ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT

Laboratory Project: Audio Frequency Based Stepper Motor Driver

Laboratory Project

Objectives

In the EE447 laboratory work, you were expected to familiarize yourself with the operation of TM4C123G and its utility modules. In this final project you are expected to gather the previous experience on the microcontroller with novel information to achieve a multi-functional task. The objectives of the project are as follows:

- Interpretation of the necessities of complex task and encapsulation into sub-task
- Fulfillment of co-operation of utility modules
- Understanding a given complex hardware and compatibility of its components
- Writing a multi-task software for a given complex setup
- Introduction to the serial communication on TM4C123G and utilization of the facility on SPI protocol.
- Introduction to the ARM CMSIS DSP software library.

1 Project Definition

In this project, you are expected to build a stepper motor driver system based on an applied audio signal's frequency. You are going to sample an audio signal with the ADC module using a microphone, calculate that signal's frequency using ARM CMSIS DSP library, and drive a stepper motor using the GPTM module. To show the current configuration and measurements, you will use a Nokia 5110 LCD, which you will drive using SPI module. You will also use GPIO for the on-board RGB LED and pushbuttons.

The system has three major functions:

1.1 Audio Sampling

The system continuously samples audio signal using a microphone, at a constant sampling frequency. It stores the audio samples in an array of 512 elements. When the array is filled, your system calculates the audio frequency and updates the output elements' states.

1.2 Frequency Detection

The system should react to the sampled audio's frequency. To detect the frequency, ARM CMSIS DSP library will be used. Usage of this library is explained in Section 4.

1.3 User Interface

The system has three output elements. Firstly, on the LCD screen, detected frequency and its amplitude as well as threshold values are displayed. Secondly, on-board LEDs are turned on or off according to the detected frequency. If no frequency is detected, the LEDs are off. If detected frequency is low, red LED is on. In middle frequencies, green LED is on, and for high frequencies blue LED is on. When an LED is on, its brightness should be proportional to detected amplitude. Finally, the stepper motor should rotate at a speed proportional to the last detected frequency. Rotation direction can be set using the on-board switches. Amplitude threshold can be set using the potentiometer.

2 Requirements and Restrictions

The overall functionalities of audio frequency based stepper motor driver system are described in Section 1. For the sake of simplicity there will be some assumptions about operation. Additionally, there will be restrictions in both the implementation and the hardware to be used.

2.1 Requirements

You will use MAX4466 as the microphone, Nokia 5110 LCD screen, on-board LEDs and the stepper motor as the user interface. You will also use a potentiometer for threshold setting. Following are the functional requirements:

1. The system should have three thresholds: low and high frequency thresholds and an amplitude threshold.
2. If the dominant frequency's amplitude is lower than the amplitude threshold, all LEDs must be off and the stepper motor must continue to its rotation at its current speed.
3. If the dominant frequency's amplitude is higher than the amplitude threshold, stepper motor's speed should be adjusted in proportion to the calculated frequency. That is, the motor should rotate slowly in low frequencies and quickly in high frequencies. Additionally,
   - If the calculated frequency is lower than the low frequency threshold, red LED should be on.
   - If the calculated frequency is between the low and high frequency thresholds, green LED should be on.
   - If the calculated frequency is higher than the high frequency threshold, blue LED should be on.

   When an LED is on, its brightness should change in proportion to dominant frequency's amplitude. For this purpose, PWM should be used.
4. Stepper motor should be able to rotate in both directions. When the on-board SW1 button is pressed, motor should rotate in CW direction. When the SW2 button is pressed, motor should rotate in CCW direction. Initial direction of rotation is left to you.
5. The user should be able to see the configured upper and lower frequency and amplitude thresholds on the screen.
6. The user should be able to see the current frequency and amplitude on the screen.
7. Frequencies are to be shown in Hertz units with four decimal digits. e.g.: `1250 Hz`. The maximum frequency is limited to 2000 Hz.
8. For the amplitude, there is no unit or scale restrictions. That is, you do not need to convert amplitude values to Volts or any other units. You may use any system that changes in proportion to audio volume.
9. The user should be able to set the amplitude threshold using a potentiometer. For this purpose, another ADC channel should be used.

2.2 Restrictions

2.2.1 Hardware Restrictions

The project uses the following components:

- MAX4466 Microphone Module
- NOKIA 5110 LCD Screen
- Potentiometer
- 2 Buttons and RGB LED Placed on the TM4C123G Board
- 28BYJ-48 Stepper Motor and ULN2003A Driver

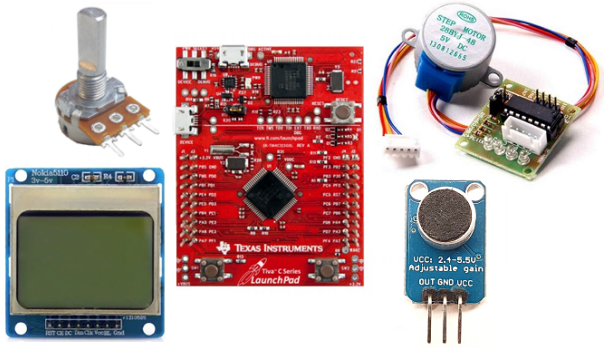

Figure 1: Components

2.2.2 Implementation Restrictions

To ensure the learning outcomes of the course and this laboratory, there are a few restrictions on implementation:

1. Microphone must be read using the ADC module.
2. Microphone ADC sampling must be done in SysTick interrupt handler. Sampling frequency should be 4000 Hz.
3. 512 point FFT should be used to detect frequency of the input, see Section 4.
4. Stepper motor must be driven using GPTM interrupts. That is, the motor signals should advance one step in the determined direction every time in a GPTM interrupt handler that you will configure. This way you can stop the motor by disabling the timer, and adjust the speed by changing the reload register value. For reasonable operation of the stepper motor, you should advance steps no sooner than 5 milliseconds.
5. GPIO interrupt or polling may be used to detect button press events of onboard buttons SW1 and SW2.

The visual style of the user interface on Nokia LCD screen is up to you as long as you fulfill the requirements specified in Section 2.1.

2.2.3 Programming Language Restrictions

1. In order to read audio samples (only reading, not ADC initialization) from the GY-MAX9814 module, only ARM assembly language must be used. That is, calling any other C function/subroutine etc. is forbidden.
2. C programming language can be used for the remaining sections.

3 Tips

On board buttons are connected to pins PF0 and PF4 [1]. Note that PF0 is locked [2] and you should unlock that pin to program it. You might want to refer pages 684 and 685 of the TM4C123GH6PM manual [2].

You will need to use interrupts and configure the priority of the interrupts. Recall that to configure an interrupt, you should know the interrupt number of the interrupt source you plan to use so that you can decide which NVIC registers (see page 141 of [2]) to be configured. You can find the interrupt number of an interrupt source from the table in page 104 of [2].

Under the given restrictions, you will read the ADC using SysTick interrupt handler and drive the stepper motor using a GPTM interrupt handler. Thus, you may do other repetitive tasks in an endless loop inside main. An example algorithm for the main loop is given below.

1. Wait for SysTick handler to store 512 readings in the array
2. Call the FFT subroutine
3. Compute the complex magnitudes of elements of the frequency domain sequence
4. Find index and magnitude of the dominant (highest-magnitude) frequency component
5. Convert that index to frequency
6. If that magnitude is larger than a threshold, turn on the corresponding LED and adjust the intensity based on amplitude. Also update the stepper motor speed
7. If more than a second has elapsed after the last LCD update, show the frequency and magnitude values on LCD
8. Loop back to 1.

Note: The ADC module in TM4C123 has 12 bits resolution, and the reading is stored in the lower 12 bits of the FIFO register. In order to not lose precision in the FFT function, you need to shift the reading to higher 12 bits of a 16 bit variable.

Note: The microphone module has 1.25V DC offset.

Note: FFT function accepts complex-valued input sequence, see Section 4.2. Thus, for a single data input, you need to store the 16 bit ADC data followed by a 16 bit zero.

4 Background Information: ARM CMSIS DSP Library

The **Common Microcontroller Software Interface Standard (CMSIS)** is a vendor-independent abstraction layer for microcontrollers that are based on Arm Cortex processors [3]. In this term project, you will use the Fast Fourier Transform (FFT) function, `arm_cfft_q15`, in CMSIS DSP library [4].

4.1 The Fast Fourier Transform

The discrete Fourier transform (DFT) is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N}, \qquad k = 0, 1, \cdots, N-1$$

The Fast Fourier Transform (FFT) is an algorithm that computes DFT of a sequence in $O(N \log N)$ time. That is, the transform takes an N-element, possibly complex-valued, sequence in time domain and transforms it into the corresponding N-element sequence in frequency domain.

Assume that the time domain signal is sampled at frequency F_s. Assume also that the time domain signal is real valued. Then, the frequency domain signal can be interpreted as follows:

$$\begin{aligned}
&X_0 = \text{DC level of input,} \\
&X_i = \text{Frequency component corresponding to } F_s \cdot \frac{i}{N}, \qquad 1 \le i \le \frac{N}{2} \\
&X_i = X^*_{N-i}, \quad \text{complex conjugate of } X_{N-i}, \qquad \frac{N}{2} + 1 \le i \le N-1
\end{aligned}$$

4.2 Input and Output Format

`arm_cfft_q15` function takes the input and gives the output as complex numbers. Real and imaginary parts of input is in q15 and those of output is in q9.7 fixed point formats. In this project, you may assume that both q15 and q9.7 formats are the same as 16-bit signed 2's complement numbers. That is, $0x7FFF$ is the maximum value and $0x8000$ is the minimum value.

A single complex number in this format takes up 4 bytes. The first 2 bytes are the real part, and the last 2 bytes are the imaginary part. For example, the number $1024 - j128$ is stored in the memory as 00 04 80 FF (note that $1024 = 0x0400$ and $-128 = 0xFF80$).

As the input, you are going to give an array of 512 complex numbers. Your input sequence should be real valued. You will store the 16 bit ADC readings in the real parts of the complex numbers, and the imaginary parts should be zero. `arm_cfft_q15` function will operate on the array in-place. That is, when the function returns, output data will be placed on the same memory location as the input data and input data will be lost.

4.3 Using CMSIS FFT Implementation

To be able to use CMSIS DSP functions, you need to add the source files to your project. When creating the project, in the Manage Run-Time Environment screen, expand the CMSIS folder and check CORE and DSP boxes (Fig. 2).

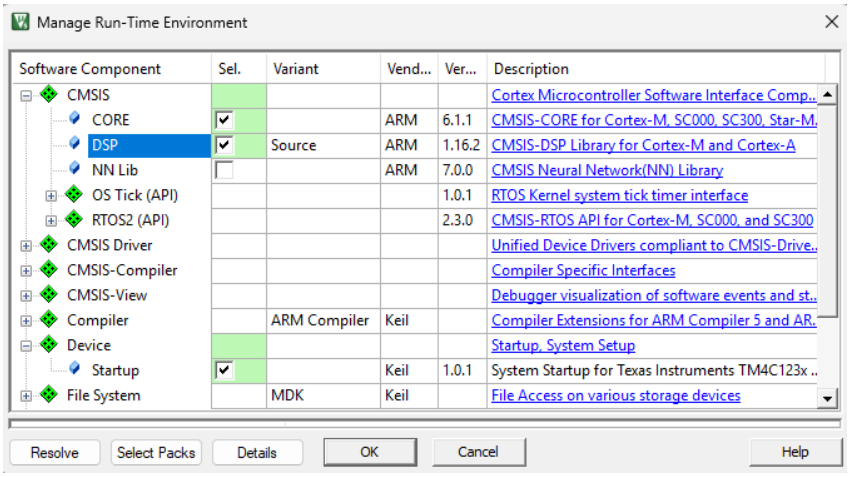

Figure 2: Adding CMSIS DSP library to the project.

This process adds several DSP functions to the project. In total, these functions take up about 900 KB program memory, which cannot fit in our microcontroller. In order to not include unused functions, you may need to turn on C/C++ optimization. From Project > Options for Target screen, under the C/C++ tab, set optimization to Level 1 (Fig. 3).

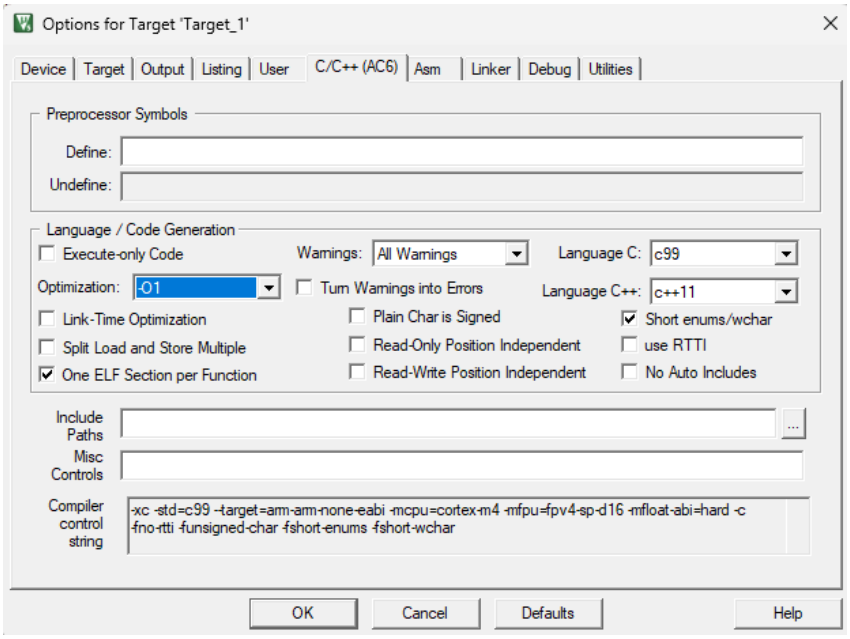

Figure 3: Setting C/C++ optimization.

The function uses a constant table, `arm_cfft_sR_q15_len512`, which is stored in the CMSIS DSP library. In your C code, you need to include `"arm_math.h"` and `"arm_const_structs.h"` files to access the function and the table. The function takes 4 input arguments [4]. First argument is the table's address, second is your array's address, third is 0, and fourth is 1. For an example code, see Fig. 4

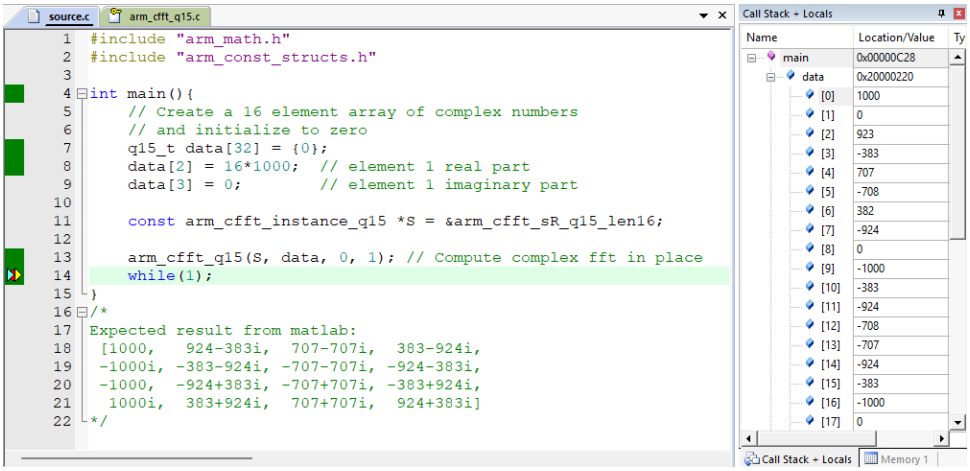

Figure 4: Calling the FFT function.

5 Background Information: Serial Peripheral Interface

Serial transmission involves sending one bit at a time, such that the data is spread out over time. Compared to parallel communication, many fewer lines are required to transmit data. This requires fewer pins but adds complexity. Serialized data is not generally sent at a uniform rate through a channel. Instead, there is usually a burst of regularly spaced binary data bits followed by a pause, after which the data flow resumes. Packets of binary data are sent in this manner, possibly with variable-length pauses between packets, until the message has been fully transmitted.

In synchronous systems, separate channels are used to transmit data and timing information. The timing channel transmits clock pulses to the receiver. Upon receipt of a clock pulse, the receiver reads the data channel and latches the bit value found on the channel at that moment.

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems and in this project it will be used. SPI involves a master and a slave, or multiple slaves. SPI interfacing involves 3 or more wires, consisting of a clock, serial data out, serial data in, and chip select if necessary. The master MCU basically sets the clock rate for the slaves, asks a specific one to listen up using the chip select port, and sends them commands via its serial data out port, and expects to receive the output from the slave through the serial data in port.

5.1 Synchronous Serial Interface in the TM4C

The TM4C has four Synchronous Serial Interface modules (SSI). The SSI is used to send synchronous serial communication to other devices, and can be configured to follow various protocols. We will use SPI in this lab, which requires that we specify which device will be sending data (Master), and which device(s) will be receiving data (Slave). When sending, the data is sent loaded into a FIFO buffer, and sent out according to the configured bit rate. Each FIFO buffer is 16 bits wide, and 8 locations deep. The size of the data can be configured to be from 4 to 16 bits wide depending on your needs.

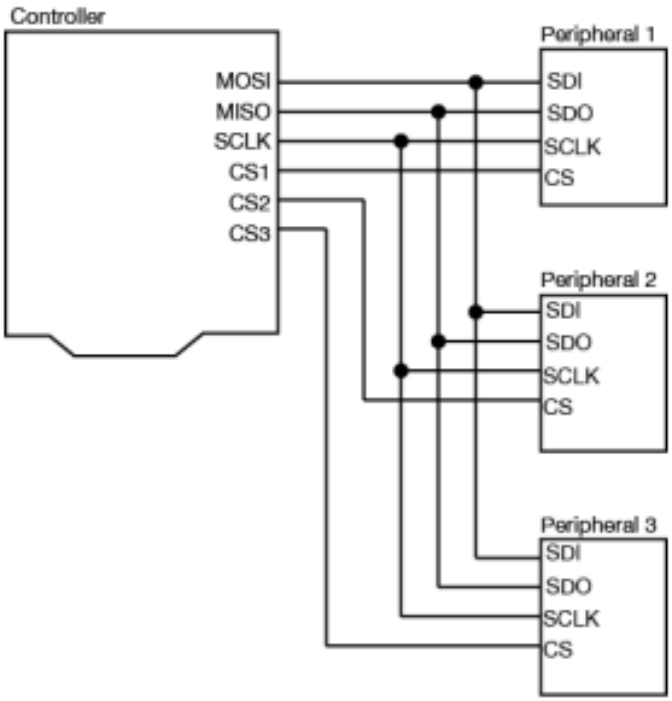

Figure 5: Synchronous Serial Interface in the TM4C

The pin connections for all SSI ports are labeled in Figure 5. When using Synchronous Serial Commu-

nication, there are typically 4 pins (lines).

- RX (MOSI) – Receiving data line
- TX (MISO) – Sending (transmitting) data
- Clk (SCLK) – The clock each bit is synched with
- Fss – Used to tell the slave that data is being sent

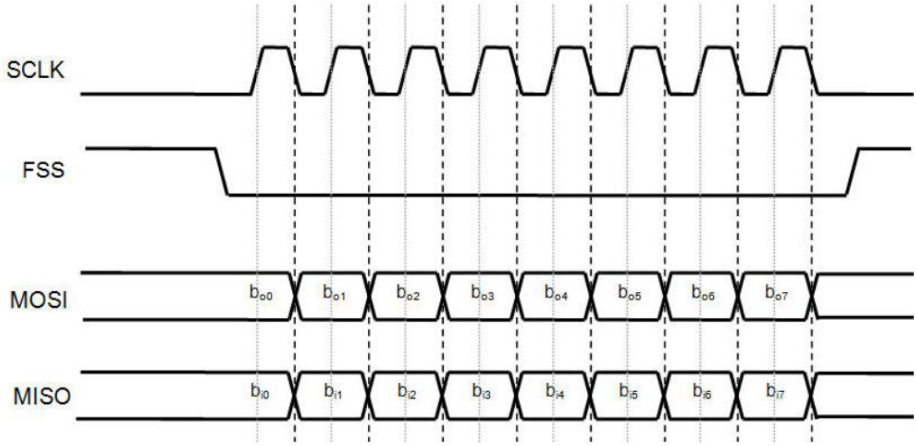

Figure 6: How SPI signals change as data is sent

5.2 SPI Configuration

In order to have Nokia 5110 functioning, 3 separate configurations are required, first of which is GPIO where the corresponding I/O pins are initialized to work as SPI pins. In the second part, the SPI module on the TM4C123 is configured to be compatible with LCD. Finally, in the NOKIA 5110 configuration part, the display is initialized for communication and to receive data to be displayed.

The Nokia 5110 uses SPI signals to receive commands, text, and images to be displayed. As to be noticed in Figure 7 on the SPI signals that there is only one data output signal. The LCD somehow, needs to distinguish whether the data being sent is data meant to be displayed, or if it is a command meant to control the screen. This is done not through the SPI module, but "manually" through a GPIO pin. The Nokia screen has a pin named Data/Command (DC). When the DC signal is low, the Nokia interprets the incoming SPI bits as a command. Similarly, if the DC signal is high, the SPI bits are interpreted as data to be displayed. The DC signal can be set high or low long before the last bit is sent.

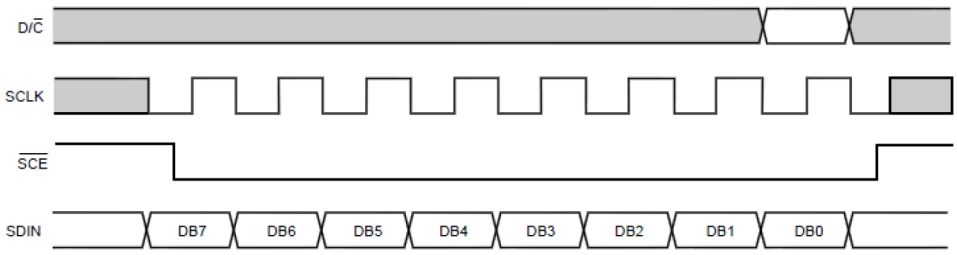

Figure 7: How the DC signal is timed with the SPI signals

For GPIO and SPI configuration parts, you may refer to page 965 of the TM4C123 Datasheet for details and 967 for register map.

GPIO Configuration:

1. Enable the clock for GPIOx **(RCGCGPIO)**
2. Wait until GPIOx is ready **(PRGPIO)**
3. Configure the CLK, CS (FSS), MOSI (Tx), and MISO(Rx) pins as a digital pin **(DEN)**
4. Set directions for the pins **(DIR)**
5. Configure the CLK, CS (FSS), MOSI (Tx), and MISO(Rx) pins for their alternate function **(AFSEL)**
6. Configure the CLK, CS (FSS), MOSI (Tx), and MISO(Rx) port control pins to route the SSI interface to the pins **(PCTL)**.

SPI Configuration:

1. Enable the clock for SSIx **(RCGCSSI)**
2. Wait for the SSI peripheral to be ready **(PRSSI)**
3. Disable the SPI interface **(CR1)**
4. Set the clock rate of the SPI Clock **(CPSR, CR0)** accordingly. Please mind the maximum data rate that the LCD is capable of working with.
5. Set the data size to be 8-bits and Freescale mode **(CR0)**
6. Set the SPI mode **(CR0)** accordingly.
7. Re-enable the SPI interface **(CR1)**

For Nokia 5110 LCD configuration you may refer to Nokia5110Datasheet.pdf provided along with the project manual, where a detailed version of the instruction set and data transmission graphs are provided for clarification.

NOKIA 5110 Configuration:

NOTE: Please do not try to connect and turn on the backlight of the display early on in development. It requires 5V which is different than the Vcc of 3.3V. Instead, make sure you can correctly drive the display without the backlight. Even then, the backlight is still optional, so you may not connect it at all if the content of the display is visible under normal conditions.

1. To initialize the Nokia screen first toggle the Reset pin by holding it low for 100ms then setting it high.
2. Send the following commands to initialize the display
   - Set H=1 for Extended Command Mode, V=0 for Horizontal Addressing
   - Set V_{OP}. You may need to sweep values between 0x[B0-C0] for correct operation.
   - Set temperature control value. You may need to sweep values between 0x[04-07] for correct operation.
   - Set voltage bias value as 0x13.
   - Set H=0 for Basic Command Mode
   - Configure for Normal Display Mode
   - Set Cursor to detemine the start address
3. Send data to be displayed.

INSTRUCTION	D/$\overline{C}$	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H = 0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	no operation
Function set	0	0	0	1	0	0	PD	V	H	power down control; entry mode; extended instruction set control (H)
Write data	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	writes data to display RAM
(H = 0)										
Reserved	0	0	0	0	0	0	1	X	X	do not use
Display control	0	0	0	0	0	1	D	0	E	sets display configuration
Reserved	0	0	0	0	1	X	X	X	X	do not use
Set Y address of RAM	0	0	1	0	0	0	Y_2	Y_1	Y_0	sets Y-address of RAM; $0 \le Y \le 5$
Set X address of RAM	0	1	X_6	X_5	X_4	X_3	X_2	X_1	X_0	sets X-address part of RAM; $0 \le X \le 83$
(H = 1)										
Reserved	0	0	0	0	0	0	0	0	1	do not use
	0	0	0	0	0	0	0	1	X	do not use
Temperature control	0	0	0	0	0	0	1	TC_1	TC_0	set Temperature Coefficient (TC_x)
Reserved	0	0	0	0	0	1	X	X	X	do not use
Bias system	0	0	0	0	1	0	BS_2	BS_1	BS_0	set Bias System (BS_x)
Reserved	0	0	1	X	X	X	X	X	X	do not use
Set V_{OP}	0	1	V_{OP6}	V_{OP5}	V_{OP4}	V_{OP3}	V_{OP2}	V_{OP1}	V_{OP0}	write V_{OP} to register

Figure 8: Instruction Format

It is possible to write data into the address of memory (DDRAM) of the Nokia LCD continuously and values of X-Address and Y-Address will be increased automatically. In this case, there are 2 methods to configure the operation format of address; firstly, Vertical Addressing Mode (V=1), 1 value of Y-Address will be increased every time; and secondly, Horizontal Addressing Mode (V=0), 1 value of X-Address will be increased every time. One may use either addressing as pleased; however it is to be noted that when the cursor is set to an arbitrary location, a very short delay, in the order of nsec is required for the cursor to settle.

6 Deliverables

You are supposed to attempt the project work in groups of two.

- **Preliminary Report:** A summary of the framework (including flowcharts and pseudo codes), at most 3 pages. **Deadline: 21.12.2025**
- **Final Report:** A full description of your work, including photos of your setup and relevant portions of code you may need to explain certain key points. Full source codes are not required, indeed discouraged, in your report. Clearly indicate how you fulfill the requirements and satisfy the restrictions. **Deadline: 04.01.2026**
- **Source Code:** A zipped Keil μvision project folder with fully executable codes is to be uploaded to ODTUClass. Your codes should be well-commented. **Deadline: 04.01.2026**
- **Lab Demo:** Demonstration of operation of the project. You will choose an available time slot on ODTUClass. **Date: 05.01.2026-07.01.2026**

References

[1] TI, "Tiva™ c series tm4c123g launchpad evaluation board user's guide." http://www.ti.com/lit/ug/spmu296/spmu296.pdf.

[2] TI, "Tiva™ tm4c123gh6pm microcontroller data sheet." http://www.ti.com/lit/ds/spms376e/spms376e.pdf.

[3] A. Developer, "Common microcontroller software interface standard (cmsis)." https://developer.arm.com/tools-and-software/embedded/cmsis.

[4] A. Developer, "Complex fft functions." https://arm-software.github.io/CMSIS_5/DSP/html/group__ComplexFFT.html.