Final Project Report

Universitat Politecnica de Catalunya

Departament d'Enginyeria Electronica

in cooperation with

Technical University of Lodz

Faculty of Electrical, Electronic,
Computer and Control Engineering

Design of a charge injection compensation system for MEMS electrostatic actuators

by

Kamil Karliński

index nr. 131180

Supervisors:

Daniel Fernández, Ph.D.

Jordi Madrenas, Asoc. Prof.

Barcelona 2010

Yesterday is history.
Tomorrow is a mystery.
And today?
Today is a gift.
That's why we call it the present.
~Babatunde Olatunji

To my dear parents,

Moriola and Jan Karlińscy

and my crazy siblings

Mateusz and Kasia

Acknowledgements

Unconditional thanks goes to my supervisors *Jordi Madrenas* and *Daniel Fernández*. First of all for accepting me as their student. Then for all the highly intelligent conversations along the way through this project. For every heavy step in the creative process of this project. And for patience in answering my question and explaining the basic of electronics sometimes. My courage was always increased after a discussion with my supervisors. These conversations are, without any doubt, the main reason that this project is now completed.

Abstract

The main goal of this project is to implement and experimentally verify various control algorithms for MEMS electrostatic actuators. Those algorithms include a charge injection compensation subsystem.

The hardware platform consists of some MEMS electrostatic actuators, a front-end sensing system (comprising a full-custom mixed-signal integrated circuit designed at UPC and a high-speed A/D converter) and a Xilinx FPGA for algorithm coding.

The work involves the following:

- Understanding the working principle of the hardware platform and the issues related to MEMS electrostatic actuators.
- Improve the hardware platform, if required by the control algorithm. This might involve some simple PCB design and test with laboratory instrumentation.
- Algorithm implementation in VHDL. This includes the signal-processing algorithms and also the ancillary routines for I/O data extraction and signal visualization.

Capacitance of MEMS electrostatic actuators were measured under different voltages conditions. It was found that positive voltage stress caused negative charging of the dielectric whereas negative voltage stress caused positive charging of the dielectric. This is consistent with the nature of traps in the silicon oxynitride dielectric used for the switches.

Report is divided into three parts, the first chapter is devoted to the description to the MEMS word. The chapter is devoted a small multiplexing board design description. And finally chapter 3 introduce a development of a VHDL code to control all system.

Contents

Abstract .... **ix**

Contents .... **xi**

1 Introduction **1**

1.1 Overview 1
1.2 Actuators: Transducers with Mechanical Output 1
1.3 Theoretical background 2
1.4 Manufacture process 3
1.5 Characterization of dielectric charging in MEMS 4
1.6 Controlling the charge 5

2 Hardware **7**

2.1 Overview 7
2.2 Delirium-I-feael 7
2.2.1 Jumpers 10
2.2.2 Potentiometers 13
2.2.3 Connections 14
2.3 MEMS actuators board 16
2.4 Multiplexing board 17
2.4.1 General description 17
2.4.2 Operational amplifier 19
2.4.3 Multiplexers 22
2.4.4 Negative voltage calculation 25
2.4.5 Printed circuit boards 25
2.4.6 Jumpers 27

3 Algorithm implementation **29**

3.1 Overview 29
3.2 Codes description 29
3.2.1 ASM chart 29
3.2.2 Clocks in design 31
3.2.3 Important instantiates 33
3.2.4 Averaging as a low-pass filter 35
3.2.5 State machine description 38
3.2.6 Do files 42
3.2.7 Program – user communication 42
3.3 Simulation 44
3.4 Measurements 47

4 Conclusion **51**

5 Bibliography **52**

Appendixes **54**

A – DELIRIUM-I-FEAEL documentation. 54
Schematic........ 54
Reference Design Assembly Drawing 55
B – Multiplexing board documentation 56
Schematic........ 56
Reference Design Assembly Drawing 57
Bills of materials 58
C – A MEMS actuators floor-plan........ 59
D – Source codes 60
MEMS_ChargeInReduce.vhd – top level 60
dsp.vhd 66
misc.vhd........ 69
Display.vhd 70
TB_MEMS_ChargeInReduce.vhd 72
sim.do 74

1 Introduction

1.1 Overview

In this chapter there will be briefly presented a theoretical background of MEMS electrostatic actuators.

1.2 Actuators: Transducers with Mechanical Output

An actuator is a device that converts energy from one form, such as electrical, mechanical, thermal, magnetic, chemical, and radiation energy, into the mechanical form. [12] (For example, resistive heating elements convert electrical energy into bending of a bimorph microstructure.) In some cases, a microactuator may convert energy into intermediate forms before resulting in the final mechanical output (such as inductively coupled heating elements, which convert electrical energy first into magnetic energy before finally resulting in thermal energy serving to deflect a microelement). Such devices are said to be based on tandem transduction [17].

Table 1.1 Transduction Methods

	Output signal	
Input signal	**Mechanical**	**Electrical**
Mechanical	Fluidics, acoustics	Piezoresistive
Electrical	Electrostatics, electromagnetic	Langmuir probe, transformer
Thermal	Thermal expansion	Pyroelectric
Magnetic	Magnetometer	Magnetoresistance
Chemical	ChemAbsorber	Ionization, ChemFET, ChemResistor

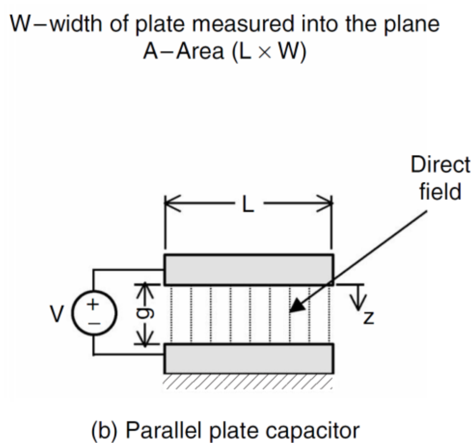
Electrical microactuators are by far the most common and diverse type of microactuator. This is primarily because of the ease with which most electrical microactuators can be produced using conventional microfabrication processes and materials. Examples of electric microactuators include static, resonant, rotary, and stepper-motor configurations. Electrical microactuators can be driven by an electrical-to-mechanical conversion that makes use either of the direct electrostatic forces between charged objects or a piezoelectric material that can mediate the

energy transformation. Electrostatic and piezoelectric transduction mechanisms, the physical relationships involved, the material properties that govern their operation.

1.3 Theoretical background

Electrostatic forces are commonly observed even on the macroscale during everyday activities, such as combing dry hair or drying clothes. The discovery of elemental charge and the concept of electric fields have led to an understanding of electrostatic energy and the forces acting between objects charged to a different extent.[1]

In the literature there are a lot of equations which allow to calculate a magnetic fields, and forces between plate as well [6] [12]. All of them base on well know Gauss's and Maxwell's laws. Important thing in the actuators analysis is amount of energy that is require to charge a plate and move them in any other position. Because in the project there will be only consider the effect of charge injection into between plate space, only derivation of accumulated charge will be given.



(b) Parallel plate capacitor

Figure 1.1: Generic electrostatic actuator

The capacitance of the generic structure shown in Figure 1.1 is given by:

$$C = \frac{\varepsilon_r \varepsilon_0 A}{g} \quad (1.1)$$

where
ε = permittivity of material between the parallel plates
(free space permittivity $8.85 \cdot 10^{-12}$ F/M)
A = plate area
g = gap between the plate

For a variable parallel plate capacitor, the movable plate moves normally to the fixed plate as defined by the coordinate, z. The capacitance for the movable capacitor is

$$C = \frac{\varepsilon_r \varepsilon_0 A}{g - z} \tag{1.2}$$

The charge Q on each plate is proportional to the capacitance C and the voltage V:

$$Q = CV \tag{1.3}$$

The equations are given only as to preview, and remind basics, next section focus on manufacture process [15].

1.4 Manufacture process

Despite the attractive performance characteristics of MEMS actuators, their commercialization is difficult because of a number of factors such as reliability and packaging cost. In particular, there can be affected by dielectric charging effects such as control-voltage drift and stiction [10]. The dielectric is typically silicon dioxide or nitride formed by plasma-enhanced chemical vapor deposition (PECVD) with a high density (~10^{18}/cm_3) of traps associated with silicon dangling bonds [11]. The board contain a MEMS electrostatic actuator which is depicture in the Figure 1.2 (and in Appendix C – A MEMS actuators floor-plan.) was next manufactured in the Multi-User MEMS Processes, or MUMPs [9]. The main purpose of this board it to provide a lot of MEMS actuators with different capacitance and dynamic characteristic.

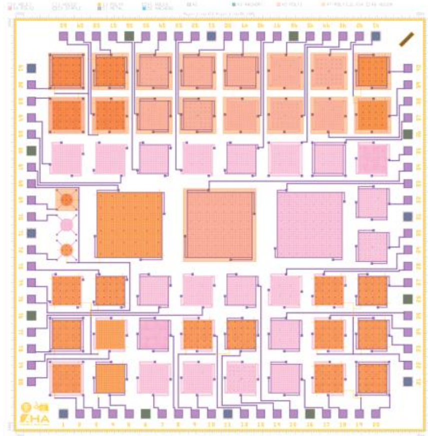


Figure 1.2: MEMS actuator chip layout.

The MUMPs process is a commercial program that provides cost-effective, proof-of concept MEMS fabrication to industry, universities, and government worldwide. MEMSCAP offers three

standard processes as part of the MUMPs program: PolyMUMPs, a three-layer polysilicon surface micromachining process: MetalMUMPs, an electroplated nickel process; and SOIMUMPS, a silicon-on insulator micromachining process.

The PolyMUMPs process is a three-layer polysilicon surface micromachining process derived from work performed at the Berkeley Sensors and Actuators Center (BSAC) at the University of California in the late 80's and early 90's. Several modifications and enhancements have been made to increase the flexibility and versatility of the process for the multi-user environment.

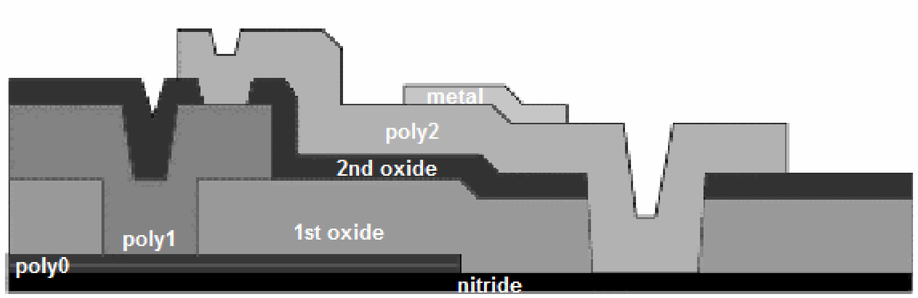


Figure 1.3: PolyMUMPs process cross section

Figure 1.3 is a cross section of the three-layer polysilicon surface micromachining PolyMUMPs process. This process has the general features of a standard surface micromachining process: (1) polysilicon is used as the structural material, (2) deposited oxide (PSG) is used as the sacrificial layer, and silicon nitride is used as electrical isolation between the polysilicon and the substrate [9]. The MEMS board was manufactured in PolyMUMPs technological process.

Knowing basic of manufactures process it is time to explain how the MEMS actuators are built.

1.5 Characterization of dielectric charging in MEMS

Figure 1.4 shows a schematic representation of an MEMS. The switch consists of two electrodes, of which the top electrode is suspended by tiny springs. The top electrode can be pulled down by applying a voltage across the air gap between the two electrodes. Above a certain voltage, the balance between the attracting electrostatic force and restoring spring force becomes unstable and the switch closes.

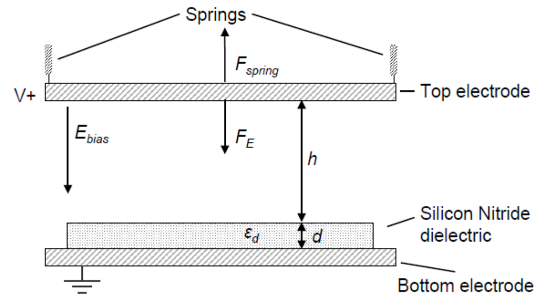


Figure 1.4: Schematic representation of an MEMS

The top electrode of a parallel plate capacitor can be pulled down by applying a voltage greater than the pull-in voltage, which is pulled up again by the springs if the voltage is lowered beneath the pull-out voltage.[10]

Figure 1.5 shows how charges are trapped into the MEMS structure.

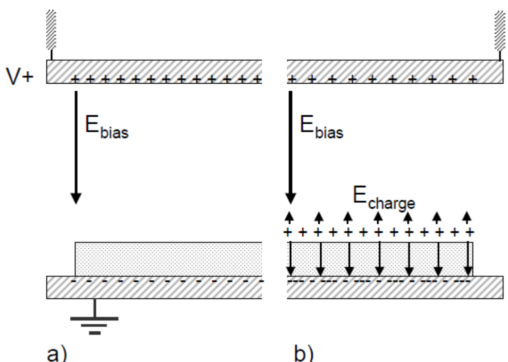


Figure 1.5: E-field in a parallel plate capacitor. a) No fixed charges in the dielectric. b) Fixed surface charge at height z = d.

The traps are amphoteric [10] so they can be either positively or negatively charged. During switch operation, the electric field across the dielectric can be of the order of 10^6 V/cm causing electrons or holes to be injected into the dielectric and become trapped. With repeated operation, charge gradually builds up in the dielectric resulting in control-voltage drift or stiction.

1.6 Controlling the charge

A graph in Figure 1.6 presents how charge is controlled by external source. At the beginning, after turn ON a power supply, there is no any loads inside. Loads pass through the electrode and nested in the space between the plates during the time. This process is very slow and depends on MEMS actuator. Some of them doesn't have such a strong injection. After changing a sequence, which means reverse a divider supply polarity, charge from previous mode is compensated at

first. When actuator is totally discharge, the reverse polarity charging process begin. The list below and figures describe this process more clear from the beginning.

1. after power supply turn ON (t = 0).
2. after short while working in sequence I.
3. while after changing a sequence
4. } presently reside in the sequence II
5. }
6. while after changing a sequence again to I

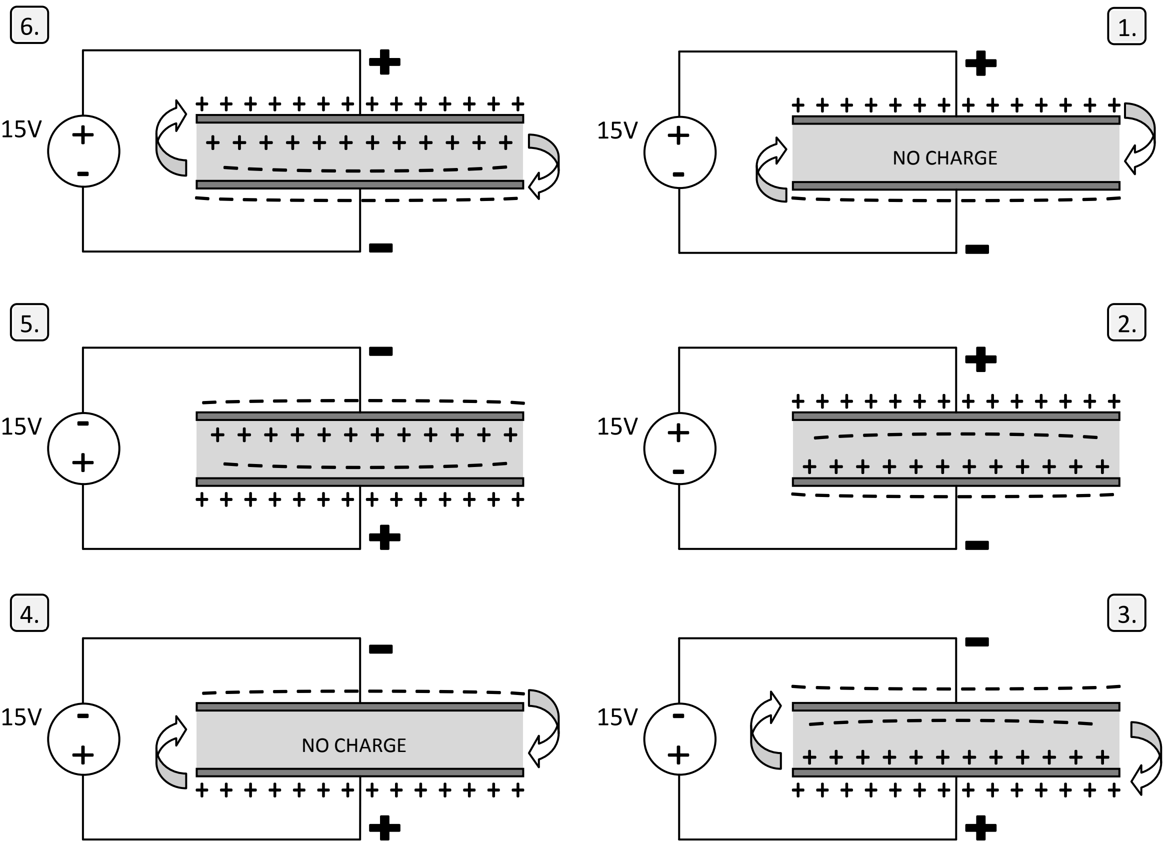


Figure 1.6: The charge injection effect

Presented train of thoughts in this section (power supply changes - as charge compensation) was carried out by multiplexers - as part of a executive hardware and state machine as system controlling part. The following chapter describe the first one.

2 Hardware

2.1 Overview

This chapter presents a hardware part of project. Here will be description of boards which were used to test a MEMS chip, and explanation how they were designed. In this chapter there will be shown how to get started with each boards and how to properly configure them. Whole project consist of three printed circuit boards and one Xilinx Spartan-3 Starter Kit [19].

2.2 Delirium-I-feael

The board was designed by Sergi Gorreta under Daniel Fernandez supervision. It contain a main integrated circuit responsible for generate appropriate signals for measuring the actuator capacitance. The board provides three voltage regulators separated for each power supply, analog to digital converter with differential driver and a big amount of jumpers and potentiometers for tuning and mode selection. Figure 2.1 shows a simple functional block diagram, but the Figure 2.2 their schematic.

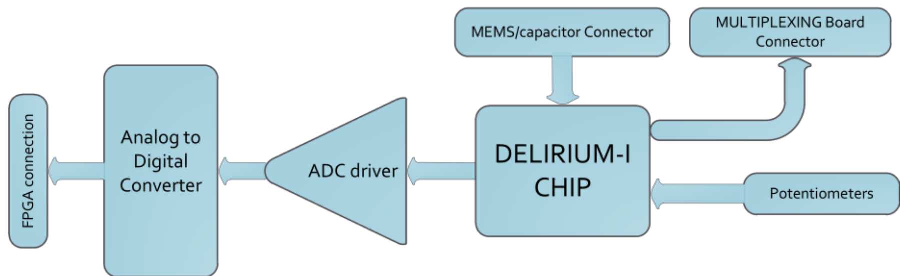


Figure 2.1: Simple functional block diagram

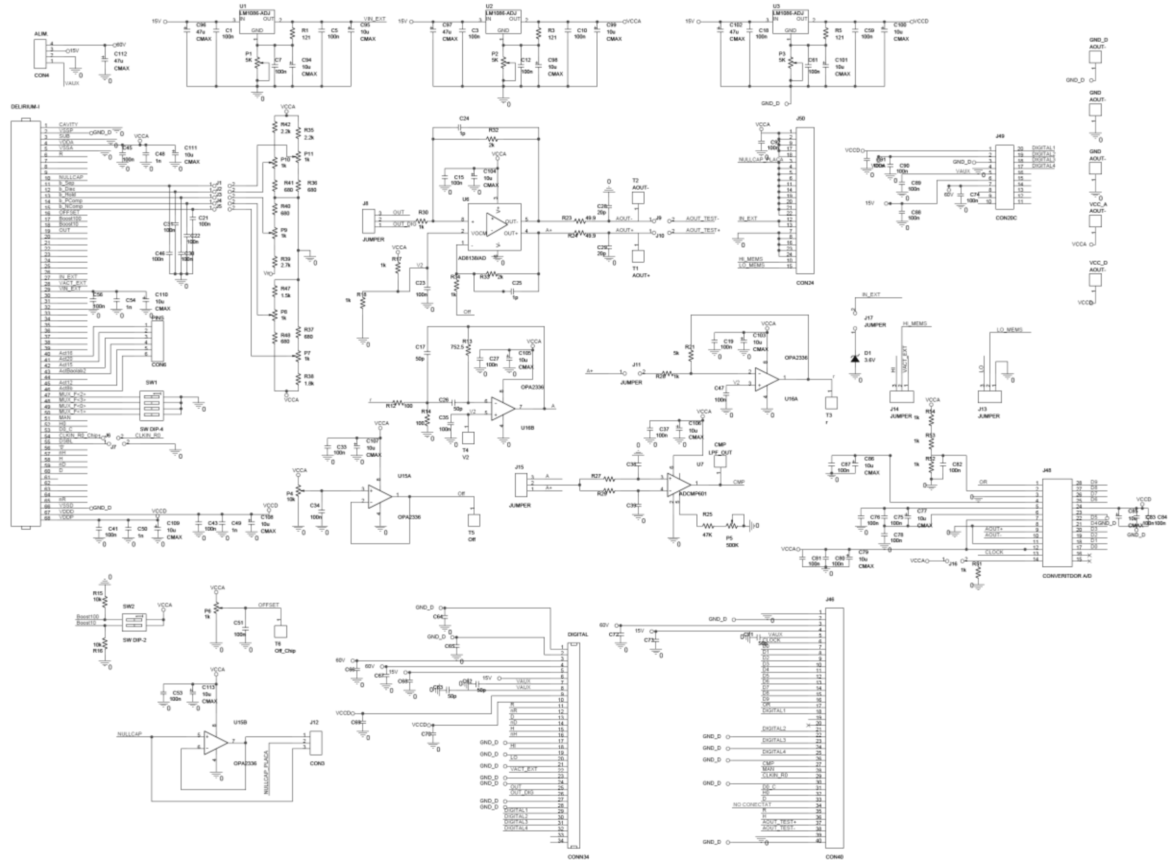


Figure 2.2 Delirium-I-FEAEL schematic

Briefly explaining the operation of this board, it should be noted that its heart is the Delirium-I chip, responsible for generating a H, D, and R control signals. The functional diagram of this chip is depictured in the Figure 2.3.

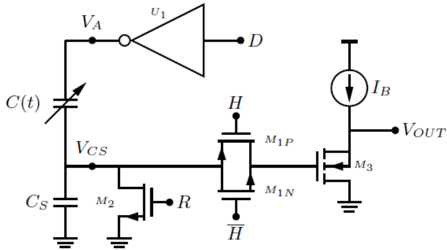


Figure 2.3 Simplified schematic of Delirium-I chip.

The circuit is better understood by analyzing the running sequence of the digital control signals D, R and H. This sequence can be summarized as follows [7]:

- In normal operation, signals D, R and H are equal to 0. In this mode, a high voltage is applied to theupper node of the actuator though the U_1 inverter, the switch M_2 is turned off and the transmission gateM_1 is turned on. Thereby, the output voltage V_{OUT} tracks V_{CS} by means of the PMOS voltage followerM_3.

- When a reset is required, the signal H is set to logic 1 (and H to 0). This disconnects the transmission gate M1 so the output voltage VOUT is hold constant regardless of VCS.
- Then, signal R is set to 1, discharging the serial capacitor Cs.
- Next, signal D is set to 1, discharging the electrostatic actuator.
- After both capacitances have been discharged, R is set back to 0 and the capacitances can be charged again.
- Finally signal D is set back to 0, charging the capacitive divider and, after a short time, signal H is set also to zero, enabling the transmission gate, so the output voltage tracks again V_{CS} and the sequence is repeated.

Figure 2.3 shows V_{OUT} voltage which provide the current MEMS actuator capacitance. This information is pass through to fully differential amplifier, where it is filtrated and a little bit gained. Then it land in high speed analog to digital converter, and in digital way goes to FPGA circuit, where it is subjected final signal processing. There is also feedback in this circuit, after analyzing it, the FPGA could impact on system operation. This device have control of main chip, and divide supply duty cycle. Figure 2.4 shows the layout of main board and next few section describe their proper configuration, started with jumper.

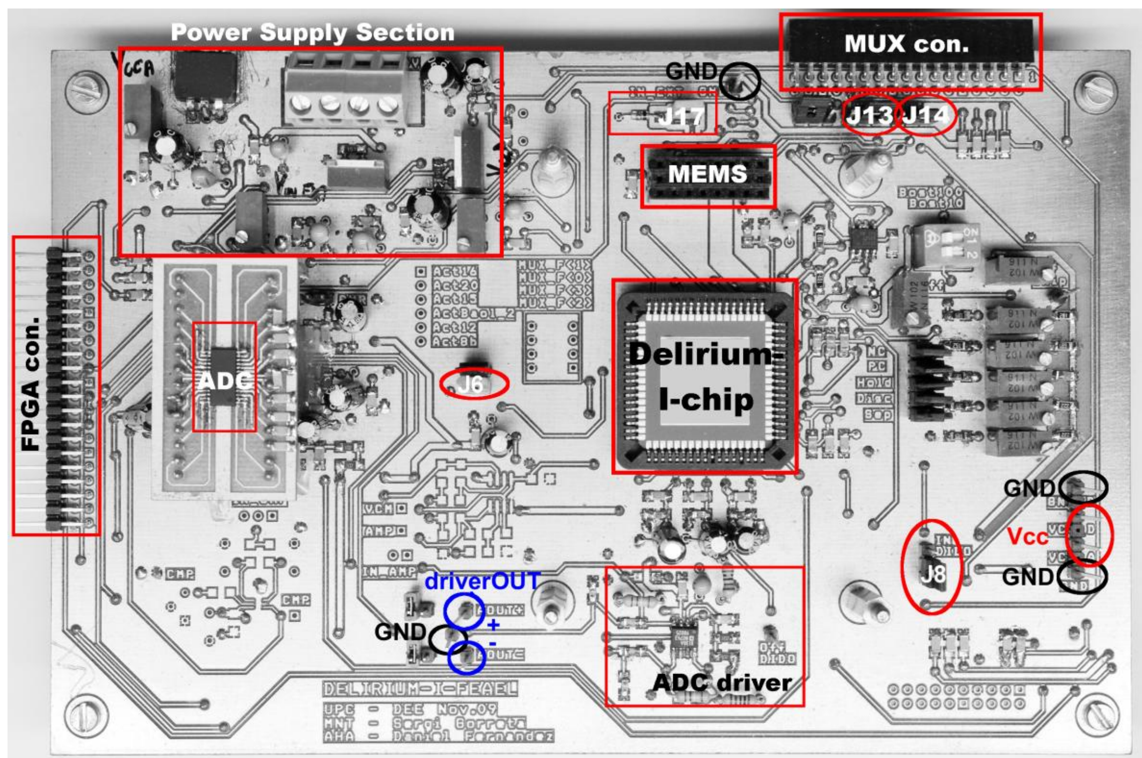


Figure 2.4: DELIRIUM-I-FEAEL board.

2.2.1 Jumpers

The most important jumper in this circuit is J17, the main objective of it is to prevent the burning of chip. It connects 3.6V Zener diode to a middle node of MEMS system, as depicted in Figure 2.5. That jumper should always be in place during making changes in the fixed capacitor or actuator. Single glitches or short overvoltages, can occur and cause chip damage.

WARNING

Before making ANY changes with capacitance divider make sure that jumper is closed and power supply is off. After work, also leave the jumper in that position.

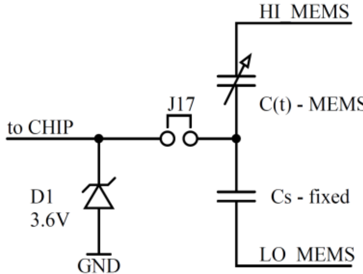


Figure 2.5: Jumper J17 application

Jumper J13 and J14 are used for voltage selection for supply a capacitive divider made up of fixed capacitor and MEMS actuator. All possibilities that can be chosen with the help of this jumper are shown in Table 2.1.

Option	Jumper J13 Setting	Jumper J14 Setting	Description
High voltage			Dependent on selected mode: **MODE_I** (positive) HI_MEMS: +15V / 0v LO_MEMS: always grounded. **MODE_II** (negative) HI_MEMS: 0v / -11.7V LO_MEMS: 3.3V / 0v
Low voltage			Irrespective of selected mode: HI_MEMS: 3.3V / 0v LO_MEMS: always grounded.
DAMAGE			This configuration may cause a damage to Delirium-I chip in mode_II. The chip is able to work only from 0 to 3.3V voltage range. In mode_II middle point of capacitive divider provide a negative voltage.
0V_out			In this configuration a capacitive divider out is always equal to ground.

Table 2.1: Capacitive divider voltage configuration

In the Table 2.1 there is one configuration with can cause a damage to circuit. It is because Delirium-I chip was designed to work only with positive voltage. In this case there is a possibility when HI_MEMS is active with -11.7V, LO_MEMS is grounded and middle point of divider goes into negative voltage.

Another jumper in that circuit is marked as J6. It can be used to disconnect Delirium chip's main clock. It is also possible to connect an external clock as showed in the picture.

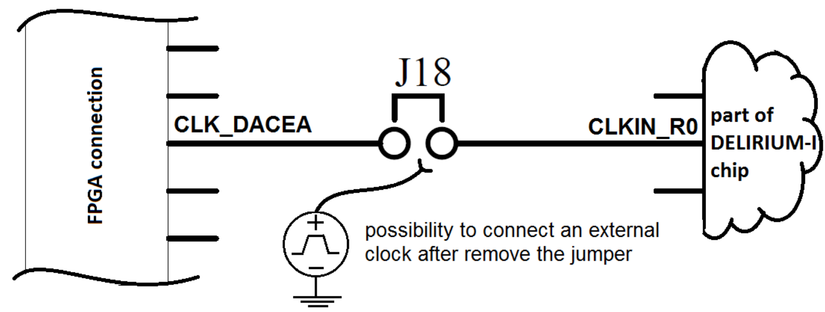


Figure 2.6: Jumper J18 application.

J8 is a jumper which allow you to chose a correct path output signal from capacitive divider as explained in the Figure 2.7 and Table 2.2.

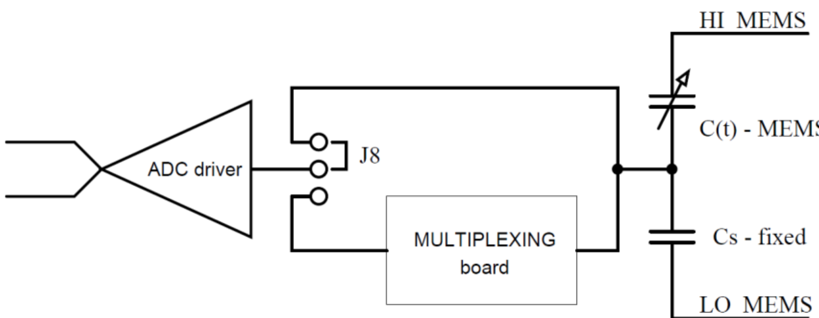


Figure 2.7: Jumper J8 application.

Option	Jumper J8 Setting	Description
MEMS_direct		This position is used to connect the middle point of divider directly to ADC driver for further processing.
MUX_board		The middle point of divider is led to multiplexing board where a few simple operations are performed depending on the mode

Table 2.2: J8 configuration

At the beginning of the work the jumper was set only in MEMS_direct position.

On the right side of this board there are still other jumpers available. By the means of them one can disconnect some potentiometers or measure appropriate current. When the board stops working without no reason it is a good idea to check the currents in the jumpers. Ranges of an allowed currents are gathered in the end of the next section.

2.2.2 Potentiometers

The board provide a several potentiometer to set appropriate value e.g. in the power section there are a possibilities to adjust a power supply voltage. Figure 2.8 shows which power voltage can be set by each potentiometer.

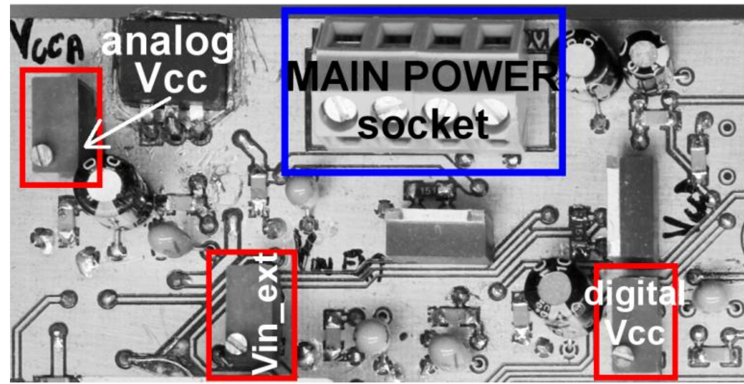


Figure 2.8: Power section description.

PIN no.	1	2	3	4	a part of board scheme
On board description	AUX	GND	15V	60V	ALIM. CON4 4 3 2 1 15V 60V C112 47u CMAX 0 0 VAUX
Recommend voltage supply	-11.7V	ground	15V	NC[1]	

Table 2.3: Main power connection/socket pin configuration.

In power supply section a simple National Semiconductor positive voltage regulators are used. Leading feature is a low dropout voltage, a maximum of 1.5V at 1.5A of load current. The circuit marked as LM1086 is available in an adjustable version, which can set the output voltage with only two external resistors included by described potentiometer. A simple application circuit can be founded in datasheet. A good habit is to separate analog and digital power supply. For that reason the board provide two different voltage regulators, moreover there is another one. This one is used to supply half-bridge PWN generator depicted in Figure 2.9. The signal VACT_EXT is led to HI_MEMS pin while – Low voltage – configuration is selected, look in Table 2.1.

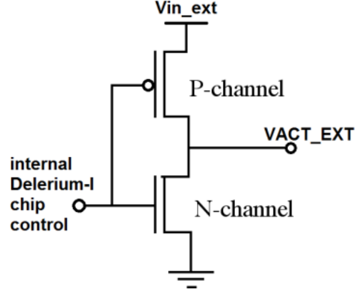


Figure 2.9: PWM half-bridge

Figure 2.10 depicts a general purpose of the right side potentiometers. The most important of them are located at the

[1] NC = NO CONNECT

bottom in row. They are responsible for setting a active width time in H, D, and R signal, what was explained in Introduction.

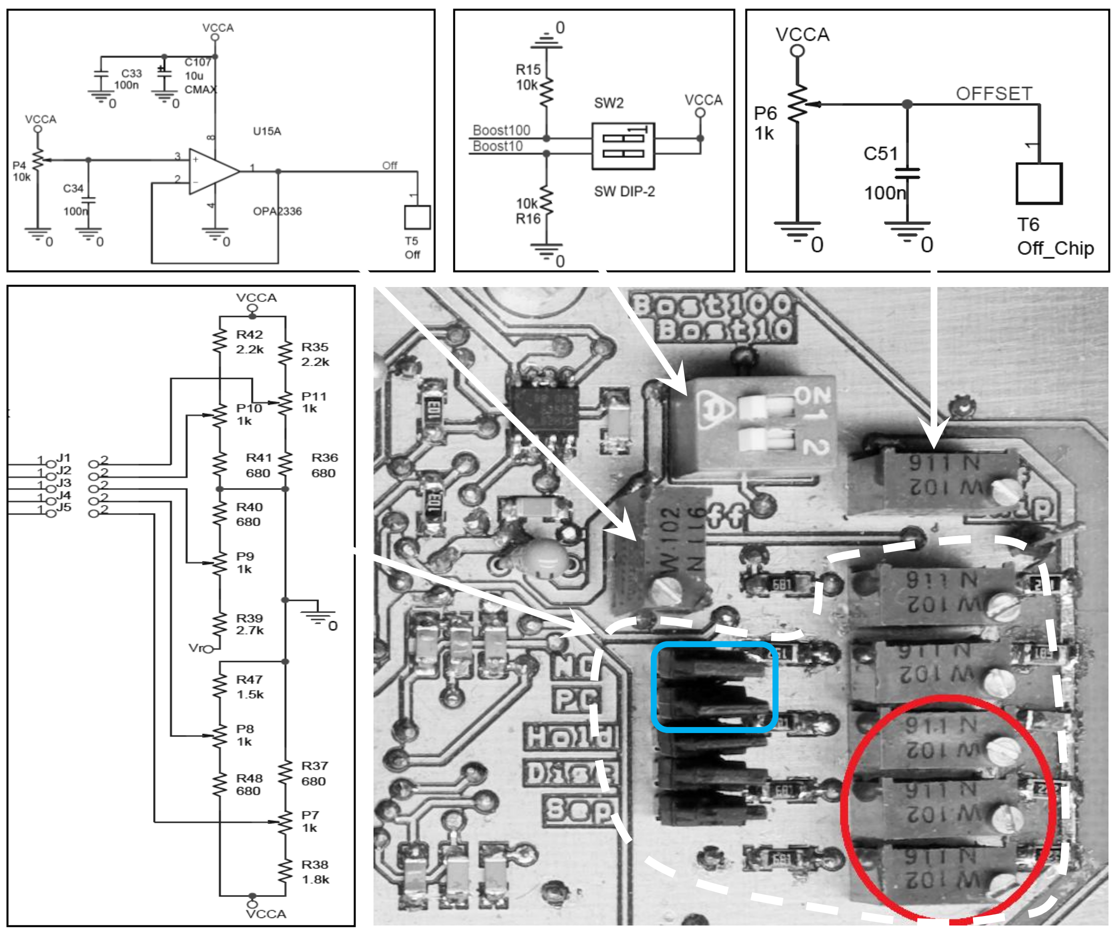


Figure 2.10: Right side potentiometer application

The Figure 2.10 shows a part of scheme containing a suitable potentiometer. The most important resistors are marked in an ellipse in the figure above. First one from the top provide a duty cycle adjustment in H signal, the middle one is used to set the same property but in D signal, and the last one in R signal. The main purpose of remaining jumpers is to measure and adjust a current. It is important to check it if something happened with the circuit. Two jumpers marked square should conduct current between 50 and 60µA, through remaining three current flow should be close to 2 µA. Adjusting a current is made by appropriate potentiometer.

2.2.3 Connections

In prototype and research department a fundamental PCB design criterion is to create a easy reconfigurable board. For that reason designing process was not focused on size but on future

usefulness. Several connections, jumpers, ports repeaters and probe pins were placed. This subsection focus on the connectors and theirs pin configuration.

Description start from the female socket marked MEMS in Figure 2.4. It is used to connect a MEMS actuators system or two fixed capacitors. The work with this project began from second option. The Figure 2.11 shows pin signals and part of the scheme containing this component.

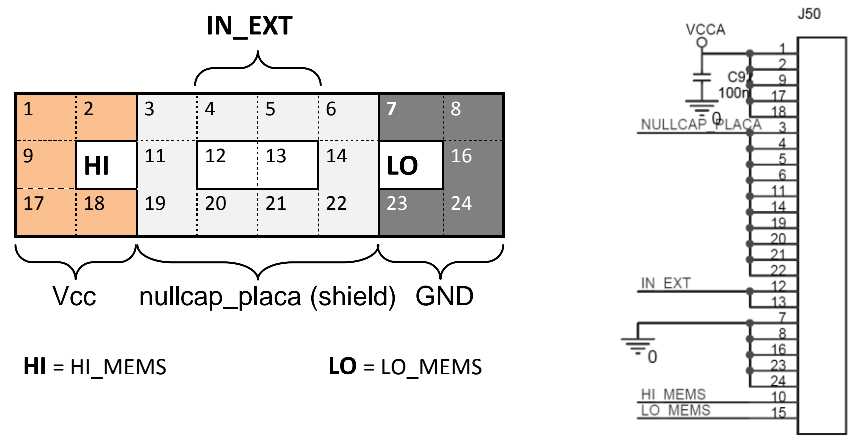


Figure 2.11: Pin used on the MEMS connector (CON24)

Another important female connector is used to connect a multiplexing board. It is located on the right side at the top. The multiplexing board will be presented in details in next section. Figure 2.10 presents a pinouts of that connector. There are two abbreviation: V_ext[22] = VACT_EXT (PWM wave generated from chip) and O_d[26] = OUT_dig – output from capacitance divider after simple arithmetic operation. OUT[25] is output directly from divider. Vaux[7,8] a negative voltage supply that goes from main power socket depicted in figure 2.5. Vccd[9,10] and Vcca[34] are respectively 3.3V digital and analog power supply, as GND[...] and GNDA[33] are digital and analog ground. R[11], D[13] and H[15] are MEMS system control signals and nR[12], nD[14], nH[16] are theirs negation. There are also multiplexer control signals marked as Dig1[29]..Dig4[32].

2 GND	4 60V	6 15V	8 Vaux	10 Vccd	12 nR	14 nD	16 nH	18 HI	20 LO	22 V_ext	24 GND	26 O_d	28 GND	30 Dig2	32 Dig4	34 Vcca
1 GND	3 60V	5 15V	7 Vaux	9 Vccd	11 R	13 D	15 H	17 GND	19 GND	21 GND	23 GND	25 OUT	27 GND	29 Dig1	31 Dig3	33 GNDA

Figure 2.12: Pin used in the multiplexing board (CONN34)

The last connector is located at the right edge. It gives a possibility to control the whole system by Xilinx FPGA board. Figure 2.13 presents a pin signals, but only previously not mentioned will be discussed. As before there are a few abbreviations: NC[19,34] = no connect, Cadc[6] = CLOCK on main scheme and represents a high speed clock connected to analog-digital converter, Cchip = CLKIN_R0 on main scheme that represents a clock connected to Delirium-I chip with possibility to

adjust the duty cycle to control and measure the MEMS capacitance. The D0[7]..D9[16] and OR signals come from ADC and are respectively data output bits and out-of-range indicator. AOUT+ and AOUT- are available as output from differential ADC driver, but only after placing the jumpers near to this area.

2 GND	4 15V	6 Cadc	8 D1	10 D3	12 D5	14 D7	16 D9	18 Dig1		22 GND	24 GND	26 GND	28 MAN	30 GND	32 H0	34 NC	36 H	38 AOUT+	40 GND
1 GNDA	3 60V	5 Vaux	7 D0	9 D2	11 D4	13 D6	15 D8	17 OR	19 NC	21 Dig2	23 Dig3	25 Dig4	27 CMP	29 Cchip	31 D0_C	33 D	35 R	37 AOUT-	39 GNDA

Figure 2.13: Pin used in the FGGA board (CON40)

There is one more connector for ADC but it is not necessary to describe it in details except of indicating where a first pin of ADC should go that is on the right side at the bottom.

2.3 MEMS actuators board

At this time is worth to mention about another small board which contains only a MEMS socket and few other connectors that allow to select an actuator and insert a fixed capacitor. On that board there are a lot of descriptors to facilitate a work with it, so here not all of them will be explained, only the most important. The board is presented in Figure 2.14, in the same figure the capacitor connector is depicted.

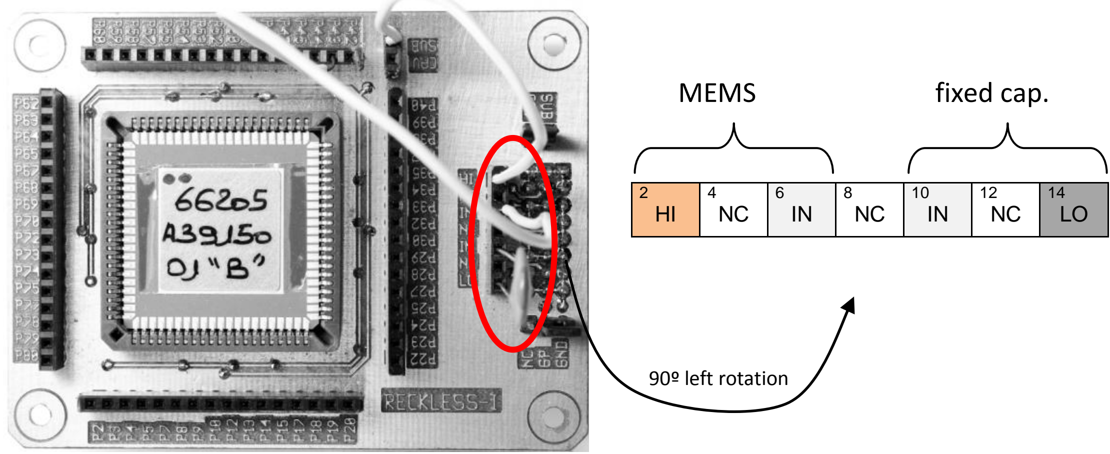


Figure 2.14: MEMS electrostatic actuators board with capacitor connector pin-outs

In this board there is a jumper between CAV and SUB signals, which are respectively the cavity of the chip and the substrate of the MEMS. CAV and SUB should be shorted. Other jumper, "NC" - "GP" - "GND", this is for selecting at which potential the ground plane (GP) should be, if it is connected to nullcap (NC) or directly to ground (GND), during measurements the GP signal should be grounded.

The best way to obtain a good performance with actuator movable plate is to connect signal marked as HI to SUB or CAV and IN signal to ones of pins located around the chip. The MEMS actuators board floor-plane is shown in the Figure 1.2. It can be seen that some of actuators have two and some of them just one available connection. The principle gave at the beginning of this paragraph concerns the second one, with only one lead called anchor. This connection have the same thickness through the entire length.

There is possibility to work with two-leads MEMS but configuration is different and deflection smaller. In this case it is necessary to combine HI signal to POLY0. In tested MEMS board this layers are affixed to paths which are shared with other actuators. This path is also thinner near actuator connection. As an example the pins number in MEMS floor-plan are given: 2, 4, 9, 19, 22, 60, 79 and so on. The second lead (anchor – similar to one-lead actuators) should be connect to IN signal. The 1.4 section presents how MEMS has been constructed, and where exactly POLY0 layer is.

Before describing how to choose an adequate fixed capacitor, it is important to mention about turning off a power supply.

WARNING

Another, faster prevention from burning the MEMS chip is to unplug a power supply.

To see a significant actuator plate movement it is important to choose appropriate fixed capacitor. The best way is to start doing this at 56pF. The fixed capacitor value strongly depend on MEMS capacitance and manufacture process, so the safest way is to start with the large one. After fastening it in a correct socket a chip clock's (not ADC clock) duty cycle should be set to 90%. The next important thing is to not turning on the power supply rapidly but increasing it slowly and observing at the same time ADC driver's differential output. The differential outputs from ADC driver should be approximately equal with properly matched capacitor and 15V power supply.

2.4 Multiplexing board

2.4.1 General description

This part has been completely designed by author of this thesis, of course, after consultation with coordinator. The main purpose is to generate a control PWM waveform in two different voltage

level combinations. The Introduction chapter presents how change a actuator capacitance with duty cycle and why the change of a voltage polarity is necessary.

The appendix B – Multiplexing board documentation presents a full user's guide of this board. In this section only a functional part and configuration will be describe. The schematic of this board is presented in Figure 2.15.

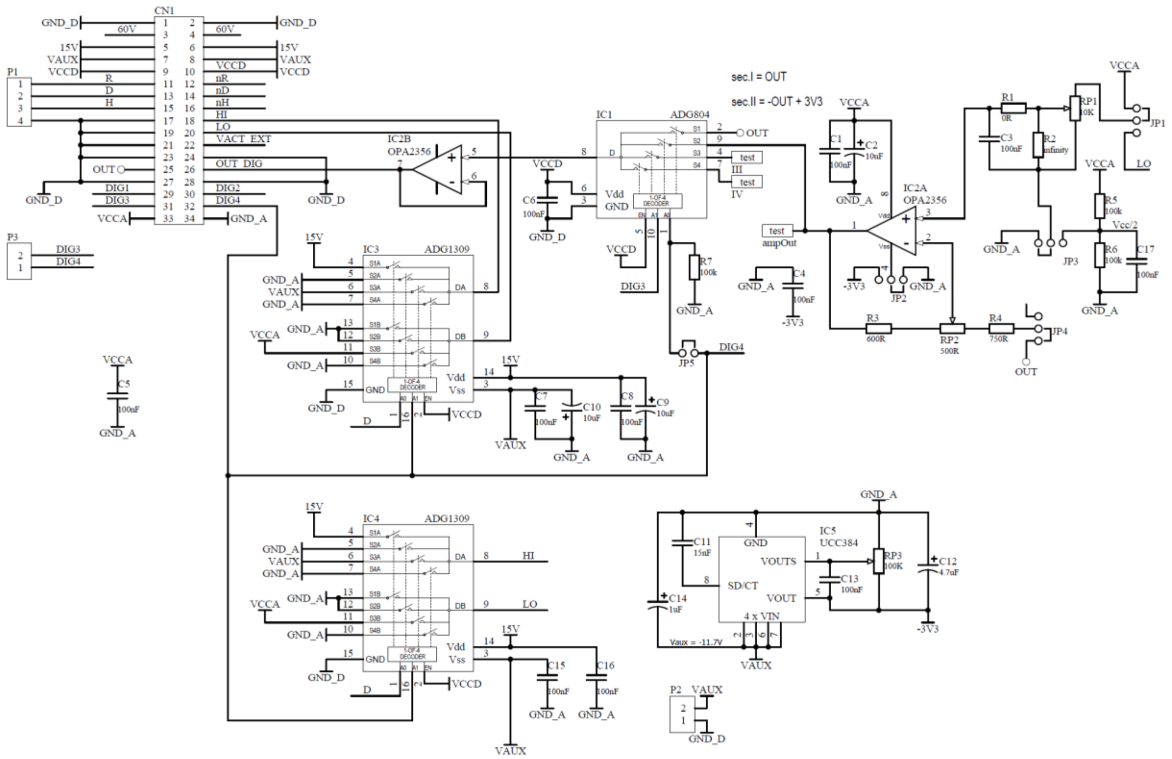


Figure 2.15: Multiplexing board’s scheme

Through the CN1 connection this board communicate with the rest of system. Among another thing there are two multiplexers to change the capacitive divider supply voltage, and one to choose a divider’s output signal path. This path could pass directly through multiplexer or firstly through inverting amplifier and then multiplexer. The output of this multiplexer is connected to voltage follower to match the impedance for both paths. There is negative voltage regulator as well, to generate a split power supply for amplifier to avoid a virtual ground introduction. The amplifier calculations are presented in the next section, and the problem with virtual ground is depicture in Figure 2.18. All multiplexers will be discussed in the 2.4.3 section.

2.4.2 Operational amplifier

A part of project was to design a circuit which should do some basic operations with capacitance divider output. Before pass on to amplifier configuration paragraph, one equation need to be derived. It is important to make the divider's output in seq_I dependent on the same output in seq_II. To make it clear les star at the beginning.

The capacitance of the electrostatic actuator C(t) and the series capacitor C_S form a capacitive divider circuit. If both capacitances are initially discharged and a voltage V_hi is applied to the upper node of the capacitive divider and V_lo voltage to the lower node, the intermediate voltage V_{OUT} follows the equation

$$V_{out} = \frac{V_{HI} - V_{LO}}{C_S + C(t)} \cdot C(t) + V_{LO} \tag{2.1}$$

The Figure 2.16 presents a capacitance divider to demonstrate the equation correctness. The electrostatic actuator is represented as the variable capacitor C(t), the series capacitor as C_S and the output as V_{OUT} .

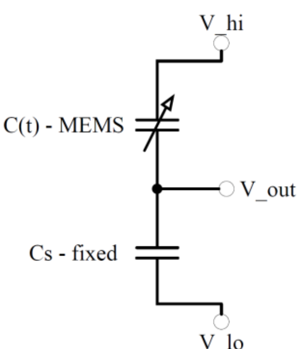


Figure 2.16: Capacitance divider.

For sequence I, where V_{HI}=15V and V_{LO}=0V the output V_{OUT} is therefore given by:

$$V_{out(I)} = 15V \cdot \frac{C(t)}{C_S + C(t)} \tag{2.2}$$

For sequence II, where V_{HI}=-11.7V and V_{LO}=3.3V output equal to:

$$V_{out(II)} = -15V \cdot \frac{C(t)}{C_S + C(t)} + 3.3V \tag{2.3}$$

The next step is to determined from equation 2.2 a $\frac{C(t)}{C_S + C(t)}$ component and insert it to 2.3. Finally desired relationship amount to.

$$V_{out(II)} = -V_{out(I)} + 3.3V \quad (2.4)$$

As equation 2.4 shows after changing between mode it is necessary to invert a output voltage and add a little offset. This task is performed by Texas Instrument's amplifier, and a scheme is presented in Figure 2.17.

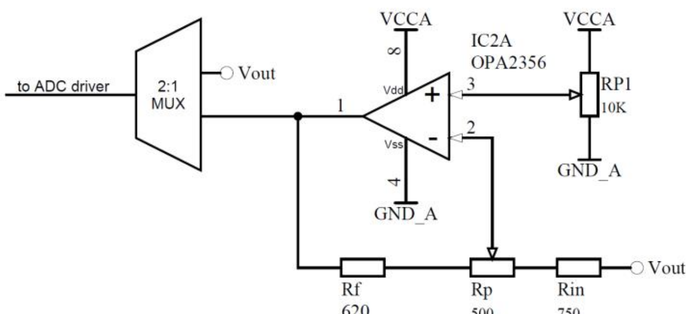


Figure 2.17: A inverting amplifier configuration

This is a simplified version of this amplifier, in original scheme there are also power supply filtering capacitors and many jumpers to select a mode configuration. The next paragraph describe all of them.

In the past there have been a lot of groups of op-amp circuits, but all of them have split power supply. While designing a single supply amplifier it is necessary to remember about specific conversion. It is not possible to change a supply from slip to single without doing modification in circuit. The next paragraph will explain this conversion. Figure 2.16 present split and single supply amplifier.

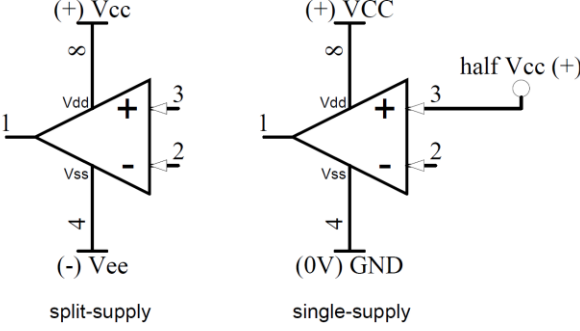


Figure 2.18: A split and single supply circuit

The power supply depicture on the left draws consist of positive and equal opposite negative pole. In this case input and output voltage are changed to the ground and swing to a limit of V_{OM}, the maximum peak-output voltage swing.

In single supply mode a new concept should be introduced. Virtual ground is a half of positive supply and ground. That is why it is necessary to add this voltage to a input signal. In fact only single supply configuration is used, and this error is compensating by adding a little less offset voltage. To clarify, the slope of input signal is controlling by a gain, and it isn't depend on power supply mode.

To make a board more useful and flexible the variable gain was designed. The gain range is 0.5 to 1.5, and the calculation is presented below.

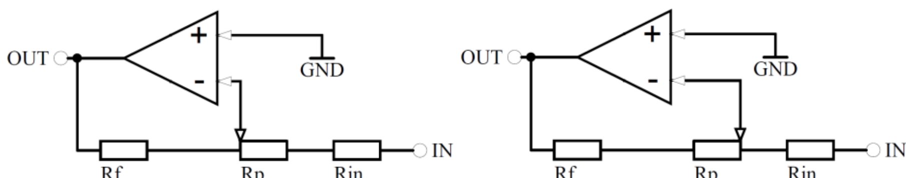


Figure 2.19: A adjustable gain on the Left G=0.5 and Right G=1.5.

Simultaneous equations presented below allow to derive all unknowns.

$$\begin{cases} \dfrac{R_f}{R_{in} + R_p} = 0.5 \\ \dfrac{R_f + R_p}{R_{in}} = 1.5 \end{cases} \quad (2.5)$$

There are three unknown, for that reason the assignment one of them is necessary. This value should be available in resistance series of types. For the potentiometer's resistance equal R_p = 500Ω other values are R_{in} = 750Ω and R_f = 625Ω. The 625Ω value doesn't exists in E24 series of types and the nearest is 620Ω. So finally the gain range for values arrangement presented below is from 0.49 to 1.5.

$$\begin{cases} R_f = 620\Omega \\ R_P = 500\Omega \\ R_{in} = 750\Omega \end{cases} \quad (2.6)$$

After explanation of all calculation in amplifier section it is time to present works in graphical way. Figure 2.20 presents only a shape without scales and any values. The dashed line present a system work with operational amplifier, but a continuous line without.

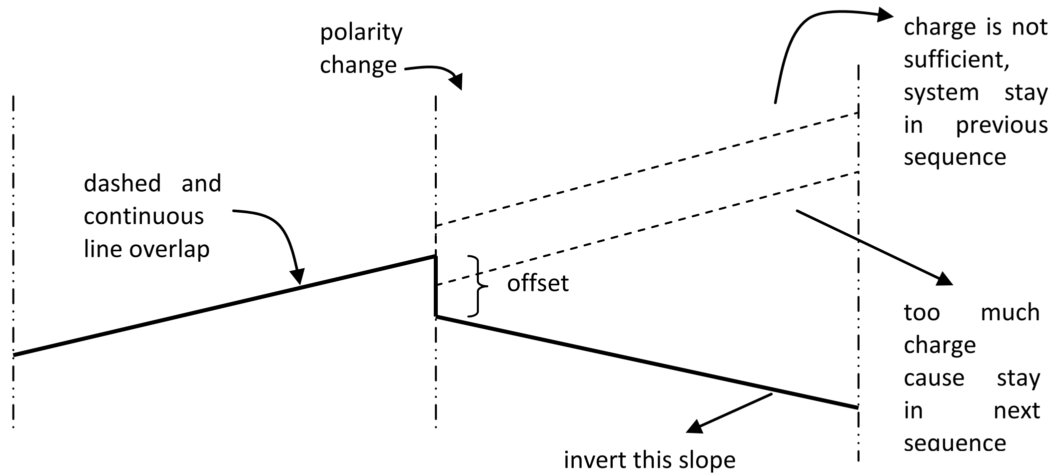


Figure 2.20: A charge injection shape with amplifier correction

Bold black line shows the course of the signal directly from the divider. In the sequence I, dashed lines show the possibilities, that may occur after changing the polarity. If stored charge is large enough, the measured capacitance is overstated after switching – this is represented by lower line, but the dashed line above present a behavioral if charge is not sufficient.

It is true that inverting and offset adding could be make by software, but much faster and easier is to do it by analog way.

Now the question arises why, after calibrating the device at the beginning of the work may appear there not enough charge. Answer is that, after the change of polarization, the capacitor is loaded opposite charge which in the next sequence should be re-charged. This situation was better describe in 1.6 Controlling the charge section.

2.4.3 Multiplexers

In this part describe a multiplexers choice to change a voltage polarity. Section takes to consider only two signals: PWM HI_MEMS supply a capacitance divider and PWM D which is controlled by Delirium-I. The high-voltage logic (+15V) inverter was implemented using the discrete commercial device ICL7667, which allows continuous operation at voltages up to 15 V. This scheme was presented in Figure 2.21.

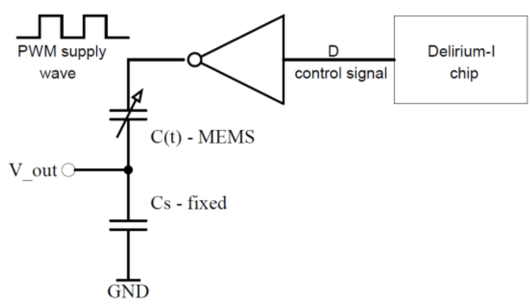


Figure 2.21: Divider's supply wave generator by inverter

This approach is good for only one mode, just for positive voltage +15V and GND wave. In design -11.7V and 3.3V voltage wave is also needful, so there was a problem to find a negative supply inverters or that circuit was very expensive. The new idea was proposed. Inserted there a symmetrical supply multiplexer solved that problem. More over this multiplexers could be order for free from Analog Device research center. Everything will be perfect but new circuit has a longer turning ON and OFF times. The solution for that problem also was founded by placed parallel two of them. The scheme of new approach was presented below in Figure 2.22. For limpidity a second multiplexer was omitted.

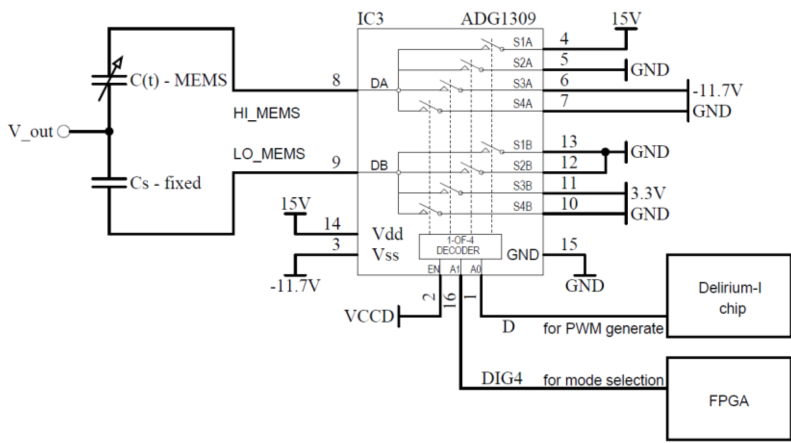


Figure 2.22: Divider's supply wave generator by multiplexer

To calculate the time constants of divider supply circuit, equivalent capacitance should be estimated at first. Substitute capacitance connected in series could be calculated by the formula 2.7, and for $C_{MEMS} \approx 20pF$ and $C_S = 68pF$, a $C_{eq} = 15pF$.

$$\frac{1}{C_{eq}} = \frac{1}{C_{MEMS}} + \frac{1}{C_S}$$
$$C_{eq} = \frac{C_{MEMS} \cdot C_S}{C_{MEMS} + C_S} \tag{2.7}$$

A switching time calculation for both approaches was presented below.

- Inverter

  At V_{CC} = 15V, the output impedance of the inverter is typically R_{out} = 7Ω. The time constant is derived in formula below.

$$\tau = R_{out} \cdot C_{eq} = 7\Omega \cdot 15pF = 105ns \tag{2.8}$$

- Multiplexer

  The output impedance of this multiplexer with dual power supply is typically R_{out} = 130Ω, and time constant is derived below.

$$\tau = R_{out} \cdot C_{eq} = 130\Omega \cdot 15pF = 1.95us \tag{2.9}$$

  This time has been reduced twice by connected two MUX in parallel. More over there are introduced several modifications in the code to slightly delay the start of sampling, it will be thoroughly explained in the chapter devote to the program.

To choose a right multiplexer a logic level capability is important. The Delirium-I chip was designed to work with TTL level, for that reason the ADG1309 device with 3V, logic-compatible digital input where: V_{IH} = 2.0 V and V_{IL} = 0.8 V was selected.

The selected multiplexer has a active high digital input to disconnect all switches. When this signal is low, the device is disabled and all switches are off. When high, Ax logic inputs determine on switches. In project there are no needs to disconnect them so the EN input is shorted to 3.3V.

The controlling signals A0 and A1 allow to select a appropriate input. As it is depicture in documentation block diagram this multiplexer has a 1-of-4 decoder and in fact consists with two identical 4 inputs and 1 outputs multiplexers and at the same time the control Ax signals connect both switches. This is useful to generate a both LO and HI_MEMS signals concurrently. One of this outputs goes to high point of actuator and another to low of fixed capacitor. This situations is depicture in figure 2.20. Another useful combination was that D signal is connected directly to A0. This makes the PWM generation independent from FPGA, because D is generated by Delirium-I

chip, and only changing between sequence is controlled by FPGA program. This situation is also presented in the Figure 2.22.

2.4.4 Negative voltage calculation

To generate a negative supply voltage the UCC384-adj Texas Instrument's linear regulator was used. In this projects it is no necessary to choose high efficiency regulators just to supply one amplifier which can consume less then 20mA current. The adjustable version was placed because of major functionality. The output voltage is defined by equation 2.10.

$$V_{out} = -1.25 \cdot \left(1 + \frac{R_1}{R_2} \right) \tag{2.10}$$

Experience shows that in prototype version it is better to use potentiometer, which allows voltage adjustment in wide range. This solution entails some danger. It is necessary to tune voltage before lead it to circuit.

2.4.5 Printed circuit boards

This section describe a board layout design. There will be presented two version of multiplexing board, and a rules which was kept during designing process. Board were made using a free 30-day trial version of Altium Designer. Despite the fact that this program provides a appreciable library items, sometimes it is needed to add a new one, and the standards are becoming very useful in this. That is why this section shall begin by introduce to the most important of them.

a) The CAD Library standard

Standard component package outlines come from industry standard organizations that specialize in component packaging data and standardization of documents and publications. One of the most well known organization is **IPC** – Association Connecting Electronics Industries. IPC is the trade association that brings together all of the company in this industry: PCB designers, PCB manufacturers, PCB assembly companies, suppliers, and original equipment manufacturers.

These standards define all aspects of facing the engineer. Collection of materials created for this project does not meet all of them, since these standards are developed for large companies where the production takes place in series. Although the multiplexing board is not complicated design, and will not be mass produced, it is worth to follow the standards for each project.

Since 1987, when an engineer needed information about the dimensions and ranges footprint tolerance, reached the standard IPC-SM-782. This standard defines the appearance of PCB components for a huge number of standard SMD components, but does not define specifically their names. In 2005 appeared the standard IPC-7351. All libraries included in Altium Designer is made in complying with this standard. At the points below there are listed the most useful standards.

- IPC-2221A: Generic Standard on Printed Board Design
- IPC-2222: Sectional Design Standard for Rigid Organic Printed Boards
- IPC-7351A: Generic Requirements for Surface Mount Design and Land Pattern Standard
- IPC-7251 (draft): Generic Requirements for Through-Hole Design and Land Pattern Standard.

All boards in this project were manufactured in milling process without metallization. That has imposed the following requirements, all tracks, pads, and vias was enlarged.

b) A first version

This version of the board had a few bugs. The first was its size 52 x 27 mm too small. It was very difficult to make any modification or measurements on the tracks or pins. In addition, in the second mode (negative divider supply), HI_MEMS and LO_MEMS signals looked the same as control D signal. Capacitors are not discharged during the reset, however, during the measurement. The layout of the first version is presented below in the figure 2.21.

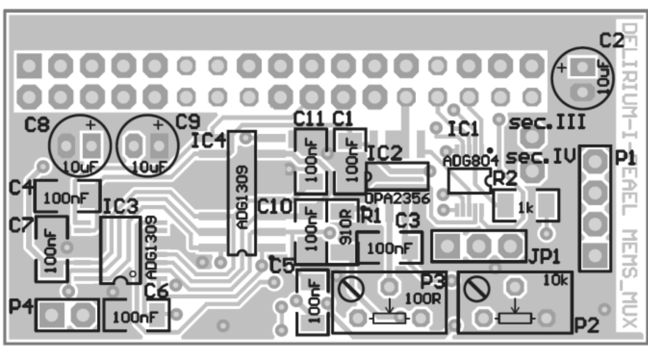


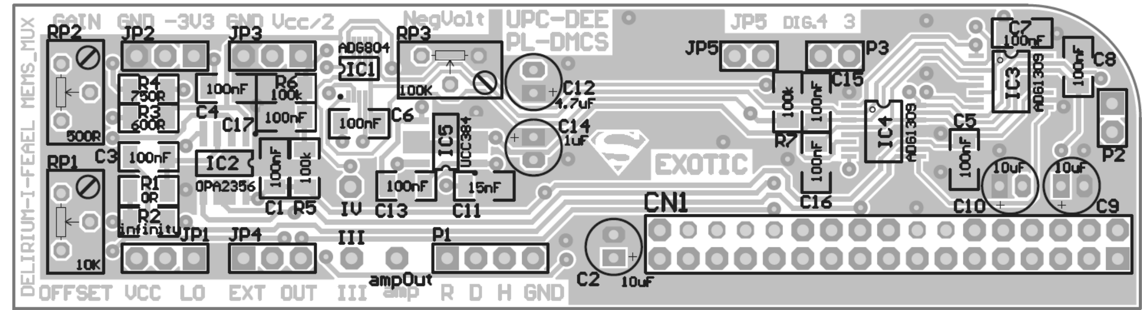
Figure 2.23: A first version of multiplexing board

c) A second version

Advantages of this board were introduced a numerous descriptions in the cooper. All elements are logically arranged and work with it is more intuitive and requiring no schema. Board is double layer with components of only one side, with the majority of surface mount elements. A vias were hand made with silver steel. All polygon plane were connected to analog ground, provide efficient shielding.

The dimension of new board are: 100 x 27 mm, and rounding in top left corner R = 10 mm. All tracks were placed according to the principle – horizontal on top, vertical at the bottom, then circuit became more clear. The maximum absorbed current is 20 mA so there is no need to increase the thickness of power paths.

TOP LAYER



BOTTOM LAYER

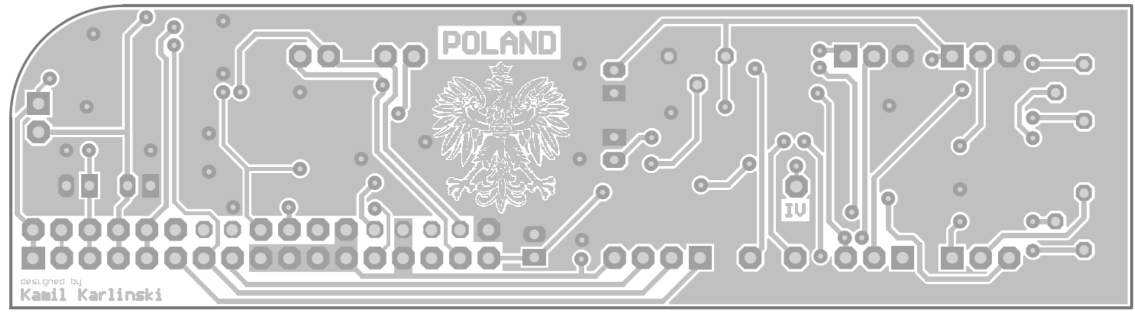


Figure 2.24: A two layers of second version of multiplexing board.

2.4.6 Jumpers

This section as before, includes a paragraph about the jumpers and their proper configuration. The majority of them comes from amplifier.

Jumper JP1 is used to select on offset voltage. In equation 2.4 this value amount to 3.3V. Therefore, this level could be achieve from power supply or LO_MEMS signal in the second mode. Assuming that the PWM voltage may be modified equation 2.4 may take the form, and offset could depend on V_{LO_MEMS} voltage.

$$V_{out(II)} = \frac{V_{HI(II)} - V_{LO(II)}}{V_{HI(I)} - V_{LO(I)}} \cdot V_{out(I)} + V_{LO(II)} + V_{LO(I)}$$

$$V_{out(II)} = gain \cdot V_{out(I)} + V_{LO} \tag{2.11}$$

Jumpers JP2 and JP3 are used to choose a amplifier's supply voltage, split or single. Difference between these modes of supply was discussed in 2.4.2 section.

Option	Jumper JP2 Settings	Jumper JP3 Settings	Description
Single supply			This position it is necessary to remember about virtual ground during a measurements.
Slit supply			This is basic and very useful configuration, amplifier operates in the manner described in every manuals
Single supply without ground shifted			This configuration was used during testing circuit, as it was describe before, gain was adjust separately and offset error was corrected by PR1 potentiometers.

Table 2.4: Amplifier power supply configuration.

The JP4 jumper is useful to test only amplifier operation without reordered circuit. Output of amplifier is available on gold pin marked as out on PCB cooper.

Option	Jumper JP2 Settings	Description
Normal mode		In this position the out from capacitance divider are connected to inverting input of amplifier, this is normal operational mode.
Test mode		This position is used to test a amplifier correct operation. There is possible to connect an external signal source – through this jumper and tune a gain and offset.

Table 2.5: The jumper's JP4 purpose

The JP5 jumper is used to disconnect a amplifier. Signal DIG(4) set a sequence mode, when the jumper is in place the multiplexer switch between first and second input depends on current sequence. In sequence II the output from divider is led thought amplifier circuit. If JP5 jumper is removed, the R7 resistor pull off the A0 control input to ground, assuring S1 terminal connect to the output drain D. Which means output from capacitance divider is permanently connected to ADC circuit, and do not depend on current sequence, the amplifier is bypassed.

Board provide 3 probe pins to observe a several control signals. All of them are depictured in the Figure 2.25.

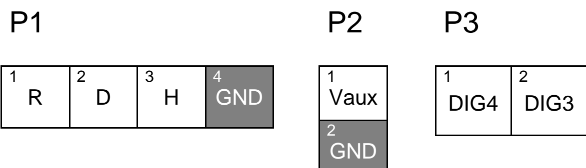


Figure 2.25: Pin signal in probe terminals.

3 Algorithm implementation

3.1 Overview

This chapter presents a hardware description and synthesis with VHDL. At the begging the block diagram will be shown with a description of each state. Then, will be description of all important pieces of code that will be included in the Appendix.

3.2 Codes description

3.2.1 ASM chart

This section present a flow chart of designed algorithm and also describe the most important part.

An FSM (finite state machine) is used to model a system that transits among a finite number of internal states. The transitions depend on the current state and external input. Unlike a regular sequential circuit, the state transitions of an FSM do not exhibit a simple, repetitive pattern.[3]

An FSM is usually specified by an abstract *state diagram* or *ASM chart* (algorithmic state machine chart), both capturing the FSM's input, output, states, and transitions in a graphical representation. The two representations provide the same information. The FSM representation is more compact and better for simple applications. The ASM chart representation is somewhat like a flowchart and is more descriptive for applications with complex transition conditions and actions. The ASM chart of the project is depictured in Figure 3.1.

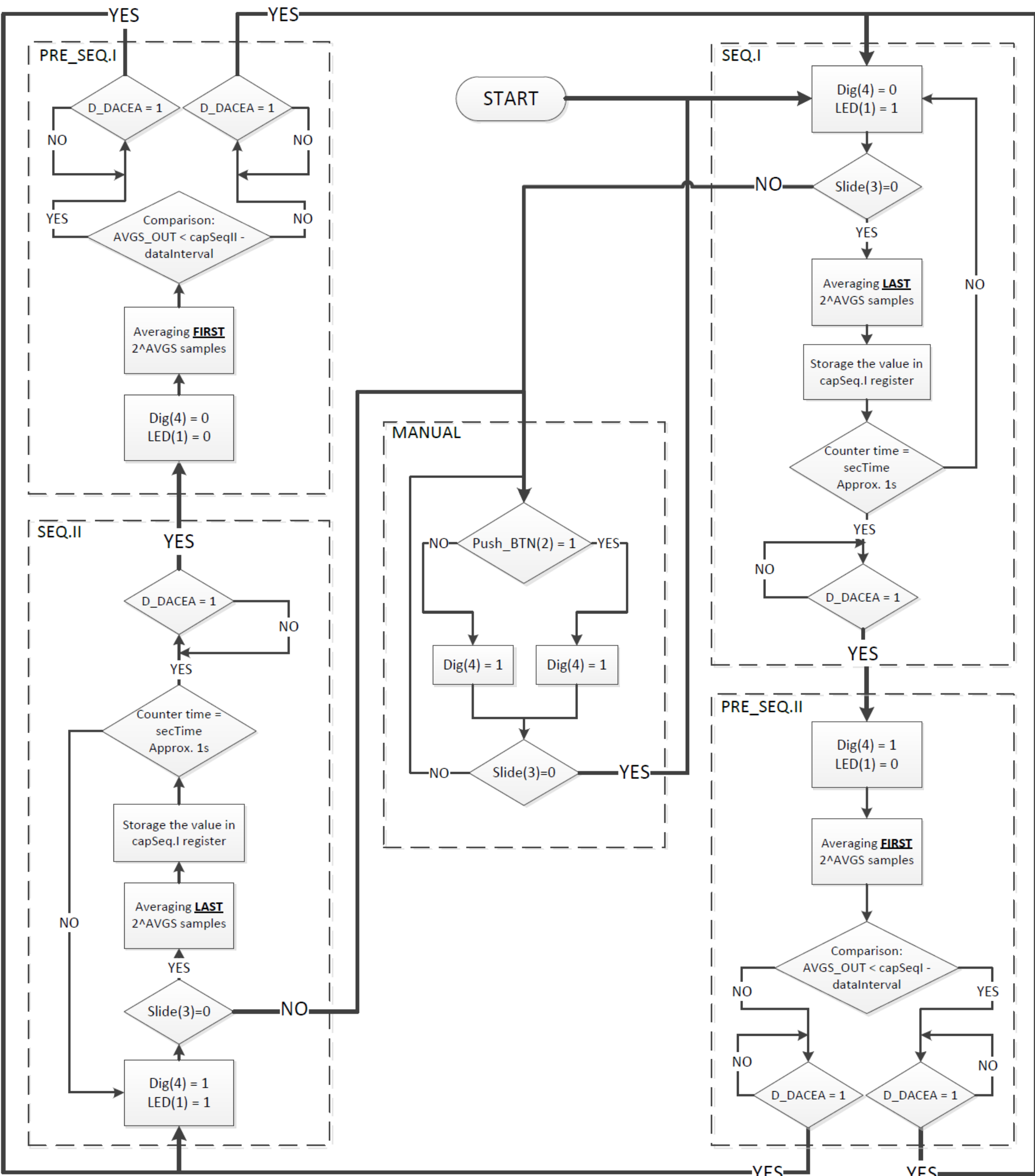


Figure 3.1: The ASM chart of charge injection algorithm.

State machine at the beginning of work, consisted of only two states, for readability and ease of troubleshooting there were introduced an additional two states, responsible for making decision in which sequence system should stay to reduce a parasitic phenomenon. There are also another state called manual, which allow to change a sequence with pushing a button. These mode is used to test circuit and to better show a charge injection effect. Program starts work from SEQ.1, and at the beginning the counter start measure a 1s time in order to change a polarity and compare the quantity of charge that has been injected into the structure. Before switching the program have to collect all samples to do average, and wait for high level of D signal. In this state (seq.I) there is also possibility to go to manual mode, by asserting a slide

button high. After meeting all requirements the program pass to pre_seq.II. Where its main task is to take a decision in which sequence program should stay further, but before that a samples should be averaged. So again collecting all 2^{AVGS} samples, do average, and decide. When a decision will be made a next step is to wait for high level of D signal. Operation principle in state seq.II and pre_seq.II is similar to seq.I and pre_sqe.I. Anyway for the transparency of the code, separates state have been introduced, thereby increasing the number of flip-flops only by 2. In a simple projects, this number does not play the role, although in larger system it should be take into account as the best code and hardware optimization. Before pass through the main part it is worth to mention about clocks in whole system.

3.2.2 Clocks in design

The Spartan-3 Starter Kit board has a dedicated 50 MHz series clock oscillator source but also provide a advanced clocking capabilities as Digital Clock Managers (DCMs) [20]. DCMs optionally multiply or divide the incoming clock frequency (50Mhz) to synthesize a new clock frequency. DCMs also eliminate clock skew, thereby improving system performance. This property has been used in the project to generate digital to analog converter's clock with 100MHz frequency. It is worth to mention that converter belong to family of monolithic, single 3V supply, and 10-bit data resolution, with three different processing speed. In the design the fastest one is used, with 105MSPS (**M**illion **S**amples **P**er **S**econd). It means that, the highest frequency which could be apply, amount 100MHz.

The main clock synchronize all processes is called CLK_IF. It is generated also from DCM instantiate as CLKFX which is an output from Frequency Synthesizer. The Digital Frequency Synthesizer (DFS) provides a wide and flexible range of output frequencies based on the ratio of two user-defined integers, a Multiplier (`CLKFX_MULTIPLY`) and a Divisor (`CLKFX_DIVIDE`). The output frequency is derived from the input clock (CLKIN) by simultaneous frequency division and multiplication. As it is depicture in the listing 3.1 (in line 126) a first variable depend of `SSFACTOR` constant, which takes the value 3. Moreover this constant is used to connects a clock frequency with output pattern width. DCM instance is presented in the Listing 3.1.

Listing 3.1 DCM instantiate – **MEMS_ChargeInReduce.vhd**

```
119  --===================================================
120  -- DCM instantiate
121  --===================================================
122     DCM1 : DCM
123     generic map (
124        CLKDV_DIVIDE => 10.0, --  Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6
125                           -- 7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
126        CLKFX_DIVIDE => 2**(SSFACTOR),   --  Can be any integer from 1 to 32
127        CLKFX_MULTIPLY => 2, --  Can be any integer from 1 to 32
128        CLKIN_DIVIDE_BY_2 => FALSE, --  TRUE/FALSE to enable CLKIN divide by two…
129        CLKIN_PERIOD => 20.0,           --  Specify period of input clock
130        CLKOUT_PHASE_SHIFT => "NONE", --  Specify phase shift of NONE…
131        CLK_FEEDBACK => "2X",          --  Specify clock feedback of NONE, 1X or 2X
132        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS", --  SOURCE_SYNCHRONOUS,
133                               --   SYSTEM_SYNCHRONOUS or an integer from 0 to 15
134        DFS_FREQUENCY_MODE => "LOW",      --  HIGH or LOW frequency…
135        DLL_FREQUENCY_MODE => "LOW",      --  HIGH or LOW frequency mode for DLL
136        DUTY_CYCLE_CORRECTION => TRUE, --  Duty cycle correction, TRUE or FALSE
137        FACTORY_JF => X"C080",           --  FACTORY JF Values
138        PHASE_SHIFT => 0,          --  Amount of fixed phase shift from -255 to 255
139        STARTUP_WAIT => FALSE) --  Delay configuration DONE until DCM LOCK,…
140     port map (
141        CLK0 => CLK_50MHz,      -- 0 degree DCM CLK ouptput
142        CLK2X => CLK_100MHzI,    -- 2X DCM CLK output
143        CLKFX => CLK_LF,    -- DCM CLK synthesis out (M/D)
144        CLKFB => CLK_100MHz,    -- DCM clock feedback
145        CLKIN => CLK_IN    -- Clock input (from IBUFG, BUFG or DCM)
146     );
148        BUF1: BUFG port map (I => CLK_100MHzI, O => CLK_100MHz);
```

In the project was being implemented additional clock divider for general purpose. It is used mainly to timing and delays. A Listing 3.2 presented its implementation.

Listing 3.2: General purpose clock divider – **MEMS_CharageInReduce**

```
224  -- General-purpose clock pre-scaler
225  process (CLK_LF)
226   begin
227      if rising_edge(CLK_LF) then
228          CLK_DIV<=CLK_DIV+1;
229      end if;
230   end process;
```

Another very important clock is a clock that control a chip's operation. In this project there is possible to select one of several output clock patterns. The main of them was designed by Daniel Fernandez and that of the ruts is very extensive, including lot of collateral types, and providing a user intervention.

Due to the lack of accurate documentation and misunderstandings of all inserter protections, a new simple clock, allowing only a period and duty cycle adjust was introduced. It is presented in the , below.

Listing 3.3: General purpose clock divider – **MEMS_ChargeInReduce.vhd.**

```
177  entity MY_PATTERN is
178      Port (
179                  clk : in  std_logic;  -- 12,5 MHz clock what means 80ns period
180                  dutyCycle : in NATURAL range 1 to 1023;
181                  patternPeriod : in natural range 1 to 1023;
182                  out_pattern : out std_logic
183          );
184  end MY_PATTERN;
185
186  architecture Behavioral of MY_PATTERN is
187   signal Counter : natural range 0 to 1023 := 0;
188  begin
189   process (clk) begin
190          if rising_edge(clk) then
191                  if (Counter < patternPeriod) then
192                          Counter <= Counter + 1;
193                  else Counter <= 0; end if;
194          end if;
195   end process;
196   out_pattern <= '1' when Counter < dutyCycle else '0';
197  end Behavioral;
```

As it was described in the Introduction chapter, a MEMS actuator capacitance could be changed with duty cycle. In the 3.2.7 section there will be description how to do it, but in the first place an important instantiates will be presented.

3.2.3 Important instantiates

At the beginning of this section it is worth to mention what is a current information path to be correctly interpreted in order to make a proper decision. The easiest way to present it is to show it on the functional block diagram.

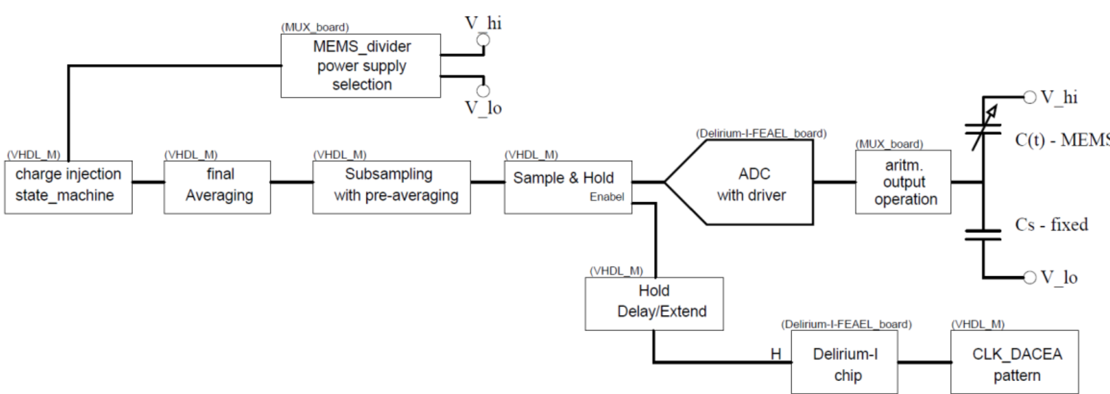


Figure 3.2: A whole system functional block diagram

In the next part there will be explanation for each module, start with `Sample & Hold`. Instantiate is presented in the Listing 3.4

Listing 3.4: S&H instantiate – **MEMS_ChargeInReduce.vhd.**

```
136  ------------------- Sample and Hold for ADC data  --------------------
137  SAHP : entity work.sampleandhold(Behavioral)
138         generic map (n=>11)
139         port map (
140                clk => CLK_100MHz,
141                hold => H_POS,
142                din => DATA_ADC_SIGNED,
143                dout => SAHP_OUT
            );
```

When the hold is high, capacitance was recharged and this kind of information should be separated from other part to prevent glitches among other things. For this reason there was introduced a module responsible for stopping the sampling at the time of capacitance reset. To control the sample and hold module it was used hold signal (H) generated by the chip, but before the signal is subordinate to a small modification. Below there is presented a listing and detail explanation.

Listing 3.5: Hold signal delayed – **MEMS_ChargeInReduce.vhd.**

```
136  ---------------    Delayed of the HOLD (H_DACEA) signal   -------------
137  HLDP : entity work.delayed_stim(Behavioral)
138         generic map(predelay=>PREHOLD_TIME, postdelay=>HPOS_TIME, preventive=>'1')
139         port map (
140                clk => CLK_100MHz,
141                inp => H_DACEA,
142                outp => H_POS
143         );
```

To explain why the signal delay was introduced it is necessary to mention about how converter operation is based. The AD9215 uses a multistage differential pipelined architecture with output error correction, and this of course means that the actual capacitance value are delayed, more over introducing this delay could prevent a short interferences appear after changing a control level. Figure 3.3 present a timing diagram, and demonstrate how ADC process the samples.

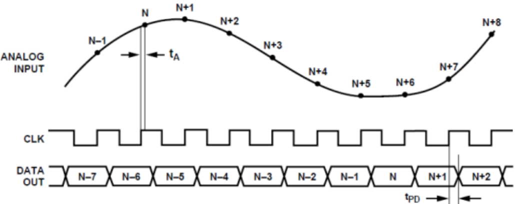


Figure 3.3: ADC timing diagram.

3.2.4 Averaging as a low-pass filter

In this section there will be averaging function description. At the beginning of project several question arise, one was: how to determine the typical amount, a valid estimate, or the true value of some measured parameter?

In the physical world, it is not easy to do because unwanted random disturbances contaminate measurements. These disturbances are due to both the nature of the variable being measured and the fallibility of our measuring devices. Each time during accurate measurement some physical quantity, a slightly different value was get. Those unwanted fluctuations in a measured value are called noise, and digital signal processing practitioners have learned to minimize noise through the process of averaging. The literature, shows not only how averaging is used to improve measurement accuracy, but that averaging also shows up in signal detection algorithms as well as in low-pass filter schemes [13] [15].

In digital signal processing, averaging often takes the form of summing a series of time-domain signal samples and then dividing that sum by the number of individual samples. Mathematically, the average of N samples of sequence x(n), denoted x_{ave}, is expressed in the equation 3.1.

$$x_{ave} = \frac{1}{N}\sum_{n=1}^{N} x(n) = \frac{x(1)+x(2)+x(3)+...+x(N)}{N} \tag{3.1}$$

Explaining the process of averaging is worth to mention about one well know filter structure. FIR (**F**inite **I**mpulse **R**esponse) low pass filters given a finite duration of nonzero input values. This filter use addition to calculate their outputs in a manner much the same as the process of averaging uses addition. In fact, averaging is a kind of FIR filter and successive time-domain outputs of an N-point averager are identical to the output of an (N–1)-tap FIR filter whose coefficients are all equal to 1/N. Figure 3.4 shows the discussed idea.

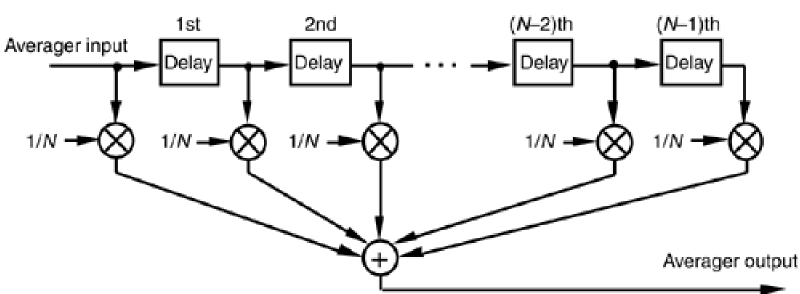


Figure 3.4 An N-point average depicture as a FIR filter

Listing 3.6 Avarage implementation in VHDL – **dsp.vhd**

```
213  ----------------------------------------------------------------------------------
214  -- Averaging device
215  ----------------------------------------------------------------------------------

221  entity averaging is
222          generic ( avgs : integer;
                       n : integer);
224          port (
225                  clk : in std_logic;
226                  din : in  STD_LOGIC_VECTOR(n-1 downto 0);
227                  dout : out  STD_LOGIC_VECTOR(n-1 downto 0)
228          );
229  end averaging;

231  architecture Behavioral of averaging is
232
233   type samples IS ARRAY ((2**avgs)-1 DOWNTO 0) OF STD_LOGIC_VECTOR(n-1 downto 0);
234   signal dataSamples : samples;

236  begin
237   process (clk)
238    variable accumulator : STD_LOGIC_VECTOR(n+avgs-1 downto 0) := (OTHERS => '0');
239    begin
240    if rising_edge(clk) then
241          -- add new sample and detele the last one (Shift right)
242          -- first LSB is deleted, then array is shifted right
243          -- and the new sample is put in MSB place
244          for index in 0 to (2**avgs)-2 loop
245                  dataSamples(index) <= dataSamples(index+1);
246          end loop;
247          dataSamples((2**avgs)-1)<=din;

249          -- average all data in array
250          accumulator := (others=>'0');
251          for index in dataSamples'range loop
252                  accumulator := accumulator + ext(dataSamples(index),n+avgs);
253          end loop;
254          dout <= accumulator(n+avgs-1 downto avgs);
255    end if;
256   end process;
257  end Behavioral;
```

Listing 3.6 presets software implementation of averaging module. Number of samples averaging in one cycle is determine by `avgs` generic, and amount to 2^{avgs} samples. All of them are accumulated in a array at first (line 247). During the time, when new sample arise, array is shifted to the right and new value goes to the MSB (**M**ost **S**ignificant **B**it) place. After that all data in the array are summed into accumulator (lines: 251-253). Division by the number of samples was carried by the logical right shift accumulator register, whose bits are simply moved a number determine by the `avgs` generic. The `n` predefined variable specify a data sample's number of bits.

As it was illustrated in the Figure 3.3, to obtain one average sample it is necessary to collect min 2^{avgs} data. That figure 3.2 shows more clearly the principal of progressive averaging. A average from rectangles marked with a cross will not be further subjected to process because of incomplete set of samples.

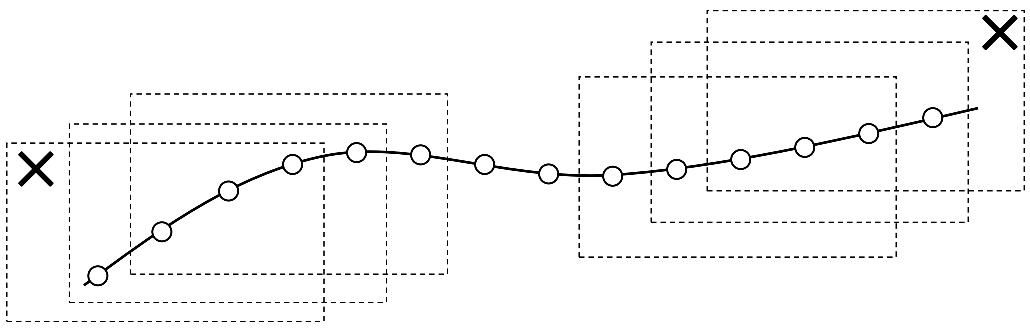


Figure 3.5 An N-point progressive average with prohibited cases

The system have to wait to collect all samples, the special counter was designed for doing that task and it is presented in the listing 3.3, but before discussing this part a short explanation is necessary. Counter is synchronize with `CLK_LF` clock, and it could be asynchronous reset by `RST` signal, or synchronous by assert a slide button high. A `DIGS_int(4)` is a signal flag to indicate and control a sequence mode. The different settings of the `DIGS_int(4)` bit have the following effects:

`DIGS_int(4) = '0':`

HI_MEMS: +15V / 0v

LO_MEMS: permanently grounded

`DIGS_int(4) = '1':`

HI_MEMS: 0v / -11.7V

LO_MEMS: 3.3V / 0v

Due to the possible changes in the number of averaging samples up-down counter was proposed. Collect all samples is indicated by setting a overflow bit in the same counter. Counter declaration is shown in the Listing 3.7.

Listing 3.7 Averaging counter declaration – **MEMS_ChargeInReduction.vhd**

```
93  signal avgCount : STD_LOGIC_VECTOR (AVGS DOWNTO 0) := (others=>'0');
```

The maximum value amount 2^{AVGS+1} not 2^{AVGS}, as should be normally. The new concept was introduced. The MSB(**M**ost **S**ignificant **B**it) is used to indicate, that 2^{AVGS} value was counted, when it happen in the next iteration counter just overload, assert meanwhile the `avgCount(AVGS)` bit.

Listing 3.8: Averaging counter implementation – **MEMS_ChargeInReduction.vhd**

```
466  -- ------------------------   avgCount   ------------------------
467  process (CLK_LF, RST) begin
468     if RST = '1' then
469         avgCount <= (others=>'0');
470     elsif rising_edge(CLK_LF) then
471         if SLIDE_FILT(3) = '0' then
472            if DIGS_int(4) = '0' then
473               if avgCount(AVGS) = '1' then avgCount <= (others=>'1');
474               else avgCount <= avgCount + 1; end if;
475            elsif DIGS_int(4) = '1' then
476               if avgCount(AVGS) = '0' then avgCount <= (others=>'0');
477               else avgCount <= avgCount - 1; end if;
478            end if;
479         else
480            avgCount <= (others=>'0');
481         end if;
482     end if;
483  end process;
```

During sequence II, `avgCount` is counting up, but in sequence I down. For that reason counter have to be loaded to one of limits value. When the `avgCount(AVGS)` bit is set, it is loaded to maximum (line 473), and when `avgCount(AVGS) = 0` counter is cleared (line 476).

After presented all the most important instantiates in the module it is time to show how the state machine was implemented.

3.2.5 State machine description

Several approaches can be conceived to design a FSM (**Fi**nite **S**tate **Ma**chine) which consist of lower and upper section. The lower one contains the sequential logic (flip-flops), while the upper section contains the combinational logic. The sequential section in this design has three inputs (`CLK_LF`, `RST` and `state_next`), and one output (`state_reg`). The Listing 3.9 shows this part.

Listing 3.9: Lower (Sequential) section of state machine – **MEMS_ChargeInReduction.vhd**

```
429  -- ####################   Lower section (registers)   ####################
430  -- ------------------------   STATE_REG   ------------------------
431  process (CLK_LF, RST) begin
432         if RST = '1' then
433                state_reg <= seqI;
434         elsif rising_edge(CLK_LF) then
435                if D_DACEA = '1' then
436                       state_reg <= state_next;
437                end if;
438         end if;
439  end process;
```

To encode the states of a state machine, there are several available styles. The default style is binary. Its advantage is that it requires the least number of flip-flops. In this case, with n flip-flops (n bits), up to 2^n states can be encoded. The disadvantage of this encoding scheme is that it requires more logic and is slower than the others. At the other extreme is the onehot encoding style, which uses one flip-flop per state. Therefore, it demands the largest number of flip-flops. In this case, with n flip-flops (n bits), only n states can be encoded. On the other hand, this approach requires the least amount of extra logic and is the fastest.

In this design the binary encode style was used, and a number of flip-flops could be calculated from the following equation: $\log_2 n$ where n is the number of states. Figure 3.6 confirm that. This is a part of RTL (**R**egister **T**ransfer **L**evel) schematic [4] representation, generated by Xilinx Synthesis Technology (XST) [18]. There are 3 flip-flop marked as D1..D3, to store one of five state possibilities (`pre_seqI, seqI, pre_seqII, seqII` and `manual`).

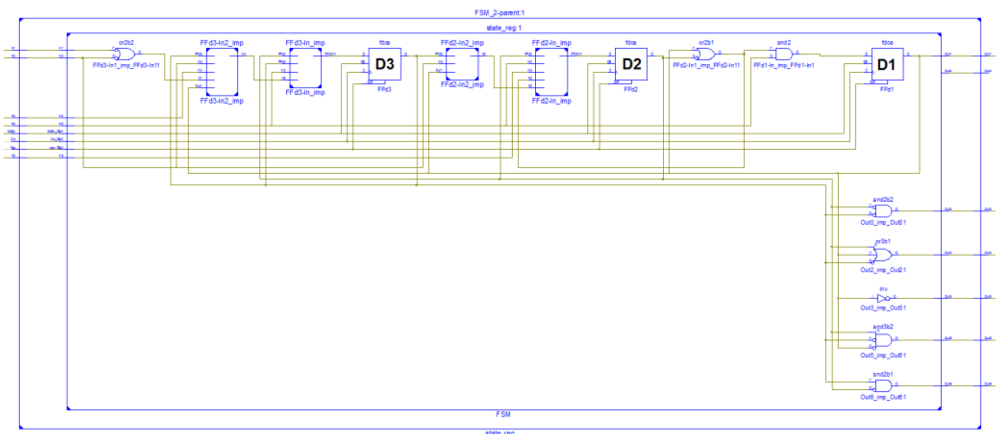


Figure 3.6: A part of RTL scheme shows state encoding.

Upper section will be discussed separately for each state, start with `seqI` state which is presented in the Listing 3.10.

Listing 3.10: A seqI state – **MEMS_ChargeInReduce.vhd**

```
502         ------------------------   SEQUENCE I   ------------------------
503         when seqI =>
504                 DIGS_int(4) <= '0';
505                 LEDS(1) <= '1';
506 
507                 if SLIDE_FILT(3) = '0' then
508                         if secCount(26 downto 23) = secTime  then
509                                 state_next <= pre_seqII;
510                         else state_next <= seqI;
511                         end if;
512                 else state_next <= manual;
513                 end if;
```

After power ON or reset signal program begin from this state. Here there is also possibility to go to manual mode after set a fourth slide button to high which is represent as `SLIDE_FILT(3)` in the program. When a counter `secCount` which will be presented in next section reach predefined value `secTime`,the program goes to `pre_seqII` state to do comparison. This part is presented in the Listing 3.11.

Listing 3.11: A pre_seqII state – **MEMS_ChargeInReduce.vhd**

```
515         ----------------------   PRE-SEQUENCE II   --------------------------
516         when pre_seqII =>
517                 DIGS_int(4) <= '1';
518                 LEDS(1) <= '0';
519
520                 if avgCount(AVGS) = '0' then
521                         if (AVGS_OUT < capSeqI - dataInterval) then
522                                 state_next <= seqII;
523                         else state_next <= seqI;
524                         end if;
525                 else state_next <= pre_seqII; end if;
```

In this state at the beginning the polarization of voltage supply capacitance divider was changed. Next there are collecting a samples to average them, if a sufficient amount was accumulated and permission to average was given program pass to comparison. The state with the lowest ADC data which mean with the lowest MEMS capacitance is chosen. Remaining states (`pre_seqI`, and `seqII`) operate in the same manner and they are shown in the Listing 3.12.

Listing 3.12: A remaining state: pre_seqI and seqII – **MEMS_ChargeInReduce.vhd**

```
490         ----------------------   PRE-SEQUENCE I   -----------------------
491         when pre_seqI =>
492                 DIGS_int(4) <= '0';
493                 LEDS(1) <= '0';
494
495                 if avgCount(AVGS) = '1' then
496                         if (AVGS_OUT < capSeqII - dataInterval) then
497                                 state_next <= seqI;
498                                 else state_next <= seqII;
499                         end if;
500                 else state_next <= pre_seqI; end if;


527         -----------------------   SEQUENCE II   -----------------------
528         when seqII =>
529                 DIGS_int(4) <= '1';
530                 LEDS(1) <= '1';
531
532                 if SLIDE_FILT(3) = '0' then
533                         if secCount(26 downto 23) = secTime then
534                                 state_next <= pre_seqI;
535                         else state_next <= seqII;
536                         end if;
537                 else state_next <= manual;
538                 end if;
```

Listing 3.13 presents a last state in which user can chance supply manually.

Listing 3.13: A manual mode state – **MEMS_ChargeInReduce.vhd**

```
540         -----------------------   MANUAL MODE   -----------------------
541         when manual =>
542                 if CLK_DIV(20) = '1' then LEDS(1) <= '1';
543                 else LEDS(1) <= '0'; end if;
544
545                 if PUSH_FILT(2) = '1' then DIGS_int(4) <= '1';
546                 else DIGS_int(4) <= '0'; end if;
547
548                 if SLIDE_FILT(3) = '0' then state_next <= seqI;
549                 else state_next <= manual; end if;
```

As it was shown in the Figure 3.1 it is possible to reach this state only form sequence I or II with putting a fourth slide button to high. A throbbing second led `LEDS(1)` inform about operations in this state. A third push button can be used to change voltage polarity.

Connected with lower section there is another one register which should be described. It is presented in the Listing 3.14.

Listing 3.14: A sequence timer – **MEMS_ChargeInReduction.vhd**

```
440  -- ------------------------   secCount   ------------------------
441  process (CLK_LF, RST) begin
442          if RST = '1' then
443                  secCount <= (others=>'0');
444          elsif rising_edge(CLK_LF) then
445                  if SLIDE_FILT(3) = '0' then
446                          -- comment that instead next line put this for sim.
447                          if secCount(26 downto 23) = secTime then
448                                  secCount <= secCount;
449                                  if D_DACEA = '1' then
450                                          secCount <= (others=>'0');
451                                  end if;
452                          -- elsif secCount = (secTime & B"000" & X"0_0000") - 1 then
453                                  -- if (SLIDE_FILT(4) = '0') then
454                                          -- secCount <= secCount + 1;
455                                          -- stopStoraging <= '0';
456                                  -- else
457                                          -- stopStoraging <= '1';
458                                  -- end if;
459                          else secCount <= secCount + 1;
460                          end if;
461                  else
462                          secCount <= (others=>'0');
463                  end if;
464          end if;
465  end process;
```

When `secCount` reach `secTime` value counter is stopped and reset only after asserted a control D signal. When it happened in the same time a `state_next` register will be updated. The `secCount` could be stopped just before achieving a `secTime` value by set a fifth slide button high. This opportunity was introduced for changing a capacitance just before switching and

interfere thereby in normal mode. Finally this option was removed due to easier way to force a sequence charge in normal mode. Difference between capacitance in two mode could be changed by the offset. As it was describe in the Figure 2.17 in 2.4.2 section, in sequence II output from capacitance divider pass through amplifier before goes to converter. By changing a offset in this amplifier user can manipulate a capacitance values in seqII, and force staying in different modes.

Before sequence changing storage a previous capacitance value is necessary. It was implemented with two D-type flip-flops as shown in the figure and listening below.

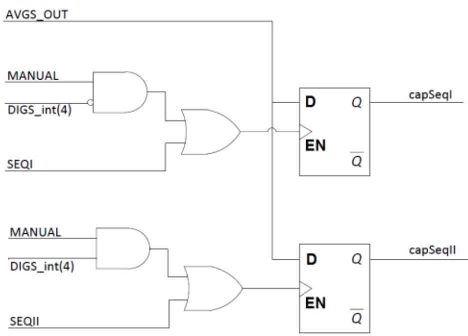


Figure 3.7: Capacitance storage section

Listing 3.15: A VHDL implementation of capacitance storage – **MEMS_ChargeInReduce.vhd**

```
553  -- to storage a ADC data before switch
554  process (CLK_LF) begin
555     if rising_edge(CLK_LF) then
556         if (state_reg = seqI) or (state_reg = manual and DIGS_int(4) = '0')
557                then capSeqI <= AVGS_OUT;
558         elsif (state_reg = seqII) or (state_reg = manual and DIGS_int(4) = '1')
559                then capSeqII <= AVGS_OUT; end if;
560     end if;
561  end process;
```

3.2.6 Do files

In situations where there are repetitive tasks to complete, it is possible to increase productivity with DO files. DO files are scripts that allow to execute many commands at once. The scripts can be as simple as a series of ModelSim commands with associated arguments, or they can be full-blown TCL(**T**ool **C**ommand **L**anguage) programs with variables, conditional execution, and so forth. Appendix D – Source codes presents all do file used in this project.

3.2.7 Program – user communication

This section describe program collaboration with users. There will be depicture, what diodes and 7-segment LED display shown, and for what are push and slide buttons responsible.

The main control is executed by slides buttons. Below in the graph form there will be presented a proper configuration.

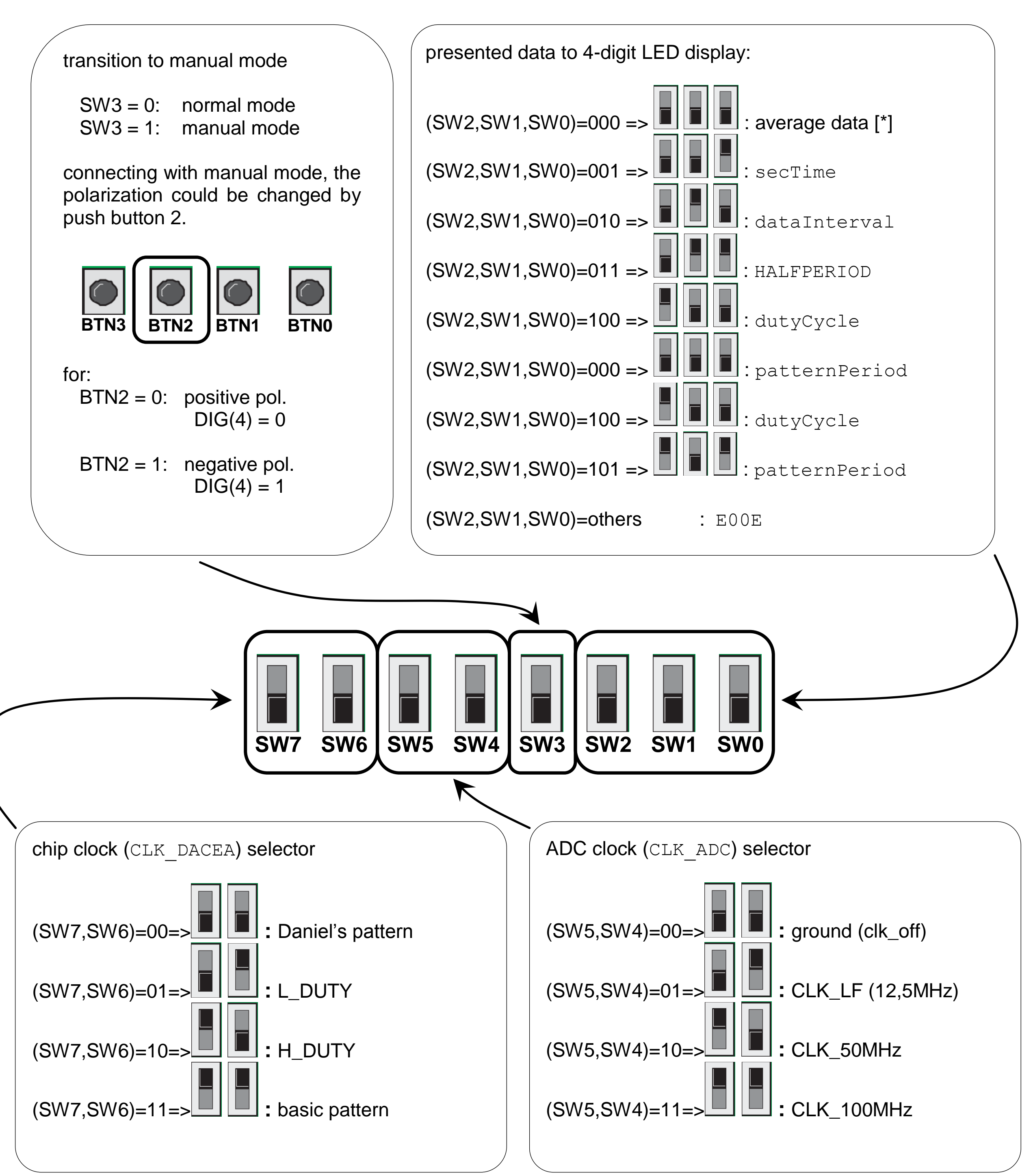


Figure 3.8: Slide switches configuration

3.3 Simulation

This section shows correctness the algorithm operation, and also demonstrate work principle in graphical way.

A changing a sequence is allow only when H (Hold) signal is active high. For that reason a capacitance divider is disconnected from measurement part and prevent a glitches occurrence. This behavior is depicture on Figure 3.9.

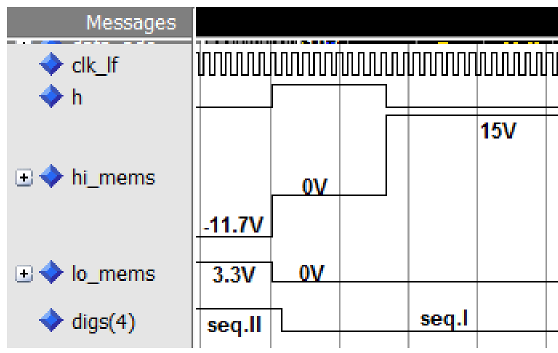


Figure 3.9 Sequence changes when H is high.

The MEMS capacitance could be changed in several ways. The most obvious is by destruction, but unfortunately in this thesis there is no description of this method. Other important way is designed by Daniel Fernandez. A distance between plates is changed with a PWM duty cycle width. By increasing a time of high level PWM the capacitance increase too [8]. The third method to change it, is by charge injection.

Because user can tune a MEMS capacitance, a charge injection phenomena could be see only after changing a power supply, as it was explained in 1.5 section. For that reason we have to capture a ADC data just before and after switch and decide which mode should be selected next.

When ADC data are equal or not exceed a interval value in both sequence the program change a voltage polarity every defined period of time(defined by `secTime` register). The Figure 3.10 presented that situation. The time is set by user because injection speed is different for each MEMS.

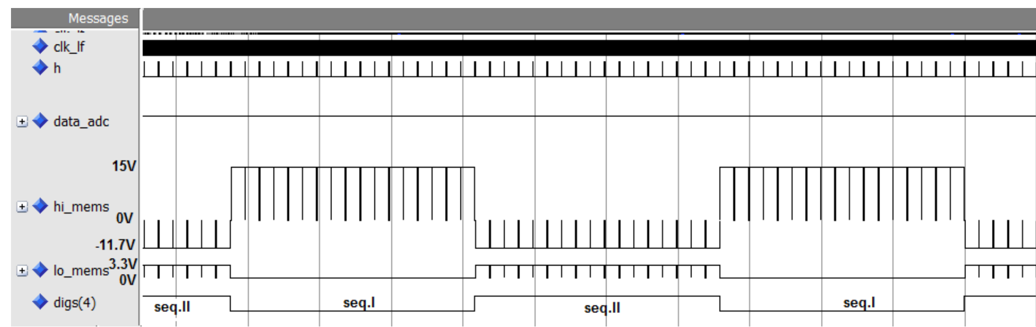


Figure 3.10: Sequence changes every secTime period.

In the Figure 3.12 the data from the ADC are not constant, but increasing due to injected charge. There are register named *charge* which indicate how large load is injected to MEMS structure. This register does not include charge's sign. The *digs(4)* flag which indicate a current sequence, helps to determinate to which pole a capacitor is loaded. This is better explain in the Figure 3.11, and the Figure 1.6 in 1.6 section will also be useful. During each sequence there are discharging from previous pole and charging to opposite one.

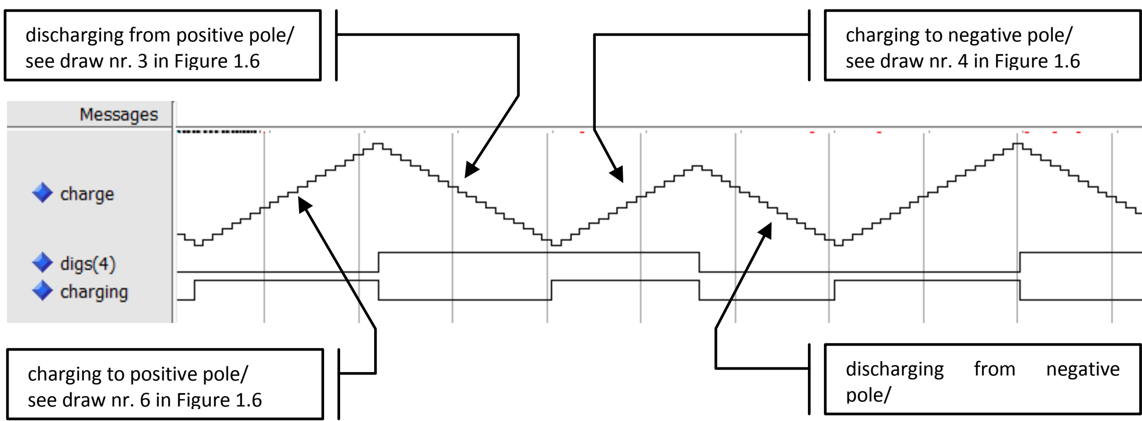


Figure 3.11: Charging sign explanation

Returning to Figure 3.12 there are two artificial addition of load, marked as discharging A and B. This stimulation was performed to show that system is able to stabilize, and also that algorithm reduce a charge to the minimum. In this figure there are another signal – *leds(1)* to indicate what is current state. When *leds(1) = 0* the program is in one of two pre-sequence, in pre_seq_I when *digs(4)=0* and in pre_seq_II when *digs(4)=1.*

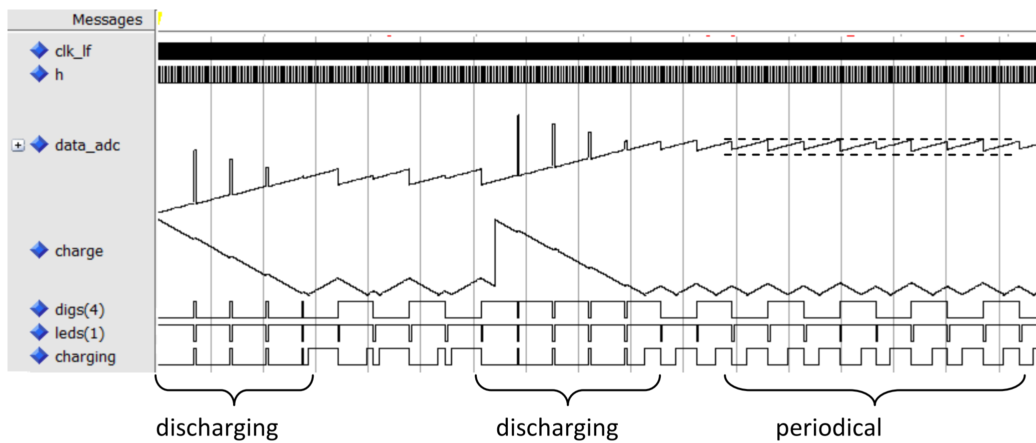
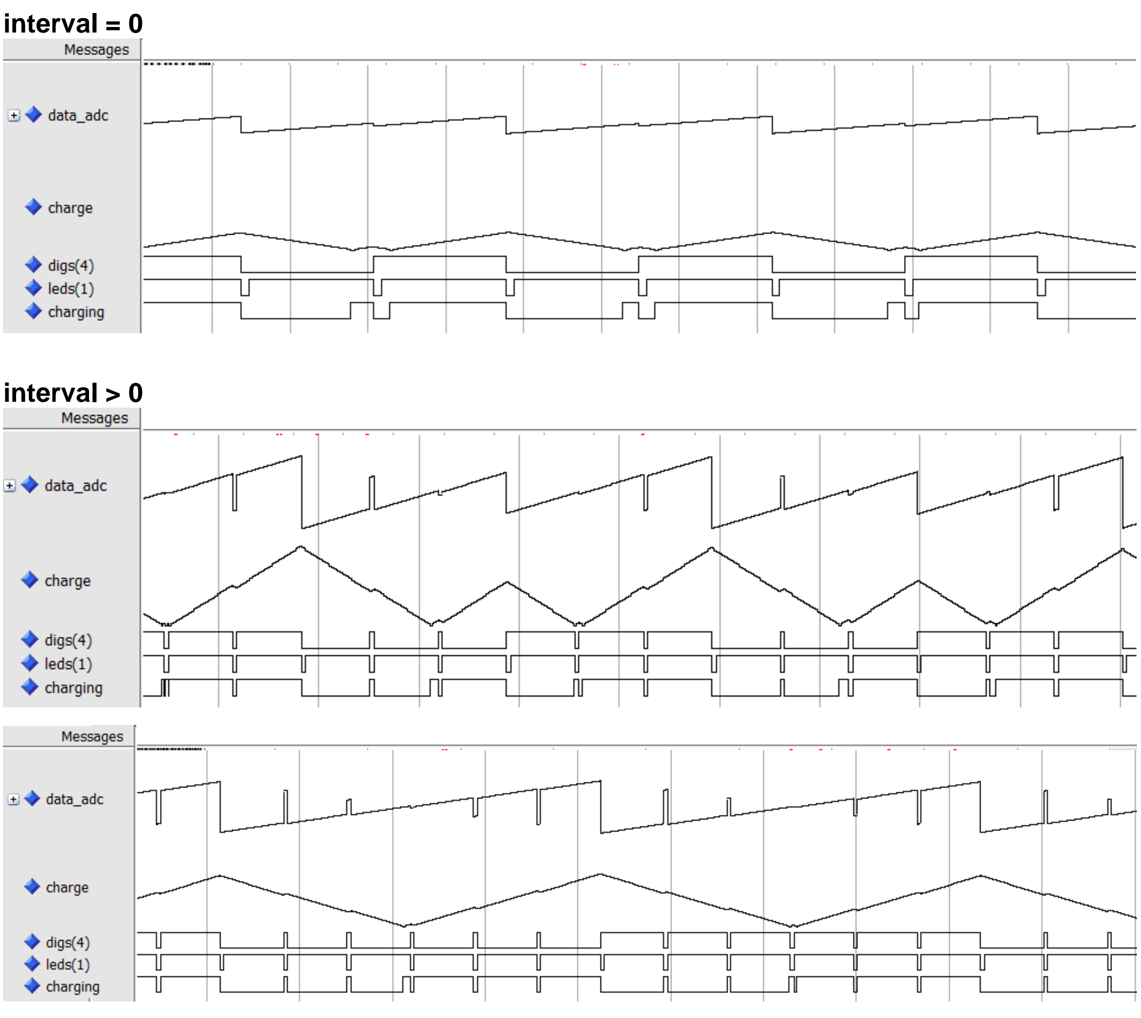


Figure 3.12: Artificial charge addition

After short explanation a few basic signals in design it's time to present a main task. Figures below depicture a ADC data – which mean MEMS capacitance stabilize in different interval times.



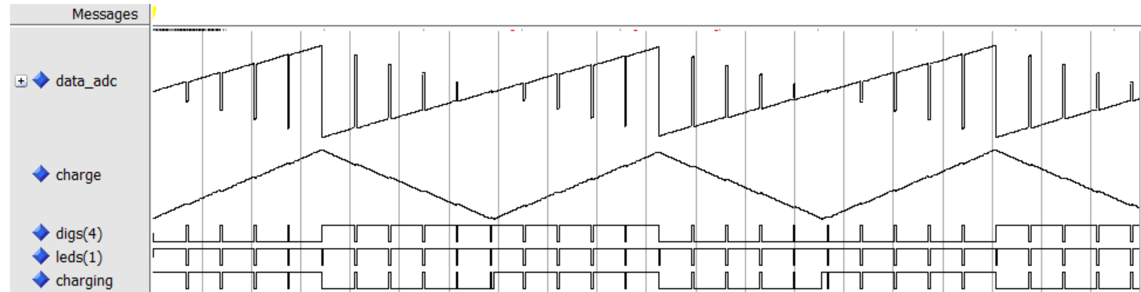


Figure 3.13: Charge reduction for different interval times.

The first curve in the figure above shows that the ADC data are stabilize as fast as possible, in this case for each 1s, and *digs(4)* signal looks like PWM wave with 50% duty cycle. The situation is different for curves where some interval value changes was inserted. This is defined by *dataInterval* register in main program. User can change this value, when it is displayed on 7-segment led display.

In the figures where *dataInterval > 0* there are short switch between sequence for checking if injected charge increase a critical value. If that happens program change sequence for next 1s, then check and decide again.

ModelSim software allow to do a long time simulation but it takes a lot of time. For that reason the switching between sequence was shorted.

3.4 Measurements

As it commonly happens in real world, simulations and experimental results may differ from each other in many cases. This follows from the fact that, all phenomena that occur in reality are not yet known. Of course it is beautiful and provides jobs for researches, whose main goal is making life easier and safer.

One of the first measurement was to review the operation of the system and test correct operation of Delirium-I-FEAEL board presented in the Figure 2.4. It gives reliability in work with capacitance estimation and driving circuit, designed by Daniel Fernandez [7]. First, system was driving by external generators sources, not by FPGA, analog to digital converter was removed as well. So whole system consist of Delirium-I-FEAEL with MEMS actuator (Figure 2.14) boards and one small with inverter. Then PWM generator was connected to CLKIN_R0 as depicture in Figure 2.6, with 10us period time, and 90% duty cycle. It was observed that gradual reduction of duty cycle, change a capacitance divider’s output voltage. This confirmed the correct operation of initial part of system, and allow to find actuators with good performance.

The next measurements were related to measure the actuator capacitance. This task has been made mainly on the basis of the Daniel's manuscript [5]. Many tests have been carried out, however, here is presented a most interesting one with MEMS connected to 10th terminal (figure attach in appendix C – A MEMS actuators floor-plan.). In the first place the curve voltage-capacitance was determinate with two fixed capacitors. One of them substitute a actuator and its value was increased, whereas the second one, represented the C_S capacitor (see Figure 2.16) remind unchanged. The ADC driver's output voltage was measured and calculated $V_{math} = V_{out+} - V_{out-}$. Observed that it is proportional to the changing capacitor's value. This dependence is presented in the Figure 3.14

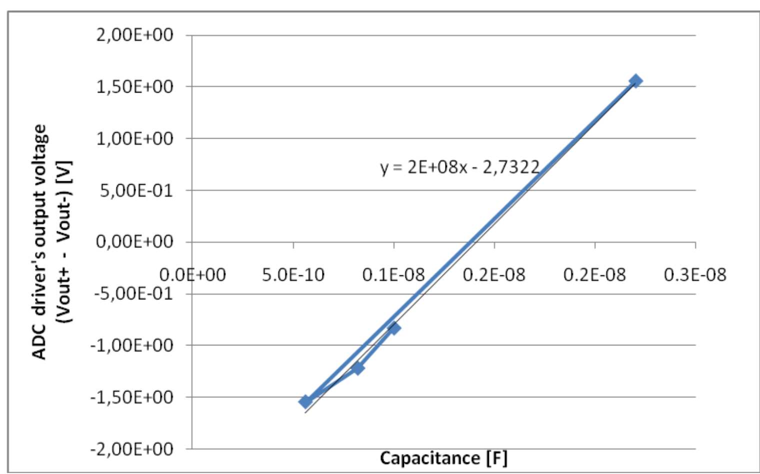


Figure 3.14 Voltage-capacitance curve, obtained from two fixed capacitance divider's output.

The linear trend line was calculated, and its formula as well. In next step these capacitors was replaced by MEMS actuator board (Figure 2.14), but this time changing the supply duty cycle change a MEMS actuator's plate distance. The capacitance was calculated from the derived formula.

Duty Cycle [%]	Vmath = V_{out+}-V_{out-} [mV]	C(t) [pF]
10%	63	14
20%	218	14,7
30%	340	15,4
40%	580	16,6

Table 3.1: The MEMS actuator capacitance calculation.

Further measurements have the task of verifying the correctness of the algorithm, however, previously it is necessary to show that the charge injection phenomenon already exists. In the figure below the courses presents a manually sequencing change, the changing aren't so appreciable. During approximately 6s a voltage change by about 100mV. Both channels presented a ADC driver outputs, and on the bottom there is a DIG(4) signal which indicate a divider supply

mode. The obtaining measurements are confirmation of assumptions erected at the beginning of 1.6 section. Figure 3.7 presents a ADC driver outputs (channel 1 and 2), and DIG(4) signal (D_1 from logic analyzer) – indicates a current sequence.

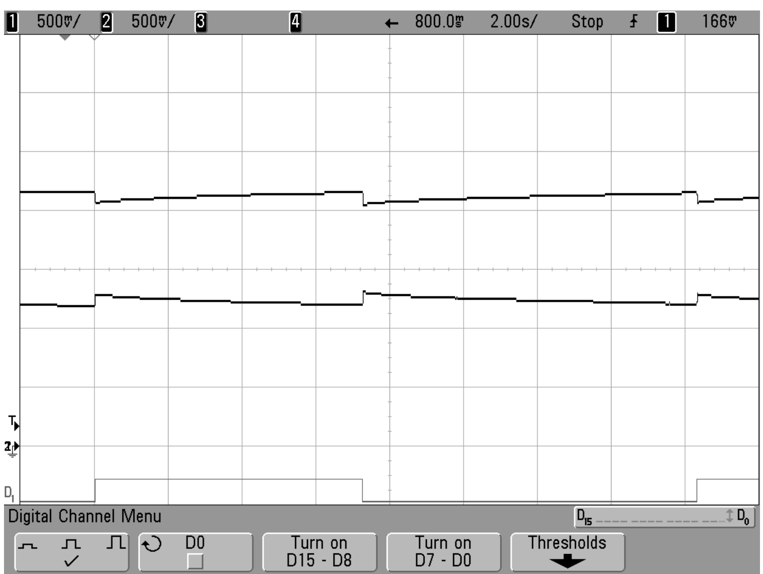


Figure 3.7: Manually mode changing and a charge injection confirmation.

It was obvious that during the measurements and work under the project a lot of problems and concerns occur. Most of them may have been due to inaccurate documentation equipment, as well as novelty and complexity of the design and wide range of material which needs to be achieve before work. Most of them have been resolved through consultation and conversation with the promoters and coordinators, but still not all of them. One of them, associated with high sensitivity device to external disturbance still occur.

In the first phase of work there was two fixed capacitor (instead of MEMS actuator) connected to the MEMS connector (CON24) depicture in the Figure 2.11. Noticed that divider is very sensitive for electromagnetically field. Bringing a hand near to system changes a characteristic curves change too.

There were also one another problem with ADC clock, or measuring equipment. When the ADC clock was switched ON the small interference goes to the output voltage. Observed that the magnitude of this interference depend on oscilloscope cables placement, and got rid of them by using the simplest averaging in the oscilloscope screen. After obtain clean output waveforms something happen to DIG(4) signal, which cannot remain stable in high level, but random change to low on the D edges. Due to a lack of time, which still has been very extended this problem remained unsolved. Figure 3.8 presents a described situation.

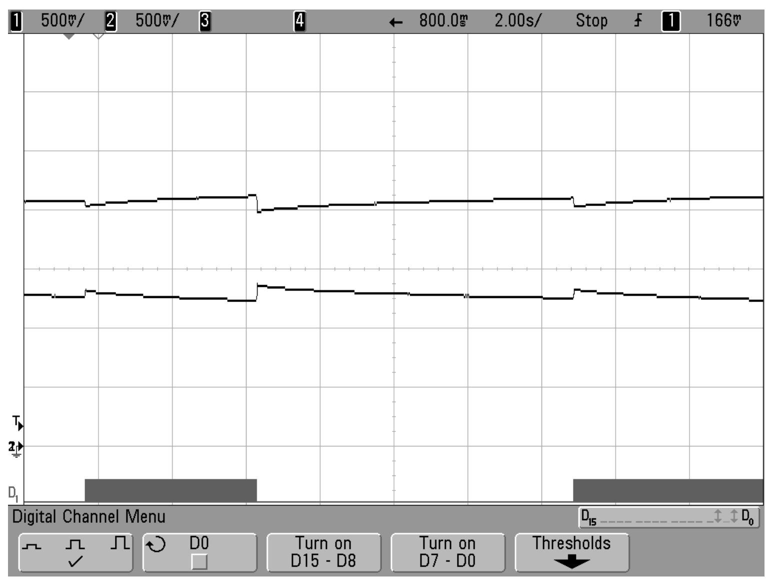


Figure 3.8: Manually mode changing with ADC clock ON.

In the 3.2.7 section there was a description how to change a ADC clock frequency which is applied to the converter. In the further work the source of such interference should be found. A few advise, that could be given know is to check all paths, cause short or bridge find.

The last measurements shows the proper work of system excluding those outside interferences. Curves show the same as in Figure 3.7 but know in automatic mode, with 100MHz ADC clock.

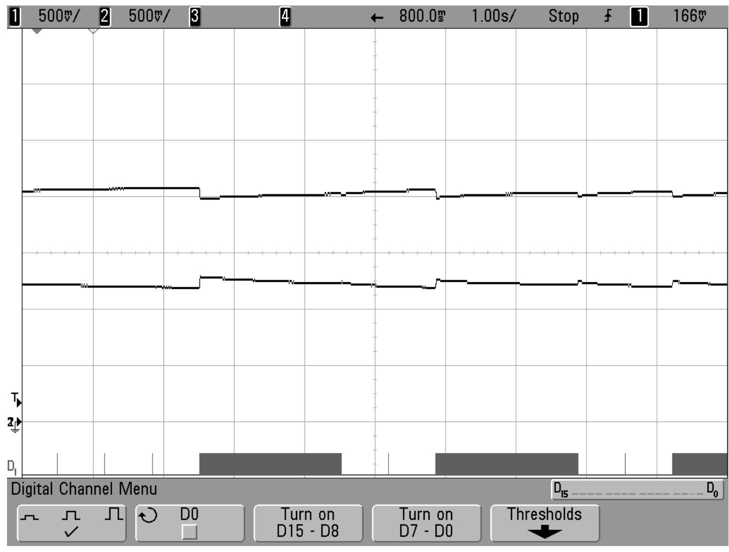


Figure 3.9: Automatic operation

4 Conclusion

Dielectric charging caused by charge injection under voltage stress was observed. The amphoteric nature of traps and its effect on switch operation was noticed under both positive and negative control voltages.

Dielectric charging is a complicated process involving different types of traps with time constants differing by orders of magnitude. The presented in this document charging effect is probably one of many charging effects that can impact switch operation and lifetime.

After lengthy attempts, it contributed to find the system that provide the minimum amount of charge inside the structure of MEMS actuators. A few measurements should be made in a vacuum chamber as a further work. It was stated previously, that these devices are very sensitive to external factors. Atmospheric pressure and humidity in the air negatively affected the accuracy of measurements.

During the execution of this project a lot of interesting problems have been corrected. World is not yet fully known, and ability what engineer must possess, sometimes outweigh its potential. Enormity of knowledge available on the web, might cause that answer to be ambiguous. Therefore, many attempts have been made before finding the right one.

Before the end, it is still worth pointing out what exactly has been done by the author of this work. At the beginning he verified the correctness of the main board, with preliminary measurements included. Then he designed the schematic and layout of multiplexing board, and finally author implemented a charge injection reduce algorithm with simulation and confirmed by measurements.

Summarizing the project has been done in correct manner what is presented in the simulation and measurement part. The main goal which was to implement and experimentally verify various control algorithm for MEMS electrostatic actuators was achieved.

5 Bibliography

[1] Allen, J. J. (2005). *Micro Electro Mechanical System Design .* Broken Sound Parkway NW, USA: Taylor & Francis Group.

[2] Ashenden, P. J. (2008). *The designer's guide to VHDL.* Burlington: Morgan Kaufman Publishers.

[3] Chu, P. P. (2008). *FPGA prototyping by VHDL examples.* Hobokon: Wiley-Interscience.

[4] Chu, P. P. (2006). *RTL hardware designe using VHDL.* Hoboken: Wiley-Interscience.

[5] D. Fernández, Jordi Madrenas. Pulse-Drive and Capacitive Measurement Circuit For MEMS Electrostatic Actuators, Analog Integrated Circuits and Signal Processing. Accepted for publication.

[6] D. Molinero, R. C. (2006). *Dielectric charge measurements in capacitive microelectromechanical switches.* Barcelona, Spain.

[7] Daniel Fernándeza, Jordi Madrenas. (2008). A Self-Test and Dynamics Characterization Circuit for MEMS Electrostatic Actuators., (pp. 3-4). Barcelona.

[8] Horsley, D. (2004). *Patent nr US 6,674,383 B2.* Berkeley, CA (US).

[9] Carter J., Cowen A., MEMSCAP Inc. (2005). *PolyMUMPs Design Handbook.* Rev. 11.0.

[10] Goldsmith C., Ehmke J., (2001). *Lifetime characterization of capacitive RF MEMS switches.,* Dig. IEEE Int. Microwave Symp., (pp. 227-230).

[11] Krick D. T., P. Lenahan, (Oct. 1988.) *Electrically active point defects in amorphous silicon nitride: An illumination and charge injection study*, (pp. 3558-3563).

[12] Korvink, J. G., & Korvink, J. G. (2006). *MEMS: A Practical Guide to Design, Analysis, and Applications.* Freiburg, Germany: Springer-Verlag GmbH & Co. KG.

[13] Lyons, R. G. (2007). *Streamlining Digital Signal Processing A Tricks of the Trade Guidebook.* Piscataway, New Jearsy : IEEE Press .

[14] Lyons, R. G. (2004). *Understanding Digital Signal Processing, Second Edition.* Upper Saddle River, New Jersey: Bernard Goodwin.

[15] M.-C. Lu, G. F. (2004). Position Control of Parallel-Plate Microactuators for Probe-Based Data Storage, Microelectromechanical Systems. pp 759–769. Journal of 13 (5).

[16] Pedroni, V. A. (2004). *Circuit Design with VHDL.* Cambridge, Massachusetts: MIT Press.

[17] Senturia, S. D. (2002). *Microsystem Design.* New York: Kluwer Academic Publishers.

[18] Xilinx. (2009). *PlanAhead User Guide.*

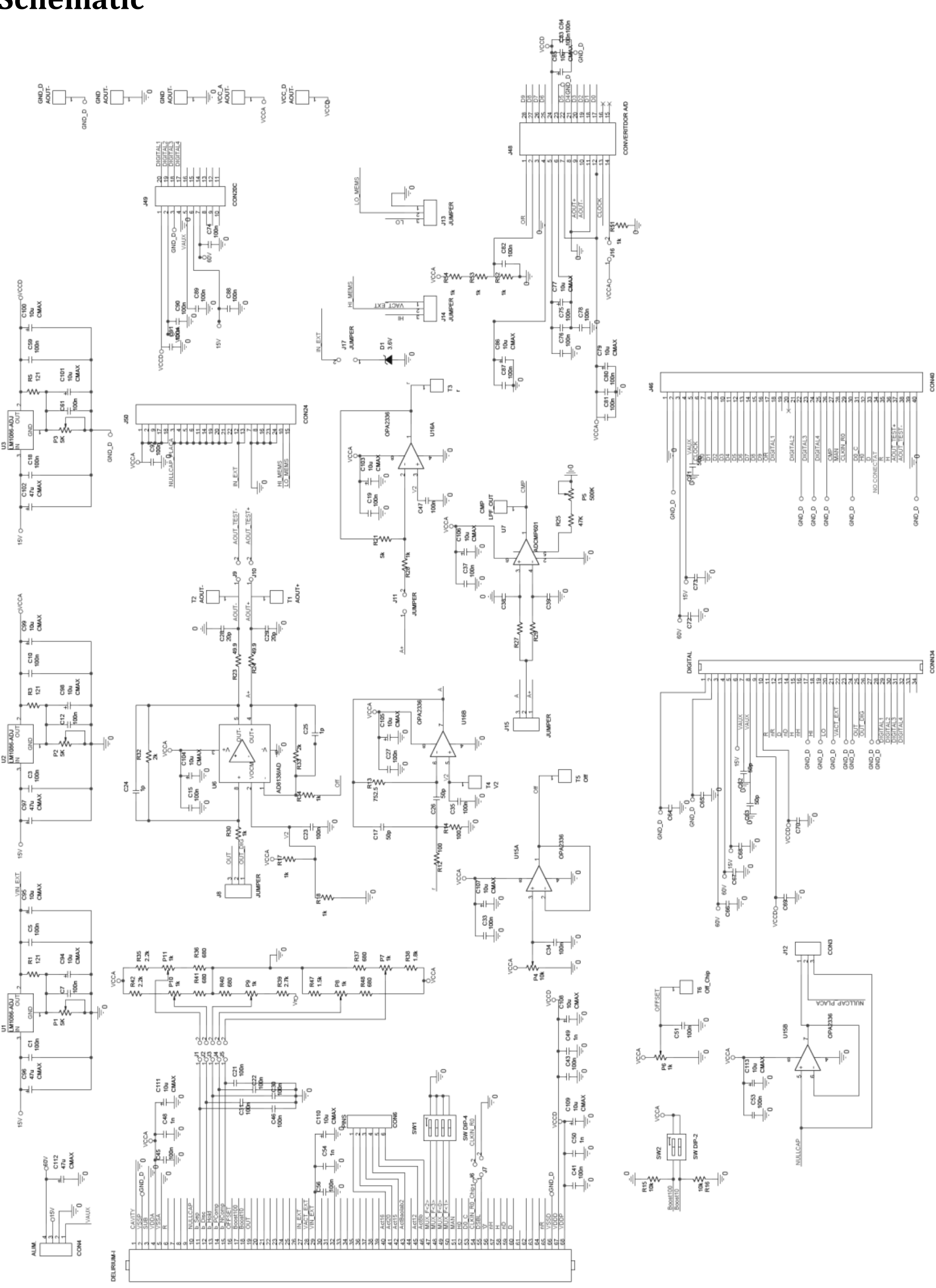
[19] Xilinx. (2005). *Spartan-3 Starter Kit Board User Guide.*

[20] Xilinx. (2006). *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs.* Application Note: Spartan-3 and Spartan-3L FPGA Families.

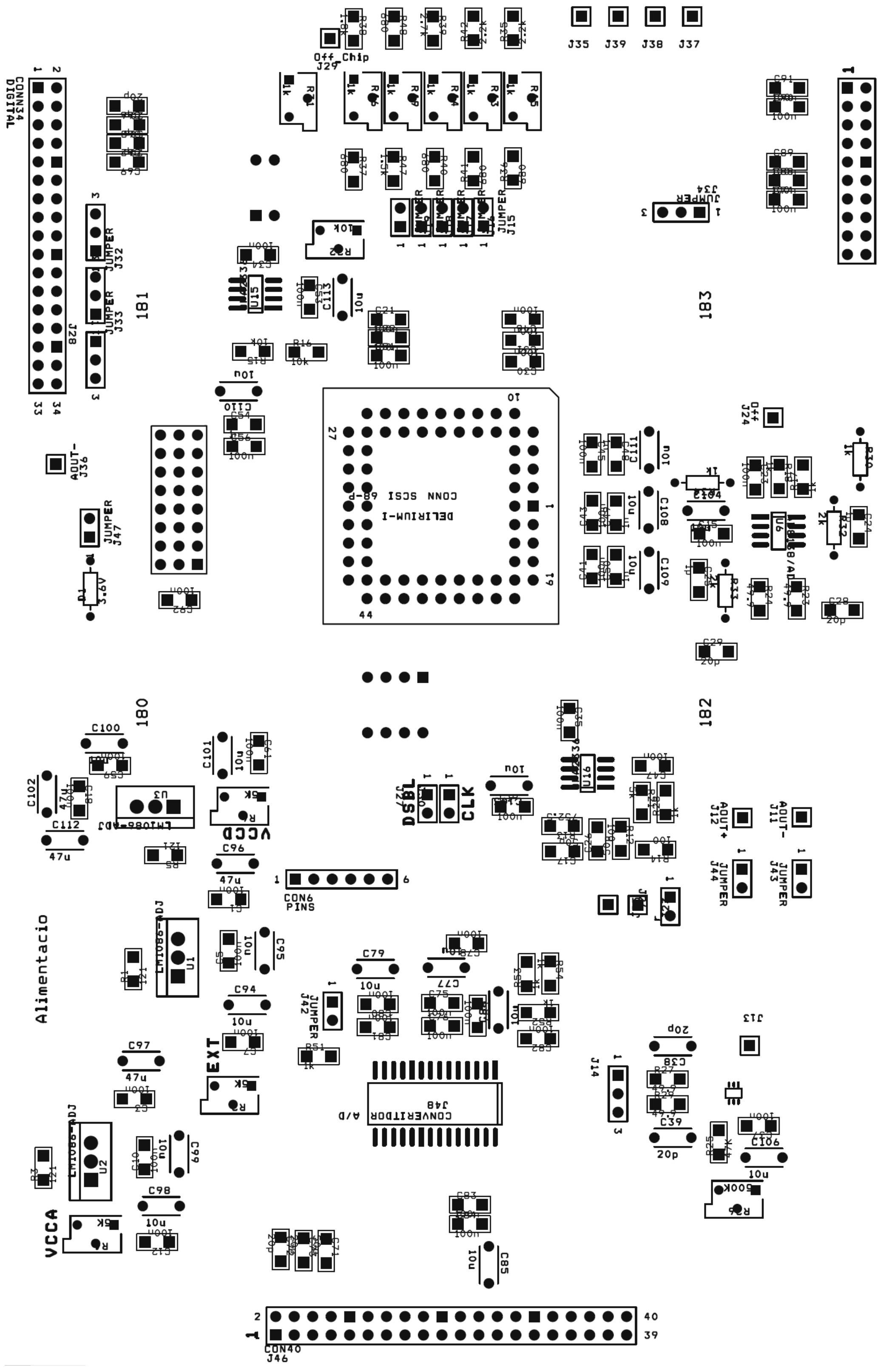
Appendixes

A – DELIRIUM-I-FEAEL documentation.

Schematic

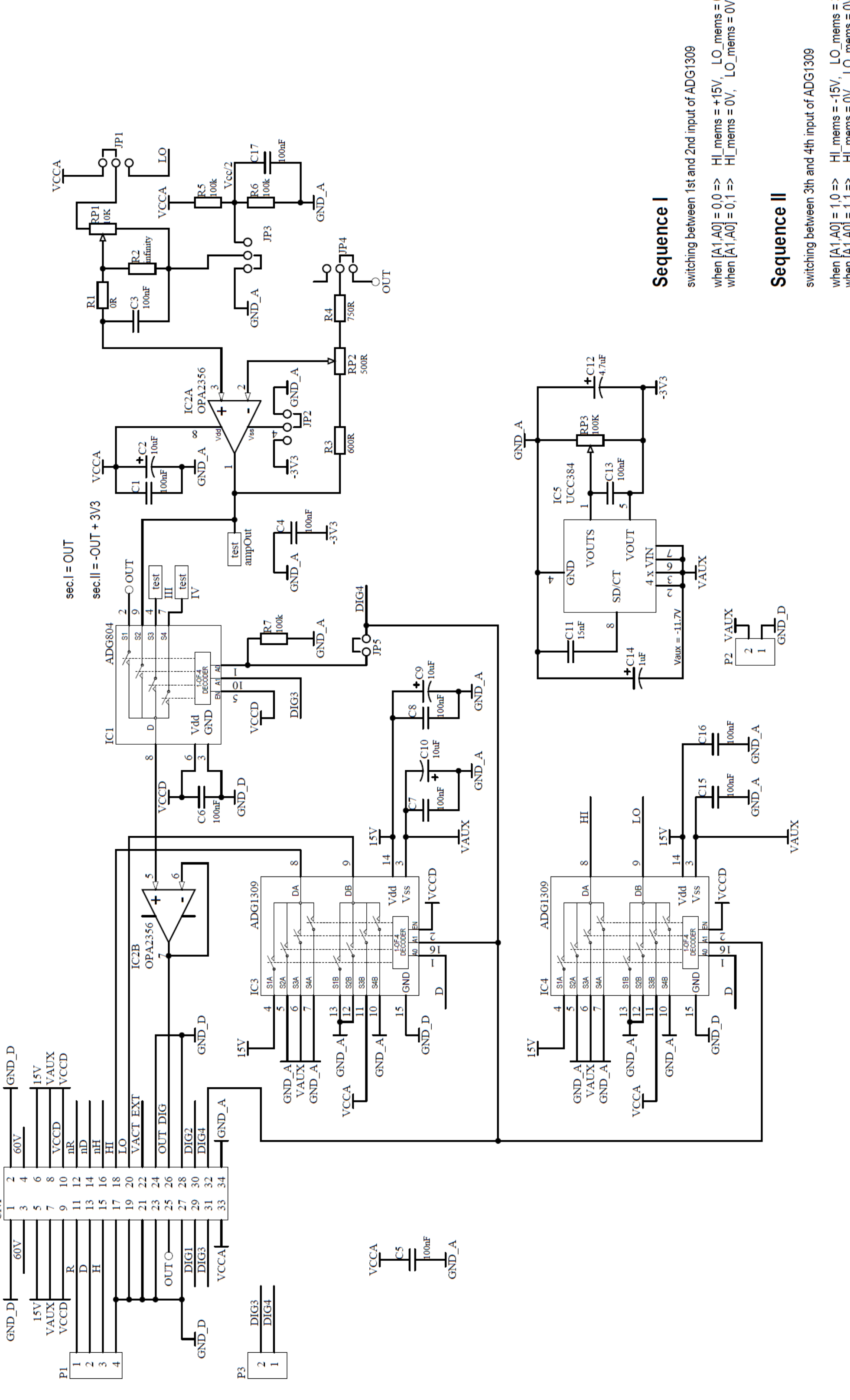


Reference Design Assembly Drawing



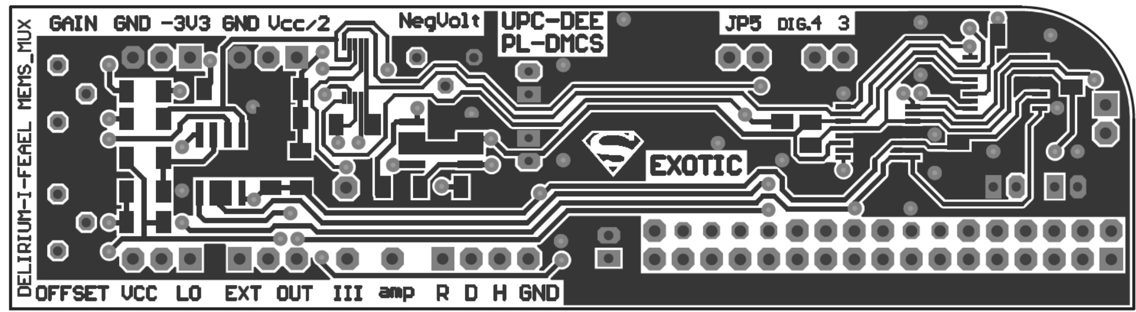
B – Multiplexing board documentation

Schematic

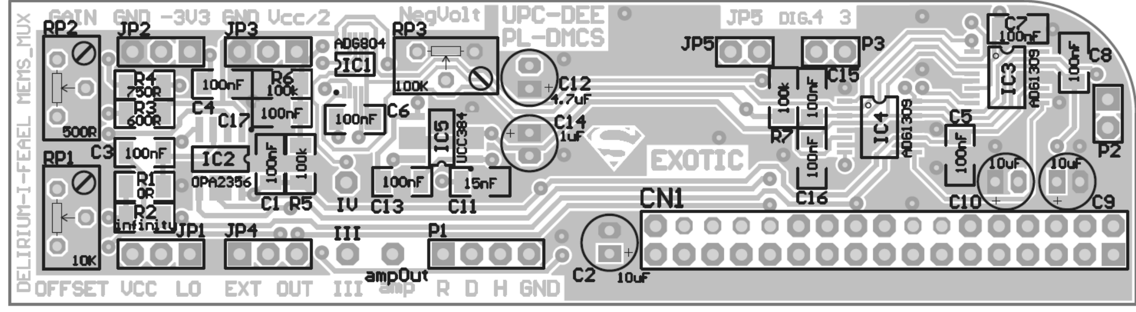


Reference Design Assembly Drawing

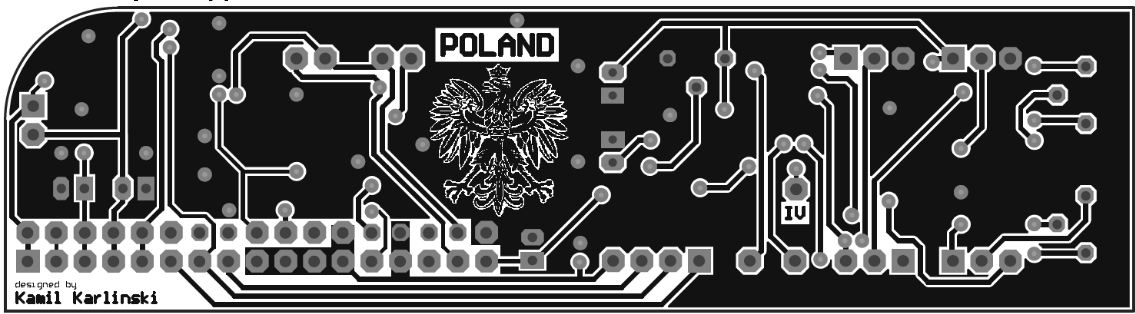
Top Layer Copper



Top Layer Assembly



Bottom Layer Copper



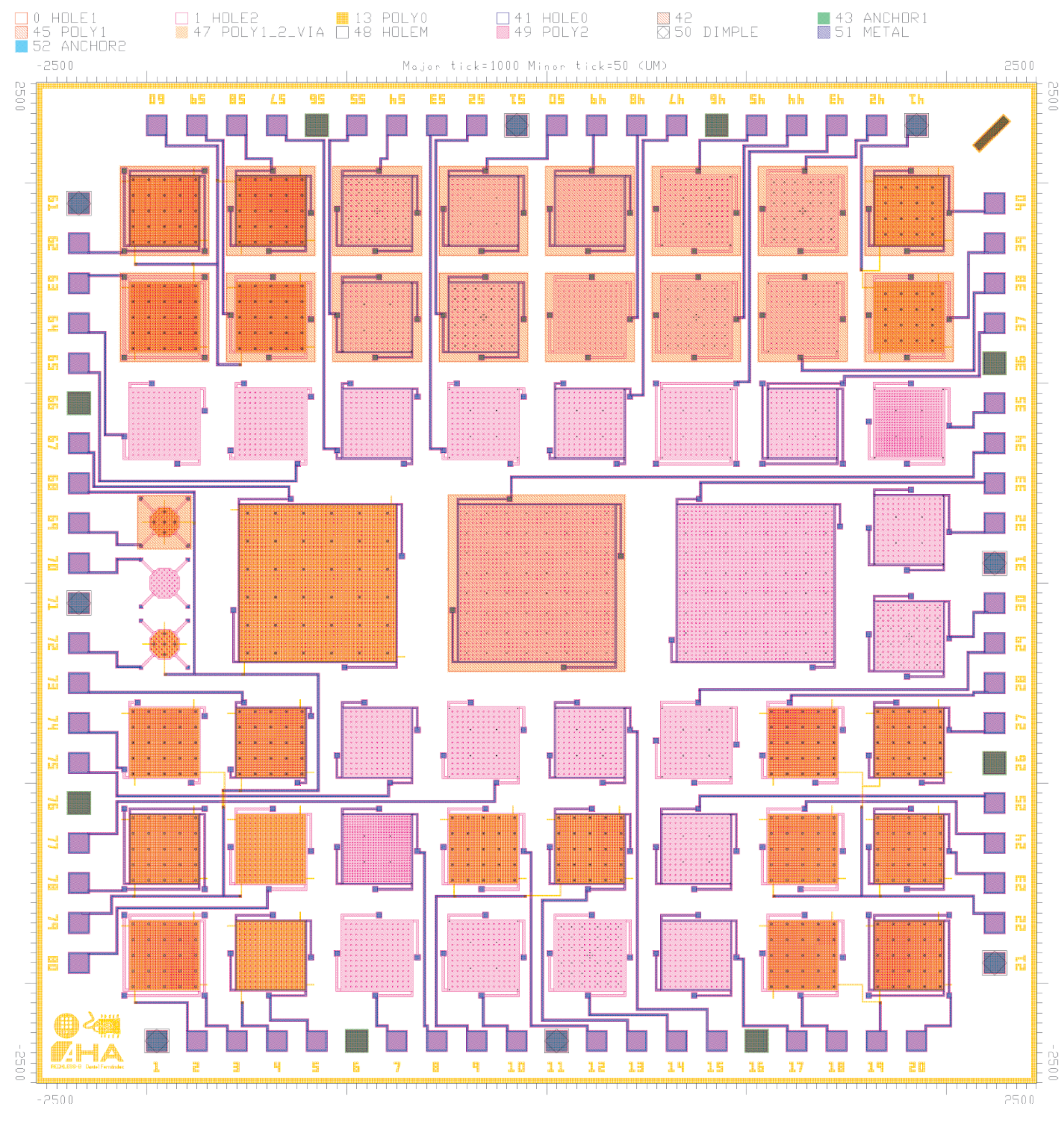
Bottom Layer Assembly

There is no components on that layer.

Bills of materials

Comment	Description	Designator	Footprint	LibRef	Quantity
test Piont		ampOut, III, IV	test PAD	test Piont	3
100nF	Capacitor	C1, C3, C4, C5, C6, C7, C8, C13, C15, C16, C17	C1206	C	11
10uF	Polarized Capacitor (Radial)	C2, C9, C10	C poly 2/5	C Pol1	3
15nF	Capacitor	C11	C1206	C	1
4.7uF	Polarized Capacitor (Radial)	C12	C poly 2/5	C Pol1	1
1uF	Polarized Capacitor (Radial)	C14	C poly 2/5	C Pol1	1
Header 34 2-rows	Header, 26-Pin	CN1	HDR2X17 2.54MM	Header 34 2-rows	1
ADG804	4-Channel ±15 V/+12 V Multiplexers	IC1	MSOP-10	ADG804	1
OPA2356	dual op. amp., 200MHZ GBW, 360V/us, rail-to-rail	IC2	SO-8	OPA2356	1
ADG1309	4-Channel ±15 V/+12 V Multiplexers	IC3, IC4	TSSOP-16	ADG1309	2
UCC384	LOW-DROPOUT 0.5-A NEGATIVE LINEAR REGULATOR	IC5	SO-8	UCC384	1
Jumper 3pin		JP1, JP2, JP3, JP4	HDR1x3 2.54mm	Jumper 3pin	4
Jumper 2pin		JP5	HDR1x2 2.54mm	Jumper 2pin	1
Header 4	Header, 4-Pin	P1	HDR1X4 2.54MM	Header 4	1
Header 2	Header, 2-Pin	P2, P3	HDR1x2 2.54mm	Header 2	2
0R	Resistor	R1	C1206	R	1
infinity	Resistor	R2	C1206	R	1
600R	Resistor	R3	C1206	R	1
750R	Resistor	R4	C1206	R	1
100k	Resistor	R5, R6, R7	C1206	R	3
10K	Potentiometer	RP1	T93_B	Pot	1
500R	Potentiometer	RP2	T93_B	Pot	1
100K	Potentiometer	RP3	T93_B	Pot	1

C – A MEMS actuators floor-plan.



D – Source codes

MEMS_ChargeInReduce.vhd – top level

```
----------------------------------------------------------------------------------
--         ######################################      JUNE    VERSION      ######################################
-- Company: Politechnical University of Catalonia (UPC)
-- Engineer: Kamil Karliński / Daniel Fernández (dfernan@eel.upc.edu)
--
-- Create Date:    17:59:21 03/23/2010
-- Design Name:
-- Module Name:    MEMS_ChargeInReduce - Behavioral
-- Project Name:
-- Target Devices: DACEA platform
-- Tool versions:
-- Description: Algorithm to reduce the failure mechanizm in elektrostatic actuators like CHARGE INJECTION
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity MEMS_ChargeInReduce is
    Port ( CLK_IN : in  STD_LOGIC;
           CLK_ADC : out  STD_LOGIC;
           DATA_ADC : in  STD_LOGIC_VECTOR (9 downto 0);
                               CLK_DACEA : out STD_LOGIC;
                               BCD_OUT : out  STD_LOGIC_VECTOR (7 downto 0);
                               BCD_SEL : out  STD_LOGIC_VECTOR (3 downto 0);
                               FILTER_OUT : out STD_LOGIC_VECTOR (10 downto 0);
                               CAPTURE : out STD_LOGIC;
                               RST : in STD_LOGIC;
                               H_DACEA : in STD_LOGIC;
                               D_DACEA : in STD_LOGIC;
                               R_DACEA : in STD_LOGIC;
                               CMPOUT : out STD_LOGIC;
                               SLIDE_SWITCH : in STD_LOGIC_VECTOR (7 downto 0);
                               PUSH_SWITCH : in STD_LOGIC_VECTOR (2 downto 0);
--                             TxD : out STD_LOGIC;
                               LEDS : out STD_LOGIC_VECTOR (7 downto 0);
--                             VGA_HS, VGA_VS, VGA_RED, VGA_GRN, VGA_BLU : out STD_LOGIC;
                               OR_ADC : in STD_LOGIC;
                               MAN : out STD_LOGIC;
                               H0 : out STD_LOGIC;
                               D0_C : out STD_LOGIC;
                               DIGS : out STD_LOGIC_VECTOR (4 downto 1)
            );
end MEMS_ChargeInReduce;

architecture Behavioral of MEMS_ChargeInReduce is

 type state_type is (pre_seqI, seqI, pre_seqII, seqII, manual);
 signal state_reg, state_next : state_type := seqI;

 signal CLK_50MHz : STD_LOGIC;
 signal CLK_100MHz : STD_LOGIC;
 signal CLK_100MHzI : STD_LOGIC;
 signal CLK_LF : STD_LOGIC;
 signal CLK_DIV : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS => '0');
 signal CLK_DACEAI : STD_LOGIC;

 signal H_POS : STD_LOGIC;    -- it's a delayed HOLD signal
 signal DATA_ADC_SIGNED, SAHP_OUT, SUBS_OUT, AVGS_OUT, SUBS2_OUT, MAX, MIN : STD_LOGIC_VECTOR (10 downto 0) :=
(others=>'0');
 signal capSeqI, capSeqII : STD_LOGIC_VECTOR (10 downto 0) := (others=>'0');
 signal AVGS_OUT_disp : STD_LOGIC_VECTOR (10 downto 0) := (others=>'0');
 signal AVGS_OUT_dispII : STD_LOGIC_VECTOR (15 downto 0) := (others=>'0');
 signal BEGINCAP_MAX, BEGINCAP_MAX_DLY : STD_LOGIC;
 signal DISPLAY_DATA_SSD : std_logic_VECTOR(15 downto 0);
 signal DOTS : STD_LOGIC_VECTOR (3 downto 0);
 signal SLIDE_FILT : STD_LOGIC_VECTOR (7 DOWNTO 0) := (OTHERS => '0');
 signal PUSH_FILT : STD_LOGIC_VECTOR (3 DOWNTO 0) := (OTHERS => '0');
 signal SELECTOR : std_logic_vector(1 downto 0) := (OTHERS => '0');
 signal COMP : STD_LOGIC;
 signal L_DUTY : STD_LOGIC;
 signal H_DUTY : STD_LOGIC;
 signal PRE_CLK_DACEA : STD_LOGIC;
 signal LOCK, nLOCK, nLOCK_DLY : STD_LOGIC;

 signal secCount : STD_LOGIC_VECTOR (26 DOWNTO 0) := (others=>'0');
```

```
 signal secTime : STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001"; -- variable, indicate a each sequence period,

                            -- approximate in 1 second unit
 constant AVGS : NATURAL RANGE 0 to 8 := 5;                 --it is a power of 2,
                  --indicate how many samlpes was taken to avarage process (in this case 2^2=4)
 signal avgCount : STD_LOGIC_VECTOR (AVGS DOWNTO 0) := (others=>'0');
          --not (AVGS-1 DOWNTO 0) because last bit (MSB) avgCout(AVGS) - can indicate when
                  -- averaging module collect all 2^avgs samples to process
 signal stopStoraging : STD_LOGIC := '0';
 signal dataInterval : NATURAL RANGE 0 to 512 := 10;
 signal DIGS_int : STD_LOGIC_VECTOR (4 DOWNTO 1) := (others=>'0');

 signal dutyCycle : NATURAL range 1 to 1023 := 10;                          --
 signal patternPeriod : NATURAL range 1 to 1023 := 250;     -- 125 => period = 10us with CLK_IF = 12.5MHz
 signal MY_CLK_DACEA : STD_LOGIC;

 constant PREHOLD_TIME : NATURAL RANGE 0 TO 255 := 2;       --the time which we add at the beginig of H_DACEA to delay
 constant HPOS_TIME : NATURAL RANGE 0 TO 255 := 70;         --the time which we add at the end of H_DACEA to delay
 constant SSFACTOR : NATURAL RANGE 2 to 5 := 3;
 constant MAXLOCK : NATURAL RANGE 1 to 131000 := 15000;
 constant GRACE : NATURAL RANGE 1 to 8191 := 200/(2**SSFACTOR);
 signal   HALFPERIOD : NATURAL RANGE 1 to 8191 := 216/(2**SSFACTOR);
 signal   HPadjSpeedUp : NATURAL RANGE 1 to 2047 := 1;
 signal   adjSpeedUp : NATURAL RANGE 1 to 63 := 1;

 constant LOW_DCP : STD_LOGIC_VECTOR(1023 downto 0) := (0=>'1', 1=>'1', 2=>'1', OTHERS=>'0');
 constant HIGH_DCP : STD_LOGIC_VECTOR(255 downto 0) := (0=>'0', 1=>'0', 2=>'0', 3=>'0', 4=>'0', 5=>'0', 6=>'0',
7=>'0',
                                                       8=>'0', 9=>'0', 10=>'0', 11=>'0', 12=>'0', 13=>'0', 14=>'0',
15=>'0',
                                                       16=>'0', 17=>'0', 18=>'0', 19=>'0', 20=>'0', 21=>'0', 22=>'0',
23=>'0',

                                                   OTHERS=>'1');
begin
--==================================================
-- DCM instantiate
--==================================================
   DCM1 : DCM
   generic map (
      CLKDV_DIVIDE => 10.0, --  Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
                          --     7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
      CLKFX_DIVIDE => 2**(SSFACTOR),   --  Can be any integer from 1 to 32
      CLKFX_MULTIPLY => 2, --  Can be any integer from 1 to 32
      CLKIN_DIVIDE_BY_2 => FALSE, --  TRUE/FALSE to enable CLKIN divide by two feature
      CLKIN_PERIOD => 20.0,            --  Specify period of input clock
      CLKOUT_PHASE_SHIFT => "NONE", --  Specify phase shift of NONE, FIXED or VARIABLE
      CLK_FEEDBACK => "2X",          --  Specify clock feedback of NONE, 1X or 2X
      DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS", --  SOURCE_SYNCHRONOUS, SYSTEM_SYNCHRONOUS or
                                             --     an integer from 0 to 15
      DFS_FREQUENCY_MODE => "LOW",     --  HIGH or LOW frequency mode for frequency synthesis
      DLL_FREQUENCY_MODE => "LOW",     --  HIGH or LOW frequency mode for DLL
      DUTY_CYCLE_CORRECTION => TRUE, --  Duty cycle correction, TRUE or FALSE
      FACTORY_JF => X"C080",           --  FACTORY JF Values
      PHASE_SHIFT => 0,          --  Amount of fixed phase shift from -255 to 255
      STARTUP_WAIT => FALSE) --  Delay configuration DONE until DCM LOCK, TRUE/FALSE
   port map (
      CLK0 => CLK_50MHz,     -- 0 degree DCM CLK ouptput
      CLK2X => CLK_100MHzI,   -- 2X DCM CLK output
      CLKFX => CLK_LF,    -- DCM CLK synthesis out (M/D)
      CLKFB => CLK_100MHz,    -- DCM clock feedback
      CLKIN => CLK_IN    -- Clock input (from IBUFG, BUFG or DCM)
   );

   BUF1: BUFG port map (I => CLK_100MHzI, O => CLK_100MHz);
--==================================================
-- Digital_Signal_Procesing components instantion
--==================================================
---------------    Delayed of the HOLD (H_DACEA) signal   -------------
HLDP : entity work.delayed_stim(Behavioral)
         generic map (predelay=>PREHOLD_TIME, postdelay=>HPOS_TIME, preventive=>'1')
         port map (
                   clk => CLK_100MHz,
                   inp => H_DACEA,
                   outp => H_POS
         );
------------------- Sample and Hold for ADC data  --------------------
SAHP : entity work.sampleandhold(Behavioral)
         generic map (n=>11)
         port map (
                   clk => CLK_100MHz,
                   hold => H_POS,
                   din => DATA_ADC_SIGNED,
                   dout => SAHP_OUT
         );
------------------- Generic subsampler v2  --------------------
SUBS : entity work.subsampler_v2(Behavioral)
         generic map (n=>11, s=>SSFACTOR)
         port map (
                   clk => CLK_100MHz,
                   clk_out => CLK_LF,
                   din => SAHP_OUT,
                   dout => SUBS_OUT
         );
--------------- Averaging the SUBS_OUT samples  -----------------
AVG : entity work.averaging(Behavioral)
         generic map ( avgs=>AVGS, n=>11)
         port map (
                   clk => CLK_LF,
                   din => SUBS_OUT,
                   dout => AVGS_OUT
         );
--==================================================================
```

```
-- Output pattern generator - the MEMS actuator pulse clock exactly (CLK_DACEA)
--===================================================================
---------------------   main pattern  ---------------------
OUTP : entity work.OUT_PATTERN(Behavioral)
          generic map (maxlock=>MAXLOCK, grace=>GRACE)
          port map (
                    clk => CLK_LF,
                    comp => COMP,
                    halfperiod => HALFPERIOD,
                    lock => LOCK,
                    out_pattern => PRE_CLK_DACEA
          );
---------------------   MY pattern  ---------------------
MyP : entity work.MY_PATTERN(Behavioral)
          port map (
                    clk => CLK_LF,
                    dutyCycle => dutyCycle,
                    patternPeriod => patternPeriod,
                    out_pattern => MY_CLK_DACEA
          );
---------------------   LOW duty pattern  ---------------------
LDCY : entity work.cyclicpattern(structural)
          generic map (n=>LOW_DCP'LENGTH)
          port map (
                    clk => CLK_LF,
                    reset => RST,
                    dada => LOW_DCP,
                    b => L_DUTY
          );
---------------------   HIGH duty pattern  ---------------------
HDCY : entity work.cyclicpattern(structural)
          generic map (n=>HIGH_DCP'LENGTH)
          port map (
                    clk => CLK_LF,
                    reset => RST,
                    dada => HIGH_DCP,
                    b => H_DUTY
          );
--====================================================
-- 7-segment LED display component instantion
--====================================================
DISP : entity work.Display(Behavioral)
          port map (
                    value => DISPLAY_DATA_SSD,
                    dots => DOTS,
                    en => "1111",
                    clkin => CLK_DIV(10),
                    bcd_out => BCD_OUT,
                    bcd_sel => BCD_SEL
          );
--====================================================
-- processes
--====================================================
-- General-purpose clock pre-scaler
process (CLK_LF)
 begin
    if rising_edge(CLK_LF) then
        CLK_DIV<=CLK_DIV+1;
    end if;
 end process;

 -- Seven-Segment LED Display refresh delay
 process (CLK_DIV(19)) begin
          if rising_edge(CLK_DIV(19)) then       -- in freq 12.5MHz, the rising_edge on proper bits in CLK_DIV occur
after time below
                    -- AVGS_OUT_disp <= AVGS_OUT;          -- rising_edge on bit:(19) after 0.08s (20) after 0.17s;
(21) after 0.34s; (22) after 0.67s; (23) after 1.34s
                    AVGS_OUT_dispII(7 downto 0) <= capSeqI(7 downto 0);
                    AVGS_OUT_dispII(15 downto 8) <= capSeqII(7 downto 0);
          end if;                                                                  -- rising_edge which means
period I count in that way: 2**(a+1)/freq
end process;                                                                -- e.g. in bit(20) 2*(20+1)/12.5MHz =
0.17s

-- Button and switch samplers
process (CLK_DIV(14))
 begin
           if rising_edge(CLK_DIV(14)) then
       SLIDE_FILT<=SLIDE_SWITCH;
       PUSH_FILT<=RST & PUSH_SWITCH;
    end if;
 end process;

-- MAX and MIN detection
process (CLK_LF)
 begin
          if rising_edge(CLK_LF) then
                    if (RST='1' or PUSH_FILT(1)='1') then
                              MAX<=(OTHERS=>'0');
                              MIN<=(10=>'0', OTHERS=>'1');
                    else
                              -- if(SUBS_OUT>=MAX) then MAX<=SUBS_OUT; BEGINCAP_MAX<='1'; else BEGINCAP_MAX<='0'; end
if;
                              -- if(SUBS_OUT<MIN) then MIN<=SUBS_OUT; end if;
                    end if;
          end if;
 end process;

-- Select the output pattern - exactly the MEMS actuator pulse clock
-- SLIDE_SWITCH 7,6 :        00 Normal mode
--                                  01 Test Mode. Apply high voltage only (CLK_DACEA is set low duty cycle)
--                                  10 Test Mode. Apply low voltage only (CLK_DACEA is set high duty cycle)
--                                  11 Test Mode. Disable MEMS clock (CLK_DACEA is set high).
```

```
 SELECTOR<=SLIDE_FILT(7 downto 6);
 with SELECTOR select
         CLK_DACEAI<=
                  PRE_CLK_DACEA when "00",
                  L_DUTY when "01",
                  H_DUTY when "10",
                  MY_CLK_DACEA when others;

-- Mux for select which data to display depending on the switches
-- SLIDE_FILT 1,0 :                  00 Displays a capacitance in both seq.:          left ->  capSeqII,
         right -> capSeqI
--                                   01 Displays a secTime value (time between seq. switch)
                                                                        [*]
--                                                              10 Displays the dataInterval value
                                                                                                       [*]
--                                   11 Displays the half period
                                                                                              [*]
-- [*] - can be adjusted with BUTTON 0 and 1 in this mode
process (CLK_LF)
 begin
         if (CLK_LF='1' and CLK_LF'event) then
                  case SLIDE_FILT(2 downto 0) is
                           when "000" =>
                                    DOTS <= B"0000";
                                    -- DISPLAY_DATA_SSD<=ext(SUBS_OUT(10 downto 0),16);
                                    DISPLAY_DATA_SSD <= ext(AVGS_OUT_dispII,16);
                           when "001" =>
                                    DOTS <= B"0000";
                                    -- DISPLAY_DATA_SSD<=ext(MAX,16);
                                    DISPLAY_DATA_SSD<=ext(secTime,16);
                           when "010" =>
                                    DOTS <= B"0000";
                                    DISPLAY_DATA_SSD<=std_logic_vector(to_unsigned(dataInterval,16));
                           when "011" =>
                                    DOTS <= B"0000";
                                    DISPLAY_DATA_SSD<=std_logic_vector(to_unsigned(HALFPERIOD,16));
                           when "100" =>
                                    DOTS <= B"0000";
                                    DISPLAY_DATA_SSD<=std_logic_vector(to_unsigned(dutyCycle,16));
                           when "101" =>
                                    DOTS <= B"0000";
                                    DISPLAY_DATA_SSD<=std_logic_vector(to_unsigned(patternPeriod,16));
                           when others =>
                                    DOTS <= B"0000";
                                    DISPLAY_DATA_SSD <= X"E00E";
                  end case;
         end if;
 end process;

  -- Sequence time switching adjustment
 process (CLK_DIV(22-SSFACTOR)) begin
         if rising_edge(CLK_DIV(22-SSFACTOR)) then
                  if SLIDE_FILT(2 downto 0) = "001" then
                           if(PUSH_FILT(0) = '1') then
                                    secTime <= secTime + 1;
                           elsif(PUSH_FILT(1) = '1') then
                                    secTime <= secTime - 1;
                           end if;
                  end if;
         end if;
 end process;

  -- Hysteresis width adjustment
 process (CLK_DIV(21)) begin
         if rising_edge(CLK_DIV(21)) then
                  if SLIDE_FILT(2 downto 0) = "010" then
                           if(PUSH_FILT(0)='1') then
                                    dataInterval <= dataInterval + 1;
                           elsif(PUSH_FILT(1)='1') then
                                    dataInterval <= dataInterval - 1;
                           end if;
                  end if;
         end if;
 end process;

 -- Half period external adjustment
 process (CLK_DIV(22-SSFACTOR)) begin
         if rising_edge(CLK_DIV(22-SSFACTOR)) then
                  -- -------------------   speedUP adjustment section   -------------------
                  if PUSH_FILT(1 downto 0) = "00" then
                           HPadjSpeedUp <= 1;
         --if buttons are released reset HPadjSpeedUp
                  else

                           if HPadjSpeedUp = 2046 then HPadjSpeedUp <= HPadjSpeedUp;
                           else HPadjSpeedUp <= HPadjSpeedUp + 1; end if;             -- if you presse '0' or '1'
push button increase the modification speed
                  end if;
                  -- ------------------------   patternPeriod   ------------------------
                  if SLIDE_FILT(2 downto 0) = "011" then                      --we can make changes during display
HALFPERIOD on SSD
                           if(PUSH_FILT(0) = '1') then HALFPERIOD <= HALFPERIOD + HPadjSpeedUp;           --press
PushButton(0) to increase duty cycle
                           elsif(PUSH_FILT(1) = '1') then HALFPERIOD <= HALFPERIOD - HPadjSpeedUp;
         -- press PushButonn(1) to decrease
                           end if;
                  end if;
         end if;
 end process;

   -- dutyCycle and period width adjustment
 process (CLK_DIV(20)) begin
         if rising_edge(CLK_DIV(20)) then
```

```
                    -- -------------------   speedUP adjustment section   -------------------
                    if PUSH_FILT(1 downto 0) = "00" then
                              adjSpeedUp <= 1;
                    else

                              if adjSpeedUp = 20 then adjSpeedUp <= adjSpeedUp;
                              else adjSpeedUp <= adjSpeedUp + 1; end if;
                    end if;
                    -- --------------------------   dutyCycle   --------------------------
                    if SLIDE_FILT(2 downto 0) = "100" then
                              if(PUSH_FILT(0)='1') then
                                        dutyCycle <= dutyCycle + 10;
                              elsif(PUSH_FILT(1)='1') then
                                        dutyCycle <= dutyCycle - 10;
                              end if;
                    -- ------------------------   patternPeriod   ------------------------
                    elsif SLIDE_FILT(2 downto 0) = "101" then
                              if(PUSH_FILT(0)='1') then
                                        patternPeriod <= patternPeriod + 125;
                              elsif(PUSH_FILT(1)='1') then
                                        patternPeriod <= patternPeriod - 125;
                              end if;
                    end if;
          end if;
 end process;

-----------------------------------------------------
-- state machine to control behavior, depend on sequence.
-----------------------------------------------------
-- DIGS_int(4)='0' - sequence I (LO_MEMS: always GND, HI_MEMS: +15V/GND)
-- DIGS_int(4)='1' - sequence II (LO_MEMS: 3.3V/GND, HI_MEMS: -11.7V/GND)
--
-- sequence I:
--        CLK:      ________----------____________
--        H:                  _________---------------______
--        D:                  _________-----------__________
--        R:                  ____________-----_____________
--        HI_MEMS:---------___________---------- hi = 15V, low = GND
-- LOW_MEMS:_______________________________       low = GND

-- sequence II:
--        CLK:      ________----------____________
--        H:                  _________---------------______
--        D:                  _________-----------__________
--        R:                  ____________-----_____________
--        HI_MEMS:_________-----------__________ hi = GND, low = -11.7V
-- LOW_MEMS:---------___________----------       hi = 3.3V, low = GND

-- switching beetwen sec. follow always when D_DACEA is High.
-- there is a counter which switching beetwen sec. every adjustable secTime, independently of avarage samples value.
-- after switching there is a comparison and making a decizion in which mode should stay.

-- ####################################   Lower section (registers)    ####################################
-- ------------------------   STATE_REG   ------------------------
process (CLK_LF, RST) begin
          if RST = '1' then
                    state_reg <= seqI;
          elsif rising_edge(CLK_LF) then
                    if D_DACEA = '1' then
                              state_reg <= state_next;
                    end if;
          end if;
end process;
-- ------------------------   secCount   ------------------------
process (CLK_LF, RST) begin
          if RST = '1' then
                    secCount <= (others=>'0');
          elsif rising_edge(CLK_LF) then
                    -- ####################   MAIN COUNTER/TIMER   ####################
                    if SLIDE_FILT(3) = '0' then
                              -- if secCount(15 downto 12) = secTime then secCount <= secCount;              --
!!!!!!!!!!!!!!!   for ModelSim   !!!!!!!!!!!!!!!!
                              if secCount(26 downto 23) = secTime then
                              -- !!!!!!!!!!!!!!!!   for synthesis !!!!!!!!!!!!!!!!
                                        secCount <= secCount;
                                        if D_DACEA = '1' then
                                                  secCount <= (others=>'0');
                                                  -- if D_DACEA = 1 then clearing the sequence timer,
                                        end if;
                              -- elsif secCount = (secTime & B"000" & X"0_0000") - 1 then
                              --        if (SLIDE_FILT(4) = '0') then
                              --                  secCount <= secCount + 1;
                              --                  stopStoraging <= '0';
                              --        else
                              --                  stopStoraging <= '1';
                              --        end if;
                              else secCount <= secCount + 1;
                              end if;
                    else
                              secCount <= (others=>'0');
                    end if;
          end if;
end process;
-- ------------------------   avgCount   ------------------------
process (CLK_LF, RST) begin
          if RST = '1' then
                    avgCount <= (others=>'0');
          elsif rising_edge(CLK_LF) then
                    if SLIDE_FILT(3) = '0' then
                              if DIGS_int(4) = '0' then
                              -- average samples counter
                                        if avgCount(AVGS) = '1' then avgCount <= (others=>'1');             -- can't
do one direction counter because there is variable avg.samples
```

```vhdl
					else avgCount <= avgCount + 1; end if;
		-- counting up when DIGS(4) = 0
				elsif DIGS_int(4) = '1' then
					if avgCount(AVGS) = '0' then avgCount <= (others=>'0');
					else avgCount <= avgCount - 1; end if;
		-- counting down when DIGS(4) = 1
				end if;
			else
				avgCount <= (others=>'0');
			end if;
		end if;
end process;

-- ####################################   Upper section (next-state logic)   ####################################
process (state_reg, D_DACEA, avgCount(AVGS), secCount(26 downto 23), capSeqI, capSeqII, secTime, PUSH_FILT,
SLIDE_FILT, CLK_DIV(23 downto 17)) begin

		case state_reg is

			-------------------------------   PRE-SEQUENCE I   -------------------------------
			when pre_seqI =>
				DIGS_int(4) <= '0';
				LEDS(1) <= '0';

				if avgCount(AVGS) = '1' then
		-- when I have all samples to do the average,
					if (AVGS_OUT < capSeqII - dataInterval) then			-- compare
(including hysteresis)
						state_next <= seqI;
						else state_next <= seqII;
					end if;
				else state_next <= pre_seqI; end if;

			---------------------------------   SEQUENCE I   ---------------------------------
			when seqI =>
				DIGS_int(4) <= '0';
				LEDS(1) <= '1';

				if SLIDE_FILT(3) = '0' then
					if secCount(26 downto 23) = secTime	then
						state_next <= pre_seqII;
					else state_next <= seqI;
					end if;
				else state_next <= manual;
				end if;

			-------------------------------   PRE-SEQUENCE II   -------------------------------
			when pre_seqII =>
				DIGS_int(4) <= '1';
				LEDS(1) <= '0';

				if avgCount(AVGS) = '0' then
					if (AVGS_OUT < capSeqI - dataInterval) then
						state_next <= seqII;
					else state_next <= seqI;
					end if;
				else state_next <= pre_seqII; end if;

			---------------------------------   SEQUENCE II   ---------------------------------
			when seqII =>
				DIGS_int(4) <= '1';
				LEDS(1) <= '1';

				if SLIDE_FILT(3) = '0' then
					if secCount(26 downto 23) = secTime then
						state_next <= pre_seqI;
					else state_next <= seqII;
					end if;
				else state_next <= manual;
				end if;

			---------------------------------   MANUAL MODE   ---------------------------------
			when manual =>
				if CLK_DIV(20) = '1' then LEDS(1) <= '1';
				else LEDS(1) <= '0'; end if;

				if PUSH_FILT(2) = '1' then DIGS_int(4) <= '1';
				else DIGS_int(4) <= '0'; end if;

				if SLIDE_FILT(3) = '0' then state_next <= seqI;
				else state_next <= manual; end if;
		end case;
end process;

-- to storage a ADC data before switch
process (CLK_LF) begin
		if rising_edge(CLK_LF) then
			if (state_reg = seqI) or (state_reg = manual and DIGS_int(4) = '0')
				then capSeqI <= AVGS_OUT;
			elsif (state_reg = seqII) or (state_reg = manual and DIGS_int(4) = '1')
				then capSeqII <= AVGS_OUT; end if;
		end if;
end process;
--==================================================
-- assignments
--==================================================
-- SLIDE_SWITCH 3,4:		00 - out directly from chip
--										01 - out from differential aplifier
--										10 - external input 3
--										11 - external input 4
--DIGS_int(3 downto 2) <= SLIDE_FILT(4 downto 3);

CLK_ADC <=  '0' when SLIDE_FILT(5 downto 4) = "00" else
```

```
                              CLK_LF when SLIDE_FILT(5 downto 4) = "01" else
                              CLK_50MHz when SLIDE_FILT(5 downto 4) = "10" else
                              CLK_100MHz;
                                                                       -- when SLIDE_FILT(5 downto 4) = "11"

DATA_ADC_SIGNED <= ext(DATA_ADC,11); --It's extantion of 10-bits ADC data to 11-bits
CMPOUT<=COMP;
DIGS_int(3 downto 1) <= "000";
--LEDS(0)<=COMP;
--LEDS(1)<=(not TxDONE) and (not SLIDE_FILT(5));           -- LED on when is transmitting
--LEDS(2)<=(not RAM_WE(0)) and SLIDE_FILT(4) ;                       -- LED on when ready to begin transmission
--LEDS(3)<=nLOCK_DLY;

--LEDS(0) <= DIGS_int(1);
--LEDS(1) <= DIGS_int(2);
--LEDS(2) <= DIGS_int(3);
--LEDS(3) <= DIGS_int(4);
--LEDS(7 downto 4)<=(OTHERS =>'0');

LEDS(0) <= DIGS_int(4);
-- LEDS(1) <= -- also busy;
LEDS(5 downto 2) <= secCount(24 downto 21);
LEDS(6) <= secCount(25);
LEDS(7) <= avgCount(AVGS);

FILTER_OUT<=(others=>'0');              --FILT_OUT(12 downto 2);
CAPTURE<=PUSH_FILT(0);
CLK_DACEA<=CLK_DACEAI;
DATA_ADC_SIGNED<=ext(DATA_ADC,11);
nLOCK<=not LOCK;
DIGS <= DIGS_int;

-- necessities for the correct operation
MAN <= '0';
H0 <= '0';
D0_C <= '0';

end Behavioral;
```

dsp.vhd

```
----------------------------------------------------------------------------------
-- Company: Advanced Hardware Architectures (AHA) / Politechnical University of Catalonia (UPC)
-- Engineer: Daniel Fern ez (dfernan@eel.upc.edu), Kamil Karliński
--
-- Create Date:    11:30:44 05/29/2008
-- Design Name:
-- Module Name:    delayed_stim - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--          0.1 - Code cleanup
-- Additional Comments:
--
----------------------------------------------------------------------------------


----------------------------------------------------------------------------------
-- Expansor of the signal duration
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity delayed_stim is
    generic (predelay : integer range 1 to 1023 := 10;
              postdelay : integer range 1 to 1023 := 10;
                 preventive : bit := '0');
    Port ( clk : in  STD_LOGIC;
           inp : in  STD_LOGIC;
           outp : out  STD_LOGIC);
end delayed_stim;

architecture Behavioral of delayed_stim is
 type state is (inputhi, justrised, justfallen, rest);
 signal pr_state, nx_state : state := rest;
 signal cnt1, cnt2 : integer range 0 to 1023 := 0;

begin
 process (clk)
  begin
     if (clk='1' and clk'event) then
     pr_state<=nx_state;
      if(pr_state = justrised) then cnt2<=cnt2-1; else cnt2<=predelay-1; end if;
      if(pr_state = justfallen) then cnt1<=cnt1-1; else cnt1<= postdelay-1; end if;
     end if;
 end process;

 process(pr_state, cnt1, cnt2, inp)
  begin
```

```
   case pr_state is
     when justrised =>
     outp<='0';
      case preventive is
       when '1' =>
         if(inp='0') then nx_state <= justfallen;
          elsif(cnt2 = 1) then nx_state <= inputhi ; else nx_state <= justrised; end if;
       when others =>
       if(cnt2 = 1) then nx_state <= inputhi ; else nx_state <= justrised; end if;
      end case;
    when inputhi =>
     outp<='1';
     if(inp='0') then nx_state <= justfallen; else nx_state <= inputhi; end if;
    when justfallen =>
     outp<='1';
     if(inp='1') then nx_state <= inputhi;
       elsif(cnt1 = 1) then nx_state <= rest;
        else nx_state <= justfallen;
      end if;
    when others =>
     outp<='0';
     if(inp='1') then nx_state <= justrised; else nx_state <= rest; end if;
   end case;
 end process;
end Behavioral;


----------------------------------------------------------------------------------
-- Simple derivator
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity derivator is
    generic (n: INTEGER := 11);
    Port ( clk : in  STD_LOGIC;
            din : in  STD_LOGIC_VECTOR(n-1 downto 0);
           dout : out  STD_LOGIC_VECTOR(n downto 0));
end derivator;

architecture Behavioral of derivator is
 signal din_old : STD_LOGIC_VECTOR(n-1 downto 0);

begin
 process (clk)
  begin
      if (clk='1' and clk'event) then
       dout<=(sxt(din,n+1))-(sxt(din_old,n+1));
        din_old<=din;
      end if;
 end process;
end Behavioral;


----------------------------------------------------------------------------------
-- Generic subsampler v2
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity subsampler_v2 is
    generic (n : integer := 11;
              s : integer := 2);
    Port ( clk : in  STD_LOGIC;
              clk_out : in STD_LOGIC;
            din : in  STD_LOGIC_VECTOR(n-1 downto 0);
           dout : out  STD_LOGIC_VECTOR(n-1 downto 0));
end subsampler_v2;

architecture Behavioral of subsampler_v2 is
 type samples IS ARRAY ((2**s)-1 DOWNTO 0) OF STD_LOGIC_VECTOR(n-1 downto 0);
 signal oldsamples, oldsamples_aux : samples;

begin
 process (clk)
  variable count : NATURAL RANGE 0 TO (2**s)-1 := 0;
  begin
    if rising_edge(clk) then
     oldsamples(count)<=din;
     if count=0 then oldsamples_aux<=oldsamples; end if;
      if(count=(2**s)-1) then count:=0; else count:=count+1; end if;
     end if;

 end process;

process (clk_out)
 variable accumulator : STD_LOGIC_VECTOR(n+s-1 downto 0) := (OTHERS => '0');
  begin
    if rising_edge(clk_out) then
     accumulator:=(OTHERS => '0');
     for i in (2**s)-1 downto 0 loop
       accumulator:=accumulator+sxt(oldsamples_aux(i),n+s);
     end loop;
     dout<=accumulator(n+s-1 downto s);
     end if;
 end process;
end Behavioral;
```

```
--------------------------------------------------------------------------------
-- Sample and Hold
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sampleandhold is
    generic (n : integer);
    Port ( clk, hold : in  STD_LOGIC;
            din : in  STD_LOGIC_VECTOR(n-1 downto 0);
           dout : out  STD_LOGIC_VECTOR(n-1 downto 0));
end sampleandhold;

architecture Behavioral of sampleandhold is
begin
 process (clk)
  begin
   if rising_edge(clk) then
        if hold='0' then dout<=din; end if;
   end if;
 end process;
end Behavioral;


--------------------------------------------------------------------------------
-- Simple sequence generator
--------------------------------------------------------------------------------
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.std_logic_unsigned.all;
   entity cyclicpattern is
    generic (n : integer);
    port (clk, reset: in std_logic;
          dada : std_logic_vector(n-1 downto 0);
             b:out std_logic);
    end cyclicpattern;
   architecture structural of cyclicpattern is
    begin
     process(clk)
      variable i: integer range 0 to dada'length-1 := 0;
      begin
       if rising_edge(clk) then
        if reset='1' then i := dada'length-1;
        else
         if i=0 then i:=dada'length-1; else i := i-1; end if;
        end if;
        b <= dada(i);
       end if;
      end process;
     end structural;

--------------------------------------------------------------------------------
-- Averaging module
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity averaging is
    generic ( avgs : integer;
             n : integer);
    port (
        clk : in std_logic;
        din : in  STD_LOGIC_VECTOR(n-1 downto 0);
        dout : out  STD_LOGIC_VECTOR(n-1 downto 0)
    );
end averaging;

architecture Behavioral of averaging is

 type samples IS ARRAY ((2**avgs)-1 DOWNTO 0) OF STD_LOGIC_VECTOR(n-1 downto 0);
 signal dataSamples : samples;

begin
 process (clk)
  variable accumulator : STD_LOGIC_VECTOR(n+avgs-1 downto 0) := (OTHERS => '0');
  begin
  if rising_edge(clk) then
    -- add new sample and detele the last one (shift right)
    -- first LSB is deleted, then array is shifted right
    -- and the new sample is put in MSB place
    for index in 0 to (2**avgs)-2 loop
        dataSamples(index) <= dataSamples(index+1);
    end loop;
    dataSamples((2**avgs)-1)<=din;

    -- average all data in array
    accumulator := (others=>'0');
    for index in dataSamples'range loop
        accumulator := accumulator + ext(dataSamples(index),n+avgs);
    end loop;
    dout <= accumulator(n+avgs-1 downto avgs);
  end if;
 end process;
end Behavioral;
```

misc.vhd

```
--------------------------------------------------------------------------------
-- Company: Advanced Hardware Architectures (AHA) / Politechnical University of Catalonia (UPC)
-- Engineer: Daniel Fernández (dfernan@eel.upc.edu)
--
-- Create Date:    11:30:44 05/29/2008
-- Design Name:
-- Module Name:    output pattern generators  - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--          0.1  - RAM_STOPCONT added
-- Additional Comments:
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity OUT_PATTERN is
    generic (maxlock : integer;
                                        grace : integer);
    Port ( clk : in  STD_LOGIC;
           comp : in  STD_LOGIC;
                  halfperiod : NATURAL RANGE 1 to 8191;
           lock : out  STD_LOGIC;
           out_pattern : out  STD_LOGIC);
end OUT_PATTERN;

architecture Behavioral of OUT_PATTERN is
 type STATE is (dist_decreasing, dist_increasing, not_locked);
 signal PAT_PRSTATE, PAT_NXSTATE : STATE := not_locked;
 signal countlock, countdist : NATURAL RANGE 0 to 131000;

begin
 process (clk)
  begin
           if (rising_edge(clk)) then
            if(PAT_PRSTATE = dist_decreasing) then countlock<=countlock-1; else countlock<=maxlock; end if;
            if(PAT_PRSTATE = dist_increasing) then countdist<=countdist-1; else countdist<=halfperiod; end if;
     PAT_PRSTATE<=PAT_NXSTATE;
           end if;
 end process;

 process(PAT_PRSTATE, comp, countlock, countdist)
  begin
   case PAT_PRSTATE is
    when dist_decreasing =>
     out_pattern<='0';
            lock<='1';
     if countlock<(maxlock-grace) then
             if(comp='1') then PAT_NXSTATE<=dist_increasing;
              elsif countlock=1 then PAT_NXSTATE<=not_locked;
                     else PAT_NXSTATE<=dist_decreasing; end if;
      else PAT_NXSTATE<=dist_decreasing;
            end if;
    when dist_increasing =>
     out_pattern<='1';
     lock<='1';
     if(countdist=1) then PAT_NXSTATE<=dist_decreasing; else PAT_NXSTATE<=dist_increasing; end if;
    when others =>
     out_pattern<='1';
     lock<='0';
     PAT_NXSTATE<=dist_increasing;
   end case;
 end process;
end Behavioral;


-- ---------------------------   MY OUT PATTERN GENERATOR   ---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity MY_PATTERN is
    Port (
                   clk : in  std_logic;                                      -- 12,5 MHz clock what means 80ns
period
                   dutyCycle : in NATURAL range 1 to 1023;
                   patternPeriod : in natural range 1 to 1023;
                   out_pattern : out std_logic
          );
end MY_PATTERN;

architecture Behavioral of MY_PATTERN is
 signal Counter : natural range 0 to 1023 := 0;
begin
 process (clk) begin
          if rising_edge(clk) then
                   if (Counter < patternPeriod) then
                            Counter <= Counter + 1;
```

```
                    else Counter <= 0; end if;
          end if;
 end process;
 out_pattern <= '1' when Counter < dutyCycle else '0';
end Behavioral;
```

Display.vhd

```
----------------------------------------------------------------------------------
-- Company: Advanced Hardware Architectures (AHA) / Politechnical University of Catalonia (UPC)
-- Engineer: Daniel Fernández (dfernan@eel.upc.edu) / Jose Luis Casas
--
-- Create Date:    10:57:12 05/27/2008
-- Design Name:
-- Module Name:    Display - Behavioral
-- Project Name:
-- Target Devices: DACEA platform
-- Tool versions:
-- Description: BCD display basic handlers
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Display is
    Port ( value : in  STD_LOGIC_VECTOR (15 downto 0);
           dots : in  STD_LOGIC_VECTOR (3 downto 0);
                                en : in STD_LOGIC_VECTOR (3 downto 0);
           clkin : in  STD_LOGIC;
           bcd_out : out  STD_LOGIC_VECTOR (7 downto 0);
           bcd_sel : out  STD_LOGIC_VECTOR (3 downto 0));
end Display;

architecture Behavioral of Display is

 component BCDto7seg is
  port (a : in std_logic_vector(4 downto 0);
  b:out std_logic_vector(7 downto 0));
 end component;
 component muxBCD is
   port (a3,a2,a1,a0:in std_logic_vector(4 downto 0);
   s:in std_logic_vector(1 downto 0);
   b:out std_logic_vector(4 downto 0));
 end component;
 component decBCD is
   port (a : in std_logic_vector(1 downto 0);
   en : in std_logic_vector(3 downto 0);
   b : out std_logic_vector(3 downto 0));
end component;

 signal bcd_in : std_logic_vector (4 downto 0);
 signal bcd_in3, bcd_in2, bcd_in1, bcd_in0 : std_logic_vector (4 downto 0);
 signal sel : std_logic_vector (1 downto 0) := (OTHERS=>'0');
begin
 BCD : component BCDto7seg port map (bcd_in, bcd_out);
 MUX : component muxBCD port map (bcd_in3, bcd_in2, bcd_in1, bcd_in0, sel, bcd_in);
 DEC : component decBCD port map (sel, en, bcd_sel);

 bcd_in3<=(dots (3) & value(15 downto 12));
 bcd_in2<=(dots (2) & value(11 downto 8));
 bcd_in1<=(dots (1) & value(7 downto 4));
 bcd_in0<=(dots (0) & value(3 downto 0));

 process (clkin)
 begin
    if (clkin='1' and clkin'event) then
        sel<=sel+1;
    end if;
 end process;


end Behavioral;


----------------------------------------------------------------------------------
-- Create Date:    10/11/2007
-- Module Name:    decBCD
-- Project Name:   display
-- Tool versions:
-- Author:         Daniel Fernández (dfernan@eel.upc.edu)
-- Description:    Decodificador 2:4.
--
----------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity decBCD is
```

```
   port (a : in std_logic_vector(1 downto 0);
   en : in std_logic_vector(3 downto 0);
   b : out std_logic_vector(3 downto 0));
end decBCD;

architecture arhmux of decBCD is
begin
   process(en, a)
   begin
           if (a="00" and en(0)='1') then b<="1110";
     elsif (a="01" and en(1)='1') then b<="1101";
     elsif (a="10" and en(2)='1') then b<="1011";
     elsif (a="11" and en(3)='1') then b<="0111";
     else b<="1111";
          end if;
   end process;
end arhmux;


--------------------------------------------------------------------------------
-- Create Date:    10/11/2007
-- Module Name:
-- Project Name:   display
-- Tool versions:
-- Author:         Jose Luis Casas / Daniel Fernández (dfernan@eel.upc.edu)
-- Description:    Decodificador BCD a 7 segmentos
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity BCDto7seg is
   port (a:in std_logic_vector(4 downto 0);
   b:out std_logic_vector(7 downto 0));
end BCDto7seg;

architecture Arch_BCDto7seg of BCDto7seg is
begin
   process(a)
   begin
      if    (a(3 downto 0)="0000") then b<=NOT a(4) & "1000000";--displays 0
      elsif (a(3 downto 0)="0001") then b<=NOT a(4) & "1111001";--displays 1
      elsif (a(3 downto 0)="0010") then b<=NOT a(4) & "0100100";--displays 2
      elsif (a(3 downto 0)="0011") then b<=NOT a(4) & "0110000";--displays 3
      elsif (a(3 downto 0)="0100") then b<=NOT a(4) & "0011001";--displays 4
      elsif (a(3 downto 0)="0101") then b<=NOT a(4) & "0010010";--displays 5
      elsif (a(3 downto 0)="0110") then b<=NOT a(4) & "0000010";--displays 6
      elsif (a(3 downto 0)="0111") then b<=NOT a(4) & "1011000";--displays 7
      elsif (a(3 downto 0)="1000") then b<=NOT a(4) & "0000000";--displays 8
      elsif (a(3 downto 0)="1001") then b<=NOT a(4) & "0010000";--displays 9
      elsif (a(3 downto 0)="1010") then b<=NOT a(4) & "0001000";--displays A
      elsif (a(3 downto 0)="1011") then b<=NOT a(4) & "0000011";--displays B
      elsif (a(3 downto 0)="1100") then b<=NOT a(4) & "1000110";--displays C
      elsif (a(3 downto 0)="1101") then b<=NOT a(4) & "0100001";--displays D
      elsif (a(3 downto 0)="1110") then b<=NOT a(4) & "0000110";--displays E
      elsif (a(3 downto 0)="1111") then b<=NOT a(4) & "0001110";--displays F
--                 elsif ()
                   end if;
   end process;
end Arch_BCDto7seg;


--------------------------------------------------------------------------------
-- Create Date:    10/11/2007
-- Module Name:    mux - arhmux
-- Project Name:   display
-- Tool versions:
-- Author:         Jose Luis Casas / Daniel Fernández (dfernan@eel.upc.edu)
-- Description:    Este modulo es un multiplexor de 4:1. Tiene 2 bits de seleccion
--                 de canal y 4 buses de 5 bits.
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity muxBCD is
   port (a3,a2,a1,a0:in std_logic_vector(4 downto 0);
   s:in std_logic_vector(1 downto 0);
   b:out std_logic_vector(4 downto 0));
end muxBCD;

architecture arhmux of muxBCD is
begin
   process(a3,a2,a1,a0,s)
   begin
      if (s="00") then b<=a0;
      elsif (s="01") then b<=a1;
      elsif (s="10") then b<=a2;
      elsif (s="11") then b<=a3;
      end if;
   end process;
end arhmux;
```

TB_MEMS_ChargeInReduce.vhd

```vhdl
--------------------------------------------------------------------------------
-- Company: DMCS
-- Engineer: Kamil Karlinski
--
-- Create Date:    22/04/2010
-- Design Name:
-- Module Name:    TB_MEMS_ChargeInReduce - testbench
-- Description:    the same as filename
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_MEMS_ChargeInReduce is
end entity;

architecture testbench of TB_MEMS_ChargeInReduce is

-- clocks section
signal clk_in, ckl_100mhz, clk_lf, clk_dacea : STD_LOGIC := '0';
signal clk_div : STD_LOGIC_VECTOR(23 downto 0);
signal clk_ADC_data, clk_charge : STD_LOGIC := '0';

signal or_adc : STD_LOGIC := '0';
signal h_duty : STD_LOGIC;
signal selector : STD_LOGIC_VECTOR (1 downto 0);
signal rst,h,d,r : STD_LOGIC := '0';
signal slide, leds : STD_LOGIC_VECTOR (7 downto 0) := (others=>'0');
signal push : STD_LOGIC_VECTOR (2 downto 0) := (others=>'0');
signal digs : STD_LOGIC_VECTOR (4 downto 1) := (others=>'0');
signal secCount : STD_LOGIC_VECTOR (26 DOWNTO 0) := (others=>'0');
signal clearSecCounter : STD_LOGIC;
signal upSecCountEnter : STD_LOGIC_VECTOR (27 DOWNTO 0) := (others=>'1');

signal data_adc : STD_LOGIC_VECTOR (9 downto 0) := B"10_0000_0000";           -- = 512 in decimal
signal capSeqI, capSeqII : STD_LOGIC_VECTOR (10 downto 0);
signal charge : NATURAL range 0 to 512 := 0;
signal Charging : STD_LOGIC := '1';
signal AVGS_OUT : STD_LOGIC_VECTOR (10 downto 0) := (others=>'0');
signal secTime : STD_LOGIC_VECTOR (3 DOWNTO 0) := B"0001";
signal avgCount : STD_LOGIC_VECTOR (5 DOWNTO 0) := B"00_0000";

signal A,B : STD_LOGIC_VECTOR (10 downto 0);
signal A_higher_B, A_equal_B, A_lesser_B : STD_LOGIC;
constant hysteresis : NATURAL range 0 to 15 := 5;
signal intCLK_2ms : STD_LOGIC := '0';
signal ADC_increase : STD_LOGIC := '0';

-- signal HI_mems, LO_mems : STD_LOGIC := '0';
signal HI_mems, LO_mems : unsigned (1 downto 0) := "00";


begin

          SIM_MEMS_unit: entity work.SIM_MEMS
          port map (
                    CLK_IN => clk_in,
                    CLK_ADC => open,
                    DATA_ADC => data_adc,
                    CLK_DACEA => clk_dacea,
                    BCD_OUT => open,
                    BCD_SEL => open,
                    FILTER_OUT => open,
                    CAPTURE => open,
                    RST => rst,
                    H_DACEA => h,
                    D_DACEA => d,
                    R_DACEA => r,
                    CMPOUT => open,
                    SLIDE_SWITCH => slide,
                    PUSH_SWITCH => push,
--                              TxD => ,
                    LEDS => leds,
--                              VGA_HS, VGA_VS, VGA_RED, VGA_GRN, VGA_BLU : out STD_LOGIC;
                    OR_ADC => or_adc,
                    MAN => open,
                    H0 => open,
                    D0_C => open,
                    DIGS => digs,
                    CLK_100MHz_sim => ckl_100mhz,
                    CLK_LF_sim => clk_lf,
                    H_DUTY_out => h_duty,
                    SELECTOR_out => selector,
                    CLK_DIV_out => clk_div,
                    AVGS_OUT_out => AVGS_OUT,
                    secCount_out => secCount,
                    secTime_out => secTime,
                    avgCount_out => avgCount,
                    upSecCountEnter_out => upSecCountEnter,
                    capSeqI_out => capSeqI,
                    capSeqII_out => capSeqII
```

```
                    );

-- clocks definition section
clk_in <= not clk_in after 10 ns;                                     -- frequency = 50MHz -> period = 20ns
ckl_100mhz <= not ckl_100mhz after 5 ns;                    -- frequency = 100MHz -> period = 10ns
clk_lf <= not clk_lf after 40 ns;                                     -- frequency = 1/4 * 50Mhz = 12,5MHz -> period =
80ns
intCLK_2ms <= not intCLK_2ms after 3200 us;                 -- frequency = ???                              -> period
4 ms
clk_ADC_data <= not clk_ADC_data after 5 us;
clk_charge <= not clk_charge after 5 us;

slide(7 downto 6) <= "11";
slide(5 downto 4) <= "11";
slide(3) <= '0';

-- manual mode stimulation
-- slide(2) <= '0', '1' after 1.8 ms, '0' after 4.7 ms;
-- push(2) <= '0', '1' after 2.5 ms, '0' after 3 ms, '1' after 3.5 ms, '0' after 4 ms;

-- for HI_mems and LO_mens I do a simulation with the following level:
-- logic "01"       ->                  GND
-- logic "11"       ->                  +15V
-- logic "10"       ->                  +3.3V
-- logic "00"       ->                  -11.7V
HI_mems <=          "01" when d='1' else
                              "11" when d='0' and digs(4)='0' else              --sec. I
                              "00" when d='0' and digs(4)='1';                            --sec. II

LO_mems <=          "01" when d='1' or (d='0' and digs(4)='0') else
                              "10" when d='0' and digs(4)='1';


chipSignalGenerator : process begin
          wait until clk_dacea='1'; wait for 10 ns;
          h <= '1'; d <= '1'; wait for 30 ns;
          r <= '1';
          wait until clk_dacea='0'; wait for 10 ns;
          r <= '0'; wait for 30 ns;
          h <= '0'; d <= '0';
end process chipSignalGenerator;

-- --------------------   ADC data   --------------------
ADC_data_Stimulator : process (clk_ADC_data, digs(4)) begin
          if digs(4)'event then
                    if Charging = '1' then
                              data_adc <= data_adc - 2*charge;
                    else
                              data_adc <= data_adc + 2*charge;
                    end if;
          elsif rising_edge(clk_ADC_data) then
                    data_adc <= data_adc  + 1;
          end if;
end process;

-- --------------------   CHARGE   --------------------
Charge_Stimulator : process(clk_charge) begin
          -- if rising_edge(intCLK_2ms) then
                    -- charge <= 127;
          -- els
          if rising_edge(clk_charge) then
                    if Charging = '1' then
                              charge <= charge + 1;
                    else
                              charge <= charge - 1;
                    end if;
          end if;
end process;

-- --------------------   chargingFlag   --------------------
charging_flag_Stimulator : process(digs(4), charge) begin
          if charge = 0 then
                    Charging <= '1';
          elsif digs(4)'event then
                    Charging <= not Charging;                         -- nie może byc = 0 że po zmianie robimy zawsze
rozłodowywanie, bo jak na chwilę
          else
                    -- wejdziemy w drugi tryb - żeby tylko zobaczyć to po powrocie musi być ładowanie
                    Charging <= Charging;
          end if;
end process;

-- adcDataSimulator : process(d, intCLK_2ms) begin
          -- if d'event then
                    -- if ADC_increase = '1' then data_adc <= data_adc + 5;
                    -- else data_adc <= data_adc - 5; end if;

          -- elsif intCLK_2ms'event then
                    -- if ADC_increase = '1' then data_adc <= data_adc + 20;
                    -- else data_adc <= data_adc - 20; end if;
          -- end if;
-- end process adcDataSimulator;
-- ADC_increase <= not ADC_increase after 2.2 ms;
-- data_adc <= B"01_0010_1100"; -- 300 in decimal

-- --------------------------------------   part to check how comparison works   ------------------

-- A <= B"000"&X"00", B"000"&X"03" after 0.3 us, B"000"&X"0F" after 0.65 us, B"000"&X"A0" after 0.9 us,
                    -- B"000"&X"01" after 1 us, B"000"&X"03" after 1.2 us, B"000"&X"0D" after 1.5 us;

-- B <= B"000"&X"00", B"000"&X"09" after 0.35 us, B"000"&X"01" after 0.6 us, B"000"&X"A0" after 0.95 us,
                    -- B"000"&X"06" after 1.05 us, B"000"&X"0A" after 1.25 us, B"000"&X"08" after 1.56 us;
```

```
-- -- histereza daje nam to, że przełączanie nie następuja tak często.

-- temporaryCheckingProcess : process(clk_lf) begin
        -- if rising_edge(clk_lf) then
                -- if (A > B + hysteresis) then A_higher_B <= '1'; A_lesser_B <= '0'; A_equal_B <= '0';
                -- elsif (A < B - hysteresis) then A_higher_B <= '0'; A_lesser_B <= '1'; A_equal_B <= '0';
                -- else A_higher_B <= '0'; A_lesser_B <= '0'; A_equal_B <= '1';
                -- end if;
        -- end if;
-- end process;

end architecture;
```

sim.do

```
quit -sim
vdel -all -lib work

vlib work

vcom Display.vhd
vcom dsp.vhd
vcom misc.vhd
vcom SIM_MEMS.vhd
vcom TB_MEMS_ChargeInReduce.vhd

#simulate without optimization
vsim -novopt work.TB_MEMS_ChargeInReduce

onerror {resume}
quietly WaveActivateNextPane {} 0

#open the window to time analize a wave
view wave

# add all items in region to wave
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/clk_in
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/ckl_100mhz
        add wave -noupdate -format Logic /tb_mems_chargeinreduce/clk_lf
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/clk_dacea
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/or_adc
        add wave -noupdate -format Logic /tb_mems_chargeinreduce/h_duty
# add wave -noupdate -format Logic -radix decimal /tb_mems_chargeinreduce/clk_div
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/rst
add wave -noupdate -format Logic /tb_mems_chargeinreduce/h
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/d
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/r
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/slide
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/push
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/selector
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/leds
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/ADC_increase
        add wave -noupdate -format Analog -min 400 -max 700 -radix unsigned -height 80 /tb_mems_chargeinreduce/data_adc
        add wave -noupdate -format Analog -min 0 -max 150 -radix unsigned -height 80 /tb_mems_chargeinreduce/charge
# add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/AVGS_OUT
        add wave -noupdate -format Analog -min 0 -max 3 -radix unsigned -height 40 /tb_mems_chargeinreduce/hi_mems
        add wave -noupdate -format Analog -min 0 -max 3 -radix unsigned -height 40 /tb_mems_chargeinreduce/lo_mems
# add wave -noupdate -format Logic /tb_mems_chargeinreduce/digs
        add wave -noupdate -format Logic /tb_mems_chargeinreduce/digs(4)
        add wave -noupdate -format Logic /tb_mems_chargeinreduce/leds(1)
        add wave -noupdate -format Logic /tb_mems_chargeinreduce/Charging
# add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/capSeqI
# add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/capSeqII
# add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/secCount
# add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/clearSecCounter
# add wave -noupdate -format Logic -radix decimal /tb_mems_chargeinreduce/secTime
        add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/avgCount
```

```
# add wave -noupdate -format Logic -radix unsigned /tb_mems_chargeinreduce/upSecCountEnter

#add wave -noupdate -format Logic -radix hexadecimal /tb_mems_chargeinreduce/A
#add wave -noupdate -format Logic -radix hexadecimal /tb_mems_chargeinreduce/B
#add wave -noupdate -format Logic /tb_mems_chargeinreduce/A_higher_B
#add wave -noupdate -format Logic /tb_mems_chargeinreduce/A_lesser_B
#add wave -noupdate -format Logic /tb_mems_chargeinreduce/A_equal_B
TreeUpdate [SetDefaultTree]

configure wave -namecolwidth 120
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits us
update

#run simulation for 1ms
# for simulate in normal mode with long D_DACEA pattern set 3ms sim. time
#for low/high duty cycle in CLK_DACEA signal (test mode)
#run 1ms

#for long CLK_DACEA pattern (normal mode)
run 10ms
```