Lab 3 – Report

Utilization of processor's accelerometer sensor to determine STMF32 Discovery Board's tilt angles

Group 7
Hoai Phuoc Truong – 260526454
Ravi Chaganti – 260469339

ECSE 426
Department of Electrical and Computer Engineering
McGill University

27th October, 2015

Table of Contents

1. Abstract

The main objectives of this experiment are to understand the peripherals that are connected to the STMF32 discovery board and utilize the accelerometer sensor provided by the processor to calculate the tilt angles along the x, y, and z axis to design a game, 'Guess the Angle'. A keypad and 4-digit seven-segment clock display will be used to take inputs from the user to guess the angle and to display the numeric value of the inputs. The processor's LEDs will then be used to show if the input guessed by the user is right (within a certain limit) and wrong (more or less than the actual value).

2. Problem Statement

In our experiment, our primary focus is to establish a link between the accelerometer sensor, which is an external MEMS sensor chipset, and the processor by using external interrupt module for the external chipset and nested vector interrupt controller (NVIC) module for the microprocessor to generate and control interrupt signals on both sides. This requirement must follow a proper initialization of SPI (serial peripheral interface) to enable communication of data between the processor and the sensor peripheral.

The system must be built in such a way that it measures the calculation of angles 100 times in one second and proper equations must be implemented to calculate the pitch and roll. Each MEMS sensor will also have a different driver that will work with the microprocessor which needs to be identified as well.

Since accelerometer sensor is not completely accurate to measure the tilt angles, we must calibrate the sensor's raw measurements using the least square approximation method. After obtaining the correct raw measurements of the tilts, these values must be filtered using moving average filter that was implemented in experiment 2 to maintain consistency of the readings.

To design the guess game, we need to use the keypad so that the user can enter a two or three digit integer. We also need to implement a non-numeric button which will 'enter' the user's answer. To display the input, we must use the 4-digit seven segment display that will display the guess in the form of $XXX°, XX.Y° \; or \; X.YY°$. We also need to implement TIM peripherals as hardware timers to generate delays for the clock display.

3. Theory and Hypothesis

3.1 Tilt Angles

To calculate the pitch (angle between the X_b axis and the horizontal plane) and roll (angle between the Y_b axis and the horizontal plane), we will be using the following trigonometric equations (Reference 1 in Appendix) to provide constant sensitivity over 360° rotation:

$$pitch = \alpha = \arctan\left(\frac{A_{X1}}{\sqrt{(A_{Y1})^2 + (A_{Z1})^2}}\right)$$

Equation 1: calculation of pitch

$$roll = \beta = \arctan\left(\frac{A_{Y1}}{\sqrt{(A_{X1})^2 + (A_{Z1})^2}}\right)$$

Equation 2: calculation of roll

Yaw (along z-axis) cannot be deduced from the accelerometer sensor readings because unlike pitch and roll, gravity force is independent of it.

3.2 External Interrupts/GPIO Mapping

Since there are 16 types of GPIO pins each for GPIOA, GPIOB, GPIOC, GPIOD and GPIOE, the external interrupt lines are defined (Reference 2 in Appendix) to the 16 types accordingly. For example, if we choose GPIOA_pin_1 on the processor to enable the external interrupt coming from the accelerometer sensor, the external interrupt line that will be used is EXTI1.

In our implementation of configuring the accelerometer an as external chipset, we will be using GPIOE_pin_0 which is defined for the external interrupt line EXTI0.

3.3 Accelerometer Data

The accelerometer measurements are sent to the following 6 registers:

1. OUT_X_H
2. OUT_X_L
3. OUT_Y_H
4. OUT_Y_L
5. OUT_Z_H
6. OUT_Z_L

These registers contain the most significant bit (MSB) that is defined by H and least significant bit (LSB) that is defined by L for all angles (Reference 3 in Appendix). The total measurement of tilt angle is calculated by adding the MSB and LSB (shifted left by 8 bits).

3.4 Least Square Approximation

To calibrate the processor's tilt angles that are measured from the accelerometer sensor, the least square approximation method will be used:

$$X = [w^T.w]^{-1}.w^T.Y$$

Equation 3: least square approximation to calculate X

$$Y = w.X$$

Equation 4: generic least square approximation

w is the N×4 matrix where N is the number of results that are being considered for calibration and 4 is the columns for 1, pitch, roll and yaw (Reference 4 in Appendix). X is the matrix which contains the normalization values or calibration parameters and Y is the output (normalized earth gravity vector) defined for a particular axis. Once the X matrix is obtained, constants will be added for each axis to each normalized values in X matrix for each axis.

3.5 Seven Segment Display

For each digit, the seven segment represents A, B, C, D, E, F and G as seen in the following figure:

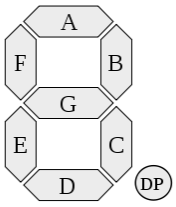

Figure 1: 7 segment display

To display digit 1, B and C must be set to 1 and all the other segments must be set as 0 (Reference 5 in Appendix). To display 8, G must be set to 0 and all the other segments must be set as 1. Table1 in Appendix summarizes the selection of segments for all possible 16 digits.

A, B, C, D, E, F and G segments for all 4 digits are connected to the same lines with pins 14, 16, 13, 3, 5, 11, and 15 on the clock display. The decimal points for all the 4 digits are connected to pin 7 on the clock display. Finally, all the 4 digits are connected to pins 1, 2, 6 and 8. This structure optimizes the usage of pins in an efficient way (using 12 pins instead of 32). All the pin connections for the clock display can be seen in Figure2 in Appendix.

3.6 Keypad

For a generic keypad with 16 digits in the form of a 4×4 matrix, there are 4 rows and 4 columns (see Reference 6 in Appendix). Initially, the rows are configured as output and the columns are configured as input. Using pull down resistors, the default logic is 0 until a button is pressed which changes the logic to 1. Once a button is pressed, the corresponding column lines are identified and saved and set as low and the input is switched from column to row. Then the corresponding row lines are read and matched with the previously saved column lines to detect the button that is pressed on the keypad. We will be using interrupts to identify whether a button has been pressed or not.

3.7 Serial Peripheral Interface (SPI) protocol

SPI protocol is used to read and write the registers from the external device with the help of 4 wires:

1. Chip Select (CS): low at the start of the transmission signal and high at the end, connected to PE3 on the microprocessor
2. Serial Port Clock (SPC): controlled by SPI master, no transmission when CS is high, connected to PA5 on the microprocessor
3. Serial Port Data Input (SPDI): connected to PA7 on the microprocessor
4. Serial Port Data Output (SPDO): connected to PA6 on the microprocessor

Data is written to the device when bit_0 is 0 (chip will enable SPDI) and data is read from the device when bit_0 is 1 (chip will enable SPDO). It will take 16 clock cycles for the read and write register commands to be completely executed. Connections of the accelerometer sensor using SPI protocol to the microprocessor and description of 4 wires can be seen in Figure6 and Reference 8 in Appendix respectively.

3.8 Moving Average Filter

In our implementation of the moving average filter to smoothen the values of the output signal, we will be using a circular buffer as it is connected end-to-end and is easy to use to buffer continuous data streams. In this model, we will be taking the average of the number of elements based on the following equation (Reference 9 in Appendix):

$$y[i] = \frac{1}{M}\sum_{j=0}^{M-1} x[i+j]$$

Equation 5: moving average filter equation

where M is the filter depth (D), x is the circular buffer and y is the continuous moving average filter.

3.9 Keypad de-bouncing

When a key is pressed, the voltage that is measured sometimes does not have a clean edge (has spikes and inconsistency). This glitch results in keypad de-bouncing. This results in an increase in the number of counts (if implementing a counter) and affects the transmission of data signals to the microprocessor (Reference 10 in Appendix). Proper assigned pins must be used to avoid this effect.

3.10 Timer's period equation

To calculate TIM's parameters of prescaler and period, the following equation will be used (Reference 11 in Appendix):

$$PWM\_frequency = \frac{\frac{default\_frequency}{prescaler + 1}}{TIM\_Period\ + 1}$$

4. Implementation

4.1 Accelerometer Sensor's Interrupt Configuration

The first step was to identify the type of processor which would define the drivers that would need to be installed. Since we have STM32F4 of type MB997C, we used the base project with LIS3DSH drivers.

We defined a struct accelerometer_info which takes the parameters of roll and pitch (in accelerometer_interface.h). For the sensor initialization, there are multiple configurations (in accelerometer_interface.c) that need to be set in order for the interrupts and data transfer to take place. A list of all the implemented configurations to initialize the accelerometer sensor can be seen in Table1.

Configuration	Key Parameters	Explanation
LIS3DSH_InitStruct	Power_Mode_Output_Data Rate Axes_Enable	Used to configure LIS3DSH drivers. Set the output DATARATE as 100Hz and enable X, Y and Z axes
DRYInterruptConfig	Interrupt_signal	Used to enable the interrupt when data is ready. To generate a high, set interrupt_signal as LIS3DSH_ACTIVE_HIGH_INTERRUPT_SIGNAL

GPIO_InitStruct	GPIO_Pin	To set pin as input for external interrupt, used GPIOE and initialized GPIO_Pin as GPIO_Pin_0
EXTI_InitStruct	EXTI_Line	To connect PE0 to Line0, set EXTI_Line as EXTI_Line0
NVIC_InitStruct	NVIC_IRQChannel NVIC_IRQChannelPreemptionPriority	To add IRQ to the NVIC as global external interrupt to MCU, NVIC_IRQChannel set as EXTI0_IRQn with preemptionPriority as 0x00 (highest priority)

Table1: Accelerometer sensor initialization

We defined the GPIO initialization struct for GPIOE_pin_0 and linked it to EXTI_Line0 by calling SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource0). To check the status of the external interrupt, we defined a method EXTI0_IRQHandler. This method will check if EXTI_Line0 is not RESET (which means that there is an interrupt request), and will initiate the accelerometer sensor's raw readings.

4.2 Accelerometer Sensor's Readings

We defined a method accelerometer_read_raw() to read the raw values from the sensor and store them in a buffer of size 6 (for 3 axes and MSB and LSB). For x_raw, y_raw and z_raw, we added the MSB and LSB (which is shifted to the left by 8 bits).

To calibrate the sensor's raw x, y and z measurements, we implemented the least square approximation method. We defined the normalizing matrix initially as {{1,0,0}, {0,1,0}, {0,0,1}} and multiplied with the raw values of x, y, and z. The actual values of the normalization matrix are described in the observation section. We passed these values continuously to the moving average filter buffer to streamline the raw values to maintain consistency of readings. This implementation can be seen in accelerometer_read method. Lastly, to convert the measurements from radians to degree, we used the equations 1 and 2 (from section 3.1) to calculate the rotation in the method accelerometer_calculate_rotation for pitch and roll.

4.3 4-digit seven segment initialization and display

To simplify the connections on the breadboard, we separated the GPIO pins for the 4 digits (GPIOB), 7 segments for each digit (GPIOD) and the DOT (decimal point). In seven_segment_init method, we initialized all the GPIOB and GPIOD pins (in sevent_segments_interface.c). The pin connections between the processor and the 7 segment digits on the clock display can be seen in seven_segments_interface.h. It also includes the GPIOB_Pin_8 for the degree symbol which is connected to pin 10 of the clock display. The pin connections for the clock display can be seen in Figure2 and Figure3 in Appendix.

Since the maximum current rating for the clock display pins is 25mA (Reference 7 in Appendix) and the input voltage is 3.3V, we connected 330Ω resistors (4band: orange, orange, brown, gold) between GPIO pins on the processor and the 7-segment pins to protect the clock display.

A list of all the implemented configurations to initialize the clock display can be seen in Table2.

Configuration	Key Parameters	Explanation
TIM_TimeBaseStructure	TIM_Period TIM_Prescaler	Defined time base configuration TIM2 to generate hardware based time delay. Set TIM_Period as 99, TIM_Prescaler as 8049Hz, enabled ITConfig and counter for TIM2
NVIC_InitStructure	NVIC_IRQChannel NVIC_IRQChannelPreemptionPriority	To add IRQ to the NVIC as global external interrupt to MCU, NVIC_IRQChannel set as TIM2_IRQn with preemptionPriority as 0x01 (second highest priority)
gpio_init_s	GPIO_Pin	Initialized all the 4 GPIOB pins for 4 digits and 8 GPIOD pins for 7 segments and 1 decimal point

Table2: Clock display initialization

To configure the interrupt handler for TIM2, we defined the method TIM2_IRQHandler which will verify if it's reset and then call seven_segment_periodic_display method which will enable the display functions of the seven segment clock display.

After the initialization of all the GPIO pins, we configured each digit by setting the digit under operation as low (ResetBits) and all the other digits to high (SetBits). This implementation can be seen in choose_digit method. To display 0-F at each digit place, we used the seven segment theory and defined an array of dimension 16×7. We set values of A, B, C, D, E, F, and G as 0 or 1 based on the information in Table4 in Appendix. This implementation can be seen in seven_segment_display method.

To display the digits in the form $XXX°, XX.Y° \; or \; X.YY°$, we defined a method seven_segment_set_display_float which will take into consideration the number of decimal

places of the user's input value. For input values less than 10, less than 100 and more than 100, we defined the decimal places as 2, 1 and 0 respectively. These implementations can be seen in seven_segments_set_display_float_smart and seven_segment_periodic_display methods.

4.4 Keypad initialization and implementation

To initialize the keypad's pins on the processor, we used 8 GPIOE pins for 4 rows and 4 columns. Table3 describes all the configurations that are used to initialize the keypad.

Configuration	Key Parameters	Explanation
TIM_TimeBaseStructure	TIM_Period TIM_Prescaler	Defined time base configuration TIM3 to generate hardware based time delay. Set TIM_Period as 199, TIM_Prescaler as 8049Hz, enabled ITConfig and counter for TIM3
NVIC_InitStructure	NVIC_IRQChannel NVIC_IRQChannelPreemptionPriority	To add IRQ to the NVIC as global external interrupt to MCU, NVIC_IRQChannel set as TIM3_IRQn with preemptionPriority as 0x02 (third highest priority)
configure_gpio_input	GPIO_Pin GPIO_Mode GPIO_PuPd	Defined to take input from keypad. Initialized all 8 pins, set GPIO_Mode to IN and GPIO_PuPd to GPIO_PuPd_DOWN (to limit the current)
configure_gpio_output	GPIO_Pin GPIO_Mode	Defined to take input from keypad. Initialized all 8 pins and set GPIO_Mode to OUT

Table3: Keypad initialization

We implemented switch_readings method to check the readings of column and row as input or output as defined section 3.6 (Theory and Hypothesis). If we are reading columns first, we set the output to row and set the input to columns. If we are reading rows first, we set the input as row and output as column.

To configure the interrupt handler for TIM3, we defined the method TIM3_IRQHandler which will verify if it's reset and then call keypad_periodic_read_state method which will select the column and row with respect to the pin state that is pressed periodically.

4.5 Guess the Angle Game

After initializing the keypad and the 4-digit seven segment display, we defined lab3_game.c which will initialize the game. We defined a buffer[100], which will store the values being entered by the user, READ_INPUT (state as 1), GUESS (state as 2), state (which is either 1 or 2) and LAB3_GAME_BOUND (which is the offset of 4 degrees). We calculated the magnitude of the actual tilt of rotation by calling the method accelerometer_calculate_angle.

If the current state is 1, then the keypad will take in the input until the $*$ symbol on the keypad is not pressed. Once the $*$ symbol is pressed, the state will change to GUESS. We calculated the magnitude of the difference between the user's guess and the actual angle of the tilt of processor.

If the difference between the user's guess and the actual angle is within the offset, we display the green LED (GPIOD_pin_12). If the difference is more than the offset, we display the blue LED (GPIO_pin_15) and if the difference is less than the offset, we display the red LED (GPIO_pin_14). We also defined a parameter LAB3_GAME_MAX_TRY (set as 5) to keep track of the number of tries the user has to guess the correct angle of tilt. If the number of tries is exceeded, the orange LED (GPIO_pin_13) is highlighted and the game will be over.

The final setup of keypad, microprocessor, 4-digit seven segment display and cable connections can be seen in Figure4 in Appendix.

5. Observation

After converting the raw values that are measured from the accelerometer sensor to degrees, we tested the accelerometer sensor by tilting the microprocessor along the x and y direction. While placing the board flat and perpendicular on a table, we observed the calculations of pitch and roll were not accurate (off by 1.5 degrees). We implemented the least square approximation method to calibrate the readings. We used MATLAB to calculate the X matrix (in equation 3 in section 3.4) and obtained the following matrix after modifying the values in the identity normalized matrix:

{1.05980676, 0.02639947, -0.00279462}
{-0.03077862, 1.1679603, -0.01424515}
{-0.01497205, 0.05370348, 1.22005476}

The constant values for x, y and z obtained are -163.44321229, -142.16213624, -112.6033728. With these values, calibration was quite accurate with an uncertainty of 0.2 degrees and worked well while placing the board flat and perpendicular on the table.

While implementing and testing the keypad, we observed that some of the pins that were connected to the processor were not functioning properly in terms of reading high and low logic values. This was due to the fact that some of the pins were not meant to be used for connection purposes (for example PA13). We manually measured the voltage across the pins by using the probe by connecting one end to the ground and the other end across the pin to be measured. We made a note of all the pins that we observed that can be used for the keypad in keypad_interface.h file. We also observed de-bouncing of switches while measuring the voltage with the probe, which was corrected by manipulating the values of duration of the button being pressed and the time delay.

We tested the clock display by first displaying all numbers from 0-F on one digit and then for all the digits. While doing this, we noticed the flickering of all the 4 digits on the clock display. This was corrected by increasing the TIM frequency which will transfer data signals at a faster rate.

While testing the moving average filter to streamline the raw measurements calculated by the accelerometer sensor (from Equation 5 in section 3.8), we used multiple values of depth (D) to find the appropriate filter average system. We observed that depth (D) of 14 was the best value which can be seen in figure 5 in Appendix.

6. Conclusion

Overall, the peripherals of keypad, clock display and the accelerometer sensor were in synchronization with respect to the processor with proper utilization of external interrupts to read and transfer data. The final game design was successfully implemented by taking user's guess (for the tilt angle) as input to show on the clock display and verify if the guess is right or wrong with respect to the angles measured from the accelerometer sensor.

7. Appendix

References

1. Application Note. (n.d.). Retrieved October 29, 2015, pp. 6,7,13, from http://www.st.com/web/en/resource/technical/document/application_note/CD00268887.pdf
2. Reference Manual. (n.d.). Retrieved November 1, 2015, pp. 208, from http://www.st.com/web/en/resource/technical/document/reference_manual/CD00171190.pdf

3. Accelerometer Application Notes. (n.d.). Retrieved November 1, 2015, pp. 13, 14, from http://www.st.com/web/en/resource/technical/document/application_note/DM00026768.pdf
4. Application Note. (n.d.). Retrieved October 29, 2015, pp. 14,15,16, from http://www.st.com/web/en/resource/technical/document/application_note/CD00268887.pdf
5. Room, T. (2010). 7-segment displays tutorial, LEDs, sample circuits. Retrieved November 1, 2015, from http://www.sentex.ca/~mec1995/tutorial/7seg/7seg.html
6. Suyyagh, A. (n.d.). A Generic Keypad and LCD Tutorial. Retrieved November 1, 2015, pp. 7,8,9, from https://github.com/gmart772/ECSE426G4/blob/master/Documentation/A Generic Keypad and LCD Tutorial.pdf
7. Li, F., & Xu, L. (n.d.). LED Numeric Display, 4 Digit. Retrieved October 23, 2015, pp. 2,3, from http://www.adafruit.com/datasheets/BL-Q56C-43.pdf
8. Application Note - LIS302DL. (n.d.). Retrieved November 2, 2015, pp. 13,14, from http://www.st.com/web/en/resource/technical/document/application_note/CD00098549.pdf
9. Smith, S. (1997). Moving Average Filters. In The scientist and engineer's guide to digitalsignal processing (pp. 277-280). San Diego, Calif.: California Technical Pub.
10. Greensted, A. (n.d.). Debounce. Retrieved November 2, 2015, from http://www.labbookpages.co.uk/electronics/debounce.html
11. Majerle, T. (2014, May 10). STM32F4 PWM tutorial with TIMERs - STM32F4 Discovery. Retrieved November 3, 2015, from http://stm32f4-discovery.com/2014/05/stm32f4-stm32f429-discovery-pwm-tutorial/

Figures

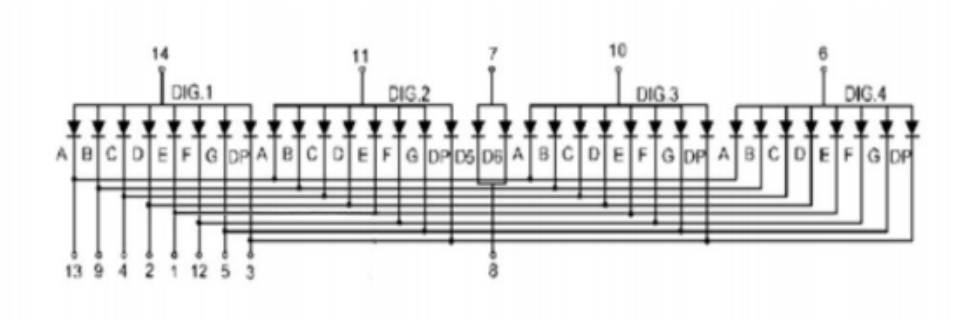

Figure2: Clock display pins (Reference 7 in Appendix)

16	15	14	13	12	11	10	9
1	2	3	4	5	6	7	8

Figure3: 4-digit seven segment pin display assignment

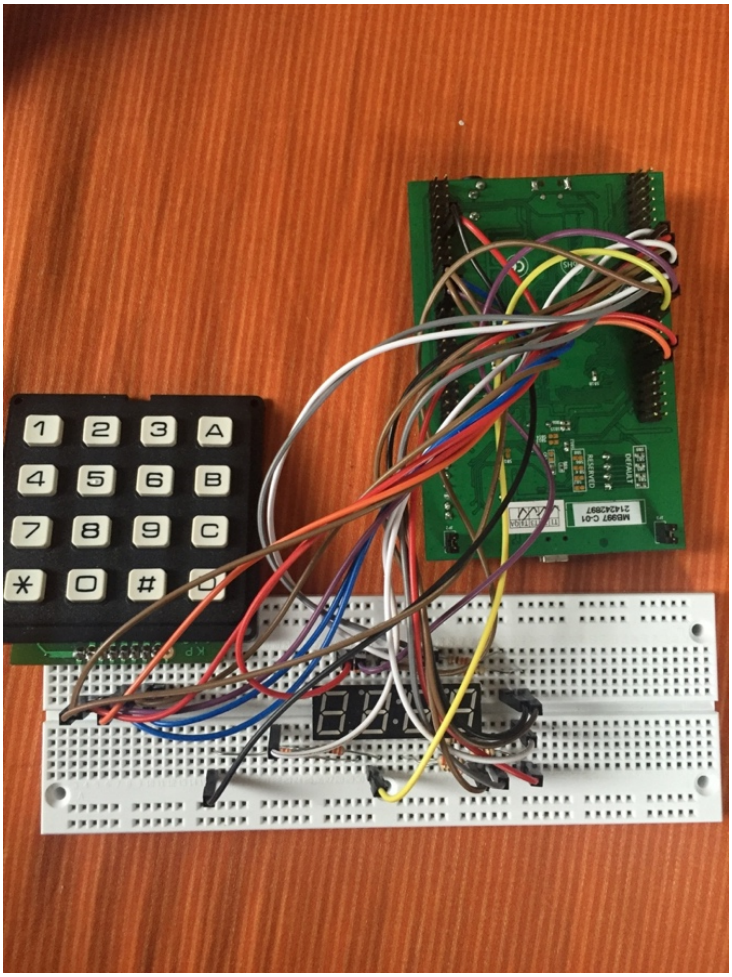

Figure4: Lab3 setup

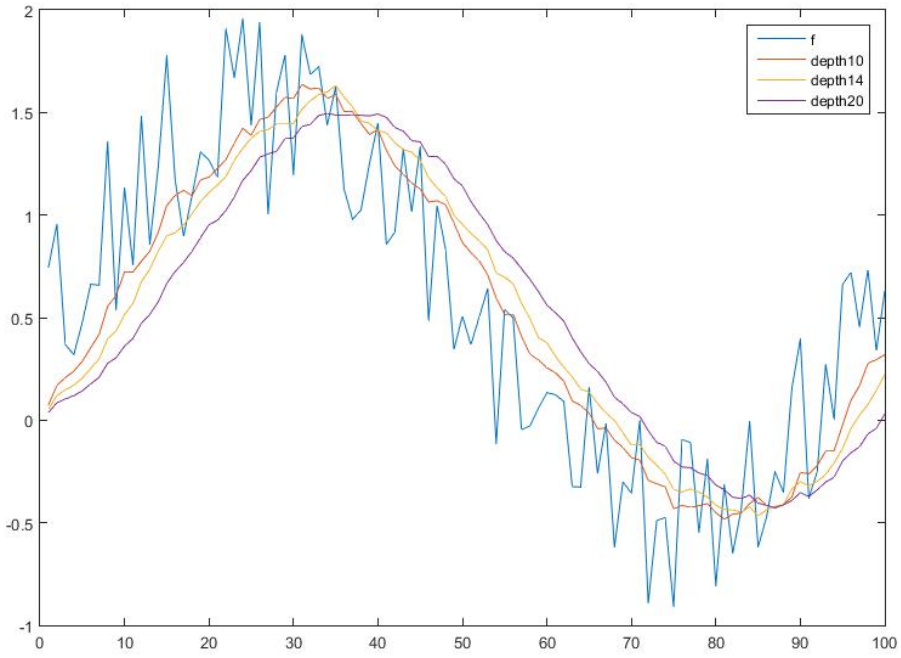

Figure5: Plots for different depths for moving average filter

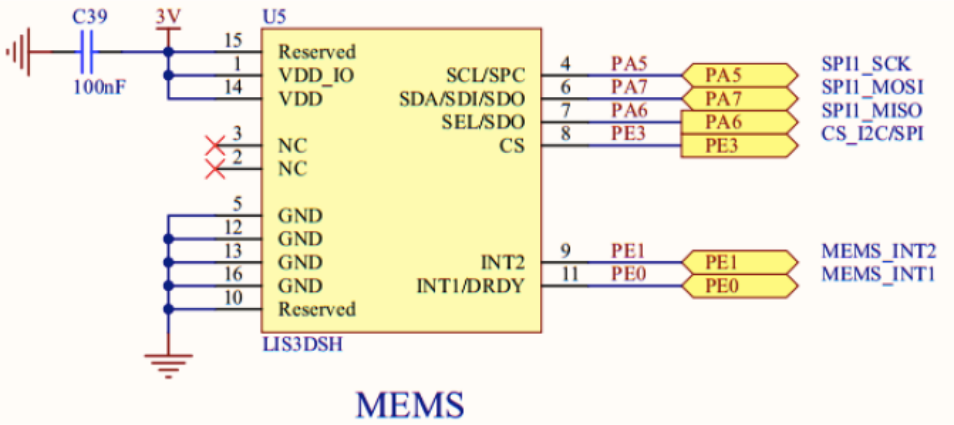

Figure6: SPI protocol and initialization

Tables

A	B	C	D	E	F	G	Digit
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	1	1	9
1	1	1	0	1	1	1	A
0	0	1	1	1	1	1	B
1	0	0	1	1	1	0	C
0	1	1	1	1	0	1	D
1	0	0	1	1	1	1	E
1	0	0	0	1	1	1	F

Table4: Seven segment display for 0-F digits (0 is off, 1 is on)