

Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Built-in self-test of analog-to-digital converters in FPGAs

Examensarbete utfört i elektroteknik
vid Tekniska högskolan vid Linköpings universitet
av

Petter Nilsson

LiTH-ISY-EX--14/4747--SE

Linköping 2014



Linköpings universitet
TEKNISKA HÖGSKOLAN

Built-in self-test of analog-to-digital converters in FPGAs

Examensarbete utfört i elektroteknik
vid Tekniska högskolan vid Linköpings universitet
av

Petter Nilsson


LiTH-ISY-EX--14/4747--SE

Handledare: **Petter Källström**
ISY, Linköpings universitet

Peng Lim
Xilinx

Examinator: **J Jacob Wikner**
ISY, Linköpings universitet

Linköping, 29 april 2014

	Avdelning, Institution Division, Department Electronics Systems Department of Electrical Engineering SE-581 83 Linköping	Datum Date 2014-04-29
---	---	--

Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN LiTH-ISY-EX--14/4747--SE <table style="width: 100%;"> <tr> <td style="width: 60%;">Serietitel och serienummer</td> <td style="width: 40%;">ISSN</td> </tr> <tr> <td>Title of series, numbering</td> <td>_____</td> </tr> </table>	Serietitel och serienummer	ISSN	Title of series, numbering	_____
Serietitel och serienummer	ISSN					
Title of series, numbering	_____					
URL för elektronisk version http://www.ep.liu.se						

Titel Title	Built-in self-test of analog-to-digital converters in FPGAs Built-in self-test of analog-to-digital converters in FPGAs
Författare Author	Petter Nilsson

Sammanfattning Abstract	<p>When designing an ADC it is desirable to test its performance at two different points in the development process. The first is characterization and verification testing when a chip containing the ADC has been taped-out for the first time, and the second is production testing when the chip is manufactured in large scale. It is important to have a good correlation between the results of characterization and the results of production testing.</p> <p>This thesis project investigates the feasibility of using a built-in self-test to evaluate the performance of embedded ADCs in FPGAs, by using the FPGA fabric to run necessary test algorithms. The idea is to have a common base of C code for both characterization and production testing. The code can be compiled and run on a computer for a characterization test setup, but it can also be synthesized using a high-level synthesis (HLS) tool, and written to FPGA fabric as part of a built-in self-test for production testing. By using the same code base, it is easier to get a good correlation between the results, since any difference due to algorithm implementation can be ruled out. The algorithms include a static test where differential nonlinearity (DNL), integral nonlinearity (INL), offset and gain error are calculated using a sine-wave based histogram approach. A dynamic test with an FFT algorithm, that for example calculates signal-to-noise ratio (SNR) and total harmonic distortion (THD), is also included. All algorithms are based on the <i>IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters</i> (IEEE Std 1241). To generate a sine-wave test signal it is attempted to use a delta-sigma DAC implemented in the FPGA fabric.</p> <p>Synthesizing the C code algorithms and running them on the FPGA proved successful. For the static test there was a perfect match of the results to 10 decimal places, between the algorithms running on a computer and on the FPGA, and for the dynamic test there was a match to two decimal places. Using a delta-sigma DAC to generate a test sine-wave did not prove feasible in this case. Assuming a brick-wall bandpass filter the performance of the delta-sigma DAC is estimated to an SNR of 53dB, and this signal is not pure enough to test the test case ADC with a specified SNR of 60dB.</p>
-----------------------------------	--

Nyckelord Keywords	FPGA, BIST, ADC, dynamic test, static test, linearity, DNL, INL, offset, gain error, FFT, SNR, THD, delta-sigma, sigma-delta, DAC, high-level synthesis, HLS, IEEE Standard 1241
------------------------------	--

Abstract

When designing an ADC it is desirable to test its performance at two different points in the development process. The first is characterization and verification testing when a chip containing the ADC has been taped-out for the first time, and the second is production testing when the chip is manufactured in large scale. It is important to have a good correlation between the results of characterization and the results of production testing.

This thesis project investigates the feasibility of using a built-in self-test to evaluate the performance of embedded ADCs in FPGAs, by using the FPGA fabric to run necessary test algorithms. The idea is to have a common base of C code for both characterization and production testing. The code can be compiled and run on a computer for a characterization test setup, but it can also be synthesized using a high-level synthesis (HLS) tool, and written to FPGA fabric as part of a built-in self-test for production testing. By using the same code base, it is easier to get a good correlation between the results, since any difference due to algorithm implementation can be ruled out. The algorithms include a static test where differential nonlinearity (DNL), integral nonlinearity (INL), offset and gain error are calculated using a sine-wave based histogram approach. A dynamic test with an FFT algorithm, that for example calculates signal-to-noise ratio (SNR) and total harmonic distortion (THD), is also included. All algorithms are based on the *IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters* (IEEE Std 1241). To generate a sine-wave test signal it is attempted to use a delta-sigma DAC implemented in the FPGA fabric.

Synthesizing the C code algorithms and running them on the FPGA proved successful. For the static test there was a perfect match of the results to 10 decimal places, between the algorithms running on a computer and on the FPGA, and for the dynamic test there was a match to two decimal places. Using a delta-sigma DAC to generate a test sine-wave did not prove feasible in this case. Assuming a brick-wall bandpass filter the performance of the delta-sigma DAC is estimated to an SNR of 53dB, and this signal is not pure enough to test the test case ADC with a specified SNR of 60dB.

Acknowledgments

I would like to thank the examiner J Jacob Wikner and my supervisor Petter Källström at the division of Electronics Systems, Linköping University, for helping me throughout the project. Special thanks to Jacob for helping me with the application for the internship that led up to this thesis project.

I also would like to thank my supervisor Peng Lim at Xilinx for his support. Furthermore I would like to thank the AMS group at Xilinx in Dublin, Ireland, and all the colleagues at Xilinx who have offered comments and suggestions during my time there. Special thanks goes to Adrian Lynam and Aidan Keady for helping me with the project.

Linköping, April 2014
Petter Nilsson

Contents

List of figures	ix
List of tables	xi
Notation	xiii
1 Introduction	1
1.1 About FPGAs	1
1.2 About Xilinx	2
1.3 Problem formulation	2
1.4 Related research	3
1.5 Method and tools	4
1.6 Thesis outline	6
2 System overview	7
2.1 What is a BIST?	7
2.2 Test case	8
2.3 The system	9
2.4 Interface between stimulus and ADC	11
2.5 Summary	11
3 Theory of ADC testing	13
3.1 About the IEEE standard 1241	13
3.2 Static test	14
3.2.1 ADC transfer function	14
3.2.2 Performance metrics	17
3.2.3 How to obtain the transfer levels	20
3.3 Dynamic test	28
3.3.1 Performance metrics	28
3.4 Summary	33
4 Theory of delta-sigma DACs	35
4.1 First-order delta-sigma modulator	35

4.2	Second-order delta-sigma modulator	42
4.3	Summary	43
5	System implementation and results	45
5.1	The delta-sigma DAC	45
5.1.1	Measurement results	46
5.1.2	Alternative approach – filtered clock signal	48
5.2	Top-level FSMs	49
5.3	Overview of C code algorithms	51
5.3.1	Static test	51
5.3.2	Dynamic test	52
5.4	BIST measurement results	54
5.4.1	Comparison between models and measurement	54
5.4.2	Utilization	60
5.4.3	Measurement time	60
5.5	Summary	61
6	Conclusions and future work	63
	Bibliography	65

List of figures

1.1	Project workflow	5
2.1	The KC705 evaluation board	8
2.2	BIST block diagram	10
3.1	Transfer function conventions	14
3.2	ADC transition levels	15
3.3	Code-edge and code-center approach	16
3.4	Best-fit and terminal based	17
3.5	Example of gain and offset	20
3.6	Example of DNL and INL	21
3.7	Servo approach to finding transition levels	22
3.8	Histogram generated with evenly distributed input signal	22
3.9	Ramp signal and its probability density function	23
3.10	Sampled sine-wave and the resulting histogram	24
3.11	A sine-wave and its probability density function	25
3.12	Spectrum of a 20 kHz sine-wave	29
3.13	8-point FFT flow graph	31
4.1	Block diagram of a first-order delta-sigma modulator	36
4.2	Spectrum of the output signal of a first-order delta-sigma modulator simulated in Matlab	36
4.3	Linear model of the first-order delta-sigma modulator.	38
4.4	Probability density function of quantization noise, assuming the noise is uniformly distributed.	38
4.5	Spectral density of quantization noise, assuming the noise is white.	39
4.6	Magnitude plot of NTF of a first-order delta-sigma modulator	40
4.7	Linear model of a second-order delta-sigma modulator	42
4.8	Magnitude plot of NTF of a second-order delta-sigma modulator	43
5.1	Block diagram of the second-order delta-sigma modulator	46
5.2	Setup for testing delta-sigma DAC	46

5.3	Spectra of the output signals of second-order delta-sigma modulators	47
5.4	Test setup for sine-wave generation with clock signal	48
5.5	Spectrum of 20 kHz clock signal	49
5.6	Flowchart of static test FSM	50
5.7	Flowchart of dynamic test FSM	51
5.8	Pseudo code for the first static test sub-function	52
5.9	Pseudo code for the second static test sub-function	52
5.10	Pseudo code for third static test sub-function	53
5.11	Struct holding complex numbers	53
5.12	Butterfly flowgraph	53
5.13	Pseudo code for the FFT algorithm	54
5.14	Static test results illustrated with data probed with ILAs.	56
5.15	DNL and INL using 700 mV input amplitude, when algorithm assumes 550 mV	57
5.16	DNL and INL using an input signal frequency which is harmonically related to the sample frequency	58
5.17	Frequency spectrum calculated by FFT algorithm and probed with ILA	59

List of tables

2.1	Specifications of test case ADC	9
3.1	Static performance metrics	18
3.2	Dynamic performance metrics	30
3.3	Bit-reversed order	32
5.1	Comparison between static BIST and code running on PC	55
5.2	Static measurement results using 700 mV input amplitude, when algorithm assumes 550 mV.	56
5.3	Static measurement results using an input signal frequency which is harmonically related to the sample frequency	57
5.4	Comparison between dynamic BIST and code running on PC . . .	58
5.5	Resource utilization for dynamic and static test including ILAs . .	60
5.6	Estimate of measurement time	61

Notation

ABBREVIATIONS

Abbreviation	Meaning
FPGA	Field-Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
ADC	Analog-to-Digital Converter
V&C	Verification and Characterization
RTL	Register-Transfer Level
DAC	Digital-to-Analog Converter
IDE	Integrated Design Environment
HDL	Hardware Description Language
VHDL	VHSIC (Very High Speed Integrated Circuit) Hard- ware Description Language
ILA	Integrated Logic Analyzer
HLS	High-Level Synthesis
FFT	Fast Fourier Transform
DFT	Discrete Fourier Transform
FSM	Finite State Machine
PDF	Probability Density Function
SQNR	Signal-to-Quantization-Noise Ratio
STF	Signal Transfer Function
NTF	Noise Transfer Function
DDS	Direct Digital Synthesizer

1

Introduction

This thesis concerns the testing of analog-to-digital converters (ADCs) that are embedded in field-programmable gate array (FPGA) chips. By exploiting the fact that an FPGA can be reprogrammed, its fabric can be utilized as part of a built-in self-test (BIST) for the embedded ADC, and the test logic can then be removed when it is not needed anymore. If the test algorithms are written in C code, and mapped to the FPGA using high-level synthesis, then the exact same algorithms can also run on a computer as part of larger test setup. The BIST is suitable for production testing and demonstrations, and the computerized test setup is suitable for more versatile characterization testing. By using the same code base it is easier to get comparable measurement results from the two. In order to test the ADC a test signal also has to be generated, and this can possibly be done with the help of a delta-sigma modulator realized in the FPGA fabric.

These two topics—using C code for an FPGA ADC test, and generating a test signal with a delta-sigma modulator in the FPGA fabric—are investigated in this thesis. The C code test algorithms are mainly based on an IEEE standard for ADC testing, and related research papers, and the delta-sigma modulator is inspired by an application note from Xilinx. C code test algorithms and Verilog code for delta-sigma modulators were developed in Xilinx's Vivado design suite.

1.1 About FPGAs

A field-programmable gate array is a type of programmable logic circuit, which can be used to realize integrated circuit designs. In contrast to an application specific integrated circuit (ASIC) an FPGA is not manufactured to fit a certain application, but can instead be used in a large variety of applications. One of

the advantages is that the development time for a design can be much shorter for an FPGA than for an ASIC, and thus the time-to-market is shorter. Another advantage is flexibility. If there is an error in the design, the bug can be fixed and the device reprogrammed with the correct version of the design. If a product is produced in large volumes, ASICs have the advantage that they are cheaper to manufacture per unit, but since there is a high initial cost for manufacturing integrated circuits, FPGAs are more cost effective for lower volumes.

1.2 About Xilinx

Xilinx is presently the largest programmable logic company in the world, with around 50% of the market. It was founded in 1984 and has around 3000 employees [1]. This thesis project was carried out as part of an internship in Xilinx's AMS group in Dublin, Ireland.

1.3 Problem formulation

When developing analog-to-digital converters validation and characterization (V&C) work is typically carried out. A number of chips are manufactured and measurements are taken in order to conclude if the performance of the ADC match what is expected from simulation results obtained during the design process. If the measurement results do not match or exceed the expectations, the circuit can either be redesigned or the requirements on the final product can be relaxed.

Another point in time at which measuring the performance of the ADC is of interest is when the chip is manufactured in large scale, i.e. production testing. Either each chip is tested or samples from the production line are taken and tested. This is to ensure the quality of the product to some defined precision and statistical significance.

These two test situations have somewhat different requirements. For V&C a limited number of parts are tested, and it is important to have a flexible setup so that the circuit can be tested in all possible modes—if more than one exists—and under many operating conditions (e.g. various temperatures). For production testing a large number of parts are tested and the test must be much more limited due to time constraints. It is important that the setup is highly automated. Due to these differences in requirements, production testing and V&C are generally done using two separate test setups. It is however very important to have a good correlation between the results of the two setups.

Xilinx develops FPGAs with embedded ADCs and these ADCs have to be tested. The purpose of this thesis project is to develop a set of algorithms written in C code that can both be run on a desktop computer as part of V&C setup, and be synthesized into a register transfer level (RTL) representation which can be written to the FPGA fabric, as part of a built-in self-test (BIST) for production testing. The feasibility of such a solution is investigated. In order to test an ADC

an input signal, a stimulus, has to be generated. This signal has to be spectrally pure because it is difficult to separate an error of the ADC from an error in the input signal. To generate this signal is one of the most difficult challenges when it comes to ADC testing. In this thesis the possibility of using a delta-sigma digital-to-analog converter (DAC) as stimulus is investigated.

1.4 Related research

This work is primarily based on the *IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters* (Std 1241-2010). This standard is intended to provide a common terminology and methods for testing ADCs [2]. The standard describes test methods and the formulas used to process measured information into metrics that quantify the errors of the ADC, but it does not attempt to derive the results used. Instead papers where this is done are cited in the standard.

The sections of the standard that are of most interest for this project primarily refer to the three papers: *Full-Speed Testing of A/D Converters*, by Doernberg et al., *Dynamic Testing and Diagnostics of A/D Converters*, Vanden Bossche et al. and *Histogram Measurement of ADC Nonlinearities Using Sine Waves*, Blair.

Doernberg et al. introduces a histogram test based on a ramp input, a histogram test based on a sine-wave input and nonlinearity analysis using fast Fourier transform (FFT). They derive a formula for calculating a statistical estimate of the transition levels of the ADC using the cumulative histogram resulting from sampling a sine-wave input. A result for calculating the number of samples needed in the test to measure the differential nonlinearity with a defined precision and statistical significance is derived. It is based on the fact that the histogram test can be considered a Bernoulli trial where each sample either goes into a specific ADC code bin or it does not go into that code bin. The Bernoulli trial has a binomial distribution which is approximated as a normal distribution to derive the result [7].

Vanden Bossche et al. derive the formula for calculating transition levels of the ADC from a sine-wave histogram in a different way. They also derive formulas for calculating the standard deviation of DNL and INL measurements. To analyze the ADC at bit-level it is suggested to use Walsh transform [4].

Blair extends the work of Doernberg and Vanden Bossche. The cumulative sine-wave histogram approach is used but it is suggested to use the average code bin width in calculating DNL instead of the ideal code bin width. Additive noise affects the result when using a sine-wave histogram approach, but this error can be reduced by overdriving the ADC, and a formula for this required overdrive vs. noise level is derived. Also a more general formula for desired accuracy and confidence level of DNL and INL calculations is derived. This gives an estimate of worst case deviation, and it is noted that this is important because if individual DNL values are calculated with a certain precision with 99% confidence, the expectation is that 1% will be out of tolerance. Since a high resolution ADC has

many code bins quite a few of the calculated DNL errors are then expected to be out of tolerance. The effect of harmonic distortion in the input signal is also studied [3]. Many of the results from this paper are found in the IEEE standard.

In [13] the possibility of using an imprecise stimulus to test ADCs is presented. Typically a stimulus of at least $N + 3$ bits of linearity is used to test an N -bit ADC but generating such a signal is difficult for a high resolution ADC. In this paper the authors proposes an algorithm which, using the ramp histogram approach, makes it possible to use a stimulus with less linearity than the ADC under test. This is done by using two ramps, where the second one is shifted by a known offset.

The delta-sigma modulator attempted in this thesis project is based off a Xilinx application note, XAPP154 [11]. This application note presents Verilog code for a first-order delta-sigma modulator, using two 10-bit adders and latches. Excess coding is used for the digital representation. It also discusses the choice of components for an RC lowpass filter to filter the DAC output.

1.5 Method and tools

The main development tool used throughout the project was Xilinx's Vivado Design Suite 2013.3. It includes, among other things, an integrated design environment (IDE) for FPGA design development. In the Vivado IDE hardware description language code ¹ (HDL) can be edited, debugged and synthesized. The design can then be simulated, implemented and generated into a bitstream which can be loaded onto an FPGA through a JTAG interface. All this can be done in one design environment. If integrated logic analyzers (ILAs) are integrated into the design, these can be used through Vivado's hardware manager when the design is running on the FPGA.

Part of the Vivado Design Suite is a tool called Vivado High-level Synthesis (HLS). With HLS, code written in C, C++ and SystemC can be synthesized into a register-transfer level (RTL) representation, which can be integrated into an FPGA design. RTL is a name for the abstraction level which is usually used when designing using HDL code.

The idea behind HLS is to use a higher abstraction level to make it easier to manage complexity of designs and to speed up the design process. In the same way as HDL code usually implies a higher abstraction level than drawing circuit schematics, writing C code allows using a higher abstraction level than writing HDL code. So HLS allows using an already existing programming language, like C, to design at a higher abstraction level.

There are some limitations to HLS though, which are important to have knowledge about when using the tool. Since the design is meant to run on an FPGA there can be no system calls in the C code which would use the operating system,

¹HDL usually refers to the languages VHDL and Verilog.

because there is of course no operating system on the FPGA. Another limitation, which is more important to this project, is that dynamic memory is not allowed. This means no recursive algorithms are allowed and all array type of data must have a defined size at compile time. The reason for this limitation is that arrays usually are stored in block RAM on the FPGA and it has to be known beforehand which resources that should be allocated to the design. If the size of an array in the design would change dynamically, the synthesis tool would not be able to predict how much memory it must allocate to this array. When analyzing ADCs, data is typically stored in arrays and the size of many of these arrays are decided by the resolution of the ADC, so this limitation means that the design must typically be recompiled for different ADC resolutions.

A third limitation is that some floating point functions are not supported. This is because floating point operations (e.g. an addition) are using floating point operators (e.g. an adder) which are mapped to operators in the Vivado IP core library. If there is no operator in the library matching a particular C function, the function cannot be synthesized. An example is the complex exponential function, `cexp`. The user guide lists the functions that are supported [18].

It is useful to have a good understanding of hardware when designing with HLS, because the user can control how the code is mapped to hardware by setting directives. The user can for example decide in what kind of memory an array should be stored, and if a loop should be pipelined to increase data throughput.

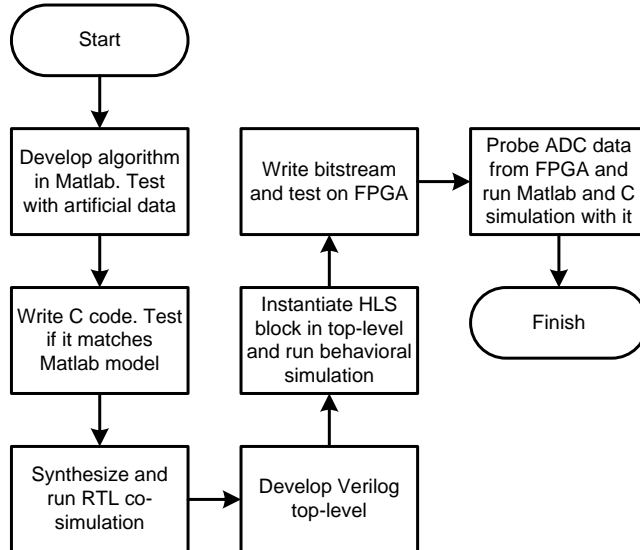


Figure 1.1: Project workflow.

The workflow followed in this project can be seen in Fig. 1.1. The algorithms were developed in Matlab from Mathworks, because it is convenient to experiment with algorithms using this tool. Matlab also has a built-in fast Fourier transform

(FFT) function which was used as reference, when developing an FFT function for the C program.

The ADC test covered in this project can be divided into two parts—static test and dynamic test—and each of these two tests consist of a set of algorithms. The algorithms were tested with artificial data generated in Matlab. For the static test an ADC model was written using an if-statement to represent the transition levels of an ideal 4-bit ADC, and then errors were introduced by changing some of the transition level from their ideal values. The model was then used to quantize a sine-wave and the resulting ADC codes were used to test the static test algorithms. For the dynamic test a sine-wave with a specific signal-to-noise ratio was generated using the built-in additive white Gaussian noise function in Matlab. Harmonics of the signal were also added to the signal and the resulting array of data was then used to test the dynamic test algorithms.

After these tests with artificial data, the algorithms were ported into C code and the C code was tested using the same artificial data to see that it matched the models in Matlab. Then the code was synthesized into a RTL representation using HLS and tested in a co-simulation. In the co-simulation both the C code and its synthesized RTL representation are simulated using the same C code test bench. This way the results can be compared to make sure the synthesized design is working as expected.

The top-levels of the BIST, one for a dynamic ADC test and one for a static ADC test, were written in Verilog. They are basically finite state machines that collect ADC data and start the test when a button is pressed. In Vivado there is a catalog of IP cores, which are ready-made pieces that can be used in designs by generating code and then instantiating it in Verilog. The HLS blocks (the synthesized C code) can be imported into this library and then instantiated into the design in the same manner as other IP cores. This way the synthesized ADC test algorithms were integrated into the Verilog top-levels.

The top-level designs with instantiated HLS blocks were simulated, synthesized, implemented and generated into bitstreams using Vivado, and then tested on the FPGA. When testing them, output codes from the ADC were probed using ILAs and then used as test data in Matlab and C simulations. In this way the results from the BIST could be compared to the results of algorithms running on a computer processing the exact same data, and thus the correlation between them could be investigated.

1.6 Thesis outline

In chapter 2 an overview of the system is given. Chapter 3 discusses the theory behind ADC testing with focus on the tests that were included in this project. In chapter 4 the basic theory behind delta-sigma DACs is explained. Chapter 5 discusses how the system was implemented and the results of the investigations. Chapter 6 contains conclusions and suggestions for further work.

2

System overview

As mentioned in the problem formulation the purpose with this project is to develop a common base of C code for both production testing and characterization work, using high-level synthesis to write the code to the FPGA fabric. Not all parts of the BIST are implemented with synthesized C code, however, instead there is a top-level written in Verilog and only the core test algorithms are written in C. This way there is a clear distinction between the code base that is intended to be a common base for production testing and characterization, and the extra logic necessary to make the BIST work. The delta-sigma modulator is also written in Verilog. The test case for the system is a Kintex-7 FPGA which has an embedded 12-bit, 1 MS/s ADC.

The goal is to have a system that is close to a true built-in self-test, in the sense that it has no active external components. To filter out quantization noise from the output signal of the delta-sigma modulator it is necessary to have an external analog filter in the signal path, before feeding the signal to the ADC input. This can be a passive filter.

2.1 What is a BIST?

Built-in self-test, or BIST, means that extra circuitry is added to a design in order for the design to be able “perform operations on [itself] to prove correct operation” [16]. This is then combined with a so called scan technique which allows the registers in the design to be read, so that the result of the test can be examined. Because of the added circuitry, there is a circuit overhead related to the test, but because the test time can be reduced the overall system cost may be lower [16]. A BIST in a FPGA has the advantage that it does not have the circuit overhead

normally associated with a BIST. When the BIST circuitry is not needed in the design anymore it can be removed and the design re-synthesized, which is a huge advantage compared to a BIST in an ASIC.

2.2 Test case

The test case for this project is a Kintex-7 325T FPGA. Kintex is Xilinx's mid-range family of FPGAs. Embedded in this FPGA is a circuit called the XADC, which contains two 12-bit resolution analog-to-digital converters capable of sampling at one mega samples per second. The XADC has system monitor capabilities which mean that the ADCs can be used to monitor on-chip temperature sensors and supply voltage, but it can also be used to convert external analog signals. There are two dedicated analog input pins to the ADC which are used in this project [17]. The FPGA is mounted on a KC705 evaluation board from Xilinx, which can be seen in Fig. 2.1.

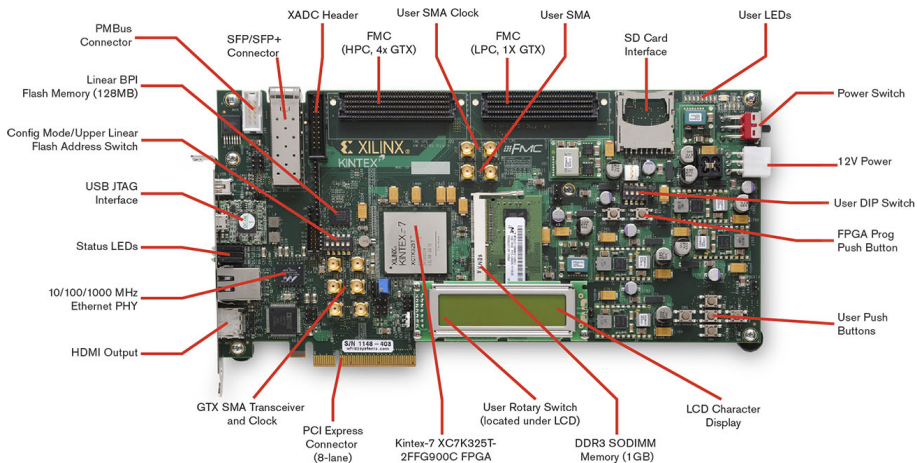


Figure 2.1: The KC705 evaluation board.

Image source: www.xilinx.com/products/boards-and-kits/EK-K7-KC705-G.htm

The KC705 features power supplies, I/O connectors, push buttons, an LCD display, crystal oscillators for clock generation, on-board memory, etc., which makes it a convenient platform for trying out a new FPGA designs. In this project the user push buttons in the bottom right corner are used as start and reset buttons for the system, and the user SMA connector in the middle of the board is used to output signals from the delta-sigma modulator for measurement with a spectrum analyzer. A 200 MHz fixed-frequency oscillator on the board is used to generate a system clock for the BIST and for evaluating the delta-sigma DAC. To the top-left of the board the XADC header is located and the two connector pins closest to the upper edge of the board are the dedicated analog input pins to the ADC. Between the input pins and the input of the ADC, soldered to the back of

the board, is an antialiasing RC-filter

Table 2.1: Specifications of test case ADC. It is a 12-bit, 1 MS/s ADC which is embedded in the Kintex-7 FPGA.

Parameter	Value
Resolution (bits)	12
Integral nonlinearity (LSB)	± 3
Differential nonlinearity (LSB)	± 1
Offset error (LSB)	± 6
Gain error (%)	± 0.5
Sample rate (MS/s)	0.1 – 1
Signal-to-noise ratio (dB)	60
Total harmonic distortion (dB)	70
Unipolar input range (V)	0 – 1
Bipolar input range (V)	-0.5 – 0.5
Unipolar common mode range (V)	0 – 0.5
Bipolar common mode range (V)	0.5 – 0.6
ADC clock frequency (MHz)	1 – 26

Specifications of the test case ADC [19] can be seen in Table 2.1. As mentioned this is a 12-bit, 1 MS/s ADC, and there are two of these in the XADC circuit embedded in the FPGA chip. The first four parameters after the resolution in Table. 2.1—integral nonlinearity, differential nonlinearity, offset and gain error—are measures of the ADC’s static performance. They will be discussed in detail in chapter 3.2. Regarding gain and offset error there is a setting for automatic calibration which gives smaller errors, but the uncalibrated values are shown here. The sample rate can be chosen between 0.1 and 1 MS/s, but usually it is the performance at the highest possible sample rate that is of most interest. Signal-to-noise ratio and total harmonic distortion are measures of the ADC’s dynamic performance, and will be discussed in detail in chapter 3.3. The ADC supports both unipolar and bipolar mode. For the bipolar mode a fully differential input signal can be used. For a 1 MS/s sample rate a 26 MHz ADC clock frequency is needed, but a higher frequency system clock can be divided down and used as ADC clock.

2.3 The system

In Fig. 2.2 a block diagram of the BIST is shown. A delta-sigma modulator in the FPGA fabric is used to generate a test signal. This signal is fed out through the I/O of the FPGA and is then filtered by an external analog filter before going to the dedicated analog inputs of the ADC. The filter is necessary to remove high-frequency quantization noise, which will be discussed in chapter 4. The test signal is generated continuously and the ADC is sampling at a fixed sample rate. The top-level Verilog block consists of a finite state machine (FSM) which will be shown in detail in chapter 5. When a start button on the KC705 board is pressed

the top-level collects samples from the ADC, and when enough samples have been collected it starts algorithms that calculates the performance of the ADC based on the collected samples. These are the algorithms that were written in C code and synthesized using HLS. Once the algorithms are finished, a done signal is returned and the top-level FSM goes to an idle state where it resides until the start button is pressed again.

Integrated into the design are ILAs which can be used to read the test results. Each ILA has a trigger and a capture signal that can be chosen by the designer. If the ILA is armed—this is done in Vivado’s hardware manager—it starts to capture data from a designated register in the design when the trigger signal goes high. The register is then sampled by the ILA each clock cycle a chosen condition, based on the capture signal, is met. If for example the condition is that a capture signal C0 is equal to one, then the register will only be sampled at the clock cycles where “C0 = 1”. If there is no capture condition set, however, the ILA will sample the register every clock cycle after the trigger goes high. Each sample that an ILA collects is stored in its memory, which consists of block RAM, and the data capture continues until the memory is full. The memory content can then be read through a JTAG interface using Vivado’s hardware manager. There is a data ready signal that goes high one clock cycle for each ADC conversion, and this signal can be used both as trigger and capture signal for an ILA that captures ADC codes from the ADC data register. Thus data from the ADC can be obtained in a convenient way.

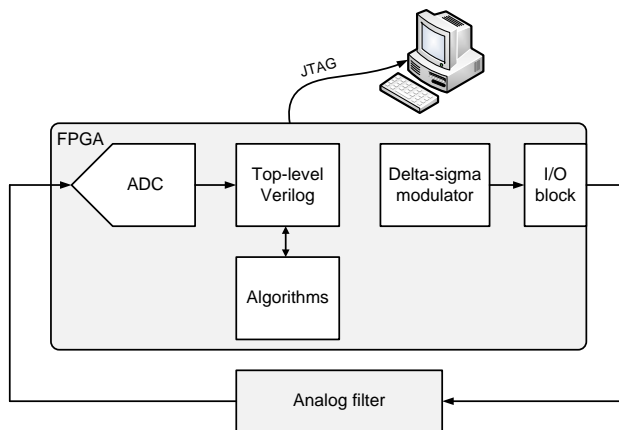


Figure 2.2: Block diagram of the BIST. The BIST consists of a delta-sigma DAC that generates a test signal, an analog filter, the embedded ADC that is tested and algorithms that calculates the performance of the ADC. Test results can be read through the JTAG interface.

One important thing to note about this setup is that there are no external active components used in the test. Typically a signal generator or external DAC would be used to generate the test signal for the ADC. The argument for not using external active components is that those components then would have to be calibrated,

or their accuracy somehow guaranteed in order to ensure that the errors found in the test are indeed in the ADC itself and not the test signal. It is desirable to avoid this situation.

2.4 Interface between stimulus and ADC

The interfaces between the I/O block, the analog filter, and the input of the ADC are in practice much more complicated than Fig. 2.2 indicates. If the output of the delta-sigma DAC is differential, a differential-to-single ended converter may be needed in front of the filter. If a differential signal is needed at the input of the ADC—which is desirable if the ADC is tested in bipolar mode—the signal has to be converted from single-ended to differential after the filter. For this test case a DC offset is needed at the input of the ADC and if a bandpass filter is used as analog filter, this will remove any DC offset from the delta-sigma DAC. A balun-circuit could be used both to convert the signal from single-ended to differential and to introduce a DC offset. It might be necessary to insert a voltage divider in the signal path. If for instance the DAC outputs a signal between 0 and 1.8 V and the input range of the ADC is 1 V, the signal has to be attenuated. But because the amplitude and offset of the test signal is desirable to know with good precision, care must be taken to avoid an unintentional attenuation at the input of the ADC.

2.5 Summary

The concept of a built-in self-test is that extra circuitry is added to a circuit so that it can test itself. In the spirit of this concept an FPGA can be programmed with algorithms that test circuitry that is embedded in the FPGA chip, and this test logic can be removed when it is not needed anymore. In this thesis project a BIST for embedded ADCs is developed, using a Kintex-7 FPGA as test case. Kintex is Xilinx's mid-range family of FPGAs and the embedded ADC is a 12-bit, 1 MS/s ADC, with performance as specified in Table 2.1.

The BIST consists of a delta-sigma modulator in the FPGA fabric, an external analog filter, the ADC under test, a top-level FSM, and test algorithms which are written in C code. The results of the test can be read through the JTAG interface by the help of integrated logic analyzers. The interfaces between the delta-sigma DAC, the analog filter and the input of the ADC is more complicated than a first look reveals. Differential vs. single-ended signaling and signal attenuation must be considered carefully.

3

Theory of ADC testing

This project is based on the *IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters* (IEEE Std 1241). The standard defines many test methods and aspects of ADC performance, but only a subset of those is discussed in this report. The performance measurements of interest here can be divided into two categories; static test and dynamic test. The static test measures an ADC's excursion from its ideal behavior when sampling a slowly varying (relatively low frequency) signal. The dynamic test measures the spectral purity of the of the ADC's digital output signal, when the ADC sample a relatively high frequency signal.

Many of the metrics used to quantify performance of ADCs can be defined in more than one way, and also measured in more than one way. This is especially true for the static test, where for example the gain error—mentioned in the ADC specifications, Table 2.1, and discussed in this chapter—can be defined either based on transfer curve end points or based on a best-fit line. A difference in definition of a metric often introduces only a small, but significant, difference in measurement result. The exact definition and test methodology used is generally not disclosed in ADC data sheets, which makes it difficult to compare two ADCs and to reproduce measurement results found in a data sheet.

3.1 About the IEEE standard 1241

The IEEE standard 1241 attempts to provide a common ground for ADC tests and testing methods. If test methods and definitions from this standard are used, and if they are disclosed properly when presenting the results, it is easier to compare ADCs and to reproduce measurement results. This makes the information more

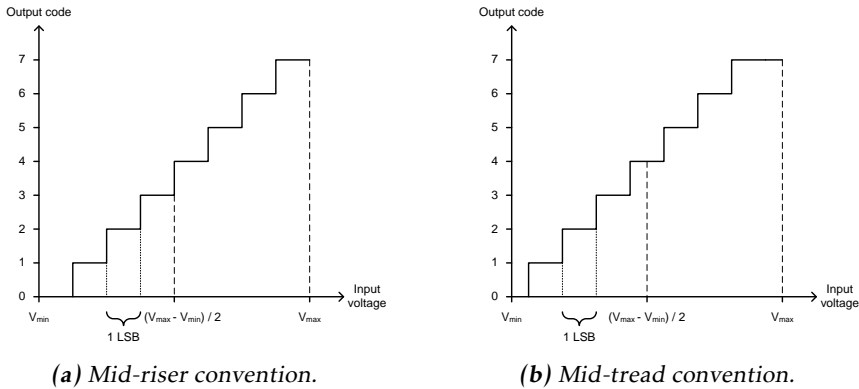


Figure 3.1: The two transfer function conventions. In the mid-riser convention the mid-point of the input voltage range is at a riser in the staircase-looking transfer curve, and in the mid-tread convention the mid-point is at a tread of this staircase.

usable, and thus more valuable.

3.2 Static test

The idea behind what is usually referred to as a static ADC test, is to excite the ADC with an input signal of low enough frequency so that the signal can be considered a DC, or static signal. The output codes of the ADC are then compared with what is expected from an ideal ADC model. When designing a static ADC test there are a number of choices to be made. There are several, equally valid, ways to define the ideal behavior and the ADC's excursion from this ideal behavior. Below these choices are discussed.

3.2.1 ADC transfer function

In Fig. 3.1 the ideal transfer function of an ADC can be seen. On the x-axis is the voltage of the input signal of the ADC, and on the y-axis are the output codes of the ADC. The ADC has 2^N output codes, from 0 through $2^N - 1$ inclusively, where N is the number of bits (also called resolution) of the ADC. The figure illustrates a 3-bit ADC so it has eight codes, 0 through 7 inclusively.

There are two different conventions for the ideal transfer function. The first one is called the mid-riser convention because the mid-point of the input voltage range is at a riser in the staircase-looking transfer function graph, and the second one is called mid-tread convention because the mid-point is at a tread of the staircase. Note that the first transition level of the ADC (the voltage for which the ADC transitions from code 0 to code 1) is one LSB up from the minimum input voltage for the mid-riser convention, but half an LSB up from the minimum voltage for

the mid-tread convention. LSB is an abbreviation for *least significant bit*, and in this context it stands for the voltage step that corresponds to a minimum code change of the ADC. Here LSB is taken to mean the ideal LSB which is defined as

$$LSB = \frac{v_{max} - v_{min}}{2^N} \quad (3.1)$$

where N is the resolution of the ADC. An alternative definition of the LSB is used in [3] where the last and the first transition levels of the ADC are obtained in some way, and the LSB is defined as

$$LSB = \frac{v_{rf}}{2^N - 2} \quad (3.2)$$

where v_{rf} is the reduced full-scale voltage, which is the difference between the last and the first transition level.

Fig. 3.1 shows an ADC with an unsigned binary representation of the code words which is often the case for an ADC in unipolar mode. For bipolar mode the codes are often given in two's complement representation. If the ADC output codes are represented with some other coding scheme, the codes should be mapped to unsigned binary representation before using the tests explained in this section [2].

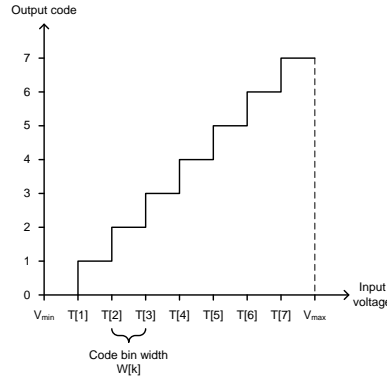


Figure 3.2: ADC transition levels. The transition levels are the voltages for which the ADC changes from one output code to the next.

Fig. 3.2 illustrates the transition levels of the ADC. As mentioned above the first transition level is the input voltage for which the ADC transitions from code 0 to code 1. In general the transition level $T[k]$ is the input voltage for which the ADC transitions from code $k - 1$ to code k . There is always noise present in an ADC, both inherent noise of the ADC itself and noise from the input signal and external circuitry. This noise can be viewed as a random addition to the input voltage and hence it is not possible to define exact voltages for which the ADC transitions take place. Instead the transition level $T[k]$ is defined statistically

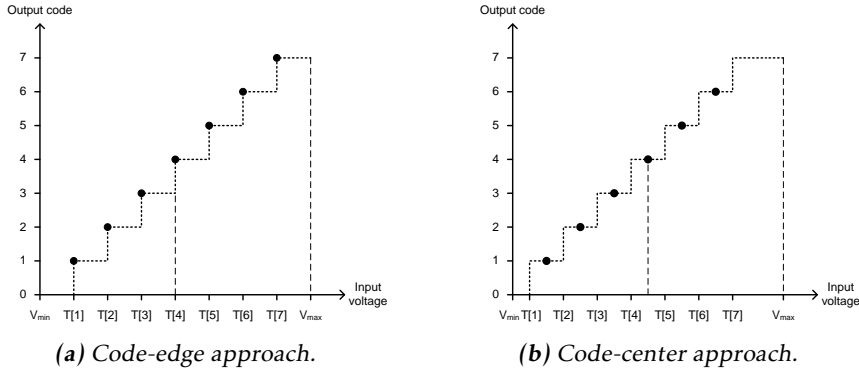


Figure 3.3: Either the transition levels can be used directly to calculate gain and offset, called code-edge approach, or the mid-points of the code bins can be used to calculate gain and offset, which is referred to as the code-center approach.

as the input voltage for which 50% of the output codes are code $k - 1$ or less, and 50% of the output codes are code k or higher. Note also from Fig. 3.2 the definition of code bin width. The voltage span between transition levels $T[k]$ and $T[k + 1]$ is referred to as the code bin k with corresponding code bin width $W[k]$, which is nominally equal to 1 LSB.

In essence, the static test consists of finding the transition levels of the ADC and compare these voltage levels to the ideal ones. The difference between the tested device and the ideal case is then quantified by four metrics: gain, offset, differential nonlinearity (DNL) and integral nonlinearity (INL).

Assume for the moment that the transition levels have been measured with satisfying precision by some method (will be discussed shortly). To decide gain and offset of the ADC, the transition levels can be used directly for comparison with the ideal case. This is called the code-edge approach and is illustrated in Fig. 3.3a. Alternatively the mid-point of each code bin can be calculated and then used to compare with the ideal case, as illustrated in Fig. 3.3b. This is called the code-center approach. As can be observed by comparing Fig. 3.3 to Fig. 3.1, the code-edge approach is typically used in conjunction with the mid-riser convention transfer function, and the code-center approach is typically used in conjunction with the mid-tread convention transfer function. These combinations are suggested in the IEEE Std 1241, and the results of the two can be directly compared, but they are not necessary choices. In [5] it is noted that an offset of plus or minus half an LSB can be introduced in the result depending on which ideal transfer function that is assumed. So if for example the code-edge approach is used with the mid-riser convention in one case, and the code-edge approach is used with the mid-tread convention in another case, an extra difference of half an LSB is to be expected between the results.

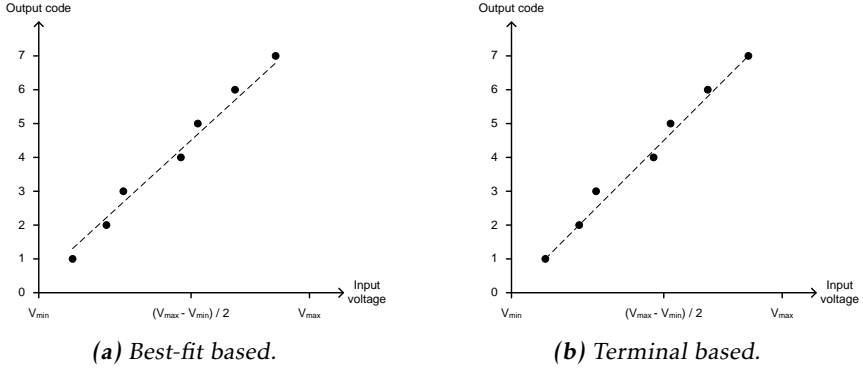


Figure 3.4: The dots indicate the transition levels of the ADC. A line is drawn based on the calculated data points, and is used to calculate gain and offset. The line can either be drawn as a least sum of squares best-fit line, as in (a), or between the endpoints, as in (b).

There are many ways to define gain and offset [5] and there is little consistency in the choice of definition. The two options that are recommended in the IEEE Std 1241 are based on drawing a line, which fits the transition levels, through the transfer function graph as illustrated in Fig. 3.4. This line is then used to calculate gain and offset. Figure (a) shows what is called the best-fit based approach, where the line is drawn as a best-fit line in the least sum of squares sense, and in (b) the terminal based approach is shown, where the line is drawn from the first to the last transition level. In this project the best-fit line based approach is used.

3.2.2 Performance metrics

As mentioned above there are four metrics which are used to quantify the difference between the transfer function of the ADC under test and the ideal transfer function. These are: gain, offset, DNL and INL, and are defined in [2]. The definitions, using the best-fit line approach, can be seen in Table 3.1.

Gain and offset

To calculate the gain and offset, the following equation is used

$$GT[k] + V_{os} + \epsilon[k] = LSB \cdot (k - 1) + T_1 \quad (3.3)$$

where G is the gain, $T[k]$ is the measured transition levels, V_{os} is the offset, ϵ is the error for each index k and T_1 is the ideal position of the first transition level. For the mid-riser convention the ideal position of the first transition level is at one LSB above the minimum input voltage, so T_1 is equal to $V_{min} + 1LSB$.

On the right-hand side of Eq. 3.3 are the ideal transition levels. For the first one,

Table 3.1: Static performance metrics as defined in the IEEE Std 1241. These four parameters quantify the static error of the ADC compared to its ideal behavior.

Name	Description
DNL	Differential nonlinearity is the difference between a specified code bin width and the average code bin width, divided by the average code bin width.
INL	Integral nonlinearity is the maximum difference between the ideal and actual code transition levels after correcting for gain and offset.
Offset	Offset is the value by which the input values are added to minimize the mean square deviation from the output values.
Gain error	Gain is the value by which the input values are multiplied to minimize the mean square deviation from the output values. Gain error is the deviation from the ideal gain in percent.

k is equal to 1, so it is located at T_1 , which is $V_{min} + 1LSB$. For the second one k is equal to 2, so it is located at $V_{min} + 2LSBs$, etc. On the left-hand side are the measured transition levels $T[k]$. The task is to find the gain G and the offset V_{os} which minimizes the mean squared deviation from the best-fit (linear regression) line. This is done by minimizing the sum of square of the errors (SSE). How to minimize the SSE is explained in statistics literature, for example in [15]. In this case Eq. 3.3 can be rearranged as

$$\epsilon[k] = LSB \cdot (k - 1) + T_1 - GT[k] - V_{os}. \quad (3.4)$$

The index k is the index of the transition levels and runs from 1 to $2^N - 1$, so the SSE can be expressed as in Eq. 3.5, assuming T_1 is equal to $V_{min} + 1LSB$.

$$SSE = \sum_{k=1}^{2^N-1} \epsilon^2[k] = \sum_{k=1}^{2^N-1} (LSB \cdot k + V_{min} - GT[k] - V_{os})^2. \quad (3.5)$$

This is a function of the two variables G and V_{os} and to find the minimum, the

partial derivatives are calculated and set to zero.

$$\frac{\partial(SSE)}{\partial V_{os}} = -2 \sum_{k=1}^{2^N-1} (LSB \cdot k + V_{min} - GT[k] - V_{os}) = 0 \quad (3.6a)$$

$$\frac{\partial(SSE)}{\partial G} = -2 \sum_{k=1}^{2^N-1} (LSB \cdot k + V_{min} - GT[k] - V_{os})T[k] = 0 \quad (3.6b)$$

Eqs. 3.6 (a) and (b) form an equation system which can be solved for G and V_{os} . Doing so results in the following expressions for gain and offset:

$$G = \frac{LSB \cdot (2^N - 1) \left(\sum_{k=1}^{2^N-1} kT[k] - 2^{(N-1)} \sum_{k=1}^{2^N-1} T[k] \right)}{(2^N - 1) \sum_{k=1}^{2^N-1} T^2[k] - \left(\sum_{k=1}^{2^N-1} T[k] \right)^2} \quad (3.7)$$

$$V_{os} = LSB \cdot 2^{(N-1)} + V_{min} - \frac{G}{2^N - 1} \sum_{k=1}^{2^N-1} T[k]. \quad (3.8)$$

Gain and offset are illustrated in Fig. 3.5. In (a) the dotted line indicates the best-fit line of the ideal transfer curve and the solid line indicates the best-fit line of the measured transition levels. The gain is nominally 1. In this example the tested ADC has a gain of 0.9, which can also be referred to as a gain error of -10%. In (b) the dotted line again shows the ideal case and the solid line the tested ADC. In this case the gain of the tested ADC is unity, but it has an offset of -1.5 LSBs. Note that this is a negative offset, which can be understood by looking at the definition: “offset is the value by which the input values are added to minimize the mean squared deviation from the output values”. If -1.5 LSBs were added to the x-coordinate of each point of the measured line, it would shift to the ideal position. Offset stated in units of LSBs is often referred to as offset error.

DNL and INL

DNL and INL are calculated with Eq. 3.9 and Eq. 3.10 respectively [2].

$$DNL[k] = \frac{G \cdot (T[k+1] - T[k]) - LSB}{LSB}. \quad (3.9)$$

$$INL[k] = \frac{G \cdot T[k] + V_{os} - LSB \times (k-1) - T_1}{LSB}. \quad (3.10)$$

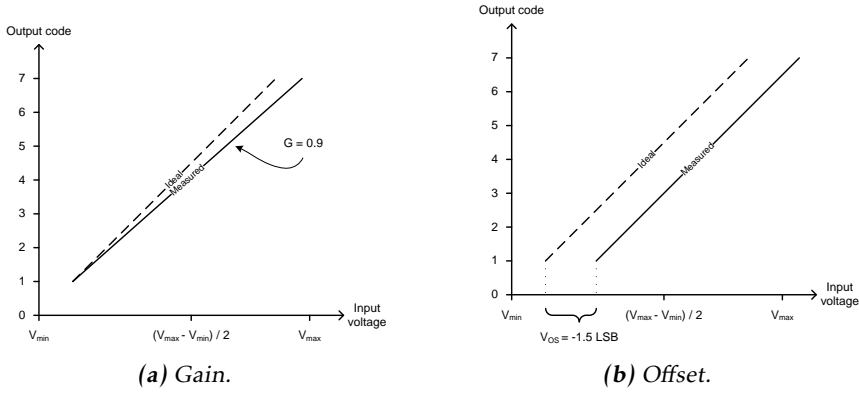


Figure 3.5: Example of gain and offset. The dotted lines indicates the ideal behavior and the solid lines are the best-fit lines based on measured transition levels.

The DNL is the difference in code bin width $W[k]$, after compensating for gain, to the ideal code bin width, which is 1 LSB, expressed in LSBs. The INL is the error of each transition level, after compensating for gain and offset, expressed in LSBs. An example of INL and DNL is shown in Fig. 3.6. Assume that the transition levels have been measured, gain and offset have been calculated, and after compensating for gain and offset the transfer function in Fig. 3.6 is obtained. After compensation, all transition levels are exactly in the right place, except for $T[3]$ which is half an LSB off. In this example we have:

$$DNL[2] = \frac{W[2] - LSB}{LSB} = \frac{1.5 - 1}{1} = 0.5LSB \quad (3.11)$$

$$DNL[3] = \frac{W[3] - LSB}{LSB} = \frac{0.5 - 1}{1} = -0.5LSB \quad (3.12)$$

$$INL[3] = 3.5 - 3 = 0.5LSB. \quad (3.13)$$

The rest of the code bins and transition levels have no errors.

3.2.3 How to obtain the transfer levels

Up until now it has been assumed that the transition levels have been measured in some way. But how are they actually obtained? There are three options listed in the IEEE Std 1241: a servo (or feedback loop) approach, a ramp histogram approach and a sine-wave histogram approach.

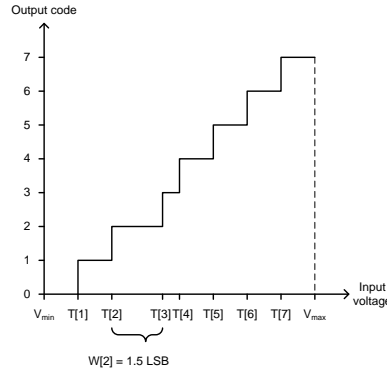


Figure 3.6: Example of DNL and INL errors after compensating for gain and offset. After compensation all transition levels are in their correct (ideal) positions except for $T[3]$ which is half an LSB off. Hence there are DNL and INL errors around that transition level.

The first two are discussed briefly here, but focus is on the sine-wave based approach, which is the one that is used in this project.

Servo approach

In the servo approach each code transition is found using a servo feedback loop as illustrated in Fig. 3.7. The DAC, which has to have higher resolution and better linearity than the ADC under test, feeds a voltage to the ADC and the output code of the ADC is compared to the desired code. If it does not match, the control logic increments or decrements the input code of the DAC. In practice this process might be more elaborate since the definition of transition level $T[k]$ is that 50% of the codes are k or higher, and 50% of the codes are lower than k . When the feedback loop has located the transition level, its voltage can be measured with a voltmeter.

If X number of samples is required to find each transition level, then $X \cdot (2^N - 1)$ samples are needed to find all transition levels. If there is some information about at which code bins the largest errors will occur, the servo method is suitable since only the transition levels of interest then need to be measured.

A drawback with this method is that a good, high resolution, DAC is needed. Typically a DAC with 3 to 4 bits higher resolution than the ADC under test is used. If e.g. a 12-bit ADC is tested a 16-bit DAC can be used. The LSB of the DAC is then 1/16 of the LSB of the ADC, so the transition levels of the ADC can be decided to the precision of 1/16 LSB.

Histogram approach

The idea behind the histogram approach is to let the ADC sample a signal and then use the sampled codes to create a histogram. Each bin in the histogram corresponds to a code bin of the ADC, so for a 3-bit ADC the histogram would

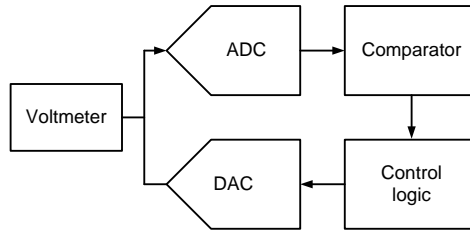


Figure 3.7: Servo (also called feedback) approach to finding transition levels. Based on information from the comparator, the control logic adjusts the DAC code, and thus the input voltage of the ADC, until a transition level between two ADC codes have been found. The transition level can then be measured with a voltmeter.

have eight bins, where the zeroth bin correspond to code 0 of the ADC, the first bin correspond to code 1 of the ADC etc.

If the input signal is evenly distributed over the input range of the ADC, the histogram would ideally have equally many hits in each of the histogram bins. But if a code bin is wider than the nominal 1 LSB the probability that a sample will end up in that bin is slightly higher and that histogram bin receives more hits. Hence the number of hits a histogram bin receives in the test is proportional to the width of the corresponding code bin. See for example Fig. 3.8 where bins 1, 3 and 5 have received slightly more hits than average and are therefore a bit wider than the nominal 1 LSB.

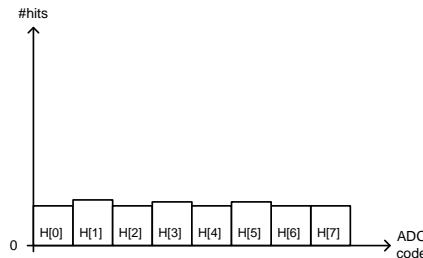


Figure 3.8: Histogram generated from the ADC codes in a test using an input signal that is uniformly distributed over the input range of the ADC. Each histogram bin corresponds to an ADC code bin.

If the number of hits in each bin is compared to the average number of hits over all bins—which is the expected number of hits per bin if all codes would be of equal width—the DNL error can be calculated. This is because the DNL is, as mentioned earlier, a measure of the difference between the ideal code bin width and the measured code bin width. Furthermore, if the first transition level can be determined by some other method, e.g. a feedback loop search, the rest of the transition levels can be calculated from the histogram. In this case the second

transition level is calculated as the first transition level plus the width of the first code bin, the third transition level is the first transition level plus the width of the first and the second code bin etc.

The input signal of choice is usually a slow ramp function. Such a function has an uniform voltage distribution over the input range of the ADC and changes slowly enough to be considered a static signal. Either a rising or a falling ramp, or a triangular wave with both rising and falling edges can be used. In theory a completely randomized signal with an even voltage distribution over the input range of the ADC can be used, but such a signal is difficult to generate. The drawback with the ramp histogram approach is that it is difficult to generate a ramp signal with good linearity [7], although a high resolution DAC can be used to generate it. Fig. 3.9 shows an example of a ramp input and its probability density function (PDF). Since the distribution is flat, a random sample from the ramp is equally likely to belong to any of the code bins and hence the histogram is expected to be flat for an ideal ADC.

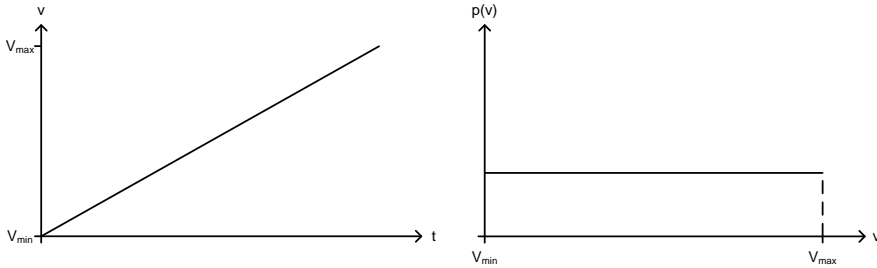


Figure 3.9: Ramp signal and its probability density function. Because it is a simple function with an uniform voltage distribution, the ramp signal is a popular choice for histogram linearity tests of ADCs.

To ensure that the entire input range of the ADC is “covered” by the input ramp, even if there is gain error and offset in the ADC, the ramp is usually chosen to overdrive the ADC. This is done by letting the ramp start at a voltage slightly lower than V_{min} and end at a voltage slightly higher than V_{max} . Doing this gives many more hits in the zeroth and in the last $(2^N - 1)$ histogram bin, because any sample with a voltage lower than V_{min} will end up code 0, and any sample with a voltage higher than V_{max} will end up code $2^N - 1$. The width of these two code bins are regarded as not clearly defined since their width is not the difference between two transition levels, and they are discarded from the calculation of DNL.

In the sine-wave based histogram approach the ADC samples a sine-wave signal (see Fig. 3.10a) and the resulting codes are used to form a histogram. The histogram will have a shape similar to what is illustrated in Fig. 3.10b which is usually referred to as a “bathtub” shape.

The voltage of a sine-wave signal changes much slower near the peaks of the

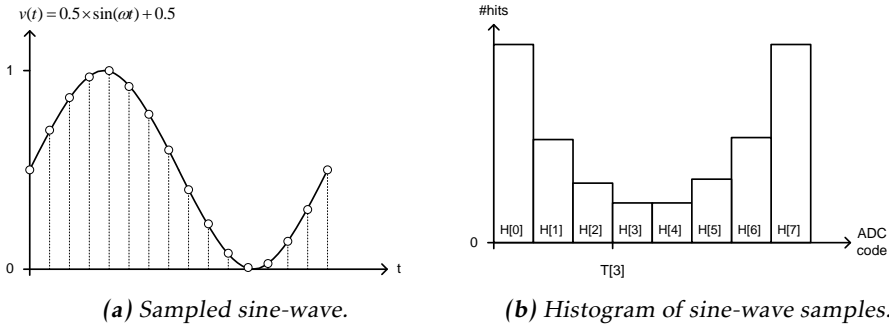


Figure 3.10: A sampled sine-wave and a histogram formed with the resulting ADC codes. Since its PDF is known, a sine-wave can be used as an alternative to input signals with uniform distribution, for histogram based linearity tests.

wave. In fact, its derivative is zero at the peaks. In contrast the voltage changes relatively fast near the mid-point of the wave. This means that if a sample is taken randomly, it is more likely that the sampled voltage will be closer to the peaks than that it will be near the middle. If the voltage is treated as a random variable the PDF of the sine-wave

$$v(t) = A \sin(\omega t) + d \quad (3.14)$$

can be derived [14], resulting in

$$p(v) = \frac{1}{\pi \sqrt{A^2 - (v - d)^2}} \quad (3.15)$$

where A is the amplitude of the sine-wave and d is the offset (in volts). An example of a sine-wave, with peak-to-peak value of 1 V and an offset of 0.5 V, and its PDF is illustrated in Fig. 3.11.

Using the histogram in Fig. 3.10a the probability that an ADC code occurs in the test can be estimated. The probability that a random sample gives code 0 can be estimated as the number of hits in the zeroth histogram bin divided by the total number of samples taken, expressed mathematically as

$$P(\text{code}0) \approx \frac{H[0]}{N_{\text{smpIs}}} \quad (3.16)$$

where $H[0]$ is the number of hits in the zeroth histogram bin and N_{smpIs} is the total number of samples taken in the test. To estimate that a random sample is

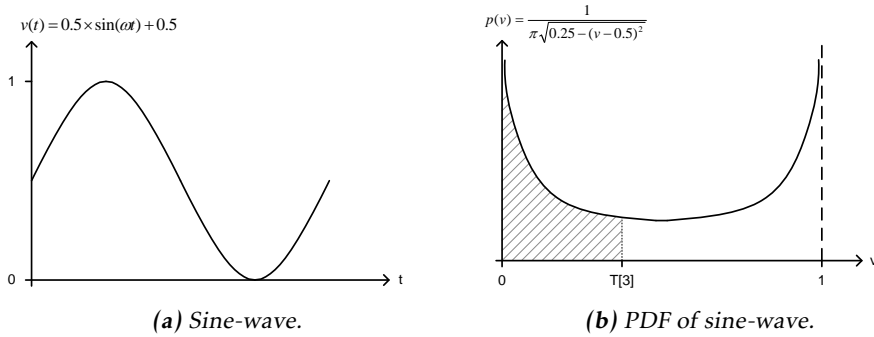


Figure 3.11: A sine-wave and its probability density function. As the PDF graph shows, the distribution is not uniform. If amplitude and offset of the sine-wave are known, however, the PDF can be derived.

either code 0, 1 or 2 the probability that it is code 0, the probability that it is code 1 and the probability that it is code 2 can be added up as

$$P(\text{code0}) \approx \frac{H[0]}{N_{\text{smpIs}}} + \frac{H[1]}{N_{\text{smpIs}}} + \frac{H[2]}{N_{\text{smpIs}}} = \frac{CH[2]}{N_{\text{smpIs}}}. \quad (3.17)$$

CH stands for the cumulative histogram, so $CH[0]$ are all the hits in bin 0, $CH[1]$ are all the hits in bin 0 plus all the hits in bin 1, $CH[2]$ are all the hits in bin 0 plus all the hits in bin 1 plus all the hits in bin 2, etc.

Now notice in Fig. 3.10b that the third transition level of the ADC, $T[3]$, is indicated in the histogram. This is a voltage and is therefore, in a sense, out of place in a histogram plot. What the figure is meant to illustrate is that the probability that a random sample from the sine-wave is of lower voltage than $T[3]$ can be estimated as

$$P(v < T[3]) \approx \frac{CH[2]}{N_{\text{smpIs}}}. \quad (3.18)$$

This estimate is the same as the estimate of the probability that the sample corresponds to code 0, 1 or 2 (Eq. 3.17). The reason for this is that if a sample is of lower voltage than $T[3]$, which is the transition level between code 2 and 3, it has to correspond to code 0, 1 or 2. There is no other possibility. The total area under the PDF in Fig. 3.11 is 1, because there is a 100% chance that a random sample is some voltage between the peak values of the sine-wave. The probability that the voltage is less than $T[3]$ is equal to the shaded area under the graph, and can be calculated by integrating the PDF from 0 to $T[3]$:

$$P(v < T[3]) = \int_0^{T[3]} \frac{1}{\pi\sqrt{0.25 - (v - 0.5)^2}} dv. \quad (3.19)$$

By equating the right-hand side in Eq. 3.18 with the right-hand side in Eq. 3.19 an equation, that can be solved for the transition level $T[3]$, is obtained. In the general case with a transition level $T[k]$ and a sine-wave with amplitude A and offset d it is

$$\frac{CH[k-1]}{N_{\text{smpIs}}} = \int_{d-A}^{T[k]} \frac{1}{\pi\sqrt{A^2 - (v-d)^2}} dv. \quad (3.20)$$

The integral on the right-hand side has a closed form and can, as suggested in [4], be solved by doing the variable change

$$v = Ax + d \Leftrightarrow x = \frac{v-d}{A}, \quad dv = A dx, \quad t[k] = \frac{T[k]-d}{A}. \quad (3.21)$$

Using the substitution in Eq. 3.21 gives the following:

$$\frac{CH[k-1]}{N_{\text{smpIs}}} = \int_{-1}^{t[k]} \frac{1}{\pi\sqrt{1-x^2}} dx = 1 - \frac{\arccos(t[k])}{\pi}. \quad (3.22)$$

Solving Eq. 3.22 for $t[k]$ and then substituting back to $T[k]$ gives

$$T[k] = d - A \cos\left(\frac{\pi CH[k-1]}{N_{\text{smpIs}}}\right), \quad k = 1, 2, 3, \dots, 2^N - 1. \quad (3.23)$$

With Eq. 3.23 a statistical estimate of the transition levels can be calculated from the histogram if the amplitude and offset of the input sine-wave are known. If the amplitude and offset are unknown this formula gives the transition levels within a gain and offset error. If any two transition levels can be measured by other means (e.g. with a feedback loop approach) the amplitude and offset can be obtained from this formula [2].

Just as in the ramp histogram approach the input signal in practice needs to slightly overdrive the ADC, i.e. the peak-to-peak voltage of the input signal has to be higher than the input range of the ADC. This is to ensure all the codes of the ADC are “covered” by the input signal even if there are offset and gain errors in the ADC. The overdrive will skew the probability density function, and there

will be significantly more hits in the end-code bins (bin 0 and bin $2^N - 1$). Since the cumulative histogram is used to calculate the transition levels, Eq. 3.23 will still hold as long as the correct amplitude information is available.

One important thing to note about histogram tests in general is that non-monotonic behavior of the ADC will not be detected [2]. Non-monotonicity of an ADC basically means that some of its code bins have switched places, but the histogram will look roughly the same even if this is the case.

Choice of frequency for the histogram approach

In the derivation of the histogram method it has been assumed that the samples are taken randomly, but in practice the ADC will not sample randomly but at a fixed sample rate. What is done instead is that the frequency of the input signal is chosen in relation to the sample frequency so that the samples are spread equally in phase over the sine-wave. This way the distribution of the codes will be the same as if random sampling was used. In order for this to happen two conditions have to be met: (1) the sampling must be done over an integer number of periods and (2) the input signal frequency must not be harmonically related to the sampling frequency. If the sampling is not done over an integer number of periods the PDF of Eq. 3.15 will not be correct. If the sampling frequency is harmonically related to the frequency of the input signal the samples will not be spread in phase and the ADC will tend to sample the same point of the sine-wave repeatedly. More hits will go into certain histogram bins and the transition level calculation will be incorrect.

These conditions can be met by choosing input frequency, the number of samples and the number of sine-wave periods as

$$\frac{f_{in}}{f_s} = \frac{M_{cycles}}{N_{smpls}} \quad (3.24)$$

where f_{in} is the input frequency, f_s the sampling frequency, M_{cycles} the number of sine-wave periods and N_{smpls} the number of samples taken. M_{cycles} and N_{smpls} have to be chosen relatively prime, i.e. with no common factors except 1 [3]. Often M_{cycles} is chosen to be a prime number, because then N_{smpls} can be any number except for integer multiples of the same prime number, but the condition relatively prime is enough.

The frequency of the input signal should be low enough that the signal can be considered static, because gain error, offset error, DNL and INL are usually considered measures of static errors. In practice higher frequencies can be used and the ADC can be tested at several different frequencies in order to determine its performance at each frequency.

How many samples are needed for the histogram approach?

Using the histogram test, the transition levels of the ADC are not measured exactly. The test instead gives a statistical estimate of the transition levels, and the

statistical accuracy—the precision and confidence level—increases with number of samples taken to form the histogram. There is a derivation in Blair’s paper [3] (and its result is also in the IEEE standard) which gives guidance to how many samples that are needed. With information of RMS noise and accuracy of input frequency, the number of samples necessary to estimate INL and DNL with a certain precision and a certain statistical significance can be calculated. This is potentially very valuable information for production testing.

3.3 Dynamic test

In the dynamic test the ADC is excited with a sine-wave signal of high enough frequency so that dynamic errors occur. A number of samples from the sine-wave are collected and the discrete Fourier transform (DFT) is used to analyze the data. The DFT can be calculated using an FFT algorithm.

3.3.1 Performance metrics

The DFT gives the frequency spectrum of the ADC codes and a number of performance metrics can be calculated from this spectrum. The definition of these, as given in the IEEE standard, can be seen in Table 3.2. An example of a spectrum obtained from the DFT is illustrated in Fig. 3.12.

The total harmonic distortion (THD) is calculated by summing up the harmonics of the fundamental signal, i.e. summing up the power in frequency bins that are at integer multiples of the frequency bin of the input signal. The ratio of the distortion power to the signal power (the difference in dB) is then the THD.

The signal-to-noise ratio (SNR) is calculated by summing up the noise, i.e. the power in all the frequency bins up to Nyquist frequency, except for the bins containing the DC, the fundamental signal and its harmonics. The ratio of the signal to the noise (the difference in dB) is then the SNR. In the example illustrated in Fig. 3.12 the SNR is 60 dB. The signal is the highest peak in the graph, located at approximately 20 kHz. Note that the noise floor appears to be much lower than 60 dB below the signal, which due to a phenomenon often called process gain of the FFT. The FFT noise floor is decreased by increasing the number of points in the FFT, and the process gain can be calculated as $10\log(M/2)$, where M is the number of points [9]. In this case 16384 points are used and the process gain is therefore 39 dB, and as can be seen in the figure the FFT noise floor is roughly $39 + 60$ dB below the signal. The reason behind process gain is that the power of the noise is divided into more frequency bins when the number of points is increased, and therefore there is less power in each bin. If the noise is summed up between two frequencies, however, the result is the same. It is important that the resolution of the FFT is high enough so that harmonics and spurs are not hidden in the FFT noise floor.

Signal-to-noise-and-distortion ratio (SINAD) is the same as SNR with the only difference that the harmonics are included in the noise summation. The spurious-

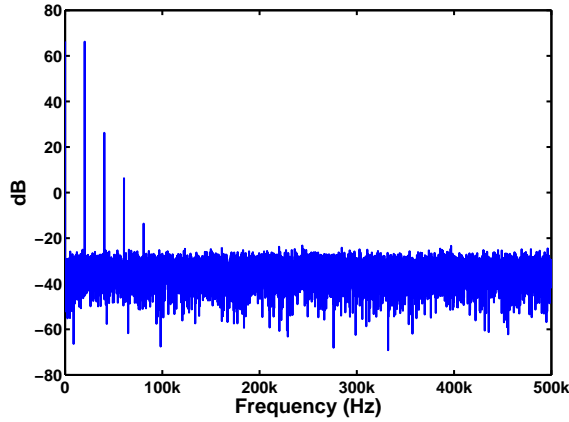


Figure 3.12: Spectrum, of a 20 kHz sine-wave, obtained using DFT. The signal-to-noise ratio is approximately 60dB, and the 2nd, 3rd and 4th harmonic are clearly visible in the spectrum.

free dynamic range (SFDR) is the ratio of signal to the highest noise spur (the difference in dB) in the spectrum. In Fig. 3.12 the highest spur is the second harmonic located at approximately 40 kHz.

Lastly the equivalent number of bits (ENOB) is calculated using the SINAD in the formula for SNR for an ideal ADC (Eq. 3.25). Replacing SNR with SINAD in this formula and solving for N gives the ENOB.

$$SNR = 6.02N - 1.76dB. \quad (3.25)$$

There is one more important thing to note about these metrics, which is that they can either be calculated with respect to the amplitude of the test signal (sometimes called vs. carrier) or they can be calculated with respect to a full-scale signal. In the second case a signal with almost full-scale peak-to-peak value is used. A complete full-scale signal is not possible to use because even a tiny error in the signal amplitude would saturate the ADC which distorts the spectrum.

The FFT

As mentioned the DFT can be calculated using FFT algorithms. In this thesis project a decimation-in-time radix-2 FFT algorithm is used. This algorithm can be derived from the analysis function of the DFT as follows [12] [8].

$$X[k] = \sum_{n=0}^{M-1} x[n]W_M^{kn}. \quad (3.26)$$

Eq. 3.26 is the analysis function of an M-point FFT, where $X[k]$ are the frequency

Table 3.2: Dynamic performance metrics as defined in the IEEE Std 1241.

Name	Description
SNR	Signal-to-noise ratio is the ratio of the rms amplitude of the ADC output signal to the rms amplitude of the output noise, using a pure sine wave input of specified amplitude and frequency. This does not include the harmonic distortion components that are used for the estimate of THD.
SFDR	Spurious-free dynamic range is the ratio of the amplitude of the ADC's output averaged spectral component at the input frequency to the amplitude of the largest harmonic or spurious spectral component observed over the full Nyquist band, using a pure sine wave input of specified amplitude and frequency.
SINAD	Signal-to-noise-and-distortion ratio is, using a pure sine wave input of specified amplitude and frequency, the ratio of the rms amplitude of the ADC output signal to the rms amplitude of the output noise, where noise includes not only random errors but also nonlinear distortion and effects of sampling time errors.
THD	Total harmonic distortion is, using a pure sine wave input of specified amplitude and frequency, the root-sum-of-squares of all the harmonic distortion components including their aliases in the spectral output of the ADC. IEEE Std 1241-2010 suggest the ten first harmonics are used to estimate THD. THD is often expressed as a decibel ratio with respect to the rms amplitude of the output component at the input frequency.
ENOB	Effective number of bits is a measure of the signal-to-noise-and-distortion ratio used to compare the actual ADC performance to an ideal ADC.

domain points, $x[n]$ are the time domain samples (the sampled ADC codes in this case) and W_M is defined as:

$$W_M = e^{-2\pi/M}. \quad (3.27)$$

The sum in the analysis function can be divided into two sums, one for odd n and one for even n as shown in Eq. 3.28, and these two sums are $M/2$ -point DFTs.

$$X[k] = \sum_{n=0}^{M-1} x[n] W_M^{kn} = \sum_{r=0}^{(M/2)-1} x[2r] W_{(M/2)}^{kr} + W_M^k \sum_{r=0}^{(M/2)-1} x[2r+1] W_{(M/2)}^{kr}. \quad (3.28)$$

The two DFTs on the right-hand side of Eq. 3.28 can each be divided into two $M/4$ -point DFT sums and if M is a power of two, this process can be continued until there are $M/2$ 2-point DFT sums left. This is done in $\log_2(M)$ steps. This is illustrated in Fig. 3.13 for $M = 8$. In $\log_2(8) = 3$ steps the 8-point DFT sum is divided into $M/2 = 4$ 2-point sums. the W 's in the flow graph are defined as in Eq. 3.27 and are called twiddle factors.

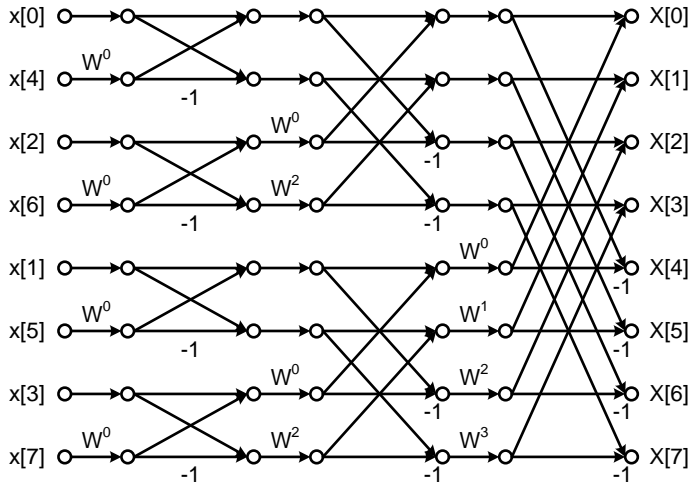


Figure 3.13: 8-point FFT flow graph. On the left-hand side are the time domain samples, and on the right-hand side is the frequency domain data.

To the left of the flow graph are the time domain samples and to the right are the frequency domain samples. One thing to note here is that the frequency domain samples are in order (0,1,2,...,7) but the time domain samples are not. They are instead in something called bit-reversed order. The reason behind this name can be understood with help of Table 3.3.

Table 3.3: Bit-reversed order. If the bits in the binary representation of each index are reversed, the indices are obtained in bit-reversed order. This is the order of the time domain samples in the radix-2 decimation-in-time FFT.

Indices in ascending order	Binary representation of indices in ascending order	Bit-reversed binary representation	Indices in bit-reversed order
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

In the left-most column are the indices in ascending order and in the second column is the binary representation of each number. In the third column the elements in the binary representation are reversed in order, so that e.g. 100 (which is 4) becomes 001 (which is 1) and 011 (=3) becomes 110 (=6) etc. The numbers with symmetric binary representation, e.g. 101 (=5), stays in their position since reversing their bits does nothing. In the right-most column are the numbers in bit-reversed order which is the order of the time domain samples in the FFT flow graph in Fig. 3.13. The reason the samples are in this order is that the FFT algorithm successively divides the time domain samples into groups of odd and even (Eq. 3.28) [12].

Choice of frequency

The frequency of the signal should be high enough for dynamic errors in the ADC conversion to occur, but the ADC can be tested at several different frequencies in order to determine its dynamic performance at each frequency.

The DFT requires that an integer number of periods of the sine-wave are sampled [10]. This is because it is assumed that the sampled signal is periodic with period M , where M is the number of samples used to calculate the DFT. Furthermore the frequency of the signal should not be harmonically related to the sample frequency. If it is, the samples will not be spread evenly over the phase of the sine-wave. Fulfilling these two requirements is referred to as coherent sampling, and note that the requirements are the same as for the static test, so the frequency of the input signal can be chosen using Eq. 3.24. When using a radix-2 FFT algorithm the number of samples N_{smples} must be chosen as a power of 2. Since the sampling frequency is typically fixed in an ADC test this somewhat limits the possible input frequencies, because the number of sine-wave periods (M_{cycles}) is the only parameter left to change.

There is one more consideration for the input frequency in the dynamic test. The frequency must be chosen so that the signal and its harmonics do not fold on top of each other (aliasing). If for example the sample frequency is 1 MHz, then the Nyquist frequency is 500 kHz. If a 250 kHz input signal is used, its third harmonic will fold back at 250 MHz and might fool the algorithms that calculate SNR, SINAD and THD.

3.4 Summary

The ADC tests covered in this thesis project can be divided into the two categories static and dynamic. In the static test the ADC's excursion from an ideal ADC model is determined by measuring its transition levels, and the excursion is quantified by the four metrics: gain error, offset, differential nonlinearity (DNL) and integral nonlinearity (INL). There are several ways to obtain the transition levels, and in this project a sine-wave based histogram approach is used. In the dynamic test the ADC is excited with a sine-wave input and the spectrum of its output codes is calculated using discrete Fourier transform, often with the help of fast Fourier transform algorithms. From the obtained spectrum signal-to-noise ratio (SNR), total harmonic distortion (THD), etc., can be calculated.

One difficulty with ADC testing is that there are many, equally valid, ways to define important performance metrics, and there are more than one way to measure them. The definition and measurement method used is usually not given in ADC data sheets, which makes it hard to compare ADCs performance. The *IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters* (IEEE Std 1241) can be used as reference. If this standard is referred to, and definitions and test methodology are stated clearly, it is easier to interpret the test results and to compare ADCs.

4

Theory of delta-sigma DACs

The delta-sigma modulator is a concept that can be used for both ADCs and DACs, and in this project it is used to implement a DAC. The idea behind a delta-sigma DAC is to input a digital signal to its modulator at a much higher rate than the frequency of the signal that is going to be generated by the DAC. This together with the use of feedback loops in the modulator increases the signal-to-quantization-noise ratio (SQNR).

4.1 First-order delta-sigma modulator

In Fig. 4.1 a diagram of a first-order modulator is shown. In this example, digital codes representing a 20 kHz sine-wave are fed to the modulator (X_{in}) at a rate of 8 MHz, and the signal comes out from the quantizer (X_{out}) at the same 8 MHz rate. The quantizer quantizes the signal into 1 bit, so there is only one bit coming out of the modulator. This bit is then turned into a voltage at the output, and this together with the modulator constitutes the delta-sigma DAC ¹.

Because the signal is quantized into only one bit, the quantization noise coming out of the modulator is very high. The high data rate and the feedback loop do however “shape” this quantization noise so that the noise is pushed out of the frequency band of interest. Looking at the spectrum of the signal coming out of the modulator Fig. 4.2 is obtained, where the generated sine-wave signal can be seen as a tone at 20 kHz. Due to the feedback loops the noise in the spectrum is shaped so that it is very low at low frequencies, but very high at high frequencies.

¹The conversion of the digital bit, coming out of the modulator, into a voltage can be thought of as the operation of a 1-bit DAC. Looking at it this way the delta-sigma DAC consists of a delta-sigma modulator connected to a 1-bit DAC.

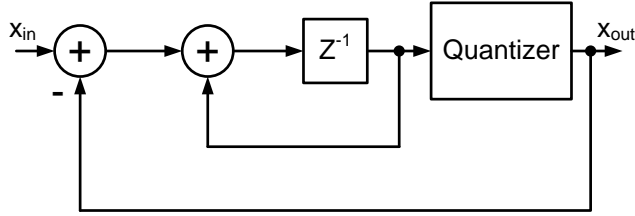


Figure 4.1: Block diagram of a first-order delta-sigma modulator.

The noise in the spectrum has a slope of 20 dB per decade. In total the noise power is higher than the signal, but if a lowpass filter with a cut-off frequency just above the frequency of the generated sine-wave is attached to the output, most of the quantization noise can be filtered out. Left is a sine-wave with a very high signal-to-noise ratio.

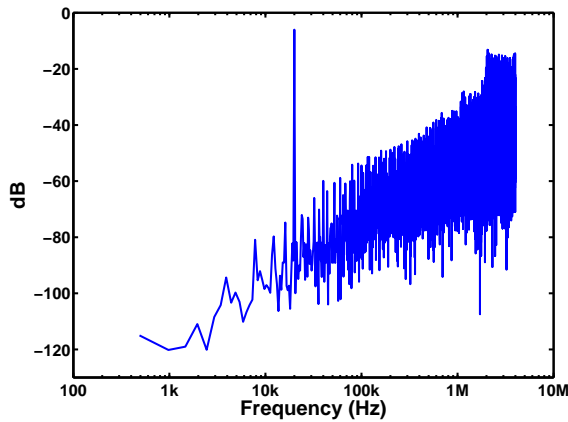


Figure 4.2: Spectrum of the output signal of a first-order delta-sigma modulator, simulated in Matlab.

To get an estimate of the performance of the delta-sigma DAC a theoretical signal-to-noise ratio can be calculated, assuming a full-scale sine-wave is generated. The SNR, expressed in decibels, is calculated as

$$SNR = 10 \log \left(\frac{V_{s,rms}^2}{V_{n,rms}^2} \right) \quad (4.1)$$

where $V_{s,rms}$ and $V_{n,rms}$ are the root mean squared voltages of the signal and the noise respectively. Let the signal be a sine-wave

$$v_s(t) = A \sin(\omega t) = A \sin\left(\frac{2\pi}{T} t\right) \quad (4.2)$$

with amplitude A and period T . Its rms value is calculated as

$$V_{s,rms}^2 = \frac{1}{T} \int_0^T v_s^2(t) dt = \frac{A^2}{T} \int_0^T \sin^2\left(\frac{2\pi}{T} t\right) dt = \frac{A^2}{2}. \quad (4.3)$$

The highest input code of an N -bit DAC is 2^N which corresponds to the highest possible output voltage: $LSB \cdot 2^N$. The amplitude of a sine-wave is equal to half its peak-to-peak voltage, and hence the amplitude of a full-scale sine-wave is

$$A = \frac{LSB \cdot 2^N}{2}. \quad (4.4)$$

By inserting this expression for amplitude into Eq. 4.3, and taking the square root of both sides of the resulting equation, the following expression for signal rms voltage is obtained:

$$V_{s,rms} = \frac{LSB \cdot 2^N}{2\sqrt{2}}. \quad (4.5)$$

The quantization error can be modeled as a linear addition of noise in the circuit; obtaining a linear model of the delta-sigma modulator seen in Fig. 4.3. By approximating the quantization noise as independent of the input signal, it can be viewed as random noise uniformly distributed between $-LSB/2$ and $LSB/2$ [6] as illustrated in Fig. 4.4. In a quantization process the signal is set to the closest quantization level, and since the distance between two adjacent quantization levels is one LSB, the quantization error can never exceed half an LSB. The assumption is that the quantization error can be regarded as uniformly distributed noise. Since the total area under the graph of a PDF must be 1, the height of the function curve in Fig. 4.4 is $1/LSB$, and the PDF is defined as

$$p(v) = \begin{cases} \frac{1}{LSB}, & \text{if } -\frac{LSB}{2} \leq v \leq \frac{LSB}{2}. \\ 0, & \text{otherwise.} \end{cases} \quad (4.6)$$

The rms value of the quantization noise can be calculated from its PDF with the following equation [6]:

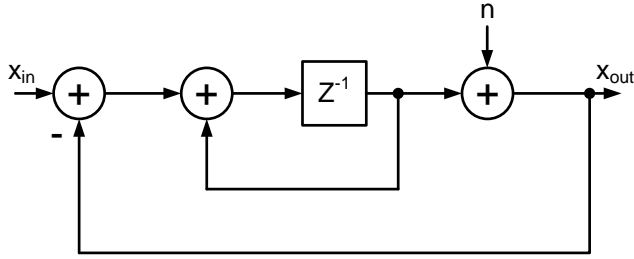


Figure 4.3: Linear model of the first-order delta-sigma modulator.

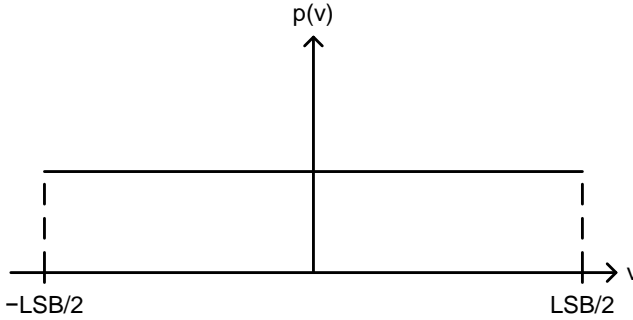


Figure 4.4: Probability density function of quantization noise, assuming the noise is uniformly distributed.

$$V_{n,rms}^2 = \int_{-\infty}^{\infty} v^2 p(v) dv = \frac{1}{LSB} \int_{-LSB/2}^{LSB/2} v^2 dv = \frac{LSB^2}{12}. \quad (4.7)$$

By taking the square root of both sides of Eq. 4.7 the rms voltage of the quantization noise is obtained as

$$V_{n,rms} = \frac{LSB}{\sqrt{12}}. \quad (4.8)$$

The next assumption about the quantization noise is that it can be considered white noise, and thus has a flat spectral density over the frequency band of interest [6]. This is illustrated in Fig. 4.5, where the height of the curve is a constant, K . This constant can be derived by noting that the total normalized power of the quantization noise is equal to $V_{n,rms}$ squared, and integrating the square of the spectral density over the whole frequency range should yield the same result [6]. So K is calculated as

$$\int_{-\infty}^{\infty} S^2(f) df = \int_{-f_s/2}^{f_s/2} K^2 df = K^2 f_s = \frac{LSB^2}{12} \Rightarrow K = \frac{LSB}{\sqrt{12}} \sqrt{\frac{1}{f_s}} \quad (4.9)$$

where f_s is the sample rate, a terminology originating from delta-sigma ADCs, which in the context of DACs means the rate at which codes are fed to the modulator. With this definition of K , the spectral density of the quantization noise is given as

$$S(f) = \begin{cases} \frac{LSB}{\sqrt{12}} \sqrt{\frac{1}{f_s}}, & \text{if } -\frac{f_s}{2} \leq f \leq \frac{f_s}{2}. \\ 0, & \text{otherwise.} \end{cases} \quad (4.10)$$

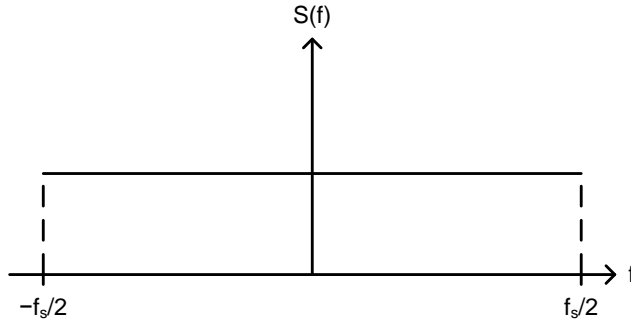


Figure 4.5: Spectral density of quantization noise, assuming the noise is white.

From the linear model of the modulator (Fig. 4.3) a transfer function from the signal input to the signal output can be derived. This is called the signal transfer function (STF) and defined as

$$STF(z) = \frac{X_{out}(z)}{X_{in}(z)} = z^{-1}. \quad (4.11)$$

In the same way a transfer function from the noise input to the signal output—called noise transfer function (NTF)—can be derived. It is defined as

$$NTF(z) = \frac{X_{out}(z)}{N(z)} = (1 - z^{-1}). \quad (4.12)$$

The signal transfer function seen in Eq. 4.11 is a delay and does not attenuate the signal. The magnitude of the noise transfer function in Eq. 4.12, however, has

sloping shape and attenuates the noise at low frequencies. A Bode plot of the magnitude of the NTF can be seen in Fig. 4.6, in which the curve has a shape very similar to the spectrum plotted in Fig. 4.2 (notice the 20 dB per decade slope).

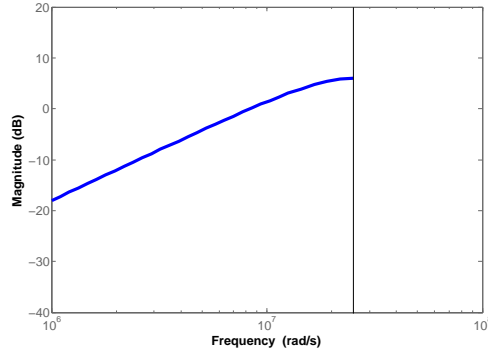


Figure 4.6: Magnitude plot of NTF of a first-order delta-sigma modulator.

The normalized power of the quantization noise that ends up within the frequency band of interest, can be calculated by evaluating the noise transfer function on the unit circle, i.e. letting $z = e^{j\omega t} = e^{j2\pi f/f_s}$. This gives

$$NTF(f) = 1 - e^{-j\frac{2\pi f}{f_s}} = 2je^{-j\frac{\pi f}{f_s}} \frac{e^{j\frac{\pi f}{f_s}} - e^{-j\frac{\pi f}{f_s}}}{2j} = 2je^{-j\frac{\pi f}{f_s}} \sin\left(\frac{\pi f}{f_s}\right). \quad (4.13)$$

For noise calculation it is the magnitude of the NTF that is of interest. Assuming that the sample frequency, f_s , is much higher than the frequency, f , small-angle approximation for the sine function can be used resulting in

$$|NTF(f)| = 2\sin\left(\frac{\pi f}{f_s}\right) \approx \frac{2\pi f}{f_s}. \quad (4.14)$$

Assuming a bandwidth of f_0 the normalized quantization noise power at the output of the modulator can be calculated as

$$P_n = \int_{-f_0}^{f_0} S^2(f) |NTF(f)|^2 df \quad (4.15)$$

where $S(f)$ is the spectral density of the noise as defined in Eq. 4.10, and NTF is the noise transfer function [6]. Inserting the expression for the absolute value of

the NTF from Eq. 4.14 and the definition of the spectral density in Eq. 4.10 into Eq. 4.15 yields

$$P_n = \int_{-f_0}^{f_0} \frac{LSB^2}{12} \frac{1}{f_s} \left(\frac{2\pi f}{f_s} \right)^2 df = \frac{LSB^2 \pi^2}{36} \left(\frac{2f_0}{f_s} \right)^3 = \frac{LSB^2 \pi^2}{36} \left(\frac{1}{OSR} \right)^3 \quad (4.16)$$

where OSR is the oversampling ratio defined as

$$OSR = \frac{f_s}{2f_0}. \quad (4.17)$$

The normalized power is equal to the rms voltage squared, so the expression for in-band noise power in Eq. 4.16 can be used together with the expression for the signal rms voltage in Eq. 4.5 to calculate SNR as defined in Eq. 4.1. Since the noise in question here is quantization noise, the calculated metric is called signal-to-quantization-noise ratio (SQNR).

$$SQNR = 10 \log \left(\left(\frac{LSB \cdot 2^N}{2\sqrt{2}} \right)^2 \middle/ \left(\frac{LSB^2 \pi^2}{36} \left(\frac{1}{OSR} \right)^3 \right) \right) \quad (4.18)$$

simplifies to

$$SQNR = 6.02N + 1.76 - 5.17 + 30 \log(OSR) \quad (4.19)$$

where N is the number of bits of the quantizer—1 in this case. The OSR, as defined in Eq. 4.17, is a measure of how many times the Nyquist rate the system is sampling at. If the desired bandwidth is f_0 —i.e. the highest possible signal frequency without getting aliasing is f_0 —and the sampling frequency, f_s , is equal to $2f_0$, then OSR is 1 (i.e. no oversampling). If f_s is equal to $4f_0$, then the OSR is 2, etc. Notice from Eq. 4.19 that the SQNR increases as the OSR increases. So by increasing the sampling rate the performance of the DAC gets better. Again, the terminology used here comes from a discussion of ADCs, and in the context of a DAC f_s is the rate at which the DAC is fed input codes and outputs a signal, and f_0 is the (highest possible) frequency of the generated signal. In the example in this chapter f_s is 8 MHz and f_0 is 20 kHz, so the OSR is 200 and the SQNR 72 dB.

When using Eq. 4.19 to estimate the performance it is important to notice that there are some assumptions that have been made when deriving the formula. Firstly it is assumed that quantization is the only, or at least dominant, source of noise and that the NTF of the modulator is perfect. When integrating the noise up to f_0 in Eq. 4.16 there is an underlying assumption that no higher frequency

noise ($f > f_0$) is included, and hence this noise has to be filtered out with a low-pass filter, as mentioned earlier. In practice such a filter is never perfect and it does not filter out all noise above f_0 .

4.2 Second-order delta-sigma modulator

To get a more effective noise shaping a higher order modulator can be used. For this project a second-order modulator, with the linear model shown in Fig. 4.7, is investigated. The noise transfer function of this modulator is

$$NTF(z) = (1 - z^{-1})^2 \quad (4.20)$$

and a Bode plot of the magnitude of this NTF can be seen in Fig. 4.8. Notice that the function curve has a slope of 40 dB per decade. Compare to the first-order modulator's NTF curve (Fig. 4.6) which has a slope of 20 dB per decade. Using the same logic as for the first-order modulator a formula for calculating the SQNR can be derived as [6]

$$SQNR = 6.02N + 1.76 - 12.9 + 50\log(OSR). \quad (4.21)$$

Notice here that the SQNR increases faster with OSR, than in the case of the first-order modulator. With the example: f_s of 8 MHz, and f_0 of 20 kHz, a second-order modulator (with a 1-bit quantizer) gives a SQNR of 110 dB.

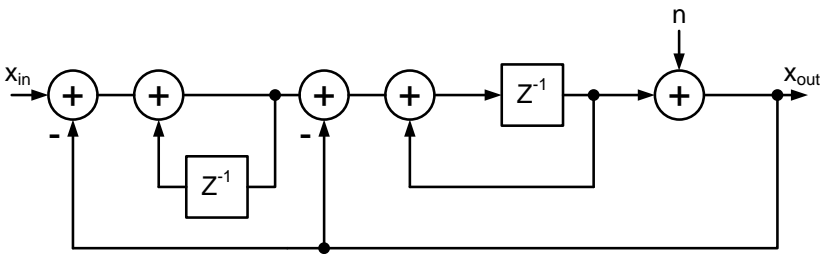


Figure 4.7: Linear model of a second-order delta-sigma modulator.

The drawback with higher order modulators is stability. Where for the first-order modulator a full-scale signal can be used, higher order modulators will become unstable for such large signals. And what is worse is that the stability of delta-sigma modulators is not well understood because the quantization makes them highly nonlinear circuits and therefore difficult to analyze [6]. As a rule of thumb a half-scale signal can be used with a second-order modulator.

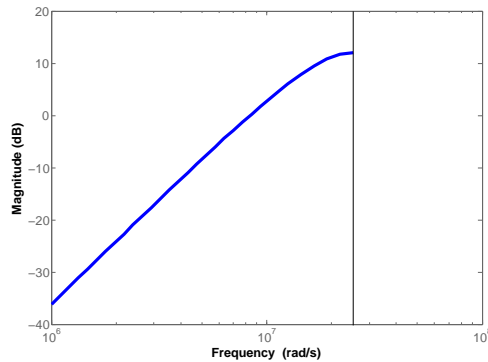


Figure 4.8: Magnitude plot of NTF of a second-order delta-sigma modulator.

4.3 Summary

The concept of delta-sigma modulation is to use oversampling and feedback loops to push quantization noise out of the frequency band of interest, so that it can be filtered out. A delta-sigma modulator can be used to both implement ADCs and DACs, and in this project it is attempted to use a delta-sigma DAC to generate a test signal for an ADC built-in self-test. An estimate of the performance of a delta-sigma DAC can be obtained by calculating the theoretical SQNR using Eqs. 4.19 and 4.21 for first-order and second-order modulators respectively. With the example: a sampling frequency, f_s , of 8 MHz and a signal frequency, f_0 , of 20 kHz, the SQNR is 72 dB for a first-order modulator and 110 dB for a second-order modulator. A higher order modulator gives a more effective noise shaping but have the disadvantage of instability for high amplitude signals.

5

System implementation and results

This thesis project consists of two main parts; investigating signal generation with a delta-sigma modulator realized in FPGA fabric, and mapping C code algorithms for ADC testing to FPGA fabric using HLS. A second-order delta-sigma modulator was investigated because it gives a sufficient SQNR with a reasonable oversampling ratio. Its implementation in Verilog is discussed below. Two BISTs for ADC testing were developed, one for dynamic test and one for static test, but those could be integrated into one piece if so desired. For development it was advantageous to keep them separate however. Not the entire BISTs were implemented in C code and synthesized with HLS, instead top-levels were written in Verilog and only the core test algorithms were developed in C.

5.1 The delta-sigma DAC

The delta-sigma modulator was implemented in Verilog and a block diagram of the design can be seen in Fig. 5.1. It consists of four adders, two flip-flops and a quantizer. The signals in the design are represented as 24-bit fixed point two's complement numbers so the buses in the design are 24 bits wide. The adders therefore add 24-bit words and the flip-flops in the diagram represent 24 parallel flip-flops each. Four of the bits are integer bits and 20 are fractional bits. It was found that four integer bits was enough to avoid overflow and 20 fractional bits gave sufficient accuracy. The quantizer is basically a comparator which feeds back +1 if the output is positive, and -1 if the output is negative¹. Through the output X_{out} one bit is fed out, and that is the inverted sign bit from the 24-bit representation. This way the DAC outputs 1, which corresponds to +VDD volts,

¹The value -1 is written as 1111.00000000000000000000 and the value +1 is written as 0001.00000000000000000000 with the 24-bit representation.

if the output of the modulator is positive and 0, which corresponds to 0 volts, if the output of the modulator is negative.

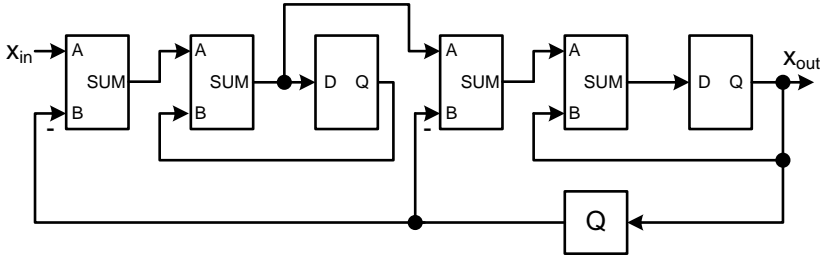


Figure 5.1: Block diagram of the second-order delta-sigma modulator.

The test setup used for the delta-sigma DAC can be seen in Fig. 5.2. A direct digital synthesizer (DDS) was used to generate a half-scale digital sine-wave of 20 kHz. The DDS is a pre-made design block which is available in Vivado's IP library and can be instantiated in the design. The sine-wave codes are fed to the delta-sigma modulator with the clock rate 8 MHz. The output of the modulator goes through an I/O, traces on the KC705 board and is then fed out to a spectrum analyzer using a GPIO SMA connector. The I/O blocks of the FPGA can be set in many modes (many I/O standards) but for this experiment the choice of standard was limited to the standard LVCMOS18. Before implementing the design two settings to the I/O could be adjusted: drive strength (4-24 mA) and slew rate (fast/slow). For this experiment 8 mA drive strength and fast slew rate were chosen.

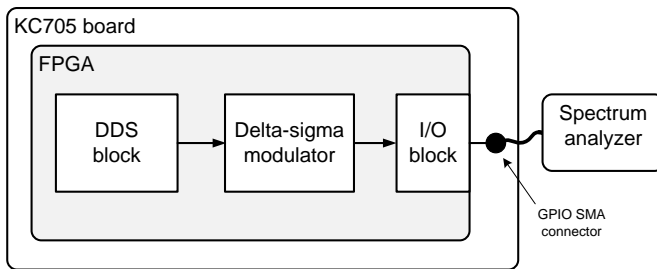


Figure 5.2: Setup for testing delta-sigma DAC. Digital sine-wave codes are generated by the direct digital synthesizer and fed to the modulator. The modulator outputs a single bit which is fed through the I/O of the FPGA to a spectrum analyzer. The I/O can be viewed as a one-bit DAC since it translates the digital bit into a voltage.

5.1.1 Measurement results

The measurement result of the experiment described above is illustrated in Fig. 5.3. To the left is the simulated ideal behavior of a second-order delta-sigma modu-

lator where the generated sine-wave can be seen as a tone at approximately 20 kHz. Notice that the noise has a slope of 40 dB per decade, which matches the NTF plot in Fig. 4.8. To the right is the spectrum measured with the spectrum analyzer.

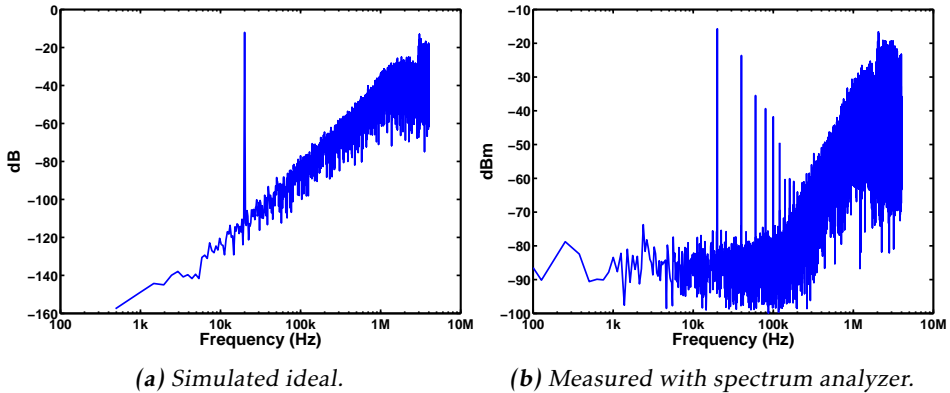


Figure 5.3: Spectra of the output signals of second-order delta-sigma modulators. In (a) the modulator is an ideal model simulated in Matlab, and in (b) the modulator runs on the FPGA, the signal is fed out through the I/O, and is measured with a spectrum analyzer.

As can be seen in the figure there are many harmonics in the measured spectrum. These are probably due to nonlinearities of the I/O, but are not a problem because they could be filtered out with a good analog filter. What is worse is that the in-band noise around the signal is much higher than in the ideal case. The noise shaping of the quantization noise seems to work properly because the 40 dB per decade noise slope significant of a second-order modulator is present in the spectrum, but the noise curve flattens out around 100 kHz and below. Assuming a brick-wall bandpass filter with a passband between 18 kHz and 22 kHz the measured SNR is 53 dB. This is too low to be able to test the test case ADC which is specified to have a SNR of 60 dB.

The digital behavior of the circuit was found to be correct by probing the output of the modulator running on the FPGA with an ILA. Therefore it can be concluded that the imperfections in the spectrum are related to the I/O and/or the board. Apart from the GPIO SMA connector output the user clock SMA outputs were also tested with no significant improvement. Two things would be desirable to test before deeming this approach unfeasible, and that is to try more I/O standards and to use two I/Os to make the DAC output fully differential. Unfortunately no suitable differential-to-single ended converter was available at the time of the experiment.

5.1.2 Alternative approach – filtered clock signal

For the sine-wave based static histogram test and for the dynamic test it is not necessary to have a very good DAC available. What is needed is to somehow generate a very pure sine-wave. An alternative approach to achieve this is to filter a clock signal, since if one of the frequency components of a square wave can be selected using a narrow bandpass filter a sine-wave is obtained. The setup for testing this idea is similar to the test setup for the delta-sigma DAC and is illustrated in Fig. 5.4. The clock signal is generated by a 200 MHz oscillator on the KC705 board and this signal is then used to generate a lower frequency clock signal in the FPGA by using a “clocking wizard” available in Vivado’s IP library. The clock signal is then divided down further to 20 kHz using a register divider before being fed out through the I/O to the spectrum analyzer in the same way as the delta-sigma output.

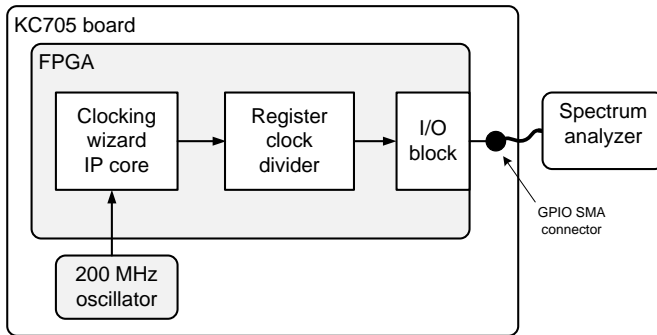


Figure 5.4: Test setup for sine-wave generation with clock signal. The clocking wizard IP core generates a clock signal based on a signal generated by a 200 MHz oscillator on the KC705 board. The clock signal is then divided down in the FPGA fabric to obtain a lower frequency clock signal. This signal is fed out through the I/O and is measured with a spectrum analyzer.

The result of this experiment can be seen in Fig. 5.5 which shows a number of tones. These are not harmonics due to nonlinearities in the I/O, but rather the frequency components of the square wave itself. There are also some spurs just around the tones of the square wave. Assuming a brick-wall bandpass filter with a passband between 18 kHz and 22 kHz the measured SNR is 64 dB. This is a more promising result than the delta-sigma DAC and if the noise spurs close to the signal can be removed this could be a feasible solution for testing the ADC. The way the clock signal is generated (divided down in the FPGA fabric) is rather crude and the result could probably be improved by using a more elaborate way of generating the clock signal.

The downside with this approach is that it is much less flexible than a delta-sigma DAC. There is no way of adjusting the amplitude of the sine-wave in the digital domain, instead the signal has to be attenuated with a resistive divider or similar before being presented at the input of the ADC. If this solution would be used

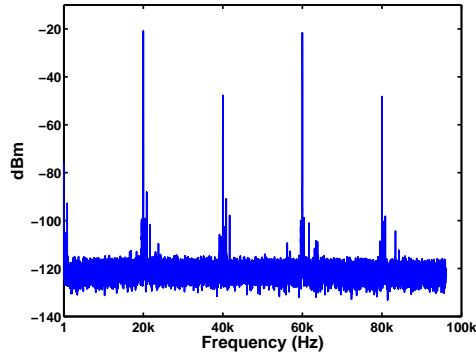


Figure 5.5: Spectrum of 20 kHz clock signal. Assuming a brick-wall band-pass filter with a passband between 18 kHz and 22 kHz its SNR is 64dB.

for a BIST containing both a static and a dynamic test, this is a problem since the two tests require different amplitudes of the sine-wave. For the static test the signal needs to overdrive the ADC, but for dynamic test it is very important not to saturate the ADC.

Since neither the delta-sigma DAC nor the filtered clock signal proved good enough to test the ADC without doing further investigations, an external signal generator was used to evaluate the BIST.

5.2 Top-level FSMs

As mentioned in the introduction the top-levels of the BISTs were implemented in Verilog as FSMs; one for the static test and one for the dynamic test. A flowchart of the static test can be seen in Fig. 5.6, and the boxes in the flowchart corresponds to states in the FSM.

The program is idle until the start button is pressed. This is one of the GPIO push buttons on the KC705 board. When the start button is pressed the block RAM that will hold the histogram is cleared by successively stepping through the memory elements and writing zeros to them. When the memory is cleared (4096 memory words for a 12 bit ADC, since it has 4096 code bins) a done signal is returned and the FSM steps to its next state.

After this follows four states which are used to read an ADC code. The signal generator and the ADC are running continuously and after each ADC conversion an end of conversion signal is set high. The ADC output data can be obtained from a register using the dynamic reconfiguration port (DRP). In order to do that the DRP is enabled and the program then waits until a data ready signal is returned from the DRP. When the data ready signal is returned the ADC code can be read from the register. The ADC is specified as having 12 bits of resolution but actually 16 bits are available. The four LSBs in the register are discarded by

the program and the 12 MSBs are used.

Each ADC code that is read from the DRP register is used as a pointer to the histogram memory, and the number held in the memory position that the ADC code points out is incremented by one. So if for example code 2037 is read from the ADC, the program points out address 2037 in the histogram memory and increments the number held in this memory position by one. This way the ADC codes are sorted into the histogram at the same pace the ADC samples the signal, instead of saving off hundreds of thousands of samples into a memory and then sort them into a histogram.

When enough samples have been collected for the histogram (the number is set in the Verilog code) the program starts the HLS block which contains the C code algorithms. This block calculates the gain error, offset error, DNL and INL and when it is finished a done signal is returned and the program goes back to idle.

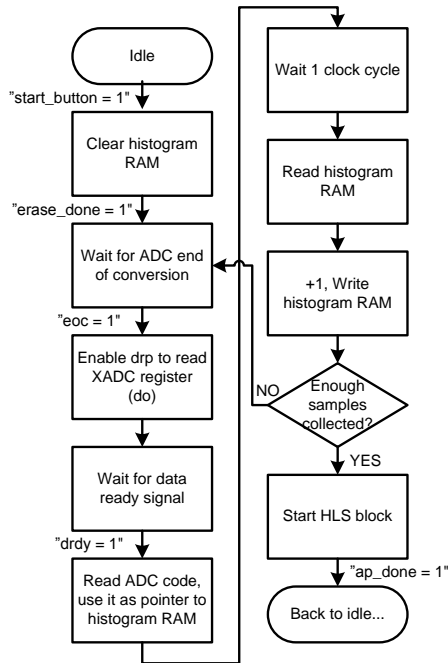


Figure 5.6: Flowchart of static test FSM.

The dynamic test functions much in the same way. A flowchart of it can be seen in Fig. 5.7. The program is idle until the start button is pressed and then it collects 16384 ADC samples reading from the DRP register in the same way as the static test does. As discussed previously, the time domain samples needs to be put into the FFT algorithm in bit-reversed order, so when the ADC codes are collected they are saved off into a memory in bit-reversed order. The C code contains functions to sort the samples into this order, but synthesizing this into an FPGA design

proved extremely inefficient, and therefore this work is instead done directly by the top-level FSM when collecting the samples.

When the 16384 samples have been collected the program starts the HLS block containing the C code algorithms that calculates the FFT and the dynamic performance parameters from the resulting spectrum. When the calculations are finished a done signal is returned and the program goes back to idle.

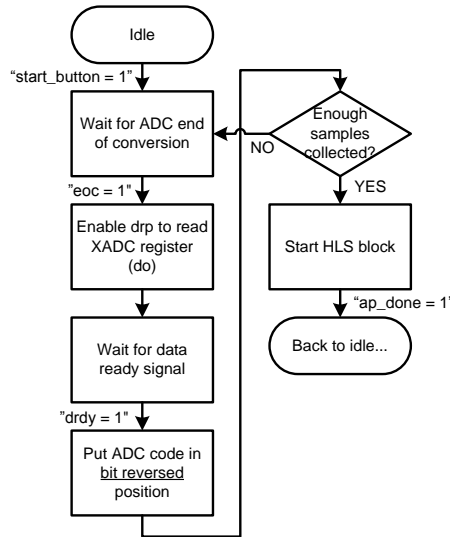


Figure 5.7: Flowchart of dynamic test FSM.

5.3 Overview of C code algorithms

In this section an overview of how the C code algorithms are implemented is presented. The algorithms are divided into static test and dynamic test, and each test consists of a top-level function and number of sub-functions.

5.3.1 Static test

The C code for the static test consists of a top-level function that calls three sub-functions which each contains a loop. In the first sub-function (Fig. 5.8) the program steps through the bins of the histogram, sums them up into the cumulative histogram and calculates the transition levels using Eq. 3.23.

In the second sub-function (Fig. 5.9) the program calculates the gain and offset. It steps through all the transition levels and calculates the three sums needed to for calculating gain and offset using Eq. 3.7 and 3.8.

In the third sub-function (Fig. 5.10) the program once again steps through the transition levels and calculate the DNL and INL using Eq. 3.9 and 3.10.

```

for (i=1; i < histogram_size; i++)
{
    Calculate transition level using Eq. 3.23
    CH = CH + H[i]
}

```

Figure 5.8: Pseudo code for the first static test sub-function.

```

for (i=1; i < num_trans_lvl; i++)
{
    sumx = sumx + T[i]
    sumxx = sumxx + T[i] * T[i]
    sumxy = sumxy + i * T[i]
}
Calculate gain using Eq. 3.7
Calculate offset using Eq. 3.8

```

Figure 5.9: Pseudo code for the second static test sub-function.

It is important to note that if there is an inaccuracy in the amplitude and offset of the input sine-wave, the algorithms cannot detect a difference between the errors in the input signal and errors in the ADC itself. An offset in the input will be perceived as an offset of the ADC, and an amplitude error in the input will be perceived as both a gain and an offset error in the ADC, because the gain goes into the offset calculation of Eq. 3.8. Since gain and offset are compensated for in the DNL and INL calculations (Eq.3.9 and 3.10) an error in the input amplitude and offset will not considerably affect the INL calculation [2] or the DNL calculation.

5.3.2 Dynamic test

The dynamic test consists of a top-level function, an FFT function and sub-functions to calculate THD, SFDR, SINAD, SNR and ENOB respectively. These sub-functions are not described in detail here, but they essentially sum up content in the spectrum frequency bins in order to calculate the performance metrics according to their definitions. They are all calculated with respect to the signal amplitude and not full-scale.

To implement the FFT algorithm a representation for complex numbers is needed. Vivado HLS does support the C99 standard which includes complex numbers, but in practice it proved difficult to get the compiler to accept these. The complex numbers are instead represented with the struct shown in Fig. 5.11.

The implemented FFT is a radix-2, decimation-in-time, in-place algorithm which is best illustrated by the flow graph in Fig. 3.13. It consists of an outer loop which steps through the $\log_2(M)$ stages of the FFT and an inner loop which steps

```

for (i=1; i < num_trans_lvl; i++)
{
    W[i] = T[i+1] - T[i] // Code bin width
    Calculate DNL using Eq. 3.9
    Calculate INL using Eq. 3.10
    Check if these DNL & INL values
    are max/min
}

```

Figure 5.10: Pseudo code for third static test sub-function.

```

struct complex_num
{
    double real, imag;
}

```

Figure 5.11: Struct holding complex numbers.

through the samples in each stage and calls a “butterfly” function. Notice that the samples in each stage in Fig. 3.13 can be calculated two-and-two from the adjacent samples in the previous stage, using the flow graph shown in Fig. 5.12. Because of its shape this flow graph is often referred to as a butterfly flow graph.

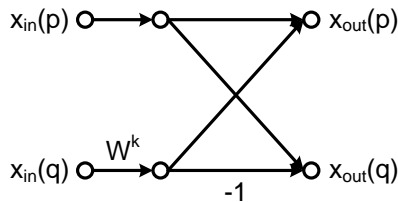


Figure 5.12: Butterfly flowgraph. This is the fundamental building block in the radix-2 FFT algorithm used in this project.

Because adjacent samples from the previous stage are used in each stage, the same memory elements that holds the input samples in Fig. 5.12 can be used to store the result of the calculation. This is the reason the algorithm is called “in-place” [12]. Pseudo code for this FFT algorithm is presented in Fig. 5.13. For each stage the algorithm goes through the samples, and for each step it calculates indices p and q as well as the exponent of the twiddle factor used in the calculation illustrated by Fig. 5.12. It then calls a function which carries out this calculation.

Each butterfly calculation requires two complex additions/subtractions and one

```

struct complex_num
for (stage=0; stage < log2(M); stage++)
{
    for (samples in stage)
    {
        Calculate index p
        Calculate index q
        Calculate exponent of twiddle factor
        butterfly(p,q,exp)
    }
}

```

Figure 5.13: Pseudo code for the FFT algorithm.

complex multiplication, and each complex multiplication requires four real multiplications and two real additions/subtractions as seen in Eq. 5.1.

$$(a + jb)(c + jd) = (ac - bd) + j(ad + bc) \quad (5.1)$$

By decreasing the number of (complex) multiplications needed to calculate the DFT, FFT algorithms can dramatically reduce the resources and/or computation time needed to obtain a frequency spectrum.

5.4 BIST measurement results

In order to evaluate if synthesizing C code algorithms for ADC testing with HLS is feasible, the test results between code running on a PC and the synthesized design were compared. Except for matching results between PC and BIST, measurement time and FPGA utilization was also considered.

5.4.1 Comparison between models and measurement

In Table 5.1 a comparison between the BIST, C code running on a PC and the Matlab model for the static test can be seen. The ADC was set in unipolar mode and an external signal generator was used to generate the input signal. The signal was a single-ended sine-wave with an offset of 500 mV and an amplitude of 550 mV, and the frequency was set to around 20 kHz but non-harmonically related to the sampling frequency. The frequency was somewhat arbitrarily chosen. A lower frequency might be desirable, but the purpose with this experiment was to evaluate the functionality of the BIST rather than evaluate the ADC itself. For that purpose any reasonable frequency chosen according to the rules in chapter 3

is acceptable, since the static test algorithms are independent of frequency ². In the test 200,000 samples were collected and used to generate the histogram. In order to compare the BIST with the C code and Matlab code running on a PC, the histogram was probed with an ILA. Using Vivado's hardware manager the data from the ILA can be obtained in a CSV-file. This way the histogram from the BIST test could be used as input data in Matlab and a C code testbench. Ideally the ADC codes would be probed directly and the histogram would be regenerated with code on the PC, but the ILAs cannot save enough samples. In this comparison, using 10 decimal places, the results match exactly and it can be concluded that synthesizing the algorithms into RTL and running them on the FPGA was successful. Comparing the result to the specifications of the ADC (see Table 2.1) it can be seen that the test results are reasonable. All results meet specifications except for the offset error, which is probably due to that the accuracy of the offset setting of the signal generator was not good enough.

Table 5.1: Comparison between static BIST test, C code running on PC and Matlab test model using probed histogram. Since there is a very good match, mapping the algorithms to the FPGA was successful.

	Matlab model using measured histogram	C code (on PC) using measured histogram	C code algorithms running on FPGA
gain error (%)	-0.1814590398	-0.1814590398	-0.1814590398
offset error (LSB)	-11.4587839949	-11.4587839949	-11.4587839949
DNL min (LSB)	-0.8265005955	-0.8265005955	-0.8265005955
DNL max (LSB)	0.6550577276	0.6550577276	0.6550577276
INL min (LSB)	-2.1075215934	-2.1075215934	-2.1075215934
INL max (LSB)	2.5614670437	2.5614670437	2.5614670437

Fig. 5.14 shows DNL and INL plots from the measurement. The INL has jumps at certain codes where the DNL is large. Note that the INL curve does not start off from zero because the best-fit approach was used and not the terminal based approach.

To illustrate that an error in the input amplitude does not affect the INL and DNL calculations, a test was run using an amplitude of 700 mV with the algorithm assuming an amplitude of 550 mV. As can be seen in Table 5.2 this error in input amplitude is perceived by the algorithm as both a gain error and an offset error, but the DNL and INL have not changed significantly. Because the input signal was not filtered and because only 200,000 samples were used to form the histogram, the DNL and INL values do change a little bit from test to test. The point here is that if DNL and INL were sensitive to an error in the input amplitude, they would deviate significantly in this test.

Looking at the DNL and INL graphs from this test (Fig. 5.15) it can be seen that

²The actual measured errors do change with frequency, since the performance of the ADC under test typically varies with frequency, but the algorithms themselves do not use frequency as an input parameter.

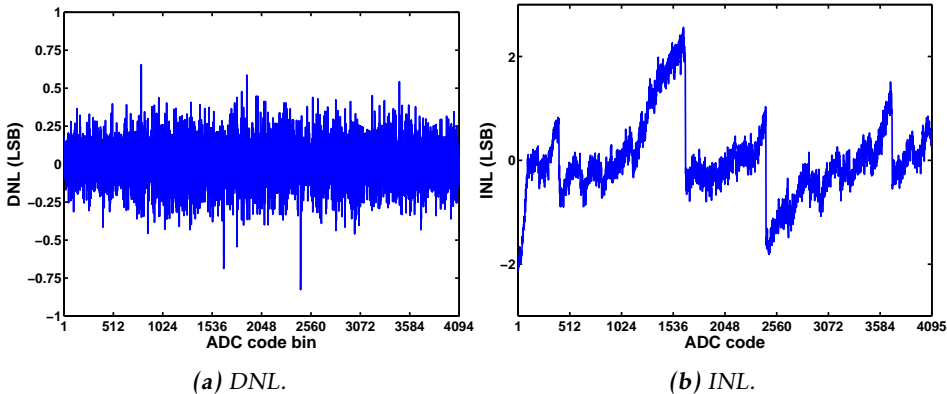


Figure 5.14: Static test results illustrated with data probed with ILAs.

Table 5.2: Static measurement results using 700 mV input amplitude, when algorithm assumes 550 mV.

	C code algorithms running on FPGA
gain error (%)	27.0
offset error (LSB)	-569
DNL min (LSB)	-0.64
DNL max (LSB)	0.71
INL min (LSB)	-2.27
INL max (LSB)	2.78

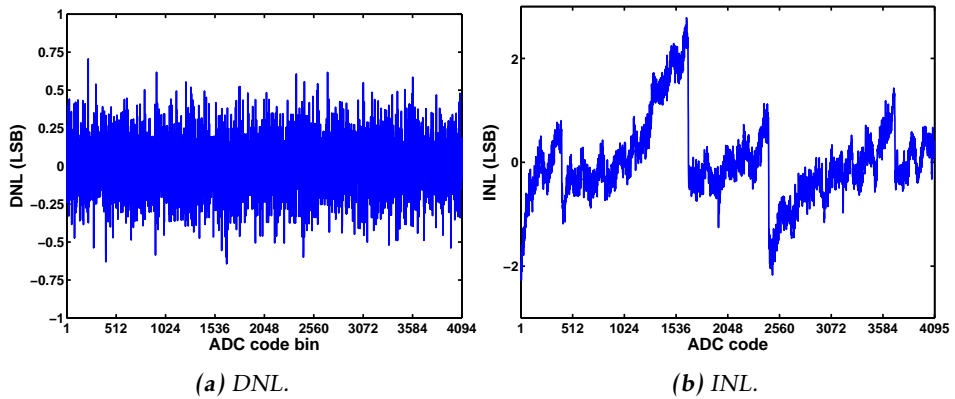


Figure 5.15: DNL and INL using 700 mV input amplitude, when algorithm assumes 550 mV.

the DNL curve is a bit more “noisy” than in the previous test, but the max/min DNL is similar. The INL curve is very similar to the previous test.

To illustrate the importance of choosing the input frequency as non-harmonically related to the sampling frequency, a test was done using an input frequency of exactly 20 kHz. The results are given in Table 5.3. Interestingly the DNL min/max values are reasonable, as well as the offset error, but the INL min/max values indicates that something is wrong. Looking at the DNL and INL graphs in Fig. 5.16 the problem becomes evident. Because of the relation between the input signal frequency and the sampling frequency some code bins get much more hits, and some code bins get much less hits, than expected. Noise spreads out the hits preventing huge DNL errors, but in the INL curve the effect is very clear.

Table 5.3: Static measurement results using an input signal frequency (20 kHz) which is harmonically related to the sample frequency (1 MHz).

	C code algorithms running on FPGA
gain error (%)	-0.27
offset error (LSB)	-11.7
DNL min (LSB)	-0.61
DNL max (LSB)	0.67
INL min (LSB)	-6.59
INL max (LSB)	7.73

In Table 5.4 measurement results from the dynamic BIST are compared with the C code algorithms running on a PC and a Matlab model. In this case the ADC codes were probed directly with an ILA, a CSV-file with the data was obtained and this data was used in a testbench on the PC. Unfortunately the last ADC

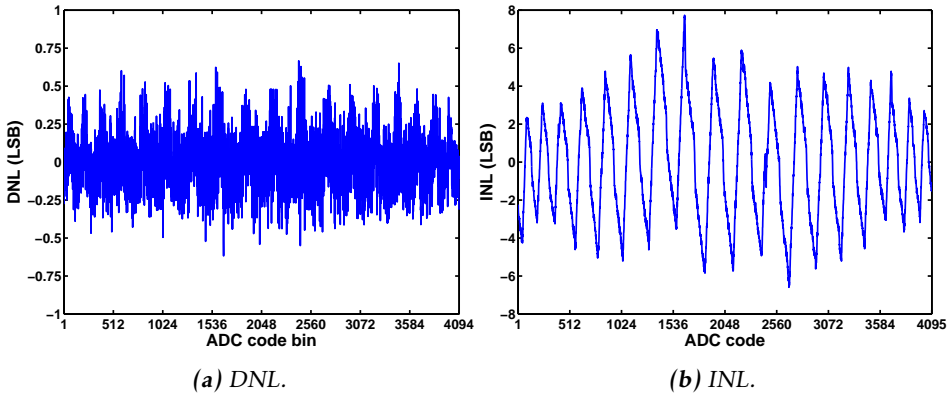


Figure 5.16: DNL and INL using an input signal frequency (20 kHz) which is harmonically related to the sample frequency (1 MHz).

sample was not probed correctly by the ILA so to be able to make the comparison, including the FFT, the code of the last sample was estimated by interpolating the curve. Despite this, the results match to two decimal places, except for the THD which matches to one decimal place. This is close enough to deem the experiment successful, and based on the experience from the static test, the match is probably even better in reality. Some difference is expected though, because the design uses floating point operations and the floating point operators used in the FPGA design are not 100% accurate [18].

Table 5.4: Comparison between dynamic BIST test, C code running on PC and Matlab test model using probed ADC codes and probed spectrum.

	Matlab model using measured ADC codes	Matlab model using measured spectrum	C code algorithms running on PC	C code algorithms running on FPGA
THD (dB)	-66.0306269531	-66.0210167941	-66.0306269531	-66.0210167941
SFDR (dB)	47.5210792495	47.5191127127	47.5210792495	47.5191127126
SINAD (dB)	42.3139953465	42.3132329784	42.3139953465	42.3132329784
SNR (dB)	42.3324899726	42.3317654022	42.3324899726	42.3317654022
ENOB	6.7365440775	6.7364174383	6.7365440775	6.7364174383

Table 5.4 does show a good match between the algorithms running on a PC and on the FPGA, but the measurement results are much lower than the specification of the ADC (Table 2.1). One reason for that is that the input signal from the signal generator was not filtered and is therefore not as spectrally pure as desired. Also, and more importantly, there is significant spectral leakage that affects the result. The reason behind the spectral leakage is probably that the frequency of the input signal could not be set with sufficient precision. This highlights a

potential difficulty of dynamic ADC testing in a BIST environment. It can be hard to control the input signal frequency accurately enough.

In Fig. 5.17 the spectrum obtained by an ILA probe during the test run can be seen. The DFT is calculated directly from the ADC codes and the amplitudes in the resulting frequency spectrum are not normalized. Since the performance metrics are calculated as ratios of the signal and distortions/noise in the spectrum, no normalization is necessary.

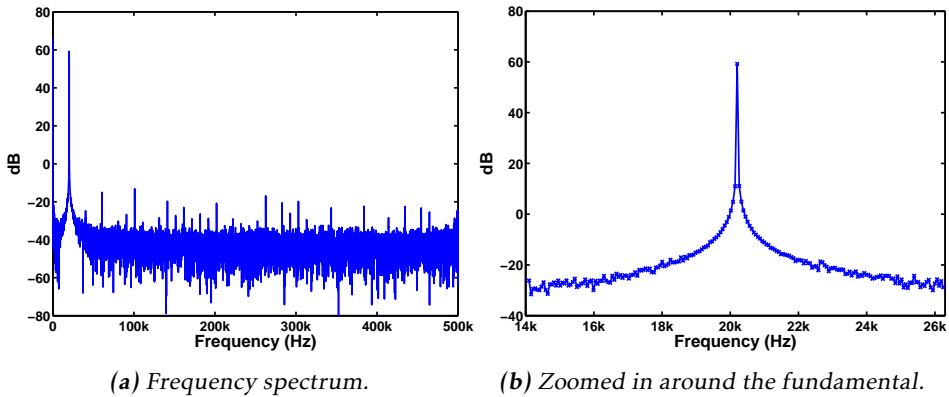


Figure 5.17: Frequency spectrum calculated by FFT algorithm and probed with ILA. Notice the spectral leakage due to inaccuracy in the frequency of the test signal.

Figure (b) shows a zoomed in view around the input signal frequency, where the crosses on the curve indicates the data points (frequency bins). Because of the spectral leakage, the two bins just beside the signal bin are perceived by the algorithms as the highest spurs in the spectrum and thus limits the calculated SFDR. These bins do in reality contain energy from the fundamental signal and are not due to distortion or noise. All the frequency bins outside the signal bin are summed up in the noise calculation, so the result is not only affected by the fact that the signal frequency bin contains less energy, but also because signal energy is leaked into the frequency bins the algorithms regard as noise. This situation of course has to be avoided, and that can be done by either setting the signal frequency with better precision or by using a windowing function and modifying the algorithms to use more than one frequency bin to calculate signal power.

In summary it can be concluded that the dynamic test algorithms work as expected and that there is a good match between the BIST and the algorithms running on a PC, but that the result in this particular test does not reflect the true performance of the ADC because of inaccuracies in the input signal.

5.4.2 Utilization

The FPGA resource requirements of the BISTs are shown in Table 5.5. The Kintex-7 FPGA used is a quite large device and the designs use a significant portion of its resources despite the fact that rather simple algorithms have been implemented. The issue here is that the data type “double” is used in the C code, which corresponds to double precision floating point numbers. All operations in floating point are mapped to floating point operators by the high-level synthesis tool and these floating point operators are very expensive.

The double precision floating point numbers also requires much memory because they are represented by 64-bit words. The FFT algorithm uses lookup tables for sin and cos functions which in total uses 16384 64-bit words. These are saved in ROM using the static and const keywords in the C code.

Table 5.5: Resource utilization for dynamic and static test including ILAs. Notice that a relatively large portion of the resources are used—the usage in percent is shown in brackets—despite the fact that these are relatively small designs. Much of the memory usage is due to that ILAs are used to probe the results.

	Available (Kintex-7*)	Dynamic test	Static test
Slice LUTs	203800	21324 (10%)	33607 (16%)
Slice registers	407600	21556 (5%)	27065 (7%)
Memory	445	202 (45%)**	66 (15%)
DSP	840	217 (26%)	71 (8%)
I/O	582	9 (2%)	9 (2%)
Clocking	32	2 (6%)	2 (6%)

* XC7K325T-2FFG900C

** 13% ILA

Another reason for that much memory is used is that there are ILAs in the design. The ILAs samples registers when certain trigger and capture conditions are met, and the samples are placed into a memory. So in many cases in these designs there is a memory containing some information, and during the program run an ILA samples the same information and places it into its own memory. This way there are two copies of the same information in memory, which of course, in theory, is unnecessary. This could be avoided by for example reading the information directly from the memories of the design and send it via an UART interface to a PC.

5.4.3 Measurement time

Table 5.6 shows an estimate of the BIST test time based on the latency of the designs, the system clock frequency, the number of samples taken and the ADC sample frequency. The ADC has a sample frequency of 1 MHz, so collecting e.g. 200,000 samples takes 200 ms. The latency of the static test (the HLS block) is

about three million clock cycles and a 104 MHz system clock is used³. So the test time, excluding the time it takes to collect the samples, is about 30 ms.

Table 5.6: Estimate of measurement time based on latency of HLS blocks, system clock frequency, number of samples collected and ADC sample frequency.

	Estimated measurement time (ms)
Static test	230 (200 collecting)
Dynamic test	76 (16 collecting)

The exact measurement time is not that interesting, it is the order of magnitude of the test time that is important. In this case the test time of the static test is limited by the time it takes to collect the samples, and for good precision in the test even more samples are needed. With a faster ADC this situation may change, and the calculation time could exceed the time it takes to collect the samples. For a very fast ADC decimation of the sampled data might be necessary if the system is not fast enough to collect the samples.

For the dynamic test a significant portion of the test time is used to collect the 16384 samples that are needed for the FFT, but the bulk time is the calculation time. The test time could be decreased significantly by improving the FFT implementation. In this implementation just one butterfly “operator” was used because the double precision floating point multiplication operators are very expensive. Using fixed point representation, more operators could be used and up to $N/2$ butterfly calculations could be done in parallel in each stage of the FFT algorithm.

5.5 Summary

In this project a second-order delta-sigma modulator realized in FPGA fabric was evaluated. The implementation in Verilog was directly mapped from the block diagram of the design, using adders, flip-flops and quantizing the signal by looking at the sign bit. Numbers are represented internally as 24-bit fixed-point two’s complement numbers, with four integer bits and 20 fractional bits. The digital behavior of the modulator was correct, but the performance—tested in a Kintex-7 device on a KC705 board and measured with a spectrum analyzer—of the DAC proved poor. There are several harmonics in the spectrum and the measured SNR assuming a brick-wall filter was 53 dB. The non-ideal behavior of the DAC is probably due to nonlinearities in the I/O and/or board. The possibility of using a filtered clock signal to generate a sine-wave was also investigated. Assuming a brick-wall filter a SNR of 64 dB was measured. The idea was to use the delta-sigma DAC (or filtered clock signal) to generate a test signal for an ADC built-in

³The design was intended to run at 100 MHz but the ADC needs a 26 MHz clock, and this clock is obtained by dividing the system clock. 100 MHz could not be divided down to 26 MHz correctly, so a 104 MHz system clock was used instead.

self-test. Without further investigation none of these method could be used to generate a pure enough signal for this purpose, and instead an external signal generator was used to test the BIST.

The BIST was actually implemented as two different BISTs—one for a static test and one for a dynamic test—but they could be integrated into one BIST if desired. They both consist of top-level FSMs written in Verilog, and core test algorithms written in C and synthesized using HLS. Mapping the algorithms to the FPGA fabric using HLS proved successful, and there was a good match between the code running on a PC and the BISTs. The quality of the input signal proved very important. For the static test it was noted that an error in amplitude of the test signal would cause an error in gain and offset calculations, but not in DNL and INL calculations. In the dynamic test it proved difficult to set the test signal frequency accurately enough to avoid spectral leakage.

6

Conclusions and future work

The concept of having a common base of C code for running on a PC and for synthesizing with HLS into a BIST running on an FPGA proved feasible. When testing ADCs a number of choices have to be made, especially for the static test. There are multiple ways to define the performance metrics of interest, there are many ways to measure them and the choice of definition and method will affect the result. The IEEE Std 1241 can be used as guidance when making these choices. Definitions and measurement methods are usually not expressed explicitly on data sheets which makes it harder to compare the performance of ADCs.

Much information was gained about the challenges that have to be met in order to build a reliable system based on the BIST concept. On the digital side floating point representation should be replaced with a fixed point representation. This is to reduce the resource requirements of the system and to be able to get a completely exact match between the algorithms running on a PC and the algorithms synthesized into an FPGA design. Rewriting the code to use fixed point representation is not a trivial task and a very good understanding of the algorithms is needed to avoid issues with overflow and inaccuracy in the calculations. In order to avoid the unnecessary memory usage of the ILAs, sending the results via UART to the PC is probably a good idea.

For the static test, it has to be considered that the accuracy of gain and offset calculations depends on the accuracy of the amplitude of the sine-wave input signal. If very accurate measurements of gain and offset are needed, either the amplitude must be controlled very precisely, or some other measurement method must be used. Perhaps the feedback loop approach can be used to find two of the transition levels and then calculate the amplitude and offset of the input using this information.

For the dynamic test, the precision of the signal frequency setting must be considered carefully. Otherwise windowing or some method to avoid spectral leakage must be used.

Regarding the stimulus, the delta-sigma DAC did not work out in this particular test case, but further investigations should be done before discarding the idea. Other I/O standards and differential output should be tested. In the longer term it would be good to develop practical methods for testing ADCs using imprecise stimulus and digital post-processing. One of the most fundamental difficulties with ADC tests is to ensure the input signal quality, and moving this problem from the analog domain to the digital domain would make a lot of sense, especially for an FPGA based BIST.

Lastly, depending on the intended applications, it could be considered to use the BIST to test more aspects of the ADC. For example step response, code noise and two-tone tests.

Bibliography

- [1] URL www.xilinx.com/about/company-overview/index.htm. Cited on page 2.
- [2] IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters. *IEEE Standard 1241-2010*, 2011. Cited on pages 3, 15, 17, 19, 26, 27, and 52.
- [3] J. Blair. Histogram Measurement of ADC Nonlinearities Using Sine Waves. *IEEE Transactions on Instrumentation and Measurement*, 43(3):373–383, June 1994. Cited on pages 4, 15, 27, and 28.
- [4] M. Vanden Bossche, J. Schoukens, and J. Renneboog. Dynamic Testing and Diagnostics of A/D Converters. *IEEE Transactions on Circuits and Systems*, CAS-33(8):775–785, August 1986. Cited on pages 3 and 26.
- [5] M. Burns and G. W. Roberts. *An Introduction to Mixed-Signal IC Test and Measurement*. Oxford University Press, Inc., New York, 2001. Cited on pages 16 and 17.
- [6] T. C. Carusone, D. Johns, and K. Martin. *Analog Integrated Circuit Design*. John Wiley & Sons, Inc., Singapore, 2 edition, 2013. Cited on pages 37, 38, 40, and 42.
- [7] J. Doernberg, H-S Lee, and D. A. Hodges. Full-Speed Testing of A/D Converters. *IEEE Journal of Solid-State Circuits*, SC-19(6):820–827, December 1984. Cited on pages 3 and 23.
- [8] W. Kester. Section 5 - Fast Fourier Transforms. URL www.analog.com/static/imported-files/seminars_webcasts/MixedSignal_Sect5.pdf. Cited on page 29.
- [9] W. Kester. *Taking the Mystery out of the Infamous Formula, "SNR = 6.02N + 1.76dB," and Why You Should Care*. Analog Devices, www.analog.com/static/imported-files/tutorials/MT-001.pdf, 2009. Cited on page 28.

- [10] W. Kester. *Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor*. Analog Devices, www.analog.com/static/imported-files/tutorials/MT-003.pdf, 2009. Cited on page 32.
- [11] J. Logue. *Virtex Synthesizable Delta-Sigma DAC, XAPP154*. Xilinx, www.xilinx.com/support/documentation/application_notes/xapp154.pdf, September 1999. Cited on page 4.
- [12] A. V. Oppenheim, R. W. Schaffer, and J. R. Buck. *Discrete-Time Signal Processing*. Prentice Hall, New Jersey, 2 edition, 1998. Cited on pages 29, 32, and 53.
- [13] K. Parthasarathy, T. Kuyel, D. Price, L. Jin, D. Chen, and R. Geiger. BIST and Production Testing of ADCs Using Imprecise Stimulus. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):522–545, October 2003. Cited on page 4.
- [14] K. Shin and J. Hammond. *Fundamentals of Signal Processing for Sound and Vibration Engineers*. Wiley, 1 edition, 2008. Cited on page 24.
- [15] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye. *Probability & Statistics for Engineers & Scientists*. Pearson, Boston, 9 edition, 2011. Cited on page 18.
- [16] N. H. E. Weste and D. M. Harris. *Integrated Circuit Design*. Pearson, Boston, 4 edition, 2011. Cited on page 7.
- [17] *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter 7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter – User Guide (UG480)*. Xilinx, October 2012. Cited on page 8.
- [18] *Vivado Design Suite User Guide – High-Level Synthesis (UG902)*. Xilinx, July 2012. Cited on pages 5 and 58.
- [19] *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS182)*. Xilinx, October 2013. Cited on page 9.