Design (E) 314 — Ontwerp (E) 314

Project Definition — Projek Definisie

2023

Dr Arno Barnard, Mr Lanche Grootboom

This document is NOT the final version. Further information will be added throughout the course.

Version History

Version	**Date**	**Changes**
0.1	14 Feb 2023	- Initial project specification and guide
0.2	24 Feb 2023	- Corrections to Demo1 UART specification, sec 4.10.1 - Corrections to UART configuration in table 1
0.3	8 Mar 2023	- Additions/Changes to UART protocol, sec 4.10 - Added International Morse Code table 5 - Updated UR3 in table 1 to refer to table 5 - Major update of preliminary report, chapter 6
0.4	9 Mar 2023	- Update/Clarification of UR3, UR5, and UR7 in table 1 - Added default start-up values to the system operational overview, in section 4.1 - Added details for system operation in sec 4.8 - Added Demo 2 details in table 8
0.5	12 Apr 2023	- Added information about the Azoteq IQS7211A trackpad: hardware section 3.5, and software section 4.12
0.6	18 Apr 2023	- Updated UR6 to match UR3, UR4 w.r.t. trackpad functionality - Updated UR4 to allow for the middle button on/off control of Mood Light - Updated UR7 to describe on/off functionality of middle button and thus align with UR2, UR3, UR4 - Updated terminology throughout the document to better fit the Azoteq datasheet terminology - Added section 2.3 for trackpad slide and colour position definition - Corrected the <Param2> meaning in ME mode, MF->ME in examples and the 'B' values for morse code, in section 4.10.1 - Updated section 4.12 with more guidance and detail - Updated table 8 for Demo 3 information
0.7	10 May 2023	- Updated sections 3.6 and 4.13 with LED and driver information - Updated user requirements in tables 1 and 7 to clarify LED output requirements - Updated table 8 with Demo 4 specification - Updated section 7, final report rubric, due to simplification of system - Updated UR3.1.a in table 1 to be consistent with the rest of the document

1 Purpose of this Document

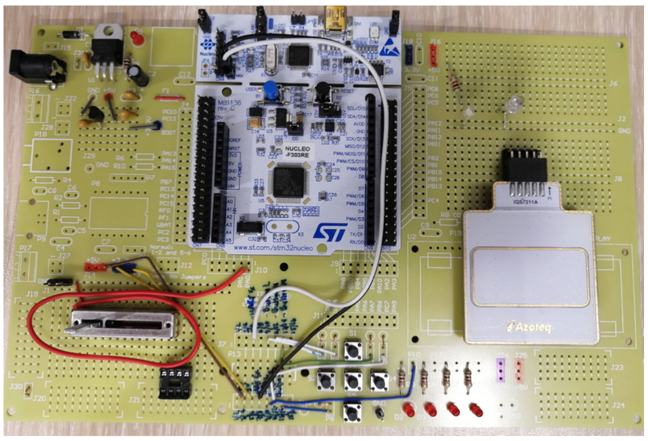
The purpose of this document is to:

- Provide students with a clear **overview and scope** definition of the project;
- Provide clear **project requirements**, that will be used to test the hardware during demonstrations;
- Provide some assistance in understanding certain **concepts/information** about the components that will used in the project;
- Identify the **critical design choices** that the student should solve.
- Provide **guidance for producing the final report** using an assessment rubric.

2 Overview

The project will consist of designing, building and testing a multi-functional light source, that:

1. operates as a flashlight with adjustable intensity
2. operates as an emergency light with adjustable flashing modes
3. operates as an adjustable coloured mood light


(a) An example of the prototype board.

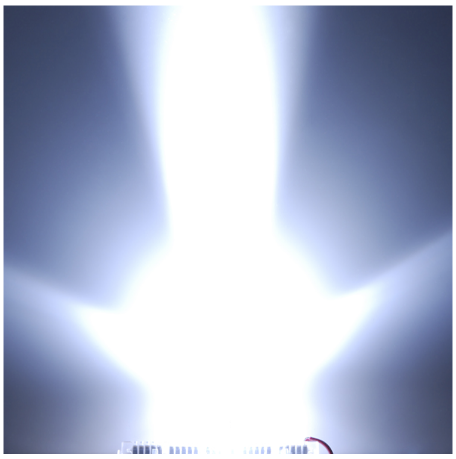

(b) Bright white LED on

Figure 1: Project examples

The requirements that should be fulfilled are listed in Table 1.

Table 1: User requirements for the project.

UR#	Description
UR1	The system shall generate its own regulated 5 V and 3.3 V supply voltage from a nominal 9 V-12 V battery or power supply.
UR2	During flashlight operation, the device shall: 1. switch the white LED on/off using the: (a) middle pushbutton (de-bounced) (b) UART command input (see section 4.10 5.3) (c) Azoteq trackpad - "press-and-hold" gesture 2. change the intensity of the white LED using the: (a) slider input via ADC (b) UART command input (see section 5.3) (c) Azoteq trackpad - slide motion (not swipe)
	Continued on next page

	Table 1 – continued from previous page
UR#	**Description**
UR3	During emergency light operation, the device shall 1. switch between the following emergency modes: (a) a strobe mode, 50% duty cycle at 0.9765 Hz (512 ms on-time by default). If the on-time is changed the duty cycle must remain at 50%. (b) an S.O.S. Morse-coded pattern, as specified in table 5 (c) a multi-character Morse-coded message received via UART command as specified in table 5 2. toggle the white LED on and off using: (a) middle pushbutton (de-bounced) (b) UART command input (see section 5.3) (c) Azoteq trackpad - “press-and-hold” gesture 3. change the intensity of the white LED using the: (a) slider input via ADC (b) UART command input (see section 5.3) (c) Azoteq trackpad - slide motion (not swipe) 4. set the strobe timing using: (a) UART command input (see section 5.3) 5. cycle through emergency modes using: (a) right pushbutton (strobe, SOS, custom msg, strobe, ...) (b) UART command input (see section 5.3) (c) Azotech trackpad - “single tap” gesture (strobe, SOS, custom mag, strobe, ...)
UR4	During mood light operation, the device shall: 1. toggle the RGB LED on/off using: (a) middle pushbutton (de-bounced) (b) UART command input (see section 5.3) (c) Azoteq trackpad - on a “press-and-hold” gesture 2. set the RGB LED colour using: (a) UART command input (see section 5.3) (b) Azoteq trackpad - XY position defines colour values as per figure 3
UR5	The device shall use a UART connection, in both transmit and receive modes. The device shall operate in a 8 data bits, one even parity bit and two stop bits configuration, at a data rate of 57600 baud. The UART information, both transmitted and received, shall be formatted as per section 5.3. The system shall receive commands defined in section 5.3 and respond accordingly. The device shall report the following information via UART: 1. the current operation mode after receiving a valid command 2. when applicable, the level of the intensity of the white LED 3. when applicable, the “on” time of the strobe light 4. when applicable, the current emergency message being flashed 5. when applicable, the intensity of each of the RGB LED channels
	Continued on next page

	Table 1 – continued from previous page
UR#	**Description**
UR6	The device shall use the connected Azoteq Rectangular Trackpad to: 1. switch the white LED on/off on a "press-and-hold" gesture 2. adjust the white LED intensity using the "sliding motion" input 3. to sequentially select the emergency mode type, choosing (in order) between strobe, Morse-coded SOS, and variable Morse-coded message (then repeat) - using the "single tap" gesture 4. switch the RGB LED on/off on a "press-and-hold" gesture 5. adjust the colour of the RGB using the touch position information as per figure 3
UR7	The device shall use push-buttons for manual configuration and control, as per section 4.8, to: 1. to switch the white LED or RGB LED on/off in their respective modes - using the MIDDLE button 2. to sequentially select the operating modes, choosing (in order) between flashlight, emergency light, and mood light (then repeat) - using the LEFT button 3. to sequentially select the emergency mode type, choosing (in order) between strobe, Morse-coded SOS, and variable Morse-coded message (then repeat) - using the RIGHT button
UR8	The device shall use four LEDs to indicate the active system state as per section 4.9
UR9	The device shall implement light sources using the DAC and 4x PWM output channels to drive the high-power white LED and RGB LED. The white LED will be driven using 1x PWM output via a transistor circuit as per section 3.6, and the RGB LED will be driven directly as per section 3.6. NB! Although the DAC will no longer be used to drive the white LED, it must still be available for monitoring!

For more detail about how these specifications will be tested, see section 5.

You will be provided with a ***baseboard*** and an STM32 microcontroller board. The microcontroller board will connect to headers, which have to be soldered onto the baseboard. Some connections to the power supply and UART are already made on the baseboard, but you will have to design certain elements of the system and make decisions about wiring and electrical connections.

You will also have to devise suitable tests to show that your system conforms to the mentioned requirements and specifications. Your design decisions and justification, along with test methods and test results should be documented in a final report, which is due at the end of this module.

Your system will be tested using automated demonstration stations, which will test whether your system meets these system specifications. It is thus important that you devise and perform similar tests to what the demo station will run, ***before*** you present your board for a demonstration.

The demonstration should not be used to substitute for your own testing since you will be penalised for multiple demonstrations.

The rest of this document describes the specifications in more detail, the hardware that you will be required to use and interface to (section 3), as well as software considerations (section 4), and finally the test methods that will be used to test your board (section 5).

2.1 UART communication

The system should use the UART port connected to the PC (**and the TIC for Demo purposes**) to report measurements and system statuses and to receive configuration commands. The full details of the communication protocol are given in section 5.3.

2.2 Application command interface

The system should implement an application command interface allowing the user to select the operating mode and modify system parameters. The command input interface shall be via the push buttons (UR7), UART, Slider, and Azoteq trackpad. The output interface shall be via the UART (UR5). The system's current state shall also be indicated using the four debug LEDs (UR8).

2.3 Azotech IQS7211A rectangular trackpad definitions

2.3.1 Slide motion for intensity setting

The "slide" motion is not defined as one of the trackpad gestures. Contrary to the left/right "swipe" gestures of the trackpad, which are fast movements with no position importance, the slide motion is a slower motion to adjust the intensity of the white LED upwards or downwards. The motion should be analogous to the slider motion.

See figure 2 for a visual example.

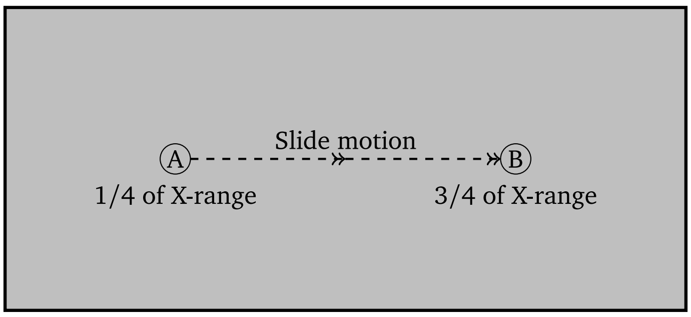


Figure 2: Trackpad example to increase the intensity of the white LED with a slide motion from left to right. The total distance travelled by the slide is 1/2 the trackpad, so intensity should increase by 1/2 the total range or +256 units.

2.3.2 RGB LED colour mapping

The coordinate system and colour mapping for colour settings is defined as:

- Bottom of the trackpad is 0 colour value
- Top of the trackpad is 512 colour value
- Values increase linearly from 0 at the bottom to 512 at the top
- Each colour is assigned a third of the trackpad width, from left to right: Red, Green, Blue

See figure 3 for a visual example, and note that the X-resolution and Y-resolution are set to 0x700 and 0x300 respectively using the calibration values for the iqs7211A. You have to convert the raw touch position values. For X-axis to select which 3rd of the touchpad you are touching. For Y-axis convert to the 0-512 colour value range.

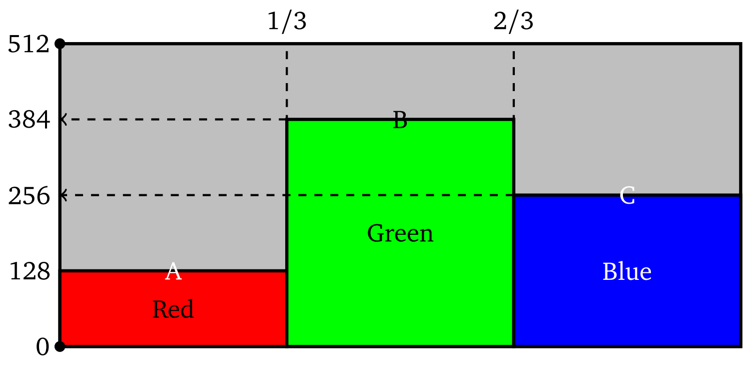


Figure 3: Trackpad positions relating to RGB colour values. In mood light mode, after points A, B, and C were pressed on the trackpad the colour should be R=128, G=384, B=256

3 Hardware

The STM32 microcontroller board that you will use is the STM32F303RE. It has a NUCLEO-64 form factor which allows it to stack through either the Arduino headers or the Morpho connectors (on top). The STM board will stack on top of the prototyping baseboard that you will use. To simplify the connection choices and possible solder problems, the following pre-determined design choices were made:

- The digital and analogue ground pins of the microcontroller board are hard-wired to the main board ground
- The NUCLEO-STM32F303RE is provided with main board power via the E5V pin (CN7-6) and F1 (fuse or link).
- To assist in building the system, some peripherals have pre-defined layout positions. These include:
  - 5 V power regulation circuit (3.3 V power regulation circuit has a predefined prototyping area where it should be built)
  - Debug / user LEDs
  - Debug / user push button switches
  - Test interface connector (used for Demonstrations)
  - Miscellaneous connectors that might be applicable to this project, eg. power socket, USB, screw terminal, audio jack.

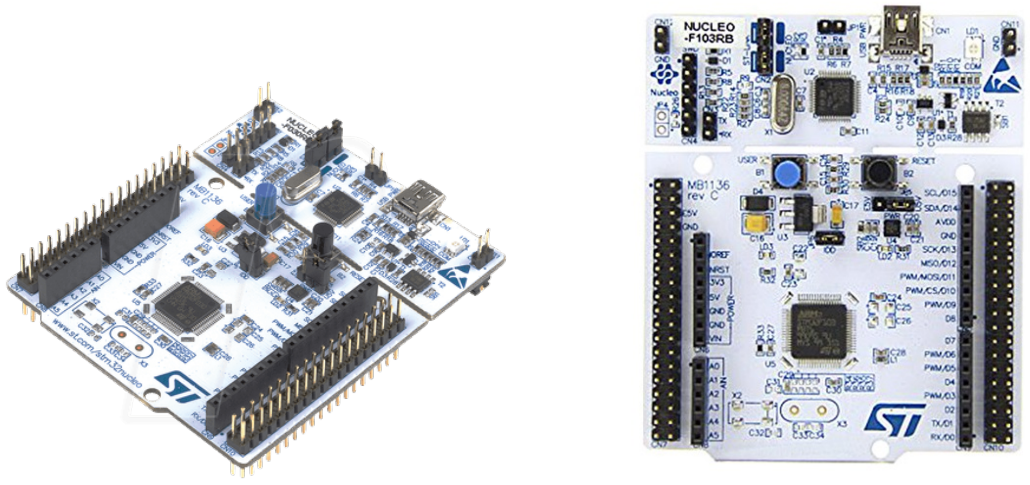


Figure 4: NUCLEO-F303RE development board [2]

3.1 Power Supply

The board must be supplied with a nominal 9 V Battery source (but you can use a bench power supply during development and testing, instead of a real battery). Your project will require you to implement both a 5 V and a 3.3 V power regulated supply. The 5 V circuit is pre-designed and shown in figure 5.

The 5 V power supply circuit on the baseboard uses a standard 7805 regulator (U1) in a TO220 package to regulate the input down to 5 V. The 5 V power is routed to 3 jumpers (J14, J16 and J25, each providing 3 pins for wiring). An LED (D6) circuit is provided to show that a voltage is present on the 5 V line.

You will have to determine the minimum supply voltage input required as well as the maximum allowable input voltage. You must also make sure that the regulator (U1) is thermally stable (by calculating the expected heat dissipation and adding thermal heat sinking hardware, if required). You must understand the role of each component in the power supply circuit.

You will have to design the 3.3 V supply circuit yourself using the LM2950 regulator, provided in a TO-92 package. Available on the baseboard, there is a 3.3 V power rail routed to 3 jumpers (J17, J18 and J26, each providing 3 pins for wiring).

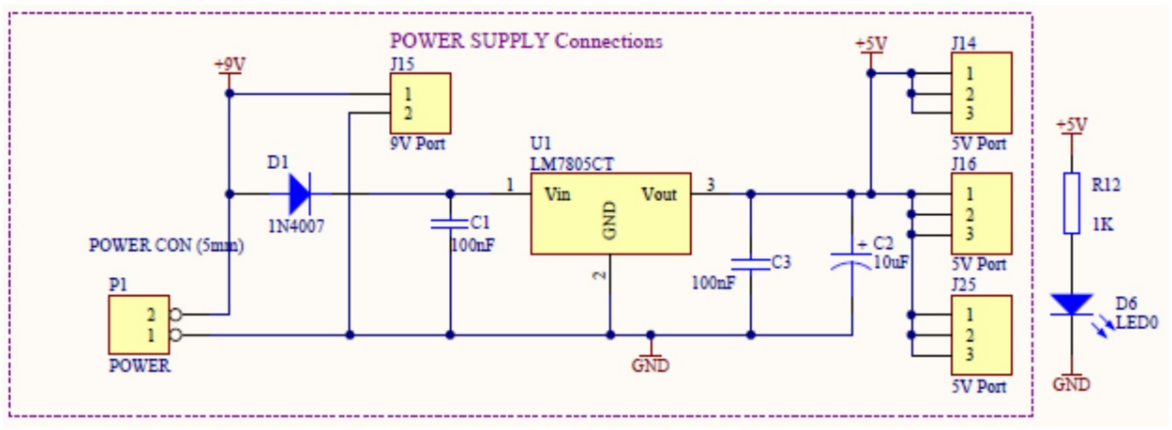
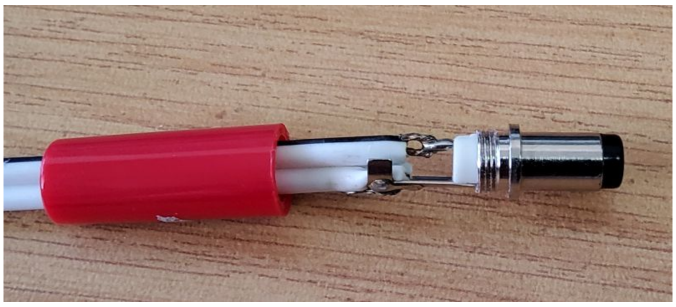


Figure 5: 5 V Power supply circuit diagram

⚠ *Note that the 3.3 V supply should NOT be connected to the NUCLEO-STM32F303RE's own 3.3 V regulated voltage!*

On your NUCLEO board, the `VIN` pin should be left floating. To select board power, move the JP5 jumper on the NUCLEO-STM32F303RE to the `E5V` (external power) position, from the `U5V` (USB power) position. Power your board using a bench power supply, set to 9 V, and connect the power to your board using the barrel jack and wire provided.


(a) Barrel jack with wires soldered

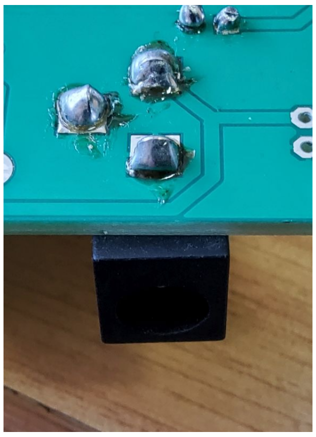

(b) Barrel socket soldered on to the PCB

Figure 6: Power connector solder connections.

⚠ *Do not power your baseboard and NUCLEO system using the USB power from the PC, as this could damage the USB port. It is safer to keep the jumper in the* `E5V` *position and use an external 9 V power supply. See more detail on this in "UM1724 User manual" PDF document, section 6.3.2 on page 21 - 22.*

ⓘ *Jumper J30 must be bridged on student boards to allow the test station to supply the student board with a nominal 9 V during demonstrations. Jumper J31 must be bridged to supply 9 V to the regulator.*

ⓘ *The fuse, F1, must be bridged on student boards to supply the STM32 board with 5 V.*

3.2 Push button inputs

Space for five push buttons are provided on the board. They are logically arranged to represent up, down, left, right and middle. You need to wire up the buttons to be active low, and read as inputs by the GPIO pins of the Nucleo board. They will also need to be connected to the TIC as per Table 6.

> ⓘ *These pins are also floating on the baseboard - they are not connected to any other signal. You need to wire them to the pins/signals you wish to use.*

> ⚠ *Remember to limit the current through the buttons by using a series resistor, otherwise you will damage your regulator circuit!*

3.3 Slider

A analogue slider has been provided. You are required to connect it to the ADC of the STM32F303RE board, and wire it up as a voltage divider. The output voltage of the slider must also be connected to the TIC.

> ⚠ *What voltage must the slider be connected to? 5 V or 3.3V? Careful! The wrong voltage can damage the MCU ADC input pins!*

> ⓘ *You will have to bend the four mounting tabs (out of the way) to insert the slider pins into the prototype area.*

3.4 Debug / user LEDs

Space for four LEDs plus series resistors are provided for display and/or debug purposes. You will have to calculate an appropriate value for the series resistors to limit the LED current to an acceptable level. These LEDs must be driven by four GPIO pins of the Nucleo board. The details of the state that is represented by each LED (or LED combination) is given in section 4.9.

> ⓘ *These pins are also floating on the baseboard - they are not connected to any other signal. You need to wire them to the pins/signals you wish to use.*

> ⚠ *Remember to limit the current through the LEDs by using a series resistor, otherwise you may damage the processor output pins!*

3.5 Azoteq Rectangular Trackpad

In addition to the buttons and UART, your system should be able to receive input commands from the Azoteq Rectangular Trackpad - IQS7211A. The hardware interface to the trackpad consists of six pins, as listed below. The trackpad uses two I2C communication lines, SDA and SCL, along with two digital lines, MCLR and RDY.

For details on the pinout of the trackpad, please see the schematic file: “iqs7211a_rectangle_trackpad_layout.pdf” on SUNLearn. The P1 header on the schematic is the female connector of the trackpad. Please note the indication of pin 1 of the header on the trackpad (see Figure 7).

A short description of each pin is listed here for convenience. For more details on each pin, please check the datasheets, guides, and example code provided on SUNLearn.

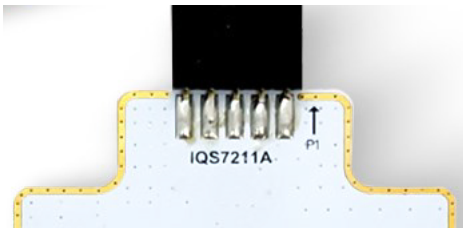


Figure 7: Indicator for pin 1 of the header

⚠ *Be careful to connect the VDD of the trackpad to the 3.3 V supply of your board. The trackpad will get damaged if connected to the 5 V supply as it exceeds its maximum input voltage!*

- 1 - GND = Ground pin, should be connected to your board ground.
- 3 - VDD = Supply pin, MUST be connected to your board 3.3V supply.
- 5 - MCLR = Reset pin (input for trackpad), can be driven low (to 0 V) to reset the trackpad. This pin can also be left floating as it has an internal pull-up resistor to bring the trackpad out of reset after power is supplied.
- 7 - SDA = I2C Data pin
- 9 - SCL = I2C Clock pin
- 10 - RDY = Output pin from trackpad to indicate new data is ready to be read. This pin can be read to know when to trigger a data read operation for your software. See the software section 4.12 for more details.

You should therefore connect the pins listed above to your selected NUCLEO pins. Please note that specific pairs of NUCLEO pins must be used together for I2C operation. If you use the peripheral configuration tool, these pins should be chosen by default for you by the tool, assuming they are not used already.

Section 4.12 provides information on how to develop software for the trackpad.

3.6 Output LEDs and drive circuit

You must drive your white LED and RGB LED from the MCU, either directly or via a transistor circuit. For the high-power white LED we will use the Cree P4 LED, and for the RGB LED we will use the Kingbright full-colour LED lamp. The datasheets for both devices are available on SUNLearn.

To provide enough current for the high-power LED you have to use a circuit to 'amplify' the current of the combined output circuit. For this, you should use the provided transistor (2N2222A) and a suitable current-limiting resistor. Since the forward voltage of the white LED is above 3.3 V you will need to provide the VCC of the transistor circuit from the 5 V power rail.

For the RGB LED, you do not require a high drive current and you can drive the 3x RGB LED channels directly from the GPIO pins using PWM. You will still have to add current limiting resistors, based on the forward voltage of each of the LED channels.

⚠ *When choosing resistors, the power rating of the resistor should also be considered. What happens if you put 1 A through a 100 Ω resistor? What is the power dissipated in the 100 Ω resistor?*

3.7 NUCLEO-STM32F303RE Connections

Table 2 gives some suggested pinouts, not all are required. The connections **that are not listed should be chosen by you** as part of your design solution. Many pins are restricted to a limited set of functionalities so consider all the pin requirements for the project when making your decision.

CPU Pin	**Port**	5V	**Connector-Pin**			**Proposed Function**
			Morpho	**Arduino**	**Baseboard Solder Pad**	
PA0	A	n	CN7-28	CN8-1	J10-1/2	
PA1	A	n	CN7-30	CN8-2	J10-3/4	
PA2	A	n	CN10-35	CN9-2	J9-13/14	UART2 TXD
PA3	A	n	CN10-37	CN9-1	J9-15/16	UART2 RXD
PA4	A	n	CN7-32	CN8-3	J10-5/6	
PA5	A	n	CN10-11	CN5-6	J11-9/10	LED2 on Nucleo
PA6	A	n	CN10-13	CN5-5	J11-11/12	
PA7	A	n	CN10-15	CN5-4	J11-13/14	
PA8	A	Y	CN10-23	CN9-8	J9-1/2	
PA9	A	Y	CN10-21	CN5-1	J11-19/20	
PA10	A	Y	CN10-33	CN9-3	J9-11/12	
PA11	A	Y	CN10-14		J5-13/14	
PA12	A	Y	CN10-12		J5-11/12	
PA13	A	Y	CN7-13		J4-13/14	
PA14	A	Y	CN7-15		J4-15/16	
PA15	A	Y	CN7-17		J4-17/18	
PB0	B	n	CN7-34	CN8-4	J10-7/8	
PB1	B	n	CN10-24		J5-23/24	
PB2	B	n	CN10-22		J5-21/22	
PB3	B	Y	CN10-31	CN9-4	J9-9/10	SWO (PC Debug)
PB4	B	Y	CN10-27	CN9-6	J9-5/6	
PB5	B	Y	CN10-29	CN9-5	J9-7/8	
PB6	B	Y	CN10-17	CN5-3	J11-15/16	**I2C1_SCL**
PB7	B	Y	CN7-21		J4-21/22	**I2C1_SDA**
PB8	B	Y	CN10-3	CN5-10	J11-1/2	
PB9	B	Y	CN10-5	CN5-9	J11-3/4	
PB10	B	n	CN10-25	CN9-7	J9-3/4	
PB11	B	n	CN10-18		J5-17/18	
PB12	B	n	CN10-16		J5-15/16	
PB13	B	n	CN10-30		J5-29/30	
PB14	B	n	CN10-28		J5-27/28	
PB15	B	n	CN10-26		J5-25/26	
PC0	C	n	CN7-38	CN8-6	J10-11/12	
PC1	C	n	CN7-36	CN8-5	J10-9/10	
PC2	C	n	CN7-35		J9-13/14	
PC3	C	n	CN7-37		J9-15/16	
PC4	C	n	CN10-34		J5-33/34	
PC5	C	n	CN10-6		J5-5/6	
PC6	C	Y	CN10-4		J5-3/4	
PC7	C	Y	CN10-19	CN5-2	J11-17/18	
PC8	C	Y	CN10-2		J5-1/2	
PC9	C	Y	CN10-1		solder pad	
PC10	C	Y	CN7-1		J4-1/2	
PC11	C	Y	CN7-2		solder pad	
PC12	C	Y	CN7-3		J4-3/4	
PC13	C	n	CN7-23		J4-23/24	Blue Pushbutton
PC14	C	n	CN7-25		J4-25/26	32 kHz Xtal
PC15	C	n	CN7-27		J4-27/28	32 kHz Xtal
PD2	D	Y	CN7-4		solder pad	
PF0	F	Y	CN7-29		J4-29/30	HE Clk 8 MHz
PF1	F	Y	CN7-31		J4-31/32	

Table 2: NUCLEO-STM32F303RE to Baseboard connections

The NUCLEO-STM32F303RE plugs into the baseboard through two separate 38-pin double row connectors, CN7 (left) and CN10 (right). These connections correspond to the NUCLEO Morpho connectors. The outer row connections are linked to solder connections J4 (for the odd numbered CN7 pins) and J5 (for the even numbered CN10 pins). The majority of the inner row CN7 (even numbered) pins that correspond to Arduino connections are brought out through J10, and the odd-numbered CN10 pins are brought out through J9 and J11.

It is good practice to trace the PCB connections on your board and familiarise yourself with the connections made via the PCB and therefore the interfaces available to you.

Table 2 provides a cross-reference to the signals available on the Arduino, Morfo and baseboard connectors. This table also contains some proposed and pre-determined signals to enable the use of UART and I2C communications. More detail on this will follow later in the course.

The 5 V compatible pins are also listed — the rest of the pins are only 3.3 V compatible and need protection if connected to a 5 V source.

You will need to refer to these pin connections whenever you need to decide on which pin to use for a specific function. Some peripherals have only one or two options, so where you have the design freedom, make sure you plan for future possible pin uses too.

DO NOT CONNECT the following pins of the NUCLEO-STM32F303RE, in your application unless you have a specific requirement - `BOOT`, `5V`, `RESET`, `VIN` *and any of the* `NC` *pins. The NUCLEO-STM32F303RE operates normally without connections to these pins.*

3.8 Test-interface Connector

A test-interface connector (TIC) is provided by P13. This connector is designed to be stackable and consists of an Amphenol Bergstik 16-pin surface mounted connector soldered to the bottom of the baseboard (solder side - see Figure 8). This connector mates with a 16-pin Amphenol Dubox connector on the Automated Test board (top side) that will be used during demonstrations. The connections provide a power source (9 V), UART and 10 other connections during testing of the system.

This connector is supported by two solder connectors (J3 and J13) to connect to various ports or circuits.

More detail about the TIC, and its complete pin descriptions are given in Section 5.

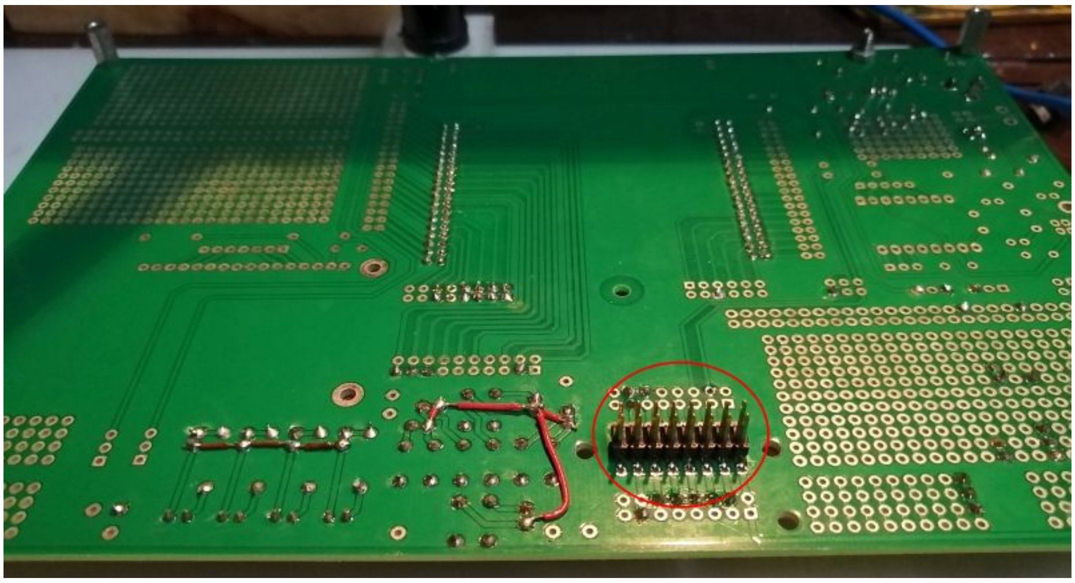


Figure 8: The TIC shown on the solder side of the baseboard.

4 Software Design and Considerations

4.1 System operational overview

As the system operates and receive user inputs (via buttons, UART, and the Azoteq trackpad) it should change to a predefined set of display states that indicate the current system information.

Menu operation is further detailed in section 4.8. and should specifically implement the functionality described.

The default start-up values of the system will be as follows:

- The operational mode shall be Flashlight with the white LED in the OFF state.
- The default emergency mode (when switched to) shall be the Strobe mode with the white LED in the OFF state.
- The default strobe "on" time shall be 512 ms, when the Strobe is switched to ON state.
- The custom Morse code message shall be set to "SOS" when switched to, if no UART message has been received to change the custom Morse code message.
- The default mood light mode (when switched to) shall be with each channel in the OFF state.
- The default mood light RGB channel intensities shall be set to 128 when switched to ON state.

4.2 General software information

To develop software for this project, a similar environment and typical development process as used in Computer Systems 245 will be used: STMCubeIDE v1.11. There is however one significant change: The use of a software version control system: GIT, described in more detail in section 4.3.

⚠ *The use of STM32CubeIDE v1.11, with F3 Firmware package 1.11.3 is NOT optional. We cannot offer support for other versions.*

ⓘ *In any of these project generator software set-ups, the user must be cautious. The software typically use comment delimiters to allow user code to be added, but this system is prone to deleting user code if the project is regenerated (adding new I/O for instance). Our advice is to add the minimum code in the project generated files and to use the project generator sparingly (ideally once only when generating the project for the first time). We recommend that you add files, with calls originating from the main C-file, which contain your user created code.*

For a small project such as this, adding the following minimum files will simplify your development:

- Header file with all global definitions and function prototypes;
- Source file with all your global variables;
- Source file with initialisation function and user code, which leaves the main while loop short. Call a run function from main to execute your user code.
- Additional files to handle each mode, inputs (trackpad, slider, and buttons), and LED driver functions are recommended.

4.3 Using GIT for Design E314

GIT is a form of source code version control. It not only keeps backup of your code, but also allows you to easily see changes in the code between checked-in versions, and facilitates collaboration on source code projects (although we will not use this latter functionality). Having knowledge of GIT and its processes is beneficial since it is used almost everywhere in industry where source code is part of the organization's Intellectual Property (IP). You will be ***required*** to use GIT and have your demo code (a separate version committed for each demo) in your GIT repository ***before*** you do your demonstration.

ⓘ *Please refer to the GIT document on SunLearn for further detail on how to setup and use GIT in this course.*

4.4 Design of your Software Structure - Functions and Loops

The next question the designer (you) will encounter is what software structure must be chosen. The suitable choices depend on the requirements at system level and the peripheral response times.

Lets take an example approach:

The designer decides to put all the software in one big main loop with each function being executed sequentially before repeating indefinitely. What could go wrong?

Let us explore the possible problems by asking some questions about the software behaviour:

- How fast will the system respond to a UART command received? Will it consistently respond in this time?
- How fast will the system be able to sample digital inputs?
- How fast will the system be able to sample ADC inputs?
- How fast will the system respond to I2C inputs from the sensors?
- How long will the system take to do all the required calculations?
- How long will the system take to change the driver output parameters?
- How fast does the system NEED to do all the above, based on the specifications?
- Do all of these inputs/outputs require the same periodic attention from the system?
- How much resources will be consumed by the code? RAM, ROM and stack?

There is also a second, more subtle, reason for making a good software structure choice at the start of the project: How will any code additions or changes in sub functions affect the rest of the code? What you don't want is to have to rewrite a big part of your main loop (and maybe some other functions too), to accommodate a new function's requirements.

ⓘ *Modularity and well defined interfaces are key to keep the reworking of code to a minimum.*

In terms of the system design, we may attempt one of the following three approaches:

- The novice software writer will not consider a synchronous time-based structure for his code. Although it is feasible just to string all the functions together into one big while loop, the responsiveness of the system will be determined by the slowest function and recovery from time-outs and other errors are non-trivial to maintain.
- By using a timer-based schedule inside the while loop with interrupts can make the software much more robust. The choice of a tick-update period equal to $2\times$ to $5\times$ the largest response delay time will simplify the code (otherwise a state machine and elapsed timer will also be required).
- The use of a real-time operating system (RTOS), such as FreeRTOS, will simplify the structure further, but add some overheads. It also has a steep learning curve. In the simple case of each thread having the same priority, the behaviour will resemble a large while loop (equal priority from interrupt but with pre-defined execution order).

The large while loop (second option above), reacting to interrupt flags, is the preferred option for this project and learning phase.

4.5 HAL vs. LL Functions

A programmer may choose to write code at register level, or may want the abstraction that the HAL (Hardware Abstraction Layer) provide. The CubeMX initialisation generator will generate code for either the HAL or the LL (Low Level) libraries, but not register-level code directly. By using the CubeMX generator, the student can only choose LL or HAL code (per peripheral), and with the documentation bias towards HAL code, it is likely that most users will start off with HAL code.

The majority of tasks for this type of project can be written using the HAL functions only. If required, direct register access is still available using the STM32F303 header file.

Interrupts are accessible by using calls with IT-ending (e.g. `HAL_UART_Receive_IT()`) and a default interrupt handler for every interrupt is generated (e.g. `USART1_IRQHandler()` in the `stm32f3xx_it.c` file). You can process the flags to determine the cause of the interrupt in this file, but the recommended call-back function (e.g. `HAL_UART_RxCpltCallback()`) is a better option.

The supplied HAL library has a number of drawbacks and/or bugs. There are significant bugs in the I2C interrupt handler — what we have identified are:

- No checking for zero condition in the transfer (which may cause additional characters to be transferred);
- The restart is conditioned by write and read direction requests. The approach works for alternating directions but does not work for repeated writes;

The effect is that the only call allowing repeated writes (utilising interrupts) fails during parameter uploading during initialisation but work fine inside the main while loop while running.

A secondary effect is caused by the action that both the I2C and UART interrupt handler disables interrupts before returning — no problem if you are using deferred interrupt handling (setting a flag in the call-back and processing it in the main while loop) but back-to-back transfers through the call-back will not work.

4.6 Regular Interval Timing

The core ARM processor provides a Systick timer as standard, which is set in the HAL library at 1 msec intervals, so there is no need to provide any additional timers for this simple purpose. A call-back from the Systick timer will provide a 1 millisecond heartbeat. This is located in the STM32Flxx_it.c file.

```
        void SysTick_Handler(void)
        {
            /* USER CODE BEGIN SysTick_IRQn 0 */
            /* USER CODE END SysTick_IRQn 0 */
            HAL_IncTick();
            /* USER CODE BEGIN SysTick_IRQn 1 */
            /* USER CODE END SysTick_IRQn 1 */
        }

```

However in some cases you need to generate shorter duration timer delays, functions that might need sub microsecond accuracy. Hardware timers should be used in such cases, and you will have to add your own code to the timer interrupt vectors and create your own functions to implement these delays. For example you would want to call such a delay function:

```
        // Wait for 5 microseconds
        myUsecDelay(5); // call self-created u-sec accurate delay function

```

4.7 ADC Conversions and measurements

4.7.1 Conversion

The ADC is quite fast (worst case 2-3 microseconds conversion), so for undemanding applications, there is no need to really use either multiple channel conversion scan chains or even simultaneous sampling schemes. A simple single channel conversion using polling can be sufficient. In this project your device will need to measure the input voltage using one channel.

The ADC can operate with as low as 1.5 clock cycle sampling for regular channels (if your input impedance is sufficiently low).

To convert a single channel, using the HAL library, we need to set up the channel, start the conversion, wait for completion and convert the result into the required scaled value.

You can also use the ADC in DMA mode to transfer the data directly to memory, bypassing the CPU. This is not required but can be useful when having to sample many times in succession.

4.7.2 Voltage measurements

You will use the internal ADC to measure the voltage from the slider as input. For this application, timing is less important, however, an accurate measurement is important as you want to set the intensity accurately.

The input voltage should be accurately measured to within 5% accuracy. Calibration can assist you greatly in achieving good accuracy. You can calibrate your system by comparing its measurements to laboratory instrument measurements, like an oscilloscope measurement. Good calibration can be achieved if the signal is compared over the full range of the input and a correction factor is implemented in software to compensate for any non-ideal system response.

4.8 Command operation

The system will implement a menu system that is controllable by using three of the five push buttons as well as the UART input. Such a menu system is usually implemented using a software state machine. The menu system required by this project is defined below:

- Use the LEFT button to cycle through the main operating modes in the following sequence: Flashlight, Emergency, Mood, Flashlight, ...
- When in Emergency mode:
  - Use the RIGHT button to cycle through the emergency modes in the following sequence: Strobe, SOS, custom message, Strobe, ...

When the system receives a command via the UART, it should directly change to the requested mode and state while immediately applying the provided parameter values. I.e. if the system is in Mood light mode and receives a UART command to change to Emergency Strobe mode with an “on” time of 256 ms, it should do so immediately and display a strobe light with an “on”.

4.9 Debug LEDS

The debug LEDs will be used as system state indicators as follows:

- D2 - Indicates the system is in flashlight mode (D2 = ON)
- D3 - Indicates the system is in emergency mode (D3 = ON)
- D4 - Indicates the system is in mood light mode (D4 = ON)
- D5 - Only for emergency mode: it indicates when Morse-coded messages are being flashed (D5 = ON)

The LED indicators are especially useful in the absence of an LCD screen

4.10 UART

The UART shall operate at a baud rate of 57600 bps, with 8 data bits, 1 parity bit, and two stop bits, using TTL signal levels. The parity used will be even parity.

UART messages make use of ASCII printable characters, to make it easy to interface with the student board debug connector using simple terminal programs, such as Termite, Realterm, and Teraterm in the absence of a test station.

The format of the UART receive HAL functions is slightly at odds with how UART communication normally occurs, the HAL functions are geared towards known fixed length packets, while in reality, we use variable length packets. To use the HAL receive function, we can set it up for single-byte mode using interrupts (in effect to prime the receiver before the byte arrives), and as soon as we received a byte we can re-prime the receiver.

For UART transmission we can use standard block transmits.

4.10.1 UART Protocol

The SB UART will be used to send and receive messages to and from the PC (TS).

The SB will output messages to the PC(TS) with the formats as in Table 3. Each message starts with a '#' character and ends with a '$' character followed by a newline character (ASCII character code 10, or indicated in C string escape notation as '\n'). Each field in the message is separated by a ':'. Except for the *StdNum* message that identifies the student board, all the transmit messages are in response to a message request received from the PC (TS).

For Demo 1 (see section 5.3): For the UART part, you will need to send the student number, using the correct message format, to the TS. Then the UART protocol will deviate for testing purposes. After sending the student number message you must echo each received character back to the TS and respond with a button press counter (range '000' up to '999') after each middle button press. The counter must be terminated by a '\n' character.

The UART will be used to receive configuration and command messages from the PC(or TS) to the SB. The SB will interpret a message following the formats as shown in Table 4. A configuration/command message starts with a '#' character and ends with a '$' character followed by a newline character (ASCII character code 10, or indicated in C string escape notation as '\n'). Each field in the message is separated by a ':'.

For both transmit and receive messages the following options are defined:

- <Mode> values can be 'MF' (flashlight), 'ME' (emergency), or 'MM' (mood)
- <State> meaning depends on the current system <Mode>
  - in 'MF' it is the intensity of the white LED in value ranging from '001' to '512' ('000' if white LED is off)
  - in 'ME' it is the intensity of the white LED in value ranging from '001' to '512' ('000' if white LED is off)
  - in 'MM' it is the intensity of the R channel of the RGB LED in value ranging from '000' to '512'
- <Param1> meaning depends on the current system <Mode>
  - in 'MF' it is always '000'.
  - in 'ME' it is the "on" time of the strobe light in milliseconds ranging from '001' to '512' ('000' means it displays the Morse code message)
  - in 'MM' it is the intensity of the G channel of the RGB LED in value ranging from '000' to '512'
- <Param2> meaning depends on the current system <Mode>
  - in 'MF' it is always '000'.
  - in 'ME' it is the three-letter custom Morse code message (default return 'SOS', when setting the value by sending a '000' sets the sub-mode to default S.O.S mode)
  - in 'MM' it is the intensity of the B channel of the RGB LED in value ranging from '000' to '512'

Some example RX messages:

Field in bytes	1	8	2
StdNum	‘#’	Student number	‘$\n’

Field in bytes	1	2	3	3	3	2
Status	‘#’	<Mode>	<State>	<Param1>	<Param2>	‘$\n’

Table 3: UART TX Message fields - Log from SB to PC(or TS)

Field in bytes	1	2	2
Request mode status	‘#’	<Mode>	‘$\n’

Field in bytes	1	2	3	3	3	2
Set mode command	‘#’	<Mode>	<State>	<Param1>	<Param2>	‘$\n’

Table 4: UART TX Message fields - Configuration/Command from PC (or TS) to SB

- Start-up message for student 12345678: `#:12345678:$(newline)`
- Status message in MF mode, with white LED on half intensity: `#:MF:256:000:000:$(newline)`

Some example TX messages:

- Set mode to flashlight, 25% intensity: `#:MF:128:000:000:$(newline)`
- Set mode to ME mode, full int, strobe with 256ms on time: `#:ME:512:256:SOS:$(newline)`
- Set mode to ME mode, 50% int, SOS/Morse message “YES”: `#:ME:256:000:YES:$(newline)`

Letter	Symbols	Letter	Symbols	Letter/Number	Symbols	Number	Symbols
A	. -	K	- . -	U	. . -	1	. - - - -
B	- . . .	L	. - . .	V	. . . -	2	. . - - -
C	- . - .	M	- -	W	. - -	3	. . . - -
D	- . .	N	- .	X	- . . -	4	. . . . -
E	.	O	- - -	Y	- . - -	5	. . . . .
F	. . - .	P	. - - .	Z	- - . .	6	- . . . .
G	- - .	Q	- - . -			7	- - . . .
H	. . . .	R	. - .			8	- - - . .
I	. .	S	. . .			9	- - - - .
J	. - - -	T	-			0	- - - - -

Table 5: International Morse Code

The timing for the different symbols and spaces for International Morse Code is:

- Dot is ONE time unit
- Dash is THREE time units
- Space between parts in same letter/number is ONE time unit
- Space between letters/numbers is THREE time units
- Space between words is SEVEN time units

4.11 I2C Communications

This section is a generic overview of I2C software and low-level drivers. More specifics for the Azoteq trackpad are given in section 4.12.

The I2C communications on the STM32 family is well supported by the HAL libraries, which remove most of the complexity from the user. The STM32F303 HAL libraries provide a variety of functions for reading from and writing to I2C devices, including calls that allow repeated start, interrupt and even DMA transfers.

Even with all the functions provided, not all the calls required are provided (no blocking-type of calls with repeated start) and there can be bugs. The use of interrupt calls provide very little benefit in this application.

The writing and debugging of the functions will be daunting to new users, and the following strategy is proposed:

1. Start off with a single transfer — preferable a memory read (which is inherently an address write-repeated start-read transaction). The ideal memory location to read is the product identification register(s).
2. Develop an I2C write call (with repeated start, using the HAL blocking write call as basis), write to one of the configuration parameter registers and use the memory read listed above to read the written info back (to make sure the the write was successful).
3. Then develop an I2C read call (current address and with repeated start) and test as above.
4. Start to string commands together, and always make sure that you try to read the info back for checking purposes.
5. Use a oscilloscope, a low value on the SCL and/or SDA lines will quickly indicate a misplaced STOP token (it is more challenging to try and figure it out from the software only).

4.12 Azoteq IQS7211A software and drivers

To assist in developing software for this device, a software example for an Arduino board is provided by the manufacturer and can be downloaded from SUNLearn. Similar to what you would encounter when working in the industry, you will have to port (re-write for the NUCLEO) the software example. You do not have to re-write all the code. Focus on incremental development and rewrite only the sections of the code you require for your project. Some of the header files can be used with little (if any) modifications.

The Azoteq IQS7211A device is quite complex and has many configuration registers. The register values can be tuned for optimal operation. However, you should use the provided header files that have a pre-determined set of values that should work well enough for the purposes of this project. Note that a newer version of the configuration header file has been added to the library - "IQS7211A_init_AZP1189A3_v0.1.h". Make sure to use this file as it matches the updated hardware of the trackpad you received.

It is strongly encouraged that you start by reading the "IQS7211A Arduino Example Code Guide.pdf" and then the "README.md" file in the /src/IQS7211A/ subfolder of the example code, to guide and assist you in developing the code for your project.

The communication interface of the trackpad with your NUCLEO board is I2C (also see section 4.11) and you will have to use the HAL I2C driver functions to realise I2C reading and writing functionality in your project.

Note that the HAL libraries do not allow for a "restart" condition at the end of the total data transfer and will generate a "stop" condition. The "stop" condition will end the communication window by pulling RDY high again. Since RDY is on an open-drain connection, the speed at which it is pulled high depends on the pull-up resistors on both the trackpad PCB and the STM32 internal pull-up resistor (if enabled). Make sure to NOT start a communication transaction (read or write) immediately after a previous transaction. On our board the RDY line took 6-8us to get pulled high.

After complete initialization, the trackpad should be in "streaming" mode, meaning it will sample data on any event input, with the frequency of updates dictated by the power mode it is in (see section 6.1 in the IQS7211A datasheet). The device is set up to change from active mode through the lower power modes to LP2 mode if there is no input detected.

The trackpad will indicate that newly sampled data is available with the RDY line going low. This should trigger a read action in your code (remember a "stop" condition at the end of your I2C transaction will cause this communication window to end). The RDY will go high when the communication window ends (if you do not read it, the device will timeout - see IQS7211A_MM_RDY_TIMEOUT register values - and end that particular communication window).

The IQS7211A_MM_GESTURES register will hold the most recent detected gesture information. By just identifying which gesture took place, it should be enough for your system to respond to that input. For slide motion and position press, your code should sample enough information to allow intensity changes and colour values to be read. Some useful information can be found in registers 0x12 to 0x1B for these purposes.

> *To mitigate the no-"restart" condition issue with the HAL drivers, you can read a set of sequential register values at the same time. As an example, by reading 6 bytes starting at register 0x00 (IQS7211A_MM_PROD_NUM), you will be reading all the contents of the first three registers which hold two bytes each.*

It is highly recommended that you use the **HAL_I2C_Mem_Read** and **HAL_I2C_Mem_Write** functions from the HAL-library, but you are free to try it your own way too.

If your device does not get initialized correctly, even though your write all the values, check that the RDY line is low for each transaction. If any transaction starts while RDY line is high it will not be handle correctly and the values will not be written/read correctly. If you read a 0xeeee or 0xffff for a register where you expect other values, it is most probably an RDY timing issue.

4.13 LED drivers

4.13.1 DAC

Although we will not use the DAC to drive the white LED, the DAC channel must still be active and provide a linear voltage for the test station to monitor via the TIC

4.13.2 PWM

You will use the PWM functionality of the MCU to drive the white LED through the transistor amplification circuit. The three channels of the RGB LED are driven directly using PWM. Only the white LED PWM channel will be monitored by the test station via the TIC. The RGB PWM signals are verified visually by monitoring the colour change of the RGB LED.

4.14 Main While Loop Statistics

Suppose we want to find out how long our loop times are for the main while loop. One way is to set up a timer (example of timer 7 here) with a prescaler that divides the 64 MHz clock frequency down to 1 microsecond intervals. The following code fragment illustrates how to do this:

```
while (1){
// Prepare timer 7 to get execution time statistics
__HAL_TIM_SET_COUNTER(&htim7, 0); // Reset the counter
HAL_TIM_Base_Start(&htim7); // Start Timer7 to get cycle time in usec

userProcess(); // Run the user process (your own cyclic program)

// Gather execution time statistics
HAL_TIM_Base_Stop(&htim7);        // Stop Timer7
elapsedTime = __HAL_TIM_GET_COUNTER(&htim7);

if (elapsedTime > maxElapsedTime)        // Update the maximum stats
maxElapsedTime = elapsedTime;
if (elapsedTime < minElapsedTime)        // Update the minimum stats
```

```
    minElapsedTime = elapsedTime;
    // Average stats:  Discrete IIR filter with 1/100 bandwidth
    aveElapsedTime  = 0.99 * aveElapsedTime + 0.01 * elapsedTime;

    __WFI();        // Wait for the next interrupt
    }
```

This code will provide minimum, maximum and average cycle times (to give you an indication of the processor loading). With the regular Systick interrupt occurring every millisecond, the example of an average elapsed time of 20 microseconds indicate a 2% average loading.

> ⚠ *To use the timer in this mode, load the period value with a large value (such as* `0xFFFF`*) as it will count up, if you leave it at the default of zero then the timer will not count!*

4.15 Auto code generation

The STM IDE provides functionality to auto-generate code. We advise you to be cautious as if you auto-generate code half way through a project, it can delete some of your code. Make sure all your code is between the correct comments if you plan to use this feature.

> ⚠ *We recommend only doing this once at the start of your project! Back up your code frequently to avoid loss of code!*

5 Testing

Your student board (SB) will be tested during demo sessions, as specified in the module framework. The testing will occur in an automated fashion. Your SB will be plugged in to a test station (TS). The TS will provide power to your SB and generate a number of signals that are connected to your SB. Certain signals from your SB will also be monitored and checked by the TS.

5.1 Test Interface connector (TIC)

In order to facilitate testing, a test-interface connector (TIC) is provided by P13. This connector is designed to be stackable and consists of an Amphenol Bergstik 16-pin surface mounted connector soldered to the bottom of the baseboard (solder side). This connector mates with a 16-pin Amphenol Dubox connector on the Test-station board (top side) that will be used during demonstrations. The connections provide a power source (9 V), UART and 10 other connections during testing of the system.

The signal connections to the TIC is detailed in Table 6, and the pin numbering of J13, J3, and P13 are as shown in Figure 9.

From your board, you will have to route the signals listed in Table 6 to the test connector, P13, by making use of the solder connectors J3 and J13. The solder pads on J3 and J13 are routed to pins on the TIC (P13).

You will be required to bridge the jumper at J30 to ensure your board receives the 9V from the TIC.

P13 pin connection	J3/J13 solder connection	Signal	Direction
1	J3 - 1,2	`V_bat` - 9 V Supply from test station	`SB <-> TS`
2	N/C	`V_bat` - 9 V Supply from test station	`SB <-> TS`
3	J3 - 3,4	Analogue signal from SB, DAC output (via added signal conditioning)	`SB -> TS`
4	J13 - 3,4	5V supply from student board (fed back to test station for verification/measurement)	`SB -> TS`
5	J3 - 5,6	UART transmit (TX from student board, RX of test station)	`SB -> TS`
6	J13 - 5,6	3.3 V supply from student board (fed back to test station for verification/measurement)	`SB -> TS`
7	J3 - 7,8	UART receive (RX from student board, TX of test station)	`SB <- TS`
8	J13 - 7,8	Analogue signal to SB, ADC input (via added signal conditioning)	`SB <- TS`
9	J3 - 9,10	Status LED - D2 (connected to TS for verification/measurement)	`SB -> TS`
10	J13 - 9,10	Analogue signal from SB, PWM output (direct from pin)	`SB -> TS`
11	J3 - 11,12	Button left (connected to TS for control/verification/measurement)	`SB <-> TS`
12	J13 - 11,12	Analogue signal from SB, OpAmp output	`SB -> TS`
13	J3 - 13,14	Button middle (connected to TS for control/verification/measurement)	`SB <-> TS`
14	J13 - 13,14	Button right (connected to TS for control/verification/measurement)	`SB <-> TS`
15	N/C	GND	`SB <-> TS`
16	N/C	GND	`SB <-> TS`

Table 6: Test-interface Connector Pin definitions

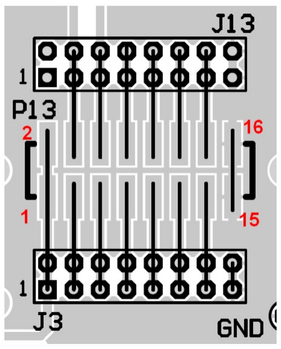


Figure 9: Pin numbers of J13, J3 and P13

Connections shown in grey in Table 6 are signals that are already routed on the PCB - you do not have to do anything to connect them.

5.2 Test method during demonstrations

The tests that the test station will execute once a student board has been plugged in, are designed to verify the requirements of the student board (detailed in Section 2). These requirements, and the method in which the test station will perform the test are listed in Table 7.

5.3 UART communications interface

To facilitate testing (both by the student and for automated tests) and debugging of the device, a UART link shall be implemented. The UART link shall operate ***in both directions***: transmitted from the student board and received by the test station or PC test program. The student board shall make use of the default UART2 channel on the STM Nucleo - This UART channel is connected to the ST-Link chip on the Nucleo module, and will automatically enumerate as a virtual COM port when the Nucleo board is connected to the PC via the USB debug cable. Thus, for debugging it will not be necessary to make any hardware connections.

For test station purposes, it will be necessary to route the UART2 to the Test Interface Connector (TIC). In this case, you will have to make a connection from the correct pins of the Nucleo board (RX/TX) to the Test Interface Connector (TIC) (see Section 3.7 and 5.1). Both methods are shown in Figure 10.

Remember that the RX/TX indicators are as seen from the ST-LINK perspective! If you are unsure, first check the direction with an oscilloscope, while sending messages from the PC, before connecting the wires to the TIC pins.

5.4 Demos

There will be a total of 4 demos which will test all the user requirements. The details of each demo are presented in Table 8.

UR	Method	Pass/Fail
UR1	The TS supplies board with 9V switched power supply. SB will feed back the generated 5V and 3.3V supplies to TIC connector.	Test passes if TS measures the 5V and 3.3V supplies to be in 5% of expected values.
UR2	The TS provide input to the SB via push-buttons, UART, and/or a user provides input via the slider or trackpad to switch the white LED on/off and adjust the white LED intensity.	Test passes if the system responds by setting the DAC and/or PWM signals to the correct levels. Output voltages from the DAC, PWM, and driver circuit of the white LED, must be accurate to within 5% of the required levels.
UR3	The TS provide input to the SB via push-buttons, UART, and/or a user provides input via the slider or trackpad to adjust the white LED strobe frequency, select emergency mode, flash the white LED to display messages in Morse code.	Test passes if the system responds by flashing the white LED to within 5% timing accuracy for each mode/setting and if the correct Morse code sequences are generated.
UR4	The TS provide input to the SB via UART, and/or a user provides input via the trackpad to switch the RGB LED on/off and adjust the RGB colour of the LED.	The test passes if the system responds by setting the RGB colour of the LED correctly.
UR5.1	The TS will receive UART messages from the SB. The TS will compare received UART messages to the expected values.	Test passes if the expected UART messages are received correctly by the TS.
UR5.2	The TS will send UART configuration/command messages to the SB. The TS will compare the SB response (via TIC/UART/Visual) to the expected responses/values.	Test passes if the SB responds in the correct way, both in function and in UART messages, to the configuration/command messages.
UR6	The user will input commands using the Azoteq trackpad's "press-and-hold", and "tap" gestures, slide motion, and position press.	Test passes if the correct events and inputs are reported via UART and if the system responds accordingly.
UR7	TS will override button signals on SB via TIC.	Test passes if buttons are successfully debounced and if the correct action occurs on the SB due to the button press.
UR8	The TS will send UART configuration/command messages to the SB and/or override buttons while monitoring the status LED(s).	Test passes if the status LEDs match the expected system states.
UR9	The SB will receive inputs from the TS and manually via buttons, UART, and trackpad.	Tests pass if the system drives the white LED and RGB LED to their required intensities/flash frequency/colours. The intensity and frequency of the white LED will be verified via the test station monitoring the TIC for the correct DAC, PWM outputs. The RGB PWM channels will be visually monitored to produce the correct colours.

Table 7: Test station test method definitions

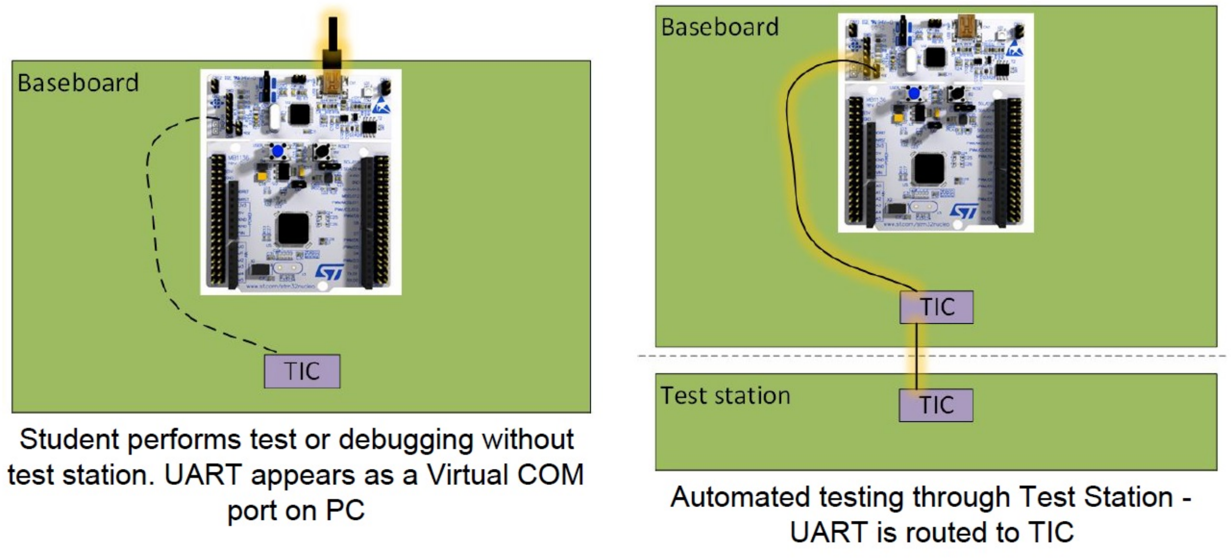

Baseboard
TIC
Student performs test or debugging without test station. UART appears as a Virtual COM port on PC
Baseboard
TIC
Test station
TIC
Automated testing through Test Station - UART is routed to TIC

Figure 10: UART connections for debugging and testing.

Demo	Description	UR's
1	On power-up, transmit your student number via UART using the prescribed message format, no sooner than 100ms after power-up and no later than 500ms after power-up. The TS will monitor and test the 3.3V and 5V lines. The TS will send a series of single characters via UART to the SB, with 500 ms delay between transmissions, and the SB must echo back each character on the UART immediately after receiving it. The TS will override the middle button and press it several times with and without bounce. Each time the middle button is pressed the SB must respond with the number of button presses received since power-up.	UR1, UR5.1 (student number, echo, and counter), UR5.2 (only echo), UR7 (de-bounce)
2	On power-up, transmit your student number via UART using the prescribed message format, no sooner than 100ms after power-up and no later than 500ms after power-up. The TS will monitor and test the 3.3V and 5V lines. The TS will override the left, middle, and right buttons (via TIC P13 pins 11, 13, 14) while monitoring the UART, DAC, and PWM outputs (via TIC P13 pins 5, 7, 3, 10, 12) to check for correct system operation. The TS will override the Slider (via TIC P13 pin 8) while monitoring the system state via UART responses received after the status request command to check for correct ADC scaling and calibration. The minimum delay between consecutive UART commands will be 500 ms. The system should update and respond IMMEDIATELY (<100ms) to any input. The slider will be manually adjusted and the system response monitored via UART to check for correct Slider functioning.	UR1, UR2 (no LED), UR3 (no LED), UR4 (no LED), UR5 (no visual), UR7 (no debug LEDs)
3	Same startup procedure as for Demo 2, except the test station will wait 2 seconds before timeout on receiving of Student Number to allow time for the trakpad initialization. Same electrical interface on TIC as Demo 2 with the exception that pin 8 is now an INPUT to the TS (you do not have to change any connection other that have the ADC input of your board connected to the TIC. **The TS will test the SB via UART interface** in both sending commands settings and receiving status reports. The button inputs will be tested as per Demo 2. The trackpad as input will be tested with manual presses/slides and the system should respond according to the URs. See the released marking rubric on SUNLearn for a more detailed breakdown.	UR1, UR2 (no LED), UR3 (no LED), UR4 (no LED), UR5 (no visual), UR6 (no LED), UR7 (no debug LEDs).
4	**Exactly the same assessment as demo 3** with the following **added**: The white LED will be visually (+ photo) verified for correct intensity change and the use of the transistor circuit. The RGB LED will be visually (+ photo) verified for the correct colour changes. The debug LEDs will be visually (+ photo) verified for the correct state indication. Additional manual inputs may be done via the slider, trackpad, and buttons to verify their responsiveness along with a system stress test at the end.	All of UR1 to UR9.

Table 8: Demo descriptions

6 Preliminary Report

Consistently documenting your work is critical in all engineering projects.

The goal of the preliminary report is to give you exercise and feedback in writing parts of a technical report. You will mark your peers' reports which should give you further insight into what is expected and some examples of interpretations of the tasks.

In the task descriptions that follow, the page limits refer to the total content length for that task excluding all cover pages, content lists, and references. Also provided, is a rubric to guide you in creating the report content. There are formatted templates for you to use on SUN Learn, in both MS Word and LaTeX formats.

6.1 Task 1

(maximum 2 pages)

To get used to the design process detail, you have to report on your design of the LED (D2) debug LED circuit and the middle push-button hardware.

Document your effort by drawing circuit diagrams (schematic diagrams) and include the necessary explanations for your design values in a short report. You may make any reasonable assumptions from the user requirements. Your diagrams and explanations should clearly cover the needed calculations and assumptions to motivate the design decisions, and how these designs will adhere to the specifications/requirements.

6.2 Task 2

(maximum 2 pages)

You have to report on the software design details to implement the control of LED brightness with the slider and ADC in your code.

You have to clearly document your specifications and requirements and proceed to design the software approach that dictates the behaviour of the ADC and changes the intensity of the LED.

Just as the final circuit diagram without proper context and design does not constitute a "design", so too for software. All code is written according to a planned structure, and those structures should be represented as flow diagrams, state machine diagrams, timing diagrams, etc. Raw code will not be marked.

6.3 Task 3

(maximum 2 pages)

Ensuring the successful operation of the project is the final step of development.

In this task you should gain experience in presenting in detail the test methods you employed, with proof of the measurements and setups used, to ensure the specifications of your project are met. The measurements and test must be of the separate design elements (Task 1) as well as the combined UR2.2a (table 1) system. Extensive testing of the system is encouraged and will help develop a mindset of continual testing of designed systems and the documentation of data.

6.4 Assessment of tasks

These tasks will be assessed through peer assessment - you will have to mark three other reports, and your report will be marked by three other students.

The marking will be carried out using the SunLearn peer assessment module. You will be given a marking rubric to use when carrying out the evaluation. The tasks and your peer evaluations will count towards your report mark - the report portion of the module mark counts 33.3%, and these tasks, and peer evaluation of them, will contribute 10 (14.285%) of the 70 marks of the report portion. The 15 marks total in the rubric will be scaled to a mark out of 10.

Misconduct in carrying out the peer evaluation (i.e. if you make up random marks, fail to complete the peer evaluation, or give marks for clearly incorrect work) will be penalized by receiving 0 for the peer evaluation. This will be done by doing checks on the evaluations of reports.

6.5 Hand-in

A single document with all the tasks (max 6 pages) should be submitted to SunLearn by Friday 14 April 2023 at 12:00. Only documents in, Turnitin readable, PDF format will be accepted.

6.6 Rubric

Criterion	Description	Mark
1.1	Based on the calculations and schematic, it should be sufficient for you, the reviewer, to see how the student designed the LED circuit to meet the requirements, including current consumed, brightness, and pin selection.	2
1.2	Based on the calculations and schematic, it should be sufficient for you, the reviewer, to see how the student designed the Push button circuit to meet the requirements, including choice of resistor, and pin selection.	2
2.1	Measurements must be provided to motivate the calibration (min, max, and linearity) and scaling of the slider and ADC combination input values in the software.	1
2.2	Based on the software design diagram presented, is it clear how the proposed software will correctly scale and calibrate the input values?	2
2.3	Details regarding software structures and components needed to implement the code, as well as the configuration of peripherals, like ADC pins and settings, must be clear and sensible	2
3.1	Testing method(s) should detail the setups and measurements to show that, • the LED circuit (2 mark) • the button circuit (2 mark) • the system functionality (2 marks), using both Slider and the ADC input , is confirmed to meet the requirements/specifications.	6
Total		15

Table 9: Student Report rubric.

7 Final Report

The Final Report is due by the end of the semester and will be in electronic format and submitted to SUN-Learn. In general, the content of the report should enable a student with the same background as yourself (3^{rd} year E&E student, 4^{th} year M&M student) to be able to repeat your project with the same results. It is not a "story" progression of how you built your project, but rather a structured design description with the following elements:

1. What the system is supposed to achieve (high-level system requirements)
2. Which design alternatives were considered
3. How to go from concept design to refining the detail elements (report on concept design, block diagram, interfaces between blocks, then elaborate on the detail of each block. Include calculations, decisions and justification, schematic diagram, software flow diagram, timing diagrams and all other relevant information)
4. How to test for various functionality, and report on the results (Test method, and test results)
5. Summary of whether the system does achieve all the required functions in (1). (Compliance)

Point 2 above would normally appear in a good design report, but as was mentioned earlier in the module – the lecturers already made some of these design choices for you in order to manage component procurement, standardizing on tests and demonstrations etc. For the EDesign report we will thus only expect you to elaborate on point 2 where there was no initial specification for the particular hardware element.

It is expected that your report is organized into the following main sections:

1. Introduction and System description/concept design
2. Hardware design and implementation
3. Software design and implementation
4. Testing
5. Conclusion

7.1 Marking Scheme

7.1.1 Overall

The following overall remarks apply:

- Use the templates provided.
- The report may not exceed 25 pages (Page count should be a maximum of 25 pages, counting from Introduction to Conclusion, and excluding appendices, table of contents, list of images, list of tables and abbreviations, and references).
- Do a spell check and proofread.
- Use diagrams and tables to supplement your text. Make sure all diagrams are properly labeled and captioned.
- Your report will be checked through TurnitIn for plagiarism.

Marks will be awarded for:

Table 10

Report element	Max. marks
Language and grammar	1
Table of contents, list of figures, tables, abbreviations, etc.	2
Overall presentation/impression	2
Sub-total	5

7.1.2 Introduction and System description / Concept Design

Give a brief description of the system. It must describe what the system does, and not how it is done. Include only the most necessary technical details. A short overview of the contents and structure of the report may be given (although this is a bit boring). Write the rest of the report first, then the introduction and lastly the summary (abstract).

- Show block diagrams of the system
- Explain the functioning of the system (in relation to the different blocks)
- Do NOT include design detail

Marks will be awarded for:

Table 11

Report Element	Max. points available
High level system description	1
Summary of requirements/user needs	2
Sub-total	3

7.1.3 Hardware design and implementation

- Provide a (high level) description of your program.
- Provide a description of the design of each hardware element.
- Show all calculations (e.g. resistor values, calibration formula etc.).
- The detail must be sufficient to enable another knowledgeable person to utilize the information to do a similar design.
- Motivate design choices.
- Show the sections of the circuit diagram to which the calculations apply. The given diagrams may be adapted and used, but acknowledge all sources.

Marks will be awarded for:

Table 12

Report Element	Max. points available
Hardware Block diagram and description of interaction	2
Power supply	2
LEDs (debug)	1
Buttons	1
Slider/ADC	2
DAC/PWM filters (or motivation/meas why not needed)	1
Output LEDs + Driver amplifier	3
Azoteq trackpad	2
Sub-total	14

7.1.4 Software design and implementation

- Provide a (high level) description of your program.
- Use state diagrams, flow diagrams, or timing diagrams, or any other applicable diagrams or explanations to explain:
  - Logic for changing states
  - Data flow and processing
- Describe which peripherals (timers, ADC, etc.) of the STM32 are used, and how they are setup. If necessary, provide calculations to motivate register values. (Do NOT include screenshots from the Code Generator)

Marks will be awarded for:

Table 13

Report Element	Max. points available
Software Block diagram and description of interaction	2
Button bounce handling	2
UART communications (protocol and timing)	2
ADC, Setup, Slider calibration and processing	2
DAC, Setup, Calibration and processing	2
PWM, Setup, Calibration and processing	2
I2C, Azoteq trackpad setup, calibration, interface	3
Debug LED + system state indication logic	1
Sub-total	16

7.1.5 Measurements and Results

- The intention of this section is to demonstrate that the built hardware will perform the functions as defined by your hardware design requirements.
- For each element, provide the method that was used to verify the functionality or perform measurements, and report on the result (measurement results, and/or successful verification).

Marks will be awarded for:

Table 14

Report Element	Max. points available
Power supply	2
UART communications	2
Buttons	1
Slider/ADC	2
DAC	1
PWM	2
Current amplifier	1
Azoteq trackpad	1
Output LEDs + debug LEDs	2
Complete system	1
Sub-total	15

7.1.6 Conclusion

- Mention any of the required specifications that your system does not comply with.

- Identify shortcomings in the design/implementation and make recommendations for future expansion or improvements.

Marks will be awarded for:

Table 15

Report Element	Max. points available
Discuss non-compliance’s	1
Identify your design short comings	1
Provide recommendations and possible improvements	1
Sub-total	3

7.1.7 Appendices

- Complete circuit diagram (schematic) must appear in an appendix.
- All connections and component values must be shown. The information must be sufficient to allow someone else to build the circuit.
- A table with the pinout of the STM32 module (which pins connect to which hardware elements on your board) must be supplied in the appendix.
- A complete and correct reference list.

Marks will be awarded for:

Table 16

Report Element	Max. points available
Complete schematic	2
STM32 pins used and their configuration	1
References	1
Sub-total	4

7.1.8 Marks Summary

Table 17

Report Element	Max. points available
Overall	5
Introduction and System description/Concept Design	3
Hardware design and implementation	14
Software design and implementation	16
Measurement and results	15
Conclusion	3
Appendices	4
Total	60

References

[1] *RM0316 Reference Manual, STM32F303xB/C/D/E advanced Arm®-based MCUs*, ST Microelectronics, DocID022558, Rev 8, filename=`rm0316-stm32f303xx`, January 2017.

[2] *NUCLEO-F303RE*, ST Microelectronics, `https://www.st.com/en/microcontrollers-microprocessors/stm32f303re.html`, accessed 2022-03-01.