


2024

Dr Arno Barnard, Dr Armand du Plessis

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

Version History

| Version | Date | Changes |
|---------|-------------|--|
| 0.1 | 13 Feb 2024 | - Initial project specification and guide |
| 0.2 | 28 Feb 2024 | Updates for Demo2+: <ul style="list-style-type: none">- Section 2.2: Table 1 - Updated UR2, UR3, UR5, UR7- Section 3.6: Updated sensor and usage information- Section 4.7: Temperature and light sensor information added- Section 4.8: Button command interface expanded and clarified- Section 4.10.1: Updated Table 3, Added Table 4, TX/RX Examples added- Added section 4.10.2 - values reported by UART- Section 5.1: Table 5 - Pin 14 description clarified- Section 5: Updated Demo 2 description, Table 6, Table 7 |

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 be 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.

For other information about the course *schedule* and *administration*, please see the Module Framework available on SUNLearn.

2 Project Overview

2.1 PV System Efficiency Monitor

Many people in South Africa rely on solar photovoltaic (PV) power as an alternative energy source for their homes and businesses. This is especially becoming an attractive solution, since the return on investment (a performance measure used to evaluate the efficiency or profitability of an investment) has significantly improved over the last five years, given the reduced costs of PV modules, inverters and batteries.

However, dirty/soiled PV modules can seriously affect the PV system power output, which will ultimately result in a financial opportunity cost. Many PV system owners fail to realise the negative impact of dust on PV module power production, which ultimately affects the return on investment.

To their defense, PV system owners are not always sure when to clean their PV system modules, especially if the system is roof mounted, making it more difficult to see the condition (clean vs. dirty) of the PV modules.

As an engineer, your task is to build a device that will help house-hold PV system owners to determine the percentage of reduced PV power output due to dust/soiling. This will help PV system owners to determine when it will be worth going to the effort of cleaning their PV modules to restore maximum power output.

2.2 User Requirements

The project will consist of designing, building and testing a multi-functional device, with the primary objective of reporting on PV module power output given the effects of dirt/soil on the PV module:

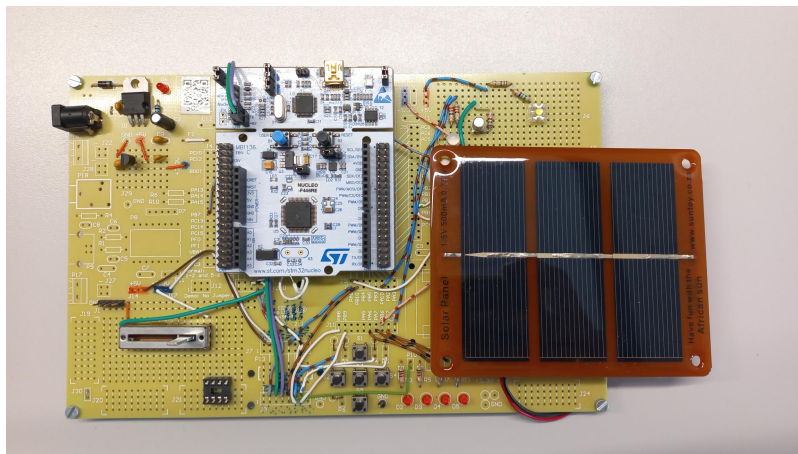


Figure 1: An example of the prototype board.

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

Table 1: User requirements for the project.

| UR# | Description |
|-----|--|
| UR1 | <p>The system shall generate its own regulated supplies from a nominal 9 V-12 V battery or power supply.</p> <ol style="list-style-type: none"> 1. 5 V supply voltage 2. 3.3 V supply voltage 3. LED (D6) must turn on to indicate that the 5V regulated voltage is present. |
| UR2 | <p>PV module measurement: Measure the PV module's operating points by changing the value of the adjustable load. Two adjustable loads will be used: a) a manual multi-turn potentiometer, and b) an automated active load. For the manual case, a time of 1 min will be allowed to adjust and measure. For the automated case, a time of 5 s will be allowed to adjust and measure.</p> <p>Specifically, the device must measure from the PV module:</p> <ol style="list-style-type: none"> 1. The open-circuit voltage [Volt] delivered by the PV module: V_{OC} 2. The short-circuit current [Ampere] delivered by the PV module: I_{SC} 3. The MPP voltage [Volt]: V_{MPP} 4. The MPP current [Ampere]: I_{MPP} <p>Using the measured V_{MPP}, I_{MPP} values above to calculate:</p> <ol style="list-style-type: none"> 5. The MPP power [Watt] of the PV module: P_{MPP} 6. The maximum efficiency, as described in section 2.5, of the PV module: P_{EFF} <p>The above mode of operation must be activated by either of the following:</p> <ol style="list-style-type: none"> 7. UART command input: (a) send SP measurement command to start the measurement sequence, and (b) send SP measurement command again to stop the measurement sequence, as per section 4.10. 8. Bottom pushbutton: (a) Press bottom-button for start- measurement command to start the measurement sequence, and (b) press bottom-button again to end the measurement sequence, as per section 4.8. <p>LED</p> <ol style="list-style-type: none"> 9. LED (D2) must flash rapidly (100 ms on, then 100 ms off) to indicate that the measurement sequence is in progress. 10. LED (D2) must stop flashing (stay on) to indicate that the measurement sequence has ended. <p>Data retrieval/display</p> <ol style="list-style-type: none"> 11. UART command input: The 4 data points and two calculated values listed above must be sent via UART to the device requesting information within 100 ms after receiving the stop measurement command, as per section 4.10. The LCD screen must display the V_{MPP}, I_{MPP}, P_{EFF}, as per section 4.11. 12. Push-button: The 4 data points and two calculated values listed above must be sent via UART to the device requesting information within 100 ms after the stop measurement button pressed, as per section 4.8, The LCD screen must display the V_{MPP}, I_{MPP}, P_{EFF}, as per section 4.11. |
| UR3 | <p>Environment Measurement: The device must provide the ambient temperature, solar panel temperature, and light intensity the board is exposed to.</p> |
| | Continued on next page |

| Table 1 – continued from previous page | |
|--|---|
| UR# | Description |
| | <p>Specifically, the device must measure:</p> <ol style="list-style-type: none"> 1. Using the analogue temperature sensor, the ambient temperature [$^{\circ}\text{C}$]: T_a 2. Using the digital temperature sensor, the solar panel temperature [$^{\circ}\text{C}$]: T_{sp} 3. Using the photodiode, the light intensity [Lux]: Lx_d <p>The above mode of operation must be activated by either of the following:</p> <ol style="list-style-type: none"> 4. UART command input: (a) send <u>EN measurement command</u> to start the measurement sequence, and (b) send <u>EN measurement command again to stop</u> the measurement sequence, as per section 4.10. 5. Top pushbutton: (a) <u>Press top-button for to start the measurement</u> sequence, and (b) <u>press top-button again to end the measurement</u> sequence, as per section 4.8. <p>LED</p> <ol style="list-style-type: none"> 6. LED (D3) must flash rapidly (50 ms on, then 50 ms off) to indicate that the measurement sequence is in progress. 7. LED (D3) must stop flashing (stay on) to indicate that the measurement sequence has ended. <p>Data retrieval/display</p> <ol style="list-style-type: none"> 8. UART command input: The 3 datapoints listed above must be sent via UART to the device requesting information within 100 ms after receiving the stop measurement command, as per section 4.10. The LCD screen must display the T_a, T_{sp}, and Lx_d, as per section 4.11. 9. Push-button: The 3 datapoints listed above must be sent via UART to the device requesting information within 100 ms after stop measurement button pressed, as per section 4.8. The LCD screen must display the T_a, T_{sp}, and Lx_d, as per section 4.11. |
| UR4 | <p>Cleanness index: Device must indicate whether cleaning is necessary when the maximum efficiency PEFF has been determined. An indication of “to clean” or “no-cleaning” will be provided with:</p> <ol style="list-style-type: none"> 1. UART command input 2. Middle push-button is pressed <p>LED</p> <ol style="list-style-type: none"> 3. LED (D5) must flash rapidly (100 ms on, then 100 ms off) to indicate that the PV panel must be cleaned. 4. LED (D5) must be on continuously if the panel is still clean enough. |
| UR5 | <p>Calibration: System owner must be able to calibrate the device. This calibration will only take place when the PV module is clean (without dust/soiling). The calibration procedure must be possible to do using either the manual or automated approaches detailed in UR2:</p> <ol style="list-style-type: none"> 1. LED (D4) must flash rapidly (200 ms on, then 200 ms off) to indicate that the calibration measurement sequence is in progress. 2. LED (D4) must stop flashing (stay on) to indicate that the calibration measurement sequence has ended. |
| | Continued on next page |

| Table 1 – continued from previous page | |
|--|---|
| UR# | Description |
| UR6 | Data storage (To be confirmed at a later stage): 1. The device will save all the measurements it performs and supply it via the UART when receiving the appropriate command. |
| UR7 | The device shall use a UART connection, in both transmit and receive modes. The device shall operate in 8 data bits, one odd parity bit and one stop bits configuration, at a data rate of 115200 baud. The UART information, both transmitted and received, shall be formatted as per section 4.10.1. The system shall receive commands as defined in section 4.10.2 and respond accordingly. 1. No sooner than 100 ms after startup, and no later than 500 ms after startup, the device will send the board owner's student number, formatted as per section 4.10.1. 2. The device shall report the following information via UART after receiving the appropriate command: (a) The most recent PV panel parameters, V_{OC} , I_{SC} , V_{MPP} , and I_{MPP} . (b) The current ambient temperature, solar panel temperature, and level of the ambient light intensity, T_a , T_{sp} , and Lx_d . (c) The last recorded / calculated MPP power and efficiency of the PV module, P_{MPP} and P_{EFF} . |
| UR8 | Real-Time-Clock (RTC): The device shall keep time using its built-in Real-Time Clock (RTC) |



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).

2.3 UART communication

The system should use the UART port connected to the PC (**and the TIC for Demo purposes** see section 5.3) to report measurements and system statuses and to receive configuration commands. The full details of the communication protocol are given in section 4.10.

2.4 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, and UART. The output interface shall be via the UART. Selected measurements and calculated values will be displayed on the LCD. The system's current state shall also be indicated using the four debug LEDs.

2.5 Solar panel I-V curve and efficiency

Details about the PV panel characteristic I-V curve, the associated parameters, and how to measure and calculate the maximum powerpoint and efficiency of the PV panel will follow in a later version of this document.

3 Hardware

The STM32 microcontroller board that you will use is the STM32F411RE. 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-STM32F411RE 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.

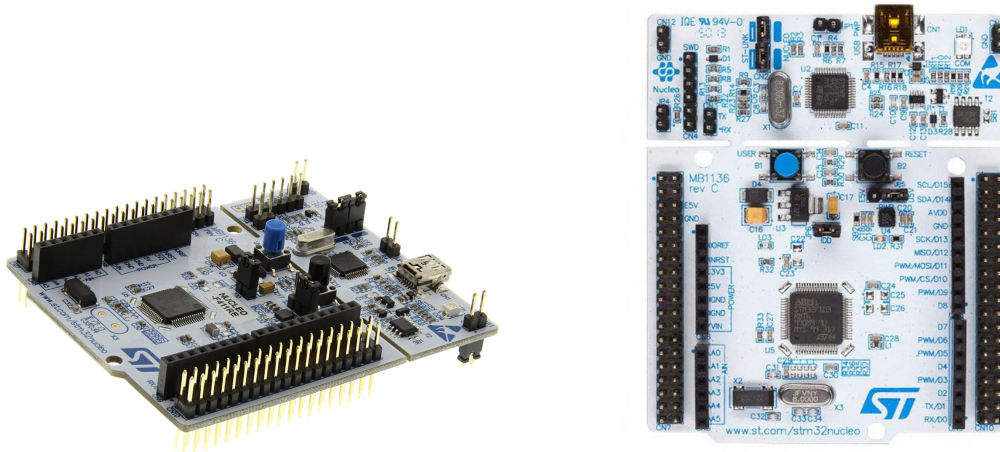


Figure 2: NUCLEO-F411RE development board [2]

3.1 Power Supply

The board must be supplied with a nominal 9V 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 3.

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.

i 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 MCP1700 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).

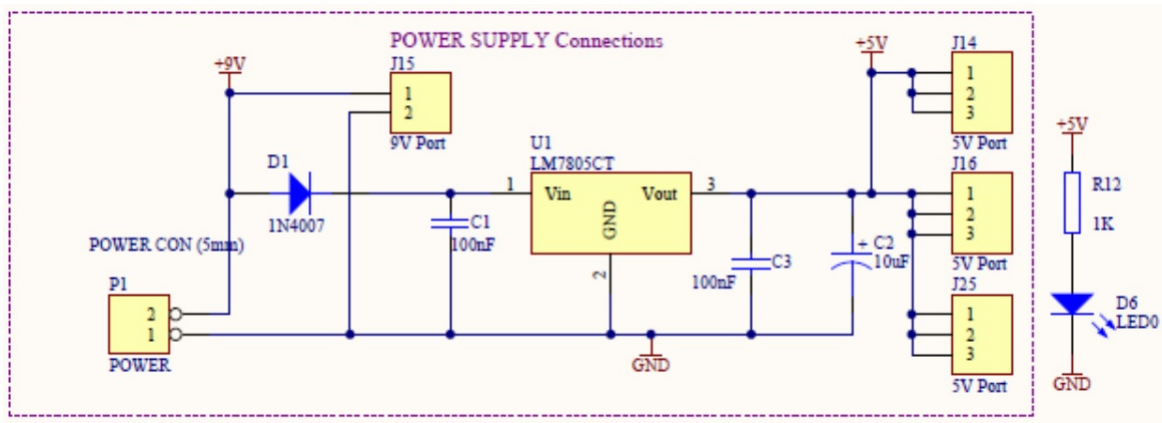


Figure 3: 5 V Power supply circuit diagram

! Note that the 3.3 V supply should NOT be connected to the NUCLEO-STM32F411RE'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-STM32F411RE 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 4: 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.


i 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.

i 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 5.


 *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 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!*

3.4 LCD Display


The information in this section will be updated once delivery of the 1602A LCD is confirmed.

You will use a 16x2 character LCD display (1602A) as the display device in this project. You should use a mail header strip to solder the LCD to your base board, as female header strips are in limited supply. The LCD connections on the PCB are pre-connected to the required VCC (5 V) and GND. Note that this LCD is 3.3 V compatible, but the baseboard is powering it from 5 V. This seems to be OK, as the 3.3 V changes are only to provide enough voltage drop (it generates a negative voltage for the LCD panel). I have run the this LCD in 5 V mode for many hours and it operates fine with 5 V supply.

You will need to connect the other LCD pins electrically to your NUCLEO pins, using wires, from P19.

 *You should only require 7 GPIO pins.*

You will have to use the LCD in 4-bit (nibble) interface mode, since the LCD's data lines D0 - D3 are hard wired to ground. The LCD further has a contrast adjustment pin, for which you should design the two resistors R8 and R9 required to provide a good contrast on the display. The model of the LCD we received is a difficult to read without the backlight. However, there is no specific allowance for connecting the backlight (LED) to the PCB, you will have to connect wires to the A and K solder terminals.


 *Do not connect the backlight A and K terminals directly between VCC (5 V) and ground! It is still an LED and it requires a current limiting resistor in series (keep the current below 10 mA). The typical voltage drop across the backlight can be as high as 4.1 V, but be safe and measure it in your circuit.*

3.5 Voltage and current sensing

The system will measure the voltage across the PV panel, and current supplied by the PV panel to the load, using an electronic circuit and the built-in Analogue-to-Digital Converter (ADC) of the STM32F411RE. More details will follow in a later version of this document.


3.6 Temperature sensing

The system will measure the ambient and PV panel temperatures using an analogue and digital temperature sensor respectively.

 Both these temperature sensors are sensitive devices, please handle them with care and do not expose them to harsh conditions or over voltages.

For the ambient temperature, the analogue sensor (LM235) together with the built-in ADC of the microcontroller should be used. The sensor needs power and ground and provides an analogue output. Please see the LM235 datasheet uploaded on SUNLearn for more information about using the device. The analogue output of the sensor must also be connected to the TIC (see section 5.1). This sensor must be mounted so that the ambient (room) temperature is measure by it.

For the solar panel temperature the digital sensor (LMT01) connected to two GPIO pins. This sensor has a two wire interface that you connect using two GPIO pins and a resistor, please see the device datasheet on SUNLearn for more information. This sensor uses a digital pulse train to communicate the measured temperature within a timing window. You will have to count the pulses using one of the GPIO pins. By using the GPIO pin as a "clock" input, through configuring one of the STM32F411 timers, the pulse counting can be accomplished. Alternatively configuring the GPIO pin with an external interrupt and counting in software is also possible. In both cases you will probably need a second timer to handle the window timing. When the solar panel is available, this sensor will b mounted to the panel to measure the solap panel temperature.

 For Demo 2 the digital temperature sensor will also measure the ambient temperature since the solar panels are not available yet.

3.7 Light sensing

The system will measure the light intensity the PV panel is exposed to a photo sensitive diode-based sensor. More details will follow in a later version of this document.

3.8 PV panel

The system will use a representative PV panel that will be used to provide a PV source for the system. More details will follow in a later version of this document.

3.9 Multi-turn potentiometer

The PV panel will have to be connected to a load. Initially the load element will be a variable resistor, more specifically a multi-turn potentionmeter. This will be used to measure the I-V curve of the PV panel using a manual sequence. More details will follow in a later version of this document.

3.10 Active load

The PV panel will have to be connected to a load. For the system to automatically measure the I-V curve an active load will be used that can be automatically controlled by the STM32F411RE controller. More details will follow in a later version of this document.

3.11 NUCLEO-STM32F411RE 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.

The NUCLEO-STM32F411RE 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-STM32F411RE, in your application unless you have a specific requirement - BOOT, 5V, RESET, VIN and any of the NC pins. The NUCLEO-STM32F411RE operates normally without connections to these pins.*

3.12 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 5). 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.

| CPU Pin | Port | 5V | Connector-Pin | | | Proposed Function |
|---------|------|----|---------------|---------|----------------------|---|
| | | | Morpho | Arduino | Baseboard Solder Pad | |
| PA0 | A | n | CN7-28 | CN8-1 | J10-1/2 | UART2 TXD UART2 RXD LED2 on Nucleo |
| PA1 | A | n | CN7-30 | CN8-2 | J10-3/4 | |
| PA2 | A | n | CN10-35 | CN9-2 | J9-13/14 | |
| PA3 | A | n | CN10-37 | CN9-1 | J9-15/16 | |
| PA4 | A | n | CN7-32 | CN8-3 | J10-5/6 | |
| PA5 | A | n | CN10-11 | CN5-6 | J11-9/10 | |
| 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 | SWO (PC Debug) |
| 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 | |
| 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 | |
| PB7 | B | Y | CN7-21 | | J4-21/22 | |
| 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 | Blue Pushbutton 32 kHz Xtal 32 kHz Xtal |
| 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 | |
| PC14 | C | n | CN7-25 | | J4-25/26 | |
| PC15 | C | n | CN7-27 | | J4-27/28 | |
| 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-STM32F411RE to Baseboard connections

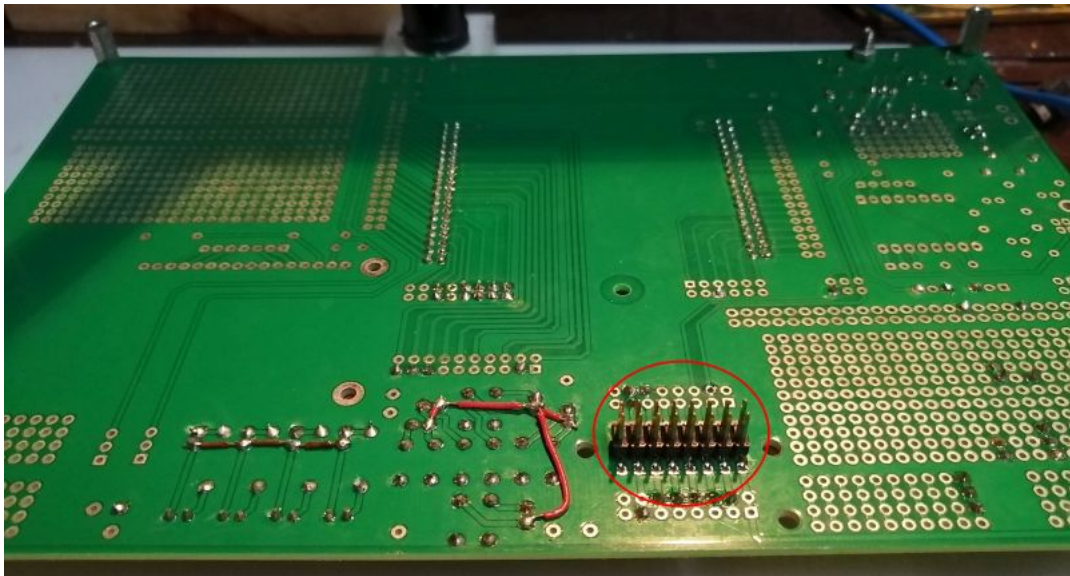


Figure 5: The TIC shown on the solder side of the baseboard.

4 Software Design and Considerations

4.1 System operational overview

As the system operates and receive user inputs (via buttons, and UART) it should perform the requested commands and output the required information via UART and LCD.


Command buttons 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 startup state will be idle.
- The system will display the real-time power as measured continuously from the solar panel.
- The RTC value will be default: 19/09/2024 16:20:00.
- The active load (when present) will be set to its maximum value.

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 F4 Firmware package 1.27.1 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.

```
1      void SysTick_Handler(void)
2      {
3          /* USER CODE BEGIN SysTick_IRQn 0 */
4          /* USER CODE END SysTick_IRQn 0 */
5          HAL_IncTick();
6          /* USER CODE BEGIN SysTick_IRQn 1 */
7          /* USER CODE END SysTick_IRQn 1 */
8      }
9
```

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:

```
1      // Wait for 5 microseconds
2      myUsecDelay(5); // call self-created u-sec accurate delay function
3
```

4.7 ADC Conversions and measurements

4.7.1 Conversion

The ADC can be quite fast (as little as 2-3 microseconds conversion time), 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 at least four (4) single input channels**: **1**) Voltage+ (PV voltage), **2**) Voltage- (Voltage after current sense resistor, to calculate the PV current), **3**) LM235 ambient temperature sensor, **4**) SFH203 photodiode output.

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, as might be required during the IV-curve measurement and calibration procedure.

4.7.2 Voltage measurements

You will use the internal ADC to measure the voltage and current from the solar panel. For this application, timing is less important, however, an accurate measurement is important as you want the current measured with as little as possible noise.

The solar panel voltage and current should be accurately measured to within 5% accuracy. Calibrating the ADC can assist you greatly in achieving good accuracy. You can calibrate your system's ADC 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.

4.7.3 Temperature measurement - LM235


The analogue output of the LM235 should be sampled using the ADC in single channel mode. Once sampled the value must be converted using the guidelines and information in the datasheet to convert the value to a temperature in degrees Celsius. This temperature must then be converted into a 3-digit string, theoretically representing temperatures from -99 to 999. However the LM235 operating temperature range is much less (see datasheet). The 3-digit string can then be sent via UART and be displayed on the LCD.

4.7.4 Light measurement - SFH203

The analogue output of the Silicon PIN Photodiode circuit must be sampled using the ADC in single channel mode. Once sampled the value must be converted to a Lux value using the datasheet information. More details about this sensor will follow in a later version of the PDD.

4.8 Button command operation

The system will implement a command interface by using the five push buttons as well as the UART input. Such a system is usually implemented using a software state machine. The interface to the system required by this project is defined below:

- Use the RIGHT button to initialise a solar panel reference point calibration 
- Use the LEFT button to select the display option for the LCD (see section 4.11)
- Use the TOP button to measure, then display (on LCD) and report (via UART) the immediate temperatures (T_a, T_{sp}) and light intensity (Lx_d). The first button press starts the measurement (averaging the samples) and the second button press ends the measurement and display / report the values.
- Use the BOTTOM button to measure, calculate, and display the immediate power PV panel efficiency. The first button press starts the measurement process and the second button press ends the measurement and display / report the values.
- Use MIDDLE button to initiate an RTC clock set menu. Once in the RTC set menu, the TOP and BOTTOM buttons must operate as increment and decrement inputs to the vales while the MIDDLE button sets each entered value in turn until all date and time values have been entered. Once the complete date and time have been entered, the device must report the time via the UART.

4.9 Debug LEDS

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

- D2 - Power efficiency measurement in progress (flashing), and completed (D2 = ON)

- D3 - Temperature and Light measurement in progress (flashing), and completed (D3 = ON)
- D4 - Calibration sequence in progress (flashing), and completed (D2 = ON)
- D5 - Cleanness index progress (flashing), and completed (D2 = ON)

i 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 115200 bps, with 8 data bits, 1 parity bit, and one stop bit, using TTL signal levels. The parity used will be odd 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 Termit, 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 an underscore '_' character. 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).**

i 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.

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 '_' character.

| | | | | | | | | | |
|----------------------|-----|-----|-------------------|-----|--------------------|-----|--------------------|-----|-------|
| Field in bytes | 1 | Sep | 8 | | | | | Sep | 2 |
| Student Number | '&' | '_' | <StdNum> | | | | | '_' | '*\n' |
| Field in bytes | 1 | Sep | 3 | Sep | 3 | Sep | 3 | Sep | 2 |
| Environment Response | '&' | '_' | <T _a > | '_' | <T _{sp} > | '_' | <Lx _d > | '_' | '*\n' |

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

Example TX message:

- Start-up message for student 12345678:

```
&_12345678_*(newline)
```

| | | | | | | |
|----------------|-----|-----|-----------|--------|-----|-------|
| Field in bytes | 1 | Sep | 2 | | Sep | 2 |
| Command | '&' | '_' | <Command> | | '_' | '*\n' |
| Field in bytes | 1 | Sep | 1 | 1 | Sep | 2 |
| Display on LCD | '&' | '_' | <RS> | <byte> | '_' | '*\n' |

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

- Start-up then 'a', '1', characters received (for Demo 1 only):

```
&_12345678_*(newline)
a1
```

- Response for environment measurement with $T_a = 25^\circ\text{C}$, $T_{sp} = 33^\circ\text{C}$, $Lx_d = 111\text{Lux}$:

```
&_025_033_111_*(newline)
```

Example RX message:

- Command to measure environment:

```
&_EN_*(newline)
```

- Display character 'A' on LCD:

```
&_1A_*(newline)
```

4.10.2 UART Value Representation

For the values transmitted and received via UART the following representation and ranges will be used:

- <StdNum>: 8-digit student number, Range: 1000000 to 40000000
- < T_a >: 3-digit (with leading zeros) ambient temperature in Celsius, Range: -99 to 999
- < T_{sp} >: 3-digit (with leading zeros) solar panel temperature in Celsius, Range: -99 to 999
- < Lx_d >: 3-digit (with leading zeros) light intensity in Lux, Range: 000 to 999
- <Command>: The command can be one of three options:
 - 'SP': Measure solar panel parameters (UR2), V_{OC} , I_{SC} , V_{MPP} , I_{MPP}
 - 'EN': Measure the environment (UR3), T_a , T_{sp} , Lx_d
 - 'CA': Calibrate the monitor (UR5)
- <RS>: 1-digit RS value for LCD operation, Range: 0 to 1
- <byte>: 1-byte parameter for LCD operation

4.11 LCD display

The objective of the LCD screen is to present useful information to the system owner. The display options of the LCD should function as follows.

- Display Mode 1: Default state of the LCD is to always indicate the most recently measured power output value of the solar panel.
- Display Mode 2: When the LEFT button is pressed 1x, then the LCD display should indicate the most recently measured temperature and light sensor values.
- Display Mode 3: When the LEFT button is pressed 2x, then the LCD display should indicate the current time: DD/MM/YYYY hh:mm:ss
- Display Mode 4: When the LEFT button is pressed 3x, then the LCD display should cycle through Display 1, 2 and 3 (as described above) at an interval rate specified by the user. The LCD exits Display mode 4 when the LEFT button is pressed 1x again.

Details on the exact format of the character displays for each of these modes will be provided in a next version of this document.

4.12 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!*

4.13 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 5, and the pin numbering of J13, J3, and P13 are as shown in Figure 6.

i From your board, you will have to route the signals listed in Table 5 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).

i 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} - 9V Supply from test station | SB <-> TS |
| 2 | N/C | V _{bat} - 9V Supply from test station | SB <-> TS |
| 3 | J3 - 3,4 | Load drive voltage as Analogue signal from SB, PWM output (direct from pin) | 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 | 3.3V 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 TS, ADC input from voltage measurement circuit | SB -> TS |
| 9 | J3 - 9,10 | Button TOP (connected to TS for control/verification/measurement) | SB <-> TS |
| 10 | J13 - 9,10 | Analogue signal to TS, ADC input from current measurement circuit | SB -> TS |
| 11 | J3 - 11,12 | Button MIDDLE (connected to TS for control/verification/measurement) | SB <-> TS |
| 12 | J13 - 11,12 | Light sensor measurement as input to TS | SB -> TS |
| 13 | J3 - 13,14 | Button BOTTOM (connected to TS for control/verification/measurement) | SB <-> TS |
| 14 | J13 - 13,14 | Analogue Ambient Temperature sensor (T_a) measurement (ADC input) as input to TS | SB -> TS |
| 15 | N/C | GND | SB <-> TS |
| 16 | N/C | GND | SB <-> TS |

Table 5: Test-interface Connector Pin definitions

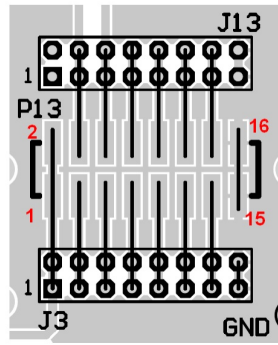


Figure 6: Pin numbers of J13, J3 and P13

i Connections shown in grey in Table 5 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 6. **This version of this document only covers test methods up to Demo 1, more detail for the other demonstrations will follow in future versions of this document.**

| 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 (1) 5V and (2) 3.3V supplies to be in 5% of expected values, and (3) the LED D6 is on when 5V is present |
| UR3 | (Light intensity EXCLUDED for Demo 2) The TS will push the TOP button via the TIC interface. The TS will compare the two temperature values received via UART, and analogue temperature measured via TIC, from the SB to the ambient temperature as measured by the TS. | Test passes if (1) the TS and SB temperatures are with 3°C of each other, (2) the analogue voltage matches the reported temperature within 5%, and (3) the digital sensor value changes appreciably when touched. |
| UR7.1 | 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. |
| UR7.2 | 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 message formats, to the configuration/command messages. |

Table 6: Test station test method definitions

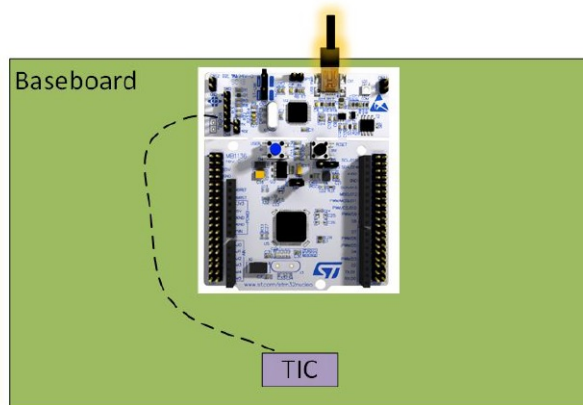
5.3 UART communications interface

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

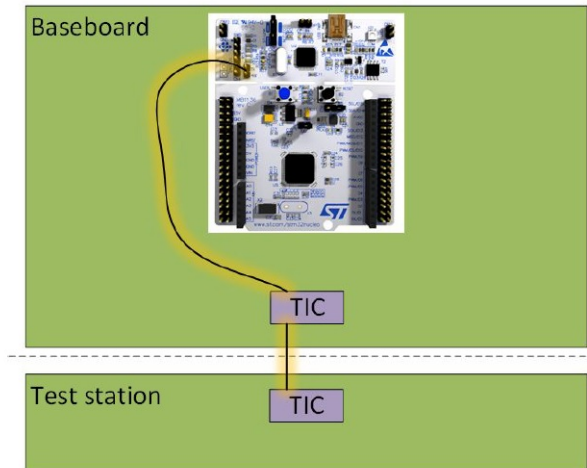
to the PC via the USB debug cable. Thus, for debugging it will not be necessary to make any hardware connections.

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 3.11 and 5.1). Both methods are shown in Figure 7.

⚠ 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.



Student performs test or debugging without test station. UART appears as a Virtual COM port on PC



Automated testing through Test Station - UART is routed to TIC

Figure 7: UART connections for debugging and testing.

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 7. In this version of this document only details up to Demo 2 is listed. Details for the other demonstrations will follow in future versions of this document.

For Demo 2 the appropriate buttons, appropriate debug LEDs, UART, and both temperature sensors must be working correctly.

| 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. | UR1, UR7.1 (student number and echo), UR7.2 (modified - only echo) |
| 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 TOP button (via TIC P13 pin 9) while monitoring the UART, LM235 outputs (via TIC P13 pins 5, 7, 14), and taking photographs (for debug LEDs) to check for correct system operation. The minimum delay between consecutive UART commands will be 500 ms. The system should update and respond IMMEDIATELY (<100ms) to any input. The TOP button will be manually pressed too and the system response monitored via UART, ADC and photos to check for correct system functioning. | UR1, UR3 (no Light sensor, no LCD), UR7 |

Table 7: 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

More details to follow in future versions of this document.

6.2 Task 2

More details to follow in future versions of this document.

6.3 Task 3

More details to follow in future versions of this document.

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 and Rubric

Hand-in dates are stated in the Module Framework schedule and will be strictly applied. A detailed Rubric will follow in future versions of this document.

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 (3rd year E&E student, 4th 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, standardising 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 organised 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.

A detailed Rubric will follow in future versions of this document.

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.