Technische Universität Braunschweig

Institut für Mikrotechnik

Final Project

Wireless evaluation electronic system based on FPGA for micro probe

Iosu Lerga Valencia

Number of matriculation 4716459

Technische Universität Braunschweig

Fakultät Maschinenbau

Written in

Institut für Mikrotechnik

Examiner: Prof. Dr. Andreas Dietzel

Supervisor: Dipl.-Ing. David Metz

Handed in on 21. July 2016

Statutory declaration

I hereby declare that I, Iosu Lerga, have created the present work independently and only with the aid of the specified means and sources.

Braunschweig, 21.07.2016

Iosu Lerga

Abstract

In this project it is presented a wireless electronic system based on FPGA to integrate a new designed microprobe in a Coordinate Measuring-Machine. A CMM is an instrument of direct measurement which function is to measure physical geometrical characteristics of an object. They consist of three parts: a motion system with 3 or more axis, a probing system and a controller.

The probing system (microprobe) can be tactile or optical. Through it, it is possible to touch an object and determine the touch points. With the measured points, an object can be geometrically determined.

In order to integrate a developed micro probing system in the CMM, a wireless electronic evaluation system has been developed. This electronic system is based on FPGA and digitalizes the analog signal using an Analog Digital Converter, sending wirelessly the values from the micro probe to the machine.

Therefore, it allows the interfacing of the micro probing system without cables. In addition to this, the electronic system can be controlled and monitored through the computer.

The developed system is able to send the values (selected by sending a command through the computer) from the micro probe to the machine and also to the PC, achieving a frequency of 1 kilohertz. This system consists of the following parts:

- Personal Computer (LabView Program, HTerm and ISE Design Suite).
- Master FPGA (Xess Xula 2).
- Slave FPGA (Xess Xula 2) with AD Converter.
- Digital parallel port (based on shift registers).
- Infrared Modules.

Keywords: Cordinate Measuring-Machine, Micro probe, FPGA interface, Wireless data transmission, AD Converter, Shift register.

Procedure

The process followed in order to achieve the objective has been chosen so as to work step by step and to allow us to optimize and build together all parts of the system. First, it was necessary to develop the LabView program in order to monitor the values of the micro probe. It was used at first to check if the Wheatstone Bridges connected to the AD Converter worked correctly.

Later, the following task was to develop and manufacture the digital parallel port. So as to achieve it, it was necessary to make four circuit boards with three shift register chips (8 bits) each one, giving place to a 4 x 24 bits digital parallel port. This piece of hardware would be needed after to connect the master FPGA to the machine.

After that, the most critical part of the project was the data transmission using the infrared modules. It was necessary to modify much of the code, but finally the reliability of the system has achieved an acceptable rating of errors (more than 99.5% of sendings are correct).

Then, the data management between both FPGA and their peripherals has been developed. The data received by slave FPGA from the AD Converter should be sent through IrDA module to the master FPGA, and then to the CMM through the shift register and to the PC through UART-USB.

In order to develop a functional and smaller prototype, a new printed circuit board was designed to integrate in the same hardware piece the AD Converter and the slave FPGA. Also, the master FPGA was replaced by a smaller one, with its own printed circuit board.

Finally, it has been needed to build all together (introducing it in a metallic box) and optimize the system until it works at any time and condition.

Contents

List of figures

List of tables

Acronyms

IMT	Institut für Mikrotechnik, TU Braunschweig
PTB	Pysikalisch Technische Bundesanstalt
CMM	Coordinate Measuring Machine
VHDL	VHSIC Hardware Description Language
FPGA	Field-Programmable Gate Array
IrDA	Infrared Data Association
RF	Radio Frequency
USB	Universal Serial Bus
PC	Personal Computer
HDMI	High Definition Multimedia Interface
VISA	Virtual Instrument Software Architecture
SPI	Serial Peripheral Interface
LED	Light-Emitting Diode
PPM	Pulse Position Modulation
EM	Electromagnetic
CRC	Cyclic Redundancy Check
UART	Universal Asynchronous Receiver Transmitter

1 Introduction

A Coordinate Measuring-Machine (CMM) is an instrument of direct measurement which function is to measure physical geometrical characteristics of an object. They consist of three parts: a motion system with 3 or more axis, a probing system and a controller.

The probing system (microprobe) can be tactile or optical. Through it, it is possible to touch an object and determine the touch points. With the measured points, an object can be geometrically determined.

In the present project at the IMT, in cooperation with the PTB, a developed microprobe has to be integrated a in a CMM. This CMM, the P40 from the company Klingelnberg (Figure 1.1), is especially designed to measure gears and rotation symmetric workpieces.

Figure 1.1 - CMM P40

The micro probing system is based on a silicon membrane on which a stylus is attached (Figure 1.2a). The probing forces transmitted through the stylus to the silicon membrane have as a result a deformation of the membrane. In the membrane some piezoresistors are diffused in order to measure this deformation.

They are wired together and form four Wheatstone bridges which enable trough a tension measure to determine the deformation of the membrane, and the force applied on the stylus tip. (Figure 1.2b)

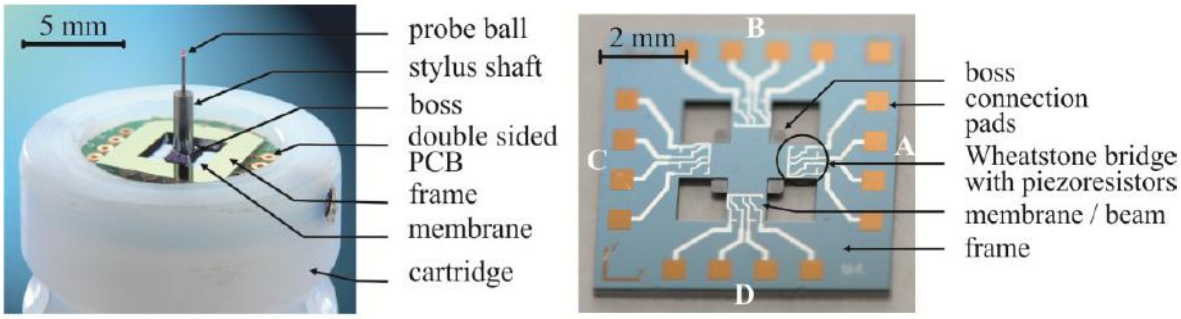


Figure 1.2 – a. The microprobing system and b. Micro probe Weatstone Bridges. [25]

In order to integrate this micro probing system in the P40, a wireless electronic evaluation system has to be developed. This electronic system will be based on FPGA and should digitalize the analog signal using an Analog Digital Converter, and send wirelessly the value from the four bridges (A, B, C, D) to the machine.

Therefore, it allows the interfacing of the micro probing system without cables, which are not possible to bring at the moment in the machine. In addition to this, the electronic system can be controlled and monitored through the computer.

In the following chapters it will be described the work done. At first the objectives and specifications of the system will be listed, showing the possibilities that the system offers. After that, it will be described all electronic devices used in the project in order to achieve a global vision of the technology required, and the reason of choosing them.

In Chapter 5, the working of the system will be displayed through some graphs, focusing on the data flow between all parts of the system. In Chapter 6, the user interface based on LabVIEW will be explained, in order to offer the user a complete manual for using it.

In Chapter 6 and 7 the FPGA programming will be described, both the high level programming and the interfaces required to manage data from/to the hardware modules.

Finally, an assessment of the work made will take place, and also the further improvements that the following version of the system should integrate.

2 Objective and specifications

Before the explanation of the work made, the specifications and objectives of the project will be described.

The main objective is to develop a wireless electronic system based on FPGA Modules for the micro probing system, which have to be integrated in the CMM. This evaluation system has to evaluate four tension values and send them to the CMM. In the mean time, this electronic has to allow monitoring in real time the four channels and be controlled by an external computer with a LabVIEW program.

The Figure 2.1 gives a general view of the system and also how the relationships are between each element. The evaluation system is composed of two FPGA Modules, the master and the slave. The master is used to make the connection between the PC, the CMM and the slave FPGA. The slave is directly connected with the microprobe and it is used to evaluate the four channels thanks to the developed AD converter shield.

The communication with the PC is through a COM port (UART) so that there is no need of special interface on the PC. Between the probing module (slave FPGA and AD Converter) and the machine module (master FPGA, computer and CMM), a wireless communication has to be used, in order to integrate easily the microprobe in the CMM (avoiding cables in the machine). An infrared data transmission system will be used, instead of Bluetooth. It enables to send the four values of the microprobe in less than 100 µs, something impossible with other systems as Bluetooth.

The values are sent from the master FPGA to the CMM through a digital parallel port.

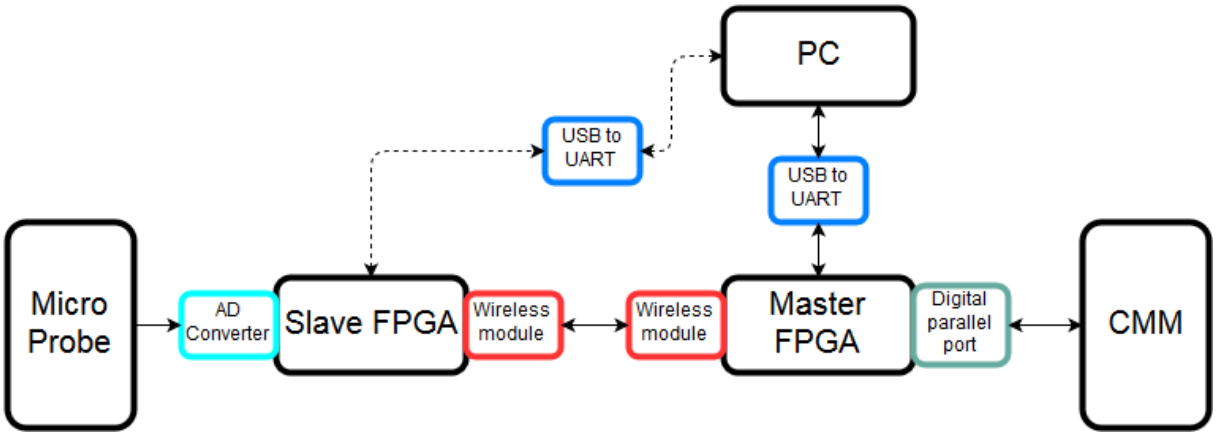


Figure 2.1 - System and relationship between elements

All functions of the electronic system have to be controlled through sending commands from the computer, using a LabVIEW interface. Also, all values have to be monitored on the PC in order to visualize the measurements or to control the behavior of the microprobe.

The system will provide the offset values of the micro probe, in order to make an offset compensation of the four channels (auto-zero). Also, it will be possible to measure the four values periodically or not.

The system has to be to be synchronized with the CMM in order to deliver the values when the CMM requests them through the strobe-signal (1 kHz, signal). Therefore, the system has to deliver the values of the four channels to the CMM every millisecond.

All values for the four channels have to be got through an analog to digital converter. In case of a filtered data, the channels are measured n-times and the mean values are set as the value to send. If not, the values sent are estimated one time. This filter enables to eliminate some noise of the four signals, what can improve the measurement uncertainty of the CMM.

To achieve all these specifications, the previous version of the system has been helpful, because it has been possible to take advantage of it. The FPGA programming interfaces were already done, and also some hardware had been manufactured (AD Converter circuit board, one digital parallel output and IrDA modules).

3 Hardware background

In this chapter it will be described all electronic devices used in the project in order to achieve a global vision of the technology required, and the reason of choosing them.

3.1 Field-Programmable Gate Array Modules

A Field-Programmable Gate Array (FPGA) is an integrated circuit designed to be configured after manufacturing, allowing customers or designers more freedom in their projects. Generally this kind of hardware is specified using a specific computer language, called hardware description language (HDL). The most common HDLs are VHDL and Verilog. The language used in the present project is VHDL and the complier used is ISE Design Suite from Xilinx with ISIM as a simulator tool (it enables to make a simulation of the programming signals before the program is loaded in the FPGA).

There are many tutorials about this programming language, but there are two which deserve a mention: VHDL Tutorial by Peter J. Ashenden [3] and FPGAs!? Now What? by Xess Company [17].

FPGAs are a synchronous circuitry that requires a clock signal, which is generated internally or externally. They contain a big array of programmable logic blocks (also called Configurable Logic Block). The most common FPGA architecture consists, apart from these logic blocks, input/output pads and routing channels. These blocks can be configured and wired as needed. For example, it is possible to wire together the blocks in order to perform complex combinational functions.

The decision of using a FPGA in this project is because of the capability of real simultaneously multitasking, what with a microcontroller not possible is. In this project the four channels have to be digitalized at the same time and a wireless sending has to take place in less than 1 ms.

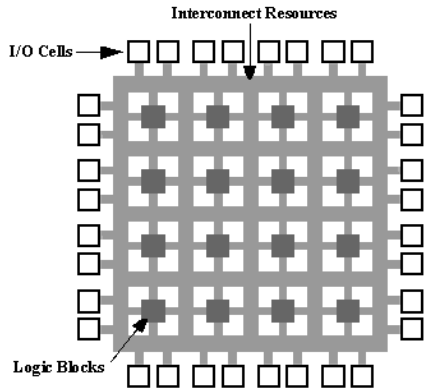


Figure 3.1 - Structure of a FPGA. [21]

In this project two models of FPGA modules have been used:

- **Xess Xula 2**

Xula 2 provides the things needed to develop a project as the present. The FPGA hardware is Spartan 6 from Xilinx. It crams more than 1 million logic gates, 32 megabyte SDRAM, 8 megabit flash memory and also the possibility to insert a microSD memory card. Also, it offers two voltage regulators: 3,3V and 1,2V. The clock can be up to 400 MHz overclocked from a 12 Mhz crystal.

The most important advantage of this FPGA board is its dimensions, because its size is 51 mm x 25 mm and it can be easily integrated in the micro probe. In addition to this, the flash memory can be used to save the configuration of the FPGA after power down. Thanks to the header 32 in/out pins are available.

- **Nexys 4 Artix-7 FPGA**

The Nexys 4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 (FPGA) from Xilinx. The Artix-7 FPGA is optimized for high performance logic, as in the present project. Also, it offers a VGA port, an amount of buttons and some LED which are already wired.

In addition to this, it has several built-in peripherals, like an accelerometer, temperature sensor, digital microphone, speaker amplifier, and some other elements that can be checked in its data sheet [26].

The second one was used at first, because it provides more functions for the debugging such as LEDs or buttons, but its dimensions are too big to this project, so it has been replaced by Xula 2.

3.2 Shift register

In this project the shift registers are used to make the digital parallel port which is used to communicate with the machine. A shift register is basically a cascade of D-flipflops (bistables), which share the same clock signal, and the data output of each flip-flop is wired to the data input of the next one in the chain. As a result, in each rising edge of the clock signal, the data present is shifted in at its input and the last bit in the array is shifted out.

Shift registers are a type of sequential logic that, unlike combinational logic, is not only affected by the present inputs, but also by the prior history (past events). Its working is as follow: the shift register produces a delay of a digital signal in *n* discrete clock times, where ***n*** is the number of the shift register stages (bits). It means that it is necessary to wait n clocks after the data input is in the output.

The most common shift registers are D flip flops or J-K flip flops. However, there is another classification of them [1] that is more useful to understand its working:

1. Serial in – Serial out (used to delay a signal)
2. Parallel in – Serial out (used to convert a parallel input to a serial output)
3. Serial in – Parallel out (Figure 3.2)
4. Universal parallel in – Parallel out (universal shift register, combines the function of the parallel-in, serial-out shift register with the function of the serial-in, parallel-out shift register).

In the present project the shift register used is the third one (Figure 3.2). It is used to send through it serial data and to present it on the parallel output of the shift register.

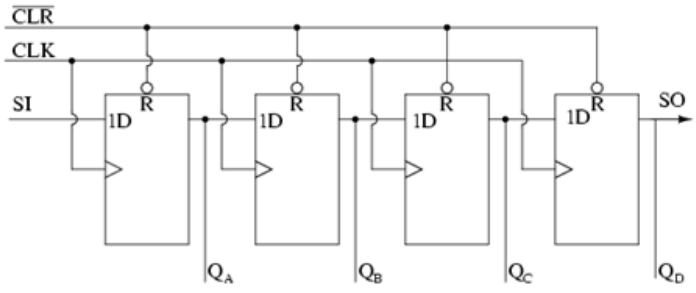


Figure 3.2 - Serial-in, parallel-out shift register (SIPO). [1]

The timing diagram of the serial in, parallel out shift register is shown in Figure 3.3, evidencing that the time to have ready the output depends on the number of bits:

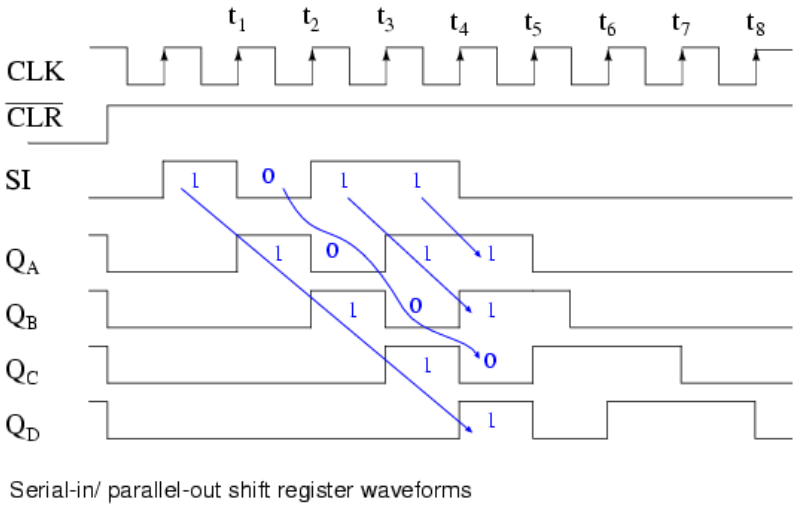


Figure 3.3 - Serial-in parallel-out shift register timing. [1]

The chip used in this project is 74HC595 [12], a 8-bit serial in, serial or parallel output shift register with output latches. After the serial data are loaded in the internal buffer register, and by receipt of the output Latch signal, the state of the buffer register is copied into the output registers and with it the data are presented on the output pins of the chip. This behavior enables to avoid any type of error in the data reading of the machine, because when the data are latched to the output all bits are actualized at the same time.

In order to make a 24 bits digital parallel port, three chips are used and placed in cascade, as it is described in Chapter 4.3.

3.3 Infrared Transceiver

The infrared transmission is an interoperable, low-cost infrared data interconnection standard that supports a walk-up, point-to-point user model that is adaptable to a board range of mobile appliances that need to connect to peripheral devices and hosts. Infrared Data Association (IrDA) is the first actor which standardized infrared protocols.

The transmission is based on emitting/receiving infrared light from a LED to a photodiode. The transmitter module emits light with a modulated signal which depends on the data sent, the receiver module detects the light and regenerate the modulated signal.

Infrared data transmission requires a direct or indirect visual contact between transmitter and receiver. The newest infrared modules are able to transmit data up to 14 Mbps.

This technology is really useful when it is not required to be continually connected to each other. In addition to this, it is required to mention that this communications model

minimizes the development cost significantly. The use of a single LED and photodiode in the transceiver enables an extremely low-cost implementation.

According to The IrDA Standards for High-Speed Infrared Communications [8] there are different IrDA physical layers according to the data rate ranges:

- 2400 to 115.200 bits/s.
- 1.152 Mbits/s.
- 4 Mbits/s.

All links are designed to be used in a line-of-sight. However, they are able to transfer data with a ±15° viewing angle. The first one is designed for easy implementation, and its architecture is the simplest one: it has a host processor (protocol, packet framing and CRC calculation), a UART converter which converts this data into a serial data, and then the encoder/decoder module. Finally, as it is shown in Figure 3.4, we have the transceiver module.

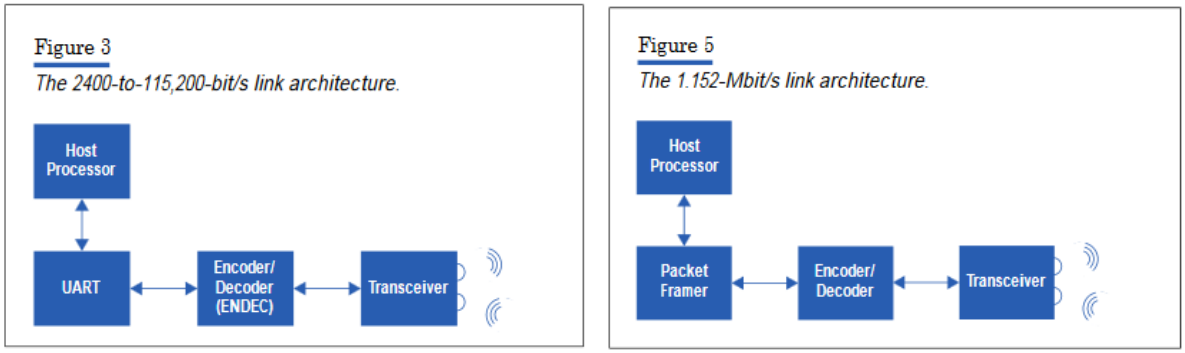


Figure 3.4 - 2400 to 115,200-bit/s and 1.152-Mbit/s link architecture. [8]

The most important difference between this architecture and the required for 1.150 Mbits/s in advance is that packet framing and CRC generation and checking are harder, and therefore it is made in hardware.

However, the physical layer used in the present project is the last one, 4 Mbits/s, in order to achieve a high frequency of data transmission between both FPGA. The chip used in the present project is RPM971-H14 from Rohm, which supports three layers according to its datasheet [4]:

- SIR mode: 9,6k to 115,2 kbps
- MIR mode: 1.152 Mbps
- FIR mode: 4 Mbps

To ensure a high speed transmission, the FIR mode has been chosen for the project. So in this project, the hardware, and not the software, is the responsible of the packet framing and CRC generation/checking. As it is possible to appreciate in Figure 3.5, this link uses a robust packet structure, which is encoded and decoded. Also, in this physical layer the sampling clock from the received signal is a phase-locked loop, which is more robust and safe.

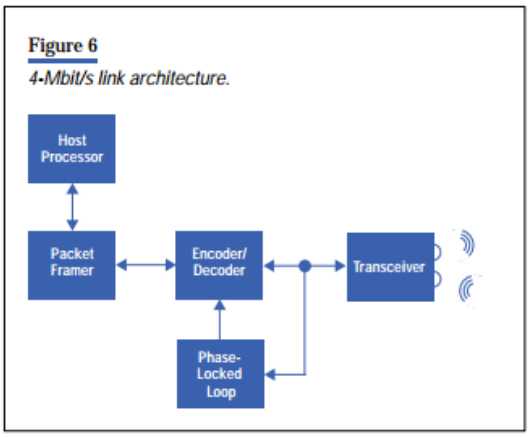


Figure 3.5 - 4-Mbit/s link architecture. [8]

There are different ways to code the packets. The most used are Pulse Width Modulation (PWM) and Pulse Position Modulation (PPM). The first one has a moderate complexity, but it has a worse power efficiency. This is the reason why the coding and packet format chosen is Pulse Position Modulation (PPM).

Its working consists on a varying of the pulse position within a cell (time, in the present project 500 us). As it is possible to realize in Figure 3.6, there are four possible positions of the pulse in the cell (four different messages); thus it is known as 4PPM.

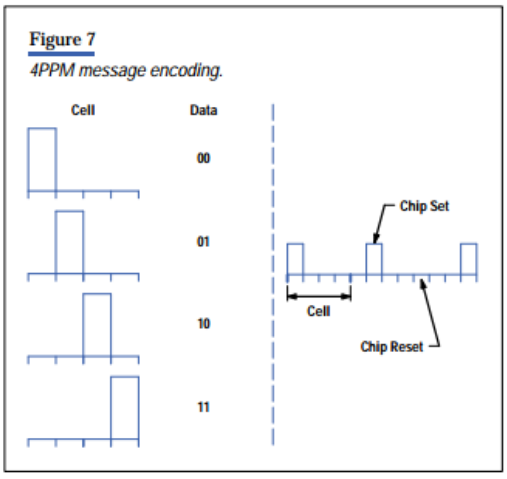


Figure 3.6 - 4PPM message encoding and 4-Mbit/s packet format. [8]

The advantages of using 4PPM are many, but the most important of them: the high speed which is possible to achieve because of the sparseness of the code, the regular timing content, the evasion of interference generated by fluorescent lighting, and also the ability to detect errors.

To conclude with this explanation, it would be important to define the packets sent. As it has been mentioned before, the 4PPM packet has a 32-bit cyclic redundancy check (CRC) field appended to it. Also, there is a preamble, a start and a stop, as it is possible to appreciate in Figure 3.7:

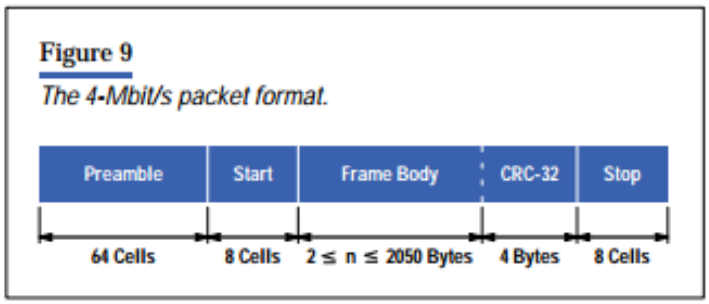


Figure 3.7 - The 4-Mbit/s packet format. [8]

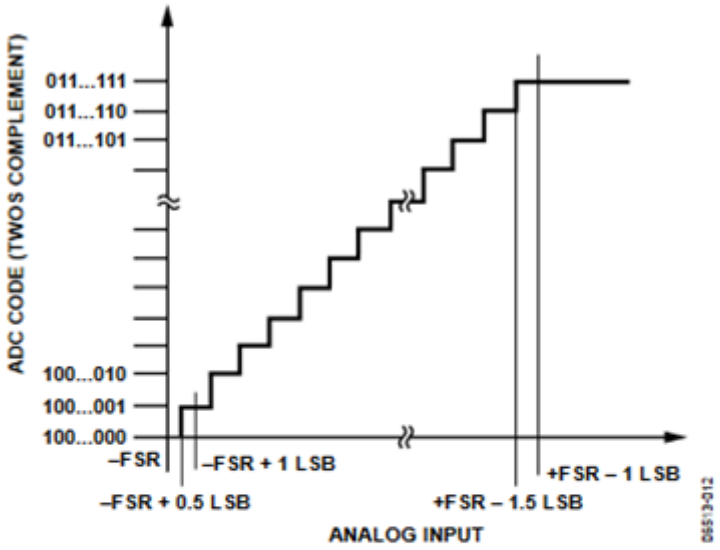
3.4 Analog Digital Converter

An analog to digital converter (ADC) is a device able to convert an analog signal (the four values of the Wheatstone Bridge in this project) into a digital value that represents the amplitude of the signal. Of course, this analog to digital conversion is based in quantization: a process of mapping a large set of input values to a (countable) smaller set. The number of digits allowed is limited, so this process introduces an error which is called quantization error.

The converter is more accurate as more digits are allowed (more discrete values it can produce over the range of analog values), and it is called resolution of the converter. These values are usually generated in binary, so the resolution is expressed in bits.

There are many different ways of implementing an electronic ADC (direct conversion, ramp comparison, pipeline ADC, etc.). The chip used in the present project, AD7982 from Analog Devices [11], is a 18 bit based on successive approximation, that uses a comparator to successively narrow a range that contains the input voltage.

In Figure 3.8 the process of quantization-codification of the chip used in the project is described and also it is possible to appreciate the importance of having a high resolution to have accurate measures.



Description	Analog Input V_{REF} = 5 V	Digital Output Code (Hex)
FSR – 1 LSB	+4.999962 V	0x1FFFF[1]
Midscale + 1 LSB	+38.15 µV	0x00001
Midscale	0 V	0x00000
Midscale – 1 LSB	–38.15 µV	0x3FFFF
–FSR + 1 LSB	–4.999962 V	0x20001
–FSR	–5 V	0x20000[2]

Figure 3.8 - Quantification and codification of AD7982. [11]

At each successive step of Figure 3.8, the converter compares the input voltage to the output of an intern DA Converter which might represent the midpoint of a selected voltage range: At each step in this process, the approximation is stored in a successive approximation register (SAR). For example, if the input is 3V, the first comparison is with 0 V (midpoint of the range +5 V to -5 V). It realizes that is bigger than 0 V, so the next comparison will be with 2,5 V (midpoint of 0 and +5). The steps are continued until the desired resolution is reached.

The data transmission with the FPGA uses a SPI protocol. To start a conversion it is necessary to make an impulse in the CNV pin. After the conversion has done, at least 19 rising edges of the SPI clock signal are required in order to get the 18 bit data conversion on the SDO pin (more details in [11]).

4 Hardware design of the electronic

In this chapter the hardware design will be described with detail: slave FPGA Module, infrared modules, the UART-USB converter, the digital parallel port and also the master FPGA Module.

4.1 Slave FPGA Module

As it has been mentioned before, the slave FPGA module is attached to the micro probe. Therefore, it should be as small as it is possible.

For the Slave FPGA module, the Xula 2 has been used. An additional circuit board (shield) has been developed to extend the FPGA module and to be able to connect it with peripherals. It integrates the AD Converter and a connector for the four channels of the microprobe. Also, it integrates two connectors that can be used to connect the module to the IrDA module and also to the PC. The supply of the module can be provided in a range of 6 up to 12V thanks to the regulators placed on the board.

This FPGA module cannot be connected directly to the electricity, so the supply can be done thanks a battery. It has to be as small as possible, but also has to ensure a duration of at least about 8 hours. The energy consumption was measured, 0,124 A. By using a battery of 1.000 mAh, it is possible to use it during 8h.

At first, a first prototype has been constructed and tested. It was based on two stacked circuit boards. After optimization, it could be possible to reduce it to one board, making the new prototype much smaller. In Figure 4.1 it is possible to appreciate the difference between the both versions. In A the old one is in the right side and the new one in the left side, and in B the old one is behind and the new one in front.

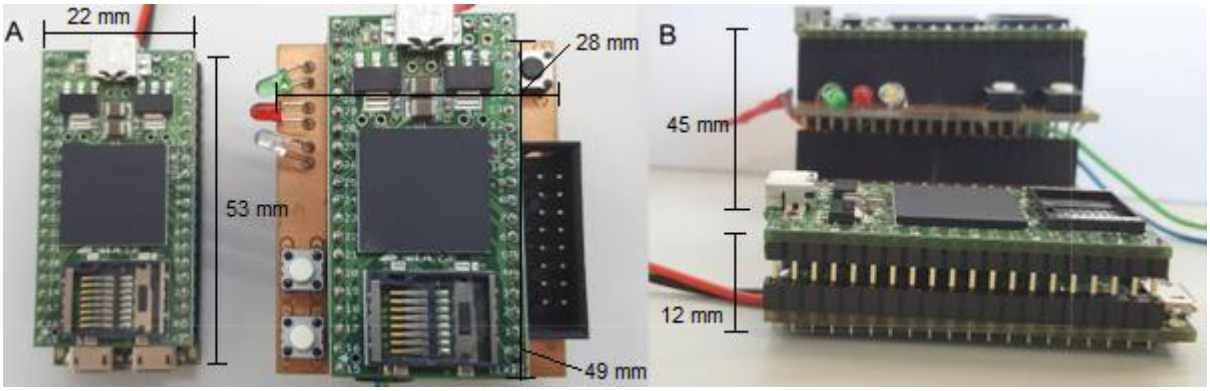


Figure 4.1 - Development of slave FPGA circuit board

The Figure 4.2 shows a view in detail of the shield. It is possible to appreciate that it has two regular USB 2.0 connectors connected to Xula 2 for IrDA module and UART-USB, and one USB 3.0 connector to receive the analog signal from the microprobe.

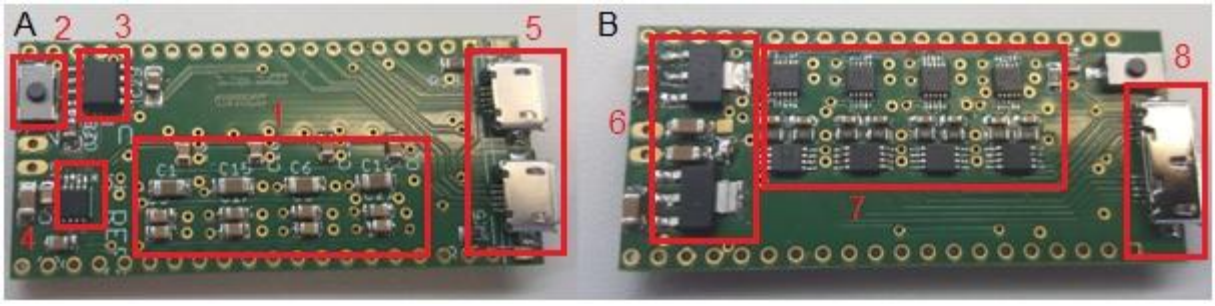


Figure 4.2 - New slave FPGA circuit board with AD Conv. integrated

In the top side (Figure 4.3, A) there are the capacitors needed by the AD converter chips (1), the reset button (2), the two USB 2.0 connectors (5) and the 5V supply for the bridge (3) and the 5V voltage reference for the AD Converter (4). In the bottom side (Figure 4.3, B) there are eight chips, which form the four channels (7), each one with one ADC, a low pass filter and a signal follower. Also, in the left part of the board there are two LDO regulators of 5 and 2.5V (6) and the USB 3.0 connector (8). It has been designed according to the ADC channel structure shown in Figure 4.3:

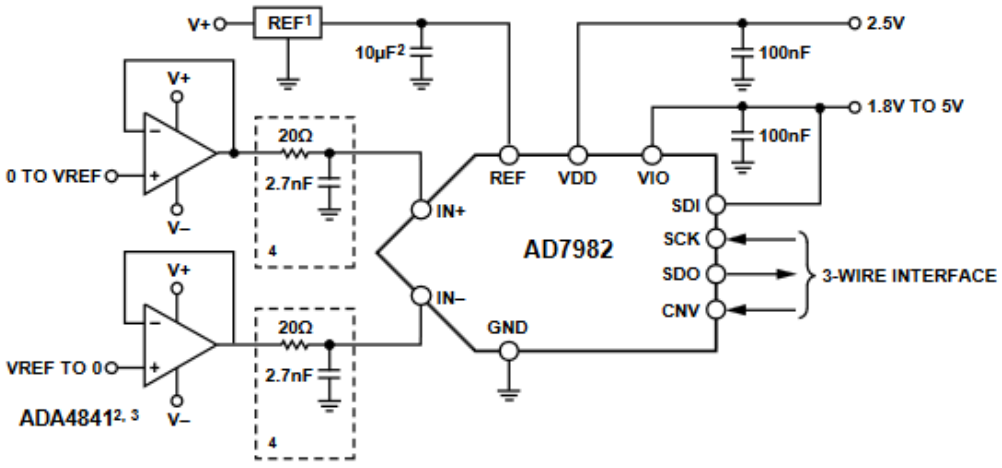


Figure 4.3 - Structure of AD7982. [11]

The connection between the FPGA and the AD Converter shield is done through 40 header pin which are directly connected to the FPGA on the Xula 2. According to circuit board designed on Eagle (see the CD appendix), it was possible set the in/out pins of the FPGA. They are described in the Table 4.1:

Table 4.1: Slave FPGA pin function and use.

Pin	Function	Module	Pin	Function	Module
CH0	-		**CH16**	-	
CH1	-		**CH17**	CONV (3)	ADC Chanel 4
CH2	-		**CH18**	MISO (3)	ADC Chanel 4
CH3	PWDOWN	IrDA	**CH19**	SCLK (3)	ADC Chanel 4
CH4	IrDA TXD	IrDA	**CH20**	CONV (2)	ADC Chanel 3
CH5	IrDA RXD	IrDA	**CH21**	MISO (2)	ADC Chanel 3
CH6	-		**CH22**	SCLK (2)	ADC Chanel 3
CH7	USB TXD	USB-UART	**CH23**	CONV (1)	ADC Chanel 2
CH8	USB RXD	USB-UART	**CH24**	MISO (1)	ADC Chanel 2
CH9	-		**CH25**	SCLK (1)	ADC Chanel 2
CH10	-		**CH26**	CONV (0)	ADC Chanel 1
CH11	-		**CH27**	MISO (0)	ADC Chanel 1
CH12	-		**CH28**	SCLK (0)	ADC Chanel 1
CH13	-		**CH29**	-	
CH14	-		**CH30**	-	
CH15	LED	USER	**CH31**	-	

The USB 2.0/3.0 connectors are used only for an easily connection, but they cannot be connected directly to the USB port of the computer. Therefore, a USB-UART converter is required (explained in subchapter 4.5).

4.2 Master FPGA Module

The master FPGA Module is composed of a Xula 2 and a StickIt Mother Board with three PMOD connector modules which enable an easy connection with the shift register, the wireless module and also the UART-USB module.

To select the correct pins as inputs and outputs, the StickIt data sheet has been used [18]. In Table 4.2 they are listed all in/out pins of the FPGA, with their corresponding hardware module and the PMOD pin associated to them (PMOD number/bit number).

Table 4.2: Master FPGA pin function and use.

Pin	Function	Module	PMOD	Pin	Function	Module	PMOD
CH0	Optional		1/1	**CH16**	-		
CH1	-			**CH17**	IrDA TXD	IrDA	2/2
CH2	Data read	Shift register	1/2	**CH18**	-		
CH3	USB RXD	USB-UART	1/9	**CH19**	IrDA RXD	IrDA	2/3
CH4	Data ready	Shift register	1/4	**CH20**	-		
CH5	USB TXD	USB-UART	1/10	**CH21**	IrDA PWD	IrDA	2/4
CH6	-			**CH22**	LED		1/8
CH7	-			**CH23**	Serial In (1)	Shift register	3/1
CH8	-			**CH24**	Serial In (0)	Shift register	3/7
CH9	-			**CH25**	Serial In (2)	Shift register	3/2
CH10	-			**CH26**	Serial In (3)	Shift register	3/8
CH11	-			**CH27**	Latch clock	Shift register	3/3
CH12	-			**CH28**	Out. Enable	Shift register	3/9
CH13	-			**CH29**	Reset	Shift register	3/4
CH14	-			**CH30**	Clock	Shift register	3/10
CH15	-			**CH31**	-		

This mother board (StickIt) has not any LED to display some information (if it is working or not and also the mode of working). Therefore, a LED (2) has been soldered using the pins of the Raspberry Pi socket (Figure 4.4). In addition to this, a reset button (1) was needed outside the box, so it has been soldered also a second reset button in parallel with the original one. The final result is shown in Figure 4.4:

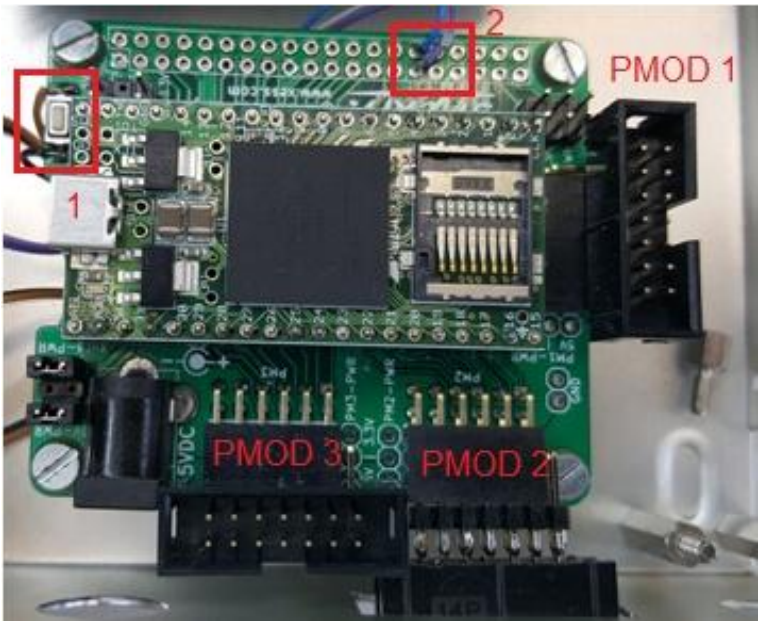


Figure 4.4 - New master FPGA circuit board

4.3 Digital parallel port

The digital parallel 4x24 bits port is required to establish the communication between the master FPGA and the CMM. It is composed on four shift register based boards with a 24 bits parallel output. Through this port it can be transmitted the signals from the CMM to the FPGA and data from the FPGA to the CMM.

The working of each shift register board can be understood looking at Figure 4.5. Each circuit board contains three 74HC595 chips [12]. The first chip receives the data from Serial in, and also the clock/reset signal, latch clock signal, output enable signal, supply and ground. All signals except the serial input are shared by all chips. They are connected in cascade: the serial output of the first one is used to send the data to the following shift register chip, and the parallel output of all of them are wired directly to the suitable connector.

There are jumper which allows selecting the serial data line 1, 2, 3 or 4 depending on which board the data have to be sent through the DSUB-25 connector (max. 24 bits).

The 24th bit of the connector can be directly connected to the FPGA in order to share signals like the strobe signal.

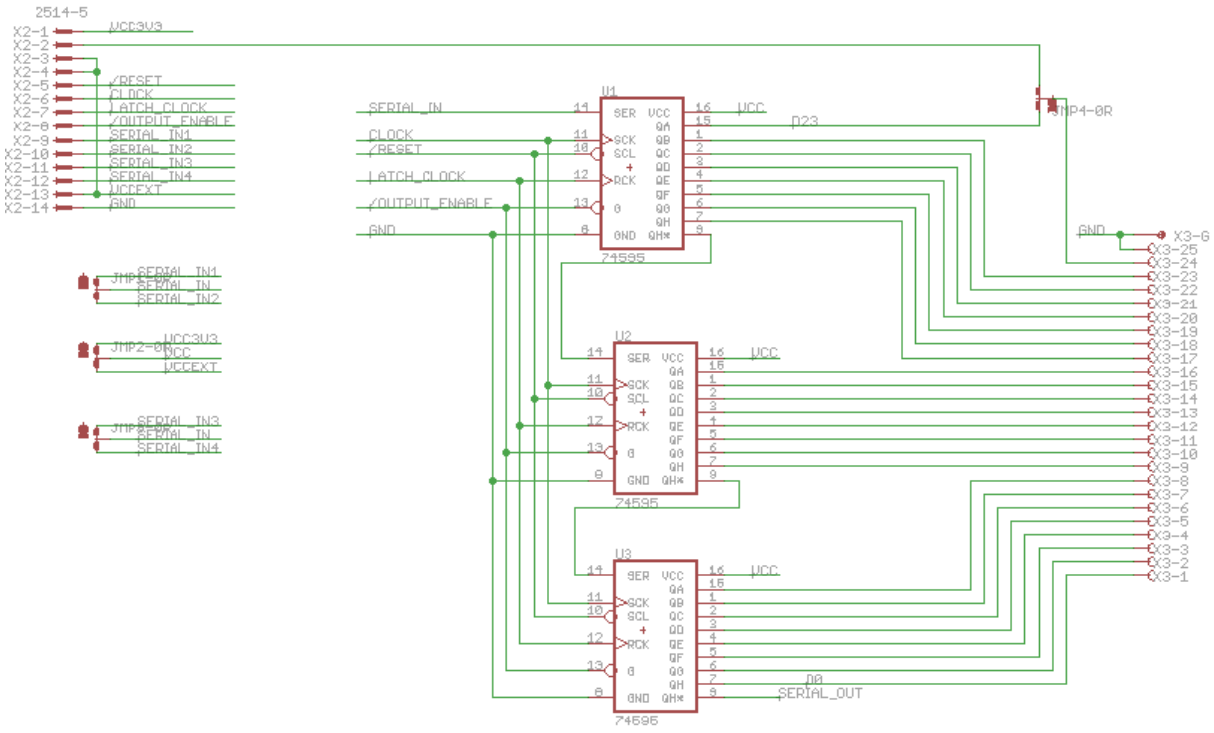


Figure 4.5 - Shift register connections

Looking at Table 4.3 it is possible to realize what the input/output 24 bit of each shift register means. The last one is not used, but it is prepared for using it when needed.

Table 4.3: Shift register pin function of the DSUB-25 connector.

Bit number	Bit function Module 1 (down)	Bit function Module 2	Bit function Module 3	Bit function Module 4 (up)
1	LSB (Data)	LSB (Data)	LSB (Data)	LSB (Data)
2-17 bits	Data	Data	Data	Data
18	MSB (Data, used)	MSB (Data, used)	MSB (Data, used)	MSB (Data, used)
19-23 bits	Not used	Not used	Not used	Not used
24	Strobe input	Data ready output	Data read input	Optional info
25	GND	GND	GND	GND

The first prototype of the shift register was made attaching the three circuit boards in a single piece, achieving a piece of hardware more functional and visual. The second prototype is based on four circuit board attached in the same way as the first one. It was developed using smaller printed industrial circuit boards, in order to achieve the same functions but in less space. In this case, four shift register boards were manufactured in order to have the possibility of sending up to 4 x 24 bits.

The difference between both versions is shown in Figure 4.6.

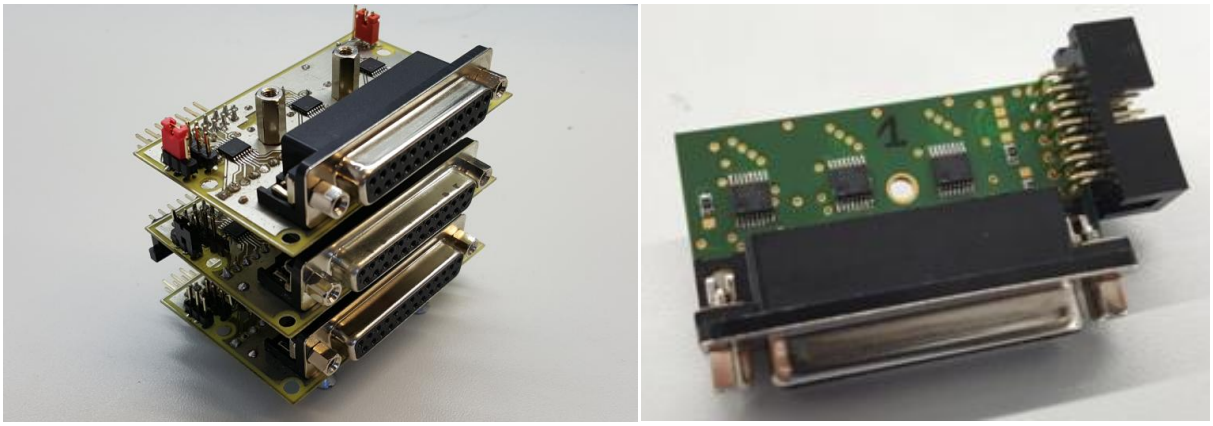


Figure 4.6 - First (left) and second (right) prototypes of the shift register.

4.4 IrDA module circuit board

According to the requirements of the project, the IrDA module should be placed in a small site, so it was required to manufacture a small circuit board. The chip used is RPM971-H14 from Rohm, and it has been wired following its data sheet [4]. The hardware used has the structure shown in Figure 4.7:

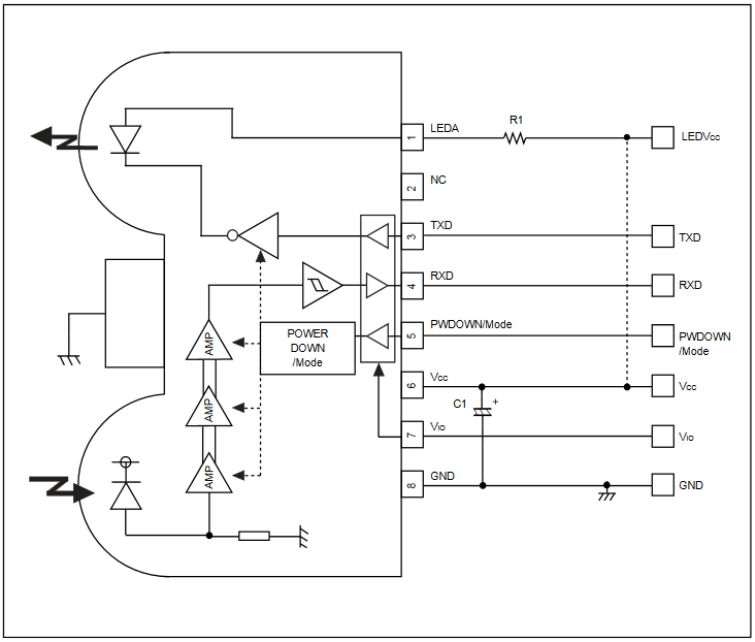


Figure 4.7 - Infrared module [4]

The circuit board design has some pads to wire directly the outputs using a cable. Also, as it is possible to appreciate in Figure 4.8, it has two holes in order to be attached to its holder in the machine. Two long cables were manufactured in order to achieve enough freedom to place them where required.

Figure 4.8 shows the hardware design and also the cable function according to its color:

Red	VCC
Green	RX Data
Blue	PWDOWN (mode)
White	GND
Yellow	TX data

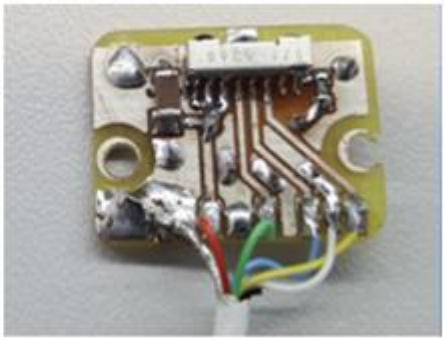


Figure 4.8 - New IrDA module

4.5 USB to UART converter

As it has been mentioned before, Xula 2 needs a converter from UART to USB to receive and transmit data. This is the reason because it has been developed a USB to UART converter. It allows communicating with the FPGA through a COM port.

The chip used is the commercial one FT232RL, from FTDI. According to its data sheet [20], it is able to convert USB to asynchronous serial data and UART to USB, which allows operating between 300 baud to 3 megabaud.

Therefore, some circuit boards were designed and manufactured, including some LED (transmitting and receiving indicators). One of them is shown in the Figure 4.9, and the order of the pins is (from up to down): VCC, GND, TX Data, RX Data, RTS and CTS (last two are not used in the project).

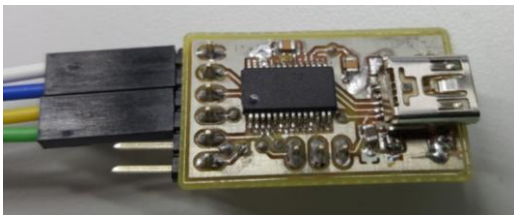


Figure 4.9 - USB to UART converter

4.6 Final design

The slave FPGA, as mentioned before, will be placed in the micro probe with its battery using a specific holder.

The master FPGA and the parallel digital port were introduced in a metallic box (Figure 4.10). The outputs of the box are the four connectors of the digital parallel port (1) and the outputs of the FPGA: a LED (3) to indicate how it is working, a basket to connect the cable of the IrDA module (5), a reset button (2), a USB connector (6) to communicate with the computer and also a hole to introduce the supply cable (4).

Figure 4.10 - Metallic box with master FPGA and shift register

5 General working of the system

In this chapter it will be displayed a general view of the system working. As it has been mentioned before, there are three different modes of working for the master FPGA. The most important differences are related to the data request source (PC or CMM). In mode 0 the request of data comes from the PC, but in modes 1 and 2 it comes from the CMM (strobe signal of 1 kHz).

Mode 0: The master FPGA asks for the value of the Bridge when it is required by a USB command, then it receives them and the data is sent to the CMM and also to the computer. This command can be also periodic: After a sending of a periodic command through the PC, the master FPGA asks for data one time and receive data periodically with a frequency of 1 kHz from the slave. However, the sending of data to the machine is independent of the strobe.

Mode 1: The CMM request the data of the bridge through the strobe signal, with a frequency of 1 kHz. The Master FPGA asks for data by a rising edge of the strobe and when they are received, they are sent to the CMM through the digital parallel port and to the PC through UART-USB.

Mode 2: The CMM request the data of the bridge through the strobe signal, with a frequency of 1 kHz. The Master FPGA receives constantly data of the bridge with an updated frequency of 2 kHz from the Slave. By a rising edge of the strobe the last updated data are sent to the CMM through the digital parallel port and to the PC through USB-UART.

All commands work in mode 0 (default mode), but in mode 1 only *0x30* and *0x31* can be used. In mode 2, only *0xD1* and *0xD0* can be used. The commands 0x30 and *0xD0* are asking for filtered data and *0x31* and *0xD1* are asking without filter (more details in Chapter 8.4).

In the following lines some typical modes of working are shown with detail and explained. In the Appendix A it is possible to find the workings modes for all commands. To explain the mode of working there is a signal wave graphic obtained with a digital oscilloscope by measuring the real signal during the function of the system used. The meaning of the channels is listed in each figure.

- MODE 0

In Figure 5.1 it is possible to appreciate the working in mode 0 by a normal/nonperiodic command. First, the master receives a command from USB (1) (as example 0x30, request of filtered data) and sends the acknowledgement (2). Then it sends through IrDA the corresponding command (3) to request filtered data to the slave. After that, the slave sends the acknowledgement (4) corresponding to the command from the master at the same time it starts the acquisition of the data (5) (in the figure only one channel of the four is displayed). When the data are available, they are sent through IrDA to the master FPGA (6), and this one sends the acknowledgement (7) to confirm the receipt.

Finally, this value is sent through the parallel port (8) and also through USB (9). In the figure it is possible to see that when the serial input is done, the latch clock make an impulse and of course the output enable is high until the transmission finishes.

Figure 5.1 - General working in mode 0. Command 30

In Figure 5.2 it is possible to appreciate the working in mode 0 by a normal/nonperiodic command. As it has been mentioned before, in this mode of working only the commands *0xD1* and *0xD0* (periodic commands) can be used. The only difference is that after receiving the USB command, the master sends through IrDA the corresponding command (steps 1-4) and the slave evaluates and sends the bridge values through IrDA (steps 5-6) with a frequency of 1 kHz. After receiving and acknowledging the data, the master

FPGA sends them to the CMM through the parallel port and also to the PC through USB (steps 7-9, Figure 5.2).

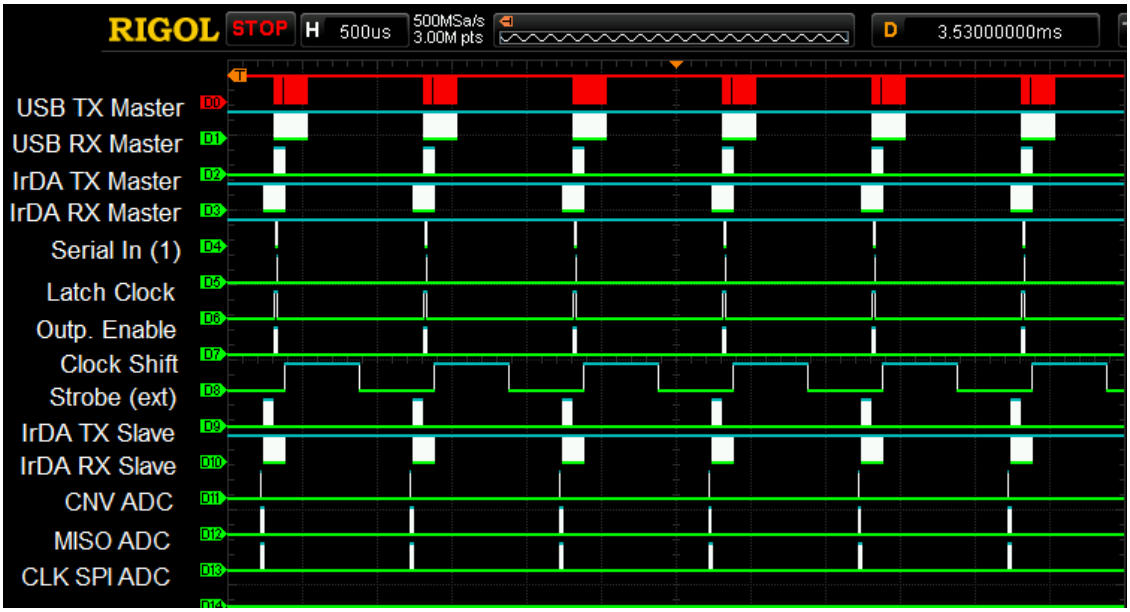


Figure 5.2 - General working in mode 0. Command D1

- MODE 1

In this mode the receiving of data has to be enabled trough the sending of the command (0x30 or 0x31) from USB (1), which will be acknowledged (2). By a rising edge of the strobe the data request is sent through IrDA from the master to the slave (3) and the slave acknowledges it (4). Then it takes place the reading of the bridge values of the AD channel (5) and it is sent trough IrDA (6) (Figure 5.3).

After receiving on the master this, the data are sent every time through to the CMM through the parallel port (7) with the same working as explained before, and through USB each ten times (8). In Figure 5.4 the process is shown in detail and it is possible to see, the dependence of the strobe signal.

Figure 5.3 - General working in mode 1. Command 30

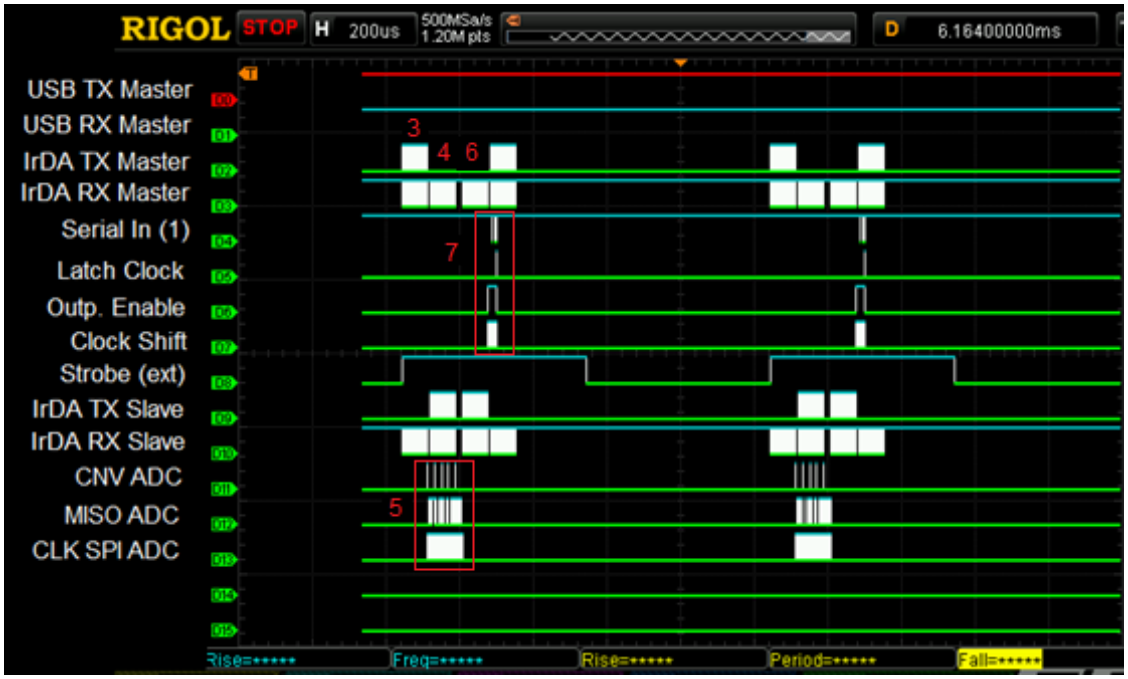


Figure 5.4 - General working in mode 1 with detail. Command 30

- MODE 2

In the mode 2 the periodic sending from the bridge values with a frequency of 2 kHz is activated trough a USB command (*0xD1* and *0xD0).* The main difference in relation to mode 1 is that the data are sent from the slave automatically and are transferred to the CMM after

the strobe rising edge. The last data updated is sent to the CMM (2) (one time each ten, also through USB), as it is shown in Figure 5.6 and Figure 5.5.

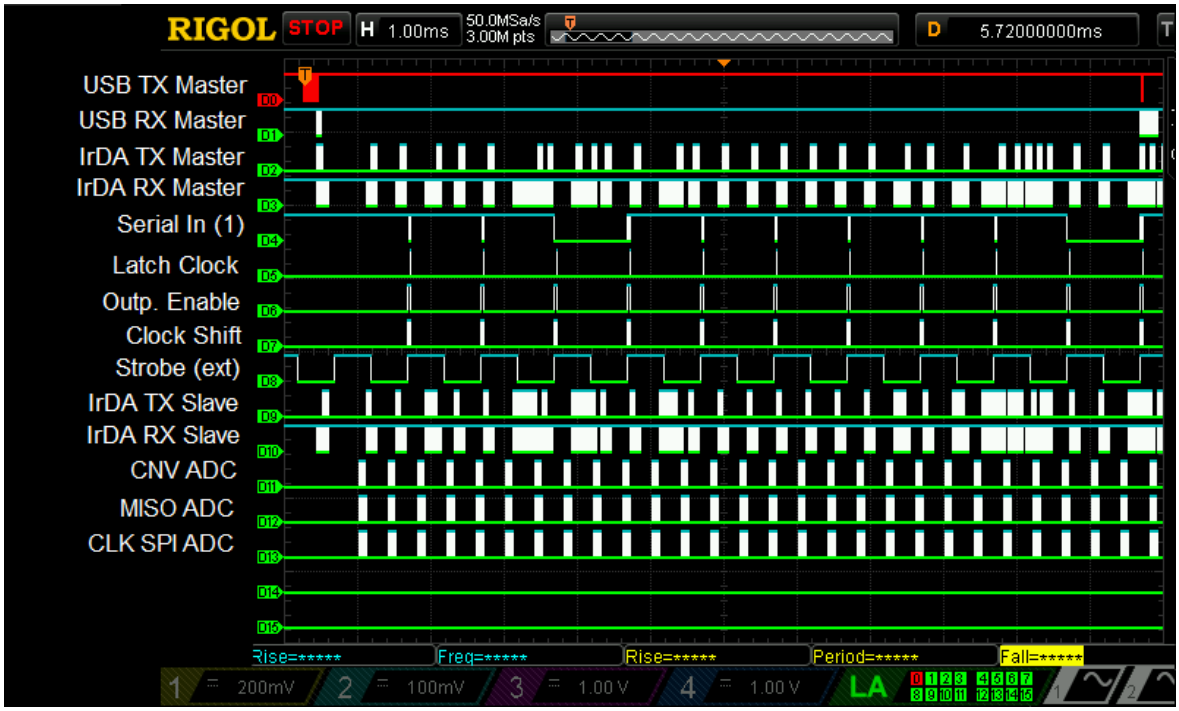


Figure 5.5 - General working in mode 2. Command D0

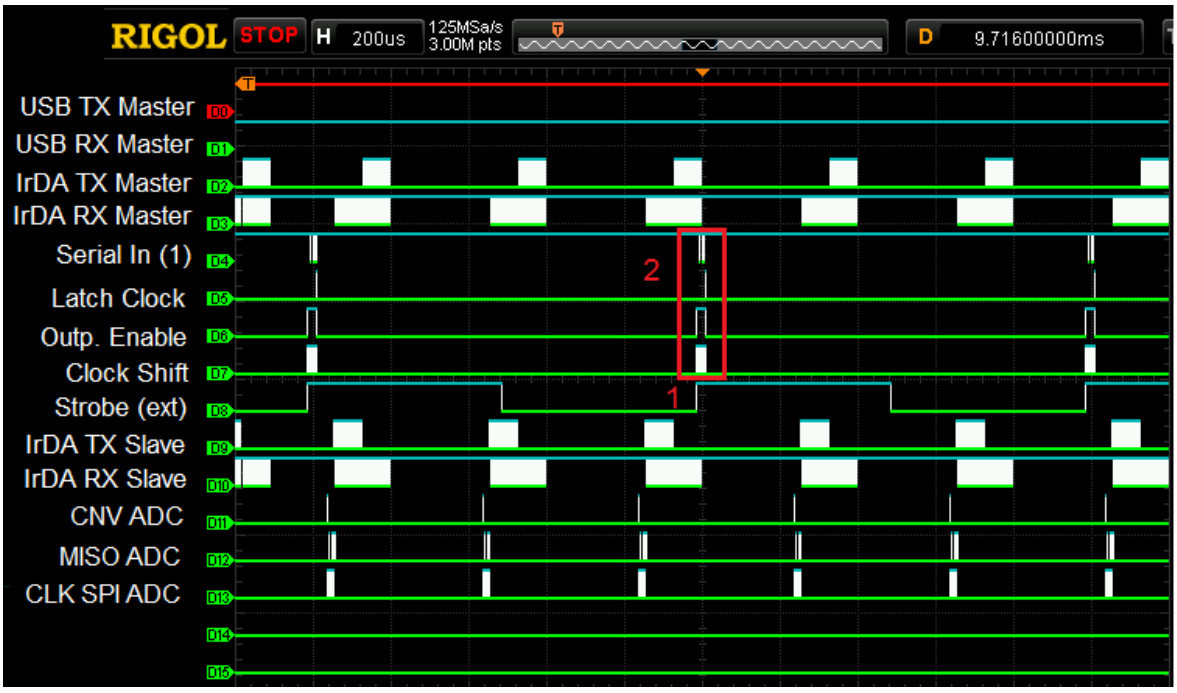


Figure 5.6 - General working in mode 2 with detail. Command D1

It is required to mention that this mode works with some problems when the measuring time is bigger than 1 minute. This is the reason why in the CD appendix there are both versions, one with 2 kilohertz of IrDA sending and other with 1 kilohertz.

6 Data visualization in real time and control system

Data visualization and the control of the system has been developed using LabVIEW software from National Instruments. The most important difference with respect to other programming languages is that this is a visual language. It means that every line of code can be replaced by its corresponding block or visual structure. Its execution is determined by the block diagram structure. These blocks are connected by wires that have different colors depending on its function (logic, data, errors…).

The LabVIEW program offers the user the possibility of selecting the COM port of the FPGA and the baudrate (1), initializing the FPGA (2), getting the offset value of the Microprobe (3), measuring in a manual way (4) or periodically (5), viewing in real time the values of the micro probe (8), sending commands manually (5), selecting the mode of data asking-receiving (6) and also selecting the features of the shift register sending (7).

This version of the LabVIEW program has a clear interface, and it is really functional. It can be used by users without great knowledge of the system. Figure 6.1 shows its interface and also the possibilities it offers in a general view:

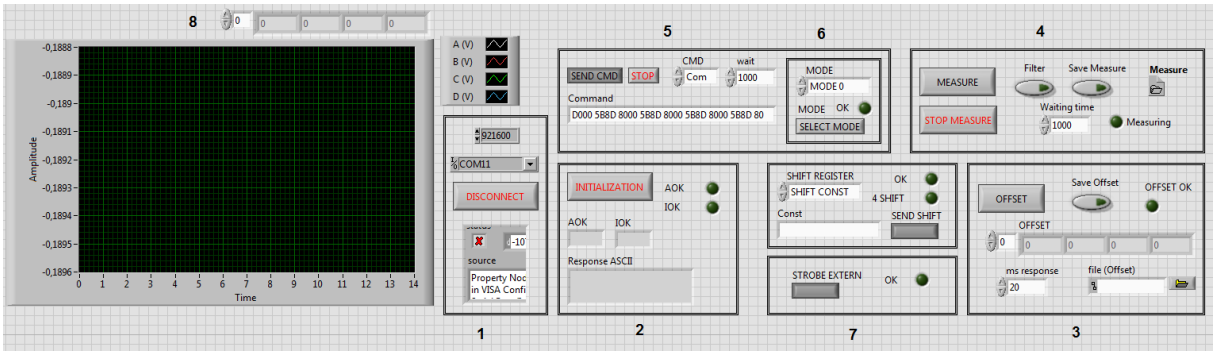


Figure 6.1 - LabVIEW visual interface

There is a while loop which is being executed every time, and the sending of command through USB to the FPGA starts only when the user press a button of the interface. It is possible also to change between functions inside the program, stopping a loop -for example the manual measure- and then to start another one –for example measure periodically-.

When a command is sent through USB and received correctly, the UART interface of the FPGA programming sends the acknowledgement (*AOK IOK* when the FPGA is initialized and *AOK* in normal times).

6.1 VISA communication protocol

Regarding the programming window, it is necessary to clarify that the protocol used to send and receive data through USB is Virtual Instrument Software Architecture (VISA), standard for configuring, programming, and troubleshooting USB interfaces.

At the begining of the program, it opens a serial communication port introducing the baudrate (default 921000), the number of bits and other variables (Figure 6.2). Then it sets a receiving buffer (32 bits) and reads the buffer in order to clear it. The error flag and the resource name have to be wired further to use the VISA serial port. The error flag by creating is displayed to be sure that the port is good opened.

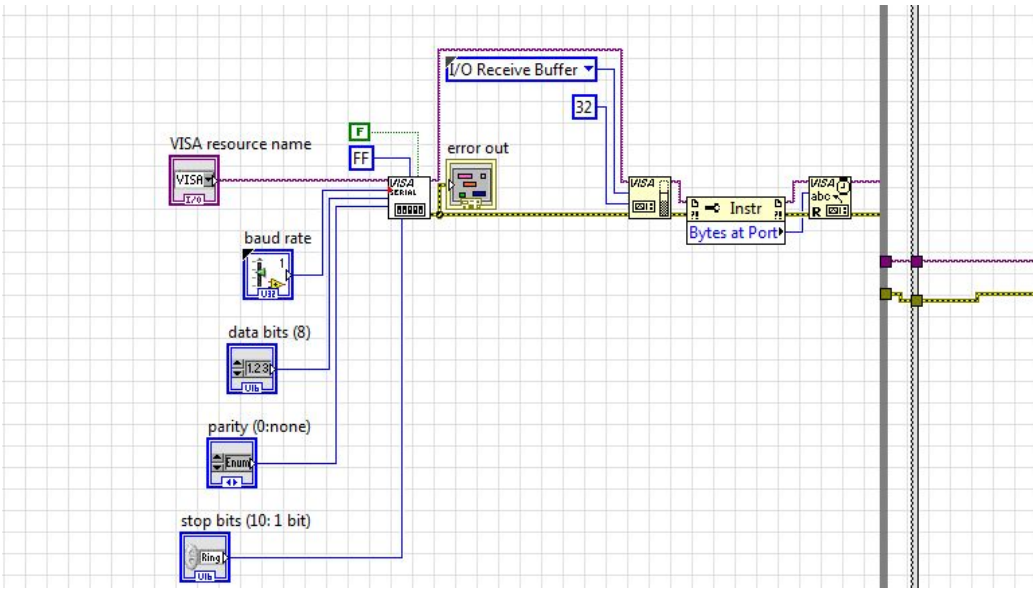


Figure 6.2 - VISA Protocol initialization

Then, this port is used to send and receive data to/from the FPGA depending on our requirements (set the offset, periodic measurement...). When the user wants to finish the communication and also the program, it is required to press the button “disconnection”. By clicking disconnection, the two blocks shown in Figure 6.3 will close the data transmission, ending the VISA protocol and also informing about errors (only in case).

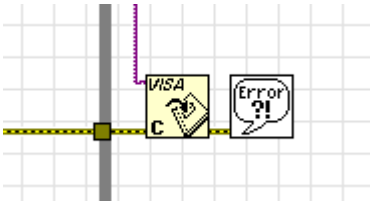


Figure 6.3 - VISA Protocol ending

6.2 FPGA initialization

In order to communicate with the FPGA trough the USB-UART port, it has to be initialized. This is a simple way to tear down the system. To initialize it, the ASCII word *INIT* (in hexadecimal *0x494E 4954*) has to be sent. When the FPGA receives this word, it answers AOK and IOK, which means that it has been good initialized.

In Figure 6.4 it is shown the initialization of the FPGA. First, it sends the INIT word and then if the acknowledgement is received, AOK and IOK indicators are set. If the acknowledgement is not good, then it is shown the word *ERROR* in IOK field. In this case, try again and reset the FPGA.

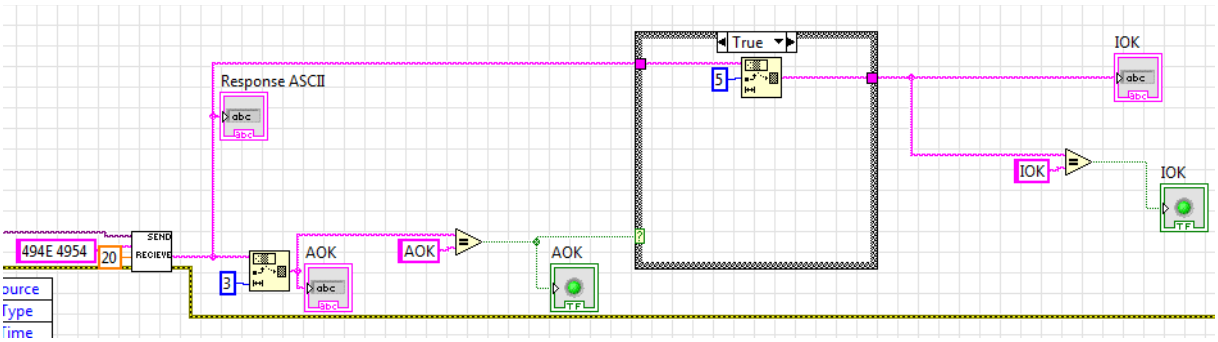


Figure 6.4 - FPGA initialization

6.3 Get offset value

By clicking the offset button the offset value of the channel (the microprobe is in rest position) is got. The command sent to the FPGA in order to receive it is 0x20, as it is possible to see in the Table 7.1.

In this case, a packet of data is received and the data have to be extracted in order to use them. The offset data packet is masked using a pattern of letters (a, b, c and d), as it is shown in Figure 6.5:

A	O	K	\r	\n
41	4F	4B	0D	0A

a	"	"	"	b	"	"	"	c	"	"	"	d	"	"	"	S	n	D	\r	\n
61	22	22	22	62	22	22	22	63	22	22	22	64	22	22	22	53	6E	44	0D	0A

Figure 6.5 - Data offset example

This is the reason why when the acknowledgement of the USB (*AOK*) is correct, it is deleted and it is only used the second part of the data string, as shows Figure 6.6.

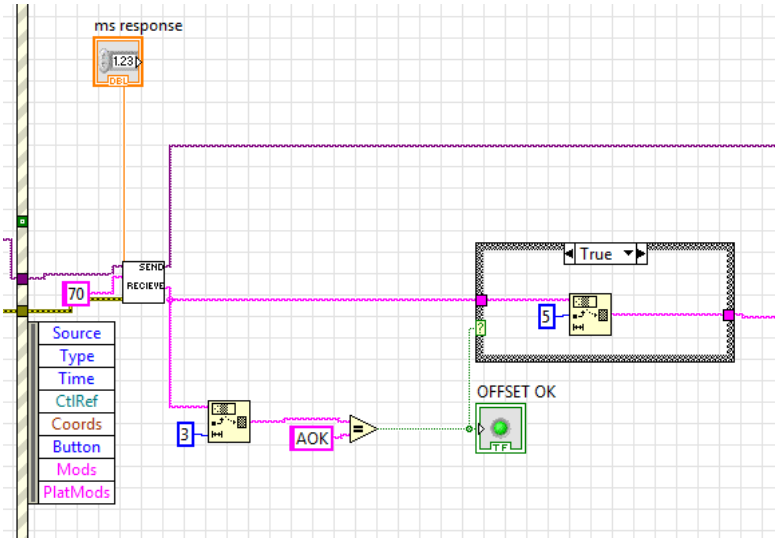


Figure 6.6 - Offset acknowledgement

Then, when there is only the part corresponding to the value, it is required to delete the mask of letters and to get only the values of the four AD channels (StringToABCD VI is explained in Chapter 6.8). After that, the data received from the AD Converter have to be converted in a voltage (String to Voltage, explained in Chapter 6.8).

Later, when the voltages are calculated, they are displayed in the computer screen (in this case, as a vector). The last part of the program (Figure 6.7) allows the user to save the voltages measured in a .txt file, only if the save offset button has been pressed before sending the command.

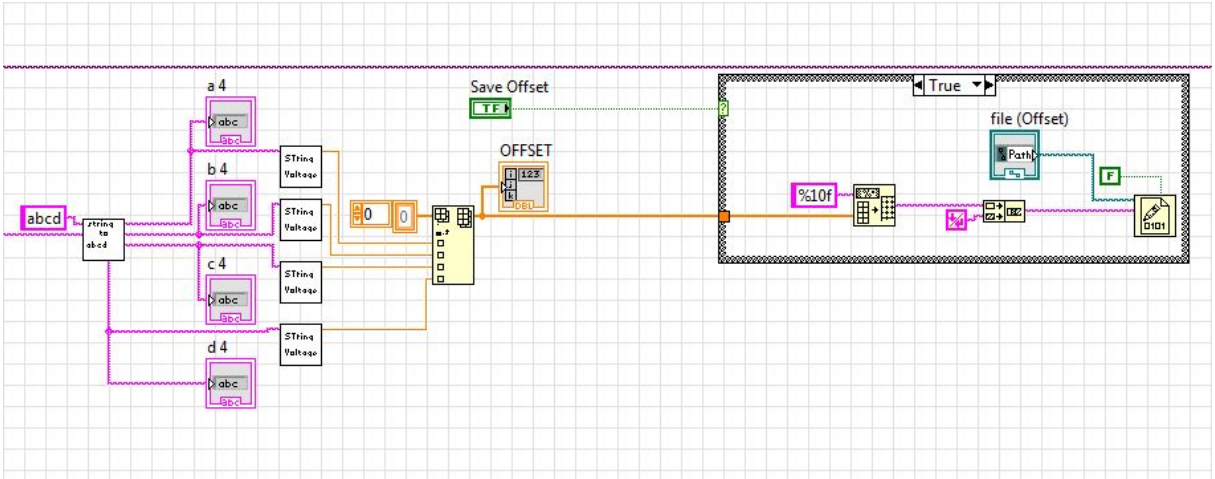


Figure 6.7 - Offset value packet extraction, conversion and saving

6.4 Manual measure

One aim of the present project is to monitor the actual value measured in the micro probe. To integrate this in the LabVIEW program, it has been necessary to add a new sequential structure which is executed after pressing the measure button. In this case, the command sent to the FPGA in order to receive that is 0x*30* (with filter) or *0x31* (without filter), as it is possible to see in the Table 7.1. This function only works in mode 0, and the sending structure is the following one:

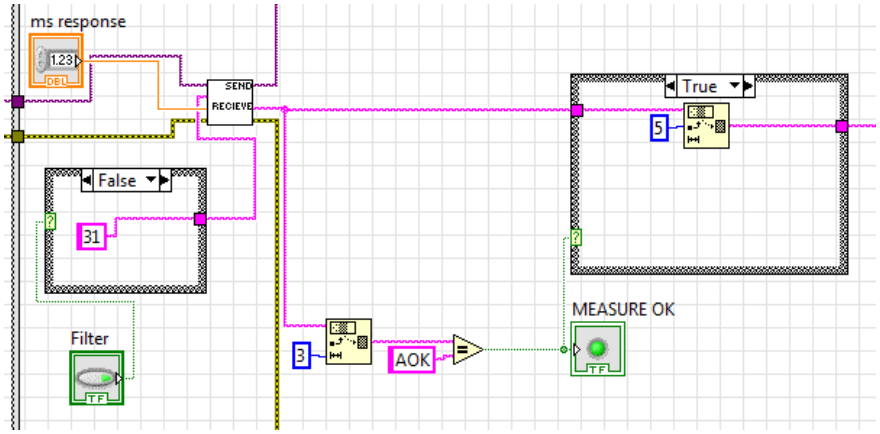


Figure 6.8 - CMD sending for manual measure

The selection between both commands is done only pressing a button if the measure required is with filter or without, as it is possible to see in Figure 6.8. The process follows with the data extraction from the packet and then there is a conversion in voltages (Figure 6.9).

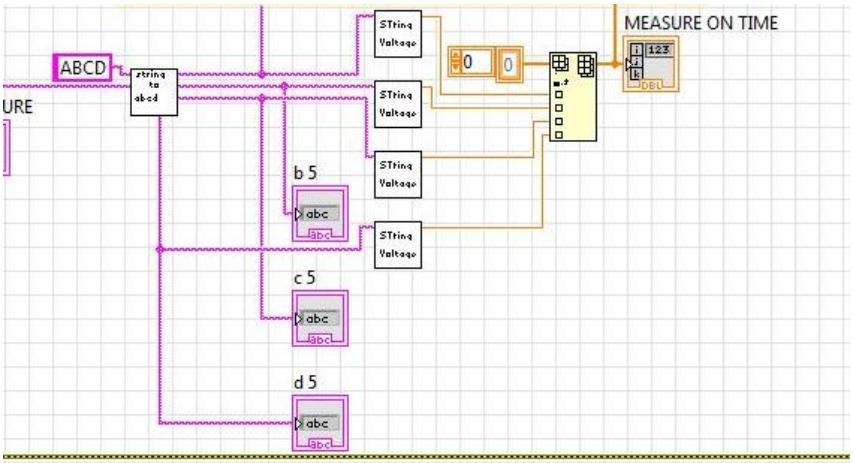


Figure 6.9 - Data packet extraction and conversion

The most relevant novelty relating to the previous subchapter is that in this one the data are displayed in a graphic chart, as it is shown in Figure 6.10. To get values periodically,

this case structure is repeated (the frequency can be configured in the LabVIEW interface through the waiting time variable). It is possible to save the data in a .txt file.

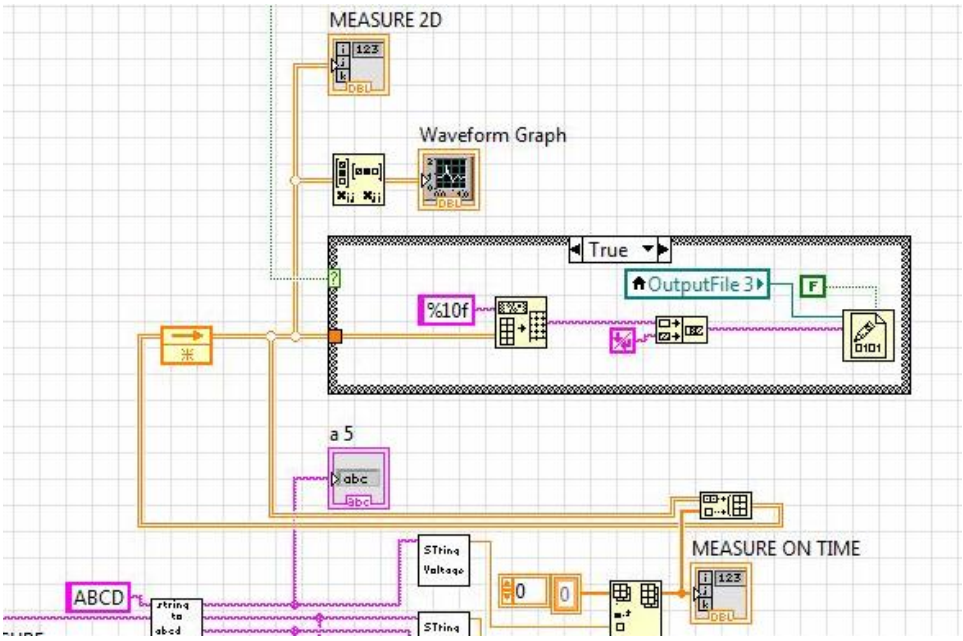


Figure 6.10 - Saving data and graph

6.5 Periodic measurement and sending commands

With the periodic measurement and sending commands panel it is possible to send to the master a command where the sending of data is done by the slave. The commands chosen to make this action are four:

- Mode 0: 0xD0 and 0xD1 (with filter and without filter, respectively).
- Mode 1: 0x30 and 0x31 (with filter and without filter, respectively).
- Mode 2: 0xD0 and 0xD1 (with filter and without filter, respectively).

More over it is possible to send some other commands from the PC to the FPGA by writing manually in the command field. Now, it is possible to send a command in hex or to pulse a button with one of the most used commands (*0x30*, *0x31*, *0xD0* and *0xD1*, in hexadecimal).

In addition to this, it is possible to send to the slave FPGA the commands *0x73* and *0x74* (hexadecimal) in order to test directly through USB the AD Channel without wireless communication. *0xD0* and *0xD1* commands need data after them, and in fact these data is useful to set the frequency of infrared data transmission.

The command sending structure is shown in Figure 6.11, where it is possible to see that two options are contemplated: sending a string or a predefined command.

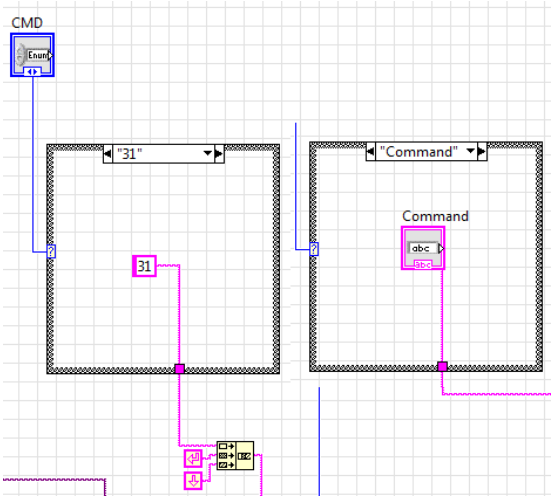


Figure 6.11 - CMD and string sending

After that, the FPGA sends through the USB the data from the Microprobe with the frequency selected, so it is necessary to make a receiving loop. The programming is similar to the last subchapter, as it is possible to appreciate in Figure 6.12.

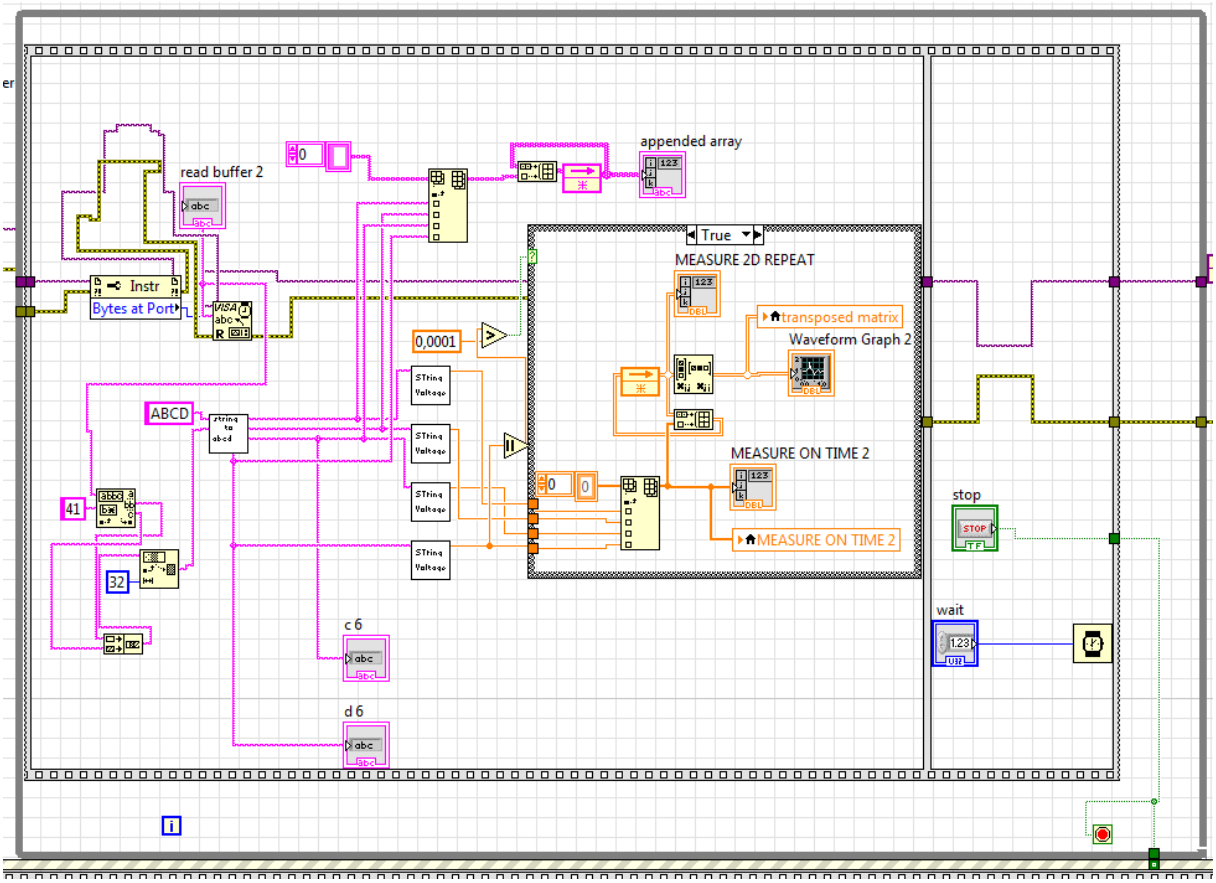


Figure 6.12 - Data receiving structure

However, there is an important difference between this loop and the previous one. In this case, it is required to wait until the first part of the data (A, *0x41* in hexadecimal) because the loop starts independently of the data received, it only depends on the frequency set by the user. In this way, every time the data shown is correct.

6.6 Select mode of operation

As it has been mentioned before, there are different possibilities of data transmission to the CMM. There are three modes of operation as mentioned, and it is possible to change between them using LabVIEW interface.

These modes will be explained with detail in chapter 7, and they can be configured by sending commands through the USB. The commands chosen are *0x90* (Mode 0), *0xF1* (Mode 1) and *0xF2* (Mode 2). As it is possible to realize, the LabVIEW programming is similar to Chapter 6.2, at least the sending structure (Figure 6.13).

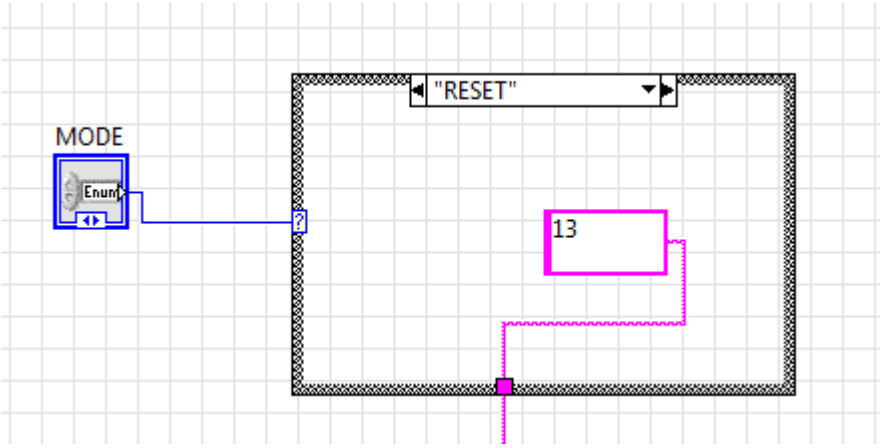


Figure 6.13 - CMD sending

In addition to this, to stop the data transmission between both FPGA it has been developed a reset case tool which allows it. When the command *0x13* is sent, then every process is stopped.

6.7 Select strobe intern/extern and shift register configuration

In order to achieve a properly data transmission between master FPGA and the CMM, the machine has as an output (input of the shift register) a 1 kHz signal. During the development of the present project it has been not possible to work with the machine, so this signal was simulated in the FPGA. This is the reason why in LabVIEW there is the possibility of selecting between using the intern strobe or extern strobe, sending the command 0x99.

In addition to this, it is possible to choose between sending three or four values, depending on the needs. In the first case, it is necessary to send three constants in order to reduce four data into three. In the constants field it is necessary to write this string (D3 is the command, x is the first constant, y the second and z the third):

0xD3-xxxxxxxx-yyyyyyyy-zzzzzzzzz-00000000

In the second case, it is required to send the command *0x98*. It would be good to clarify that the process and the diagram used is similar to the previous subchapter. The user interface is shown in the Figure 6.14:

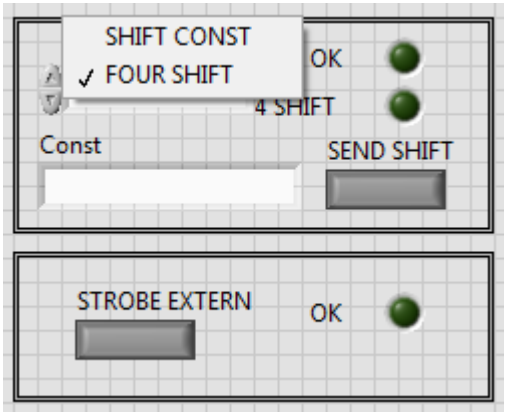


Figure 6.14 - Select strobe and shift register configuration

6.8 General used VIs (programming blocks)

Inside all case structures, it has been possible to see tree blocks every time: send-receive, stringtoabcd and stringtovalue. All of them have been created only for this project and they will be explained in the following lines.

6.8.1 Send-Receive block

This block allows sending and receiving data through USB. As it is possible to appreciate in Figure 6.15, the data sent is a concatenated string with data (command or command plus data), end of line and carriage. Then, this string goes to the writing VISA block, which sends the string through USB.

After that, a short period of time (default 20 ms) is needed in order to wait until the new data is received. Then, the Read VISA block allows the program to use it.

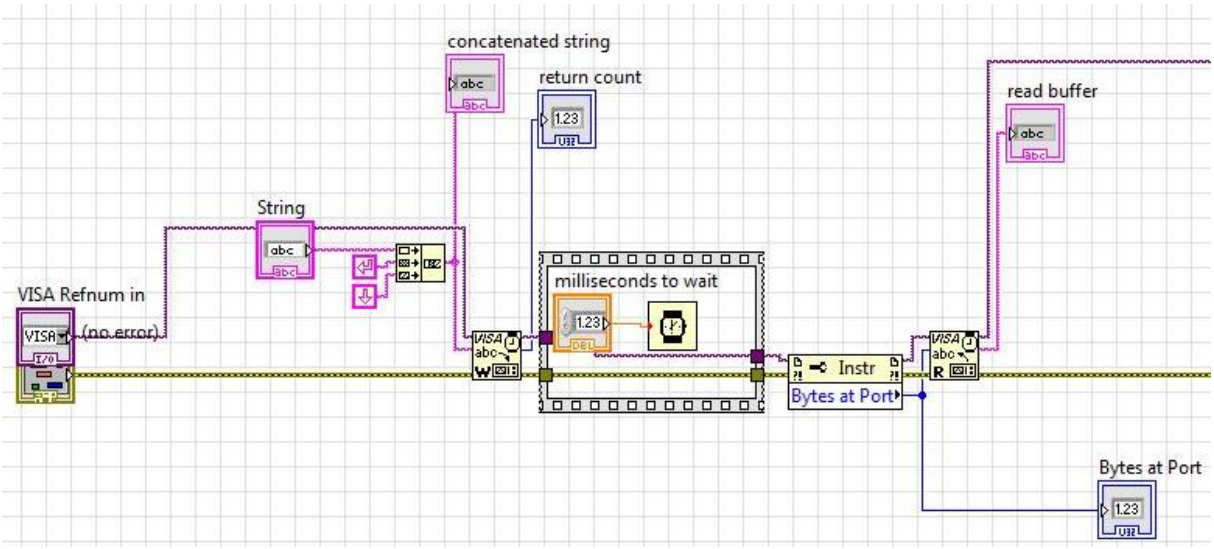


Figure 6.15 - Send-receive block

6.8.2 StringToABCD block

This block allows converting the string received from the USB into 4 smaller strings, one from each AD Channel. In order to achieve this, it is necessary to check if the string received has the suitable structure (for example the measure value has a mask using A, B, C and D as it is shown in Figure 6.16).

A	"	"	"	B	"	"	"	C	"	"	"	D	"	"	"	S	n	D	\r	\n
41	22	22	22	42	22	22	22	43	22	22	22	44	22	22	22	53	6E	44	0D	0A

Figure 6.16 - Data string example

Therefore, this string has to be split into four smaller strings, deleting the last 12 bits. After that, when it has been checked that the mask is correct, it only takes the values corresponding to the AD Channels, as it can be appreciated in Figure 6.17.

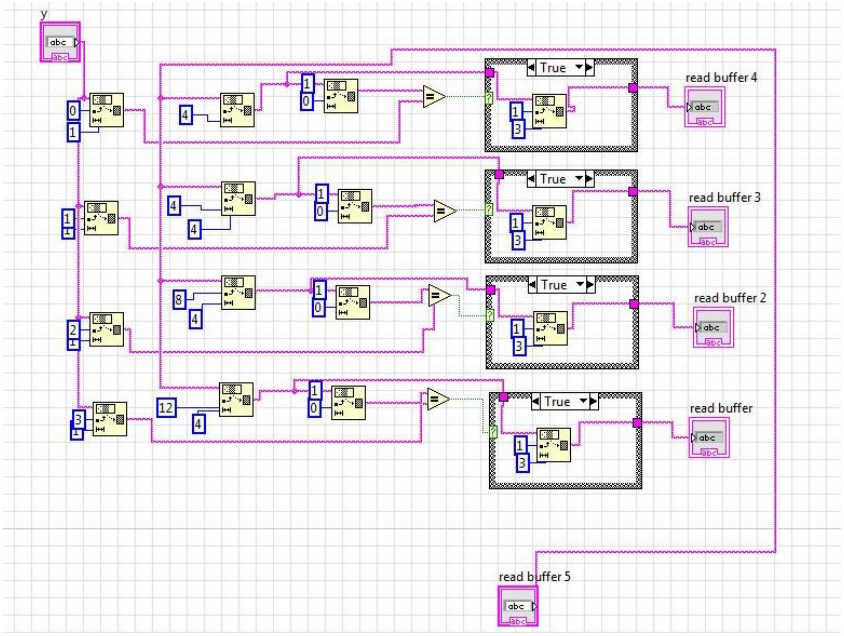


Figure 6.17 - String to ABCD block

6.8.3 StringToValue block

The values received from each AD Channel have to be converted in voltage, because when an analog-digital conversion takes place, there are some values that have relevance: resolution and supply.

According to this, the string received from each AD Channel has to be converted in an unsigned integer which is coded in two complement. There are values also negative, so following the AD Converter datasheet [2], the process needed to know the sign of the data is displayed in Figure 6.18:

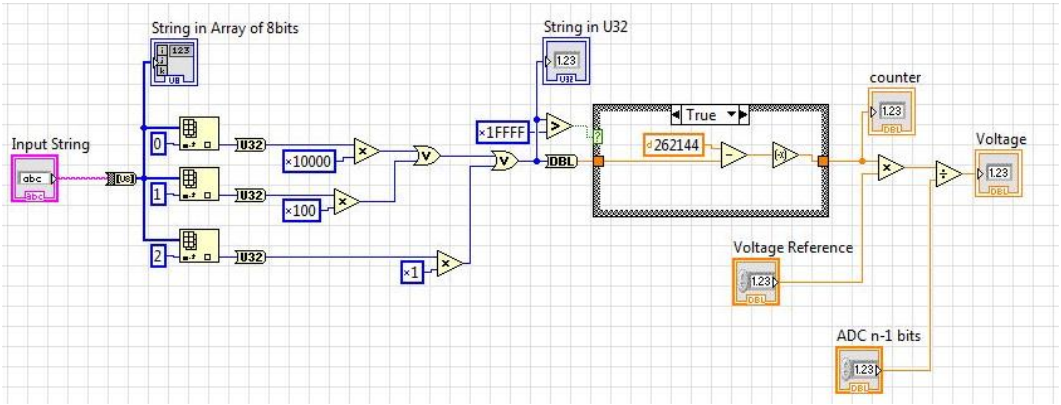


Figure 6.18 - String to value block

If the value is negative, then it is necessary to take the A2 complement. Finally, this value is divided by the resolution and multiplied by the supply of the AD Converter.

7 FPGA Programming

As it has been mentioned before, the master FPGA is the brain of the system developed in the present project. It should be able to manage information between all elements connected to it (digital parallel port/shift register/CMM, personal computer and the slave FPGA). However, the slave FPGA has also its importance in the system because it enables to measure the bridge (AD Converter, through SPI) which is going to be sent through IrDA to the master FPGA.

The VHDL program developed in this project is the same for both FPGA. In order to achieve it in an easy and clear way, the VHDL code has been structured in the way showed in Figure 7.1:

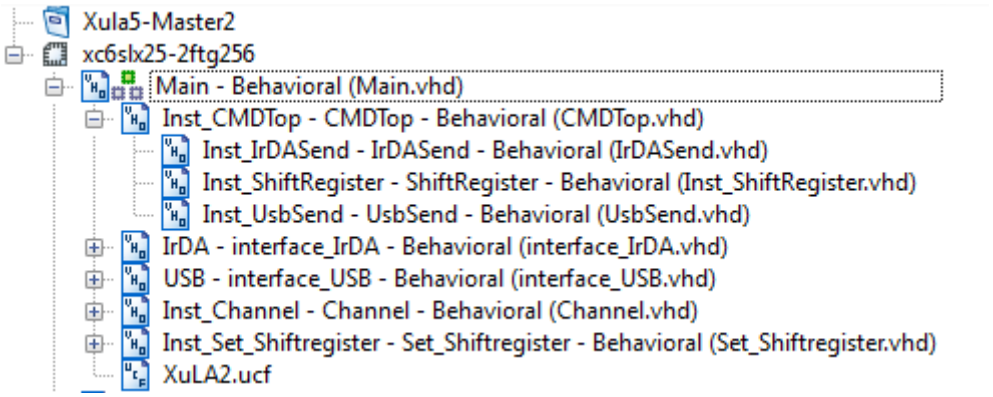


Figure 7.1 - Code hierarchy

The only difference between master and slave FPGA is Xula2.ucf file (configuration of input/output pins) and of course, the ports definition in the main program, as it is shown in Figure 7.2:

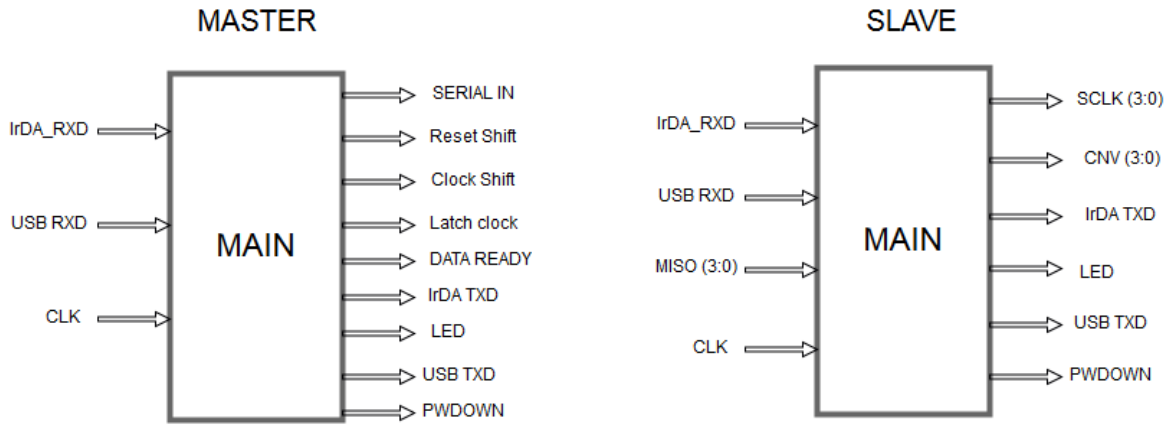


Figure 7.2 - Master and slave main code

The main code is the highest level of programming, where all inputs and outputs of CMDTop and the interfaces for the peripherals are wired. The most important module is *CMDTop*, because it is there where the command management takes place and where the data in/out of the interfaces are managed.

For each peripheral (USB-UART, IrDA, parallel port) there is an interface with which the receiving and sending of data is managed: AD Channel (AD Converter interface), UART interface, Shift register interface and IrDA interface. In chapter 8 all will be explained.

7.1 CMDTop

7.1.1 Generals

The principal function of this program is to manage data between all elements connected to the FPGA. It can be understood as a box, which has as input the data received from the interfaces (UART-USB, Infrared module, shift register or AD Converter) and as output the data to send through them.

These data from the interfaces provides data and also the information about the state of the communication modules, in order to know if it is possible send data –if it is busy- or if new data has arrived in a peripheral module (UART-USB for example).

This program operates with an intern command, used to manage the data origin and destination, and also to set the start of the data transmission. The extern functions (*IRDASend*, *ShiftRegister* and *USBSend*) and also AD Converter interface react (send) when there is a change in the intern command. The command flow is described in Figure 7.3:

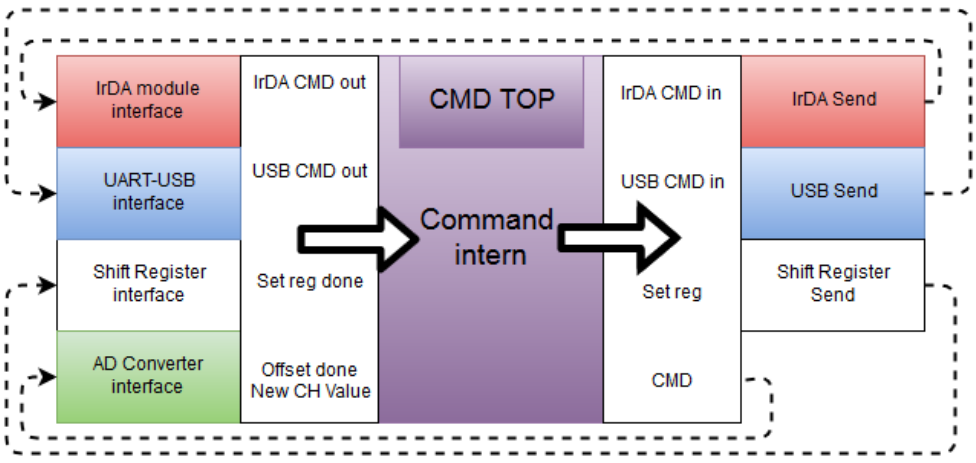


Figure 7.3 - Command intern flow

As it is possible to see, the intern command manages data between all modules. For example, if it has received new data from the USB and the data should go through the infrared module, this command takes the required value to order a wireless sending. The codification used to identify all communication modules (origin and destination) has been listed in the structure of the intern command (32 bit vector), which is the following one.

Bit	0-7	8-15	20-23	24-27	31
Function	IrDA/USB Command	AD Channel command	Data origin USB-UART: 0x1 IrDA: 0x2	Data destination USB-UART: 0x1 IrDA: 0x2 USB + IrDA: 0x3 Shift register: 0x4 USB + Shift register:0x5	Sending flag

In the following lines inputs and outputs of *CMDTop* will be listed, in order to achieve a global vision of the program.

INPUTS of CMDTop

- **CMDout_USB** (information about the USB data received and state of UART)

Bit	0-7	8-15	16	17	18	19-32
Function	USB Command	USB Data	New data received	Acknowledgement sent	UART busy	Not used

- **Dataout_USB** (4 standard logic vector of 32 bits each one, data received from UART-USB module)
- **Dataout_ IrDA** (4 standard logic vector of 32 bits each one, data received from IrDA module)
- **CMDout_IrDA** (32 bits vector, information about the IrDA module state)

Bit	0	1	1-2	3	4
Function	Data received ready	RX busy	TX busy	Data received latched	Data transmitted
Bit	**5**	**6**	**7**	**8-15**	**16-32**
Function	Acknowledgement done	New CMD	New data	IrDA command	Not used

- **CH_Value** (4 standard logic vector of 32 bits each one, measure value from all AD Channel)
- **Offset_Value** (4 standard logic vector of 32 bits each one, offset value from all AD Channel)
- **New_CH_Value** (flag to indicates there is a new measure value, AD Channel)
- **Offset_done** (flag to indicates there is a new offset value, AD Channel)
- **Set_reg_done** (4 bits vector, indicates that the data latching in the shift register has finished)
- **Reset**
- **Clock**

OUTPUTS of CMDTop

- **CMDin_USB**
  - Bit 0: Flag to start USB data transmission
- **Datain_USB** (4 standard logic vector of 32 bits each one, data to send through UART-USB module)
- **CMDin_ IrDA**
  - Bit 17: Flag to start IrDA data transmission
- **Datain_ IrDA** (4 standard logic vector of 32 bits each one, data to send through IrDA module)
- **CMD** (8 bits standard logic vector to request offset/measure to *AD Channel*)
- **Value** (4 standard logic vector of 32 bits each one, value to send through shift register)
- **Set_REG** (flag to init the shift register latching)

Before explaining the working of the program, it will be listed (Table 7.1) all commands (in hexadecimal) used in this program in order to know the meaning of all of them. All commands are defined for a specific source and receiver and only commands *from USB* are extern, the rest are intern.

Table 7.1: Commands

CMD	Function	Data	From	To
11	Get status register	-	USB	Master FPGA
20	Get the offset values	-	USB	Master FPGA
30	Get CH Values with filter	-	USB	Master FPGA
31	Get CH Values without filter	-	USB	Master FPGA
D0	Get periodic CH Values with filter	baudrate	USB	Master FPGA
D1	Get periodic CH Values without filter	baudrate	USB	Master FPGA
13	Reset	-	USB	Master FPGA
D3	Set shift register´s constants	Shift const	USB	Master FPGA
99	Select number of shift registers	-	USB	Master FPGA
98	Select strobe intern or extern	-	USB	Master FPGA
90	Select MODE 0	-	USB	Master FPGA
F1	Select MODE 1	-	USB	Master FPGA
F2	Select MODE 2	-	USB	Master FPGA
62	Get status register	-	Slave IrDA	Master FPGA
62	Get the offset values	-	Slave IrDA	Master FPGA
63	Get CH Values with filter	-	Slave IrDA	Master FPGA
64	Get CH Values without filter	-	Slave IrDA	Master FPGA
E7	Get periodic CH Values with filter	baudrate	Slave IrDA	Master FPGA
E8	Get periodic CH Values without filter	baudrate	Slave IrDA	Master FPGA
65	Reset	-	Slave IrDA	Master FPGA
70	Get the offset values	-	USB	Slave FPGA
71	Get CH Values with filter	-	USB	Slave FPGA
72	Get CH Values without filter	-	USB	Slave FPGA
73	Get periodic CH Values with filter	baudrate	USB	Slave FPGA
74	Get periodic CH Values without filter	baudrate	USB	Slave FPGA
13	Reset	-	USB	Slave FPGA
51	Get status register	-	Master IrDA	Slave IrDA
52	Get the offset values	-	Master IrDA	Slave IrDA
53	Get CH Values with filter	-	Master IrDA	Slave IrDA
54	Get CH Values without filter	-	Master IrDA	Slave IrDA
E5	Get periodic CH Values with filter	baudrate	Master IrDA	Slave IrDA
E6	Get periodic CH Values without filter	baudrate	Master IrDA	Slave IrDA
55	Reset	-	Master IrDA	Slave IrDA

As it has been mentioned before, there are different working modes. All commands work in mode 0 (default mode), but in mode 1 only *0x30* and *0x31* can be used. In mode 2, only *0xD1* and *0xD0* can be used. In the following subchapters the programming of the modes will be described.

7.1.2 Configuration of the system working

As it has been explained in Chapter 6.7, there are different options in the shift register performance. It is possible to select if the strobe is intern or extern (from the machine), the number of data used and also the constants required if the four read data have to be converted into three.

The process required to configure it only requires sending through USB the corresponding command (it can be consulted in Table 7.1). After this, the state machine goes to idle (the first state) and then it can work in the mode required (explained in the following subchapters).

7.1.3 Mode 0

To select this mode of working, first of all it is necessary to send through USB from the PC to the master FPGA the command 0x*90*. If it has been selected a periodic measurement, the command *0x13* stops every process and it goes to the initial state.

To clarify it, the process in this mode is in this way: first, the master receives a command from USB (which means a function to do), and then this is sent through IrDA. The slave sends the appropriate data and the master receives them, and of course sending through UART and shift register. It is possible to appreciate with detail in the Appendix A, where there are different figures describing the process in mode 0.

The flow chart of both FPGA will be described in Figure 7.4 (slave) and Figure 7.5 (master). First it will be displayed the working of the master FPGA and then slave FPGA. It should be noticed that in the slave FPGA flow chart it has been described also the process of direct measurement (asking for AD Channel data directly from USB).

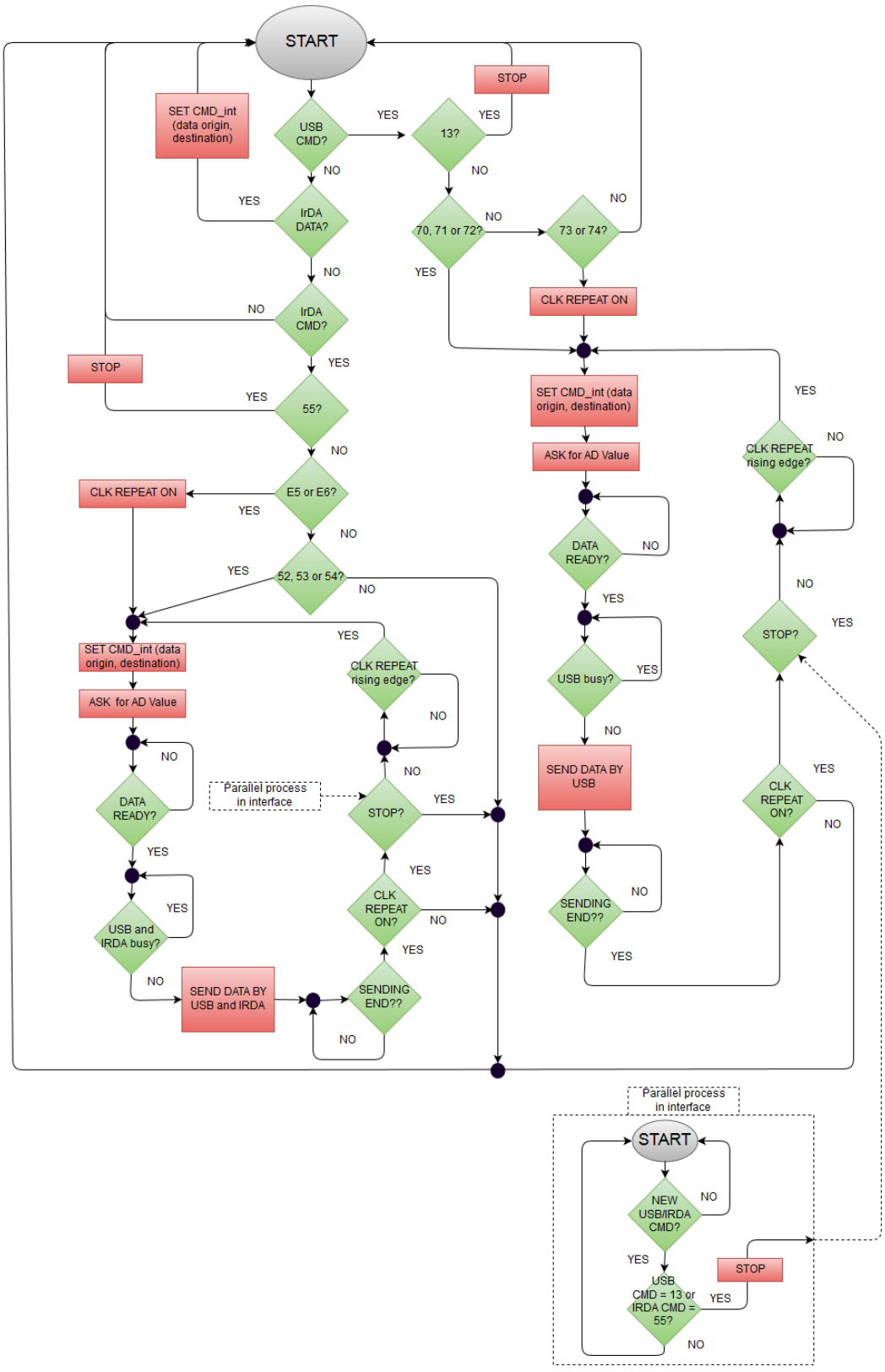


Figure 7.4 - Slave FPGA programming algorithm in Mode 0

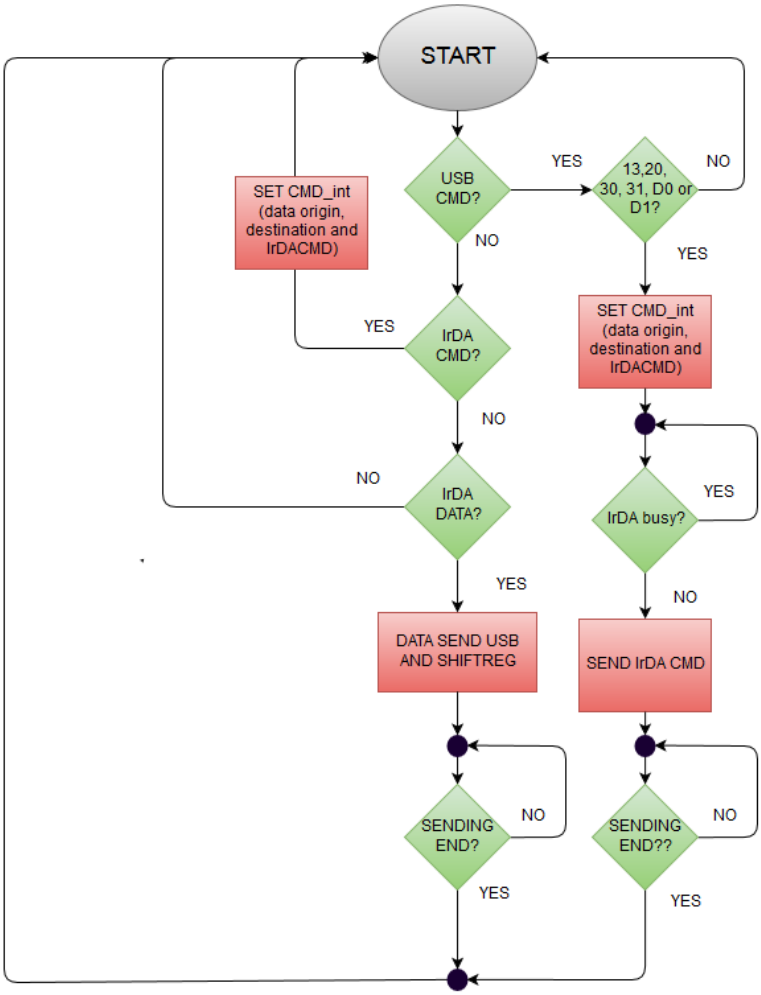


Figure 7.5 - Master FPGA programming algorithm in Mode 0

7.1.4 Mode 1

To select this mode of working, first of all it is necessary to send through USB from the PC to the master FPGA the command *0xF1* with data. In this mode, the IrDA transmission frequency is 1 kHz (enough for the desired data frequency of the machine).

In this case, as it has been possible to see in Chapter 5 and also Appendix A with more detail, the master receives a command (*0x30* or *0x31*) from USB and then it is sent through infrared module by a strobe rising edge. Then, the slave sends the measured data.

After receiving them, the data are sent every time through the parallel port. For the USB the data are sent each 10 times (10 ms). The programming of both FPGA will be described in Figure 7.6 (master) and Figure 7.7 (slave).

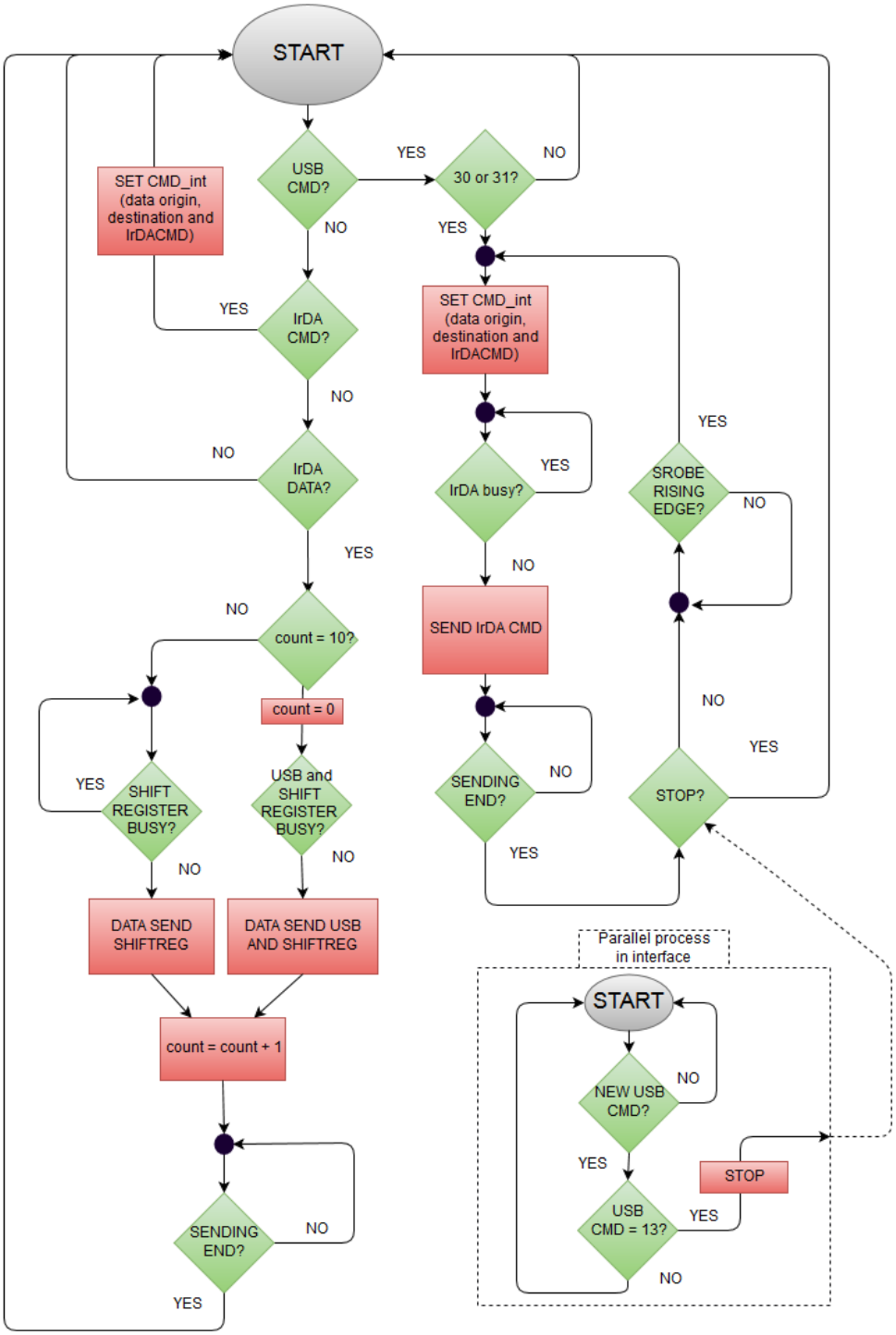


Figure 7.6 - Master FPGA programming algorithm in Mode 1

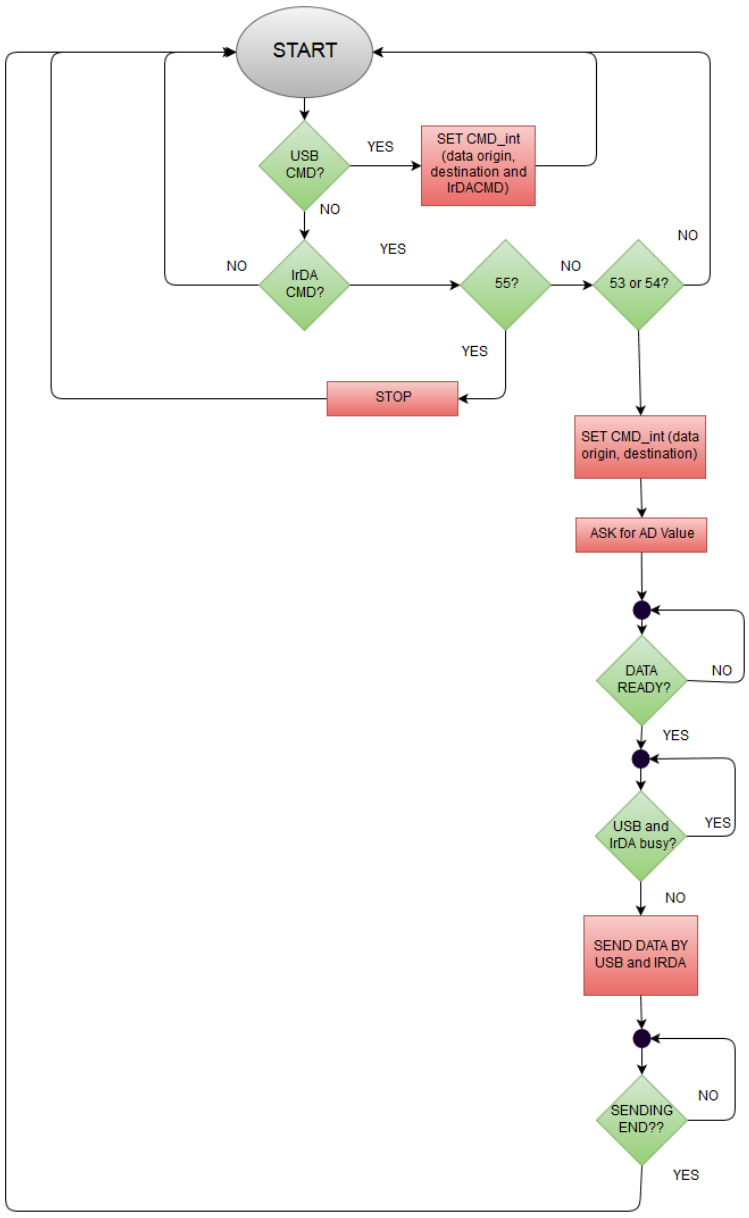


Figure 7.7 - Slave FPGA programming algorithm in Mode 1

7.1.5 Mode 2

To select this mode of working, first of all it is necessary to send through USB from the PC to the master FPGA the 0x*F2* command with data. After that, the master sends it to the slave and then this one sets the frequency of the IrDA sendings (2 kHz).

The main difference to mode 0 is that now, the master receives a command (*0xD0* or *0xD1*) from USB and then it is sent through IrDA. After that, the slave sends the appropriate data every time with the new frequency set. Of course, the master receives them but only when the strobe is rising edge, the last data is sent through UART-USB and/or shift register (only one of ten sendings is through UART-USB).

This one is the most suitable mode to achieve high frequencies of data transmission, because the asking of data every time is not needed, so the waiting is near the half. Therefore, it is possible to reach IrDA data transmission with frequencies much bigger than 1 kHz.

It will be described the programming of both FPGA in Figure 7.8 (master) and Figure 7.9 (slave).

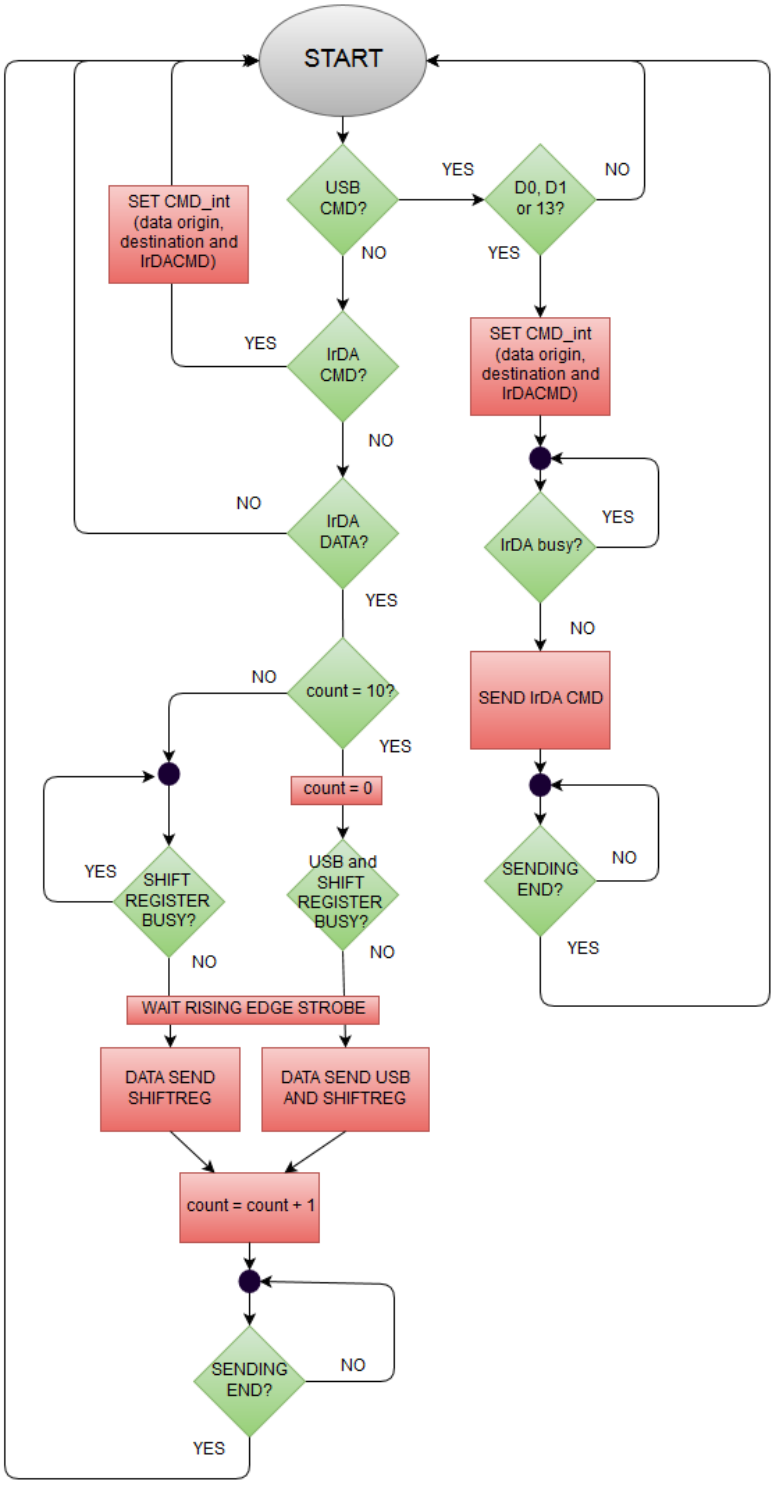


Figure 7.8 - Master FPGA programming algorithm in Mode 2

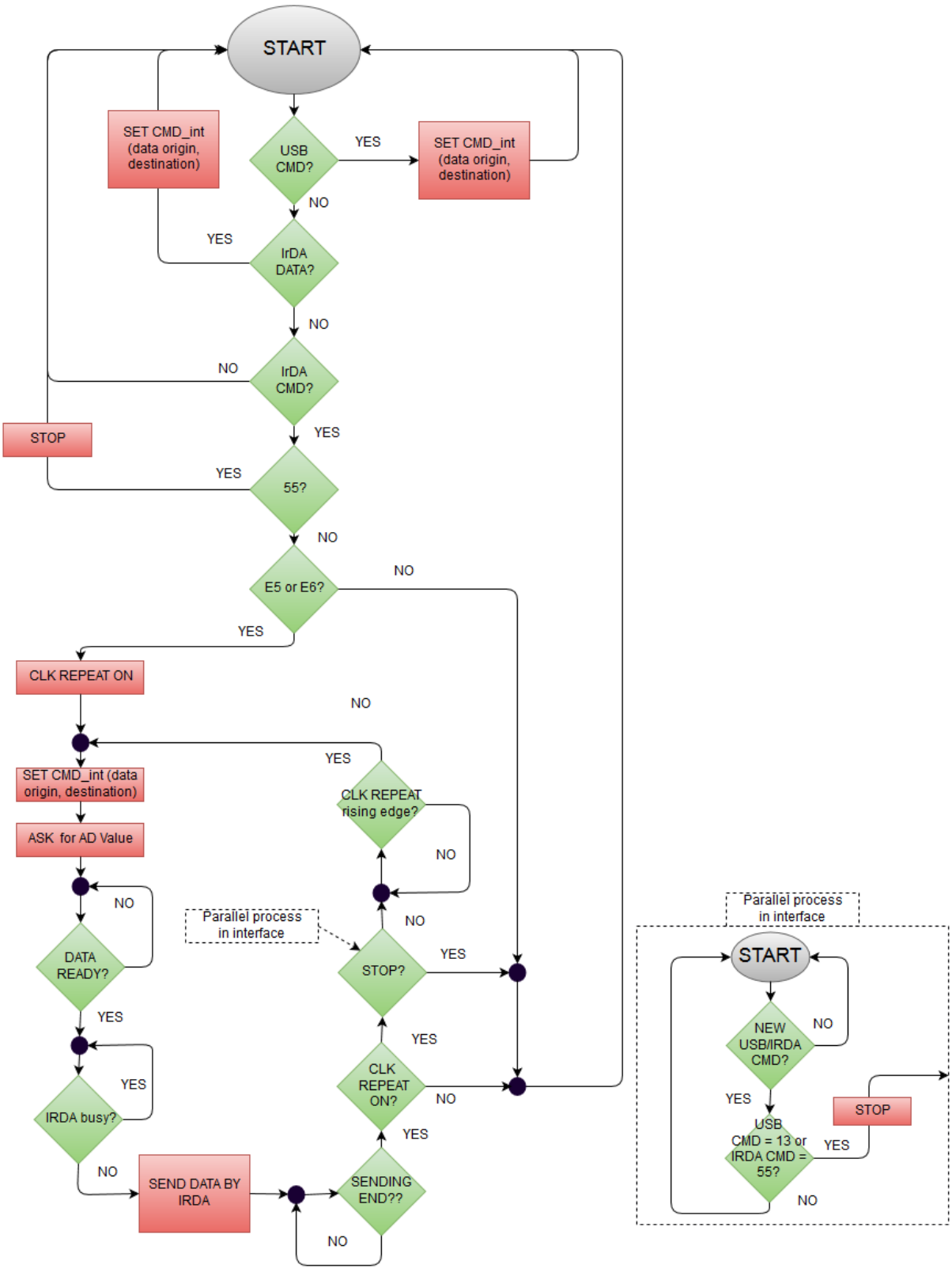


Figure 7.9 - Slave FPGA programming algorithm in Mode 2

7.2 IrDASend and USBSend

As it has been shown before, the working of an entity can be explained as a box. The inputs of the IrDA box are four: CMD_int (command intern), data, rst (reset) and the clock. As outputs we have: data_irda and CMD_irda (in fact, CMDin_IRDA).

In the USB case, the inputs of the box are also four: CMD_int (command intern), data, rst (reset) and the clock. As outputs we have: Data_USB and CMD_USB (in fact, CMDin_USB).

When the FPGA (master or slave, both) receives a command which involves a sending through the infrared or UART module, the command intern (which is used to manage the data origin and destination, and also to set the start of the data transmission) allows this box to do its function: sending through the infrared/UART module.

The state machine has three states (Figure 7.10). In the first one is every time looking for a desire of sending through the infrared/UART module. When it happens, then the second state takes place, and inside it the data to send is set and of course the sending flag is activated. After this, in the third state there is a waiting until the data transmission has finished and then it goes to the first state again.

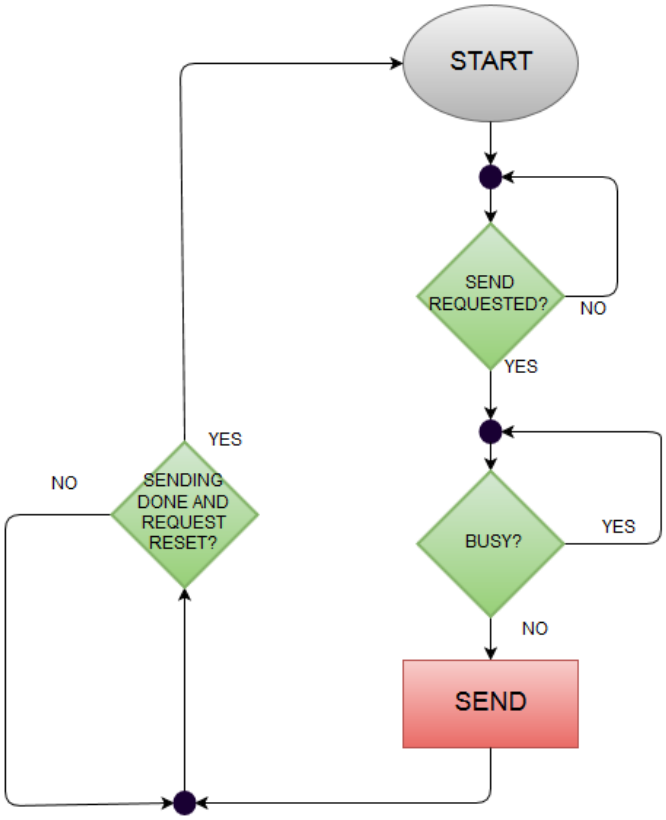


Figure 7.10 - IrDA and USB send programming algorithm

7.3 Shift register

As it has been shown before, the working of an entity can be explained as a box. The inputs of the box are six: CMD_int (command intern), Set_reg_done (a flag which says if the shift register can be used or not), data, rst (reset), shift_const (some constants to convert the four AD Channel values to the three values sent through the shift register) and the clock. As outputs we have: value (the data which is going to be sent), data ready (a flag to indicates the data is ready) and set_reg (a flag which allows the SPI protocol to start the transmission).

When the master FPGA receives a command which involves a sending through the shift register, the command intern (it is used to manage the data origin and destination, and also to set the start of the data transmission) allows this box to do its function: sending through the shift register.

The state machine has three states (Figure 7.11). In the first one is every time looking for a desire of sending through the shift register. When it happens, then the second state takes place. First, it looks if the shift register is busy and if it is not busy, the data to send is set (according to the number of shift registers to use) and of course the sending flag is activated. After this, in the third state there is a waiting until the data transmission has finished and then it goes to the first state again.

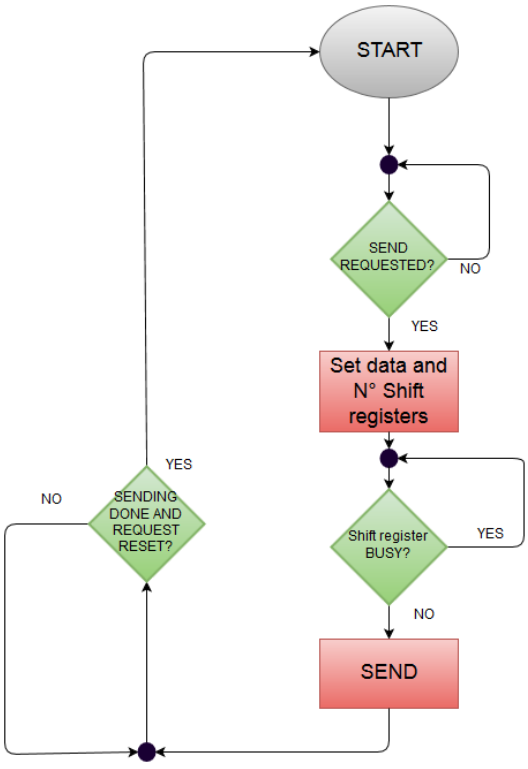


Figure 7.11 - Shift register send programming algorithm

The most important state is the second one, where the data to send is calculated, and of course the set_reg flag is activated. To calculate the data which is going to be sent through the shift register, it is necessary to know first the number of shift register which are going to be used. As it has been mentioned before, it is set using the command *0x99*. If it is selected the option of 4 shift registers, the data of each AD channel is sent through each shift register.

If the option selected is the other one, it is required a matrix to convert 4 data in 3 data. However, it is really small so it will be presented here as a simple operation:

$$Value(0) = Shift\_const(0) \cdot (Data(0) - Data(2)) \quad 7\text{-}1$$

$$Value(1) = Shift\_const(1) \cdot (Data(1) - Data(3)) \quad 7\text{-}2$$

$$Value(2) = Shift\_const(2) \cdot (Data(0) + Data(1) + Data(2) + Data(3)) \quad 7\text{-}3$$

$$Value(3) = 0 \quad 7\text{-}4$$

Value: Value to send through the shift register to the machine

Shift_constant: Constant set by the command *0xD3*

Data: Value received from the AD Channel

When the data is ready to be sent through the shift register, the flag data ready is set to '1', allowing the CMM to know the data is ready. In Figure 7.12 it is possible to appreciate the simulation of a shift register´s setting (data = 0x222222): a rising edge of set_reg starts the setting of the shift register, then the output is disenabled (output_enable high) and the data are shifted out of the FPAG in accordance to clock_s (shift clock). When all data are transferred, then the output is enabled and the data are latched on the ouptut register through the rising edge on latch_clock_s.

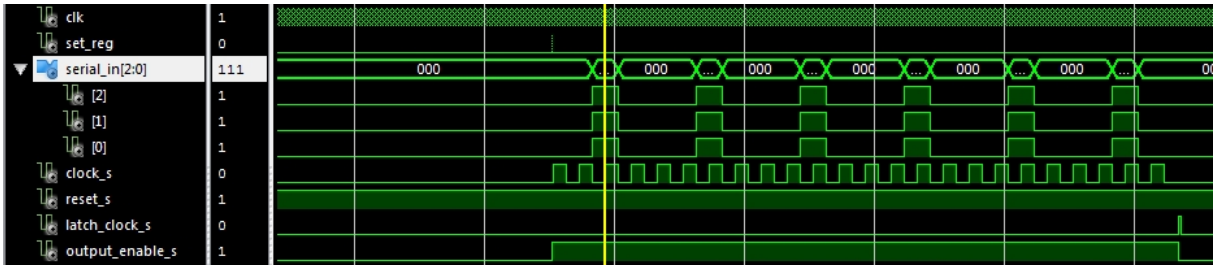


Figure 7.12 - Simulation of shift register´s working

8 FPGA Programming: Communication interfaces

In this chapter the communication interfaces used and implemented in FPGA will be explained. To enable the communication between FPGA and PC, UART has been used; Infrared transmission interface to enable the wireless data transmission between both FPGAs and Serial Peripheral Interface to enable the communication between FPGA and AD Converter and also between FPGA and shift register.

8.1 UART interface

The UART interface can be explained as a box (Figure 8.1 - UART interface.), with five inputs: CMD in (more details in Chapter 7.1), DATA in (data to send through UART, 4 x 32 bits), clock, reset and USB RXD (physical input from the USB cable). As outputs we have the CMD out (more details in Chapter 7.1), DATA out (data received, 4 x 32 bits) and also USB TXD (physical output of the USB cable).

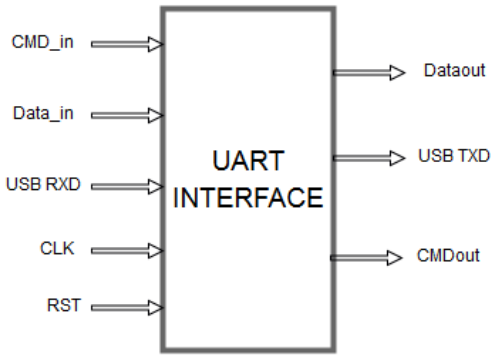


Figure 8.1 - UART interface.

Before explaining the processes that take place in the programming, it will be described the communication protocol used. The UART interface uses the protocol RS232 to send and receive data. It defines the signal used in communication and also the hardware to transfer signal between devices, as shown Figure 8.2:

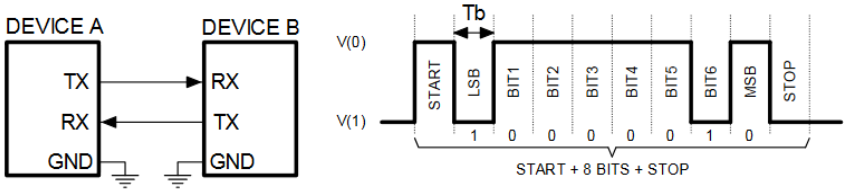


Figure 8.2 - RS232 Protocol. [27]

As it is shown in the figure, this standard defines the number of bits transferred within one pack, eight. There is a start bit and then 8 bits of data. Finally, there is a stop bit that finishes the pack sent. RS232 defines also the speed of the transmission (baudrate), commonly 9600. However, in this project it is 921600.

After explaining the basics of the protocol, it is going to be explained the management of data received from USB and the process to get the information from it.

There are four important processes in this interface:

- The latching of data received
- Data received management
- USB sending
- Commands management

The first process is looking continually for new data received through the UART RX wire, which is signalized thought a signal rising edge from the UART decoder. By receiving a new data this process memories it for further treatment.

The second one, data received management, is a process which is every time looking for a new data word. The first word received has to be INIT (attached with cr lf at the end) in order to initialize the interface. If this sending has finished successfully, then it sends AOK IOK. After this, the UART interface is able to receive commands with or without data (4 x 32 bits). Every time it receives a successful command (with or without data), it sends the acknowledgement AOK.

In addition to this, there is a filtering of commands: if the command is bigger than 0xC0 then it waits for data (128 bits). If it is smaller than 0xC0 but bigger than 0x10 it receives the command but it does not wait for data. If the data are cr (0x0d) or lf (0x0a), it does nothing but it is useful to end the communication and to return to the idle state of the UART interface.

In the third process, the USB sending process, there are three states. The idea is that in the first state, it is every time looking for a rising edge of a flag which indicates the desire of sending data. When it happens, then it goes to the second state. In the second state, the sending of 4 x 32 bits takes place. Finally, in the third state, it sends SnD with crlf in order to finish the sending of data.

Finally, the acknowledge management of the UART interface. This process is the responsible of sending the corresponding acknowledgement. It sends AOK IOK when the INIT word is received in order to initialize the interface; AOK when a correct command is received and ERROR when an incorrect command is received.

8.2 IrDA interface

The IrDA interface can be explained as a box (Figure 8.3), with five inputs: CMD in (more details in Chapter 7.1), DATA in (data to send through IrDA, 4 x 32 bits), clock, reset and IrDA RXD (physical input from the IrDA module). As outputs we have the CMD out (more details in Chapter 7.1), DATA out (data received, 4 x 32 bits) and also IrDA TXD (physical output of the IrDA module).

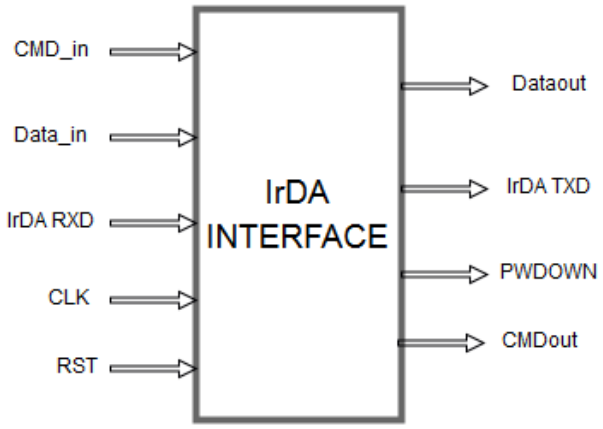


Figure 8.3 - IrDA interface.

The general working of the IrDA interface is as follow: one of the two FPGA sends a packet of data which is composed of a command (with data or not). After receiving the packet and testing if the data are valid with the CRC, the acknowledgement is sent within a time delay and then, the conversation ends. It is displayed in Figure 8.4:

Figure 8.4 - IrDA data transmission simulation

As it has been explained in Chapter 3.3, the IrDA protocol chosen for the present project is 4PPM. According to this, there are two different modules in the IrDA interface: transceiver and receiver. In the following lines they will be explained in detail.

The packets sent through IrDA have the same structure as it is shown in the Figure 3.7. As it has been mentioned before, the 4PPM packet has a 32-bit cyclic redundancy check (CRC) field appended to it. Also, there is a preamble, a start and a stop. Depending on the command, the data sent (frame body) changes. However, the frame body has every time the same length: 4 x 32 bits, so 16 bytes in all.

8.2.1 IrDA interface programming hierarchy

The structure of this interface programming is shown in Figure 8.5:

Figure 8.5 - IrDA interface VHDL code hierarchy

TX module

In the TX module, the top level of programming does not have other function than wiring the interface and the 4PPM encoder.

The sending of data through the IrDA module is managed in the TX interface, and of course it uses the data generated in the two lower levels: FIFO temp memory and CRC32 generator. The first one (FIFO, first in first out register) is the responsible of saving the data which is going to be sent through the IrDA module and CRC32 is responsible of generate the CRC32 before sending.

The 4PPM encoder, as it has been explained in Chapter 3.3, encodes the data packet and generates the 4PPM signal. The TX data encoder has this working:

- "00" => "0001"
- "01" => "0010"
- "10" => "0100"
- "11" => "1000"

The 4PPM data sent structure is (each data stream has 32 bits):

- data_stream(0): "80ACDE80” (Preamble)
- data_stream(1): "80ACDE80” (Preamble)
- data_stream(2): "000C000C" (Start)
- data_stream(3): Data(0)
- data_stream(4): Data(1)
- data_stream(5): Data(2)
- data_stream(6): Data(3)
- data_stream(7): CRC 32
- data_stream(8): "00060006" (Stop)

RX module

In the RX module, as it is in TX module, the top level of programming does not have other function than wiring the interface and the 4PPM encoder.

The receiving of data through the IrDA module is managed in the RX interface, and of course it uses the data generated in the two lower levels: FIFO temp memory and CRC32 generator. The first one (FIFO, first in first out register) is the responsible of saving the data received through the IrDA module and CRC32 is responsible of generate the CRC32 and compare it with the CRC32 received. If they are the same, the data has been successfully received.

The 4PPM decoder, as it has been explained in Chapter 3.3, checks first if the CRC is correct and then decodes the data packet (in the opposite way as it has been encoded in the TX 4PPM interface) and rebuilds the data received to able upper levels of programming an easy management of the data.

IrDA transmission interface

This is the highest level of programming of the interface. In this entity there are 3 important processes:

1. Set the transceiver in FIR mode 4Mbits/s

According to the basics explained in Chapter 3.3, the IrDA module has to be set in FIR mode (Fast Infrared Mode), allowing it to operate with a high frequency.

2. Sending of data

This process is looking every time for a desire of sending data through IrDA module. When it is required, then the data to send is set according to the command (Table 7.1). After this, the data stream is set with the data to send and the transmission starts. When it has finished (it means the acknowledgement is received), the FIFO is cleared. If the acknowledgement does not come, then the data is resent.

3. Receiving of data

When new data is received through IrDA module, this process latches the data received to data out (the output of the interface). In addition to this, there is a filter of data: if the first 32 bit packet is the same as the second one, the flag new IrDA command receives is set to '1'.

Also, if the data received has the mask of letters used to transmit the data from the Wheatstone Bridge (A, B, C, D), the flag new IrDA data is set to '1'. Both flags are integrated in the CMD out explained in Chapter 7.1.

8.2.2 Problems found and correction

The IrDA interface was already done at the beginning of this project, but there were some errors that made the system unstable. In order to achieve an acceptable reliability, many changes were necessary.

The main problem was that when the state machine came to the idle state, sometimes it started the process when the last loop had not ended, so it was necessary to add one condition more in order to be sure that the start of the next loop is in the correct moment.

```
if  tmp_ird_rx = '0' and ird_rx = '1'  and irda_stream_start = '0' then
      state1 <= init;
      busy <= '1';
end if;
```

This last part of the code has been modified in TX and RX modules, receiver and transmitter. Due this changes, the IrDA system now offers a 99,4% of reliability (it has been subjected to a test of 10.000 sending-receiving, 10 per second).

In addition to this, it was required a filter, allowing the system to send only when the data is different to zero and of course avoiding any type of instability. This change is possible to see in the following part of code:

```
if rx_end = '1' and rx_end_tmp = '0' then
       if data_stream(2) = x"000C000C" and data_stream(4) /= x"00000000" then
              fifo_clr_en <= '0';
              state1 <= run;
              data_stream_index1 <= nb_Preamble + 1;
       end if;
end if;
```

8.3 Serial Peripheral Interface

In spite of not being a hardware module interface, it has been used to transmit data between FPGA and shift register and also between FPGA and AD Converter, so it will be explained. Serial Peripheral Interface (SPI) is a synchronous serial communication interface used for short distance communication. It is based on full duplex cables, using a master-slave architecture, as it is possible to see in the Figure 8.6 (SCLK is the serial clock, SS means slave select, MOSI is the master data output and MISO the master data input). It is possible to have different slaves, selecting the select slave (SS) pin.

Figure 8.6 - SPI structure. [15]

After setting the SS low, the data are clocked out in accordance of the SPI mode on MISO and MOSI simultaneously. Four modes of working are possible in SPI, defined by CPHA (clock phase) and CPOL (clock polarity). They determine when the data are available on MOSI pin (if CPHA = 1, data are available). This is shown in the Figure 8.7 (red lines represent CPHA=0 and the blue ones CPHA=1):

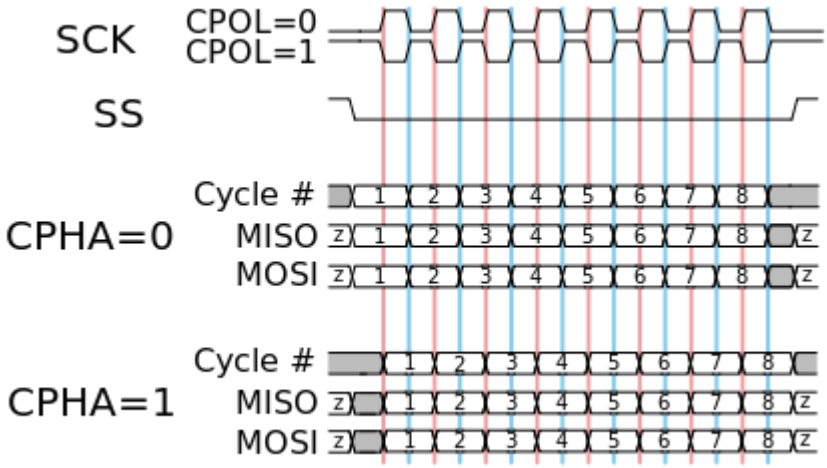


Figure 8.7 - SPI timing diagram. [15]

Mode 3 is used in the present project to send data through the shift register, and also to get digital data from AD Converter AD5782. The master working is described by two parallel processes:

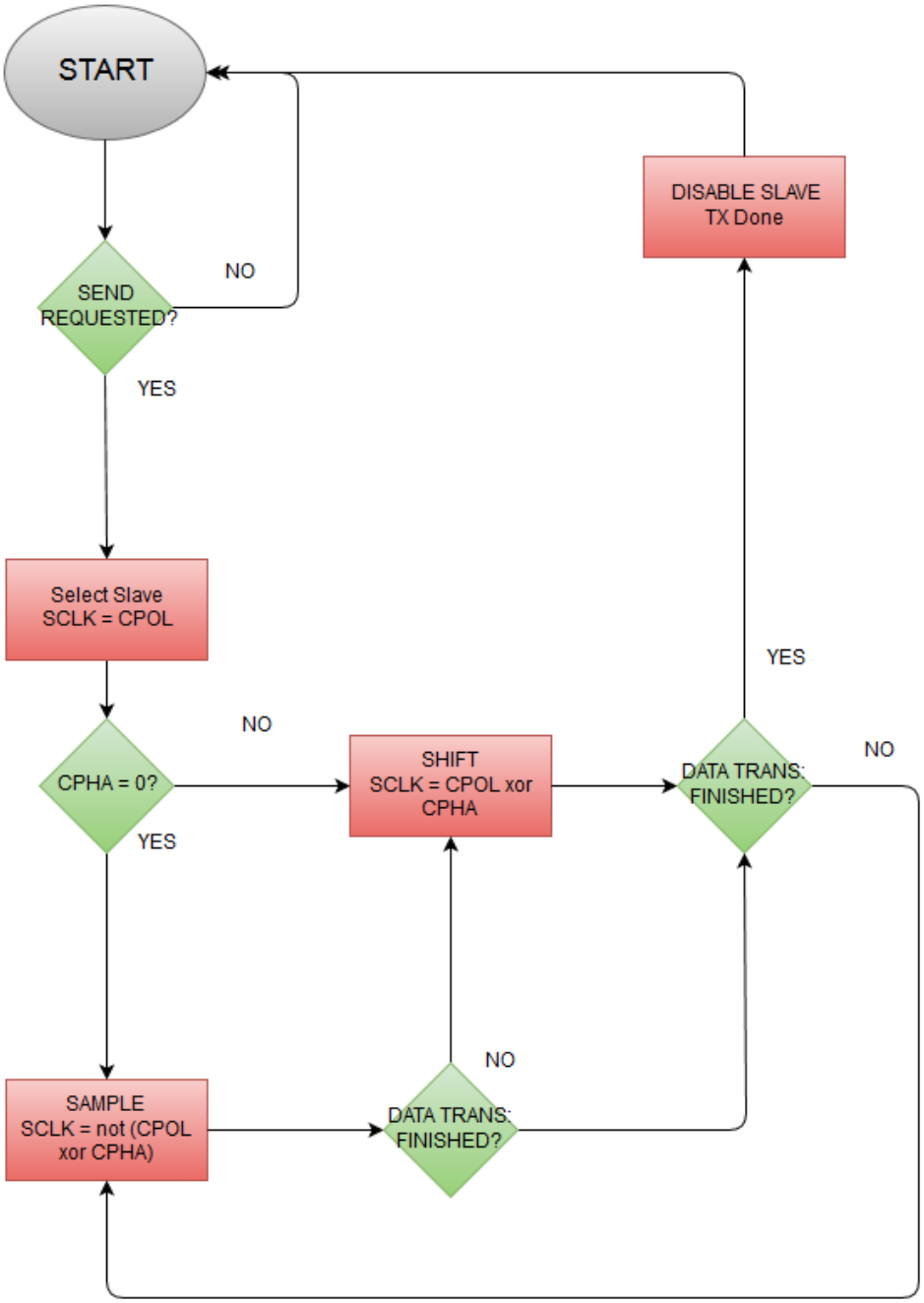


Figure 8.8 - SPI programming algorithm

In Figure 8.8 it is shown the main process. However, the other process takes place in a parallel way when the state machine is sampling or shifting. When it is sampling, the data received is saved in a RX register and when it is sending, the data to send is set by a TX register.

8.4 AD Converter interface

The AD Converter interface can be explained as a box, with some inputs and some outputs, that is used by the *CMDTop* to get the values from each AD Converter.

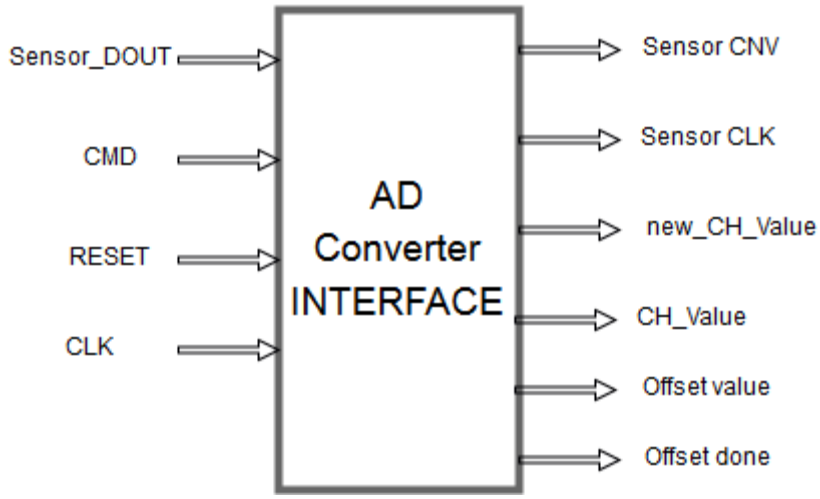


Figure 8.9 - AD Converter interface. [15]

As inputs, it should be necessary to mention these:

- CMD (8 bit vector to select the function to do)
- RESET
- Sensor_DOUT (Value from the sensor, MISO)
- CLK

As outputs, it should be necessary to mention these:

- Sensor_CLK (SPI clock)
- Sensor_CONV (MOSI)
- CH_Value (18 bits vector with measure value)
- New_CH_Value (flag which indicates there is new channel value)
- Offset_Value (18 bits vector with offset value)
- Offset_done (flag which indicates there is new offset value)

The protocol used to communicate with the AD Converter and of course to get the values from it is the SPI protocol, so in a lower level there is the VHDL code of this process. As it has been explained in Chapter 4.1, to start the conversion of the voltage difference between positive and negative input pins of the AD Converter, it is necessary to make an impulse in Sensor_CNV signal. After the conversion has done, at least 19 rising edges of the

SPI clock signal (Sensor_CLK) are required in order to get the 18 bit data conversion through SDO pin (acquisition).

There are four important processes in the interface. The first one is the command management, the second one is the regular measurement and the third one is the measurement with filter. The last process activates the SPI protocol when the flags Read Once, Start Offset or Start Measure are activated in order to get the AD Value.

In the following subchapters the first three will be explained in detail. However, in spite of explaining the processes only for one channel, in the main program four channels are generated. Therefore, the processes are done also in the rest of the channels and it is possible to get the data from the four bridges.

8.4.1 Commands management

This process is really important in order to achieve a correct management of data. Depending on the type of measurement required, there are five input commands which are able to select between these options:

- CMD x"41": Read AD Once
- CMD x"42": Read AD - Offset
- CMD x"43": Offset (with filter)
- CMD x"44": Measure with filter
- CMD x"45": Measure with filter – Offset

It is evident that if the measure is done only one time, the probability of having errors is really high. This is the reason why a filter has been implemented in order to get the value at least 5 times, and then the mean value is the data which takes *CMDTop* to send through IrDA or UART. In Figure 8.10 it is shown the difference between making a measure with or without filter.

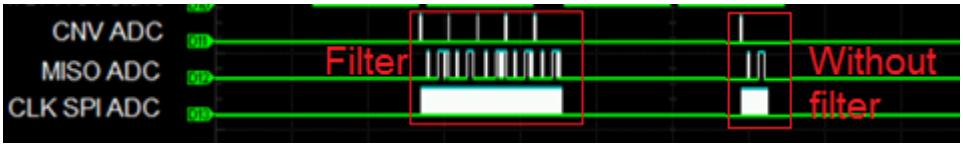


Figure 8.10 - Difference between measuring with/without filter

Depending on the type of measure required (a regular measure or a offset measure), the data is masked using capital letters (A,B,C,D) or lowercase (a,b,c,d), as explained in

Chapter 6. The following flow chart (Figure 8.11) describes the programming of the command management:

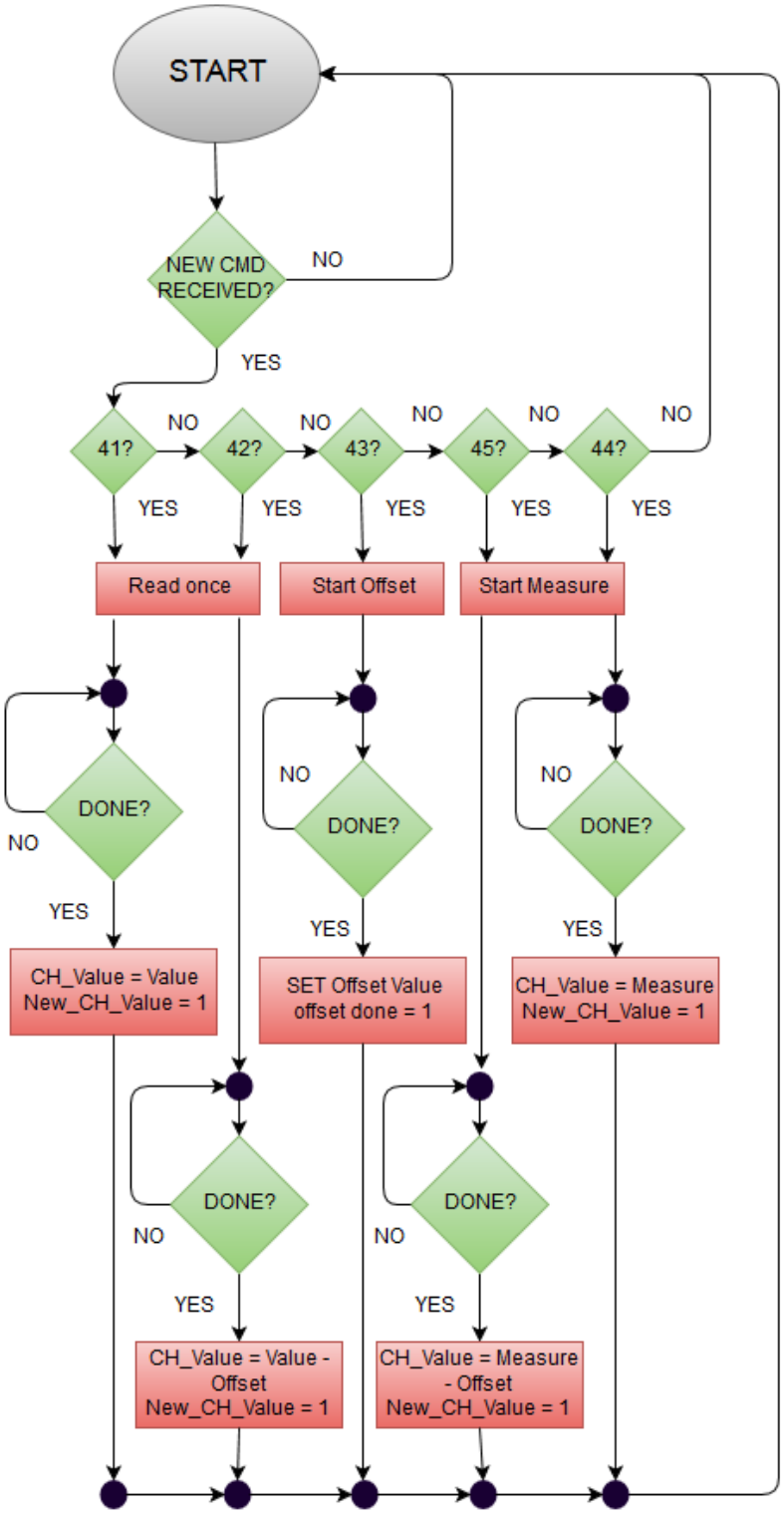


Figure 8.11 - ADC command management programming algorithm

8.4.2 Offset and measure with/without filter

The process required to get the offset and measure with or without filter is the same, the only difference is that when a measure with filter is required, the process is repeated 5 times and the value to send is the mean of the five measures. When an offset value/measure is required, then it waits until the value is available and activates a flag to say it has finished in order to able *CMDTop* knowing when there is a new value. It is described in Figure 7.11:

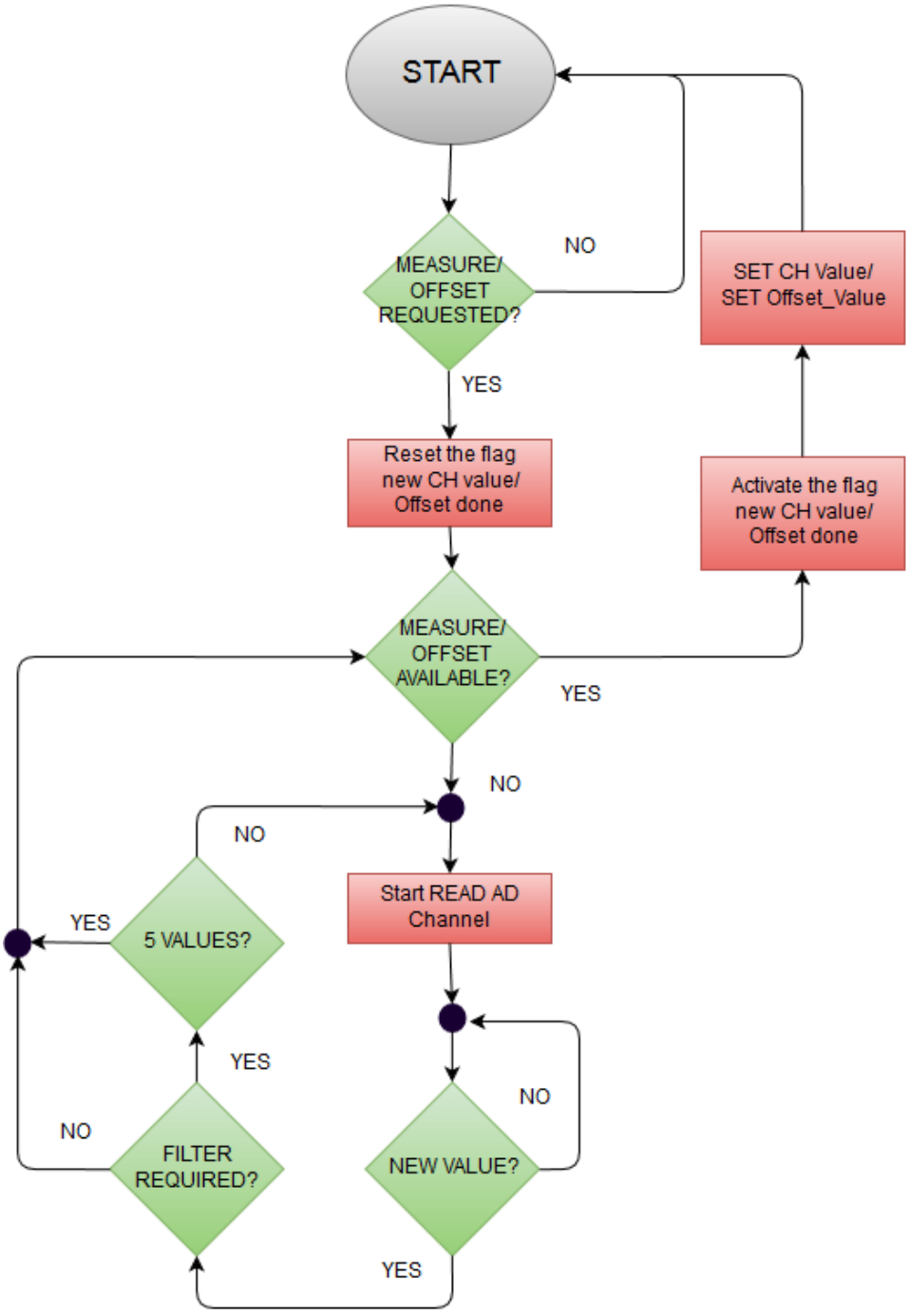


Figure 8.12 - ADC offset and measure programming algorithm

8.5 Shift register interface

The interface of the shift register can be explained using the simile of a box: the inputs are in the left side and the outputs are in the right side, in Figure 8.13. Its function is to set all variables required to send data through the shift register, when *CMDTop* ask for a sending.

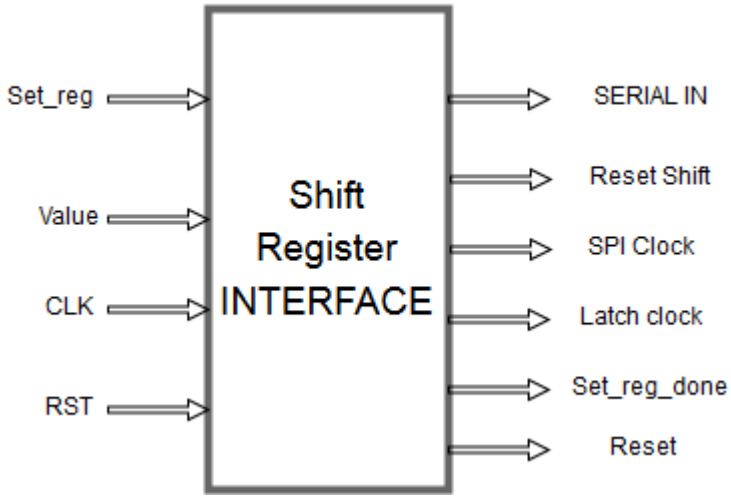


Figure 8.13 - Shift register interface

As inputs, it should be necessary to mention these:

- Value (data to send through the shift register)
- Set_reg (flag which means the desire of the data transmission)
- RST (reset flag)
- CLK (FPGA clock)

As outputs, it should be necessary to mention these:

- Serial_in (data for each shift register)
- SPI clock
- Reset
- Latch_clock
- Output_enable
- Set_reg_done (flag to indicate the latching of data has finished)

In spite of explaining the processes only for one shift register, in the main program four channels are generated. Therefore, the processes are done also in the rest of the shift registers and it is possible to send the data through all of them.

As it has mentioned before, the data transmission uses the SPI protocol, so in a lower level there is the VHDL code of this process. When the *CMDTop* asks for a sending of data, the SPI process starts and when all data is latched, then the output is enabled for a short time and then it goes again to the start. It is described in Figure 8.14:

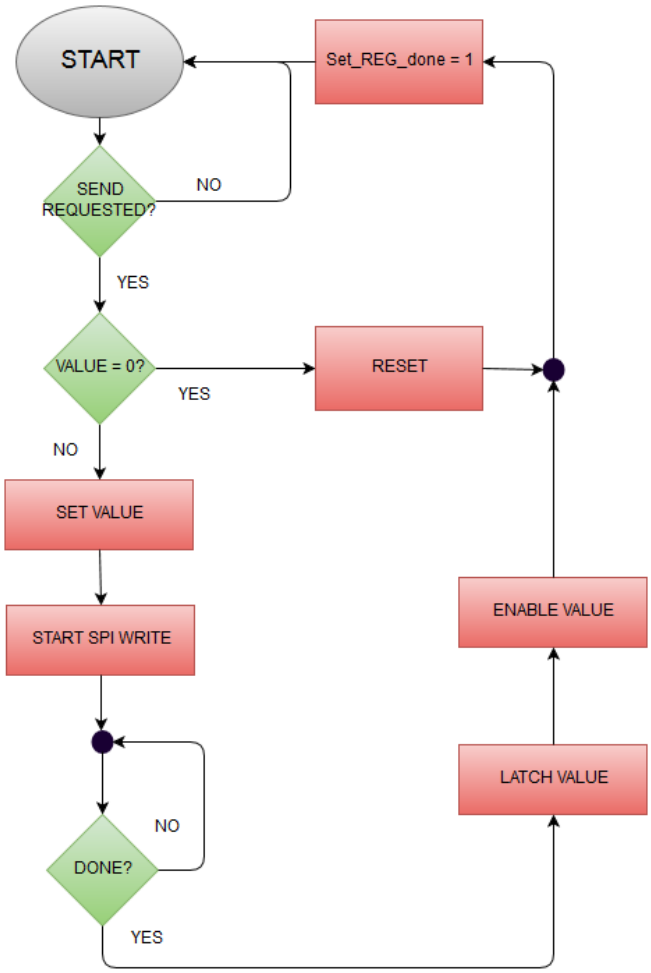


Figure 8.14 - Shift register interface programming algorithm

9 Summary and outlook

In this project it has been developed a wireless electronic system to integrate a new microprobe in the CMM. This evaluation system evaluates four tension values from the Wheatstone Bridges of the microprobe and sends them to the CMM with a frequency of 1 kHz. In the mean time, this electronic allows monitoring in real time the four channels and be controlled by an external computer with a LabVIEW program.

Therefore, it allows the interfacing of the micro probing system without cables, which are not possible to bring at the moment in the machine. According to the final results of the system developed, it is possible to affirm that the objectives of the project have been satisfied. The requirements of the system have been achieved, and also some more functions have been developed in order to improve it even more.

In terms of programming, the wireless transmission has been the most critical part of the development. In order to get the transmission stable and usable, a lot of changes were required in the IrDA interface programming, but finally the result is acceptable.

In spite of having developed a system whose reliability is acceptable, it would be required to improve the system even more in order to achieve a performance to be used in industry. It would be required a more robust data transmission, because now it is needed direct or indirect visual contact between transmitter and receiver and it is not easy to assure this.

For example, instead of developing a wireless data transmission using IrDA modules, in order to achieve a safer and robust data transmission system, it would be interesting to develop a radio frequency data transmission system. Also, if more than one micro probe is required (it means, there are more than one slave FPGA), a RF system will be needed because it is impossible to connect more than two devices using IrDA.

Other improvement that could be interesting is to develop a circuit board for the slave FPGA with less power consumption, so as to require a battery with smaller dimensions and therefore to have more duration of it.

10 Bibliography

1. Allaboutcircuits: **Introduction to Shift Registers** : Shift Registers - Electronics Textbook. Available online at http://www.allaboutcircuits.com/textbook/digital/chpt-12/introduction-to-shift-registers/, checked on 6/3/2016.

2. Analog Devices, Inc. (2014): **AD7982 Datasheet**. In Nicolai Hartmann, Eugene Kelly (Eds.): Aesthetics. Berlin, Boston: DE GRUYTER. Available online at http://www.analog.com/media/en/technical-documentation/data-sheets/AD7982.pdf, checked on 5/27/2016.

3. Ashenden, Peter J. (2002): **The designer's guide to VHDL**. 2nd ed. San Francisco CA: Morgan Kaufmann. Available online at http://ati.ttu.ee/~alsu/VHDL_TUTORIAL_Ashenden.pdf, checked on 5/27/2016.

4. CO, ROHM; LTD: **RPM971-H14 : IrDA Infrared Communication Modules**. Available online at http://www.mouser.com/ds/2/348/rpm971-h14-208858.pdf, checked on 5/19/2016.

5. Digi International: **RF Basics - Digi International**. Available online at http://www.digi.com/resources/standards-and-technologies/rfmodems/rf-basics, checked on 6/3/2016.

6. Eric López Pérez: **SPI Protocol**, checked on 6/3/2016.

7. Guadilla, M.: **Tutorial Eagle**. Available online at https://www.datsi.fi.upm.es/docencia/Micro_C/eagle/tutorial-spa.pdf, checked on 5/27/2016.

8. Hewlett Packard: **The IrDA Standards for High-Speed Infrared Communications**. Available online at http://www.hpl.hp.com/hpjournal/98feb/feb98a2.pdf, checked on 6/3/2016.

9. National Instruments (2010): **Einführung in RF- und Wireless-Kommunikationssysteme - National Instruments**. Available online at http://www.ni.com/tutorial/3541/de/, updated on 1/28/2010, checked on 6/3/2016.

10. National Instruments (2012): **National Instruments VISA - National Instruments**. Available online at https://www.ni.com/visa/, updated on 11/15/2012, checked on 6/3/2016.

11. Nordic Semiconductor: **nRF24l01+ Datasheet**, checked on 6/3/2016.

12. Semiconductors, N. X.P.: **74HC595; 74HCT595 8-bit serial-in, serial or parallel-out shift register with output latches; 3-state**, checked on 5/27/2016.

13. Texas Instruments; Incorporated [SLAP127; *]: **RF Basics, RF for Non-RF Engineers**. Available online at http://www.ti.com/lit/ml/slap127/slap127.pdf, checked on 6/3/2016.

14. Wikipedia (Ed.) (2016a): **LabVIEW** - Wikipedia, the free encyclopedia. Available online at https://en.wikipedia.org/w/index.php?oldid=712826349, updated on 5/13/2016, checked on 6/3/2016.

15. Wikipedia (Ed.) (2016b): **Serial Peripheral Interface**. Available online at https://de.wikipedia.org/w/index.php?oldid=153969517, updated on 5/19/2016, checked on 6/17/2016.

16. Wikipedia (Ed.) (2016c): **Coordinate-measuring machine** - Wikipedia, the free encyclopedia. Available online at https://en.wikipedia.org/w/index.php?oldid=722588394, updated on 5/29/2016, checked on 6/3/2016.

17. Xess: **FPGAs!? Now What?** Available online at http://www.xess.com/static/media/appnotes/FpgasNowWhatBook.pdf, checked on 5/23/2016.

18. Xess: **StickIt Xula 2 Manual**. Available online at http://www.xess.com/static/media/manuals/StickIt-manual-v4_0_1.pdf, checked on 6/7/2016.

19. Xess: **XuLA2 Manual**, checked on 5/27/2016.

20. Yang, Verny: **FT232RL Manual**. Available online at http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf, checked on 5/27/2016.

21. Wikipedia (Ed.) (2016c): **Field Programmable Gate Array** - Wikipedia, the free encyclopedia. Available online at https://es.wikipedia.org/wiki/Field_Programmable_Gate_Array, updated on 5/30/2016, checked on 6/3/2016.

22. Wikipedia (Ed.) (2016c): **Shift Register** - Wikipedia, the free encyclopedia. Available online at https://en.wikipedia.org/wiki/Shift_register, updated on 6/13/2016, checked on 6/3/2016.

23. Carlos E. Canto Quintal: **UART basics**, checked on 6/3/2016.

24. Wikipedia (Ed.) (2016c): **Analog to digital converter** - Wikipedia, the free encyclopedia. Available online at https://en.wikipedia.org/wiki/Analog-to-digital_converter, updated on 6/21/2016, checked on 6/23/2016.

25. Publication bibliography IEEE (2013): **Seventh International Conference on Sensing Technology** (ICST), 2013. 3 - 5 Dec. 2013, Wellington, New Zealand. Piscataway, NJ: IEEE, checked on 6/22/2016.

26. Digilent: **Nexys 4 FPGA Board reference manual**, checked on 5/27/2016.

27. FMF-Uni: **Serial communication RS232** – FMF-Uni. Available online at http://www.fmf.uni-lj.si/~ponikvar/STM32F407%20project/SerialCommunication-RS232.pdf, checked on 6/24/2016.

Appendix A: Figures of all modes of working

Table 1: Signals and channels.

Channel	Signal
D0	USB TX Master
D1	USB RX Master
D2	IrDA TX Master
D3	IrDA RX Master
D4	Serial in (1) Shift Register
D5	Latch Clock Shift Reg
D6	Output Enable Shift Reg
D7	Clock_S (Shift Register)
D8	Strobe
D9	IrDA TX Slave
D10	IrDA RX Slave
D11	Conv (AD Converter)
D12	MISO (AD Converter)
D13	Clock SPI (AD Converter)

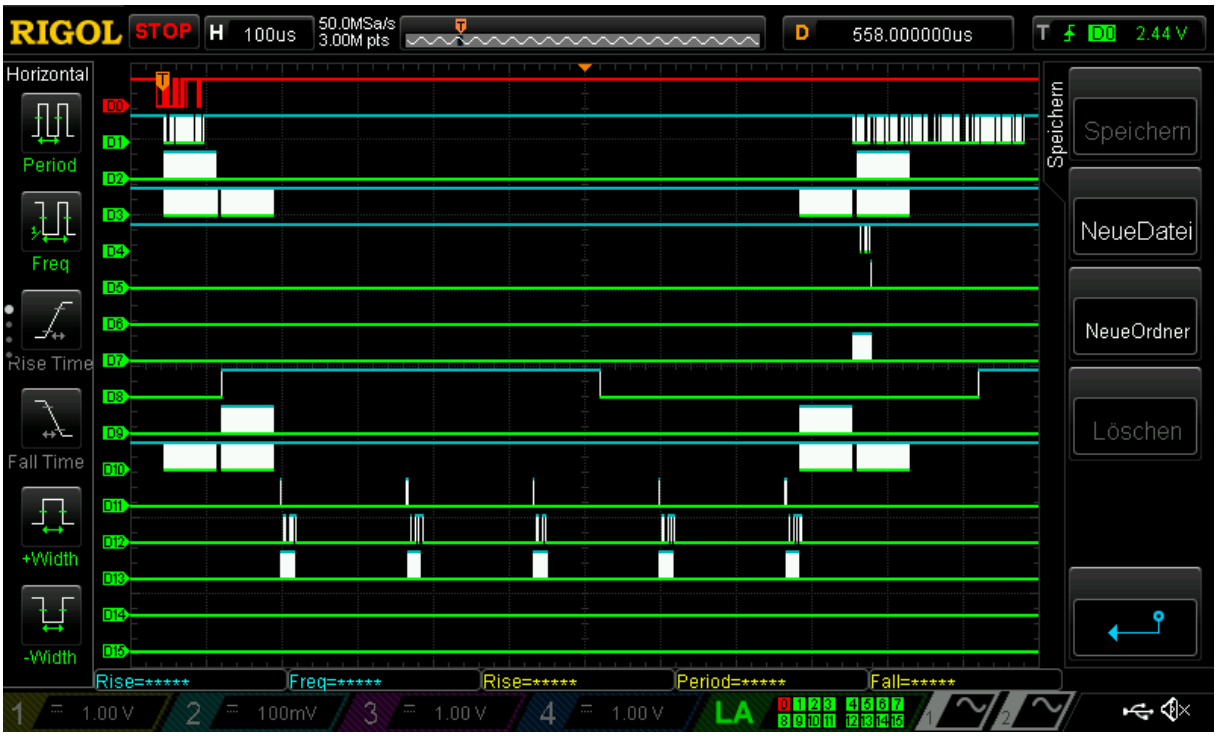


Figure 1. Mode 0 – CMD 20

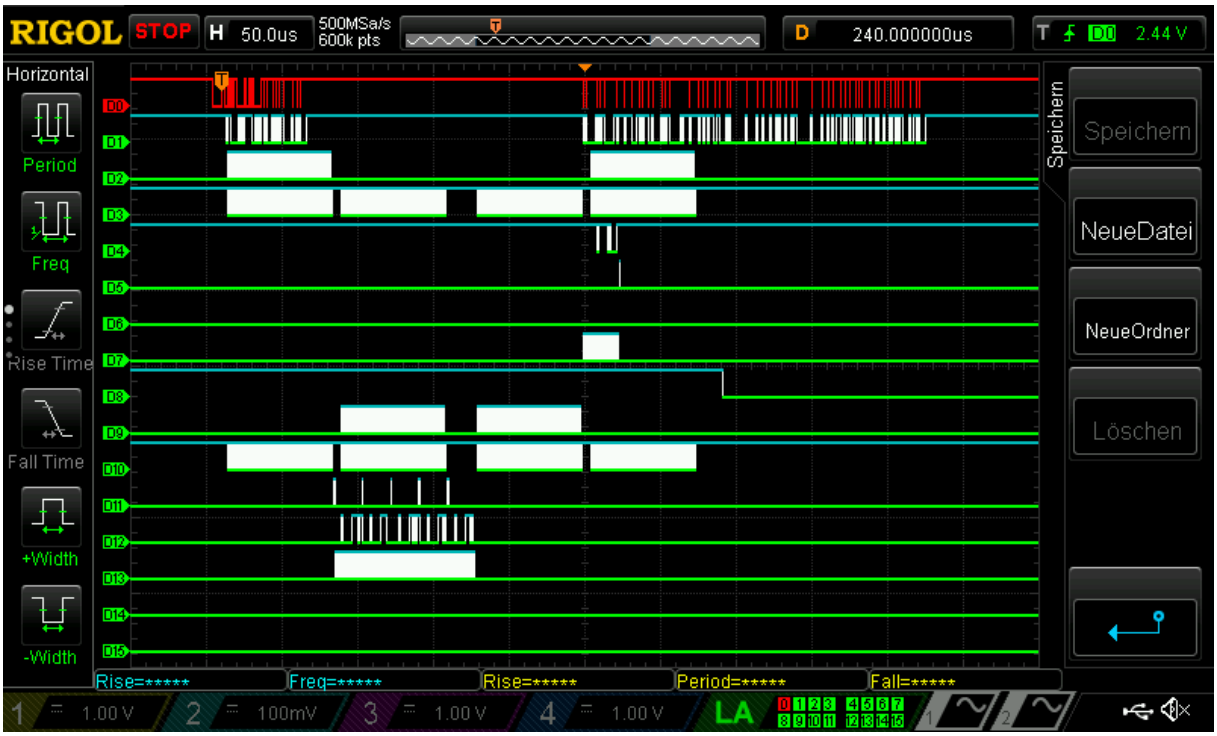


Figure 2. Mode 0 – CMD 30

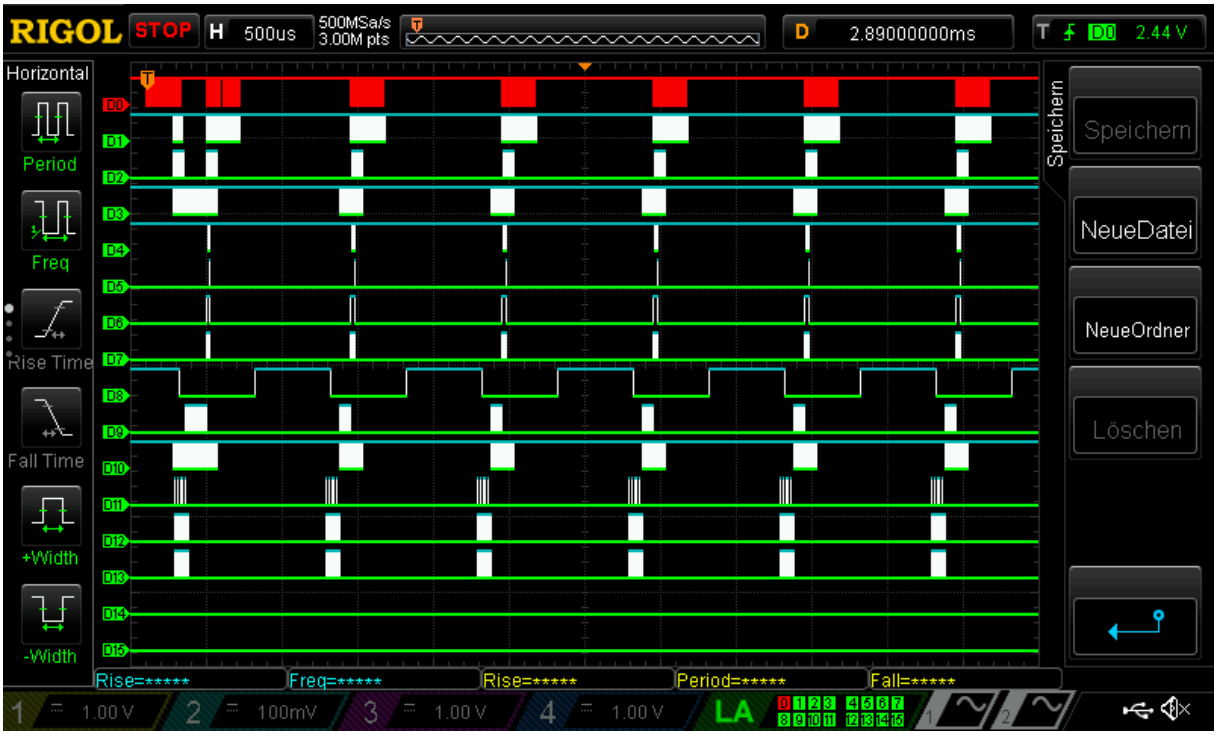


Figure 3. Mode 0 – CMD D0

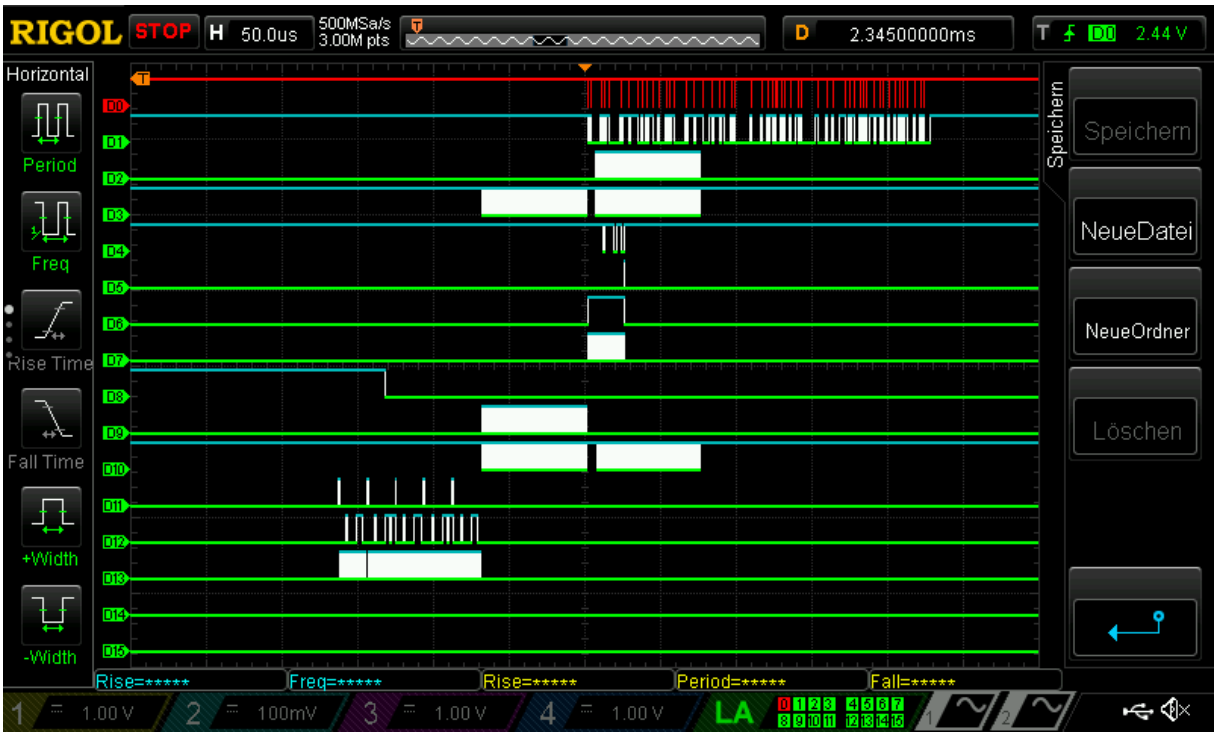


Figure 4. Mode 0 – CMD D0 with detail

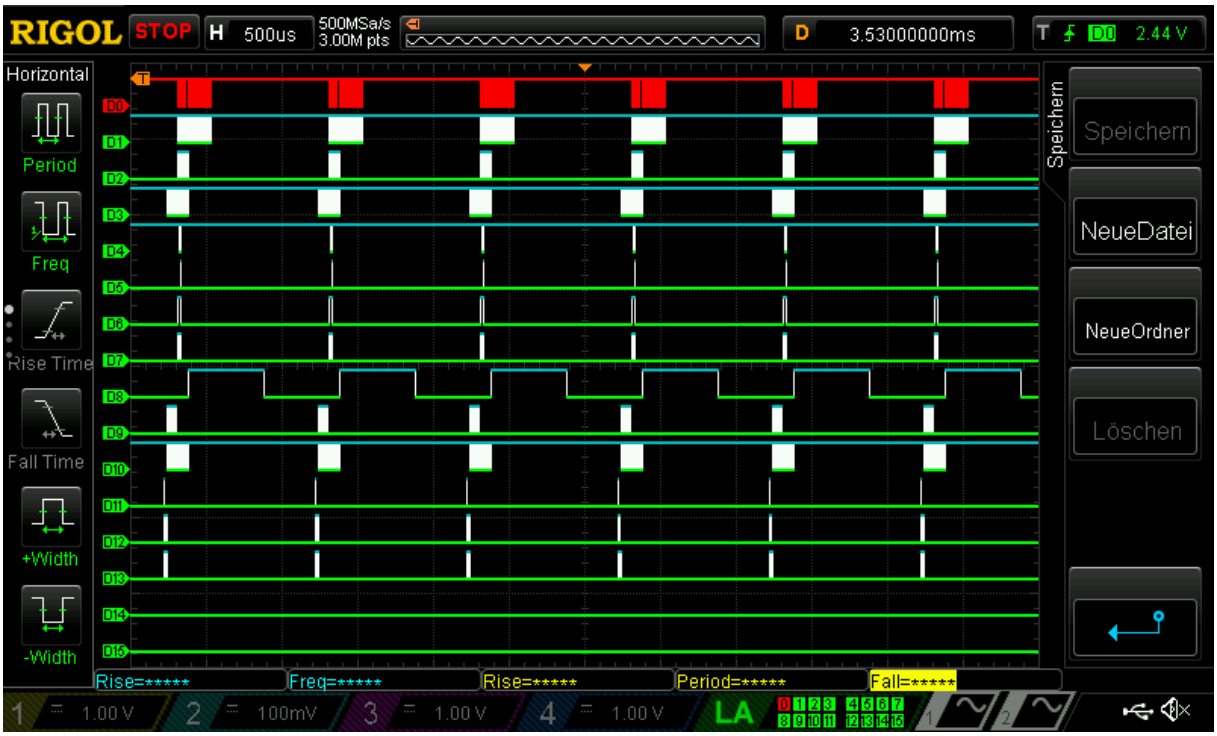


Figure 5. Mode 0 – CMD D1

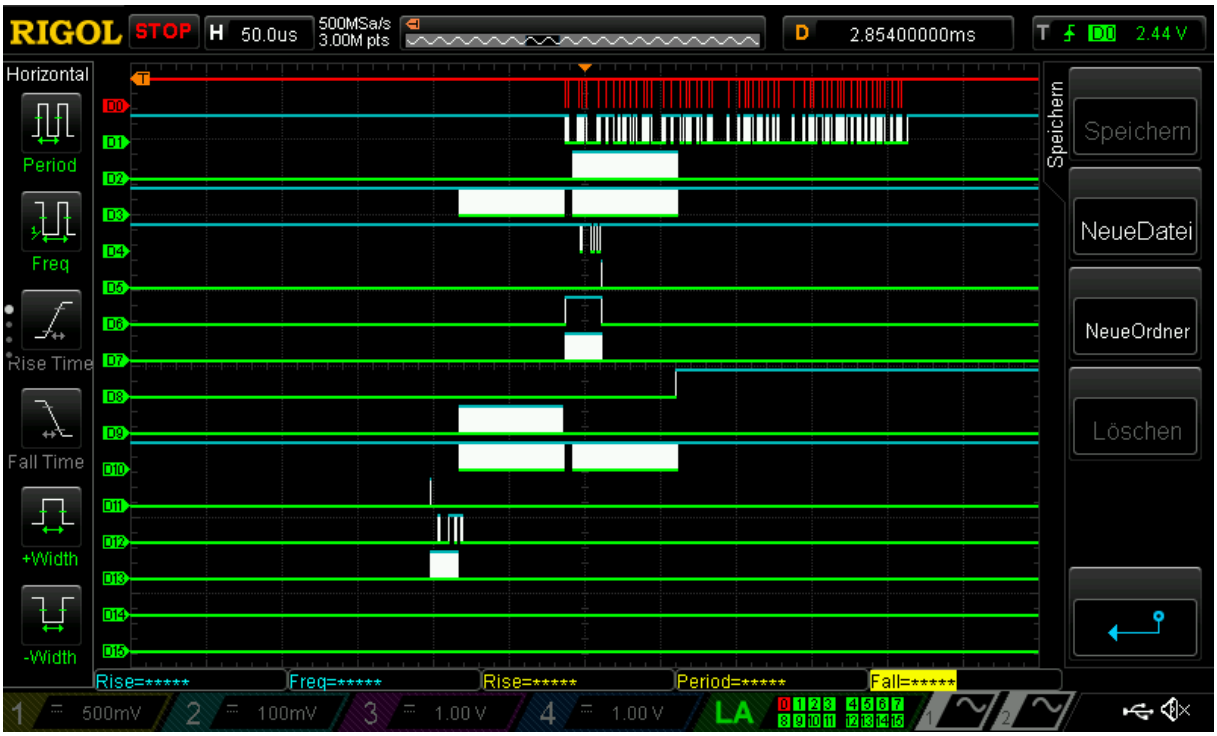


Figure 5. Mode 0 – CMD D1 with detail

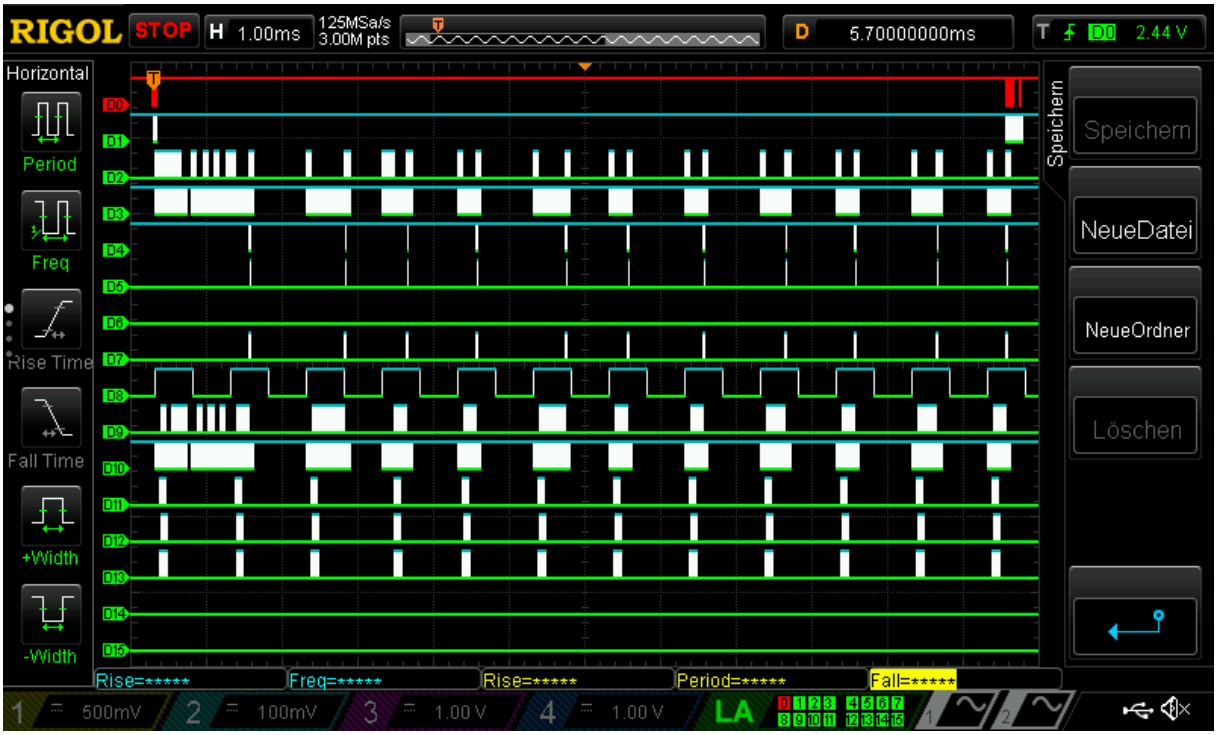


Figure 6. Mode 1 – CMD 30

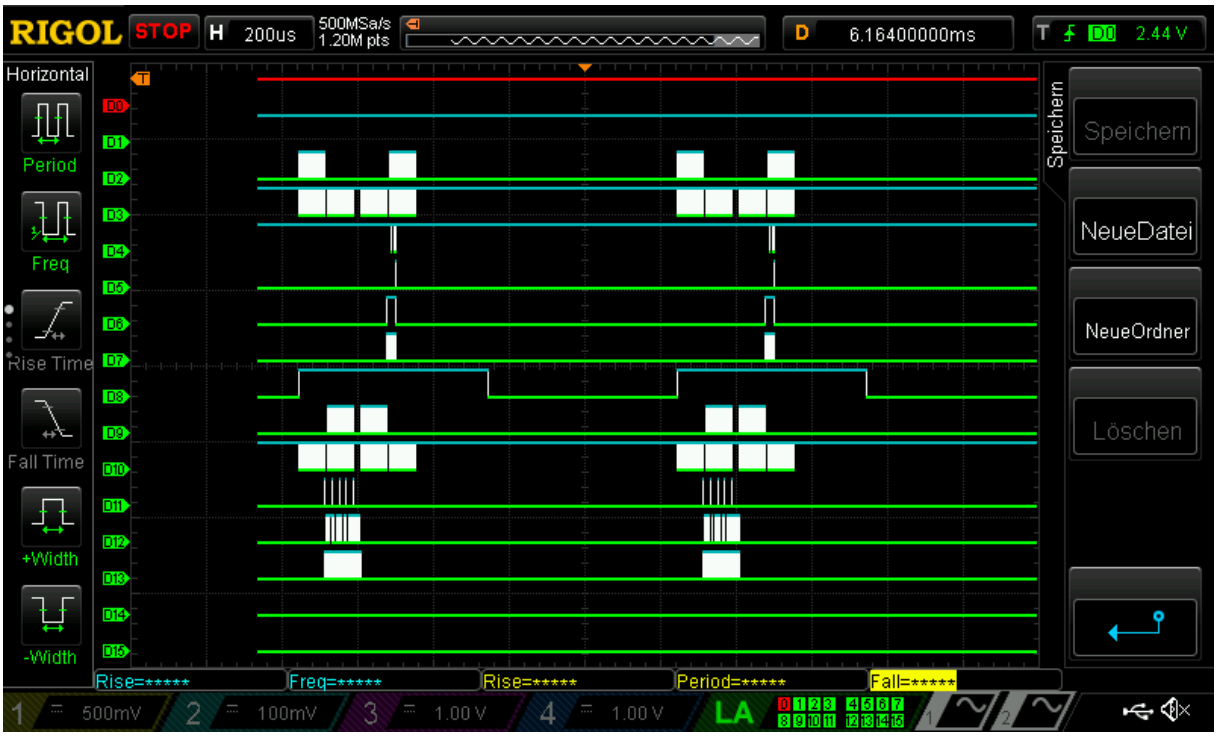


Figure 7. Mode 1 – CMD 30 with detail

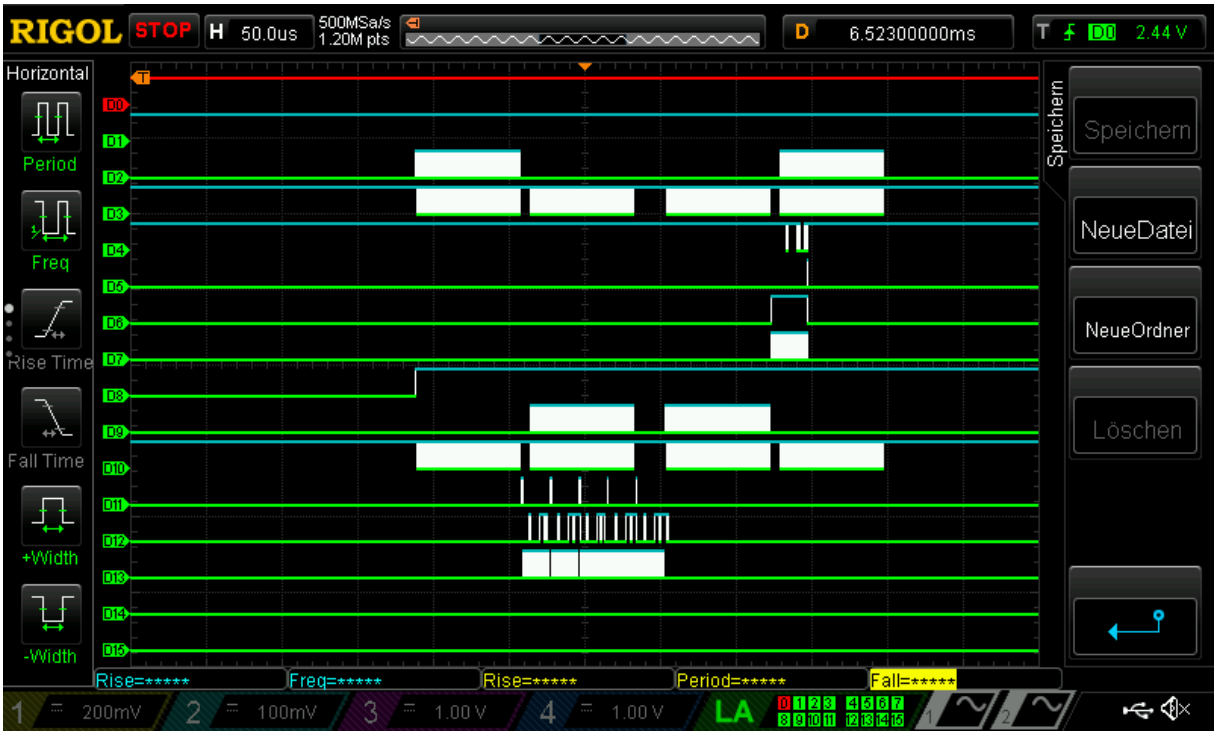


Figure 8. Mode 1 – CMD 30 with more detail

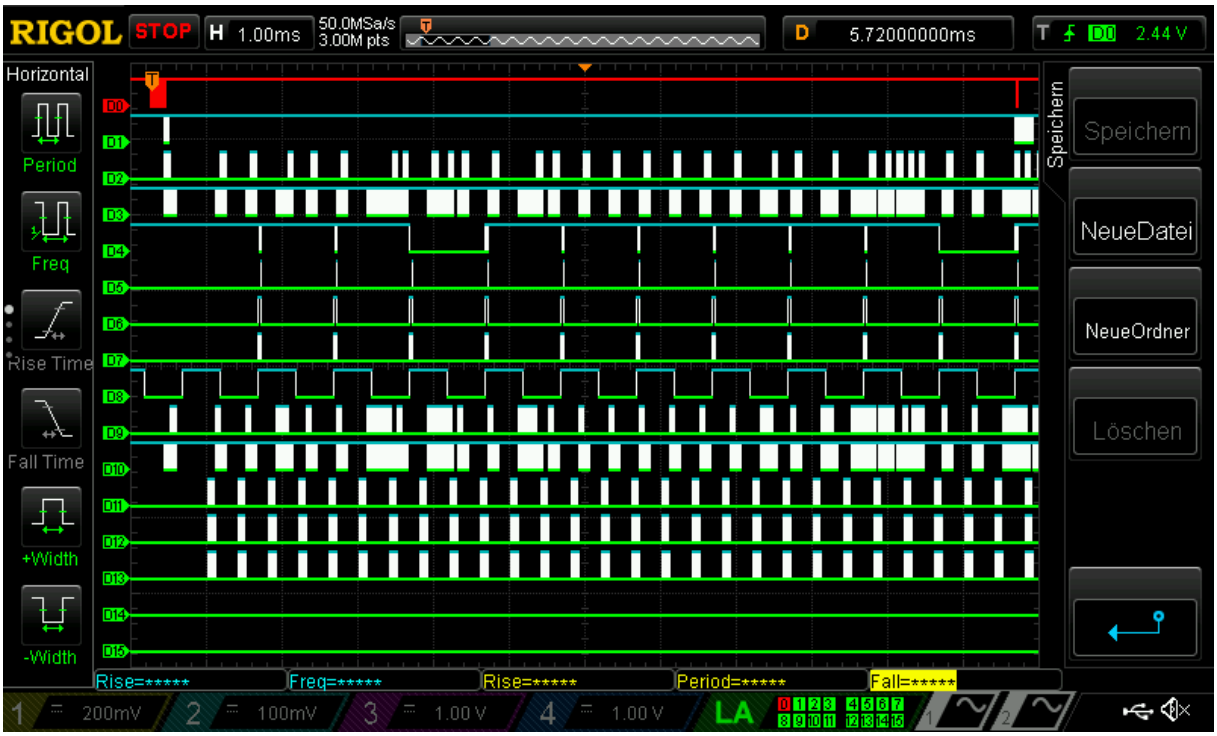


Figure 9. Mode 2 – CMD D0

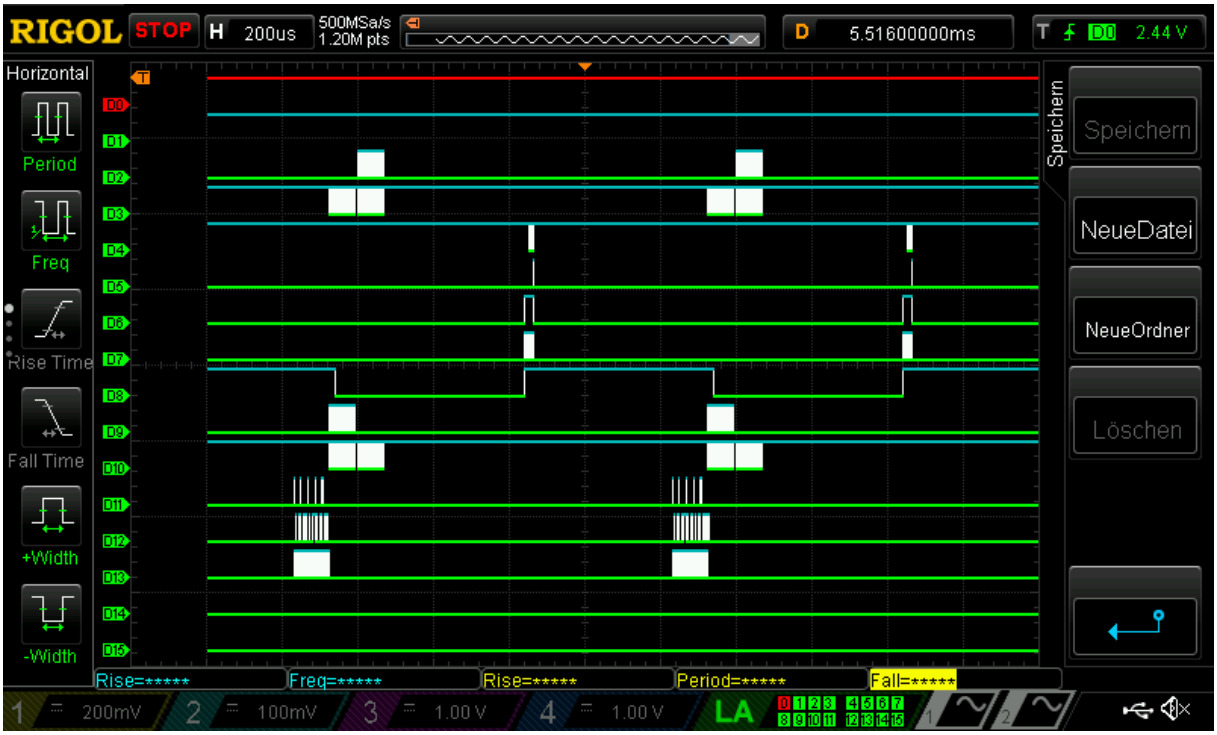


Figure 10. Mode 2 – CMD D0 with detail

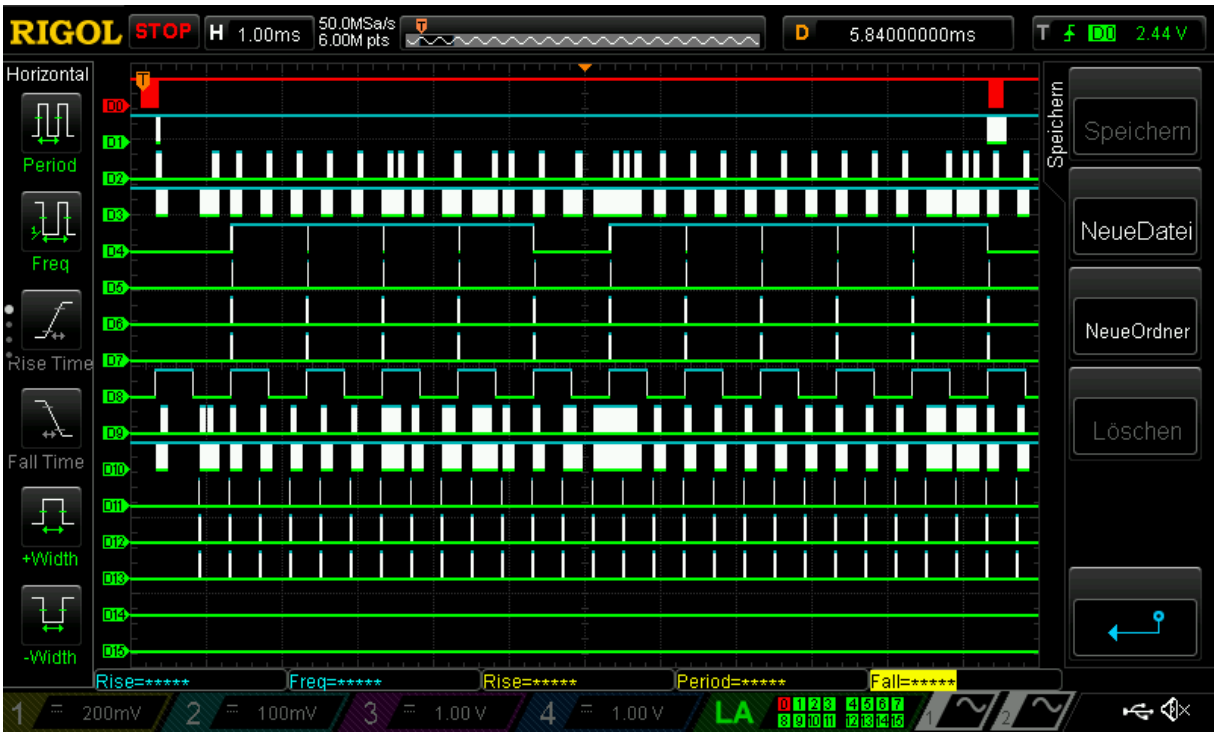


Figure 11. Mode 2 – CMD D1

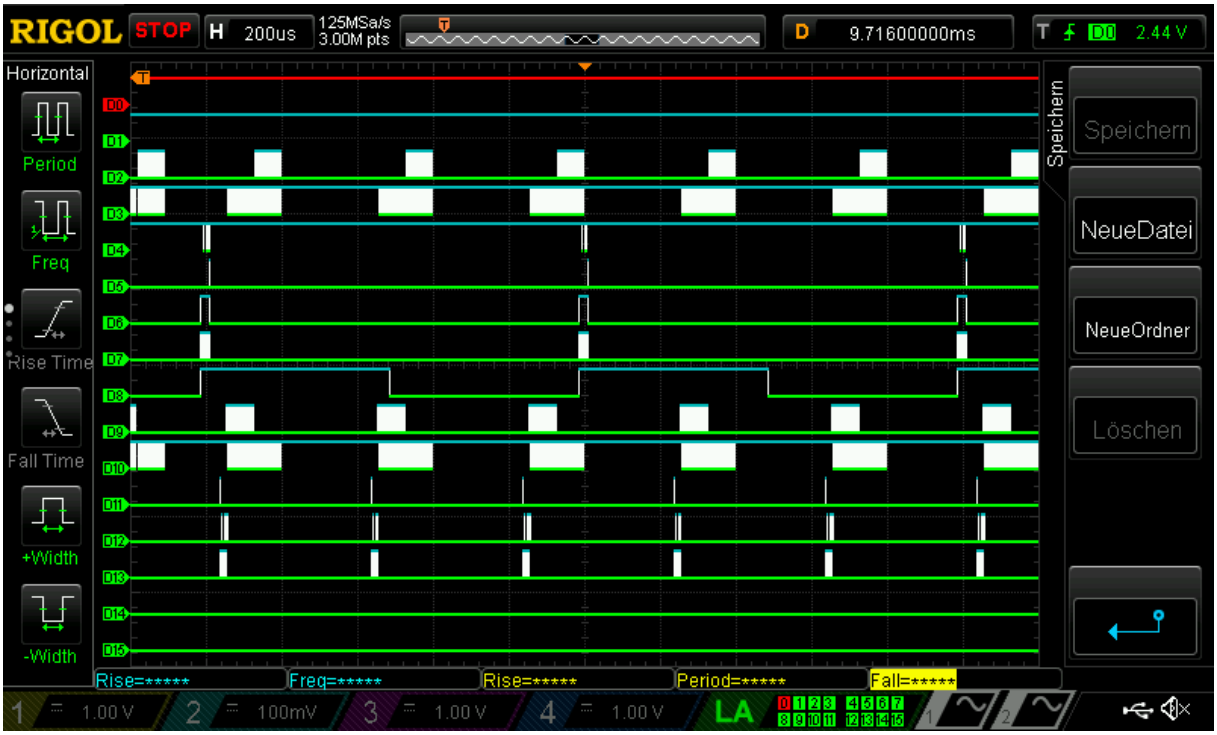


Figure 12. Mode 2 – CMD D1 with detail

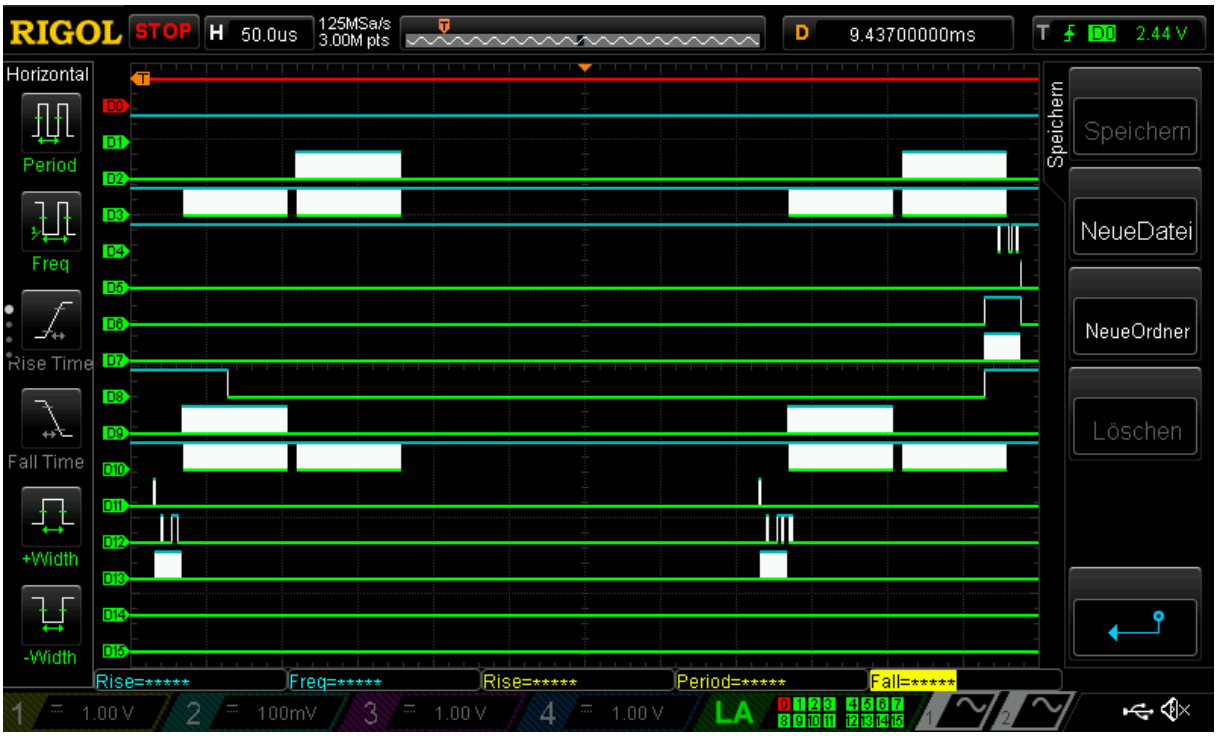


Figure 13. Mode 2 – CMD D1 with more detail

Appendix B: CD

A CD has been attached to the project. On it, it has been included all archives involved in the development of the project. It has been separated in different folders according to their nature:

- Documentation
- Eagle
    - AD Converter
    - IrDA module
    - Shift register
    - USB adaptor
    - USB-UART converter
- Figures
    - Oscilloscope
    - Flow charts
- LabView
- FPGA programming
    - Master
    - Slave