Evaluation and design of readout electronics for electron and proton detectors

A thesis by

Aleksander Kårstad Nes

for the degree of

Master of Science in Physics

Department of Physics and Technology

University of Bergen

June 2017

Abstract

Researchers at Birkeland center of space science have initiated a project where the precipitating flux of energetic electrons and protons into the middle atmosphere will be measured. An instrument will be attached to a low earth orbit (LEO), polar satellite. At the start of this thesis, only the radiation detectors were specified. It was also determined that the detector readings would be oversampled by an analog to digital converter (ADC), and processed on a field programmable gate array (FPGA). Finding an ADC that can be used in this project, and creating interfaces and an evaluation system to it are the main goals of this thesis.

A block diagram overview is made for the entire measurement system. The discussion around this proved helpful to define tasks needed in this project, and it will continue to be helpful for future work. A potential candidate for an ADC is identified after an extensive search. Potential reliability concerns are discussed, as well as the analog signal processing that might be necessary with the ADC in question. FPGAs that are potential candidates for this project are considered. Digital design methods for ADC data acquisition and ADC control are discussed in relations to the project and the FPGA alternatives. All methods are at various levels realized and tested.

A synthesizable VHDL model of the ADC is made to test the digital designs close to the real-life application. To test against the actual ADC, a testboard is made. This testboard also contains electronics for differential conversion, electronics that enable having multiple sample-clock sources, as well as voltage level mitigation circuitry. A VHDL testbench is made to verify the digital designs in the development phase. A system on chip (SoC) is made to interface with the ADC testboard. The functionality of the SoC includes setting various output data from the ADC and checking readout data automatically, accessing ADC configuration memory, testboard-component control, ADC sample-clock generation, "real-time" monitoring of readout data, and error tracking and error alerts for long term testing.

The SoC is tested and verified in all design stages before implementation. A system is made where the SoC is internally connected to the ADC model. This is tested and verified in a computer-aided testbench in the pre-synthesis-, post-synthesis-, and post-layout- stage. After implementation, the SoC is tested in a physical, software-based testbench. A similar system is made where the signals between the digital designs and the model are looped back through I/Os on the FPGA. This is also tested in computer-aided- and physical- testbenches. The testboard SoC is also tested and verified in a computer-aided testbench. The testboard is realized at a late stage in this thesis. It was not enough time to properly test the testboard SoC and testboard together. Some issues are found, but it is possible that the first version of the testboard is fully functional. If so, further testing can commence.

Acknowledgments

First and foremost, I would like to give special thanks to my supervisors **Johan Alme** and **Kjetil Ullaland** for giving me support and perspective, as well as always being available when help was needed.

I would also like to thank **Hilde Tyssøy** and **Johan Stadsnes** for the help they gave me regarding the project background.

I would like to thank **Bilal Hasan Qureshi** for designing the PCB for my circuit designs, and for performing measurements on it post-assembly. I must also thank **Shiming Yang** for supervising the entire testboard process, and **Per Heradstveit** for assembling the components on the testboard.

My fellow students deserve thanks for a great environment at our office these two years at UiB.

Last but not least, I would like to thank my friends, and especially my family, for supporting me throughout my studies, and in general.

Contents

CHAPTER 1

Introduction

This chapter gives an introduction to the background of the distribution of energetic electrons and protons (DEEP) project. The goals of this thesis are also defined, and an overview of the thesis structure is given.

1.1 DEEP background and motivation

The content in this section is mostly based on relevant work performed by Linn-Kristine Glesnes Ødegaard [1], and on the conceptual design report of the project [2].

In the simplest way, the earth can be seen as a dipole magnet. A magnetic field resides from the south hemisphere (magnetic north) to the north hemisphere (magnetic south). It protects the earth from charged particles coming from the sun, also known as the solar wind. This is mostly composed of electrons and protons. The magnetic field deflects most of these particles and effectively shields the earth and its atmosphere from the direct impact of this radiation [3]. Figure 1.1 shows an illustration of the magnetic environment around the earth. This is generally called the magnetosphere [3].

Some of the radiation gets trapped in belts that surround the earth and its atmosphere. These are called Van Allen belts, named after their discoverer. Figure 1.2 shows an illustration of the belts. It is generally believed that the solar wind is the source of the particles in these belts [3]. The inner belt mostly contains protons, while the outer belt mainly contains electrons. The energy range in the outer belt is depicted in figure 1.1.

The belts are caused by interactions between charged particles and the magnetic field. One process causes particles to move between the magnetic poles in a bouncing motion. This depends on the angle between the particle velocity vector and the magnetic field. This angle is defined as the pitch angle. The pitch angle grows larger when the particle approaches one of the poles. If it becomes 90° at some point, it mirrors. Hence the bouncing motion. However, if this does not happen before the particle has entered the atmosphere, it can lose its energy to atmospheric atoms. This is called particle precipitation.

Some of the precipitating particles have energies that allow them to travel into the middle atmosphere before their energy is lost. This transfer of energy can affect the

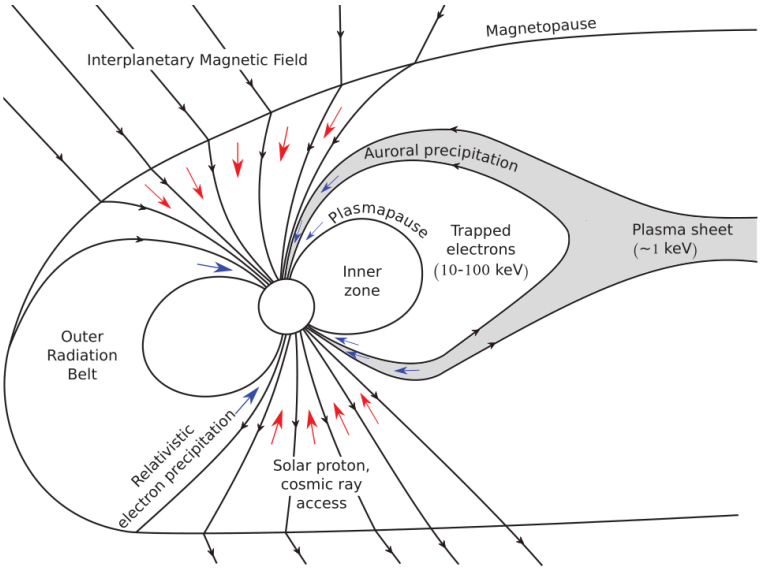


Figure 1.1: Illustration of the magnetosphere [4]

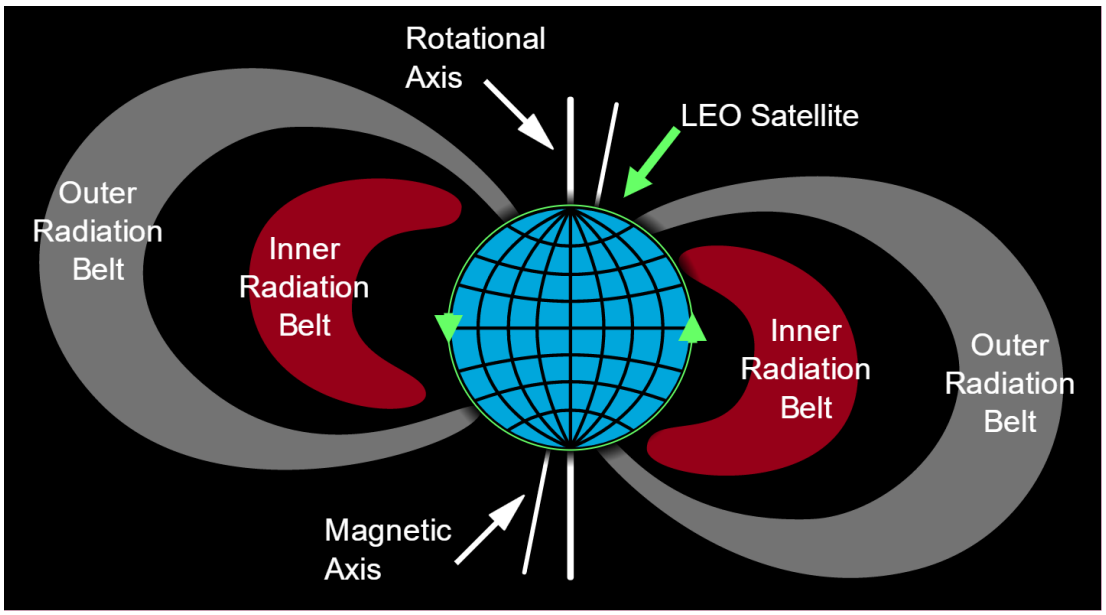


Figure 1.2: Illustration of Van-Allen radiation belts. [5].
Satellite trajectory of an LEO satellite has been added.

atmosphere in different ways. One effect is the creation of certain gasses that destroy ozone gas. Ozone gas has the well known function, amongst others, of absorbing harmful ultraviolet radiation from the sun. To accurately quantify the effects, the number of particles, their energy, and where in the atmosphere they precipitate should be measured globally and continuously. Results from the satellites NOAA POES with the particle detector MEPED provide the closest match to these criteria. However, the measurements have insufficient pitch angle coverage, proton contamination effects of measured electron data, and vice versa. This complicates reliable use of the obtained data.

By using newly designed detectors, it is possible to obtain the data that is needed to correct the pitch-angle uncertainties. Birkeland centre of space science has initiated the creation of an instrument that measure, process and transfer the measurement data to earth. The instrument will be attached to an LEO satellite in a polar orbit, as illustrated in figure 1.2. The launch is expected to be in 2019. Starting the development of this instrument is the focus in this thesis.

1.2 About this work

At the time of writing, the detectors are the only specified parts in the system. The functionality of the system is decided, but specifications on how to achieve it are not entirely specified. In this thesis, a general overview of the necessary parts in the measurement system is made. The primary objectives are however to find a way to convert the detector readings into a digital format, create and implement interfaces for the digitized data, and test against the device. Testing it imply that a test system must be made as well.

1.3 Thesis outline

Chapter 2 provides a basic understanding of the challenges microelectronics face in space, as well as for radiation mitigation techniques. It also provides a basic overview of the functionality that needs to be implemented in the measurement system.

Chapter 3 contains information about components that were found to be possible alternatives in the measurement system. This concerns an ADC with necessary analog circuitry and an FPGA which is used as the platform for digital functionality.

Chapter 4 discusses methods of interfacing from the FPGA to the ADC. Different alternatives are discussed, and considerations are made towards the measurement system.

Chapter 5 describes the test and verification systems that were made in order to evaluate the digital designs that were made, as well as the actual ADC.

Chapter 6 presents an overview of the testing that was performed and the test results.

Chapter 7 contains a discussion and conclusion about the work that is done in this thesis, and a discussion about future work of the project.

Appendices contain information about methodologies used in this thesis, tools that were used, details about components that were considered, and how to set up the FPGA systems that were made.

1.4 Citations

It was chosen to deviate from standard citation-placement rules in certain cases. The following rules apply, in falling order: If the citation is before the punctuation, it applies to statements in the sentence. If it is after the punctuation, it applies to all statements made before the citation. If citations are made at the top of a section, it applies to all the statements in the section.

CHAPTER 2

Background and related work

This chapter goes through some aspects of the environment for a LEO satellite, and challenges of electronics in space. It also introduces a basic overview of the different parts of the instrument that will measure the radiation, and process the information.

2.1 Radiation

In this project, radiation is a source of information, as well as a source of concern. Thus, a brief explanation should be made. Radiation is a term used to describe mass and energy transportation through space. Examples of such radiation are electrons, protons, heavy ions, neutrons, and gamma. These different types of radiation have different properties. Radiation may interact with matter, i.e. a medium composed of an atomic nucleus and extranuclear electrons. The interaction process depends on properties of the radiation, as well as properties of the medium. The common factor is that if an interaction occurs, energy is transferred to the medium. This absorption of energy may influence the medium. As will be discussed in 2.3.1, this transfer of energy can be utilized so that the energy of the radiation can be measured. As will be discussed in section 2.2, this energy transfer can cause problems in microelectronic devices. [6, 7]

2.2 Electronics in space

As mentioned in section 1.1, the instrument will be attached to a low earth, polar orbit satellite [2]. The trajectory will be at a height of about 600 km above the surface of the earth. Each orbit will take around 100 minutes. Most of the mass of the atmosphere is below 100 km, and thus, this is where space begins [1]. Due to the low density of mass and low pressure, regions above this altitude can be considered a vacuum. The environment is quite different compared to the environment experienced on the surface of the earth. Some of the challenges are: [8, 9]

- The experienced temperature will cycle between -20° C to+60° C [2].

- The heat generated in the electronic circuits can only be dissipated through radiation.
- The absence of an atmosphere provides no shielding against the high-energy radiation present in this region.

Charged-particle-radiation effects deserve some extra attention because of its effects on integrated circuits (IC). In a low earth orbit, the electronics will be exposed to electrons and protons in the inner radiation belt, and solar energetic protons and galactic cosmic rays in the polar regions [2]. Long term exposure to these sources will cause degradation of the electrical properties. At some point, the device will fail to function. It is important that the device can handle the expected radiation over the entire operation lifetime. Single event effects (SEE) is a term that covers multiple effects caused by a single particle strike. Examples of these are single-event latchup and single-event upset (SEU). Latchup can cause a short circuit in CMOS transistors, which will lead to device malfunction if not detected in time [10]. SEU causes data corruption as it changes the content stored in a memory element, such as a D-flip-flop (DFF) or an SRAM-cell. [11, 12]

Shielding around the electronic device can decrease the effects of charged-particle radiation, but it can not prevent them entirely. Microelectronic devices can be made more radiation tolerant by using different techniques. Radiation-hardening by design implies using specialized architectures on memory elements. By increasing the area, and applying feedback, an SEU error can be corrected [10]. Error detection-and-correcting codes is a method where control bits are added to memory data. By including logic circuitry, an eventual error can be detected, and in some cases corrected. Scrubbing is a method where content in a memory is read often and checked for errors. Tripple-majority voting (TMR) is a method where a logic path is implemented three times in different areas, where their outputs are connected to a majority voter. Thus, if an SEU happens in one path, it will be neglected as the data in the other two paths have the majority [10, 13]. The problem of latchup can be removed by using silicone-on-insulator (SOI)-transistors, which are immune from latchup [10]. By employing these mitigation techniques, reliable operation of microelectronics in space can be achieved.

2.3 Measurement system

This section presents a high-level overview of the functionality that should be included in the system. Some parts of the system are discussed in more depth than others. The idea behind this is to provide support for the decisions that are made in this thesis, as well as a visual representation for the future work. The content is mostly based on the project design report [2].

Figure 2.1 shows the block diagram of the main circuit-board-mounted components in the measurement system. Charged-particle detectors composed of multiple sensors are the sensing elements of the system. When a sensor is struck by a particle, an electrical signal that is proportional to the absorbed energy is generated. The signal magnitude and duration are often too small for further use. Therefore, the magnitude is amplified. This is followed by circuitry that changes the shape of the signal, and consequently, increases the duration of the pulse. [14]

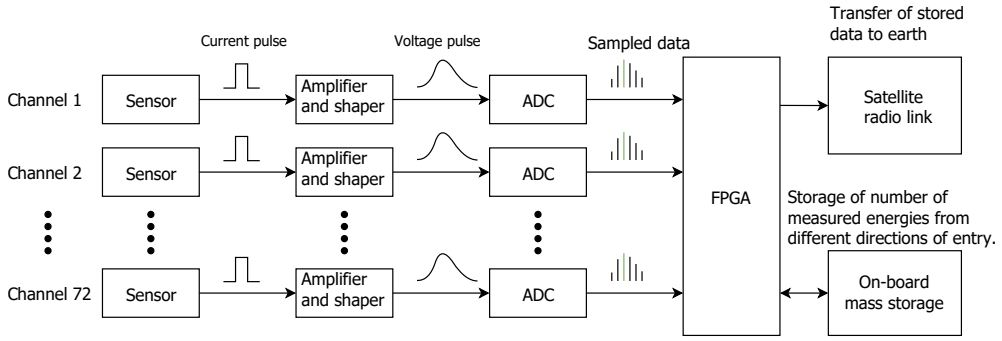


Figure 2.1: Conceptual measurement system

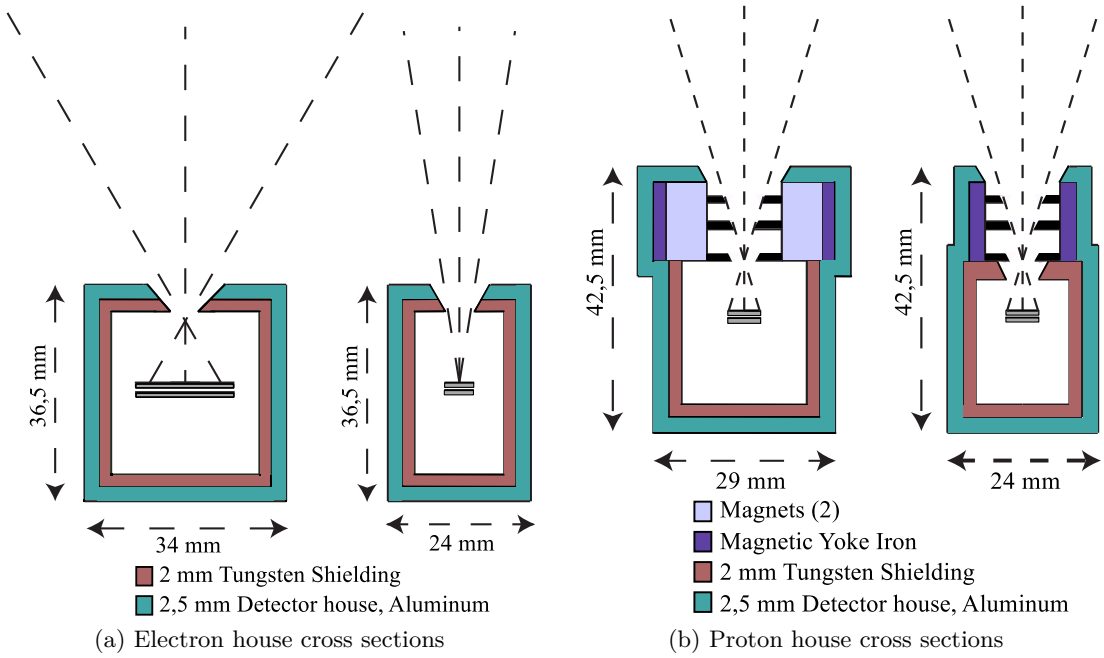
The processed signal is sourced to an ADC. The pulse magnitudes are converted from a continuous signal to a set of digital codes. These codes are transferred into an FPGA, which contains functionality specific to this project. The main functionality of the implemented system is to determine particle energies, the number of particles with said energies, and from which direction the particle came from. This information is then stored in a digital storage unit. After each orbit, stored data is wirelessly transferred to a ground station on earth via a satellite radio link.

2.3.1 Radiation measurement

In this project, the primary scientific objective is to measure electrons that precipitate into atmospheric heights between (50-100) km. The particle energy corresponding to this is (30-500) keV. This is the energy range of the electron sensors. Proton measurement is mandatory because protons will contaminate electron data. Thus, these data can be used to correct the electron measurement data. The range of energies that is monitored for protons is 30 keV-10 MeV. The extended range also allows measurements of other phenomena, but this is not discussed further.

Figure 2.2 shows the detector houses. Electrons and protons will be measured by three houses each. The houses will point in different directions so that the combined field of views cover the angles of interest. A combination of aluminum and tungsten forms a house around the detectors to stop electrons of energies less than 6 MeV and protons of less than 45 MeV. The opening has a nickel foil shield to reduce light sensitivity, and to stop low energy protons. The proton housing is made from the same materials to prevent unwanted particles in the same energy range. Unlike the electron-detector housing, a magnetic field of 0.2 Tesla is applied at the entrance of the collimator. This prevent electrons of energies less that 1 Mev from reaching the sensors.

Figure 2.3 shows the structure of the detectors. An electron house consists of 16 sensors divided evenly in two layers, while a proton house has 8 sensors. Each sensor has its own readout electronics, which gives a total of 72 readout channels. The setup makes it possible to determine the direction of entry. As a first order explanation, a particle that enters through the hole will strike two vertically adjacent sensors. Thus, the total particle energy is the combination of both sensor outputs. If only one sensor in a pair provides an output, the particle did not enter through the hole. If this is the case, the measurement is not valid.



(a) Electron house cross sections (b) Proton house cross sections

Figure 2.2: Illustration of detector houses with incoming radiation [2]

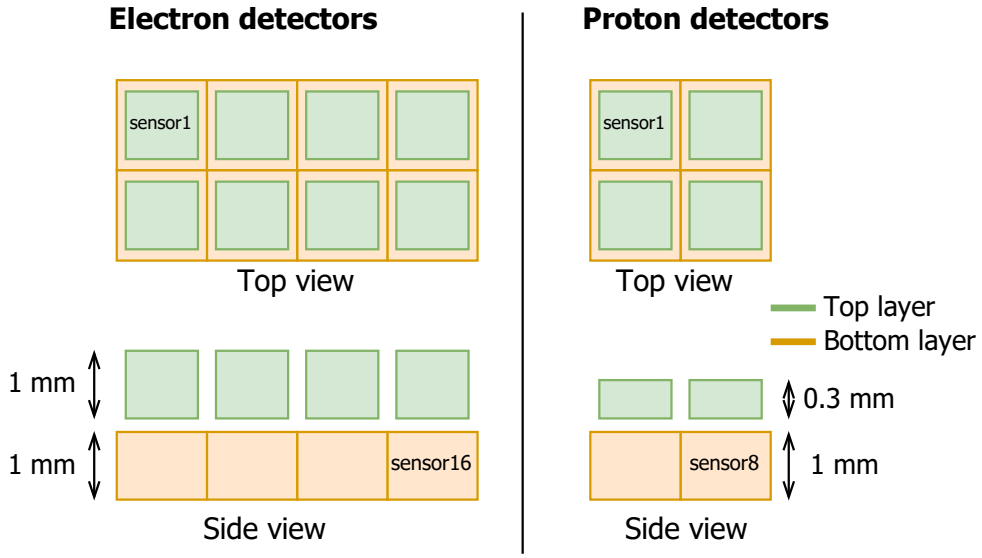


Figure 2.3: Illustration of detector configurations

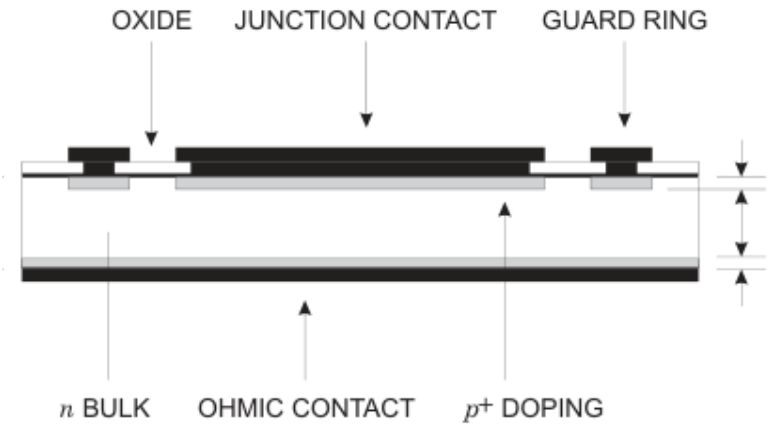


Figure 2.4: Semiconductor detector [14]

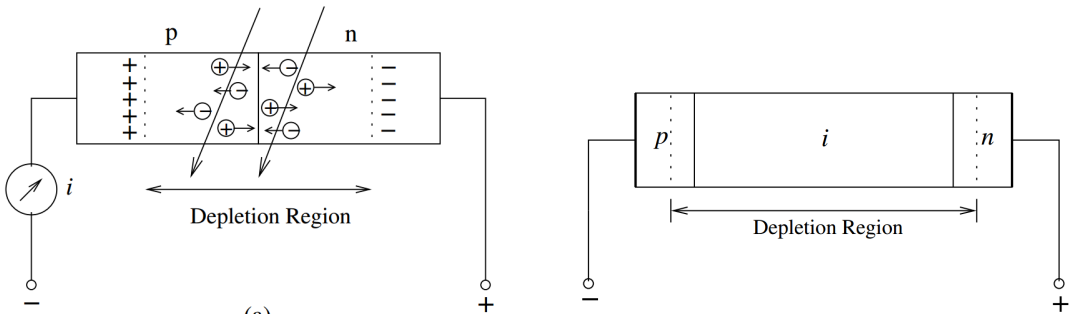
2.3.1.1 Sensors

The detectors are made from a solid-state, semiconducting material. N-doped semiconductors have mobile, negatively charged electrons in the material. P-doped semiconductors have too few electrons in the crystal structure, which is commonly referred to as having mobile, positively charged holes in the material. When a p-type and an n-type semiconductor are connected, a pn-junction is formed. Electrons from the n-type material will diffuse into the p-type material, and vise versa. This causes a region between the two junctions where there are no mobile charge carriers. [6] Figure 2.4 shows a version of a semiconductor detector. A silicon wafer is made to have a lightly n-doped bulk. A heavily p-doped layer is made on the wafer top to create a pn-junction. A metallic contact is placed on top of this layer. [14]

The depleted pn-junction in the diode, commonly called the depletion region, is quite narrow. By applying a reverse bias voltage, this region increases. This is the area that is sensitive to radiation. If a particle traverses through it, electron-hole pairs are created. The number of generated pairs are proportional to the incident particle energy. Because of the reverse-bias potential, the electrons are pulled toward the positive terminal, while the holes are pulled to the negative terminal. [14] Thus a current is flowing. The described process is illustrated for a pn-junction in figure 2.5a.

The depletion region should be as large as possible to have a large, radiation-sensitive area. In a pn-diode, this must be done by increasing the reverse-bias voltage. If the voltage is greater than a certain magnitude, the structure of the crystals will breakdown. Therefore, it can be difficult to get a sufficient depletion region in this type of detector. A way of mitigating this is to insert an undoped layer between the p-and n- type materials, called an intrinsic layer. Figure 2.5b illustrates this. This reduces the need for a large reverse-bias voltage. This is a common method, and this is the type of sensor that will be used in this project. [6]

As shown in figure 2.3 the thickness of the sensors in each layer is equal in the electron houses. A thickness of 1000 µm is specified for these sensors. The thickness of the sensors in the proton houses is different in each layer. The top layer is specified to be 300 µm, while the bottom layer is 1000 µm. Electrical contact is enabled through a 20 µgcm^{-2} thick aluminum film. The pn-junction is going to be totally depleted,



(a) Charged particles traversing through the depletion region of a pn-junction[6]

(b) PIN diode [6]

Figure 2.5: PN and PIN radiation detectors

meaning that the thickness of the depletion region extends to the negative terminal of the sensor [2, 14].

2.3.2 Charge sensitive amplifier and pulse shaping

Figure 2.6 shows the amplifier and shaping circuitry, as well as the shape of the electrical signal in different stages.

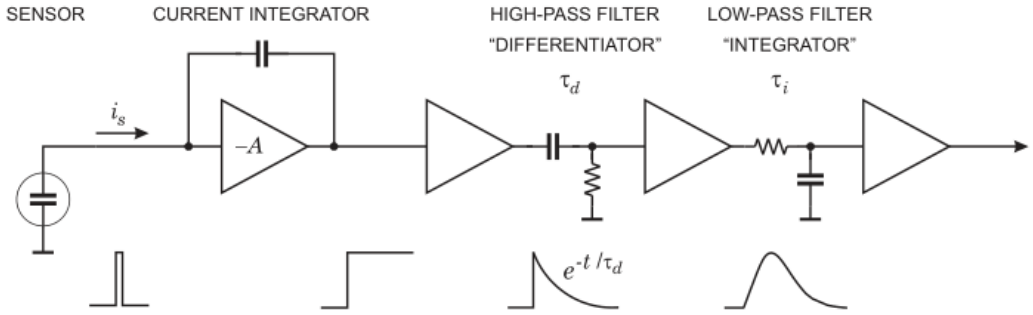


Figure 2.6: Charge sensitive amplifier with pulse shaping [14]

The input stage is a charge sensitive amplifier (CSA). The current signal from the detector is injected into the feedback capacitor which acts as a charge integrator. The resulting output is a voltage step with a short rise time and long decay time. The voltage amplitude depends on the amount of charge that is injected into the feedback capacitor. Thus, it is proportional to the particle energy. [14]

If another particle strikes the detector before the voltage has returned to the baseline voltage, the amplitude of the output voltage from the CSA will represent a combination of multiple particle energies. This is called pile up. To avoid this, a high-pass filter is used to decrease the decay time to a suitable value. The resulting voltage signal is a narrow pulse. As the amplitude is of interest, an increase in the time where the amplitude is present is practical for further use as this reduces the sample rate requirements of the following ADC. The high-pass- and low-pass- filters change the rise time and decay time, and as a result of this the shape of the pulse. In the frequency domain, this translates to a reduction in the bandwidth of the signal. This improves the signal-to-noise (SNR) ratio as the thermal noise contribution is reduced. [14]

A maximum shaping time of 1 µs has been determined to be necessary to avoid pile up. It is wished to use an application specific integrated circuit (ASIC) where both CSA and pulse shaping is integrated. At the time of writing, this part has not been determined. Finding a suitable ASIC was not a part of this thesis, and will not be discussed any further.

2.3.3 Analog to digital conversion

An ADC converts the time- and amplitude- continuous signal from the previous stage into a discrete set of values that represent the magnitude of the sampled analog signal. A general ADC operation begins by sampling the input signal at fixed time intervals. The value is held between each sample. It is then quantized, i.e. rounded to the nearest value of a finite set of discrete values. The value is then encoded, i.e. converted to a number format such as the binary number system. [15]

In order to get a proper representation of the sampled signal, the sampling frequency must be $F_s \geq 2F_{max}$, where F_{max} is the highest signal frequency component [15]. The resolution of the ADC is a combination of a reference voltage and the number of bits of the output code, or:

$$V_{res}=\frac{V_{ref}}{2^n}$$

where n is the number of bits. If the input signal V_{p-p} is larger than the reference voltage, the ADC will saturate. [16]

Fluctuations in temperature, supply voltage and atmospheric pressure amongst others, affect the transfer function of an ADC. These can however be compensated for. Intrinsic non-linearities will also create transfer-function deviations. Unlike environmental effects, these cannot be compensated for. In certain cases, this can result in missing output codes, which can be severe for low-resolution converters. [16, 15]

The non-linear transfer function causes harmonic frequency products, which consequently causes distortion. The noise that is present in the signal paths will decrease SNR. Signal-to-noise-and-distortion (SINAD) is a term that combines the effects of distortion and noise, as a measure of the total dynamic performance of the device. It is common to convert this to the effective-number-of-bits (ENOB). ENOB and the specified number of bits should be as close as possible. [16, 17]

Finding applicable ADCs was the starting point in this thesis. This is discussed further in section 3.3.

2.3.4 FPGA

An FPGA is chosen as the platform where digital processing and system control will be implemented. FPGAs are ICs that can be configured to perform any logical function. Figure 2.7 shows the architecture of IGLOO2 (IG2), which is an FPGA from Microsemi. Naming convention differs between vendors, but the implementation is similar. The fabric, which is the programmable section, is composed of multiple logic clusters. The clusters can be connected together through an interconnecting array, which is also programmable. Each logical cluster contains multiple logical elements, as shown in figure 2.8. A lookup table (LUT) is the programmable element. It can be configured to act as any combinational element of up to 4 variables [10]. If more variables are needed, multiple LUTs can be connected. The logical elements also contain a sequential element. Both can be used separately or together. [18]

To configure an FPGA, the logic functionality must be described in a hardware description language (HDL). In this thesis, the language very high speed integrated circuit HDL (VHDL) is used. Syntactically, it has resemblances to software languages such as C. However, where software is sequential instructions of what a microprocessor will do, VHDL describes the implementation of digital hardware. Examples are combinational logic, sequential elements, and memory cells.

Special purpose sub-systems are often also included in the device as hard components. Typical examples are phase locked loops (PLL) for clock conditioning, storage memory, and mathblocks. A microprocessor, i.e. a CPU, can be implemented in the fabric. This is referred to as a soft implementation. However, certain devices contain a hard CPU to increase performance. Such a system-on-chip, or SoC, gives the possibility to implement functionality in both hardware, and software.

For the system that will be implemented, it will most likely be necessary to use the SEU mitigation techniques that were presented in section 2.2.

2.3.5 Data storage

Storage of data between each satellite orbit is necessary. In applications where the data amount is small, the internal FPGA memory could have been used. Most likely, this does not apply for this project. An external, higher capacity data-storage unit must be used to store the data of interest. The required memory capacity will depend on the number of incoming particles in the energy ranges and directions of interest. It is crucial that the memory is not corrupted by radiation. Therefore, the memory must be radiation hardened, or at least made radiation tolerant by employing the radiation-mitigation techniques mentioned in section 2.2.

2.3.6 FPGA-implemented system

Figure 2.9 shows a conceptual block diagram of the application-specific system that needs to be implemented on the FPGA.

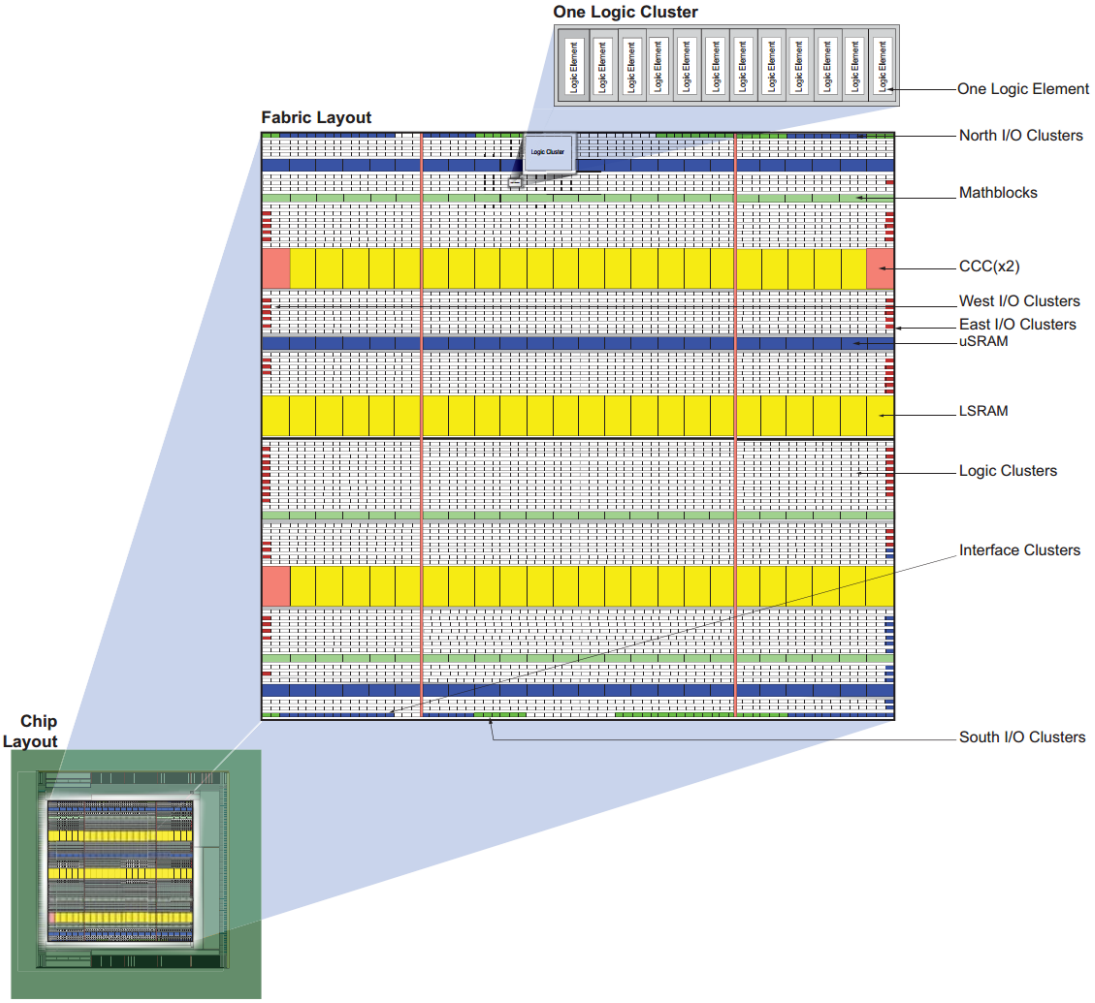


Figure 2.7: Smartfusion2/IGLOO2 fabric architecture [18]

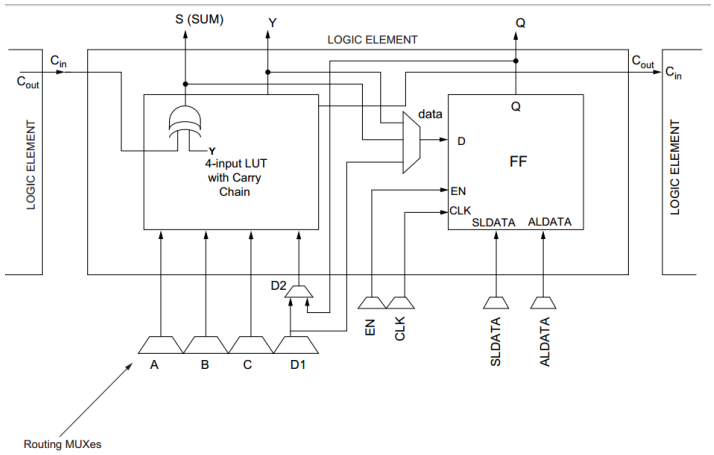


Figure 2.8: Smartfusion2/IGLOO2 LUT [18]

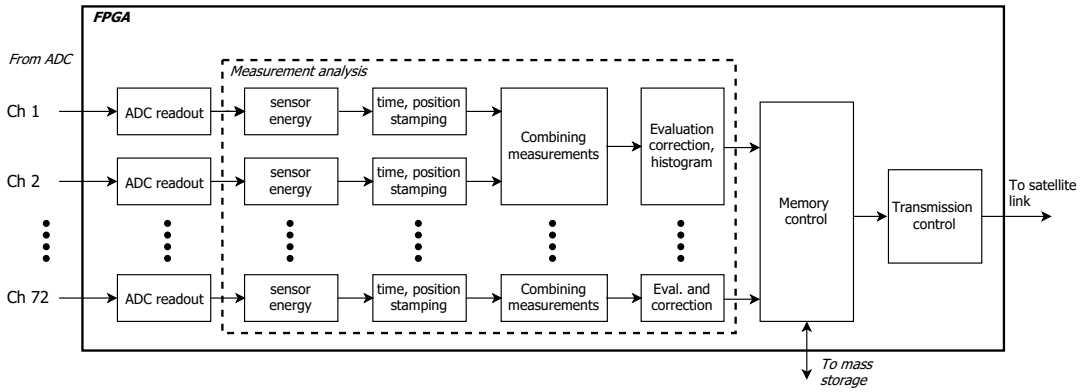


Figure 2.9: FPGA internal system

2.3.6.1 ADC readout

The first stage consists of logic that acquires the data from the ADC. The functionality of this block depends on the ADC output format. Data can be transferred serially or on a parallel bus. The transfer standard can be differential, such as low-voltage differential signaling (LVDS), or single-ended such as LVCMOS. The transfer protocol can be a common protocol such as serial peripheral interface (SPI), or it can be custom to the device. Independent on how it is transferred, data must be captured correctly for further processing.

2.3.6.2 Measurement analysis

As discussed in 2.3.2, the voltage amplitude of the output from the amplifier-and-shaping circuitry is proportional to the particle energy. As illustrated in figure 2.10, there are several methods that can be applied to extract information about such a signal. Peak search is the process of searching for the peak value in the sampled data set. Curve fitting calculates the amplitude by using an equation that describes the signal shape [19]. The third method calculates the area of the pulse by integrating the sample-data set.

As the ADC samples data at a fixed interval, data that does not contain information about particle energies will be captured as well. Ideally, the voltage input to the ADC should be at a fixed level between particle hits. In reality, noise from circuit-board components and electromagnetic interference can cause the voltage to fluctuate. It is necessary to include circuitry that evaluates if the readout data is caused by noise, or if it is an actual measurement.

The previously mentioned voltage fluctuations will mix with the signal voltage. As this influences the amplitude of the signal, it corrupts the energy information. Thus, filtering of this noise must be done. Instead of using the methods mentioned to extract the needed information, an averaging filter can be used. This will not give an accurate representation of the energy, but it will filter away the noise. If all details about the transfer function and uncertainties are known, it should be possible to determine the actual energy. The advantages of this method are that the needed logic uses a small area, and it is not complex. This reduces the probability of SEU, and the chances of

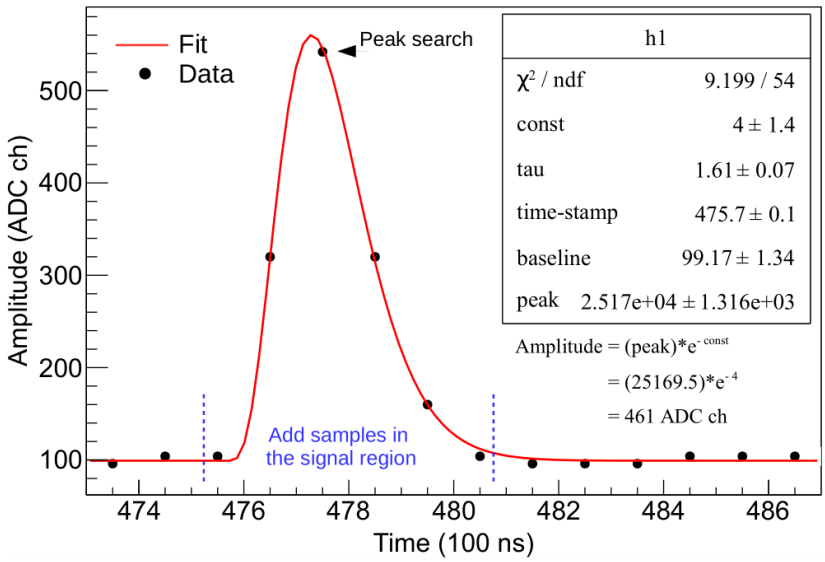


Figure 2.10: Different methods of analyzing a semi-gaussian pulse: peak detection (arow), curve fitting (red line) and area calculation (integrating samples between blue lines) [19]

something going wrong due to inadequate testing, respectively. A proper study of the methods mentioned in this section were beyond the scope of this thesis.

As mentioned in section 2.3.1, the detectors are composed of two layers, where the energy of a particle is the sum of the outputs of two vertically adjacent sensors. Protons will contaminate electron measurements and are amongst other reasons also measured. Functionality that adds the sensor-pair-measurement data to get the total particle energy must be included. Before this can be done, it must be determined that both sensor pairs give an output. Adding a time stamp to the measurement data can be used to determine if the measurements are valid. If the two sensors provide data that is close in time, it is likely that they originated from the same particle hit. One method is to create logic that only evaluates the belonging pairs. If both sensors provide valid data that are close in time, the values are added. An evaluation of proton contamination must then be performed. If so, a correction must be made to get the actual electron energy. As the input angles of particles also are of interest, this information must be added. Each sensor pair in each detector cover certain angles. Therefore, the angle can be measured by adding a stamp that tells which sensors provided the data.

The determined particle energy with angle information can be used as inputs to circuitry that counts the occurence of different energies, for different input angles. Keeping track of all measured energies for each input angle require a huge amount memory. Therefore, using histograms with bins of certain energy ranges is a more likely solution. Some energies might be more interesting than others. Hence, the size of the bins should be programmable, so that it is possible to alter the bin sizes. Another parameter can be how long data is collected before a histogram is stored, as some phenomena depend on time. The data that will be stored after analysis is done, is thus a factor of input angle, energy ranges, and data-collection time. A tag that makes it possible to identify what the data represents must therefore be added.

2.3.6.3 Memory control

A controller for the storage unit must be implemented. This controls where the analyzed data is stored, as well as read and write operations to the memory. Read and write operations depend on what type of memory is used, and the transfer protocol of the chosen memory. Multiple memories might be used, so multiple RW-controllers might be necessary. To ensure that all data is stored, the operations must be sufficiently fast. Including memory buffers might be necessary to store data temporarily before it is written to the memory. Error detecting and correcting circuitry as mentioned in section 2.2 must be implemented to correct any potential SEU.

2.3.7 Other functionality

Each satellite orbit takes approximately 100 minutes. After each orbit, the stored data will be sent to earth via radio transmission. This will be achieved by a satellite radio link, which is a module that handles the radio communication between the satellite and a ground station on earth. Once transmission is due, the module will receive a stream of data. Internally on the FPGA, logic that controls when data will be transferred must be added. This logic must also initiate the transfer of the data stream from the storage unit. Thus, a timer could control when data transfer must occur. This, however, would require that the orbit time is exact for each orbit. Further investigations on how this can be done in a reliable way were not done.

So far, only the measurement-related functionality has been mentioned. The measurement system will have a default setup, I.e. all components will have a default configuration that assures optimal functionality. This can change after the satellite is deployed. Therefore, it must be possible to reconfigure all the components in the system. This includes resetting digital devices, power cycling analog devices, and altering the content in devices with internal memories. Environmental effects such as temperature and supply-voltage fluctuations can change the transfer function characteristics of components. This must be measured so that it is possible to correct the deviations. Current must be measured in case latchup occurs, so that the charge in the component can be drained before it causes permanent damage. To have the possibility of doing all this, a control system must be implemented on the FPGA. It must monitor the entire measurement system , and perform the necessary operations once an event occurs.

CHAPTER 3

Component selection and considerations

In this chapter, general specifications are set for electronic components that are alternatives in the project. A discussion is made about what types of FPGA that might be used in the project. One of the goals in this thesis was to find an applicable ADC based on some predefined requirements. The ADC that was found to be the best alternative is presented. Some considerations about the reliability of the ADC is also discussed. The signal might have to be processed before the ADC input. This is also discussed.

3.1 General consideration

Certain manufacturers produce space qualified electronics. There are strict requirements concerning manufacturing and testing of such ICs. The resulting components are expensive, and often outdated compared to commercial products because the process is time consuming. While finding components in this thesis, performance, capability and cost of the parts had priority over reliability. Electronics made for commercial use were not considered as it is unlikely that they would work in a space environment. Space qualified electronics were not considered as they generally are to expensive. The alternative option is to acquire a component which is not space qualified and test it, i.e. upscreen it to the requirements set for space operation. The latter is the preferred method in this project. The initial general requirement that was set was that the device must be able to operate in the expected temperature range of -20° C to +60° C [2].

3.2 FPGA

Memory-based FPGAs are alternatives to fuse-based FPGAs and ASICs. An issue with memory-based FPGAs is the possibility of radiation-induced upsets in the configuration data. This can change the intended functionality, and power-cycling or re-configuration must be done. Configuration-memory technologies used in modern FPGAs are generally SRAM or FLASH. Standard 6-transistor SRAM cells are more susceptible to SEU than FLASH-transistors. The energy transfer that is necessary to flip a bit in the latter is large, so FLASH cells are often referred to as SEU immune. The mitigation techniques

mentioned in section 2.2 can be employed to make SRAM more radiation tolerant. This can however increase the cost, reduce performance, etc. On this basis, flash-based FPGAs will be considered for this project. [20, 21, 13].

The main vendor of these types of FPGA is Microsemi. They deliver FPGAs for many applications, ranging from commercial to space applications. Differences lie in specialized features. Examples are integrated high-bandwidth transceivers, integrated microcontroller system, double-data-date (DDR) memory systems and number of inputs/outputs (I/O). Other factors are number of logic clusters, number of PLLs, power consumption, etc. Table 3.1 lists FPGA alternatives. Although all of the listed devices are options for this project, SmartFusion2 (SF2) and IG2 are mostly treated as the FPGAs of interest in the remaining sections. These and PolarFire can be delivered with military classification. PolarFire was however announced in the midst of this thesis, and was not evaluated beyond the contents in table 3.1. RTG4 is a radiation-hardened, space qualified FPGA.

Table 3.1: Microsemi FPGA comparison

	RTG4	**SF2/IG2**	**PolarFire**
Logic clusters	151824	146124	481000
DSP	462	240	1480
PLL	8	8	8
High speed interface	24	16	24
Hard CPU	No	Yes/no	No
Class	Space	Military	Military

SF2 and IG2 have the same fabric, as shown in figure 2.7. The difference between the two is that SF2 has a hard microcontroller subsystem (MSS). Figure 3.1 shows an internal overview. It contains an ARM Cortex-M3 CPU which connects to hard peripherals through a bus matrix. The logic implemented in the fabric can be connected to the MSS trough fabric interface controllers (FIC). Thus, custom logic can be accessed and controlled by a software program.

3.3 ADC

Specifications listed in table 3.2 applies for the ADC. The sample rate requirements of the ADC is set so that the signal is oversampled by a factor of 5-10. This should be enough to detect the peak voltage of the signal with a tolerable uncertainty. A resolution of 10-12 bit is specified for the same reason. Multiple channels in one package reduce the printed circuit board (PCB) footprint. A serial output interface is desired as a parallel interface requires a vast amount of inputs on the FPGA. Based on the CSA-and-shaper ASICs that were previous alternatives, the ADC should expect a single-ended input. No power budget is specified for the ADC. The power consumption should however be as small as possible. Details regarding the process of finding an ADC, and the alternatives that were considered, is presented in “A.1: Finding an applicable ADC” in “Appendix A: Methods”.

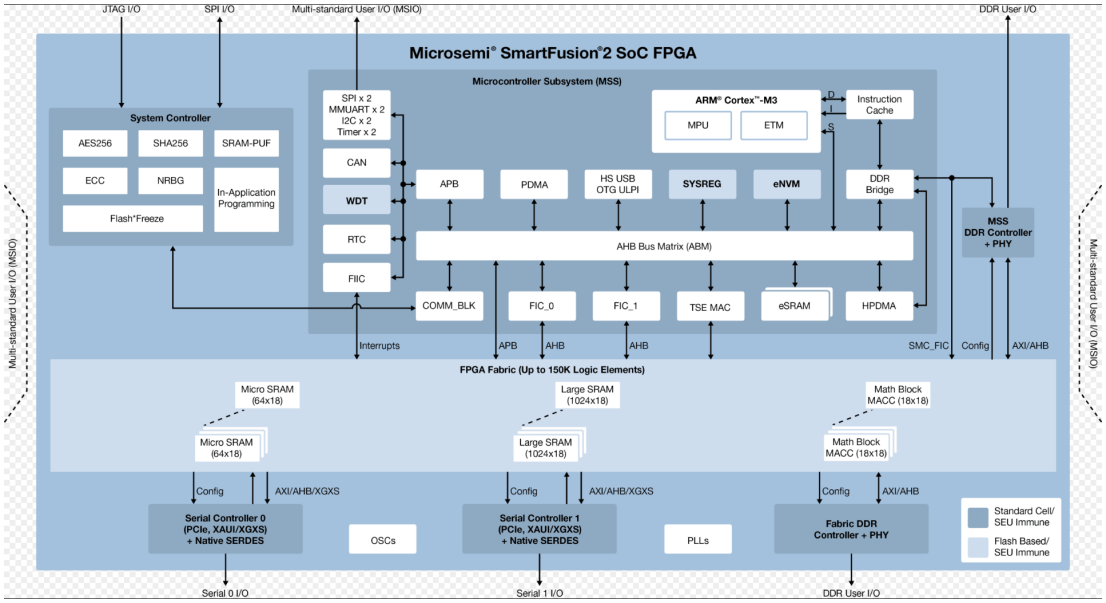


Figure 3.1: Smartfusion2 overview [22]

Table 3.2: ADC specifications

Sample rate	5 - 10 MSPS
Bit resolution	10 - 12
Input phys interface	Single-ended
Output phys interface	Serial
Power usage	Low
Nr. of channels	16
Rating	Military

3.3.1 AD9257-EP

Information is this section is obtained from the datasheets of AD9257-EP [23] and AD9257 [24].

AD9257-EP, hereby referred to as AD9257, was determined to be the best alternative. This was based on its high number of channels per package, low power consumption, and serial output interface. It is evaluated in the military temperature range, but not classified as a military device. Figure 3.2 shows a block diagram of the converter.

AD9257 is a 14-bit ADC with a sample rate of 10-65 MSPS. It has eight individual channels which are composed of multiple pipelined stages. All stages but the last are low-resolution flash ADCs in combination with a DAC. The final stage is a flash ADC, delivering the final digitized data to the output stage. The output stage aligns and corrects errors in the data before it is passed to the output buffers. Here, data is serialized and aligned to a frame clock output (FCO), and a data clock output (DCO). [24, p.18] These clocks can be used by the readout system to capture and deserialize data. Sampled data is coded into a 14-bit word with a reference of +1V. Table 3.3

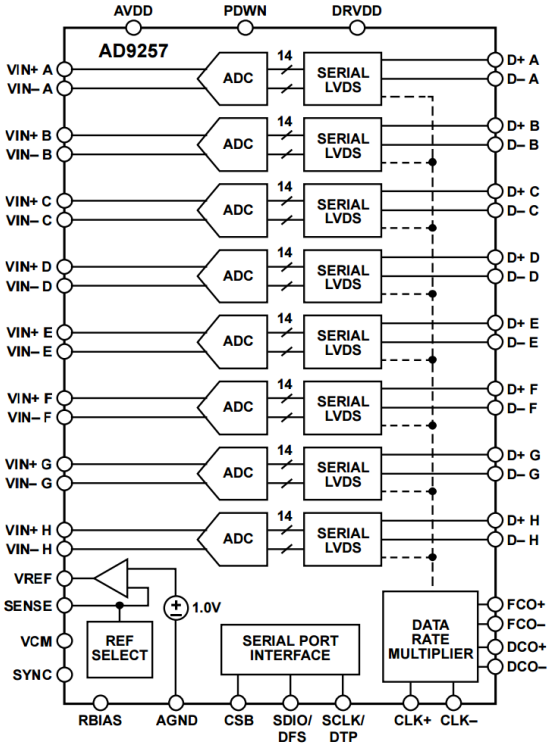


Figure 3.2: Block diagram of AD9257

Table 3.3: AD9257 output coding format[24]

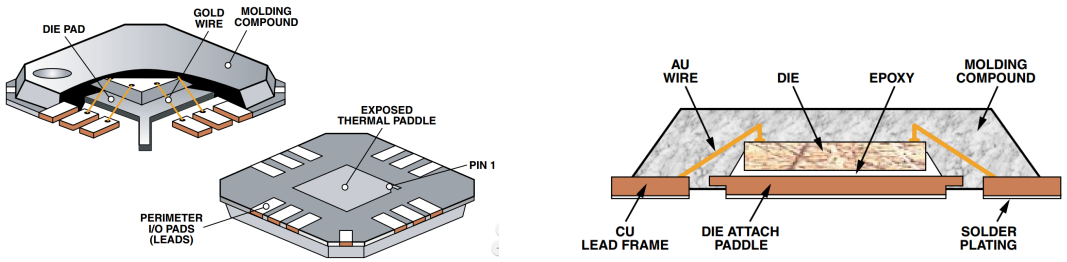
Input (V)	Condition(V)	Offset binary	Twos complement
(VIN+) − (VIN−)	> (+VREF) − (0.5 LSB)	11 1111 1111 1111	01 1111 1111 1111
(VIN+) − (VIN−)	= (+VREF) − (1.0 LSB)	11 1111 1111 1111	01 1111 1111 1111
(VIN+) − (VIN−)	= 0	10 0000 0000 0000	00 0000 0000 0000
(VIN+) − (VIN−)	= (−VREF)	00 0000 0000 0000	10 0000 0000 0000
(VIN+) − (VIN−)	< (−VREF) − (0.5 LSB)	00 0000 0000 0000	10 0000 0000 0000

shows the available output coding formats. Twos complement is default.

The converter can be configured through a three-wire SPI-bus. One particularly helpful feature for testing and verifying the readout logic is the ability to set output data to built-in- or user-defined-patterns. The configuration memory is loaded with default values if the device is reset. The operating state can be toggled via SPI or via a dedicated input `PDWN`. If multiple devices are used in a system, as would be the case in this project, synchronous sampling can be achieved by asserting the dedicated input `SYNC`. More detailed considerations regarding the outputs and the control section are presented in chapter 4, where the digital designs are discussed.

3.3.1.1 Reliability considerations

This device is an enhanced version of a commercial AD9257. It is rated in the military temperature range of -55°C to +125°C. Leadframes are coated with SnPb or NiPdAu to prevent the growth of tin whiskers, which can happen with RoHS-compliant, i.e. lead-



(a) LFCSP top and bottom

(b) LFCSP cross section

Figure 3.3: LFSCP package overview [29]

free, devices. Wire-bonds are of a non-copper material. There are stricter requirements for manufacturing as one assembly site, one test site and one fabrication site are used for all enhanced products. Manufacturing is done via a single processing flow baseline. Qualification data can also be provided by request.[23, 25, 26]

The device has a plastic, LFCSP package which is displayed in figure 3.3. It has I/O pins beneath the package, i.e. it has no leads. Packages that have leads are more robust if the circuit board is exposed to mechanical shocks. This is because the leads offer some mechanical flexibility. [27] Vibrations will ripple through the circuit board during satellite launch [28], and thus directly to the package, and can therefore pose a reliability problem.

Hermetically sealed ceramic- or metal- packages are often used in high-reliability applications. The use of plastic packages poses many reliability concerns, such as outgassing, and thermal-cycling-induced degradation of the molding compounds. However, parts with plastic packages have been used in space before. Though not as reliable, they offer advantages in areas such as size, weight, and cost. There is an abundance of components that use these packages. Consequently, there are more available options. [28, 30] Screening this device to ensure reliable operation in the LEO environment must be done. An in-depth analysis of what to test, and how to upscreen this device to higher reliability requirements were beyond the scope of this thesis.

3.4 Analog front-end of AD9257

AD9257 has differential, switched capacitor inputs. The signals applied to the inputs should therefore also be differential. The input voltage on each pin on the differential input must not exceed 2 V, and a common-mode voltage of 0.9 V must be present. The differential signal voltage-span should not be greater than $2\,V_{p-p}$ as this equals the full-scale output. As shown in table 3.3 this would result in a differential voltage greater than $\pm V_{ref}$=$\pm$1 V. [24]

At the time of writing, it is believed that the signal from the CSA-and-shaper has to be converted from a single-ended signal to a differential signal and that it must be attenuated. This depends on the ADC front-end circuitry that is used for the project, which is not yet decided. Table 3.4 lists different elements that perform single-end-to-differential (SED) conversion. The generated differential signals should ideally have the same magnitude, and have a 180° phase relationship. The differential output voltage is $(+V) - (-V)$. Mismatches in either magnitude or phase of one or both of these causes distortion, which in turn will degrade the overall accuracy of the system.

Table 3.4: Single-ended to differential conversion

Converting element	**Category**
transformer coupling / flux coupling	passive
balun / transmission line transformer	passive
Two operational-amplifiers (opamp)	active
Fully differential opamp (FDO)	active

The passive devices have advantages when it comes to high-frequency operations, noise contribution, and power consumption. Having other gains than unity will however complicate the circuit design. [31, 32, 33] This is a relatively easy process when using active devices, as attenuation can be achieved by setting the gain to < 1. There are many factors that must be considered to determine which method suits this project the best. A thorough comparative study was however beyond the scope of this thesis. To have the possibility to test the different methods, they were included on a test-and-verification circuit-board. This is presented in section 5.6.

The switched-capacitor front-end can be modeled as a resistor and capacitor in parallel. This makes the impedance frequency dependent. The value of the capacitor also changes depending on if the ADC tracks the input signal, or if it is in hold mode. To ensure proper signal integrity, and optimal performance of the ADC, an impedance compensation network should be included between the SED converter and AD9257. A proper compensation network will minimize the effects of the imaginary part, and also set the real part to a value that matches the source output, and PCB trace impedance. [34]

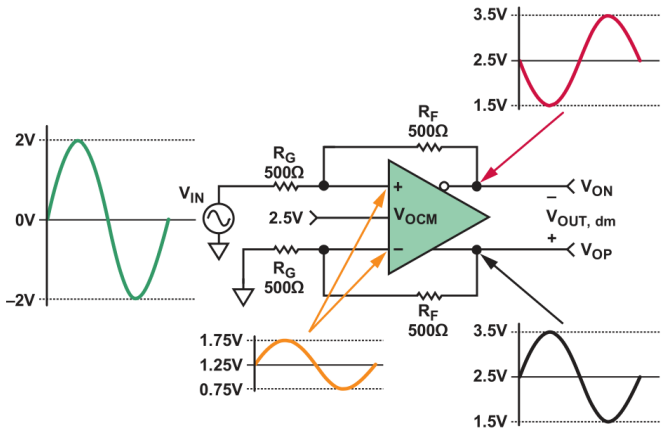


Figure 3.4: Fully differential amplifier, single-end to differential conversion [36].

FDOs are made to produce differential output signals, while the two-opamp configuration simulates such an architecture. Based on this, the FDO was given some extra attention. Figure 3.4 shows an FDO and the signal propagation with unity gain. This is achieved by setting R_G=$R_F$in both feedback paths. The signal will be attenuated if R_G>R_F. The resistor values must be closely matched to avoid phase unbalance. A common and practical feature on FDOs is the inclusion of an input that sets the output common-mode voltage. Thus, the required 0.9V can be set via this input. An FDO that could be an alternative in this project is THS4524-EP [35]. As AD9257-EP, this is an enhanced commercial product.

CHAPTER 4

Digital design

In order to capture data from AD9257, as well controlling it, systems that match the transfer protocols of the device must be made. This chapter discusses different methods, and which methods can be used in the final project. To fully understand the structures of the VHDL designs that are presented in this chapter, "A.2: VHDL design" and "A.3: Design flow" in Appendix A: Method should be read.

4.1 AD9257 readout logic

In order to capture the data from AD9257, a readout system must be designed. In this section, several readout methods are considered. Table 4.1 gives an overview of the methods that are discussed.

Table 4.1: AD9257 data-capture methods

Readout method	**Implementation**
SERDESIF	Hard
DDR-MSIO	Hard/soft
DDR-fabric	Soft

4.1.1 Readout source considerations

Information in this section is obtained from the datasheet of AD9257 [24]. Figure 4.1 shows the timing diagram of the ADC outputs.

CLK± is the sample clock. After an input has been sampled and digitized, data is serially loaded onto the output. FCO± is asserted each time the first bit of the 14-bit word is put on the output. FCO± has the same frequency as CLK±. This clock can be used to separate the captured words in the readout system. DCO± has a frequency of 7×CLK±, and is aligned to the center of the data output D±. Both edges of this clock can be used to capture each bit in the 14-bit word. All of the outputs are differential and conform to the ANSI-644 LVDS standard.

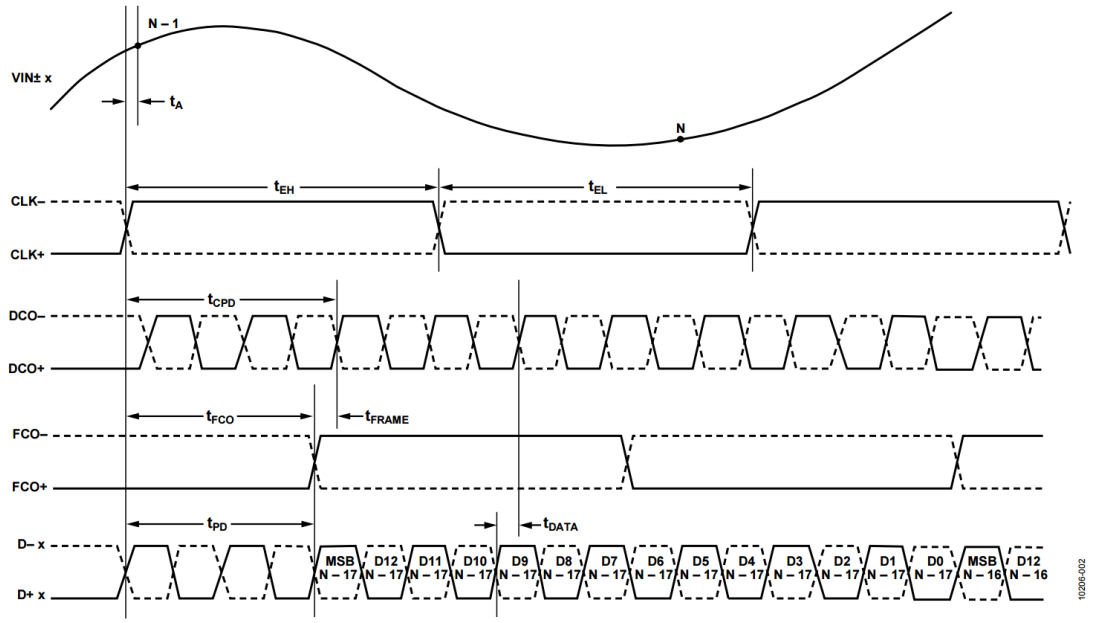


Figure 4.1: AD9257 sampled data output.[24]

4.1.2 Specifications

IO
The IOs must be configured to receive LVDS signals. To get data from all 72 sensors, 9 AD9257 must be used. This yields 8 data outputs and 2 clock outputs per ADC, for a total of 90 differential inputs.

DDR data capture
The minimum sampling rate of AD9257 is 10 MHz. In regards to the specifications of the project, this is closest to the desired sample frequency. This means that the minimum capture clock frequency is 70 MHz, as DCO is 7×CLK. The data from all channels, per AD9257, will be captured using this clock.

Deserialization
The data rate with minimum sampling is 140 Mbps. After 14 consecutive bits have been captured in each channel, data must be deserialized. This puts the data on a 14-bit parallel bus for further processing.

4.1.3 Serializer/deserializer interface

SF2 has hard serializer/deserializer interface (SERDESIF) blocks with dedicated inputs that can be used for capturing and deserializing data of differential signals. The block can be configured to work with standard serial protocols, as well as custom protocols through an external physical codings sub-layer mode. It is however not fully customizable. The data bus widths from the deserializer can only be set to predefined widths. 14 bits is not an alternative. The minimum input reference clock is 100 MHz. This is beyond the target 70 MHz. The minimum supported data rate is 1 Gbps. The target data rate in this project is 140 Mbps. Neither of the FPGAs discussed in section 3.2 have enough SERDESIFs to cover all detector outputs.[37] Using SERDESIF, if possible,

would require multiple FPGAs, and sampling the data inputs at a much higher rate than what is required.

4.1.4 Custom logic

Custom-made readout logic can be implemented in the fabric of the FPGA. The outputs from the ADC will in this case connect to the fabric through multi-standard I/Os (MSIO). These can be configured to accept differential LVDS inputs by connecting the input signal to compatible N- and P-inputs. Versions of SF2 and IG2 with enough I/Os to capture data from all sensors are also obtainable. [18]

Figure 4.2 shows a 14-bit capture- and deserialize-circuit. The circuit is based on the 12-bit circuit in [38]. The data input and `DCO` connect to a DDR capture block. In this block, data is captured by DFFs on both edges of `DCO`. The outputs are connected to two separate serial in, parallel out, shift-registers: one for rising-edge data, and one for falling-edge data.

Data is subsequently shifted through the two shift registers. The entire word should be shifted in by the time data from a new word is present on the input. This is signaled by the rising edge of `FCO`. Content stored in the two shift-registers is latched into a 14-bit, frame capturing register that is triggered by `FCO`. The now deserialized data which is stored in this register must match the serial input data. This can be achieved by connecting the outputs from both shift registers in an interleaved manner, as pictured in the figure. The data on the outputs of this register can be used for further processing.

Clocks should be distributed to the register elements with minimum delay between clock edges. Routing through the fabric is not necessarily uniform, so the length of the wires between DFFs can differ. This can result in a skew between the clock edges on subsequent flip-flops, and timing margins could be worsened. For this reason, global or local clock nets should be used. Usage of such is indicated by the green wires in figure 4.2.

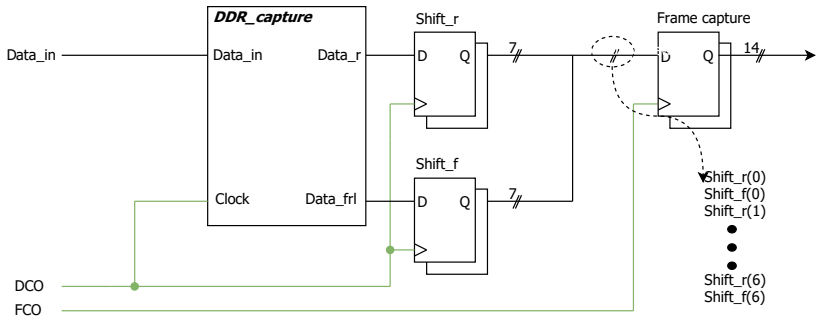


Figure 4.2: Readout circuit

4.1.4.1 DDR-MSIO

MSIOs can be configured to perform DDR capture by instantiating a DDR-macro in the VHDL design [18]. Figure 4.3 shows the schematic of the macro[1]. Rising-edge data is captured directly by the top-right flip-flop. Falling-edge data appears on data input `D` while `CLK` is high, as shown in figure 4.1. Since the latch is transparent when the clock is high, the state of `D` is stored on the output of the latch once the clock goes low. Data is then transferred to the output `QF` on the following rising-edge of `DCO`.

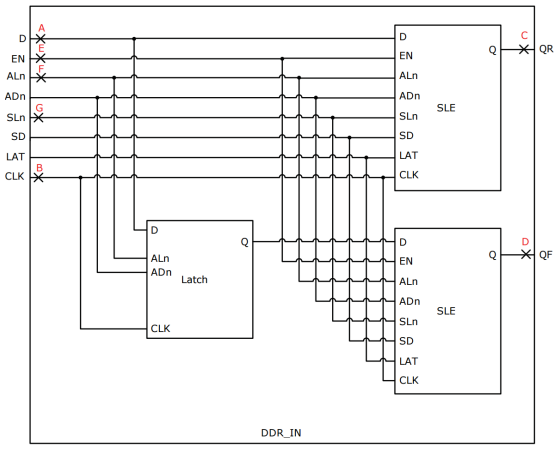


Figure 4.3: DDR_IN macro[39]

Timing when using `DDR_IN` in place of `DDR_capture` in figure 4.2 is shown in figure 4.4. When `FCO` is asserted, all bits must be captured. The blue lines represent data captured on the rising edge, while the orange lines represent data captured on the falling edge. As the diagram shows, it is not possible to latch all of the data bits into the frame register when `FCO` is asserted. The reason for this is that the first capture edge of `DCO`, as shown in figure 4.1, is a rising edge. This creates a skew between the two capture paths, where the data captured on falling edge trails behind the data captured on rising edge by one clock cycle. When `FCO` is asserted, rising-edge data is present on the outputs of `shift_r` and `data_r`. Because of the skew, the last captured falling-edge data will not be present on `data_frl`. `Data_f` is not reachable. The data latched into the frame capture register is for this reason not valid.

It is however possible to use MISOs configured for DDR if the phase of `DCO` is shifted by 180° before it is connected to the clock inputs in figure 4.2. This removes the skew that caused the non-valid readout. Timing in this case is shown in figure 4.5. The phase shift can be achieved with an inverter or a PLL.

Using an inverter to shift the phase is possible. It will however add delay to `DCO`, which results in a change in the timing relationship between it, `FCO` and the data. The significance of this increases with increased frequency, as illustrated in figure 4.6. This is true assuming the delay contribution from the inverter is independent of the clock frequency. As the target frequency of `DCO` is 70 MHz in this project, the added delay might not be critical.

[1] In the timing diagrams, signals in the DDR macro: D = Data_in, Q on the latch = Data_f, QR = Data_r, QF = Data_frl

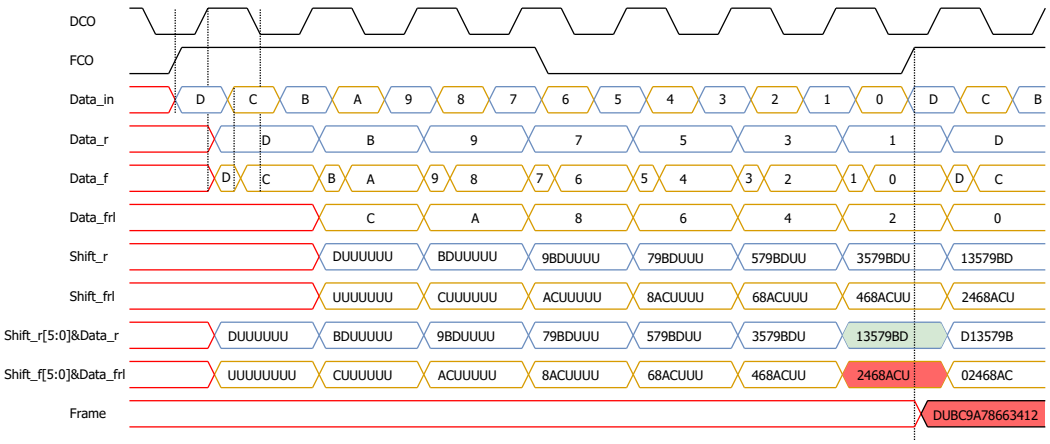


Figure 4.4: DDR capture using DDR_IN macro timing.
The blue lines represent data captured on rising edge. Falling edge is represented by orange lines. Green fill marks that all bits are captured before FCO is asserted. Red fill marks the opposite.

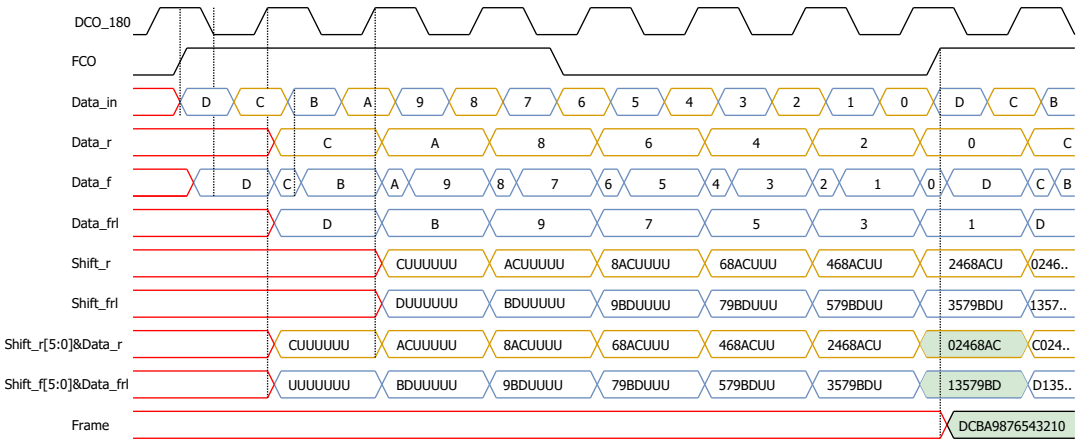


Figure 4.5: DDR capture when DCO is skewed 180 degrees timing.
The orange lines represent data captured on rising edge. Falling edge is represented by blue lines. Green fill marks that all bits are captured before FCO is asserted.

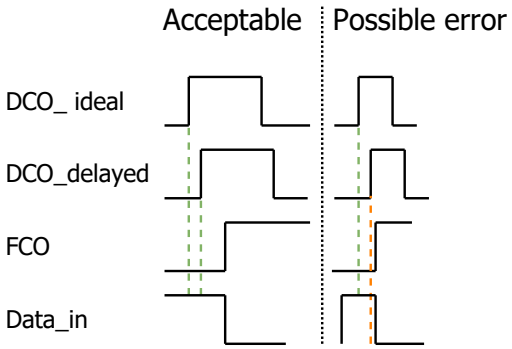


Figure 4.6: Significance of skew on DCO in relations with FCO and data.

By using a PLL, both a phase shift and/or a delay can be added to `DCO`. The advantage of a PLL is that it corrects the output if a change is detected between the output and the conditions that were set. Thus, the PLL can be configured so that `DCO` is inverted without any change in the relationship between `DCO`, `FCO` and the data. Using PLLs would however require two FPGAs as none of the alternative FPGAs mentioned in section 3.2 have more than 8 PLLs. The final alternative is to use input delay elements. These are configurable through I/O-constraining. The delay can be set to D+6.3 ns where D is the intrinsic delay in the I/O [18]. For the target frequency of DCO, the delay needed to invert the signal would be:

$$\frac{1}{2 \times f_{DCO}}$$

which is about 7.1 ns. Hence, this option depends on D.

4.1.4.2 DDR-fabric

Figure 4.7 shows an alternative way of performing DDR capturing without inverting `DCO`. Rather than configuring the MSIOs to do the capturing, they are used as ordinary inputs. Two DFFs are implemented in place of the DDR_capture block in figure 4.2. The timing is shown in figure 4.8. This works by combining the shifted data with the latest data on the DDR capture block outputs. An inverter is used on the DFF that captures falling-edge data. This will worsen the timing margins between `data_f` and `shift_f`. The timing margin is already tight as the two DFFs are triggered by opposite clock edges. Unlike the previously discussed design with an inverter in section 4.1.4.1, this is the only path that is affected by the additional delay.

Another alternative is to create the circuit shown in figure 4.9. Compared to the DDR-macro, the latch is replaced by a falling-edge DFF. As the timing shows in figure 4.10, this implementation will work by combining the shift registers with all sequential outputs. This also uses an inverted version of DCO on only one DFF. It does not have any advantages timing-wise compared to the circuit in figure 4.7. In fact it has a slight disadvantage as it uses one additional DFF per capture block.

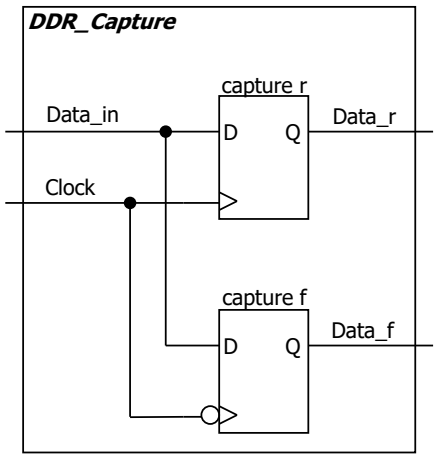


Figure 4.7: Alternative DDR capture logic

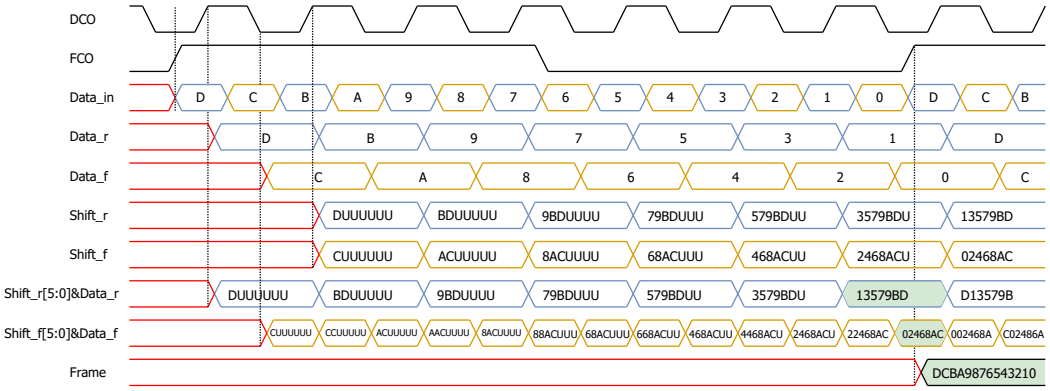


Figure 4.8: Alternative capture timing-
The blue lines represent data captured on rising edge. The falling edge data is represented by orange lines. Green fill marks that all bits are captured before FCO is asserted.

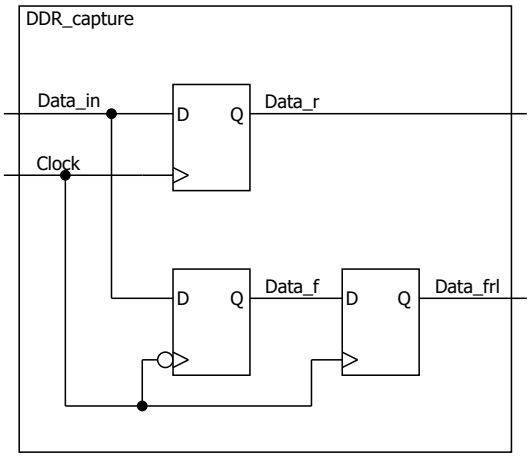


Figure 4.9: Macro circuit implemented in fabric

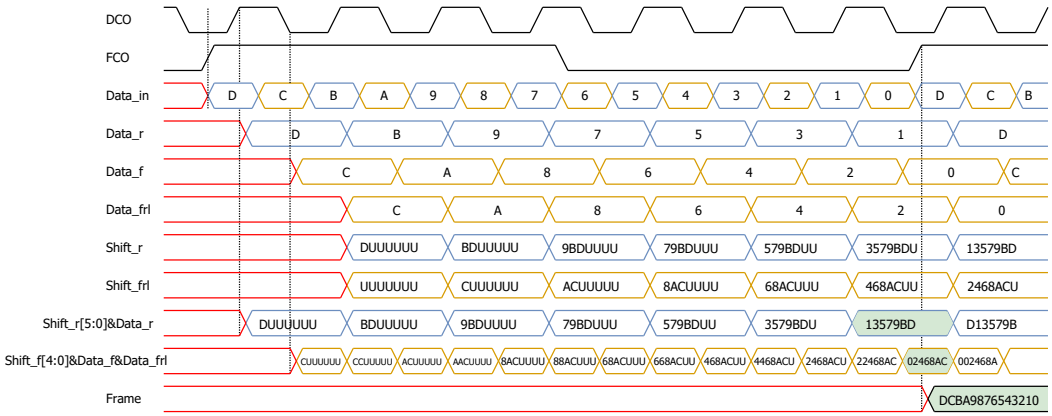


Figure 4.10: DDR_macro circuit implemented in fabric timing.
The blue lines represent data captured on rising edge. The falling edge is represented by orange lines. Green fill marks that all bits are captured before FCO is asserted.

4.1.5 Preliminary conclusion

Using SERDESIF[2] or PLLs require multiple FPGAs. This is of course an option, but it would increase the footprint of the PCB which should be as small as possible. Therefore, the other methods are considered to be better alternatives. The designs using DDR-MSIOs in combination with an inverted `DCO`, and the DDR-fabric using two flip-flops are considered to be the best of the custom methods. Their timing is depicted in figures 4.5 and 4.8 in the previous section.

Critical timing paths in figure 4.5 are on the bit capture in the DDR_blocks, and the last captured bit before the frame is capured. Due to the skew on `DCO` from the inverter, this is worsened. In the DDR_block, hold time violation is of concern for `data_r,` and also the latch `data_f`. This is caused by the delayed capture of their inputs. On the last bit before `FCO` is asserted, the delayed capture on `data_frl` may cause a setup time violation on the frame capture register. This can be mitigated by delaying `FCO` by e.g $T_{DCO}/2$. The delay can be added either by buffers or input delay, as mentioned in section 4.1.4.1.

The design using two DFFs in fabric has its critical path on `data_f`. This can cause a hold time violation in the falling-edge-capture DFF, and a setup violation in the frame capture register. The latter can be mitigated by delaying `FCO` by e.g $T_{DCO}/8$ by the same means mentioned in the previous paragraph. Since the target frequency is relatively low, and by applying proper timing constrains, both designs will probably work.

It should be mentioned that the skew from the inverter is one of many variables that affect timing margins. Process variations will introduce random skews. Environmental changes in temperature can cause a drift where the skew changes over time. High-frequency environmental variations can cause jitter on clock edges. All these variables must be accounted for. It is therefore important to verify that timing requirements are met. Microsemi provides a timing analysis tool that analyses timing with four-corner analysis, where process, voltage, and temperature are the variables. [10, 40]

[2]Due to certain restrictions, it is not certain if this can be configured to work against AD9257.

4.1.6 Realized readout design

The custom made DDR-fabric, two-DFF design was used in all systems that will be explained in later chapters. This section describes the structural design of the readout logic.

4.1.6.1 Fabric DDR-capture and deserializer

The top-level structure of the readout logic for one AD9257 can be seen in figure 4.11. In addition to the serial data and clock inputs, and the deserialized data outputs, a synchronous active low reset that can clear all the internal registers is included. For multiple ADCs, this structure can be instantiated multiple times in a top level file. The internal structure is shown in figure 4.12. It is comprised of three structures which are explained below.

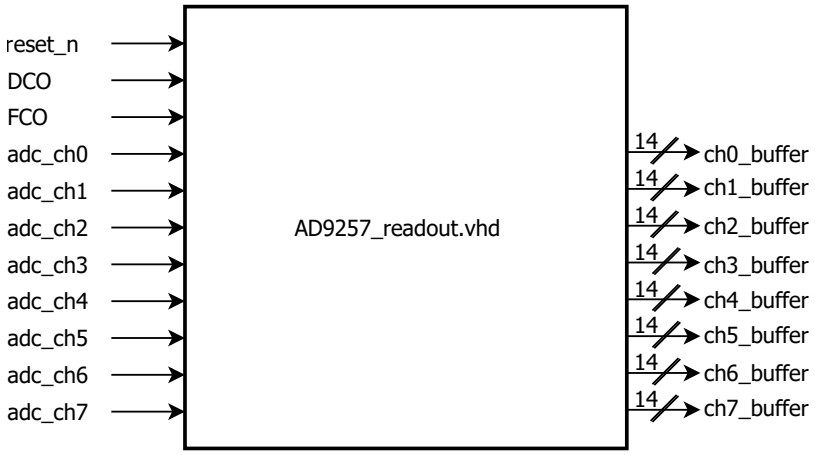


Figure 4.11: Readout logic top

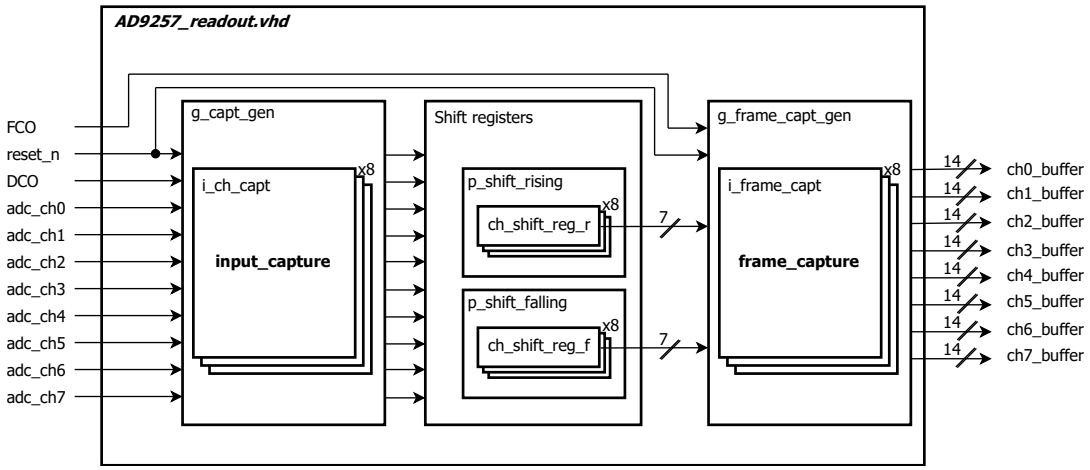


Figure 4.12: Readout logic internal

Input_capture
`Input_capture` is instantiated for all channels inside a generate statement. Each

instance contains the DDR capture logic in figure 4.7. If it is decided to change the method of DDR-capture at a later stage, it is only necessary to replace this instance.

P_shift_rising/falling
`P_shift_rising` and `p_shift falling` create shift registers for all channels by using for-loops. These processes are located in the top level.

Frame_capture.
`Frame_capture` is instantiated for all channels inside a generate statement. Each instance latches readout data from each channel into 14-bit parallel registers on the rising edge of FCO.

The combined register transfer level (RTL) structure is shown for one channel in figure 4.13.

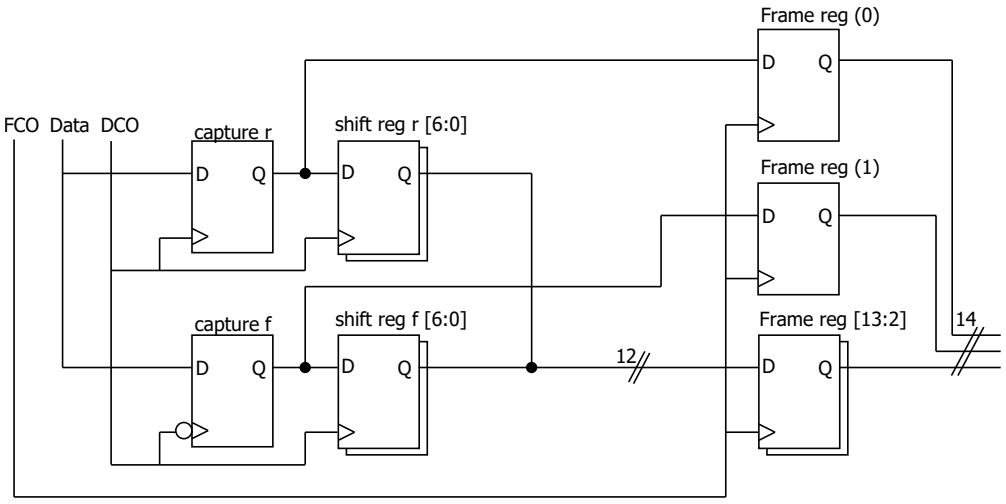


Figure 4.13: Logic for input capture, shifting and frame capture

4.2 AD9257 control logic

AD9257 has accessible registers which control certain settings. Having the possibility to change settings of the ADC can be useful. As an example: writing to register address 0x0D lets the user set test patterns on the ADC output. In this case, the analog front end is internally disconnected from the outputs. It is also possible to change the phase of DCO. If e.g. environmental variations cause the phase to change during operation, control logic that corrects this by adjusting the phase could be implemented. The register control section is accessed by a three wire SPI interface, where data in and data out are combined onto a single, bidirectional pin.[24] In the following sections, different methods of implementing an SPI master is discussed. Table 4.2 lists the methods that were considered.

4.2.1 SPI slave considerations

The general transfer protocol is divided into two phases: instruction phase and data phase. The instruction phase consists of transferring 16 bits from master to slave. The

Table 4.2: Methods of implementing an SPI master

Control method	Implementation
Custom fabric master	Soft
Microsemi CoreSPI	Soft
Microsemi MSS SPI	Hard

instruction contains data telling the slave whether the data phase is a master read- or write-operation, how many bytes of data to transfer, and the memory address. Once the instruction has been decoded, the data phase is executed.

Data can be transferred to the ADC in different ways. A transfer is initiated by setting the chip select bar (CSB) signal low. This tells the state machine in AD9257 to process the signals on the serial clock pin (SCLK), and on the serial data I/O (SDIO). The slave captures data on the rising edge of SCLK, while it puts data on the bidirectional pin SDIO on the falling edge of SCLK. SCLK has a minimum period of 40 ns [24]. I.e. the maximum frequency is 25 MHz. If multiple SPI slaves are used, as is the case in this project, CSB must be stalled at logic 1 when there is no active transfer.

Two of the instruction bits decide the number of data bytes to transfer in the data phase. The options are one, two or three bytes, or streaming. The latter option transfers bytes continuously until CSB is asserted. For all but streaming, CSB can be set to 1 between each transferred byte, or remain low for the entire transfer. This also applies for the instruction phase. Between bytes, the register address is incremented automatically. The latter applies to all modes. Figure 4.14 shows the timing of different transfer modes.[41]

4.2.2 SPI master specifications

Changing settings or resetting AD9257 is not functionality where speed is critical. At this stage in the design process, this system will be used to characterize the controllability of AD9257. The minimum requirements of the SPI master is as follows:

- Transfer signals to and from the SPI slave at a rate of no more than 25 MHz.
- Load data onto SDIO on SCLK falling edge.
- Register data from SDIO on SCLK rising edge.
- Transfer of data between master and slave must support the minimum:
  - Transfer one byte of data in the data phase.
  - Transfer instruction and data one byte at the time, stalling CSB at logic '1' between each transferred byte.

4.2.3 Custom SPI master in fabric

Figure 4.15 shows the internal structure of an SPI master that can be implemented in the FPGA fabric.

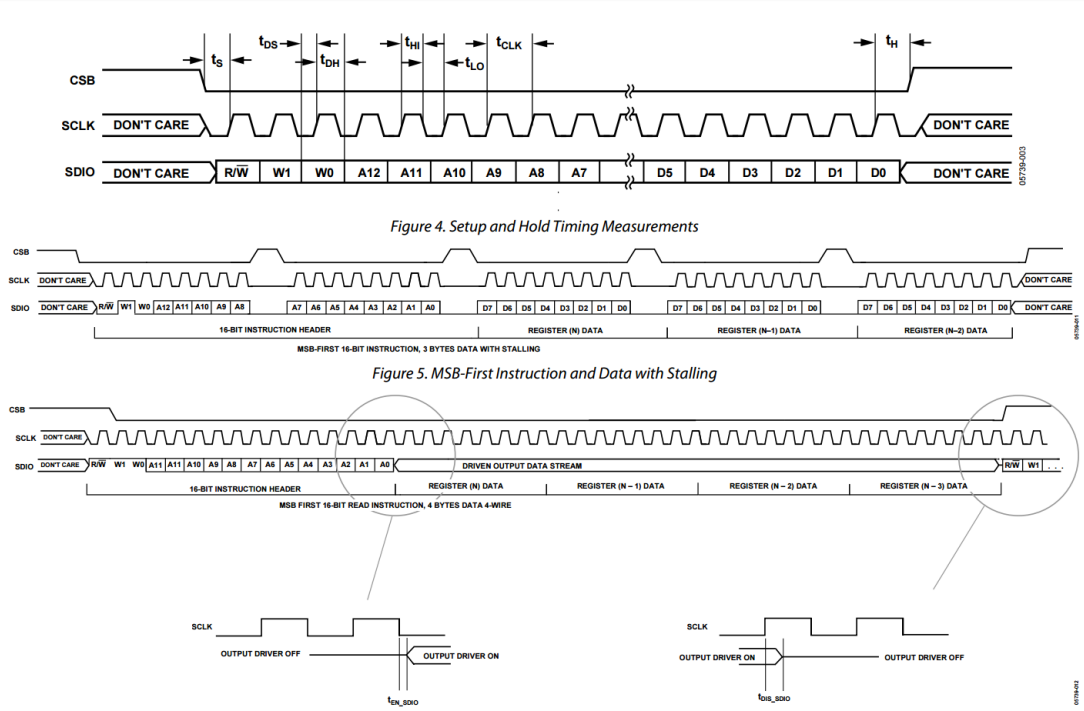


Figure 4.14: AD9257 SPI interface timing diagrams[41]

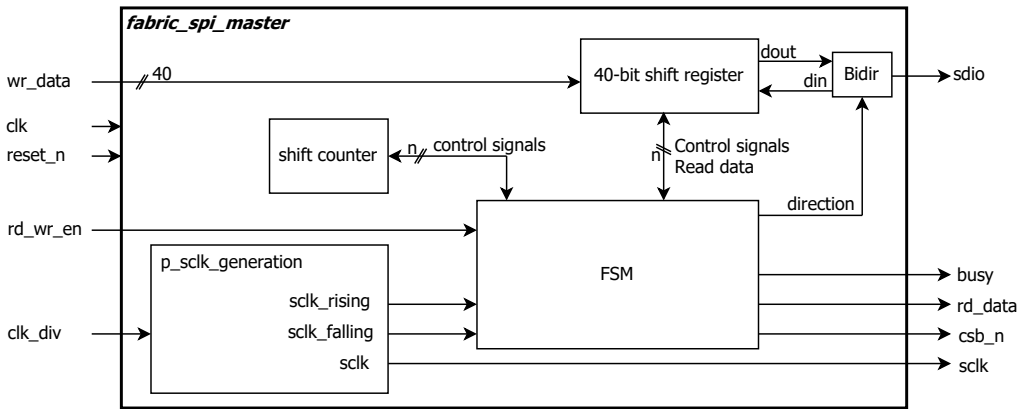


Figure 4.15: Fabric SPI master. Note that `bidir` must be an IO as high impendance can not be implemented in the fabric

Data input `wr_data` is loaded with up to 40 bits. This is equivalent to one instruction and three data bytes. `Clk_div` sets a factor that decides the frequency of `SCLK`, which is generated in `p_sclk_generation`. When these have received valid data, `rd_wr_en` can be asserted for at least one period of the system clock to enable a transfer. It must then be deasserted. A 40-bit shift register shifts data in from the `din` input when reading from the slave, or shifts data onto `dout` when writing data to the slave. A counter is used to keep track of the shifting process. When a transfer is in progress, `busy` is asserted to notify the master control system.

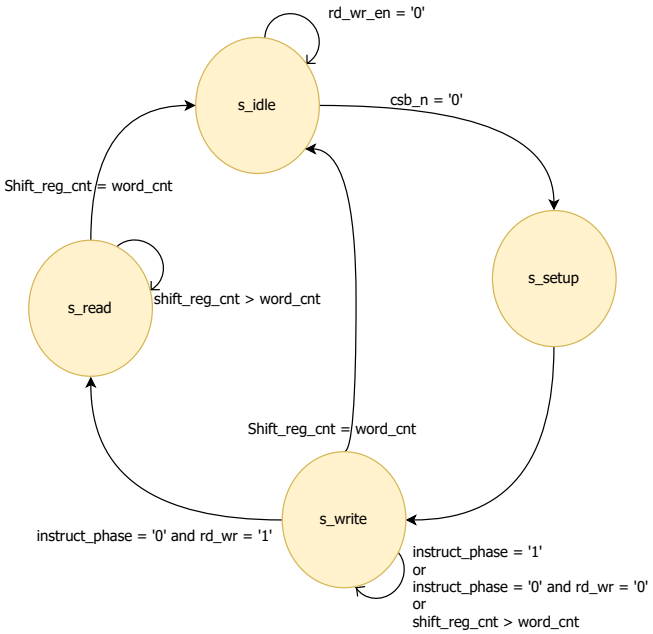


Figure 4.16: Fabric SPI master FSM

A finite state machine (FSM) is used to control all operations. It is implemented in one process which is sensitive to a system clock. The state diagram is shown in figure 4.16. The FSM is initially in the `s_idle` state. In this state, no transfer is active. When `rd_wr_en` is asserted, the state is changed to `s_setup`. In this state, data is loaded into the shift register, `SCLK` generation is enabled, `CSB_n` is de-asserted and `busy` is asserted. Certain instruction bits are loaded into internal registers to be used in following states. An instruction signal is also asserted as the first operation is writing the instruction to the slave. The next state is then directly changed to `s_write`.

After the instruction is shifted to the slave, the instruction signal is deasserted, and data is written to the slave if this was instructed. If a read from the slave was instructed, the state is changed to `s_read`. Here, data is shifted into the shift-register until the instructed number of bytes have been received. Shifting and next-state are controlled by a counter. Once the transfer is complete, the state is changed back to `s_idle` where registers are set to default values. `Busy` is deasserted, telling the controller that the transfer is complete.

The main advantage of creating a custom SPI master is that the interface from the controller is custom. It doesn't have to conform to any bus standards, as for example advanced peripheral bus (APB), where a 40-bit data bus might not be an option.

Another advantage is that it is not technology dependent, so it can be used on different types of FPGAs. On the other hand, the functionality that controls the actions of the master must be implemented in the fabric since a CPU uses standard bus interfaces. A software controller could be less complex to implement in some cases, but at the time of writing, this is only a speculation.

4.2.4 CoreSPI

CoreSPI is an intellectual property (IP) that is provided by Microsemi. This is a soft IP which is implemented in the fabric of the FPGA. It is supported by different Microsemi devices, including the ones listed in table 3.1. Figure 4.17 shows the top level structure. The core can be configured to fit various applications by setting generics to predefined values. Examples of these settings are setting it to act as slave or master, the SPI transfer protocol, bit size of the words that are transferred, FIFO depths and the frequency of SCLK. The FIFO memories buffer data that is written and read. This core has a 4-wire SPI interface, where data in and out have separate ports. The core has interrupt signals which signal certain events, such as if received data is ready to be read by the controller. Internal registers can be accessed via an APB interface. [42]

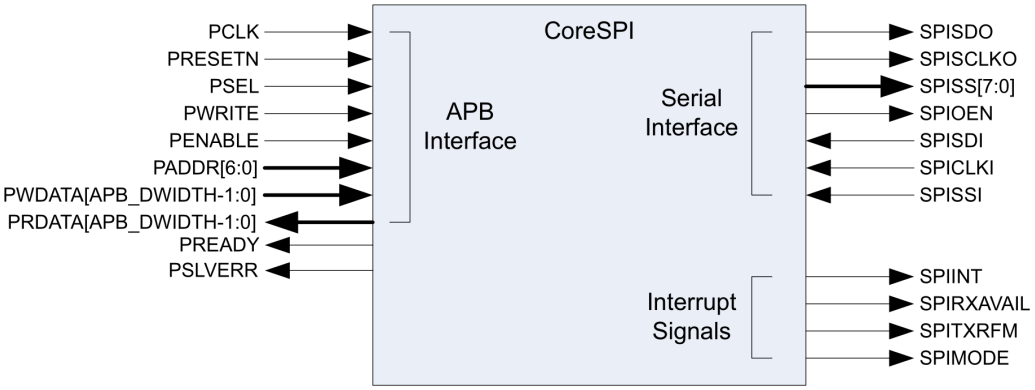


Figure 4.17: CoreSPI top level[42]

Setting the transfer protocol to Motorola mode 0, pictured in figure 4.18, will satisfy the specifications in section 4.2.2. To transfer the instruction and up to two bytes without stalling CSB between bytes, the FIFO depths can be configured to 1, and word size to 32 bits. If stalling CSB is desired, the fifo depth can be set to 3, 4 or 5, and word size to 8 bits.

All transfer related actions, e.g setting write data and enabling a transfer are set by changing content in internal registers via the APB interface. The controller that handles this can be software, executed by a CPU or a custom APB master in the fabric. This is more flexible compared to the custom master discussed in the previous section. It can also be assumed that the core is properly verified by Microsemi. On the other hand, this core requires an APB interface. Also, initiating a transfer requires five[3] operations. With the custom master, this can be achieved in two operations.

[3]This number is based on what was required in a testbench. It is possible that less operations are required.

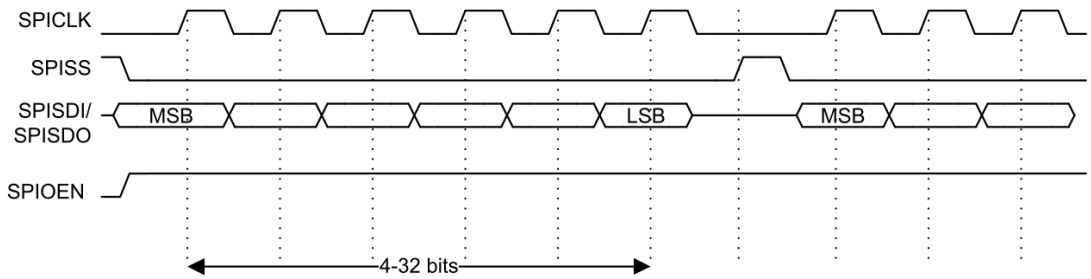


Figure 4.18: Multiple bytes transfer with SPI protocol Motorola, mode 0[42]

Table 4.3: MSS SPI write and read

SPI write	MSS_SPI_transfer_block(// Transfer 3 bytes
	&g_mss_spi0, // SPI to be instantiated
	tx_buffer1, // Data to send
	sizeof(tx_buffer1), // nr of bytes of type * array_index
	0, // indicates no reading
	0 // indicates no reading
	);

SPI read	MSS_SPI_transfer_block(// Transfer 2 bytes, read 1 byte
	&g_mss_spi0, // SPI to be instantiated
	tx_buffer2, // Data to send
	sizeof(tx_buffer2), // nr of bytes of type*array_index
	&rx_buffer, // Receive buffer
	sizeof(rx_buffer) // Size of receive buffer (should be 1)
	);

4.2.5 MSS SPI peripheral

This option is only available on SF2, as the others do not have an MSS. This is a hard-implemented SPI core. It can be accessed by the Cortex-M3 CPU through an MSS bus matrix. The functionality and configurations mentioned for coreSPI in the previous section 4.2.4 are similar to this core. E.g. Motorola mode-0 can be selected, the width of words that are transferred, and the number of bytes to transfer is configurable. Transfer-related settings are also set via an APB interface. This core also has a 4-wire SPI interface. These pins can be set to dedicated I/Os or connected to the fabric where it can interface with custom logic, or routed to I/Os. [43]

Microsemi provides firmware for this core. Using it is relativly straight forward. The functionality of the drivers are well documented in the driver file, as well as in a document that is supplied with the firmware. To use the core, the supplied firmware must be included as headers in the application software. The frequency of SCLK, master or slave mode etc. must be initialized and configured before it is used. This is equivalent to the configuration set by generics in section 4.2.4. Different functions for transfer are predefined. One example that meet the specifications in section 4.2.2 is to use the function shown in table 4.3. [43]

`Tx_buffer1` is an array where the instruction and data is stored, while `rx_buffer`

is where read data is stored. When writing, the function makes the SPI core write the number of bytes stored in `tx_buffer1`. When reading, the instruction stored in `tx_buffer2` is written to the slave. The following received data is then stored in `rx_buffer`. [44].

Using the method described in this chapter is the least complex way, as all that needs to be done is the creation of application software. It does however require the FPGA to be SF2. If this should be the case, it is a good alternative. Inclusion of a CPU also requires an embedded RAM to run the software, and an embedded ROM to store the software incase the system must be reset. The included parts will probably increase the power consumption as it runs at all times.

4.2.6 Preliminary conclusion

The choice of the method to control AD9257 strongly depends on which FPGA is chosen. It also depends on what needs to be configured, and how often the configuration memory must be checked or altered. The previously discussed methods offer different advantages. Thus, at this stage in the design process, no conclusion can be made.

4.2.7 Realized SPI master

All methods were realized to a certain level. The custom master was designed at an early stage in the design process[4]. The MSS SPI is used in all physical test and verification systems due to, amongst other reasons, the ease of implementation. This is explained further in chapter 5. Microsemi has not made a full behavioral model of the MSS SPI [43]. Therefore, coreSPI was used for computer-aided testing and verification. As the main functionality that need to be tested is that the chosen SPI transfer protocol is correct, this was considered to be an acceptable solution.

[4]Due to practicalities, the other methods were used in later design stages. Modifications, an proper testing must be done if this should be the final choice.

CHAPTER 5

Test and verification systems

This chapter describes the different systems that were designed for testing and verifying the digital designs described in chapter 4, as well as the actual AD9257. Verification was of great focus in this thesis. Therefore, everything that was realized was tested at all stages of the design process. To better understand the presented structures of the VHDL designs, and design flow of the systems that were implemented on the SF2-dev-board, "A.2: VHDL design", "A.3: Design flow", and "A.4: Verification" in Appendix A: Method should be read prior to this chapter.

5.1 Introduction

In order to test the digital designs in the previous chapter against an actual version of AD9257, the system illustrated in figure 5.1 was set up. A testboard containing the ADC was made to connect to an SF2-dev-board. Figure 5.2 shows the development board. A top level file containing the digital designs was made to program the FPGA. Also included in the top level, were the MSS and MSS peripherals necessary to establish connection to a host computer. Software was made so that the entire system could be controlled in a terminal window on the computer. The software functionality includes test and verification procedures, access to the memory of AD9257, and a visual representation of the readout data.

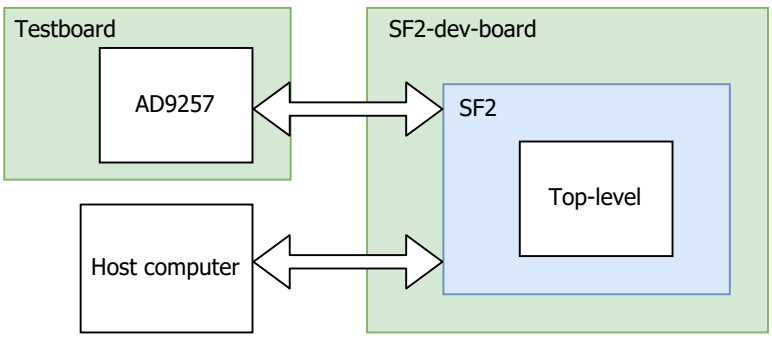


Figure 5.1: Final test-and-verification system

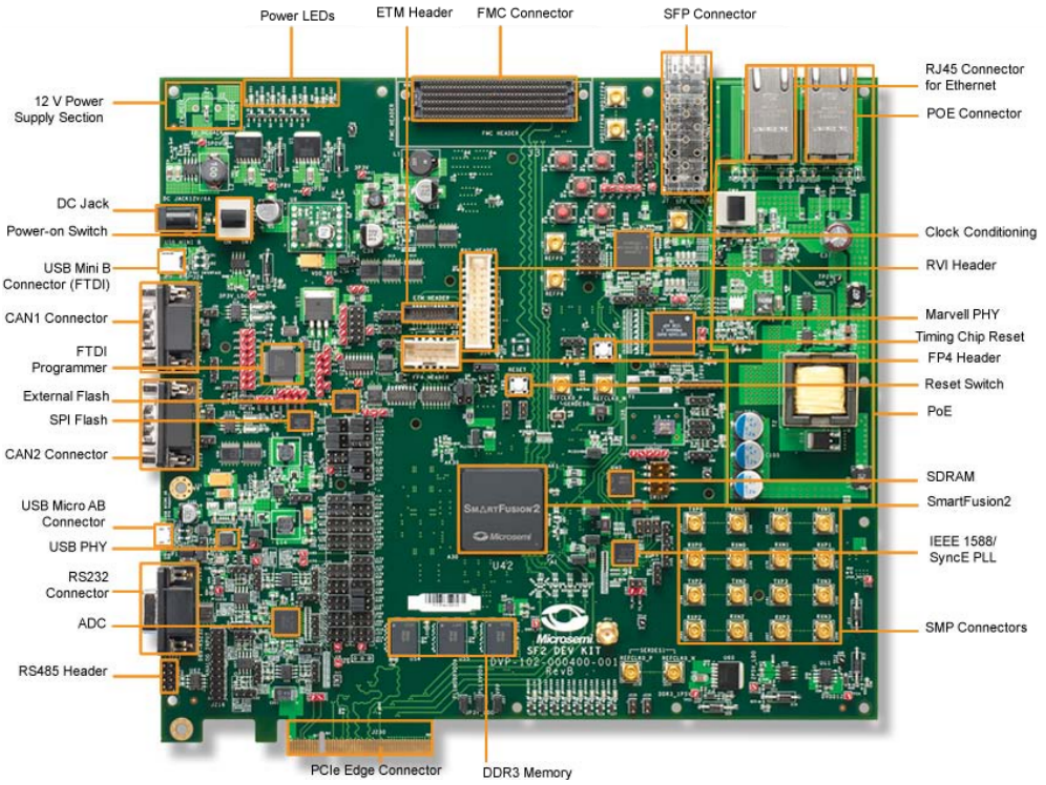


Figure 5.2: SmartFusion2 developement kit board

In order to create a error-free SoC, test-and verification systems were made from the start of the design phase. This allowed evaluation of the SoC in all design stages until the final SoC was ready. The testbenches were structured in a way that made it possible to use the same test and verification sequences for all systems. Table 5.1 lists the systems that were made.

Table 5.1: Verification systems

Test and verification systems
Development testbench
FPGA-internal SoC & model
FPGA-loopback SoC & model
AD9257-testboard Soc & testboard

The development testbench was made in order to test and verify the digital designs from the start of the design process. FPGA-internal SoC was made to test the digital designs on the SF2, and to develop the software. FPGA-loopback SoC is a similar system. AD9257-testboard SoC is the system that is made to interface with an AD9257 testboard. Before any of the SoCs were implemented on SF2, they were tested and verified in a computer-aided testbench. After they were implemented, they were tested and verified in a physical, software testbench. In order to test in an environment that was as close as possible to the real-life application, a VHDL model of AD9257 was

developed, and used in the test and verification systems.

The general difference between the computer-aided and the physical testbenches in this thesis is shown in figure 5.3. In the computer-aided systems, a VHDL test sequencer in a process acts as the controller that performs all the tests. As mentioned in section 4.2.7, coreSPI is used in computer-aided testing as the MSS SPI does not have a behavioral model. This system is used to verify pre-synthesis-, post-synthesis- and post-layout-HDL. In the physical test systems, a test sequencer written in C, together with the SF2 MSS, work as the controller. Here, the MSS SPI peripheral is used instead of coreSPI.

In addition to the readout- and control- logic, testboard-related functionality was added in all systems. The remaining parts in this chapter present this functionality, and more details about each of the systems that were designed. The testing that was performed, and the results of these tests are presented in chapter 6.1. The following sections in the chapter should be read in the order they are listed.

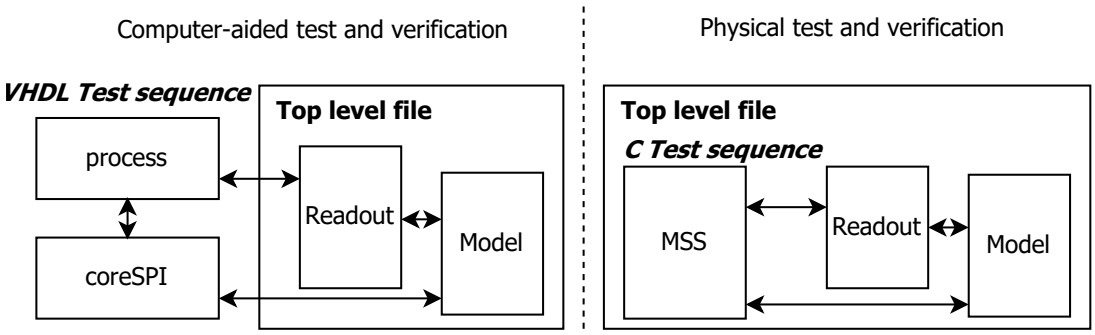


Figure 5.3: Structural difference between computer-aided- and physical- testbenches

5.2 Development testbench

A testbench framework was set up at the very beginning of the design stage. This was used to gradually test and verify as functionality were added throughout the design phase. Figure 5.4 shows the final testbench structure. All tests are implemented and performed sequentially in the test sequencer. The general test method is to enable a test from the test sequencer, awaiting the response from the devices under test, and then comparing the response with the expected result. The tests that were performed is explained in chapter 6.

The dashed boxes represent packages. These are used to increase readability by information hiding, and to include functionality. Universal VHDL verification methodology (UVVM) utility library is an open source test-bench-verification infrastructure that is developed by Bitvis[1]. The library contains functionality that eases the creation of a structured testbench. Examples are verbosity-controlled logging, setting and handling alerts for test cases, and random generators. After a test sequence is complete, the log can be compared directly to the verification plan.

A bus-functional model (BFM) models the interface of a bus between a controller and the device being controlled. A BFM for APB3-bus was made between the test sequencer

[1] www.bitvis.no

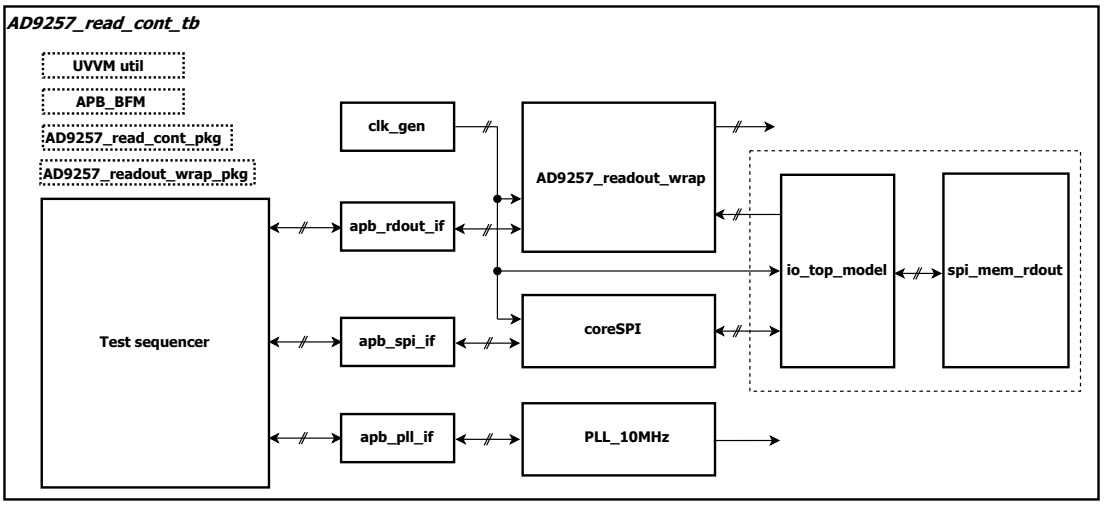
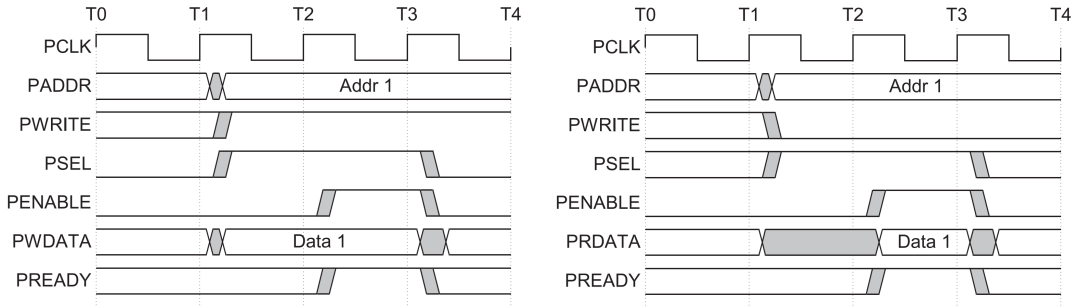


Figure 5.4: Testbench framework top



(a) Write (b) Read

Figure 5.5: APB3 timing diagrams[45]

and every device under test. Figure 5.5 shows the signals and timing of this bus. A premade BFM made by Bitvis was modified to act in accordance with the APB3 standard. APB3 was chosen because this bus was used between the CPU and its slaves in the FPGA-implemented systems discussed in sections 5.3, 5.4 and 5.5. A more accurate method would be to create a BFM where all slave interfaces connect to one data bus through a bus handler. This was however not prioritized. Instead, a bus handler was made for shared APB3-buses in the top level `AD9257_read_cont_tb`. The remaining packages contain custom procedures and constants. Examples are `spi_write()` which is used when writing data via coreSPI, and memory map addresses. The following sub-sections present the different components in figure 5.4.

5.2.1 HDL model of AD9257

A VHDL model of AD9257 was designed for test and verification purposes. The advantage of doing this is that the entire testbench will be a model of the real-life application. The model was made synthesizable so that it could be used in both computer-aided testing and physical testing on an FPGA, prior to testing against an actual AD9257.

5.2.1.1 Specifications

Making a complete model of AD9257 would be very time consuming. As it is the SPI- and readout-functionality that needs to be verified at this stage, only the SPI slave, memory, and digital back-end is needed. Since it is possible to set the output data from AD9257 by writing to certain memory addresses via the SPI interface, this functionality was also included. The advantage of this is that the stimulus to the readout logic can be changed, and since it is a direct response to the use of the SPI interface, a change in output data can be used to verify the SPI-master. The minimum specifications that were set for the model are:

- It must contain the number of data and clock outputs equivalent to one AD9257.
- The clock and data relationship on the outputs should be exactly like the timing in figure 4.1.
  - As the target sample frequency is 10 MHz, the serial data outputs must have a frequency of 70 MHz.
- It must contain an SPI slave with the following functionality:
  - Receive one byte of data following the transfer of the instruction.
  - Receive instruction and data one byte at the time, while CSB is stalled at logic '1' between each transferred byte.
- It must contain a memory with memory addresses as specified in the datasheet [24] which can be accessed through the SPI interface.
- It must at least be possible to set user-defined output patterns.
- All parts of the model must be synthesizable.

5.2.1.2 Model top-level structure

Figure 5.6 on the next page shows the structural top level of the model. `Spi_mem_rdout` is the synthesizable part of the model, while `io_top_model` models some of I/Os of the ADC. The reason for structuring it this way will be made clear in the following sub-chapters. All internal logic can be reset by an active-low signal. `Clk` and `dcoX2_0_deg` come from the same 140 MHz clock source. These are used to trigger the synchronous circuits. Note that `DCO` and `FCO` is not generated inside the model, as was specified. They were for practical reasons generated by the clock conditioning circuitry (CCC) that also generates the system clock in the physical test-and-verification systems. To create a more accurate model, they should have been generated by having a separate CCC in `spi_mem_rdout`. This was however not prioritized.

5.2.1.3 Model internal structure

`Spi_mem_rdout` contains all the functionality of the model. The internal structure is shown in figure 5.7. `AD9257_spi_model` contains the SPI slave. This connects to `AD9257_memory`, which has the same memory map as the actual AD9257 [24]. It also

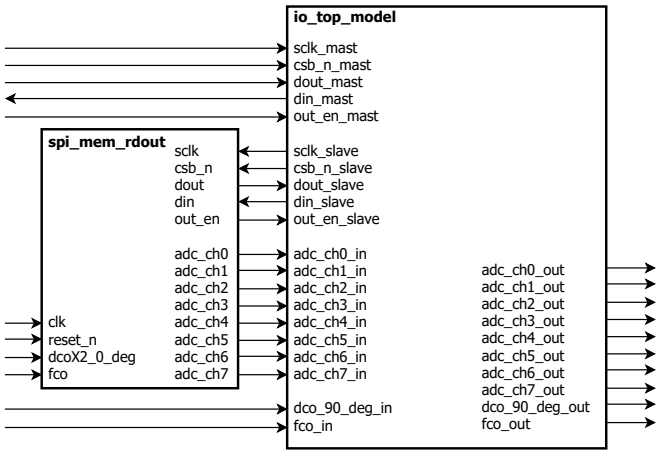


Figure 5.6: ADC model top

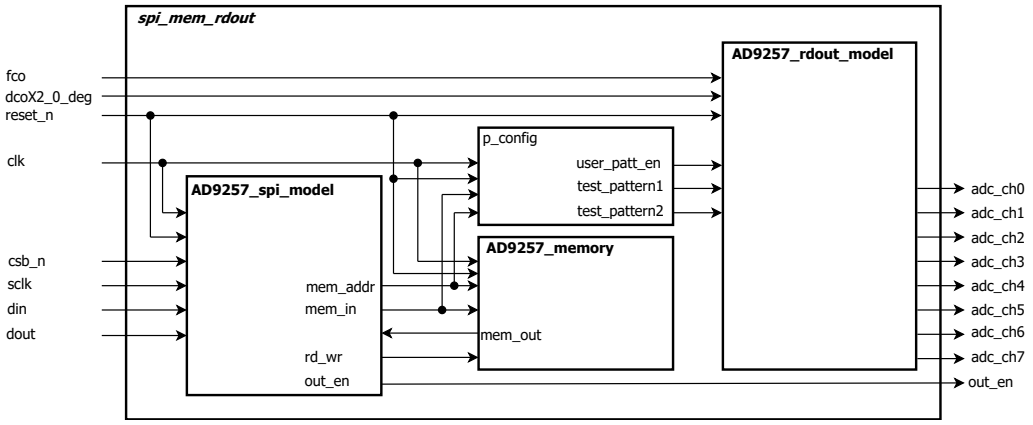


Figure 5.7: spi_mem_rdout

connects to `p_config`. This process chooses output patterns based on the content that was written to the memory. These patterns and a trigger signal are connected to `AD9257_rdout_model` where the patterns are loaded onto the ADC outputs. The following sections provide more detailed informaion about the different modules.

AD9257 SPI slave

The internal structure of `AD9257_spi_model` is similar to the structure for the SPI-master discussed in section 4.2.3. A shift register is used to shift data in and out, and an FSM acts as a controller. As this is an SPI-slave, it receives `SCLK` and `CSB`. The FSM is sensitive to the system clock which runs at a higher frequency than SCLK. Enable pulses are generated on the rising and falling edge of SCLK to ensure that the data is transferred at the correct rate. The state transition diagram is shown in figure 5.8, and is explained below.

s_idle
The FSM is idle when no transfer is initiated from the master. All signals of importance are set to their default values in this state. If `CSB` is de-asserted, it means that a transfer is about to occur. This changes the state to `s_instruct`.

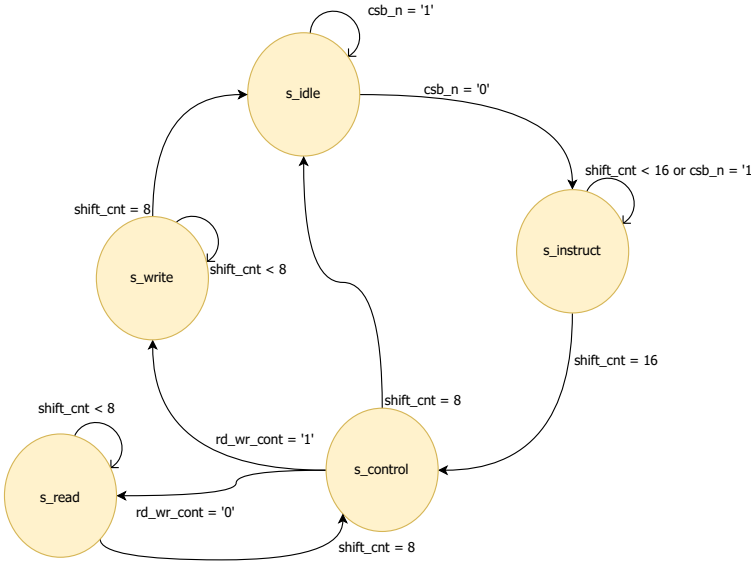


Figure 5.8: AD9257 SPI model state machine

s_instruct
In this state, a two-byte instruction is shifted in from the data input. Shifting is performed each time the pulse made on the rising edge of `SCLK` is asserted. A counter controls the next-state action. It remains in this state until the entire instruction has been received. Once this is complete, the state changes to `s_control`.

s_control
Here the instruction is decoded. The actions and next-state are based on whether a master read or write are instructed. If it is to write to the slave, it goes directly to `s_read`. In this case the state will be changed back to `s_control` once the read is complete. The content that is received from the master is stored in the received memory-address location before the state is changed to `s_idle`.

If the master instructed to read from the slave, memory data from the address contained in the instruction is loaded into the shift-register. MSB of the data is put directly on the output so that the master can sample data on the next rising edge of sclk. Since the data I/Os goes to a bi-directional I/O, an output-enable signal is asserted. The state is then changed to `s_write`.

s_write
As long as the FSM is in this state, data is shifted out when the pulse made on the falling edge of `SCLK` is asserted. A counter controls the next-state action. An output-enable signal is generated, and can be used to control the direction of data flow in a bidirectional I/O. Once complete, the state is changed to `s_idle`.

s_read
As long as the FSM is in this state, data is shifted in when the pulse made on the rising edge of `SCLK` is asserted. A counter controls next-state action. Once complete, the state is changed to `s_control`.

AD9257 internal memory

This component models the internal registers of AD9257. Accessible registers are defined in the datasheet [24]. It has input ports to receive an address, a read/write enable signal, input data, active-low reset, and the system clock. It also has one memory-data output.

If the master reads from the memory, the read/write signal will stay low. Data from the memory address is placed directly on the output. If the master writes to the memory, the read/write signal will pulse for one period of the system clock. This ensures that no memory locations are overwritten unless it is instructed. Data is then stored in the memory address at the rising edge of the system clock.

p_config

When the master writes to certain addresses, the output data from the model is changed. The functionality that handles this is made in `p_config`. Based on received address and data, either predefined or user-defined patterns are set. The predefined patterns are similar to the real predefined patterns stored in AD9257. The process of setting user defined patterns is made to be equal as for the actual ADC. Thus, two 14-bit patterns can be made from content that is stored in dedicated memory addresses. The user can choose to only set one pattern as output data, or both patterns as back-to-back outputs. [41]

AD9257 output

The internal structure of `AD9257_rdout_model` is illustrated in figure 5.9. The user patterns are loaded into a shift register where the output of the last flip-flop is fed back to the input of the first flip-flop. This makes the pattern repeat itself continuously. If `test_pattern1` and/or `test_pattern2` is changed, the previously loaded 14-bits are shifted out before the new patterns are loaded. This is controlled by a counter. The two input test patterns can be equal or not. If they are equal, only `test_pattern1` is loaded into the register. If they are unequal, the patterns are loaded back-to-back, starting with `test_pattern1`. `User_patt_en` is the signal that controls this. The clock of which the shift register is sensitive to must be 2×`DCO`for this model to work. By using this frequency, all the bits can be shifted out on the rising edge of the clock. It must also have the same phase as `FCO`. If this is the case, the timing in figure 4.1 is achieved. If `reset_n` is asserted, all DFFs are set to '0'. Functionality that makes sure that new data is not put on the outputs before the next rising edge of `FCO` is implemented, but not shown. In this way, the data is synced to `FCO` regardless of when the module is reset.

5.2.1.4 I/O model

The I/O interface between AD9257 and an FPGA was modeled as a separate component. The internal structure is shown in figure 5.10. The model only contains the I/O port types and pull circuitry of the SPI interface. Modeling the differential I/Os of the ADC

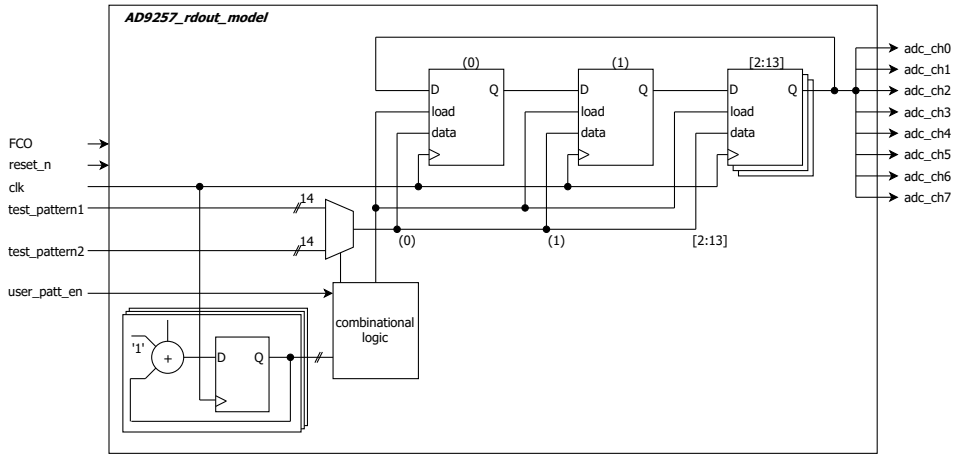


Figure 5.9: Readout model RTL

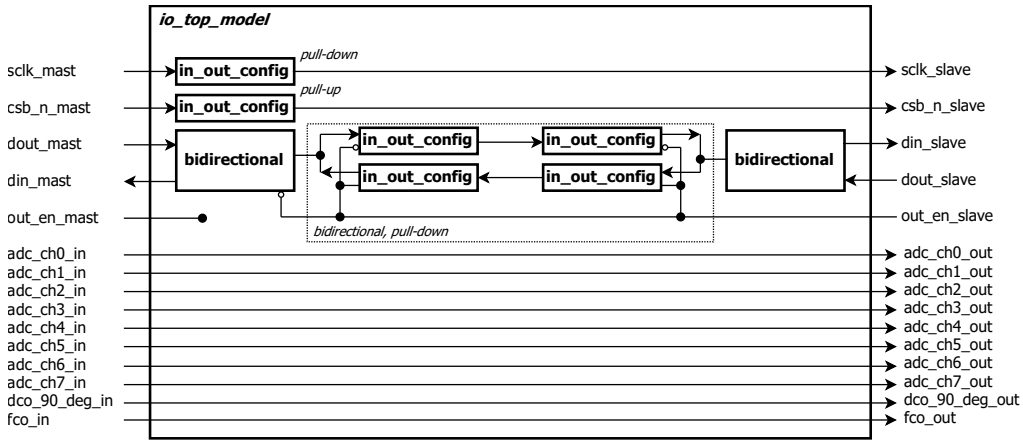


Figure 5.10: io_top_model

outputs were not included. The signals were despite this routed through the module in case this would be done at a later stage. This module was made separate from `spi_mem_rdout` because it is not possible to implement this functionality when all of the logic is connected internally in the FPGA. This is the case for the system that is presented in section 5.3.

`SCLK` and `SDIO` have internal pull-down resistors, while `CSB` has an internal pull-up resistor. `In_out_config` is a component that was made to model pull-up- or pull-down-behaviour. The type of pull is set by generics when the component is instantiated. `SDIO` is a bi-directional signal, while the others are unidirectional. The component `bidirectional` was made to act as a bidirectional port. An output-enable signal decides if data that is sourced to the component is put on the bi-directional line. The data output of the component is always connected to the line. To control the direction of data transfer, the output-enable signal from the SPI-slave, and an inverted version of said signal is used.

5.2.2 Clock generation

For practical reasons, the system clock, `DCO`, and `FCO` were generated by a CCC in the test systems that used the model internally on SF2. This was modeled by creating a behavioral model of a CCC. This was used in all computer aided testbenches. Ideally, this component should have been replaced with the HDL of the generated CCC, but this was not prioritized.

5.2.3 Readout wrapper

As presented in section 4.1.6, the component `AD9257_readout` captures and deserializes data from AD9257. So that the test sequencer can access the deserialized data, a bus wrapper was made where the readout logic is instantiated. The top level structure of the readout wrapper is shown in figure 5.11. The ports are composed of readout-logic ports, APB3-bus ports, AD9257-testboard control signals and a debug signal. The purpose of the control signals is made clear in section 5.6 where the circuits on the testboard are explained. Figure 5.12 shows the internal structure. The signal flow is explained for one channel in the following text. It is however valid for all channels as they are equal.

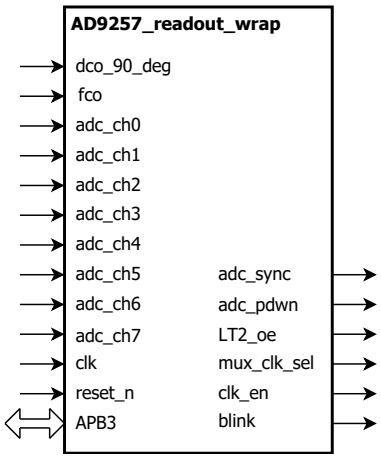


Figure 5.11: Readout wrapper top

`P_sync` synchronizes the readout data to the system clock before it is used any further. This is done by passing it through two cascaded system-clock-triggered DFFs. This reduces the probability of metastability when the signals cross clock domains. [10]

The synchronized readout data is fed to error-detecting circuitry `p_read_check`. If a change in readout data is expected, the new data is latched into a specific register. A register that signals the arrival of new valid data is then asserted. If readout data changes unexpectedly, it is latched into a specific error-data register. A register that signals the erroneous readout is also asserted. If readout data is latched into either register, they remain unchanged until `new_data_clear` or `irr_clear` are asserted. This ensures that the first instance of correct - or erroneous- data can be checked. In order for this to work, the circuitry must be notified before new data is expected. This is done by asserting `new_data_clear` before a new output pattern is set in the model. It is also possible to bypass this functionality, by asserting `manual_read`. In this case, the latest received readout daa is latched into the data registers.

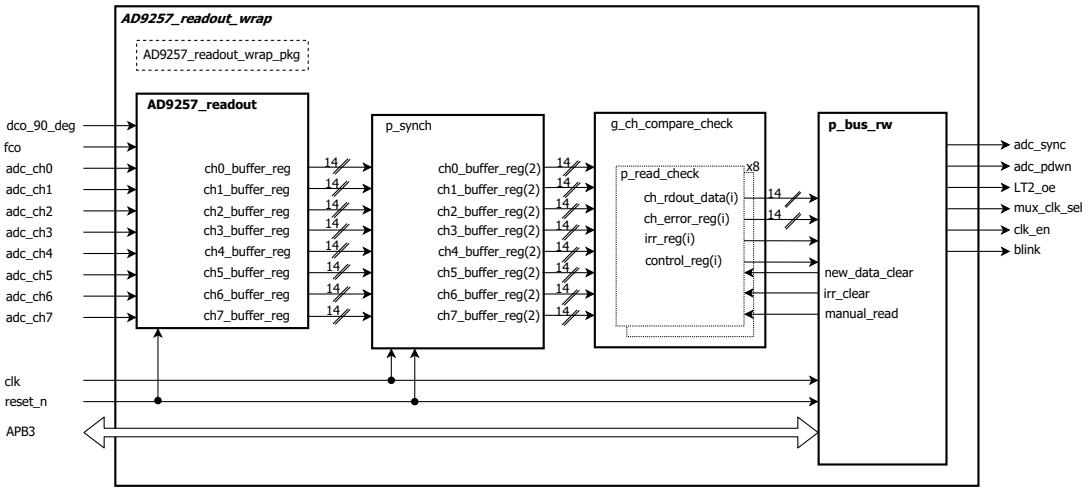


Figure 5.12: Readout wrapper internal

Everything mentioned above is accessed through the bus handler `p_bus_rw`. All of the channel-specific readout-data registers are assigned to specific addresses, while all channels share common control-signal registers. The control registers are made so that the APB3-master, i.e. the test sequencer and CPU can detect readout-and erroneous data by polling. The testboard control signals, and the debug signal `blink` are also controlled by writing to specific addresses. All addresses are defined in the package `AD9257_readout_wrap_pkg`. The bus handler complies with the APB3 transfer protocols depicted in figure 5.5.

5.2.4 Dynamic PLL

This component was included to generate an adjustable sampling clock that can be used by AD9257 on the testboard. The schematic of the testboard clock circuitry is presented in section 5.6.2.8. A PLL is located in a clock conditioning circuitry (CCC). The output frequency can be adjusted by changing the values in its internal registers. The registers are accessed via an APB3 interface. Figure 5.13 shows the top-level structure of the component. In addition to the APB3 ports, it has a reference clock input, an asynchronous active-low reset, and a powerdown input. Outputs consist of the clock, and a lock signal that is asserted when the output frequency is stable. The input reference clock, and default output clock are chosen when the component is generated in Libero SoC. The CCC generated for this purpose uses an RC-oscillator internal to SF2 as a reference clock, and has a default output frequency of 10 MHz.

Figure 5.14 shows the internal structure of the CCC. The output frequency is decided by a range of factors inside the CCC. These are changed when the content in their corresponding registers is altered. Examples of these factors are `RFDIV` and `FBDIV`. They decide the reference clock and feedback division factor, respectively. There are also restrictions on the minimum frequency into the PLL, as well as the maximum frequency out of the PLL. The output of the PLL can go through the GPD block seen in the figure, or it can bypass it. The GPD can also be used to reduce the frequency. [46]

There is no obvious way to know which factors to change, and how to route the clock through the CCC. When the component is generated in Libero SoC, a document that

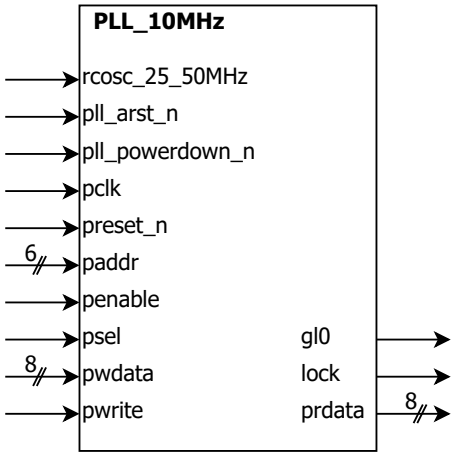


Figure 5.13: Dynamic CCC top

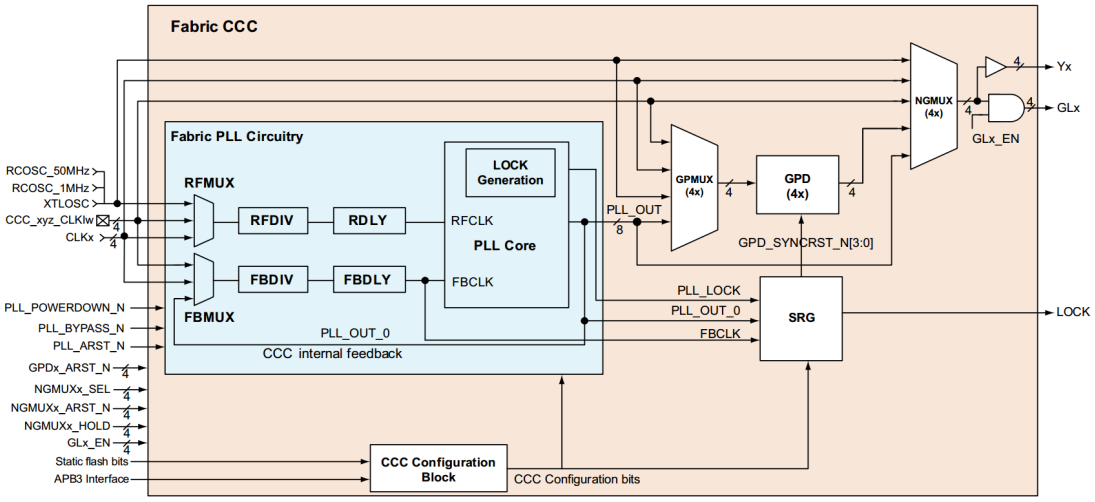


Figure 5.14: Dynamic CCC internal[46]

shows the content in each register is included in the folders. By generating multiple components, and comparing these, commonalities were found. Table 5.2 lists these registers and their values with respect to the output frequency. These values can be loaded into the CCC when any of the frequencies will be used. For other than the listed frequencies, a new component must be generated to get the proper register content.

5.3 FPGA-internal SoC

The system explained in this section was designed in order to test the readout logic on an FPGA before testing with an actual AD9257. This was possible since the model was made synthesizable. The system was also used during development of the software that is presented in section 5.7.

Figure 5.15 illustrates the implemented SoC. It contains a CPU, a RAM and the buses that are needed to execute the software. The SPI peripheral discussed in section 4.2.5 is used as the SPI master. A hard universal asynchronous receiver/transmitter (UART)

Table 5.2: CCC register-values in hex for different frequencies

Names\freq [MHz]	10	12	17	20	30	40	50	200
FCCC_RFDIV_CR	0x04	0x18	0x18	0x04	0x04	0x04	0x01	0x01
FCCC_FBDIV_CR0	0x01	0x0b	0x10	0x01	0x02	0x03	0x01	0x07
FCCC_NGMUX0_CR0	0x08	0x08	0x08	0x07	0x07	0x07	0x07	0x07
FCCC_GPMUX0_CR	0x47	0x47	0x47	0x58	0x58	0x58	0x58	0x58
FCCC_GPD0_CR	0x02	0x02	0x02	0x01	0x01	0x01	0x01	0x01
FCCC_PLL_CR7	0x05	0x05	0x04	0x05	0x05	0x04	0x04	0x02
FCCC_PLL_CR10	0x05	0x02	0x02	0x05	0x05	0x05	0x07	0x07
FCCC_GPD0_SYNC_CR	0x03	0x03	0x03	0x02	0x02	0x02	0x02	0x02

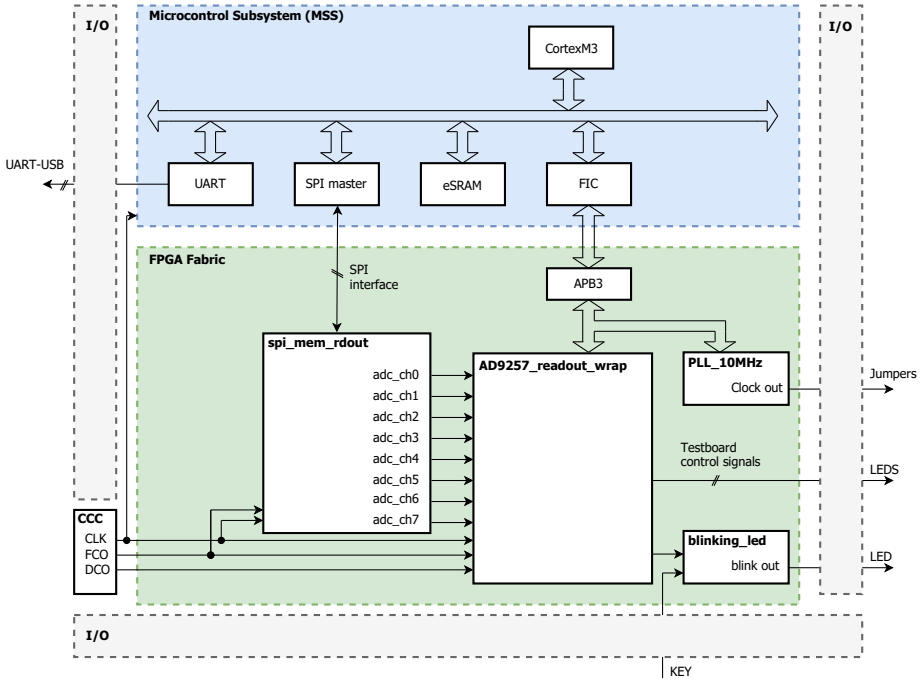


Figure 5.15: Illustration of SoC for FPGA internal

is included in the MSS. In this system it enables the creation of a user interface. It connects to an UART-USB converter external to SF2 on the dev-board. This is further connected to a mini-USB port.[47] This makes it possible to interact with the system from a host computer.

A CCC is used to generate a system clock, `DCO`, and `FCO`. The dynamic PLL `PLL_10MHz` that was explained in section 5.2.4 is also implemented. Its outputs are connected to jumpers on the SF2-dev-board. A debug component, `blinking_led`[2] was added and connected to a LED. This can be activated through a key on the board, or by writing to a specific address in `AD9257_readout_wrap`. All testboard control signals are also connected to LEDs. The CPU connects to `AD9257_readout_wrap` and `PLL_10MHz` through a fabric interface controller. This was configured to create an APB3-bus master in the fabric, as this is the bus interface in the two modules.

[2]This component was provided by Kjetil Ullaland (kjetil.ullaland@uib.no)

The following sub-sections present the system that was made for testing in Modelsim, and the system that was implemented on the SF2-dev-board. As explained in section 5.1, the difference between the two systems is that the system for Modelsim does not use the MSS. Internally, all signals connect in the same way. How to set up the project and control it from a computer is explained in Appendix C: Project setup.

5.3.1 Computer-aided-verification system

Figure 5.16 shows the structure of the testbench. `Io_top_tb` is the top level where all functionality except the MSS is instantiated. This structure was used when testing the system pre-synthesis, post-synthesis and post-layout. The latter two were created in Libero SoC with `io_top_tb` as the top level.

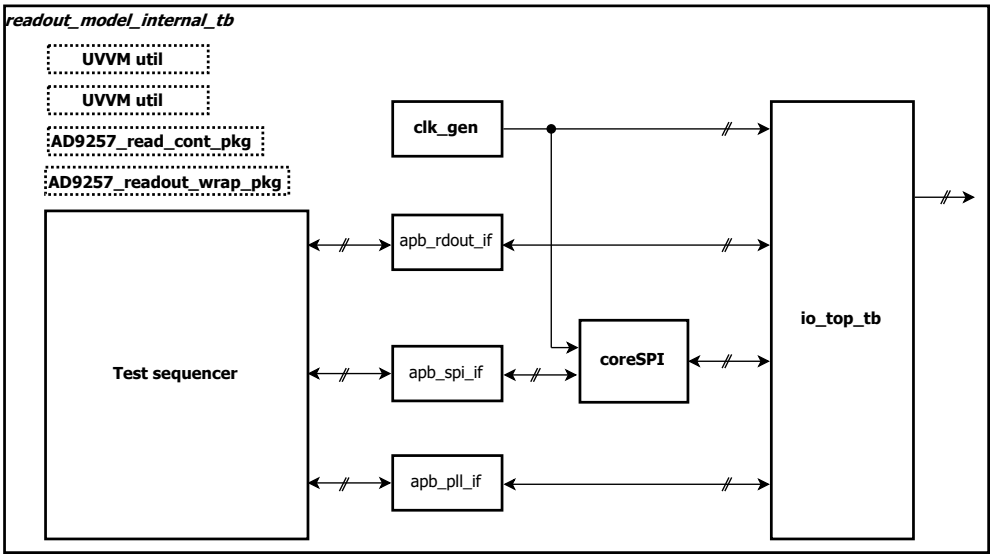


Figure 5.16: testbench structure for FPGA internal

Figure 5.17 shows the top level interfaces of `io_top_tb`. The system clock and an active low reset are connected to all sequential elements. `Blink_in` is a port for debugging. Asserting this generates a toggling signal on `blink_out`. `DCO` and `FCO` are used in the readout logic, and are generated in `clk_gen` along with the system clock. Note that the APB3-bus signals for `apb_rdout_if` and `apb_pll_if` are drawn as a single bus. This is because most of the signals from the APB3 master are shared between the fabric slaves. The exceptions are the data-read- and slave-select- signals.

Figure 5.18 shows the internal structure of `io_top_tb`. `Spi_mem_rdout`, which is the synthesizable part of the model, is connected directly to the readout logic through `AD9257_readout_wrap`. The clocks are distributed to the components on global nets. Hence the green wires. The gray components are Microsemi cores. `Rosc_25_50mhz` is the internal clock oscillator that was chosen as the reference clock to `PLL_10MHz` when it was configured. All I/O signals are routed through I/O-buffers. The clock output of `PLL_10MHz` is converted to a differential signal. The reason for this is explained in section 5.6.2.8. Compared to the illustration of the system in figure 5.15, the only difference is that the MSS is removed from this top level. The results from the tests performed on this system are presented in chapter 6.

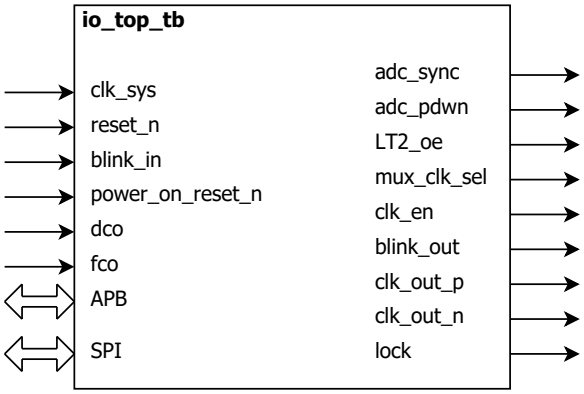


Figure 5.17: io_top_tb top level

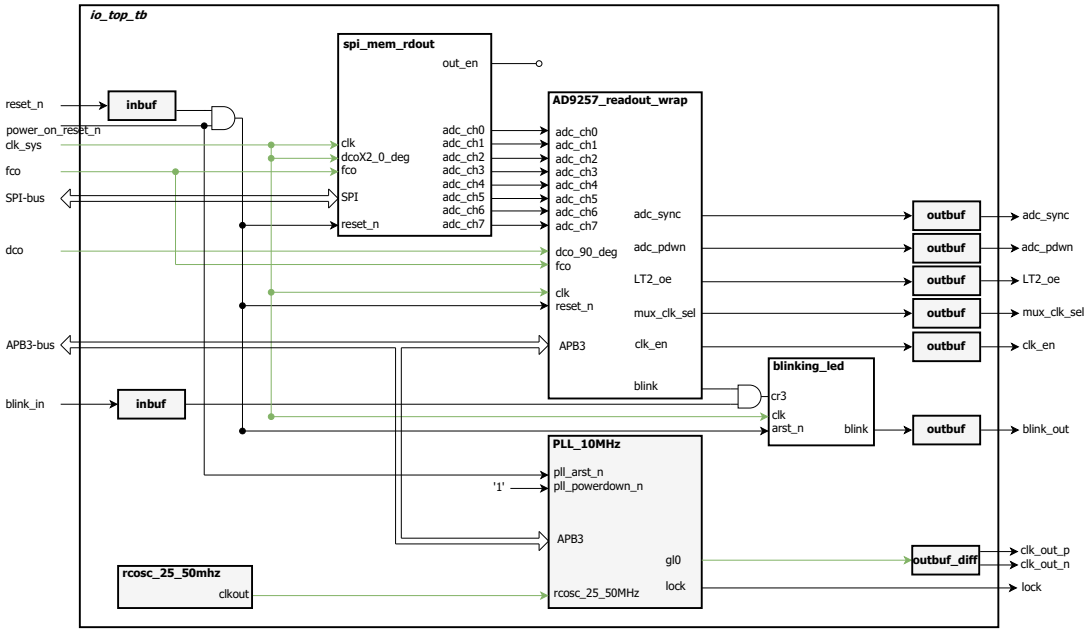


Figure 5.18: io_top_tb pre-synthesis

5.3.2 FPGA-internal SoC structure and design process

This section presents the structure of the system that was implemented for physical verification on the SF2-dev-board. Figure 5.19 shows the top-level interfaces to `io_top`, while figure 5.20 shows the internal structure with the MSS included. Compared to figure 5.18, all but the APB3-, SPI-, and UART-signals are the same.

The inputs consist of reset-signals `reset_n` and `devrst_n`, debug-signal `blink_in`, and UART-receive `mmuart_0_rxd_f2m`. `Devrst_n` is an active-low reset-signal that is active for some time when the board is powered on. `Reset_n` and `blink_in` are connected to keys. `Mmuart_0_rxd_f2m` is used to receive USB-data from the host computer. The outputs consist of the circuit board control signals, AD9257 sample-clock, debug signal `blink_out`, and UART transmit `mmuart_0_txd_m2f`. The Control - and debug- signals are connected to LEDs, and the differential sample-clock signal is connected to a jumper pair. `Mmuart_0_txd_m2f` is used to transmit the on-chip data via USB to the host

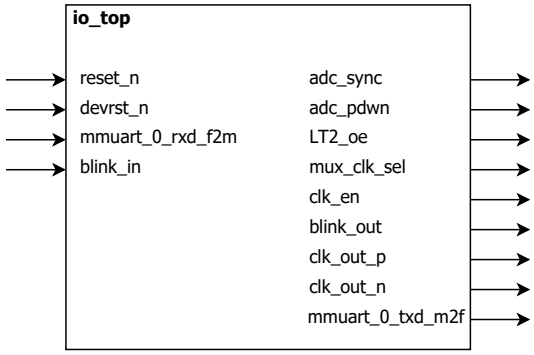


Figure 5.19: Soc top level

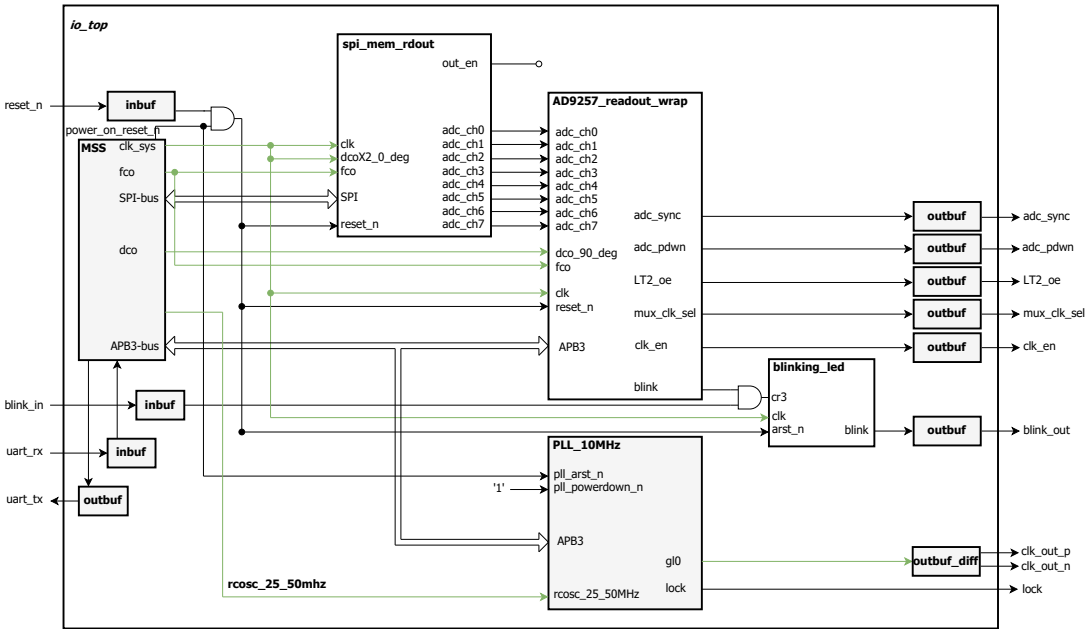


Figure 5.20: Illustration of io_top for testing on FPGA.
Gray boxes are Microsemi IP's. Note that some signals from the MSS are not included.

computer.

Figure 5.21 shows the MSS top-level while figure 5.22 illustrates its internal structure. The CPU, hard MSS peripherals and FIC explained in the beginning of section 5.3 are instantiated inside `MSS_comp_gen_sb_MSS_0`. As the FIC was configured to use an APB3-bus, `CoreAPB3` was generated. `FABOSC_0` and `CCC_0` generates the system clock, `DCO`, and `FCO`. `SYSRESET` and `CoreResetP` were added by the system configurator to generate reset signals during power-up.

The MSS was generated by using the systembuilder option in Libero. After the component had been generated, it was converted to a smartdesign where more settings are accessible. These terms are explained in "B.2: Software" in Appendix B: Tools. Following this, `DCO` and `FCO` were added as generated clocks from the CCC. This was considered more practical than using a separate CCC instance in the model. In the early design phase, an issue regarding communication with `AD9257_readout_wrap` in the application software was fixed by changing the APB3-slave. Its memory slot was

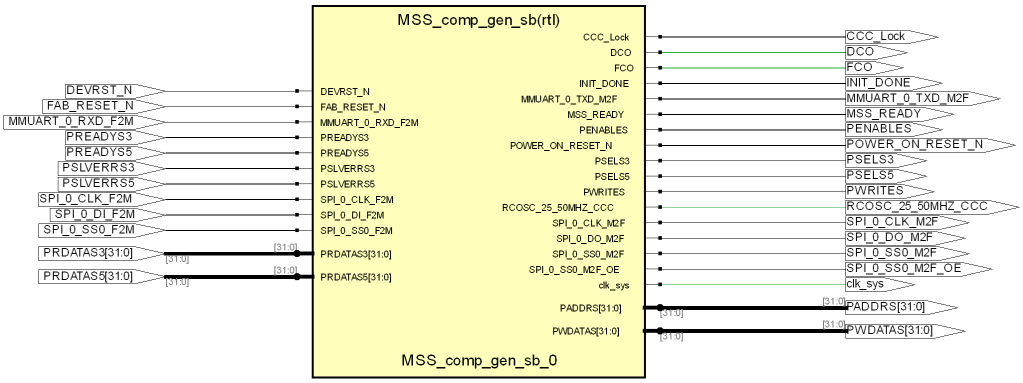


Figure 5.21: MSS top level

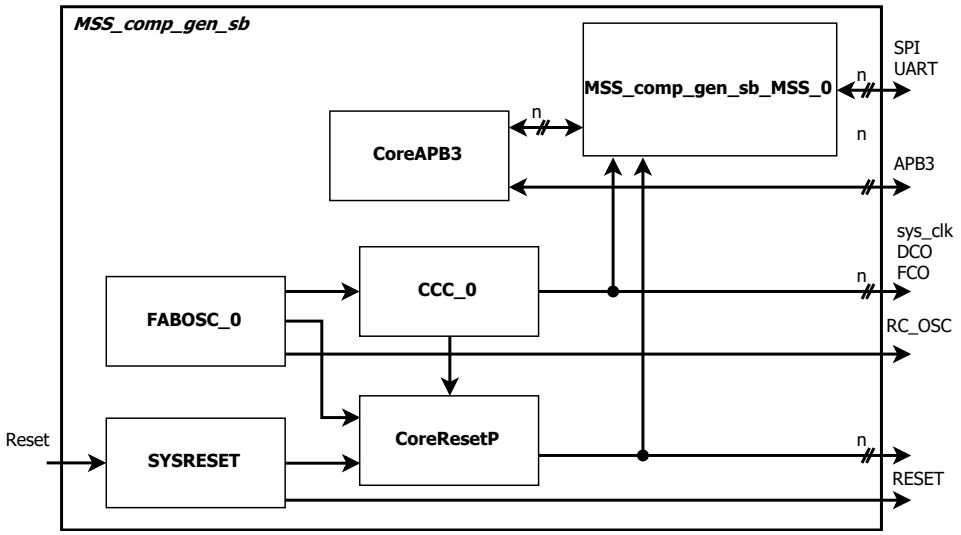


Figure 5.22: MSS internal components

changed so that it matched the fabric memory address region designated to the FIC. This was not examined further and is therefore not explained in greater detail.

Clock- and I/O- constraints were added prior to synthesis and place-and-route. Timing analysis was then performed. Hold-time violations were reported in the data path between the model output and the readout logic. Figure 5.23 shows the CCC configuration in Libero. The frequencies and times presented after the multiplexer are as intended, where the output `GL1` is used as `DCO` and `GL2` is used as `FCO`. The times at the outputs causes a deviation from the intended times. This was corrected by applying a minimum-delay constraint in the signal path until no hold-time violation was reported. After this, a programming file for SF2 was created.

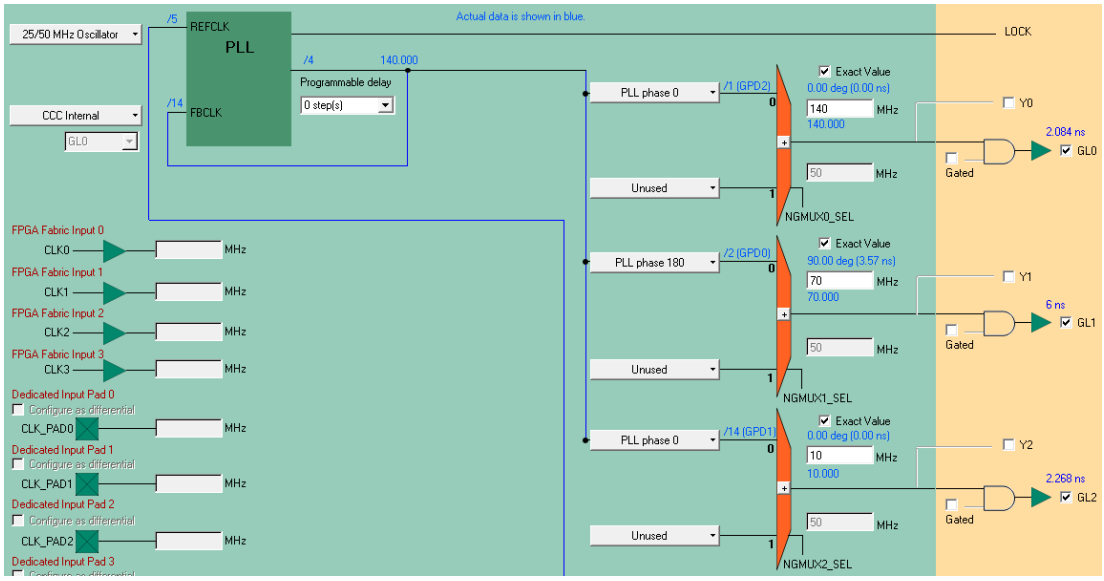


Figure 5.23: CCC configurator

5.4 FPGA-loopback SoC

This system was designed to test if the digital designs works as intended when the signals are sent out of SF2, through jumpers on the SF2-dev-board, and back into SF2. This is the only significant difference compared to the internal system in section 5.3. Figure 5.24 illustrates the SoC. Due to a limited number of jumpers, only one data channel, in addition to the capture clocks, was looped back.

The following sub-sections present the system that was designed for computer-aided testing, and the system that was implemented on the SF2 for physical testing. The difference between the two systems is explained in 5.1. The process of setting up the system in Modelsim, and how it is implemented and tested on the SF2-dev-board is described in Appendix C: Project setup.

5.4.1 Computer-aided verification

Figure 5.25 shows the structure of the testbench. `Io_top_tb` is the top level where all functionality except the MSS is instantiated. This structure was used when testing the

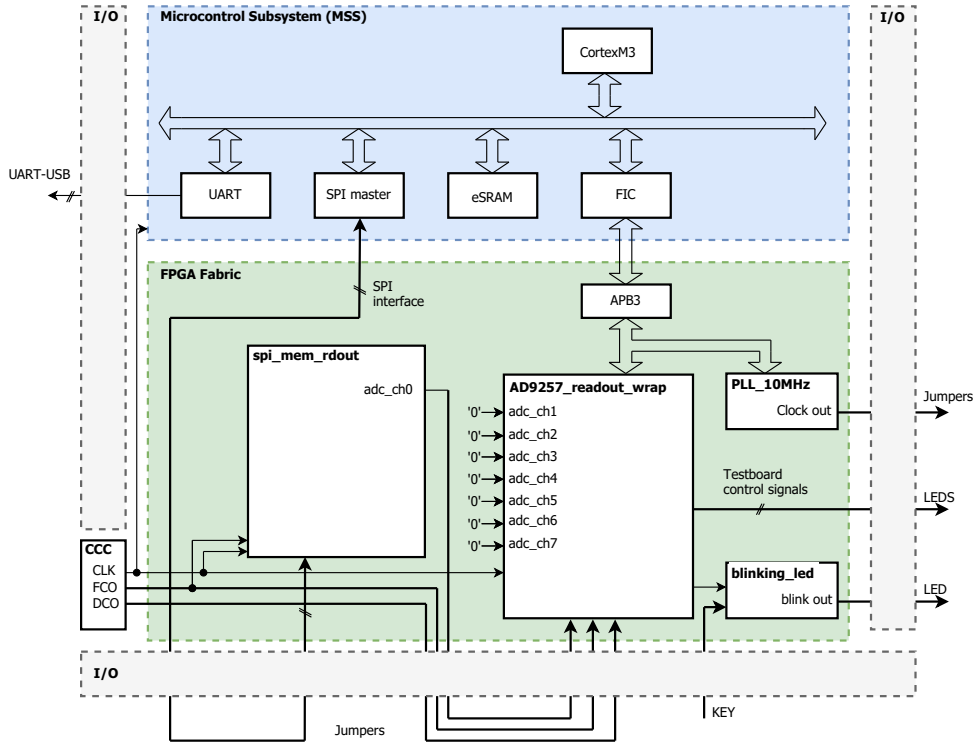


Figure 5.24: Illustration of SoC for FPGA loopback

system pre-synthesis, post-synthesis, and post-layout. The latter two were created in Libero SoC with `io_top_tb` as the top level. Figure 5.25 shows the testbench structure. Compared to the FPGA-internal system that was presented in section 5.3.1, the only differences are that the outputs from the model, the capture clocks, and the SPI-bus go through I/Os. The rest of this section only mentions the differences in this system compared to the system in section 5.3.1. Therefore, it should be read prior to this section.

Figure 5.26 shows the top-level structure. The internal structure is similar to the internal structure of the internally connected system in figure 5.18. The exceptions are the routing, and the addition of I/O-buffers. Figure 5.10 illustrates how the SPI interface is connected. In this system, I/O-buffers from Microsemi are used instead of the I/O models. Thus, the `SPI master` signals connect to outputs, which are given the name `SPI master out`. These can be looped back to `SPI slave in`, which connects to the model `spi_mem_rdout` through input buffers.

As previously mentioned, only one data channel is connected between the model and the readout logic. The other outputs from the model were left floating, while the corresponding inputs in the readout logic were set to a fixed value. Data-output channel 0, and the capture clocks, are converted to differential outputs. The readout logic data input is converted back to a single-ended signal before it is connected. The capture-clock inputs are put on global nets before they are connected to the readout logic. The results of the tests for this system are presented in chapter 6.

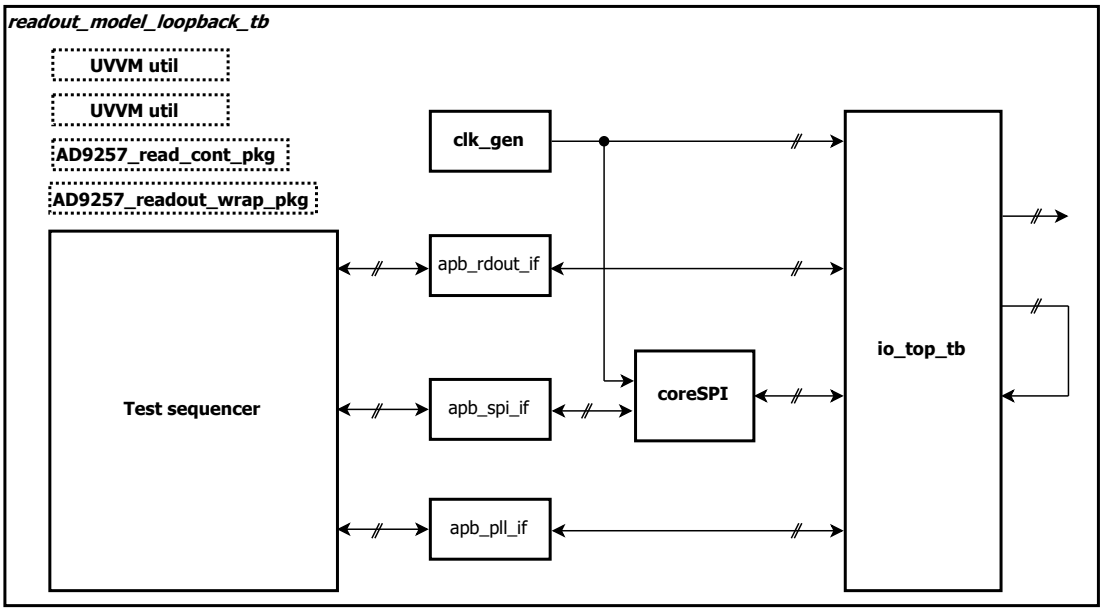


Figure 5.25: testbench structure for FPGA loopback

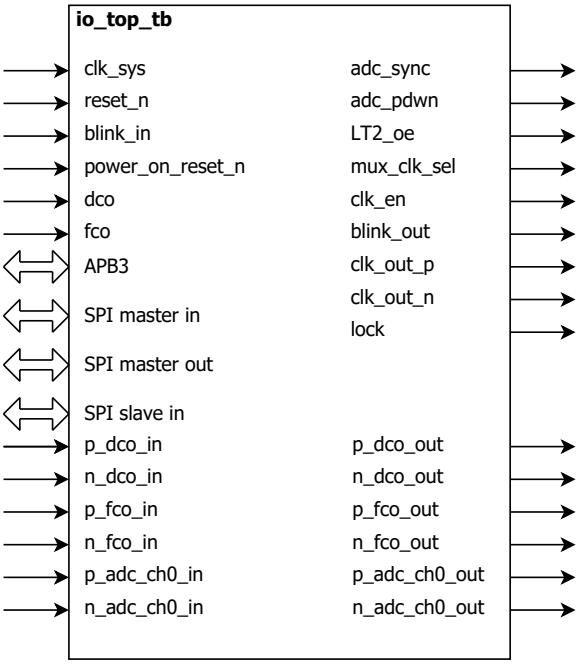


Figure 5.26: io_top_tb top level

5.4.2 Implemented FPGA-loopback SoC

The only difference between the top level in this system compared to the implemented system for FPGA-internal in figure 5.20, is that the signals between the MSS SPI and readout logic, and the model goes through I/Os. Clock constraints and I/O constraints were added before the programming file for the SF2 was created. Timing analysis was performed after place-and route where no errors were reported.

5.5 AD9257-testboard SoC

This system connects to the AD9257 testboard that was made. Figure 5.27 illustrates the SoC. The signals between SF2 and the board goes through certain I/Os that are connected to jumpers on the SF2-dev-board. These are located to the left of the FPGA in figure 5.2. All data channels and both clock outputs are connected to `AD9257_readout` through `AD9257_readout_wrap`. Testboard control signals and the SPI-bus are also connected to the testboard through jumpers. The details about the testboard is presented in section 5.6.

As for sections 5.3 and 5.4, the following sub-sections present the systems that were made for verification and implementation. Appendix C: Project setup explains how to use the systems.

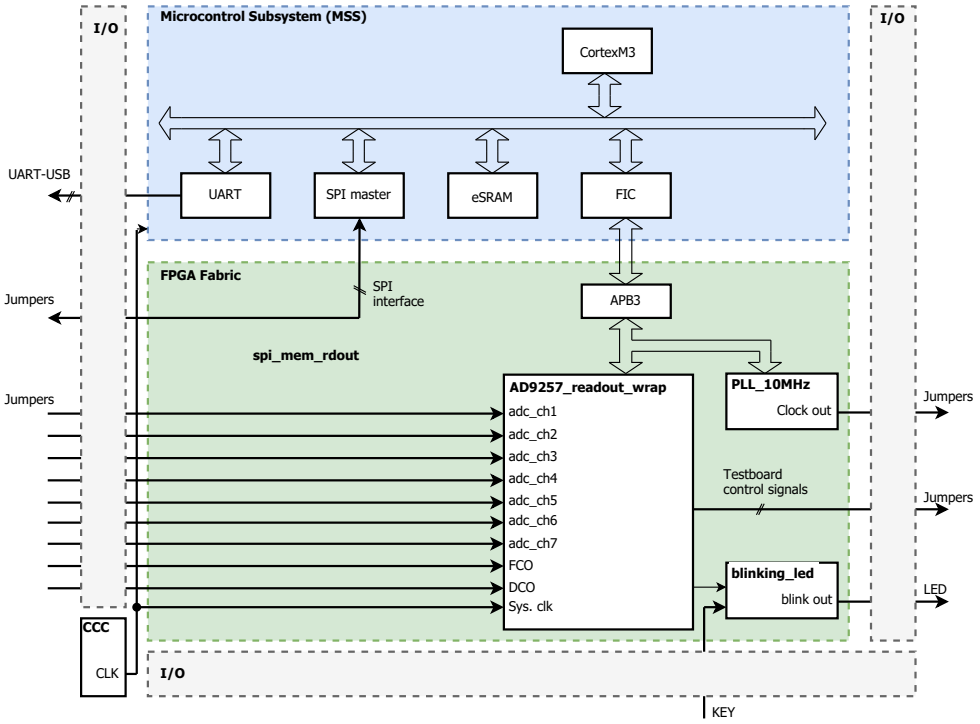


Figure 5.27: Illustration of SoC for testboard

5.5.1 Computer-aided verification

Figure 5.16 shows the structure of the testbench. `Io_top_tb` is the top level where all functionality except the MSS is instantiated. This structure was used when testing the system pre-synthesis, post-synthesis and post-layout. The latter two were created in Libero SoC with `io_top_tb` as the top level. Figure 5.28 shows the testbench setup for computer-aided testing of this system. Unlike the previously explained systems, the model is not an instance in `io_top_tb`. As it is not used in the SoC that is implemented, the model is added as an instance in the testbench top level `readout_testboard_tb` through `io_top_model`. In this way, the testbench structure remains the same. `Io_top_model` is connected through Microsemi I/O buffers so that it is possible to synthesize the model as well.

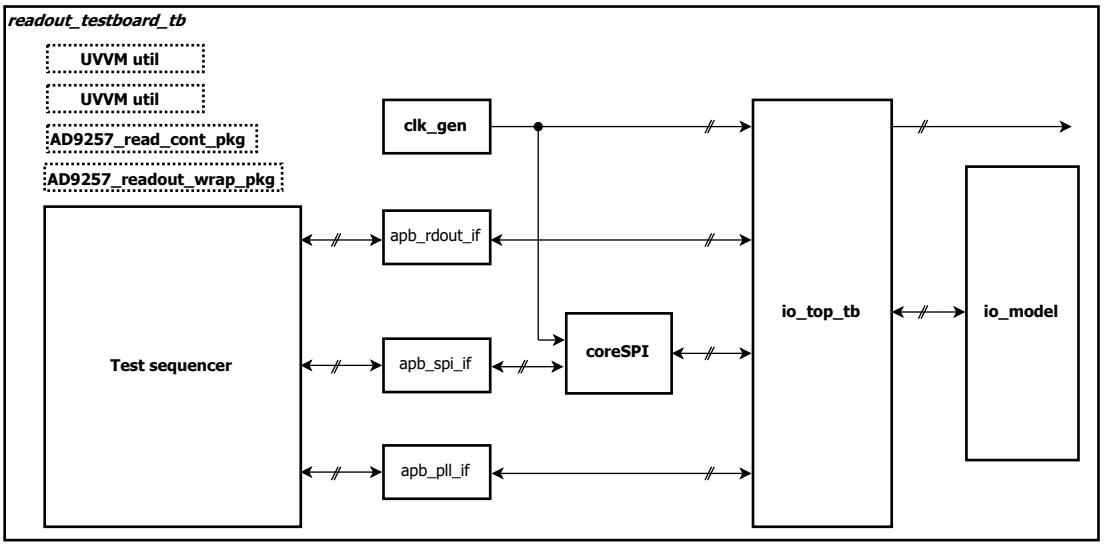


Figure 5.28: Testbench structure for testing against circuit board

Figure 5.29 shows the top-level interface of `io_top_tb`. The internal structure is similar to the internal structure of the internally connected system in figure 5.18. The exceptions are, as mentioned, that the model `spi_mem_rdout` is removed, and that the data inputs and capture clocks come from differential inputs. The results of the tests that were performed on this system are presented in chapter 6.

5.5.2 Implemented AD9257-testboard SoC

The only difference between the top level in this system compared to the system in figure 5.29, is that the MSS is added as an instance in the top level. Thus, the bus-ports `APB3` and `SPI master in` are removed and instead connected internally to the MSS. An UART interface is also added. A new MSS component was generated for this system. The only difference between this and the MSS in the other systems is that the CCC clocks `DCO` and `FCO` are removed. Clock constraints and I/O constraints were added before the programming file for the SF2 was created. Timing analysis was performed after the design had been placed and routed. No errors were reported.

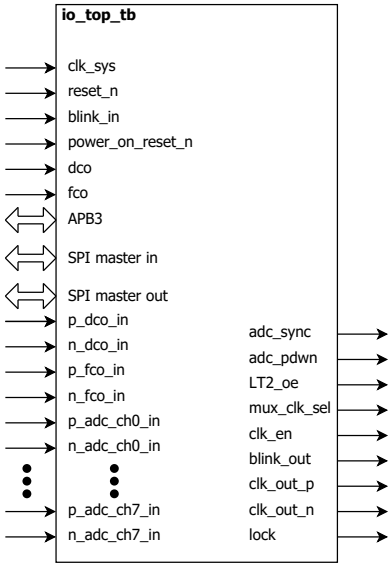


Figure 5.29: io_top_tb top level

5.6 AD9257 testboard

Analog devices provide an evaluation board for AD9257. This board is made to connect to a data acquisition board. By connecting this board to a computer, the ADC can be evaluated using software provided by Analog devices. [48] It was however decided that instead of using this configuration, a circuit board containing the ADC should be made. This decision was made because the evaluation board's connectors does not fit the connectors on the SF2-dev-board. It was also of interest to test all the analog-front-end SED-methods that is mentioned in section 3.4, especially the fully-differential opamp.

5.6.1 Practical considerations

5.6.1.1 Digital back-end

AD9257 has eight data outputs and two clock outputs common to all the channels. The output-stage buffers are made to comply with the ANSI-644 LVDS standard. Traces longer than 61 cm can result in timing errors. The output traces on the circuit board should be close together and should have equal lengths.[24]

The SF2-dev-board is pictured in figure 5.2. To receive LVDS-signals on SF2, a matching pair of inputs must be used. These are denoted by n and p, i.e. one negative, and one positive terminal. The FMC-connector at the top of the figure was first intended to be used. Unfortunately, none of the connections from the connector to the pins of SF2 is usable for LVDS. The connections are mainly to MSIOs and DDRIOs on the FPGA. The issue with the pins that connect to MSIOs is that they are connected through a 2.5 V to 3.3 V voltage-level translator. The LVDS signal has a common mode voltage of about 1.2 V, and swing between 1.4 V and 1.1 V [23]. DDRIO is a MSIO optimized for DDR-memories [18]. They do not support LVDS [18]. They do support HSTL which is a differential standard, and it is possible to convert LVDS to HSTL by terminating the

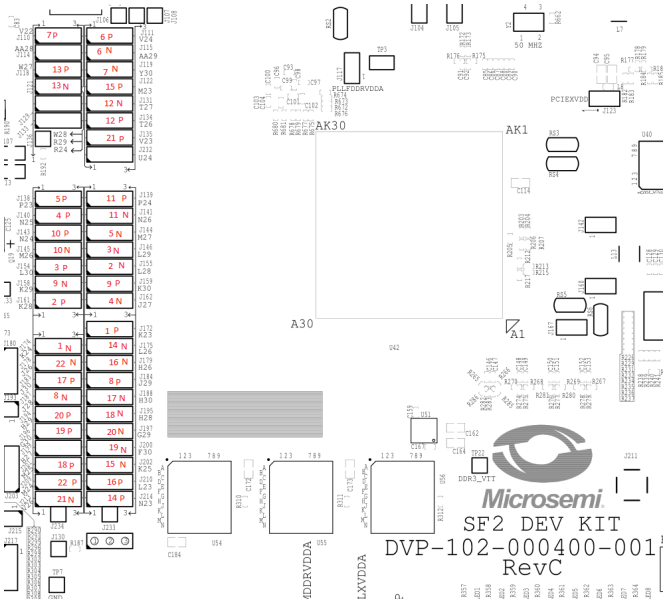


Figure 5.30: Positive and negative IO pairs on jumpers

inputs in a certain manner [49]. HSTL is however not possible to use since the bank voltage where the DDRIOs are located only enables the use of LVCMOS25 [47].

All other connectors on the SF2-dev-board were also found to be unusable. The only possible way to connect the needed amount of pairs to the FPGA was through the on-board jumpers. Figure 5.30 shows the jumpers, where the negative and positive I/O-pairs are marked. The intended use of the jumpers is to switch FPGA ports between the voltage-level translators leading to the FMC-connector, or to on-board components such as SPI-memory, and UART-to-USB converters.

Using the jumpers is generally not a good idea for several reasons. As figure 5.30 shows, the I/O pairs are not always routed to adjacent jumpers. This is an issue as it is difficult to match the length of the traces. The differential signals between the testboard and the dev-board are rather far apart. This increases the chance of uneven distribution of externally coupled noise. The source- and characteristic- impedance of the signal paths should be matched to reduce reflections. The impedance in the jumpers are unknown, and might be slightly different between jumper pins. This complicates the matching. Another board was also considered. At this stage in the design process, it was not desirable to use other FPGAs than the SF2. Emcraft systems provides the development kit M2S-FG484 SOM Starter Kit. This is displayed in figure 5.31[3]. The connections on this board are however similar to the jumpers on the SF2-dev-board. Despite the issues, it was decided to continue with this board.

5.6.1.2 Analog front-end

Since eight channels are available on AD9257, it was decided to include all the different methods for SED-conversion that was mentioned in 3.4. The primary analog signal source is a signal generator. Signal generators are often terminated internally with a

[3]Picture was taken from http://www.emcraft.com/products/255#starter-kit

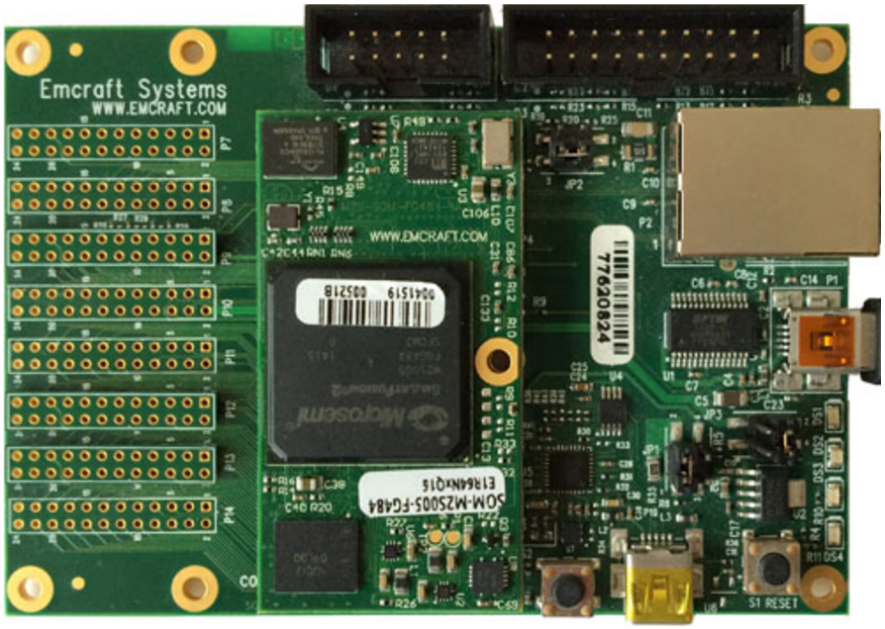


Figure 5.31: M2S-FG484 SOM Starter Kit from Emcraft systems.

50 Ω resistor. The trace impedance should match this. A termination resistor at the receiving end should also be included to have the possibility of terminating near the receiving end. It should be possible to both AC- and DC-couple the paths between the signal generator and the receiver as this it is interesting to test both cases for some of the SED-converters.

5.6.1.3 Single-ended signals

MSIOs in banks powered by a 3.3V supply voltage. This is not a problem for signals that use the LVDS standard, but it poses an issue on any single-ended signal between SF2 and AD9257 because the inputs of AD9257 have an absolute maximum of 2 V [24]. The SPI interface uses single wires. To mitigate this, some form of voltage-level shifting must be done. A ground connections must also be made between the boards so that the single-ended signals have a current return path.

5.6.1.4 Power supply decoupling

AD9257 and the other active devices that were included on the board should have decoupling capacitors on power supply inputs. The number of capacitors, and their values, are based on what is recommended in the component's datasheet.

5.6.2 Design of the AD9257 testboard

A block diagram of the realized circuit board is shown in figure 5.32. All channels of the AD9257 are used. All single-ended-to-differential conversion methods, except the transformer, are implemented twice on the board so that both AC-coupled and DC-coupled versions can be tested at the same time. The transformer will intrinsically

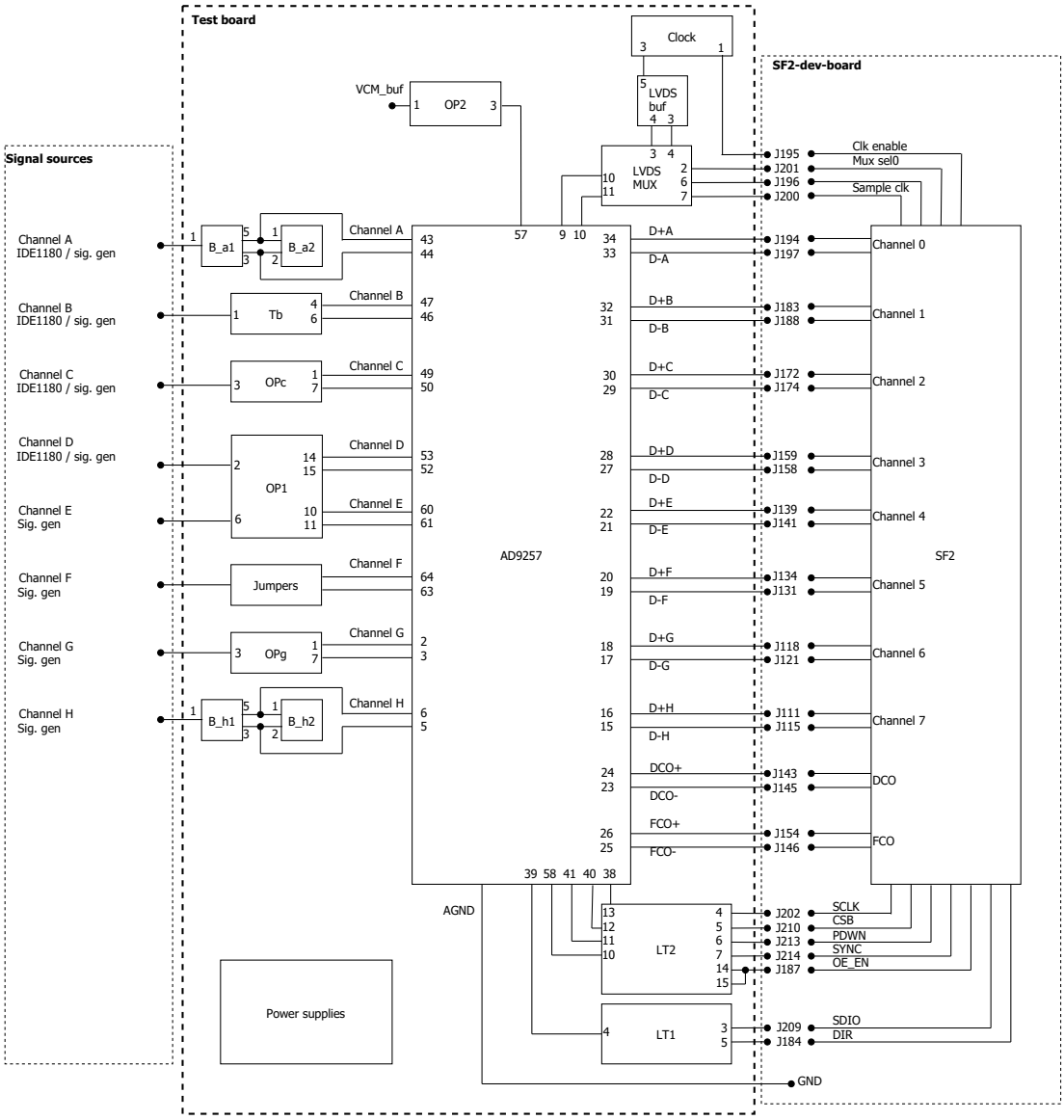


Figure 5.32: AD9257 testboard block diagram

block DC, so two implementations were not necessary. The extra circuit was replaced by a jumper configuration where different voltages can be applied. The ADC data channels and capture clocks connect to the SF2 through the jumpers on the dev-board. An adjustable sample clock is sent differentially to an LVDS mux. A control signal from the FPGA chooses whether this clock or an on-board 10 MHz clock is fed to the clock inputs of AD9257. The SPI signals and AD9257 control signals are sent through voltage-level shifting circuitry before they are connected to AD9257. The direction of data flow on `SDIO` through `LT1` is decided by `DIR`, which is controlled by an output-enable signal that is generated by the MSS SPI peripheral.

The following sections present the different circuits that were implemented on the board. All components used for SED-conversion on the testboard are listed in Appendix D: AD9257 testboard extras.

5.6.2.1 Impedance compensation circuitry

The impedance compensation circuit in figure 5.33 is part of all the SED-converter circuits, and is therefore explained prior to the other circuits. It is implemented between the ports of all SED-converter outputs and AD9257 to compensate for the switched-capacitor inputs of AD9257, as mentioned in section 3.4. The circuit also allows for an implementation of a low-pass filter.

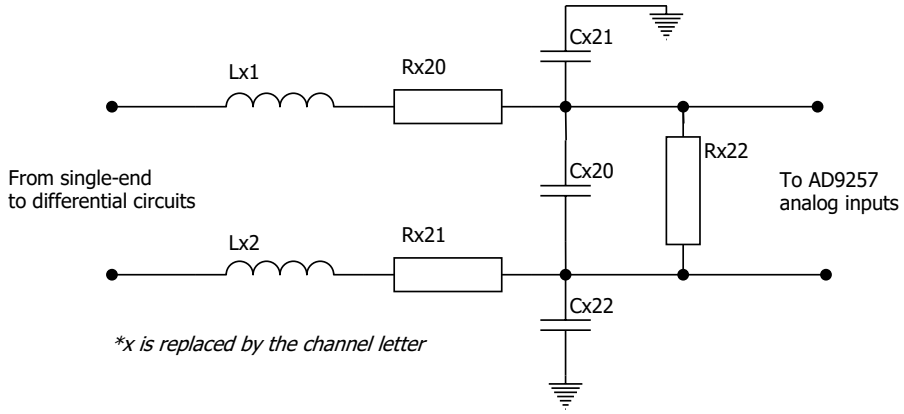


Figure 5.33: Compensation circuitry

5.6.2.2 Voltage buffer circuitry

AD9257 provides 0.9 V through output `VCM`. As mentioned in section 3.4, AD9257 requires this voltage on both inputs of each channel. `VCM` is buffered before it is distributed on the board. This removes a potential issue that could occur if the circuits draw more current than the output can deliver, which would cause a voltage drop. This should not be a problem since the current that is sinked from this voltage source is low, but it was implemented as a precautionary measure. Figure 5.34 shows the circuit. The opamp that was used is located in a dual package. Only one of the opamps are used.The unused opamp is terminated so that the voltage on every port is kept mid-supply.

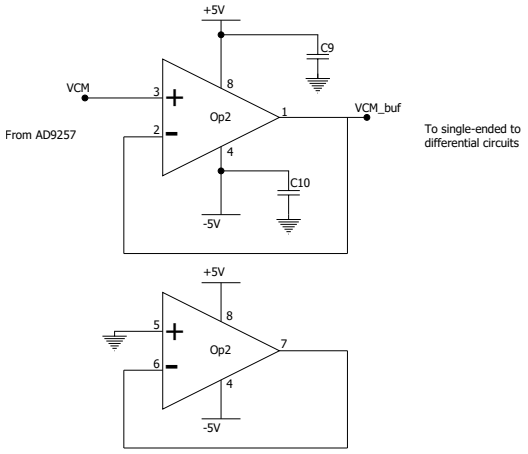


Figure 5.34: VCM output from AD9257 is buffered as it drives eight different circuits

5.6.2.3 Jumper configuration

The jumper configuration circuit is connected to channel F in figure 5.32. Figure 5.35 shows the jumper implementation and its signals. If the circuit is not in use, `VCM_buf` can be connected to channel F by setting the jumpers between pins `Jf5-Jf8` and `Jf1-Jf4`. By other jumper placements, the signal source, `gnd`, or `VCM_buf` can be connected to one or both inputs. This circuit can be useful when AD9257 is characterized. E.g the linearity of the transfer function can be determined by applying a slow moving voltage-ramp to both inputs.

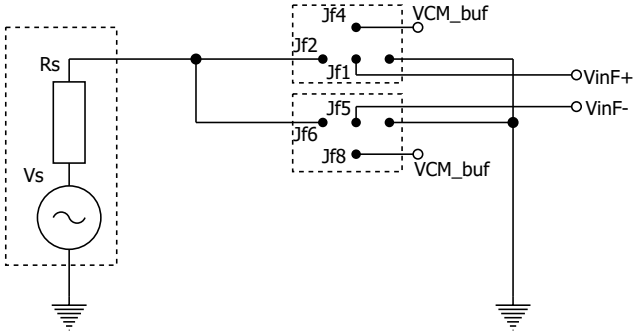


Figure 5.35: Jumper configuration

5.6.2.4 Balun configuration

A double balun circuit is connected to channel A and H in figure 5.32. Figure 5.36 shows the circuit schematics. It is one of two circuits which are recommended by Analog devices to perform the SED-conversion for AD9257 [24, p. 19]. `Ra6` and `Ra7` were added to have the possibility of attenuating the input signal, and to terminate the input.

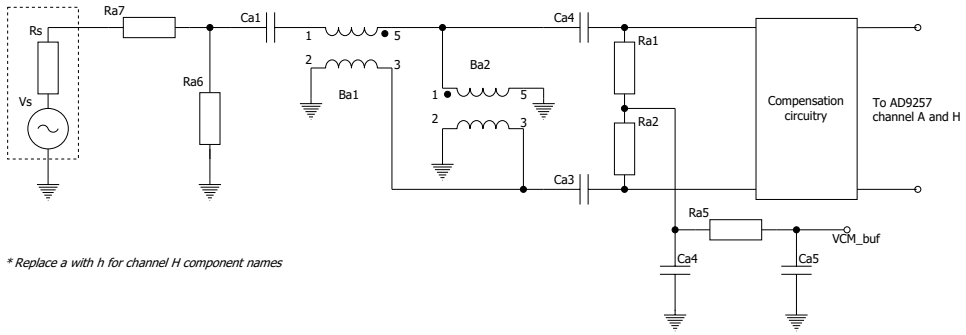


Figure 5.36: Double balun configuration

5.6.2.5 Transformer configuration

The circuit that uses a transformer is connected to channel B in figure 5.32. Figure 5.37 shows the schematics of the circuit that uses a transformer convert from SED. This circuit is the other circuit that is recommended by Analog devices to perform SED-conversion for AD9257 [24, p. 19]. `Ra5` and `Ra6` were added to have the possibility of attenuating the input signal, and to terminate the input.

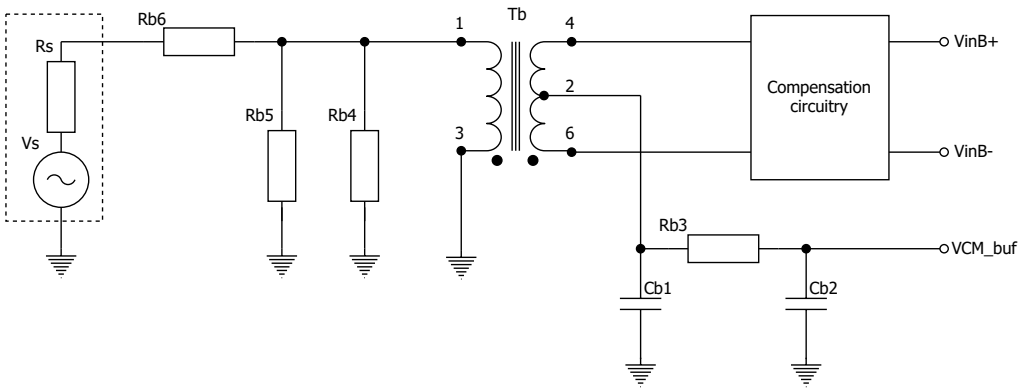


Figure 5.37: Transformer configuration

5.6.2.6 Two opamp configuration

The active circuit that uses two opamps is connected to channel C and G in figure 5.32. Many different configurations of two opamps can perform SED-conversion. The circuit shown in figure 5.38 matches the delay contribution for both outputs. Hence the phase relationship between both output signals should be close to the ideal 180 degrees. `Rc3` through `Rc6` should be of equal value. Because of this, 1 % resistors should be used so to keep the values closely matched. The gain of the entire circuit is set by the relationship `Rc2/Rc1`. [50] It should be noted that this circuit does not output a true differential signal. That is, setting the circuit gain to unity will double V_{p-p} of the input signal.

5.6.2.7 Fully differential opamp configuration

Circuits with fully-differential opamps are connected to channel D and E in figure 5.32. Figures 5.39 and 5.40 shows the implemented circuits that use an FDO. It is mentioned in more detail in section 3.4. This opamp is connected to channels D and E with slightly different designs. Channel D can be used for both AC-coupled and DC-coupled inputs, as the non-signal input can be connected to `gnd`, or `VCM` through jumper `JPD1`. Channel E should primarily be AC-coupled since an offset voltage can not be applied to the input of the non-signal input. Capacitors were added on the outputs in case it at some stage is interesting to apply the common-mode voltage here. In this case, `Vocm` on the opamps can be set to `gnd` through jumpers `JPD2` and `JPE1`.

5.6.2.8 Clock circuitry

Figure 5.41 shows the circuitry that sources the sampling clock to AD9257. It was decided to use two separate clock sources: One fixed-frequency on-board clock, and one adjustable-frequency clock which was generated in the FPGA. How this is done is explained in section 5.2.4. The clock is sent as LVDS to the testboard where it is connected to one input of a 2:1 LVDS multiplexer. The on-board clock is generated by a crystal. This has a single-ended output, and is therefore converted to a differential LVDS signal in an LVDS buffer before it connects to the inputs of the multiplexer. The output is connected to the clock inputs of AD9257. Two control signal are sent from the

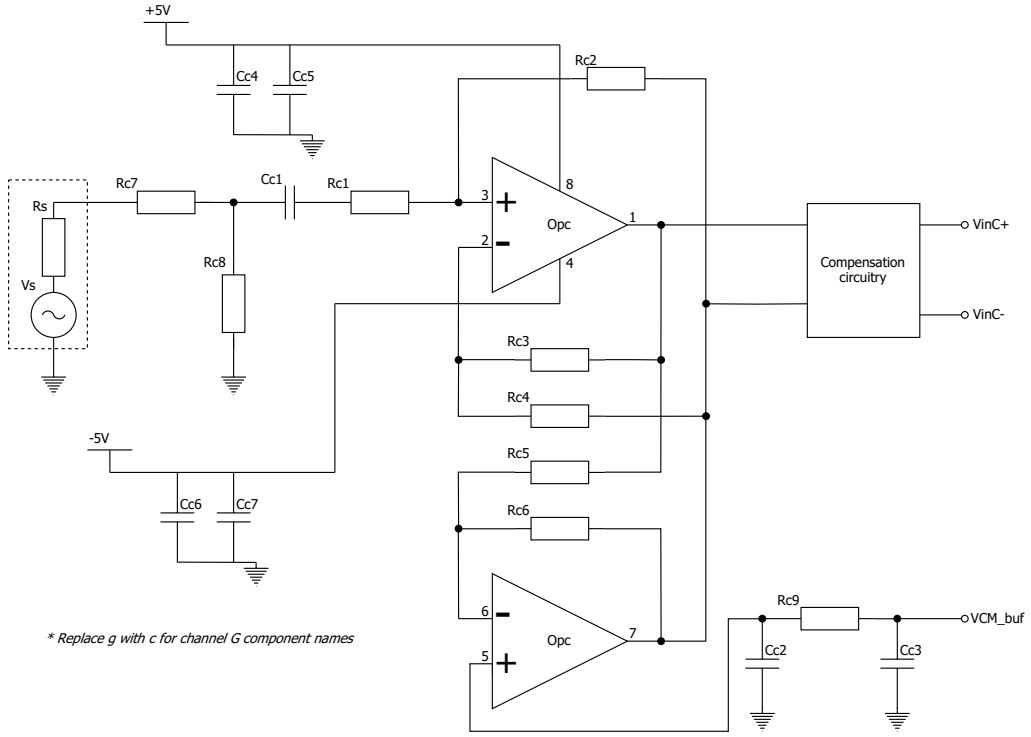


Figure 5.38: Dual opamp configuration

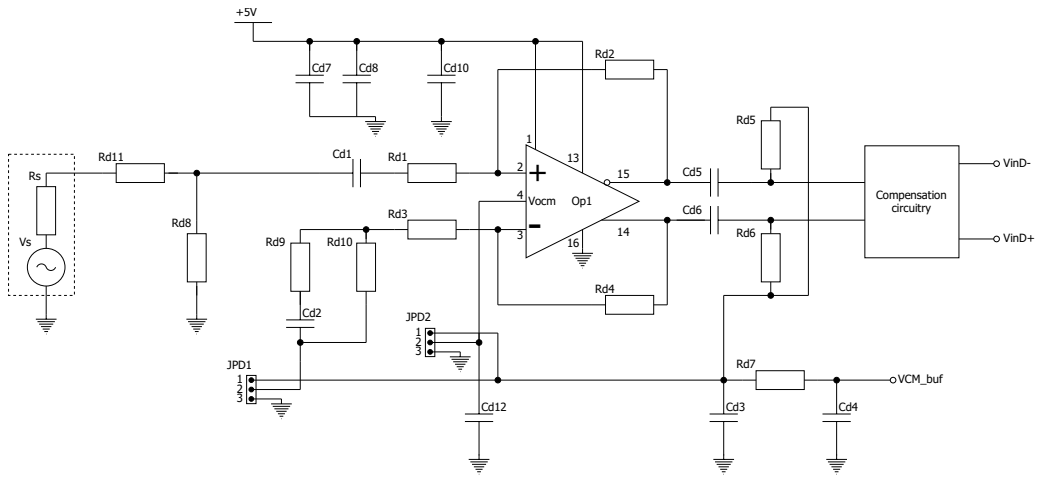


Figure 5.39: Fully differential opamp configuration for channel D

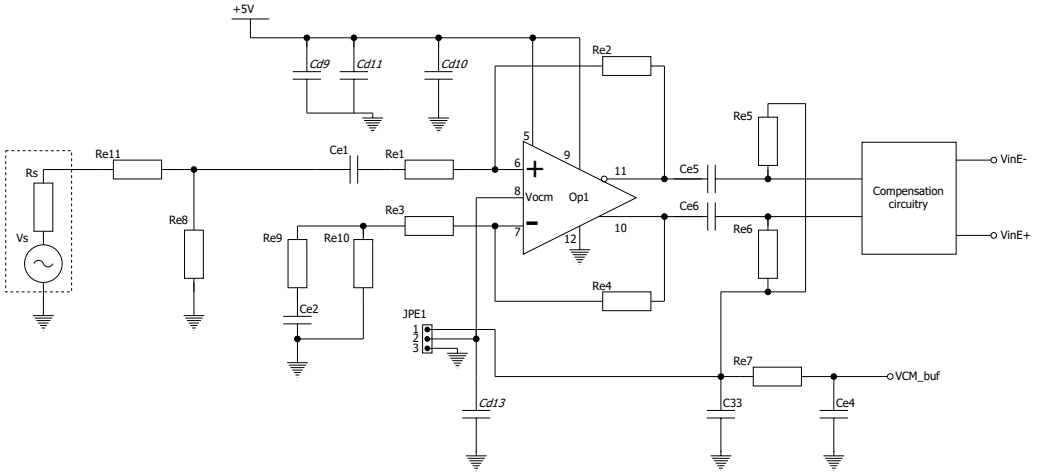


Figure 5.40: Fully differential opamp configuration for channel E

FPGA. One signal enables/disables the on-board clock, and the other chooses which clock is sourced to AD9257.

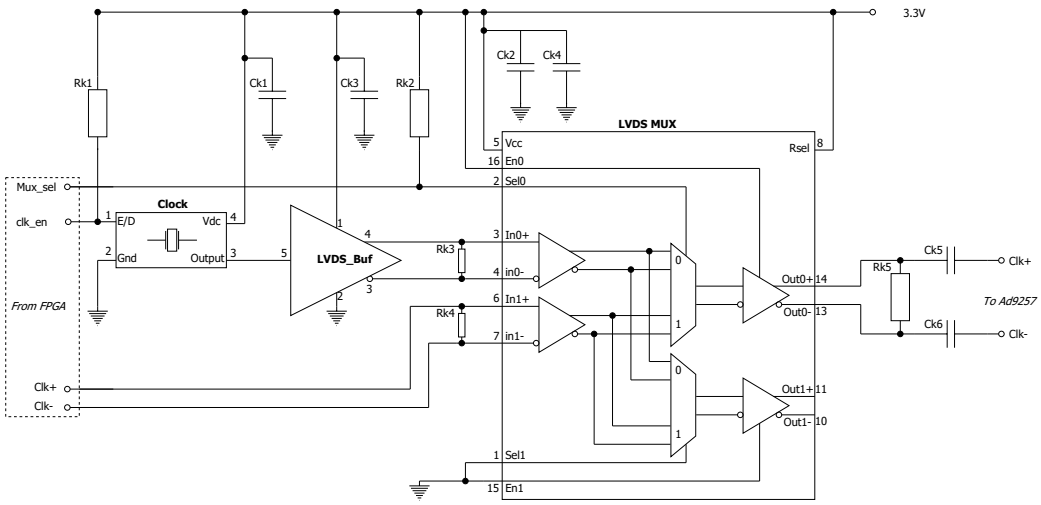


Figure 5.41: Clock circuitry

5.6.2.9 Voltage level translation

As mentioned in section 5.6.1.3, the voltage of the single-ended signals from SF2 do not match the input voltage on AD9257. To mitigate this, the circuit in 5.42 is used to shift the voltage levels on the SPI signals. `PDWN` and `SYNC` were mentioned section 3.3.1, and require the same treatment. LT1 and LT2 are voltage-level converters. They convert an input from one voltage domain to an output in a different voltage domain. All unidirectional signals are connected through LT2 which is configured so that it only convert inputs from the FPGA 3.3 V domain to the AD9257 1.8 V domain. It has one control signal: `OE`. When this is high, all outputs are high-impedance. It must be set low for normal operation. `SDIO` is connected through `LT1`. This level shifter must be bidirectional. Control signal `DIR` chooses the direction of transfer. This signal is

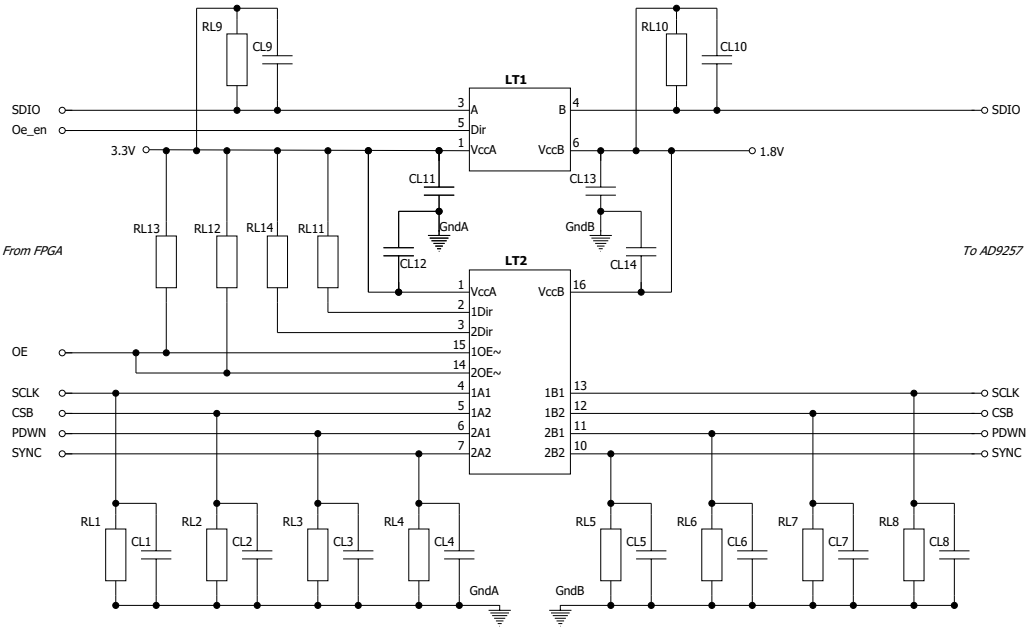


Figure 5.42: Voltage level translation circuitry

generated in the MSS SPI, and is the same signal that is used on the bidirectional buffers for `din`, `dout`, and `SDIO` in SF2.

The inputs of AD9257 have internal pull-down or pull-up resistors. LT1 and LT2 drives or sink the output. This should not pose any problems. The resistors and capacitors in the schematic are pull-up/down-, compensation-, and decoupling- components. These are included based on recommendations in the datasheets.

5.6.3 PCB layout

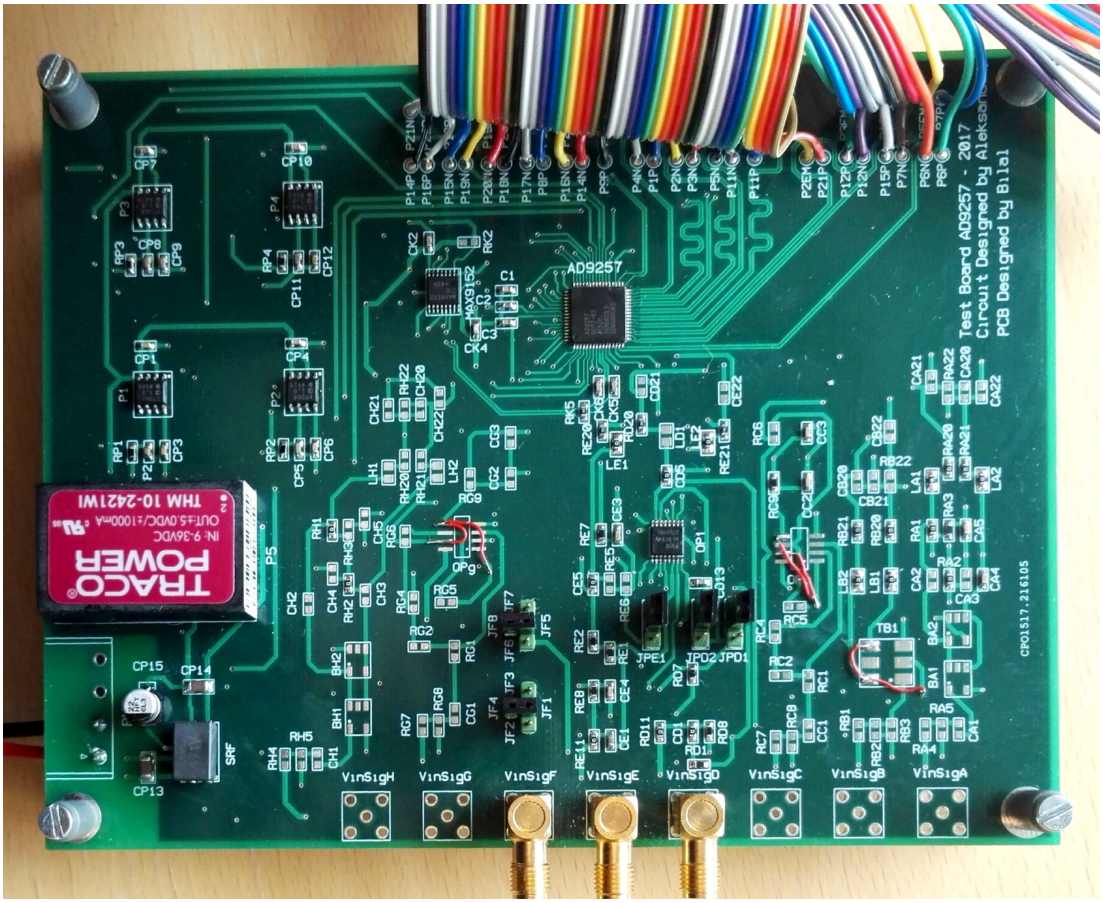
The design and layout of the testboard PCB as well as the design of the power supply were performed by senior engineers[4] at the department of physics and technology. The resulting board has 4 layers. All signal paths are located at the top or bottom layer. The middle layers are ground and power planes. The schematics for the top and bottom layer are included in Appendix D: AD9257 testboard extras.

5.6.4 1st board configuration

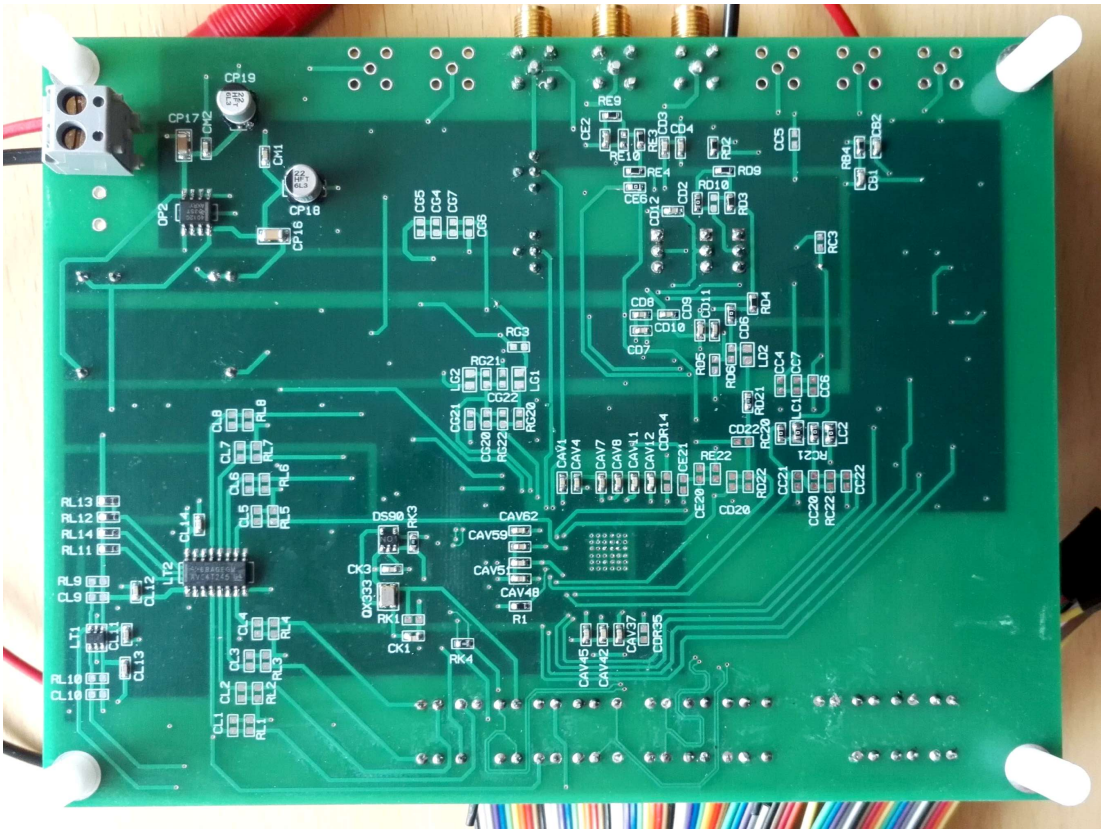
The first configuration of the board did not utilize all input channels. Only components for channel D, E, and F in figure 5.32 are assembled on the board. The unused channel inputs of AD9257 are tied to the buffered AD9257 VCM output. The assembly was performed by a senior engineer[5] at the department of physics and technology. All testboard component values are listed in Appendix D: AD9257 testboard extras. Figure 5.43 shows the board after the components were assembled.

[4]Layout by Bilal Hasan Qureshi (bilal.qureshi@uib.no). Supervised by Shiming Yang (shiming.yang@uib.no)

[5]Components were assembled by Per Heradstveit



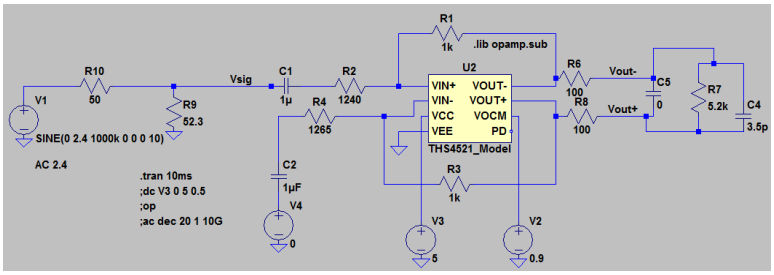
(a) Testboard top side



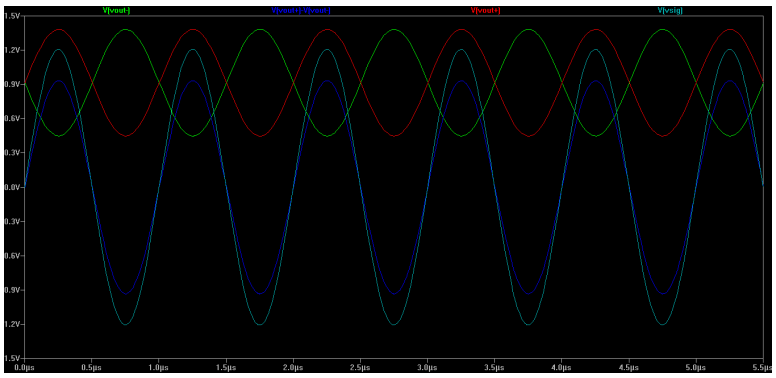
(b) Testboard bottom side

Figure 5.43: Post-assembly testboard

The component values in the FDO circuitry were calculated the way it is recommended in the datasheet of the FDO [35]. The circuit was simulated in LTspice with the calculated values before implementation. A thorough analysis was however not performed. A spice model[6] of the opamp was used in the simulation. Figure Simulation of FDO in LTspice shows the result of simulating with AC-coupled front-end. A transient analysis was performed with a 2.4 V_{AMP}, 1 MHz source. The gain of the FDO is set so that the differential output voltage was approximately -1≤ V_{diff} ≤+1.



(a) Circuit for AC-coupled simulation



(b) Transient analysis

Figure 5.44: Simulation of FDO in LTspice

5.7 Embedded software

Software was written in order to utilize and access the hardware components in the SoCs that are presented in sections 5.3, 5.4 and 5.5. The UART enables connection to a host computer, where the content sent from the SoC can be read by a terminal that connects to the UART through USB, and vise versa. The software was designed in a way that provides a user interface in the terminal. Menu choices are entered from the keyboard on the computer. These menu choices enable control of the hardware components in the SoC, such as data transfer by the MSS SPI, or reading the captured data in the readout logic. All the software functionality that was made is explained in the remaining section.

Figure 5.45 shows the high-level hierarchy of the software. Microsemi provides embedded software for SF2 MSS peripherals. Custom drivers had to be made for the interfaces to `AD9257_readout_wrap` and `PLL_10MHz`. They make it possible to write too, or read

[6]THS4522IPW. Downloaded from www.farnell.com

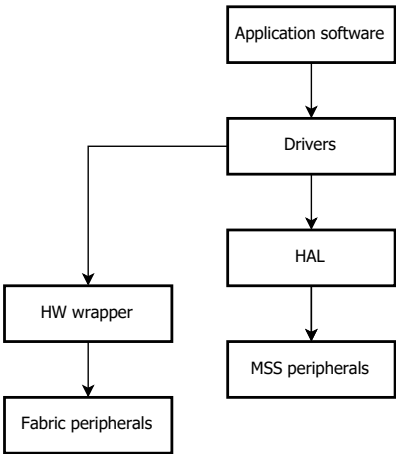


Figure 5.45: Software hierarchy

from the hardware registers. Table 5.3 lists the drivers that were created for the two modules. The drivers are simply a pointer to the base address of the peripheral, where a specific register offset is added. Offsets for the PLL are predefined [46], while the offsets for the readout wrapper are defined in `AD9257_readout_wrap_pkg.vhd`.

Multiple functions were made for this project. These are listed in table 5.4. Note that the function parameters are excluded. All functions are defined in one file. The main menu functions contain big-picture functionality. An example is `test_board_control`(). This function has underlying functionality where amongst others control signals can be toggled, and the frequency of the dynamic PLL can be altered.

Support functions were made to increase readability of the code main menu functions, and for frequently used functionality. Examples are `To_hex()` and `to_asci()`. These are used to convert integers to ASCII, and vice versa. This was necessary because characters received from the host computer were ASCII-encoded. Some functionality required the integer value as they represented hex-numbers. Similarly, integer values such as the readout data had to be converted to ASCII to display the correct values in the terminal. Another example is `set_output pattern()`. When this function is called, data output in AD9257 is set via the SPI interface. The function polls the readout-data-control register in `AD9257_readout_wrap`. When the register signals that new data has been captured, it reads the readout-data-registers for the channels that reported new data. The data is then compared against the pattern that was set as output data in AD9257, and prints the results to the host computer.

To apply a hierarchy to the design, all functionality is defined in the file `func_and_const.h`. The top level file `main.c` is a short menu where the functionality of each menu choice is briefly explained. Table 5.5 shows the two highest levels of the software functionality. Note that the names in the software differ slightly from the names in the table.

SPI functionality enables the user to read and write to every address defined for AD9257 [24]. When reading, the content is displayed in the terminal window. When writing, the content is read back, and displayed after the new content has been written. *Readout wrapper register access* enables manual access to all the registers `AD9257_readout_wrap`. E.g. readout data can be read, or test board control signals can be toggled.

Readout functionality has an underlying menu. *Read all channels* reads the latest content

Table 5.3: Drivers
_xx() is a placeholder for the readout-channel numbers.
_yy() is a placeholder for rd and wr.

File name	**Functions**
AD9257_readout_wrap_func.h	*Read*
	channel_yy()
	error_data_yy()
	irr_reg()
	id()
	control_reg()
	Write
	clear_irr()
	sync_enable()
	blink()
	manual_read()
	new_readout_en()
	pdwn_enable()
	LT2_oe()
	mux_clk_sel()
	clk_en()
	sw_reset()
dynamic_pll_func.h	*PLL register rd/wr*
	FCCC_RFMUX_CR_xx()
	FCCC_RFDIV_CR_xx()
	FCCC_FBMUX_CR_xx()
	FCCC_FBDIV_CR0_xx()
	FCCC_FBDIV_CR1_xx()
	FCCC_NGMUX0_CR0_xx()
	FCCC_NGMUX0_CR1_xx()
	FCCC_GPMUX0_CR_xx()
	FCCC_GPD0_CR_xx()
	FCCC_PLL_CR0_xx()
	FCCC_PLL_CR1_xx()
	FCCC_PLL_CR2_xx()
	FCCC_PLL_CR3_xx()
	FCCC_PLL_CR4_xx()
	FCCC_PLL_CR5_xx()
	FCCC_PLL_CR6_xx()
	FCCC_GPDS_SYNC_CR_xx()
	FCCC_PLL_CR7_xx()
	FCCC_PLL_CR8_xx()
	FCCC_PLL_CR9_xx()
	FCCC_PLL_CR10_xx()
	FCCC_GPD0_SYNC_CR_xx()
	FCCC_PDLY_CR_xx()

Table 5.4: Application software

File name	**Functions**
func_and_const.h	*Main menu functions*
	AD9257_SPI_memory_rd_wr()
	Wrapper_register_access()
	AD9257_readout_func()
	test_board_control()
	test_sequencer()
	Support functions
	to_hex()
	to_asci()
	nibble_to_asci()
	readout_to_uart()
	set_output_pattern()
	set_user_patt()
	pseudo_rand_test()
	poll_irr_reg()
	spi_write()
	spi_read()
	compare_data()

Table 5.5: Software functionality

main.c	**func_and_const.h**	
SPI functionality	1	Read or write to addresses defined in AD9257 datasheet
Readout wrapper register access	1	Read or write to addresses defined in AD9257_readout_wrap_pkg.vhd
Readout functionality	1	Read all channels
	2	Set ADC output pattern and confirm readout
	3	Monitor readout data
Test-board control	1	Check status of control signals
	2	Toggle control signals
	3	Run board start-up procedure
	4	Run board shut-down procedure
	5	Send clock from PLL to AD9257
Test sequencer	1	Run test similar to test in Modelsim

in the readout registers. *Set ADC output pattern and confirm readout* enables the user to set the output patterns in the model/AD9257. The user can set the patterns that are built into the model, or set user-defined patterns. The user-defined patterns can be entered manually via the terminal, or generated by a pseudo-random generator. In both cases, two patterns are made and stored in the model/AD9257. When they are equal, they remain as output data. When unequal, the output data is set to all-zeroes after one transfer. When a pseudo-random generator is chosen, the user must enter a number of patterns to generate. The function will then generate the specified number of patterns, set them as output data, and test the correctness of the readout logic. When the number is reached, the number of errors is printed.

For the mentioned readout patterns, a test is performed to verify the readout data. Once, the output pattern is set, `control_reg` in `AD9257_readout_wrap` is polled for a certain amount of time. If no new data is detected, an error message is printed. If one or more channels report new data, the content of all channels is read and compared against the output pattern that was set. Channel-specific messages are then printed, telling the user if the readout data is correct or not.

Once the menu choice *Set ADC output pattern and confirm readout* has been entered, an error counter will track the number of errors until the user exits to the main menu. Thus, multiple patterns can be tested, while keeping track of the total amount of errors. If a single pattern is previously set while in the function, the error register explained in section 5.2.3 is polled. If readout data changes unexpectedly, an error message will be printed, as well as added to the error counter. As mentioned before, this functionality would not work if the input data changes back-to-back more than once. This is the reason why the pattern is set to all-zeroes after one iteration of unequal patterns. As long as the user remains in this function, all readouts, correct or incorrect, can be logged by storing the terminal session.

The final choice in *Readout functionality, monitor readout data* provides "real-time" monitoring of the readout data. Reading all the channels before new data has arrived is not possible in this system, so the data shown is not always the first specimen of new data. This is however tolerable, since this is not the purpose of the function. The purpose is to be able to see the data on each channel continuously. When the analog inputs of the ADC are used, checking if the sampled data corresponds with the applied voltage without having to manually read the output registers over and over again is practical. The user can choose to print all channels on one line, or in a list. The latter has to choices: read and print as fast as possible, or print in a readable manner.

Test-board control contains options for control and status of the circuit board. The control signals used on the circuit board can be toggled and checked manually. A start-up and shut-down procedure is also included, where all of the control signals are set to correct states. Furthermore, this is also where the clock frequency of the CCC that generate the AD9257 testboard sample clock is changed. The user can enter the register values presented in section 5.2.4 in the terminal window.

The final menu choice is the *test sequencer*. This was made so that a test similar to the tests used in the computer-aided test systems could be executed on the FPGA-implemented systems. The tests in this test sequencer are explained in chapter 6. This option should not be used on the actual AD9257, as certain bits in its internal memory should not be overwritten.

CHAPTER 6

Tests and results

This chapter describes the tests that were performed on all of the different systems described in chapter 5. The results from the testing and verification that was done in this thesis are also presented. Chapter 4 and chapter 5 should be read before this chapter in order to understand the functionality that is tested, and the structures of the test and verification systems.

6.1 Tests

Computer-aided testing was done in Modelsim. The VHDL testbenches contain a test sequencer where all tests are performed sequentially. Once the testing is complete, a log of the tests and the results is made. If any tests fail, it is reported in the log. Physical testing is done by enabling the tests described in 5.7 from the terminal on a host computer. By logging the terminal session, the tests and results can be stored.

A verification plan was made for the readout logic and SPI master in an early stage of the thesis. This is described in Appendix A: Method: "A.4: Verification". As the project progressed, other components were added to the system. Thus, they are not included in the original verification plan. The following sections describe the tests that were made to verify the readout logic, SPI-master, dynamic PLL and testboard control signals.

6.1.1 Computer-aided test sequence

The test sequence implemented in VHDL for the computer-aided test systems mostly comply the verification plan that is described in appendix A.4: Verification. In addition, tests were implemented for the dynamic PLL and testboard control signals. The test sequencer can be divided into different main sections. By selecting the highest level of verbosity control, the generated log after a test equals the sections below. Lower levels result in a more detailed log, which is helpful during debugging. A description of the tests is given in the following sub-sections.

6.1.1.1 ABP3 defaults

Here, the default values on the APB3-bus inputs to `AD9257_readout_wrap`, `coreSPI` and `PLL_10MHz` are checked. All inputs should initially be set to all-zeroes. Any errors result in error messages.

6.1.1.2 Dynamic CCC

For this component, it was necessary to test that the output frequencies changed, and was held at the intended frequency after a change was initiated. The tests for the dynamic CCC starts by checking that the default register values in `PLL_10MHz` are the same as the values that are listed in the configuration file. Next, the output clock of the CCC is changed to 12 MHz, 30 MHz and then back to its default frequency 10 MHz. After the register content for a specific frequency has been written, the sequencer waits until the lock signal is asserted. Once this is true, the sequencer can continue with the next frequency.

Errors in the default value test results in messages that notes the deviation. Clock frequencies must be checked manually in the waveguide. UVVM utility library contains functionality where pulse trains can be checked, but this was not implemented. It should also be noted that if the division factors in the CCC are improperly set, a built-in feature in the behavioral model of the PLL prints a message in the Modelsim console window.

6.1.1.3 Testboard control signals

It was necessary to test that the control signals were asserted by writing to their specific addresses in `AD9257_readout_wrap`. All control signals are part of a multiple-bit control register. The default value of this register is checked, before the signals are asserted, checked, deasserted and checked again. The signals are asserted for 4 periods of the system clock before they are deasserted again. Any errors result in error messages.

6.1.1.4 `AD9257_readout_wrap` control signals

These signals are also a part of the control register that was mentioned in 6.1.1.3. These signals are used to control the error-detection circuitry in `AD9257_readout_wrap`. This was mentioned in section 5.2.3. These signals are only asserted for one period of the system clock, and are only accessible through the APB3 bus. Their correctness must therefore be manually checked in the waveguide.

6.1.1.5 SPI master

As mentioned in 4.2.7, SPI protocol Motorola mode 0 had to be tested to assure that this would work against AD9257. CoreSPI was configured to meet the specifications that were set for the SPI master. Testing here starts by checking readable register-defaults in the coreSPI component. Content is then written to read/write-registers and read back for confirmation. Next, every address defined in the datasheet of AD9257 is written

and read. Selected addresses are tested with random data, and corner cases all-zeroes and all-ones. The number of random data iterations is specified by changing a generic in the top of the testbench. For all other addresses, one write and read is done. The data that is written to each of these is different. According to the verification plan in appendix A.4: Verification, every address should have been tested with random data and corner cases, but it was chosen to deviate from this requirement. Any errors result in error messages. Any error would imply a problem with either the transfer protocol, the configuration of the SPI master, or with the model of AD9257.

6.1.1.6 Readout built-in patterns

This part contains the first tests of the readout logic, which is the main functionality that needs to be tested. Here, all of the readout patterns that are stored in the model are set as output patterns. Amongst the patterns are all-zeroes, all-ones and alternating-ones and -zeroes. The output patterns are changed by writing to a specific address via coreSPI. The written data is immediately read back to confirm correct SPI transfer. The readout-data registers for each channel is then read via the APB3-bus, and compared to the pattern that was set. Note that between each new pattern, the error-detecting circuitry in `AD9257_readout_wrap` must be notified. As explained in section 5.2.3, if this is not done, the readout data will latch into error-data registers.

A loop that resets the model and readout logic multiple times at different points in the readout cycle is also included here. This was included to test how the system responded. The number of iterations is specified by changing a generic in the testbench. A channel-specific error message is generated if readout data deviates from the test pattern that was set.

6.1.1.7 Unequal user-defined fixed pattern

Here, two unequal patterns are stored in the model memory via coreSPI. Following this, coreSPI is used to set these patterns as back-to-back, user-defined output data from the model. Once the patterns have been set as output data, the sequencer will wait until at least one channel have received new data. This is done by polling the new-data control register. The channel specific readout-data registers are then read via the APB3-bus and compared against the first pattern

Since the second pattern was read by the readout logic before the error-detecting circuitry was notified, this pattern is latched into the channel specific error-data registers. Thus, the second pattern can be checked by reading these registers via the APB3-bus and comparing against the second pattern. However, before data is read, the error-data control register is polled until it signals error data for at least one channel. This also checks that the error detecting circuitry works as it should. A channel-specific error message is generated if readout data deviates from the pattern that was set.

6.1.1.8 Equal user-defined fixed pattern

In this test, equal patterns are stored in the model memory via coreSPI. Following this, coreSPI is used to set one pattern as output data from the model. Once this is done,

the sequencer will wait until at least one channel has received new data. This is done by polling the new-data control register. The channel specific readout-data registers are then read via the APB3-bus, and compared against the pattern. A channel-specific error message is generated if readout data deviates from the pattern that was set.

6.1.1.9 Equal random user-defined patterns

The method of setting output data, and checking the readout data is equal to what is described in section 6.1.1.8. The difference is that the patterns are pseudo-randomly generated before they are stored in the model memory. This is all confined in a loop. A generic in the top of the testbench the number of iterations of this test. In this loop, the error-data control register is polled after each iteration to see if data is latched into the error-data register. A commented section can be uncommented to test if this polling works as intended, i.e. if data changes unexpectedly in a channel, an error message should be generated.

6.1.1.10 Unequal random user-defined patterns

This test is the combination of what is described in sections 6.1.1.7 and 6.1.1.9. That is, unequal, pseudo-random patterns are set as output patterns in the model. A generic controls the number of iterations.

6.1.2 Physical test sequence

6.1.2.1 AD9257 model

The test sequence implemented in software complies with verification plan in appendix 7.1. Opposed to the test sequence explained in 6.1.1 on page 78, this was not updated to include the functionality that was added at a later stage. Checking of default bus-input values and default register values were also excluded. The focus here was to check readout and SPI master functionality. Thus, the tests using random data, built-in patterns, and equal and unequal user-defined patters the same. In this sequencer, all memory addresses defined for AD9257 was written too with corner case data and pseudo-random data,

Functionality that creates log was made. This was inspired by the log that is generated with the functionality in Bitvis UVVM library in the computer-aided tests. Verbosity control is not included. Thus, what is tested, and correct and incorrect results are printed in the terminal while the test is running. The total number of errors for SPI and likewise for readout is also printed once the test is complete.

Though not included in the test sequencer, the circuit board control signals can be verified manually by altering and checking their state as explained in . They are also connected to LEDs on the SF2-dev-board for visual inspection. The output frequency of the dynamic CCC `PLL_10MHz` is connected to external jumpers, and must be verified by measuring with an oscilloscope.

6.1.2.2 AD9257 testboard

As mentioned in section 5.7, the software test sequencer should not be used against the actual AD9257 on the testboard. Instead, the menu choice for readout functionality where all readout errors are tracked and logged can be used. At this stage in the design process, the control signals and output clock are in actual use. Thus, the correctness of these signals must have been verified at a previous stage. However, they can still be verified as explained in the previous sub-section 6.1.2.1.

6.2 Results

The following sections present the results from the testing that was done in this thesis. Appendix C.2: Simulating in Modelsim explains how the different tests are started in Modelsim, and what must be done to setup and test the different systems on the SF2-dev-board.

6.2.1 Custom SPI-master

The custom SPI-master that was discussed in 4.2.3 was made at an early design stage, and was quickly replaced by the other SPI-master methods. It was however tested in a testbench where Bitvis UVVM utility library was used as the framework. The test performed were to shift 24, 32 and 40 bits to a simple SPI-slave. After data was written to the slave, it was read back, and compared against the written data. No errors were reported at the end of the test. However, some alterations must be done before this will work against the model of AD9257. This master transfers all bits without stalling between bytes, and a model of a bidirectional I/O which connects to SPI ports `din`, `dout` and `sdio` is implemented in the component. If it at some point becomes interesting to use a custom SPI, this component would be a good starting point.

6.2.2 Readout methods

Some of the readout methods that were discussed in section 4.1 were tested in addition to DDR-fabric, which was used in all other systems. The additional methods are DDR-MSIO where the data and clock signals are directly connected, and DDR-MSIO where DCO is inverted by an inverter. A separate testbench system was not made for this. Instead, it was instantiated in the readout wrapper and compiled when it was simulated in Modelsim. This is not an advisable verification method, but creating a separate testbench was not prioritized. The results, however, confirmed that the DDR-MSIO with an inverted DCO works, while the DDR-MSIO where DCO is used directly failed on each readout.

6.2.3 Development testbench

The development testbench confirmed that the digital designs, and system setup, behaved as intended. Hence, no errors were reported. All testing that is done for the readout logic, and for the SPI master is thus verified. Also, all the tests that required manual

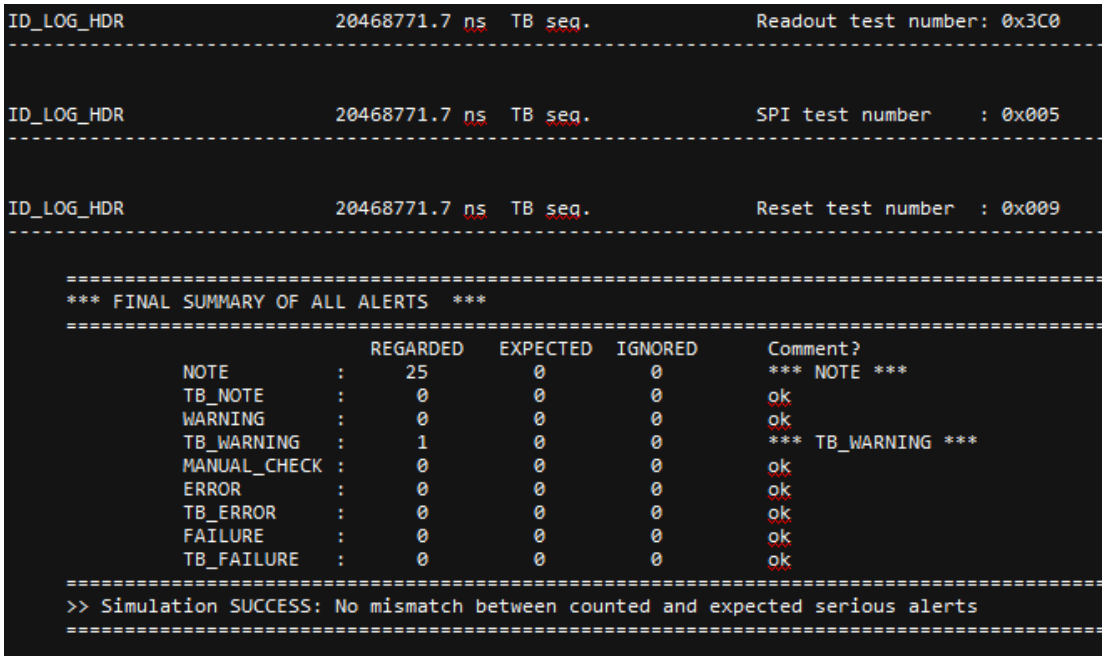


Figure 6.1: Testbench summary after test

verification in the Modelsim waveguide were deemed to work as intended. This concerns the clock frequencies of the dynamic PLL, and the control signals. Some tests did however return unexpected results. Figure 6.1 shows the summary in the log from the last performed test in this system. Note that the number of iterations has a limit. The frequency of `DCO`, causes a period where the number is irrational. Even with maximum time resolution in Modelsim, the edges will be skewed compared to FCO and the data. Thus, the readout will fail if the simulation runs long enough.

The `TB_warning` concerns Bitvis logging of time stamps, and can be neglected. The reported `Note` concerns reading default register data in `PLL_10MHz`, and `in coreSPI`. `Note` was set as the alert level for failed tests that was deemed to be of no concern. The following sub-sections discuss the reason behind the reported `Notes`. These notes occur in all the computer based test-systems that were made. Some other issues were present in `coreSPI`. These are also mentioned below.

6.2.3.1 Dynamic CCC

The reported `notes` were caused by two reasons. The first reason is that the PLL only uses the eight least significant bits of the data bus. The APB3-interface to the instance was 32-bits. A selection from the log shows the first reported note. All bits that were not driven was set to high impedance `'Z'` in the bus handler. The second reason is caused by the illegal values represented by `'X'`. Some bits that should be `'0'` have this value. Otherwise, the 8 bits would have been correct. This can happen if a signal is not given an initial value, or if it is driven simultaneously by two sources. `PLL_10MHz` was reset before any tests were made, and the read-bus was not driven by any other sources. No further investigation was done, as the default output frequency of the component is correct, and as all configuration changes worked.

```
*** NOTE #1 ***
UVVM: 251.785773 ns APB BFM
UVVM: apb_check(A:x"04", x"0B") => Failed.  slv Was x'ZZZZZZXB
(b"ZZZZZZZZZZZZZZZZZZZZZZZZXXX01011")'.  Expected x'B'
UVVM: Checking default FCCC_RFMUX_CR (RFMUX configuration register)
value
```

6.2.3.2 SPI master

The reported `notes` were caused by wrong default values in the registers compared to the values in the coreSPI manual. Some values were simply different, some were undefined, while some were in an illegal state. The core did however work for its purpose of confirming that SPI protocol Motorola mode 0 works as intended. Some other problems with the core was found. A `command` register with the purpose of resetting the TX- and RX- FIFOs does not seem to work. Writing to it had no effect. This caused a problem since the FIFOs needed to be cleared after each transfer. This was mitigated by asserting the reset signal of the core between each transfer. This did not change the configuration of the core. No further analysis of the problem was made, so it is possible that something was misunderstood. This must however be done if this core should become the SPI master of choice. It should also be mentioned that Microsemi has made a new version of the core, but this was not tested.

6.2.4 FPGA-internal SoC

As previously shown in section 5.3, testbenches were made where pre-synthesis-, post-synthesis-, and post-layout- VHDL were tested. Furthermore, when the SoC was implemented on the SF2-dev-board, testing was continued with the software test sequencer, and by capturing different patterns over time, while error events were monitored. Table 6.1 lists the results. The systems marked valid reported no errors for the tests presented in sections 6.1.1 and 6.1.2.1. As shown, the post-synthesis system did not work. This is discussed in section 6.2.6.

Table 6.1: Results of FPGA-internal tests

Test system	**Result**
Pre-synthesis	Valid
Post-synthesis	Not valid
Post-layout	Valid
Soc	Valid

6.2.5 FPGA-loopback SoC

Table 6.2 lists the results of the loopback systems. For this design, the post-synthesis testing worked, while post-layout failed. This was also the case for the test explained in section 6.2.6. Thus, this is discussed there. The tests performed in the SoC are marked

Table 6.2: Results of FPGA-loopback tests

Test system	**Result**
Pre-synthesis	Valid
Post-synthesis	Valid
Post-layout	Not valid
Soc	Semi-valid

semi-valid. The reason for this is that only the SPI-interface worked when it was tested. A selection of the test-sequencer log is shown below. The readout data seemed to be skewed compared to the output pattern from the model. The reason is believed to be caused by the additional delays on the outputs of the CCC that generated `DCO` and `FCO`. This was mentioned in section 5.3.2. At the time of testing, an attempt to correct this was not prioritized.

```
Channel 0:
Readout data 0x02aa != test pattern 0x2aaa => ERROR!
Channel 0:
Readout data 0x0555 != test pattern 0x1555 => ERROR!
```

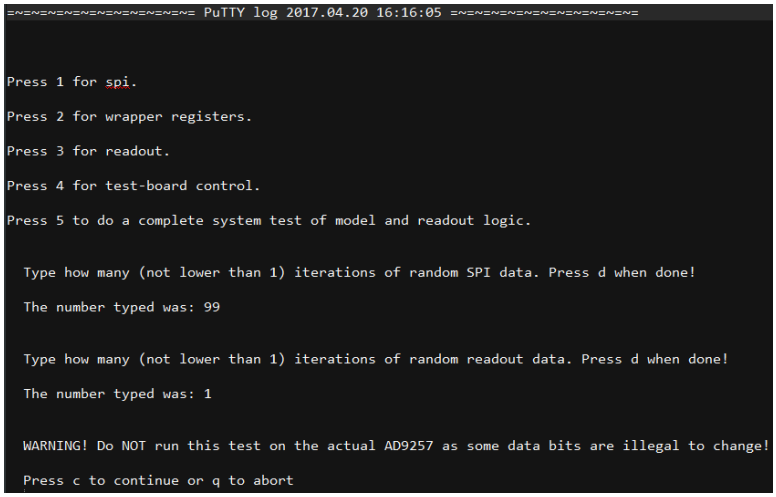
The main reason behind the implementation of this system on the SF2-dev-board to verify that the output-enable signal from the MSS SPI only was asserted when data was written to the slave. This was primarily questioned because this was not the case for coreSPI. For this core, the signal was asserted when both writing and reading to the slave. As this signal is used to control the direction of the bidirectional I/O, and the voltage-level translator where `SDIO` passes through, it had to be verified. The test sequencer was configured to test SPI with 99 pseudo random patterns in addition to corner cases for each address. As shown in figure 6.2, no errors were reported. If the signal had been asserted for both read and write, `SDIO` would have been driven simultaneously by the master and the slave. A correct readout is unlikely if this was the case. Therefore, it was concluded that the output-enable signal works as intended.

6.2.6 AD9257 testboard SoC & testboard

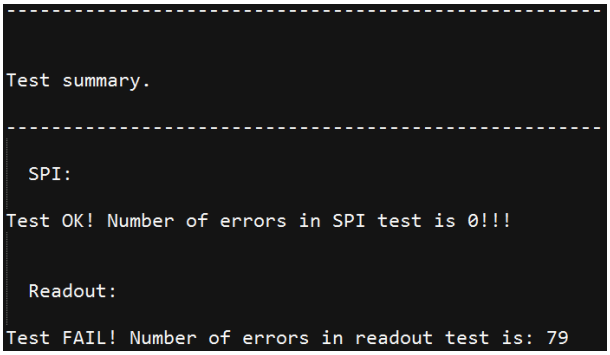
All test results for this system is shown in table 6.3.

Table 6.3: Results of AD9257 testboard SoC tests

Test system	**Result**
Pre-synthesis	Valid
Post-synthesis	Valid
Post-layout	Not valid
Soc	Not properly verified
Testboard	Currently not operational



(a) Setting number of pseudo-random iterations



(b) Test summary

Figure 6.2: Physical test sequecer for FPGA-loopback SoC

The pre-synthesis and post-synthesis testing for AD9257 testboard SoC reported no errors. The post-layout system, however, did not work. This was also the case for the FPGA loopback design, while in the FPGA internal design, it did not work for the post-synthesis system. Some effort was put into finding the cause behind this, but it was not discovered. The problem for each failed system was that certain signals did not behave as they should. The only difference between the different testbenches for each design stage was that the instance `io_top_tb` was replaced with the files that were generated in Libero SoC. Thus, it should not be caused by incorrect signal connections. It is believed that the way the top levels were created, and how they were added to Libero SoC before it was synthesized and placed and routed, caused the problem. The signals that should have been connected to the MSS, were instead added as unconstrained ports. It is a chance that the ports were neglected by the tools because of this. Since pre-synthesis testing was verified, which was the case for the FPGA-internal SoC, it is likely that it will work as intended when it is implemented.

Before the testboard was connected to the dev-board, measurements were performed to check if the board was operational. For the results mentioned, all DC-voltages were measured with a multimeter. AC-signals were measured with an oscilloscope. The voltage supplies pictured in figure 7.5, dropped voltage over the resistors that are in series with the outputs. To mitigate this, they were replaced by 0 Ω resistors. The

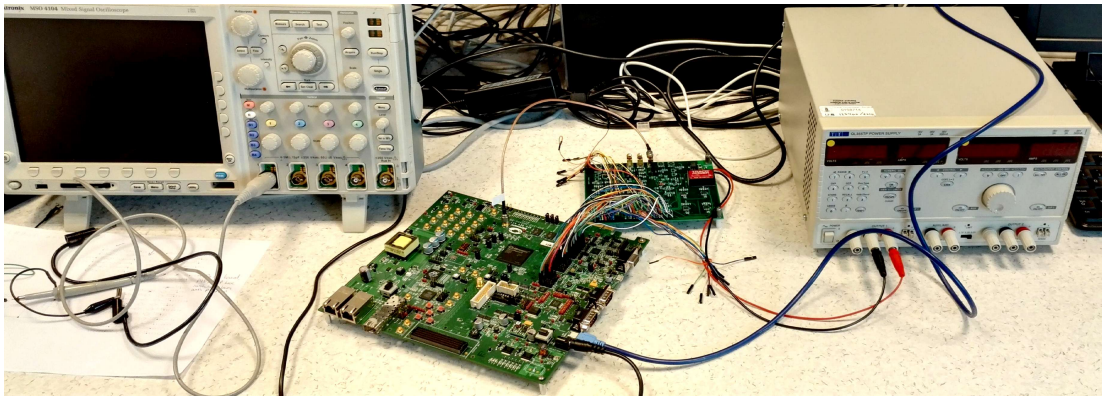


Figure 6.3: Testboard (top) connected to SF2-dev-board (bottom) where the SoC is implemented on the FPGA.

`VCM_buf` voltage was then measured on the jumpers in figure 5.35. This was confirmed to be about 0.9 V, which is the correct value. As this voltage is delivered by the ADC through a buffer, it confirmed that these devices receive power. Next, the common-mode voltage on one of the differential outputs from AD9257 was measured to be about 1.3 V. This is correct in regards to the voltage listed in the datasheet [24].

Next, the testboard was connected to the SF2-dev board where the SoC was implemented. The setup is displayed in figure 6.3. A `Gnd` connection was made by connecting the channel F SMA connector to a SMA connector on the dev-board. The first test was to read the device ID from the internal memory in AD9257 via the SPI-interface. No data was returned. Next, the control signals that enable the on-board clock, and switch between the clock inputs on the LVDS multiplexer, were measured on the testboard. Both were confirmed to operate as intended. I.e, when they were toggled from the user interface, the voltages switched to correct values. This also confirmed that the `gnd` connection was valid.

The last test that was performed was a measurement of the input clock to AD9257. Only one of the signals of the differential pair was measured. The signal looked random, and not like a clock signal. However, if the control signal to the clock was set to disable the clock, the signal disappeared. This means that the signal that was measured originated from the on-board-clock, and was put on the output of the LVDS multiplexer. This also means that the signal propagated through the LVDS buffer.

When the clock was measured, the readout monitoring functionality was enabled in the user interface. While measuring, the readout data changed sporadically. This indicates that the ADC transferred fixed data when it was not disturbed. When only one signal of the differential clock was measured, the phase relationship of the differential signals might have been changed due to the probe impedance. This can explain why the output data changed. When no measurements were performed, the readout data were at a fixed value. This did not change even when the wires between the boards were touched. Unfortunately, the initial data output before the measurements was not checked. There was not enough time to perform any more measurements. A test plan has however been devised for further measurments on the testboard. This is located in D.1: Testboard test plan.

CHAPTER 7

Discussion & conclusion

In this thesis, an overview of all the functionality in the DEEP measurement system was made as it had not been done prior to this thesis. It will be valuable for the future work in the DEEP project as it visualizes the system, and as it discusses the measurement flow from start to finish. After a thorough search for an applicable ADC, AD9257-EP is presented as a potential candidate. Possible ADC control- and readout- interfaces between it and the FPGA-candidates were discussed in relationship to the measurement system. The chosen methods will depend on the FPGA that will be used in this project. All interfaces of interest were at some level realized and tested. Hence, the work done in this thesis has provided solutions that are independent of the final choice of FPGA.

The ADC in question is a high-reliability device, but it is not qualified for space. Therefore, it must be upscreened to higher reliability requirements before any conclusion can be made. If it is proven to work, it is a great alternative due to the high number of channels, serial outputs, and low power consumption. In order to qualify the ADC, an evaluation system composed of an FPGA SoC, and a testboard that contains the ADC in question were designed and realized. The system is controlled and monitored from a user interface on a host computer. Functionality of the system includes controlling and configuring the ADC, monitoring the readout data, board component-control, generation of an adjustable sample-clock, and automatic verification of readout data. The testboard also contains circuitry that allows testing of various single-end-to-differential-conversion methods. This system can be used to verify that the interfaces that have been designed work against the actual ADC. It can be used to characterize the ADC by applying different analog input signals. This also includes characterizing the different methods of SED-conversion. It can be used to monitor the ADC while it undergoes environmental stress-testing, such as radiation testing, and temperature testing. As the current system runs on a SmartFusion2 development board, it can also be used during environmental testing of the FPGA.

Proper verification before implementation was of great focus in this thesis. Computer-aided- and physical- test-and-verification-systems were designed and used from the start of the design phase. A synthesizable model of the ADC was designed so that the test environment was close to the real-life application in all verification stages. The structure of the testbench is a model of the SoC that is implemented on the FPGA, as a test sequencer in combination with BFMs act as the software-CPU system that is

implemented. All computer-aided testbenches are simulated in Modelsim by running scripts. Thus, verification before implementation is fully automated. A log containing the results of the test is generated once the test is finished. Evaluating the designs, and potential future changes are because of this a simple process. Programming files for the hardware that must be implemented on the SF2-dev-board are also provided along with detailed descriptions on how the designs can be implemented and tested. The information that is printed in the user interface when testing in the physical testbenches is presented in a similar manner as the computer-aided logs for easy comparison. A verification plan was made at the start of the design phase. The combined testing confirmes that the readout- and control- interfaces, as well as the other included functionality in the SoC work as intended.

Testing for either post-synthesis and post-layout failed for all systems. The method chosen for testing by removing the MSS interferes with the system. When the post-VHDLs were made, the wires from the MSS had to be interpreted as FPGA I/Os instead of internal signals. Thus, the design was not equal to the implemented design. For post-synthesis, this should not have a great impact, as the purpose of this test is to confirm that the VHDL-functional code is converted to vendor-specific cells while retaining the same functionality. For post-layout, however, this means that the physical structure on the FPGA changes. Thus, testing this with timing information would not give accurate results. For this reason, this step was not performed for any of the designs. In hindsight, the designs should have been implemented after the pre-synthesis verification. If they had not worked, post-synthesis and post-layout simulation could have been performed.

The testbenches have some small issues that should be mitigated. Because DCO and the system clock have their period represented by an irrational number, their relationship with each other, and FCO will change over time. One way could be to generate a CCC component like the one that was used in the designs and implement this in the testbench. Another option is to implement skew-correction functionality in the clock generation model that was designed. The pseudo-random generator used in the software random tests seems to be limited. During SPI testing in the FPGA-loopback project, many of the same patterns appeared when the log was evaluated. It should also be revised if the test patterns that have been used provide a sufficient fault coverage.

There was not enough time to properly test the final system. Some initial measurements show that the first version of the board has the potential of being fully operational. It powers on, a sample clock is present, and certain control signals from the SoC were confirmed. The ADC also generated output data when the sample clock was measured. The ADC did not return data when the internal memory of the ADC was read. Thus, the control interface and its belonging testboard circuitry must be checked. A test plan for further testing is made, and is included in D.1: Testboard test plan. If the final system is operational, it can be used to perform further testing of the AD9257-EP.

If the ADC is proven to not be suitable for the environment, a new ADC must be found. In any case, the discussions regarding the Microsemi FPGA technology for readout and control still remain valid. Also, the designs, design structure, and the test methodology provide a basis that can be used further in the project. The VHDL designs are properly documented, both by comments in the VHDL files, and by the structural schematics that are presented in this thesis. Thus, the existing designs can be modified to work with other ADCs. The testbenches can also easily be modified to fit new designs.

7.1 Future work

Before any decision can be made about AD9257-EP, it must be upscreened. Testing how it operates in a radiation environment, i.e. testing how sensitive it is to SEE, and the radiation dose it can handle must be performed. SEE is important to test since it can alter the configuration memory of the ADC. The total dose will determine if it can operate throughout the duration of the project. If SEE is an issue, it must be determined if rewriting the configuration memory often can result in reliable operation. Using the SPI ports can, however, degrade the converter performance [24]. Therefore it must be determined if this is acceptable.

The qualification data from when it was evaluated by the vendor must be evaluated so that the additional testing that must be done can be determined. E.g. it must be determined if the temperature cycling testing is sufficient in regards to the temperature cycling it will experience in the LEO environment. It must be determined if the plastic package can be used, since e.g. outgassing might be a problem if there are optical systems on the satellite. Mitigation techniques for plastic packages in a vacuum must be investigated. It should also be looked into if the chip can be delivered without the plastic package so that it can, if possible, be packed in a ceramic or metal package. Vibration testing must be done to determine if the leadless package style handles the vibrations during launch.

If it is deemed to be reliable enough, testing of the readout and control interfaces should continue. Which FPGA that will be used should be determined, so that a firm decision can be made on how the SPI-master will be implemented. If the inclusion of MSS brings other advantages, it is considered as the best alternative. Once the CSA-and-shaper has been determined, if SED-conversion is required, the method of achieving this must be firmly decided. Most likely, the deciding factors will be if the noise that the active devices contributes is acceptable, and if the low frequency response of the passive devices is acceptable. If the testboard is functional, it can be used to characterize the different methods.

Should the AD9257-EP be determined to not be reliable enough for this project, an alternative must be found. It is possible that other non-space devices than the ones that were evaluated in this thesis exist or were made available in the midst of this thesis. If not, space qualified devices must be looked at. If no alternatives are found, the method of digitizing the sensor data must be reevaluated. Should however another ADC be found, it will be necessary to modify the VHDL interfaces to fit the new ADC.

As for the other parts of the measurement system, the circuit board components must be specified. The functionality that will be implemented on the FPGA, must also be determined. It must be determined if it is possible to implement everything that was presented in section 2.3 with the number of available programmable blocks and DSPs in the FPGA that is chosen. As triple majority voting probably must be used for SEU mitigation, this will, depending on the method of implementing it, reduce the number of available blocks by a factor of three. Error detecting and correcting codes, and possibly scrubbing, must also be used for the memories. Another aspect of what to implement, is that it is crucial that the system works at all times. The complexity of the system should be kept to a minimum, as more things can go wrong, and as it is more difficult to properly verify complex systems. Also, the active area on the FPGA should be kept to a minimum, since it reduces the probability of interacting with the radiation.

Appendix A: Method

A.1: Finding an applicable ADC

An extensive search was conducted to find an applicable ADC. Table 7.1 lists the manufacturers and distributors visited during the search. The approach used to find the components was to first visit the distributor pages, and using their parametric search engines. The selection of ADCs is vast, so applying filters to narrow the search was necessary. For the most part, only components that met the military temperature criteria were considered. Some devices with automotive temperature were also considered.

The web pages of the listed manufacturers were also visited. When visiting these web pages, it was firstly determined if the manufacturer produced ADCs beyond commercial applications. If this was the case, parametric searches, if possible, were used. In addition, product pages that contain information about devices which could meet specifications were looked at. Some manufacturers also have special guides for their high-reliability devices. These were also looked at. All devices that showed promise were listed in a table so that the best alternative could be found by comparison. ADCs that were considered as alternatives are listed in tables 7.2 and 7.3.

Table 7.1: Manufacturers and distributors

Analog devices	Exar
Texas instruments	IDT
Honeywell	Nuvoton
Intersil	NXP
E2V	ON Semiconductor
Linear technology	Silicon labs
Microchip	STMicroelectronics
Maxim integrated	Cirrus logic
Advanced linear devices	digikey.com
AMS	mouser.com
CEL	parts.io

Table 7.2: Alternative ADCs, enhanced products

(a) Enhanced products

Name	AD9257-EP	AD9253-EP	ADS6445-EP	ADS6444-EP	AD9266	ADS5463-EP	ADS4245-EP
Rating	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC
Operating temp[C]	-55 to 125	-55 to 125	-55 to 125	-55 to 125	-55 to 125	-55 to 125	-55 to 125
Bit resolution	14	14	14	14	16	12	14
Effective Bit, ENOB	min:11.5 typ:~12	min:11.5 typ:~12	10,95	11,3	12,30	10,50	11,50
Conv. rate MSPS[max/min]	65/10	125/10	125/5	105/5	65/3	500/20	125/80
Input bandwidth[MHz]	650	650	500	500	700	2200	2V:400 1V:600
Supply [V]	1,8	1,8	3,3	3,3	1,8	5/3,3	1,8
Total Power[mW]	55/ch @65Msps. Tot:547	110/ch @125 Tot:480	1800	1500	122mW	2575	277 @max. digital power:179
Input Voltage range	2	2	Diff: 2	Diff:2	2	Diff:2,2 Single:1,1	2V
Channels	8 separate	4 separate	4 separate	4 separate	1	1	2 separate
Input Interface	Diff	Diff	Diff	Diff	Diff	Diff.	Diff
Output Interface	serial LVDS	serial LVDS	Serial LVDS	Serial LVDS	interleaved parallel	Parallel LVDS	LVDS - Parallel, Parallel
Package	LFCSP	LFCSP	QFN	QFN	LFCSP	TQFP	QFN

(b) Enhanced products cont.

Name	THS1206-EP	ADS5500-EP	THS1408-EP	AD9648-EP	ADS5444-EP	ADS5440-EP
Rating	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC	EP/AQEC
Operating temp[C]	-55 to 125	-55 to 125	-55 to 125	-55 to 125	-55 to 125	-55 to 125
Bit resolution	12	14	14	14	13	13
Effective Bit, ENOB	10,17	11,30	11,20	11,80	11,2	11,4
Conversion rate MSPS[max/min]	6/channel	80/10 w.o DLL	8/3	125/10	250/10	210/10
Analog Input bandwidth[MHz]	Diff:96 Single:54	750	140	650	800	800
Supply [V]	5	3,3	4	1,8	5	5
Total Power[mW]	216	875	360	300	2370	2350
Input Voltage range[V_p-p]	diff:2 single:3,5	2,3	3	2	2,2	2,2
Channels	1 @6Msps	1	1	2	1	1
Input Interface	single_end or differential	Diff	recom: Diff.	Diff	Diff	Diff
Output Interface	Parallel	Parallel	Parallel	interleaved and serial? *	parallel lvds	parallel lvds
Package	TSSOP	HTQFP	PQFP	LFCSP	HTQFP	HTQFP

Table 7.3: Alternative ADCs, military and automotive

Name	AD872A	AD871	AD10242	LTC2311	LTC2315-12	AD7356
Rating	MIL-883B	MIL-883B	MIL-883B	industrial & automotive	automotive	
Operating temp[C]	-55 to 125	-55 to 125	-55 to 125	-40 to 125	-40 to 125	-40 to 125
Bit resolution	12	12	12	14	12	12
Effective Bit, ENOB	N/A	N/A	N/A	N/A	11,7	N/A
Conversion rate MSPS[max/min]	10 /	5/	40/5	5	5	5
Analog Input bandwidth[MHz]	35	15	60	100	130/5	110
Supply [V]	Vdd:5V Vss:-5V	Vdd:5V Vss:-5V	Vdd:5V Vss:-5V	5/3	2,7-5,25	2,5
Total Power[mW]	1030	1300	2000	60/40	100	60
Input Voltage range	+-1V	+-1V	+-2	8V	2/4	VCM ± VREF/2
Channels	1	1	2	1	1	2
Input Interface	Diff and single	Diff and single	Single end	Diff	Single end	Diff
Output Interface	parallel	parallel	parallel	Serial cmos/lvds	Serial SPI	serial SPI etc..
Package	CLCC	CLCC	CLCC	MSOP	TSOT	TSSOP

A.2: VHDL design

The content in this section is based on content from [51] and [10].

VHDL is a programming language that is made to describe digital logic. Certain parts of the language can be used to implement digital logic on an FPGA. Other parts cannot be realized, but is useful when models of digital electronics must be made. It also enables testing and verification of logical designs on a computer before the design is implemented. This is done by enclosing the designs that must be tested in a testbench, where stimulus to the designs-under-test is created. Thus, by combining all parts of the language, reliable digital systems can be made.

Methodology

Designing in a structural manner is vital in order to create a reliable system. A system is composed of three domains: functional, structural and geometric. Each domain describes different aspects of the system, but together they define all aspects. Gajski-Kuhn Y chart, depicted in figure 7.1 can be used to visualize this concept.

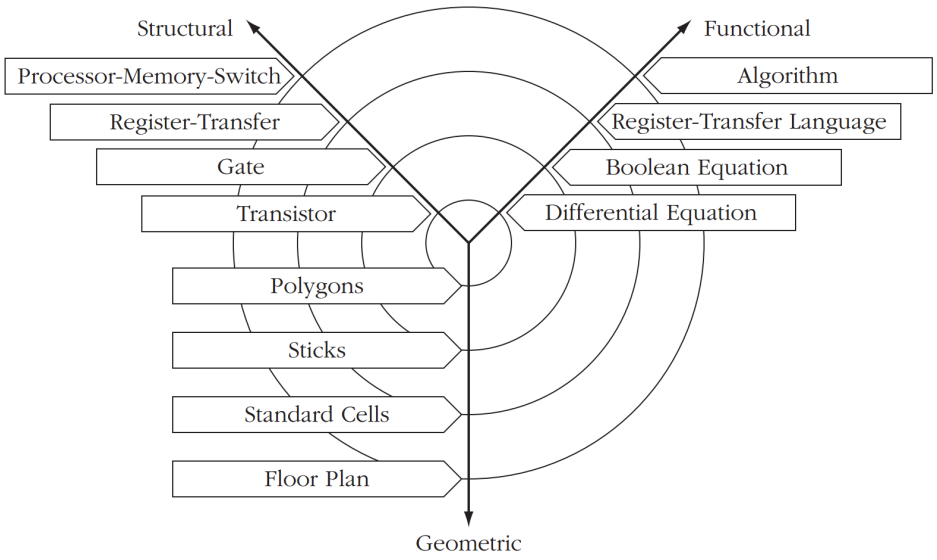


Figure 7.1: Gajski-Kuhn Y[51]

The chart is composed of the three domains. It also shows different abstraction levels of the design. The functional domain describes the functionality of the system. The structural domain describes the structure chosen to achieve the behavior at the same abstraction level. The geometric, or physical domain describes the physical placement of the structure. Once all domains are defined in one abstraction level, it can be repeated at a lower abstraction level.

Considerations for FPGA design

Structure of an entity

An entity in VHDL is composed of an entity declaration, and an architecture body of

the entity. This can be looked at as a box with input and output ports. The entity declaration specifies the ports, or interfaces to the box. The architecture body describes the internal functionality of the body, which connects to the ports of the entity.

There are many ways to create a component. The most basic component has an architecture with one process, describing one sequential or combinational function. An architecture consisting of one or more processes is called a behavioral architecture. Including to much code in one file, or creating different functionality in one file makes the design difficult to interpret. Hierarchy is a method of preventing issues like this. Hierarchy is applied by dividing the system into separate modules. These modules can be connected together by instantiating them in a higher hierarchy file. That is, instead of having an architecture with many processes, it is composed of substantiations to other entities. Connections between the entities are created by connecting signals in the top level. An architecture composed of one or more instances of other entities is called a structural architecture. It is also possible to create an architecture that is a mixture of behavioral and structural. I.e, the architecture is composed of both processes and instances of other entities. In the cases where multiple instances or processes are used, they can be interconnected by creating signals.

Logic design method
Circuits can be designed at different abstraction levels in VHDL. A circuit can be described in a behavioral manner. In this case, the functionality of the system is described using statements similar to statements used in software languages. Examples are if-else and loop statements. The lowest level of logic description in VHDL is at gate level. I.e one can create a logic system by interconnecting the combinational and sequential circuits necessary for a specific function. Both methods can be used to represent the same logic functionality. When writing RTL, the user specifies the architecture of the system. This can be time consuming, but the logic can be exactly as specified. When writing behavioral, the tool creates the system based on the behavioral description. This requires less work, as the architecture is chosen by the tool. If the interpretation of the behavior is correct, or if the best possible logic architecture is chosen, will depend on the quality of the tool.

In either case, the design must be verified, for both functional requirements and timing requirements. Utilizing the concept of regularity increases the reliability, and design time of the system. This is the concept of using the same component everywhere the functionality of a certain component is needed. This saves time compared to creating many different components when the end functionality is the same. It also saves time since the component only has to be verified once. In order to use the concept of regularity, important characteristics of entities must be clearly defined. As examples, the port names should be understandable, constraints for the module must be available and so on. This is the concept of modularity. That is, a module must be designed in a way that makes it possible to use it as it was intended, every time.

A.3: Design flow

“B.2: Software” in Appendix B: Tools should be read before this section is read as this chapter describes the software tools that were used during development. The design flow used in this thesis is pictured in figure 7.2. This is the design flow in Libero Soc v11.7,

which was used during development systems that were implemented on SmartFusion2. This is a general design flow for FPGA design. The design flow chosen for this thesis mostly follow this flow. Specifications of how the VHDL designs were implemented in this thesis is described in the following paragraphs.

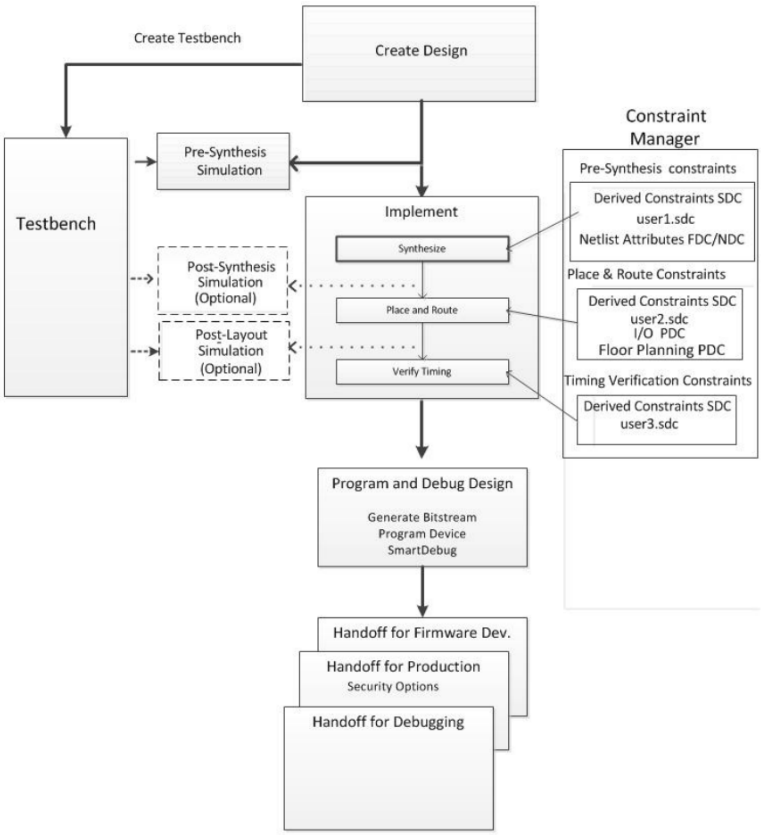


Figure 7.2: Libero Soc v11.7 design flow [52]

Before a design was created, requirements of the design were specified. If IPs supplied from Microsemi were used, the components were generated separately in Libero Soc. The generated VHDL of the components were then connected together with custom made logic by instantiating all modules in a top level file. Any text editor can be used for this. In this thesis, Notepad++ was the preferred editor. The reason behind this approach is that it is not necessary to connect custom logic to IPs in the design canvas if the project is remade. A better way would be to create scripts that execute all the stages, but this was not prioritized in this thesis. Version control is obtained by using GIT. Before designs were implemented on SF2, functional verification was performed. It was chosen to create the testbenches, and perform the verification outside of the Libero SoC design flow. Details concerning verification are located in "A.4: Verification".

Architectural structures in this thesis are mostly behavioural or mixed. As for the logic design method, behavioural descriptions of synthesizable code are utilized unless it fails to meet design requirements. If an RTL description is not to complex for a component, this will be utilized. Designs that were made in this thesis also follow coding guidelines at the department of physics and technology, which is where this thesis was assigned. It should also be noted that in this text, component, entity, instance, module and system

are used to describe either a part of a design, or the entire design, interchangeably.

If a design will not be synthesized, non-synthesizeable VHDL is utilized. E.g, the type *time* can be used. Regarding the domains in figure 7.1, structures are made for designs to the level the design was visualized. The structures will be named with the functional description, i.e the name of the VHDL-file or files. The physical domain will be handled by a place-and-route tool unless design requirements are not met. The concepts of hierarchy, regularity etc. will be utilized to structure the designs in an understandable way. Also, all sequential elements in the designs are resettable through an active-low reset signal. Note that the reset signals are not included in the RTL-structrures that were made.

A.4: Verification

Functional verification plan

In order to obtain enough confidence about the correctness of the system, the following testing must be performed:

AD9257 readout logic

- Check that readout can be performed at a rate of 70 MHz.
- Check that deserialization can be performed at 10 MHz.
- Check readout data for all channels.
  - Data read out must be of different composition. Patterns that must be checked are:
    - All zeroes.
    - All ones.
    - Alternating ones.
    - Random patterns.

AD9257 register access via SPI

- Write to all memory addresses specified in the datasheet.
  - Data written must be of different compositions. Patterns that must be written are:
    - All zeroes.
    - All ones.
    - Alternating ones.
    - Random patterns.
- Read from all memory addresses specified in the datasheet.

Method of test and verification

To test in an application-like environment, a model of the ADC that is chosen must be developed. The tests in the section above, in combination with the model, are to be performed in all levels of the design phase:

- Computer-aided verification using Modelsim for:
  - Pre-synthesis firmware.
  - Post-synthesis cells.
  - Post-layout cells.
  - Post-layout cells, with timing information.
- Verification of design implemented on an FPGA:
  - Internal connection of model and readout logic.
  - Loopback connection of model and readout logic, and model and SPI master where signals from the model are sent out of the FPGA, and connected directly to FPGA inputs connected to the readout logic.
- Verification against an actual AD9257 on a testboard.

Making a structured testbench can be very time consuming. Instead of building the testbench from scratch, Bitvis utility library has been used. The library provides useful functionality for making a good testbench. Some of the features are:

- Logging and alert.
- Procedures and functions with string handling, and log- and alert- integration.
- Scope and verbosity control.
- Provided BFM models of different interfaces.
- Provided SBI BFM can be modified to fit a user-defined BFM.

Example projects with scripts can be downloaded[1] and used as a basis for the testbench in this project. When verifying with Modelsim, it is required to use the Bitvis UVVM utility library as the framework. This gives the possibility of generating a log of the tests that have have been performed, as well as the results of the test. This log must show that all the steps in the verification plan have been executed. When testing the design implemented on an FPGA, a test, user-interface must be made. As when testing with Modelsim, a log should be generated here as well, so that it can be checked against the verification plan.

[1] www.Bitvis.no

Appendix B: Tools

This appendix lists the tools that were used to create and implement the digital systems in this thesis.

B.1: Hardware

- SF2-DEV-KIT-PP [47]
  - SmartFusion2 Development Board with M2S050T-FGG896. Shown in figure 5.2.
  - FlashPro4 JTAG programmer/debugger for the development board
- All work was done on a personal computer

B.2: Software

- Libero Soc v11.7 and supporting software. All software is explained in the following section
- Putty v0.67 as a terminal for serial communication between computer and development board.
- LTspice v4.23I for analog circuit simulation

Microsemi software was downloaded from Microsemi website with a free gold license. The software that was included and used, and brief descriptions are given below.

Libero Soc v11.7

Libero SoC is a design suite for Microsemi SoC FPGAs [52]. All steps from a circuit specification is made, to a functional system is implemented on an FPGA can be performed in this program. The functionality and programs that are included in the design flow are explained in the following paragraphs.

Creating a design
Creating a design with the MSS can be done with the Smartbuilder. This is a GUI setup

of the MSS, clocks, security, MSS peripherals, and more. In a few clicks, a component where the MSS and peripherals are connected. A global clock network, and reset network is automatically added. All components can also be connected together by choosing SmartDesign. Here, everything must be chosen and connected manually. However, more settings are available, such as PLL phase-and-delay settings. It is possible to first generate a system with the Smartbuilder, then convert to Smartdesign to access the additional settings. After the component is generated, it can be connected to custom logic in a design canvas. The final choice is to generate components separately, manually connect instances in a top level HDL-file, and add the files before synthesis. If the MSS is not used, this is the option to use.

Verification with Modelsim ME 10.4c
Modelsim from Mentor Graphics is the program that is used to verify that the logic functionality works as it was intended. Sequential and concurrent hardware is simulated. The propagation of signals can be viewed in a waveguide. A testbench can be generated from the system top level file in Libero SoC. The user can use this file to set the desired input stimulus to the design. Verification can be performed for pre-synthesis, post-synthesis and post-layout in the design flow. Scripts for compiling and simulation are generated automatically. Modelsim can also be used as a stand-alone program if the user wishes to perform outside of the Libero Soc design flow.

Constraining
Applying timing constraints is an important part of the design process. Without information about timing requirements, synthesis and place-and-route tools cannot implement the design in a reliable way. All clocks used for sequential operations must be specified. This is also utilized by static timing analysis tools to determine if signals propagate between sequential elements without violating setup- or hold-requirements. I/O-constraints must be added so that signals in and out of the FPGA are connected to the correct I/Os. Type of port, drive strength, pull-circuitry and other I/O characteristics are also defined here. Placement constraining can be done to specify where on the chip the logic will be implemented. Constraining tools are implemented in Libero SoC.

Synthesis with Synplify Pro J-2015.03M-SP1-2
Synthesis is the process of transforming technology-independent HDL firmware to technology-specific logic cells. Synthesis is performed by Synplify Pro from Synopsis. The program can also be used as a stand-alone program, or used within the design flow in Libero SoC. This program offers many helpful functions, such as optimization of the design, RTL-, technology- and FSM- structural views, and timing analysis. It is also possible to perform automatic TMR implementation. For the sake of this project, it was only used for synthesis and structural views within the design flow.

Place and routing
The place-and-route tool defines where the logic cells are placed on the chip, as well as the routing between cells. The requirements of the procedure can be timing or power driven. This tool is implemented in Libero SoC.

Static timing verification
This is performed after place-and-route to verify that the design meets timing requirements. This is done by testing if path delays violate setup- and hold- requirements on sequential elements. A violation must be mitigated by applying constraints, or by changing the directory. E.,g inserting a DFF in the paths where the propagation delay of the combinational logic causes setup-violation, or inserting delay in paths where

hold-time violations are reported. The program SmartTime is implemented in Libero SoC, and can be used to perform 4-corner analysis where temperature, voltage and process variation are the variables [40].

Verify power
This tool calculates both dynamic and static power consumption of the system. This is implemented in Libero SoC, and was not used in this thesis.

Implementing design on SF2 with FlashPro v11.7.1.11
Programming of the SF2 is done after timing has been verified. A bitstream of configuration data must first be generated. Programming is done by using FlashPro software, and a Flashpro4 JTAG programmer which is connected between a computer and the board. Programming can be done directly in the design flow, or by using FlashPro as a stand-alone program. In the latter case, a STAPL file must be generated. This can be done in Libero by exporting the bitstream.

Software development and debugging with SoftConsole v4.0
Software development and debugging is done in the program SoftConsole.

Appendix C: Project setup

C.1: Project directory structure

The top structure of the project directory is shown in figure 7.3. The folder *AD9257_rd_cont* contains all files that are made in this project except for the custom SPI-master which is contained in the folder *Custom_SPI_master*. *UVVM_v1_4_0* contains Bitvis UVVM utility library and BFMs used in computer-aided testing and verification.

Figure 7.4 shows the sub-structures in the folder *AD9257_rd_cont*. The folders are structured according to the SoCs that were made, and explained in chapter 5. Thus the files for the project FPGA-internal are contained in the folders *readout_model_internal_x*, etc. The development-testbench files are contained in the *model_x* folders.

All of the files for the readout system described in section 4.1.6 are located in *src\Readout*. The other folders in *src* are the top-level files for the different systems. *Tb* contains all testbench related files. The files of the model, and other testbench files are located in *common_tb_files*. Here, folders for the SoCs contain the testbench file, the top-level file for computer-aided testing, pre-synthesis, post-synthesis and post layout files. *Scripts* contains scripts to run all testbenches in Modelsim. This is explained in a following section. *Constraints* contains timing and I/O constraints. *Microsemi_cores* contains all the IP's that were generated and used. Most cores are common for the systems, except for the *MSS_x* folders. *Software* contains all custom made software, and Microsemi firmware that was used in the project. This is located in the *common_files* folder. The other folders in *software* contains SF2 programming files for all SoCs.

C.2: Simulating in Modelsim

It is recommended to use the same project directory structure described inC.1: Project directory structure. In this thesis, the folder *ADC_readout_control* was put in C:\. To simulate the designs: open Modelsim, and change the directory to the script folder of the testbench of interest, and run **compile_and_sim_all.do**. This will compile and simulate all the necessary files. When the simulation is finished, a log file containing the tests performed are generated in the script folder. There are certain settings that can be set in testbench files. E.g, for the development testbench, this is the file *tb\model_tb***AD9257_read_cont_tb**. The settings are:

- A generic can be changed to set the number of pseudo random patterns that are written/read via SPI. Setting this to 0 gives 1 iteration.

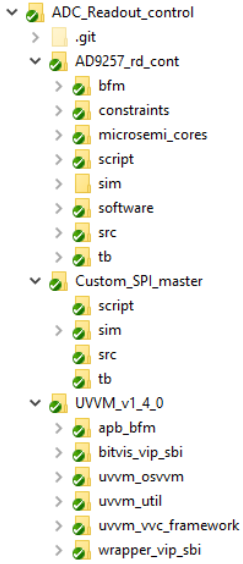
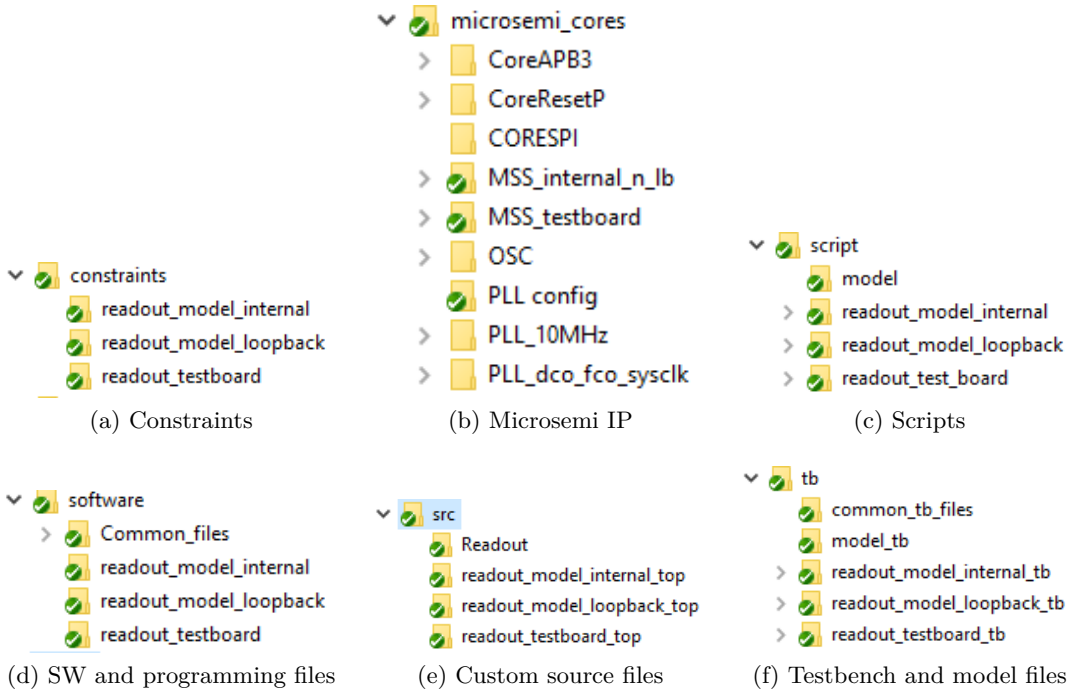


Figure 7.3: Project directory structure



(a) Constraints (b) Microsemi IP (c) Scripts

(d) SW and programming files (e) Custom source files (f) Testbench and model files

Figure 7.4: Project directory sub-structure

- A generic can be changed to set the number of skewed-reset iterations. Setting this to 0 gives 1 iteration.
- A generic can be changed to set the number of pseudo random patterns that are generated, set as output patterns in the model, captured and tested. Setting this to 0 gives 1 iteration.
- Logging abstraction levels can be changed by uncommenting or commenting verbosity control statements. Default is to only log the headers.

C.3: Configuration of SF2-DEV-KIT-PP

The configuration described in this sub section is the configuration used during this thesis. Before using the board, all jumpers were set to default positions as described in [47].

1. Set jumper across pins 2-3 on connector J129. Set jumper across pins 2-3 on connector J133. This connects the MSS UART RX and TX pins to the on-board FT4232 USB controller.
2. Connect a USB mini-B cable from the computer to the mini USB mini-B connector on the board. This plug connects to the FT4232 USB controller (FTDI interface).
3. Connect the JTAG Flashpro4 programmer from the host computer to the FP4 header on the board. This is necessary to program and debug the design.
4. Connect a power cord and toggle the power switch.
5. Program the SF2 as described in the following sections.

C.4: Running pre-made programming files on SmartFusion2

1. Perform the steps described in "C.3: Configuration of SF2-DEV-KIT-PP".
2. Open Flash-pro program.
3. Create a new project. The directory does not matter.
4. Add programming file.
   (a) Browse to ..*Software* and select the pre-made **.stp** file in the folder for the Soc of interest.
5. Press program.
6. When complete, close the program.
7. On the computer, use a terminal program. Putty was used in this thesis. Putty settings:

(a) Chose a COM port. Which one is correct must be checked at a later stage.

(b) Set speed/baud rate to 57600.

(c) Set data bit to 8.

(d) Set stop bit to 1.

(e) Select none parity and flow control.

(f) To enable logging, go to logging tab and chose "All session output". choose a directory to save the .log file. Also chose "Always overwrite it" under "what to do if the file already exists.

8. Create a folder called *SoftConsole* somewhere in the project directory. E.g *ADC_readout_control\SoftConsole.*

9. Open the program SoftConsole.

   (a) Choose this folder made in the previous step to be the workspace.

      i. Note! This workspace can be used for all SoCs.

10. Create a new project by following the steps from 1.7.3 to 1.8 in TU0546 [53].

    (a) Additional notes:

       i. Import the files from *..\software\common_files.* It is not necessary to add any new files.

       ii. newlib-nano was not enabled during this thesis.

       iii. In step 1.7.4, write M2S050T instead of M2S090.

       iv. If the "play" symbol is showing after debug was enabled, press it to start the program.

11. Run program.

    (a) If text appear in the putty terminal window, follow the instructions.

    (b) If no text appear.

       i. Press some key other than 1 or 2. If no text appears, restart putty with same setting except that another COM must be chosen. Re-do this step until text appears in the terminal.

       ii. If text saying wrong input appear in terminal, press 1 if model is used, or 2 if testboard is used. Difference is whether the test sequencer menu choice is an option.

12. Reprogram software.

    (a) If change is made in software, and software must be reprogrammed:

       i. Terminate debugging session by pressing the red "stop" symbol.

       ii. Right click "Debug" next to "C/C++" in the top right corner, and close the tab.

       iii. To program, chose Run -> Debug configuration -> Debug.

C.5: Generating programming files in Libero

These steps are shown graphically in TU0546 [53].

1. Open Libero and choose “new project”. Enter a project name and some directory.
2. Choose device M2S050T-FG896.
3. Set Default I/O technology to LVCMOS 1.5V. Set PLL supply voltage to 3.3V and power on reset delay to 100ms.
4. Choose “none” under design templates.
5. Press next under “add HDL source files”.
6. under “add constraints”, link the files for the Soc of interest in *..\constraints.*
7. Press finish. If prompted for what type of constraint flow to use, choose “use enhanced constraint flow”.
8. Select window “Design hierarchy”.
9. Press “File -> Link files ->” to link the files for the SoC of interest. E.g. for FPGA-internal:
   (a) “-> Create link folders” *..\src\readout.*
   (b) “-> Create link ” to the top level file to SoC of interest in *..\src.*
   (c) “-> Create link folders” *\microsemi_cores\CoreAPB3\4.1.100\rtl\vhdl\core.*
      i. right click the “work” library in the “design hierarchy” and add a VHDL library COREAPB3_LIB.
      ii. Mark the files from the APB-folder, right-click, and move to the new library.
   (d) “-> Create link folders” *\microsemi_cores\CoreResetP\7.1.100\rtl\vhdl\core.*
   (e) “-> Create link folders” *\microsemi_cores\OSC.*
      i. Remove duplicate with ending _pre.
   (f) “-> Create link folders” *\microsemi_cores\MSS_internal_n_lb.*
      i. Remove duplicate with ending _pre.
      ii. This one is also used in FPGA-loopback.
   (g) “-> Create link folders” *\microsemi_cores\PLL_10MHz.*
   (h) “-> Create link” *\tb\PLL_10MHz\common_tb_files* and mark files:
      i. AD9257_memory.vhd.
      ii. AD9257_rdout_model.vhd.
      iii. AD9257_spi_model.vhd.
      iv. blinking_led.vhd.
      v. spi_mem_rdout.vhd.
10. Switch to the “Design flow” window.

11. Press “manage constraints”.
    (a) Under “I/O attributes, assign io_top.pdc to place and route”.
    (b) Under “timing”assign:
        i. io_top_derived_constraints.sdc to all.
        ii. timing.sdc to all.
        iii. timing_min_delay to all but synthesis.
12. Click “generate bitstream”. This will execute all the necessary steps until this step in the design flow.
    (a) Timing and power can be verified after this under “Verify post layout implementations” in the Design flow window.
13. The design can now be programmed onto an FPGA in the following manner:
    (a) Before programming, configure the board as described in “C.3: Configuration of SF2-DEV-KIT-PP”.
    (b) Click “Run program action”. This programs the FPGA automatically. Following this, perform the steps from 7. in “C.4: Running pre-made programming files on SmartFusion2”.
    (c) Another method is to click “Export bitstream”. Select “STAPL” and press ok. This generates a programming file in the project directory. Then the steps in “C.4: Running pre-made programming files on SmartFusion2” can be performed.
14. To configure and use the software, follow the steps in 7.1.

C.6: Generating SoC testbench top levels in Libero

Perform same steps as in “C.5: Generating programming files in Libero” except for:

1. Use top level file for the project of interest in *..\tb* instead of top level in *..\src.*
2. Do not generate programming file or program SF2.
3. post-synthesis file is generated in project directory *..\synthesis.*
4. To generate post-layout file, press “generate back annotated files” in the “design flow” window.
    (a) File is generated in *..\designer\io_top_tb* in the project directory.

C.7: Generating MSS used in SoCs

1. Open Libero and choose “new project”. Enter a project name and directory.
2. Choose device M2S050T-FG896.

3. Set Default I/O technology to LVCMOS 1.5V. Set PLL supply voltage to 3.3V and power on reset delay to 100ms.
4. Choose “Systembuilder” under design templates.
5. No source or constraint files are added. Press next.
6. Press finish. If prompted for what type of constraint flow to use, choose “use enhanced constraint flow”.
7. Enter a name for the system, e.g. MSS_comp. Now the systembuilder interface is opened.
8. Under “device features”, press next.
9. Under “peripherals”, choose:
   (a) MSS peripherals.
      i. MM_UART_0. Press the settings icon. Change “connect to” from IO to fabric.
      ii. MSS_SPI_0. Press the settings icon. Change “connect to” from IO to fabric.
   (b) Drag “Fabric AMBA slave” from “Fabric slave cores” to “MSS_FIC_0 - MSS master subsystem” under subsystems. Press the settings icon. Change “interface type to APB3” and press OK.
   (c) Press next button.
10. Under “clocks”:
   (a) change the system clock to “on-chip 25/50MHz RC oscillator ”.
   (b) Set M3_CLK to 140MHz.
   (c) Click the “fabric CCC” Tab.
      i. Note! Do not add the clocks below if this is the core to the MSS in the testboard SoC.
      ii. Enable FAB_CCC_GL1. Set to 70MHz.
      iii. Enable FAB_CCC_GL2 Set to 10MHz.
11. Press the next button until “Memory map”. Press “Finish”.
12. The MSS is displayed in the design canvas.
13. Choose “Design hierarchy” in the design flow window.
14. Expand the MSS design. Right click the IP, and choose “convert to SmartDesign”.
15. Click the settings icon on the “CoreAPB3”.
   (a) Set APB master data bus width to 32-bit.
   (b) Under “Enabled APB slave slots”, choose slot 3 and slot 5.
   (c) Press OK.

(d) Right click the port “S3” on the output from CoreAPB3, and chose “Promote to top level”. Perform same step for “S5”.

16. Double-click the settings icon on xxx_MSS_0.

    (a) Double-click on FIC_0.

    (b) Under “FPGA fabric address regions (MSS master view)”, select “Fabric region 0 (0x300000000 - 0x3FFFFFFF)” and “Fabric region 0 (0x500000000 - 0x5FFFFFFF)”.

    (c) Press OK.

17. In the current tab, save (ctrl + s). Press the generate button. Close the tab when finished.

18. Double-click the CCC_0 to open the Fabric CCC configurator.

    (a) Skip this step if it is the MSS to the testboard SoC.

    (b) Make sure GL0, GL1 and GL2 are selected, and that the frequencies are the same as in step 11.

    (c) go to “advanced” tab. For GL1, set “PLL phase 180”. The actual phase, is written in blue, and should be 90 deg.

    (d) Press ok to exit the window. Save the current tab. Press generate in the top left corner. Close the tab when finished.

19. Right click the xxx_sb_0 and chose “update instance with latest components”. Save, and generate. The file is now generated and saved in the project directory.

C.8: Generating other Microsemi components used in SoCs

Similar to what is written above in “C.7: Generating MSS used in SoCs”. Choose SmartDesign instead of systembuilder. Drag component of interest, e.g PLL from the catalog window onto the design canvas. Here it can be configured. Press generate in top left corner when finished. Component is then generated and stored in the project directory.

Appendix D: AD9257 testboard extras

D.1: Testboard test plan

Future testing should start by figuring out the issues with the testboard-SoC system, which is explained in section 6.2.6. The clock output from the LVDS multiplexer must be measured again with a differential probe and confirmed to be 10 MHz. If this is not the case, the signal should be measured back to the source. Next, the reason why no data was read via the SPI interface must be found. This could be caused by the voltage-level-shifting circuitry, by the wires between the boards, or if the ADC does not operate properly. Thus, it should be checked again that the wires are correctly connected. Then, it must be confirmed that data is sent from the SF2 to the level shifters on the test board. It should also be checked that the timing relationships of the SPI signals are correct. It should then be confirmed that the signals appears at the output of the level shifters, and next on the inputs of AD9257.

If the issues in the above paragraph are solved, reading from the memory of AD9257 should be done next. One address contains an ID specific to AD9257. This can be read to verify that the SPI functionality and interface are working correctly. Refer to the software functionality that was made for accessing the memory of AD9257 in section 5.7. The next step should be to set output patterns from the ADC, and start verifying if the readout logic is working correctly. If readout works, or partly works, the wires between the boards should be made as short as possible. If none of the mentioned testing works, signals should be measured with an oscilloscope. Microsemi also provides a tool that enables monitoring of internal signal propagation inside the SF2. To debug the readout functionality, a source could be connected to one of the channels that is assembled on the board. The software implemented monitor functionality can be chosen in the terminal. Setting some voltage on one input of the AD9257 should then produce an output pattern from the ADC. If nothing appears in the terminal window, the outputs from the ADC should be measured with an oscilloscope.

If the setup works, the testing explained in chapter 6 for testing against the AD9257 testboard must be done. If the readout data is skewed, it is most likely caused by path delay between the ADC and FPGA. The data paths should be measured together with DCO and FCO. If the timing between the paths deviate from the ideal output timing diagram in figure 4.1, input delays should be added in the I/O-editor in Libero SoC to correct the deviations. Alternatively, input timing constrains can be added on the inputs and outputs of the FPGA in the SDC timing file, or through the timing-constraint editor in Libero SoC. The only timing constraints that has been added is for the internal clock

on the FPGA, and for the received clocks DCO and FCO. If the readout is incorrect, and if the skew is not fixed, the wires between the boards, and connections on the connector, should be checked. If the readout seems to work, a proper test with different patterns over time should be done to check that the system works as it should. Once the readout logic and control interface are deemed as functional, further work regarding evaluation of AD9257 can commence.

D.2: Schematics and component overview

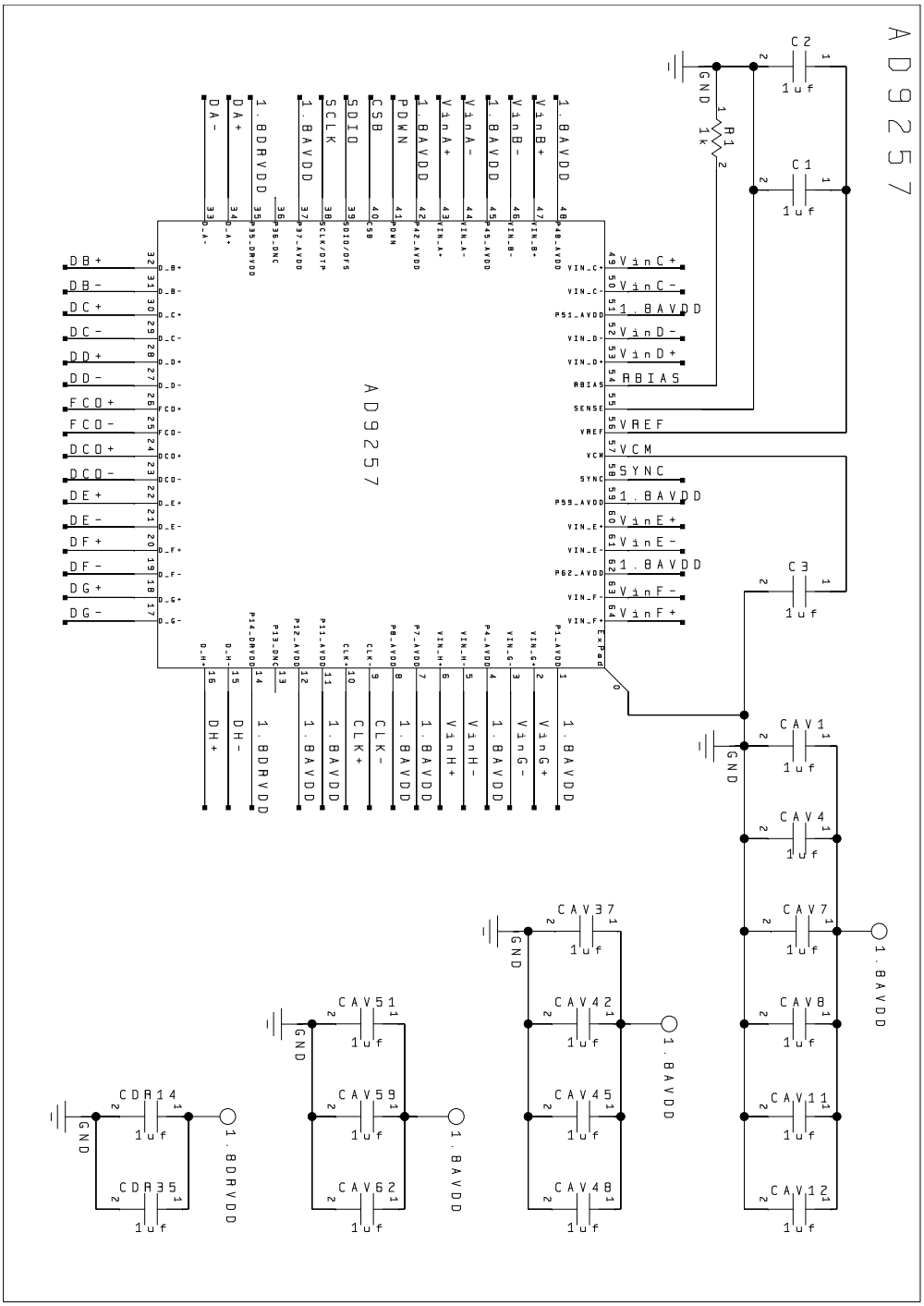


Figure 7.5: AD9257 circuit board

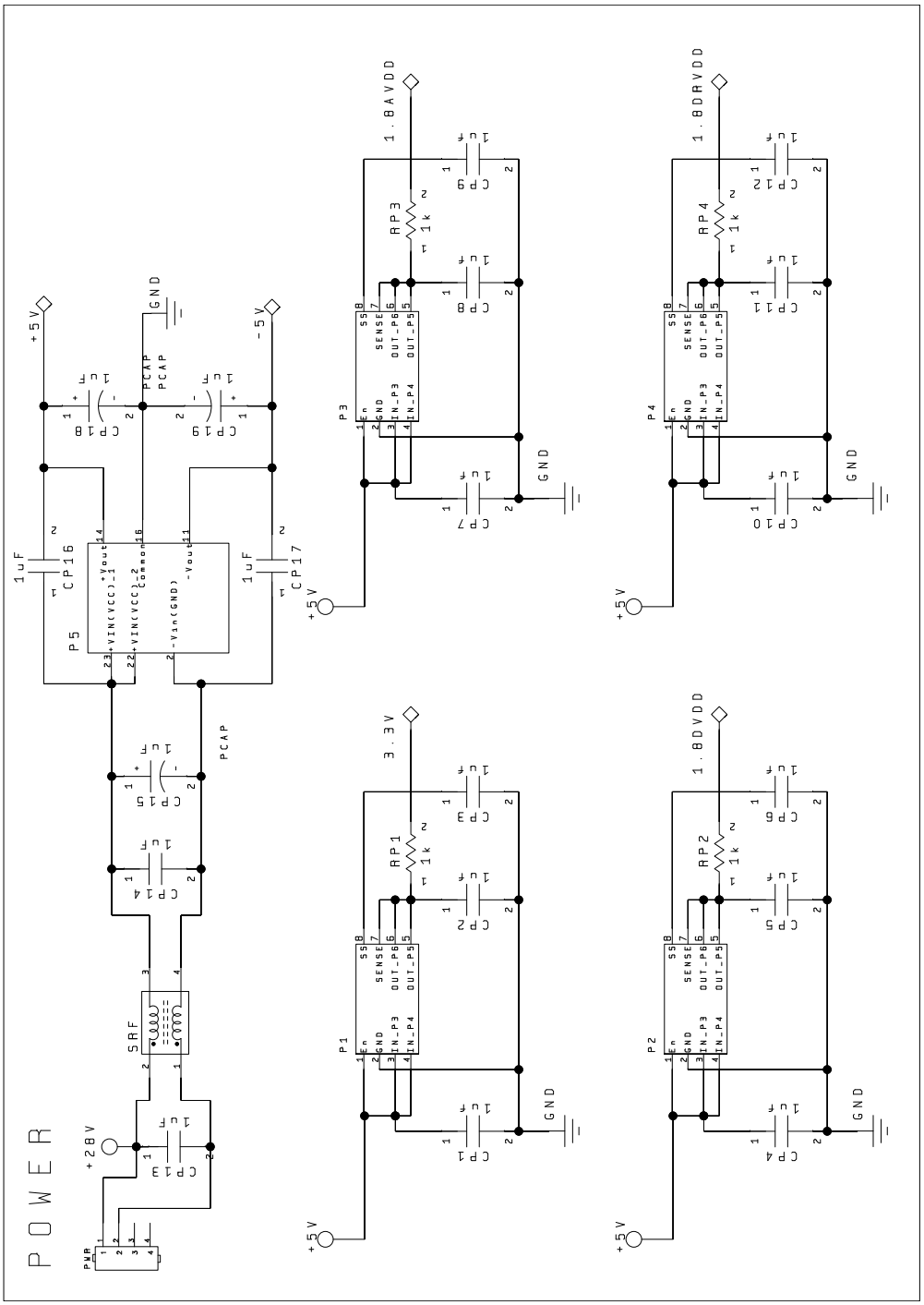


Figure 7.6: Power supplies on testboard

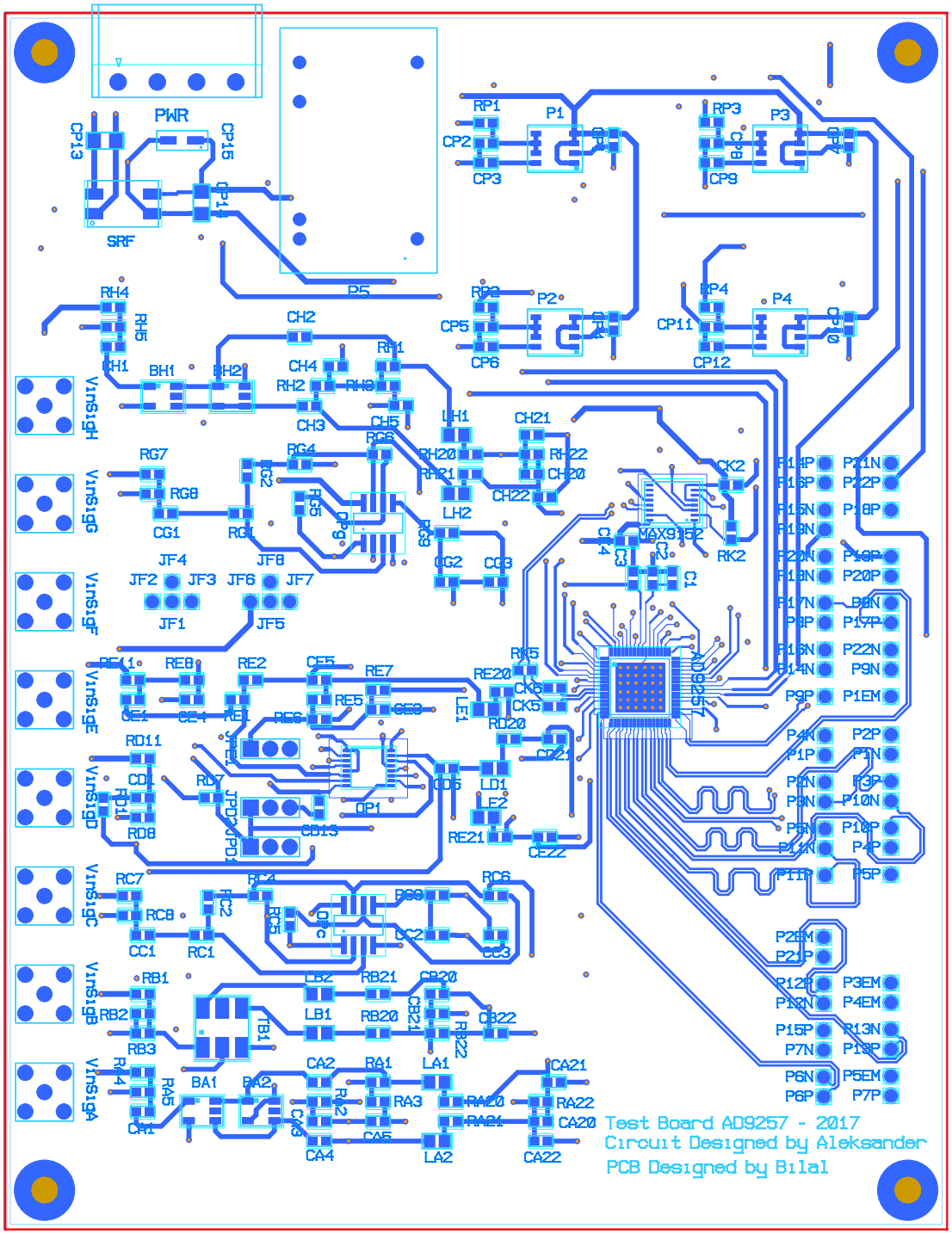


Figure 7.7: Testboard PCB front side

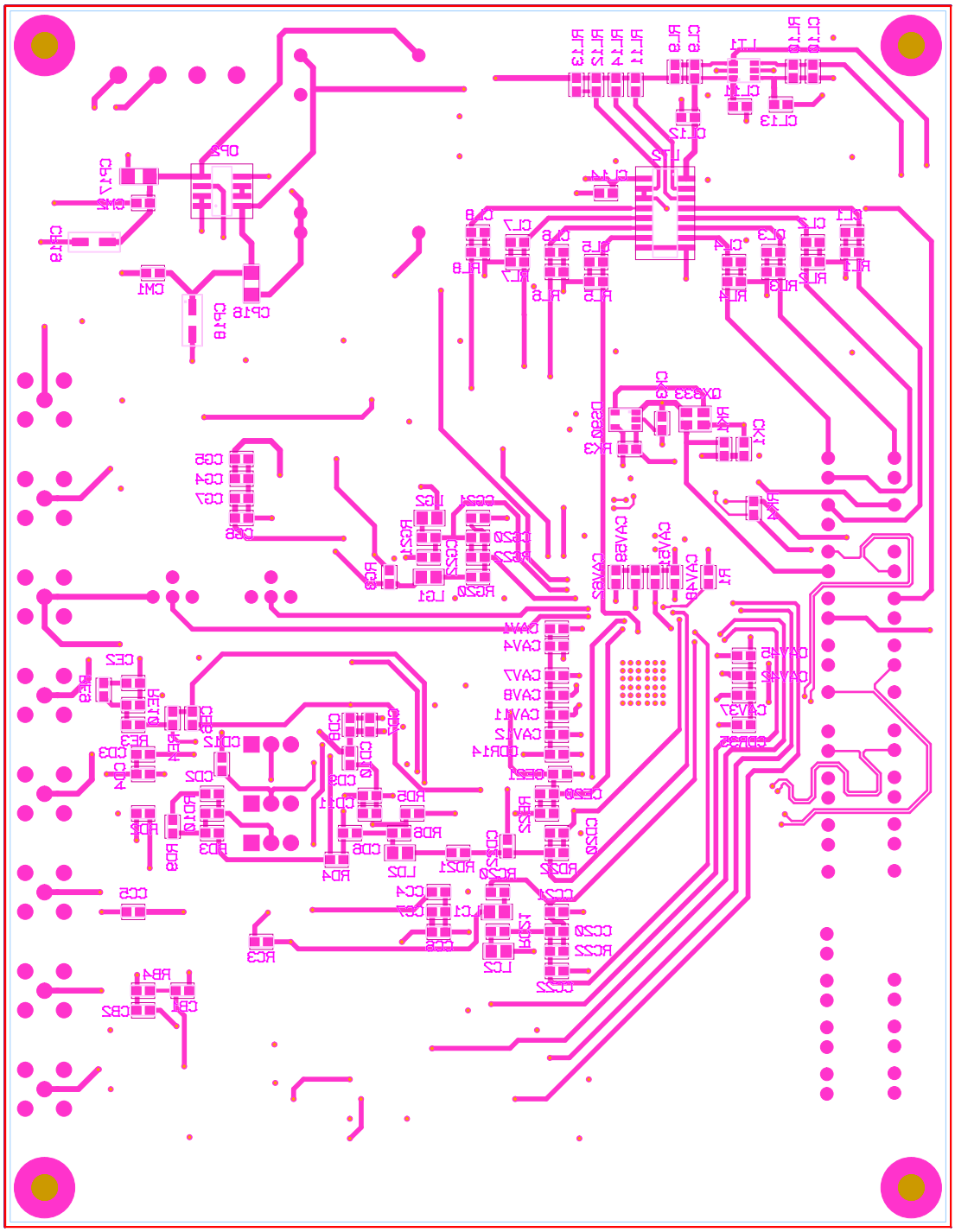


Figure 7.8: Testboard PCB backside

Table 7.4: ICs on testboard & channel D in 1st board configuration

Name	Device	Voltage [V]
ADC	AD9257	1.8
LT1	SN74AVC1T45	1.8, 3.3
LT2	SN74AVC4T245	1.8, 3.3
Op2, Opc, Opg	THS4012	±5
Clock	QX3 Series	3.3
LVDS buffer	DS90LV011A	3.3
LVDS mux	MAX9152	3.3
Ba1,Ba2,Bh1,Bh2	ETC1-1-13	
Tb	ADT1-1WT	
P1	ADP1706ARDZ-3.3-R7	5, 3.3
P2, P3, P4	ADP1706ARDZ-1.8-R7	5, 1.8
P5	THM 10-2421WI	28, 5
SRF	SRF0905-471Y	

(a) ICs on testboard

Name	Exact value	E96	Assembly
Rd1	1244	1240	yes
Rd2	1000	1000	yes
Rd3	1244	1240	yes
Rd4	1000	1000	yes
Rd5			open
Rd6			open
Rd7	200	200	yes
Rd8	52	52.3	yes
Rd9	25	24,9	yes
Rd10			open
RD11	0		short
Cd1	0		short
Cd2	0		short
Cd3	0.1uF		yes
Cd4	0.1uF		yes
Cd5	0		short
Cd6	0		short
Cd7	0.1uF		yes
Cd8	0.1uF		yes
Cd9	0.1uF		yes
Cd10	0.1uF		yes
CD11	0.1uF		yes
CD12	0,22uF		yes
CD13	0,22uF		yes
JPD1	jumper		1-2 short
JPD2	jumper		1-2 short
Ld1	0		short
Ld2	0		short
Rd20	100	100	yes
Rd21	100	100	yes
Rd22			open
Cd20			open
Cd21			open
Cd22			open
OP1			yes

(b) Channel D

Table 7.5: 1st board configuration.

(a) Voltage level shift circuitry

Name	Exact value	E96	Assembly
RL1			open
RL2			open
RL3			open
RL4			open
RL5			open
RL6			open
RL7			open
RL8			open
RL9			open
RL10			open
RL11	30k		YES
RL12	30k		YES
RL13	30k		YES
RL14	30k		YES
CL1			open
CL2			open
CL3			open
CL4			open
CL5			open
CL6			open
CL7			open
CL8			open
CL9			open
CL10			open
CL11	0.1uF		yes
CL12	0.1uF		yes
CL13	0.1uF		yes
CL14	0.1uF		yes
LT1			YES
LT2			YES

(b) Power supply

Name	Exact value	E96	Assembly
RP1	39	39	yes
RP2	39	39	yes
RP3	39	39	yes
RP4	39	39	yes
CP1	4.7uF		yes
CP2	4.7uF		yes
CP3	10nF		yes
CP4	4.7uF		yes
CP5	4.7uF		yes
CP6	10nF		yes
CP7	4.7uF		yes
CP8	4.7uF		yes
CP9	10nF		yes
CP10	4.7uF		yes
CP11	4.7uF		yes
CP12	10nF		yes
CP13	4.7uF 50V		yes
CP14	4.7uF 50V		yes
CP15	22uF 50V		yes
CP16	1000pF 2000V		yes
CP17	1000pF 2000V		yes
CP18	22uF 50V		yes
CP19	22uF 50V		yes
SRF			yes
P1			yes
P2			yes
P3			yes
P4			yes
P5			yes

Table 7.6: 1st board configuration cont.

(a) AD9257

Name	Exact value	E96	Assembly
R1	10kΩ, 1%	10k	yes
C1	1,0uF		yes
C2	0,1uF		yes
C3	0,1uF		yes
CAV1	0,1uF		yes
CAV4	0,1uF		yes
CAV7	0,1uF		yes
CAV8	0,1uF		yes
CAV11	0,1uF		yes
CAV12	0,1uF		yes
CAV37	0,1uF		yes
CAV42	0,1uF		yes
CAV45	0,1uF		yes
CAV48	0,1uF		yes
CAV51	0,1uF		yes
CAV59	0,1uF		yes
CAV62	0,1uF		yes
CAV14	0,1uF		yes
CAV35	0,1uF		yes
AD9257			yes

(b) Channel A & Channel H

Name	Exact value	E96	Assembly
Ra1			open
Ra2			open
RA3	200	200	yes
RA4			open
Ra5			open
Ca1			open
Ca2			open
Ca3			open
Ca4	0.1uF		yes
Ca5	0.1uF		yes
La1	0		short
La2	0		short
Ra20	0		short
Ra21	0		short
Ra22			open
Ca20			open
Ca21			open
Ca22			open
BA1			open
BA2			open

Table 7.7: 1st board configuration cont.

(a) Channel C & Channel G

Name	Exact value	E96	Assembly
Rc1			open
Rc2			open
Rc3			open
Rc4			open
Rc5			open
Rc6			open
RC7			open
RC8			open
Rc9	200	200	yes
Cc1			open
Cc2	0.1uF		yes
Cc3	0.1uF		yes
CC4			open
CC5			open
CC6			open
Cc7			open
Lc1	0		short
Lc2	0		short
Rc20	0		short
Rc21	0		short
Rc22			open
Cc20			open
Cc21			open
Cc22			open
OPC			open

(b) Channel E

Name	Exact value	E96	Assembly
Re1	1244	1240	yes
Re2	1000	1000	yes
Re3	1244	1240	yes
Re4	1000	1000	yes
RE5			open
RE6			open
Re7	200	200	yes
Re8	52	52.3	yes
Re9	25	24,9	yes
Re10			open
Re11	0		short
Ce1	1uF		yes
Ce2	1uF		yes
Ce3	0.1uF		yes
Ce4	0.1uF		yes
Ce5	0		short
Ce6	0		short
JPE1	jumper		1-2 short
Le1	0		short
Le2	0		short
Re20	100	100	yes
Re21	100	100	yes
Re22			open
Ce20			open
Ce21			open
Ce22			open

Table 7.8: 1st board configuration cont.

(a) Channel B & Channel F

Name	Exact value	E96	Assembly
RB1			open
RB2			open
Rb3			open
Rb4	200	200	yes
Cb1	0.1uF		yes
Cb2	0.1uF		yes
Lb1	0		short
Lb2	0		short
Rb20	0		short
Rb21	0		short
Rb22			open
Cb20			open
Cb21			open
Cb22			open
Tb1			open
Channel F			
Name	**Exact value**	**E96**	**Assembly**
JF1 - JF8			yes

(b) Buffer & Clock circuitry

Vcm Buf			
Name	**Exact value**	**E96**	**Assembly**
C9	0.1uF		yes
C10	0.1uF		yes
OP2			YES
Clock			
Name	**Exact value**	**E96**	**Assembly**
RK1			open
RK2			open
RK3	100		yes
RK4	100		yes
RK5	100		yes
CK1	0.1uF		yes
CK2	0.1uF		yes
CK3	0.1uF		YES
CK4	1nF		YES
CK5	0.1uF		YES
CK6	0.1uF		YES
QX333			YES
DS90			YES
MAX9152			YES

Table 7.9: Connections between board components and SF2 through jumpers on dev-board

AD9257					**SmartFusion2**	
Pin nr.	**Name**	**Direction**	**Jumper**	**jumper pair number**	**Pin nr.**	**Name in [4]**
15	D-H	Output	J115	6N	AA29	MSIO8NB3/CAN_TX_EN_N/GPIO_4_A/USB_DATA2_A
16	D+H	Output	J111	6P	V24	MSIO8PB3/CAN_RX/GPIO_3_A/USB_DATA1_A
17	D-G	Output	J121	13N	W28	MSIO13NB3/SPI_0_SS0/GPIO_7_A/USB_NXT_A
18	D+G	Output	J118	13P	W27	MSIO13PB3/SPI_0_SDO/GPIO_6_A/USB_STP_A
19	D-F	Output	J131	12N	T27	MSIO21NB3/GPIO_28_A
20	D+F	Output	J134	12P	T26	MSIO21PB3/GPIO_27_A
21	D-E	Output	J141	11N	N26	MSIO31NB2/GPIO_30_A
22	D+E	Output	J139	11P	P24	MSIO31PB2/GPIO_29_A
23	DCO-	Output	J145	10N	M26	MSIO36NB2/GPIO_8_B
24	DCO+	Output	J143	10P	N24	MSIO36PB2/GPIO_7_B
25	FCO-	Output	J146	3N	L29	MSIO33NB2/GPIO_2_B
26	FCO+	Output	J154	3P	L30	MSIO33PB2/GPIO_1_B
27	D-D	Output	J158	9N	K29	MSIO37NB2/GPIO_10_B
28	D+D	Output	J159	9P	K30	MSIO37PB2/GPIO_9_B
29	D-C	Output	J174	1N	K24	MSIO48NB1/I2C_0_SCL/GPIO_31_B/USB_DATA1_C
30	D+C	Output	J172	1P	K23	MSIO48PB1/I2C_0_SDA/GPIO_30_B/USB_DATA0_C
31	D-B	Output	J188	17N	H30	MSIO41NB1/MMUART_1_TXD/GPIO_24_B/USB_DATA2_C
32	D+B	Output	J183	17P	J30	MSIO41PB1/GB10/VCCC_SE0_CLKI/USB_XCLK_C
33	D-A	Output	J197	20N	G29	MSIO42NB1/MMUART_1_RXD/GPIO_26_B/USB_DATA3_C
34	D+A	Output	J194	20P	H29	MSIO42PB1/GB14/VCCC_SE1_CLKI/MMUART_1_CLK/GPIO_25_B/USB_DATA4_C
0	AGND	-	-	P14N,P16N,P22N	GND	N/A

Table 7.10: Connections between board components and SF2 through jumpers on dev-board cont.

AD9257					**SmartFusion2**		
Pin nr.	**Name**	**Directi**	**Jumper**	**jumper pair number**	**Pin nr.**	**Name in [4]**	
SN74AVC4T245					**SmartFusion2**		
4	1A1	Input	J202	15N	K25	MSIO44NB1/MMUART_0_DSR/GPIO_20_B	SCLK
5	1A2	Input	J210	16P	L23	MSIO47PB1/MMUART_0_RXD/GPIO_28_B/USB_STP_C	CSB
6	2A1	Input	J213	21N	V26	MSIO11NB3/CCC_NE1_CLKI0/I2C_1_SCL/GPIO_1_A/USB_DATA4_A	PDWN
7	2A2	Input	J214	14P	N23	MSIO39PB1/CCC_NE0_CLKI1/MMUART_1_CTS/GPIO_13_B	SYNC
14, 15	1OE, 2OE	Input	J187	8N	J28	MSIO38NB1/MMUART_1_DTR/GPIO_12_B	OE
SN74AVC1T45					**SmartFusion2**		
3	A	InOut	J209	22P	M24	MSIO43PB1/MMUART_0_RTS/GPIO_17_B/USB_DATA5_C	SDIO
5	DIR	input	J184	8P	J29	MSIO38PB1/MMUART_1_RTS/GPIO_11_B	
MAX9152					**SmartFusion2**		
6	In1+	Input	J196	19P	G30	MSIO45PB1/MMUART_0_RI/GPIO_21_B	
7	in1-	Input	J200	19N	F30	MSIO45NB1/MMUART_0_DCD/GPIO_22_B	
2	sel0	Input	J201	18P	M25	MSIO40PB1/CCC_NE1_CLKI1/MMUART_1_RI/GPIO_15_B	
QX3 Series					**SmartFusion2**		
1	Enable	Input	J195	18N	H28	MSIO40NB1/MMUART_1_DCD/GPIO_16_B	

Acronyms

AC	Alternating current
ADC	Analog to digital converter
APB	Advanced peripheral bus
ASCII	American Standard Code for Information Interchange
ASIC	Application specific integrated circuit
BFM	Bus functional model
CMOS	Complementary metal–oxide–semiconductor
CCC	Clock conditioning circuitry
CPU	Central processing unit
CSA	Charge sensitive amplifier
CSB	Chip select bar
DC	Direct current
DCO	Data capture clock
DDR	Double data rate
DSP	Digital signal processor
DFF	D-flip-flop
EP	Enhanced product
ENOB	effective number of bits
eV	electron volt
FDO	Fully-differential operational amplifier
FIC	Fabric interface controller
FIFO	First-in-first-out
FPGA	Field programmable gate array
FSM	Finite state machine
HSTL	High-speed transceiver logic
IC	Integrated circuit
IG2	IGLOO2
I/O	Input/output
IP	Intellectual property
LFCSP	Lead Frame Chip Scale Package

LSB	least significant bit
LUT	Lookup table
LVDS	low-voltage differential signaling
LVCMOS	low-voltage-CMOS
MSIO	Multi-standard I/O
MSS	Microcontroller subsystem
Opamp	Operational amplifier
PLL	Phase-locked-loop
PCB	Printed circuit board
RAM	Random access memory
ROM	Read only memory
RTL	Register transfer level
SCLK	Serial clock pin
SDIO	Serial data input/output pin
SED	Single-ended-to-differential
SEE	Single-event effects
SEU	Single event upset
SERDESIF	Serializer/deserializer interface
SINAD	Signal to noise and distortion
SF2	SmartFusion2
SNR	Signal to noise ration
SOI	Silicon on insulator
SPI	Serial peripheral interface
SRAM	Static RAM
SF2	SmartFusion2
TMR	Tripple-majority voting
UART	universal asynchronous receiver/transmitter
USB	Universal Serial Bus
UVVM	Universal VHDL verification methodology
VHDL	VHSIC hardware description language
VHSIC	Very high speed integrated circuit

Bibliography

[1] L.-K. G. Ødegaard, *Energetic particle precipitation into the middle atmosphere : optimization and applications of the NOAA POES MEPED data.* Bergen: University of Bergen, 2016.

[2] H. N. Tyssøy, J. Stadnes, F. Søraas, K. Ullaland, and D. Röhrich, "Distribution of energetic electron and proton (DEEP) instrument conceptual design report," Tech. Rep., 2016.

[3] C. T. Russel, "The configuration of the magnetosphere, in E.R. Dyer (ed.), Critical problems of magnetospheric physics," Washington, D.C, p. 15, 1972.

[4] R. M. Thorne, "The importance of energetic particle precipitation on the chemical composition of the middle atmosphere," *Pure and Applied Geophysics PAGEOPH*, vol. 118, no. 1, pp. 128–151, mar 1980. [Online]. Available: http://link.springer.com/10.1007/BF01586448

[5] C. Martin, "Van Allen radiation belt," 2006. [Online]. Available: https://commons.wikimedia.org/wiki/File:Van_Allen_radiation_belt.svg

[6] S. N. Ahmed, *Physics and Engineering of Radiation Detection.* Burlington: Elsevier Science, 2007.

[7] C. Herman and T. E. Johnson, *Introduction to health physics.* New York: McGraw-Hill Medical, 2009.

[8] F. aviation administration, "The Space Environment," 2017, ch. 4.1.2, p. 26. [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/avs/offices/aam/cami/library/online_libraries/aerospace_medicine/tutorial/media/III.4.1.2_The_Space_Environment.pdf

[9] M. M. Finckenor and K. K. de Groh, "Space Environmental Effects," NASA, Tech. Rep., 2015. [Online]. Available: https://www.nasa.gov/sites/default/files/files/NP-2015-03-015-JSC_Space_Environment-ISS-Mini-Book-2015-508.pdf

[10] N. H. Weste and D. M. Harris, *Integrated circuit design*, 4th ed. Boston, Nass: Pearson, 2011.

[11] J. L. Barth, "Space and Atmospheric Environments: from Low Earth Orbits to Deep Space," p. 19, 2003. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20030053331.pdf

[12] S. Duzellier, "Radiation effects on electronic devices in space," *Aerospace Science and Technology*, vol. 9, no. 1, pp. 93–99, jan 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1270963804001129

[13] Microsemi, "Understanding Single Event Effects (SEEs) in FPGAs," Tech. Rep., 2011. [Online]. Available: http://www.actel.com/documents/SEE_WP.pdf

[14] H. Spieler, *Semiconductor detector systems.* Oxford: Oxford University Press, 2005.

[15] J. P. Bentley, *Principles of measurement systems.* Harlow: Pearson education limited, 2005.

[16] Atmel, "AVR127: Understanding ADC Parameters," Atmel, Tech. Rep., 2016. [Online]. Available: https://goo.gl/hs7U3C

[17] W. Kester, "MT-003 TUTORIAL Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor," Analog Devices Inc, Tech. Rep., 2009. [Online]. Available: http://www.analog.com/media/en/training-seminars/tutorials/MT-003.pdf

[18] Microsemi, "UG0445 User Guide SmartFusion2 SoC FPGA and IGLOO2 FPGA Fabric," Microsemi, Tech. Rep., 2017. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/132008-ug0445-smartfusion2-soc-fpga-and-igloo2-fpga-fabric-user-guide

[19] G. Tambave and A. Velure, "Qualification of the ALICE SAMPA ASIC with a High-Speed Continuous DAQ System," p. 6, 2017.

[20] Microsemi, "Single Event Effects - A Comparison of Configuration Upsets and Data Upsets," no. November, 2015. [Online]. Available: https://goo.gl/j2G6uX

[21] S. Habinc, "Suitability of reprogrammable FPGAs in space applications," Gaisler Research for ESA, Tech. Rep., 2002. [Online]. Available: http://microelectronics.esa.int/techno/fpga_002_01-0-4.pdf

[22] Microsemi, "SmartFusion2 SoC FPGA Family." [Online]. Available: https://www.microsemi.com/products/fpga-soc/soc-fpga/smartfusion2

[23] Analog Devices, "AD9257-EP (Rev. A)," Analog Devices, Tech. Rep. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/AD9257-EP.pdf

[24] Analog Devices, "AD9257 (Rev. A)," Analog Devices, Tech. Rep., 2013. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/AD9257.pdf

[25] Analog Devices Inc, "AEROSPACE AND DEFENSE," p. 8, 2015. [Online]. Available: http://www.analog.com/media/en/news-marketing-collateral/solutions-bulletins-brochures/Aerospace-and-Defense-brochure.pdf

[26] Analog Devices Inc, "HIGH RELIABILITY COMPONENTS AND SOLUTIONS," p. 4, 2016. [Online]. Available: http://www.analog.com/media/en/news-marketing-collateral/solutions-bulletins-brochures/High-Reliability-Components-and-Solutions.pdf

[27] R. Ghaffarian, "Accelerated Thermal and Mechanical Testing of CSP Assemblies," 2001. [Online]. Available: https://nepp.nasa.gov/DocUploads/AFEF0D54-E6B6-49E3-AB978C0630AA6198/reza.pdf

[28] C. Leonard, "Challenges for Electronic Circuits in Space Applications The Harsh Environmental Conditions of a Spacecraft and the Hazards Posed to the Electronics," 2017. [Online]. Available: http://www.analog.com/media/en/technical-documentation/technical-articles/Challenges-for-Electronic-Circuits-in-Space-Applications.pdf

[29] G. Griffin, "A Design and Manufacturing Guide for the Lead Frame Chip Scale Package (LFCSP)," Analog Devices Inc, Tech. Rep., 2006. [Online]. Available: http://www.analog.com/media/en/technical-documentation/application-notes/AN-772.pdf

[30] G. L. Rose, N. Virmani, and J. S. Kadesch, "Plastic Encapsulated Microcircuit (PEM) Guidelines for Screening and Qualification for Space Applications," p. 36, 1997. [Online]. Available: https://nepp.nasa.gov/DocUploads/F2ED9134-5D10-48AE-A953EAC31069A797/pemqual.pdf

[31] R. Reeder, "Transformer-Coupled Front-End for Wideband A/D Converters," Analog Devices, Tech. Rep., 2005. [Online]. Available: http://www.analog.com/media/en/analog-dialogue/volume-39/number-2/articles/transformer-coupled-front-end-a-d-converters.pdf

[32] R. Reeder and R. Ramachandran, "Wideband A/D Converter Front-End Design Considerations. When to Use a Double Transformer Configuration," Analog Devices, Tech. Rep., 2006. [Online]. Available: http://www.analog.com/media/en/analog-dialogue/volume-40/number-3/articles/wideband-a-d-converter-front-end-design-considerations.pdf

[33] R. Reeder and J. Caserta, "Amplifier- or Transformer Drive for the ADC?" Analog Devices, Tech. Rep., 2007. [Online]. Available: http://www.analog.com/media/en/analog-dialogue/volume-41/number-1/articles/wideband-adc-design-considerations-2.pdf

[34] R. Reeder and Analog Devices, "Frequency Domain Response of Switched-Capacitor ADCs , Rev A," Tech. Rep., 2005. [Online]. Available: http://www.analog.com/media/en/technical-documentation/application-notes/58717399805791156408708165573049671384533529037444108 3AN_742_a.pdf

[35] Texas Instruments, "THS4524-EP: VERY LOW POWER, NEGATIVE RAIL INPUT, RAIL-TO-RAIL OUTPUT, FULLY DIFFERENTIAL AMPLIFIER," p. 48, 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/ths4524-ep.pdf

[36] J. Ardizzoni and J. Pearson, "High Speed Differential ADC Driver Design Considerations," Tech. Rep., 2015.

[37] Microsemi, "UG0447 User Guide SmartFusion2 and IGLOO2 FPGA High-Speed Serial Interfaces," Microsemi, Tech. Rep., 2017. [Online]. Available: https://www.microsemi.com/document-portal/doc_view/132011-ug0447-smartfusion2-and-igloo2-fpga-high-speed-serial-interfaces-user-guide

[38] Vinod Paliakara and Shantanu Prabhudesai, "Understanding Serial LVDS Capture in High-Speed ADCs Application Report," Texas instruments, Tech. Rep., 2013. [Online]. Available: http://www.ti.com/lit/an/sbaa205/sbaa205.pdf

[39] Microsemi, "DS0128 Datasheet IGLOO2 FPGA and SmartFusion2 SoC FPGA," Tech. Rep., 2016. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/132042-ds0128-igloo2-and-smartfusion2-datasheet

[40] Microsemi, "Advanced Static Timing Analysis Using SmartTime," Microsemi, Tech. Rep., 2011. [Online]. Available: https://www.microsemi.com/document-portal/doc_view/129843-ac379-advanced-static-timing-analysis-using-smarttime-app-note

[41] Analog Devices, "AN-877 APPLICATION NOTE Interfacing to High Speed ADCs via SPI by the High Speed Converter Division," Analog Devices, Tech. Rep., 2007. [Online]. Available: http://www.analog.com/media/en/technical-documentation/application-notes/AN-877.pdf?doc=AD9670

[42] Microsemi, "HB0089 Handbook CoreSPI v5.1," Microsemi, Tech. Rep., 2016.

[43] Microsemi, "UG0331 User Guide SmartFusion2 Microcontroller Subsystem," Microsemi, Tech. Rep., 2016. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/130918-ug0331-smartfusion2-microcontroller-subsystem-user-guide

[44] Microsemi, "SmartFusion2 MSS SPI Driver User's Guide," Tech. Rep., 2015.

[45] ARM, "AMBA 3 APB Protocol Specification v1.0," ARM Limited, Tech. Rep., 2004. [Online]. Available: http://web.eecs.umich.edu/$\sim$prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf

[46] Microsemi, "UG0449 User Guide SmartFusion2 and IGLOO2 Clocking Resources," Tech. Rep., 17. [Online]. Available: https://www.microsemi.com/document-portal/doc_view/132012-ug0449-smartfusion2-and-igloo2-clocking-resources-user-guide

[47] Microsemi, "SmartFusion2 SoC FPGA Advanced Development Kit UG0557 User Guide Table of Contents," Microsemi, Tech. Rep., 2013. [Online]. Available: https://www.microsemi.com/document-portal/doc_view/130919-smartfusion2-development-kit-user-guide

[48] Analog Devices, "EVAL-AD9257 Evaluation Board." [Online]. Available: http://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad9257.html#eb-documentation

[49] K. Mustafa and C. Sterzik, "AC-Coupling Between Differential LVPECL, LVDS, HSTL, and CML," Texas Instruments, Tech. Rep., 2007. [Online]. Available: http://www.ti.com/lit/an/scaa059c/scaa059c.pdf

[50] B. Carter, "Buffer Op Amp to ADC Circuit Collection," Texas Instruments, Tech. Rep., 2002. [Online]. Available: http://www.ti.com/lit/an/sloa098/sloa098.pdf

[51] P. J. Ashenden, *The Designer's Guide to VHDL*, 3rd ed. Elsevier Science, 2008.

[52] Microsemi, “Libero SoC for Enhanced Constraint Flow v11.7 User’s Guide,” Tech. Rep., 2016. [Online]. Available: http://coredocs.s3.amazonaws.com/Libero/11_7_0/Tool/libero_ecf_ug.pdf

[53] Microsemi, “SoftConsole v4.0 and Libero SoC v11.7 TU0546 Tutorial,” Tech. Rep., 2016. [Online]. Available: https://www.microsemi.com/document-portal/doc_view/133700-tu0546-softconsole-v4-0-and-libero-soc-v11-7-tutorial