Institute of Computer-aided Product Development Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3249

Development of procedures and evaluation strategies for novel field-effect transistor sensors

Michael Lee Parker

Course of Study: Information Technology (INFOTECH)

Examiner: Univ-Prof. Hon-Prof. Dr. Dieter Roller

Supervisor: Dr. Thomas Brosche (Robert Bosch GmbH)

Commenced: 20. July 2011

Completed: 20. Jan 2012

CR-Classification: B.5.0, C.3, G.1.0, J.7

Acknowledgments

I would like to express my appreciation to all of the people who have helped and assisted me throughout my master thesis.

At Robert Bosch GmbH, I would like to thank my supervisor, Dr. Thomas Brosche, for the helpful guidance and the CR/ARE4 group for the nice working atmosphere. Furthermore, I would like to thank Prof. Dr. Dieter Roller, for providing me the opportunity to do this work.

Stuttgart, January 2012

Michael Parker

Entwicklung der Verfahren und Strategien zur Bewertung von Feldeffekttransistor-Sensoren

Abstrakt

Zur Beurteilung der neuartigen Sensoren auf der Basis der Feldeffekttransistor-Technologie wurden kosteneffiziente Mess- und Prüfverfahren entwickelt. Aufgrund der Anfälligkeit einiger der Sensortypen für Drift und Rauschen wurde ein Messsystem nach dem sog. switched-biasing-Ansatz realisiert, welches bei geeigneter Konfiguration nachweislich die Drift reduziert.

Innerhalb dieser Arbeit wurde auf einem FPGA die Vorfilterung sowie die Dezimation der gemessenen und mit einem schnellen Analog-Digital-Wandler abgetasteten Daten implementiert, wobei die Abtastrate des hierfür verwendeten CIC-Dezimationsfilters flexibel einstellbar ist. Hierdurch wird das Signal-Rausch-Verhältnis erhöht und die erforderliche Datenübertragungsrate reduziert. Die Erfassung der Messdaten und das Messsystem werden intern von einem Mikrocontroller gesteuert. Er überträgt die Messergebnisse über die USB-Schnittstelle auf den PC zu einem übergeordneten System, z.B. ein MATLAB-Programm. Es können mehrere Messsysteme von einem übergeordneten System kontrolliert und deren Daten parallel erfasst werden. Systematische Fehler im Zusammenhang mit Einschränkungen der Mess-Hardware, wie Offset, Temperatur und Drift werden durch eine Kalibrierung weitgehend kompensiert.

Das Ergebnis dieser Arbeit ist ein software- und hardware-seitig umgesetztes System, welches sowohl die Ansteuerung eines Transistors mit switched-biasing als auch eine ziemlich präzise Messung seiner Performance ermöglicht. Hiermit konnte die Reduktion der Drift unter verschiedenen switch-biasing-Konfiguration untersucht und die Effektivität des Ansatzes anhand ausgewählter Messungen validiert werden.

Stichworte:

Feldeffekt-Transistor-Sensor, Switched-Biasing, CIC-Filter,
Transistor Charakterisierung, Analog-Digital-Wandlung,
Digital-Analog-Wandlung, Mikrocontroller-Schnittstellen,
FPGA-Schnittstellen

Development of procedures and evaluation strategies for novel field-effect transistor sensors

Abstract

In order to evaluate new types of sensors based on the field-effect transistor technology, a cost-effective measurement and control system is developed. Because some new types of transistor-based sensors are particularly prone to drift and noise, a measurement system is built around evaluating the effect of a biasing technique known as switched biasing, which has been shown to reduce drift under certain configurations.

The result is an implementation of software and hardware that is both able to control a transistor with switched biasing, explore drift-reducing switched biasing configurations, and accurately measure its performance with relatively high precision. Pre-Filtering of the measured data coupled with a fast actuation of an analog-to-digital converter is realized and implemented on a FPGA in the form of a rate-adjustable CIC decimation filter, which increases the signal-to-noise ratio and reduces the required data-transfer rate. The measurement system is controlled internally by a microcontroller and is interfaced through a USB interfaces to a higher-level system, such as a computer running MATLAB, and allows for multiple measurement systems to be operated in parallel. Systematic errors related to limitations of measurement hardware such as offset, temperature and drift are evaluated and compensated for through calibration.

Keywords:

Field-effect-transistor-sensor, Switched-Biasing, CIC-Filter,
Transistor Characterization, Analog-Digital-Conversion,
Digital-Analog-Conversion, Microcontroller, FPGA

Contents

List of Abbreviations

ADC	Analog-to-digital converter
API	Application programming interface
CIC	Cascaded integrator-comb
CR	Carriage return
DAC	Digital-to-analog converter
DCO	Digitally-controlled oscillator
DVI	Data Valid Input
DVO	Data Valid Output
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMI	Electromagnetic interference
FET	Field-Effect Transistor
FLL	Frequency locked loop
FPGA	Field Programmable Gate Array
FSR	Full scale range
Hz	Hertz (cycles per second)
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IrDA	Infrared Data Association
JTAG	Joint Test Action Group
LED	Light-emitting diode
LF	Line feed
LIN	Local Interconnect Network
LSB	Least-significant bit/byte

MCU	Microcontroller Unit
MSB	Most-signifiant bit/byte
MSP	Mixed signal processor
Op-amp	Operational Amplifier
PC	Joint Test Action Group
PC	Personal Computer
PC	Personal computer
pc	Printed circuit
PID	Proportional-Integral-Derivative
RMS	Root mean square
RST	Reset signal (pin)
SCPI	Standard Commands for Programmable Instruments
SPI	Serial Peripheral Interface
SPS	Samples per second
TI	Texas Instruments
UART	Universal asynchronous receiver/transmitter
UCS	Unified Clock System
UCSI	Universal Serial Communication Interface
UI	User Interface
USB	Universal Serial Bus
VCP	Virtual COM port

Chapter 1

Introduction

1.1 Motivation

In the development of new types of sensors, the field-effect transistor (FET) is a device that being developed for a variety of sensing applications. In order to evaluate the performance of transistors, e.g. using different types of materials under varying temperatures and conditions, high-performance measurement equipment is needed that can measure the characteristics of the transistor. During the development of these sensors, it was found that certain sensors are prone to errors such as temperature drift and biasing drift. A biasing technique known as switched biasing has also been shown to reduce these effects. Therefore research into the configuration of the switched biasing under varying conditions is being conducted in order to determine the best configuration of the switched biasing technique under varying conditions.

While the concept of switched-biasing has shown to improve performance, it is still necessary to test the performance of new types of transistors, variants of existing transistors, obtain statistically sound samples size, and perform long-term testing. The capability to perform this kind of research already exists in the form of expensive test equipment, however because the equipment is usually developed for generic applications and is capable of more than is needed for this one specific research application, the cost of a test station using commercially available equipment becomes prohibitively expensive, both in terms of price of equipment, cost of operation as well as physical space required in the test environment. A proposed solution to this problem was the development measurement system built especially for the purpose of controlling and measuring the performance of transistor-sensors under a variety of conditions, while still providing a level of accuracy and precision comparable to or exceeding that which commercially available equipment provides, but in a cost-effective form.

1.2 Previous Work

In the work by [Winkelman 2009], it was first shown that it was possible to reduce drift and improve transient response under certain temperature conditions for new types of transistor sensors by biasing them with the switched biasing technique. The measurement station used to perform this testing was composed of a Eurotherm PID regulator for heating control, an Agilent frequency generator, and Keithly Sourcemeter for control via a PC running Labview [Winkelman 2009].

Further research into improving the response of transistor sensors with switched biasing led to the development of a measurement system that could also reduce the cost of measurement by reducing the number of interconnection between multiple sensors through multiplexing of certain signals. This involved a system incorporated a microcontroller with built in analog-to-digital conversion for control and measurement, multiplexing circuitry and a PC system running Labview to which measurement results are sent and displayed graphically [Ganz 2010]. It was found through this work that the multiplexing system was not an effective for measurements and that determination of the drift behavior is required.

In response to the work by [Ganz 2010], further development of a measurement system was conducted, as documented by [Winkelman 2009]. The proposed system includes high-performance components to measure and control the tested transistor sensor. This work included characteristics of a measurement system components important required, these characteristic included:

- Square wave output for switched biasing signal
- frequency of the switched biasing signal: (1kHz to 100 kHz)
- Configurable high and low switched bias voltages (-4V to 2V)
- Amplitude of the switched biasing signal
- Switched biasing duty cycle ($<=$ 20%)
- Variable Temperature
- Transistor drain-source voltage ($\pm$15V)

As a result of the work by [Winkelman 2009], a printed circuit board (pc board) incorporating components capable of meeting and exceeding these design requirements was developed. The improved hardware was designed to test two transistors simultaneously, and incorporates high-resolution (16-bit) components, a high performance MCU, and an FPGA to provide fast data processing and filtering of incoming ADC data.

1.3 Problem Definition

The goal of this thesis is to program the components of the newly designed hardware board, verify component interoperability, suggest changes or improvements to the hardware, and program all components so the hardware is ultimately capable of providing precise and accurate measurements of two transistors simultaneously.

During the course of the thesis, the specific goals are to:

- Incorporate the existing hardware and program the user and hardware interfaces.
- Create a strategy to calibrate the measurement system
- Implement pre-filtering of ADC measurement data in the FPGA using VHDL
- Creation of a program for the MCU, written in C to control the measurement process, set voltage levels, configure the analog-to-digital conversion timing,

and transfer data to a higher-level system (e.g. a PC running MATLAB or LabView)
- Creation a program for data acquisition on a the higher-level system and visualize the results.
- Evaluation of drift compensation using switched biasing

In the introductory phase of the work (Chapter 2), the selection of components, their capabilities, and their interoperability is reviewed. Existing measurement system concepts are reviewed and the role of software for the MCU and FPGA in introduced.

This is followed by the implementation phase of the work (Chapter 3, in which how each software and FPGA component was implemented as part of the thesis. Programming in this phase includes writing and testing software for the MCU, FPGA and higher-level system.

The implementation is broken down into two parts, MCU and FPGA. For the MCU, this is the software written in C for: component initialization, component interfacing with the MCU, control of switching logic used for calibration, the calibration logic itself, current flow switching control and the user interface (UI), with which which the higher-level system interacts to set system parameters and received measurement data.

The second phase of implementation is an explanation of VHDL coding on the FPGA, whose role is to control logic for the ADC, and provide received filtered data for retrieval by the MCU. FPGA implementation is broken down into *components*. These components work on the black-box principle, each of which has its own internal input and output and plays a role in the overall functionality of the FPGA design. These components are: MCU IO, the communication and storage component; ADC READ the analog-to-digital control logic; CIC, the pre-filtering and decimation logic; ADC CH the filer selection logic; CALC CH, triggering control for each of the two transistor measurement ports; and RESET SYNC, FPGA reset logic.

In chapter 5, the capabilities and limitation of both the hardware and the implementation are discussed. This includes a discussion of the possible error that can be introduced from noise, or limitations of the hardware; capabilities of the overall system, including system limits. This is followed by actual testing of transistors with and without switched biasing, drift behavior of a transistors and of internal calibration resistance that help determine the offset, precision, accuracy and linearly of the overall hardware and software design.

The thesis is concluded with a short discussion of known issues, and further work. Issues found in the course of the thesis include a possible silicon bug on the MSP430, pc board power supply issues with the DAC during start-up, and error checking and handling weaknesses in the implementation. Further work includes implementation in software on the MCU of a dynamic calibration logic to compensate for component losses and gain error, a PID control loop for control of a heating element, improvement of the higher-level system software, further analysis for the need of calibration of individual digital-to-analog output channels to improve accuracy.

Chapter 2

Fundamentals

2.1 Measurement system overview

In this chapter, the newly designed hardware for the simultaneous measurement of two transistors utilizes several high-performance hardware devices in order to precisely and accurately measure the performance of transistors. In this chapter, the measurement system, its components are discussed. Including, criteria for selection of hardware components, their capabilities and their role in the overall measurement system.

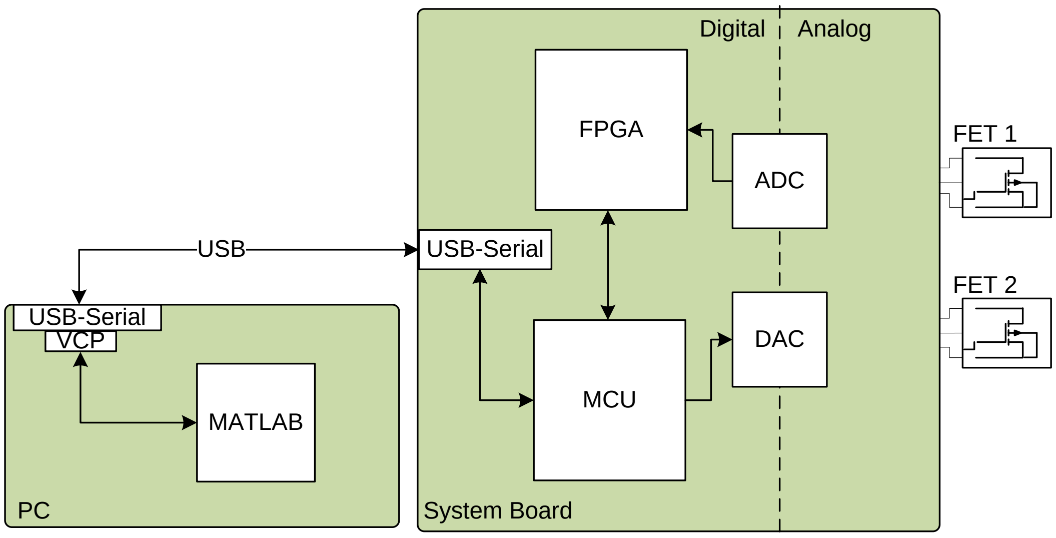


Fig. 2.1: Component overview with connections

The measurement system consists of the hardware in form of a printed circuit board and a higher-level system, typically a computer workstation running MATLAB or Labview. In actual operation, a heating element will also be used to heat the transistor under test, with its temperature regulated by PID controller implementation on the MCU, and the temperature is measured and sampled by the ADC.

The printed circuit board contains all of the ports to transistor terminals, interfaces to external devices, and hardware components with which the measurements will be made. This including all of the support for the devices, such as power supplies which supply voltage to the various components. A simplified overview of the measurement system, including important components for measurement, the higher-level system, and field-effect transistors (FET) under test is shown in figure 2.1).

The circuit board connects via USB to the higher-level system, shown here as a personal computer (PC) running MATLAB, interfaced to the measurement board via USB and a Virtual COM port (VCP)[1]. The USB interface can also be used to with other software, such as LabView, or terminal-emulation software running on a PC via the VCP and USB.

2.1.1 Field Effect Transistor

A basic field-effect transistor is a semiconducting device consisting of three terminals, Gate (G), Drain (D) and Source (S), through which the amount of current that flows between the source and drain electrodes is controlled by an electric field at the gate electrode. Conventional current entering the channel at S is designated by I_S. Conventional current entering the channel at D is designated I_D. Drain to Source voltage is V_{DS} and by applying voltage to G, one can control ID. Applying voltage to the gate electrode of the transistor such that the current may flow between drain and source is referred to as “biasing”[Winkelman 2009].

2.2 Measurement concepts

The transistor characteristic of most interest in our measurement system is the relationship between voltage at the terminals and current flow between source and drain, and current flow between gate and drain. Using these current characteristics, we can *characterize* a transistor, i.e. describe the voltage-current relationships of a transistor. In the measurement circuit, voltage is applied at the drain, and measured at the source.

2.2.1 Current measurement circuit

The current measurement circuit in our system uses the DAC to apply voltages to the transistor and the ADC to measure the resulting current flow. Because the ADC measures voltage and not current, additional circuitry is required in order to create a voltage proportional to the current flow through the transistor.

An operational-amplifier is used in a current-to-voltage configuration, using one of two high-precision reference resistances to create feedback, creating a *virtual ground* at the source terminal of the transistor[Jacob Millman 1985]. The op-amp compensates for voltage differential at its input by creating voltage at output which, in this configuration is fed back into its input through one of two reference resistances. These known resistances are then used later to calculate the resulting current flow by taking the measured value of the voltage at the output of the op-amp (connected to, and measured by the ADC) and applying Eq. 2.1.

[1]VCP is a software driver interfacing the legacy COM port interface typically found on a PC with the USB interface

An overview of the biasing and current measurement circuit is shown in figure 2.2

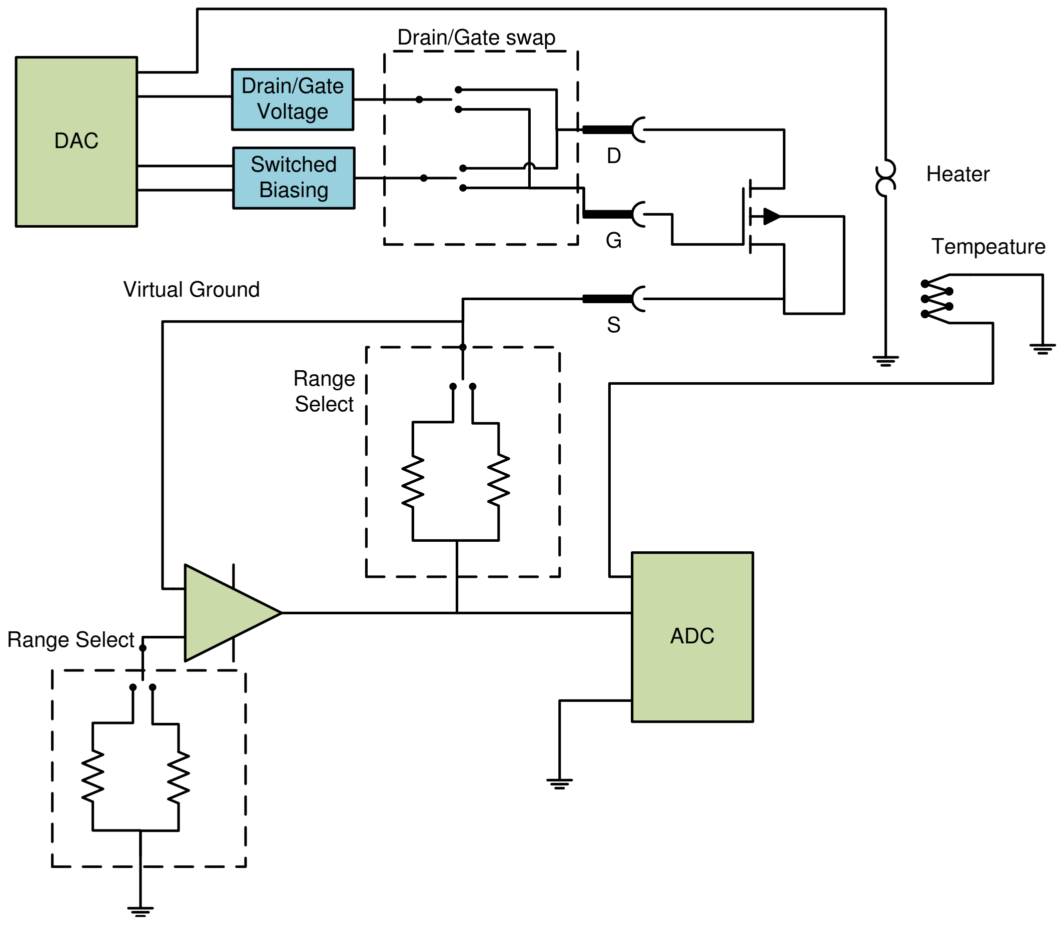


Fig. 2.2: Biasing and current measurement overview

Current is measured simply by observing Ohm's law[Fetzer 1965],

$$I = \frac{E}{R} \tag{2.1}$$

where E is the voltage in volts (V), I is the current in Ampere(A) and R is the resistance in Ohm(Ω).

2.2.1.1 Measurement ranges

In our measurement circuit, one of two reference resistance is used, switched by a relay control by the MCU. The reference resistance, 2.5KΩ±0.02% and $25K\Omega$±0.02% and the maximum measurable input range of the ADC (5V) gives two measurement ranges with which the circuit can operate:

$$I_{25K} = \frac{5V}{25K\Omega} \tag{2.2}$$
$$= 200\mu A \tag{2.3}$$
$$I_{2.5K} = \frac{5V}{2.5K\Omega} \tag{2.4}$$
$$= 2mA = 2000\mu A \tag{2.5}$$

Because the ADC is dual supply, measurement $\pm 5V$ is possible, therefore the actual measurement range is:

$$I_{low} = \frac{\pm 5V}{25K\Omega} \tag{2.6}$$
$$= \pm 200\mu A \tag{2.7}$$
$$I_{high} = \frac{\pm 5V}{2.5K\Omega} \tag{2.8}$$
$$= \pm 2mA = \pm 2000\mu A \tag{2.9}$$

An important consideration when interpreting the resulting ADC values is that the value will be inverted. This is because current flows toward virtual ground in the current-measurement circuit.

2.2.1.2 Drain-Source, Gate currents

When measuring I_{DS}, a typical FET will not be ideal and the measured current will include leakage current from the gate, although normally negligible, a characterization of the FET should take this into consideration. measured current is $IDS + I_G$. The drain current can be found by removing the voltage applied at the drain terminal while applying the bias voltage to the gate and measuring the resulting current. This current is then subtracted from the previously measured drain-source current[Jacob Millman 1985]:

$$I_D = I_G + I_{DS} \tag{2.10}$$

In order to determine these values, there are relays in parallel across the drain and source of the FET (named the IG relay) and in series between the drain terminal and DAC (named the VDS Relay). In order to measure I_G, the VDS relay is opened, leaving only the voltage on the gate, then current is measured. The IG relay can be closed in order to create a low-resistance path from drain to source, bypassing the FET.

In normal measurement state, the IG relay is open and the VDS relay is closed.

2.2.1.3 Drain-Gate swap

In the standard measurement configuration, the SWB signal is always applied at the gate terminal. In case it is desired to apply biasing at the FET drain terminal, cables would have to be physically swapped from one terminal to the other. Therefore a “swap” relay was built into the other, the purpose of which is to make swapping biasing more convenient and less prone to physical damage to the board, test cables, or device under test.

When “swapped” the two gate switch-biasing reference voltages: VDAC0/VDAC1 (port 1) and VDAC4/VDAC5 (port 2) are swapped with the drain reference voltages of each port, VDAC2 (port 1) and VDAC6 (port 2).

This is controlled by the SWAP relay, for more details section 3.2.9, drain-gate swap implementation.

2.2.2 Switched biasing

Switched biasing, refers to “switching” the bias to the FET between two voltages levels, at a particular frequency and duty cycle in order to improve certain characteristics of transistor performance , such as transient response and especially drift caused by "1/f" noise [Winkelman 2009]. Since this is such an important concept to the performance of the transistor, much of the measurement system is built around measuring the effects of this technique.

The switched biasing voltage levels for each transistor to be measured can be set independently from each other. For measurement port 1, this between which the biasing switches are set independently for each transistor.

2.2.3 Sampling windows

Measurement I_DS is divided into two measurements, depending on the current resulting from one of the two switched biasing levels applied during measurement. These two levels are referred to hereon as A or B. Furthermore, is desirable to measure the current first only after transient effects of switching have settled, therefore it is possible to define periods within the switched biasing period, in which the current measurement voltage is sampled, these are defined as “sampling windows” and displayed in figure 2.3.

Definition of the sampling window depends on the desired measurement configuration, which consists of the following:

Period is the time at which the both levels of the signal are applied, to the transistor, the value of *period* is measured in number of FPGA system clock cycles: $(1_{CLK} = 50ns)$

Width is the time at which the the switched biasing is set at the A voltage level; the difference between period and width is the time in which the SWB is at is “B” level.

Measurements of transistor current occur withing *sampling windows*, bounded by the values “phase a” and “phase b” for the A sampling window and the values “phase c” and “phase d” for the B sampling window, as shown in figure 2.3.

Current measurement is not instantaneous, the ADC begins its internal analog-to-digital conversion process after receiving a conversion start (CONVST) signal, as controlled by the FPGA process ADC READ (see section 3.3.3). The conversion process typically 3.1 μs, plus an additional 0.9μs to send out the data, giving a total *conversion cycle* time of approximately 4μs. Therefore it is important to configure the sampling windows based on these, and other limitations (for more information, see System Limitations, sec. 4.2.

Depending on the speed of the switched biasing signal, 1 or as many as 25000 samples can occur within a sampling window. Conversion cycles occur one after another as it is within a sampling window. A sampling delay can be configured between conversion cycles, figure 2.4. This is can be used to adjust the rate at which the ADC samples without adjusting the sampling window configuration or switched biasing configuration.

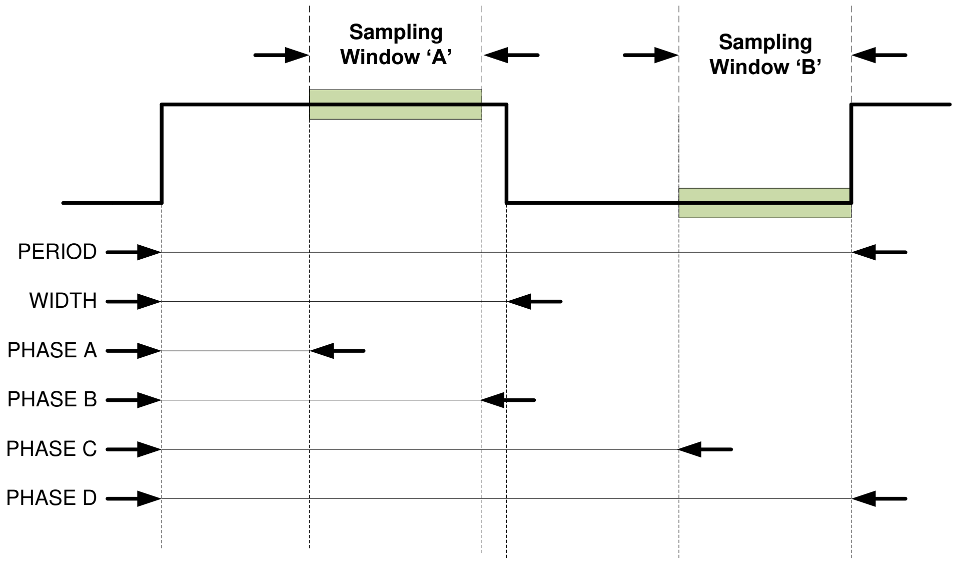


Fig. 2.3: Sampling window, period, width

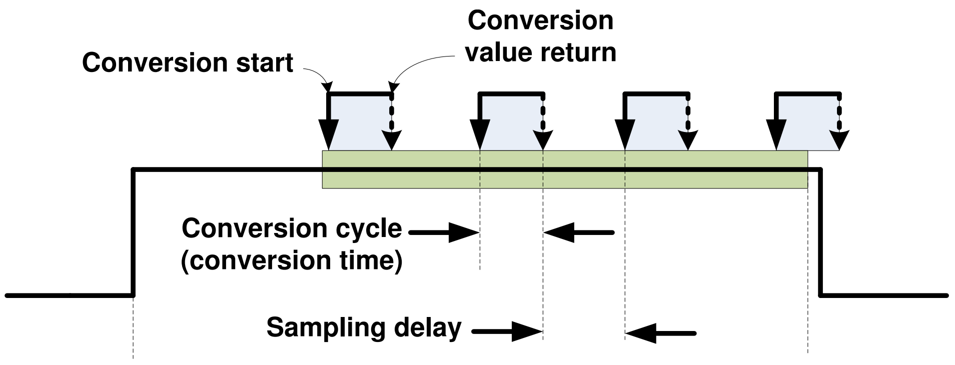


Fig. 2.4: Sampling delay, conversion time

2.2.3.1 Filtering and decimation

The configuration of the sampling window may result in a sample every 4μs, depending on the configuration of the sampling window, this can result in an sample rate as high as 250000 per second, however by reducing data rate by sending the samples through a type of moving-average filter known as a CIC filter, we achieve two things: our data output rate to the higher-level system is reduced, and the signal-to-noise ratio of the signal is improved [Kester 2008].

Samples taken at the A switched biasing level are sent through the *A filter* on the FPGA, while samples taken at the B switched biasing level are sent through the *B filter* on the FPGA. Each measurement port has its on set of A and B filters, additionally, temperature measurements of each transistor are sampled by the ADC and sent through their own filter (T filter).

Through digital averaging filtering of the samples, the noise-free resolution of the the ADC is effectively increased .

2.2.4 Calibration

Calibration is performed in order to adjust for offsets and losses as a result of switch resistance or gain error in the operational amplifier.

2.2.4.1 Zero-offset calibration

Zero-offset errors can occur in both the ADC and DAC. Offset error for the ADC is the non-zero representing value returned when its input voltage is zero; for the DAC, it is the non-zero output value of an output, when that output is configured for 0V [Instruments 1995].

In order to compensate for zero-offset for the zero offset error, all output values that would result in current flow are set to 0V. Since the output values are the DAC are set to 0V, any difference in the actual output value of the DAC measured at the ADC, which may offset of it own is then compensated for storing the returned, filtered value in this configuration. Then during measurement with voltages applied, this value is removed from the measured value.

2.2.4.2 Dynamic calibration

To compensate for switch ON-resistance and for operational amplifier gain error during measurement, a dynamic calibration circuit is included to determine the *error factor*, as shown in figure 2.5.

The dynamic calibration circuit consists of two relays and two high-precision reference resistances, Rc1 ($10\text{K}\Omega \pm 0.01\%$) and Rc2($100\text{K}\Omega \pm 0.01\%$). During measurement of a transistor, the relay in series with the transistor is disabled, preventing current flow through the transistor under test. Then either Rc1 or Rc2 is connected to the previously connected DAC voltage and to the current measurement circuit. Current is measured and the second calibration resistance is measured.

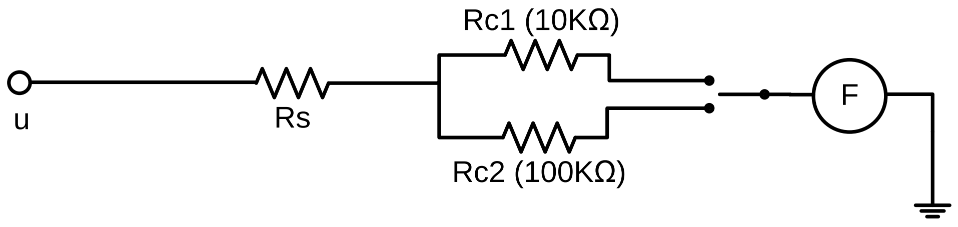


Fig. 2.5: Dynamic calibration

Since the value of the two resistances is known (within the confidence interval), and the value of the DAC voltage output is known (within the tolerance of the DAC), a system of equations with two equations and two unknowns can be used to determine the value of Rs, the ON-resistance and the the full-scale error factor as shown in figure 2.5 and the following equations:.

$$I_1 = \frac{u}{F * (Rc1 + Rs)} \tag{2.11}$$

$$I_2 = \frac{u}{F * (Rc2 + Rs)} \tag{2.12}$$

$$u = I_1 * (Rs + Rc1) \tag{2.13}$$

$$= I_2 * (Rs + Rc2) \tag{2.14}$$

$$(I_1 - I_2) * Rs = I_2 * Rc2 - I1 * Rc1 \tag{2.15}$$

$$Rs = \frac{I_2 * Rc2 - I_1 * Rc1}{I_1 - I_2} \tag{2.16}$$

$$F = \frac{u}{I_1 * (Rs + Rc1)} \tag{2.17}$$

$$= \frac{u}{I_2 * (Rs + Rc2)} \tag{2.18}$$

$$= \frac{u(I_2 - I_1)}{I_1 * I_2(Rc1 - Rc2)} \tag{2.19}$$

2.2.5 Heater and temperature

Since the temperature of a transistor plays an important role in the characterization of a transistor, facilities to both measure temperature and to heat the transistor are considered for the measurement system. Although analysis of the effects of temperature and heat were beyond the scope of this thesis, it was important to built in the capability for future development of these factors.

As shown in figure 2.2, temperature will be measured and voltage level corresponding to that temperature will be sampled by the ADC. For heating control, a pulse-width modulated signal, whose signal is a result of a regulation routines in the MCU is planned for future implementation.

When considering the design of the measurement system software, these techniques by which temperature and heat were considered in the design. A filter for this temperature has also been implemented in the FPGA for this purpose, and filtered sampled values can be retrieved from the FPGA. For the heating control, a program structure on the MCU compatible with PID control was implemented.

2.3 Measurement hardware

In this section, a description of the hardware components that make up the printed circuit board is given. Described are the important elements of the measurement system, including criteria for their selection, their capabilities and their role in the measurement system.

2.3.1 Printed circuit board and housing

The printed circuit board, approximately A4 in size is a multiple layer printed circuit board, on which all components are placed and to which source, gate, and drain cables lead to the transistor under test. USB, programming ports, debugging pins all connect directly to ports on the board. The circuit board is to be enclosed in a housing to both protect its components and to prevent EMI[2] from affecting other test equipment in the vicinity.

During the course of the thesis, a second revision of the printed circuit board was developed based on intermediate findings of this work. The first revision, served as a prototype on which the hardware and its implementation could be tested. As problems with the design were found, the design of the board and changes were made, the second revision of the board was produced.

The printed circuit board and its hardware elements were chosen so that two transistors can be tested simultaneously, each driven by independent DAC channels and measured by independent ADC channels, each with its own set of relays and transistor connection ports.

2.3.2 FPGA

The Field Programmable Gate Array is an integrated circuit, whose operation is specified a hardware description language, such as VHDL and re-programmable. The FPGA provides fast, parallel processing capability, making it ideal for simultaneous filtering role it plays in this measurement system. While its primary role is to filter data, it also designed to control the ADC and provide an interface to the MCU to return filtered sample values (for the FPGA design, see FPGA implementation, section 3.3). Additionally, the FPGA has been designed to create the *switched biasing control signal*, a signal which causes the voltage level to switch between the two configured levels at the measured transistor, as well as control the timing of the ADC sampling.

The capacity of the FPGA is measured in logic blocks or logic elements (LEs), the capacity determines how much memory (registers) and signals can be used in

[2]Electromagnetic interference

the In order to sample the current measurement values as quickly as possible and provide filtered results, a FPGA with the a large enough capacity to (logic-elements (LEs)) to accommodate the design.

The FPGA chosen is the Altera EP1C6Q240C6N, which belongs to Altera Cyclone family of FPGAs. The EP1C6 variant to which the this FPGA belongs, includes 5980 LEs, 20 RAM blocks, 92160 RAM bits, 2 PLLs, and 185 I/O pins [Altera 2007].

2.3.3 Microcontroller (MCU)

A microcontroller is comparable to a small computer on a chip, containing a processor, memory, and input/output capability. It runs software written especially for the MCU, is typically written in either assembly or in the C programming language. The main role of the MCU in the measurement system is to provide a user interface with which all measurements can be coordinated, provide an interface to the FPGA and DAC, and control the transistor heating process. Filtered sample values are retrieved by the MCU and returned to the user via the USB interface (for MCU software implementation, see section 3.2.

The MCU chosen for this project is the Texas Instruments (TI) MSP430F5438A-IPZ. It is a general purpose microcontroller in TIs MSP430 range of products.

The product number describes characteristics of the MSP430
variant [Instruments 2011a].

MSP	Mixed-signal processor
430	430 MCU platform
F	Includes flash memory
5	Generation 5xx
4	Family
38	Series and device number
I	Temperature (I:range -40 to 85 C)
PZ	Packaging

Features of this MCU is 16-bit memory, 256KB flash memory, 16KB RAM, 4 USCI (communication interface useful for interfacing Serial-USB conversion component and DAC), global and pin interrupts and high clock rate capability (20 MHz).

2.3.4 Analog-to-Digital Converter

The analog-to-digital converter (ADC) is used to convert an analog voltage at its input to a discrete voltage level. The *resolution* of an ADC determines the number of discrete voltage levels over its full-range of input. The rate at which it performs this is its sampling rate, and is measured in samples per second (SPS).

The role of the ADC in the measurement system is to convert the analog voltage from the current-measurement circuit to a voltage level proportional to the measured current and to convert the a voltage for temperature measurement. An ADC was chosen that provides both high resolution, multi-channel independent input and high sampling rate (SPS) as determined by [Kolka 2010]. The ADC chosen for this system is is the Analog Devices AD7656, a 6-input, 16-bit resolution ADC with a maximum sampling rate of 250 KSPS. The measurement system design uses 4 channels (2 per measurement port), however future design may include the remaining two for other purposes such as calibration.

2.3.5 Digital-to-Analog Converter (DAC)

The digital-to-analog converter (DAC) translates a range of discrete input values (codes) into an analog voltage level. The number of voltage levels, i.e. step size, is determined by its resolution.

The role of the DAC in the measurement system is to bias the transistor with two voltage levels for switched biasing and to apply a voltage to the drain terminal of the transistor. Additionally the DAC will provide a voltage for control of a heating element, which heats the transistor to a temperature measured by the ADC and regulated by the MCU.

The Texas Instruments DAC8718 was chosen because of its 16-bit resolution which is able to provide fine control of the configured voltage levels, and 8-channel output all of which are utilized by the system. The DAC8718 also is interfaced via the SPI interface (see section 2.3.10.1).

The output level is determined by a number of factors including reference voltage, the input code, calibration registers, and user configured gain, configurable to 4 or 6. A correction engine and configurable offset allow the user to calibrate the DAC output. All configurable aspects of the DAC are programmable by writing to the registers, which occurs directly over the MCU-DAC interface (see DAC interface implementation section 3.2.6).

2.3.6 DAC reference voltage

The reference voltage provides a stable reference voltage with which the DAC creates the output voltage. The reference voltage chosen for this measurement system is the 5V TIREF5050 from Texas Instruments. It provides low-noise ($3\mu V_{PP}/V$), low temperature drift (8ppm/Â°C (max)) and a precision 5V voltage reference.

2.3.7 Serial-USB interface

The serial-USB interface provides a internal serial connection for the MCU externally a USB connection for the higher-level system (PC with MATLAB), a Serial-USB chip was chosen that both can handle a high baud rate, and interface the EEPROM chip that is used to serialize the measurement boards.

The chip chose for this purpose is the FT232BL from FTDI. The FT232BL is the lead-free variant of the FT232BM, which provides complete support for asynchronous serial transfer between the MCU and USB/higher-level system.

For the interface between the MCU and the FT232BL was chosen for its high baud rate in serial mode(up to 1 MBaud), and its interface with EEPROM that provides serial number support.

When interfacing with the higher-level system, there are two possibilities: via the the virtual COM port driver, a software drive emulating the functionality of a legacy COM port; and the D2XX interface, a programming API[3] from FTDI provides higher data rates and access to special features of the FT232BL.

2.3.8 EEPROM

The EEPROM (Electrically Erasable Programmable Read-Only Memory) provides storage for the FTDI serial-USB interface. An interface between both the FT232BL and EEPROM and between the MCU and EEPROM is used to provide a single storage point for a serial number.

The serial number is used by the VDP and D2XX drivers to uniquely identify a device. For our measurement system, the serial number is retrieved by the MCU into uniquely identify individual measurement systems that may be connected to the same machine, or to identify for repeatability, a particular system board.

2.3.9 Relays and Switches

The use of relays in the measurement system is intended to control the flow of current at various points of the measurement system. The switches are control via control signals set by the MCU, depending on the the configuration of the measurement (i.e. input voltage levels, biasing voltage levels and frequency can first be configured, then switched on.

Switches are used to control current flow to: dynamic calibration resistors, between DAC and the transistor under test, short-circuit FET terminals for calibration, in the drain-gate swap circuitry, and in switched biasing of the transistor. terminals for calibration, in the drain-gate swap circuitry, and in switched biasing of the transistor.

In the first revision of the printed circuit board, solid state relay switch PS710BL was used. However it was found that it suffer from capacitance loading, resulting unintended current flow and causing poor transient response. In the second revision of the board, the this component was replaced with the REED SIP DPST analog relay switch, which has shown to provide much better performance.

2.3.10 Interfaces and buses

USB

The Universal Serial Bus (USB) is a standard found on most modern computer workstations and it is used by the measurement system to provide data at a relatively high rate of speed. The USB interface provided by the serial-USB Converter FT232BL, supports both USB1.1 and USB2.0 standards.

[3]Application programming interface, provides an interface between software components

JTAG

The JTAG programming and debugging interface, or more officially the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture [Wikipedia 2011] is the primary method of debugging and programming the MCU. A newer alternative to this interface is the SPI-Bi-Wire interface [Instruments 2011b]. Although this interface has the advantage of requiring only two pins (TEST and RST), it is however slower than the 4-wire JTAG interface. The JTAG interface was selected in later hardware revisions for this reason.

The interface between the PC and the JTAG programming interface is dependent on the device for which programming and/or debugging is intended, therefore there are two JTAG programming ports are available on the board. For the MCU, this port is interfaced with the PC via the MSP430 USB Debug-Interface MSP-FET430UIF. For the FPGA, programming and debugging is achieved with the Altera USB Blaster interface. Both of which allow interfacing with the programming and debugging via a USB interface. The Altera FPGA USB Blaster interface supports the following functions for programming, configuration and logic analysis[Altera 2009].

2.3.10.1 SPI

The Serial Peripheral Interface (SPI) bus is a serial bus consisting of three or four-wire configuration, used in the measurement system between the MCU and DAC. These four signals are: STE, SIMO (SDI), SOMI (SD), and CLK. The purpose of STE is to share interconnections and to select the chip in multi-chip configurations (chip-select). SIMO is short for Slave-in Master-out (also SDI, slave data-in), and SOMI is Slave-out, Master-in (also SDO, slave data out).

In the SPI bus, one device is designated the *master* with one more *slaves*. Messages from the master to the slave occur exclusively on the SIMO signal, while messages from the slave to the master occur exclusively on the SOMI line. The CLK signal synchronizes the bits of the signal, reading on either edge of the clock signal.

2.4 Higher-level system

The higher-level system is the named designed to the computer used to connect to the measurement system via the USB interface. It will typically run software that can configure the aspects of the measurement system, retrieve measurements, and display or otherwise output the results of the measurement.

The higher-level system chosen for during implementation and testing is a PC running Microsoft Windows and the MATLAB software. MATLAB was chosen because of its built-in ability to interface with the COM port interface, analyze measurement data, display results graphically and export data in a variety of formats.

CHAPTER 3
Implementation

3.1 Tools

In this section the language in which the elements of the project are written, the software packages used to write, program, debug, test the software and hardware descriptions for this project are briefly described, including their capabilities and limitations.

3.1.1 Software

Texas Instruments Code Composer Studio IDE

For the MCU, the Texas Instruments Code Composer Studio (CCS) CCStudio comprises a suite of tools used to develop and debug embedded applications. It includes a cross-compiler designed with extensions written explicitly for the MSP430 variant used in this project. The IDE includes a source code editor, project build environment, debugger, profiler, simulators, real-time operating system and many other features.[Instruments a]

The majority of code was written in the C programming language, CCS is used to build, compile and program the MCU via the JTAG interface. Especially useful are the debug and watch functions, which allow program execution to be halted so that the value of variables can be analyzed.

In addition to program code, system registers and ports of the MCU can be read, again especially during program is the ability view the I/O ports, mode, direction.

The free version of CCS is limited to 16KB code size and some features of real-time analysis are disabled.

Quartus II

The Altera Quartus II software is used to compile VHDL code, analysis, place and fit, and assign ports to signals for the variant FPGA chip used in the project.

Quartus II specifically is used to interpret the VHDL code for errors, constraints and possible design issues such timing issues. Upon successful compilation: synthesis, further timing analysis, placement and routing, power optimizations and programming. It achieves these through the following components:ing issues. Upon successful compilation: synthesis, further timing analysis, placement and routing, power optimizations and programming. It achieves these through the following components:

Design Entry: input VHDL code Functional Simulation: ModelSim Synthesis: Placement and Route TimeQuest: static timing analysis PowerPlay: power analysis

Additional debugging support, is provided by SignalTap II logic analyzer and the transceiver toolkit. These are unavailable in the free web edition versions.

ModelSim

Full compilation using Quartus II to a FPGA can not only be a a time consuming process, but is also possible that hardware damage, e.g. through a improperly configured port direction can occur. In the VHDL design process therefore it is normal to first write VHDL code and simulate it. This has the advantage of being able to look at all signals in the design very quickly in order to determine if the design is working properly.

ModelSim is a digital ASIC and FPGA simulation and verification tool from Mentor Graphics and supports VHDL, Verilog and SystemC designs. In this project, all hardware descriptions were written with VHDL and verified in this way, using test benches.

Test Benches

The test bench concept for VHDL is simply a model to generate waveforms with which to test a circuit model. [Rushton 2011] They were used extensively in the project to simulate the response of the ADC and MCU in the design of the FPGA-ADC interface (described later in this chapter).

It was important to consider the simulated and actual delay of the devices simulated with the test bench. For example, the response time of the MCU varies depends on whether it is currently processing an interrupt or not, or if other factors such as port configurations are not considered. The test bench must therefore always used cautiously. On several occasions, delays that had been unknown or not considered were not implemented in the test bench leading to a functional circuit, however incorrect output.

3.2 Microcontroller software

3.2.1 Functional specifications

In the design of the software for the MCU, several requirements were written considering how measurements would be made. In order to characterize a transistor, measurements and commands would have to be sent within short periods of one another, data should "stream". These same requirements are used to determine the effectiveness of the implementation during testing of the system.

The design requirements were determined as follows:

For simplicity and high interoperability between different higher-level software systems, an ASCII interface (rather than pure binary) would be designed. This is to allow the user to connect to the measurement system with the simplest of terminal emulation software with the ability to send commands and receive output.

1. Commands are sent and checked for validity before being processed
2. The ability to send commands via the UI to set measurement system configuration values
3. Configure the DAC voltage levels of each port
4. Configure the switched biasing frequency of each port
5. Configure the duty cycle of each port
6. Configure the ADC sampling windows
7. Configure the FPGA data output rate
8. Ability to put the DAC into power-down mode and return to normal operation
9. Ability to set the group offset, channel offset, group gain and channel gain as well as other configuration registers of the DAC
10. To manual configure the measurement range between "high current (2mA)" and "low current (200μA) modes.
11. The ability to set an automatic range selection mode and a corresponding hysteresis switching mode.
12. The ability to set all each relay in the system and ensure that certain relay switching combinations are disallowed.
13. Test functions to test the speed, output and accuracy of the DAC voltage levels.
14. Safety features to disable current flow across the the transistor would be implemented, disabling current flow by default and allowing configuration of the voltage levels before allowing current to flow through the measurement port.
15. Measurement data should be achieved as a "streaming" output, i.e. the ability to send a command over the USB interface while simultaneously receiving a complete measurement data at a rate of at least 10 times per second (10 Hz).
16. The ability to stop measurement, change voltage levels, and make changes to the sampling window while simultaneously receiving a data stream.
17. Interaction with the hardware design of the FPGA must occur in a predictable and reliable manner.
18. The system should have the ability to calibrate the "zero-point" of each channel. That is when no current is following, the current measurement values read become the reference zero for all future measurements until the next calibration cycle.
19. A unique serial number can be assigned and read through the UI. This is to distinguish between multiple measurement boards connected to the sample higher-level system.
20. The ability to "soft-reset" both the MCU and the FPGA via the UI.
21. Software should be easily modifiable and well documented.

3.2.2 Overview

The main loop of the program checks for the following conditions as depicted in the flow diagram:

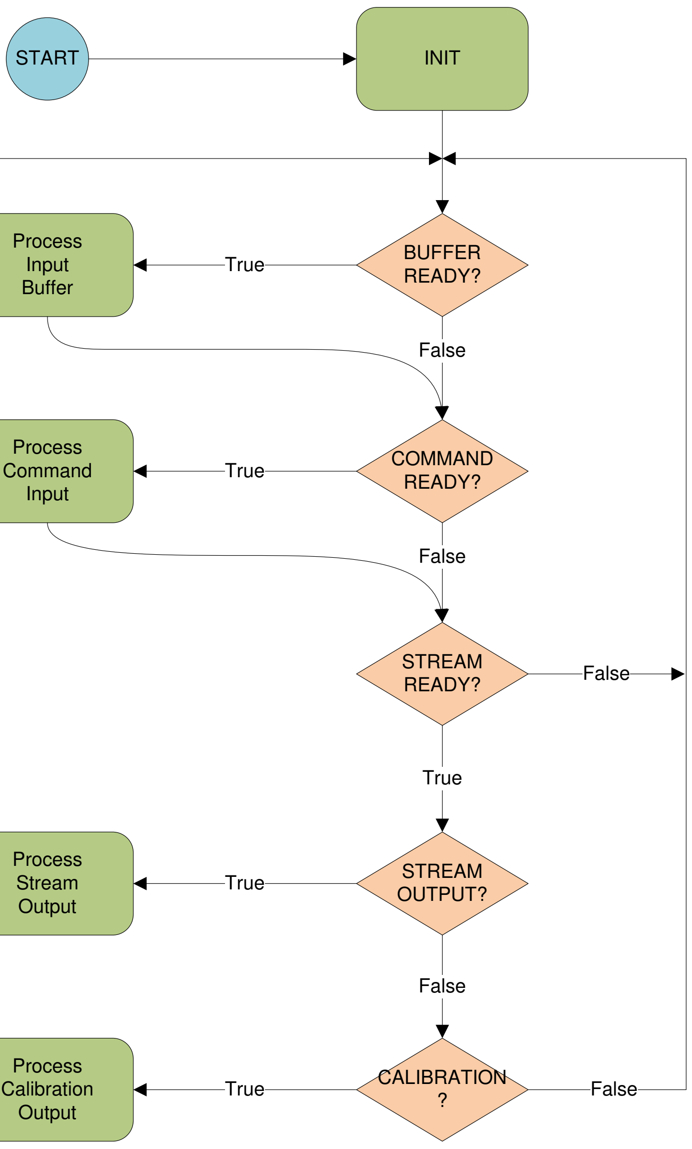


Fig. 3.1: Main operation flow

3.2.3 Power-on initialization (INIT)

When power is applied to the system, the system components are configured and the default values are set and local mode is entered.

The order of initialization is as follows:

1. Initialize system clock
2. Initialize system ports
3. Initialize FPGA interface
4. Initialize DAC interface
5. Serial number retrieval
6. Initialize Serial-USB interface
7. Initialize system interrupts

3.2.4 Disable watchdog

One feature of the MCU is called the watchdog timer, which resets the CPU if certain conditions are not met. This feature is not used in the implementation and must be disabled, otherwise the MCU will automatically reset. The watchdog timer is disabled by setting the watchdog control register during MCU initialization:

```
WDTCTL = WDTPW+WDTHOLD;
```

Initialize system clock

Our hardware includes an external 20 MHz resonator specifically for the MCU as system clock input. The MCU uses a system called the Unified Clock System (UCS) to drive clocking signals. UCS allows two clock references, XT1 and XT2. XT1 can either be driven by the internal low frequency resonator, or an external reference. [Instruments b] XT2 allows for an optional external high frequency resonator. In our configuration, we are disabling the internal XT1 reference and replacing it with the external 20 MHz reference.

The XT1 and XT2 drive three internal clock signals: Auxiliary clock (ACLK), Master clock (MCLK), and Subsystem master clock (SMCLK). These clocks are used to drive peripheral modules such as the SPI bus and serial interface baud rate, and therefore must be carefully configured.

Setting the system clock consists of specified order of operations, which includes configuring the port to which the external crystal is connecting, configuring UCS registers, waiting until the oscillator stabilizes as shown in figure 3.2 [Instruments b].

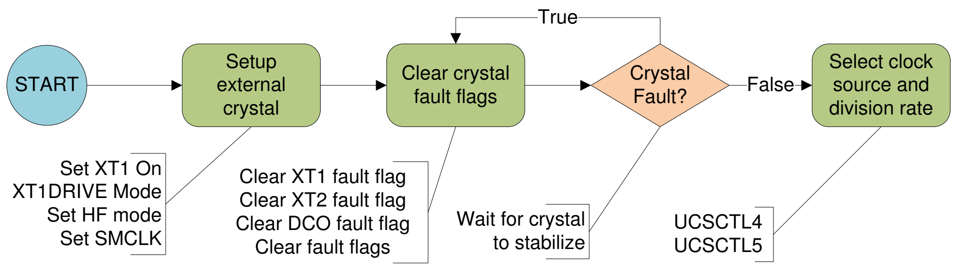


Fig. 3.2: MCU clock initialization

XT1 can be configured for LF mode, which uses the internal 32768 Hz reference. XT1 is driven by an external reference, connected to its peripheral port. XT1DRIVE

Tab. 3.1: MCU port configuration registers

Register	Purpose
PxIN	Input read in from port
PxOUT	Output to write to port
PxDIR	Direction
PxREN	Pull-up/down resistor
PxDS	Drive strength
PxSEL	Port function select (General I/O / Peripheral)

Tab. 3.2: Additional configuration registers for P1 and P2

Register	Purpose
PxIV	Interrupt vector
PxIES	Interrupt edge select (rising, falling edge)
PxIE	Interrupt enable

mode must be set to match the range of the external oscillator[Instruments b]. XT2 and the digitally controlled oscillator (DCO) are not used in this design and are left uninitialized.

Once XT1 is enabled, sometime must be allowed to pass before the oscillator is fully functional. The XT1 fault flag is raised when the oscillator is not running. The fault flags are checked and cleared until they are no fault is raised.

After XT1 has been started, it is selected as the source for internal clock signals ACLK, MCLK and SMCLK in control register UCSCTL4. Clock division is set to 1 (none) in UCSCTL5.

Initialize system ports

The MCU includes multiple I/O ports (P1 - P11), designated Px, where x stands for the port number. They must be configured to match their intended use as a peripheral mode or general I/O [Instruments b]. If a port is used in general I/O mode, port properties must be configured during initialization, these registers are listed in table 3.1 and 3.2). In peripheral mode, some attributes are automatically configured (e.g. direction) depending on the assigned use for that pin [Instruments b].

The port configuration has been carefully chosen to avoid port conflicts with other devices that could lead to excessive current flow and possibly component damage.

Port 1 and 2 have additional capability not available for the other ports:

FPGA interface initialization

Because the MCU can be reset independently from the FPGA and DAC, a software reset has been written to reinitialize FPGA registers during MCU initialization (see section 3.3.7).

From the MCU, the reset is performed by: 1. setting the memory location associated assigned to the FPGA_SW_RESET (FPGA RAM location 29, bit 0), 2. waiting 1 ms, 3. reinitializing FPGA configuration (if not FPGA default). The MCU function written for this purpose is cmd_sys_fpga_init().

Initialize DAC interface

The MCU includes support for the intra-IC serial protocol SPI, before it can be used, the Universal Serial Communication Interface (USCI) module of the MCU must be initialized and configured for the connected device.

The USCI interface consists of two modules, USCI_A and USCI_B, which support difference interfaces in different modes. USCI_A handles the UART, IrDA, LIN, and SPI interfaces, and USCI_B handles I2C and SPI interfaces [Instruments b].

USCI_B itself is further divided into two modules, named USCI_B0 and USCI_B1. Because our DAC supports SPI, and we will also utilize the UART module for the serial/USB communications, the DAC is configured for USCI_B, only one of which is required, USCI_B0.

Initialization of the USCI module is as depicted in figure 3.5 [Instruments b].

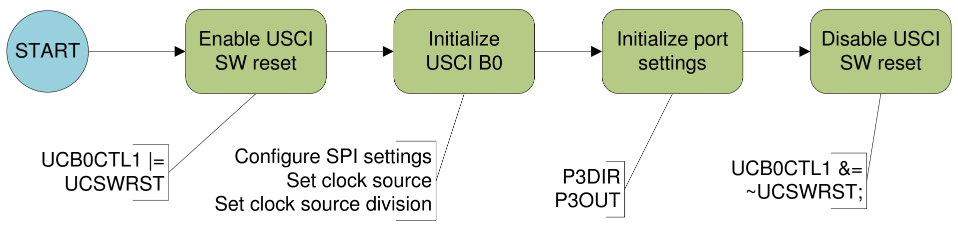


Fig. 3.3: DAC interface initialization

Holding the USCI in SW reset prevents unexpected results on the connected device during initialization. Initialization of the B0 registers configures SPI settings, such as 3-wire/4-wire and master/slave mode.

Although port settings have already been initialized, they are initialized again to ensure the correct settings are applied, in case they may have been inadvertently changed. Upon disabling the USCI SW reset, the SPI interface to the DAC is ready to be used.

Serial number retrieval

Connected to the FTDI USB-serial conversion module is an EEPROM module for storage of USB configuration settings, including the USB serial number. In order to set the system serial number, so that it may be easily queried over the serial interface, it is desirable to read the serial number stored in the EEPROM module.

However, while the serial-USB module is active, the EEPROM is not directly accessible to the MCU. In order to read the EEPROM, which is connected both to the FTDI USB-serial module and the MCU, it is first necessary to but the FTDI USB module in a reset state, at which time MCU port-pins directly connected to the EEPROM module can be accessed.

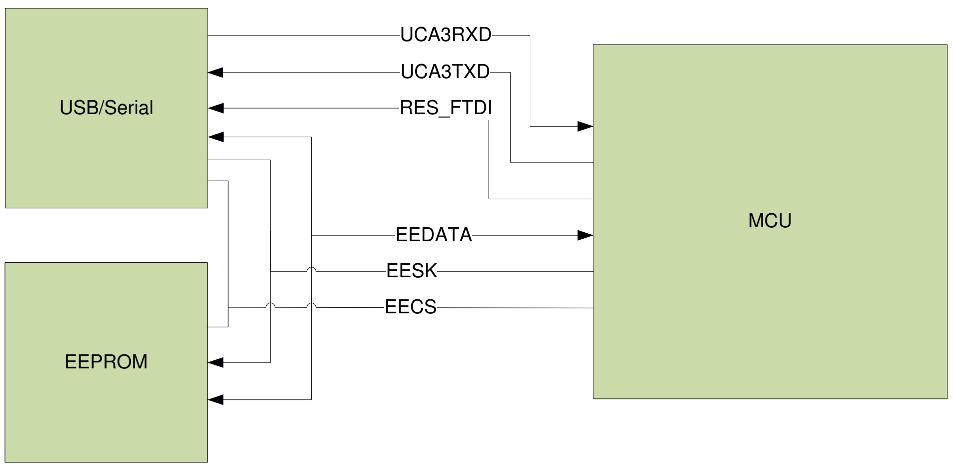


Fig. 3.4: Serial/USB interface initialization

Setting up the MCU I/O ports involves setting P11 pins which are shared both connected to the FTDI module and the EEPROM module. The FTDI module pins connected to the EEPROM module are tri-stated, meaning that when not in a reset condition they are configured to communicate with the EEPROM module. When in a reset condition, they are in a high impedance state, allowing the MCU I/O pins connected to the EEPROM to directly read and write to EEPROM module pins.

When MCU I/O pins connect to the EEPROM module are left in the INPUT direction, which does not result in excessive current flow.

Getting the serial number involves invoking get_eeprom_serial() which performs the following:

1. Set and hold FTDI-USB module in reset
2. Wait for reset condition
3. Configure MCU I/O ports for EEPROM
4. Read EEPROM length and offset of serial number
5. Read EEPROM serial number
6. Reconfigure MCU I/O ports
7. Release FTDI-USB module from reset

Initialize serial-USB interface

Initialization of the serial/USB interface is similar to the DAC interface. In this case, the UART module of USCI_A is used, USCI_A3.

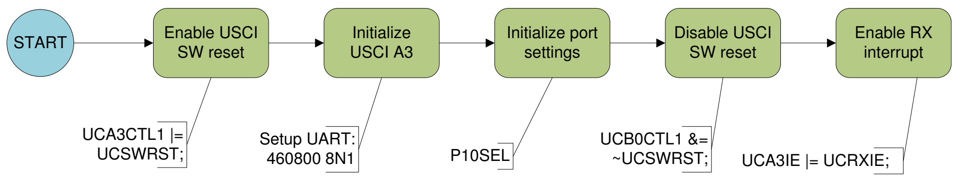


Fig. 3.5: Serial/USB interface initialization

The maximum baud rate supported by the MCU with a system clock of 20 MHz is 460800, this baud rate is detected automatically by the the FTDI USB-serial conversion module[Ltd 2011]. This baud rate is fixed and automatically detected by the FTDI module, set by configuring the MCU registers UCBRx, UCBRSx, UCBRFx [Instruments b].

Using the baud rate (r_baud), the number of characters per second (r_{cps}) this speed supports can be calculated. 1 ASCII character is 8 bits, plus 1 start bit, and 1 stop bit, 10 bits per character [Luecke 2005].

$$r_{cps} = \frac{b_r}{10} = \frac{460800}{10} = 46080 \text{ characters per second} \tag{3.1}$$

To meet the minimum output rate of 10 measurements/second, we can find the maximum number of characters per measurement output line (cpm):

$$\frac{r_{cps}}{rmeas} = \frac{46080}{10} = 4608 \text{ characters per measurement (max)} \tag{3.2}$$

Initialize system interrupts

The final stage of initialization is to set the Global Interrupt Enable (GIE) bit, which enables maskable interrupts, which allow for among others, Serial-USB interface interrupts (i.e. command input via serial).

3.2.5 FPGA Interface

All measured data is acquired by the ADC which is connected to the FPGA, where they are stored in memory and signaled ready for retrieval (for more detail see section FPGA implementation). In order to reliably retrieve the measurement data a memory bus was designed with which the MCU can communication with the FPGA.

The MCU and FPGA are physically connected at MCU ports 1, 4, 2, and 8 from which an address bus, control bus, and data bus have been conceived. In the new designed bus, the MCU is designated the "master" and always retrieves data from the "slave", the FPGA. To distinguish the direction of input/output I/O, the names MISO (Master-in, Slave-out) and MOSI (Master-out, Slave-in) were given to the read and write procedures developed for the interface, similar to the naming convention used for the SPI interface.

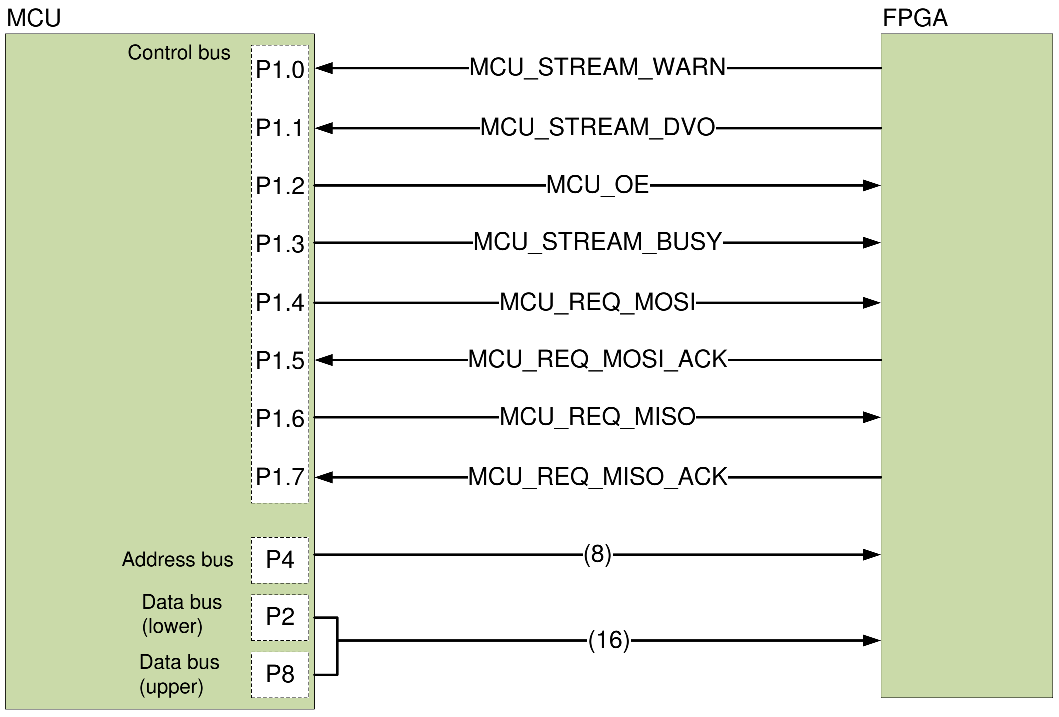


Fig. 3.6: MCU-FPGA interface

Control bus

The control bus activates processes on the FPGA and interrupts program flow on the MCU. Port 1 or port 2 of the MCU has must be used since only these two ports have pin-specific interrupt capability that allows the signals from the FPGA to interrupt MCU program flow.

The following control signals have been defined:

STREAM_WARN Set by the FPGA when streaming data is incoming from the ADC/Filter combination faster than it is being retrieved by the MCU. The state of this pin is checked each time data is retrieved in streaming mode and an warning character (e.g. ASCII "!") during streaming measurements.

MCU_STREAM_DVO Stream Data Valid Output - this is pin is raised by the FPGA to signal (i.e. trigger) the MCU that configured trigger data is ready for retrieval. P1.1 is configured as an interrupt on the rising edge of this signal. The P1.1 interrupt is enabled when in STREAM or CALIBRATION mode.

MCU_OE used to control the direction of the data bus. The bi-directional data bus direction is set to either an output or input on the FPGA side based on that state of this pin.

MCU_STREAM_BUSY is set by the MCU to signal to the FPGA that data from memory is being retrieved. This signal can be configured to "lock" the memory from being changed until the signal is cleared. This is due to some values being stored in multiple memory locations, to help ensure that retrieved data is consistent.

MCU_REQ_MOSI set by the MCU when the master is ready to send data. Confirms that the address bus is ready to be read.

MCU_REQ_MOSI_ACK set by the FPGA in response to MCU_REQ_MISO. Confirms that the values on the data bus has been stored at the address on the address bus.

MCU_REQ_MISO set by the MCU to signal that address set on the address bus being requested.

MCU_REQ_MISO_ACK set by the FPGA in response to MCU_REQ_MISO. Confirms that the values on the data bus is the contents of address set on the address bus.

Address bus

The address bus is an 8-bit bus, on which the address to read, or the address to which to write is set by the MCU. The address refers to a memory location on the FPGA, which contains e.g. ADC measurement values, configuration data, and triggering information.

For a complete table of assigned address values on the FPGA, refer to appendix app:fpga.

Data bus

Two options considered for the data bus were, either two 8-bit data buses, one for MISO data, the second for MOSI data. However, these would result in a more complicated implementation since data and a larger address space than what the the 8 bit address bus could provide would be required. Therefore, the tri-state buffering feature of the FPGA and the ability to switch the direction of the address pins during runtime on the MCU were used to create one 16-bit bidirectional bus. Since ports of the MCU are 8 bit, the data bus has been divided into the 8bit upper-byte (port 8) and an 8-bit lower-byte (port 2) of the 16-bit data word.

The direction of the data bus on the FPGA was set by a new control signal MCU_OE (output enable).

3.2.5.1 Read operation (MISO) handshaking

A four-signal handshaking scheme was created to transfer data between the MCU and FPGA in order to synchronize the two clock domains of the MCU and FPGA, and because the bus is parallel, the lengths of traces are not all equal, a handshaking system is utilized to ensure that all bits on all pins are set and ready to be read or written.

A read operation is controlled by the master (MCU). The operation occurs in three stages: setup, request and clear. The setup phase is entered first when the MISO ACK (from FPGA) is cleared. This is to ensure that a previous read operation has fully completed. In the setup phase, the address to be read from the FPGA is setup on the address bus and the OE bit is cleared to ensure that the data bus is in the MISO state (MCU data ports set to input, FPGA data pins set to output). After setup, in the request phase, the MISO request bit is set confirming a valid address on the address bus and proper configuration of the data port. This triggers the FPGA to set the requested memory location on the bus, followed by a MISO acknowledgment (ACK) from the FPGA, signaling that bits on the data bus are valid and ready to be read. After the data has been processed by the MCU, MISO REQ is cleared signaling a complete read. The MISO REQ ACK is in turn set low and the MISO process has been completed. A message sequence diagram depicting a MISO read operation is displayed in figure 3.7.

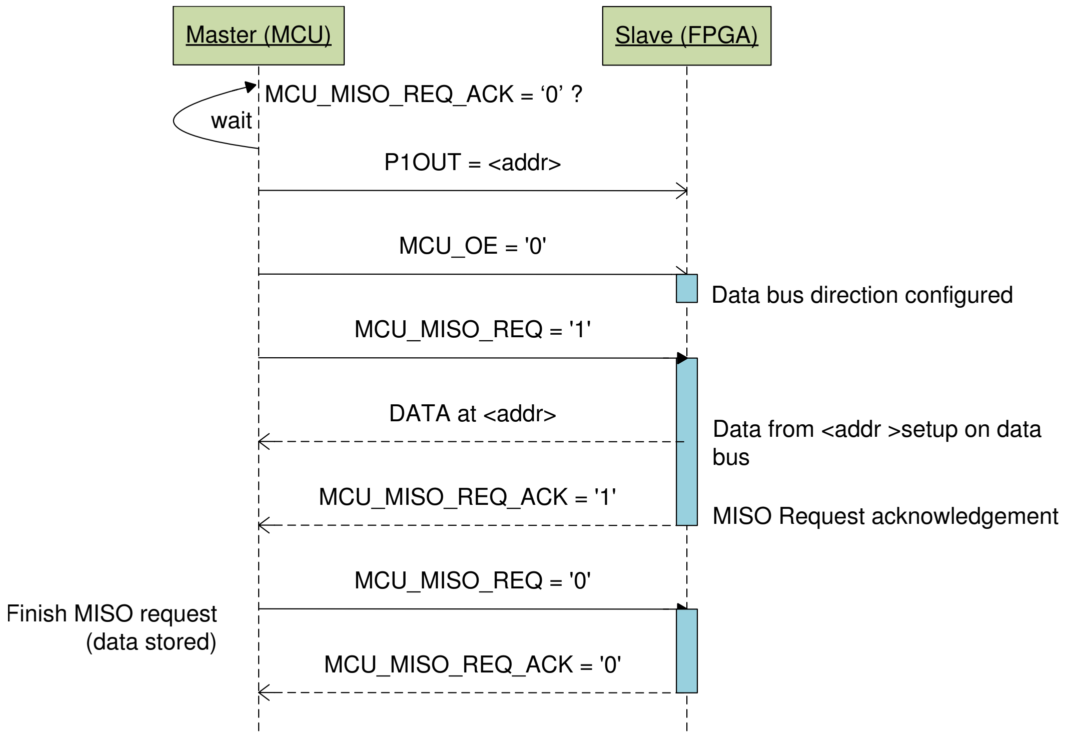


Fig. 3.7: MISO operation

3.2.5.2 Write operation (MOSI) handshaking

In a MOSI operation, the MCU writes data to memory on the FPGA at a specified memory location. As in a MISO operation, this occurs in three phases: setup, request, clear. In setup, first OE is set, the FPGA reacts by configuring the tri-state buffers of the data port as an input. For a short time, the MCU and FPGA data ports are both configured as inputs, that is until the MCU data port is reconfigured as an output. Also in the setup phase, the address to which data is to be written is setup, and the most and least significant bytes of the word are setup on the data bus. The MOSI operation is displayed in figure 3.9.

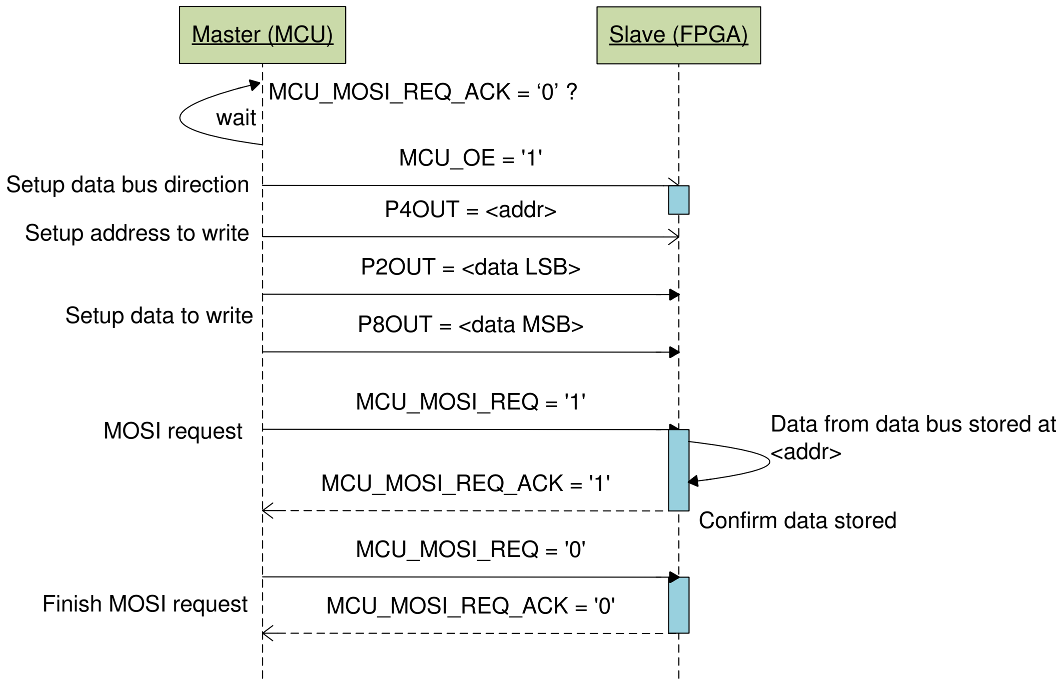
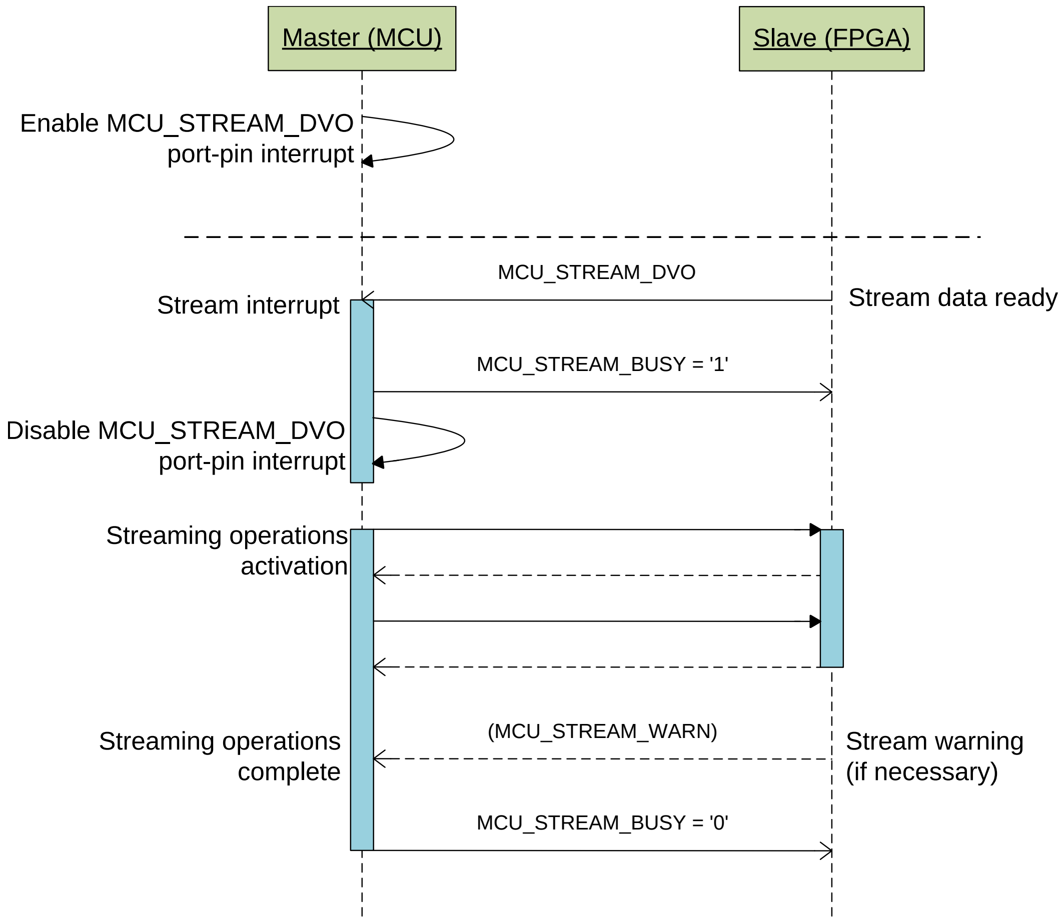


Fig. 3.8: MOSI operation

Streaming operations

In streaming and calibration modes, data is retrieved based on an incoming data valid output signal (DVO) from the FPGA. This DVO bit is configurable and matched to the configured triggered ADC filter output. This bit is always set when the trigger is configured on the FPGA. When the system mode is set to STREAM or CALIBRATION, the port-pin interrupt for DVO is enabled. On the rising edge of DVO, an interrupt sets MCU_STREAM_BUSY, disables the port-pin interrupt temporarily, and sets the system variable flag STREAM_READY $\bar{1}$.

When the main loop of the MCU checks this flag, streaming or calibration functions are performed as needed. At the conclusion of the stream, stream_busy is cleared, the interrupt is cleared, and the port-pin interrupt for DVO is re-enabled.



Master (MCU)
Slave (FPGA)
Enable MCU_STREAM_DVO port-pin interrupt
MCU_STREAM_DVO
Stream interrupt
Stream data ready
MCU_STREAM_BUSY = '1'
Disable MCU_STREAM_DVO port-pin interrupt
Streaming operations activation
(MCU_STREAM_WARN)
Streaming operations complete
Stream warning (if necessary)
MCU_STREAM_BUSY = '0'

Fig. 3.9: Streaming MISO operation

3.2.6 DAC Interface

The MCU physically interfaces with the DAC and is completely controlled by the MCU, as opposed to the ADC which is physically connected to the FPGA and can only indirectly interface with the ADC.

The built-in SPI protocol is utilized by configuring the SPI specific pins (P3.0, 3.1, 3.2, 3.3) as in *peripheral* mode. This uses the pins in the pre-configured mode assigned to these pins. This timing of the SPI interface clock, SIMO, and SOMI pins are automatically by reading and writing to the special function register RXBUF and TXBUF.

Pins P3.4, P3.5, P3.6 and P3.7 are left as general I/O and their logic is controlled by functions accessible via the user interface. CLR clears previously configured DAC registers, RST resets the DAC, LDAC loads the DAC output registers and WAKEUP is a type of sleep function.

In order to program the DAC, functions were written specifically for DAC read-/write operations. Through the user interface, these are:

Tab. 3.3: MCU-DAC Interface

Pin	MCU Port	Signal	Purpose
33	P3.0	SPI DAC STE	SPI protocol slave select
34	P3.1	SPI DAC SIMO (SDI)	SPI protocol slave-in, master-out
35	P3.2	SPI DAC SOMI (SDO)	SPI protocol slave-out, master-in
36	P3.3	SPI DAC CLK	SPI protocol communications clock
39	P3.4	$\overline{CLR}$ DAC	Clear DAC
40	P3.5	$\overline{RST}$ DAC	Reset DAC
41	P3.6	$\overline{LDAC}$ DAC	Load DAC
42	P3.7	DAC WAKEUP	DAC wake-up (low-power)

mem:dacget:<addr> Get DAC register at DAC register address <addr>

mem:dacset:<addr>:<val> Write the value <val> at DAC register address <addr>

Internally, the a request to read DAC memory calls the functions depicted in figure 3.10.

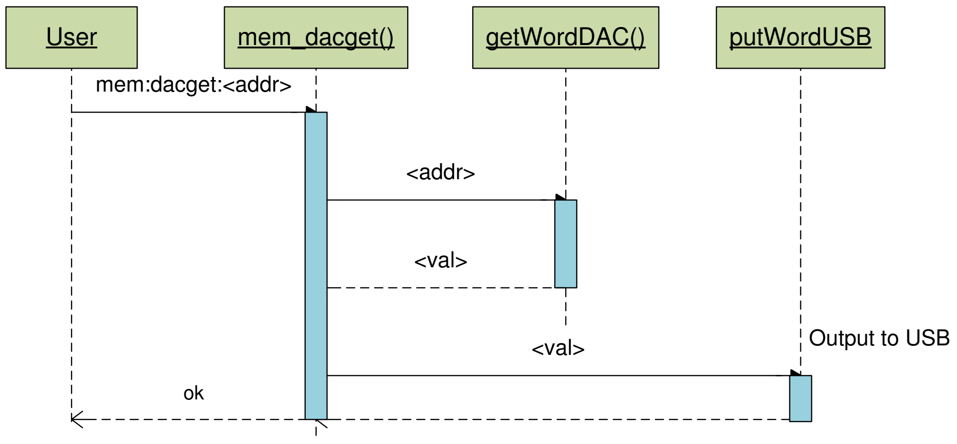


Fig. 3.10: mem:dacget function calls

Memory retrieval is somewhat involved due to the packet delay in retrieving register data. Packets to the DAC are composed of 24 bits, as described in the data sheet. In standalone mode, the first 24-bit packet specified in the first 8 bits, the address to be read, followed by dummy data. In the second 24-bit packet, a NOP command is written, while simultaneously data from the address of the last packet is read in [Instruments 2009].

In order to retrieve this data, the process of reading and writing simultaneously is split into two phases: phase 1 is the input word specifying the register to be read, phase 2 writes NOP data and reads data from the selected register.

Note: during development of this function, a possible MCU (MSP430F5438 rev. A) silicon bug was discovered and a workaround was needed. The details can be found in section 6.3.

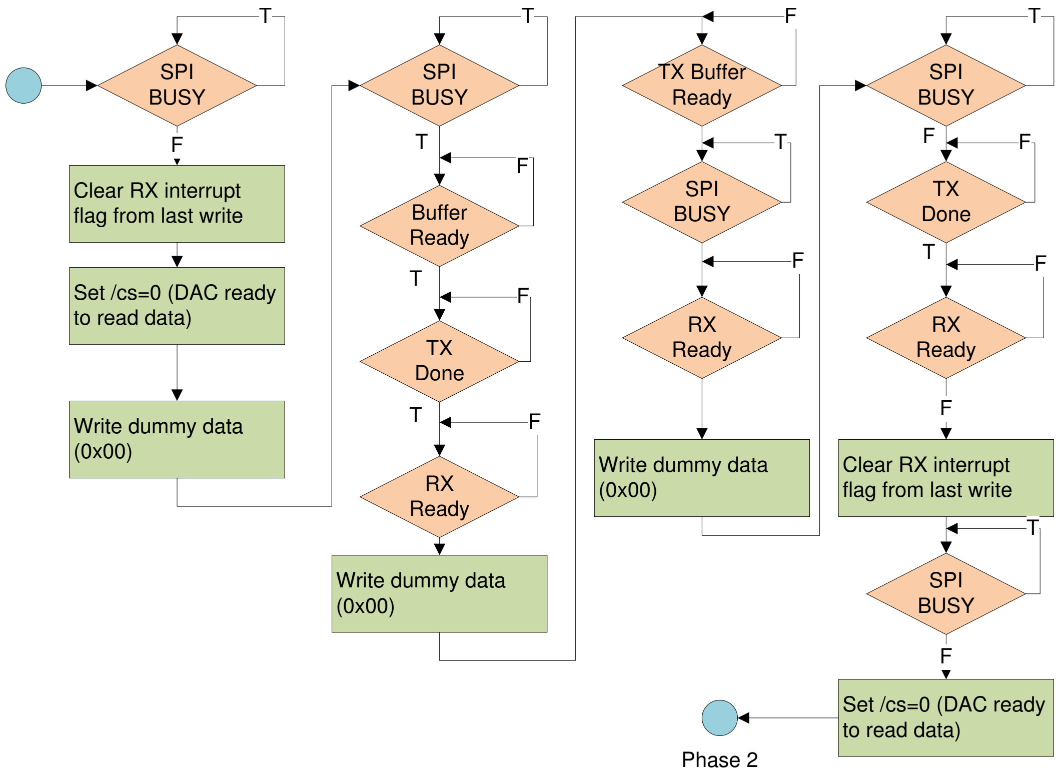


Fig. 3.11: Reading a DAC register, getWordDAC() phase 1

Writing to the DAC, e.g. when programming the INPUT_CODE to set the output level, is less complicated because only one 24-bit packet needs to be set. The UI function calls are depicted in figure 3.13

Internally, putword_DAC() writes the bytes to the MCU SPI buffer UCB0TXBUF, sets DAC Chip Select, and finally loads the value with LDAC as specified in the DAC8718 data sheet. The process is displayed in 3.14.

3.2.7 Serial-USB Interface

As with the DAC SPI interface, the MCU has built-in serial functions which are assigned to specific ports, which allow simplified serial access.

The MCU physically connect to the FTDI serial-USB conversion chip. The ports connected to this chip are:

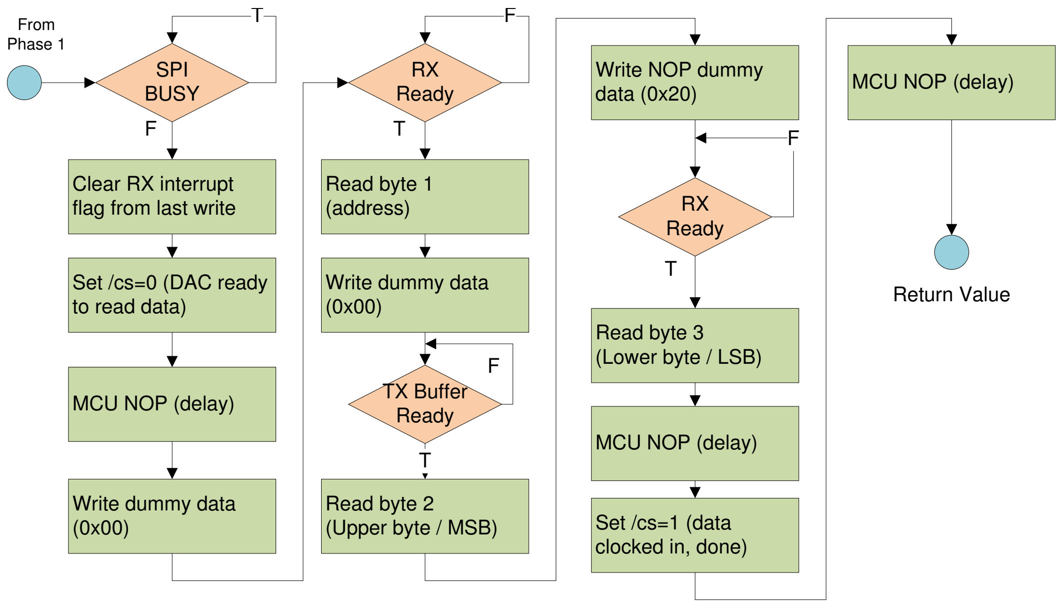

From Phase 1
SPI BUSY
T
F
Clear RX interrupt flag from last write
Set /cs=0 (DAC ready to read data)
MCU NOP (delay)
Write dummy data (0x00)
RX Ready
F
T
Read byte 1 (address)
Write dummy data (0x00)
TX Buffer Ready
F
T
Read byte 2 (Upper byte / MSB)
Write NOP dummy data (0x20)
RX Ready
F
T
Read byte 3 (Lower byte / LSB)
MCU NOP (delay)
Set /cs=1 (data clocked in, done)
MCU NOP (delay)
Return Value

Fig. 3.12: Reading a DAC register, getWordDAC() phase 2

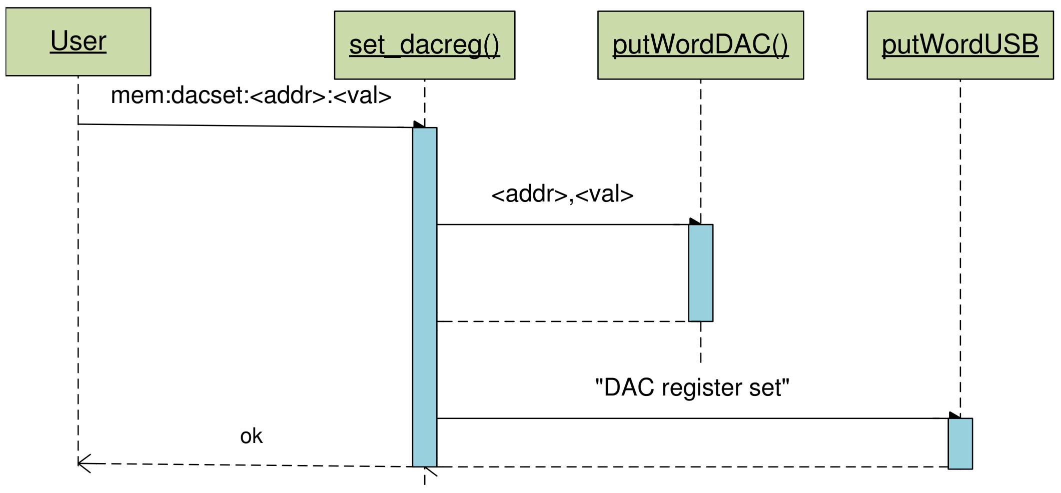


Fig. 3.13: UI command "mem:dacset" function calls

MCU port	Description	Purpose
P10.4	UCA3TXD	MCU data to FTDI Serial-USB
P10.5	UCA3RXD	FTDI serial-USB data to MCU

3.2.8 User interface (External view)

The user interface operates by interpreting ASCII input sent over the Serial-USB interface. User commands are divided into six main divisions, each of which is

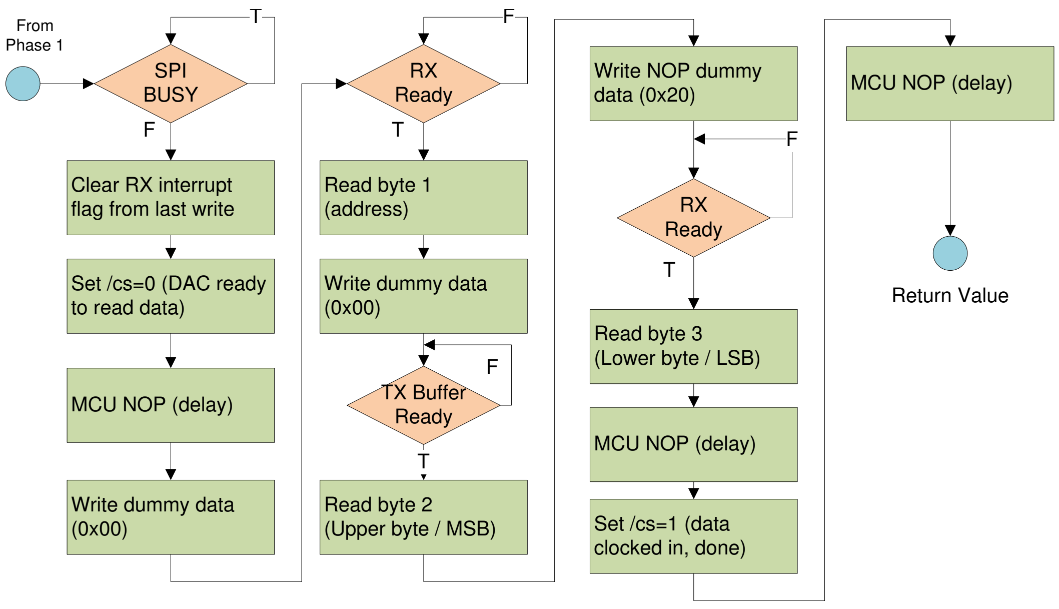

From Phase 1
SPI BUSY
T
F
Clear RX interrupt flag from last write
Set /cs=0 (DAC ready to read data)
MCU NOP (delay)
Write dummy data (0x00)
RX Ready
F
T
Read byte 1 (address)
Write dummy data (0x00)
TX Buffer Ready
F
T
Read byte 2 (Upper byte / MSB)
Write NOP dummy data (0x20)
RX Ready
F
T
Read byte 3 (Lower byte / LSB)
MCU NOP (delay)
Set /cs=1 (data clocked in, done)
MCU NOP (delay)
Return Value

Fig. 3.14: Writing to a DAC register, putWordDAC()

further divided into commands that control all elements of the system, as shown in figure 3.15.

Each of these branches break down further to either direct system commands as in the CMD branch (figure 3.16) or commands that take input arguments, as is the case with the MEAS branch (figure 3.17).

Each command has an associated number of valid command parts, i.e *meas:fet* (two input *parts*) is not a valid command, nor is *meas:fet:a* (3 parts), however *meas:fet:a:5* is a valid command (4 parts). This was done by setting a counter which stores the current number input commands parts and comparing in before calling the associated function.

Numerical input arguments are converted from ASCII string to an unsigned long using the library function strtoul(). This was done because of the functions built in ability to handle decimal, octal and hexadecimal input. Hexadecimal input begins is prefixed by 0x; octal input is prefixed by 0, and the rest is decimal. If no valid conversion could be performed, a zero value is returned [Peter Prinz 2006].

A complete command may exist of 1 to 4 parts and is entered in the following format:

<part1> <part2> <part3> <part4> <LF>

Where LF is the line feed sent by the higher-level system For a complete listing of UI input commands and a their usage, refer to appendix, section B.

3.2.8.1 System modes

To distinguish between manual UI command input, higher-level system automated input, streaming output and calibration modes, the microcontroller was programmed

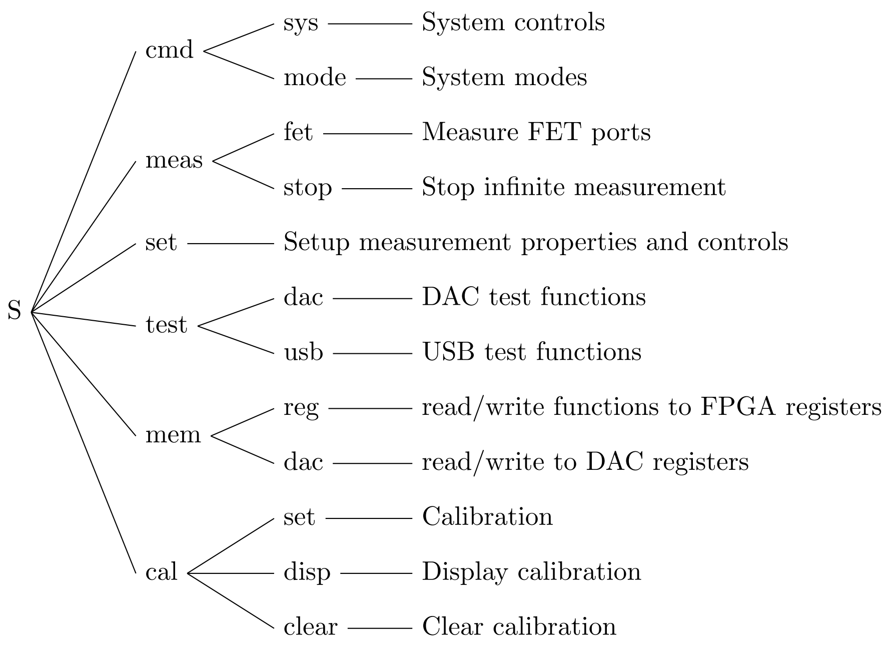

S
cmd
sys — System controls
mode — System modes
meas
fet — Measure FET ports
stop — Stop infinite measurement
set — Setup measurement properties and controls
test
dac — DAC test functions
usb — USB test functions
mem
reg — read/write functions to FPGA registers
dac — read/write to DAC registers
cal
set — Calibration
disp — Display calibration
clear — Clear calibration

Fig. 3.15: User command branches

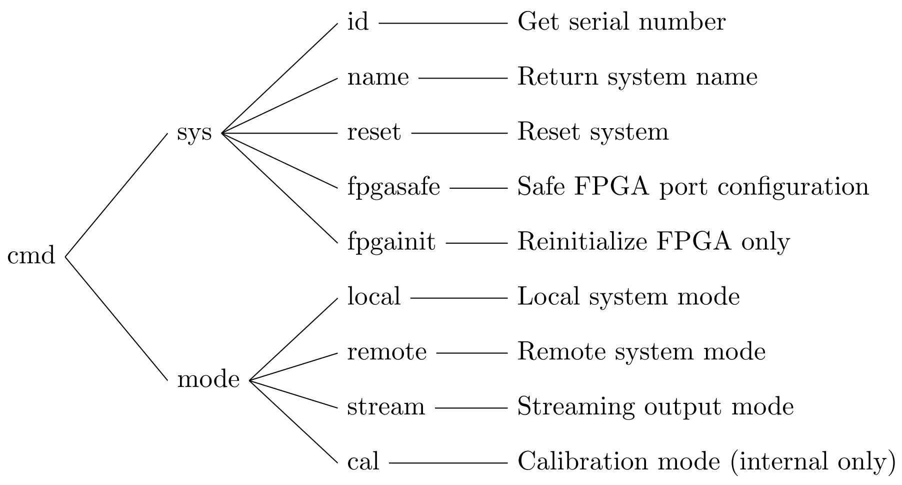

cmd
sys
id — Get serial number
name — Return system name
reset — Reset system
fpgasafe — Safe FPGA port configuration
fpgainit — Reinitialize FPGA only
mode
local — Local system mode
remote — Remote system mode
stream — Streaming output mode
cal — Calibration mode (internal only)

Fig. 3.16: CMD branch

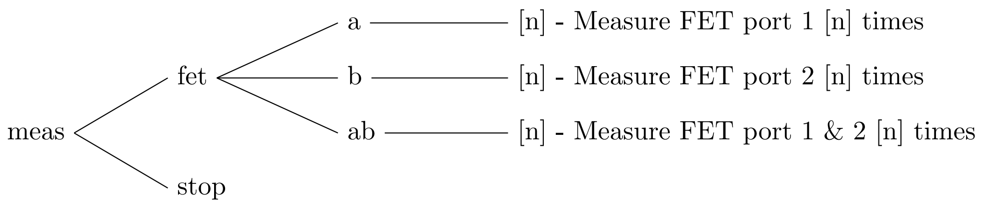

meas
fet
a — [n] - Measure FET port 1 [n] times
b — [n] - Measure FET port 2 [n] times
ab — [n] - Measure FET port 1 & 2 [n] times
stop

Fig. 3.17: MEAS branch

to operate in four modes of normal operation, these are called: local, remote, stream, and calibration.

Local mode (LOCAL)

After power-up initialization, the user interacts in "local" mode by default. In this mode all ASCII text command written to the terminal are echoed back to the terminal.

Remote mode (REMOTE)

Remote mode is similar to the local mode, however is intended for interfacing with higher-level system program in which the commands sent are known and only the responses are of interest. In this mode text that is sent to the MCU is not echoed back to the user in order to reduce the about of traffic over the interface.

Measurement modes STREAM and CAL

Two measurement modes: streaming output and calibration Interrupts and Triggers Stream warning memory lock bit on fpga

3.2.8.2 Input buffer

Incoming text are stored in a ring buffer, a flag INPUT_READY is set and command interpretation and execution occurs. In order to achieves this, an incoming character triggers an interrupt on the UCSIA3 interrupt vector, which is handled by the interrupt handler USCI_A3_VECTOR. Interrupt vector 0x02 is raised by an incoming character, RXIFG.

The interrupt handling is kept as simple as possible to allow to avoid long processing time in the interrupt, on receiving (RX) a character, the interrupt is handled as depicted in figure 3.18.

In order to send commands text quickly, any ASCII text sent to the MCU triggers an interrupt on the MCU; the received character is stored in a circular buffer and processed by the MCU after the interrupt exits. This allows incoming text (and complete commands) to be sent via the UI faster than they can be processed.

When the the read pointer falls behind the write pointer, buffering handling begins. Because commands are loosely based on the input style of SCPI (IEEE-488.1,488.2 / IEC/60488-1, -2) [Consortium 1999], consisting of ASCII textual strings, format and are divided into "parts". Each command part defines the command more specifically. Commands are terminated by an ASCII 'CR' (Hex: 0x0D) or by semi-colon ";" (Hex 0x3B).

Commands are processed by comparing the READ and WRITE pointers in the main process. When READ WRITE then command processing begins. The buffer scans each character in the ring buffer until a command part terminator is found. Characters that are non alphanumeric (A-Z, a-z, 0-9) are not stored in the command array. Upper-case letters are converted to lower case before being stored to ease command interpretation. This is repeated for each command part until the

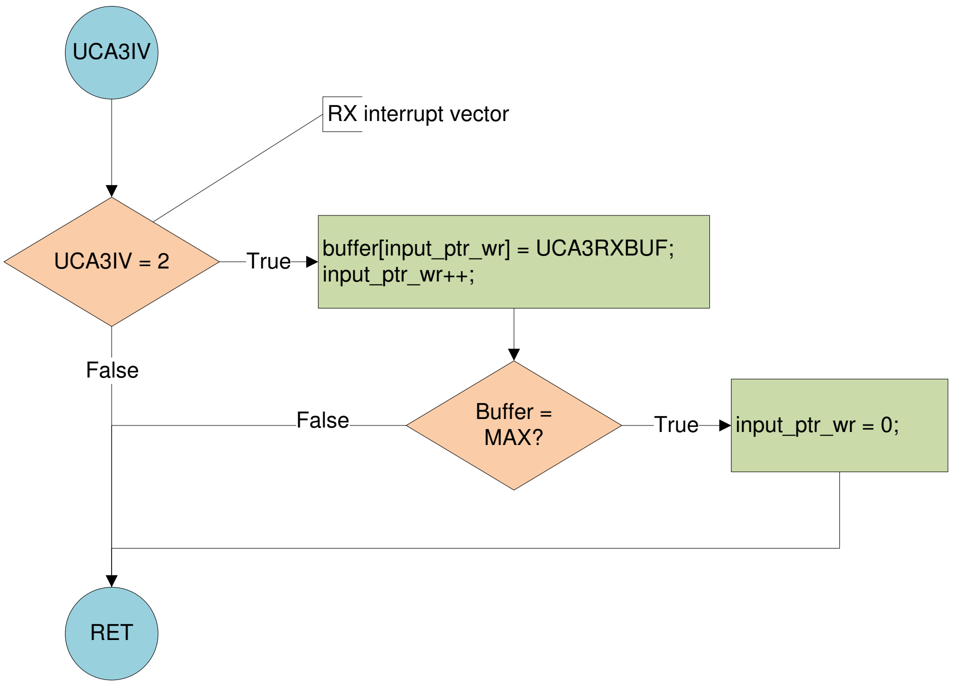


Fig. 3.18: Character input interrupt handling

command terminator is found, at which time a command-ready flag, to signal an available command.

The process is depicted in figure 3.19.

3.2.8.3 Command buffer

Once a command terminator has been found, a flag is raised to signal command interpretation. Similar commands are grouped in a tree-like format, see figure 3.15. In this design, there is a trade-off between source code efficiency and readability. The tree-like design has the advantage of being easily modifiable, while requiring more code space on the MCU.

3.2.8.4 Measurement mode

Measurement mode is an extension of REMOTE mode, and is entered as the result of system mode be setting to STREAM or CAL and an incoming STREAM_DVO interrupt setting the STREAM_READY flag, as depicted in the figure.

3.2.8.5 STREAM mode

The streaming output occurs once per iteration of the main loop, in this way, new incoming commands are always processed between measurement values, therefore allowing parameter adjustments during endless measurements.

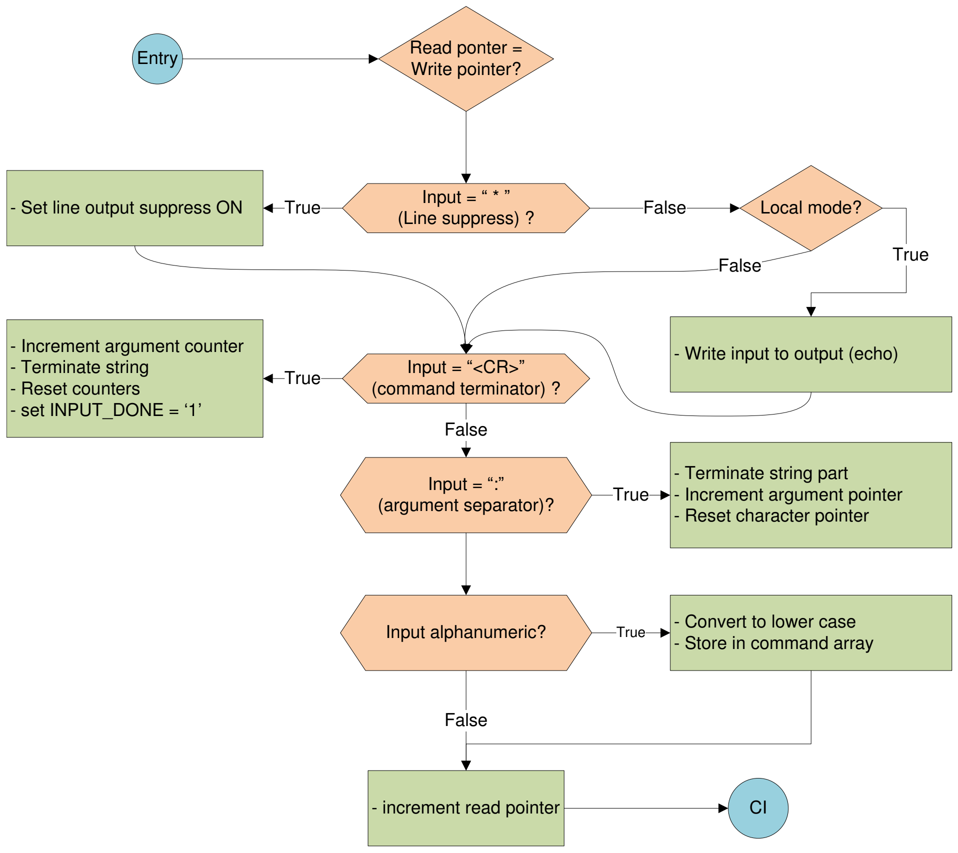

Entry
Read ponter = Write pointer?
Input = " * " (Line suppress) ?
True
- Set line output suppress ON
False
Local mode?
True
False
- Write input to output (echo)
Input = "<CR>" (command terminator) ?
True
- Increment argument counter
- Terminate string
- Reset counters
- set INPUT_DONE = '1'
False
Input = ":" (argument separator)?
True
- Terminate string part
- Increment argument pointer
- Reset character pointer
Input alphanumeric?
True
- Convert to lower case
- Store in command array
False
- increment read pointer
CI

Fig. 3.19: Command array

When the output flag is set (by configuration of the trigger and an incoming ready flag from the FPGA), it is handled as follows.

Delay counter

A delay counter is first checked before any output values are retrieved or displayed. The output delay counter discards incoming values that would otherwise be not useful for measurement values. Such a situation arises when the sampling window is changed and the filter values have not yet stabilized (filter group delay)[Mitra 1993].

This value is not configurable, but is easily changed in the source code define DEFAULT_FILTER_DELAY , and is set by default to 3 (the sampling delay of the 1st order CIC filter, see section CIC filter implementation, section 3.3.5).

Two modes of streaming measurement output are available, infinite and finite. For the finite output mode, an internal variable is set and remains set to the number of output values desired. Internally a second variable counts through the iterations of the measurement cycle. In infinite output, an internal variable is set and remains set to zero. On each iteration of the main loop, this cycle variable is checked. As long is it remains zero, FET measurement values are retrieved and output. To stop

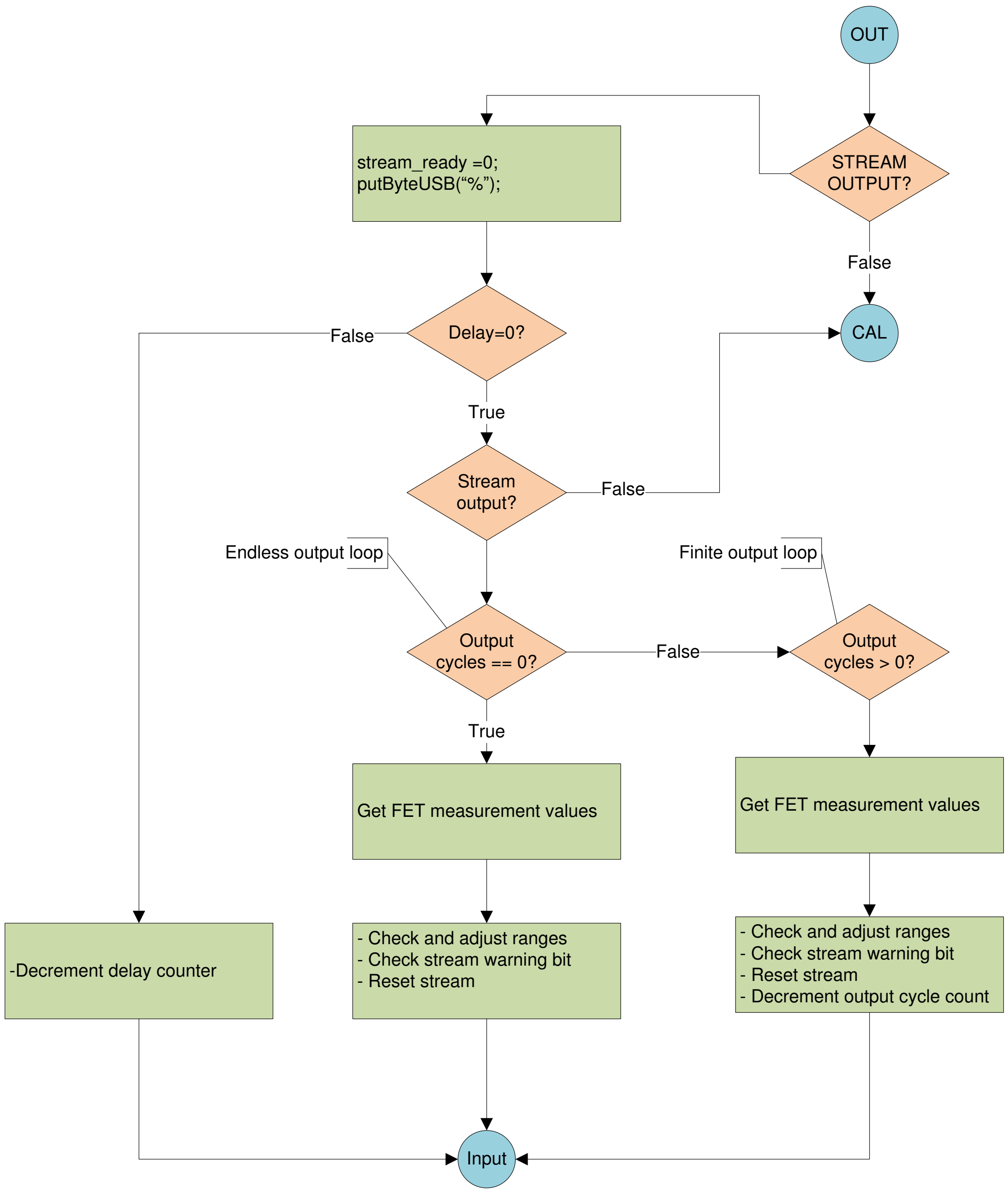


Fig. 3.20: Measurement mode output

measurement in infinite output mode, the cycle counter is set to 1, which results in one additional measurement before the measurement cycle ends, as depicted in figure 3.21.

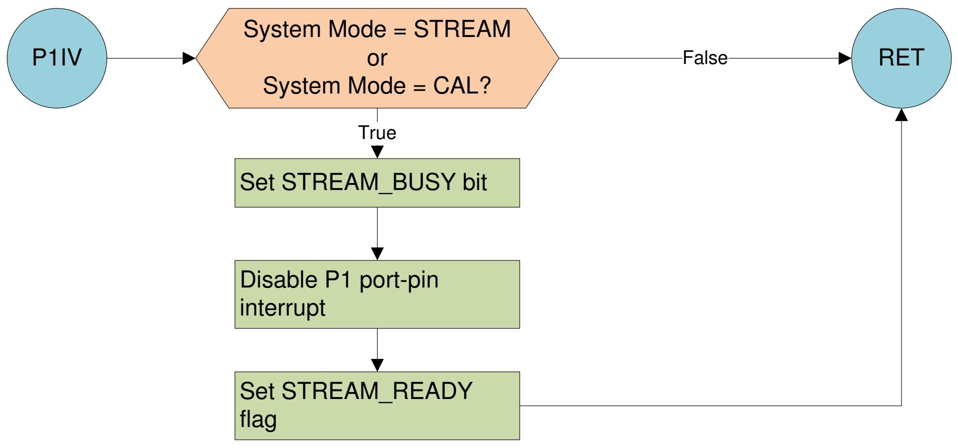


Fig. 3.21: Measurement STREAM_DVO interrupt handling

Get FET measurement values and auto-trigger

Getting FET measurement values involves querying the memory locations on the FPGA for the correct values. This is done internal via the function meas_fet().

The get FET measurement process is complicated due to automatic triggering, i.e. retrieving only the port and port-filter that has new data, and automatic range adjustment. The process is depicted in the figure 3.23.

mem_get_filter()

The mem_get_filter() function called in meas_fet() gets the most and least-significant bytes of the called port/filter from defined memory locations, shifts them appropriately and returns a value.

RAW output mode

Display formatted is an abstraction of the filter value retrieval process. When in RAW mode, the hexadecimal value returned by mem_get_filter() are displayed in CSV format (values separated by semicolon, lines terminated by carriage return).

Formatted mode current calculation

In formatted output, the actual values are calculated and displayed in CSV format. The calculation of the current depends on the range (i.e. the reference resistance). Because the output is in twos-complement format, numbers of values around zero are not exactly symmetric. The 16-bit ADC requires one bit for the sign bit, giving a range of ADC values from -32768 to +32767.

16-bit values are returned signed, effectively 2^{15} positive value including zero and 2^{15} negative values. For the high-current range, a 2500Ω resistance is used. Range is -32768 to 32767.

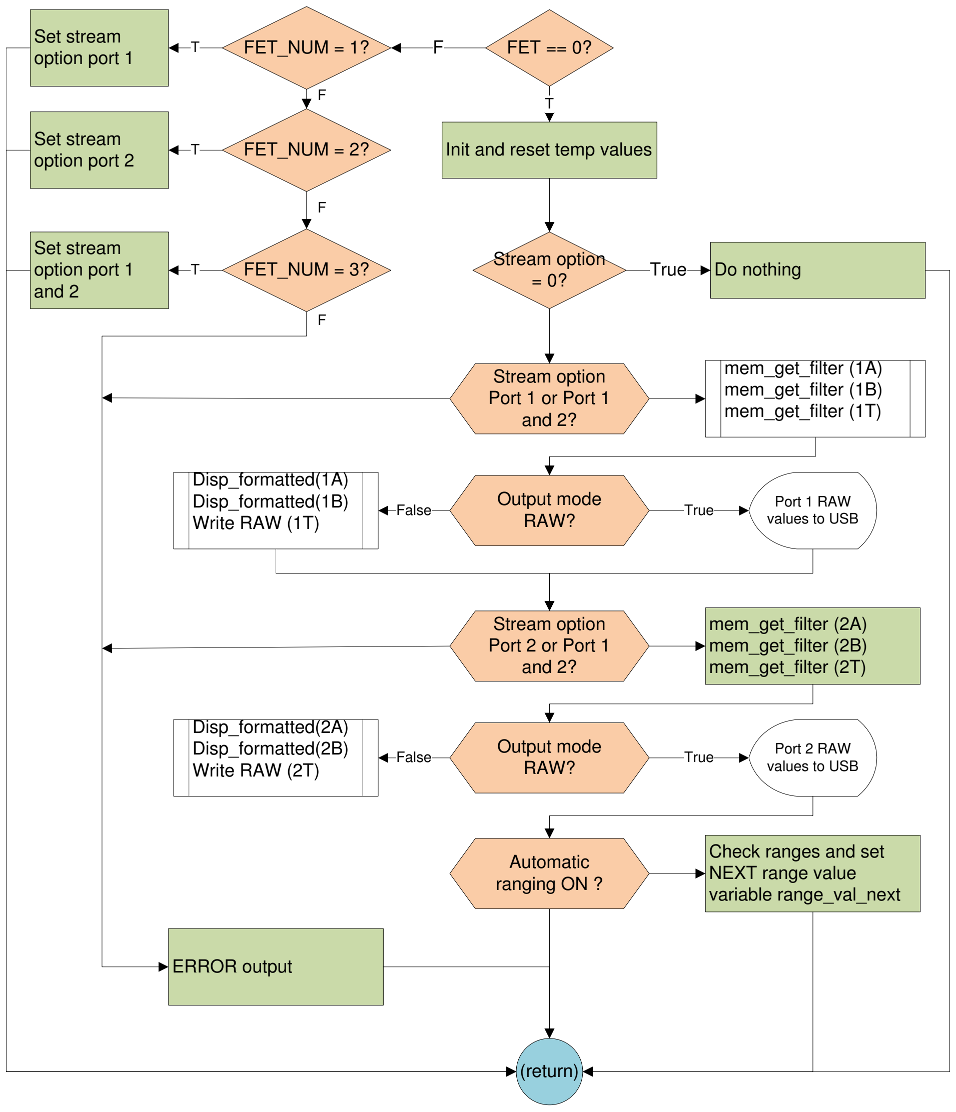

Set stream option port 1
FET_NUM = 1?
FET == 0?
Set stream option port 2
FET_NUM = 2?
Init and reset temp values
Set stream option port 1 and 2
FET_NUM = 3?
Stream option = 0?
Do nothing
Stream option Port 1 or Port 1 and 2?
mem_get_filter (1A) mem_get_filter (1B) mem_get_filter (1T)
Disp_formatted(1A) Disp_formatted(1B) Write RAW (1T)
Output mode RAW?
Port 1 RAW values to USB
Stream option Port 2 or Port 1 and 2?
mem_get_filter (2A) mem_get_filter (2B) mem_get_filter (2T)
Disp_formatted(2A) Disp_formatted(2B) Write RAW (2T)
Output mode RAW?
Port 2 RAW values to USB
Automatic ranging ON ?
Check ranges and set NEXT range value variable range_val_next
ERROR output
(return)
T
F
True
False

Fig. 3.22: mcu meas_fet()

Therefore the calculation of the current is as follows:

Low current

$$i_{meas} = value_{ADC} * 1LSB_{ADC} * 32768 \quad (3.3)$$
$$= value_{ADC} * 6.103516nA \quad (3.4)$$

with i_{meas} the measured current value, $value_{ADC}$ the value returned by the ADC, and LSB_{ADC} the step size of the ADC (see 4.1.1).

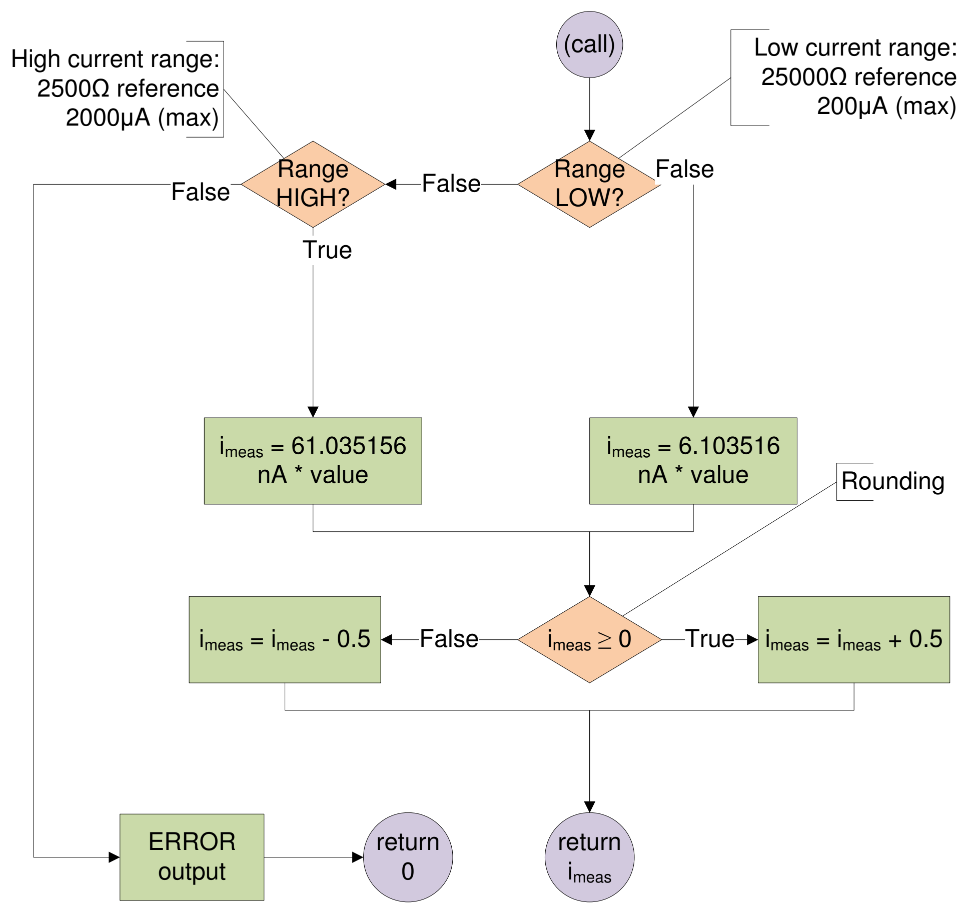


Fig. 3.23: Display calculated, formatted values disp_formatted_I()

In high current mode, the step size is 61.035156 nA (see 4.1.1).

$$i_{meas} = value_{ADC} * \frac{2000\mu A}{32768} \tag{3.5}$$

$$= value_{ADC} * 61.035156nA \tag{3.6}$$

In the low-current range, a 25000Ω resistance is used, maximum input voltage is ±5V, therefore current can range from -200 to +200 μA, a total range of $400\mu A$. ADC values are typical expressed in LSBs, which is the step resolution based on the resolution of the ADC and the full scale value [Kester 2008]. In the low-range, the value of $1LSB_{ADC-LOW}$ can be calculated:

$$1LSB_{ADC-LOW} = \frac{400\mu A}{2^{16}} \tag{3.7}$$

$$= 6.103516nA \tag{3.8}$$

In the high-current range, 2500Ω resistance is used, maximum input voltage is 5V, therefore current can range from -2000 to $+2000\ \mu A$, a total range of $4000\mu A$.

$$\begin{aligned} 1LSB_{ADC-HIGH} &= \frac{4000\mu A}{2^{16}} && (3.9)\\ &= 61.03516nA && (3.10) \end{aligned}$$

Enough significant digits are used to ensure the resolution of the range is maintained, including rounding factors. With these, the lower and upper limits of measure for low-current and high-current modes are therefore:

$$\begin{aligned} RANGE_{low-lower} &= 6.103516nA * -32768 = -200000.0nA && (3.11)\\ RANGE_{low-upper} &= 6.103516nA * 32767. = 199993.9nA && (3.12)\\ RANGE_{high-lower} &= 61.03516nA * -32768. = -2000000.nA && (3.13)\\ RANGE_{high-upper} &= 61.03516nA * 32767. = 1999939.nA && (3.14)\\ & && (3.15) \end{aligned}$$

Range adjustment and overload checking

The motivation behind range adjustment is two-fold, 1. give the most highest resolution possible for a given measurement, 2. as a precaution against damage to the op-amps or ADC due to high current/voltage.

The two elements providing capability are are the measurement relays, which completely disable current flow and the range selection which selects the reference resistance for a given measurement.

After measurement relays have been switched on and a measurement is performed, the last stage of the measurement is to check the measured values against the limits and then to configure the range relays in case of

Automatic range adjustment and overload checking occurs each time a measurement is made, in two stages:

1. check the measured values against range limits and
2. adjust range / shut-off current

Range limits

In order to configure the automatic range hysteresis, the following parameters are defined:

LOW-SWUP	low-to-high current ADC transition value
LOW-MAX	maximum current value when in low-current mode
HIGH-SWDOWN	high-to-low ADC transition value
HIGH-MAX	maximum current value when in high current mode

Both LOW-MAX and HIGH-MAX determine the overload values, when the returned measured value meets are exceeds these values, the measurement relays are switched off.

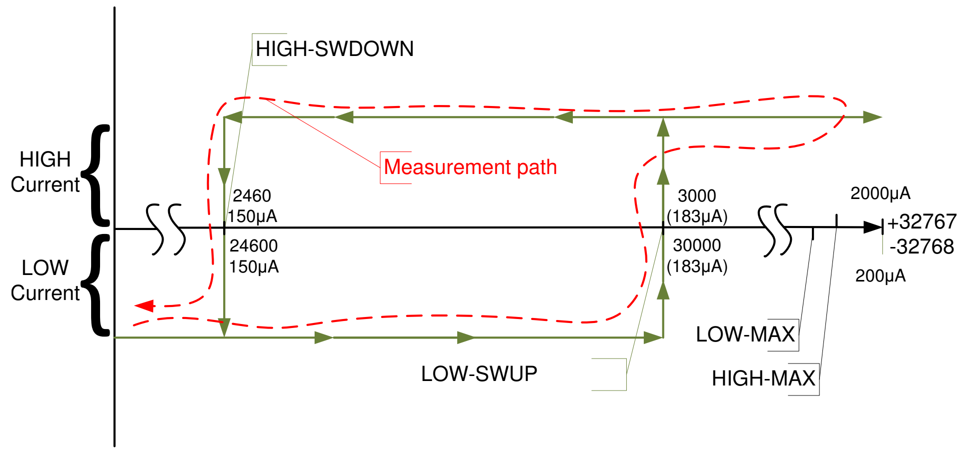


Fig. 3.24: Range limits

STREAM_WARN

Stream loss warning checks the STREAM_WARN bit and outputs the STREAM_WARN_CHAR (default: ASCII "!") to USB. This serves as a warning of misconfiguration that can be checked for in the first field of the CSV values by the higher-order system.

stream_reset()

This function simply clears the MCU_STREAM_BUSY bit, re-enables the port-pin interrupt, allowing MCU_STREAM_DVO to start streaming input again.

3.2.8.6 Calibration mode (CAL)

In order to compensate for offset at 0V DAC output within in the system, a technique to calibrate the zero-point offset has been developed. After calling the command to start calibration, the variable cal_cycles is set greater than 0, this allows the entire calibration process to run n number of times, in case longer average of calibration values is desired.

The calibration must also be able to handle future development of a dynamic calibration, which calibrates switches from measurement of the FET to a precision reference resistance.

3.2.8.7 Set zero point

Starting the calibration is depicted in figure 3.25.

Setting calibration mode sets the system mode to CAL, which then enables streaming input from the FPGA. This means, just as in streaming output mode, the streaming data from FPGA is used internally to calibrate the zero-offset. The calibration decision process is depicted in figure 3.26.

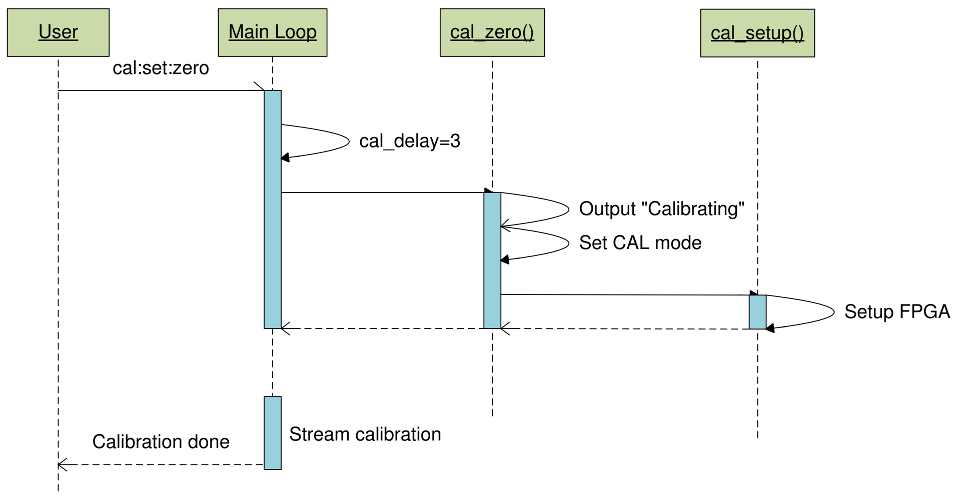


Fig. 3.25: Calibration setup function calls

cal_cycles counts the number of complete calibration cycles.

cal_delay is similar to output delay, discards readings to ensure the output has settled to a reliable value.

3.2.9 Relay control

Three sets of relays are important to the control of the current flow during measurement and calibration.

Modes of relay operation are:

OFF	No current flow
ON	Current flow in the normal direction
SWAP-ON	Current flow, swapped terminals

The controlled relays are:

GATE	Used to measure the gate current
VDS	Used to disable drain-source current

The calibration relays support ON and OFF modes:

CAL1	enable/disable the 10KΩ reference
CAL2	enable/disable the 100Ω reference

It is important that both relays are not "on" at the same time, and that one switches off and the next switches on with a delay to prevent damage. Sending an ON commands when already on, results in a brief (40 ms) off pulse because each relays are disabled before turned on for safety.

3.3 FPGA hardware architecture

3.3.1 System Overview

For the design of the FPGA, the following goals were to be met:

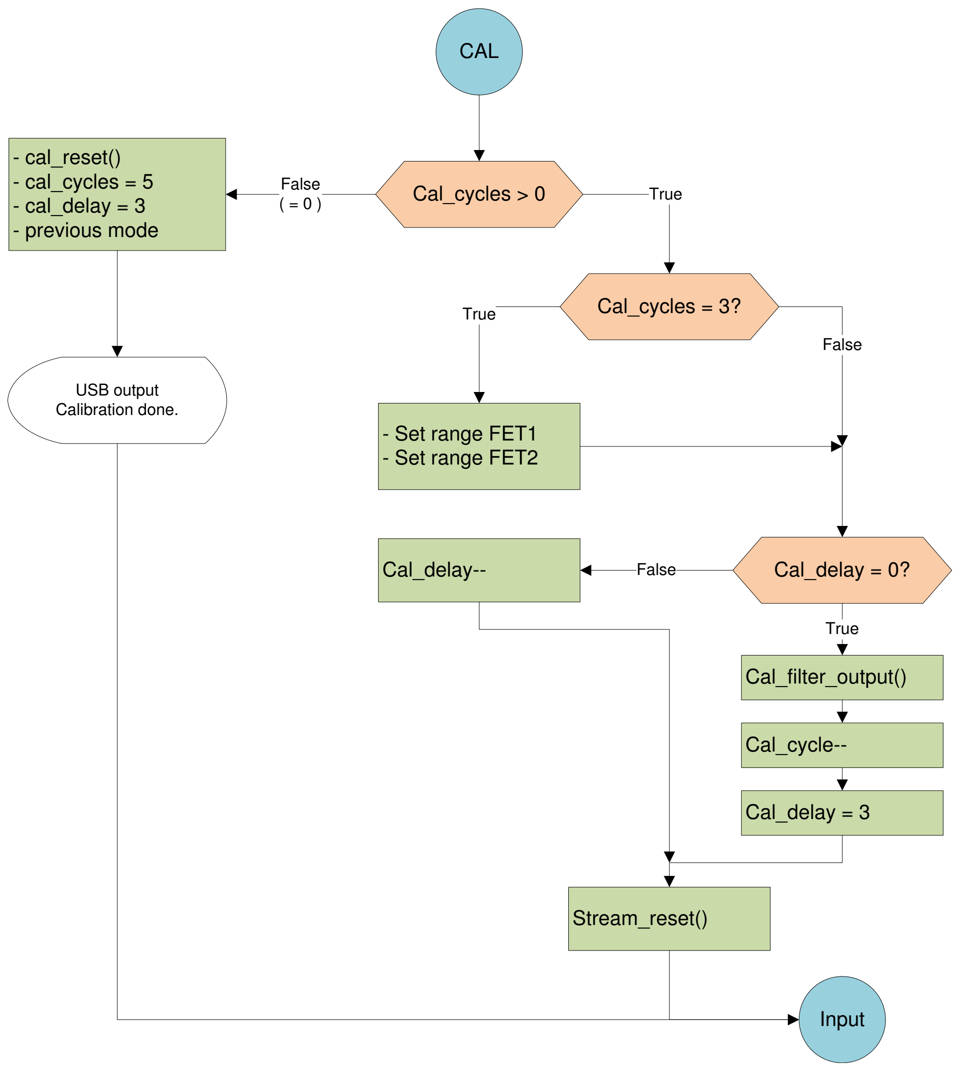


Fig. 3.26: Calibration decision

1. Create an period and duty-cycle adjustable switched biasing signal in the form of a square wave for each FET port.
2. Sample the voltage level of the current-measurement circuit for both FET ports according to the configuration of the sampling windows and state of the switch bias signal.
3. Efficiently filter and decimate ADC samples such that measurements can be sent over USB in real-time and such that noise efficiently reduced
4. Provide filtered/decimated measurement values independently for both switch biasing states and temperature samples for each port.
5. Configuration of the current measurement range relays.
6. Ability to reset and configure the ADC and FPGA configuration values.

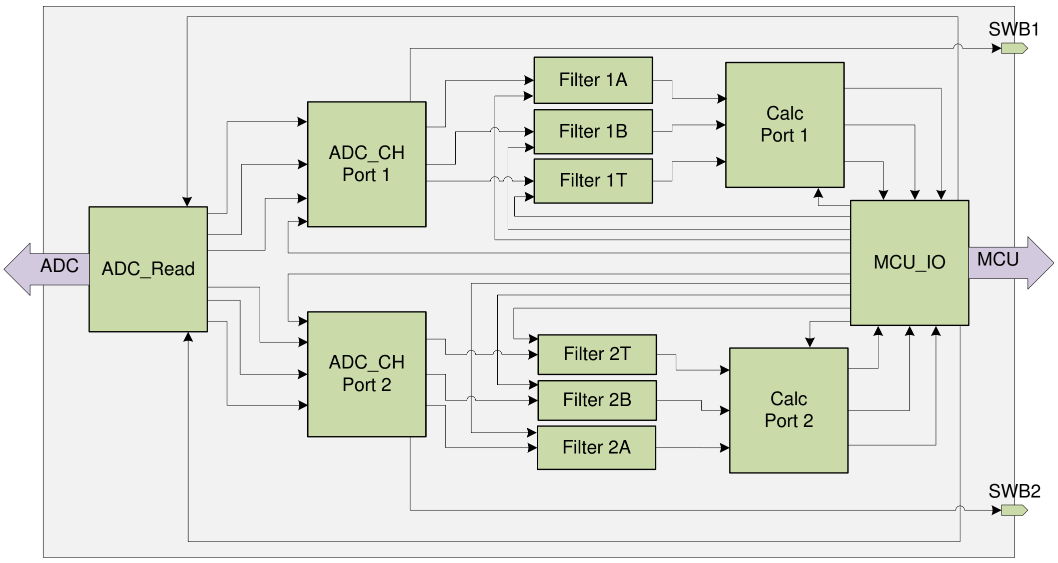


Fig. 3.27: FPGA implementation overview

In order to meet these requirements, the overall role of the FPGA was subdivided into components, each of which is part of the over design, instantiated one or multiple times as entities, each of which provides its own part of the overall functionality of the system. These components summarized in table 3.4.

Tab. 3.4: FPGA component summary

Component name	Description
MCU_IO	Memory, MCU Interface
CIC	CIC decimation filter
ADC_READ	ADC configuration and sampling
RESET_SYNC	Synchronization of incoming asynchronous reset signal
ORgate_2	Two-input OR gate
ADC_CH	Route incoming ADC data to the appropriate filter, create switch-biasing signal
CALC_CH	Calculations, adjustments to outgoing filter data, triggering controls
LED	Debugging LED control

Components are organized in a shallow hierarchy, each belonging to the root component *FET_ROOT*, which is the name assigned to the root of the component which routes and contains all components of the system. Several of the components are initialized multiple times, each playing a different role with different routes in FET_ROOT. These entities are summarized in table 3.5.

Tab. 3.5: FPGA component-entity summary

Entity	Component	Description
ADC_CH1	ADC_CH	FET Port 1 ADC_CH functionality
ADC_CH2	ADC_CH	FET Port 2 ADC_CH functionality
Filter 1A	CIC	CIC filter for FET port 1, sample window A values
Filter 1B	CIC	CIC filter for FET port 1, sample window B values
Filter 1T	CIC	CIC filter for FET port 1, temperature values
Filter 2A	CIC	CIC filter for FET port 2, sample window A values
Filter 2B	CIC	CIC filter for FET port 2, sample window B values
Filter 2T	CIC	CIC filter for FET port 2, temperature values
Calc Port 1	CALC_CH	FET Port 1 CALC_CH functionality
Calc Port 2	CALC_CH	FET Port 2 CALC_CH functionality

The root component FET_ROOT also routes all external input/output signals which connect to the external components on the system board. External components are the ADC, switched biasing control logic, range control relays, MCU and debugging LEDs and pins. These list of these external I/O signals are shown in table A.5.

When internal process logic is used to create output signals to other system components, they are first *registered*. Internal versions of these signals that are *registered* are assigned with the **_i** appendix to its name.

In the following sections, the implementation of the FET_ROOT components will be described in detail. All components are written using the standard IEEE libraries, with the header

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

For brevity, all signal vector functions relying on these libraries, such as those dealing with addition and integer-signal vector conversion (e.g. std_logic_vector(), signed(), unsigned()) are omitted from the component descriptions. For example, this signal assignment from the CIC component appears in the VHDL source code as:

```
count_out_i <= (others => ’0’);
c1  <= std_logic_vector(signed(c0) - signed(i2d1));
```

would be displayed in block diagrams and descriptions of the component as:

```
count_out_i <= 0
c1  <= c0 - i2d1
```

Each component is designed to be synchronous with the rising-edge of the FPGA system clock (separate clock as MCU). This is recommended design, as there is a period of instability due to propagation delays through the FPGA [Altera 2007]. In the routing specification in the Quartus II TimeQuest Timing Analyzer software, a timing requirement to match the FPGA system clock (20 MHz) was added to ensure that propagation delay between modules and of signals does not exceed the period of the system clock, i.e. the *setup and hold time*. The clock itself is assigned to a the global clock of the Altera FPGA, which also helps ensure timing requirements are met[Altera 2007].

Output signals that are intended for external pin assignment are registered in the module in which they are used to help ensure that the setup and hold times are not violated. Those incoming signals, such as reset and MCU handshaking signals are made synchronous to the FPGA system clock through the use of synchronizers.

3.3.2 Component MCU_IO

The MCU_IO module is primarily responsible for communication between the MCU and the other modules in FET_ROOT by setting up internal signals, providing handshaking between the MCU and FPGA, and memory allocation and assignment for various configuration and measurement values.

The module includes many internal I/O signals (table), as it heavily interconnected, however is functionality can be broken down into just three processes, summarized in table 3.6.

Tab. 3.6: Module MCU_IO process summary

Process	Purpose
COMM_PROC	MCU handshaking, Memory assignments
STREAM_PROC	set mcu_stream_dvo (interrupts MCU, output to MCU) based on configured channel trigger (in CALC_CH) and CH1/CH2 two-bit selector
STREAM_WARN_SET	Set a warning bit to be check by the MCU if data is internally triggered while the MCU is still processing the previous triggered request. Maintains a count "lost" measurements

There are many internal signals, which serve to interconnect the processes and to create registers (storage), these signals are summarized in table A.6

The signal RAM16 serves as a generic RAM storage for measurement values, sampling window configuration values, data output rate, trigger information, etc. is implemented. The configuration values are set by the COMM_PROC process to fixed locations and assigned to entities through entity-external signal assignments. The 64 16-bit memory locations are addressed and accessed internal in by their array location, RAM16(0) to RAM16(63). For a list of memory location assignments, refer to the appendix.

3.3.2.1 MCU_IO process COMM_PROC

The communications process (figure 3.28) is responsible for resetting memory, assigning default values, controlling accesses to the data bus, and MCU handshaking logic.

3.3.2.2 MCU_IO process STREAM_PROC

This process (figure 3.29) sets mcu_stream_dvo (interrupts MCU when in a streaming mode) based on configured channel trigger (set by CALC_CH) and the CH1/CH2 two-bit channel selector (set by the user, stored at RAM16(27)).

The DVO (data-valid output) signal to the MCU is held for at least two clock cycles, in order to ensure that signals are stable while they are being retrieved by the MCU, this is done by chaining the signal assignment at each rising clock edge and ORing the signal assignment with the registered output, as shown in the following code snippet:

```
elsif rising_edge(clk) then
mcu_stream_dvo_d1 <= ch1_en or ch2_en;
mcu_stream_dvo_d2 <= mcu_stream_dvo_d1;
mcu_stream_dvo_i <= mcu_stream_dvo_d2 or mcu_stream_dvo_d1 or ch1_en or ch2_en;
```

where the signals *mcu stream dvo d1*, *mcu stream dvo d2* are delay for delay, and *mcu stream dvo i* is the internal signal for a DVO that is set for the the MCU.

3.3.2.3 MCU_IO process STREAM_WARN

This process checks for a rising edge of DVO from the filter trigger configuring during stream_processing (MCU busy). If this is the case, the MCU is still busy retrieving measurement values over the FPGA-MCU interface, and therefore a measurement has been *lost*. In order to warn the user, a flag is raised that is checked by the MCU at the conclusion of each measurement retrieval cycle, additionally a counter of lost measurements is maintained and stored in memory (RAM16(51)), which can be retrieved by the MCU.

3.3.3 Component ADC_READ

This component starts sampling in response to an enable signal, the ADC, sending the appropriate signals with the correct timing to the ADC and outputting the result on internal signals for the ADC_CH component. The module is divided into two processes:

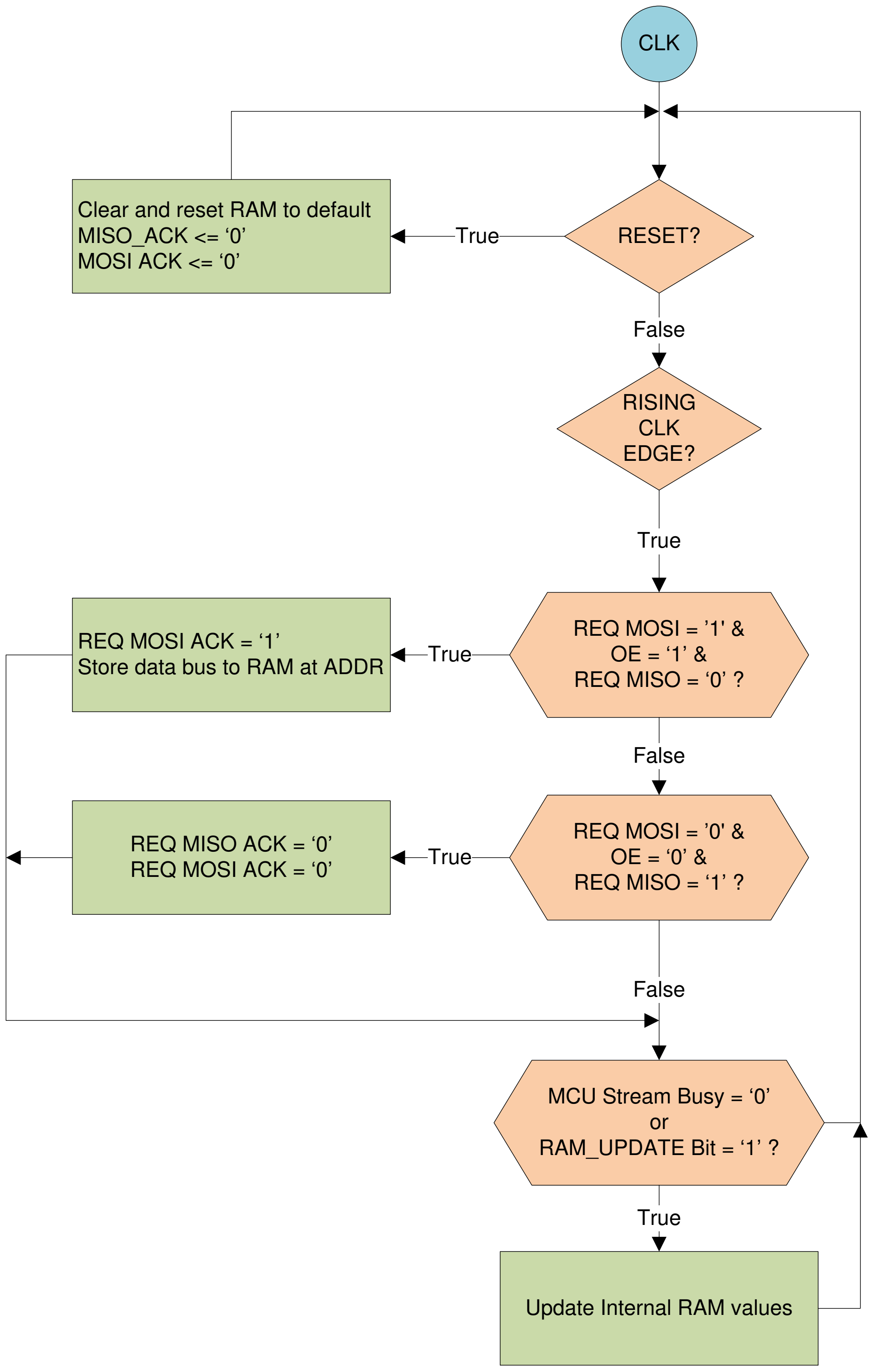

CLK
RESET?
True
Clear and reset RAM to default
MISO_ACK <= '0'
MOSI ACK <= '0'
False
RISING CLK EDGE?
True
REQ MOSI = '1' & OE = '1' & REQ MISO = '0' ?
True
REQ MOSI ACK = '1'
Store data bus to RAM at ADDR
False
REQ MOSI = '0' & OE = '0' & REQ MISO = '1' ?
True
REQ MISO ACK = '0'
REQ MOSI ACK = '0'
False
MCU Stream Busy = '0' or RAM_UPDATE Bit = '1' ?
True
Update Internal RAM values

Fig. 3.28: COMM_PROC

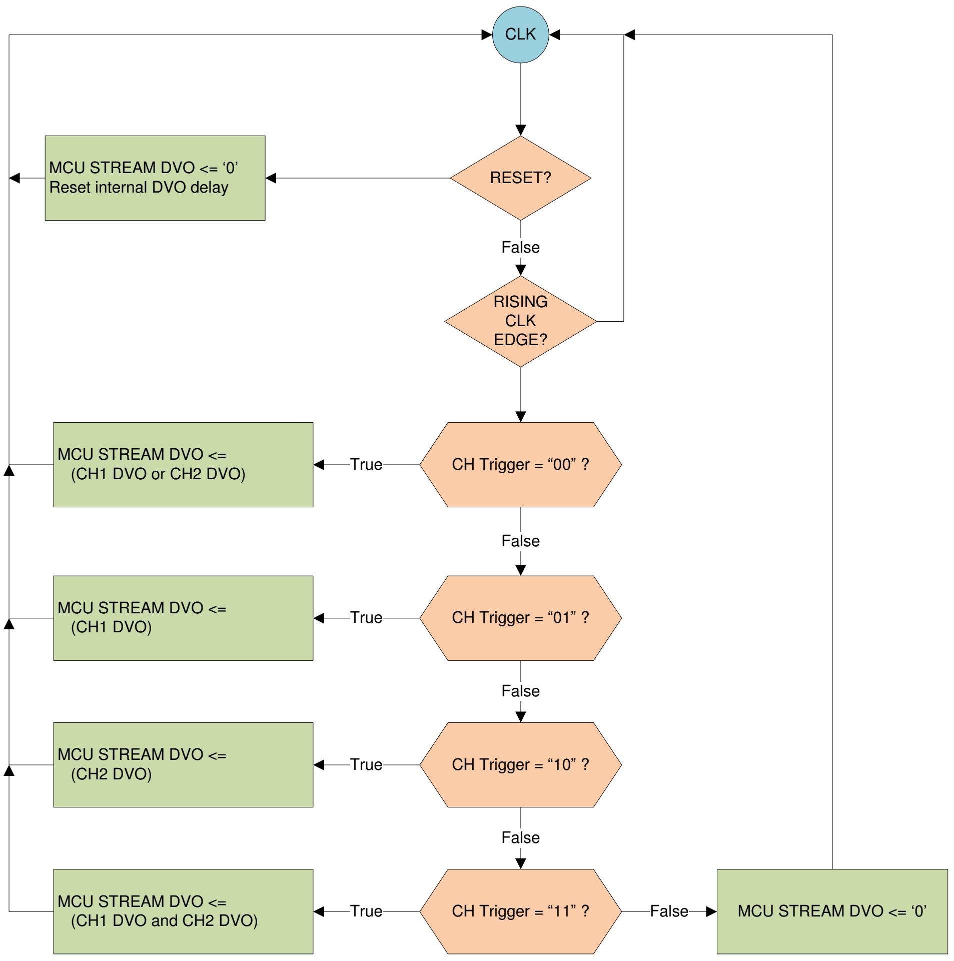


Fig. 3.29: STREAM_PROC

Tab. 3.7: ADC_READ processes

Process	Purpose
ADC_COUNT_PROC	The count process creates the logic of when to sample, based on the input signal EN and an internal counter.
DATA_COUNT	counter for control of the filters
ADC_LOOP_PROC	sets the signals according to the timing diagram of the ADC and the state of the enable (en) signal.

3.3.3.1 Process ADC_COUNT_PROC

The EN signal is not used completely traditionally, in a way an enable signal would also disable the component when cleared. Instead, it behaves more like a start

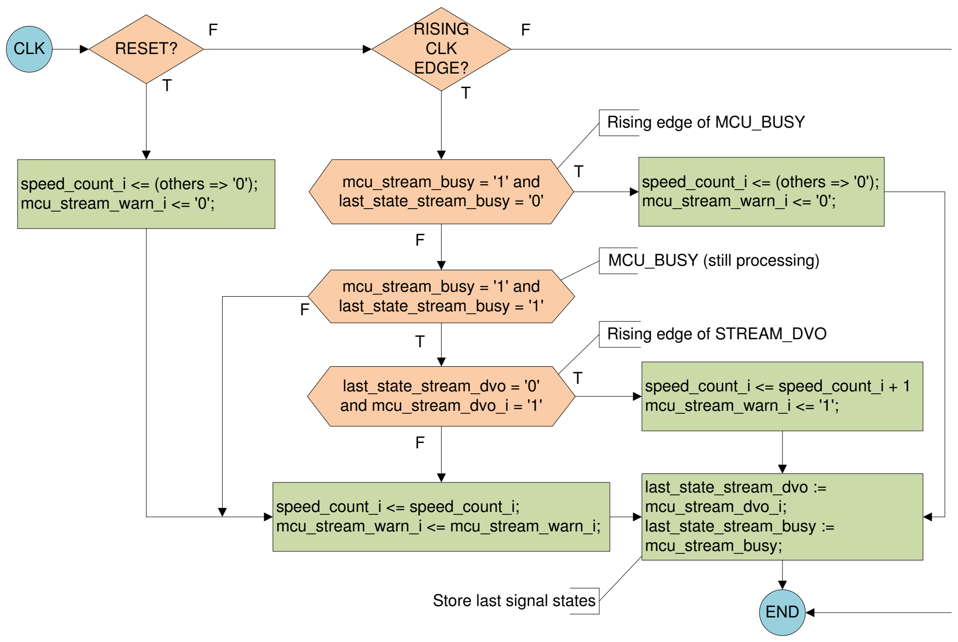

CLK
RESET?
F
T
RISING CLK EDGE?
F
T
speed_count_i <= (others => '0');
mcu_stream_warn_i <= '0';
Rising edge of MCU_BUSY
mcu_stream_busy = '1' and last_state_stream_busy = '0'
T
speed_count_i <= (others => '0');
mcu_stream_warn_i <= '0';
F
MCU_BUSY (still processing)
mcu_stream_busy = '1' and last_state_stream_busy = '1'
F
T
Rising edge of STREAM_DVO
last_state_stream_dvo = '0' and mcu_stream_dvo_i = '1'
T
speed_count_i <= speed_count_i + 1
mcu_stream_warn_i <= '1';
F
speed_count_i <= speed_count_i;
mcu_stream_warn_i <= mcu_stream_warn_i;
last_state_stream_dvo := mcu_stream_dvo_i;
last_state_stream_busy := mcu_stream_busy;
Store last signal states
END

Fig. 3.30: STREAM_WARN

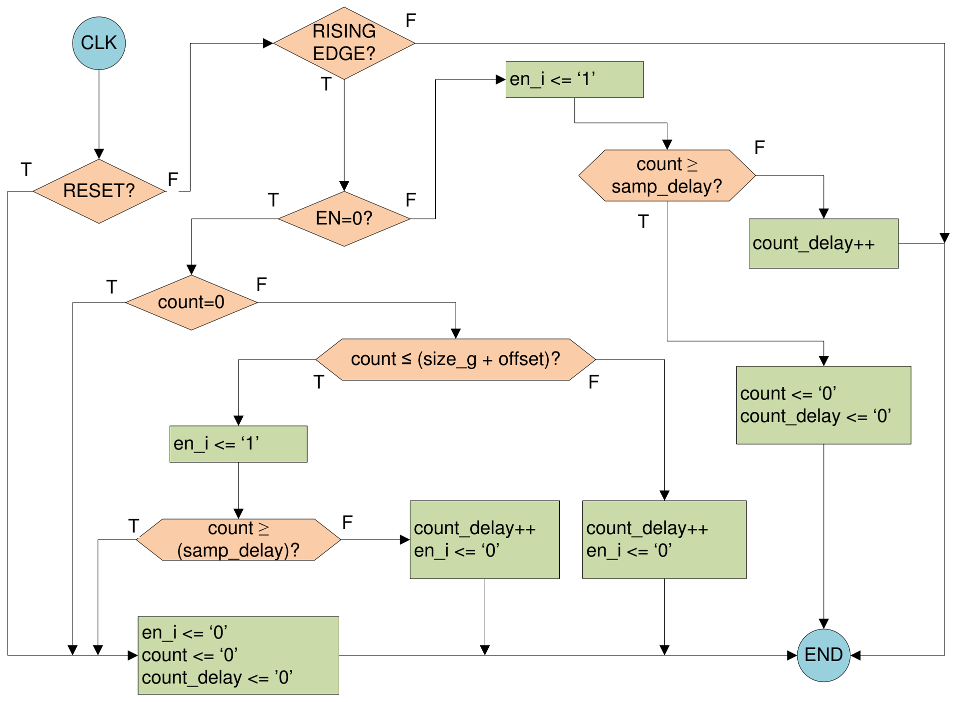


Fig. 3.31: ADC_READ:ADC_COUNT_PROC

conversion. Once the signal EN goes high, the internal signal en_i is used to *lock-in* the enable signal until the conversion cycle (count to size_g) is complete is used to lock-in the enable signal.

signal **count_delay** is used for sampling delay, while signal **count** is used by the ADC_LOOP_PROC for the timing of signals to the ADC.

3.3.3.2 Process ADC_LOOP_PROC

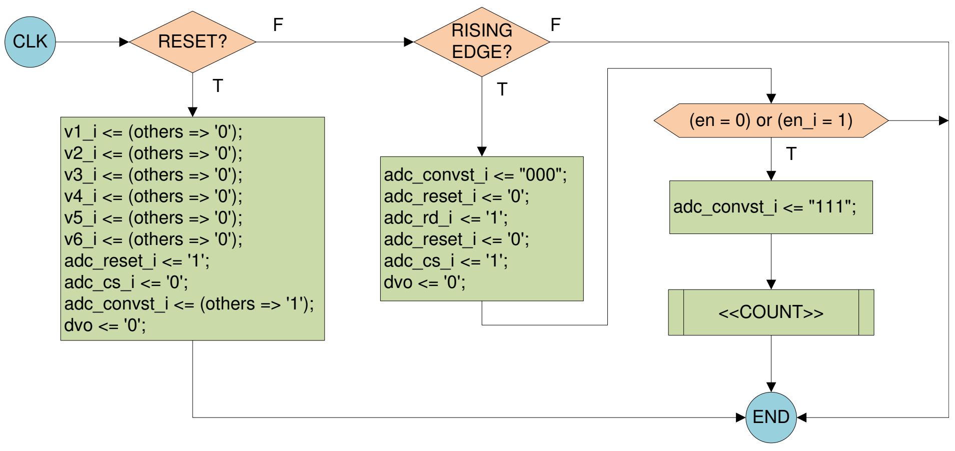
Several implementations were tested for the design of this process, including a state machine. The simplest solution was found to employ a counter to time count clock cycles and set output at predefined counts. The timing of ADC control signals is achieved by counting FPGA system clock cycles. By observing the timing requirements of the ADC, the ADC control signals can be set accordingly and samples stored, as summarized in table A.9. With a 20 MHz clock, the sampling cycle time (t_{sc}) can be calculated:

$$t_{sc} = (20MHz)^{-1} * 81_{CLK} = 4.05\mu s f_{sc} \qquad = tsc^{-1} = 246913.6 \tag{3.16}$$

This is slightly less than the maximum rate ADC sampling rate of 250 KSPS, however exceeds our minimum specification requirements.

By using a counter instead of state machine, the timing of the process remains deterministic. The entire process is depicted in figure 3.32.

Similarly, after the sampling cycle, a sampling delay counter (counting CLK cycles) adds delay before the next sampling cycle can begin.



```
v1_i <= (others => '0');
v2_i <= (others => '0');
v3_i <= (others => '0');
v4_i <= (others => '0');
v5_i <= (others => '0');
v6_i <= (others => '0');
adc_reset_i <= '1';
adc_cs_i <= '0';
adc_convst_i <= (others => '1');
dvo <= '0';
```

```
adc_convst_i <= "000";
adc_reset_i <= '0';
adc_rd_i <= '1';
adc_reset_i <= '0';
adc_cs_i <= '1';
dvo <= '0';
```

```
adc_convst_i <= "111";
```

Fig. 3.32: ADC_READ:ADC_LOOP_PROC

This process takes advantage of the fact that the last signal assignment in the process is the signal assignment that is output (first) at the end of the process. The signal assignments in the block "COUNT LOGIC ASSIGNMENT" are shown in table A.9.

In the timing design, several points has to be observed concerning how to handle the inter-process and inter-component signal delay:

1. **Start-up**: How to configure the ADC during power-up.

2. **Switched-biasing status** How to determine the status of the switched biasing signal during conversion start
3. **Sampling-delay** How to incorporate the sampling delay without affecting the timing
4. **CONVST state** State and timing of the conversion start signal at the end of the count, before the next sampling cycle

Start-up. Several ADC pins during during power-on are important to configure the ADC. The following code sets this pins statically in the design, and can not be changed:

```
adc_range  <= '1';
adc_wb  <= '0';
adc_ser_par_sel  <= '0';
adc_hs  <= '0';
adc_wr_refen_dis  <= '1';
```

where adc_range sets the range, wb sets a word/byte mode, ser par sel sets serial or parallel mode, hs sets hardware or software mode and wr ref en dis enables/disabled the reference. It would found by attempting to configure these pins such that the user can change the values, the start-up mode during system power-up played an important role, and resulted in unpredictable performance of the ADC-FPGA interface.
Switched-biasing status In order to determine the true conversion start of the ADC (between the enable start signal from ADC CH and the true CONVST signal to the ADC, the output signal adc_conv_intern was added with the logic:

```
adc_conv_intern <= adc_convst_i(0) or adc_convst_i(1) or adc_convst_i(2);
```

where adc conv intern the signal setting the conversion start process on the ADC, for each of the three ADC groups A, B and C. A starts conversion for channels 1 and 2, B starts conversion for 3 and 4, and C starts conversion for 5 and 6. Setting all bits simultaneously starts a parallel conversion, but setting requires a complete conversion cycle to pass before the next group can be set [Devices].

CONVST state While other signals are assigned default values before the COUNT LOGIC block, the CONVST has two default states before the COUNT LOGIC block depending on the status of the en and en_i signal. This allows conversion to start as soon as an enable signal is started and keeps it high during the sampling cycle. When the sampling cycle is complete, the CONVST returns to 0.

3.3.4 Component ADC_CH

This module has two main functions: sending the switched biasing signal for each port, according to the configured period and duty cycle; and directing the returned ADC sampling information from all 6 channels to the correct filtered, based on the state of the switched biasing signal. In order to achieve these tasks, the module is divided into 6 processes:

Tab. 3.8: ADC_CH Processes

Process	Purpose
SW_COUNT	counter for the switched biasing signal
DATA_COUNT	counter for control of filter decimation rate (USB data output rate)
SW_PROC	Create the switched biasing signal
DEC_PROC	Control signals to the filters
ADC_CONTROL	control signals for ADC READ module
LAST_STATE	Retain state of switch biasing signal for return ADC value filter routing
FILTER_CONTROL	Control signals for the A,B and T filters based on the sampling states

Tab. 3.9: ADC_CH external signals

Signal	Direction	Purpose
clk	in	FPGA system clock (20 MHz)
reset	in	Synchronous reset
va	in	Input ADC value (voltage value representing current)
val_t	in	Input ADC value (voltage value representing temperature)
adc_busy	in	ADC_BUSY signal from ADC (high during conversion)

3.3.4.1 Internal signals

The size of counter sw_cnt dictates the period of the switched biasing signal:

Tab. 3.10: Switched biasing counter output frequencies

n_{CLK}	Period	Frequency
1	50ns	20 MHz
80	4000 ns	250 kHz
20000	1ms	1 kHz
$20 * 10^6$	1s	1 Hz
2^{32}	214.75s	$4.6 * 10^{-3}$

The number of bits required for any desired output switched bias control frequency $f_{desired}$ can be calculated:

$$n_{bits} = \left\lceil log_2 \left(\frac{f_{CLK}}{f_{desired}} \right) \right\rceil \qquad (3.17)$$

The same is true for data_cnt, however the filter COUNT value limits how many actual samples can be integrated. (See section 3.3.5)

The 32-bit width was chosen due to memory size on the MCU, and because it easily accommodates the minimum requirements of the system.

3.3.4.2 Process SW_COUNT

The SW COUNT is a count down timer. The value from which it counts down is set only if period is greater than zero. In this configuration, the switch biasing signal remains unset until it is configured from the user. The duty cycle is handled when the signal is actually created in the SW_PROC process. The counting logic is depicted in figure 3.33.

In order to configure switch biasing, the configuration value of sw_width = sw_period are set equal, the difference between then is 0 and the value of cnt (due to inter-process delay) remains above zero, always enabling the switch biasing signal.

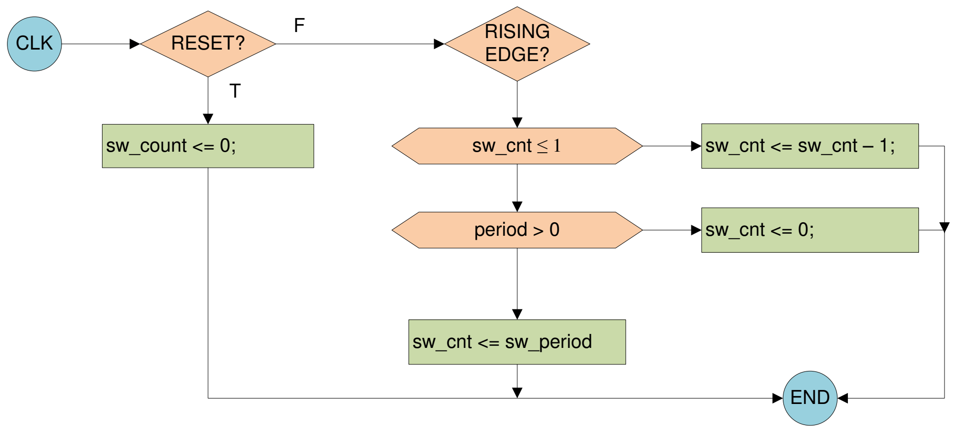


Fig. 3.33: ADC_CH:SW_COUNT process

3.3.4.3 Process DATA_COUNT

This process is logically identical to the SW COUNT process, creating a separate periodic signal used for control of the filters by DEC_PROC. A separate counter is used because the data output rate normally differs from the switched biasing control signal frequency.

It is configured by the value of the data_period input signal, which is set by the user (stored in MCU_IO memory) to control the rate at which decimation, i.e. measurement output rate.

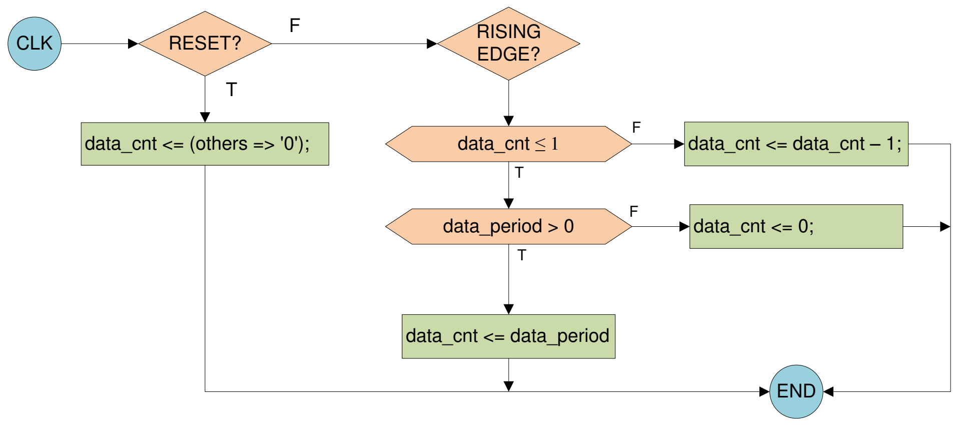


Fig. 3.34: ADC_CH:DATA_COUNT process

3.3.4.4 Process SW_PROC

The switch biasing signal's period is dictated by the value of the period and width signals, which are set by the user.

The duty cycle is set by the value of width, which is use as shown in the figure.

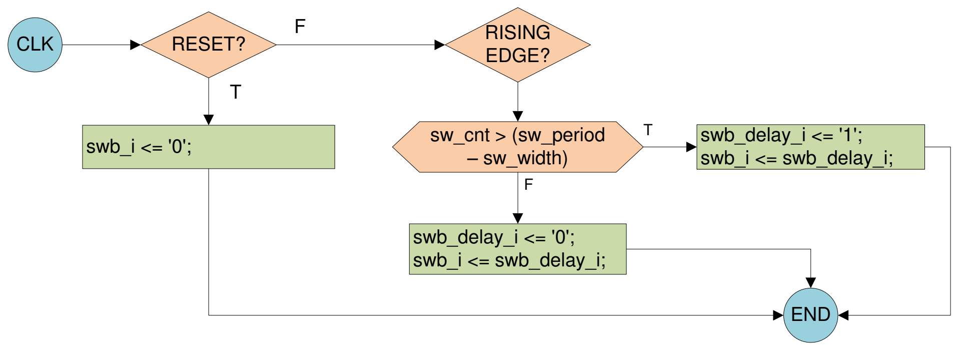


Fig. 3.35: ADC_CH:SW_PROC process

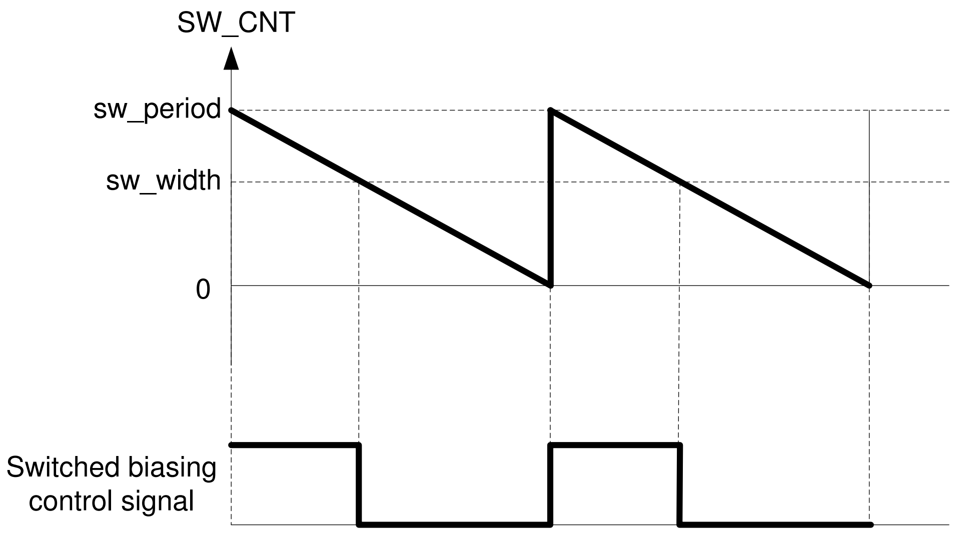


Fig. 3.36: ADC_CH:SW_PROC variable duty cycle

3.3.4.5 Process DEC_PROC

The CIC filter decimation rate, and hence the data output rate to USB is determined by this process, which sends a signal to the CIC filters when a decimation should occur. This in combination with a separate counter (number of incoming signals, see CIC filter process) allows for a variable decimation rate.

The CIC filter control signal dec is set when the counter reaches 1 and not at 0 so that it is only set when the counter is running, and not after a reset or before it has been configured.

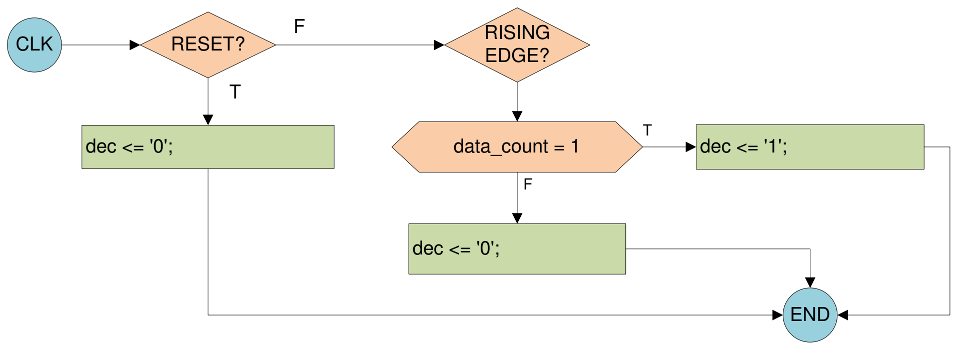


Fig. 3.37: ADC_CH:DEC_PROC decimation control process

3.3.4.6 Process ADC_CONTROL

In this process, the ADC is controlled as determined by the PHASE A, PHASE B, PHASE C, PHASE D (delay 1, delay 2, delay 3, delay 4) configuration signals, i.e. the timing relationship between the period, width and phase configuration.

Pseudo code The internal signal adc_en_i is set as follows:

```
if sw_cnt > (sw_period - delay1), then adc_en_i <= '0'
elsif sw_cnt > (sw_period - delay1), then adc_en_i <= '0'
elsif sw_cnt > (sw_period - delay1), then adc_en_i <= '0'
elsif sw_cnt > (sw_period - delay1), then adc_en_i <= '0'
else adc_en_i <= '0'
end if;
```

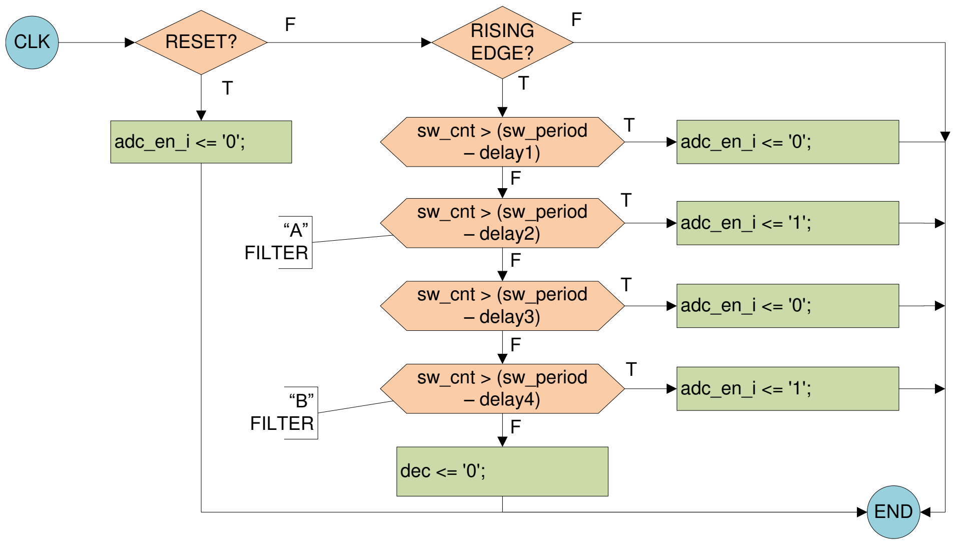


Fig. 3.38: ADC_CH:ADC_CONTROL ADC control process

3.3.4.7 Process LAST_STATE

The last state process is used to create a signal that retains the state of the switching biasing signal when the ADC conversion start signal CONVST was sent to the ADC. Since it is desirable to sample as close as possible to the edge of the switched biasing signal (due to settling time of the voltage-current measurement circuitry) an ADC sample may occur when the switched biasing control signal is high, however return the sampled value when it is low. Therefore when the DVO signal is received, the state of the switching biasing signal is needed in order to route it to the correct filter.

An important consideration is that this process does not take into account the actual delay between the setting of the switched biasing control signal and the actual current-represented voltage that results from sampling. This is due to settling time of the current-measurement circuitry in reaction to the switched biasing control signal.

An alternative to this process would be to implement a new user-configurable signal by which the user could set the number clock cycles to shift the last_state signal. Otherwise, this delay must be observed and accounted for when setting the sampling windows.

The LAST STATE itself process (figure 3.39) employs the use of a variable *edge_state* to store the *last state* of the ADC BUSY signal from the ADC. The ADC BUSY signal is used here instead of the CONVST signal since the ADC can not make an instantaneous conversion. There is a slight delay for the sample-and-hold circuitry, the ADC BUSY signal is set during conversion therefore we ensure the sample-and-hold circuitry is holding the sampled value.

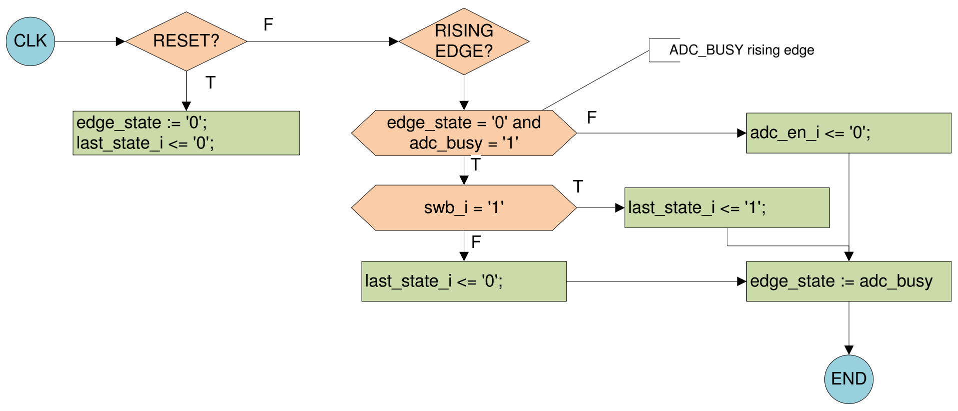


Fig. 3.39: ADC_CH:LAST_STATE process

3.3.4.8 Process FILTER_CONTROL

After an ADC conversion has completed, a data-valid input signal is generated (adc_dvi). In order to determine which filters to enable (A, B or T), an external signal connected to the CIC filter module is used to *enable* the filter. Additional registers are used to hold that value until the next incoming value for that filter. val_h_i (A-filter) and val_l_i (B-filter) hold the values.

val_h_en enables the A-filter, val_l_en enables the B-filter, and for each condition the val_t_en is enabled, since the value of the temperature is always wanted.

R	the rate change
M	the differential delay
N	the number of comb sections / integrator stages

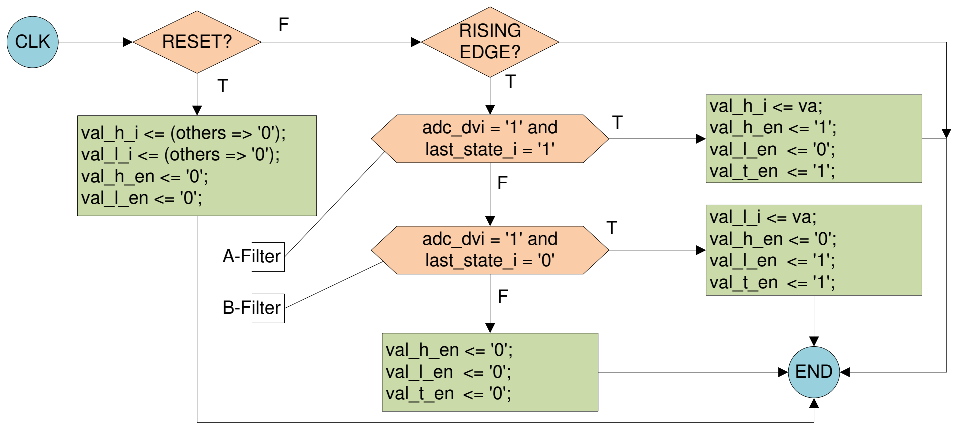


Fig. 3.40: ADC_CH:FILTER_CONTROL process

3.3.5 Component CIC

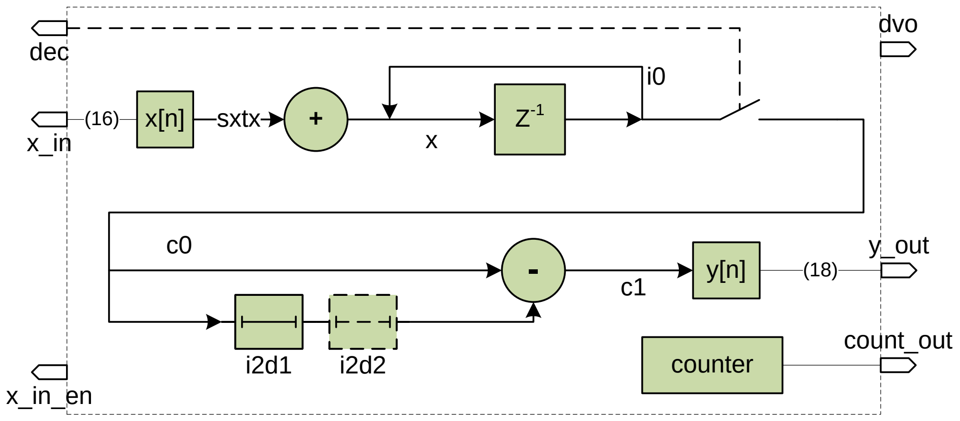


Fig. 3.41: CIC filter module overview

The filter chosen for this design is a cascade integrator comb, CIC filter, type of moving average filter. [Jahromi 2007]

The purpose of the CIC component is to create a filter that can be enabled and disabled, according to the current sampling window configuration, but also adjust to sampling rate changes. This is necessary because the sampling window of the switched biasing signal is user configurable, and since the input into the filter alternates between the A and B filters at a variable rate, the downsampling rate itself is also variable.

Together, these give us the transfer function of the filter:

$$H(z) = H_I(z) * H_C(z) \tag{3.18}$$

$$= \left(\frac{1 - z^{-RD}}{1 - z^{-1}}\right)^N \tag{3.19}$$

Additionally factors that determine the shape of the CIC filter, the number of incoming bits (16-bits in our case, from the ADC).

The range of decimation possible must be fixed in order to determine the bit width of the internal signals, it is necessary to determine by how much the summing in the integrator stage before decimation can grow.

This is determined by considering the maximum sampling rate of the ADC possible and measurement value output rate over USB, which has been specified as 10 measurements per second. Considering a configuration in which there is no switch-biasing (constant bias) the maximum number of ADC samples per second is 250000. With an measurement rate of 10, the maximum decimation rate R_{max} is:

$$R_{max} = \frac{\text{ADC SPS (max)}}{\text{USB output rate (max)}} = \frac{250000}{10} = 25000 \tag{3.20}$$

The gain of the stages is

$$G = (RM)^N \tag{3.21}$$

From this value, we can determine the maximum signal size that results in the integration stage, we can calculate the maximum bit growth, B_{growth}:

$$B_{growth} = \lceil N * log_2(RM) \rceil \tag{3.22}$$

The maximum bit size is then:

$$B_{MAX} = B_{in} + B_{growth} \tag{3.23}$$

Choosing a delay M = 1, and because we are mostly concerned with the values around DC and are interested in removing higher frequency components, a one-stage CIC filter was chosen, which gives us a B_{MAX} value:

$$B_{MAX} = B_{in} + \lceil N * log_2(RM) \rceil \tag{3.24}$$

$$= 16 + \lceil 1 * log_2(25000 * 1) \rceil \tag{3.25}$$

$$= 16 + \lceil 1 * 14.6096 \rceil \tag{3.26}$$

$$= 16 + 15 = 31 \tag{3.27}$$

The maximum rate change is the next highest power of two, 32768. Because our memory is 16-bit, we easily increase the bit width to 32, meaning our maximum rate change is 65536.

The second parameter of interest size of the output. That is, the *minimum* bit size, to shift (divide) by in order to obtain a bit width at the output that accommodates the *smallest* rate change. This value, s, is determined by:

$$s = \frac{2\left\lceil Nlog_2RM \right\rceil}{(RM)^N} \tag{3.28}$$

However, calculating and shifting by this value is is complicated when the decimation rate R is not a power of two, resulting in variable gain that must be adjusted for at the MCU before the measurement is output. Because the decimation rate can vary slightly depending on the configuration of the sampling windows, the timing of both FET ports, exactly how many samples s are made are difficult to determine. Instead, the full bit width (32 bits) of the integration, along with the number the number summed values, count is transferred along with the integrated value to the MCU. There, the actual measurement value is calculated.

Because multiple filters are in use, each filter has a *enable*, which allows ADC_CH to select the appropriate filter. The *dec* signal is set by the ADC_CH module, which determines the output rate filter (i.e. USB measurement output rate). The dec signal results in a DVO signal, which signals that the y_out value and the count value are valid.

3.3.5.1 Implementation

The implementation of this design uses a simple state machine consisting of two states, SAMPLE and HOLD. When in HOLD, the integrator section is active, summing ADC values. In SAMPLE, the comb section overtakes the last value of the integrator. This is accomplished using the following processes:

Process	Purpose
dec_proc	counts the number of values integrated, switches the filter to sample when decimate signal is received
cic_count_proc	Counts the number of samples, sets the state (SAMPLE and HOLD)
cic_sxt_proc	Takes input signal, adjust to configured B_{MAX} word length, retains upper bits (sign bit)
cic_int_proc	Integration section
cic_comb_proc	Comb section

3.3.5.2 Process dec_proc

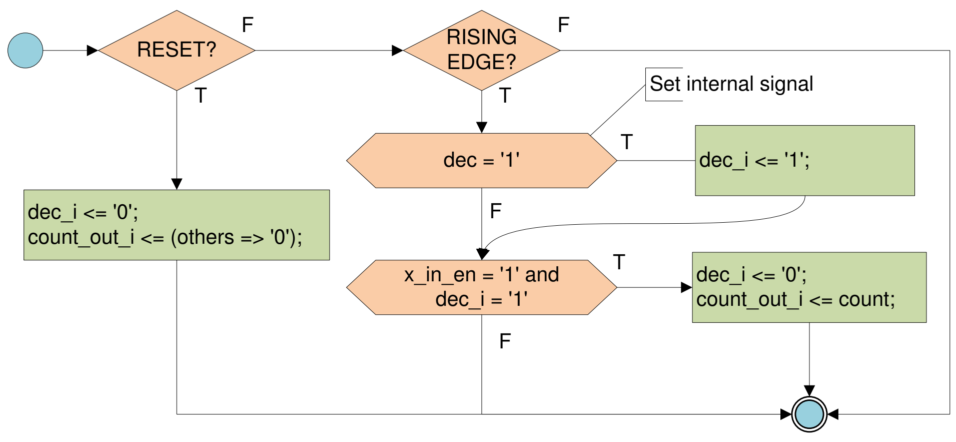


Fig. 3.42: CIC process DEC_PROC

The filter control signals from ADC_CH select which of the 6 filters is activated at any given ADC output. In case the decimate and enable signal do not occur at the same time, the dec (decimate) signal updates an internal signal (dec_i) which is used by the cic_count_proc to change filter state to SAMPLE and reset the count the next time an enable signal is received.

3.3.5.3 Process cic_count_proc

The process works by counting each incoming dec signal while it is

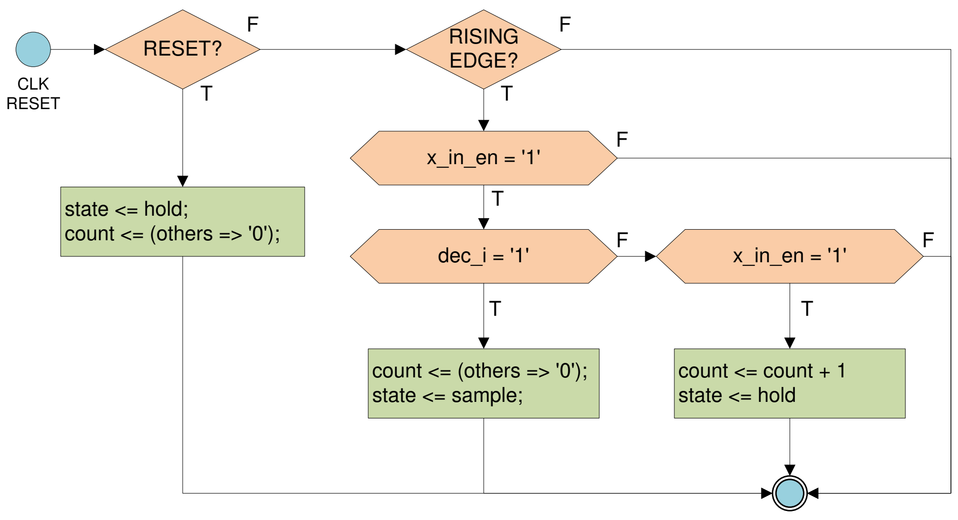


Fig. 3.43: CIC process DEC_PROC

3.3.5.4 Process cic_sxt_proc

This process sets assigns the incoming value x_in to the internal signal sxtx. Since sxtx is the set to the internal word size (32 bits), all upper bits are assigned the sign bit to maintain the two's complement signing of the value. In case of a negative value, the additional 1s are adjusted for in post-processing on the MCU with the value of the signal *count*.

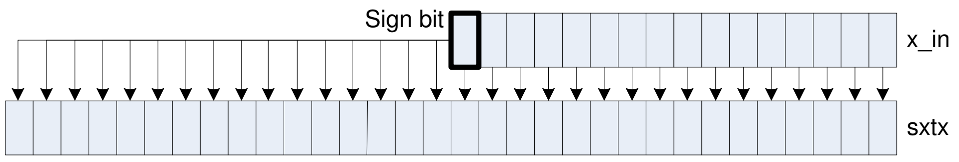


Fig. 3.44: CIC process CIC_SXT_PROC

This is done as shown in the code below, using the VHDL attribute 'high to read the sign bit, and assignment in a for loop.

```
sxtx((input_size-1) downto 0) <= std_logic_vector(x);
   for k in (word_size-1) downto (input_size) loop
      sxtx(k) <= x(x'high);
   end loop;
```

where sxtx is the internal version of the incoming signal.

The signal sxtx is a large bit width so that it can be assigned and summed with delayed versions of itself in process CIC_INT_PROC more easily.

3.3.5.5 Process cic_int_proc

This is the integrator section of the CIC filter, which operates at the f_s the sampling speed of the incoming values, but is only active when the filter is enabled (as configured by the sampling window).

In our one-stage version, we assign the values input *x in* to *x*:

```
x  <= x_in;
i0 <= std_logic_vector(signed(i0) + signed(sxtx));
```

In a two- or three-stage version, we could simply add additional signals here, as follows:

```
x  <= x_in;
i0 <= std_logic_vector(signed(i0) + signed(sxtx));
i1 <= std_logic_vector(signed(i1) + signed(i0));
i2 <= std_logic_vector(signed(i2) + signed(i1));
```

where i0, i1, and i2 connect each of the three internal stages.

On each process activation, the value of the signal propagates through. The signed() function is used to maintain the sign during addition.

In our one-stage version, the signal i0 is assigned to the comb section signal c0 when sampling (see figure 3.41).

3.3.5.6 Process cic_comb_proc

This filter operates at fraction of the speed of the integrator section, as configured by the decimation rate. When in the SAMPLE state, the value of i0 is assigned to c0.

The delay elements i2d1 and i2d2 adjust the value of M the delay parameter. In our configuration, we have $M = 1$ by using assigning internal registers c0 to i2d1, as in the following code:

```
if state = sample then
  c0  <= i0;
  i2d1  <= c0;
  --i2d2  <= i2d1; -- extra delay M = 2
  c1  <= std_logic_vector(signed(c0) - signed(i2d1));
  dvo <= '1';

end if;
```

A version of this code which sets $M = 2$ would be:

```
if state = sample then
  c0  <= i0;
  i2d1  <= c0;
  i2d2  <= i2d1; -- extra delay M = 2
  c1  <= std_logic_vector(signed(c0) - signed(i2d2));
  dvo <= '1';
end if;
```

It is also in this process that the DVO process is set, which is used by the process CALC_CH auto-trigger process, which determine which filters are being activated sets a signal so that output to USB is automatically configured.

3.3.5.7 Output assignments

The output signals

```
y_out <= c1;
count_out <= std_logic_vector( unsigned(count_out_i) + 1 );
```

3.3.6 Component CALC_CH

The CALC_CH component provides automatic triggering based on the incoming DVO signals from the filters. This triggering information is sent to MCU_IO where it is stored, MCU_IO then sets a stream DVO which is sent to the MCU. Deciding when to trigger that a particular port is "ready" depends on the the configuration of the sampling windows. A port may be ready each time an A and T trigger occurs, or in other configuration A, B and T. Furthermore, after a sampling window configuration change, the trigger may also change. Therefore the logic of the auto-trigger processes decides if filter triggering since last decimated out is same as previous out, then trigger, otherwise wait for all data valid signals. This is handled in just one process:*AUTO_TRIGGER*.

Internal signals are listed in table A.11

3.3.6.1 Process AUTO_TRIGGER

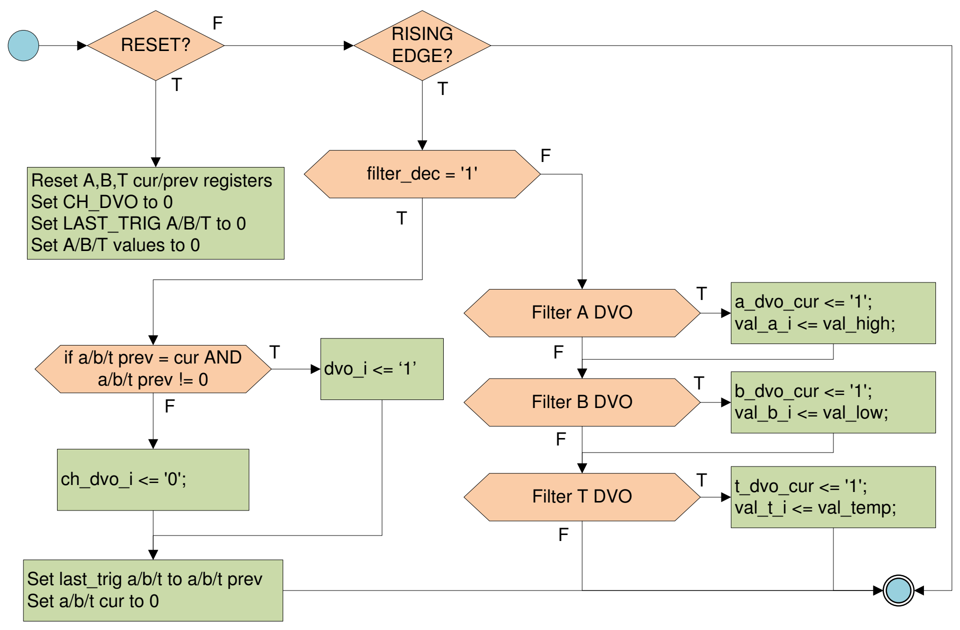


Fig. 3.45: Component CALC CH process AUTO TRIGGER

3.3.7 Component RESET_SYNC

This module concerns with the handing of resets in the system. Each component is designed using asynchronous resets, i.e. the CLK and the internal reset signal is sensitivity list of each process). One problem with asynchronous resets, is the release of the reset state which might occur near a clock edge, which could result in a metastable (i.e. unstable/unknown). To avoid this problem, the reset signal itself is synchronized, using a two flip-flop reset synchronizer circuit [Rushton 2011].

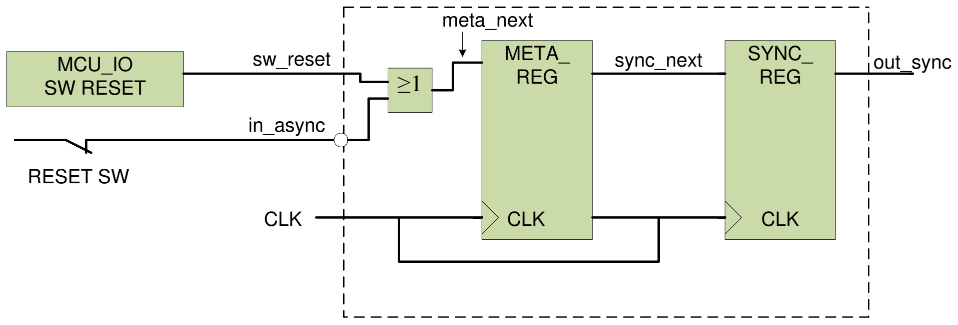


Fig. 3.46: Component RESET_SYNC process RESET_SYNC

The internal signals needed for this two-flip flop synchronizer are:

Tab. 3.11: Component CIC internal signals

Signal	Purpose
meta_reg	Input register
sync_reg	Output register
meta_next	Input to to meta_reg
sync_next	Output of meta_reg

3.3.7.1 Process SYNC

The incoming asynchronous resets is OR'd so that either will cause a reset. The RESET_SW is normally closed, the fore the signal is inverted at input. At the output of the OR gate, the asynchronous reset is stored at the clock edge of meta_reg. To further reduce the chance of metastability, a second flip-flop is used, sync_reg. The output of sync_reg is assigned to out_sync and drives the asynchronous reset circuity within the rest of the system.

3.3.8 Component OR GATE 2

This is a simple two-input gate used by directly in FET_ROOT, used to OR the enable signals that enable ADC_READ. This is contained within FET_ROOT for simplicity of the ADC_READ component, but could also be directly implemented in ADC_READ.

When only one of the ports is configured, this circuit allows the either to trigger ADC_READ conversion cycles. When the sampling windows of port 1 and port 2 do not match, the enable signal will be set, but will not result in in a ADC_READ conversion cycle if a one is already active, therefore it is important to configure the sampling windows of both ports with this in mind.

This component's usage in FET_ROOT is displayed in figure 3.47.

Tab. 3.12: MATLAB Instrument Control Toolbox objects[Mathworks 2011]

Object type	Object name	Description
Interface	serial	Create a serial port object
Interface	fclose	Disconnect the interface object from the instrument
Interface	fopen	Connect interface object to instrument
Data input	fscanf	Read data from instrument, and format as text
Data output	fprintf	Write text to instrument
Information	instrhwinfo	Information about available hardware
Information	instrfind	Read instrument objects from memory to MATLAB workspace

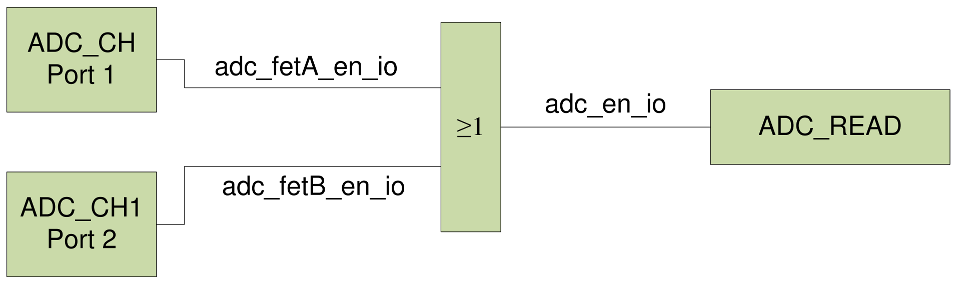


Fig. 3.47: Component OR_GATE_2 usage

3.4 Higher-level system software

The higher-level system refers generally to a computer or workstation running an operating system such as Windows or Linux and software such as MATLAB or LabView to automate the measurement process. A simple interface was developed using the MATLAB programming environment with the Instrument Control Toolbox.

MATLAB Instrument Control Toolbox

This software-addon for MATLAB, which automates serial port communication, is used to communicate with the virtual COM port serial driver that interfaces with the USB interface connected to the measurement system. Serial communication is configured by configuring and a creating a serial object, to which and from which data can be read[Mathworks 2011] via the MATLAB environment.

Instrument control toolbox functions and objects needed to interface the measurement system via the virtual COM port driver are summarized in table 3.12.

MATLAB programming environment

Using the C-like MATLAB programming language, a program was written utilizing the instrument control toolbox objects listed in table 3.12 to create a connection via the serial/USB interface that configures the measurement system, stores measurement data, and displays the results graphically.

CHAPTER 4

Measurement certainty

References to *precision* refer to the repeatability of multiple measurements to show the same results, while *accuracy* refers to the closeness of a measurement to its true value [Drosg 2009]. This chapter discuss limitations of hardware that could affect precision and accuracy of measurements, limitations of the system design, and error propagates through the system affection over all measurement accuracy and precision.

4.1 Sources of Error

The components that make up the measurement system are limited in their capability and prone to external error. This section discusses major limitations.

4.1.1 ADC Quantization

The ADC's resolution is limited to 16-bit, i.e. the number of distinguishable voltage levels is discrete and limited to $2^{16} = 65536$ levels. The difference between the actual value and the returned code is refereed to as *quantization error*[Instruments 1995]. The smallest step is defined as

$$1\ LSB_{ADC} = \frac{FSR}{2^n - 1} \tag{4.1}$$

where FSR is the full-scale range of the ADC and n is the number of bits. Four our ADC, 1 LSB is:

$$1\ LSB_{ADC} = \frac{10V}{2^{16} - 1} = 152.6\mu V \tag{4.2}$$

with FSR determined by the internal ADC reference voltage (5V) and the ADC RANGE configuration pin (2X), $FSR_{ADC} = 2 * 5V = 10V$.

Since the ADC samples voltage levels of the current-measurement circuit, and the current is measured in two ranges: low ($\pm 200\mu A$) and high ($\pm 2000\mu A$), the smallest distinguishable step in measured current is then for both ranges:

$$step_{low} = \frac{400\mu A}{2^{16} - 1} = 6.10nA \tag{4.3}$$

$$step_{high} = \frac{4000\mu A}{2^{16} - 1} = 61.0nA \tag{4.4}$$

4.1.2 DAC

Like the ADC, the DAC is limited to 2^{16} discrete output values. The smallest voltage step possible is defined as 1 LSB in the same way as the ADC [Instruments 1995]:

$$1LSB_{DAC} = \frac{FSR}{2^n - 1} \tag{4.5}$$

$$1LSB_{DAC} = \frac{30V}{2^{16} - 1} = 457.8\mu V \tag{4.6}$$

FSR is 30V as configured by the DAC reference voltage (5V) and the configuration of the DAC gain register (6X) ($FSR_{DAC} = 6 * 5V = 30V$).

Since the sets the bias and drain voltages in the FET measurements, the step does not directly introduce error in the current-measurement circuit, however differences in expected DAC output and actual output would cause error in the representation of the current flow at a certain biased/sourced voltage.

4.1.3 Offset error

Offset error can occur in both the ADC and DAC. Offset error for the ADC is the mid-step value returned when the input voltage is zero; for the DAC, it is the step value (output value) when digital input code is zero [Instruments 1995]. The ADC offset can be determined through by grounding the input and applying the input-histogram technique described in section 4.1.6.1. The DAC offset can be calibrated by setting the corresponding input code for 0V (0x8000) and measuring the output, setting an offset.

In our measurement system, the offset of both these is adjusted through zero-point offset calibration and dynamic calibration, limited by the precision of the calibration reference resistances. (See section 4.2.4)

4.1.4 Gain error

Gain error is defined as the difference between the nominal and actual gain points on the transfer function after the offset error has been corrected [Instruments 1995]. The DAC also provides the capability to adjust for this error in hardware through its user calibration techniques[Instruments 2009].

4.1.5 Other error

Other factors which may affect the ADC and DAC include [Instruments 1995][Devices]:

Differential Non-Linearity (DNL) The difference between the measured and ideal 1 LSB change between any two adjacent codes.

Integral Non-Linearity (INL) The maximum deviation of the ADC/DAC transfer function from a straight line.

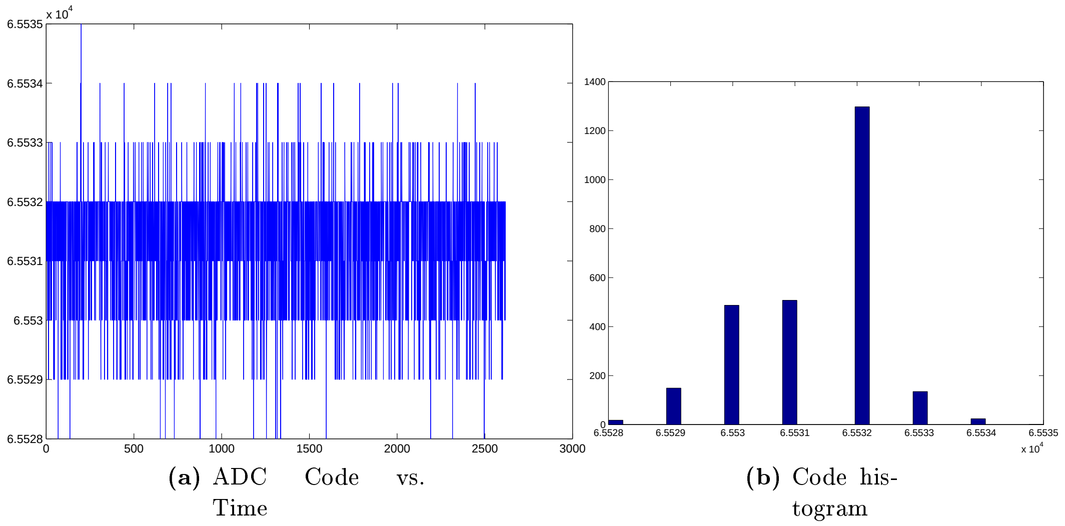
Aperture error The time which the sample/hold circuity needs to go from sample to hold.

4.1.6 Noise

4.1.6.1 ADC input noise

Considering the ADC, which measures the voltage level of the current-measurement circuit, there are three primary sources of error: quantization, ac signals, and wide-band noise [Ruscak 1995].

Of these, wide-band noise is measurable by the *input histogram technique* [Ruscak 1995]. Code transition noise, or input-referred noise, can be measured by a technique referred to as grounded-input histogram. In this test, the ADC is connected to ground and ADC is plotted on a histogram. The resulting histogram should be approximately Gaussian, otherwise there may problems with pc board layout, grounding or the power supply[Kester 2008]. Tests of the revision 1 board, as shown in figure 4.1, show an offset as well as a skew from Gaussian distribution in the histogram, with an expected value of 65536 for 0 in this particular test.



(a) ADC Code vs. Time **(b)** Code histogram

Fig. 4.1: ADC grounded input test

The input RMS noise is given as the standard deviation of the grounded input histogram, using this, peak-to-peak input noise can be calculated by [Kester 2008]:

$$\text{P-P Input Noise} \approx (6.6) * (\text{RMS Noise}) \tag{4.7}$$

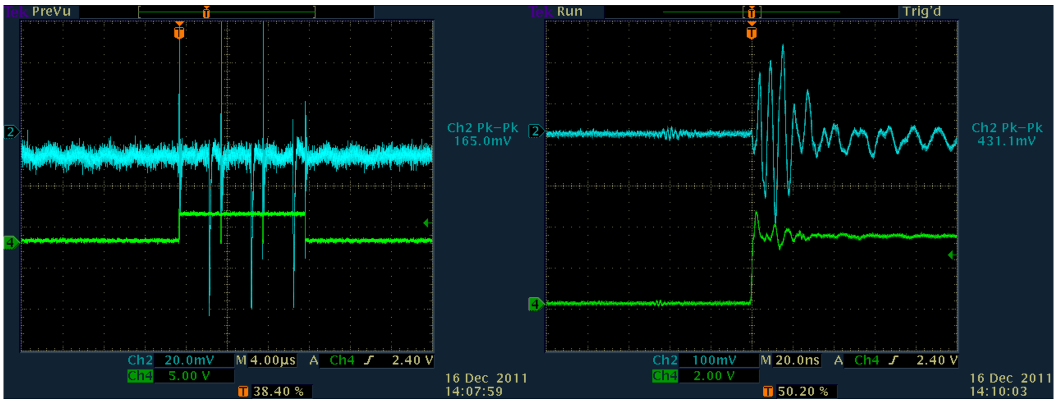
Effective resolution is the ratio of the full-scale range to the rms input noise (LSBs) [Kester 2008]:

$$\text{Effective resolution} = \left(\frac{2^N}{\text{rms input noise}}\right) \tag{4.8}$$

where N is the number of bits in the ADC, with $N = 16$
The effects of input noise are reduced through the use of the CIC filter [Hogenauer 1981].

Conversion noise

The ADC conversion process after CONVST creates noise due to resistor noise [Kester 2008], shown in figure 4.2 (I_{meas} (top/blue), ADC_BUSY (bottom/green)). While this is less of an issue after a CONVST signal has switched the sampled-and-hold circuitry of the ADC to hold, it may result in increased EMI that may interfere with other measurement boards.



(a) (b)

Fig. 4.2: ADC conversion noise after CONVST

4.1.6.2 Environmental noise

The test environment itself (i.e. lab) may lead to noise, the intended application of the measurement system involves multiple measurement boards performing measurements in close proximity. The factor that such an application plays is not yet tested.

If such a noise is an issue, a possible solution might be adjustment of the DRIVE STRENGTH port configuration on the MCU, to help reduce EMI [Instruments b].

4.1.7 Temperature

The temperature of the component can cause a loss of precision. The voltage reference REF5050 changes by 3ppm/C.

4.1.8 Component Losses

Some components may change over time, e.g. the voltage output of the reference used by the DAC may drift over time. The reference voltage is however relatively low-drift, changing by 0.001% over 1000 hours of usage [Instruments c].

4.2 FET Measurement system limitations

4.2.1 Sampling Window

Sampling window speed

The sampling window configuration determines when the ADC starts sampling, how often it samples and when it will stop sampling. Configuration of the sampling window depends on limits of the ADC, esp. maximum sampling speed. ADC_READ is designed to start sampling and set DVO when all six ADC channels have returned values. This process takes 81 clock cycles (at 20 MHz FPGA clock) to complete.

Therefore, the sampling window configuration must be configured with this limitation in mind. Because the sampling window is configured in number of clock cycles, the minimum switched biasing PERIOD determined by this process, $T_{samp-ADCREAD}$, is then 81 clock cycles. The actual maximum ADC sampling rate $T_{samp-ADC}$, is:

$$T_{samp-ADCREAD} = 81 * 50ns = 4.05\mu s \tag{4.9}$$
$$T_{samp-ADC} = (250KHz)^{-1} = 4\mu s \tag{4.10}$$

For the sake of safety and possible future optimizations, calculations use the worst case sampling rate, $T_{samp-ADC}$ or $T_{samp-ADCREAD}$ depending on the calculation.

If configured less than to a value less than 81, a CONVST will be set on the ADC but an ADC_READ DVO will never be output.

Filter selection

At the minimum period, only the A filter would be active, since the entire period of the switched biasing signal would cycle, returning to the A filter. The minimum speed for to sample both switched biasing levels, is then:

$$T_{samp-full} = 2 * 81 * 50ns \tag{4.11}$$
$$= 162 * 50ns = 8.1\mu s = 123456.79Hz \tag{4.12}$$
$$\tag{4.13}$$

MIN PERIOD (1 filter)	81 (clock cycles, 4.05 μs)
MIN PERIOD (2 filter)	162 (clock cycles, 8.1 μs)
MAX PERIOD	2^{32} (clock cycles, 214 s)

Current measurment delay

The sampling window logic was designed based on the switched biasing *control* signal. This signal controls the switched biasing signal, which in turns results in a change in the current-measurement circuit. This change however requires a certain amount of time (experiments show approximately 3 clock cycles, or 150 ns) to compensate for the new voltage. Therefore the ADC sampled voltage level directly *after* the switched biasing controlled voltage switch is unstable (see figure 4.3). Therefore it is preferable not to configure the sampling window START too close to this edge, instead at least 3 clock cycles after to give the current-measurement circuit time to adjust.

Conversions that occur close to the end of the switched biasing window are less of an issue, however the settling time of I_{meas} may play another factor.

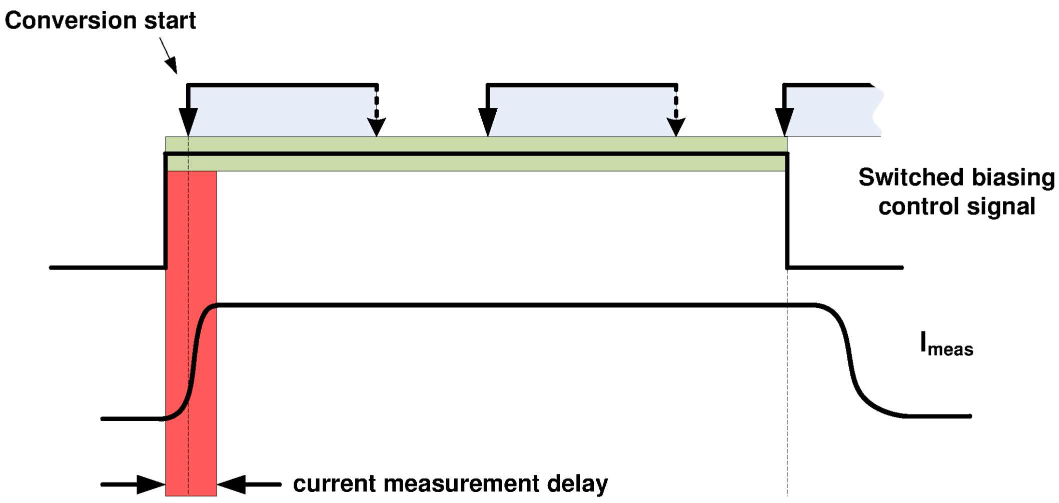


Fig. 4.3: Switched biasing control signal delay

Figure 4.4 shows an actual measurement of this delay, with the switched biasing control signal bottom/yellow and the current measurement signal I_{meas} top/green. In this case the delay was 40.8 ns.

Settling time

Depending on the speed of the switched biasing signal, the current measurement delay may play a more significant factor in measurements, especially with high-frequency switched biasing configurations, meaning that samples close to the end of the sampling period are more precise.

4.2.2 Decimation rate & measurement speed (data output rate)

The measurement speed configuration register (data output rate) on the FPGA determines the speed at which MCU_IO DVO signals to the MCU are set. This value is stored at two 16-bit memory locations per measurement port, for a total of 32-bits each.

This value dictates the decimation rate of the CIC filters, setting this value too large would lead to integration larger than what the CIC registers can store. This is determined by considering the CIC gain (G):

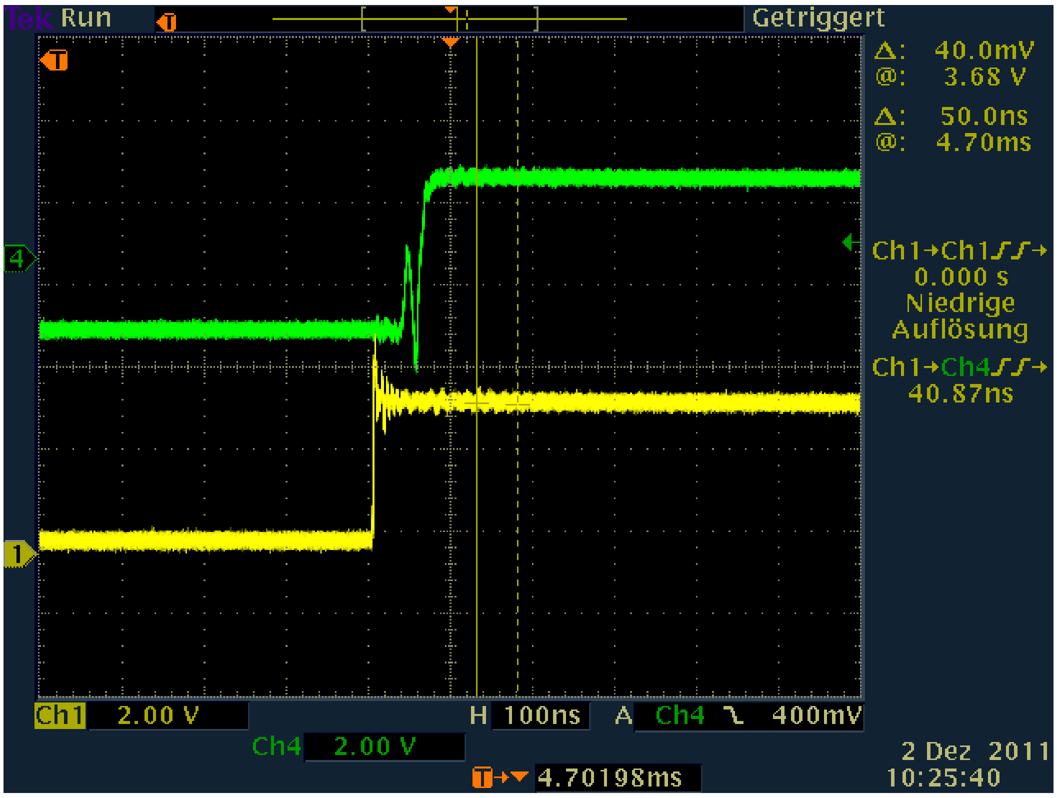


Fig. 4.4: Switched biasing transient delay, switched biasing control (bottom) and current measurement signal (top)

$$G = (R * M)^N \tag{4.14}$$

where N is the number of CIC stages (N=1), R is the decimation rate, and M is the comb delay (M=1).
The measurement speed and CIC register store was designed to work together at a measurement output rate of at least 10 Hz (100 ms), therefore measurement speed rate should be be set no higher than 100 ms (2000000 clock cycles)

The maximum decimation rate R, can be determined by the specified minimum data output frequency 10 Hz (100 ms), $f_{meas-min} = T_{meas-max}$ and the minimum sampling period of the ADC (4 μs).

$$R_{max} = \left\lceil \frac{T_{meas-max}}{T_{samp-min}} \right\rceil = \left\lceil \frac{100ms}{4\mu s} \right\rceil = 25000 \tag{4.15}$$

The minimum decimation rate is met when the ADC sampling rate is the same as the data output rate, $R = 1$.

The number of bits required for CIC integrator storage is:

$$B_{out} = \lceil N * log_2 R * M + B_{in} \rceil \tag{4.16}$$

where B_{out} = number of output bits, and B_{in} = input bits from ADC = 16.

Because the data output rate register is 32 bits, and the CIC output register is also 32 bits, the maximum output rate would be

$$\begin{aligned} B_{out} &= \lceil N * log_2 R * M + B_{in} \rceil & (4.17) \\ 32 &= \lceil 1 * log_2 R * 1 + 16 \rceil & (4.18) \\ R &= 65536 & (4.19) \end{aligned}$$

This would give a gain in the CIC integration stage of:

$$G = (R * M)^N = (65536 * 1)^1 = 65536 \tag{4.20}$$

The number of bits of storage required for this gain is based on the minimum and maximum values of ADC input values, because the ADC is 16-bit, two's complement, the range of input is -32768 to 32767, giving a maximum and minimum storage requirement of:

$$\begin{aligned} 2^{16} * 65536 &= 4294967296 & (4.21) \\ b_{storage} &= \lceil log_2(4294967296) \rceil = 32 & (4.22) \end{aligned}$$

which is within the limits of the storage registers, therefore the true lower data rate is:

$$65536 = \frac{R_{(max)}}{4.05 \mu s} = 265.4208 ms (3.78 Hz) \tag{4.23}$$

The maximum data output rate depends on the speed of the MCU request, which varies depending on the number of memory locations must be retrieved by a streaming data request, which itself is dependent on the sampling window configuration. Exceeding the rate that the MCU stream process can handle will result in lost measurements. In this case, the STREAM_WARN bit will be set. Measurements of the STREAM BUSY signal set by the MCU during data output are an indicator of absolute maximum output frequency. For one port, one filter (minimum for output), STREAM BUSY is set for 1.804 ms. For both measurement ports, both filters on both ports, STREAM BUSY is set for 4.637 ms. Therefore we can conclude absolute maximum output frequency of:

$$\frac{1}{1.804 ms} = 554.3 Hz \tag{4.24}$$

However, this rate will be actually lower due to overhead time between STREAM BUSY output periods.

4.2.3 Current range

The current measurement circuit is designed to handle currents up to $200\mu A$ in the low-current mode, and $2000\mu A$ (2mA) in the high current range. This is limited by the input range to the ADC. Although the ADC can be configured to handle a range of $4 * V_{REF} = 4 * 5V = 20V = \pm 10V$, the ADC's RANGE pin is configured to the 2X setting, i.e. $2 * V_{REF} = 2 * 5V = 10V = \pm 5V$ in order to give a smaller LSB_{ADC} of 0.152mV rather than that resulting from the 4X setting ($4 * V_{REF} = 4 * 5V = 20V = \pm 10V$), and resulting in a LSB_{ADC} of 0.305mV.

Exceeding the current limits may be possible if the measured FET breaks down quickly, causing a short circuit and the measurement rate is slow (thereby current protection logic is not quickly activated). Tests exceeding the current limits were not performed and its exact behavior is unknown.

4.2.4 Calibration

To compensate a variety of losses within the system, e.g. contact resistance in relays, calibration is used to correct for these errors. Two calibration techniques are used, zero-point and dynamic. In the first revision of the system board, the calibration techniques had not yet been integrated onto the pc board. During the course of the thesis, a second revision of the pc board was produced which integrates these circuits into the design.

Zero-point (offset) calibration

In pc board revision 1, it was shown through grounded input measurements, that the 0V was shifted -4 LSB_{ADC}. This offset depends on a number of factors, including relay resistance, amplifier error, component temperature and with variations in component contact resistance with the pc board, i.e. soldered joints on the pc board.

To compensate this error, zero-point calibration technique was implemented in order to improve the accuracy of measurement. During measurement however, the resistance value through the relay R_{ON} changes depending on contact voltage and component temperature. Therefore, measurement of a FET across a range of voltages leads to a changing R_{ON} value, changing accuracy. For this reason, a dynamic calibration circuit was integrated as well into the second revision of the pc board to adjust for changing voltages.

Dynamic calibration

The dynamic calibration uses two high-precision reference resistances 10KΩ and 100KΩ to determine the offset factor by which the measurement is shifted due to R_{ON} and due to the amplification error of the operational amplifiers (for more detail, refer to section 2.2.4.2). This circuit is itself limited by the precision of the reference resistance which can change slightly with temperature and inherent manufacturing differences.

Dynamic calibration resistance have a precision of ±0.01%. Due to time constraints, implementation and measurement on rev. 2 pc boards was not possible.

4.3 Error Propagation

In additon to the accuracy of the reference resistances, the op-amp and ADC also play an important role in the overall accuracy of the system. Gain and offset error of the op-amp and offset error of the ADC lead to error that propagates throughout the measurement process.

For the ADC used in the system, accuracy is specified as ±0.023% FSR max, and ±0.004% FSR typical error (bipolar zero-scale error)[Devices]. With a FSR of 10V (±5V), and a LSB of 0.152mV, maximum error resulting from the ADC is [Devices]

$$0.152mV \pm 0.023\% = 0.152mV \pm 34.96nV \tag{4.25}$$

and typical error of

$$0.152mV \pm 0.004\% = 0.152mV \pm 6.08nV \tag{4.26}$$

This is combined with the maximum specified offset error of the OPA4277UA operational amplifier used in the current measurement circuit($\pm 50\mu V$), causing a maximum error of the voltage representing current of [Instruments 2005]:

$$\pm 50\mu V \pm 34.96nV = \pm 50.03\mu V \tag{4.27}$$

which is less than 1 LSB_{ADC}, but may cause rounding and a resulting error $\pm 1 LSB_{ADC}$. For the low measurement range, this 6.1nA, and for the high range, 61nA.

Chapter 5

Measurements and results

In this chapter, measurements demonstrating the capabilities and limitations of the measurement system are performed in order to demonstrate measurement system functionality, accuracy and performance. The measured components are the two internal calibration resistances and a field-effect transistor connected to the measurement port of the system.

5.1 Calibration resistances

In order to determine measurement system accuracy, normal measurement relays (GATE and VDS) are switched off and the calibration switches (CAL1 and/or CAL2) are enabled. The two high-precision (±0.01%) reference resistances (10KΩ and 100KΩ) can be independently enabled and disabled through the user interface. By setting the drain voltage the current through the calibration relays can be measured by the system without any external connections. Accuracy can then be determined by performing multiple measurements over a range of voltages and comparing measured with the expected results.

Accuracy can be determined by calculating the standard deviation (σ) of k measured current values I at a given drain voltage[Drosg 2009]:

$$\sigma = \sqrt{\frac{1}{k}\sum_{n=1}^{k}(I_i - \mu)^2} \tag{5.1}$$

where

$$\mu = \frac{1}{k}\sum_{i=1}^{k} x_i \tag{5.2}$$

To evaluate performance over a range of voltages, the standard deviation can be used to determine the overall accuracy over a range of N measurements [Drosg 2009]:

$$\overline{\sigma} = \sqrt{\frac{1}{N}\sum_{N=1}^{N} \sigma^2} \tag{5.3}$$

In order to determine precision, a linear fit of the measured current and applied voltage is performed, giving current as a function of voltage, the deviation of which from the expected behavior of the calibration resistance determines the amount of bias and gain error[Fetzer 1965].

The absolute error, e, for any given current measurement, is determined by finding the difference between the measured value of the current I_{meas}, and the expected current, I_{calc} , at a particular drain voltage E_{drain}:

$$e = I_{meas} - I_{calc} \tag{5.4}$$

Where

$$I_{calc} = \frac{E_{drain}}{R_{ref}} \tag{5.5}$$

where R_{ref} is the resistance of reference resistance.

5.1.1 100KΩ characterization

First, the 100KΩ resistance is measured N times using a measurement system configuration as summarized in table 5.1. With this resistance, the maximum current that can flow as result of applying the maximum output voltage at the DAC ±15V can be found by:

$$I_{max} = \frac{\pm 15V}{100K\Omega} \tag{5.6}$$

$$I_{max} = \pm 150\mu A \tag{5.7}$$

Tab. 5.1: 100K, low decimation rate test configuration

Parameter	**Value**
Zero-point calibration	Enabled
Dynamic calibration	none
Drain voltage step size	501 (DAC input code, prime)
Sampling period	10KHz
Data output rate	10 Hz
Duty cycle	100% (constant bias)
Phase A (A-filter start)	$0.5\mu s$
Phase B (A-filter end)	$25\mu s$
Phase C	0 (disabled)
Phase D	0 (disabled)
Range	Auto
Auto low-to-high range	$183\mu A$
Auto high-to-low range	$150\mu A$

This puts the range of measurement in low-range for the system, as configured by auto-ranging and the configured high-to-low and low-to-high range switching points.

5 measurements, 0 to 15V

The voltage, E_{drain} is measured in the range from 0 to 15V, five times (N=5). The results of all five measurements across the full range of measurement are plotted with voltage on the x-axis and current on the y-axis, as shown in figure 5.1. The graph shows no large deviations and is apparently linear. In figure 5.2 an enlarged view in the range from 0 to 1.6V is given, shows deviations between individual measurements.

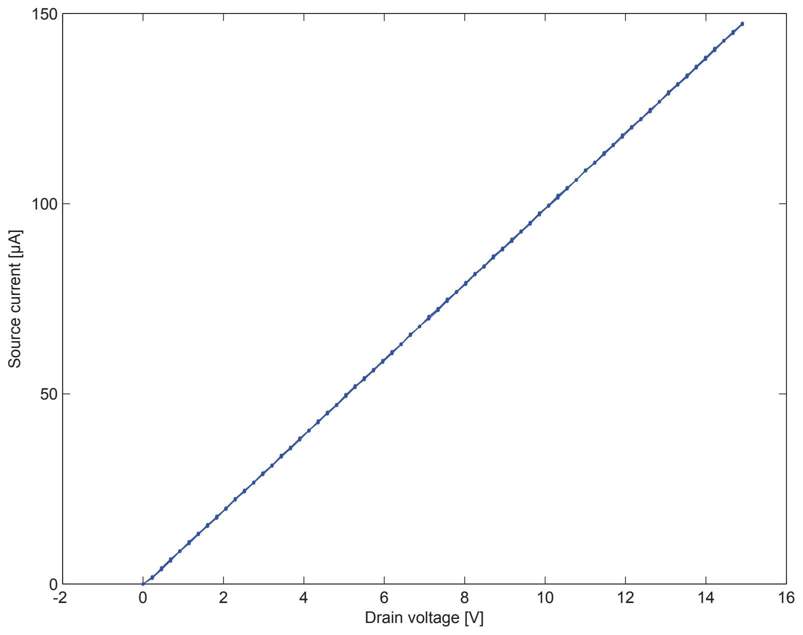


Fig. 5.1: 100KΩ characterization

Absolute error

In order to determine the error of a measurements across the *full* range of output given by the DAC ($\pm 15V$), the test is repeated and the error is calculated for each value of the drain voltage E_{drain} as given by Eq. 5.4. The absolute error is then plotted with applied drain voltage on the x-axis and the error on the y-axis, as shown in figure 5.3. It can be seen that the absolute error is in the range of $\pm 1.5\mu A$.

Accuracy and Precision

To determine the precision of the measurement, the test is repeated 100 times to give a more larger sample size. Voltage is measured in the range from 0 to 2V. The mean of the measurements at each voltage is calculated and plotted in figure 5.4, with drain voltage on the x-axis and current on the y-axis. The standard deviation was calculated, multiplied by three and plotted as error bars at each measurement point to show the likely range in which a measurement will fall.

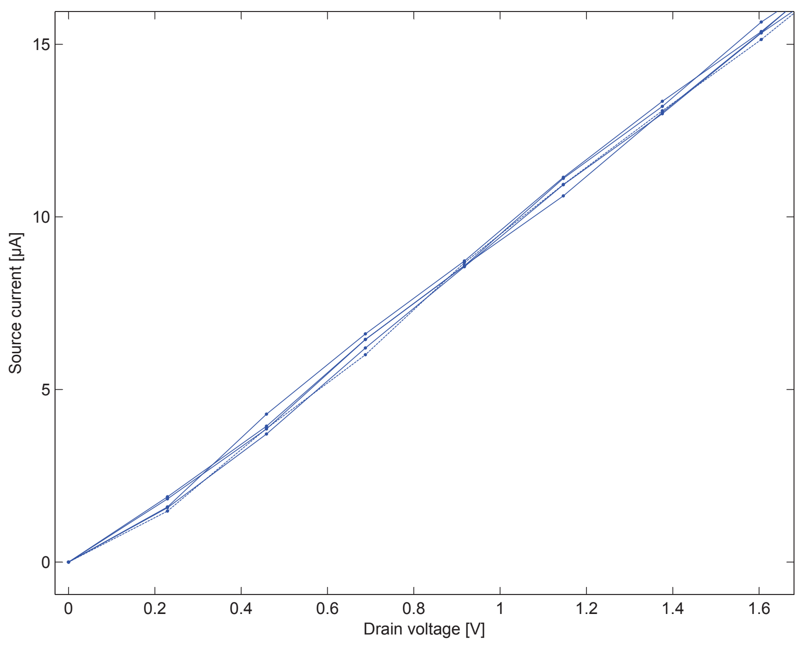


Fig. 5.2: 100KΩ, variations in measurements

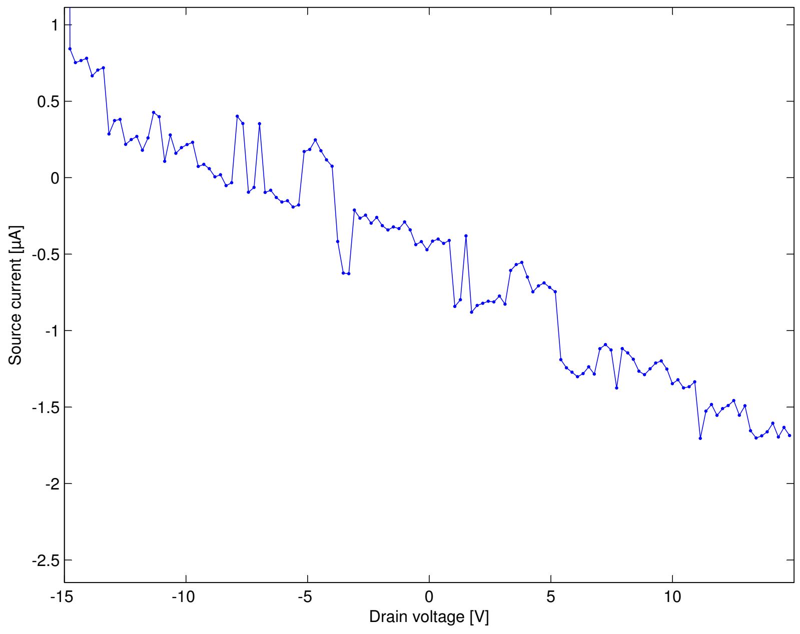


Fig. 5.3: 100 KΩ full-scale absolute error

The resulting measurement mean values to a linear model, by using the MATLAB polyfit() function, with n=1 for a linear model, returned is a function for the measured current at any given drain voltage v derived from the mean of the

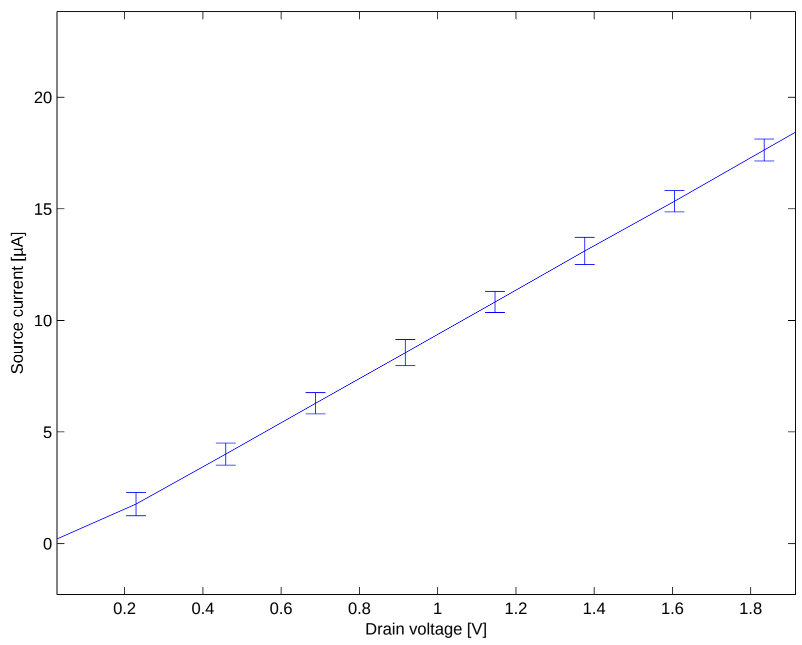


Fig. 5.4: $100K\Omega$, average of 100 measurements, enlarged 0 to 2V

measurements:

$$I_{meas} = 9.85 * 10^{-6} * v + 344 * 10^{-9} \tag{5.8}$$

Although the measurement system is calculated for zero-offset, it can seen in eq. 5.8 that there there still remains an offset of approximately 344nA. Additionally, by analyzing the first coefficient of eq. 5.8, the measurement includes additional gain offset which can be compensated for with calibration.

Increased decimation rate

In figure 5.3 it can be seen that the absolute error is not constant. This is due to input noise, causing variations in measurement. By increasing the sampling window size, the decimation rate is also increased. In figure 5.5, the result of repeating the measurement such that the sampling window size is increased by a factor of 10 is shown. In doing so, the signal-to-noise ratio of the measurement is increased. Additionally, the sampling window is changed to a later period within the sampling period.

5.1.2 10KΩ characterization

In order to determine the behavior in the system in high current range ($\pm 2000 \mu A$), the calibration resistance 100KΩ resistance is disabled and the 10KΩ calibration

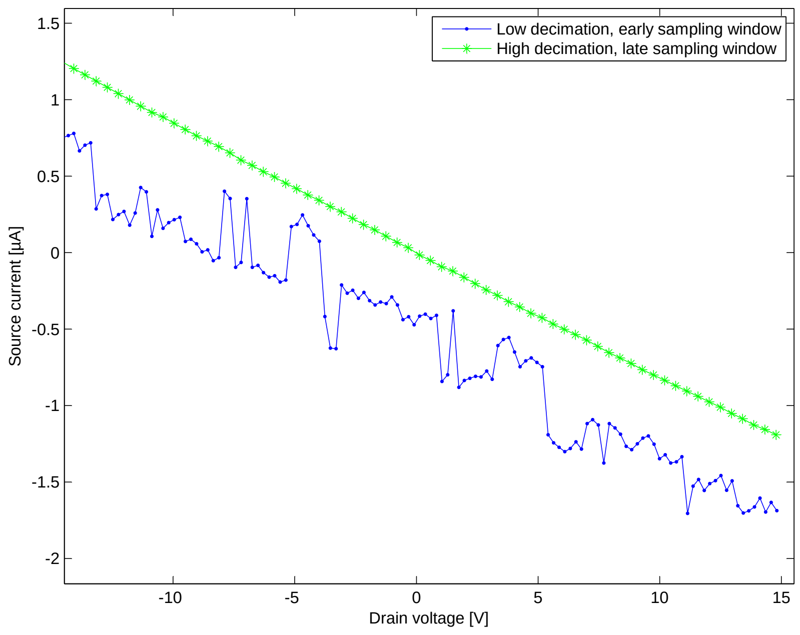


Fig. 5.5: $100K\Omega$, low decimation rate vs. high decimation rate

reference resistance is enabled. Measurements in the range from of $\pm 15V$ are performed with applied voltage E_{drain} on the x-axis and measured current on the y-axis as shown in figure 5.6. The resulting absolute error as calculated by eq.5.4 is plotted as shown in figure 5.7. There it can be seen that the absolute error of the measured current varies between -0.5 μ and $+3\mu$.

Measurements with the 10KΩ resistance along the full-scale range of possible voltage ±15V puts the resulting current into both the high and low current ranges, with the maximum current given by:

$$I_{max} = \frac{\pm 15V}{10K\Omega} \tag{5.9}$$

$$I_{max} = \pm 1500\mu A \tag{5.10}$$

The point at which the measurement system switched from low current mode to high current mode is dictated by the configuration of low-to-high and high-to-low auto range switching points. At current levels higher than the low-to-high level ($\pm 183\mu V$) the value of the measured current is limited to the resolution in the high current range (61nA). Although worse than that provided by the low current range(6.1nA), the difference is negligible due to the amount of current resulting from the 10K and 100K resistances. In the following section, these small changes of current are more apparent.

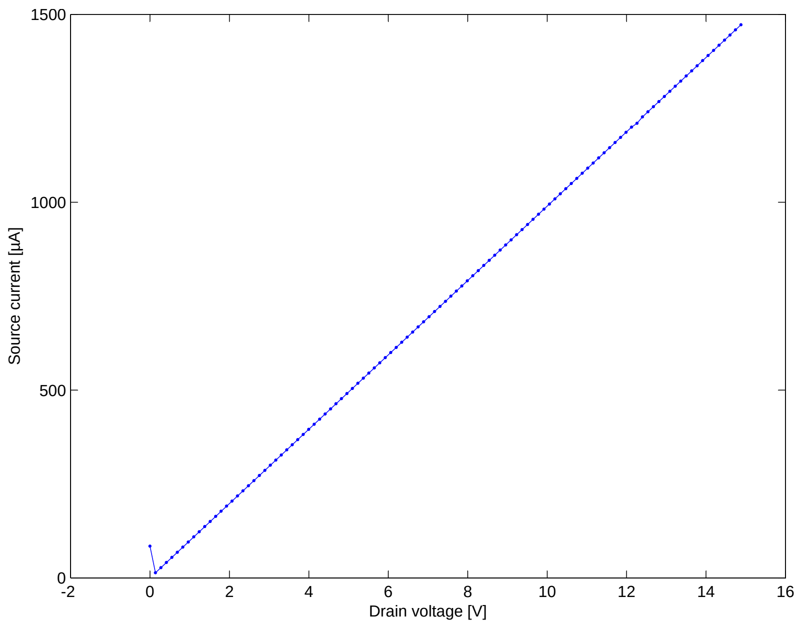


Fig. 5.6: 10 $K\Omega$ measured current

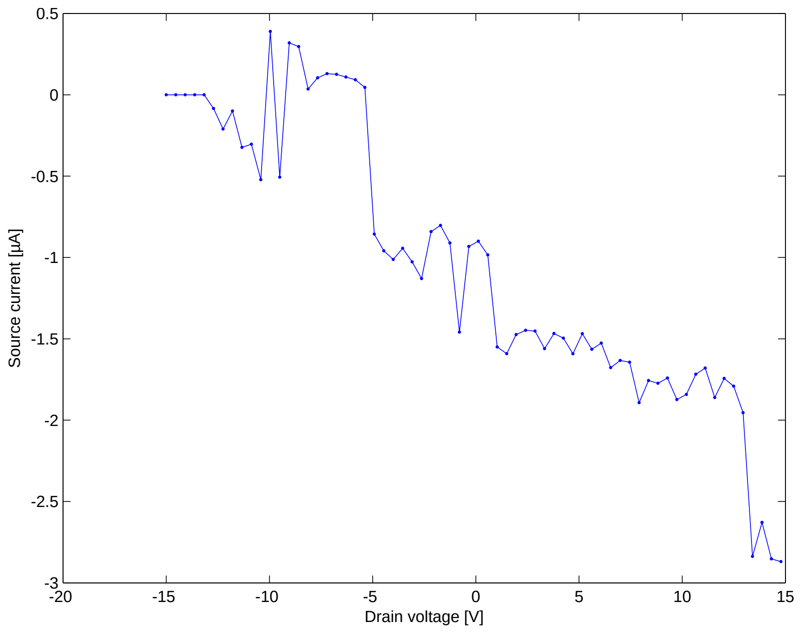


Fig. 5.7: $10K\Omega$, full-scale absolute error

5.2 Field Effect Transistor characterization

In this section, a field effect transistor is connected to the measurement port and characterized, i.e. a range of voltages is applied to its drain terminal and at each drain voltage, a range of biasing voltages is applied. The resulting current at each range produced a *family* of curves. The characterization of the transistor is the set of individual curves which describe the behavior of the transistor under various conditions.

5.2.1 Constant biasing of a transistor

In this measurement, a transistor is biased with constant voltage (no switched biasing). The measurement configuration is similar to that of the 100K resistor characterization, with the system configuration summarized in table 5.2.

Tab. 5.2: Transistor biasing configuration

Parameter	**Value**
Zero-point calibration	Enabled
Dynamic calibration	none
Drain voltage step size	501 (DAC input code, prime)
Gate voltage stop size	1V
Sampling period	10KHz
Duty cycle	100% (constant bias)
Phase A (A-filter start)	0.5 μs
Phase B (A-filter end)	25μs
Phase C	0 (disabled)
Phase D	0 (disabled)
Range	Auto
Auto low-to-high range	183μA
Auto high-to-low range	150μA

Two-curve characterization

Here, the transistor drain terminal is applied with a voltage in the range from 0 to 15V for each gate voltage 0V and 1V. The measured source current is plotted against the applied drain current for each gate voltage, producing the family of curves describing the tested transistor shown in figure 5.13. The plot shows the importance of the automatic ranging, providing higher resolution of the resulting characterization during the exponential rise in current flow at low applied drain voltages.

5.2.2 Eight-curve characterization

In figure 5.9, the characterization of the transistor by applying voltage to the drain of the transistor in the range of 0 to 15V for each of eight gate voltages is demonstrated, the gate terminal is biased from 0V to 8V with 1V increments.

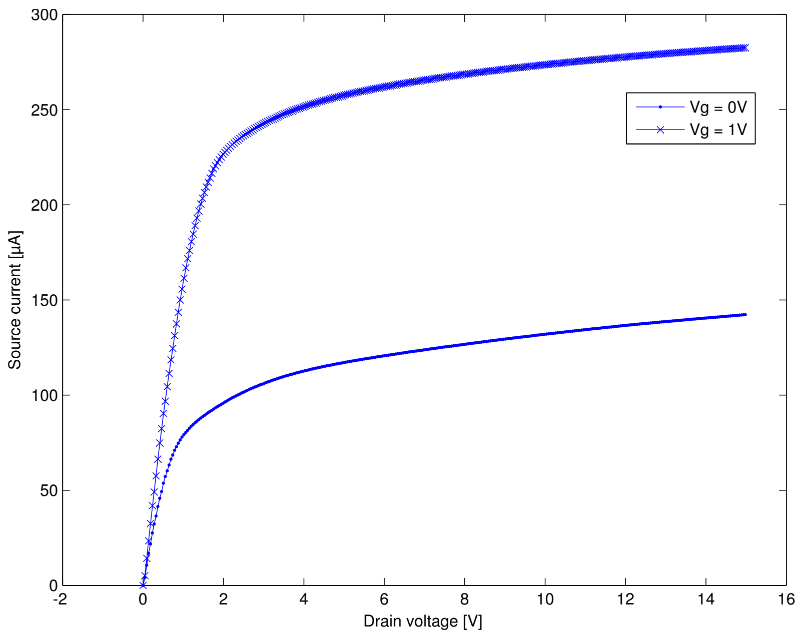


Fig. 5.8: Output characteristic curves, offset calibration

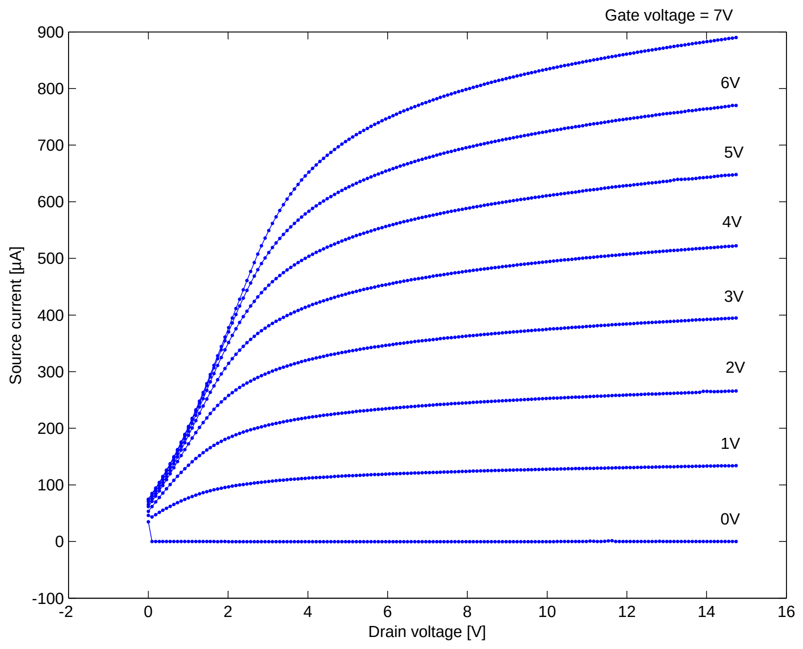


Fig. 5.9: Eight-curve characterization of a transistor

5.2.3 Transistor drift

One of the problems described by [Winkelman 2009] is transistor source current drift during constant biasing. In order demonstrate this effect, a constant gate and drain voltage is applied to the transistor, then a series of 1000 measurements (over a period of 5 minutes) is performed. In figure 5.10, the source current is plotted in the y-axis for each measurement, in the x-axis, showing the effect of constant biasing over time.

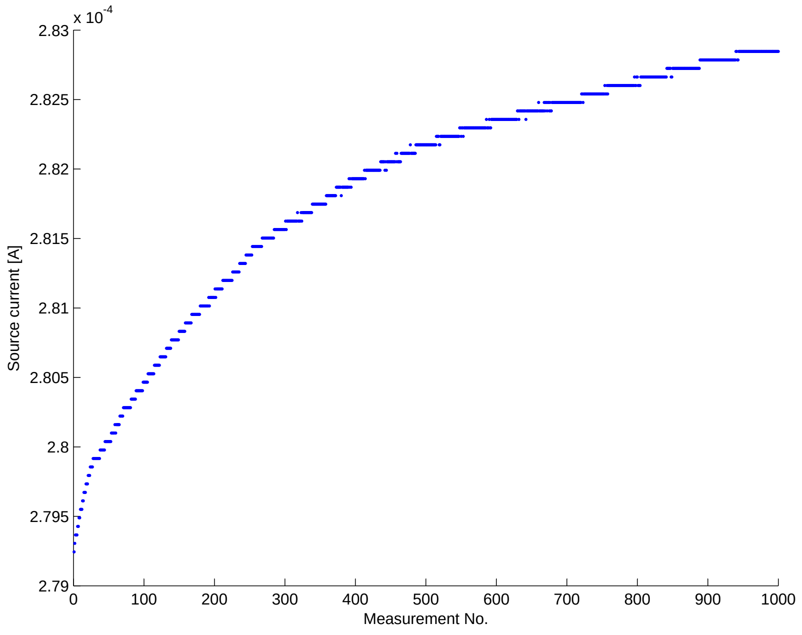


Fig. 5.10: Transistor constant biasing drift (5 minutes)

Voltage is applied to the transistor such that the resulting current is in the high-current measurement range, meaning that the smallest distinguishable change in current is 61nA. The effect is demonstrated in an enlarged view shown in figure 5.11. The individual current-measurement steps are clearly distinguishable. Overlap between steps is the result of noise, causing the rounded value to alternate between each step. Increasing the decimation rate would therefore increase the signal-to-noise ratio and reduce overlap between the steps.

In figure 5.12, the resulting current after biasing the transistor such that resulting current flow is in the low-current range. Again 1000 measurements are performed over five minutes, and the resulting current is plotted in the y-axis. Here the low-current 6.1nA step size and overlap is clearly distinguishable.

5.2.4 Switched biasing

To demonstrate the effectiveness of switched biasing as a technique to reduce transistor drift, two measurements are performed. First transistor is applied with a constant voltage of 2V at its drain and biased with a switched biasing signal, switching between 1V and 1.6V at a frequency of 10KHz and a duty cycle of 50%. The resulting source current from 300 measurements (over 1.5 minutes time), are shown in figure 5.13(top). Second, the measurement is repeated, again applying 2V to the

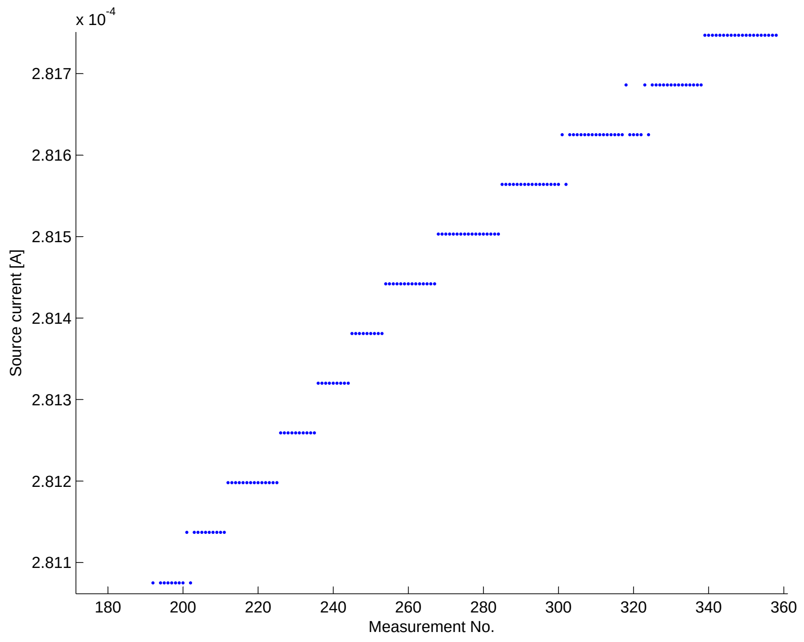


Fig. 5.11: High-measurement range 61nA resolution steps with overlap due to noise

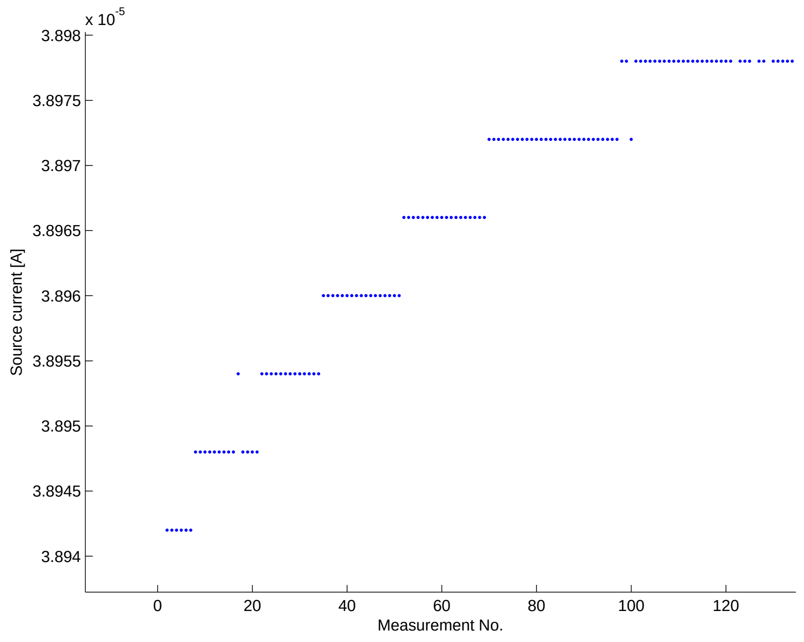


Fig. 5.12: Low-measurement range 6.1nA resolution steps with overlap due to noise

drain but with a constant bias voltage of 1V to the gate, the resulting source current is shown in figure 5.13(bottom).

In the case of constant biasing, the transistor continues to drift, from 280μA ($2.80 * 10^{-4}$A) to approximately 283μA ($2.83 * 10^{-4}$A) over the entire measurement period of 1.5 minutes. In the case of switched biasing, the amount of current flow starts to settle at around 275μA ($2.75 * 10^{-4}$A). in comparison to the constant biasing. The difference in current flow between the two transistors is beyond the scope of this thesis.

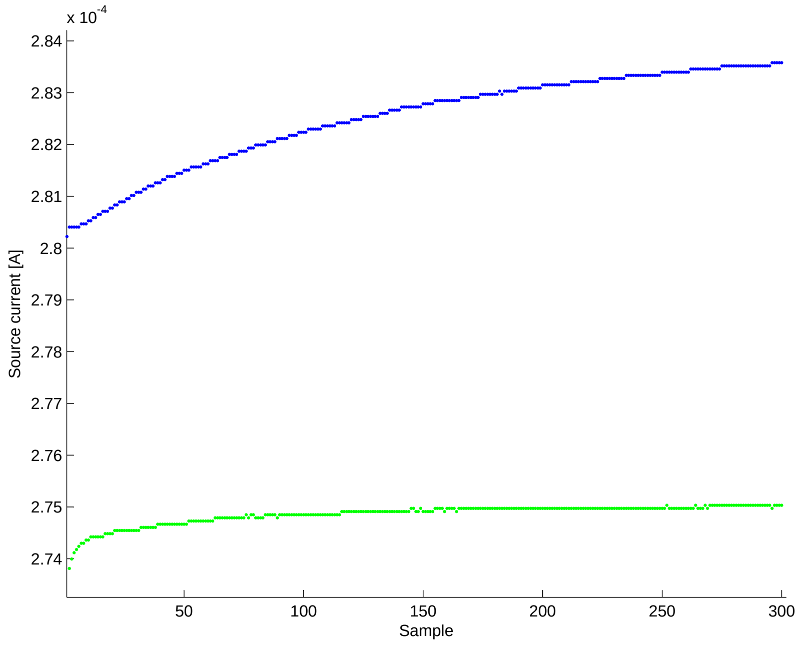


Fig. 5.13: Transistor constant biasing (top) and switched biasing (bottom) (1.5 minutes)

Chapter 6

Conclusions and recommendations

6.1 Conclusion

The primary goal of this work was to show that a FPGA-based hardware solution combined with a microcontroller- and pc-based software solution written for custom-built hardware using high-performance components, is able to accurately measure the characteristics of a transistor important for research. This is achieved while simultaneously coordinating the timing of these measurements with the switched-biasing technique, important for the reduction of undesirable effects common to transistor based sensors.

The implementation in chapter 3, explains the method by which this is achieved through software which is able to coordinate the sampling of a multi-channel analog-to-digital converter with the input of multiple CIC filters designed especially for this purpose. The timing of the filters is coordinated such that only valid measurement data is sent to the higher-level system, coordinated by the configuration of a measurement rate, that ensures constant and consistent measurement data is not lost due to speed limitations of the interface between the filter and the higher-level system.

The measurement data, ultimately sent from the filters to the higher-level system, is coordinated by software written for a microprocessor. While it would be possible to simply transfer raw measurement data from the filters to the PC, the microcontroller is itself able to further process the data such that the output data arrives at the PC in a formatted, useful form directly suitable for analysis. Measurements are able to continue indefinitely in this way, allowing for large amounts of detailed measurement data, with little or no post-processing required on the higher-level system, expediting the analysis of measurement data and reducing the chance for user or programming errors on the higher-level system.

Aside from handling measurement data, the microcontroller is also used to control other aspects of the system, such as the relays that control current flow, and the multi-channel digital-to-analog converter used to set bias and transistor voltages. Temperature control of the measured transistor through heating control is a planned for future development, and the capabilities for this were built into the system. Other parts of the microcontroller implementation, such as automated controls that react by automatically adjusting the current measurement range, maintain the highest resolution possible, while also protecting the system from overload damage.

A secondary goal of this work was to show that measurement data itself is in fact valid and useful for measurements. A discussion of system limitations and probable error is discussed in chapter 4. Through knowledge of this error, the precision of the system can be increased through calibration techniques developed as part of this work, discussed in chapter 2 and implemented, as described in chapter 3. The offset and gain error that is present can be compensated for in order to increase system precision and accuracy. The outlook for this measurement system is good as the measurement system for the purpose of transistor-based research appears to be a valid.

6.2 Recommendations

Here, suggestions for future development are given, including new features and improvements upon existing features in the system.

Matlab / Labview / Other Interfaces

The full potential of software for the higher-level system has yet to be fully developed. As a next step, a user interface that is capable of error and plausibility checking of input configuration values would be desirable. Automated testing of a range of configurations is especially interesting.

Temperature measurements

While the FPGA and ADC are configured to make and filter ADC inputs values intended for temperature measurement, there are no functions written on the MCU to handle temperature data or to output it over the interface. Implementation involves reading the measured temperature data during streaming output, calculating the temperature based on the temperature sending hardware being utilized and outputting the temperature data along side current measurements (as an addition CSV field).

Heating control PID regulator

A PID regulation routine has not yet been implemented on the MCU and should be implemented in order to control the heating elements that are to be connected to the system. As a suggestion for implementation, the heating control would be configured as a timer interrupt with highest priority. This higher priority would ensure that temperature regulation always occurs (except in the case of a software fault) for safety purposes.

Switched biasing window delay compensation

A technique to compensate for the transient delay between the switched biasing control signal and the actual current measurement voltage appearing at the ADC input would allow the current measurement system to sample closer to the falling edge of the A filtering window. This is especially useful in the case of high switched biasing speeds, where the transient delay (rising/falling) of the current measurement signal plays a more significant role.

Implementation involves addition of a configurable delay in the FPGA ADC CH process, which introduces a delay between the *last_ state_ i* register which stores the last state of the switching signal during a ADC *conversion start* and the switching biasing control signal, *swb*.

An alternative solution to this problem is an automatic determination of a stable current measurement signal. Implementation of such a function might be to begin sampling at successive points along the sampling window, in order to determine when the current measurement signal is stable for any particular sampling window configuration.

Unused ADC channels

Currently, the FPGA ADC READ process samples all 6 channels when a conversation start signal is received. In the first revision of the printed circuit board, these 4 ADC channels were divided between the three ADC sampling groups A,B, and C; making impossible to read just four channels of the ADC. Since the second revision of the hardware, these 4 ADC channels have been move to ADC sampling groups A and B, making it possible to read just four of the six channels. By taking this into account, it is possible to reduce the number of cycles required to complete a ADC conversion cycle from 4.1μs, to the minimum conversion cycle time (4 μs = 250 KSPS).

Alternatively, other uses for the two unused ADC sampling channels could be found, one possibility is in the automatic recognition of the DAC output level, as a possible verification for the error described in section 6.3.

Command interface standardization

The current user interface, while based on the SCSI standard, is not compliant with the standard. Implementation involves renaming and slight reconfiguration of user interface logic block in the MCU software. User interface commands standardization would aid in the writing of custom module software, for higher-level system packages such as Labview.

MCU flash memory storage usage

The MCU contains its own flash memory which is currently unused. When power is removed from the measurement system, all configuration information is lost. An implementation utilizing this flash memory was not realized due to reported MCU silicon errors on rev. A and rev. B of the the MSP430F5438A that was used in revision 1 measurement system hardware. Revision 2 of the measurement system hardware uses rev. E of the MSP430F5438A, making possible for flash memory to be utilized.

Utilization of secondary USB interface

A secondary, unused USB interface on the printed circuit board interfaces USB directly with the FPGA. Possible uses for this secondary USB interface a might be high-speed data acquisition, possible directly from the filters on FPGA. Implementation involves VHDL coding of a FIFO interface to the MAX3232E RS-232 transceiver.

Lower power modes to reduce system noise

It may be possible to improve the signal-to-noise ratio of measurements with low decimation rates by investigating the effects of reduced drive strength from the MCU, or low/lower power modes in system components [Instruments b]. Ideally, just before and during an ADC conversion cycles, system noise should be minimal. Possible implementations might involve timing power down key devices to match the conversion cycle of the ADC or the data acquisition from FPGA to MCU to USB.

Optimization of CIC filter

The CIC filter used in the implementation is a 1-stage, 1 comb-section delay, CIC filter. A 2- or 3-stage filter with 1 or 2 comb-section delay elements, would result in better low-pass filter response, but at the cost of a higher number of required logic elements in the FPGA. Tests during implementation show that a 2-stage of 3-stage filter would exceed the number of logic elements available in the FPGA. However, bit-trimming techniques, as suggested by [Hogenauer 1981] would reduce the bit-width between the stages of the filter, reducing overall logic element requirements, allowing for a higher stage filter in the FPGA implementation.

6.3 Known Issues

In this section, known problems in hardware and software are summarized. In the event of further development of either th hardware or software components of the system, these points should be observed.

5438A silicon bug and workaround

In testing of the DAC SPI interface, a possible silicon bug in revision A of the MSP430F5438A was found. The problem is related to the shift register of the SPI interface when interfacing the DAC8718. This problems results in the least-significant bit in the most-significant bit position, with all other bits by one position. A workaround was implemented by simply shifting the bits back into their proper position.

Board power supply and digital-to-analog converter power-up mode

The digital-to-analog converter DAC8718 support two input modes which program its analog output, two's complement and straight binary. The mode is selected based on the power-up states of configuration pins, however the delayed nature of some of the power supply components on the system board, result in unexpected input mode configurations after power-up that result in an incorrect analog output signal.

The workaround for this problem is implemented in software. The solution is to add code on the microcontroller during its initialization of the DAC8718. The software performs the following steps to ensure the correct mode is configured:

1. wait 100 ms (delay)

2. Set power-down mode (write 0x1800 to DAC CONFIG register)

3. Wait 3 seconds to ensure all power supplies are stable

4. Set a software of the DAC8717 by writing 0xA000 to DAC CONFIG register (0x30)

Appendix A

FPGA Information

Here tables relavant to the programming as described in chapter 3 are given. Tables A.1,A.2,A.3, andA.4; list memory addresses and any associated signals used by the FPGA internall or by the MCU (e.g. for mearsument data retireval) are given. Tables A.6, A.7, A.10, and A.11 list internal signals used by the named component. Table A.7 is a listing of the timing used by the component ADC READ for timing control and read-out of the ADC.

Tab. A.5: FET_ROOT I/O signals

Name	**Direction**	**Bits**	**Purpose**
areset	in	1	13.65
clk	in	1	Clock signal, 20 MHz
areset	in	1	Asynchronous reset from a switch
clk	in	1	Clock signal
adc_rd	out	1	ADC read
adc_convst	out	3	ADC conversion start
adc_cs	out	1	ADC chip select
adc_reset	out	1	ADC reset signal
adc_stby	out	1	ADC standy-by (low-power state) control
adc_wb	out	1	ADC word/byte select
adc_ser_par_sel	out	1	ADC selection of serial or parallel mode
adc_hs	out	1	ADC hardware/software control mode
adc_wr_refen_dis	out	1	ADC internal/external reference select
adc_busy	in	1	ADC conversion busy signal
adc_data	in	16	16-bit ADC data bus
sw_a	out	1	Switch biasing control signal for FET Port 1
sw_b	out	1	Switch biasing control signal for FET Port 2
range_ch1	out	1	Range control for FET Port 1

Continued on next page

Tab. A.5 – continued from previous page

Name	**Direction**	**Bits**	**Purpose**
range_ch2	out	1	Range control for FET Port 2
mcu_addr	in	8	MCU Address bus
mcu_data	inout	16	MCU Data bus
mcu_oe	in	1	MCU OE output enable (control of data bus direction)
mcu_stream_dvo	out	1	singal MCU when triggered filter data is ready
mcu_stream_busy	in	1	Signal from MCU when data retrieval is in progress
mcu_stream_warn	out	1	stream warning signal, set when incoming trigger data while MCU stream_busy is set
mcu_req_mosi	in	1	MCU MOSI operation handshaking signal
mcu_req_mosi_ack	out	1	MCU MOSI operation handshaking signal
mcu_req_miso	in	1	MCU MISO operation handshaking signal
mcu_req_miso_ack	out	1	MCU MISO operation handshaking signal

Tab. A.1: FPGA assigned memory locations

DEC	HEX	Bits	Signal Name
0	0	16	tperiod_a
1	1	16	tperiod_a
2	2	16	twidth_a
3	3	16	twidth_a
4	4	16	tperiod_b
5	5	16	tperiod_b
6	6	16	twidth_b
7	7	16	twidth_b
8	8	16	phaseA_a
9	9	16	phaseA_a
10	A	16	phaseB_a
11	B	16	phaseB_a
12	C	16	phaseC_a
13	D	16	phaseC_a
14	E	16	phaseD_a
15	F	16	phaseD_a
16	10	16	phaseA_b
17	11	16	phaseA_b
18	12	16	phaseB_b
19	13	16	phaseB_b
20	14	16	phaseC_b
21	15	16	phaseC_b
22	16	16	phaseD_b
23	17	16	phaseD_b
24	18	16	adc_samp_delay
25	19	16	adc_samp_delay
26	1A	16	

Tab. A.2: FPGA assigned memory locations (continued)

DEC	HEX		Bits	Signal Name
27	1B		16	ch_trig
		bit0	1	CH1_a
		bit1	1	CH1_b
		bit2	1	CH1_c
		bit3	1	CH1_t
		bit4	1	CH2_a
		bit5	1	CH2_b
		bit6	1	CH2_c
		bit7	1	CH2_t
		bit8	1	
		bit9	1	
		bit10	1	
		bit11	1	
		bit12	1	
		bit13	1	
		bit14	1	CH1 / CH2 select
		bit15	1	CH1 / CH2 select
28	1C		16	ADC_SETUP / CONFIG
		bit0	1	adc_cs
		bit1	1	adc_stby
		bit2	1	adc_range
		bit3	1	adc_wb
		bit4	1	adc_ser_par_sel
		bit5	1	adc_hs
		bit6	1	adc_wr_refen_dis
		bit7	1	Stream Values RAM Always Update
		bit8	1	
		bit9	1	
		bit10	1	
		bit11	1	
		bit12	1	
		bit13	1	
		bit14	1	
		bit15	1	

Tab. A.3: FPGA assigned memory locations (continued)

DEC	HEX		Bits	Signal Name
29	1D		16	SW Reset, Range and Heating Control
		bit0		SW_Reset
		bit1		MB_UMSA
		bit2		MB_UMSB
		bit3		HEIZA PWM
		bit4		HEIZA PWM
		bit5		HEIZB PWM
		bit6		HEIZB PWM
		bit7		
		bit8		
		bit9		
		bit10		
		bit11		
		bit12		
		bit13		
		bit14		
		bit15		
30	1E		16	
31	1F		16	
32	20		16	ch1_a
33	21		16	ch1_a
34	22		16	ch1_b
35	23		16	ch1_b
36	24		16	ch1_c
37	25		16	ch1_c
38	26		16	ch1_t
39	27		16	ch1_t
40	28		16	ch2_a
41	29		16	ch2_a
42	2A		16	ch2_b
43	2B		16	ch2_b
44	2C		16	ch2_c
45	2D		16	ch2_c
46	2E		16	ch2_t
47	2F		16	ch2_t

Tab. A.4: FPGA assigned memory locations (continued)

DEC	HEX	Bits	Signal Name
48	30	16	fet1_a_count
49	31	16	fet1_b_count
50	32	16	fet1_c_count
51	33	16	fet1_t_count
52	34	16	fet2_a_count
53	35	16	fet2_b_count
54	36	16	fet2_c_count
55	37	16	fet2_t_count
56	38	16	ch1_data_period
57	39	16	ch1_data_period
58	3A	16	ch2_data_period
59	3B	16	ch2_data_period
60	3C	16	
61	3D	16	
62	3E	16	
63	3F	16	

Tab. A.6: Module MCU_IO internal signals

Signal	Bits	Purpose
mcu_req_mosi_ack_i	1	Registering of external signal mcu_req_mosi_ack
mcu_req_miso_ack_i	1	Registering of external signal mcu_req_miso_ack
mcu_data_i	16	Registering of external signal mcu_data
mcu_stream_dvo_i	16	Registering of external signal mcu_stream_dvo
mcu_stream_warn_i	16	Registering of external signal mcu_stream_warn
speed_count_i	16	Count of "lost" measurements
mcu_stream_dvo_d1	1	1. chained delay signal for DVO signal
mcu_stream_dvo_d2	1	2. chained delay signal for DVO signal
RAM16	64x16	Array of 16-bit memory locations

Tab. A.7: Component ADC_READ internal signals

Signal	**Bits**	**Purpose**
count	32	Counter of CLK rising edges for timing
count_delay	32	Count of CLK rising edges for timing of sample delay
en_i	1	Internal enable, activated by en input, stays active until of sampling cycle
v1_i	16	Holds ADC V1 value for complete sampling cycle
v2_i	16	Holds ADC V2 value for complete sampling cycle
v3_i	16	Holds ADC V3 value for complete sampling cycle
v4_i	16	Holds ADC V4 value for complete sampling cycle
v5_i	16	Holds ADC V5 value for complete sampling cycle
v6_i	16	Holds ADC V6 value for complete sampling cycle
adc_cs_i	1	Internal signal for external output signal adc_cs
adc_reset_i	1	Internal signal for external output signal adc_reset
adc_rd_i	1	Internal signal for external output signal adc_rd
adc_convst_i	3	Internal signal for external output signal adc_convst
adc_stby_i	1	Internal signal for external output signal adc_stby
adc_range_i	1	Internal signal for external output signal adc_range
adc_wb_i	1	Internal signal for external output signal adc_wb
adc_ser_par_i	1	Internal signal for external output signal adc_ser_par
adc_hs_i	1	Internal signal for external output signal adc_hs
adc_wr_refen_dis_i	1	Internal signal for external output signal adc_wr_refen_dis

Tab. A.9: Comonent ADC READ timing

Time [ns]	Count	adc_read	adc_convst	adc_reset	adc_cs	adc_stby	dvo	output
0	0		1					
50	1							
100	2							
150	3							
3000	60							
3050	61							
3100	62	0	1		0			
3150	63	0	1		0			v1
3200	64		1		0			
3250	65	0	1		0			
3300	66	0	1		0			v2
3350	67		1		0			
3400	68	0	1		0			
3450	69	0	1		0			v3
3500	70		1		0			
3550	71	0	1		0			
3600	72	0	1		0			v4
3650	73		1		0			
3700	74	0	1		0			
3750	75	0	1		0			v5
3800	76		1		0			
3850	77	0	1		0			
3900	78	0	1		0			v6
3950	79		1		1		1	
4000	80		0		1			
4050	81		0					

Tab. A.10: Component ADC_CH internal signals

Signal	Bits	Purpose
sw_cnt	32	Rising edge of CLK counter
data_cnt	32	Rising edge of CLK counter
val_h_i	16	"high" switched biasing value, to A-Filter
val_l_i	16	"low" switched biasing value, to B-Filter
adc_en_i	1	ADC enable signal
swb_i	1	Switched biasing control signal
last_state_i	1	Last state of switched biasing signal (during sampling)
swb_delay_i	1	Used to add delay to swb_i

Tab. A.11: Component CALC_CH internal signals

Signal	Bits	Purpose
val_a_i	32	Incoming A-filter measurements
val_b_i	32	Incoming B-filter measurements
val_t_i	32	Incoming T-filter measumrents
ch_dvo_i	1	hold dvo signal for MCU_IO
a_dvo_cur	1	A filter dvo received since last dec signal
b_dvo_cur	1	B filter dvo received since last dec signal
t_dvo_cur	1	T filter dvo received since last dec signal
a_dvo_prev	1	A filter dvo previously received since last dec signal
b_dvo_prev	1	B filter dvo previously received since last dec signal
t_dvo_prev	1	T filter dvo previously received since last dec signal

Tab. A.12: Component CIC I/O signals

Signal	Dir.	Purpose
clk	in	FPGA system clock
in_async	in	Asynchronous reset signal
sw_reset	in	Software reset signal from MCU_IO
out_sync	out	FPGA clock synchronous reset

Appendix B

Microcontroller

Here, user interface are listed and described in tables B.1, B.3,B.4,A.1,B.6, and B.7. In tables B.8 and B.9 the interface pin assignment is given.

Tab. B.1: CMD functions

Command	Use
cmd:sys:id	Returns the same serial number assigned to the USB port
cmd:sys:name	Returns the name of the system. Intended for use by higher-level systems in combination with the serial number to verify that the connected system is the intended system (e.g. board revision).
cmd:sys:reset	Reset the ADC, DAC, and FPGA, reinitialize all values to default.
cmd:sys:fpgasafe	During programming and reconfiguration of FPGA ports, set MCU ports to a safe state that will no result in excessive current flow.
cmd:sys:fpgainit	Reset the FPGA, reset to default values. Also resets the ADC.
cmd:mode:local	Set the system mode to LOCAL. Intended for manual reconfiguration, all typed characters are echoed back to the Serial-USB interface.
cmd:mode:remote	Same as system mode LOCAL, except typed characters are not echoed back to the terminal. Intended for use by higher-level system (e.g. MATLAB scripts). Reduces Serial-USB traffic.
cmd:mode:stream	Included for debugging only, sets system to STREAM mode. This is normal set and unset automatically.
cmd:mode:cal	Included for debugging only, sets system to CAL mode. This is normally set and unset automatically.

Tab. B.2: MEAS functions

Command	Use
meas:fet:a:n	Make n measurements on port 1. n=0 : non-stop measurements
meas:fet:b:n	Make n measurements on port 2. n=0 : non-stop measurements
meas:fet:ab:n	Make n measurements on Ports 1 & 2. n=0 : non-stop measurements
meas:stop:	Stop Measurement (alternate command)

Tab. B.3: SET functions

set	:	sys	:	format	:	n	Set output format (raw ADC values / current values)
set	:	adc	:	sdelay	:	n	Set ADC delay between sampling
set	:	swA	:	period	:	n	FET_A: Clock cycles of one period
set	:	swA	:	width	:	n	FET_A: Clock cycles of high switchd biasing signal
set	:	swB	:	period	:	n	FET_B: Clock cycles of one period
set	:	swB	:	width	:	n	FET_B: Clock cycles of high switchd biasing signal
set	:	DAC	:	0	:	n	Set VDAC0 voltage
set	:	DAC	:	1	:	n	Set VDAC1 voltage
set	:	DAC	:	2	:	n	Set VDAC2 voltage
set	:	DAC	:	3	:	n	Set VDAC3 voltage
set	:	DAC	:	4	:	n	Set VDAC4 voltage
set	:	DAC	:	5	:	n	Set VDAC5 voltage
set	:	DAC	:	6	:	n	Set VDAC6 voltage
set	:	DAC	:	7	:	n	Set VDAC7 voltage
set	:	DAC	:	pd	?		Set DAC Power-down
set	:	DAC	:	pu	?		Set DAC Power-up
set	:	DAC	:	reset	:	en	Enable RESET (pin) of DAC
set	:	DAC	:	reset	:	dis	Disable RESET (pin) of DAC
set	:	DAC	:	reset	:	pulse	Quick RESET pulse of DAC
set	:	DAC	:	clear	:	en	Enable CLEAR (pin) of DAC
set	:	DAC	:	clear	:	dis	Disable CLEAR (pin) of DAC
set	:	DAC	:	clear	:	pulse	Quick CLEAR pulse of DAC
set	:	rangeA	:	mode	:	n	Select between 200 μA and 2 mA range measurments
set	:	rangeB	:	mode	:	n	Select between 200 μA and 2 mA range measurments
set	:	rangeA	:	lowswup	:	n	Set port 1 low switch up current value
set	:	rangeA	:	lowmax	:	n	Set port 1 max low value (ADC value)
set	:	rangeA	:	highswdown	:	n	Set port 1 high switch down
set	:	rangeA	:	highmax	:	n	Set port 1 high maximum value
set	:	rangeA	:	auto			Set port 1 automatic ranging

Tab. B.4: SET functions (continued)

set	:	rangeB	:	lowswup	:	n		Set port 2 low switch up current value
set	:	rangeB	:	lowmax	:	n		Set port 2 high maximum value
set	:	rangeB	:	highswdown	:	n		Set port 2 high switched-down value
set	:	rangeB	:	highmax	:	n		Set port 2 high maximum value
set	:	fetA	:	phaseA	:	n		Set port 1 phase A value
set	:	fetA	:	phaseB	:	n		Set port 1 phase B value
set	:	fetA	:	phaseC	:	n		Set port 1 phase C value
set	:	fetA	:	phaseD	:	n		Set port 1 phase D value
set	:	fetB	:	phaseA	:	n		Set port 2 phase A value
set	:	fetB	:	phaseB	:	n		Set port 2 phase B value
set	:	fetB	:	phaseC	:	n		Set port 2 phase C value
set	:	fetB	:	phaseD	:	n		Set port 2 phase D value

Tab. B.5: MEM functions

mem	:	regset	:	0-63	:	n	Set memory location(8bit) to value(16bit)
mem	:	regget	:	<addr>			Get 16-bit memory value from 8bit address
mem	:	dacget	:	<addr>			Get specified DAC register address
mem	:	dacset	:	<addr>	:	n	Set specified DAC register address to specified value

Tab. B.6: CAL functions

cal	:	set	:	zero	?	Calibrate filter output (sets mode CAL, returns after)
cal	:	disp	:	zero	?	Display zero calibration values
cal	:	clear	:	zero		Clear zero point calibration values

Tab. B.7: TEST functions

test	:	dacloop	:	n	x	Set DACn output to 0V, then 0x0000 to 0xffff, then 0V
test	:	usbloop	:	n		Output memory locations 63–58 as fast as possible

Tab. B.8: MCU-FPGA Inteface

MCU "Master"			FPGA "Slave"		
Pin	Port	Direction	Pin	Direction	Signal
43	P4.0	out	88	in	mcu_addr[0]
44	P4.1	out	87	in	mcu_addr[1]
45	P4.2	out	86	in	mcu_addr[2]
46	P4.3	out	85	in	mcu_addr[3]
47	P4.4	out	84	in	mcu_addr[4]
48	P4.5	out	83	in	mcu_addr[5]
49	P4.6	out	82	in	mcu_addr[6]
50	P4.7	out	81	in	mcu_addr[7]
17	P1.0	in	120	out	mcu_stream_warn
18	P1.1	in	119	out	mcu_stream_dvo
19	P1.2	out	118	in	mcu_oe
20	P1.3	out	117	in	mcu_stream_busy
21	P1.4	out	116	in	mcu_req_mosi
22	P1.5	in	115	out	mcu_req_mosi_ack
23	P1.6	out	114	in	mcu_req_miso
24	P1.7	in	113	out	mcu_req_miso-ack
25	P2.0	IN*/out	106	inout	mcu_data[0]
26	P2.1	IN*/out	105	inout	mcu_data[1]
27	P2.2	IN*/out	104	inout	mcu_data[2]
28	P2.3	IN*/out	101	inout	mcu_data[3]
29	P2.4	IN*/out	100	inout	mcu_data[4]
30	P2.5	IN*/out	99	inout	mcu_data[5]
31	P2.6	IN*/out	98	inout	mcu_data[6]
32	P2.7	IN*/out	95	inout	mcu_data[7]
57	P8.0	IN*/out	68	inout	mcu_data[8]
58	P8.1	IN*/out	67	inout	mcu_data[9]
59	P8.2	IN*/out	66	inout	mcu_data[10]
60	P8.3	IN*/out	65	inout	mcu_data[11]
61	P8.4	IN*/out	64	inout	mcu_data[12]
65	P8.5	IN*/out	63	inout	mcu_data[13]
66	P8.6	IN*/out	62	inout	mcu_data[14]
67	P8.7	IN*/out	61	inout	mcu_data[15]

* denotes default value

Tab. B.9: EEPROM interface

MCU port	Description	Purpose
P11.0	EEDATA	EEPROM data pin
P11.1	EESK	EEPROM data clock
P11.2	EECS	EEPROM chip select

Appendix C

EEPROM Programming

In table C.1, the memory locations useful for reading the USB serial number directly from the EEPROM is given.

Tab. C.1: EEPROM memory locations (for USB) (continued)

	Address				
word		byte	byte	Bit	Description
0	00	0	00		High Current
	00	1	01		
1	01	2	02		Vendor ID
	00	3	03		
2	02	4	04		Product ID
	00	5	05		
3	03	6	06		(0x07 0x06) Chip Type
	00	7	07		
4	04	8	08		Config descriptor
	00			7	Always 1
	00			6	1 if self powered, 0 if bus powered
	00			5	1 if this device uses remote wakeup
	00			4	1 if this device is battery powered
	00			3	
	00			2	
	00			1	
	00			0	
	00	9	09		Max power consumption: max power = value * 2 mA
5	05	10	0A		
	00			7	Bit 7: 0 - reserved
	00			6	Bit 6: 0 - reserved
	00			5	Bit 5: 0 - reserved
	00			4	Bit 4: 1 - Change USB version
	00			3	Bit 3: 1 - Use the serial number string
	00			2	Bit 2: 1 - Enable suspend pull downs for lower power
	00			1	Bit 1: 1 - Out EndPoint is Isochronous
	00			0	Bit 0: 1 - In EndPoint is Isochronous
	00	11	0B		Invert data lines
6	06	12	0C		USB version low byte when 0x0A bit 4 is set
	00	13	0D		USB version high byte when 0x0A bit 4 is set
7	07	14	0E		Offset of the manufacturer string + 0x80
	00	15	0F		Length of manufacturer string
8	08	16	10		Offset of the product string + 0x80
	00	17	11		Length of product string
9	09	18	12		Offset of the serial string + 0x80
	00	19	13		Length of serial string
10	0A	20	14		CBUS function: CBUS0, CBUS1
	00	21	15		CBUS function: CBUS2, CBUS3
11	0B	22	16		CBUS function: CBUS5
	00	23	17		Strings

Tab. C.2: EEPROM memory locations (for USB) (continued)

	Address				
word		byte	byte	Bit	Description
12	0C	24	18		
	00	25	19		
13	0D	26	1A		
	00	27	1B		
14	0E	28	1C		
		29	1D		
		30	1E		
		31	1F		
		32	20		
		33	21		
		34	22		
		35	23		
		36	24		
		37	25		
		38	26		
		39	27		
		40	28		
		41	29		
		42	2A		

List of Figures

List of Tables

Bibliography

[Altera 2007] Altera. *Cyclone FPGA Family Data Sheet*, 2007. [Online; accessed 9-December-2011]. (Cited on pages 24 and 61.)

[Altera 2009] Altera. *USB-Blaster Download Cable User Guide*, 2009. [Online; accessed 9-December-2011]. (Cited on page 27.)

[Consortium 1999] SCPI Consortium. *Standard Commands for Programmable Instruments (SCPI)*, 1999. (Cited on page 48.)

[Devices] Analog Devices. *AD7656/AD7657/AD7658 Data Sheet.* Rev. C. (Cited on pages 67, 86 and 94.)

[Drosg 2009] Manfred Drosg. Dealing with uncertainties: A guide to error analysis. Springer, 2009. (Cited on pages 85 and 95.)

[Fetzer 1965] Dr. Viktor Fetzer. Mathematik für elektrotechniker. Dr. Alfred Hüthig Verlag, 1965. (Cited on pages 17 and 96.)

[Ganz 2010] Dennis Ganz. Umsetzung und evaluierung dynamischer betriebsstrategien für feldeffekttransistorarrays. Diplom thesis, Hochschule Kempten, 04 2010. (Cited on page 12.)

[Hogenauer 1981] E. Hogenauer. *An economical class of digital filters for decimation and interpolation.* Acoustics, Speech and Signal Processing, IEEE Transactions on, vol. 29, no. 2, pages 155 – 162, apr 1981. (Cited on pages 87 and 110.)

[Instruments a] Texas Instruments. *Code Composer Studio (CCStudio) Integrated Development Environment (IDE) v5 Overview.* [Online; accessed 12-December-2011]. (Cited on page 29.)

[Instruments b] Texas Instruments. *MSP430 5xx Family User Guide.* (Cited on pages 33, 34, 35, 37, 88 and 110.)

[Instruments c] Texas Instruments. *Voltage Reference REF5050 Data Sheet.* (Cited on page 88.)

[Instruments 1995] Texas Instruments. *Understanding Data Conversion TI DAC*, 1995. (Cited on pages 21, 85 and 86.)

[Instruments 2005] Texas Instruments. *OPA277/OPA2277/OPA4277 Data Sheet*, Apr 2005. (Cited on page 94.)

[Instruments 2009] Texas Instruments. *DAC8718 Digital-to-Analog Converter Data Sheet*, May 2009. (Cited on pages 43 and 86.)

[Instruments 2011a] Texas Instruments. *MSP430 Part number decoder*, 2011. [Online; accessed 10-January-2011]. (Cited on page 24.)

[Instruments 2011b] Texas Instruments. *MSP430 Programming Via the JTAG Interface*, 2011. [Online; accessed 9-December-2011]. (Cited on page 27.)

[Jacob Millman 1985] Satyabrata Jit Jacob Millman Christos C Halkias. Electronic devices and circuits. McGraw-Hill international book company, 1985. (Cited on pages 16 and 18.)

[Jahromi 2007] Omid Jahromi. Multirate statistical signal processing. Springer, Dordrecht, 2007. (Cited on page 74.)

[Kester 2008] Walt Kester. *The Good, the Bad, and the Ugly Aspects of ADC Input Noise? Is No Noise Good Noise?*, 2008. (Cited on pages 21, 54, 87 and 88.)

[Kolka 2010] Robert Kolka. *Dokumentation über das Konzept für den Entwurf einer elektronischen Schaltung zur Ansteuerung und Auswertung eines FETs*, 2010. (Cited on page 24.)

[Ltd 2011] Future Technology Devices International Ltd. *FT232BL USB UART IC (Lead Free Package) Data Sheet*, 2011. (Cited on page 37.)

[Luecke 2005] Gerald Luecke. Analog and digital circuits for electronic control system applications - using the msp430 microcontroller. Newnes, 2005. (Cited on page 37.)

[Mathworks 2011] Mathworks. *MATLAB product documentation Instrument Control Toolbox*, 2011. (Cited on pages 82 and 135.)

[Mitra 1993] Sanjit Mitra. Handbook for digital signal processing. J. Wiley & Sons, New York, 1993. (Cited on page 50.)

[Peter Prinz 2006] Tony Crawford Peter Prinz. C in a nutshell. O'Reilly, 2006. (Cited on page 46.)

[Ruscak 1995] Steve Ruscak and Larry Singer. *Using Histogram Techniques to Measure A/D Converter Noise*, 1995. (Cited on page 87.)

[Rushton 2011] Andrew Rushton. Vhdl for logic synthesis. Wiley, 2011. (Cited on pages 30 and 80.)

[Wikipedia 2011] Wikipedia. *Joint Test Action Group — Wikipedia, The Free Encyclopedia*, 2011. [Online; accessed 9-December-2011]. (Cited on page 27.)

[Winkelman 2009] Sven Winkelman. Entwicklung von betriebsstrategien für neuartige feldeffekttransistoren zur driftreduzierung. Diplom thesis, Fachhochschule Jena, 10 2009. (Cited on pages 11, 12, 16, 19 and 103.)