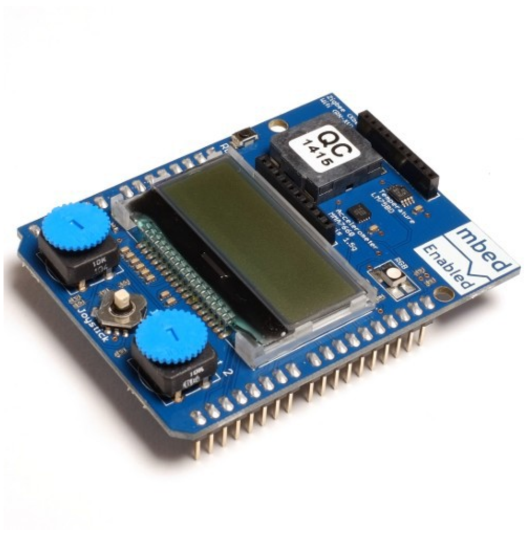


Hands-on experience

MBED LCD-Display with Accelerometer, Temperatursensor, RTC, ADC, SPI and I2C

Documentation

Studies:	Micro- and Medicaltechnology
Autor:	Matthias Renner
Client:	BFH TI Bienne, Mechatronik
Date:	08.10.2016/24.02.2016

Versionen

Version	Date	Status	Notes
1.1	10.10.2016	Release	Usage migration to V5 and packs
1.0	24.02.2016	Draft	Initial exercise documentation

Contents

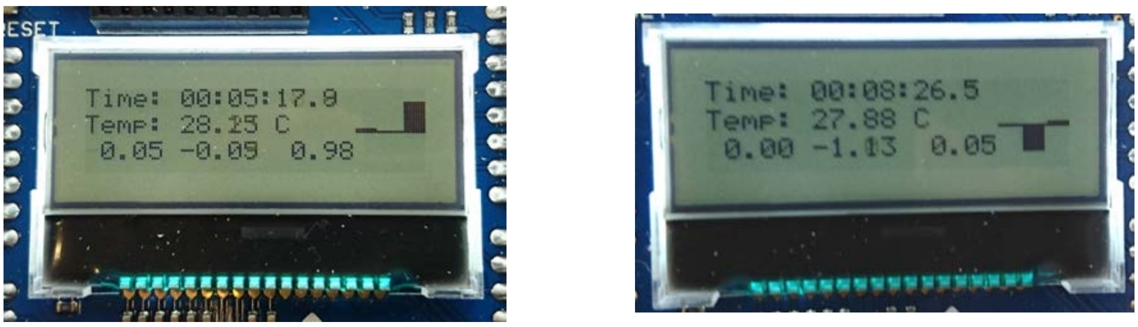
1. Introduction

For the project MecProj (Mechatronik 2016), one sub task is to control a tiny LCD display on the electronics cube. This display could, for example, work as a little alarm-clock.

For the following task you will be provided with a STM32F103 Nucleo board as well as an MBED application shield with a Graphic LCD display, a 3-Axis Accelerometer and a temperature sensor. The Newhaven Display of the type NHD-C12832A1Z-FSW-FBW-3V3 comes as closed unit with a ST7565R controller which is accessed by SPI. The display is a non-colour transflective graphic version with 128x32px. Both, the Freescale MMA7660 Accelerometer as well as the NXP LM75B temperature sensor are connected and accessed by the I2C bus.

In this laboratory you have to initialize the SPI, the I2C bus and the GPIO Pins to gain access to the display and the sensors. Furthermore you have to initialize the RTC and the A/D converter. To control the display and access the sensors, different libraries will be provided.

Your goal is to develop a similar display as shown in picture 1.1 below. It consists of the Time (RTC), the temperature and the three axis accelerations (displayed in numbers and as bars). These are the minimal requirements, feel free to implement further informations.



(a) standing flat on the table

(b) tilt 90 degree to the right

Figure 1.1.: Display showing time, temperature and acceleration

Further informations about the boards, the display and the sensors can be found in the appendix A on page 13.

2. Tasks

2.1. LCD

2.1.1. SPI

Before we can write (eg. text, graphics) to the LCD Display, we have to initialize the SPI bus. The STM32F103RB Nucleo board comes with an integrated SPI controller. The Standard Peripheral Library gives you all functions needed to initialize and communicate over SPI.

In our ×Vision project look for the file *configuration.c*. There we will find the functions void GPIO_configuration(void), void RCC_configuration(void) and void SPI_configuration(void). In this functions we will find the following comment lines: //$TASK SPI. Below these lines we have to fill in the needed commands.

Hint: There are several pins which are needed to communicate with the Display, have a look at the *mbed-016.1.pdf* in the *doc/pdf* folder and the image 2.1 below to determine which pins are needed.

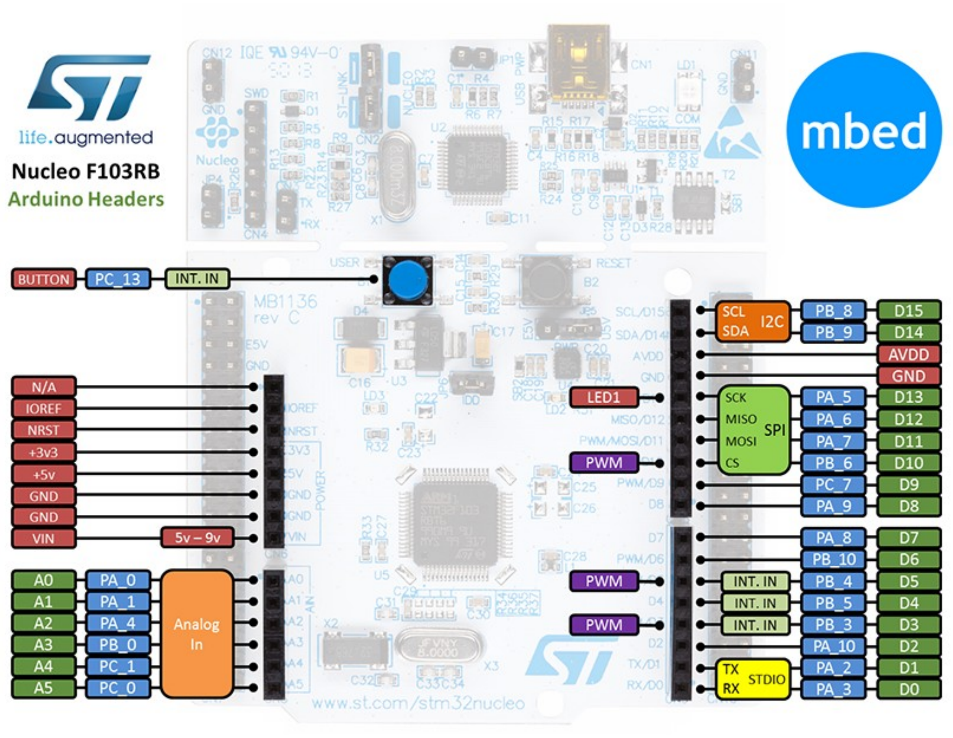


Figure 2.1.: STM32F103RB Nucleo Arduino Header

ToDo:

- enable the clocks (RCC) needed, be aware of the GPIO ports
- set up the GPIO pins needed for communication with the display, deselect display (SS)
- Deinitialise SPI
- configure SPI as:
  - only TX (RX not needed)
  - Master Mode
  - Data size 8 bit
  - Clock polarity low
  - Clock phase: 1 edge
  - Slave select (SS) by software
  - clock speed: max. 2.5MHz
  - First bit is MSB
  - CRC polynomial is 7
- Enable SPI
- test SPI communication, uncomment following line (around line 75) in the *main.c* file: //glcd_command(0xA5); while(1); If the display shows a black screen after reset, the SPI bus is working. This line is only for debugging, if you see the black screen comment out the mentioned line again!

2.1.2. glcd Library

As the SPI bus is working now, we can go further and write out to the display. Therefore the glcd library by Andy Gock (Homepage GitHub) is provided. The library has been slightly modified in order to work with our setup.

As the library is designed for different types of displays and display controllers we need to set some global defines in Keil µVision:

In Keil µVision go to *Project>Options for Target>C/C++* There we find the section *Preprocessor Symbols* and further down a textbox *Define*. Add the following defines into this textbox:

```
GLCD_LCD_WIDTH=128

GLCD_LCD_HEIGHT=32

GLCD_INIT_NHD_C12832A1Z_FSW_FBW_3V3
```

But because every micro controller board has different pins in use, we need to complete the following functions in file *display.c*: void glcd_spi_write(uint8_t c) and void glcd_reset(void). In this functions we will find the following comment lines: //$TASK glcd. Below these lines we have to fill in the needed commands.

ToDo:

- Write the glcd_spi_write routine:
    - Select LCD
    - Send data over SPI (given)
    - Deselect LCD
- Write the glcd_reset routine:
    - Select LCD
    - LCD reset low
    - 1ms delay (given)
    - LCD reset high
    - Deselect LCD
- Test your routines by writing out *hello world* to the display (add your code after line 79 in file *main.c*). Use the glcd library, examples can be found in *unit test.c*:
    - set tiny font
    - clear buffer
    - draw a string with *hello world*
    - write out to display

2.1.3. RTC

RTC is short for Real Time Clock. The RTC can be used as an alarm clock, calendar or as periodic wakeup unit. As we want to show a time clock on our display we can make use of the integrated RTC of the STM32F103RB. The RTC can be driven by three clock sources LSE, LSI or HSE (see *doc/pdf/stm32F1xxx_RM.pdf* chapter 8.2 for further information). As we do not have soldered an 32.768kHz low speed external crystal (LSE) yet, we will use the LSI clock source. Disadvantage of the LSI is its imprecision. If we want to have a higher precision we have to solder en external crystal. Normally the prescaler divides the LSI (or LSE) to get exactly 1Hz (important for calendar function). But as we want do show 1/10 seconds we need a 10Hz clock.

In our ×Vision project look for the file *configuration.c*. There we will find the functions void RCC_configuration(void), void void NVIC_configuration(void) and void RTC_configuration(void). In this functions we will find the following comment lines: //$TASK RTC. Below these lines we have to fill in the needed commands.

ToDo:

- Enable the clocks (RCC) needed
- Enable the RTC interrupts (NVIC)
- Configure RTC with LSI clock on 10Hz:
  - Allow access to BKP Domain
  - Reset Backup Domain (deinitialise)
  - Wait till RCC_FLAG_LSIRDY is equal RESET
  - Select LSI as RTC Clock Source
  - Enable RTC Clock
  - Wait for RTC registers synchronization
  - Wait until last write operation on RTC registers has finished
  - Enable the RTC Second interrupt
  - Wait until last write operation on RTC registers has finished
  - Set RTC prescaler: set RTC period to 1/10 sec
  - Wait until last write operation on RTC registers has finished

Now our RTC should generate a interrupt every 1/10s which calls the function void RTC_IRQHandler(void) in file *stm32f10x_it.c* and sets the external variable TimeDisplay = 1. In the *main.c* file this leads to calling the function void Display(RTC_GetCounter(),readTemperatur(), ax, ay, az); (around line 100). Have a look at this function in file *display.c*. Within this function write the code sequence to display the time.

ToDo:

- Reset the RTC counter when Time is 23:59:59
- Compute hours, minutes, seconds, and 1/10 of seconds
- Write the above values to the display

Our display should now show the time with updates every 1/10 second.

Hint: Keep in mind, the RTC keeps working during reset and programming. To reset the RTC (eg. when prescaler was changed) cut off the power connection. The RTC starts counting from zero, feel free to implement an option to set the RTC to the current time.

2.1.4. ADC

The contrast of the display is adjustable. The function to do so is given by the glcd library. To adjust the contrast we will make use of the potentiometer 1 soldered on the MBED extension board. Therefore we have to initialise the A/D-Converter.

In our µVision project look for the file *configuration.c*. There we will find the functions
void RCC_configuration(void), void GPIO_configuration(void) and
void ADC_configuration(void). In this functions we will find the following comment lines: //$TASK ADC. Below these lines we have to fill in the needed commands.

Hint: Have a look at the *mbed-016.1.pdf* in the *doc/pdf* folder and the image 2.1 to determine which pin is needed to read the potentiometer 1.

ToDo:

- Enable the clocks (RCC) needed, be aware of the GPIO ports
- Set up the GPIO pins needed for reading the potentiometer 1
- Set ADC clock to max. 12 MHz
- Deinitialise ADC
- Configure ADC as:
  - independent
  - do not scan all channels
  - continuous conversion
  - do not trigger conversion by external event
  - right align data
  - set number of channels
- Enable ADC
- Reset calibration and wait till done
- Start new calibration and wait till done
- Set ADC channel and sample time
- Start continuous conversion
- As the ADC is now initialised, got to the *main.c* file (around line 92), set the LCD contrast by reading the ADC value

Hint: Have a look at the glcd library functions input value range for setting contrast, process the ADC value accordingly before forwarding it to the set-contrast function.

We now should be able to set the display contrast by turning potentiometer 1 on the MBED extension board.

2.2. I2C

Besides the SPI bus which is used to control the display, there is an other bus I2C to be found on the STM32F103RB. This bus is used to communicate with temperature sensor LM75B and accelerometer MMA7660. As well as the SPI bus, the I2C bus has to be initialised.

In our µVision project look for the file *configuration.c*. There we will find the functions void RCC_configuration(void), void void GPIO_configuration(void) and void I2C_configuration(void). In this functions we will find the following comment lines: //$TASK I2C. Below these lines we have to fill in the needed commands.

Hint: Be aware that the pins for I2C can be remapped to other than the standard pins. Check standard peripheral library for remapping functions. Be also aware that remapping pins need a specific clock source to be enabled. Have a look at the *mbed-016.1.pdf* in the *doc/pdf* folder and the image 2.1 to determine which pins are needed to communicate over I2C bus.

ToDo:

- Enable the clocks (RCC) needed, be aware of the GPIO ports
- Set up the GPIO pins needed for I2C bus
- Configure I2C as:
  - I2C Mode
  - Dutycycle low/high = 2
  - Set own slave address to 0x3c
  - Enable acknowledge
  - Set ack address to 7bit
  - set clock speed to fast mode

Our I2C bus should now be initialised.

2.2.1. Temperature sensor

As the I2C bus is working now, we can go further and read some temperature values from the LM75B sensor. As we know, every I2C bus participant has its own unique address. Compare the LM75B datasheet (*doc/pdf/LM75B.pdf*) with schematics of the MBED extension board (*doc/pdf/mbed-016.1.pdf*) to determine the address and register of the sensor (be aware of the address bitshift!). Open file *lm75b.c* and look for the line /* Private define. Change the values of the defines accordingly (default set to 0x00).

Hint: For I2C write/read commands have a look at the I2C library (*i2c.c*)

Got to the function float readTemperatur(void) and write the sensor reading sequence. In the mentioned functions we will find following comment lines: //$TASK LM75B. Below these lines we have to fill in the commands needed.

ToDo:

- Over I2C: write which register to read, read the two bytes returned
- Manipulate the two returned bytes according to the information the LM75B datasheet gives (only 11 useful bits, two's complement).
- return the correct value (float)

In file *display.c* go to function void Display(uint32_t TimeVar, float TempVar, float axVar, float ayVar, float azVar). Write the temperature value TempVar on the second line of the display.

2.2.2. Accelerometer

First determine the I2C slave address as well as the different register of the MMA7660 accelerometer. We will find the information in the MMA7660 datasheet (*doc/pdf/MMA7660FC.pdf*). Open file *mma7660.c* and look for the line /* Private define. Change the values of the defines accordingly (default set to 0x00).

In the same file go to function void MMA7660setMode(uint8_t mode) and void MMA7660setSampleRate(uint8_t rate). Add appropriate commands to set mode and sample rate.

In the mentioned functions we will find following comment lines: //$TASK MMA7660. Below these lines we have to fill in the commands needed.

Go to function void MMA7660getAcceleration(float *x, float *y, float *z) and write the sensor reading sequence (x, y and z): **ToDo:**

- Write the x acceleration register, read 3 bytes (x,y and z) to the buffer. Repeat as long as one of the three accelerations has its alert flag set (check datasheet for bit definitions).
- Calculate acceleration (be aware of the two's complement and the sensitivity)
- No return needed as x, y and z are given as pointer

Hint: x, y and z are flot variables given as pointer.

In file *main.c* call the function which does the acceleration reading (around line 96).

In file *display.c* got to function void Display(uint32_t TimeVar, float TempVar, float axVar, float ayVar, float azVar). Within this function write the code sequence to display the x, y and z-acceleration:

ToDo:

- Write the values as decimal to the display
- Draw a bar graph as shown in image 1.1 on page 1. If no acceleration occurs, the display shows only a vertically centred line (next to the clock, the temperature and the decimal acceleration). Negative acceleration is shown as a bar from the centre down, positive acceleration is shown as a bar in to up-direction. Have a look at the glcd library (*unit test.c*, *graphs.c* and *graphics.c*) for appropriate functions.

Our display should now look like in image 1.1 on page 1. Feel free to add more elements on the display or add more functionality.

List of Figures

A. Datasheets and information links

A.1. Datasheets

Link: Nucleo STM32F103RB User Manual
Link: STM32F103RB Datasheet
Link: STM32F103RB Reference Manual
Link: Newhaven C12332A1Z
Link: Freescale MMA7660
Link: NXP LM5B

A.2. Additional Info

Link: General information about the MBED application shield
Link: Schematic MBED application shield
Link: STM32F103RB Schematics (ZIP)