Universidad Carlos III de Madrid
www.uc3m.es

FINAL DEGREE THESIS

DIGITAL INCLINOMETER

GRADO EN INGENIERÍA EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

AUTOR: Marta Arce de la Torre
TUTOR: Michael García Lorenz

Madrid, 22 de Junio de 2014

Abstract

In this document is present a final degree thesis which main goal is to design, development and construction of a digital inclinometer. The project consists on a multitasking system organised by a microcontroller. The input data is given by an accelerometer connected via SPI. The microprocessor will analyse this data and will obtain the angle. After it will show this angle in a lcd screen, also connected to microcontroller. It can be also set a tare angle with a pushbutton. This is useful to measure angles between different surfaces. This system will be presented soldered and in a plastic case, obtained with a 3D printer.

Resumen

En este documento se presenta un trabajo fin de grado cuyo objetivo principal es el diseño, desarrollo y construcción de un inclinómetro digital. El proyecto consiste en un sistema multitarea organizado por un microcontrolador. Los datos serán obtenidos por un acelerómetro conectado a través de SPI. El microprocesador los analizará obtendrá el ángulo. Después lo mostrará en una pantalla lcd, también conectada al microcontrolador. Se puede también definir un ángulo de tara con un pulsador. Esto es útil para medir ángulos entre diferentes superficies. Este sistema será presentado soldado y en una caja de plástico, obtenida con una impresora 3D.

Agradecimientos

Gracias a mis padres y a toda mi familia y amigos por todo el apoyo y los ánimos que me han dado a lo largo de todos mis estudios. Gracias a todos los compañeros con lo que he coincido a lo largo de la carrera y que han aportado algo a mis estudios.

Quiero agradecer especialmente a Rodrigo G, Flor A y Javier RC el apoyo que me han dado en este proyecto.

¡Muchas gracias a todos!

Index

Listado de Acrónimos

SPI	Serial Protocol Input
I2C	Inter-Integrated Circuit
IDE	Integrated Development Enviroment
LSB	Least significant bit
MSB	Most significant bit
CS	Chip Select
SCK	Serial Clock
SDI	Serial Data Input
SDO	Serial Data Output
MOSI	Master Out Slave In
MISO	Master In Slave Out

1 Introduction

An inclinometer is a device for measuring the angle of inclination of a surface with respect to its gravity. A digital inclinometer is an instrument form by a sensor which detects accelerations and microprocessor to calculate the angle and communicate it to another device or to the user through a output peripheral, for example, an screen.

In this document is presented the design of a digital inclinometer, from the hardware selection to the firmware development and the results obtained. This section introduces the motivation of the project, the main objectives and the document organization.

1.1 Motivation

A great number of applications require an instrument to measure angles or tilt. Inclinometers are essential in construction, aviation, boating, transportation, industrial, robotic and automation industries.

Furthermore, in the past years, position detection instruments had increased hugely its market due to its incorporation into consumer electronics. Accelerometers, among other complementary sensors, had become very important in videogame consoles, smartphones and wearable technology for fitness or health care.

Because of its requirement in a large application fields and in its great proliferation among all king of electronic devices, it was decided to design an inclinometer. Knowing the angles of a device with respect to gravitational force allows improving or increment its functionalities. For example, smartphones improve the user experience by flipping its screen when its orientation changes.

1.2 Objectives

The main objective of this project is to design and build a digital inclinometer with an accelerometer, a microcontroller and a screen. In order to achieve the main goal it has been specify these objectives:

- Communicate accelerometer data to the microcontroller through SPI or I2C.
- Show in the screen the angle between the accelerometer and the reference surface.
- Include a pushbutton to set the tare angle.
- Develop a multitasking system that allows parallel process of the previous tasks.
- Solder the circuit into a board and power it with a battery.
- Design a plastic case for the circuit and print it in a 3D printer.

2 State of the art

This section introduces the different types of inclinometers of the market and it current uses for angle detection and objects position detection.

2.1 Types of inclinometers

The inclinometer, as defined above, is an instrument which measure angles and/or tilt. It can be analog or digital. Analog inclinometers are based on the movement of a mass or a bubble through a previously graduated path. Digital inclinometers sample the effect of the gravitational force on the sensor.

The inclinometer, also known as clinometers, has been important along the history. Egyptians had a basic analog inclinometer shown in the illustration 1. It consists in three bars placed together forming a right triangle. In the right angle corner it is tied up a mass, which will move according to the inclination of this triangle. The triangle side opposite to the right angle it is graduated, so the angle is obtain by looking to the position of the mass over the graduated bar.

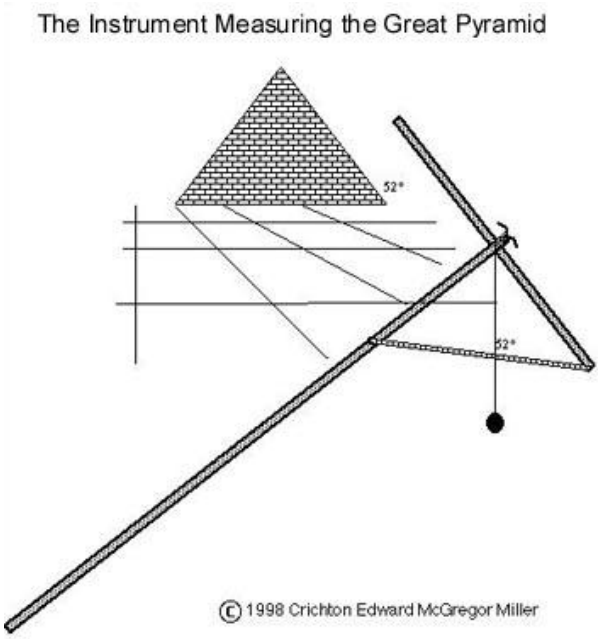


Illustration 1. Egyptian inclinometer **¡Error! No se encuentra el origen de la referencia.**

- Pendulum inclinometer: Similar to the previous one but improved by putting a circumference arc shape measured bar instead of the straight bar.

- Well's inclinometer: It consists on a graduated disk which contains another disk half-full of a heavy liquid. With no inclination, the liquid level will be at the horizontal line of cero degrees of the graduated disk. With inclination, the liquid will mark the inclination angle.

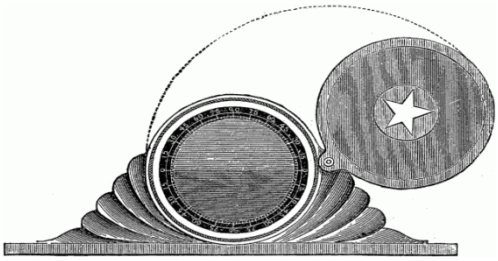


Illustration 2. Well's inclinometer

- Bubble gas inclinometer: This one holds a glass tube with liquid and an air bubble inside. When the position of the instrument varies, the bubble stays level. Its position shows the incline angle on a scale.

Ilustration 3. Spirit of Saint Louis bubble inclinometer

-With an accelerometer sensor. There type are explained broadly in the next section.

1.1 Accelerometer

An accelerometer is a device that measures proper acceleration. The acceleration can be measured by different ways:

- Capacitive: The accelerometer senses the capacitance change between a static condition and the dynami state.

- Piezoelectric: When stress is applied on a piezoelectric material an electrical charge is created.

- Piezoresistive: This type works measuring the electrical resistence of a material when stress is applied.

- Hall Effect: The accelerometer detects changes in the magnetic field around it.

- Heat transfer: Acceleration causes heat in the accelerometer. With thermoresistors the variationss of heat can be detected.

2.2 Current usages

Accelerometers have many applications in many different sectors of the industry. They can be used to measure vehicle acceleration that allows evaluation of vehicle performance, impact loads, earthquakes...

Nowadays the use of these devices in consumer electronic is very popular. Mobiles phones, tablets, computers... use this type of sensor to show an orientated image on the display in dependence of the position of the device.

Nintendo introduced this sensor in their console offering a different gaming experience. This accelerometer is a three axis one been able to registers de movements of the user's remote control in the space.

Some electronic of consume brands have developed a way to quantified the exercise their users do with accelerometers. They have included an accelerometer in small portable devices and with the accelerometer it counts the number of steps, how many times the user have moved while he was sleeping...

Illustration 42. Sony smart band

3 System propose

The system proposed for the implementation of a digital inclinometer has three main elements: an accelerometer sensor, a microcontroller and a screen.

The sensor will provide the input data, which is the gravitational acceleration over the horizontal axis. The microcontroller will calculate the angles and organize the different threats. The screen will show the angles.

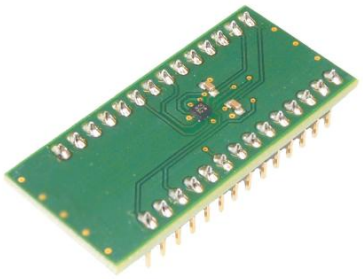
In addition to the main elements, the system will have a push button to select the tare angle and an interrupter between the battery and the system. The power system will consist in a battery and voltage regulator.

In this section it is going to be explained the election of the main elements and its advantages among others. It is also explain how are connected all the elements.

3.1 Acelerometer BOSH BCM050

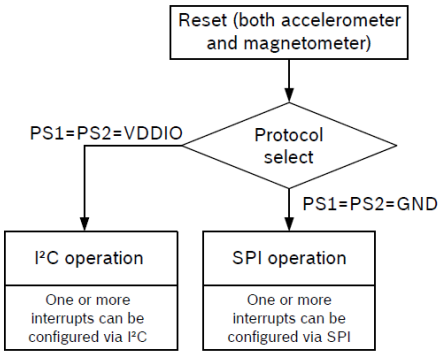
The BOSH BCM050 accelerometer was selected because it meets the requirements for this project and it was available. It was not necessary to purchase it because it had been in previous applications.

The accelerometer chip is very small but it has been use one in a PCB prepared to be connected into a board. This is a great advantage, because make it easy to connect it with the microcontroller trough jumpers wires.



One of the requirements asked is that it has to be able to communicate with the microcontroller via SPI or I2C protocols. In the case of this accelerometer have both

possibilities. The protocol is selected by hardware, powering or connecting to group the protocols selections pins, as we can see in illustration.



Another important fact of the accelerometer is the resolution. It has to sufficient to measure each angle. The available acceleration range and resolution are in the table below.

Range	Acceleration measurement range	Resolution
0011	±2g	3.91mg/LSB
0101	±4g	7.81mg/LSB
1000	±8g	15.62mg/LSB
1100	±16g	31.25mg/LSB
others	reserved	-

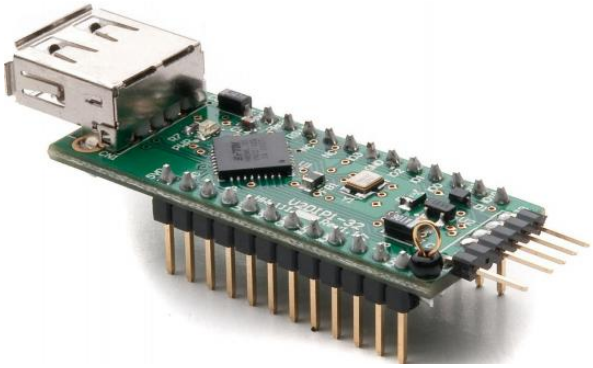
The reference measure will be the gravitational accelerometer, which is 1g. With the minim resolution, 3.91 mg/LSB we obtain 265 different acceleration values.

$$\frac{1000\ mg}{3.91\ mg/LSB} = 255.75\ \text{LSB} \sim 265\ \text{LSB}$$

It also have a extensive datasheet where it is documented all the information related with the accelerometer specifications and functionalities.

3.2 Microcontroller FTDI V2DIP1-32

The microcontroller chosen for this application is the V2DIP1-32. It is a development module of the Vinculum microprocessor of the FTDIs family.



The main advantage of this microcontroller is that a many functions for the development of a multitasking program. It also incorporates an SPI protocol, necessary to connect the accelerometer and obtain the acceleration data. All this multitasking and SPI functions are documented in the microcontroller User Guide. As it not a very popular microcontroller for this types of applications, the firmware had to been completely developed with only the User Guide documentation.

It is very easy to learn how the microprocessor IDE works. It also has a wizard assistant to configure the peripherals connections, which allows configuring the connections quickly and without errors. This helps a lot in development, as the only focus is in the code, and no the connections. Also, the IDE has a short access to the microprocessor User Guide, very helpful to a quickly check of the available functions.

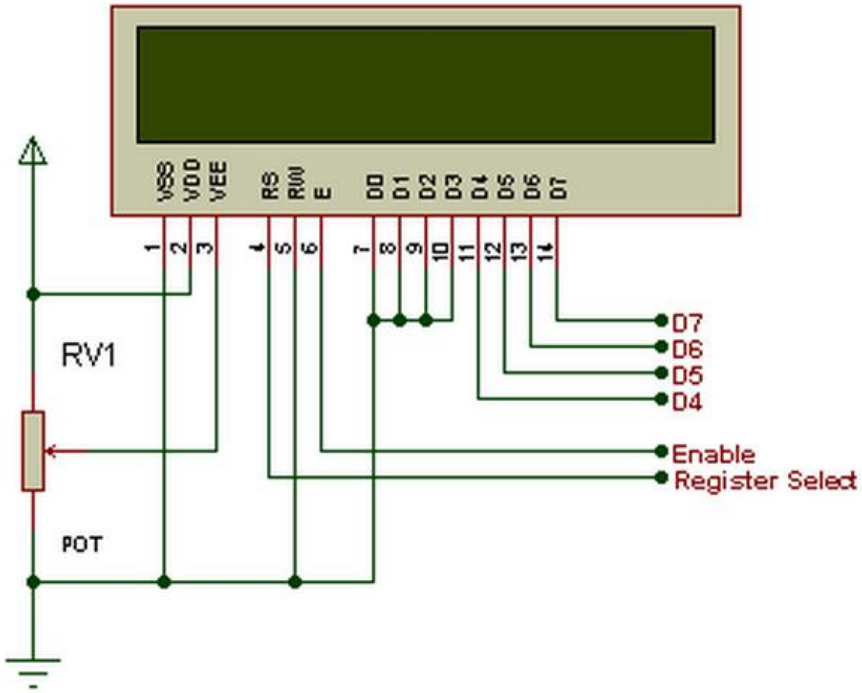
The size of the development board is smaller than other possible development boars, like Arduino, but it has enough input and output pins to connect the all the peripherals (12 pins). The development module easily placed in a prototype board thanks to its output pins.

It needs an extra module to debug the programs into the microcontroller but it is remove for the final board, so the microcontroller board is half-size. Both, microcontroller and debugger boards low prices, so they do not increase to much the final device price.

3.3 SCREEN MIDAS MC21605A&W-SPR

This is a lcd screen with 2 lines and 16 character. This type of screen is very popular in these type of applications. The size is big enough to show short sentences but it not over size the project, as it has similar size to the accelerometer and microcontroller boars together.

This screen needs 8 lines for communications, plus 3 control lines and 3 lines to power it up. It also has a second connection option which only requires 4 communication lines and 2 control line. The three power lines are: VCC, GND and a line for the screen contrast adaption. These 3 lines are connected to a 10kΩ potentiometer, so the screen contrast can be regulated, and also to the power alimentation.



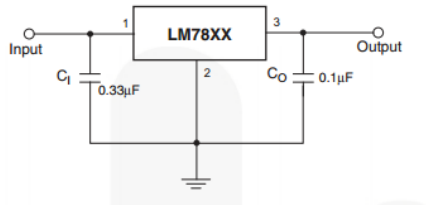
It is chosen the second connection type because it only requires 6 output pins from microcontroller. Although communications takes more time, is less than the frequency of

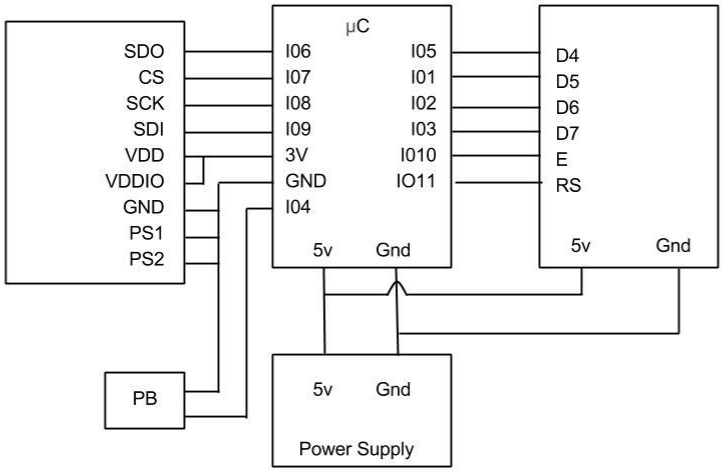
reading the accelerometer measures. In the case the first communication is wanted, the microcontroller will need to be upgraded to one with more pins, which will make the devices unnecessarily more expensive.

Figure shows the block diagram for the screen.

3.4 Block diagram

This are the block diagrams for the power supply and for all the system.





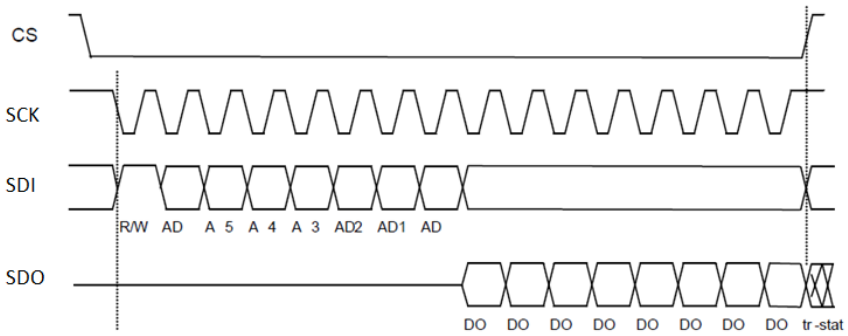
4 Firmware development

4.1 Accelerometer driver

4.1.1 SPI communication

The accelerometer data is obtained from the accelerometer sensor through an SPI connection. SPI is a protocol of serial communication with two communication lines MOSI and MISO plus the clock line and the chip select.

One of the devices is the master, the one which controls the communication, in this case the microcontroller. The others are the slaves, they only communicate when the master allows them. In this case there is only one slave, the accelerometer sensor.



The accelerometer SPI protocol is show in the figure, it only will be listening to the microprocessor when the chip select line (CS) is put to cero. Then it will read 8 bits form the MOSI line (SDI) bon the rising edge of the clock signal (SCK), which means the microprocessor has to write in the falling edge of the clock line. The fist one indicates the operation type: read or write, the other seven left are the register address it is wanted to read or write.

A register is a part of the memory of the accelerometer where some information is placed. Some of them can be written, like the configuration ones. Others are read only register, like the acceleration data registers. There are also a few registers reserved registers for internal use of the accelerometer.

Once the accelerometer has read the 8 first bit, in a write operation it will read the following 8 bits of the same line, which are the data to write in the register. In a read operation it will

write in the MISO line (SDO) the 8 bit data contained is the indicated register. If in a read operation, the master continue with the CS line at low, the accelerometer will write the data of the next register to the indicated one. It will continue writing data until the CS line is high.

The SPI master communication is configured with this information in the function spi_setup(). This is the configuration set:

- Clock Phase: data is latched on the clock leading edge
- Clock Polarity: clock is active in the high level
- Data transmit order: MSB fist.
- Clock Frequency: 10kHZ, it is 10 times lower than the accelerometer maximum frequency. there is no need of a faster communication and with a lower frequency the sensor is not overloaded.
- Automatically toggle the slave select: active

The functions use by the microprocessor to read SPI data are:

vos_dev_read(VOS_HANDLE h, unsigned char *buffer, unsigned short len, unsigned short *read);
vos_dev_write(VOS_HANDLE h, unsigned char *buffer, unsigned short len, unsigned short *written);

h: SPI hadler

buffer: Pointer to a buffer from which to send data to the device (read) or to receive data from the device (write).

Len: number of clock cicles in bytes (for 8 clock cycles, len=1)

Read/written: number of bytes read from or written to the device.

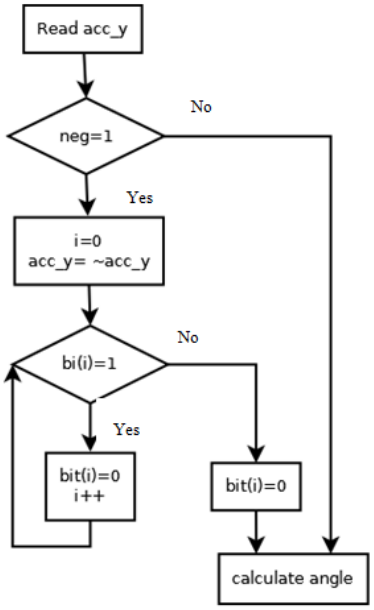
The accelerometer has six registers where accelerations are written, two for each axis. One of them contains the MSB part of the acceleration in one axis (bits 9-2). The other one contains the LSB part (bits 1-0). The data contained in these register is given in two's complement.

To calculate the angle of the inclinometer with the horizontal is going to be read both y-axis containing the y-accelerometer data. Then, data is going to be transform from two's complement into binary converted and safe in one hexadecimal variable.

4.1.2 Data transformation

Before operating with the obtained data from the accelerometer, it has to be transformed. First thing to do is to safe data of both y-axes register into one variable of 2 bytes in the rigth order of bits.

Then it is check if the most significant bit (acc_y bit 9) is cero, which means is a positive number, or is one, which means is a negative one. If it is cero it does not need more transformation, because positive number are the same in two's complement that in binary.

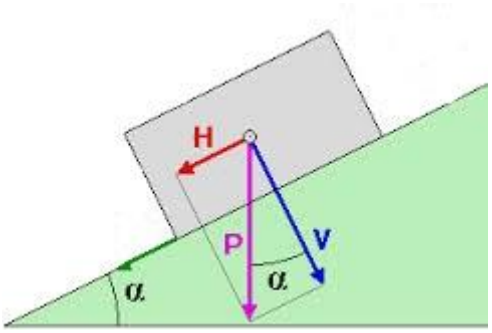


If it is negative, the binary number will be the inverse of the number in two's complement representation (acc_y=~ acc_y) plus one. As binary addition does not take into account the carry of the addition, this is doing manually. If the fist LSB is 1, the result of an addition of 1 bit will be cero and the carry number will be one. Then is check the next LSB, bit in position 2, and if it is one, carry number will be one and the thirty position bit will be check. This will continue until carry number is 0.

When the transformation ends, the data will be in binary variable. The only operation left is to get the angle from the acceleration.

4.1.3 Angle calculation

In the figure, the angle α will be equal *arcsin (H/P)*. In this project, P is the gravitational acceleration, equal to 1000mg and H will be the acceleration in the y axis (acc_y), so the equation will be :



$$\alpha = arcsin\left(\frac{acc_y}{1000}\right)$$

http://enciclopedia.us.es/index.php/Plano_inclinado

But the microprocessor does not include math.h library, which is the one containing trigonometric functions, so arcos() function is not available. In order to calculate the desired angle it is necessary to make linear approximations to this function.

Analysing the graphical representation of sin(x) , it is decided to divide this function in four parts, each one with different approximated equations. This functions are obtain taken two points of the range where the equation is going to be apply and obtaining the equation parameters.

$$\begin{cases} y_1 = m * x_1 + n \\ y_2 = m * x_2 + n \end{cases}$$

These are the equation obtained:

Range acc_y	x1	x2	y1	y2	m	n	Equation
0-139	0	0.5	0	30	60	0	$angle = \frac{acc_y * 23}{100}$
140-213	0.65	0.8	40.54	53.13	83.92334	-14.0086	$angle = \frac{acc_y * 33}{100} - 14$
214-248	0.88	0.95	61.64	71.81	145.1823	-66.1181	$angle = \frac{acc_y * 57}{100} - 66$
249-255	0.98	0.99	78.52	81.89	336.8726	-251.614	$angle = \frac{acc_y * 132}{100} - 252$
>265							$angle = 90$

The range of values of acc_y was previously obtain from the accelerometer resolution:

$$\frac{1000\ mg}{3.91\ mg/LSB} = 255.75\ \text{LSB} \sim 265\ \text{LSB}$$

The range of values for each equation its set by analysing the error between the approximation function and the real equation [arcosin ()]. When the difference between the angle obtained by approximation differed more than one degree from the real angle a new range and a new approximation to that range was calculated. This is true for all value except for the angles between 77-90. The error in this range is between 1-5 degree because of the little change of acc_y when the angle changes.

It the results table, x percentage from 0 to 1 of acc_y and y is the arcosin of x. The points are randomly selected from the range of acc_y. The parameters of the equation *m* and *n* are obtain by the equation system written above.

$$x = \frac{acc_y * 3.91}{1000} = \frac{acc\_y}{256}$$

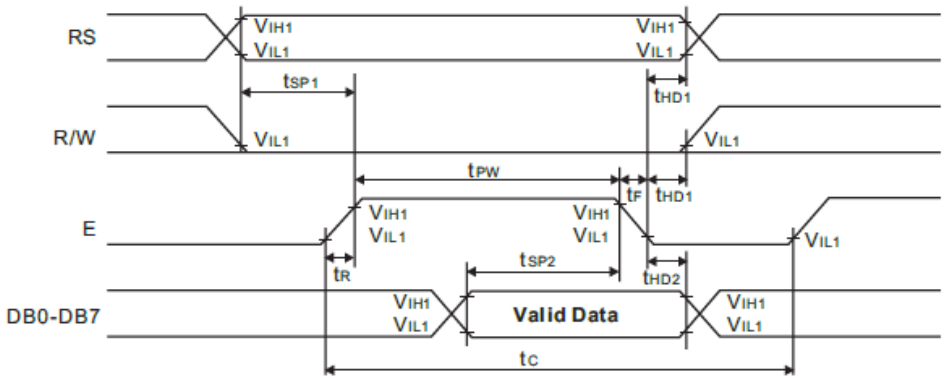
The final equation is a simplification of the all of equations. Decimals are not taken into account in the microprocessor, so the some calculation has been done in advance to avoid error because of losing the decimals in the calculation.

$$angle = m * x + n = \frac{acc_y * m * 0.391}{100} - n$$

Final equations number had been approximated, so the equation number has no decimals.

4.2 Screen driver

The microprocessor communicates data to the screen and the screen print it. Communications only works from microprocessor to screen. This communication is done by writing the necessary bit in the screen data and control lines.



RS line: will be 0 to write a command and 1 to write in the screen.

R/W line: this is set by hardware to 0 (write), because no data is read from the screen.

Data lines (DB0-DB7): where the command or letter code is written. With the 4 lines (DB4-DB7) data connection, data has to be written in to steps. Fist the MSB and then the LSB.

E line: every time new data is wanted to be read from RS o Data lines this line as to be toggle.

The function above allows the microprocessor to change output lines from high to low and form low to high. The variable value is the code of the characters or commands, which are specify in the lcd data sheet.

```
vos_gpio_set_port_mode (GPIO_PORT_A, 0xFF);
vos_gpio_write_port(GPIO_PORT_A, value);
```

To initialize the screen, the following flowdiagram has to be follow:

4.3 Multitasking system

In a multitasking system, the microprocessor switches the attention from one task to another to make it appear that many tasks are happening all at the same time. The program will have different task or threats running simultaneously.

If one thread needs to wait for another thread to do something before continuing it activity is created an interruption. The first thread will wait until the interruption happens to continue. This interruption can be activated by another thread or by an external input, like a pushbutton.

If main functions are running at the same times they may want to access to the same variables and overwrite information. This solve with the mutex function, a thread can lock a mutex to a variable so other threads are unable to access to this variable until the mutex is unlock.

For this application it is use the function semaphore, which is similar to the interruption function but instead of waiting for one event, it is able to wait for two events and continue when both appends or when one of them appends.

The main inclinometer program is divided in three threads: button, accelerometer and screen.

- Button thread: it consists on an interruption which is active when the button is pushed. When it is active it will trigger a semaphore interruption for the screen to change the tare angle and wait 100ms to avoid the button debounce and restart again.
- Accelerometer thread: it gets the accelerations and calculates the angle. Then it triggers the semaphore interruption and waits 250ms before taking a new measure.
- Screen thread: it waits until accelerometer or button semaphore. With accelerometer semaphore the screen will show the new angle in the screen. With the button, the angle is recalculated with the new tare an changed in the screen.

5 Mechanical desingn

During the development process, the component where connected with a board and . In the final mechanical design all the components were soldered into a board so the circuit is more resistant.

A 3D plastic case was design with the 3D modeller program Sketpchup. With this program simple 3d models can be easily design and modify. It also have plugging to exports the design into .stl, which is the extension recognized by 3d printers.

These designs made with a 3d printer. This process was chosen because is cheaper than others, pieces obtain can be easily personalise and modify and it is more accessible than others processes, for example, extrude the plastic into a mold.

6 Conclusions and future work

The final inclinometer design has achieved all the goals proposed for this project.

The hardware design had allowed achieving many firmware goals such as an SPI communication between accelerometer and microcontroller and the multitasking options of the microcontroller.
The screen and the pushbutton added important functions to the project, such as angle visualization and the tare function, and complemented perfectly the accelerometer and microcontroller system.

The final mechanical design makes the inclinometer more rousted and attractive, and makes easy it usage.

After testing the product, the angle accuracy was different from the calculated in theory. Comparing the inclinometer angels with a protractor angles and it differed more than 5 degrees.

For future work, the first goal is to improve the angel calculation or change the microprocessor to another with more mathematical functions.

Other goal can be to add more functions to the inclinometer, such as recording the angles in memory or send the angle information to other devices (computer or smartphone) through a wireless communication such a Bluetooth.

References

[1] Oxford University Press. *Oxford English dictionary*, 2014. Available from http://www.oxforddictionaries.com/definition/english/inclinometer?q=inclinometer [cited 21 May 2014].

[2] *Tour Egypt*, 2011. Available from http://www.touregypt.net/egypt-info/magazine-mag03012001-magf7.htm [cited 21 May 2014].

[3] Wikipedia, 2014. Available from http://en.wikipedia.org/wiki/Inclinometer [cited 21 May 2014].

[4] Riekerinc, 2014. Available from http://www.riekerinc.com/m-inclinometer/ryan_nyp_spirit_of_st.htm [cited 21 May 2014].

[5] Sensr, 2014. Available from http://www.sensr.com/pdf/practical-guide-to-accelerometers.pdf [cited 21 May 2014].

[6] Sony, 2014. Available from http://www.sonymobile.com/global-en/products/smartwear/smartband-swr10/ [cited 21 May 2014].

[7] Vinculum, 2014. Available from http://www.ftdichip.com/Firmware/Precompiled/UM_VinculumFirmware_V205.pdf [cited 22 Jun 2014].

[8] BOSH MNC050 datasheet [cited 22 Jun 2014].

[9] MC216056W-SPR datasheet [cited 22 Jum 2014].

Appendix A : Firmware

```
#ifndef _screen_H_
#define _screen_H_

#include "vos.h"
#include "GPIO.h"
#include "stdio.h"
#include "errno.h"
#include "stdlib.h"
#include "string.h"


// LCD control signalls
#define RS_cmd   0x00
#define RS_write   0x40
#define E    0x80
```

```
// LCD control comands
#define clear   0x01
#define home   0x02
//#define rigth    0x05
//#define left    0x07

#define ON   0x0C
#define OFF   0x08
#define position    0x80
#define move    0x18
#define movedelay    250


// LCD functions
void lcd_cmd (unsigned char pins);
void lcd_write(unsigned char *str);
void lcd_setup(void);


/* FTDI:EDC */

/* FTDI:SXH Externs */
/* FTDI:EXH */

#endif /* _screen_H_ */
```

```
#include "screen.h"

// Variables

unsigned char RS=0x00;



void lcd_cmd (unsigned char pins)
{
   unsigned char data;

            // Write High nibble data to LCD
            data = (((pins >> 4) &0x0F) | (RS|E) );
            vos_gpio_write_port(GPIO_PORT_A, data);

            // Toggle 'E' pin
            data &= (~E);
            vos_gpio_write_port(GPIO_PORT_A, data);

            // Write Low nibble data to LCD
            data = ((pins &0x0F) | (RS|E) );
            vos_gpio_write_port(GPIO_PORT_A, data);

            // Toggle 'E' pin
            data &= (~E);
            vos_gpio_write_port(GPIO_PORT_A, data);
}



void lcd_write(unsigned char *str)
```

```
{
        RS=RS_write;

   while(*str != '\0')
   {
      lcd_cmd(*str);
      ++str;
   }

        RS=RS_cmd;
}


void lcd_setup(void)
{
        vos_gpio_set_port_mode(GPIO_PORT_A, 0xFF);
        vos_gpio_write_port(GPIO_PORT_A,0x00);

   vos_delay_msecs(100);
   // Send Reset command
   lcd_cmd(0x03);
   vos_delay_msecs(2);
   // Send Function Set
   lcd_cmd(0x28);
   vos_delay_msecs(2);
   lcd_cmd(0x28);
   vos_delay_msecs(2);
   // Send Display control command
   lcd_cmd( 0x0C);
   vos_delay_msecs(2);
   // Send Display Clear command
```

```
    lcd_cmd(0x01);
    vos_delay_msecs(2);
    // Send Entry Mode Set command
    lcd_cmd(0x06);
    vos_delay_msecs(2);
}
```

```
#ifndef _inclinometer2_H_
#define _inclinometer2_H_

#include "vos.h"

/* FTDI:SHF Header Files */
#include "ioctl.h"
#include "SPIMaster.h"
#include "GPIO.h"
#include "stdio.h"
#include "errno.h"
#include "stdlib.h"
#include "string.h"
/* FTDI:EHF */

/* FTDI:SDC Driver Constants */
#define VOS_DEV_SPI_MASTER 0
#define VOS_DEV_GPIO_PORT_A 1

#define VOS_NUMBER_DEVICES 2



// LCD control signalls
#define RS_cmd   0x00
#define RS_write   0x40
#define E    0x80


// LCD control comands
#define clear   0x01
```

```
#define home   0x02
//#define rigth    0x05
//#define left    0x07

#define ON   0x0C
#define OFF   0x08
#define position    0x80
#define move    0x18
#define movedelay    250

// SPI control signalls
#define write   0x80
#define read   0x00


/* FTDI:EDC */

/* FTDI:SXH Externs */
/* FTDI:EXH */

#endif /* _inclinometer2_H_ */
```

```
#include "spi_read_defaultvalues.h"

/* FTDI:STP Thread Prototypes */
vos_tcb_t *tcbFIRMWARE;

void firmware();
/* FTDI:ETP */

/* FTDI:SDH Driver Handles */
VOS_HANDLE hSPI_MASTER; // SPIMaster Interface Driver
/* FTDI:EDH */

/* Declaration for IOMUx setup function */
void iomux_setup(void);

/* Main code - entry point to firmware */
void main(void)
{
        /* FTDI:SDD Driver Declarations */
        // SPI Master configuration context
        spimaster_context_t spimContext;
        /* FTDI:EDD */

        /* FTDI:SKI Kernel Initialisation */
        vos_init(50, VOS_TICK_INTERVAL, VOS_NUMBER_DEVICES);
        vos_set_clock_frequency(VOS_12MHZ_CLOCK_FREQUENCY);
        vos_set_idle_thread_tcb_size(512);
        /* FTDI:EKI */

        iomux_setup();
```

```
	/* FTDI:SDI Driver Initialisation */
	// Initialise SPI Master
	spimContext.buffer_size = VOS_BUFFER_SIZE_128_BYTES;
	spimaster_init(VOS_DEV_SPI_MASTER,&spimContext);



	/* FTDI:EDI */

	/* FTDI:SCT Thread Creation */
	tcbFIRMWARE = vos_create_thread_ex(20, 4096, firmware, "Application", 0);
	/* FTDI:ECT */

	vos_start_scheduler();

main_loop:
	goto main_loop;
}

/* FTDI:SSP Support Functions */
/* FTDI:ESP */

void open_drivers(void)
{
    /* Code for opening and closing drivers - move to required places in Application Threads */
    /* FTDI:SDA Driver Open */
    hSPI_MASTER = vos_dev_open(VOS_DEV_SPI_MASTER);
    /* FTDI:EDA */
}

void attach_drivers(void)
{
```

```
	/* FTDI:SUA Layered Driver Attach Function Calls */
	// TODO attach stdio to file system and stdio interface
	//fsAttach(hFAT_FILE_SYSTEM); // VOS_HANDLE for file system (typically FAT)
	//stdioAttach(hUART); // VOS_HANDLE for stdio interface (typically UART)
	/* FTDI:EUA */
}

void close_drivers(void)
{
	/* FTDI:SDB Driver Close */
	vos_dev_close(hSPI_MASTER);
	/* FTDI:EDB */
}

/* Application Threads */




void setup()
{
		// SPI Master Context
		spimaster_context_t spimasterCtx;
		// SPI Master IOCTL request block
		common_ioctl_cb_t spim_iocb;

		/* Open Handles */
		hSPI_MASTER = vos_dev_open(VOS_DEV_SPI_MASTER);

		/* Setup SPI Master */
		// set clock phase
```

```
        spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SCK_CPHA;
        spim_iocb.set.param = SPI_MASTER_SCK_CPHA_0;
        vos_dev_ioctl(hSPI_MASTER, &spim_iocb);
        // set clock polarity
        spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SCK_CPOL;
        spim_iocb.set.param = SPI_MASTER_SCK_CPOL_0;
        vos_dev_ioctl(hSPI_MASTER, &spim_iocb);
        // set data order
        spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_DATA_ORDER;
        spim_iocb.set.param = SPI_MASTER_DATA_ORDER_MSB;
        vos_dev_ioctl(hSPI_MASTER, &spim_iocb);
        // enable DMA
        spim_iocb.ioctl_code = VOS_IOCTL_COMMON_ENABLE_DMA;
        spim_iocb.set = DMA_ACQUIRE_AND_RETAIN;
        vos_dev_ioctl(hSPI_MASTER, &spim_iocb);
        // set clock rate
        spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SET_SCK_FREQUENCY;
        spim_iocb.set.spi_master_sck_freq = 1000000;
        vos_dev_ioctl(hSPI_MASTER, &spim_iocb);
        // set auto toggle
        spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_AUTO_TOGGLE_SS;
        spim_iocb.set.param = SPI_MASTER_SS_AUTO_TOGGLE_ENABLE_SS_0;
        vos_dev_ioctl(hSPI_MASTER, &spim_iocb);
}



void firmware()
{
```

```
/*****************************/
// Variables initialization //
/*****************************/

unsigned char dataout[2], datain[8];
unsigned short datalen=7,*p=NULL, acc_y, carry=0x01, angle;
unsigned char full=0xFF,neg=0, ref=0xFA;
int i=0, f=0;



setup();

dataout[0] = 0x82 ;
dataout[1] = 0x00;



while(1)
{
        for(f=0;f<20;f++)       {       datain[f]=0x00;    }

        vos_dev_write(hSPI_MASTER, dataout, datalen, NULL);
        vos_dev_read(hSPI_MASTER, datain, datalen, NULL);

        //Safe y-axis acceleration into a one varible (short)
        p=&datain[3];
        acc_y=*p >> 6;

        //Check positive or negative number
        if ((acc_y >> 9) == 0x01)
        {
```

```
                neg=1;
                acc_y=~acc_y & 0x03FF ;

                for (i=1; i<=12; i++)
                {
                        if ((acc_y & carry) == 0x00)
                        {
                                acc_y |= carry;
                                carry = 0x01;
                                break;
                        }
                        else
                        {
                                acc_y ^= (carry);
                                carry *= 2;
                        }
                }
        }

        //Angle calculation - revisar valores de intervalos (731)

        if (acc_y < 0x8C)   angle = acc_y * 17 / 100;
        else if ((acc_y >= 140) && (acc_y <= 213))     angle = acc_y * 17 / 100;
        else if ((acc_y >= 214) && (acc_y <= 248))     angle = acc_y * 17 / 100;
        else if ((acc_y >= 249) && (acc_y <= 255))     angle = acc_y * 17 / 100;
        else        angle=90;

    }

}
```

```
/*
** Filename: inclinometer_iomux.c
**
** Automatically created by Application Wizard 2.0.0
**
** Part of solution inclinometer in project inclinometer
**
** Comments:
**
** Important: Sections between markers "FTDI:S*" and "FTDI:E*" will be overwritten by
** the Application Wizard
*/
#include "vos.h"

void iomux_setup(void)
{
	/* FTDI:SIO IOMux Functions */
	unsigned char packageType;

	packageType = vos_get_package_type();
	if (packageType == VINCULUM_II_32_PIN)
	{
		// GPIO_Port_A_0 to pin 23 as Output.
		vos_iomux_define_output(23, IOMUX_OUT_GPIO_PORT_A_0);
		// GPIO_Port_B_1 to pin 24 as Input.
		vos_iomux_define_input(24, IOMUX_IN_GPIO_PORT_B_1);
		vos_iocell_set_config(24, 0, 1, 0, 2);
		// SPI_Master_MISO to pin 25 as Input.
		vos_iomux_define_input(25, IOMUX_IN_SPI_MASTER_MISO);
		// SPI_Master_CS_0 to pin 26 as Output.
		vos_iomux_define_output(26, IOMUX_OUT_SPI_MASTER_CS_0);
```

```
            // SPI_Master_CLK to pin 29 as Output.
            vos_iomux_define_output(29, IOMUX_OUT_SPI_MASTER_CLK);
            // SPI_Master_MOSI to pin 30 as Output.
            vos_iomux_define_output(30, IOMUX_OUT_SPI_MASTER_MOSI);
            // GPIO_Port_A_3 to pin 15 as Output.
            vos_iomux_define_output(15, IOMUX_OUT_GPIO_PORT_A_3);
            // GPIO_Port_A_2 to pin 14 as Output.
            vos_iomux_define_output(14, IOMUX_OUT_GPIO_PORT_A_2);
            // GPIO_Port_A_1 to pin 12 as Output.
            vos_iomux_define_output(12, IOMUX_OUT_GPIO_PORT_A_1);
            // Debugger to pin 11 as Bi-Directional.
            vos_iomux_define_bidi(199, IOMUX_IN_DEBUGGER, IOMUX_OUT_DEBUGGER);
            // GPIO_Port_A_7 to pin 32 as Output.
            vos_iomux_define_output(32, IOMUX_OUT_GPIO_PORT_A_7);
            // GPIO_Port_A_6 to pin 31 as Output.
            vos_iomux_define_output(31, IOMUX_OUT_GPIO_PORT_A_6);

        }

        /* FTDI:EIO */
}
```

```
/*
** Filename: inclinometer.h
**
** Automatically created by Application Wizard 2.0.0
**
** Part of solution Solution_1 in project inclinometer
**
** Comments:
**
** Important: Sections between markers "FTDI:S*" and "FTDI:E*" will be overwritten by
** the Application Wizard
*/

#ifndef _inclinometer_H_
#define _inclinometer_H_

#include "vos.h"

/* FTDI:SHF Header Files */
#include "ioctl.h"
#include "SPIMaster.h"
#include "GPIO.h"
#include "stdio.h"
#include "errno.h"
#include "stdlib.h"
#include "string.h"
/* FTDI:EHF */

/* FTDI:SDC Driver Constants */
#define VOS_DEV_SPI_MASTER 0
#define VOS_DEV_GPIO_PORT_A 1
```

```
#define VOS_DEV_GPIO_PORT_B 2

#define VOS_NUMBER_DEVICES 3
/* FTDI:EDC */

// LCD control signalls
#define RS_cmd   0x00
#define RS_write   0x40
#define E    0x80


// LCD control comands
#define clear   0x01
#define home   0x02
#define ON   0x0C
#define OFF   0x08
#define position    0x80
#define move    0x18
#define movedelay    250

/* FTDI:SXH Externs */
/* FTDI:EXH */

#endif /* _inclinometer_H_ */
```

```
/*
** Filename: inclinometer.c
**
** Automatically created by Application Wizard 2.0.0
**
** Part of solution Solution_1 in project inclinometer
**
** Comments:
**
** Important: Sections between markers "FTDI:S*" and "FTDI:E*" will be overwritten by
** the Application Wizard
*/

#include "inclinometer.h"



/* FTDI:STP Thread Prototypes */
vos_tcb_t *tcbSCREEN;
vos_tcb_t *tcbBUTTON;
vos_tcb_t *tcbACCELEROMETER;

void screen();
void button();
void accelerometer();
/* FTDI:ETP */

/* FTDI:SDH Driver Handles */
VOS_HANDLE hSPI_MASTER; // SPIMaster Interface Driver
VOS_HANDLE hGPIO_PORT_A; // GPIO Port A Driver
VOS_HANDLE hGPIO_PORT_B; // GPIO Port B Driver
/* FTDI:EDH */
```

```
/* Declaration for IOMUx setup function */
void iomux_setup(void);

/* Main code - entry point to firmware */
void main(void)
{
        /* FTDI:SDD Driver Declarations */
        // SPI Master configuration context
        spimaster_context_t spimContext;
        // GPIO Port A configuration context
        gpio_context_t gpioContextA;
        // GPIO Port B configuration context
        gpio_context_t gpioContextB;
        /* FTDI:EDD */

        /* FTDI:SKI Kernel Initialisation */
        vos_init(50, VOS_TICK_INTERVAL, VOS_NUMBER_DEVICES);
        vos_set_clock_frequency(VOS_12MHZ_CLOCK_FREQUENCY);
        vos_set_idle_thread_tcb_size(512);
        /* FTDI:EKI */

        iomux_setup();

        /* FTDI:SDI Driver Initialisation */
        // Initialise SPI Master
        spimContext.buffer_size = VOS_BUFFER_SIZE_128_BYTES;
        spimaster_init(VOS_DEV_SPI_MASTER,&spimContext);

        // Initialise GPIO A
        gpioContextA.port_identifier = GPIO_PORT_A;
```

```
	gpio_init(VOS_DEV_GPIO_PORT_A,&gpioContextA);

	// Initialise GPIO B
	gpioContextB.port_identifier = GPIO_PORT_B;
	gpio_init(VOS_DEV_GPIO_PORT_B,&gpioContextB);



	/* FTDI:EDI */

	/* FTDI:SCT Thread Creation */
	tcbSCREEN = vos_create_thread_ex(20, 4096, screen, "Screen", 0);
	tcbBUTTON = vos_create_thread_ex(20, 1024, button, "Button", 0);
	tcbACCELEROMETER = vos_create_thread_ex(20, 4096, accelerometer, "Accelerometer", 0);
	/* FTDI:ECT */

	vos_start_scheduler();

main_loop:
	goto main_loop;
}

/* FTDI:SSP Support Functions */
/* FTDI:ESP */

void open_drivers(void)
{
    /* Code for opening and closing drivers - move to required places in Application Threads */
    /* FTDI:SDA Driver Open */
    hSPI_MASTER = vos_dev_open(VOS_DEV_SPI_MASTER);
    hGPIO_PORT_A = vos_dev_open(VOS_DEV_GPIO_PORT_A);
```

```
    hGPIO_PORT_B = vos_dev_open(VOS_DEV_GPIO_PORT_B);
    /* FTDI:EDA */
}

void attach_drivers(void)
{
    /* FTDI:SUA Layered Driver Attach Function Calls */
    // TODO attach stdio to file system and stdio interface
    //fsAttach(hFAT_FILE_SYSTEM); // VOS_HANDLE for file system (typically FAT)
    //stdioAttach(hUART); // VOS_HANDLE for stdio interface (typically UART)
    /* FTDI:EUA */
}

void close_drivers(void)
{
    /* FTDI:SDB Driver Close */
    vos_dev_close(hSPI_MASTER);
    vos_dev_close(hGPIO_PORT_A);
    vos_dev_close(hGPIO_PORT_B);
    /* FTDI:EDB */
}
```

```
/* Application Threads */



void screen()
{
While (1)
        {
        Wait_semaphore()
        Lcd_write(angle);
        }
}

void button()
{
        /* Thread code to be added here */
/*
        unsigned char result=0x07;
        unsigned char num=0x00;
        unsigned char *data;

        data="waiting";
        lcd_setup;
        vos_gpio_set_port_mode(GPIO_PORT_B, 0x00);
        lcd_cmd(clear);
        lcd_write(data);

        while (1)
        {
```

```
        result = vos_gpio_enable_int(GPIO_INT_0,  GPIO_INT_ON_LOW_STATE , GPIO_B_0);
        result=0x04;
        result = vos_gpio_wait_on_int(GPIO_INT_0);



                        num++;

        }
        */

}

void accelerometer()
{

        unsigned short datalen=7;
        unsigned char dataout[8], datain[8];
        unsigned char x[2],y[2],z[2],full=0xFF,neg=0;
        int i=0;
        int f=0;



        setup();
while (1)
        {
        angle=get_angle ;
        vos_delay (250);
        }

}
```