CEG 3136

COMPUTER ARCHITECTURE

uOttawa

Author: D! (pgaur@uottawa.ca)

Table of Contents

List of figures

List of Tables

Credits and error reporting information

This course and its lab components are the direct result of multiple teams working together, here at uOttawa. The need to implement a new course arose due to current technological advancements in microcontroller architecture as well as the need for academia to catch up with the industry demands. It is of primary interest that you, as a student, will gain some experience designing embedded systems using what's being used extensively in various industries.

We hope that you will make extensive use of this lab/course and will be well prepared to design your own embedded systems for industry.

Following people were involved making this lab/course possible:

Name	Role	Contact information
Dr. Miodrag Bolic	Head professor	Miodrag.Bolic@uottawa.ca
Mr. Pierre Adam	Project manager/Lab Manager	padam@uottawa.ca
D! (Puranjan Gaur)	Course and Lab designer Author of this document	pgaur@uottawa.ca
Simon Trembley	Manufacturing base boards	strem078@uottawa.ca
Jacques Sincennes	Software/IT support	jack@uottawa.ca

Table 1. Contact Information

If you find any errors in this document, please reach out to D! (pgaur@uottawa.ca).

Lab introduction

Welcome! In this lab/course you will learn how you can design, implement, test, debug, and troubleshoot embedded systems. The complexity of embedded systems that you can design and implement using this lab/course manual depends on your level of creativity. Appendix B lists some project ideas that you can implement by going through this course.

First, there are two parts to this course: *Theory* and *Labs*. **Theory** is taught in classroom and will focus on the architectural side of embedded systems. This means that theory is more general discussion revolving around architecture of a microcontroller system. We decided to teach you the architecture of ARM microcontrollers because of their wide use in the industrial systems. **Labs** are specific to a particular family of microcontrollers. This lab uses ***STM32L552ZET6QU*** microcontroller, which is based on ***ARM Cortex M33*** microcontroller architecture, and which is implemented by STM32 in their ***STM32 Nucleo L552ZE-Q*** microcontroller.

Second, while it is up to your professor to decide what labs you will be performing, we have created this document to help you out during your journey of understanding how to design and implement embedded systems. We want to make sure that no matter what section you are taking (French or English), you get the same learning experience. Therefore, we have created this document that can be directly accessed by ***scanning the QR code*** in your lab board. It is up to you to follow this lab guide and reach out to us if you have any questions.

Third, due to the lab time restrictions and the limited availability of lab spaces after lab hours, we had a challenge of making this course accessible to students and keeping them engaged throughout this course. Another challenge was to prevent the use of AI tools to plagiarize the labs and deter students from breaching academic

integrity policies set out by the university. There are very few things we can do for lab schedule, and there is nothing we can do about you using AI tools to cheat.

Fourth, we have tried to make it simpler for you to perform your labs within a given amount of lab time — and without compromising the quality of education that we aim to deliver. You can refer to ***Appendix B for example projects***. These projects are designed to help you immensely when you are developing your own custom embedded system; no matter what sensor, which communication protocol, which programming style, which peripheral you intend to use.

Finally, don't forget to try out new ideas. ***You will learn more by designing actual systems than just reading about them***.

Enjoy the process of creating something of your own.

-D!

Hardware description

System overview

The ***System*** for this course/lab consists of following components:

i. Lab hardware platform (it has a name: “Leafy”)

ii. Programming toolchain (STM32CubeIDE or Keil IDE)

The image below highlights all the subsystems within the lab hardware. **Click on individual block** to learn more about that subsystem. The system architecture is explained in the next section along with listing datasheets of all individual subsystems.

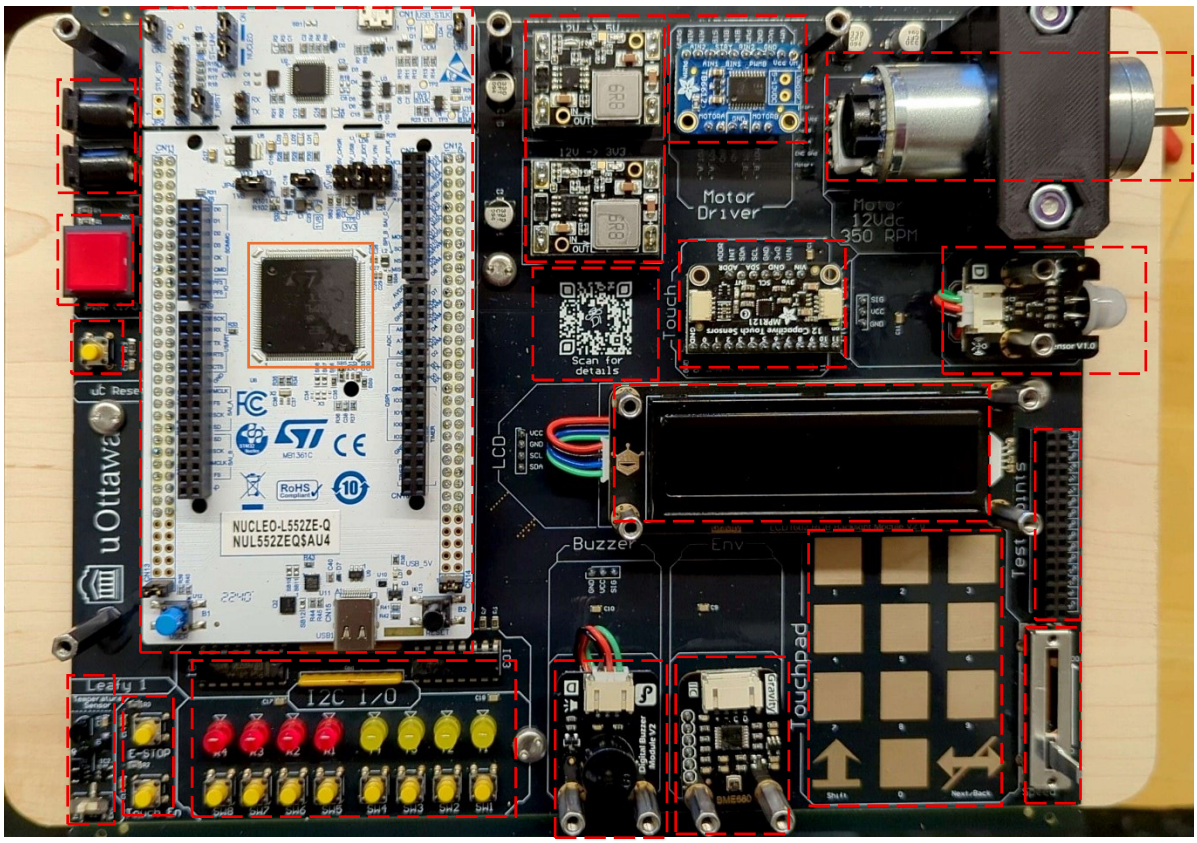


Figure 1. Lab Hardware Platform

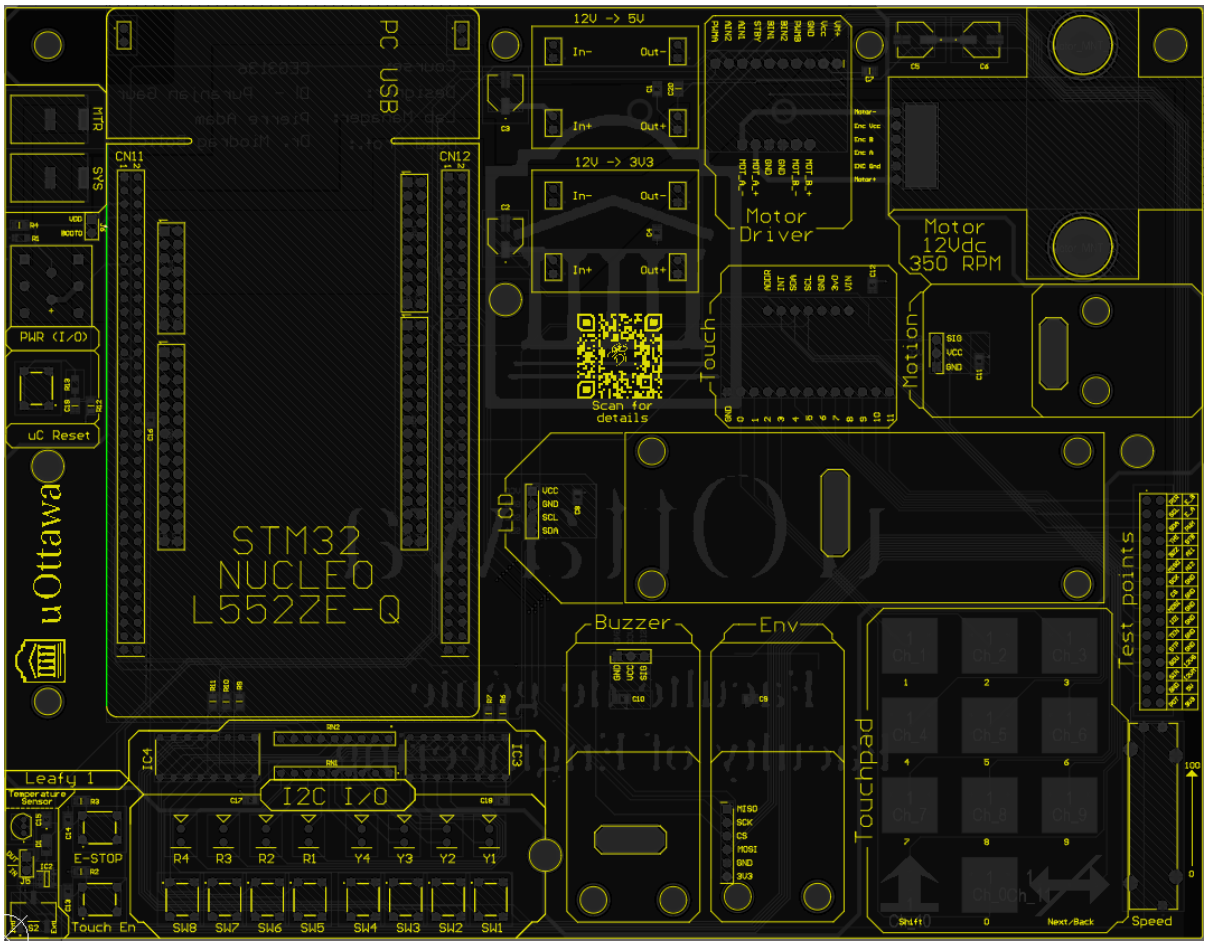


Figure 2. Lab Hardware Platform Outline

The Lab hardware is designed as a ***Platform board***. It provides provisions to connect multiple sensors, inputs and outputs of the system. Additionally, the lab hardware can host multiple types of sensors using wide range of communication protocols such as I2C, SPI or CAN. The lab hardware is also not dependent on a specific STM32 platform (for example: STM32 Nucleo L552ZE-Q). If you wish, you can interface external boards through the GPIO header (labeled 'Test points' on the board) and communicate with onboard sensors and IOs. ***Appendix A*** shows the block diagram of ***STM32L552ZET6QU*** microcontroller in more detail.

This system implements various technologies such as different types of sensing technologies, interfacing subsystems using different communication protocols,

different types of I/O technologies and interfacing subsystems to control power electronics. Each sensor employed in this system uses a completely different technology to sense the environment variable; therefore, by doing the labs in this course, you will be exposed to different types of sensors, interfacing techniques, programming approaches, debugging methods and troubleshooting skills.

Besides hosting all the modules (sensors, I/O devices, actuators, etc.), the lab hardware also acts as ***Power rail*** and ***Signal Bus***, thus providing all the electrical connections for powering the modules, all the provisions to minimize the system integrity issues, and all the communication or signal path between microcontroller and modules.

The board is designed to be fully modular and vendor agnostic. This will allow the course instructor/professors to switch to newer technology if they wish.

The section titled Power distribution provides details of how energy is distributed in the system. The section titled System architecture provides additional details on how different subsystems are connected with the microcontroller. Chapter 3 covers the programming toolchain that can be used for this lab hardware platform.

System architecture

Block diagram

The lab hardware employs ***STM32L552ZET6QU*** as its main microcontroller. This is the microcontroller that will be programmed by the user/student. The access to this microcontroller, and the required hardware to power and operate this microcontroller is provided by *STM Nucleo L552ZE-Q* development board. This is the board that can be used to debug any user application.

The block diagram of the overall system is shown below. Click here to view the high-quality version of the block diagram.

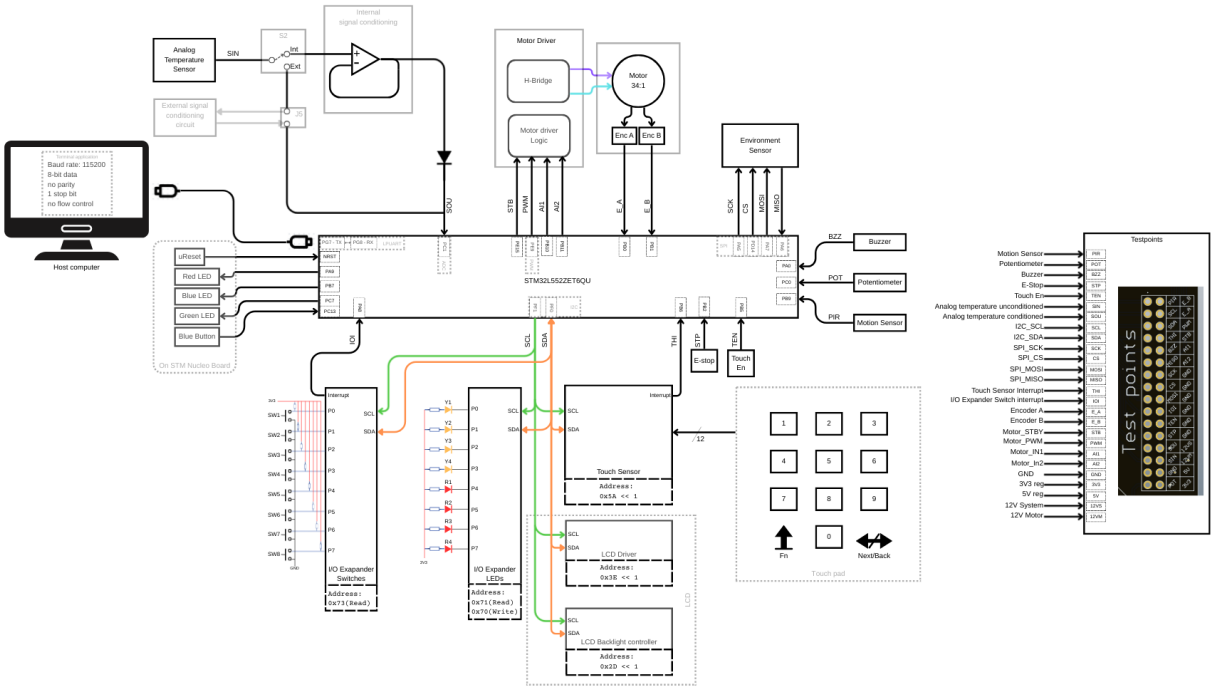


Figure 3. Block diagram

The development board is hardwired to different sensors, I/O devices and drivers. However, this hardwired configuration can be changed by the users by using the

Test-Points headers. For example, one of the push-button is connected directly to pin *PB2* of the development board (labeled *E-Stop*), but access to pin *PB2* is also given in *Test-Points* header; hence the student can interface any other type of input or output device by providing an external connection to the required I/O. Furthermore, if *PB2* pin on the microcontroller supports any other alternate functions (such as ADC, DMA, I2C, etc.), the user can program it to be used for such purposes as well.

The output/input of most subsystems can be probed using oscilloscope. Additionally, external signal conditioning can be performed by extracting out the raw sensor output and passing it through external signal conditioning block. The output of this external signal conditioning block can be interfaced back with the microcontroller. This will significantly improve the precision of certain systems, such as analog temperature sensor.

The following table lists all major subsystems of the lab hardware platform. Click on the name of the subsystem to get more information.

TYPE	SUBSYSTEM
CONTROLLER	Nucleo L552ZEQ dev board
	STM32L552ZET6QU – datasheet
	Motor driver
SENSING	Analog temperature sensor
	Touch sensor
	Environmental sensor
	Motion sensor
I/O	RGB LED
	I/O Expander
	Touchpad
	Potentiometer
	LCD
ACTUATOR	Motor

Table 2. System components datasheet

Power distribution diagram

The power distribution diagram of the overall system is shown below. Please note that the system ground (common ground or chassis ground) is not shown in the diagram below to avoid cluttering the diagram. Also note that ***if you intend to interface the lab hardware with external system*** (such as external signal conditioner for temperature sensor, external I/O, sensor etc.), you must connect the ground of external circuit/modules with the ground of this system. If possible, use external power supply to power external modules.

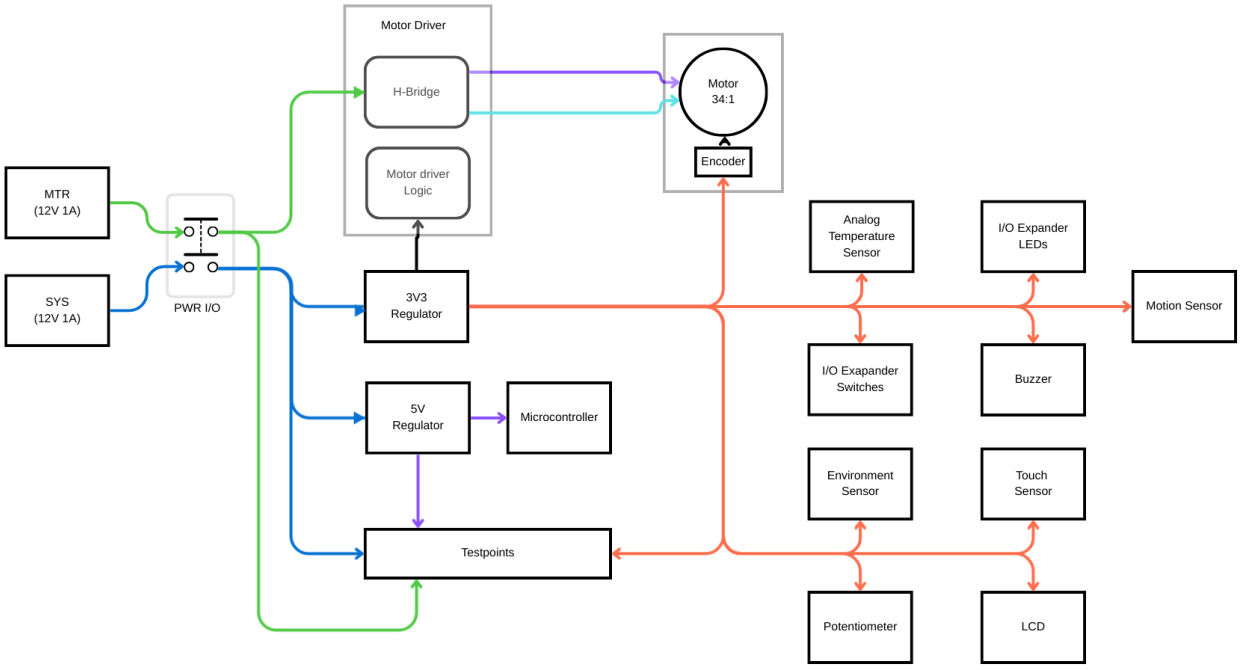


Figure 4. Power distribution diagram

Please see the section titled *System Power Supply Unit (PSU)* for complete description of how the energy and power is distributed and delivered inside the lab hardware.

System components

This section describes all the system components in detail. Users are encouraged to refer to the datasheet of the individual systems to gain better understanding of the inner workings of the systems and how to program/use/interface each subsystem. The datasheet of every subsystem is provided in *Appendix D* .

i. System power supply unit (PSU):

Main power connectors

Starting from top left, we see the two power jacks – which mate with the output of the power adapter in the lab. These power jacks supply electrical power to all the subsystems. The top power jack, labeled ***MTR***, is responsible for supplying electrical power to ***Motor only***. This is used to isolate the Motor from rest of the system; thus, making this system a truly independent low-voltage high-voltage system. This design choice was also made to mimic the actual controller interfacing with actuators found in almost every control application.

Figure 5. Power connectors

The bottom power jack, labeled ***SYS***, feeds the two **voltage regulators** (5V and 3.3V). Further details of voltage regulators are provided later in this section.

Power switch

Below the power jacks is Power Switch, labeled ***PWR (I/O)***.

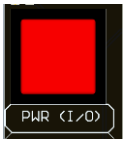


Figure 6. Power switch

This switch is the single on/off switch for the whole system. ***This switch controls the power from both jacks***. An additional feature of this switch is that it has an embedded fuse and an LED inside it, thus integrating the main safety features in one switch.

Voltage regulators

Right beside the microcontroller development board, you will notice two voltage regulator, as shown below.

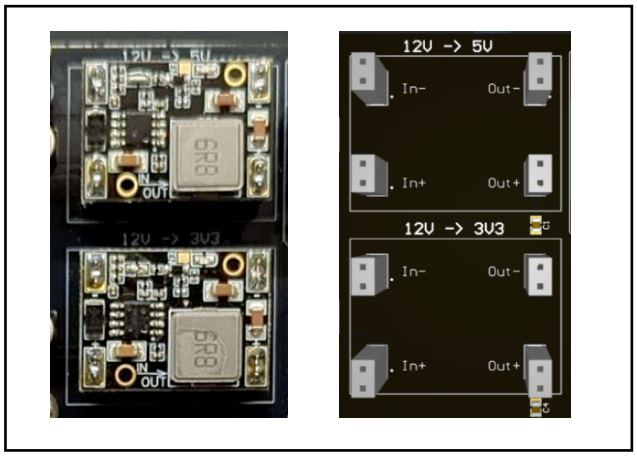


Figure 7. Voltage regulators

One of them is configured to supply a stable ***3.3V*** (*3V3*) to all the sensors and I/O modules. The other voltage regulator is configured to supply a stable ***5V*** to the microcontroller. When the system is powered on, the voltage regulators will turn on a blue light connected with the system – indicating that they are receiving power and are working properly.

The ***5V*** regulator is exclusive to the microcontroller, thus isolating the power rail noise for sensors and I/Os. This design choice was made to prevent any power rail noise or interference from any other subsystems. This was done to ensure that the system was robust and highly efficient.

If a user wishes to interface additional external sensors, I/Os, actuators, etc. with the lab hardware, they can access the output of these voltage regulator, as well as the output of the power jacks; thus, giving them 12V, 5V and 3.3V stable supply. These can be accessed via *Test-points* headers. However, the ***user must understand the loading effect of their circuits*** and hence must be careful to connect circuits that don't load these power supply units. If possible, try to avoid connecting 5V and 12V motor only connector with your external circuit.

Please see the section titled *Test points* in this document or click here to know where to get the stable output of several PSU of this lab hardware.

ii. Microcontroller:

Overview

The microcontroller that is connected with this lab platform is ***STM32L552ZET6QU*****.** The lab hardware platform is designed to interface with any microcontroller with appropriate number of GPIO and other peripherals. By default, the main development board used for this lab is ***Nucleo L552ZE-Q*** (shown below). Click on the image below to open the datasheet of the development board.

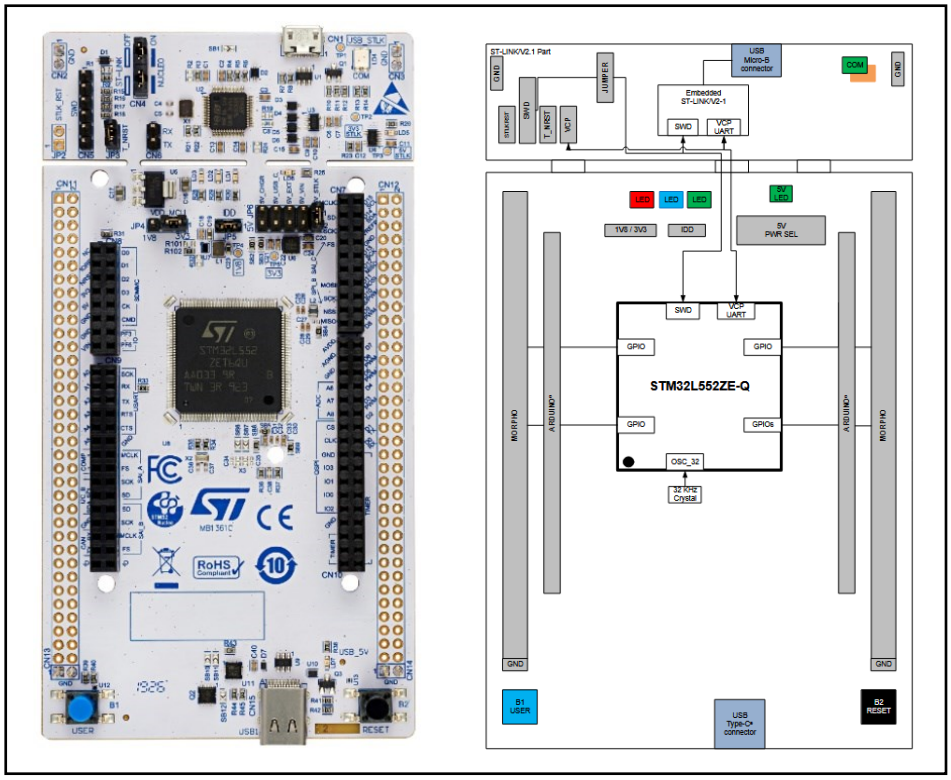


Figure 8. Development board (microcontroller)

It can be noticed that the development board has also has Red, Blue, and Green LEDs, as well as a user programmable momentary push button. A complete discription on these I/Os is provided in the next section titled *On board LEDs and Push buttons*.

Pin diagram

The picture below shows how different modules are connected to the microcontroller – or click on the image to get the high-quality version of the image. The pin table for the system, which includes a brief description of the pins, is also provided in this section and in Appendix A.

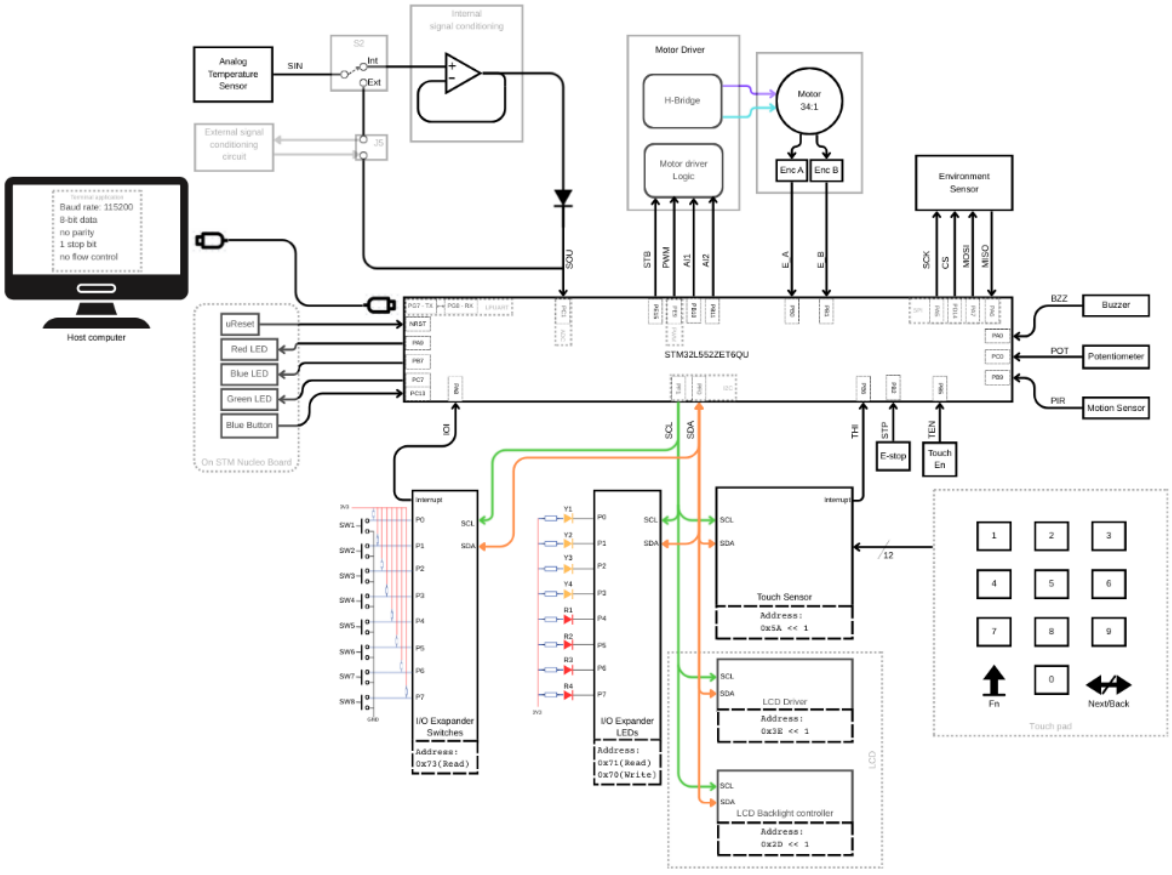


Figure 9. Pin diagram

Table 3. Pin description table

Pin name	Pin mode	Signal name	Signal description
NRST	RESET	NRST	A logic LOW on this pin will reset the microcontroller.
PA0	GPIO – Output TIM2_CH1 – PWM TIM5_CH1 – PWM	BZZ	This is the signal from microcontroller to **B**u**ZZ**er module. If configured in GPIO – Output mode, the Buzzer produces a constant beep is the Output is set to logic HIGH. If configured in PWM mode, the volume of the buzzer can be set by setting the DUTY CYCLE, and the frequency of the buzzer can be set by setting the FREQUENCY
PA5	SPI1_SCK	SCK	This signal is the **S**PI **C**loc**K** signal for all SPI modules in the system.
PA6	SPI1_MISO	MISO	This signal is **M**aster **I**n **S**lave **O**ut. This is the data line from Slave (SPI_subordinate) to Master (SPI_Controller)
PA7	SPI1_MOSI	MOSI	This signal is **M**aster **O**ut **S**lave **I**n. This is the data line from Master (SPI_Controller) to Slave (SPI_subordinate)
PA8	GPIO – External Interrupt	IOI	This signal is the **I/O I**nterrupt from I/O Expander module connected with the on board switches. This signal stays high if any of the switches of I/O expander (SW1 to SW8) is pressed. The signal goes low automatically when the microcontroller reads the switch state.

PA9	GPIO – Output TIM1_CH2 - PWM	Red_LED	This is the signal for on board RED LED light. The brightness can be adjusted if configured in PWM mode.
PB0	GPIO – External Interrupt	E_A	This is the signal coming from **E**ncoder **A** (attached to the motor). By knowing the relative location of Encoder_A with respect to Encoder_B, the direction of motor can be determined. By sensing the period of tick interval in this signal, the speed of motor can be determined.
PB1	GPIO – External Interrupt	E_B	This is the signal coming from **E**ncoder **B** (attached to the motor). By knowing the relative location of Encoder_B with respect to Encoder_A, the direction of motor can be determined. By sensing the period of tick interval in this signal, the speed of motor can be determined.
PB2	GPIO – Input GPIO – External Interrupt	STP	This is the raw signal from the E-**St**o**p** switch. Please note: This signal has not been debounced. Hence users must employ their switch debouncing logic in their software to use this signal properly.

PB5	GPIO - Input	TEN	This is the raw signal from the **T**ouch **En** switch. Please note: This signal has not been debounced. Hence users must employ their switch debouncing logic in their software to use this signal properly.
PB6	GPIO – External Interrupt	THI	This is the signal from **T**ouc**h** sensor **I**nterrupt pin. The interrupt line goes HIGH momentarily whenever a user touches or releases the any electrodes of the touchpad.
PB7	GPIO – Output TIM4_CH2 – PWM TIM17_CH1N - PWM	Blue_LED	This is the signal for onboard BLUE LED light. The brightness can be adjusted if configured in PWM mode.
PB9	GPIO - Input GPIO – External Interrupt	PIR	This signal comes from the Motion sensor (**PIR** sensor). A logic HIGH on this signal means motion is detected.
PB10	GPIO - Output	AI1	This is the Channel **A_1** control signal from microcontroller to the motor driver. This signal, along with AI2 and STB determines the behaviour of the motor driver controller.
PB11	GPIO - Output	AI2	This is the Channel **A_2** control signal from microcontroller to the motor driver. This signal, along with AI1 and STB determines the behaviour of the motor driver controller.

PC0	ADC	POT	This is the signal from Speed/Throttle/Potentiometr to the microcontroller. User must use ADC to detect the proper level. In addition to ADC, user may enable DMA to automatically capture and process the input signal levels.
PC1	ADC	SOU	This is the **S**ignal conditioner **OU**tput of analog temperature sensor to the microcontroller. The signal conditioner block can be internal or external, depending on the switch S2. User must use ADC to detect the proper level. In addition to ADC, user may enable DMA to automatically capture and process the input signal levels.
PC7	GPIO – Output TIM3_CH2 - PWM	Green_LED	This is the signal for on board GREEN LED light. The brightness can be adjusted if configured in PWM mode.
PC13	GPIO – Input GPIO - EXTI	Blue_Button	This is the signal from the User switch (Blue button). Please note: This signal ***has been debounced*** in hardware. However, users are encouraged to employ their switch debouncing logic in their software to use this signal properly.

PD14	GPIO - Output	CS	This signal is the **C**hip **S**elect signal for an individual SPI modules in the system. This signal is hardwired to select the Environmental Sensor on the lab hardware platform.
PE9	TIM1_CH1	PWM	This is the **P**ulse **w**idth **m**odulated signal from the microcontroller to the motor driver. This signal is used to control the speed of the motor. The direction of rotation and the status of the motor (Standby, running CW, running CCW, stopped) is selected by STB signal
PE15	GPIO - Output	STB	This is the **ST**and**B**y control signal from microcontroller to the motor driver. This signal, along with AI1 and STB determines the behaviour of the motor driver controller.
PF0	I2C_SDA	SDA	This is the bidirectional **S**erial **DA**ta signal line from I2C Master (microcontroller) to I2C Slaves (Touch sensor, I/O Expander, LCD character driver, LCD backlight driver).
PF1	I2C_SCL	SCL	This is a **S**erial **CL**ock signal from microcontroller to all I2C slaves. The frequency of the signal has to be compatible with all the devices using this signal. For the lab, the frequency is 100 kHz.

PG7	LPUART1_TX	TX	This signal is the Transmit (**TX**) pin from the LPUART module on Nucleo Board. This is used to enable communication between the Nucleo board and the host computer.
PG8	LPUART1_RX	RX	This signal is the Receive (**RX**) pin from the LPUART module on Nucleo Board. This is used to enable communication between the Nucleo board and the host computer.

Please click here to get the complete pin mapping table for the microcontroller (including the alternate functions).

External hardware connection diagram

Please see the block diagram of the system.

Programming and communicating with the microcontroller

Please see the section titled *Programming/Debugging and Communications* in this document.

Reset mechanism

The small yellow switch labeled ***uC reset***, just below the power switch, is coupled with ***NRST*** pin of microcontroller. This pin is used for resetting the microcontroller only.

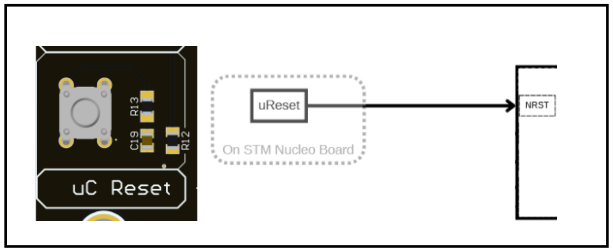


Figure 10. Microcontroller reset button

Pressing this switch does not power cycle the I/Os, or the sensors. To power cycle the board, please use the power switch. Access to this signal is not provided from the *Testpoint* header.

Pin name	Pin mode	Signal name	Signal description
NRST	RESET	NRST	A logic LOW on this pin will reset the microcontroller.

Table 4. Pin table for Reset button

iii. System I/O

On board LEDs and Push buttons:

On the development board itself (Nucleo), there are three LEDs (Red, Blue, Green) and two buttons labeled ***User*** and ***Reset***. User button is hardwired to pin *PC13* on the microcontroller. Reset is hardwired with NRST pin on the microcontroller; thus acting as ***uC Reset*** button which was previously discussed.

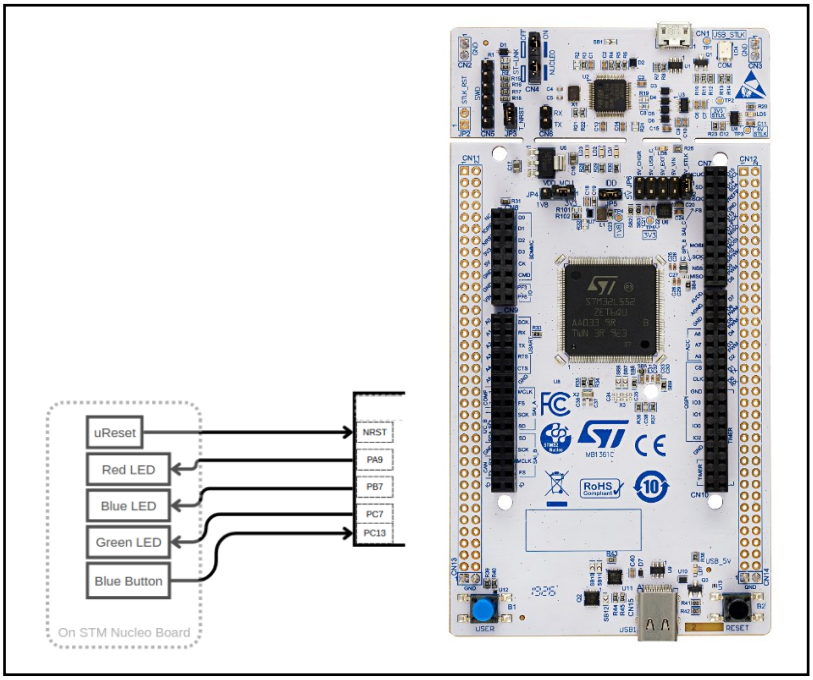


Figure 11. On board LEDs and Buttons - 1

Similar to the button 'User' (blue button on development board), there are two additional buttons that are directly coupled with the microcontroller: ***E-Stop*** and ***Touch En***.

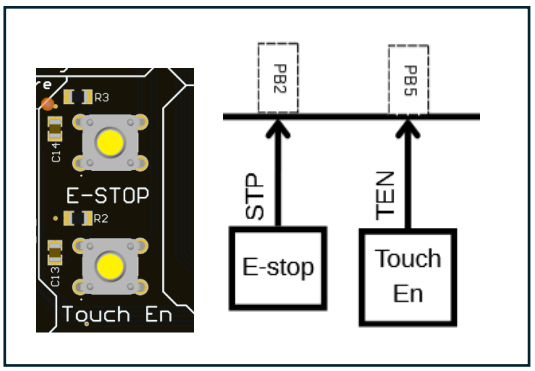


Figure 12. On board Buttons - 2

E-Stop is connected to pin *PB2*, and Touch En is connected to *PB5*. Please note that the difference between these buttons labeled *USER* is that these buttons are directly coupled with the microcontroller, therefore ***these buttons are NOT DEBOUNCED***! The *USER* button is actually debounced in hardware, however, if a user presses the user button too often, the signal may still not be debounced. Therefore, users are encouraged (and it is good practice in general) to make sure that their application software implements a switch debouncing code.

Additionally, these buttons are given descriptive names so that the user can assign these buttons for specific reason: To stop the system in emergency, or to enable the touch keyboard. ***Please note that the user's application will dictate the correct usage of these buttons***, however, to remind the user that every system needs an emergency stop function, as well as the performance of every environment sensitive sensing application (such as capacitive touch sensor) is significantly improved if employed with an added interlocking feature (In state machine design, these types of interlocking features are called ***Guard conditions***). For example, to prevent the false triggering of the touch sensor the user can program in his/her application that the interrupt generated by touch sensor should only be serviced if the ***Touch En*** button was pressed.

Both programming approaches – Polling and Interrupts – can be implemented using these buttons. Implementing Interrupts is complicated, but it is recommended over polling for some obvious reasons that you will study in your classes. Please note that in order to efficiently program your embedded system, ***use the Flags in your ISR*** and then take the action in your main loop. Don't take any action in your ISR.

Pin name	Pin mode	Signal name	Signal description
NRST	RESET	NRST	A logic LOW on this pin will reset the microcontroller.
PA9	GPIO – Output TIM1_CH2 - PWM	Red_LED	This is the signal for on board RED LED light. The brightness can be adjusted if configured in PWM mode.
PB7	GPIO – Output TIM4_CH2 – PWM TIM17_CH1N - PWM	Blue_LED	This is the signal for onboard BLUE LED light. The brightness can be adjusted if configured in PWM mode.
PC7	GPIO – Output TIM3_CH2 - PWM	Green_LED	This is the signal for on board GREEN LED light. The brightness can be adjusted if configured in PWM mode.
PC13	GPIO – Input GPIO - EXTI	Blue_Button	This is the signal from the User switch (Blue button). Please note: This signal ***has been debounced*** in hardware. However, users are encouraged to employ their switch debouncing logic in their software to use this signal properly.

Table 5. Pin table for on board LEDs and Buttons

I2C I/O Expander Switches and LEDs

The section right beside the *E-Stop* and *Touch_En* is labled ***I2C I/O***. In this section, you will notice that there are ***four red LEDs*** *(labeled R1, R2, R3, R4)*, ***four yellow LEDs*** *(labeled Y1, Y2, Y3, Y4)*, and ***eight yellow switches*** *(labeled SW1, SW2, …, SW8)*. The following diagram shows how I/O Expander ICs are connected to the microcontroller.

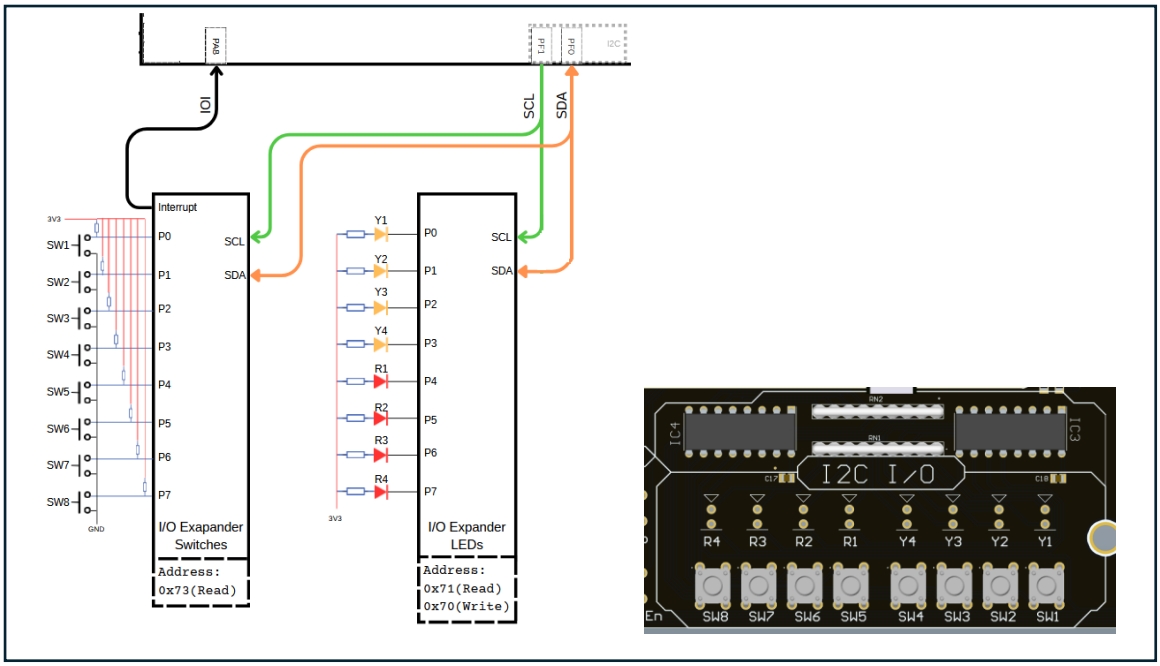


Figure 13. I/O Expander block diagram

As it can be noted from the diagram above, these LEDs and switches are not directly connected with microcontroller. Instead, an IC is used to access these IOs. These ICs are known as IO Expander, because these ICs only need two pins of microcontroller to control many IOs. These two pins are part of a communication protocol called I2C (or I^2C).

Basically, the microcontroller can command the IC to turn on a LED or turn of an LED by using I2C. Similarly, when a user presses a switch, these ICs can generate interrupt and let the application running on the microcontroller know that the

user pressed the switch. It is also possible for microcontrollers to sense the switch state without receiving the interrupt from the IC.

	INPUTS			I²C BUS responder 8-BIT READ ADDRESS	I²C BUS responder 8-BIT WRITE ADDRESS
	A2	A1	A0		
LEDs	L	L	L	113 (dec), 71 (hex)	112 (dec), 70 (hex)
Switches	L	L	H	115 (dec), 73 (hex)	

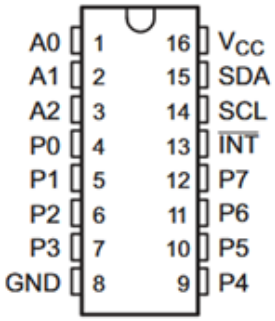


Table 6. I/O Expander I2C Address Table

The image above shows the I2C address details of I/O Expander. To use the LEDs and Switches, the user must write the drivers to access these IOs. In order to turn any *LED* ***ON*** or ***OFF***, a user must write the I/O Expander LED driver that sends command at address *0x70*. A user must set the port state (P0, P1, etc.) to logic LOW to turn ON the LED, and logic HIGH to turn OFF the LED. Therefore, the LEDs are connected as if there was an inverter between them and the microcontroller. In addition to turning any LED ON or OFF, you can also know the status of any LED (whether it is currently ON or OFF), the user's driver must send a read request to the I/O Expander IC that is connected to the LEDs. This IC will respond to the read request if the command is sent to address: *0x71*.

The I/O Expander that is connected to the switch works as follows: By default, the logic level on the switch I/O Expander is pulled up to logic HIGH. When a user presses any switch (S1, S2, …, S8), the appropriate port will have logic LOW. In addition to this, the IC will also send an interrupt request to the microcontroller on IOI signal line. When the microcontroller's ISR services that interrupt and reads the status of the switch I/O expander – by sending the read request on address 0x73 – the interrupt line goes back to low. Similarly, when the user

depresses the button, another interrupt is generated and is reset when the user code reads the status of the switch.

Pin name	Pin mode	Signal name	Signal description
PA8	GPIO – External Interrupt	IOI	This signal is the **I/O I**nterrupt from I/O Expander module connected with the on board switches. This signal stays high if any of the switches of I/O expander (SW1 to SW8) is pressed. The signal goes low automatically when the microcontroller reads the switch state.
PF0	I2C_SDA	SDA	This is the bidirectional **S**erial **DA**ta signal line from I2C Master (microcontroller) to I2C Slaves (Touch sensor, I/O Expander, LCD character driver, LCD backlight driver).
PF1	I2C_SCL	SCL	This is a **S**erial **CL**ock signal from microcontroller to all I2C slaves. The frequency of the signal has to be compatible with all the devices using this signal. For the lab, the frequency is 100 kHz.

Table 7. Pin table for I/O Expander

LCD with RGB backlight

Figure 14. LCD module

On the center of the lab hardware, you will notice a LCD. This LCD employs two different subsystems. One subsystem is responsible for communicating with the microcontroller on what to display and how to display it; and the other subsystem is solely responsible for the backlighting feature of this LCD (Color and brightness).

The following block diagram shows how the module is connected to the lab hardware. The block diagram of the module itself is provided next.

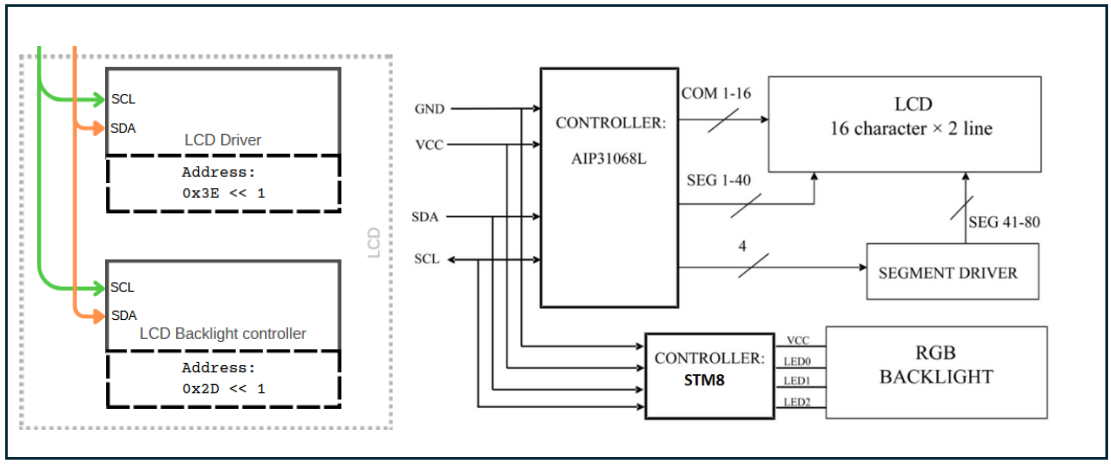


Figure 15. LCD module block diagram

The discussion of how the module works and how to program it is outside of the scope of this document. Readers are encouraged to read its datasheet to gain better understanding of the module itself. However, some information on the datasheet is not accurate for the LCD Backlight feature.

Briefly, this LCD employs another small microcontroller (STM8) which acts as a Slave device. This microcontroller is only responsible for backlight and brightness support. This microcontroller is configured (pre-wired) for I2C communication. It has an address of *0x2D << 1*. This LCD also employs a I2C LCD driver (address: *0x3E << 1*), along with an SDRAM module to store characters. This allows additional characters to be stored and displayed without any intervention from the Master device (Nucleo).

Module name	Module address	Register name	Register address	Value
LCD Backlight	0x2D << 1	Red light brightness	0x01	8-bit (0x00 – 0xFF) Different value represents different shades of RED. 0x00 is minimum, while 0xFF is maximum.
		Green light brightness	0x02	8-bit (0x00 – 0xFF) Different value represents different shades of GREEN. 0x00 is minimum, while 0xFF is maximum.
		Blue light brightness	0x03	8-bit (0x00 – 0xFF) Different value represents different shades of BLUE. 0x00 is minimum, while 0xFF is maximum.
LCD Driver	0x3E << 1	Datasheet		

Table 8. LCD RGB backlight module register description

Pin name	Pin mode	Signal name	Signal description
PF0	I2C_SDA	SDA	This is the bidirectional **S**erial **DA**ta signal line from I2C Master (microcontroller) to I2C Slaves (Touch sensor, I/O Expander, LCD character driver, LCD backlight driver).
PF1	I2C_SCL	SCL	This is a **S**erial **CL**ock signal from microcontroller to all I2C slaves. The frequency of the signal has to be compatible with all the devices using this signal. For the lab, the frequency is 100 kHz.

Table 9. Pin table for LCD module

Buzzer

The buzzer is located just beside the I/O Expander section on the board. This module can generate tone either by programming pin *PA0* in ***GPIO – Output*** mode or ***TIMER – TIM2_CH1/TIM5_CH1*** mode. If configured as GPIO – Output mode, the buzzer will produce a constant beep when the Output is set to logic HIGH. However, if configured as TIM2_CH1 with PWM generation enabled, the frequency and volume of the tone can be changed by changing the duty cycle and frequency of the PWM signal. Since the input is directly coupled with microcontroller, PWM controller can be used to change the signal characteristic of the buzzer.

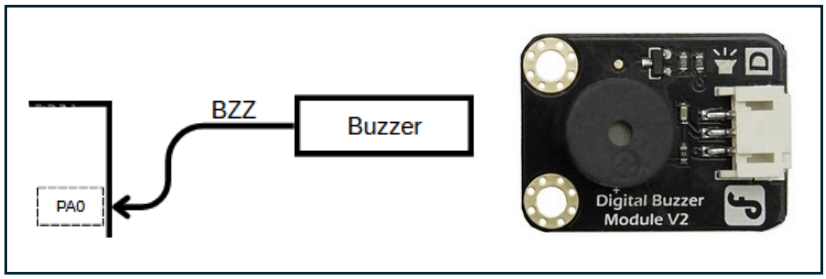


Figure 16. Buzzer module

Pin name	Pin mode	Signal name	Signal description
PA0	GPIO – Output TIM2_CH1 – PWM TIM5_CH1 – PWM	BZZ	This is the signal from microcontroller to **B**u**ZZ**er module. If configured in GPIO – Output mode, the Buzzer produces a constant beep is the Output is set to logic HIGH. If configured in PWM mode, the volume of the buzzer can be set by setting the DUTY CYCLE, and the frequency of the buzzer can be set by setting the FREQUENCY

Table 10. Pin table for Buzzer module

Speed throttle / Potentiometer

On the bottom right corner of the board, you will see a component which slides vertically. This component is labeled 'Speed'. This is a Potentiometer. The value of this potentiometer is not important, because it essentially is connected to act like a voltage divider. The wiper of this potentiometer is directly coupled with the microcontroller pin *PC0*. Therefore, to use the potentiometer, the user will have to use ADC and convert the voltage reading into the % change. By default, the down position of the slider is at 0% and the up position is 100%; however, the user can also design the application employing the opposite behavior. The value of voltage present at the wiper pin of potentiometer can also be read by probing the Test-point. One additional note: if a user wish to use a different type of Potentiometer (such as rotatory pot., etc.), or if a user wish to use a complete different type of analog sensor, he/she may interface the sensor via the Test-point socket for potentiometer, or via the external signal conditioning block, as discussed in the section titled *Analog temperature sensor*.

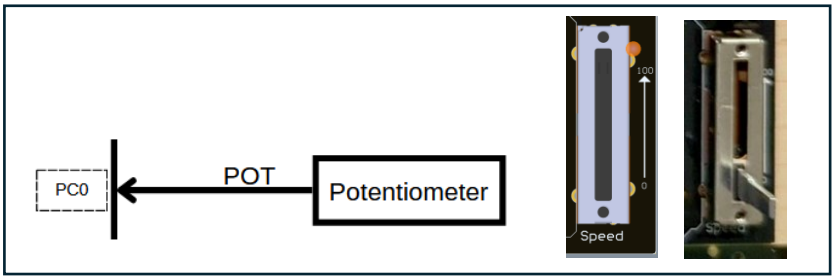


Figure 17. Speed throttle/Potentiometer module

Pin name	Pin mode	Signal name	Signal description
PC0	ADC	POT	This is the signal from Speed/Throttle/Potentiometer to the microcontroller. User must use ADC to detect the proper level. In addition to ADC, user may enable DMA to automatically capture and process the input signal levels.

Table 11. Pin table for Speed module

Motor and Motor driver

The block diagram of the motor system is shown below.

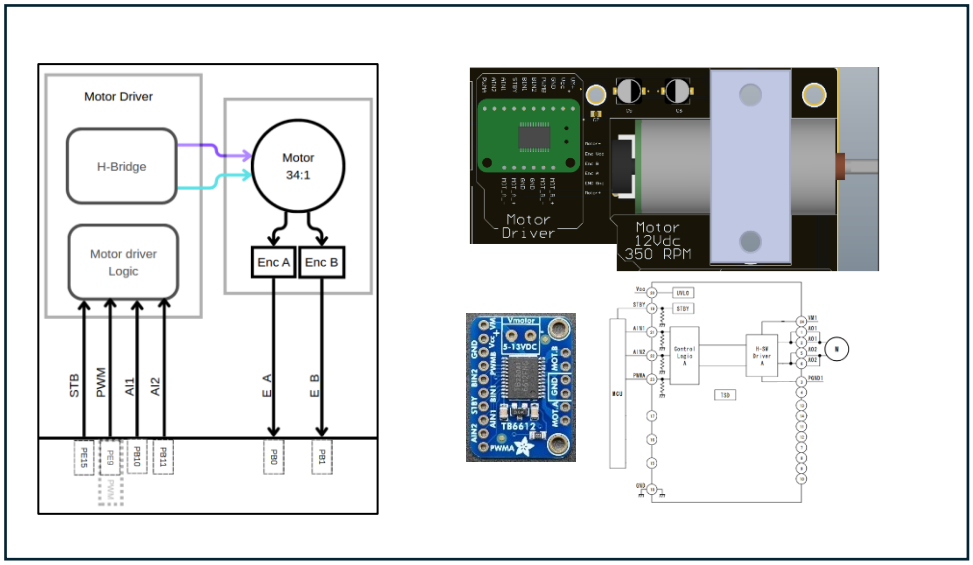


Figure 18. Motor driver block diagram

The power ports for the motor (violet and blue wires on the diagram) are directly connected with the motor driver. This ***motor driver*** gets four signals from the microcontroller: ***STB*** (Standby signal – connected to pin *PE15*), ***PWM*** (Speed control signal – connected to pin *PE9*), and ***AI1*** and ***AI2*** (direction control signal – connected to pin *PB10* and *PB11*).Therefore, by controlling these signals, the motor driver can be used to drive the motor in both directions with variable speed.

The complete description of how the motor driver works is not within the scope of this document, however interested readers are encouraged to read the data sheet (Appendix D). The function table for the motor driver is given in Table 12 (next page).

H-SW Control Function

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Table 12. Control function table for Motor driver module

The ***motor*** connected to the board is a ***brushed dc motor*** with two channel hall-effect encoder (***Enc A*** and ***Enc B*** – connected to pin *PB0* and *PB1*), as seen below. These encoders form the **Feedback loop**, to let the microcontroller know the exact speed and direction of the motor. Therefore, a user can create an open-loop or closed-loop feedback control system.

Figure 19. Motor and Encoders

The ***encoder*** resolution is 11 pulse signals per revolution (each channel). The rated speed for this motor is 350 RPM at 12V; however, the speed is greatly reduced due to the gear box that is attached to the shaft of the motor. The ***gear***

box ratio is 34:1, which means that the output of the gear box (output shaft) revolves one time when the input shaft rotates thirty-four times. Therefore, the encoder signals from each channel will be 374 (34 * 11) for each revolution of the output. The following table summarizes this information:

Motor rated no-load speed	**350rpm @ 12V**
Number of Encoders	2
Resolution of each encoder	11 pulse per revolution (motor shaft)
Gear ratio	34:1
Resolution of each encoder	374 pulse per revolution (output shaft)

Table 13. Motor data

Table 14. Pin table for Motor driver module

Pin name	Pin mode	Signal name	Signal description
PB0	GPIO – External Interrupt	E_A	This is the signal coming from **E**ncoder **A** (attached to the motor). By knowing the relative location of Encoder_A with respect to Encoder_B, the direction of motor can be determined. By sensing the period of tick interval in this signal, the speed of motor can be determined.
PB1	GPIO – External Interrupt	E_B	This is the signal coming from **E**ncoder **B** (attached to the motor). By knowing the relative location of Encoder_B with respect to Encoder_A, the direction of motor can be determined.

			By sensing the period of tick interval in this signal, the speed of motor can be determined.
PB10	GPIO - Output	AI1	This is the Channel **A_1** control signal from microcontroller to the motor driver. This signal, along with AI2 and STB determines the behaviour of the motor driver controller.
PB11	GPIO - Output	AI2	This is the Channel **A_2** control signal from microcontroller to the motor driver. This signal, along with AI1 and STB determines the behaviour of the motor driver controller.
PE9	TIM1_CH1	PWM	This is the **P**ulse **w**idth **m**odulated signal from the microcontroller to the motor driver. This signal is used to control the speed of the motor. The direction of rotation and the status of the motor (Standby, running CW, running CCW, stopped) is selected by STB signal
PE15	GPIO - Output	STB	This is the **ST**and**B**y control signal from microcontroller to the motor driver. This signal, along with AI1 and STB determines the behaviour of the motor driver controller.

Test points

The ***Test Points*** section on the board is a 16x2 socket (female header). The name of this connector reflects its core functionality: to probe the system's raw output. During the development of the lab hardware, some provisions were required to test the systems to understand how it was performing in terms of the underlying signal integrity, interference, system noise, power consumption and power rail noise. This debugging system evolved over time with each iteration of the board; now it acts as an interfacing/debugging system for the external circuits and test equipment.

Therefore, all the *system variables can be probed* by connecting voltmeter/ oscilloscope to this header. Furthermore, a user can also *inject signals* to the system, emulating the sensor output. Care should be taken when probing the Test-points: Ground wire of the test equipment must be properly connected to GND sockets to prevent damaging the system.

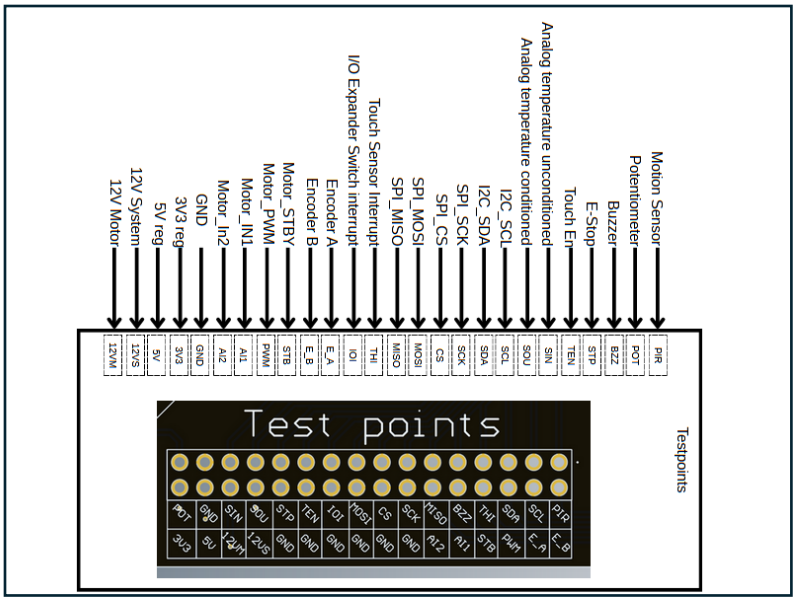


Figure 20. Test points signals

The signal names are written on the board to help the user to identify the subsystems that are connected with individual headers on the Test point connector. Appendix A describes these signals in detail.

iv. Sensors

Touch sensor

The ***Touchpad*** is located just beside the Environment sensor. The ***Touch sensor*** is located just above the LCD. Touchpad consists of ***twelve electrodes***, and each electrode is connected directly to the touch sensor. These electrodes essentially acts as a *big metal plate which can hold certain amount of electric charge*. The *touch sensor works by detecting the drastic changes in the quantity of the charge that was previously stored*. Therefore, this type of input sensing is called ***capacitive touch sensing***, since the *underlying principle is based on detecting the change in energy stored in the electric field of the electrodes*. Touch sensor is coupled with additional hardware filters to reduce any false triggering.

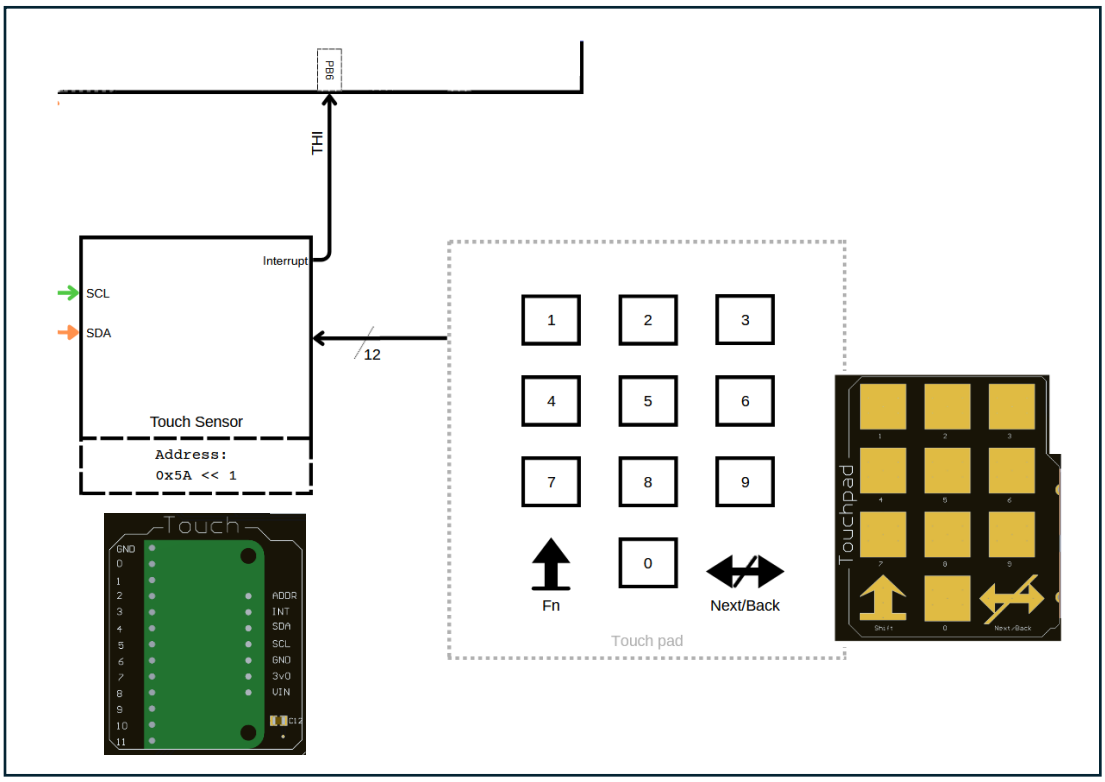


Figure 21. Touch sensor module

The touch sensor communicates with the microcontroller using I2C. The I2C pins on the microcontroller are SDA (pin *PF0*) and SCL (pin *PF1*). Interestingly, the

touch sensor implemented in this lab hardware is highly configurable, and hence the sensitivity level of these electrodes can be individually configured. Other parameters of the sensor can also be adjusted, such as base charge on the electrode, filtering resolution, etc. Hence, user must understand how the sensor works in order to properly configure it.

Additionally, the touch sensor also send an pulse signal to the microcontroller (from its interrupt pin) every time the sensor detects the change in capacitance of any electrodes connected to it. This interrupt signal is directly coupled to the microcontroller on pin *PB6* and it can be also probed using the *Test point* header. The touch sensor also supports the multi-touch; meaning it can detect multiple touch input simultaneously. This allows users to develop applications that need full integration of the interrupt driven touch sensing.

Users should note that they can create different applications that configure the touchpad as a numeric input, or alpha-numeric input. Therefore, the user is required to implement a state machine and use the touchpad as they wish. In addition to that, users are encouraged to use additional *guard conditions* using the ***Touch En*** button, to create robust embedded system.

Table 15. Pin table for Touch sensor module

Pin name	Pin mode	Signal name	Signal description
PB5 (Optional)	GPIO - Input	TEN	This is the raw signal from the **T**ouch **En** switch. Please note: This signal has not been debounced. Hence users must employ their switch debouncing logic in their software to use this signal properly.

PB6	GPIO – External Interrupt	THI	This is the signal from **T**ouc**h** sensor **I**nterrupt pin. The interrupt line goes HIGH momentarily whenever a user touches or releases any electrodes of the touchpad.
PF0	I2C_SDA	SDA	This is the bidirectional **S**erial **DA**ta signal line from I2C Master (microcontroller) to I2C Slaves (Touch sensor, I/O Expander, LCD character driver, LCD backlight driver).
PF1	I2C_SCL	SCL	This is a **S**erial **CL**ock signal from microcontroller to all I2C slaves. The frequency of the signal must be compatible with all the devices using this signal. For the lab, the frequency is 100 kHz.

Analog temperature sensor

On the bottom left section of the board, you will notice a switch labeled ***S2***, a socket (female header) labeled ***J5*** and a semi-circular electronic component. These components belong to the section titled ***Temperature sensor*** on the board. The semi-circular electronic component is the analog temperature sensor itself. The block diagram and physical placement of components is shown below.

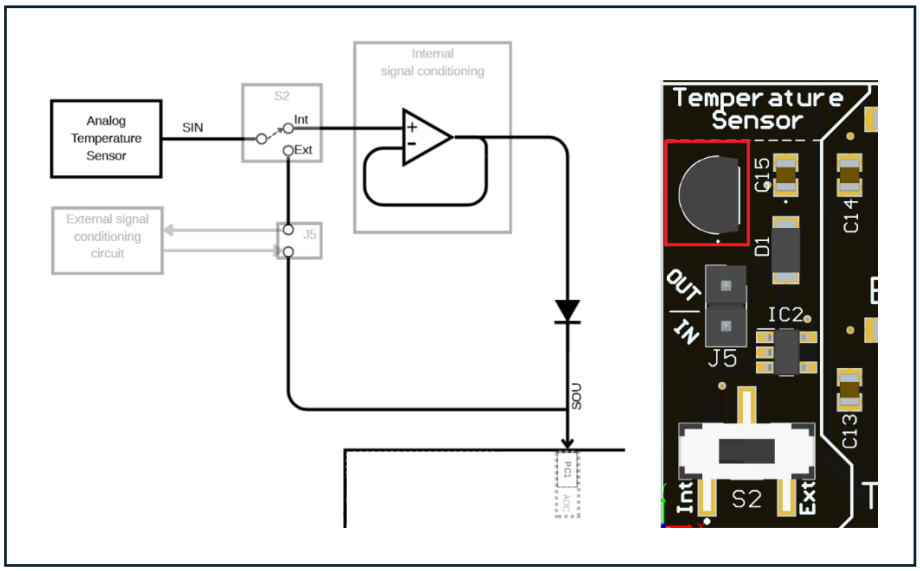


Figure 22. Temperature sensor module

The user has ability to route or channel the raw output of temperature sensor to the *internal signal conditioning block*, or to the **user defined** *external signal conditioning* block. This can be done by using the switch ***S2***. The reason for this design is due to wide temperature sensing range of the sensor (from -50°C to +150°C). Also, the output of temperature sensors is approximated to be linear for temperature range of -30°C to +90°C at **-10.9 mV/°C**. However, for the lab environment, this range is still too wide and therefore a user may not be able to get precise data due to quantization error that will result from sampling the voltages in the ADC of the microcontroller. For this reason, if a user wants to get more precise temperature sensor reading, he/she is encouraged to create their own external signal conditioning block that can provide necessary gain for the

expected temperature ranges and therefore get maximum accuracy and precision in their application.

Internal signal conditioning

When the switch ***S2*** is in *left* position, the temperature sensor is coupled to the microcontroller via internal signal conditioning block. As mentioned previously, the ***internal signal conditioning block has no gain***; it simply acts as a buffer/driver.

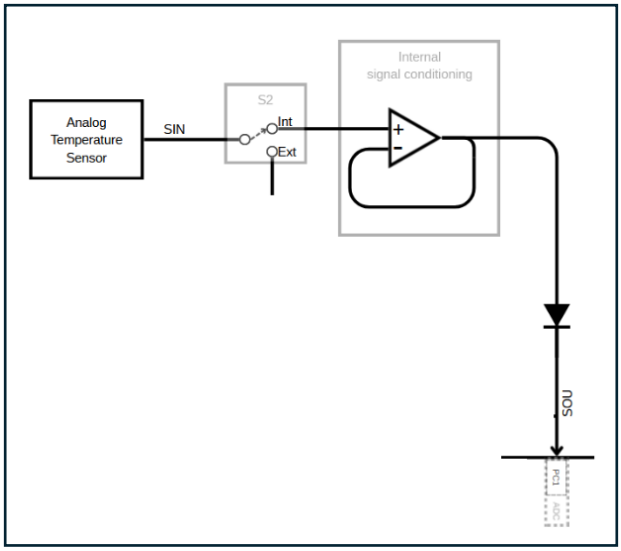


Figure 23. Internal signal conditioning block diagram

External signal conditioning

When the switch is in right position, the raw output of the temperature sensor appears on the socket pin labeled ***OUT***. A user will simply connect his/her external signal conditioning block by *connecting the* ***OUT pin to the input*** *of the external signal conditioning circuit*, and the ***output of the external signal conditioning circuit*** *will go back* ***to IN pin*** *of the socket*. The following image shows the external signal conditioner with the system.

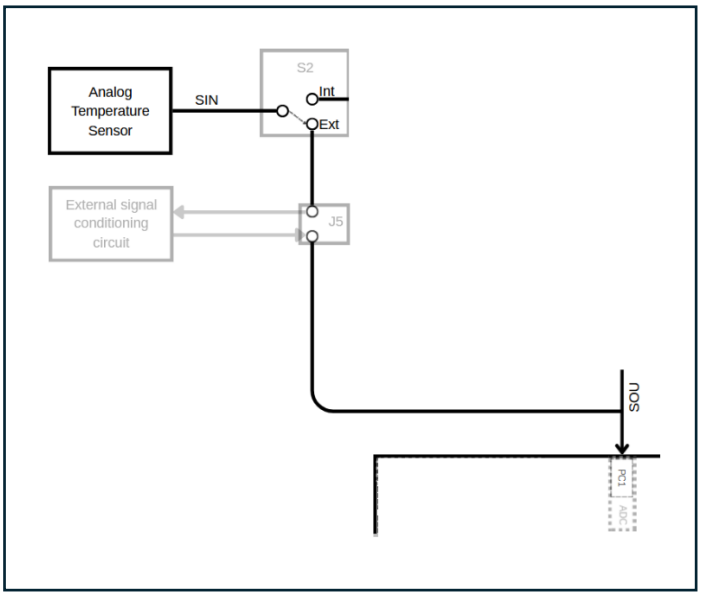


Figure 24. External signal conditioning block diagram

Pin name	Pin mode	Signal name	Signal description
PC1	ADC (with or without DMA)	SOU	This is the **S**ignal conditioner **OU**tput of analog temperature sensor to the microcontroller. The signal conditioner block can be internal or external, depending on the switch S2. User must use ADC to detect the proper level. In addition to ADC, user may enable DMA to automatically capture and process the input signal levels.

Table 16. Pin table for Analog temperature sensor module

Environmental sensor

Just beside the buzzer, you will see a small module with a very small metal enclosed square component in the middle. This is the ***environment sensor***. The square component is the actual sensor, which can measure *Pressure, Humidity, Temperature* and *Volatile Organic Compound (VOC)*. The diagram below highlights the actual temperature sensor as well as provides the block diagram/connection details of the sensor with the microcontroller.

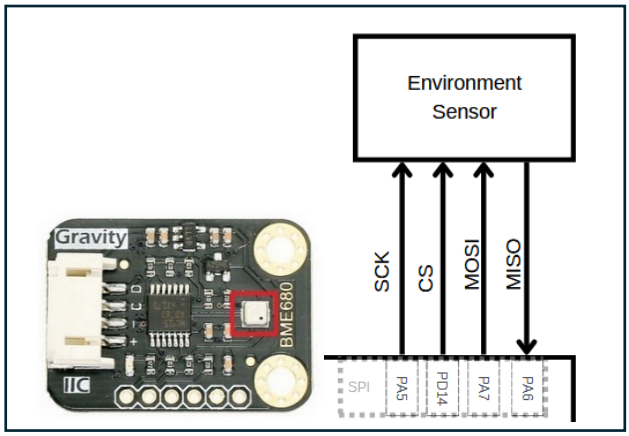


Figure 25. Environmental sensor module

This sensor uses ***SPI protocol*** (not I2C) to communicate with the host microcontroller. SPI protocol requires four signal lines to work properly: ***SCK*** (Serial ClocK – connected to pin *PA5*), ***CS*** (Chip Select – connected to pin *PD14*), ***MOSI*** (Master Out Slave In – connected to pin *PA7*), and ***MISO*** (Master In Slave Out – connected to pin *PA6*). Please note that this is an advanced sensor which requires knowledge of sensor fusion. You must configure this sensor via SPI to use it. The complete description is provided in the datasheet of this module (Appendix D).

Pin name	Pin mode	Signal name	Signal description
PA5	SPI1_SCK	SCK	This signal is the **S**PI **C**loc**K** signal for all SPI modules in the system.
PA6	SPI1_MISO	MISO	This signal is **M**aster **I**n **S**lave **O**ut. This is the data line from Slave (SPI_subordinate) to Master (SPI_Controller)
PA7	SPI1_MOSI	MOSI	This signal is **M**aster **O**ut **S**lave **I**n. This is the data line from Master (SPI_Controller) to Slave (SPI_subordinate)
PD14	GPIO - Output	CS	This signal is the **C**hip **S**elect signal for an individual SPI module in the system. This signal is hardwired to select the Environmental Sensor on the lab hardware platform.

Table 17. Pin table for Environment sensor module

Digital Motion sensor

Just above the LCD, on the right-hand side of the board, you will see a module with white lens. This module is the ***Motion sensor***, and it *senses the motion* of any object nearby. Please note that this is ***not a distance sensor***; it only *tells you if any motion is detected*. It is *not used to measure the distance* of the object that is moving. You can, however, interface an additional distance sensor by connecting the Test-point header's socket that connects the Motion sensor to the microcontroller. The sensor looks like the image shown below.

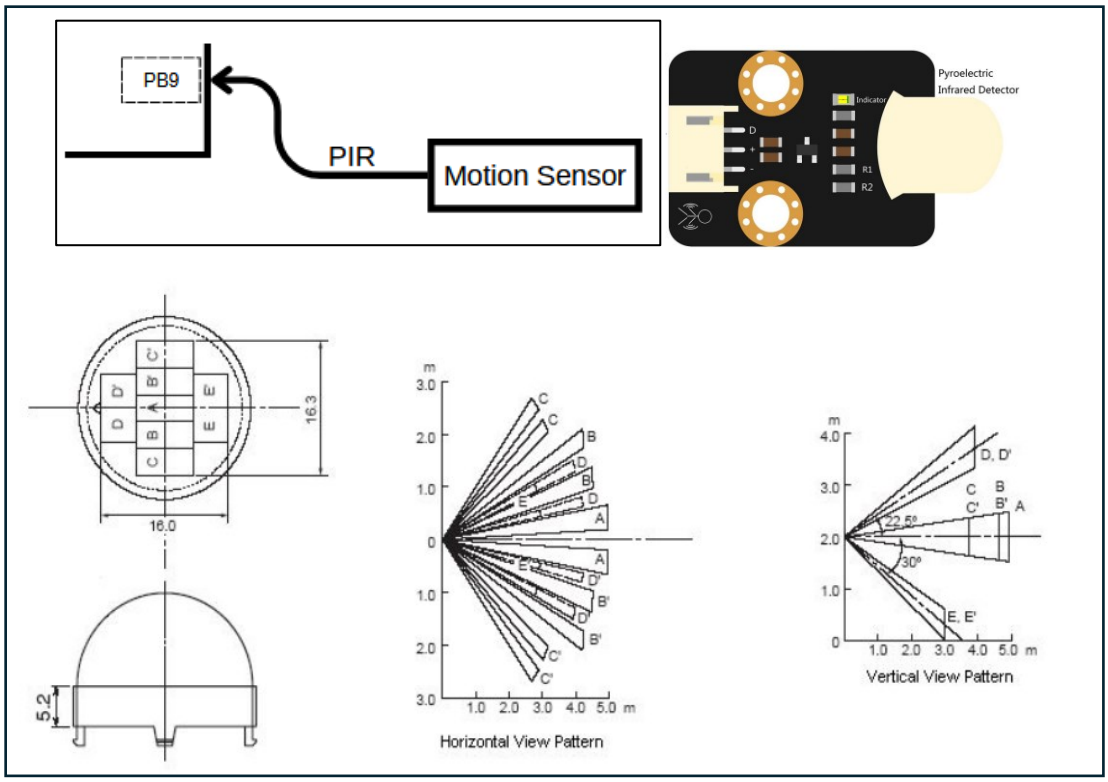


Figure 26. Motion sensor module and data

The motion sensor is connected to pin *PB9* of the microcontroller. Whenever any motion is detected, the motion sensor makes the signal go logic HIGH for a set duration. If the motion continues, the motion sensor is triggered continuously, and the signal will stay HIGH until the set duration after the motion has stopped. This signal level changes can be detected optimally by setting up the appropriate

interrupt line (EXTI) to edge-trigger mode or level-trigger mode. The following image summarizes this information.

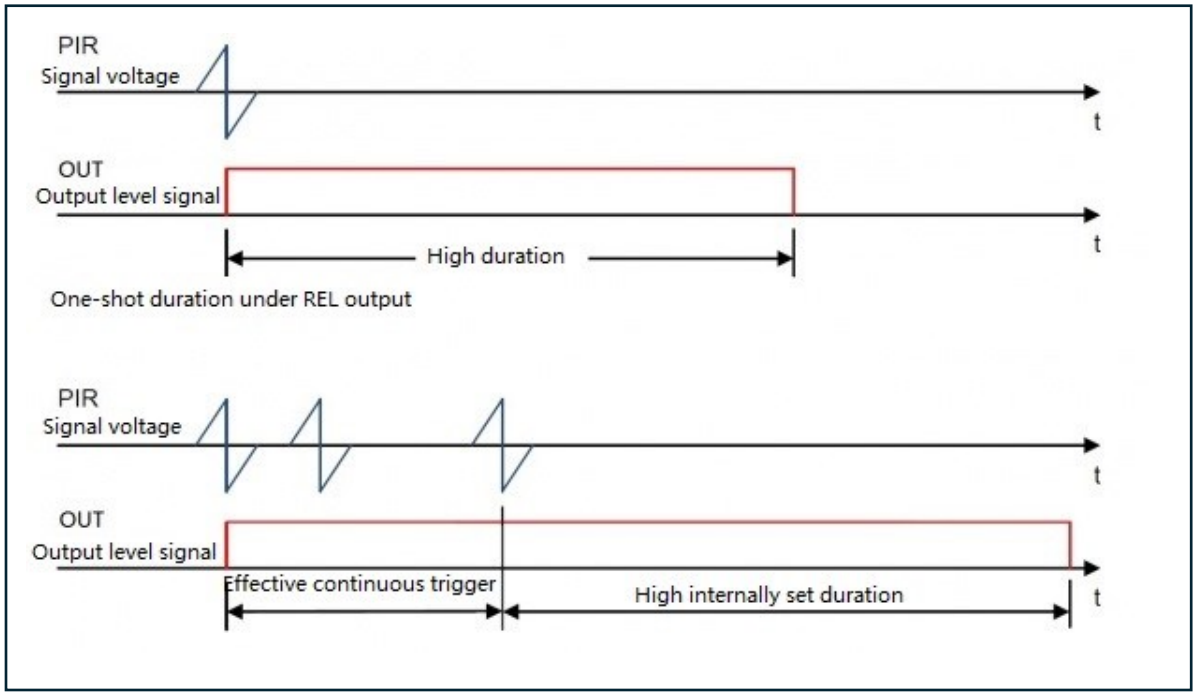


Figure 27. Motion sensor signal waveform

Pin name	Pin mode	Signal name	Signal description
PB9	GPIO - Input GPIO – External Interrupt	PIR	This signal comes from the Motion sensor (**PIR** sensor). A logic HIGH on this signal means motion is detected.

Table 18. Pin table for Motion sensor module

Programming/debugging and Communication

Programming and debugging the microcontroller

The programming of the *STM32L552ZET6QU* is done via the ***ST-Link/V2*** integrated chip that is located on the top section of *Nucleo L552ZE-Q* development board, as shown below.

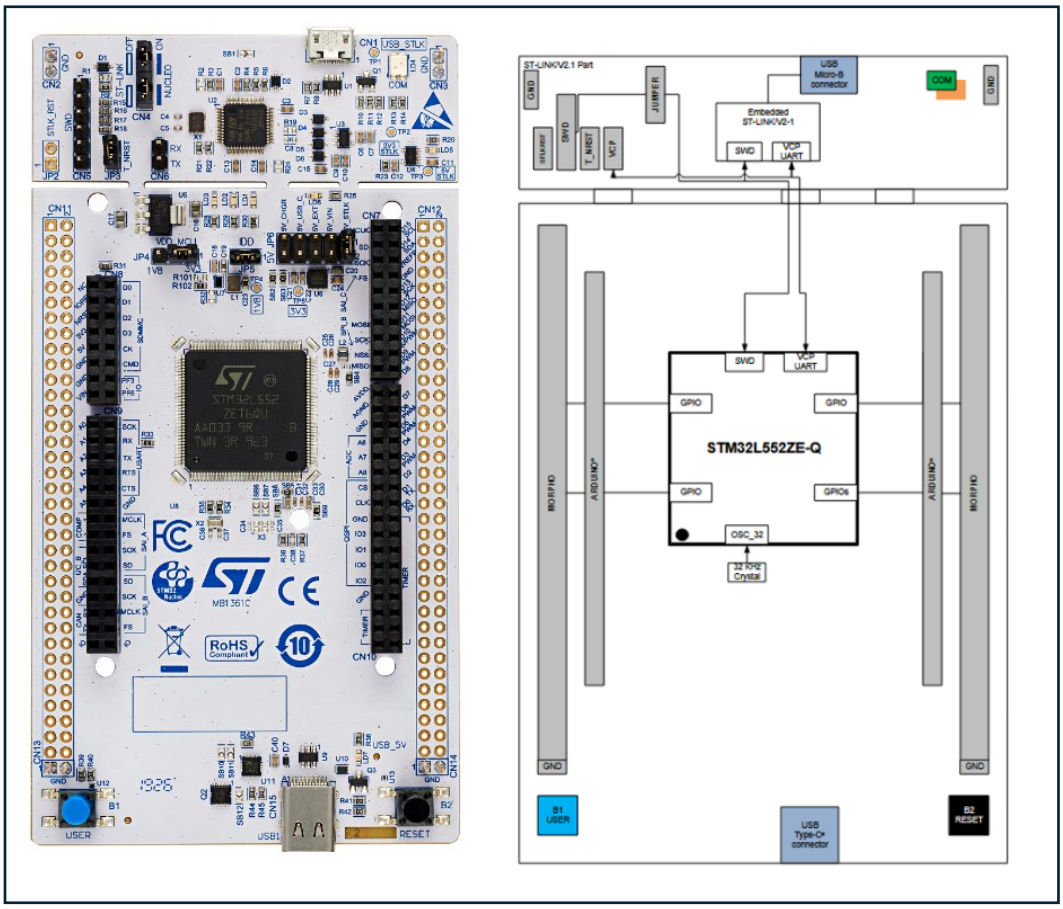


Figure 28. Programming and debugging block diagram

The technical name of this type of programmer is ***In-Circuit Debug Interface*** (*ICDI*). Therefore, ST-Link/V2 is a type of In-circuit debugger/programmer, which allows the user to access the **Serial Wire Debugging interface** (SWD) of the microcontroller. This is the signal line which allows advanced debugging possible with STM32 microcontrollers. For example, user can issue software command to the

microcontroller to halt the operation at any given time, or at any given instruction. Users can also inspect the register content and system state in real time by invoking ST-Link/V2 chip.

Communicating with the microcontrollers

In addition to providing a way for user to program and debug the microcontroller, ST-Link/V2 also makes it easy to set up communication between the host pc and the microcontroller. This is done via the ***Virtual COM Port*** (*VCP*). The VCP port provides direct access to the Low Power UART (Universal Asynchronous Receiver Transmitter), or *LPUART*_1, peripheral of the microcontroller. The Transmit (***TX***) pin of the LPUART_1 is connected to pin *PG7*, and the Receive (***RX***) pin of LPUART_1 is connected to pin *PG8* of the microcontroller, as shown below.

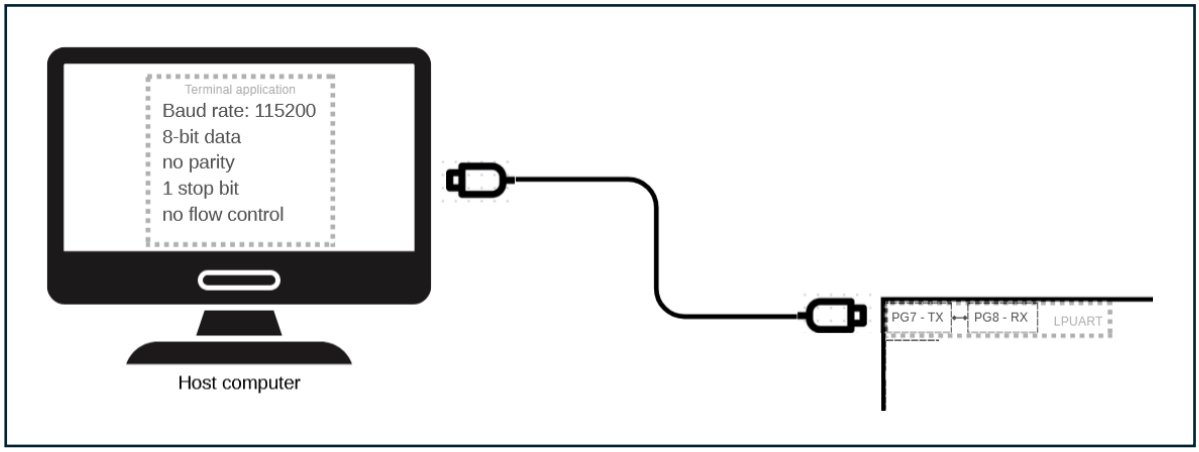


Figure 29. Communication with the development computer.

microcontroller is established via UART protocol, any terminal emulation applications such as ***PuTTY*** or ***MobaXTerm*** can be used to establish and maintain the communication with the microcontroller. By default, the following UART protocol parameters are applied to the ***LPUART_1*** peripheral on the microcontroller:

BAUD RATE	**115200 BPS**
DATA	8-bit
PARITY	None
STOP BIT	1
FLOW CONTROL	None

Table 19. UART communication parameters

It is possible to change the default parameters, but user must ensure that both the terminal application as well as the microcontroller are using the same communication parameters.

Pin name	Pin mode	Signal name	Signal description
PG7	LPUART1_TX	TX	This signal is the Transmit (**TX**) pin from the LPUART module on Nucleo Board. This is used to enable communication between the Nucleo board and the host computer.
PG8	LPUART1_RX	RX	This signal is the Receive (**RX**) pin from the LPUART module on Nucleo Board. This is used to enable communication between the Nucleo board and the host computer.

Table 20. Pin table for UART module

Communication status

The status of the communication between the target (ST-Link/V2) and the PC can be visually seen on the Nucleo L552ZE-Q board by watching the state of ***LD4*** led on the top right corner of the development board. This tricolor (Green, orange, and red) LED (LD4) provides information about ST-LINK communication status.

- *The LD4* ***default color*** *is red*.
- LD4 **turns to green** to indicate that the communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

LED	LED STATE	DESCRIPTION
RED	Slow blinking / OFF	At power-on before USB initialization
	Fast blinking / OFF	After the first correct communication between PC and ST-LINK/V2
	ON	When the initialization between the PC and ST-LINK/V2-1 is complete
RED-GREEN BLINKING	-	During communication with the target
GREEN	ON	After a successful target communication initialization
ORANGE	ON	Communication failure

Table 21. Communication LED status codes

Integrated Development Platform (IDE)

There are several software that are available in the market that unify the design, development, compilation, programming and debugging of embedded systems applications. These software are often called ***IDEs*** (or Integrated Development environment) or Programming toolchain. IDEs often integrates different software for a specific task, for example GCC for programming and debugging, text editor for editing the source files, GNU C/C++ compiler for compilation, and sometimes GUI for hardware initialization for specific microcontroller. For developing users' application for STM32 microcontrollers, and communicating with ST-Link/V2 IC, you have following options:

- STM32CubeIDE
- STM32CubeProgrammer
- STM32CubeMonitor
- Keil®
- IAR™

Either use the tools officially supported by STMicroelectronics (STM32CubeIDE), STM32CubeProgrammer, STM32CubeMonitor or use the tools officially supported by ARM (Keil IDE) or by third party IDE (IAR™). They are all supported by STM32 architecture.

Here is the difference: STM32CubeIDE gives you a visual/GUI for their boards. Meaning, the assignment of GPIO pins, setting their functions, etc. is very beginner friendly as it has a middle layer that automatically generates the code for administrative tasks. So, if you are just beginning your journey in embedded systems and if your focus is on application development rather than diving deep into the internal details, you should go with STM32CubeIDE.

STM32CubeIDE also gives you a set of management software – STM32CubeMX, which is a Hardware Abstraction Layer (HAL) with GUI interface for their microcontrollers, so that you can quickly program your application and let the HAL

worry about the details. The image on next page shows this integration and highlights some useful features of STM32CubeIDE.

Here is the link to example projects.

Please follow this link to know more about STMCubeIDE.

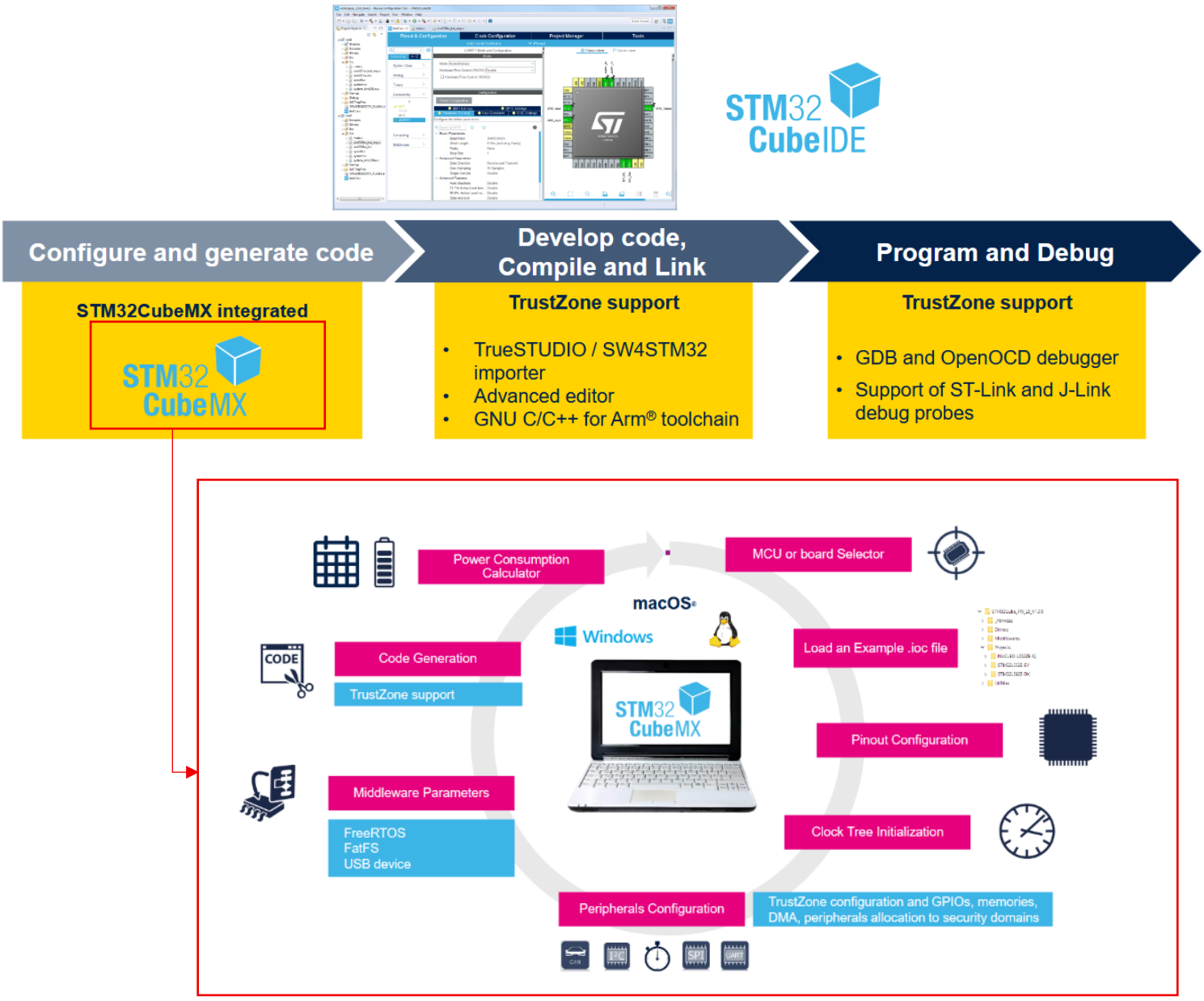


Figure 30. STM32CubeIDE ecosystem

Another IDE that is officially supported by STM32 microcontroller architecture as well as ST-Link/V2 IC is Keil IDE. Keil IDE is popular among developers who prefer

vendor neutral IDE for ARM architecture. To understand what that means, you must understand what ARM really is and how STMicroelectronics fits into the picture. The detailed description is provided in Appendix B.

The discussion of IAR™ IDE is not within the scope of this manual.

Appendix

A. Block diagram and Pin description table of microcontroller

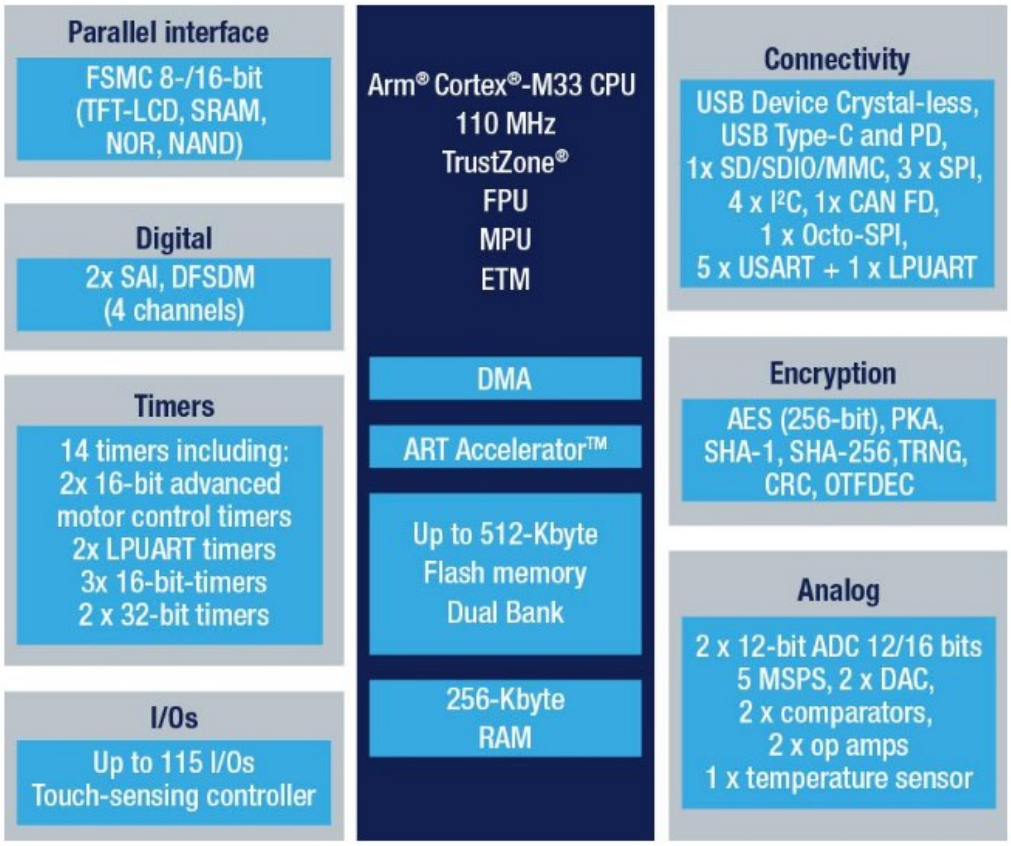
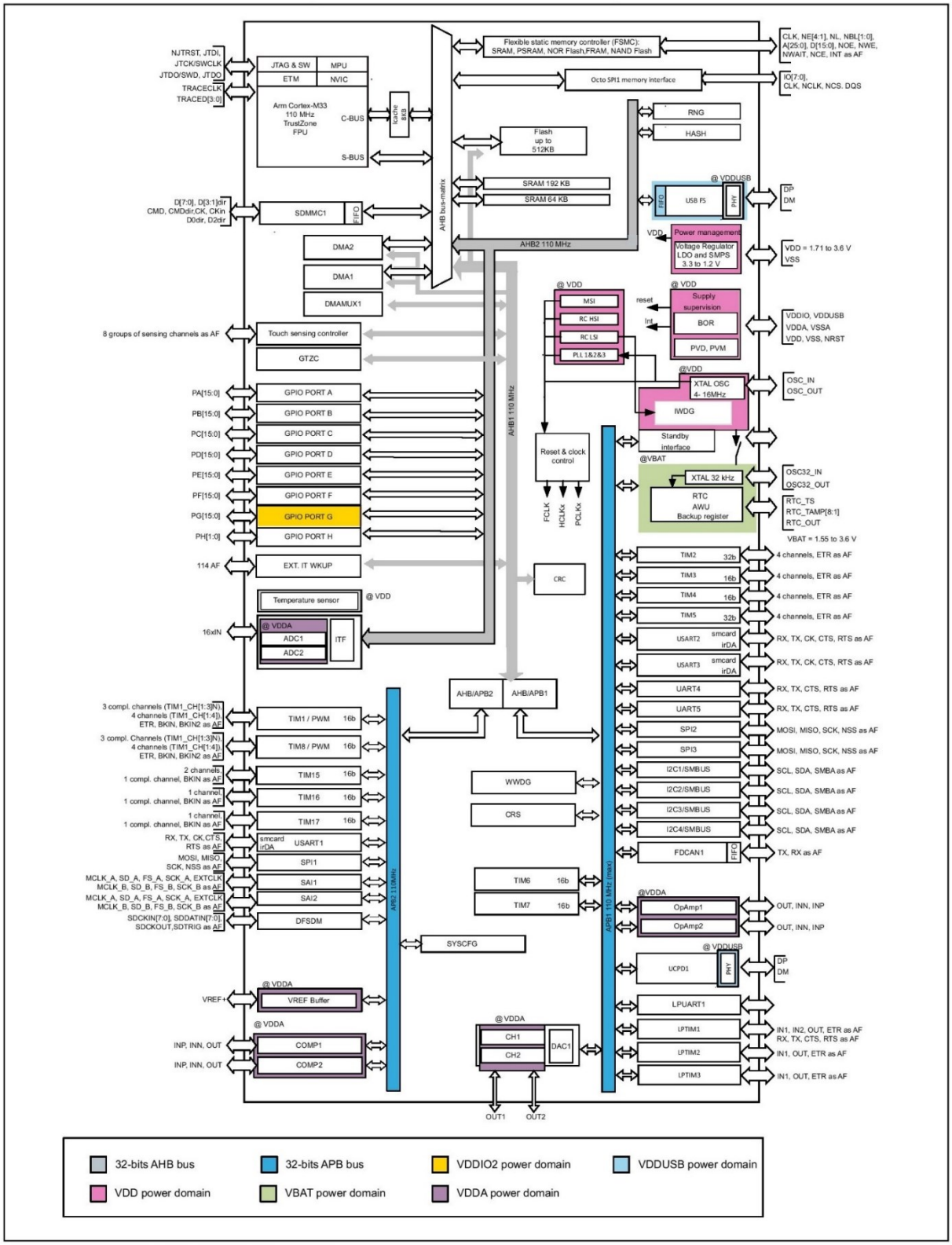


Figure 31. Peripheral diagram of STM32L5



1. AF: alternate function on I/O pins.

Figure 32. Block diagram of STM32L5 microcontroller

Table 22. Pin description table for entire system

Pin name	Pin mode	Signal name	Signal description
NRST	RESET	NRST	A logic LOW on this pin will reset the microcontroller.
PA0	GPIO – Output TIM2_CH1 – PWM TIM5_CH1 – PWM	BZZ	This is the signal from microcontroller to **B**u**ZZ**er module. If configured in GPIO – Output mode, the Buzzer produces a constant beep is the Output is set to logic HIGH. If configured in PWM mode, the volume of the buzzer can be set by setting the DUTY CYCLE, and the frequency of the buzzer can be set by setting the FREQUENCY
PA5	SPI1_SCK	SCK	This signal is the **S**PI **C**loc**K** signal for all SPI modules in the system.
PA6	SPI1_MISO	MISO	This signal is **M**aster **I**n **S**lave **O**ut. This is the data line from Slave (SPI_subordinate) to Master (SPI_Controller)
PA7	SPI1_MOSI	MOSI	This signal is **M**aster **O**ut **S**lave **I**n. This is the data line from Master (SPI_Controller) to Slave (SPI_subordinate)
PA8	GPIO – External Interrupt	IOI	This signal is the **I**/**O** **I**nterrupt from I/O Expander module connected with the on board switches. This signal stays high if any of the switches of I/O expander (SW1 to SW8) is pressed. The signal goes low automatically when the microcontroller reads the switch state.

PA9	GPIO – Output TIM1_CH2 - PWM	Red_LED	This is the signal for on board RED LED light. The brightness can be adjusted if configured in PWM mode.
PB0	GPIO – External Interrupt	E_A	This is the signal coming from **E**ncoder **A** (attached to the motor). By knowing the relative location of Encoder_A with respect to Encoder_B, the direction of motor can be determined. By sensing the period of tick interval in this signal, the speed of motor can be determined.
PB1	GPIO – External Interrupt	E_B	This is the signal coming from **E**ncoder **B** (attached to the motor). By knowing the relative location of Encoder_B with respect to Encoder_A, the direction of motor can be determined. By sensing the period of tick interval in this signal, the speed of motor can be determined.
PB2	GPIO – Input GPIO – External Interrupt	STP	This is the raw signal from the E-**St**o**p** switch. Please note: This signal has not been debounced. Hence users must employ their switch debouncing logic in their software to use this signal properly.

PB5	GPIO - Input	TEN	This is the raw signal from the **T**ouch **En** switch. Please note: This signal has not been debounced. Hence users must employ their switch debouncing logic in their software to use this signal properly.
PB6	GPIO – External Interrupt	THI	This is the signal from **T**ouc**h** sensor **I**nterrupt pin. The interrupt line goes HIGH momentarily whenever a user touches or releases the any electrodes of the touchpad.
PB7	GPIO – Output TIM4_CH2 – PWM TIM17_CH1N - PWM	Blue_LED	This is the signal for onboard BLUE LED light. The brightness can be adjusted if configured in PWM mode.
PB9	GPIO - Input GPIO – External Interrupt	PIR	This signal comes from the Motion sensor (**PIR** sensor). A logic HIGH on this signal means motion is detected.
PB10	GPIO - Output	AI1	This is the Channel **A_1** control signal from microcontroller to the motor driver. This signal, along with AI2 and STB determines the behaviour of the motor driver controller.
PB11	GPIO - Output	AI2	This is the Channel **A_2** control signal from microcontroller to the motor driver. This signal, along with AI1 and STB determines the behaviour of the motor driver controller.

PC0	ADC	POT	This is the signal from Speed/Throttle/Potentiometer to the microcontroller. User must use ADC to detect the proper level. In addition to ADC, user may enable DMA to automatically capture and process the input signal levels.
PC1	ADC	SOU	This is the **S**ignal conditioner **OU**tput of analog temperature sensor to the microcontroller. The signal conditioner block can be internal or external, depending on the switch S2. User must use ADC to detect the proper level. In addition to ADC, user may enable DMA to automatically capture and process the input signal levels.
PC7	GPIO – Output TIM3_CH2 - PWM	Green_LED	This is the signal for on board GREEN LED light. The brightness can be adjusted if configured in PWM mode.
PC13	GPIO – Input GPIO - EXTI	Blue_Button	This is the signal from the User switch (Blue button). Please note: This signal ***has been debounced*** in hardware. However, users are encouraged to employ their switch debouncing logic in their software to use this signal properly.

PD14	GPIO - Output	CS	This signal is the **C**hip **S**elect signal for an individual SPI modules in the system. This signal is hardwired to select the Environmental Sensor on the lab hardware platform.
PE9	TIM1_CH1	PWM	This is the **P**ulse **w**idth **m**odulated signal from the microcontroller to the motor driver. This signal is used to control the speed of the motor. The direction of rotation and the status of the motor (Standby, running CW, running CCW, stopped) is selected by STB signal
PE15	GPIO - Output	STB	This is the **ST**and**B**y control signal from microcontroller to the motor driver. This signal, along with AI1 and STB determines the behaviour of the motor driver controller.
PF0	I2C_SDA	SDA	This is the bidirectional **S**erial **DA**ta signal line from I2C Master (microcontroller) to I2C Slaves (Touch sensor, I/O Expander, LCD character driver, LCD backlight driver).
PF1	I2C_SCL	SCL	This is a **S**erial **CL**ock signal from microcontroller to all I2C slaves. The frequency of the signal has to be compatible with all the devices using this signal. For the lab, the frequency is 100 kHz.

PG7	LPUART1_TX	TX	This signal is the Transmit (**TX**) pin from the LPUART module on Nucleo Board. This is used to enable communication between the Nucleo board and the host computer.
PG8	LPUART1_RX	RX	This signal is the Receive (**RX**) pin from the LPUART module on Nucleo Board. This is used to enable communication between the Nucleo board and the host computer.

B. CEG3136 course files

Unfortunately, due to the static nature of this document, it was not possible to include example projects, project ideas and some useful programming references that are specific to the lab hardware. For that reason, we decided to maintain an active repository of some helpful documents and projects that the user may find useful when developing their own labs for this course. This repository contains:

- Sample labs
- Sample project ideas
- Sample completed projects
- Modules datasheets
- Programming reference materials
- Design reference materials

The course website hosts this repository, which can be accessed by scanning the QR code on the lab hardware, which is also shown below.

Figure 33. . CEG3136 course website QR code

The direct link to the website is: https://wantedbull.com/ceg3136/

An alternate GitHub repository for this course is also available here: https://github.com/DLovesHorses/CEG3136

C. Extra training materials

Besides the course website, we have gathered several freely available training materials directly from the STMicroelectronics and ARM learning platforms. These training materials are highly specific to the microcontroller used in Nucleo L552ZE-Q board. Users are highly encouraged to at least check out these resources, since they include almost all aspects of embedded systems development on STML5 series microcontrollers.

Please follow the links provided below to access these training materials:

- *Training material for STM32L5 series microcontroller – from STMicroelectronics.*
- *Training material for ARM Cortex M33 microcontroller architecture – from ARM.*

D. Useful links

This section provides links to all the up-to-date documentations for all modules that are used in this lab platform. This also includes a link to the most up-to-date version of this document. If a reader finds any mistake/error in this document, they are encouraged to check out the latest version of this document and then report the issue if the issue is still not fixed.

TYPE	SUBSYSTEM
DOCUMENTATION	This user manual
	High quality block diagram
CONTROLLER	Nucleo L552ZEQ dev board
	STM32L552ZET6QU – datasheet
	STM32L552ZET6QU – Reference manual
	STM32L552ZET6QU – Programming manual
	ST-Link/V2
	Motor driver
SENSING	Analog temperature sensor
	Touch sensor
	Environmental sensor
	Motion sensor
I/O	RGB LED
	I/O Expander
	Touchpad
	Potentiometer
	LCD
ACTUATOR	Motor

Table 23. Link to module datasheets