Design (E) 314 — Ontwerp (E) 314
Project Definition — Projek Definisie

2021

Dr. Arno Barnard, Dr. Callen Fisher

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

Version History

Version	Date	Changes
0.1	15 Mar 2021	- Initial project specification and guide
0.2	29 Mar 2021	- Updated slider voltage specifications (overview and testing)
0.3	7 April 2021	- Added "Specification" Report requirements
0.4	4 May 2021	- Added velocity and visible specifications to overview
0.5	13 May 2021	- Added final report information and rubric
0.6	21 May 2021	- Added Tennis 2 player specifications (overview) and accelerometer information (overview, testing and hardware)

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.

2 Overview

The project will consist of designing, building and testing of an arcade style game, making use of a dot matrix display.

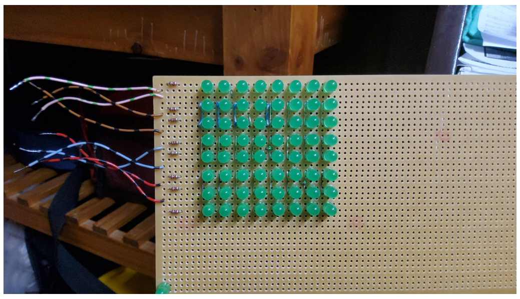


Figure 1: An example of the dot matrix consisting of 8x8 LEDs.

In this project you will build a system, controlled by a single STM32F103 microcontroller, to sense the user input (from buttons, potentiometer and accelerometer) to play one of three games.

The first game involves moving a ball through a selected pre-defined maze. The objective of the game is to move the ball from the start position to the end position, without passing through any of the maze walls/obstacles.

The second game involves playing a tennis style game. The objective of the game is to keep bouncing the ball against the back wall, if you miss the ball you lose. After a set number of hits, the ball increases in speed. You can move the bat up, down, left and right.

The third game is optional, if successful, bonus marks will be awarded. The game involves playing tennis against another user. This game follows the same rules as the single player mode, just instead of bouncing off the back wall, it goes onto the opposition's dot matrix.

The requirements that should be fulfilled are:

UR1	The system will generate it's own regulated 5 V and 3.3 V supply voltage from a nominal 9 V battery or power supply.
UR2	During game play, the device should sample (rate depending on the game) the following sensors: 1. Accelerometer 2. Potentiometer 3. Buttons
UR3	The device shall use a UART connection, in transmit mode only, that operates in an 8N1 configuration (8 data bits, no parity bits and a single stop bit), at a data rate of 115200 baud. The UART information shall be formatted as per section 5.3. The device shall report the following information via UART: 1. ball position (X and Y) 2. bat position (X and Y, top of the bat)
UR4	The LED dot matrix display must: 1. consist of 8 by 8 (64) LEDs, 2. size constraint: 60 mm by 60 mm (23 holes by 23 holes), 3. located at the top right of the baseboard, 4. be updated a row/column at a time every millisecond by the STM32, 5. display a test pattern at startup that lights each row/column at a time to verify all the LEDs are working.
UR5	Using buttons, a game must be selected as follows: 1. Left button: maze 2. Middle button: tennis (single player) 3. Right button: tennis (two player)
UR6	See below for game 1 (maze) specifications
UR7	See below for game 2 (tennis, single player) specifications
UR8	See below for game 3 (tennis, multiplayer) specifications
UR9	Once the game ends, show the calibration pattern (light up the four corners of the display) and wait for another game to be selected.

Table 1: User requirements for the project.

Game 1 (maze) specifications:

1. Select one of 4 possible pre-defined mazes. Use up and down buttons to cycle through the mazes. Display selection on debug LEDs (D2-D5) as well as the dot matrix and use the middle button to select the desired maze.
2. Use the accelerometer to move the ball through the maze to the goal position. The ball can move one light (up/down/left/right, not diagonally) every 300 ms. Alternatively use the buttons to move the ball (required for initial demo).

3. The position of the ball must be transmitted via UART every 300 ms.
4. The maze boundaries must remain on, where as the ball must flash (toggle) every 300 ms, and the goal must flash (toggle) every 100 ms.
5. The game ends when the ball reaches the goal position, or the quit button (middle button) is pressed.

Game 2 (tennis, single player) specifications:

1. The bat is the size of two lights, and remains vertical. It can move vertically and horizontally
2. The bat is moved using multiple methods (requirements for different demos):
   (a) Buttons for vertical and horizontal motion
   (b) The slider for vertical motion. When the output of the slider is at supply voltage, the ball must be at the top, when it is at GND, the ball must be at the bottom. (with buttons for horizontal motion)
   (c) The accelerometer
3. The bat can only move one light every 100 ms (the position of the bat (top light) and ball must be transmitted via UART every 100 ms).
4. The ball starts off moving one light every 700 ms, after every 3 times it hits the bat, decrease the update time of the ball by 50 ms (a maximum of 10 times).
5. The game ends once you miss the ball (goes off the screen behind the bat) or you press the quit button (middle button).
6. The ball starts on light [7,4], see Section 3.3.
7. See Section 2.1 for how the ball moves.

Game 3 (tennis, two player) specifications:

1. Same specifications as above, just the ball does not bounce off the back wall, it goes to player 2.
2. Connect the two boards together using a crossover cable and 3 pin connectors (1: GND, 2:TX from your board, 3: RX from your board). The same UART specifications are implemented (Baud rate etc).
3. When your board is displaying the calibration pattern (4 corner lights), it is considered to be in slave mode, monitor the second UART channel (NOT THE DEFAULT UART USED ON THE TIC) for communications.
4. One of the boards will have to become the "Master" and start the 2 player game (by pushing the correct push button) and sending an initial UART message to the slave. For now, the master will always be on the left, and the slave will be on the right. The UART message is a blank tennis message ("2_______ \n")
5. Once the master has started the game, the ball starts on light [7,4] and moves left. The master plays the game as normal, WITHOUT SENDING UART COMMS TO THE SLAVE.
6. When the ball crosses the back wall, a tennis UART message is sent to the slave. It will play the game as normal, without sending messages until the ball crosses the back wall again.
7. Remember, for the slave, the back wall is on the left.
8. Also take note that the tennis UART message will have the X and Y position relative to the board sending the message, so you will need to determine where is is meant to be on your screen.
9. If the middle button on either device is pressed, the game quits. If any player misses the ball the game quits.

10. If the game quits due to you missing the ball, or pressing the middle button, send a blank tennis message to the other device so it can quit too, and return to the calibration pattern.
11. For this game, do not worry about increasing the velocity.

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

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

You will also have to devise suitable tests to show that your system conforms to the mentioned requirements. 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 penalized for multiple demonstrations.

The rest of this document describes the hardware that you will be required to use and interface (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 Tennis ball motion

When the tennis game starts, the ball will begin moving horizontally left. When the ball hits the bat, a number of different motions will occur. Example motions are shown in Figure 2. Looking at this figure, if the ball hits the edge of the bat (first row), it bounces back in the same direction it came. If the ball hits the bat (second row) at an angle, it bounces off as expected. If the ball hits the bat straight on (will only ever occur as the game starts), then the ball bounces off at an angle (upwards if its the top of the bat, downwards if it is the bottom of the bat).

2.1.1 Direction naming convention

Each direction that the ball can move in has a number. This number will be part of the UART package transmitted from the SB to the TS (transmitted as a character!!). The naming convention is shown in Figure 3.

2.1.2 Velocity naming convention

Initially the start velocity is set to 1 (transmit as a character!!). After 3 hits, the velocity increases and becomes 2 etc. This number will be part of the UART package transmitted from the SB to the TS (transmitted as a character!!).

2.1.3 Bat position naming convention

The bat consists of two (vertical) LEDs. For the Y position of the bat, use the position of the top LED of the bat. This number will be part of the UART package transmitted from the SB to the TS (transmitted as a character!!).

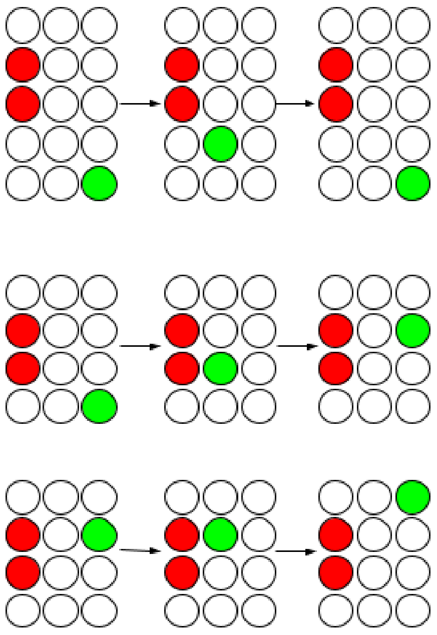


Figure 2: Example motion.

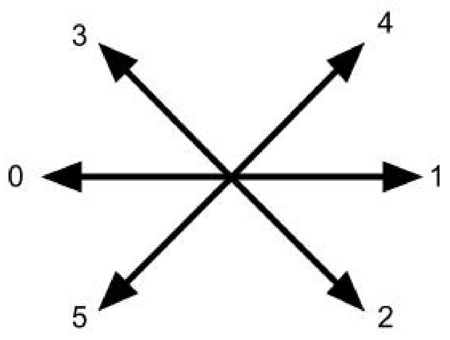


Figure 3: Naming convention for the direction of the ball

2.2 Visible naming convention for maze

For the maze game, the ball and goal position flash at a set frequency. As the game starts, both must be on. If the ball/goal LED is on, transmit a 1 as a character. If it is not visible (the LED is off), transmit a 0 as a character. This number will be part of the UART package transmitted from the SB to the TS (transmitted as a character!!).

2.3 IMU naming convention

You need to determine the angle of the accelerometer, any angle over 30 degrees is regarded as a valid input to make the bat/ball move. Effectively, if the accelerometer is at an angle of 30 degrees, it mimics the relevant button press. If the accelerometer is tilted 30 degrees left, then mimic the left button press and transmit an 'L' for the IMU byte in the UART packet. For right, up and down, transmit 'R', 'U' and 'D' respectively. If the IMU is at an angle less than 30 degrees, transmit an 'N' for none.

We will not test the case where it is registering more than one angle (for example left and down).

3 Hardware

The STM32 microcontroller board that you will use is the STM32F103RB. 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-STM32F103RB 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.

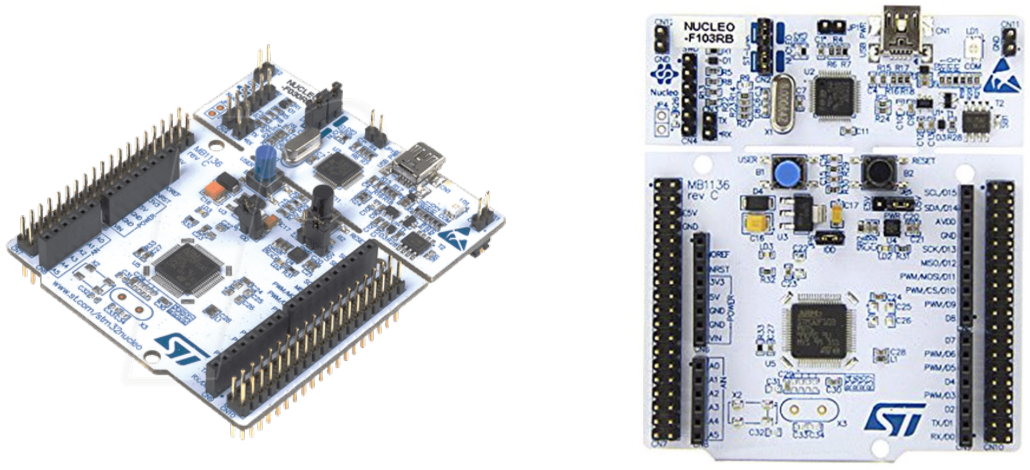


Figure 4: NUCLEO-F103RB development board [2]

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.

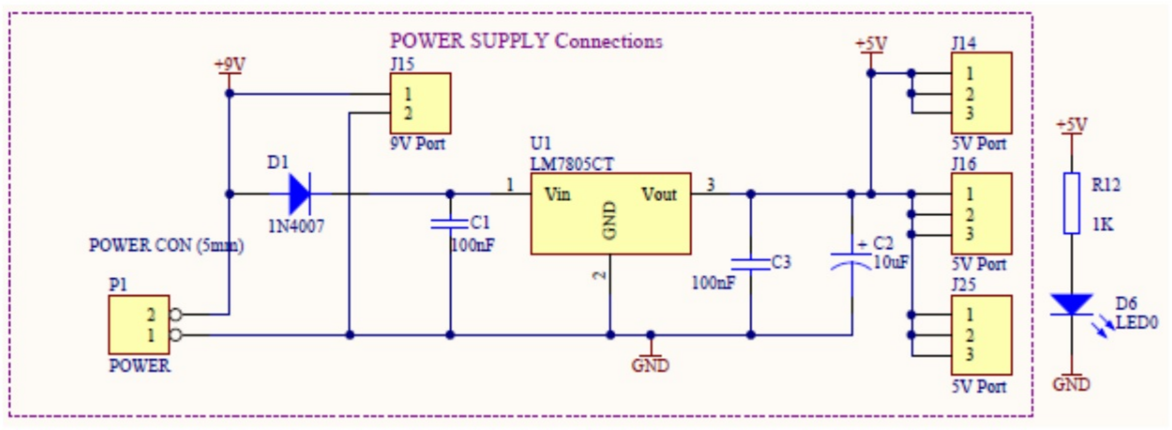


Figure 5: 5 V Power supply circuit diagram

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).

> ⚠ *Note that the 3.3 V supply should NOT be connected to the NUCLEO-STM32F103RB'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-STM32F103RB 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.

> ⚠ *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.*

> ⓘ *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 Switches

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. They will also need to be connected to the TIC.

> ⓘ *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 Dot matrix

You are required to wire up an 8 by 8 dot matrix using 64 LEDs (with 8 series resistors to limit current), 8 MOSFETs (do you need gate resistors for the MOSFETs?) and 16 GPIO pins on the STM32F103RB. A suggested wiring diagram for the LEDs is provided in Figure 6. In terms of numbering the LEDs, the top left LED is considered [0,0] (x,y), one LED to the right is [1,0].

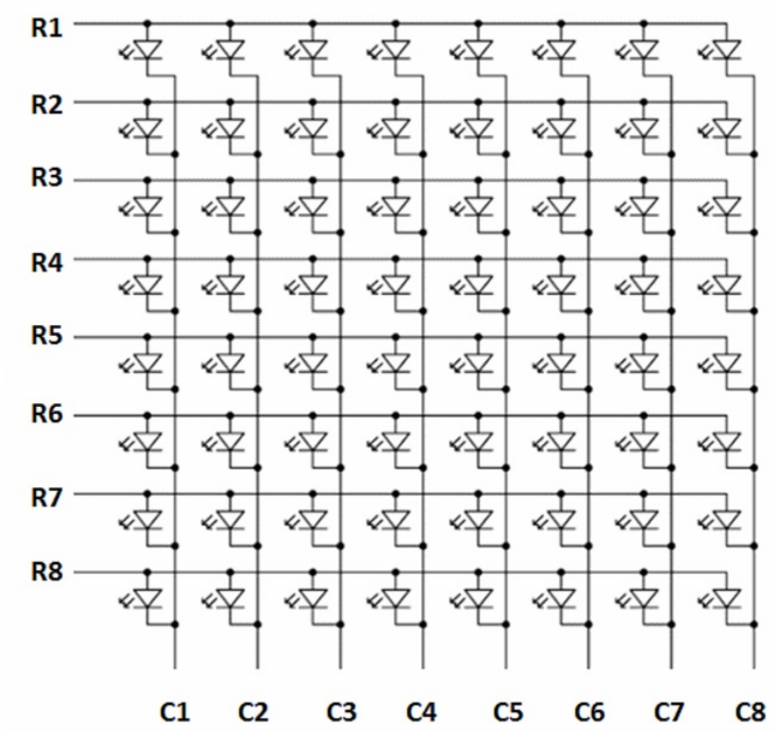


Figure 6: An example wiring diagram of the dot matrix display.

You should only require 16 GPIO pins.

Remember to limit the current through the LEDs by using a series resistor. Make use of the provided MOSFETs to ensure a consistent brightness is achieved. Where will you place the MOSFETs and where will you place the series resistors?

You can get more information about the internal circuit of the GPIO pins in the STM32F103RB's Reference Manual (cd00171190), section 9.1.

3.4 Slider

A analogue slider has been provided. You are required to connect it to the ADC of the STM32F103RB 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? 5V or 3.3V?

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

3.5 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 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!*

3.6 NUCLEO-STM32F103RB 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	SPI1_NSS
PA5	A	n	CN10-11	CN5-6	J11-9/10	SPI1_SCK (LED2)
PA6	A	n	CN10-13	CN5-5	J11-11/12	SPI1_MISO
PA7	A	n	CN10-15	CN5-4	J11-13/14	SPI1_MOSI
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	Y	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	n	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	Y	CN10-25	CN9-7	J9-3/4	
PB11	B	Y	CN10-18		J5-17/18	
PB12	B	Y	CN10-16		J5-15/16	
PB13	B	Y	CN10-30		J5-29/30	
PB14	B	Y	CN10-28		J5-27/28	
PB15	B	Y	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	UART1 TXD
PC5	C	n	CN10-6		J5-5/6	UART1 RXD
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-STM32F103RB to Baseboard connections

The NUCLEO-STM32F103RB 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-STM32F103RB, in your application unless you have a specific requirement -* `BOOT`, `5V`, `RESET`, `VIN` *and any of the* `NC` *pins. The NUCLEO-STM32F103RB operates normally without connections to these pins.*

3.7 LIS3DH - 3-axis MEMS accelerometer

See the LIS3DH datasheet provided on SUN Learn for the details about the operation of this sensor. You are required to use the I2C protocol.

You are required to measure the *accx*, *accy*, and *accz* values from the accelerometer. From these measurements, determine the angle of the device. For the project, any angle over 30 degrees will be regarded as a valid input to make the ball/bat move. You will report this in the UART package (IMU character, more details in Testing section).

The accelerometer will be attached to your PCB using 10cm long wire. This way you can rotate it while playing the games. For more information on how the accelerometer works, watch the lecture video on SunLearn.

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 7). This connector mates with a 16-pin Amphenol Dubox connector on the Test-Jig 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.

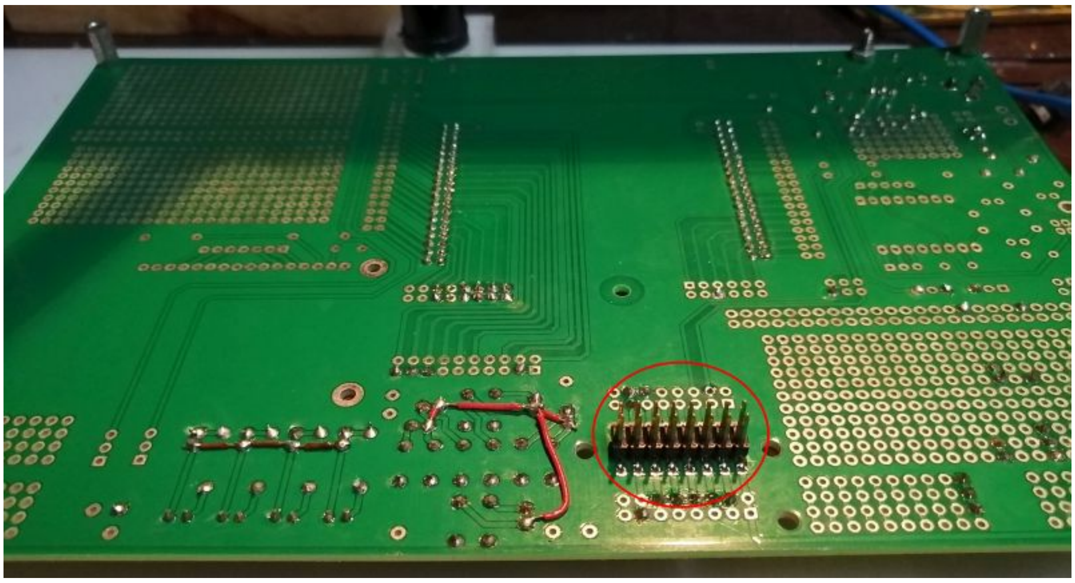


Figure 7: The TIC shown on the solder side of the baseboard.

4 Software Considerations

4.1 General

To develop software for this project, the same environment and typical development process as used in Computer Systems 245 will be used: STMCubeIDE v1.3. There is however one significant change: The use of a software version control system: GIT, described in more detail in section 4.2.

> ⚠ *The use of STM32CubeIDE v1.3 is NOT optional.*

> ⓘ *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 game, accelerometer and dot matrix display functions are recommended.

4.2 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).

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

4.3 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 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 2–5× 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.

4.4 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 STM32f103 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 `stm32f1xx_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.

4.5 ADC Conversions

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 is sufficient.

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.

4.6 Calibration Sequence

On startup, you are required to individually light up each column of the dot matrix display, with a 1 second interval between each column. You are also required to output the current row on the UART (see Section 5.3 for more details on UART). Once you have cycled through each column, you need to display the calibration pattern (light up the 4 corners) as shown in Figure 8.

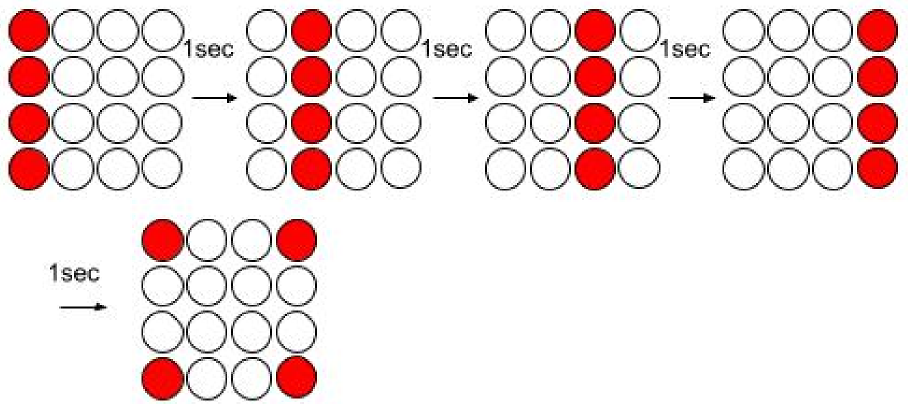


Figure 8: The calibration method for a 4x4 dot matrix.

4.7 Displaying numbers

For the maze game, you are required to select one of 4 pre-defined mazes. You are required to display the number 1, 2, 3 or 4, using the buttons to cycle through the choices. The numbers must be displayed as in Figure 9. You must also display the number on the debug LEDs (maze 1 on D2, maze 2 on D3 etc.).

4.8 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.

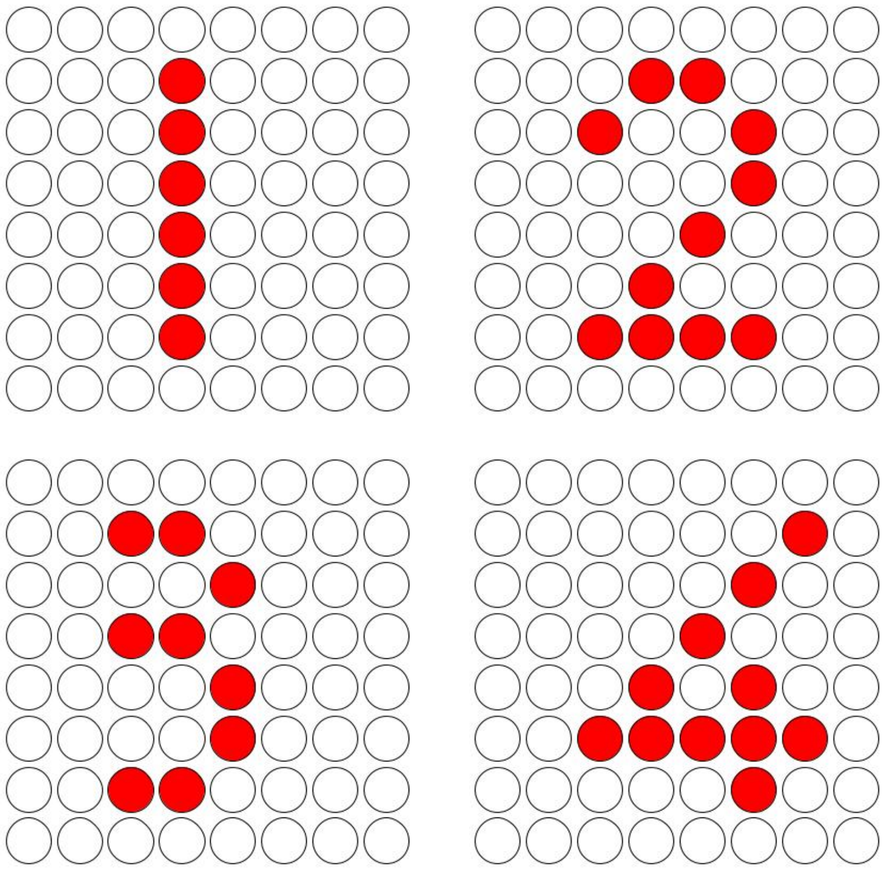


Figure 9: Displaying numbers on the dot matrix.

```
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 */
}
```

4.9 UART

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 only in one direction: 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 the test station purposes, it will be necessary to route the UART2 output to the Test Interface Connector (TIC). In this case, you will have to make a connection from the correct pin of the Nucleo board to the Test Interface Connector (TIC) (see Section 3.6 and 5.1). Both methods are shown in Figure 10.

The format of the UART receive HAL functions are 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

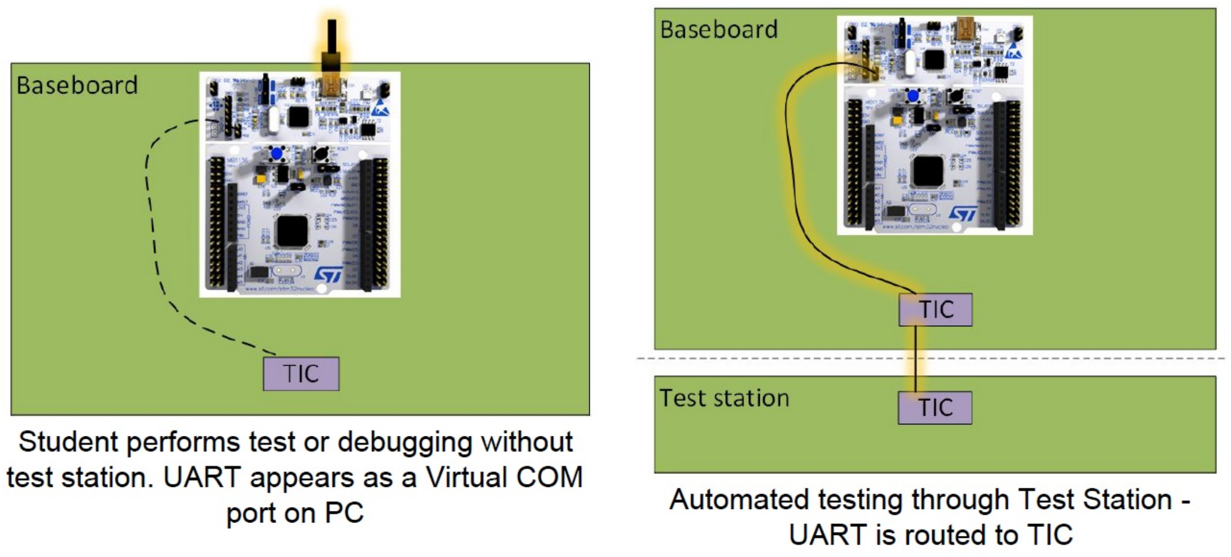

Baseboard
TIC
Student performs test or debugging without test station. UART appears as a Virtual COM port on PC
Baseboard
TIC
Test station
TIC
Automated testing through Test Station - UART is routed to TIC

Figure 10: UART connections for debugging and testing.

receiver.

For UART transmission we can use standard block transmits.

4.10 I2C Communications

The I2C communications on the STM32 family is well supported by the HAL libraries, which remove most of the complexity from the user. The STM32F103 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 are 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).

4.11 SPI Communications

This protocol is not used in this project (Use I2C to interface to the accelerometer).

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 Suggested game code structure

We recommend having a c-code structure for each game. In the structure you can have variables to keep track of parameters such as the ball position, bat position, the maze outline etc.

4.14 Suggested dot matrix code structure

We recommend creating a global buffer that stores the required state of each LED. This buffer will be periodically updated depending on what game is being played etc. We further recommend calling a function to update the state of LEDs every millisecond using the sysTick timer (see stm32f1xx_it.c). In this function, you will update one row or column of your dot matrix. Therefore it will take 8ms to update the whole dot matrix. You need to devise a method of keeping track of what row/col needs to be updated next.

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

    /* USER CODE END SysTick_IRQn 1 */
}
```

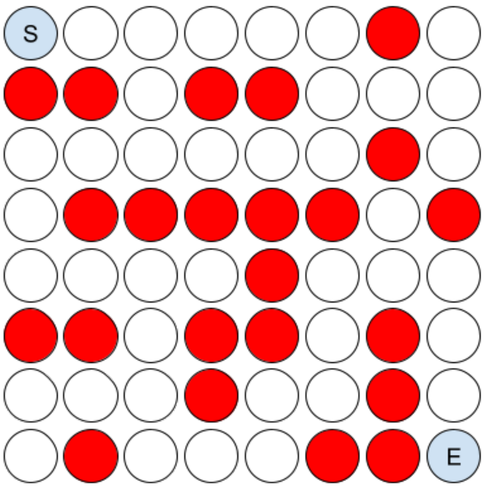
4.15 Auto code generation

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.

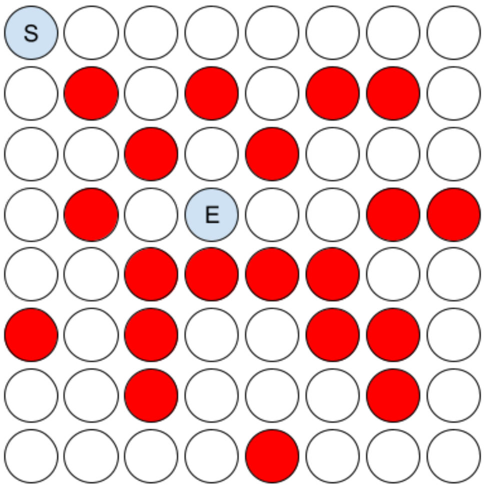
> ⚠ *We recommend only doing this once at the start of your project! Back up your code frequently to avoid loss of code!*

4.16 Pre-defined mazes

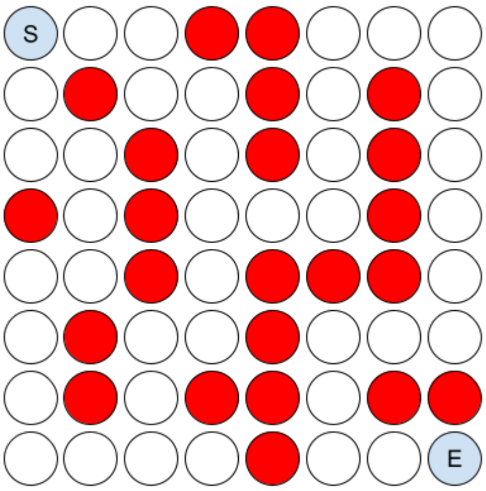
In total, four different mazes can be selected. They are shown in Figure 11.



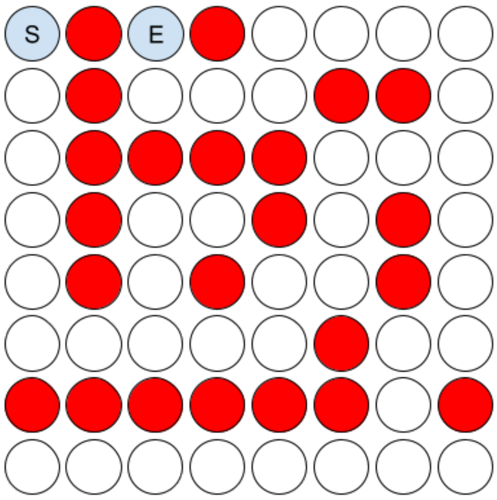
(a) Maze 1



(b) Maze 2



(c) Maze 3



(d) Maze 4

Figure 11: The 4 maze options.

5 Testing

Your student board (SB) will be tested during demo sessions, as specified in the study guide. 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.

Pin numbering of J13, J3 and P13 are as shown in Figure 12.

From your board, you will have to route the signals listed in Table 3 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	5V supply from student board (fed back to test station for verification/measurement)	SB -> TS
4	J13 - 3,4	ADC signal from slider (fed back to test station for verification/measurement)	SB -> TS
5	J3 - 5,6	3.3 V supply from student board (fed back to test station for verification/measurement)	SB -> TS
6	J13 - 5,6	**UART transmit (TX from student board, RX of test station)**	SB -> TS
7	J3 - 7,8	IMU SDA	SB -> TS
8	J13 - 7,8	IMU SCL	SB -> TS
9	J3 - 9,10	N/C	
10	J13 - 9,10	Button left (fed back to test station for verification/measurement)	SB <-> TS
11	J3 - 11,12	Button right (fed back to test station for verification/measurement)	SB <-> TS
12	J13 - 11,12	Button up (fed back to test station for verification/measurement)	SB <-> TS
13	J3 - 13,14	Button down (fed back to test station for verification/measurement)	SB <-> TS
14	J13 - 13,14	Button middle (fed back to test station for verification/measurement)	SB <-> TS
15	N/C	GND	SB <-> TS
16	N/C	GND	SB <-> TS

Table 3: Test-interface Connector Pin definitions

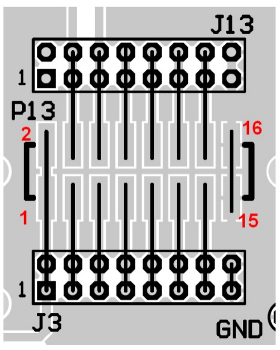


Figure 12: Pin numbers of J13, J3 and P13

Connections shown in grey in Table 3 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 4

5.3 UART communications interface

The UART shall operate at a baudrate of 115200 bps, with 8 data bits, 1 stop bit and no parity checking, using TTL signal levels.

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 Teraterm and Realterm, in the absence of a test station.

The UART will be used to transmit from the SB to the PC or TS. The SB will output a message with the format as in table 5. The message length is fixed at **10 (offset 0 - 9)** characters, and all the fields in the message also have fixed length. The message starts with a '$' character and ends with a newline character (ASCII character code 10, or indicated in C string escape notation as '\n').

5.4 Demos

There will be a total of 4 demos which will test all the user requirements:

UR	Method	Pass/Fail
UR1	TS supplies board with 9V switched power supply. SB will feed back the generated 5V and 3.3V supplies using TIC connector.	Test passes if TS measures the 5V and 3.3V supplies to be in 5% of expected values.
UR2	The TS will sample the sensors (through the TIC) at the required time, it will "play" the same game and compare its output with your output.	Test passes if TS has same output on LEDs as SB.
UR3	The TS will record the UART message and compare its data to the LEDs	Test passes if data matches the LEDs.
UR4	An image of the SB will be taken. The calibration sequence will be filmed	Test passes if each LED is equal brightness, the matrix is the correct size and in the correct location on the SB. Individual lights can be commanded and updated every 8ms (1ms per column).
UR5	TS will override button signals on SB via TIC.	Test passes if buttons are successfully debounced and if the correct action occurs due to button press.
UR6	The TS will override the buttons and will play the game. The TS will record the data received via UART, take pictures of the dot matrix and track where its ball is.	Test passes if the SB UART and dot matrix matches that of the TS ball position. All 4 mazes will be tested.
UR7	The TS will monitor the ADC/IMU and UART while the user plays the game.	Test passes if the SB UART information matches that of the TS.
UR8	This will be manually marked by a lecturer. The lecturers board will be the slave, you are required to initiate the game and show two player mode working.	Test passes if the ball can be passed between players multiple times (using buttons, slider or accelerometer).
UR9	During a game, the TS will override the buttons and exit a game.	Test passes if game successfully exits and calibration sequence (light up the 4 corners of the display) is displayed

Table 4: Test station test method definitions

MESSAGE	0	1	2	3	4	5	6	7	8	9
StdNum	‘$’	Student number, 8 bytes								‘\n’
Calibration	‘$’	‘1’	Column	‘_’	‘_’	‘_’	‘_’	‘_’	‘_’	‘\n’
Tennis	‘$’	‘2’	X pos ball	Y pos ball	Velocity	Direction	X pos bat	Y pos bat	IMU	‘\n’
Maze	‘$’	‘3’	X pos	Y pos	Visible ball	Visible goal	IMU	‘_’	‘_’	‘\n’

Table 5: UART Message fields

Demo	Description	UR's
1	On power up, transmit your student number via UART. The TS will monitor and test the 3.3V and 5V line. Perform the calibration sequence (scan through column by column, 1 second per column, outputting the current column on UART) for a 4 by 4 matrix.	UR1, UR3.1, UR4 (4x4), UR9 (no-button)
2	Expand the dot matrix to an 8x8 dot matrix. On power up, transmit your student number via UART and perform the calibration sequence. Use the middle button to start and end the "game". The "game" consists of moving the ball left and right with buttons, up and down with the slider (when the output of the slider is at supply, the ball must be at the top and vice versa). The ball starts on light [4,4] and cannot move off the screen. Move the ball every 100 ms and transmit the position of the ball via UART every 100 ms. When not in game mode, show the calibration display (light up the 4 corners of the display).	UR2.3, UR3.2, UR3.4, UR4
3	On power up, transmit your student number via UART and perform the calibration sequence. Use buttons to select one of two games (maze (left button) or tennis single player (middle button)). The maze game will have a default maze (automatically start on maze 1 without selecting a maze) and you must move the ball using buttons. During the tennis game, you can only move the bat up and down using the slider (no left/right movement).	UR2.2, UR3.3, UR5, UR6, UR7, UR9
4	Same as demo 3, but with an option of 4 mazes. You will use buttons to select the maze. You will use the IMU to control the bat in tennis (left, right, up and down), and the ball in the maze. Adding 2-player tennis will generate extra marks. Should the IMU not work, buttons can be used, but marks will be subtracted.	UR2.1, UR6, UR7 UR8, UR9

Table 6: Demo descriptions

6 “Specification” Report

Due to the nature of this year’s project, and specifically the schedule limitations, the peer reviewed “Specification” report has been changed to a peer reviewed “Student” report that is based on the two tasks described in the following sections.

6.1 Task 1

To get you used to the design process, you have to do a preliminary concept design for the entire system - select components and justify their use. Document your effort by drawing a system block diagram and include the necessary explanations in a short report (max 2 pages). You may make any reasonable assumptions from the user requirements. Your diagram and explanation should clearly show the power and communication interfaces between different components. (a detailed circuit diagram or schematic is not required).

6.2 Task 2

Perform a full design of the LED matrix section of the device. The specifications listed in this document should be adhered to. Make sure you justify the need for MOSFETs, calculate the current through the LEDs (as well as the current sourced and sunk by the GPIOs) and the duty cycle of the LEDs (provide some details on timing and updating the matrix).

You have to document the following (3 pages max):

- Design choices and their justification
- Circuit diagram (schematic) of the LED matrix
- Test method(s), to ensure the specifications are adhered to

6.3 Assessment of task 1 and 2

These two tasks will be assessed through peer assessment - you will have to mark three other reports, and your report will be marked by three 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 counts 33.3% of the final mark for the module, and these two tasks and peer evaluation of them will contribute 10% of the report mark.

Misconduct in carrying out the peer evaluation (i.e., if you make up random marks, or fail to complete the peer evaluation) will be penalized.

6.4 Hand-in

A single document with both tasks (max 5 pages) should be submitted to SunLearn by Friday 14 May at 14:00. Only PDF format will be accepted.

6.5 Rubric

Criterion	Description	Mark
1.1	Based on the block diagram, it should be sufficient for you, the reviewer, to see how the student plans to implement the system - i.e., which parts does it consist of, and how do they connect and interact.	2
1.2	Based on the explanation that is given in the report, you, the reviewer, should be able to understand why the student decided on specific parts or components and their interfaces.	3
2.1	Based on the explanation given in the report, has the use of MOSFETs been sufficiently justified? Have they implemented the MOSFETs correctly in the circuit diagram?	3
2.2	Calculations must be provided to determine the GPIO current that is sourced/sunk as well as the LED current and duty cycle.	3
2.3	Details should be provided on how the timing will be achieved and how the LED matrix is updated (Timers? interrupts? Column scan? etc)	2
2.4	Testing method should detail the logical approach to building and testing the dot matrix display.	2

Table 7: 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 (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, 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
2. System description/concept design
3. Hardware design and implementation
4. Software design and implementation

5. Testing
6. 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 8

Report Element	Max. points available
Language and grammar	3
Table of contents, list of figures, tables, abbreviations, etc.	2
Overall presentation/impression	2
Sub-total	7

7.1.2 Introduction

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.

Marks will be awarded for:

Table 9

Report Element	Max. points available
Summary of requirements/user needs	2
High-level description of system	2
Sub-total	4

7.1.3 System description / Concept Design

- 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 10

Report Element	Max. points available
Block diagram	2
How system operation is achieved through interaction between blocks. Describe interfaces between blocks	4
Sub-total	6

7.1.4 Hardware design and implementation

- 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 11

Report Element	Max. points available
Power supply	2
UART communications (protocol and timing)	2
Buttons	2
LEDs (debug)	1
Dot Matrix	2
Slider	1
IMU	2
Sub-total	12

7.1.5 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 state (tennis game, maze game).
  - 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 12

Report Element	Max. points available
High-level description of program	1
Control logic	2
Button bounce handling	1
Data flow and processing	3
IMU interface	2
Peripheral setup (timers, ADC, UART, I2C etc)	4
Sub-total	13

7.1.6 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 13

Report Element	Max. points available
Power supply	2
UART communications	2
Buttons	2
LEDs (debug)	1
Dot Matrix	2
Slider	2
IMU	2
Sub-total	13

7.1.7 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 14

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

7.1.8 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.

Marks will be awarded for:

Table 15

Report Element	Max. points available
Complete schematic	3
STM32 pins used and their configuration	1
References	1
Sub-total	5

7.1.9 Bonus marks

- Complete description of the two player tennis game.
- Provide a protocol for the UART communication.

Marks will be awarded for:

Table 16

Report Element	Max. points available
Complete description	1
Protocol for UART communication	2
Sub-total	3

7.1.10 Marks Summary

Table 17

Report Element	Max. points available
Overall	7
Introduction	4
System description/Concept	6
Hardware design and implementation	12
Software design and implementation	13
Measurement and results	13
Conclusion	3
Appendices	5
Bonus marks	3
Total	63

References

[1] *RM0364 Reference Manual, STM32F103xx advanced Arm$^{®}$-based 32-bit MCUs*, ST Microelectronics, DocID025177, Rev 3, filename=`en.DM00093941.pdf`, September 2017.

[2] *NUCLEO-F103RB*, ST Microelectronics, `https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus/stm32f1-series/stm32f103/stm32f103rb.html`, accessed 2021-03-04.