

TALLINN UNIVERSITY OF TECHNOLOGY  
Faculty of Information Technology  
Thomas Johann Seebeck Department of Electronics



TALLINNA TEHNIKAÜLIKOOL  
TALLINN UNIVERSITY OF TECHNOLOGY

IEE70LT

Sekyanzi Badru IVEM 132062

# **Evaluation of the Internal ADC of Piccolo and Tiva Microcontrollers**

Master`s

Supervisor: Olev Märtens

PhD

Lead Researcher

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL  
Infotehnoloogia teaduskond  
Thomas Johann Seebecki Elektroonikainstituut



TALLINNA TEHNIKAÜLIKOOL  
TALLINN UNIVERSITY OF TECHNOLOGY

IEE70LT

Sekyanzi Badru IVEM 132062

# **Piccolo ja Tiva mikrokontrollerite sisemise ADM evalveerimine**

Magistritöö

Juhendaja: Olev Märtens

PhD

Lead Researcher

Tallinn 2016

## **Author's declaration of originality**

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Badru Sekyanzi

22.05.2016

## **Abstract**

The piccolo control stick (TMS320F28069) and Tiva C series (TM4C123GHM6PM) are microcontrollers produced by Texas Instruments. These microcontrollers can be used in a number of applications. The technology behind the success of these two micro controller boards is low cost and high precision from the analog digital converter. Analog to Digital Converter (ADC) is the world's largest volume mixed-signal circuit. It is also a key building block in nearly all system on chip (SoC) solutions involving analog and mixed-signal functionalities. The purpose of this project was to test the ADC used in these two boards. In this work the static and dynamic performance of the ADC is analysed with comparison to what is written in the data sheet. Static performance test is done by analysing the ADC data logs of the collected 32 ADC digital output samples and 2000 analog input samples within a rail range of (0-3.3) V. The dynamic performance is tested using a sine wave input to the ADC at different frequencies and analysing the ADC digital output code. Simple scripts of code run in MATLAB and R-language are used to analyse gain error, offset error, INL, DNL, SNR, SINAD, ENOB and THD. The goal is to help engineers that are developing designs using the piccolo control stick (TMS320F28069) and Tiva C series (TM4C123GHM6PM) to make accurate decisions regarding ADC selection before producing first prototypes.

This thesis is written in English and is 57 pages long, including 4 chapters, 50 figures and 20 tables.

# **Annotatsioon**

## **Piccolo ja Tiva mikrokontrollerite sisemise ADM evalveerimine**

Piccolo controlSTICK (TMS320F28069) ja Tiva C seeria (TM4C123GHM6PM) on mikrokontrollerid, mille tootjaks on Texas Instruments. Kõige olulisem asi, mida mikrokontrollerite valimisel silmas pidada, on analoog-digitaalmuundurite (ADC-de) jõudlus. Analoog-digitaalmuundur on elektrooniline seade, mis muudab tegeliku maailma analoogsignaali masinloetavaks või binaarseks või digitaalseks formaadiks. Kõik mikrokontrollerid vajavad seda komponenti, mis saab muuta tegeliku maailma analoogsignaali pingesignaalideks. See pingeline sisse ADC-sse ja tulemuseks on kahendnumbrid, mida saab sõltuvalt vajadusest edasi töödelda. Analoog-digitaalmuundur (ADC) on maailma suurim segasignaalkõitega volulering. See on ka peamine element peaaegu kõikide selliste kiibisüsteemide (SoC) lahenduste puhul, mis hõlmavad analoog- ja segasignaali funktsioone. Mikrokontrollereid saab kasutada mitmetel juhtudel. Nende kahe mikrokontrolleri plaadi tehnoloogilise edu taga on madal kulu ja kõrge täpsus analoog-digitaalmuunduri abil. Projekti eesmärgiks oli testida nendes kahes plaadis kasutatud ADC-de jõudlust. On väga oluline saada põhjalikult aru, kuidas ADC toimib enne disaini loomist. ADC jõudluse analüüs avaldab peensused, mis viivad sageli soovitud väiksema jõudluseni. Analoog-digitaalmuunduri täpsusel on mõju üldisele süsteemi tõhususele. Selleks et täpsust parandada, peab aru saama vigadest, mis on seotud ADC-ga, ja neid mõjutavatest parameetritest. Käesolevas töös on ADC staatilist ja dünaamilist jõudlust analüüsitud võrdlusena sellega, mis on kirjas andmelehel. Staatiline jõudlustest viiakse läbi, analüüsides kogutud 32 digitaalväljundi diskreedi ja 2000 analoogdiskreedi ADC andmelogisid vahemikus 0-3,3 V. Staatiline jõudlustest keskendub sisemiste ADC-de, ilma mingisuguse välise väljundita, testimisele. Dünaamilist jõudlust testitakse kasutades erinevatel sagedustel siinuslaine sisendit, mis on ühendatud ADC-ga. Lihtsaid koodiskripte, mis töötavad MATLAB-is ja R-keeles, kasutatakse võimendusvea, eelpingevea, SNR-i, SINAD-i ja THD analüüsiks. Selle projekti eesmärgiks on aidata inseneridel, kes kasutavad disainide loomiseks Piccolo controlSTICKi (TMS320F28069) ja Tiva C seeriat (TM4C123GHM6PM), teha täpseid otsuseid seoses ADC valikuga enne esimeste prototüüpide tootmist.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 57 leheküljel, 4 peatükki, 50 joonist, 20 tabelit.

## **Acknowledgements**

I would like to offer special thanks to Olev Märtens, my supervisor for giving me this opportunity to write a thesis in this interesting field and for his knowledge and technical support during the project. I also would like to thank my family for the valuable support.

## List of abbreviations and terms

ACQPS	ADC Acquisition (Sample and Hold) Window
ADC	Analog-to-Digital Converter
ADC <sub>IN</sub>	Analog-to-Digital Converter Input
ADCSAC	Analog-to-Digital Converter Sample Averaging Control
C <sub>ADC</sub>	Input Capacitance Analog-to-Digital Converter
CPU	Central Processing Unit
DMA	Dynamic Memory Access
DNL	Differential Nonlinearity
DUT	Device Under Test
EMI	Electromagnetic Interference
EPWM	Enhanced Pulse Width Modulation
GPIO	General Purpose Input Output
INL	Integral Nonlinearity
KHz	Kilohertz
Ksps	Kilosample(s) per second
LSB	Least Significant Bit (digits)
MCU	Micro-controller Unit
MHz	Megahertz
PWM	Pulse Width Modulation
R <sub>ADC</sub>	Input Resistance Analog-to-Digital Converter
R <sub>x</sub>	Receive
SD	Standard deviation
SH	Sample and Hold
SINAD	Signal-Noise-Ratio+Distortion
SNR	Signal-Noise-Ratio
SOC	Start of Conversion
SysCtl	System Control
TBPRD	Time Base Period Register
THD	Total Harmonic Distortion
T <sub>x</sub>	Transmit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VREF	Voltage Reference
VREF <sub>HI</sub>	Voltage Reference High
VREF <sub>LO</sub>	Voltage Reference Low

## Table of contents

1 Introduction .....	13
1.1 Task specification.....	13
1.2 Work flow.....	14
1.3 Development tools.....	14
1.3.1 Tiva (TM4C123GHM6PM) microcontroller.....	14
1.3.2 Piccolo Control Stick (TMS320F28069) microcontroller.....	14
1.3.3 Matlab.....	15
1.3.4 R-language.....	15
1.2.5 Code Composer Studio.....	15
1.2.6 Eclipse CDT .....	15
1.4 Micro-Controllers .....	15
2 Piccolo Control Stick (TMS320F28069) microcontroller.....	18
2.1 TMS320F28069 features.....	18
2.2 Testing Piccolo board.....	20
2.2.1 Piccolo (TMS320F28069) board testing .....	20
2.3 ADC Static Evaluation for Piccolo Control Stick .....	23
2.3.1 Input impedance of Analog pin .....	24
2.3.2 Sample and hold circuit.....	24
2.4 Static performance parameters of Piccolo control stick .....	25
2.4.1 External voltage reference.....	25
2.4.2 Offset Error and Full scale error.....	27
2.4.3 Sample and hold window .....	29
2.4.4 Gain Error.....	30
2.4.5 Integral Nonlinearity (INL) .....	30
2.4.6 Differential Nonlinearity (DNL) .....	31
2.5 Dynamic performance parameters.....	32



2.5.1 Piccolo board dynamic performance.....	32
2.5.2 Enhanced Pulse Width Modulator (ePWM).....	32
3 Tiva C Series - TM4C123GH6PM.....	38
3.1 Tiva ADC electrical characteristics.....	40
3.2 Testing Tiva board.....	41
3.3 ADC Static Evaluation of Tiva board.....	42
3.3.1 Single ended Input.....	42
3.3.2 Noise Error.....	43
3.3.3 Differential input.....	45
3.3.4 Dithering.....	46
3.3.5 Integral Nonlinearity (INL).....	46
3.3.6 Differential Nonlinearity (DNL).....	47
3.4 Dynamic Evaluation Tiva C board.....	48
3.4.1 Run-Mode Clock Gating Control 0 (RCGC0) register.....	48
4 Results.....	53
4.1 Piccolo (TMS320F28069) results.....	53
4.2 Tiva C (TM4C123GH6PM) results.....	55
4.3 Future work.....	56
4.4 Conclusion.....	56
Reference.....	58
Appendix 1 - Program Codes.....	60
Appendix 2 - Blinking LED code.....	60
Appendix 3 - Hello World program code.....	62
Appendix 4 - ADC Differential sampling code.....	65
Appendix 5 - ADC Single Ended Sampling code.....	69
Appendix 6 - Matlab code used to develop FFT.....	74
Appendix 7 - R-language code to analyse ADC data logs.....	77

## List of figures

Figure 1. Micro controller block diagram. ....	16
Figure 2. Piccolo control stick microcontroller. ....	18
Figure 3. Architecture of functional block Piccolo diagram. ....	19
Figure 4. Simple “hello world” code and “Blinking LED”. ....	21
Figure 5. "hello world" console output. ....	21
Figure 6. Program code that collects ten samples from the ADC. ....	22
Figure 7. Ten collected samples from the ADC. ....	22
Figure 8. Piccolo ADC architecture. ....	23
Figure 9. ADCIN pin ....	24
Figure 10. sample and hold circuit. ....	24
Figure 11. Block diagram of the equipment set up. ....	25
Figure 12. ADC register for sample and hold window. ....	26
Figure 13. Program code that controls the input voltage. ....	26
Figure 14. Program code collects 20000 samples. ....	26
Figure 15. Matlab code to analyse datalogs. ....	26
Figure 16. Piccolo transfer function. ....	27
Figure 17. Program code in R-language to calculate offset. ....	28
Figure 18. Offset Error and full scale error. ....	29
Figure 19. Gain error of the piccolo control stick ADC. ....	30
Figure 20. Piccolo INL error ....	31
Figure 21. Piccolo DNL error. ....	31
Figure 22. ePWM modules. ....	32
Figure 23. Matlab code analyse Input sine wave. ....	33
Figure 24. Sinusoidal input at 1KHz. ....	34
Figure 25. Matlab code for FFT output ....	34
Figure 26. Single sided fft of the input sinusoidal signal. ....	35
Figure 27. Program code for windowing. ....	35
Figure 28. Blackman window Sine input. ....	36
Figure 29. Normalized FFT. ....	36

Figure 30. Tiva C board .....	38
Figure 31. Architecture of Tiva TM4C123x series . .....	39
Figure 32. ADC modules in a TM4C123GH6PM.....	39
Figure 33. Program code for the temperature sensor in the Tiva board. ....	41
Figure 34. ADC results from temperature sensor.....	41
Figure 35. Test set up of Tiva C series board.....	42
Figure 36. Transfer function of Tiva c series board. ....	43
Figure 37. R language script used to analyse the data log.....	44
Figure 38. ADC output code and the analog input. ....	44
Figure 39. Program code for differential input mode.....	45
Figure 40. ADC output of a Tiva board with dithering. ....	46
Figure 41. Tiva INL error. ....	47
Figure 42. Tiva DNL error. ....	47
Figure 43. Program code for sample rate register. ....	48
Figure 44. System control register.....	48
Figure 45. Tiva sine wave FFT output spectrum.....	49
Figure 46. FFT spectrum of a sine signal. ....	49
Figure 47. Matlab code for half sided spectrum.....	50
Figure 48. Single sided spectrum in dB.....	50
Figure 49. Blackman window Sine Input. ....	51
Figure 50. FFT using Hamming window . ....	51

## List of tables

Table 1. Electrical characteristics of the TMS320F28069. ....	20
Table 2. Different piccolo logs tested.....	29
Table 3. TBPRD register. ....	33
Table 4. Program code for ePWM register.....	33
Table 5. Electric characteristics of Tiva board. ....	40
Table 6. Analysed Tiva logs .....	45
Table 7. Dynamic characteristics of the Tiva board. ....	49
Table 8. Data sheet offset error. ....	53
Table 9. Actual Offset error.....	53
Table 10. Piccolo Test results.....	53
Table 11. Piccolo INL/DNL (Datasheet).....	54
Table 12. Piccolo INL/DNL test results. ....	54
Table 13. Piccolo Data logs (INL/DNL). ....	54
Table 14. Dynamic parameters of piccolo (TMS320F28069) .....	54
Table 15. Tiva static results.....	55
Table 16. Tiva INL/DNL error (Data sheet).....	55
Table 17. Tiva INL/DNL error (Test results). ....	55
Table 18. Data logs INL/DNL errors. ....	55
Table 19. Dynamic performance written in data sheet. ....	56
Table 20. Test results for dynamic performance. ....	56

# 1 Introduction

Engineers use microcontrollers for different applications. These microcontrollers have analog-to-digital converters (ADC) they use to convert analog values like voltages to digital code [1]. In the real world, signals are mostly available in analog form. To use a microcontroller in this type of system, an ADC is required, so that analog signals can be converted to the digital values. Successive-Approximation ADCs (SAR), this is the most common architecture used for data acquisition applications. The ADCs in Piccolo and Tiva boards fall in the SAR category. Since 1940s the architecture has been utilized in experimental pulse-code-modulation (PCM) systems by Bell Labs [2]. The SAR ADCs come with different resolutions and sampling rates in MHz. These ADCs have an input sample and hold window that helps to maintain a constant signal during the conversion cycle.

ADCs come with different resolution (8bits, 12bits) depending on the requirements of the application. A 12-bit-resolution analog-to-digital converter (ADC) does not necessarily mean the system will have 12-bit accuracy [1], [3]. Much to the surprise of engineers, some ADCs will exhibit much lower performance as compared to what is written in the datasheet. When this is highlighted during the first prototype test, engineers will panic looking for what is affecting the performance of ADC. Many hours are spent reworking the design as the deadline for preproduction builds fast approaches [4].

It is very important to carry out a thorough understanding of ADC performance before building the design [5]. Analysis of the ADC performance will reveal subtleties that often lead to less than desired performance. The accuracy of analog to digital converter has an impact on overall system efficiency. To improve accuracy you need to understand the errors associated with the ADC and the parameters affecting them. This is the main reason; you need to do some careful preparation before starting your development. Understanding ADC performance will help in selecting the right ADC for an application [6].

## 1.1 Task specification.

The assignment of this project was to analyze and evaluate the ADC performance of the Piccolo control stick and Tiva c series board in real time. ADC testing is a challenging task. A comparison was made between what is written in the data sheets of these boards and what really happens in the real world when the ADC is tested. Often datasheets include ADC performance characteristics but when the ADC is tested in an application it performs below expectation. Static and dynamic Parameters were tested to measure the ADC's performance for the above mentioned boards. These parameters are critically important to instrumentation applications, medical applications, image processing, in which accuracy of each converted code is of major concern.

## Goals

The goals of this project are the following

- Overview of micro controllers in general and their importance in the development of applications.
- Analyze and establishing an over view of piccolo control stick (TMS320F28069) and Tiva (TM4C123GHM6PM) microcontrollers.
- Perform static and dynamic parameter evaluation of analog digital converters for both micro controller boards mentioned above.
- Test ADC performance at electrical absolute maximum values.
- Draw a comparison between the information written in the data sheets of both micro controllers with what is actually obtained from the real time test

## 1.2 Work flow

This thesis work is composed of four chapters. Chapter 1 is the introduction chapter. In this chapter an overview of the goals and task specification is described. A general look at the micro controllers and the tools used is also discussed. In chapter 2 the Piccolo control stick is introduced, static and dynamic tests are performed. In chapter 3 The Tiva board is introduced, static and dynamic performances are discussed. In chapter 4 results are discussed in detail and in section 4.4 a conclusion about the work is made.

## 1.3 Development tools.

This section describes the tools that have been used for development of the test software and hardware in this project.

### 1.3.1 Tiva (TM4C123GHM6PM) microcontroller.

The Tiva C Series is a low-cost microcontroller from Texas Instruments (TI) [7]. This platform, together with the integrated development environment (IDE) Code Composer Studio (CCS) [7] provides tools to develop and debug embedded applications with C/C++ programming. Configurable modules with pre-defined function libraries allow development at a high abstraction level that is easy to use.

### 1.3.2 Piccolo Control Stick (TMS320F28069) microcontroller

TMS320F28069 Piccolo Microcontroller device is members of the C2000™ Piccolo MCU platform for use within embedded control applications [8]. It's a high performance low-cost 12 bit microcontroller. This platform, together with the integrated development environment (IDE) Code Composer Studio (CCS) provides tools to develop and debug embedded applications with C/C++ programming.

### **1.3.3 Matlab**

MathWorks is a useful tool for signal analysis in this project [9], MATLAB is chosen as a tool for analyzing the data logs collected for signal processing.

### **1.3.4 R-language**

R is a language and environment for statistical computing and graphics [10]. R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering and graphical techniques, and is highly extensible [10].

### **1.2.5 Code Composer Studio**

This tool was used to analyze the functionalities of the boards and to verify that each hardware module was implemented and functioning as intended. CCS is also used to run programs that trigger the conversions of the on board ADCs.

### **1.2.6 Eclipse CDT**

The CDT Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform [11]. This tool is used to connect to the data acquisition device that collects the samples codes for the ADC.

## **1.4 Micro-Controllers**

As early as 1971 Texas instruments had started producing microcontrollers [6]. The TMS 1802 from Texas instruments was used in applications such as cash registers, watches and measuring instruments. In 1974 Texas instruments introduced the TMS 1000 that included memory i.e ROM, RAM and I/O all on a single chip [3]. Other companies that contributed to early microcontroller development were Intel with the Intel 8048 microcontrollers and Motorola with 68HCxx series of microcontrollers [12].

Today, billions of microcontrollers are produced per year, and the controllers are integrated into many appliances we have grown used to, like household appliances (microwave, washing machine, coffee machine) [13]. The internal architecture of microcontrollers is closely related. Figure 1 shows the block diagram of a typical microcontroller. All components are connected using an internal bus and are all integrated on one chip. The modules are connected to the outside world via I/O pins.

## Microcontroller block diagram

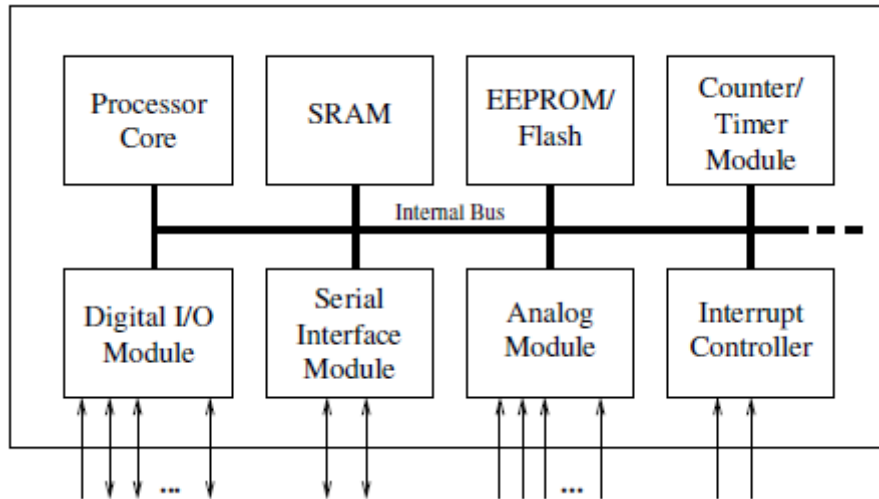


Figure 1. Micro controller block diagram [4].

**Processor Core:** This is the Central Processing Unit (CPU) of the controller. It's made up of the arithmetic logic unit, the control unit and the registers (stack pointer, program counter, accumulator register, register file) [3].

**Memory:** The memory is divided into program memory and data memory. In larger controllers, a DMA controller handles data transfers between peripheral components and the memory.

**Interrupt Controller:** Interrupts are useful for interrupting the normal program flow in case of (important) external or internal events. In conjunction with sleep modes, they help to conserve power.

**Timer/Counter:** Microcontrollers carry with them 2-3 Timer/Counters; these are used to timestamp events and measure intervals.

**PWM (pulse width modulation):** This is usually used to drive motors or for safety. The PWM output can be joined with an external filter to realize a cheap digital/analog converter.

**Digital I/O:** Parallel digital I/O ports are one of the main features of microcontrollers. The number of I/O pins varies from 3 to over 90, depending on the controller family and the controller type.

**Analog I/O:** Most microcontrollers have integrated analog/digital converters, which differ in the number of channels and their resolution. The analog module is also integrated with an analog comparator.

**Interfaces:** Controllers have at least one serial interface which can be used to download the program and for communication with the development PC in general. Serial interfaces can also be used to communicate with external peripheral devices, most controllers offer several interfaces like SPI and SCI [2]. Larger microcontrollers contain PCI, USB and Ethernet interfaces.



Watchdog Timer: The biggest application area of microcontrollers is in safety-critical systems, it is important to guard against errors in the program or the hardware. The watchdog timer major function is to reset the controller in case of software “crashes”.

Debugging Unit: Microcontrollers have additional hardware to allow remote debugging of the chip from the PC [2].

## 2 Piccolo Control Stick (TMS320F28069) microcontroller

TMS320F28069 Piccolo Microcontroller device is members of the C2000™ Piccolo MCU platform for use within embedded control applications [8]. It's a high performance low-cost 12 bit microcontroller [8]. In this chapter the TMS320F28069 board internal performance is tested. The test investigation will be centred on the static and dynamic performance of the ADC. The aim is to analyse the ADC performance with in normal values and at absolute maximum. Draw a comparison between obtained results and datasheet results.

### 2.1 TMS320F28069 features.

The following are some of the main features [8].

- Quick and easy evaluation of all of the advanced capabilities for just \$39
- Convenient and easy-to-use GUI provides hands-on experimentation with the floating point capabilities of the Piccolo F2806x MCU.
- Slightly larger than a memory stick
- On-board emulation, access to all I/O pins
- Detailed example software and documentation
- Complete hardware schematics, Gerber files.

The figure 2 below shows a piccolo control stick. All components are connected using an internal bus and are all integrated on one chip. The modules are connected to the outside world via peripheral header pins.

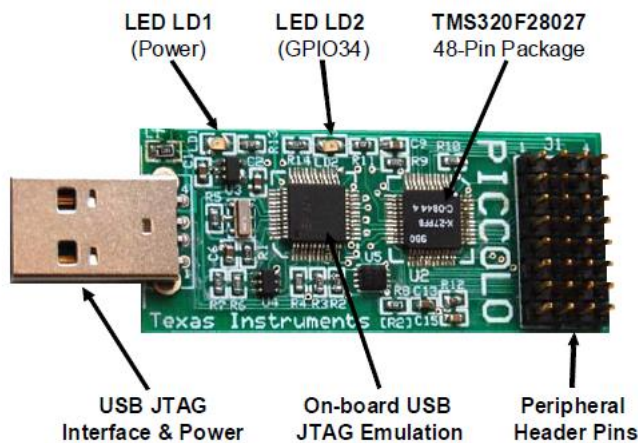


Figure 2. Piccolo control stick microcontroller [8].

A basic CPU architecture is depicted in Figure 3 below. It consists of the functional block diagram of the device and the communication busses.

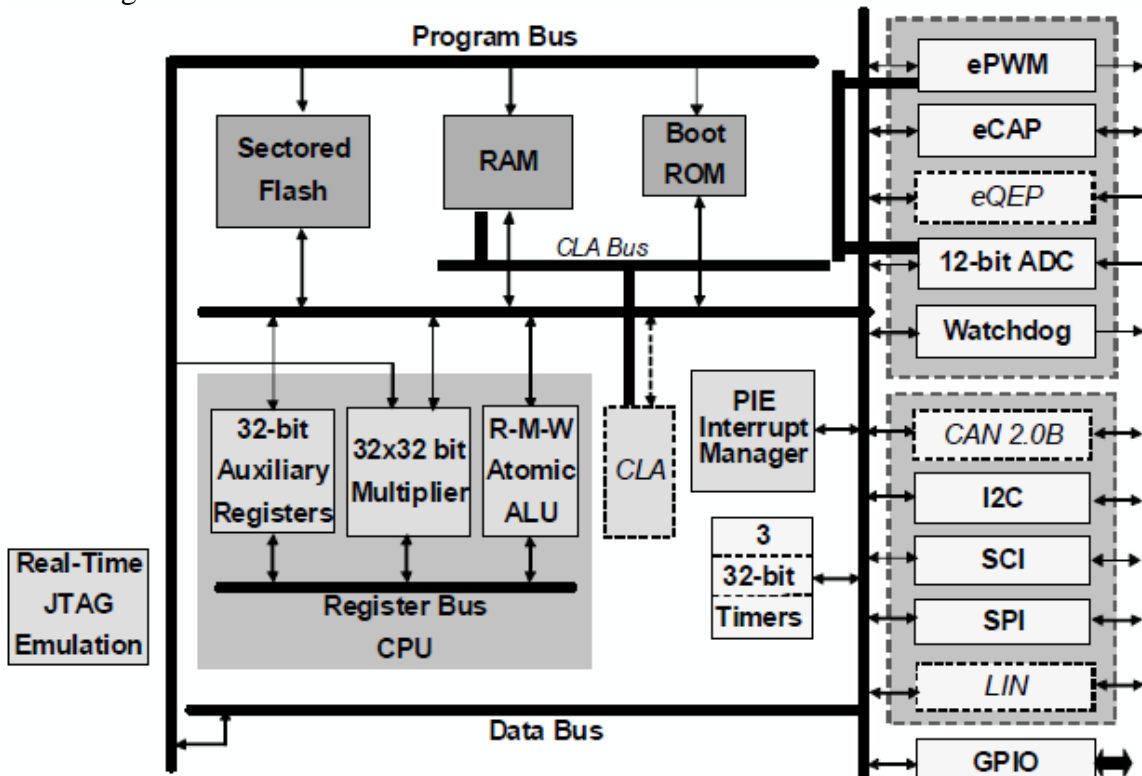


Figure 3. Architecture of functional block Piccolo diagram [8].

The architecture of a TMS320F28069 Piccolo can be divided into three parts i.e peripherals, Memory, CPU and bussing. The processor core (CPU) is the main part of the microcontroller made up of the 32 bit auxiliary registers [8], 64bit multiplier and an atomic arithmetic logical unit. The design is highly integrated, high performance solution for demanding control applications. The memory space on the F28069 is divided into program memory and data memory. There are several different types of memory available that can be used as both program memory and data memory [8].

This chapter is centered on the ADC performance of the TMS320F28069 piccolo control stick. The ADC block is a 12-bit converter. It has up to 16 single-ended channels pinned out. The ADC also contains two sample-and-hold units for simultaneous sampling with a full range analog input: 0 V to 3.3 V fixed.

The table 1 below shows the electrical characteristics of the TMS320F28069

PARAMETER		MIN	TYP	MAX	UNIT
<b>DC SPECIFICATIONS</b>					
Resolution		12			Bits
ADC clock	90-MHz device	0.001		45	MHz
Sample Window		7		64	ADC Clocks
<b>ACCURACY</b>					
INL (Integral nonlinearity) <sup>(1)</sup>		-4		4	LSB
DNL (Differential nonlinearity), no missing codes		-1		1.5	LSB
Offset error <sup>(2)</sup>	Executing a single self-recalibration <sup>(3)</sup>	-20		20	LSB
	Executing periodic self-recalibration <sup>(4)</sup>	-4		4	
Overall gain error with internal reference		-60		60	LSB
Overall gain error with external reference		-40		40	LSB
Channel-to-channel offset variation		-4		4	LSB
Channel-to-channel gain variation		-4		4	LSB
ADC temperature coefficient with internal reference			-50		ppm/°C
ADC temperature coefficient with external reference			-20		ppm/°C
V <sub>REFLO</sub>			-100		μA
V <sub>REFHI</sub>			100		μA
<b>ANALOG INPUT</b>					
Analog input voltage with internal reference		0		3.3	V
Analog input voltage with external reference		V <sub>REFLO</sub>		V <sub>REFHI</sub>	V
V <sub>REFLO</sub> input voltage <sup>(5)</sup>		V <sub>SSA</sub>		0.66	V
V <sub>REFHI</sub> input voltage <sup>(6)</sup>		2.64		V <sub>DDA</sub>	V
	with V <sub>REFLO</sub> = V <sub>SSA</sub>	1.98		V <sub>DDA</sub>	

Table 1. Electrical characteristics of the TMS320F28069 [8].

## 2.2 Testing Piccolo board.

Verifying the boards was carried out in several ways and had to be done thoroughly, it is the only way to guarantee that the board is in good working conditions. In this section, the board was first tested separately. The piccolo control stick was tested using the example projects that come with the control suite package using the CCS debugger. This tool was used to analyze the function of the control stick to verify that all modules function as intended.

### 2.2.1 Piccolo (TMS320F28069) board testing

A simple hello world example that returns hello world (//! A very simple ``hello world" example). It simply displays ``Hello World!" to the console (Real Terminal) and ‘Blinking LED’ that makes the on board LED blink were some of the example projects run to test for proper functionality.

Below is a simple “hello world” code and “Blinking LED”

```
// Hello!  
//  
UARTprintf("Hello, world!\n");  
  
//  
// We are finished. Hang around doing nothing.  
//  
while (1)  
{  
    //  
    // Turn on the BLUE LED.  
    //  
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);  
  
    //  
    // Delay for a bit.  
    //  
    SysCtlDelay(SysCtlClockGet() / 10 / 3);  
  
    //  
    // Turn off the BLUE LED.  
    //  
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);  
}
```

Figure 4. Simple “hello world” code and “Blinking LED”.

The output is printed to the real terminal console via uart and the blue Led is turned on.

A screenshot of a terminal window with a black background. The text "Hello, world!" is displayed in yellow. To the right of the text, there is a small blue cursor icon. The text is contained within a black rectangular box.

Figure 5. "hello world" console output.

The piccolo control stick has a 12 bit ADC. To test the ADC of the piccolo control stick board a simple code that returns ten ADC values is used to collect ten samples when “M” is pressed in the console terminal.

The code below shows an interrupt that collects ten samples from the ADC when M command is pressed in the console.

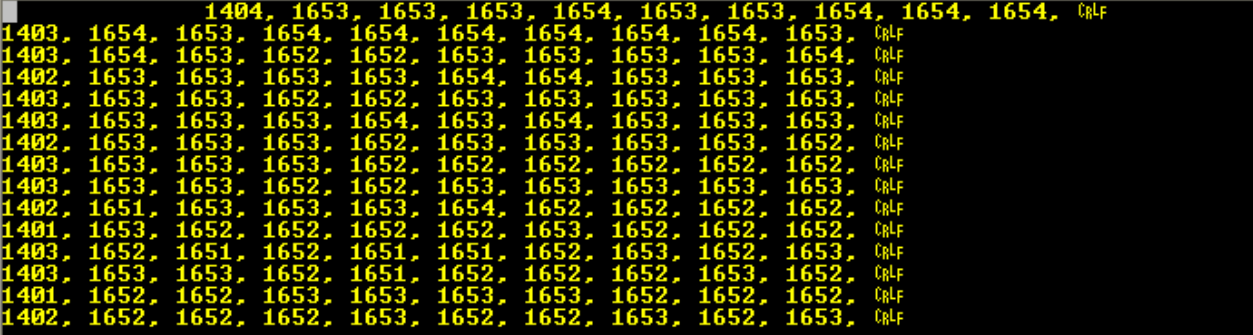
```
// Wait for ADC interrupt + Badru: wait for 'm' command to start collecting ADC
for(;;)
{
    if(SciaRegs.SCIFFRX.bit.RXFST == 1) //If there is some user input from SCI communication just loop
    {
        char ReceivedChar = SciaRegs.SCIRXBUF.all;
        switch (ReceivedChar)
        {
            case 'm': case 'M':
                ConversionCount = 0; RunFlag=1; ReadyFlag=0;
                break;
            default: break;
        }
    }

    if (ReadyFlag)
    {
        short k;
        char xxbuff[30];
        for (k=0; k< SMPLCNT; k++)
        {
            sprintf (xxbuff, "%d, ", Voltage1[k]); scia_msg (xxbuff);
        }
        scia_xmit (13); scia_xmit(10); // EndOfLine (CR+LF)

        ReadyFlag=0; // Badru:ready- results now sent out
    }
}
```

Figure 6. Program code that collects ten samples from the ADC.

On the console ten samples from the ADC are collected when “M” command is pressed in the console.



```
1404, 1653, 1653, 1653, 1654, 1653, 1653, 1654, 1654, 1654, CR+LF
1403, 1654, 1653, 1652, 1652, 1653, 1653, 1653, 1653, 1654, CR+LF
1402, 1653, 1653, 1653, 1653, 1654, 1654, 1653, 1653, 1653, CR+LF
1403, 1653, 1653, 1652, 1652, 1653, 1653, 1653, 1653, 1653, CR+LF
1403, 1653, 1653, 1653, 1654, 1653, 1654, 1653, 1653, 1653, CR+LF
1402, 1653, 1653, 1653, 1652, 1653, 1653, 1653, 1653, 1652, CR+LF
1403, 1653, 1653, 1653, 1652, 1652, 1653, 1652, 1652, 1652, CR+LF
1403, 1653, 1653, 1652, 1652, 1653, 1653, 1653, 1653, 1653, CR+LF
1402, 1651, 1653, 1653, 1653, 1654, 1652, 1652, 1652, 1652, CR+LF
1401, 1653, 1652, 1652, 1652, 1652, 1653, 1652, 1652, 1652, CR+LF
1403, 1652, 1651, 1652, 1651, 1651, 1652, 1653, 1652, 1653, CR+LF
1403, 1653, 1653, 1652, 1651, 1652, 1652, 1652, 1653, 1652, CR+LF
1401, 1652, 1652, 1653, 1653, 1653, 1653, 1652, 1652, 1652, CR+LF
1402, 1652, 1652, 1652, 1653, 1652, 1652, 1653, 1652, 1653, CR+LF
```

Figure 7. Ten collected samples from the ADC.

## 2.3 ADC Static Evaluation for Piccolo Control Stick

An Analog Digital Converter (ADC) is a module that is made up of mainly three components i.e. analog input, reference voltage input and digital outputs [8]. The ADC's main function is to convert the analog input signal to a digital output value that represents the size of the analog input comparing to the reference voltage [12]. In other words it samples the input analog voltage and produces an output digital code for each sample taken.

The block diagram below shows piccolo ADC architecture.

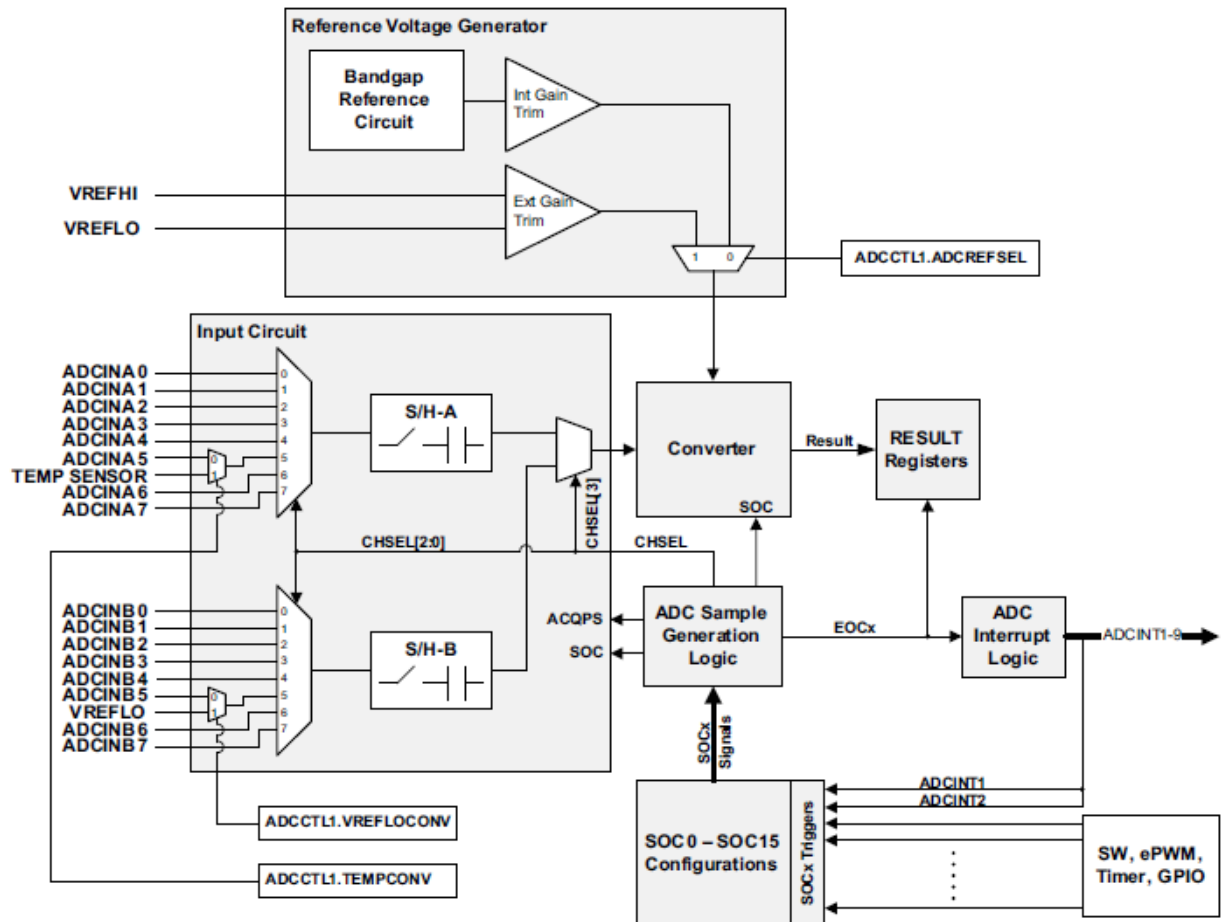


Figure 8. Piccolo ADC architecture [8].

As previously mentioned the ADC of the piccolo control stick is a 12 bit ADC with a voltage range of (0-3.3) V. The 12 bit converter is fed by two sample and hold circuit with up to 16 analog input channels. It's important to note that the ADC is not sequencer based; this makes it easy to create a series of conversions with only a single trigger. Different parameters determine the performance of the ADC for a piccolo control stick. These are relied on to determine the accuracy of the ADC. In this chapter, parameters have been grouped into static performance parameters and dynamic performance parameters.

### 2.3.1 Input impedance of Analog pin

Before looking at the parameters it's important to discuss the input peripheral pins and the sample and hold circuit. The pins can be designed as a RC circuit. The  $C_{ADC}$  is the hold capacitor and the  $R_{ADC}$  is the resistance caused by the sampling switch [8]. The figure 9 below shows the ADCIN pin.

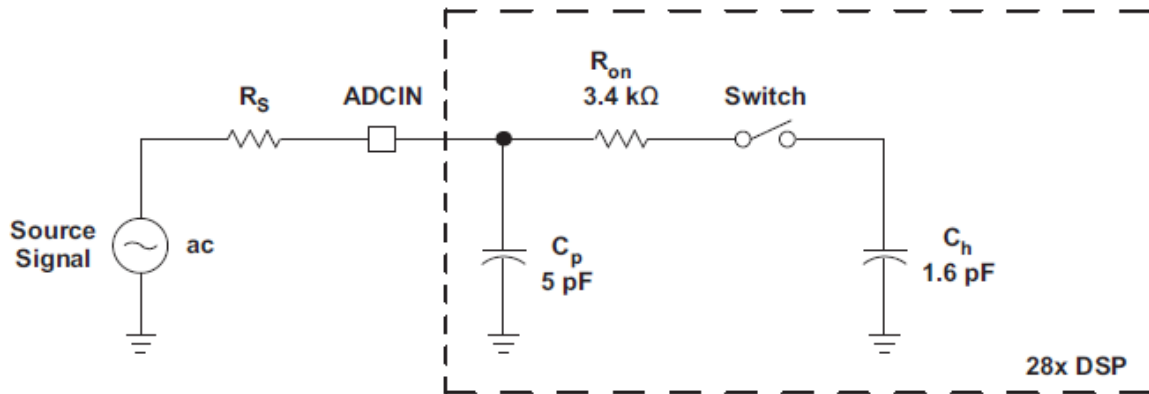


Figure 9. ADCIN pin [8].

Minimum impedance is achieved when the hold capacitor is fully discharged. The current that flows to the pin reduces when the capacitor is charged again.

### 2.3.2 Sample and hold circuit.

The hold capacitor is charged by this circuit. The sample and hold capacitor circuit is responsible for sampling the input signal. The analog pin is disconnected and the voltage across the capacitor is converted to digital code using successive approximation.

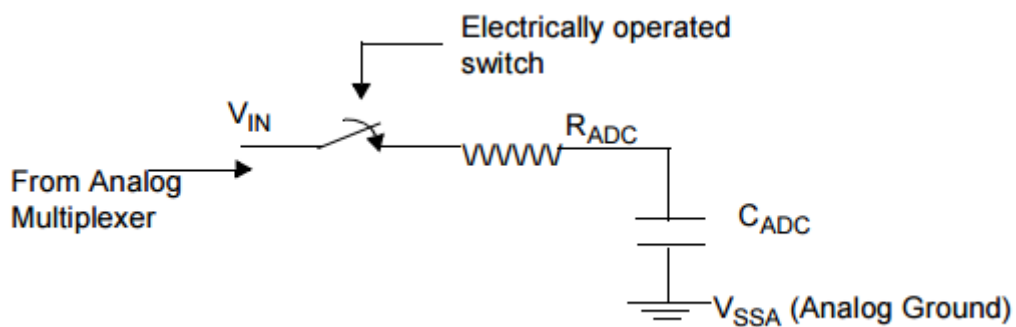


Figure 10. sample and hold circuit [8].

Immediately when the ADC conversion starts, the switch is closed. This connects the hold capacitor to the analog input along the internal ADC resistance  $R_{ADC}$ . This causes a charging current to flow into the analog input and the capacitor starts to charge.



## 2.4 Static performance parameters of Piccolo control stick

Static performance parameters, these are parameters that are not related to ADC's input signal. The test is influenced by the DC component of the input voltage. The concept is to put one input voltage value and see 31 ADC output codes for each input. Static parameters include gain error, offset error, full scale error and linearity errors. For the piccolo control stick these parameters have been shown in the accuracy section in table1.

### 2.4.1 External voltage reference.

Two pins are chosen to generate reference voltage. The internal band gap is responsible for the choosing the reference voltage for the ADC. The band gap converts the voltage based on a fixed scale (0-3.3v) range. In this case the internal band gap is not used; VREFLO and VREFHI are externally controlled.

A set up was made to test the absolute maximum and minimum voltage in takes for the ADC. The block diagram below shows components that are used to test for the voltage range input and the ADC output values of the piccolo control stick 12 bit converter.

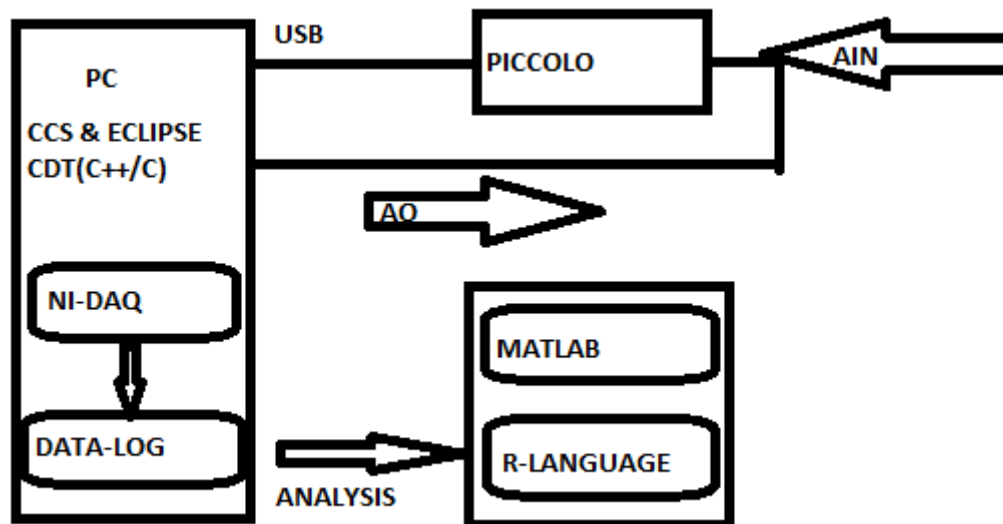


Figure 11. Block diagram of the equipment set up.

In this test the full range (-0.9-3.4) V is divided into 2000 equal steps. The NI DAQ takes 31 samples at every voltage input. A connection is established between the piccolo control stick and the PC via a USB connection. A program code is debugged and run in code composer studio. The code collects 31 samples from the ADC. Channel ADC<sub>A0</sub> is connected to the national instrument data acquisition device. R-language and Matlab are used to analyze the collected data logs. The ePWM is used to trigger the start of conversion event (SOC). The ACQPS is used to determine the sample and hold window size. The minimum number of sample cycles possible is 7 cycles which equates to (6 ACQPS). The total sampling time = sample window size + conversion time of the ADC. In case of over sampling SOC<sub>0</sub> and SCO<sub>1</sub> can be given the same value e.g. 28 cycles = 27 ACQPS

```

// Configure ADC
EALLOW;
AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1; // Enable non-overlap mode
AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1; // ADCINT1 trips after AdcResults latch
AdcRegs.INTSEL1N2.bit.INT1E = 1; // Enabled ADCINT1
AdcRegs.INTSEL1N2.bit.INT1CONT = 0; // Disable ADCINT1 Continuous mode
AdcRegs.INTSEL1N2.bit.INT1SEL = 1; // setup EOC1 to trigger ADCINT1 to fire
AdcRegs.ADCSOC0CTL.bit.CHSEL = 4; // set SOCO channel select to ADCINA4
AdcRegs.ADCSOC1CTL.bit.CHSEL = 2; // set SOC1 channel select to ADCINA2
AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 5; // set SOCO start trigger on EPWM1A, due to roun
AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 5; // set SOC1 start trigger on EPWM1A, due to roun
AdcRegs.ADCSOC0CTL.bit.ACQPS = 6; // set SOCO S/H Window to 7 ADC Clock Cycles, (6
AdcRegs.ADCSOC1CTL.bit.ACQPS = 6; // set SOC1 S/H Window to 7 ADC Clock Cycles, (6
EDIS;

```

Figure 12. ADC register for sample and hold window.

A program code that controls the input voltage is run in eclipse simultaneously. The physical channel and the data acquisition device are specified as well. The code below shows data acquisition channel and the voltage range being used. In this case channel ADCINA0 and a voltage range of (-0.9 – 3.410) V is used. It should be noted that the absolute maximum voltage range based on the piccolo control stick data sheet is (-0.3 – 4.6) V.

```

/-- Definitions for DAQ
#define physicalChannel "NI_USB_6281/ai6,NI_USB_6281/ai7" // "Dev1/ai0"
#define physicalChannel_AO "Dev1/ao0"
#define terminalConfig DAQmx_Val_RSE //DAQmx_Val_NRSE
#define terminalConfig_AO DAQmx_Val_RSE
#define minVal_AI -2.0
#define maxVal_AI 2.0
#define minVal_AO -0.09
#define maxVal_AO +3.410 //
#define units DAQmx_Val_Volts

```

Figure 13. Program code that controls the input voltage

This program code collects 20000 samples of different voltage input that are converted to digital values by the ADC.

```

#define NNN 20000 //20000 //300 // Number of AO samples

```

Figure 14. Program code collects 20000 samples.

The samples are collected in a data file and analyzed using Mat lab program.

```

>> plot(Analog,ADC,'red')
>> hold on
>> %plot(x,y,'blue')
>> xlabel('Analog input-v')
>> ylabel('ADC code')
>> title('ADC REDAING')
>> legend('ADCoutput','ideal','Location','southeast')

```

Figure 15. Matlab code to analyse datalogs.

The figure 16 below shows a transfer function - plot of analyzed data of the analog input against the ADC values.

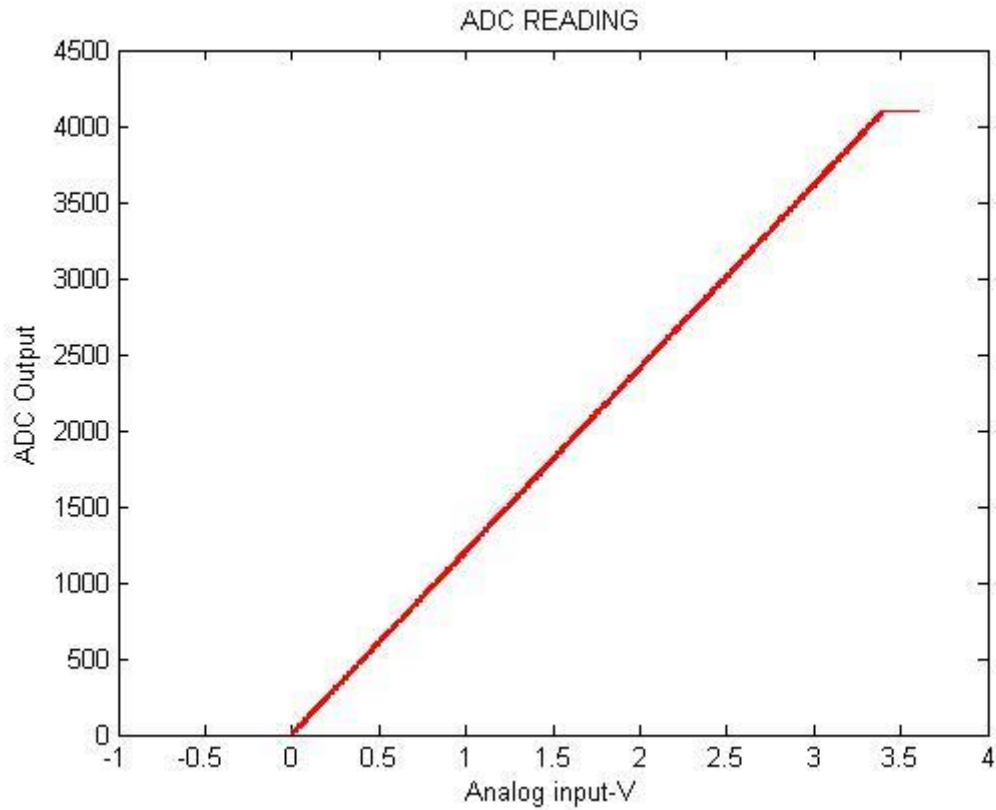


Figure 16. Piccolo transfer function.

The transfer function of an ADC is a plot of the voltage input to the ADC versus the output codes from the ADC. The plot is not continuous but is a plot of  $2^N$  codes, where  $N$  is the ADC's resolution in bits. In this case  $N=12$  bits giving us 4096 codes. The ideal transfer function plot is a straight line. It's obtained by connecting the codes at the code-transition boundaries. Figure 16 doesn't show a straight line but something close to a straight line. When the input voltage is less than 0, the digital value returned by the ADC is 0. When the voltage input is around 3.3974V the digital value returned is 4095 codes. Beyond 3.397V the ADC begins to saturate hence the horizontal flat line as shown in figure 16.

#### 2.4.2 Offset Error and Full scale error

Offset error is caused by the deviation from the ideal voltage input at code 0. At code 0 the input voltage is not 0 volts. Full scale error, this is the difference between the actual last transition voltage and the ideal last transition voltage. To obtain these errors ADC data from the piccolo control stick was analyzed using a program code in R-language. A compensation coefficient is used to get the horizontal output of the data. Below is the code used to plot figure 18.

```

> Kcorrect= 1.031
> #Kcorrect= 1.016
> #Kcorrect= 1.01851
>
> ADCcode0= Uin* (4095/ (3.3* Kcorrect)) ##*1.013))
>
> for (k in (1:xlen))
+ {
+
+ yy=as.numeric (dd[k, 2:ylen1])
+ Dout1[k]= yy[1] # 1-st sample only
+ Dout0[k]= mean(yy) # average
+ Dout_pp[k]= max(yy) -min (yy)
+ Dout_sd[k]= sd(yy)
+ Derr0 [k]= Dout0[k]- ADCcode0[k]
+ Derr1 [k]= Dout1[k]- ADCcode0[k]
+ }
> plot (Uin, Dout0, type='l')
> grid()
> xrr= c (0.11, 3.29) ##(0.1, 0.2)
> yrr= c(-20,40 )
> #plot (Uin, Derr, type='l', ylim= c(-6, 7) )
> plot (Uin, Derr1, type='l', ylim= yrr, col='red', xlim= xrr)
> lines (Uin, Derr0, type='l', col= 'blue' )
> #lines (Uin, Dout_pp, type='l', col= 'brown' )
> #lines (Uin, Dout_sd, type='l', col= 'grey' )
> grid()
> txt= paste (xfile, "Kcorr=",Kcorrect )
> title (txt)

```

Figure 17. Program code in R-language to calculate offset.

The offset error in this case is -13.3 digits and full scale error is 0.019 digits. Offset error can be a positive or negative value. The full scale error is calculated by subtracting the relative scale (RelScale) from 1. In figure 18 below the black color represents the ADC codes plus noise and the blue is the average of the noisy codes. The average also shows the internal noise in the converter. The ADC starts to sample when the input voltage is around 0.11V. The full scale is attained at around 3.29V. Offset Error in volts is calculated using this equation. Offset Error (V) = Error in LSB × Maximum Input / (number of bits). Therefore in this case the full scale error in volts would be -0.01071V.

The figure 18 below shows the offset error and full scale error.

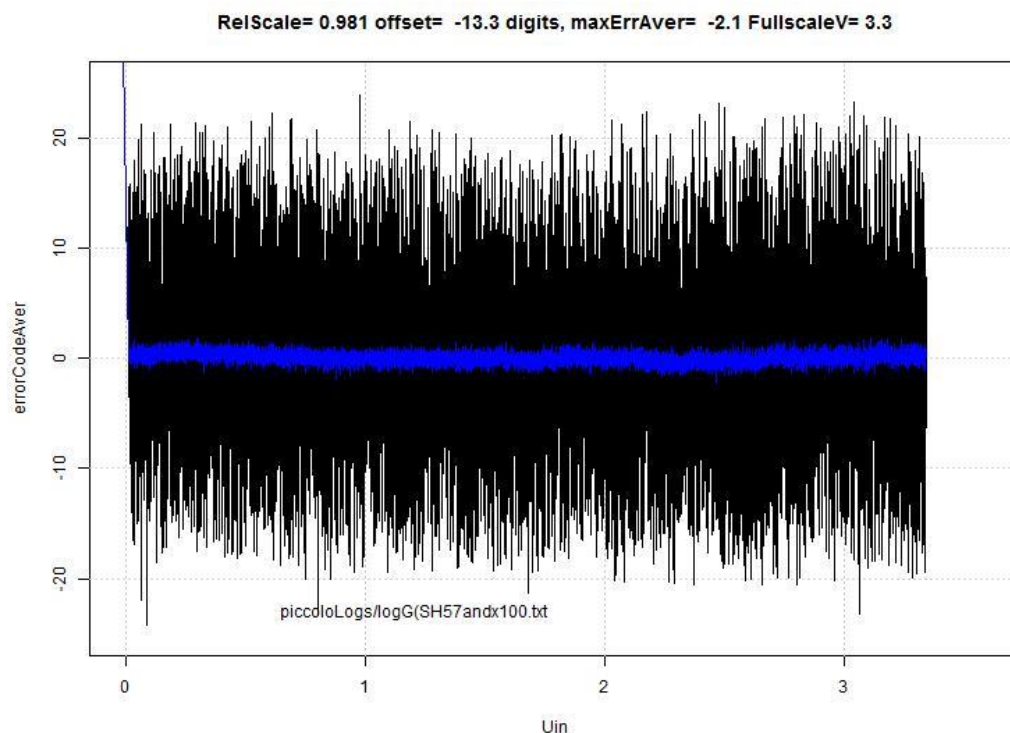


Figure 18. Offset Error and full scale error.

The black plot shows the noise distribution for each code from 0 to full scale. It's very important to understand the noise performance of the converter across all codes. In this case the noise of the converter is a function of its codes. In this particular acquisition log file, we have a stable distribution of the noise across all codes. The blue plot in figure 18 shows the distribution of noise, perturbation at the beginning codes and at the end in higher codes. The blue plot also shows us that the noise distribution is even across all codes.

### 2.4.3 Sample and hold window

The sample and hold window gives flexibility to the rate at which the sampling capacitor of the ADC is charged. Every ADCSOCxCTL register is made of a 6 bits field. The sample and hold window size is determined by this register. In this work different sample and hold windows were used to observe their effect on the ADC output code from four different data logs.

	Sampleand hold	Frequency (Hexa decimal)	Offseterror (digits)	Full scale error(digits)	Maximum Average error (digits)	SD
Log D	6	X80	5.6	0.031	4.7	1.39
Log E	13	X 28	-17	0.017	-2.3	1.41
Log F	27	X100	-12	0.019	1.8	1.40
Log G	57	X100	-13	0.019	-2.1	1.20

Table 2. Different piccolo logs tested.

From the table 2 above its observed that the sample and hold window has so much effect on the offset and gain error and on the stability performance of the ADC result e.g. log D and log E.

#### 2.4.4 Gain Error

The error in ADC is measured in comparison with the slope of the ideal transfer function. The ideal transfer function line is perfectly linear. At 0V the output code is exactly 0 and at 3.3V(Vref) the output code is exactly 4095. In real world such a linear output cannot be archived. The figure 19 below shows the gain error of the piccolo control stick ADC. The gain error in this case is 46 digits.

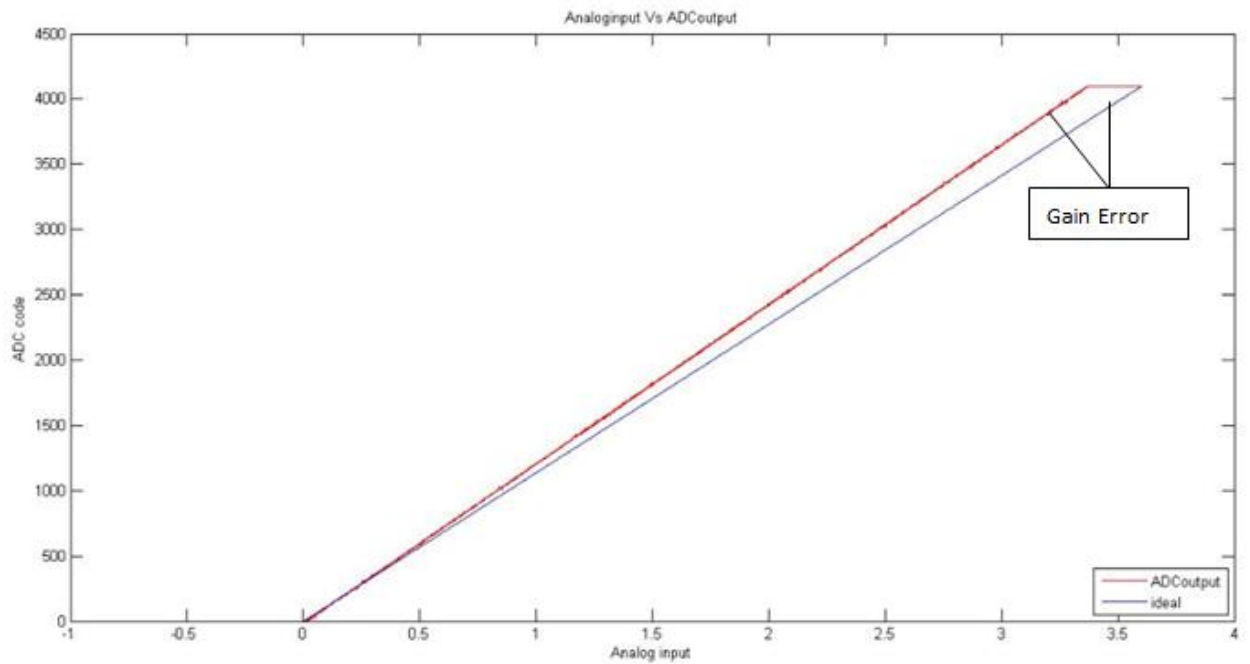


Figure 19. Gain error of the piccolo control stick ADC.

#### 2.4.5 Integral Nonlinearity (INL)

Integral nonlinearity is the deviation of individual code from the ideal straight line drawn from zero to full scale. At point zero, there is 0.5LSB before the first transition code and full scale occurs at one half LSB beyond the last code transition. To measure the deviation, one has to start from the center of each particular code to the ideal straight line between two points. The Piccolo board integral nonlinearity is tested using the static ADC data logs collected. This is done with the histogram method using the R-code. INL error can be positive or negative i.e. +1LSB or -1LSB. INL in this case is approximately +5/-10 LSB. The figure 20 below shows piccolo INL error.

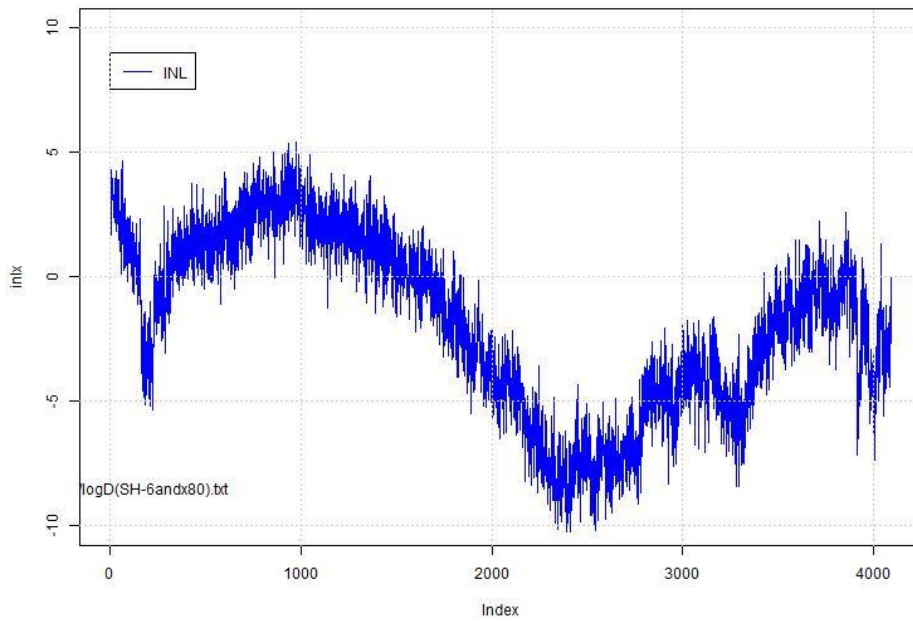


Figure 20. Piccolo INL error

#### 2.4.6 Differential Nonlinearity (DNL)

In an ideal transfer function each code transition is 1LSB apart. In reality there are deviations from the ideal LSB value. A DNL error can be positive or negative i.e. +1LSB or -1LSB. The piccolo control stick DNL is tested with the static data logs collected. This is done with the histogram method using the R - code previously used to test the piccolo control stick. The figure 21 below shows DNL error. DNL= +4/-4 LSB

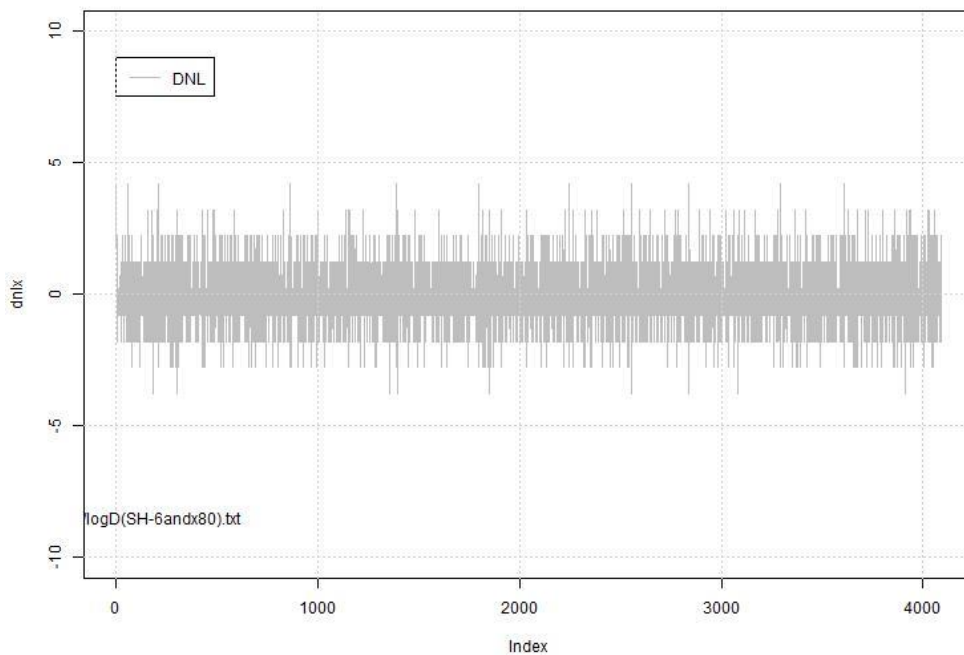


Figure 21. Piccolo DNL error.

## 2.5 Dynamic performance parameters

Dynamic performance parameters, these are parameters related to ADC's input signal. These parameters include signal-to-noise ratio (SNR), total harmonic distortion (THD), signal to noise and distortion (SINAD) and effective number of bits (ENOB). This section shows test results from the dynamic test of the Piccolo ADC.

### 2.5.1 Piccolo board dynamic performance.

The Piccolo control stick is connected to a signal generator using two wires, one of the wires is connected to ground and the other is connected to ADCIN4. The board is connected to a pc via a USB connection. The program code used in the previous tests to trigger samples from the ADC is used to collect data logs of 300 samples. In this test a sinusoidal signal of different frequencies is used to analyze the behavior of the on board ADC. According to the data sheets the board has a fast conversion rate: Up to 80ns. Before starting the test a better understanding of the sample rate control is discussed.

### 2.5.2 Enhanced Pulse Width Modulator (ePWM)

In this project the F28069 control stick uses the ePWM1 to drive the ADC sampling rate. The ePWM in the piccolo is built up from smaller single channel modules with separate resources, together these can operate as required to form a system [8]. Figure 22 below shows the different modules.

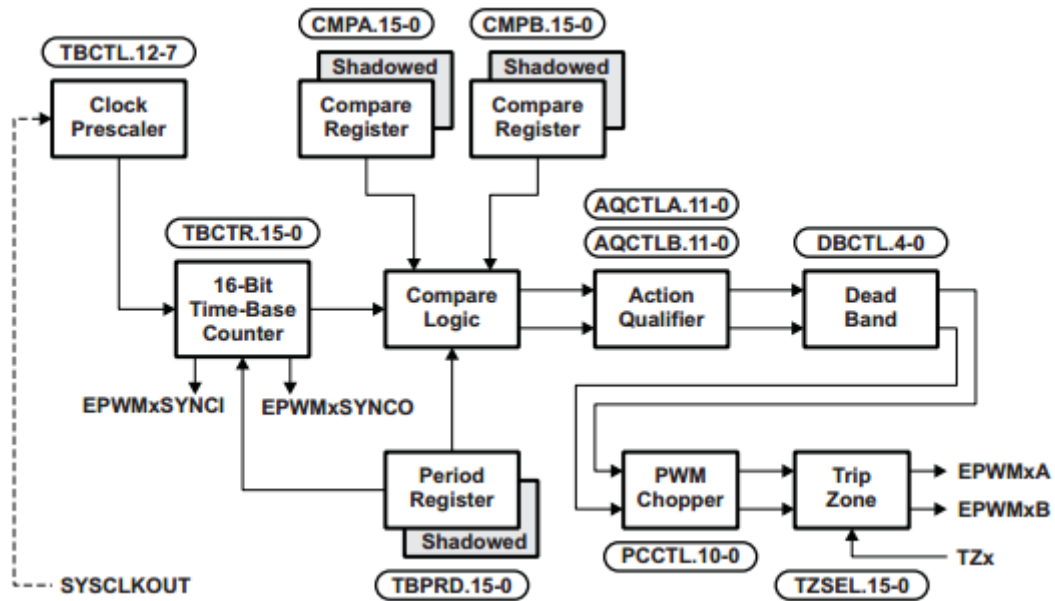


Figure 22. ePWM modules [8].

In this project the Time-Base Period Register (TBPRD) is configured in code composer studio to control the sampling rate of the ADC. The TBPRD register sets up the pwm frequency using (0-15) bits. These bits set the period of the time base counter.



The table 3 below shows the fields of the TBPRD register.

Bits	Name	Value	Description
15-0	TBPRD	0000-FFFFh	<p>These bits determine the period of the time-base counter. This sets the PWM frequency.</p> <p>Shadowing of this register is enabled and disabled by the TBCTL[PRDL] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> <li>• If TBCTL[PRDL] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the active register will be loaded from the shadow register when the time-base counter equals zero.</li> <li>• If TBCTL[PRDL] = 1, then the shadow is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware.</li> <li>• The active and shadow registers share the same memory map address.</li> </ul>

Table 3. TBPRD register [8].

The code used to determine the sample rate of 300ksp is shown below.

```
Assumes ePWM1 clock is already enabled in InitSysCtrl();
EPwm1Regs.ETSEL.bit.SOCAEN = 1; // Enable SOC on A group
EPwm1Regs.ETSEL.bit.SOCASEL = 4; // Select SOC from CMPA on upcount
EPwm1Regs.ETPS.bit.SOCAPRD = 1; // Generate pulse on 1st event
EPwm1Regs.CMPA.half.CMPA = 0x0080; // Set compare A value
EPwm1Regs.TBPRD = 300; // 0x1388; // Set period for ePWM1
EPwm1Regs.TBCTL.bit.CTRMODE = 0; // count up and start
```

Table 4. Program code for ePWM register.

The first sample data collected is used to analyze the behavior of the piccolo control stick. The sine wave input has a frequency of 1 KHz and the sampling rate is about 300ksp.

The output ADC codes are analyzed using the following Matlab code.

```
>> % badru 2016
signal = load('pp2.txt');
N = length(signal);
KHz=1e3;
fs = 300*KHz; % 300 samples per second
fnyquist = fs/2; %Nyquist frequency
Pk2Pk= max(signal) - min(signal)
rms= std(signal)
plot(signal)

Pk2Pk =

    1286

rms =

    449.3244

>> title('sinsoidal input at 1KHz Pk_Pk =1286 digits, RMS=449.3digits,
>> ylabel('Magnitude');
>> xlabel('Frequency')
```

Figure 23. Matlab code analyse Input sine wave.

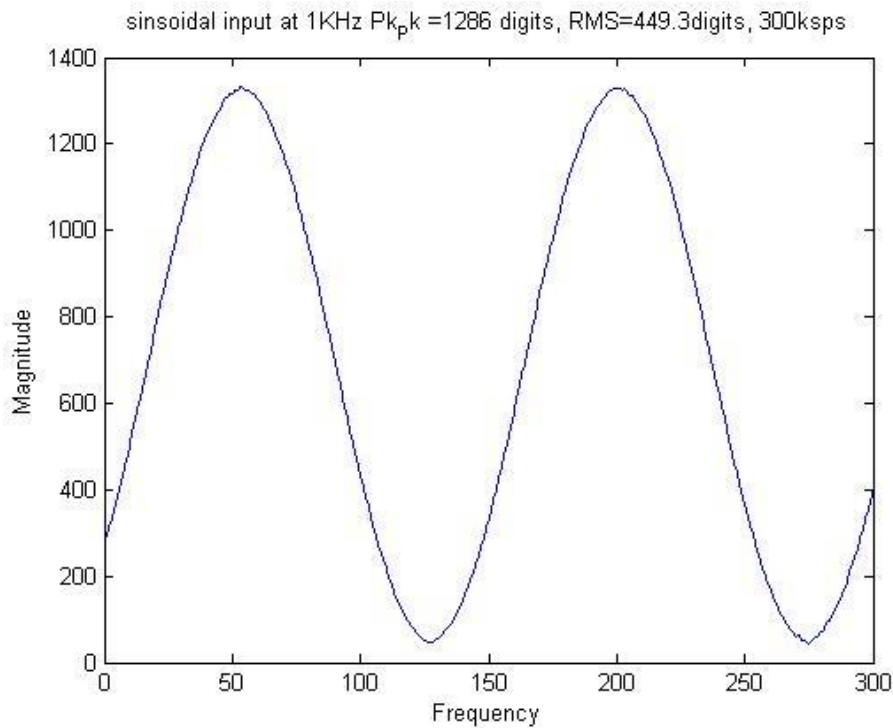


Figure 24. Sinusoidal input at 1KHz.

In order to understand the noise component of the signal and the harmonics a logarithmic scale is used and the magnitude is converted to dB. The FFT output is a single sided spectrum. The frequency scale is in KHz to correctly position the spectrum.

```
>> %Half magnitude spectrum in decibels and KHz
X_mags = abs(fft(signal));
bin_vals = [0 : N-1];
fax_Hz = bin_vals*fs/N;
N_2 = ceil(N/2);
plot(fax_Hz(1:N_2), 10*log10(X_mags(1:N_2)))
xlabel('Frequency (KHz)')
ylabel('Magnitude (dB)');
title('Half Magnitude spectrum (1kHz) 300ksps,pk_pk= 1286,rms= 449.3');
```

Figure 25. Matlab code for FFT output

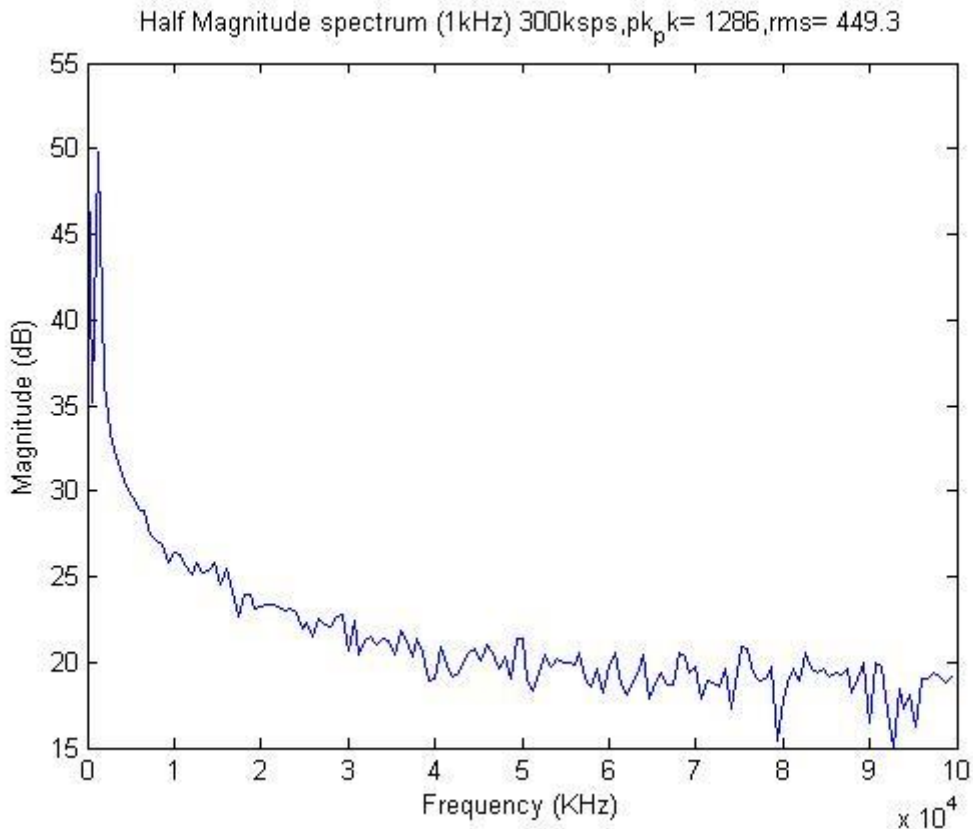


Figure 26. Single sided fft of the input sinusoidal signal.

The FFT output is used like an analog spectrum analyzer to measure the amplitude of the various harmonics and noise components of a digitized signal.

#### Windowing

Although performing an FFT on a signal can provide great insight, it is important to know the limitations of the FFT and how to improve the signal clarity using windowing. The code below is used to analyze the sinusoidal in put using Blackman window.

```
>> signal = load ('pp2.txt');
>> N = length (signal);
>> w = blackman (N)
```

Figure 27. Program code for windowing.

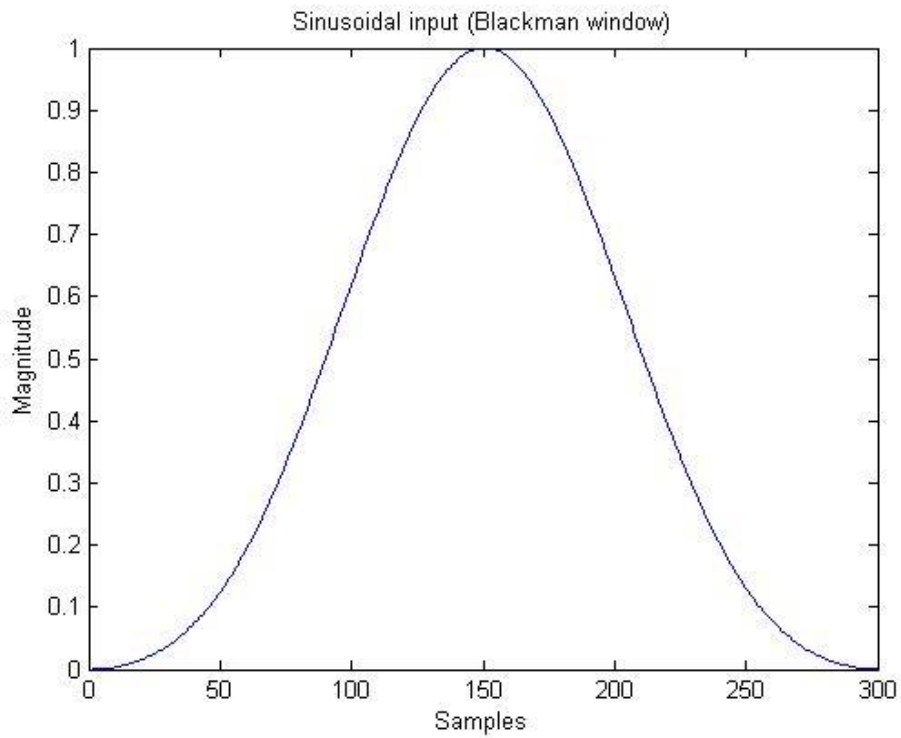


Figure 28. Blackman window Sine input.

The figure 29 below shows the normalized FFT output using Hamming window. The main lobe is -3dB and relative side lobe attenuation of -6.1dB.

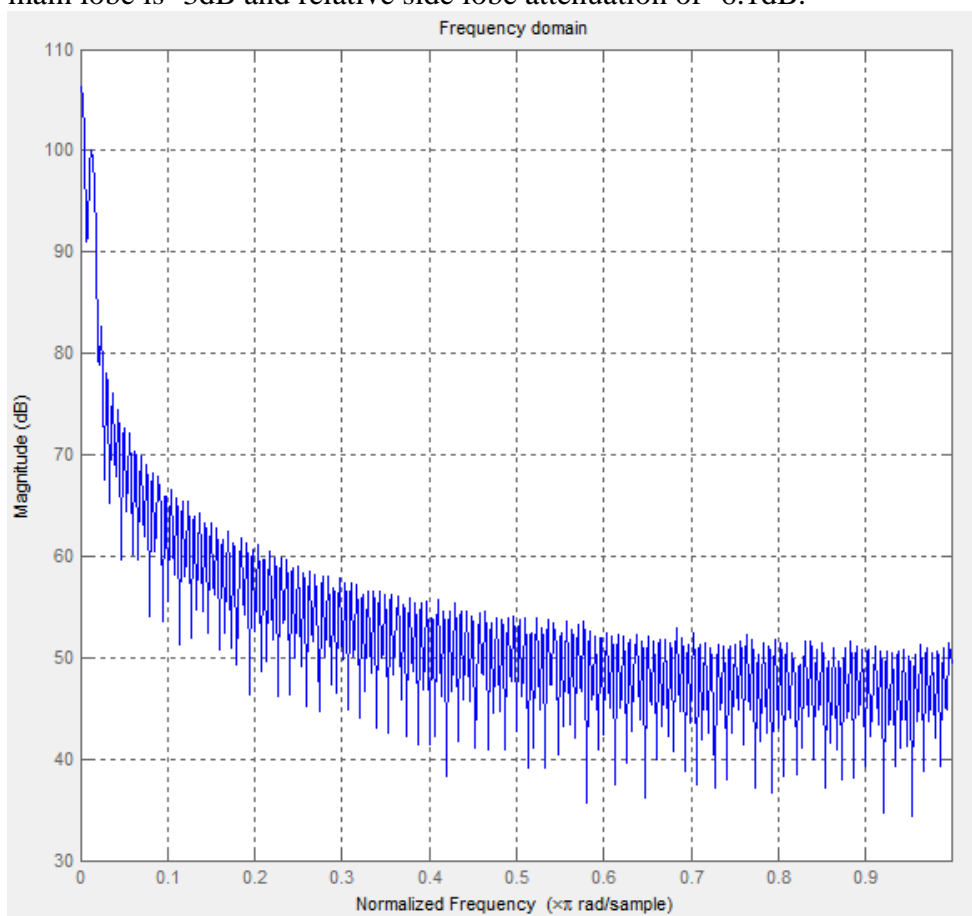


Figure 29. Normalized FFT.

### Signal to Noise Ratio (SNR)

SNR is used to characterize the quality of the signal of the analog digital converter. SNR is determined by many noise sources in addition to quantization noise. The analog digital converter's resolution and quantization level both help to establish its noise floor. The actual SNR for a 1KHz sinusoidal input signal can therefore be calculated in Matlab.

```
p= snr(signal,300*KHz,6)
SNR = 49.4103(dB)
```

### Signal-to-Noise Ratio + distortion (SINAD)

SINAD is defined as the ratio of signal plus noise plus distortion and noise plus distortion. SINAD is the ratio of the rms value of the measured input signal to rms sum of all other spectral component below nyquist frequency. The SINAD is computed using matlab

```
w= sinad(signal,300*KHz)
SINAD = 48.6854(dB)
```

### Total Harmonic Distortion (THD)

THD is the ratio of the rms sum of the first 9 harmonic components of the rms value of the measured input sinusoidal signal. The actual THD is calculated in matlab.

```
r= thd(signal,300*KHz,9)
r = -51.3(dB)
THD = -51.3(dB) = 0.0027%
```

### Effective Number of Bits (ENOB)

ENOB can be directly calculated as  $ENOB = (w - 1.76) / 6.02$   
ENOB = 7.7949 bits ~8bits

### Spurious-Free Dynamic Range (SFDR)

SFDR is the difference between the rms amplitude of the input signal and the RMS value of the largest spur observed in the frequency domain. SFDR is calculated in dB. This is a crucial specification that is used to characterize the dynamic performance of a signal generator The actual SFDR is calculated in matlab. SFDR = 53.4551(dB)

### 3 Tiva C Series - TM4C123GH6PM

The Tiva C Series TM4C123GH6PM is a low-cost evaluation platform from Texas Instruments (TI) [7]. This platform, together with the integrated development environment (IDE) Code Composer Studio (CCS) provides tools to develop and debug embedded applications with C/C++ programming. It has an ARM Cortex-M4 microcontroller that provides high performance and advanced integration [7].

The following are some of the features of the Tiva C Series - TM4C123GH6PM [14].

- Motion control PWM
- USB Micro-AB connector
- Device mode default configuration
- Host/OTG modes supportable
- RGB LED
- Two user switches (application/wake)
- Available I/O brought out to headers on a 0.1-in (2.54-mm) grid
- On-board ICDI
- Switch-selectable power sources:
- ICDI
- USB device
- Reset switch
- Preloaded RGB quick start application

The board is comes with USB library, the peripheral driver library and stackable headers to expand the capabilities.

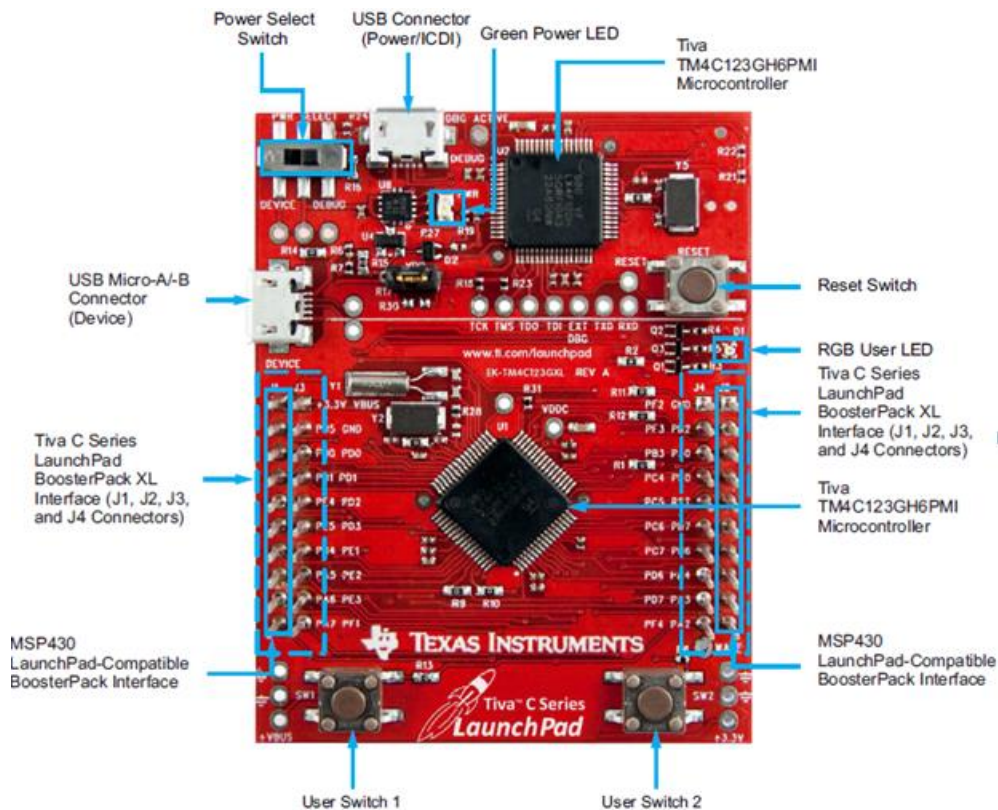


Figure 30. Tiva C board [15].

The Tiva C Series platform makes use of the most advanced ARM architecture core for MCUs, the Cortex-M4 is 32-bit processor architecture.

The block diagram below shows the architecture of the Tiva TM4C123x series.

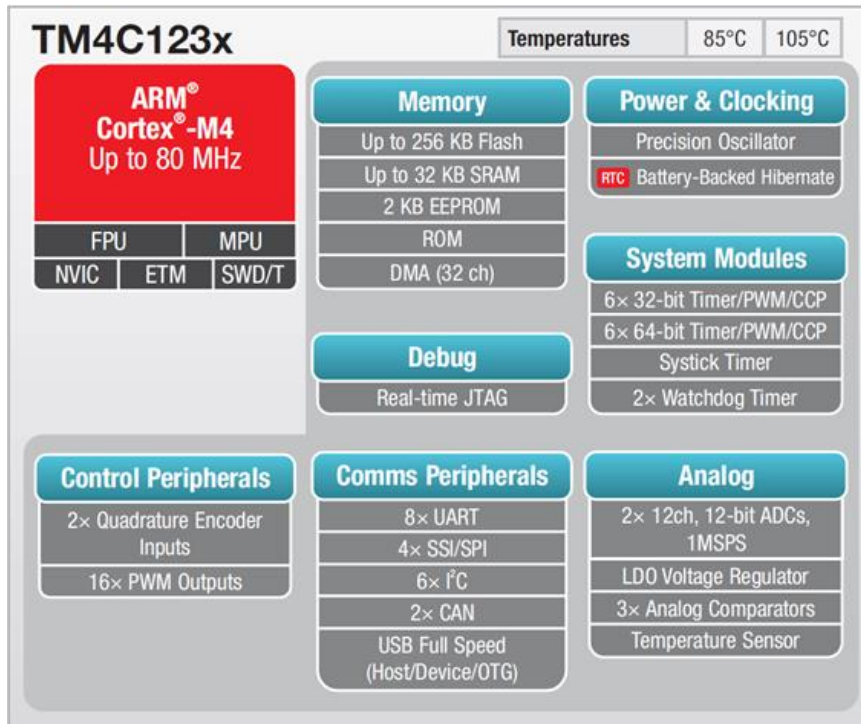


Figure 31. Architecture of Tiva TM4C123x series [15].

Tiva TM4C123GH6PM MCUs has two ADC modules (ADC0 and ADC1) that can be used to convert continuous analog voltages to discrete digital values. The ADC operates from both the 3.3V analog and 1.2V digital power supplies. Each ADC module is independent and has 12-bit resolution. The ADC is made up of 12 shared analog input channels and maximum sample rate of one million samples/second. Both modules have 8 digital comparators.

The block diagram below shows two ADC modules in a TM4C123GH6PM

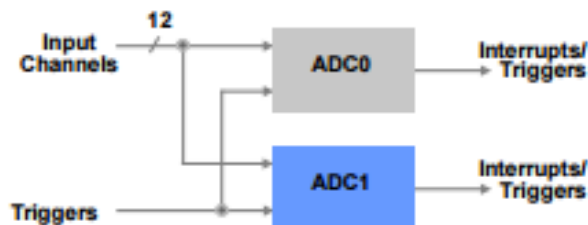


Figure 32. ADC modules in a TM4C123GH6PM [7].

### 3.1 Tiva ADC electrical characteristics

Parameter	Parameter Name	Min	Nom	Max	Unit
<b>POWER SUPPLY REQUIREMENTS</b>					
V <sub>DDA</sub>	ADC supply voltage	2.97	3.3	3.63	V
GNDA	ADC ground voltage	-	0	-	V
<b>VDDA / GNDA VOLTAGE REFERENCE</b>					
C <sub>REF</sub>	Voltage reference decoupling capacitance	-	1.0 // 0.01 <sup>c</sup>	-	μF
<b>ANALOG INPUT</b>					
V <sub>ADCI<sub>N</sub></sub>	Single-ended, full-scale analog input voltage, internal reference <sup>de</sup>	0	-	V <sub>DDA</sub>	V
	Differential, full-scale analog input voltage, internal reference <sup>df</sup>	-V <sub>DDA</sub>	-	V <sub>DDA</sub>	V
V <sub>INCM</sub>	Input common mode voltage, differential mode <sup>g</sup>	-	-	(V <sub>REFP</sub> + V <sub>REFN</sub> ) / 2 ± 25	mV
I <sub>L</sub>	ADC input leakage current <sup>h</sup>	-	-	2.0	μA
R <sub>ADC</sub>	ADC equivalent input resistance <sup>h</sup>	-	-	2.5	kΩ
C <sub>ADC</sub>	ADC equivalent input capacitance <sup>h</sup>	-	-	10	pF
R <sub>S</sub>	Analog source resistance <sup>h</sup>	-	-	500	Ω
<b>SAMPLING DYNAMICS</b>					
F <sub>ADC</sub>	ADC conversion clock frequency <sup>j</sup>	-	16	-	MHz
F <sub>CONV</sub>	ADC conversion rate	1			Msp/s
T <sub>S</sub>	ADC sample time	-	250	-	ns
T <sub>C</sub>	ADC conversion time <sup>j</sup>	1			μs
T <sub>LT</sub>	Latency from trigger to start of conversion	-	2	-	ADC clocks
<b>SYSTEM PERFORMANCE when using internal reference</b>					
N	Resolution	12			bits
INL	Integral nonlinearity error, over full input range	-	±1.5	±3.0	LSB
DNL	Differential nonlinearity error, over full input range	-	±0.8	+2.0/-1.0 <sup>k</sup>	LSB
E <sub>O</sub>	Offset error	-	±5.0	±15.0	LSB
E <sub>G</sub>	Gain error <sup>j</sup>	-	±10.0	±30.0	LSB
E <sub>T</sub>	Total unadjusted error, over full input range <sup>m</sup>	-	±10.0	±30.0	LSB
<b>DYNAMIC CHARACTERISTICS<sup>no</sup></b>					
SNR <sub>D</sub>	Signal-to-noise-ratio, Differential input, V <sub>ADCI<sub>N</sub></sub> : -20dB FS, 1KHz <sup>p</sup>	70	72	-	dB
SDR <sub>D</sub>	Signal-to-distortion ratio, Differential input, V <sub>ADCI<sub>N</sub></sub> : -3dB FS, 1KHz <sup>qf</sup>	72	75	-	dB
SNDR <sub>D</sub>	Signal-to-Noise+Distortion ratio, Differential input, V <sub>ADCI<sub>N</sub></sub> : -3dB FS, 1KHz <sup>rst</sup>	68	70	-	dB
SNR <sub>S</sub>	Signal-to-noise-ratio, Single-ended input, V <sub>ADCI<sub>N</sub></sub> : -20dB FS, 1KHz <sup>u</sup>	60	65	-	dB
SDR <sub>S</sub>	Signal-to-distortion ratio, Single-ended input, V <sub>ADCI<sub>N</sub></sub> : -3dB FS, 1KHz <sup>rf</sup>	70	72	-	dB
SNDR <sub>S</sub>	Signal-to-Noise+Distortion ratio, Single-ended input, V <sub>ADCI<sub>N</sub></sub> : -3dB FS, 1KHz <sup>stu</sup>	60	63	-	dB

Table 5. Electric characteristics of Tiva board [7].

It should be noted that analog input pads can handle voltages beyond 0 - 3.3V range but the analog input voltages must remain within the limits prescribed by Table 5 to produce accurate results.



## 3.2 Testing Tiva board

The Tiva C series board was also tested using the inbuilt projects that come with Tiva ware package using the CCS debugger. The Hello world example, blinking Led and temperature sensor examples were used to test the proper functioning of the board as well as the ADC. The figure 33 below shows sample output of the temperature readings from the board in Celsius (C) and Fahrenheit (F).

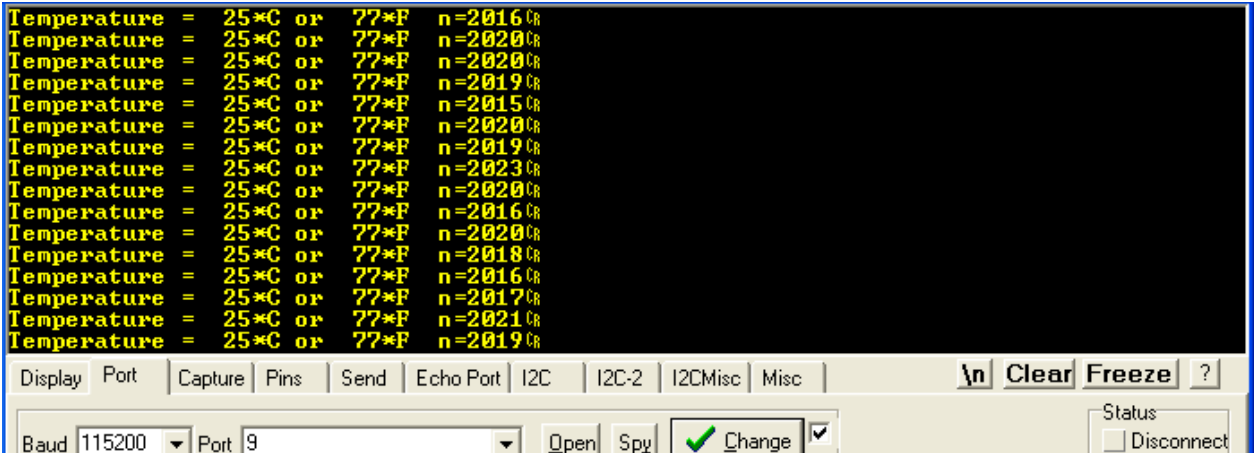
Below is a simple code for the temperature sensor in the tiva board.

```
ui32TempValueC = (2025 - (2475UL * pui32ADCValue[0]/4095) -550)/10;
//
// Get Fahrenheit value. Make sure you divide last to avoid dropout.
//
ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

// Display the temperature value on the console.
//
UARTprintf("Temperature = %3d*C or %3d*F n=%3d\r", ui32TempValueC,
           ui32TempValueF, pui32ADCValue[0]);
```

Figure 33. Program code for the temperature sensor in the Tiva board.

Output showing different temperatures in Celsius and Fahrenheit.



```
Temperature = 25*C or 77*F n=2016 C
Temperature = 25*C or 77*F n=2020 C
Temperature = 25*C or 77*F n=2020 C
Temperature = 25*C or 77*F n=2019 C
Temperature = 25*C or 77*F n=2015 C
Temperature = 25*C or 77*F n=2020 C
Temperature = 25*C or 77*F n=2019 C
Temperature = 25*C or 77*F n=2023 C
Temperature = 25*C or 77*F n=2020 C
Temperature = 25*C or 77*F n=2016 C
Temperature = 25*C or 77*F n=2020 C
Temperature = 25*C or 77*F n=2018 C
Temperature = 25*C or 77*F n=2016 C
Temperature = 25*C or 77*F n=2017 C
Temperature = 25*C or 77*F n=2021 C
Temperature = 25*C or 77*F n=2019 C
```

Figure 34. ADC results from temperature sensor.

### 3.3 ADC Static Evaluation of Tiva board.

Static performance parameters were tested for the Tiva board as well. The test is influenced by the DC component of the input voltage. The concept is to put one input voltage value and see 31 ADC output codes for each input. These parameters include gain error, offset error, full scale error inl, dnl. The block diagram below shows the set up used to test the Tiva C series board.

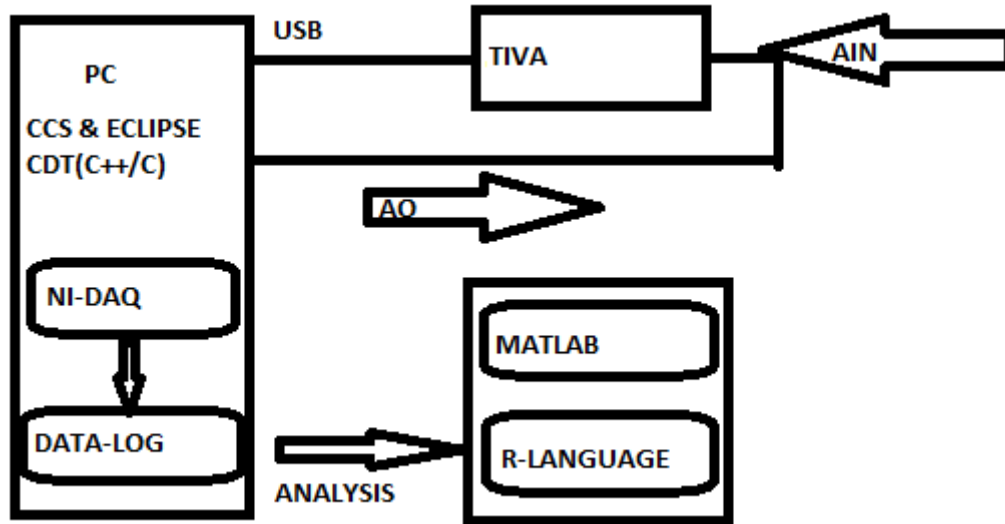


Figure 35. Test set up of Tiva C series board

A connection is established between the Tiva board and the PC via a USB connection. A program code is debugged and run in code composer studio. The code collects 31 samples from the ADC. Channel ADCAIN0 is connected to the national instrument data acquisition device. A program code that controls the input voltage is run in eclipse simultaneously with code composer studio. The data collected is sent to the data log and later analyzed in R-language and Mat lab/Simulink.

#### 3.3.1 Single ended Input

Single-ended input measure the voltage between the input channel (AIN0/PE3) and analog ground common to all in puts. The single ended input logs were collected and analyzed. In this test, single-ended input was connected with one wire from AIN0 to the data acquisition input interface and the other wire to the data acquisition input interface ground. In this test, it is assumed that the ground is at a constant 0V, but in reality the ground is at a different level in different places. The difference in levels can drive large currents, known as ground loops. This can lead to noise errors when using single-ended inputs. These noise errors are shown in the transfer function below.

The figure 36 below shows the transfer function of Tiva c series board.

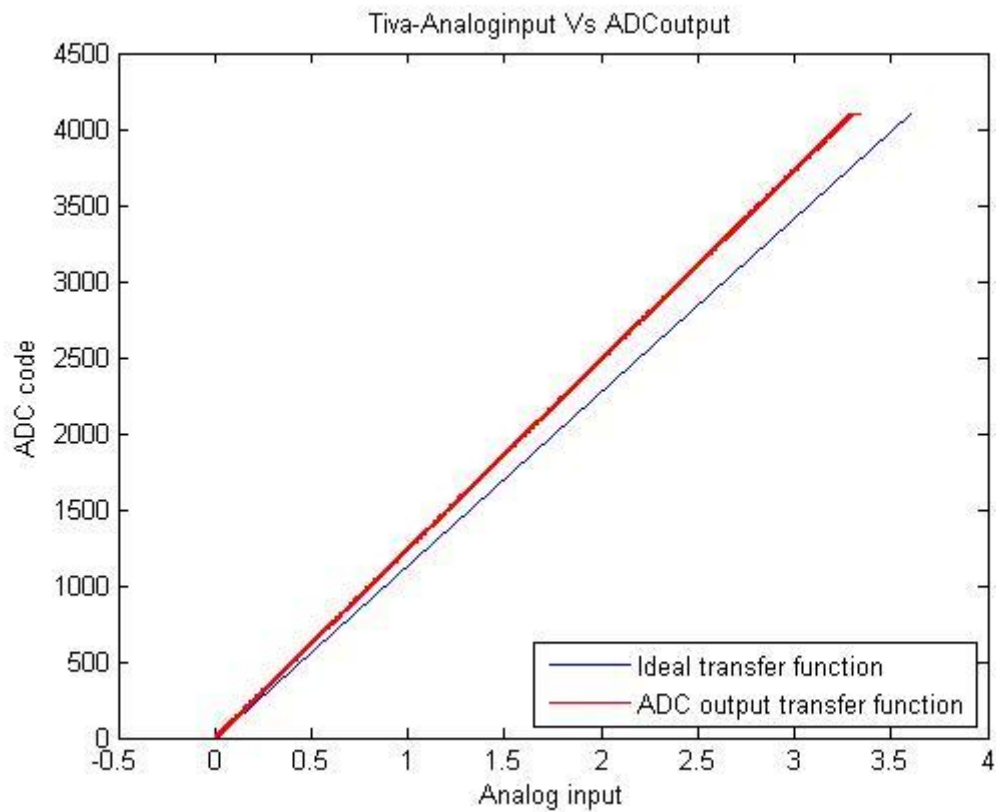


Figure 36. Transfer function of Tiva c series board.

The ADC transfer function shown in the figure 36 above doesn't look like the ideal transfer function. This is caused by the internal noise within the board and the external noise errors caused by grounding. The transfer function of an ADC is a plot of the voltage input to the ADC versus the output code's from the ADC. The plot is not continuous but is a plot of  $2^N$  codes, where  $N$  is the ADC's resolution in bits. In this case  $N=12$  bits giving us 4096 codes. The ideal transfer function plot is a straight line. The gain error is calculated by finding the difference in slope of the ideal transfer function and the ADC transfer function. In this case the gain error is positive with 52.6digits.

### 3.3.2 Noise Error

Single-ended inputs are sensitive to noise errors. Noise is unwanted signals picked up within the board and environmental electrical activity. Single-ended inputs make it hard to distinguishing between the signal and the noise. This is one of the reasons that affect the ADC output leading to a noisy transfer function. The transfer function cannot clearly show the noise across each individual code of the ADC. In this project analysis is done to see how the noise is spread across all the 4095 ADC codes.

R language script is used to analyze the data log.

```
plot (Uin, xdataAver, type='l', col= 'blue')
k1 = 4000 # code for "near-zero"
k2 =18000 # code for "near- full-scale"
Kscale= (xdataAver[k2] - xdataAver[k1]) / (Uin[k2]- Uin[k1]) # dCode/dUin
Kscale0= 4095/3.3
RelativeScale= Kscale /Kscale0
OffsetCode= xdataAver[k1]- Kscale *Uin[k1]
OffsetV= OffsetCode/Kscale
FullscaleV = OffsetV + 4095/Kscale
```

Figure 37. R language script used to analyse the data log.

In this test the full range (-0.05-3.6) V is divided into 2000 equal steps. The NI DAQ takes 31 samples at every voltage input. The figure 38 below shows the ADC output code and the analog input from 0 to full scale. The black color shows all the codes and the noise associated with each code. The blue color shows the average error code across the ADC output. The noise is evenly distributed across all codes.

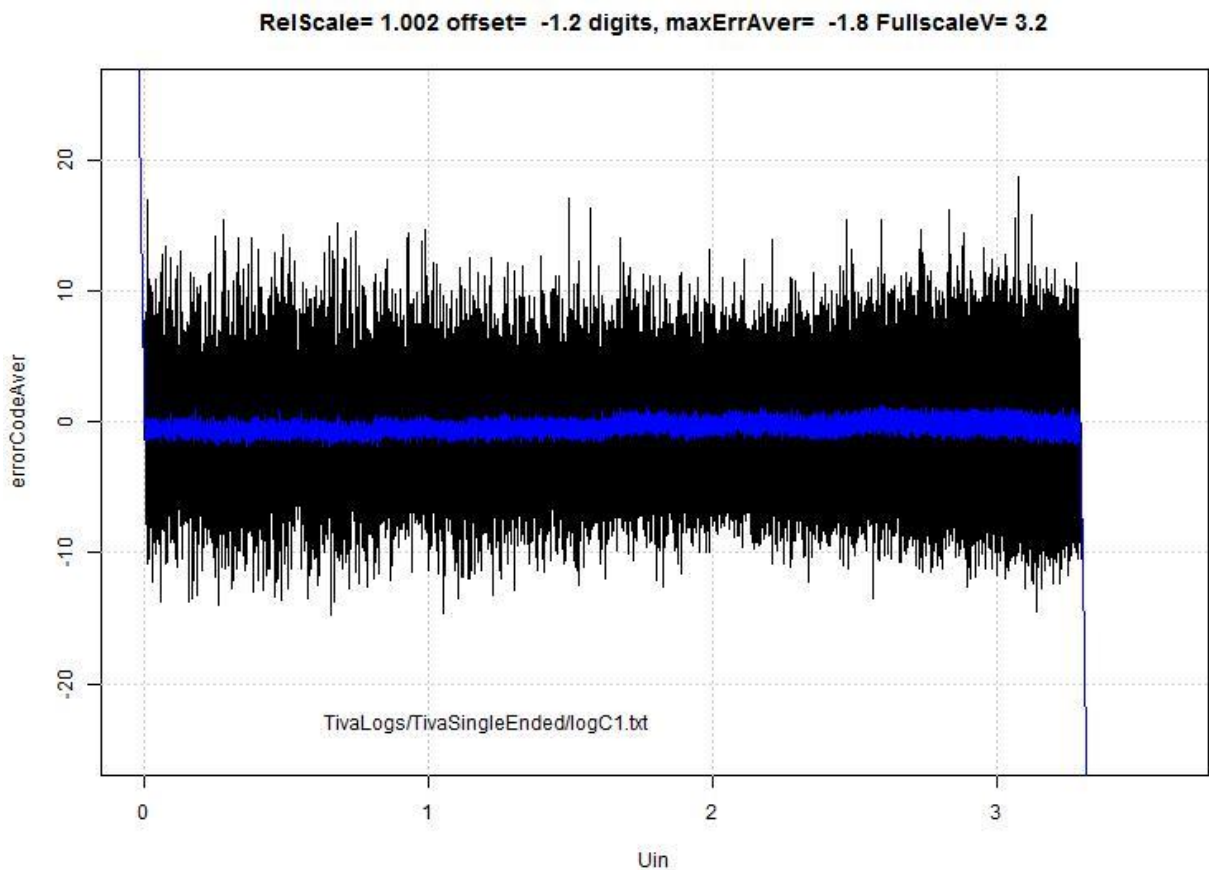


Figure 38. ADC output code and the analog input.

Four data logs were analyzed with 2000 analog samples and 31 ADC output codes. The table 6 below show different errors from the data analyzed from different logs.

	Offset error (digits)	Full Scale Error (digits)	Maximum Average Error (digits)	SD	Relative scale
LogA1	-2	-0.002	-1.3	1.23	1.002
LogB1	-2.8	-0.003	-2.1	1.22	1.003
LogC1	-1.2	-0.002	-1.8	1.26	1.002
LogD1	-1.9	-0.003	-1.5	1.18	1.002

Table 6. Analysed Tiva logs

### 3.3.3 Differential input

To solve the problem of noise error caused by single ended input, differential input is used. This measures the voltage between two individual inputs within a common mode range. The measurement is independent of the low level ground which makes it more immune to noise. The two wires in this case are both exposed to electromagnetic interference (EMI). The national instrument DAQ input measures only the difference in voltage between the two wires, and the EMI common to both is ignored. The code below is used to collect ADC samples in differential input mode.

```
// Select the analog ADC function for these pins.
// TODO: change this to select the port/pin you are using.
//GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_7);
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_7 | GPIO_PIN_6);

// Enable sample sequence 3 with a processor signal trigger. Sequence 3
// will do a single sample when the processor sends a signal to start the
// conversion. Each ADC module has 4 programmable sequences, sequence 0
// to sequence 3. This example is arbitrarily using sequence 3.
ADCSequenceConfigure(ADCO_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

// Configure step 0 on sequence 3. Sample channel 0 (ADC_CTL_CHO) in
// single-ended mode (default) and configure the interrupt flag
// (ADC_CTL_IE) to be set when the sample is done. Tell the ADC logic
// that this is the last conversion on sequence 3 (ADC_CTL_END). Sequence
// 3 has only one programmable step. Sequence 1 and 2 have 4 steps, and
// sequence 0 has 8 programmable steps. Since we are only doing a single
// conversion using sequence 3 we will only configure step 0. For more
// information on the ADC sequences and steps, reference the datasheet.
//ADCSequenceStepConfigure(ADCO_BASE, 3, 0, ADC_CTL_CHO | ADC_CTL_IE |
//ADC_CTL_END);
ADCSequenceStepConfigure(ADCO_BASE, 3, 0, ADC_CTL_D | ADC_CTL_CHO |
ADC_CTL_IE | ADC_CTL_END);
```

Figure 39. Program code for differential input mode.

A few lines of code were changed from the single ended input to sample the board in differential input mode.

### 3.3.4 Dithering

Dithering is the adding of a bit of white noise to the ADC output [7]. The dither bit is made active by adjusting the ADCCTL register which is used to reduce random noise in ADC sampling and keep the ADC operation within the specified performance limits. The dither bit should be enabled in the ADCCTL register along with hardware averaging in the ADC Sample Averaging Control (ADCSAC) register. It should be noted that the dither bit is disabled by default at reset. When dithering is introduced in the Tiva board the noise increases and the codes are not easily differentiated they are covered in a noise cloud. The figure 40 below shows the ADC output of a Tiva board with dithering.

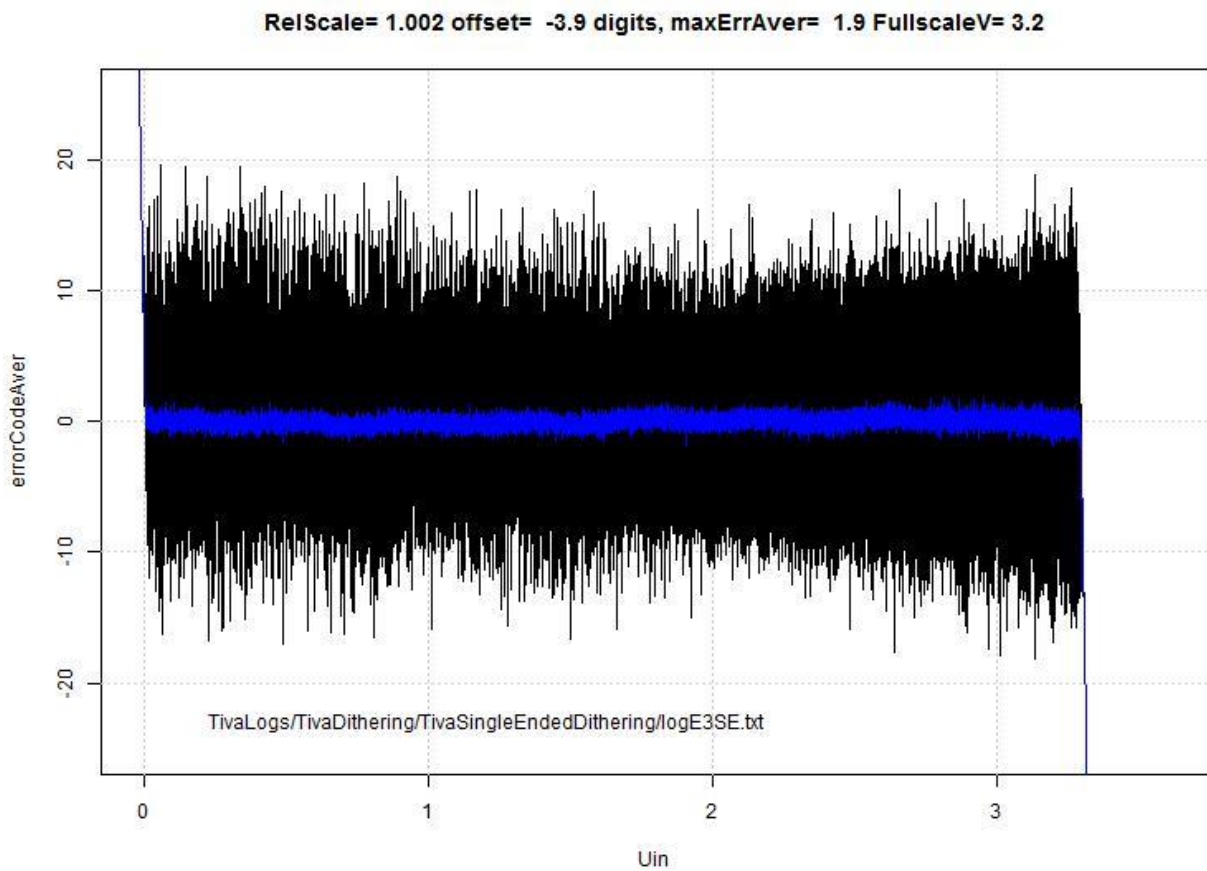


Figure 40. ADC output of a Tiva board with dithering.

### 3.3.5 Integral Nonlinearity (INL)

Integral nonlinearity is the deviation of individual code from the ideal straight line drawn from zero to full scale. At point zero, there is 0.5LSB before the first transition code and full scale occurs at one half LSB beyond the last code transition. To measure the deviation, one has to start from the center of each particular code to the ideal straight line between two points. The Tiva board integral nonlinearity is tested with the static ADC data logs collected using histogram method. In this method the first and last bin (codes 0 and 4095) are eliminated then we calculate the average /expected count of the codes. INL is the sum from the beginning up to current code (bin) of DNLs. The INL error is  $\pm 7$ -12 LSB.

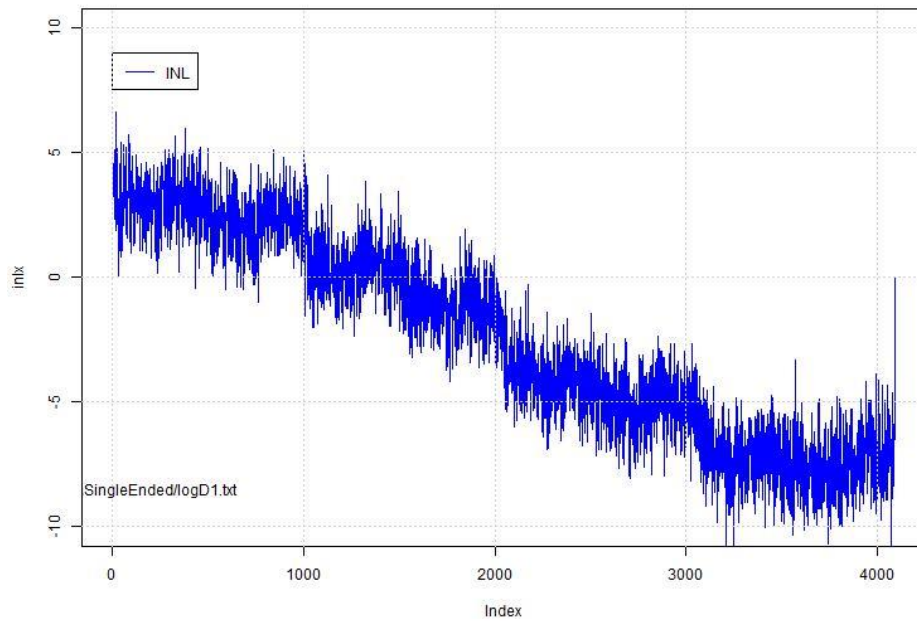


Figure 41. Tiva INL error.

### 3.3.6 Differential Nonlinearity (DNL)

In an ideal transfer function each code transition is 1LSB apart. In reality there are deviations from the ideal LSB value. A DNL error can be positive or negative i.e. +1LSB or -1LSB. The Tiva board DNL is tested using the collected data logs. This is done with the histogram method using the R-code code previously used to test the Tiva board. In this method the first and last bin (codes 0 and 4095) are eliminated then we calculate the average /expected count of the code(s). DNL is the difference at every bin (code) of actual count from average. The DNL is around  $\pm 3$  LSB.

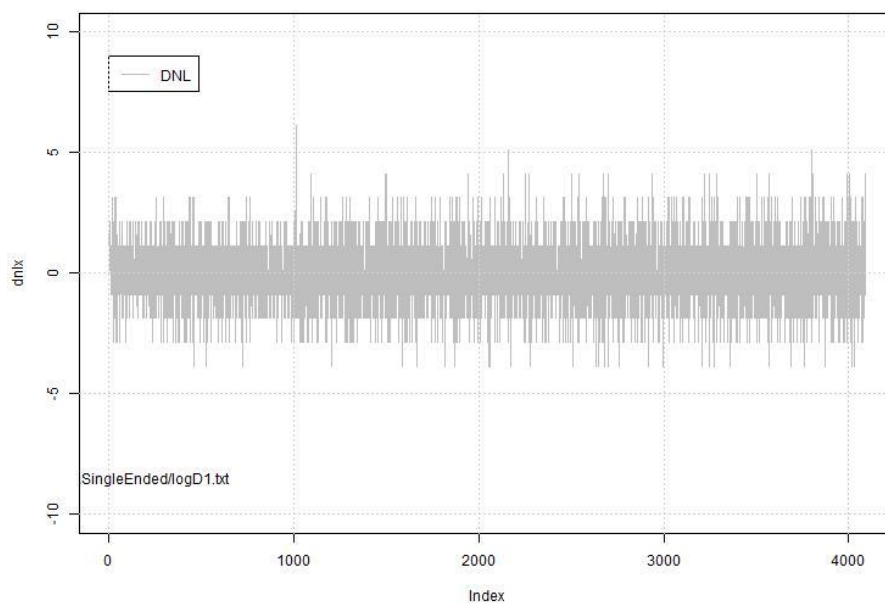


Figure 42. Tiva DNL error.

### 3.4 Dynamic Evaluation Tiva C board

The Tiva board is connected to a signal generator using two wires, one of the wires is connected to ground and the other is connected to the Tiva board peripheral pin PE3. The board is connected to a pc via a USB connection. The code used in the previous tests to trigger samples from the ADC is used to collect data logs of 3000 samples. The samples are collected using real terminal console. In this test a sinusoidal signal at different frequencies is used as the input of the on board ADC. The Sampling rate of the ADC is the maximum back-to-back conversion rate. If it's not modified then it is 1MSPS maximum rate. The board has two ADCs and each has a conversion rate of up to 1MSPS. This means that the board can archive a conversion rate of 2MSPS.

#### 3.4.1 Run-Mode Clock Gating Control 0 (RCGCO) register.

The RCGCO register is used to drive the sampling rate of the Tiva board [7]. This register determines the clock gating logic in normal run mode [7]. In other microcontrollers that are closely related to the Tiva microcontroller like Stellaris the ADC peripheral configuration register is used to determine the sample rate of the ADC. In this project the RCGCO is configured in code composer studio to control the sampling rate of the ADC. The ADC sample rate can be set to 125K (0x0), 250K (0x1), 500K(0x2), or 1M (0x3) samples/second. According to the Tiva micro controller datasheet the sample rate is set by configuring the MAXADCOSPD register bits. The SYSTCTL\_RCGCO\_ADCOSPD register is specifically used to set the ADC sample speed in this project work. A pointer is used to insert the desired sample rate value into the memory address of the register.

```
#define xSYSTCTL_RCGCO 0x400FE100 // SYSTCTL_RCGCO_R |= SYSTCTL_RCGCO_ADCSPD500K;

xxptr= xSYSTCTL_RCGCO ; // ADCCTL or ADC_CTL_R 0x38, dithering, Badru

xx32= *xxptr; // x0100 , 0x0200
xx32m= xx32 & 0x0300;

*xxptr= (xx32 + 0x0200);
```

Figure 43. Program code for sample rate register.

The system control delay also affects the sample rate. A change in the system delay will show significant change in the output signal.

```
// SysCtlDelay(SysCtlClockGet() / 12);
SysCtlDelay( 8); //100
```

When the changes have been made and saved in the code composer studio. The program is run. The table below shows the sample rate changed to 200Ksps.

1010 0101	SYSTCTL_RCGCO_ADC1SPD	00 -	ADC1 Sample Speed
1010 0101	SYSTCTL_RCGCO_ADC0SPD	10	ADC0 Sample Speed
....			

Figure 44. System control register.



The first sample data collected is used to analyze the behavior of the Tiva board. The sine wave used has a frequency of 1Khz and the sampling rate is about 200ksp. The FFT output spectrum of the signal is obtained by using the following Matlab code.

```
>> %Badru 2016
%Double-sided magnitude spectrum with frequency axis (in bins)
fax_bins = [0 : N-1]; %N is the number of samples in the signal
plot(fax_bins, abs(fft(signal)))
xlabel('Frequency (Bins)')
ylabel('Magnitude');
title('Double-sided Magnitude spectrum (bins)');
axis tight
```

Figure 45. Tiva sine wave FFT output spectrum.

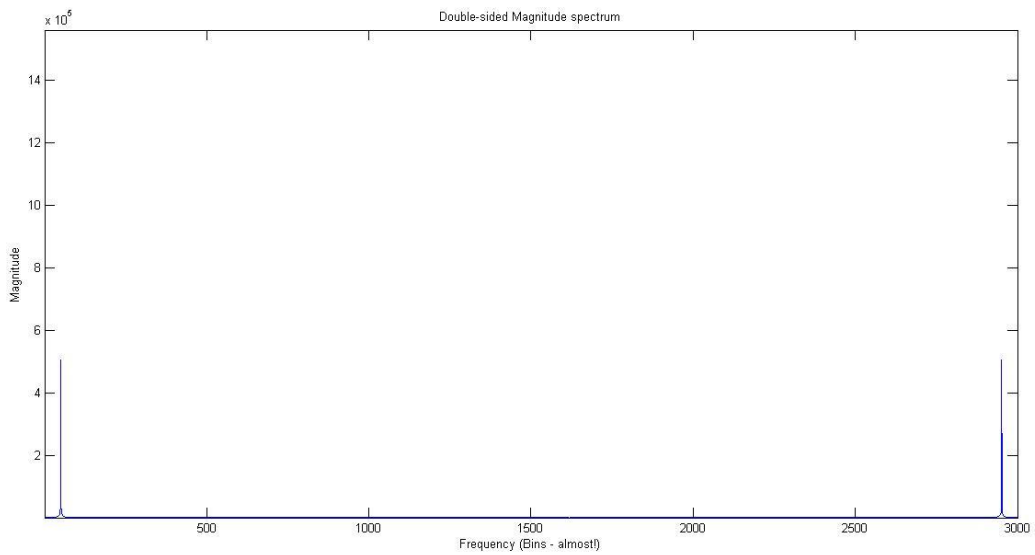


Figure 46. FFT spectrum of a sine signal.

The figure shows the fft spectrum of a sine signal with a frequency of 1KHz and sampled at a rate of 200ksp. After 1500 bins the spectrum is mirrored at 2000bins.

A comparison is made between the dynamic characteristics in the Tiva ADC data sheet and the results from analyzed data. The table below shows the dynamic characteristics of the Tiva board as given in the data sheet.

DYNAMIC CHARACTERISTICS <sup>no</sup>					
SNR <sub>D</sub>	Signal-to-noise-ratio, Differential input, V <sub>ADCIN</sub> : -20dB FS, 1KHz <sup>P</sup>	70	72	-	dB
SDR <sub>D</sub>	Signal-to-distortion ratio, Differential input, V <sub>ADCIN</sub> : -3dB FS, 1KHz <sup>pqr</sup>	72	75	-	dB
SNDR <sub>D</sub>	Signal-to-Noise+Distortion ratio, Differential input, V <sub>ADCIN</sub> : -3dB FS, 1KHz <sup>pst</sup>	68	70	-	dB
SNR <sub>S</sub>	Signal-to-noise-ratio, Single-ended input, V <sub>ADCIN</sub> : -20dB FS, 1KHz	60	65	-	dB

Table 7. Dynamic characteristics of the Tiva board.

Comparing the analysis done in this work and the dynamic characteristics of the Tiva board provided in the data sheet, a Matlab code was used to generate half of the spectrum of the signal in dB.

```
>> % badru 2016
signal = load('tt1.txt');
N = length(signal);
KHz=1e3;
fs = 200*KHz; % 200 samples per second
fnyquist = fs/2; %Nyquist frequency
Pk2Pk= max(signal) - min(signal);
rms= std(signal);
plot(signal);
>> wvtool(signal);
>> %Half magnitiude spectrum in decibels and KHz
X_mags = abs(fft(signal));
bin_vals = [0 : N-1];
fax_Hz = bin_vals*fs/N;
N_2 = ceil(N/2);
plot(fax_Hz(1:N_2), 10*log10(X_mags(1:N_2)))
xlabel('Frequency (KHz)')
ylabel('Magnititude (dB)');
title('Half Magnitude spectrum (1kHz) 200ksps,pk_pk= 1307,rms= 456.6');
```

Figure 47. Matlab code for half sided spectrum.

The figure below shows half of the spectrum in dB.

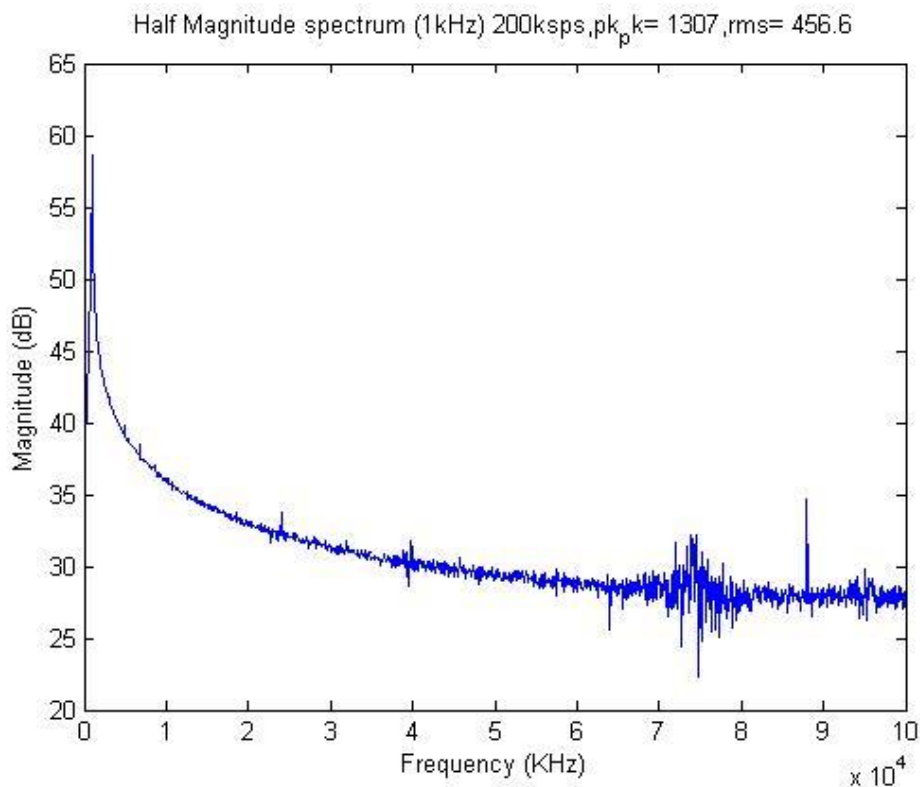


Figure 48. Single sided spectrum in dB.

## Windowing

Although performing an FFT on a signal can provide great insight, it is important to know the limitations of the FFT and how to improve the signal clarity using windowing. The Matlab code 47 is used to analyze the sinusoidal input using Blackman window.

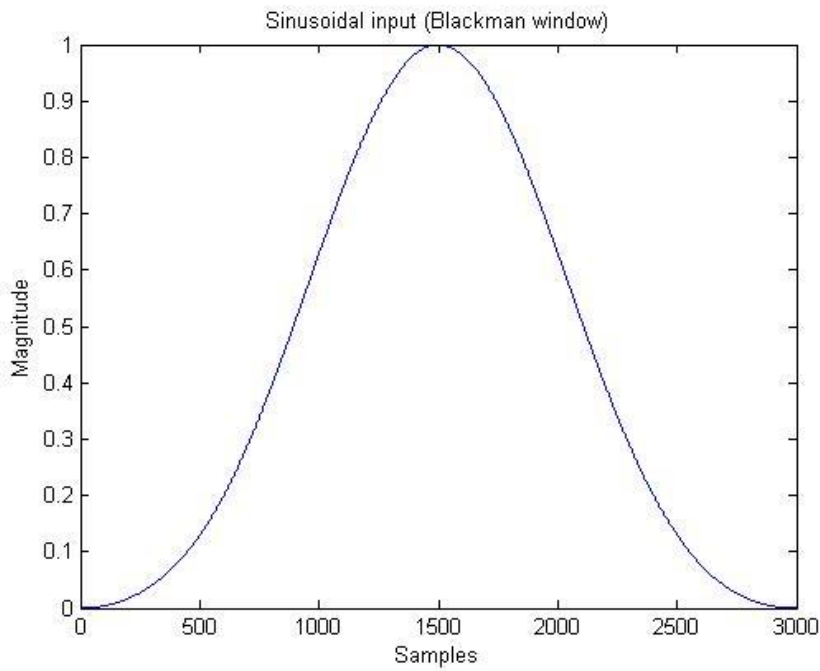


Figure 49. Blackman window Sine Input.

Figure below 50 shows FFT of the input signal frequency normalised in radians.

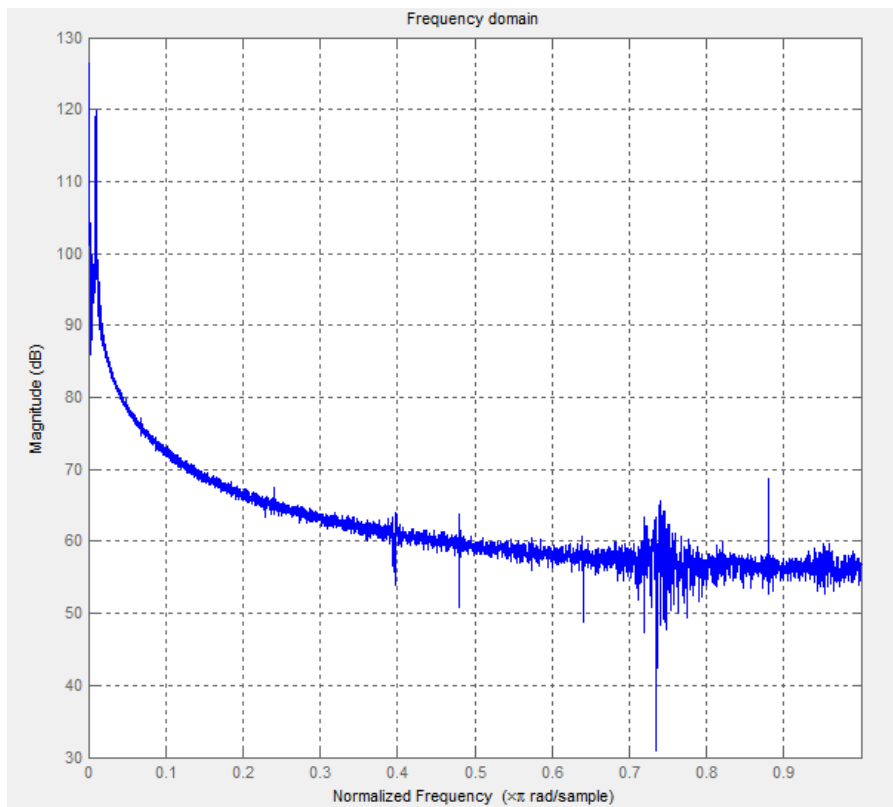


Figure 50. FFT using Hamming window .

The figure 46 above shows a clear FFT using Hamming window the main lobe width is -13.3dB and the relative side lobe -13.3dB.

#### Signal to Noise Ratio (SNR)

SNR is determined by many noise sources in addition to quantization noise. The analog digital converter's resolution and quantization level both help to establish its noise floor. The actual SNR for a 1KHz sinusoidal input signal can therefore be calculated in Matlab.

$p = \text{snr}(\text{signal}, 200 * \text{KHz}, 6)$  – Where 200Ksps is equal to the sampling frequency.

SNR = 43.5549(dB)

The SNR is equal to 43.5549dB. This SNR represents 66% of the ~65dB SNR exhibited by an ideal 12-bit ADC at 1KHz. As shown in the data sheet.

#### Total Harmonic Distortion (THD)

THD is the ratio of the rms sum of the first 5 harmonic components of the rms value of the measured input sinusoidal signal. The actual rms is calculated in mat lab.

$r = \text{thd}(\text{signal}, 200 * \text{KHz}, 5)$

THD = -55.0482 (dB).

THD is equal to -55.0482(dB) = 0.0017%

#### Signal to Noise Ratio + Distortion (SINAD)

SINAD is the ratio of the rms value of the measured input signal to rms sum of all other spectral component below nyquist frequency. The SINAD is computed using matlab.

$w = \text{sinad}(\text{signal}, 200 * \text{KHz})$

w = 43.2435(dB)

The SINAD is equal to 43.2435dB. This SINAD represents 68.6% of the ~63dB SINAD exhibited by an ideal 12-bit ADC at 1KHz. As shown in the data sheet.

#### Effective Number of Bits (ENOB)

ENOB can be directly calculated as  $\text{ENOB} = (\text{SINAD} - 1.76) / 6.02$ . ENOB= 8.9 bits. This is for the sinusoidal input of 1 KHz.

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

This is a crucial specification that is used to characterize the dynamic performance of a signal generator. SFDR determines the relationship between the amplitude of the input signal frequency being generated and the amplitude of the most prominent harmonic. In an ideal world, the frequency domain of a pure analog signal has all power concentrated at the desired frequency. However, due to noise and the nonlinearity of components, frequencies are generated at different harmonics.

$q = \text{sfdr}(\text{signal}, 1 * \text{KHz})$

q = 41.4803 (dB)

## 4 Results

All the goals, presented in section 1.1, for each of the two boards are achieved. An overview of the Piccolo (TMS320F28069) and Tiva (TM4C123GH6PM) micro controllers was done in chapter 1. The dynamic and static parameters affecting the performance of the ADC of the two boards mentioned above are discussed and analysed in chapters 2 and 3. In chapter 4 the results were presented in tabular form in comparison with those presented in the data sheets of both boards.

### 4.1 Piccolo (TMS320F28069) results.

Offset error.

Maintaining the default sample and hold window at 6, x80 frequency the offset error is shown in the table 8 below.

The offset error in the Piccolo (TMS320F28069) data sheet is indicated as follows.

Parameter	Minimum	Maximum	Units
Offset Error		10	LSB

Table 8. Data sheet offset error.

The results obtained for the test conducted show the offset error as follows. Maintaining the default sample cycles and frequency (Log D).

Parameters	Minimum	Maximum	Units
Offset Error		5.6	digits

Table 9. Actual Offset error.

The offset error obtained in the test of the piccolo board is a round 5.6 digits ~ 6 digits.

Overall performance results from all the data logs collected.

	Sample and hold	Frequency (Hexa decimal)	Offseterror (digits)	Full scale error(digits)	Maximum Average error (digits)	SD
Log D	6	X80	5.6	0.031	4.7	1.39
Log E	13	X 28	-17	0.017	-2.3	1.41
Log F	27	X100	-12	0.019	1.8	1.40
Log G	57	X100	-13	0.019	-2.1	1.20

Table 10. Piccolo Test results

The results in table 10 above were obtained from four data logs with different sample and hold window bits at different frequency. The default sample cycles in log D S/H value and frequency are maintained in comparison to the datasheet. It's observed that the sample and hold window has so much effect on the offset and gain error and on the stability performance of the ADC result e.g. log D and log E.

Looking at the piccolo (TMS320F28069) data sheet the INL and DNL errors are presented as follows.

Parameter	Minimum	Maximum	Unit
Integral Nonlinearity	-2	+2	LSB
Differential Nonlinearity	-1	+1	LSB

Table 11. Piccolo INL/DNL (Datasheet).

From the results obtained from the tests done in this work the data from one log is presented in the table 12 below.

Parameters	Minimum	Maximum	Units
Integral Nonlinearity	-10	6	LSB
Differential Nonlinearity	-4	4	LSB

Table 12. Piccolo INL/DNL test results.

INL/DNL results from the four data logs.

Data logs	Integral Nonlinearity	Differential Nonlinearity	Units
Log D	+6/-10	+4/-4	LSB
Log E	+10/-7	+4/-4	LSB
Log F	+7/-4	+4/-4	LSB
Log G	+6/-5	+4/-5	LSB

Table 13. Piccolo Data logs (INL/DNL).

The piccolo (TMS320F28069) data sheet does not clearly specify the dynamic parameters of the board.

During the test for the dynamic performance of the piccolo board the following results were obtained.

Parameter	Value	Units
Signal to Noise Ratio (SNR)	49.4103	dB
Total Harmonic Distortion (THD)	-51.37 (0.027%)	dB/%
Spurious Free Dynamic Range (SFDR)	53.4551	dB
Effective Number Of Bits (ENOB)	7.7949	Bits
Signal to Noise Ratio + Distortion (SINAD)	48.6854	dB

Table 14. Dynamic parameters of piccolo (TMS320F28069) .

## 4.2 Tiva C (TM4C123GH6PM) results

The results obtained during the test of the ADC in the Tiva board are compared with what is written in the data sheet. The data sheet doesn't clearly state the values regarding the static parameters. Looking at the Tiva (TM4C123GH6PM) test results the static performance errors are presented as follows.

	Offset error (digits)	Full Scale Error (digits)	Maximum Average Error (digits)	SD	Relative scale
LogA1	-2	-0.002	-1.3	1.23	1.002
LogB1	-2.8	-0.003	-2.1	1.22	1.003
LogC1	-1.2	-0.002	-1.8	1.26	1.002
LogD1	-1.9	-0.003	-1.5	1.81	1.002

Table 15. Tiva static results.

Four logs were tested and the average offset error is -0.49 digits and variation of the average full scale error -0.0006 digits.

The results obtained during the test of the ADC in the Tiva board are compared with what is written in the data sheet. Looking at the Tiva (TM4C123GH6PM) data sheet INL/DNL performance errors are presented as follows.

Parameters	Minimum	Maximum	Units
Integral Nonlinearity (INL)	-/+0.8	+2.0/-1.0	LSB
Differential Nonlinearity (DNL)	-/+1.5	-/+ 3.0	LSB

Table 16. Tiva INL/DNL error (Data sheet).

The following results were obtained when a test for static parameters was made for Tiva board.

Parameters	Minimum	Maximum	Units
Integral Nonlinearity (INL)	-3	+3	LSB
Differential Nonlinearity (DNL)	-3	4	LSB

Table 17. Tiva INL/DNL error (Test results).

The table 18 below shows the INL/DNL results from four data logs.

Data logs	Integral Nonlinearity	Differential Nonlinearity	Units
Log A1	+6/-12	+4/-3	LSB
Log B1	+3/-3	+4/-3	LSB
Log C1	+4/-4	+4/-4	LSB
Log D1	+7/-4	+6/-4	LSB

Table 18. Data logs INL/DNL errors.

Looking at the Tiva (TM4C123GH6PM) data sheet the Dynamic performance errors are presented as follows.

Parameter	Minimum	Normal	Units
Signal to Noise Ratio (SNR)	60	65	dB
Signal to Noise Ratio + Distortion (SINAD)	60	63	dB

Table 19. Dynamic performance written in data sheet.

The following results were obtained when a test for dynamic parameters was made for Tiva board.

Parameter	Minimum	Normal	Units
Signal to Noise Ratio (SNR)		43.55	dB
Signal to Noise Ratio + Distortion (SINAD)		43.243	dB
THD		-54.7 (0.017%)	dB/%
ENOB		8.9	Bits
SFDR		41.48	dB

Table 20. Test results for dynamic performance.

From the table 14 test results the SNR and SINAD appear to be lower than those written in the data sheet.

### 4.3 Future work

The two boards tested in this project still have a few improvements or tests to be carried out. The first thing to improve if continuing with this project would be to test the Tiva board with a differential input and dither bit active. In the current test, this is not implemented. A few tests were done with the dither bit active but the results obtained were strange. So any one continuing with the test of these boards can look into that.

### 4.4 Conclusion

All logs collected are noisy. This is not caused by the chip but the entire board. The USB connection and the NI DAQ connection also contribute to the noise generation. The noise in the Tiva board can be reduced to a greater extent using the differential input method to evaluate the ADC performance. The Piccolo control stick noise can also be improved by using a different input pin as analog ground and avoiding the digital ground.

This project work should have provided some guidance on how to look at general static and dynamic ADC inaccuracy errors for the Piccolo (TMS320F28069) and Tiva C (TM4C123GH6PM) micro controllers. A greater understanding of ADC errors, how



these errors influence the ADC performance of the Piccolo (TMS320F28069) and Tiva C (TM4C123GH6PM) is very vital for engineers planning to use these boards in various applications. A comparison has been made between the information written in the data sheet and what actually happens when the boards are tested in real time. A deviation in the parameters can affect the performance of prototype designs created by engineers when tested. This can be in form of lower performance of the ADC as compared to what is written in the datasheet.

## Reference

- [1] Walt Kester, "Analog-digital conversion," 2015.
- [2] Walt Kester, "Which ADC architecture is right for your application," in *EDA Tech Forum*, vol. 2, 2005, pp. 22-25.
- [3] Ying Bai, "ARM Microcontroller Development Kits".
- [4] N Senthil Kumar, M Saravanan, and S Jeevananthan, *Microprocessors and Microcontrollers.*: Oxford University Press, Inc., 2011.
- [5] Bin Le, Thomas W Rondeau, Jeffrey H Reed, and Charles W Bostian, "Analog-to-digital converters," *Signal Processing Magazine, IEEE*, vol. 22, no. 6, pp. 69-77, 2005.
- [6] Robert H Walden, "Analog-to-digital converter survey and analysis," *Selected Areas in Communications, IEEE Journal on*, vol. 17, no. 4, pp. 539-550, 1999.
- [7] Texas Instruments, "Tiva™ TM4C123GH6PM Microcontroller," *TM4C123GH6PM datasheet*, 2007.
- [8] Texas Instruments, "'TMS320F2806X- Piccolo Microcontrollers," *available online: <http://www.ti.com/product/tms320f28069>*.
- [9] MATLAB User's Guide, "The mathworks," *Inc., Natick, MA*, vol. 5, p. 333, 1998.
- [10] Brian D Ripley, "The R project in statistical computing," *MSOR Connections. The newsletter of the LTSN Maths, Stats \& OR Network*, vol. 1, no. 1, pp. 23-25, 2001.
- [11] Danila Piatov, Andrea Janes, Alberto Sillitti, and Giancarlo Succi, "Using the Eclipse C/C++ Development Tooling as a Robust, Fully Functional, Actively Maintained, Open Source C++ Parser.," *OSS*, vol. 378, p. 399, 2012.
- [12] Franco Maloberti, *Data converters.*: Springer Science \& Business Media, 2007.
- [13] Katie Enderle, "TMS320F2802x/TMS320F2803x to TMS320F2806x Migration Overview," 2011.
- [14] Myke Predko, *Handbook of microcontrollers.*: McGraw-Hill, Inc., 1998.
- [15] C Tiva, "Series TM4C123G LaunchPad Evaluation Kit User's Manual, 15 Apr 2013," *SPMU296*.
- [16] Leon S. Sterling, *The Art of Agent-Oriented Modeling*. London: The MIT Press, 2009.
- [17] Alex Tessarolo, "Application report F2810, F2811, and F2812 ADC Calibration," *Texas Instruments*, 2004.
- [18] Raymond B Ridley, "A new, continuous-time model for current-mode control [power convertors]," *Power Electronics, IEEE Transactions on*, vol. 6, no. 2, pp. 271-280, 1991.
- [19] Priyesh Pandya and Vikas Gupta, "Enhancing analog to digital converter resolution

- using oversampling technique," *Int. J. Innovative Sci. Mod. Eng*, vol. 2, no. 5, pp. 37-40, 2014.
- [20] Vilmos P, Tam, and Istv, "Full information ADC test procedures using sinusoidal excitation, implemented in MATLAB and LabVIEW," *ACTA IMEKO*, vol. 4, no. 3, pp. 4-13, 2015.
- [21] Vilmos P, Tam, and Istv, "Full information ADC test procedures using sinusoidal excitation, implemented in MATLAB and LabVIEW," *ACTA IMEKO*, vol. 4, no. 3, pp. 4-13, 2015.
- [22] Todd D Morton, *Embedded microcontrollers.:* Prentice Hall PTR, 2000.
- [23] Mary McCarthy, "Peak-to-peak resolution versus effective resolution," *Application Note AN-615. Analog Device Inc*, 2003.
- [24] ADC Maxim, "DAC glossary," Application note 641,>
- [25] Paul Logsdon and Ian Bonthron, "Digital Bat Ears," 2014.
- [26] Jipeng Li and Un-Ku Moon, "Background calibration techniques for multistage pipelined ADCs with digital redundancy," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 50, no. 9, pp. 531-538, 2003.
- [27] Bin Le, Thomas W Rondeau, Jeffrey H Reed, and Charles W Bostian, "Analog-to-digital converters," *Signal Processing Magazine, IEEE*, vol. 22, no. 6, pp. 69-77, 2005.
- [28] Istv and Jerome J Blair, "Improved determination of the best fitting sine wave in ADC testing," *Instrumentation and Measurement, IEEE Transactions on*, vol. 54, no. 5, pp. 1978-1983, 2005.
- [29] Walt Kester, "ADC input noise: the good, the bad, and the ugly. Is no noise good noise?," *Analog Dialogue*, vol. 40, no. 02, pp. 1-5, 2006.
- [30] Texas Instruments, "TMS320x280x, 2801x, 2804x enhanced pulse width modulator (ePWM) module," *Texas Instruments Inc., Dallas, Texas*, 2004.
- [31] Texas Instruments, "TMS320x280x, 2801x, 2804x enhanced pulse width modulator (ePWM) module," *Texas Instruments Inc., Dallas, Texas*, 2004.
- [32] Levi J Hargrove, Kevin Englehart, and Bernard Hudgins, "A comparison of surface and intramuscular myoelectric signal classification," *Biomedical Engineering, IEEE Transactions on*, vol. 54, no. 5, pp. 847-853, 2007.
- [33] BOB Flaviu Ilie, Nicolae Cristian PAMPU, and Liviu Teodor CHIRA, "Improving analog-to-digital converter's resolution using the oversampling technique," 2011.
- [34] Divya Chacko and Mrs Kanchan Chavan, "Time to Amplitude Converter for Phase Shift Detection".
- [35] M Bossche, J Schoukens, and J Renneboog, "Dynamic testing and diagnostics of A/D converters," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 8, pp. 775-785, 1986.

## Appendix 1 - Program Codes

This thesis was controlled to a great extent by the programming involved in both the collection and analysis of the data. With the data recorded, every subsequent stage of the test, analysis and presentation of the data was done within the Code Composer Studio, Matlab, Rlanguage and C++ programming language in Eclipse CDT environment.

## Appendix 2 - Blinking LED code

```

//*****

//blinky.c - Simple example to blink the on-board LED.

// Copyright (c) 2011-2015 Texas Instruments Incorporated. All rights reserved.
// Software License Agreement
// Modified Badru 2016.
// #include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
//*****
//! \addtogroup example_list
//! <h1>Blinky (blinky)</h1>
//! A very simple example that blinks the on-board LED.
//*****
// Blink the on-board LED.
//*****

int
main(void)
{
    volatile uint32_t ui32Loop;
    // Enable the GPIO port that is used for the on-board LED.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    // Check if the peripheral access is enabled.

```

```

//
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG))
{
}
// Enable the GPIO pin for the LED (PG2). Set the direction as output, and
// enable the GPIO pin for digital function.
//
GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2);
// Loop forever.
//
while(1)
{
    // Turn on the LED.
    //
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, GPIO_PIN_2);
    // Delay for a bit.
    //
    for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)
    {
    }
    // Turn off the LED.
    //
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);
    // Delay for a bit.
    for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)
}

```

## Appendix 3 - Hello World program code

```
/**
 *
 */
// hello.c - Simple hello world example.
//
// Copyright (c) 2011-2015 Texas Instruments Incorporated. All rights reserved.
// Software License Agreement
// Modified Badru 2016
/**
 *
 */
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/fpu.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "gplib/gplib.h"
#include "drivers/cfal96x64x16.h"
#include "utils/uartstdio.h"
#include "driverlib/gpio.h"

/**
 *
 */
/*! \addtogroup example_list
 *! <h1>Hello World (hello)</h1>
 *! A very simple ``hello world" example. It simply displays ``Hello World!"
 *! on the display and is a starting point for more complicated applications.
 *! This example uses calls to the TivaWare Graphics Library graphics
 *! primitives functions to update the display. For a similar example using
 *! widgets, please see ``hello_widget".
 */
/**
 *
 */
// The error routine that is called if the driver library encounters an error.
/**
 *
 */
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

/**
 *
 */
// Configure the UART and its pins. This must be called before UARTprintf().
/**
 *
 */
void
ConfigureUART(void)
{
    // Enable the GPIO Peripheral used by the UART.
    //

```

```

ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
// Enable UART0
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

// Configure GPIO Pins for UART mode.
ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

// Use the internal 16MHz oscillator as the UART clock source.
UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
// Initialize the UART for console I/O.
UARTStdioConfig(0, 115200, 16000000);
}
//*****
// Print "Hello World!" to the display.
//*****
int
main(void)
{
    tContext sContext;
    tRectangle sRect;
    // Enable lazy stacking for interrupt handlers. This allows floating-point
    // instructions to be used within interrupt handlers, but at the expense of
    // extra stack usage.
    //
    ROM_FPULazyStackingEnable();
    // Set the clocking to run directly from the crystal.
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL |
SYSCTL_XTAL_16MHZ |
        SYSCTL_OSC_MAIN);
    // Initialize the UART.
    //
    ConfigureUART();

    UARTprintf("Hello, world!\n");
    // Initialize the display driver.
    //
    CFAL96x64x16Init();
    // Initialize the graphics context.
    //
    GrContextInit(&sContext, &g_sCFAL96x64x16);
    // Fill the top 24 rows of the screen with blue to create the banner.
    //
    sRect.i16XMin = 0;
    sRect.i16YMin = 0;
    sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
    sRect.i16YMax = 23;
    GrContextForegroundSet(&sContext, ClrDarkBlue);

```

```

GrRectFill(&sContext, &sRect);
// Put a white box around the banner.

GrContextForegroundSet(&sContext, ClrWhite);
GrRectDraw(&sContext, &sRect);
// Put the application name in the middle of the banner.
GrContextFontSet(&sContext, g_psFontCm12);
GrStringDrawCentered(&sContext, "hello", -1,
                    GrContextDpyWidthGet(&sContext) / 2, 10, 0);
// Say hello using the Computer Modern 40 point font.
GrContextFontSet(&sContext, g_psFontCm12/*g_psFontFixed6x8*/);
GrStringDrawCentered(&sContext, "Hello World!", -1,
                    GrContextDpyWidthGet(&sContext) / 2,
                    ((GrContextDpyHeightGet(&sContext) - 24) / 2) + 24,
                    0);
// Flush any cached drawing operations.
GrFlush(&sContext);
// We are finished. Hang around doing nothing.
//
while(1)
{
}
}

```



## Appendix 4 - ADC Differential sampling code

```
/**
 * differential.c - Example demonstrating how to configure the ADC for
 * differential operation.
 *
 * Copyright (c) 2010-2015 Texas Instruments Incorporated. All rights reserved.
 * Software License Agreement
 * Modified 2016
 */
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
/**
 * \addtogroup adc_examples_list
 * <h1>Differential ADC (differential)</h1>
 * ! This example shows how to setup ADC0 as a differential input and take a
 * ! single sample between AIN0 and AIN1. The value of the ADC is read and
 * ! printed to the serial port.
 * ! This example uses the following peripherals and I/O signals. You must
 * ! review these and change as needed for your own board:
 * ! - ADC0 peripheral
 * ! - GPIO Port E peripheral (for ADC0 pins)
 * ! - AIN0 - PE7
 * ! - AIN1 - PE6
 * ! The following UART signals are configured only for displaying console
 * ! messages for this example. These are not required for operation of the
 * ! ADC.
 * ! - UART0 peripheral
 * ! - GPIO Port A peripheral (for UART0 pins)
 * ! - UART0RX - PA0
 * ! - UART0TX - PA1
 * ! This example uses the following interrupt handlers. To use this example
 * ! in your own application you must add these interrupt handlers to your
 * ! vector table.
 * ! - None.
 */
// This function sets up UART0 to be used for a console to display information
// as the example is running.

void
```

```

InitConsole(void)
{
    // Enable GPIO port A which is used for UART0 pins.
    // TODO: change this to whichever GPIO port you are using.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // Configure the pin muxing for UART0 functions on port A0 and A1.
    // This step is not necessary if your part does not support pin muxing.
    // TODO: change this to select the port/pin you are using.
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    // Enable UART0 so that we can configure the clock.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    // Use the internal 16MHz oscillator as the UART clock source.
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    // Select the alternate (UART) function for these pins.
    // TODO: change this to select the port/pin you are using.
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    // Initialize the UART for console I/O.
    UARTStdioConfig(0, 115200, 16000000);
}

/*****
// Configure ADC0 for a differential input and a single sample. Once the
// sample is ready, an interrupt flag will be set. Using a polling method,
// the data will be read then displayed on the console via UART0.
*****/

int
main(void)
{
    #if defined(TARGET_IS_TM4C129_RA0) || \
        defined(TARGET_IS_TM4C129_RA1) || \
        defined(TARGET_IS_TM4C129_RA2)
        uint32_t ui32SysClock;
    #endif

    // This array is used for storing the data read from the ADC FIFO. It
    // must be as large as the FIFO for the sequencer in use. This example
    // uses sequence 3 which has a FIFO depth of 1. If another sequence
    // was used with a deeper FIFO, then the array size must be changed.
    //
    uint32_t pui32ADC0Value[1];
    // Set the clocking to run at 20 MHz (200 MHz / 10) using the PLL. When
    // using the ADC, you must either use the PLL or supply a 16 MHz clock
    // source.
    // TODO: The SYSCTL_XTAL_ value must be changed to match the value of the
    // crystal on your board.
    //
    #if defined(TARGET_IS_TM4C129_RA0) || \
        defined(TARGET_IS_TM4C129_RA1) || \
        defined(TARGET_IS_TM4C129_RA2)
        ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |

```

```

        SYSCTL_OSC_MAIN |
        SYSCTL_USE_PLL |
        SYSCTL_CFG_VCO_480), 20000000);
#else
    SysCtlClockSet(SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL |
SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_16MHZ);
#endif
// Set up the serial console to use for displaying messages. This is just
// for this example program and is not needed for ADC operation.
//
InitConsole();
// Display the setup on the console.
//
UARTprintf("ADC ->\n");
UARTprintf(" Type: differential\n");
UARTprintf(" Samples: One\n");
UARTprintf(" Update Rate: 250ms\n");
UARTprintf(" Input Pin: (AIN0/PE7 - AIN1/PE6)\n\n");
// The ADC0 peripheral must be enabled for use.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
// For this example ADC0 is used with AIN0/1 on port E7/E6.
// The actual port and pins used may be different on your part, consult
// the data sheet for more information. GPIO port E needs to be enabled
// so these pins can be used.
// TODO: change this to whichever GPIO port you are using.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
// Select the analog ADC function for these pins.
// Consult the data sheet to see which functions are allocated per pin.
// TODO: change this to select the port/pin you are using.
//
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_7 | GPIO_PIN_6);
// Enable sample sequence 3 with a processor signal trigger. Sequence 3
// will do a single sample when the processor sends a signal to start the
// conversion. Each ADC module has 4 programmable sequences, sequence 0
// to sequence 3. This example is arbitrarily using sequence 3.
//
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
// Configure step 0 on sequence 3. Sample channel 0 (ADC_CTL_CH0) in
// differential mode (ADC_CTL_D) and configure the interrupt flag
// (ADC_CTL_IE) to be set when the sample is done. Tell the ADC logic
// that this is the last conversion on sequence 3 (ADC_CTL_END). Sequence
// 3 has only one programmable step. Sequence 1 and 2 have 4 steps, and
// sequence 0 has 8 programmable steps. Since we are only doing a single
// conversion using sequence 3 we will only configure step 0. For more
// information on the ADC sequences and steps, refer to the datasheet.
//
ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_D | ADC_CTL_CH0 |

```

```

        ADC_CTL_IE | ADC_CTL_END);
//
// Since sample sequence 3 is now configured, it must be enabled.
//
ADCSequenceEnable(ADC0_BASE, 3);
//
// Clear the interrupt status flag. This is done to make sure the
// interrupt flag is cleared before we sample.
//
ADCIntClear(ADC0_BASE, 3);
// Sample AIN0/1 forever. Display the value on the console.
//
while(1)
{
    // Trigger the ADC conversion.
    //
    ADCProcessorTrigger(ADC0_BASE, 3);
    // Wait for conversion to be completed.
    //
    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }
    // Clear the ADC interrupt flag.
    //
    ADCIntClear(ADC0_BASE, 3);
    // Read ADC Value.
    //
    ADCSequenceDataGet(ADC0_BASE, 3, pui32ADC0Value);
    // Display the [AIN0(PE7) - AIN1(PE6)] digital value on the console.
    //
    UARTprintf("AIN0 - AIN1 = %4d\r", pui32ADC0Value[0]);
    // This function provides a means of generating a constant length
    // delay. The function delay (in cycles) = 3 * parameter. Delay
    // 250ms arbitrarily.
    //
#ifdef TARGET_IS_TM4C129_RA0 || \
    defined(TARGET_IS_TM4C129_RA1) || \
    defined(TARGET_IS_TM4C129_RA2)
    SysCtlDelay(ui32SysClock / 12);
#else
    SysCtlDelay(SysCtlClockGet() / 12);
#endif
}
}

```

## Appendix 5 - ADC Single Ended Sampling code.

```
/**
// single_ended.c - Example demonstrating how to configure the ADC for
// single ended operation.
//
// Copyright (c) 2010-2015 Texas Instruments Incorporated. All rights reserved.
// Software License Agreement
// Modified Badru 2016
**/
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
/**
//! \addtogroup adc_examples_list
//! <h1>Single Ended ADC (single_ended)</h1>
//! This example shows how to setup ADC0 as a single ended input and take a
//! single sample on AIN0/PE7.
//! This example uses the following peripherals and I/O signals. You must
//! review these and change as needed for your own board:
//! - ADC0 peripheral
//! - GPIO Port E peripheral (for AIN0 pin)
//! - AIN0 - PE7
//! The following UART signals are configured only for displaying console
//! messages for this example. These are not required for operation of the
//! ADC.
//! - UART0 peripheral
//! - GPIO Port A peripheral (for UART0 pins)
//! - UART0RX - PA0
//! - UART0TX - PA1
//! This example uses the following interrupt handlers. To use this example
//! in your own application you must add these interrupt handlers to your
//! vector table.
//! - None.
**/
// This function sets up UART0 to be used for a console to display information
// as the example is running.
/**
void
InitConsole(void)
{
// Enable GPIO port A which is used for UART0 pins.
// TODO: change this to whichever GPIO port you are using.

```

```

//
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
// Configure the pin muxing for UART0 functions on port A0 and A1.
// This step is not necessary if your part does not support pin muxing.
// TODO: change this to select the port/pin you are using.
//
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
// Enable UART0 so that we can configure the clock.
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

// Use the internal 16MHz oscillator as the UART clock source.
//
UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

// Select the alternate (UART) function for these pins.
// TODO: change this to select the port/pin you are using.
//
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
// Initialize the UART for console I/O.
//
UARTStdioConfig(0, 115200, 16000000);
}
//*****
// Configure ADC0 for a single-ended input and a single sample. Once the
// sample is ready, an interrupt flag will be set. Using a polling method,
// the data will be read then displayed on the console via UART0.
//
//*****
int
main(void)
{
#if defined(TARGET_IS_TM4C129_RA0) || \
    defined(TARGET_IS_TM4C129_RA1) || \
    defined(TARGET_IS_TM4C129_RA2)
    uint32_t ui32SysClock;
#endif

// This array is used for storing the data read from the ADC FIFO. It
// must be as large as the FIFO for the sequencer in use. This example
// uses sequence 3 which has a FIFO depth of 1. If another sequence
// was used with a deeper FIFO, then the array size must be changed.
//
uint32_t pui32ADC0Value[1];

// Set the clocking to run at 20 MHz (200 MHz / 10) using the PLL. When
// using the ADC, you must either use the PLL or supply a 16 MHz clock
// source.
// TODO: The SYSCTL_XTAL_ value must be changed to match the value of the
// crystal on your board.
#if defined(TARGET_IS_TM4C129_RA0) || \

```

```

        defined(TARGET_IS_TM4C129_RA1) || \
defined(TARGET_IS_TM4C129_RA2)
ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
        SYSCTL_OSC_MAIN |
        SYSCTL_USE_PLL |
        SYSCTL_CFG_VCO_480), 20000000);
#else
    SysCtlClockSet(SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL |
SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_16MHZ);
#endif
// Set up the serial console to use for displaying messages. This is
// just for this example program and is not needed for ADC operation.
//
InitConsole();
// Display the setup on the console.
//
UARTprintf("ADC ->\n");
UARTprintf(" Type: Single Ended\n");
UARTprintf(" Samples: One\n");
UARTprintf(" Update Rate: 250ms\n");
UARTprintf(" Input Pin: AIN0/PE7\n\n");

// The ADC0 peripheral must be enabled for use.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

// For this example ADC0 is used with AIN0 on port E7.
// The actual port and pins used may be different on your part, consult
// the data sheet for more information. GPIO port E needs to be enabled
// so these pins can be used.
// TODO: change this to whichever GPIO port you are using.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

// Select the analog ADC function for these pins.
// Consult the data sheet to see which functions are allocated per pin.
// TODO: change this to select the port/pin you are using.
//
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_7);

// Enable sample sequence 3 with a processor signal trigger. Sequence 3
// will do a single sample when the processor sends a signal to start the
// conversion. Each ADC module has 4 programmable sequences, sequence 0
// to sequence 3. This example is arbitrarily using sequence 3.
//
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

// Configure step 0 on sequence 3. Sample channel 0 (ADC_CTL_CH0) in
// single-ended mode (default) and configure the interrupt flag

```

```

// (ADC_CTL_IE) to be set when the sample is done. Tell the ADC logic
// that this is the last conversion on sequence 3 (ADC_CTL_END). Sequence
// 3 has only one programmable step. Sequence 1 and 2 have 4 steps, and
// sequence 0 has 8 programmable steps. Since we are only doing a single
// conversion using sequence 3 we will only configure step 0. For more
// information on the ADC sequences and steps, reference the datasheet.

ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE |
                        ADC_CTL_END);

// Since sample sequence 3 is now configured, it must be enabled.
ADCSequenceEnable(ADC0_BASE, 3);

// Clear the interrupt status flag. This is done to make sure the
// interrupt flag is cleared before we sample.
//
ADCIntClear(ADC0_BASE, 3);

// Sample AIN0 forever. Display the value on the console.
//
while(1)
{
    //
    // Trigger the ADC conversion.
    //
    ADCProcessorTrigger(ADC0_BASE, 3);

    // Wait for conversion to be completed.
    //
    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }
    // Clear the ADC interrupt flag.
    //
    ADCIntClear(ADC0_BASE, 3);

    // Read ADC Value.
    //
    ADCSequenceDataGet(ADC0_BASE, 3, pui32ADC0Value);

    // Display the AIN0 (PE7) digital value on the console.
    //
    UARTprintf("AIN0 = %4d\r", pui32ADC0Value[0]);

    // This function provides a means of generating a constant length
    // delay. The function delay (in cycles) = 3 * parameter. Delay
    // 250ms arbitrarily.
    //
#if defined(TARGET_IS_TM4C129_RA0) || \
    defined(TARGET_IS_TM4C129_RA1) || \

```



```
    defined(TARGET_IS_TM4C129_RA2)
        SysCtlDelay(ui32SysClock / 12);
#else
        SysCtlDelay(SysCtlClockGet() / 12);
#endif
}
```

## Appendix 6 - Matlab code used to develop FFT

```
% badru 2016
signal = load('pp2.txt');
N = length(signal);
KHz=1e3;
fs = 300*KHz; % 300 samples per second
fnyquist = fs/2; %Nyquist frequency
Pk2Pk= max(signal) - min(signal)
rms= std(signal)
plot(signal)

%When roughly interpreting this data half way along x-axis corresponds to half the
sampling frequency
plot(abs(fft(signal)))
xlabel('Frequency (Bins - almost!')
ylabel('Magnitude');
title('Double-sided Magnitude spectrum');
axis tight

%Double-sided magnitude spectrum with frequency axis (in bins)
fax_bins = [0 : N-1]; %N is the number of samples in the signal
plot(fax_bins, abs(fft(signal)))
xlabel('Frequency (Bins)')
ylabel('Magnitude');
title('Double-sided Magnitude spectrum (bins)');
axis tight

%Single-sided magnitude spectrum with frequency axis in bins
X_mags = abs(fft(signal));
fax_bins = [0 : N-1]; %frequency axis in bins
N_2 = ceil(N/2);
plot(fax_bins(1:N_2), X_mags(1:N_2))
xlabel('Frequency (Bins)')
ylabel('Magnitude');
title('Single-sided Magnitude spectrum (bins)');
axis tight

%Single-sided magnitude spectrum with frequency axis in Hertz
%Each bin frequency is separated by fs/N Hertz.
X_mags = abs(fft(signal));
bin_vals = [0 : N-1];
fax_Hz = bin_vals*fs/N;
N_2 = ceil(N/2);
plot(fax_Hz(1:N_2), X_mags(1:N_2))
xlabel('Frequency (Hz)')
ylabel('Magnitude');
title('Single-sided Magnitude spectrum (Hertz)');
```

axis tight

```
%Single-sided magnitude spectrum with frequency axis normalised  
%Normalised to Nyquist frequency. Very common to use this method of normalisation  
in matlab
```

```
X_mags = abs(fft(signal));  
bin_vals = [0 : N-1];  
fax_norm = (bin_vals*fs/N)/fnyquist; % same as bin_vals/(N/2)  
N_2 = ceil(N/2);  
plot(fax_norm(1:N_2), X_mags(1:N_2))  
xlabel({'Frequency (Normalised to Nyquist Frequency. ' ...  
      '1=Nyquist frequency)'}))  
ylabel('Magnitude');  
title('Single-sided Magnitude spectrum (Normalised to Nyquist)');  
axis tight
```

```
%Single-sided magnitude spectrum – frequency in rads per sample
```

```
X_mags = abs(fft(signal));  
bin_vals = [0 : N-1];  
fax_rads_sample = (bin_vals/N)*2*pi;  
N_2 = ceil(N/2);  
plot(fax_rads_sample(1:N_2), X_mags(1:N_2))  
xlabel('Frequency (radians per sample)')  
ylabel('Magnitude');  
title('Single-sided Magnitude spectrum (rads/sample)');
```

```
%Single-sided magnitude spectrum in decibels and KHz
```

```
X_mags = abs(fft(signal));  
bin_vals = [0 : N-1];  
fax_Hz = bin_vals*fs/N;  
N_2 = ceil(N/2);  
plot(fax_Hz(1:N_2), 10*log10(X_mags(1:N_2)))  
xlabel('Frequency (KHz)')  
ylabel('Magnitude (dB)');  
title('Single-sided Magnitude spectrum (1kHz) 300ksps,pk_pk= 1286,rms= 449.3');
```

```
%Single-sided power spectrum in decibels and Hertz
```

```
X_mags = abs(fft(signal));  
bin_vals = [0 : N-1];  
fax_Hz = bin_vals*fs/N;  
N_2 = ceil(N/2);  
plot(fax_Hz(1:N_2), 20*log10(X_mags(1:N_2)))  
xlabel('Frequency (Hz)')  
ylabel('Power (dB)');  
title('Single-sided Power spectrum (Hertz)');  
axis tight  
axis tight  
axis tight
```

```
%Single-sided power spectrum in dB and frequency on a log scale
X_mags = abs(fft(signal));
bin_vals = [0 : N-1];
fax_Hz = bin_vals*fs/N;
N_2 = ceil(N/2);
semilogx(fax_Hz(1:N_2), 20*log10(X_mags(1:N_2)))
xlabel('Frequency (Hz)')
ylabel('Power (dB)');
title({'Single-sided Power spectrum' ...
      '(Frequency in shown on a log scale)'});
axis tight
```

## Appendix 7 - R-language code to analyse ADC data logs.

```
#kf=1 #Badru 2016
filenames= c(
"piccoloLogs/logD(SH-6andx80).txt",
"piccoloLogs/logE(SH-13andx28).txt",
"piccoloLogs/logF(SH27andx100).txt",
"piccoloLogs/logG(SH57andx100).txt",
"TivaLogs/tivaDiffSample/logA2.txt",
"TivaLogs/tivaDiffSample/logB2.txt",
"TivaLogs/TivaSingleEnded/logA1.txt",
"TivaLogs/TivaSingleEnded/logB1.txt",
"TivaLogs/TivaSingleEnded/logC1.txt",
"TivaLogs/TivaSingleEnded/logD1.txt",
"TivaLogs/TivaDithering/TivaDifferentialDithering/logE1.txt",
"TivaLogs/TivaDithering/TivaDifferentialDithering/logE2.txt",
"TivaLogs/TivaDithering/TivaSingleEndedDithering/logE3SE.txt")

for (kf in (1:length(filenames)))
{
xx= read.table (filenames[kf], header = FALSE, sep = ",", dec = ".")
xlen= dim (xx)[1] # e.g. 20k lines od data
xnn= dim (xx)[2] -2 #e.g. 31 samples per line

Uin=as.numeric (xx[ ,1]) # 1-st col-n= Uin values(s)
xdata= rep (NA, xlen*xnn)
dim (xdata)= c (xlen, xnn)
errorCode= xdata

xdataAver= rep (NA, xlen)
#idealCode= rep (NA, xlen)

for (kk in (1:xlen))
{
yy= as.numeric( xx [kk, 2:(xnn+1)])
xdata [kk, 1:xnn]=yy
xdataAver[kk]= mean (yy)
}

plot (Uin, xdataAver, type='l', col= 'blue')
k1 = 4000 # code for "near-zero"
k2 =18000 # code for "near- full-scale"
Kscale= (xdataAver[k2] - xdataAver[k1]) / (Uin[k2]- Uin[k1]) # dCode/dUin
Kscale0= 4095/3.3
RelativeScale= Kscale /Kscale0
OffsetCode= xdataAver[k1]- Kscale *Uin[k1]

idealCode= +OffsetCode + Kscale *Uin # ideal code, corrected offset and scale, input
vecotr-> code vector
```

```

errorCodeAver= xdataAver -idealCode
for (kk in (1:xlen))
{
errorCode [kk, ]= xdata[kk,] -idealCode[kk]
}

maxErrAver_index= k1-1+which.max (abs(errorCodeAver [k1:k2] ))
maxErrAver = errorCodeAver [maxErrAver_index]

xtxt= paste("RelScale=", as.integer(RelativeScale*1000)/1000, "offset= ",
as.integer(OffsetCode*10)/10, "digits, maxErrAver= ",
as.integer(maxErrAver*10)/10)

#print (xtxt)

xname= paste(kf, ".jpg", sep="")
jpeg(filename = xname, width = 760, height = 560)
plot (Uin, errorCodeAver, xlim= c(-0.01, +3.6), ylim= c(-25,25), type='l', col= 'blue')
grid()
title (xtxt)
for (kk in (1:xlen))
{
Ux= rep (Uin[kk], xnn)
lines (Ux, errorCode[kk, ], col= 'grey')
}
lines (Uin, errorCodeAver, col='blue') # overwrite
lines (maxErrAver_index, maxErrAver, type='o', col= 'red')
text (1.2, -23, filenames[kf])
dev.off()

}

```