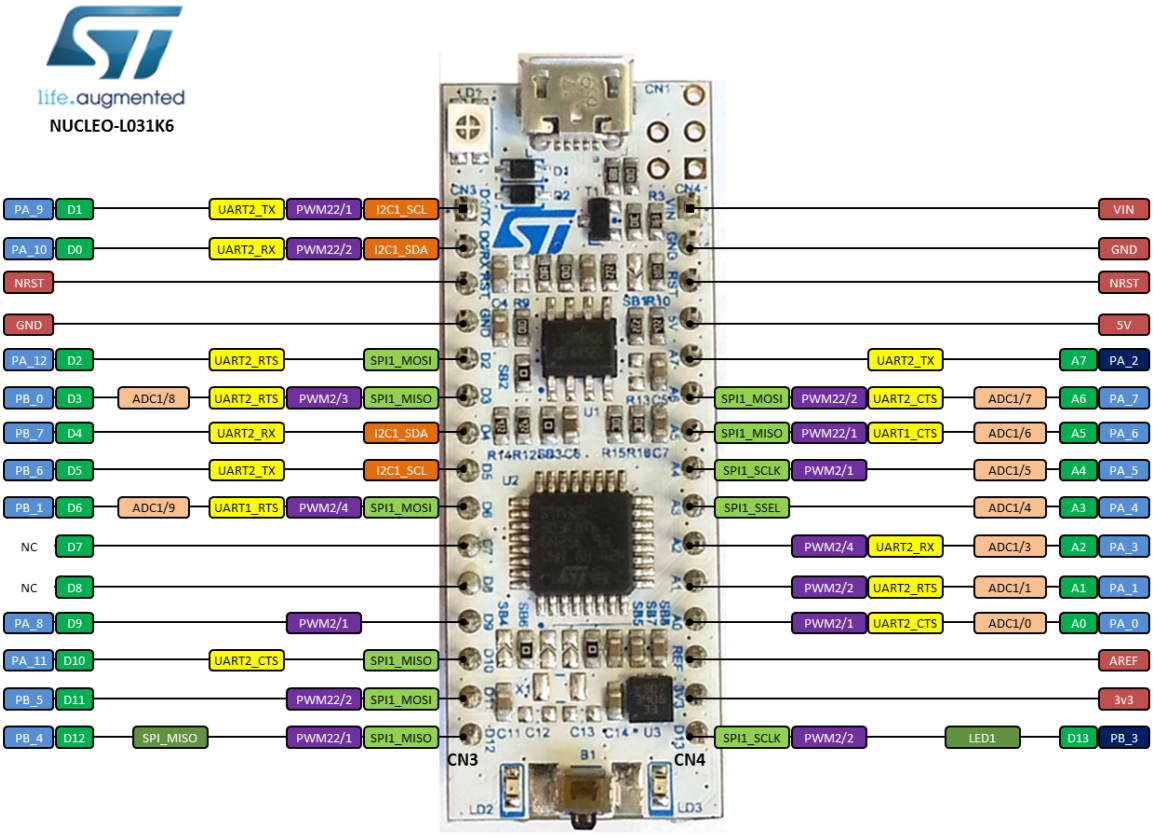
Device Test

Constructing a Test Suite for Commonly Used STM32 Peripherals

(Joel Taina)

Project Guide - Rev. 1



Contents

List of Figures

List of Tables

Revision Log

Date	Revision	Changes
23-Feb-2022	1	Initial Release

Table 0.1: Revision Log

1 Abstract

1.1 Executive Description

The project “Device Tests” is a test suite for commonly used STM32 peripherals. There are seven tests to choose from; users may configure the settings in the user application file `main.h` to select the test and pins required. Users may select a test for various electronic components such as a LCD text display (all tests), buttons, and potentiometer-based dials, as well as STM32 peripherals, such as GPIO, TIM, EXTI, I2C, ADC, and DMA. The project is intended as a demonstration tool; users may adopt portions of the firmware for other projects as desired. Link to GitHub repository here:

https://github.com/tainaj/baremetal-stm32

This project is dedicated to Vivonomicon, who created the “STM32 Baremetal Examples” blog series that inspired me to continue practicing my embedded software development skills that I initially gained from college. Link to his blog series here:

https://vivonomicon.com/category/stm32_baremetal_examples/

1.2 User Guide

This document is a guide for the project, going over hardware considerations, explanations for all source/header files used, and a detailed overview for each test, with pictures included. Each test demonstration will use a NUCLEO-L031K6 development board; other boards may be used. Links to ST documentation will be in the Bibliography section.

2 Hardware overview

2.1 STM32 Development Boards

This project is applicable to the following STM32 development boards:

- STM32F0DISCOVERY – a Discovery kit that includes everything required for beginners and experience users to get started quickly (source: STM32 site). All peripherals used by the project are supported by external pins.
- NUCLEO-L031K6 – a Nucleo-32 development board that provides an affordable and flexible way for users to try out new concepts and build prototypes (source: STM32 site). All peripherals used by the project are supported by external pins, though some peripheral mappings are unavailable.

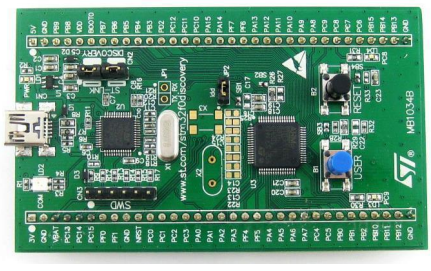
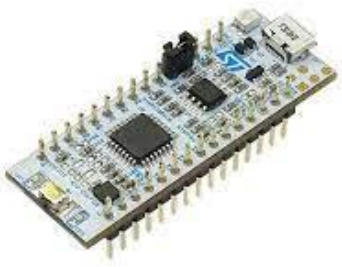
	STM32F0DISCOVERY	NUCLEO-L031K6
Microcontroller	STM32F051R8T6 [1]	STM32L031K6T6 [2]
Core	Cortex-M0	Cortex-M0+
Flash memory	64 KB	32 KB
SRAM	8 KB	8 KB
Max frequency	48 MHz	32 MHz
Package	LQFP64	LQFP32
Image		

Table 2.1: Hardware specifications of two STM32 development boards

Note: Feel free to try out other boards. Just be aware of the available GPIO pins, peripherals, and other stuff.

2.2 Digital Input Devices

This Project uses the following digital and analog input devices (see corresponding link for schematic example) (or add picture of device here)

- 2 x Push buttons – consists of two halves of pins connected by a SPST (single-pole-single-throw) switch, which is closed when the button is pressed.
- 1 x A3144 Hall effect switch – consists of a ground pin (left), source pin (right), and open-drain output (center), which is pulled low when a strong magnetic field of a particular orientation is detected.
- 2 x 10 kOhm potentiometer – consists of a ground-source pin pair (select either edge pin as ground or source), and output (center), which is set to a variable analog value between source and ground, dependent on the dial position.

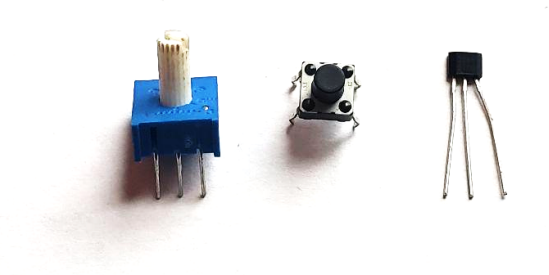


Figure 2.1: Three digital input devices, from left to right - 10 kOhm potentiometer, push button, A3144 Hall effect switch

Other digital and analog input devices may be used instead of those listed above. The above are intended as examples.

A diagram for a hardware debounce circuit is also provided here; connect the output pin of the switch to the green circle on the diagram below:

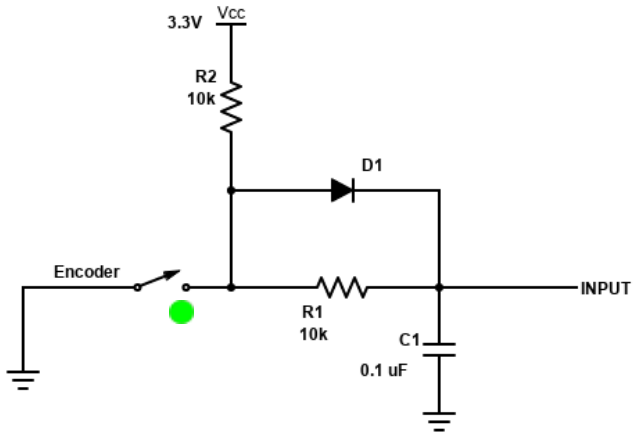


Figure 2.2: Hardware debounce circuit for digital input devices

2.3 Digital Output Devices

This Project uses the following digital output devices: (add pictures)

- 3 x LEDs – these may already be attached to the board (see datasheet), or can be externally connected.
- 1 x TC2004 LCD text display – used to display text output from the project. Connected to the STM32 via I2C interface (see software overview for available pins.

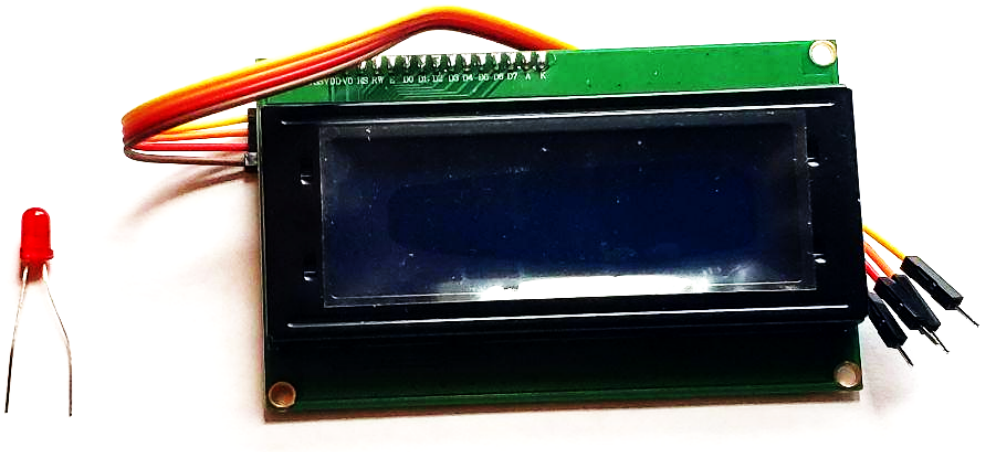


Figure 2.3: Front view of TC2004 LCD text display, and LED

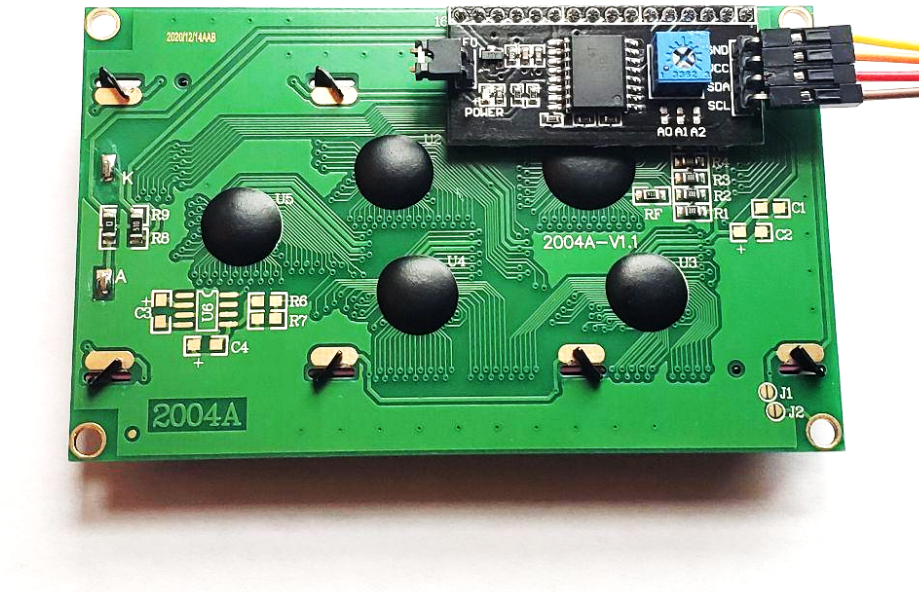


Figure 2.4: Back view of TC2004 LCD text display, with I2C connector exposed

Other digital input devices may be used instead of those listed above. The above are intended as examples.

3 Software Overview

3.1 Header file dependency graph

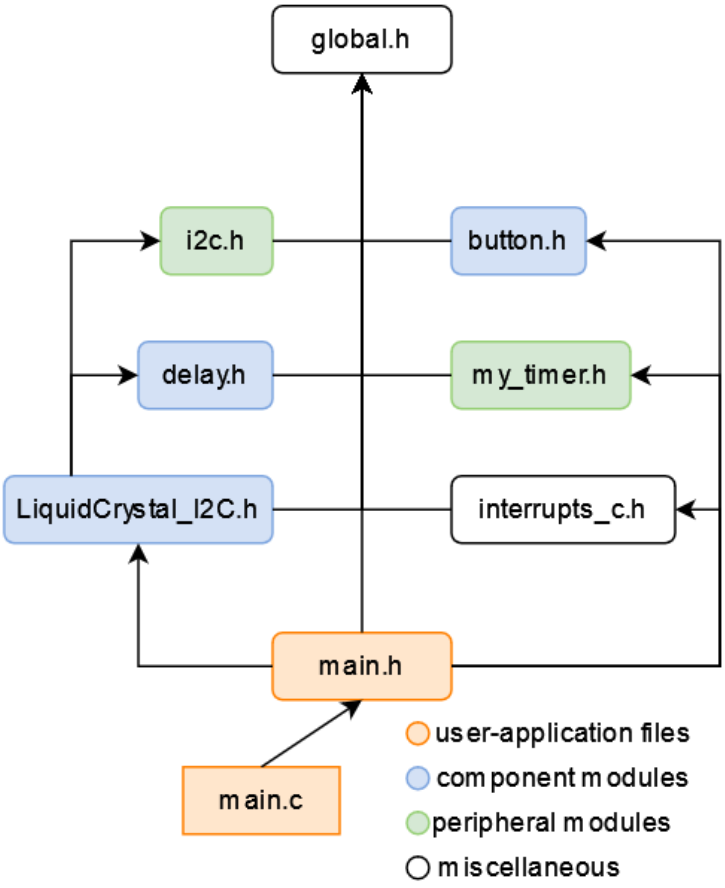


Figure 3.1: Header file dependency graph

Note 1: Every header file underneath global.h is included by their corresponding source file (not shown)

Note 2: The top header file `global.h` includes CMSIS files for STM32 devices.

3.2 User application files

3.2.1 main

The main.h header file contains definitions for the test subroutine used, as well as the definitions for every peripheral register and pin used for those tests. Currently, test numbers 1 to 7 are available. To determine which pins to use for a test, consult the section in this document for said test. Notes are also provided over the definitions for additional guidance.

Main.h uses resources referenced with the following #include 's:

- Global.h – Interrupt flag bits, other shared resources (see global.h section)
- LiquidCrystal_I2C.h – control of LCD text display
- Button.h – control of button input with software debounce (used in Test 6)
- My_timer.h – custom setup module for TIMx peripherals
- Interrupts_c.h – Interrupt service handlers

The main.c source file contains subroutines for each test (lines 179-672), as well as support functions and the main subroutine run by all tests (lines 3-177). Starting from main(), the general setup for each test is as follows:

- Clock configuration (lines 680-732). The `core_clock_hz` global variable is configured to the target device's top speed (F0 devices set to 48 MHz, L0 devices set to 32 MHz)
- Enable I2Cx peripheral and pins used for LCD text display (lines 734-752)
- Initialize delay module (lines 754-758). Used by LCD text display module.
- Start up LCD text display (lines 762-765). Print initial message on top line.
- Call the appropriate test subroutine (line 768). See Section X.1: Tests, for more.

3.3 Component modules

3.3.1 LiquidCrystal_I2C

This module is a C++ to C conversion of a LiquidCrystal Arduino library for I2C LCD displays, sourced from John Rickman [5].

LiquidCrystal_I2C.h uses resources referenced with the following #include 's:

- Global.h – shared resources (see global.h section)
- I2c.h – implementation of I2C functions used to print text (i2c_init, i2c_start, i2c_stop, and i2c_write_byte)
- Delay.h – implementation of delay() (in microseconds)

The following modifications to the original library are as follows:

- To preserve the concept of C++ class instances from the original library when transcribing to C, the parameter “this”, a struct instance of LiquidCrystal_I2C that contains class variables, is passed to all class functions.
- Public functions are prefaced with “LC_I2C_”, indicating the class name in the original. (ex. Print() → LC_I2C_print(this, ...)).
- Two print functions (not including printByte) are included:
    - LC_I2C_print(str) – intended for printing a constant character string `str` (ex. “Hello

World!")
    - LC_I2C_print_pad(str, length) – intended for printing a character buffer `str` at desired length, while ignoring `\0` characters in between (ex. Test 6 char buffer with 3 counters, with '\0' between each)

3.3.2 delay

This submodule implements time-tracking functions for either microsecond or millisecond use. Three uses exist:

- Track time elapsed between start and end points – use millis() or micros() to generate a start time, and compare a later measurement with an expected elapsed time.
- Delay for a duration of time– use delay() or micro_wait() for microsecond durations, or use milli_wait() for millisecond durations.

Note: The SysTick timer is used to track milliseconds. To track microseconds, select a timer peripheral (this project uses TIM2) to use.

delay.h uses resources referenced with the following #include 's:

- Global.h – shared resources (see global.h section)

3.3.3 button

This submodule implements 8-sample software debouncing for digital input devices (push button, Hall effect sensor, etc.). Any number of inputs may be configured; the project uses three inputs. To use the button submodule:

- Initialize button inputs – With initInputs(), initialize the module to any number of inputs, and specify the input and output GPIO pin numbers (output typically connected to LEDs)
- Update inputs – with updateInputs(), upon receiving a button press or release, toggle the corresponding LED output.

button.h uses resources referenced with the following #include 's:

- Global.h – shared resources (see global.h section)

3.4 Peripheral modules

3.4.1 i2c

This module implements the following I2C interface for the STM32:

- i2c_init(addr) – initializes an I2C peripheral to communicate to device with I2C address `addr`
- i2c_start() - send 'Start' condition, and wait for acknowledge
- i2c_write_byte(dat)
- i2c_stop() - send 'Stop' condition, and wait for acknowledge

3.4.2 my_timer

This module implements the following TIM interface for the STM32 used by all tests:

- reset_timer() - enable the timer on RCC, turn off the timer, reset the peripheral

- start_timer() - configure prescaler/autoreload values, enable update interrupt, enable timer
- start_timer_psc_timer() - start two timers, with one timer prescaled by the other (Test 4). Requires:
    - master_timer() - configure timer to master mode (also used in Test 7)
    - slave_timer() - configure timer to slave mode
- ccr_timer() - enable all channel outputs available for TIMx. Requires:
    - ccr_count() - returns number of channels available for TIMx

3.5 Miscellaneous

3.5.1 interrupts_c

This module was created to house each interrupt service handler called by the STM32. Each ISR sets the corresponding bit on the global variable **flags** (see global.h).

3.5.2 global.h

Global header linked by all header files in the project. Contains:declaration of interrupt status variable **flags**, as well as device header files “stm32f0xx.h” or “stm32l0xx.h” (depending on Makefile)

4 Tests

This project contains seven test subroutines which demonstrate common peripheral features. In main.h, modify `TEST_NO` to indicate the test number to run:

1. Still text – demonstrates the TC2004 LCD I2C text display (used by all tests).
2. Toggle 2 lines – Two lines of text near the bottom toggle on and off, alternating one over the other.
3. Counter – A TIM peripheral is configured to increment a counter value every 0.5 seconds. All available channel outputs are enabled.
4. Double clock counter – Two TIM peripherals are configured to increment a counter value every five seconds, one acting as prescaler for the other.
5. EXTI button – a button release increments a counter value. One button.
6. 3 buttons – a button release, with an 8-sample software debounce, increments its counter value. Three buttons.
7. 2 potentiometers – an ADC peripheral reads the value for each potentiometer every 1 ms, ranging from 0 to 255.

Test subroutines generally follow this pattern:

- Print test label on 2nd line, indicating the name of the test and test number.
- Configure GPIOx pins to input (button, Hall effect sensor, etc.) or output (LEDs)
- Configure other peripherals (TIMx, EXTIx, ADC). Setup a chain of events as needed. Examples of a chain of events: (replace with picture of flow chart)
    - Test 2: TIMx → interrupt (set **flags**)
    - Test 3: TIMx + CCR → interrupt
    - Test 7: TIMx → ADC → DMA → interrupt
    - See each test's section to understand why the chain of events are as they are.
- Print initial content for 3rd or 4th lines, depending on test number.
- Forever loop. When a bit from **flags** is set, acknowledge **flags** bit by clearing it and performing an update action (depends on test number).

4.1 Test 1 - Still text

4.1.1 Overview

This test demonstrates the TC2004 LCD I2C text display, a device that displays string data. The string “1: Still text” is printed.

4.1.2 Procedure

1. Select the I2C peripheral (I2C_X, I2C_X_GPIOx, PX_SCL, PX_SDA)

MCU class	Peripheral	AFy -> (SCL, SDA)		
F0	I2C1	AF1 -> (PB6, PB7)	AF1 -> (PB8, PB9)	
	I2C2	AF1 -> (PB10, PB11)		
L0	I2C1	AF1 -> (PA9, PA10)	AF1 -> (PB6, PB7)	AF4 -> (PB8, PB9)*

Table 4.1: I2C peripheral mappings for each MCU class [1, pp.37-38] [2, pp. 45-46]

GPIO pins for the pair (PB8, PB9) are not available for NUCLEO-L031K6.

4.1.3 Results

The LCD text display should display the words “1: Still Text” on the 2nd line.

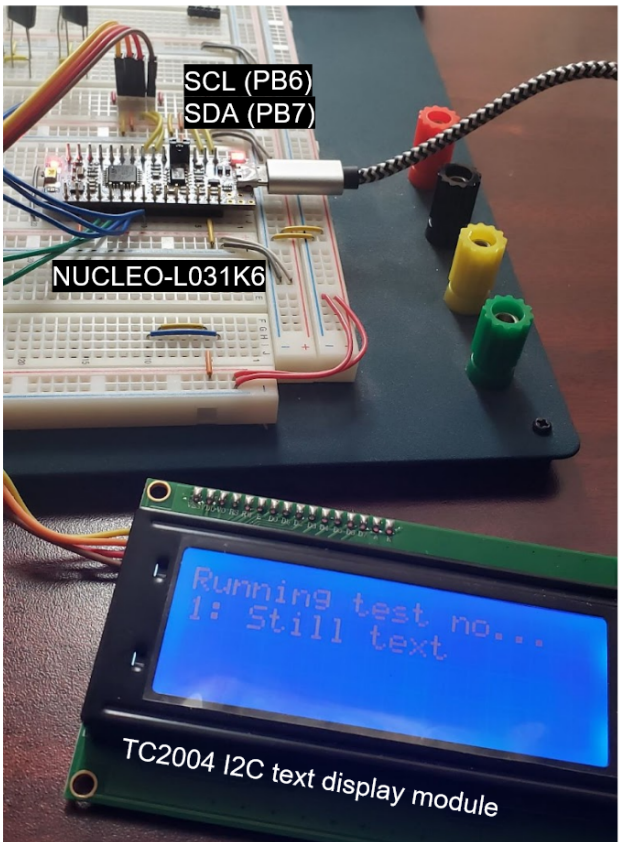


Figure 4.1: Breadboard setup of Test 1

4.2 Test 2 - Toggle 2 lines

4.2.1 Overview

Two lines of text near the bottom toggle on and off every second, alternating one over the other.

TIM_X update interrupt → LED toggle

4.2.2 Procedure

1. Select the I2C peripheral (See Test 1)
2. Select a TIM peripheral (TIM_X, TIM_X_IRQn*)

MCU class	Peripherals				
F0	TIM3	TIM14	TIM15	TIM16	TIM17
L0	TIM21	TIM22			

Table 4.2: Available TIM peripherals for each MCU class [1, p.22] [2, p. 29]

The format for the name of TIM_X_IRQn is {TIM_X}_IRQn
TIM2 is reserved for the delay module

3. Select an output GPIO peripheral (LED_GPIOx, LED_PIN_1. The following pins are available:
    a. STM32F0DISCOVERY - PA0-15, PB0-15, PC0-15
    b. NUCLEO-L031K6 - PA0-12, PB0-1, PB3-7

4.2.3 Results

The LCD text display should display the words “2: Toggle 2 lines” on the 2nd line.

- The green LED (the yellow circle) toggles on or off once every TIM22 update interrupt.
- On the 3rd and 4th lines, the texts “Fryday \ STM32 Baremetal” and “Hello World! \ Cookie!” should toggle also.

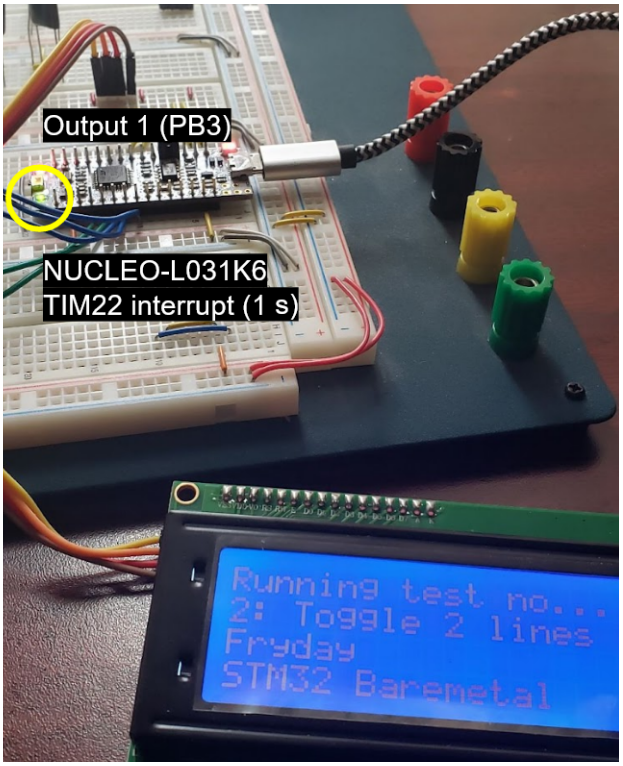


Figure 4.2: Breadboard setup of Test 2

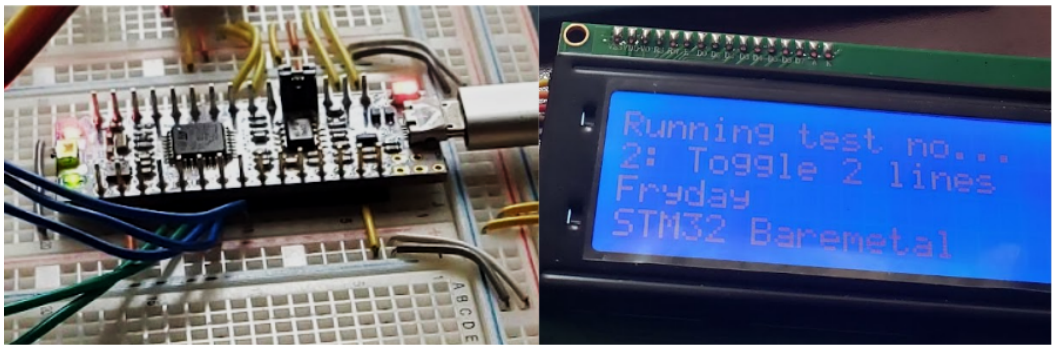


Figure 4.3: Text display with LED toggled on

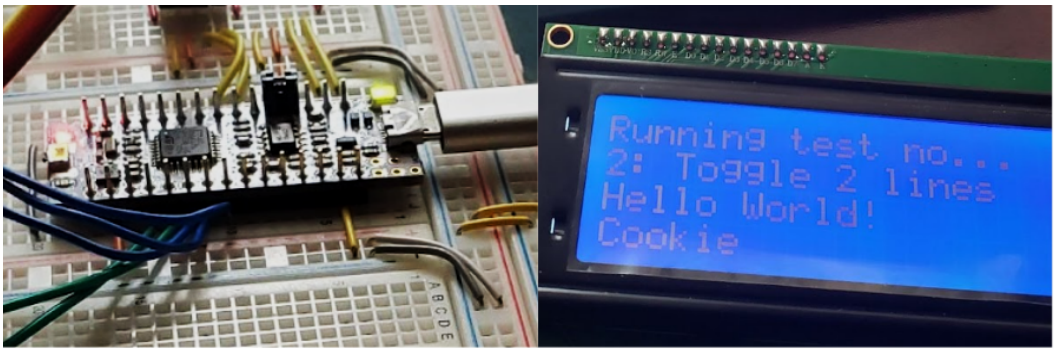


Figure 4.4: Text display with LED toggled off

4.3 Test 3 - Counter

4.3.1 Overview

A TIM peripheral is configured to increment a counter value every 0.5 seconds. All available channel outputs are enabled, resulting in a run of toggles from each channel before the main LED output toggle.

CH1 toggle → CH2 toggle → CH3 toggle → CH4 toggle → TIM_X update interrupt → LED toggle

4.3.2 Procedure

1. Select the I2C peripheral (see Test 1)
2. Select a TIM peripheral (see Test 2)
3. Select the main output GPIO peripheral (see Test 2)
4. Select the TIM channel output GPIO peripheral (TIM_X_CCR_*)
    a. GPIOx, AFy, PIN_1, PIN_2, PIN_3, PIN_4
    b. Not all TIM peripherals use PIN_2, PIN_3, and/or PIN_4. Ignore if not needed.

MCU class	TIM_X	PIN_1 (CH1)	PIN_2 (CH2)	PIN_3 (CH3)	PIN_4 (CH4)
F0	TIM3	PA6	PA7		
		PB4	PB5	PB0	PB1
	TIM14	PA4, PA7			
		PB1			
	TIM15	PA2	PA3		
		PB14	PB15		
	TIM16	PA6			
		PB8			
	TIM17	PA7			
		PB9			
L0	TIM21	PA2	PA3		
		PB6			
	TIM22	PA6, PA9	PA7, PA10		
		PB4	PB5		

Table 4.3: TIM channel output mappings for each MCU class [1, pp.37-38] [2, pp. 45-46]

For a given TIM peripheral, all channels selected must be of the same GPIO peripheral (TIM_X_CCR_GPIOx)

If you wish to disable a channel, or the number of channels are fewer than the maximum available

for the TIM peripheral, go to line 306 and replace the standard value with TIM_DISABLE for all unused channels.

4.3.3 Results

The LCD text display should display the words “3: Counter” on the 2nd line.

- Once every TIM22 update interrupt, the green LED (bottom yellow circle) toggles on or off, and a printed counter value increments by 1.
- After this toggle, the red timer channel LEDs (top yellow circle) toggle after each other, starting from CH1, in increasing order.

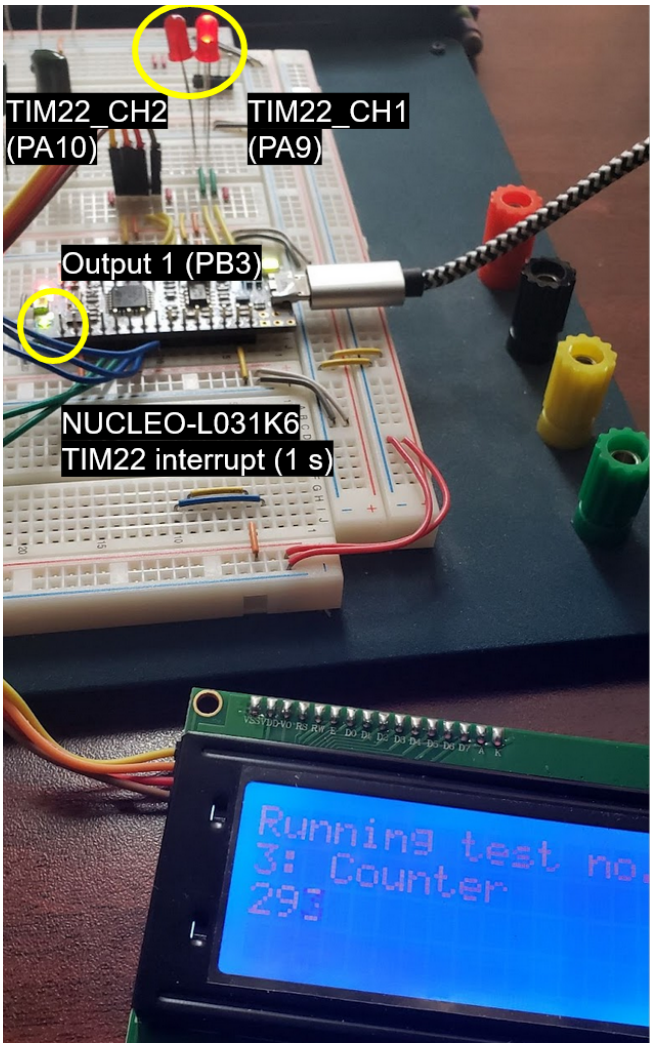


Figure 4.5: Breadboard setup of Test 3

4.4 Test 4 - Double clock counter

4.4.1 Overview

Two TIM peripherals are configured to increment a counter value every five seconds, one acting as prescaler for the other. The prescaler timer is configured to increment the final timer once every second.

TIM_Y update → TIM_X update interrupt → LED toggle & counter++

4.4.2 Procedure

1. Select the I2C peripheral (see Test 1)
2. Select the main output GPIO peripheral (see Test 2)
3. Select the prescaler (TIM_Y) and final (TIM_X) peripherals (TIM_Y, TIM_X, TIM_X_IRQn, TIM_X_TS) from the available mappings below:

MCU class	TIM_Y	TIM_X	TIM_X_TS
F0	TIM3	TIM15	001 [3, Table 63]
	TIM15	TIM3	010 [3, Table 58]
L0	TIM21	TIM22	000 [4, Table 81]
	TIM22	TIM21	001 [4, Table 81]

Table 4.4: TIM acting as prescaler for the other, peripheral mappings

The format for the name of TIM_X_IRQn is {TIM_X}_IRQn

4.4.3 Results

The LCD text display should display the words “4: 2-clock counter” on the 2nd line.

- Once every TIM21 update, the TIM22 counter increments by 1.
- Once per TIM22 update interrupt, the green LED (yellow circle) toggles on or off, and the printed counter value increments by 1.

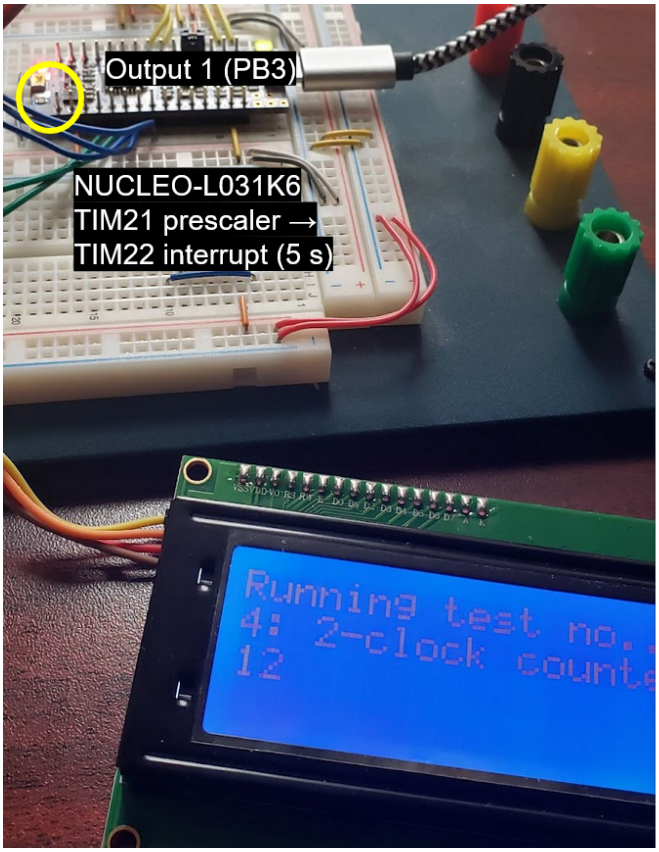


Figure 4.6: Breadboard setup of Test 4

4.5 Test 5 - EXTI button

4.5.1 Overview

Using an EXTI rising edge interrupt, a button release increments a counter value. One button is enabled. A hardware debouncing circuit is recommended for accurate counting (see Section 3.3).

Button release → EXTI interrupt → LED toggle & counter++

4.5.2 Procedure

1. Select the I2C peripheral (see Test 1)
2. Select an input GPIO peripheral (INPUT_GPIOx, INPUT_1) and output GPIO peripheral (LED_GPIOx, LED_PIN_1). The following pins are available:
    a. STM32F0DISCOVERY - PA0-15, PB0-15, PC0-15
    b. NUCLEO-L031K6 - PA0-12, PB0-1, PB3-7
3. Specify the EXTI interrupt number (EXTI_X_IRQn) for the selected input pin:
    a. PA0-PA1, PB0-PB1, PC0-PC1 → EXTI0_1_IRQn
    b. PA2-PA3, PB2-PB3, PC2-PC3 → EXTI2_3_IRQn
    c. PA4-PA15, PB4-PB15, PC4-PC15 → EXTI4_15_IRQn

4.5.3 Results

The LCD text display should display the words “5: EXTI button” on the 2nd line.

- When a button is released, the green LED toggles on or off.
- When a button is released, the printed counter value increments by 1.

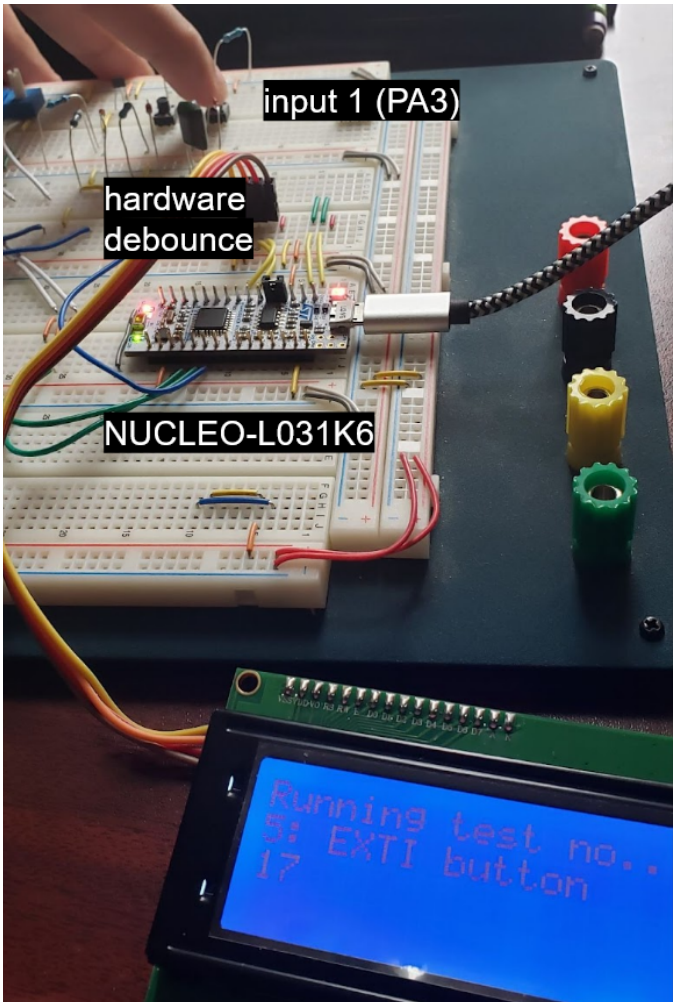


Figure 4.7: Breadboard setup of Test 5

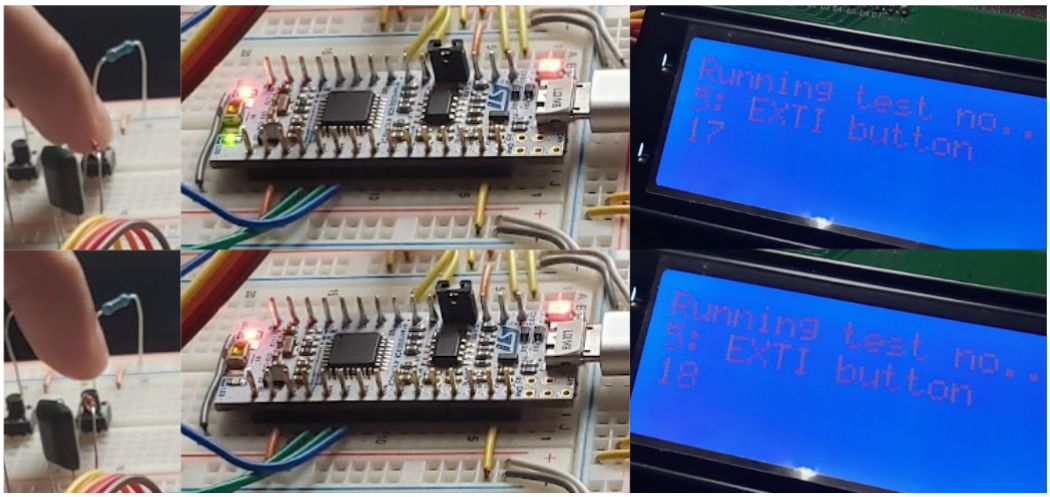


Figure 4.8: Button press turns LED on; button release turns LED off and increments counter

4.6 Test 6 - 3 buttons

4.6.1 Overview

Using a TIM interrupt, a button release increments a counter value. Three buttons are enabled. A software debouncing method is provided by the button module that registers a result after 8 samples spaced 1 ms apart, sufficient to eliminate signal bounce for this project's input devices.

Button press + TIM_X update interrupt → 8 checks → LED toggle

Button release + TIM_X update interrupt → 8 checks → LED toggle & counter++

4.6.2 Procedure

1. Select the I2C peripheral (see Test 1)
2. Select a TIM peripheral (see Test 2)
3. Select three input GPIO peripherals (see Test 5, but add INPUT_2, INPUT_3).
4. Select three output GPIO peripherals (see Text 5, but add LED_PIN_2, LED_PIN_3).

4.6.3 Results

The LCD text display should display the words “6: 3 buttons” on the 2nd line.

- Once per TIM22 update interrupt, the program checks the logic value for each button.
- Once per eight checks, the button module can register a button press or release, if detected.
- When a button press/release is detected, the corresponding red LED toggles on or off.
- When a button release is detected, the corresponding counter value increments by 1.

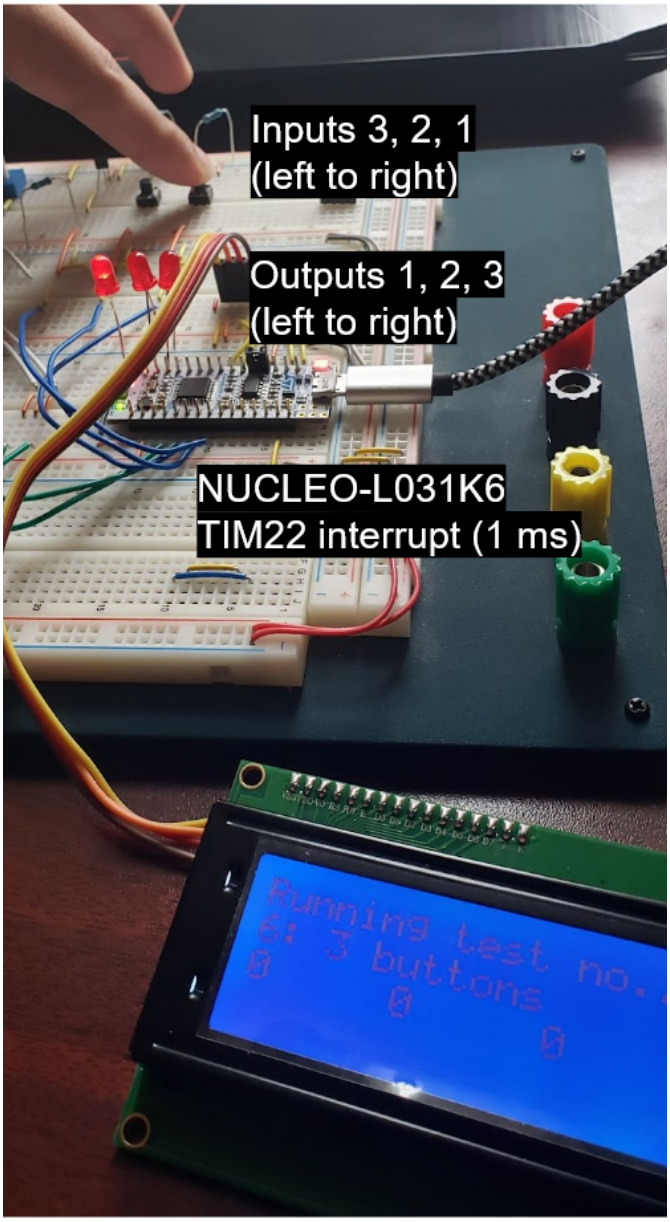


Figure 4.9: Breadboard setup of Test 6

Figure 4.10: Input 1 is right button, left LED, leftmost counter

Figure 4.11: Input 2 is left button, center LED, center counter

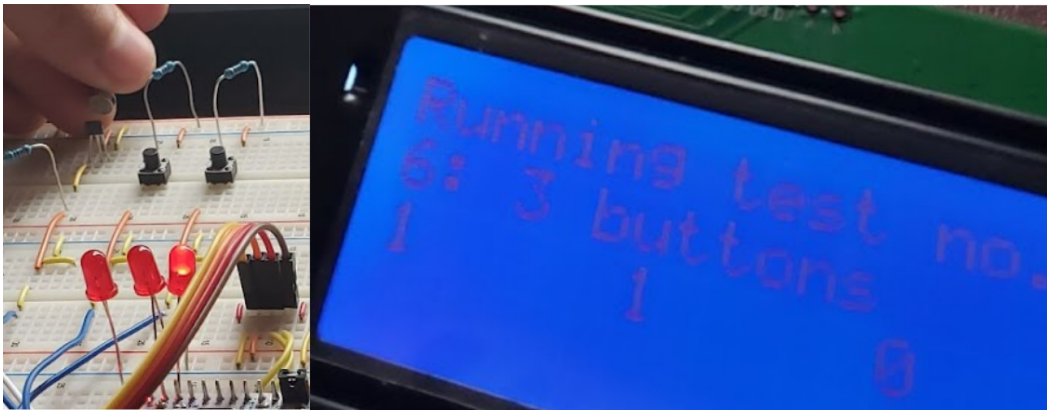


Figure 4.12: Input 3 is Hall effect switch, right LED, right counter

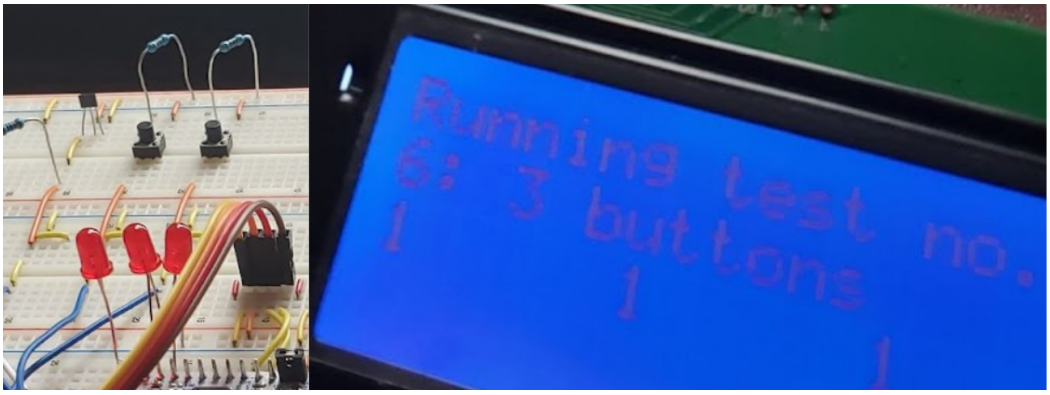


Figure 4.13: All inputs have been used once

4.7 Test 7 - 2 potentiometers

4.7.1 Overview

An ADC peripheral reads the value for each potentiometer every 1 ms, ranging from 0 to 255. To ensure accuracy while reducing processor load, 100 samples are recorded via DMA transfer, and the average is reported to the LCD text display.

TIM_X update interrupt → ADC + dial position → 100 DMA transfers → average reading

4.7.2 Procedure

1. Select the I2C peripheral (see Test 1)
2. Select two input GPIO peripherals (INPUT_GPIOx, INPUT_1, INPUT_2). EAch GPIO pin must be associated with an ADC channel (ADC1_INz_1, ADC1_INz_2), according to the following:

ADC_INz	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pin	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7	PB0	PB1	PC0	PC1	PC2	PC3	PC4	PC5

Table 4.5: GPIO mappings for each ADC channel [3, Table 13] [4, Table 15]

ADC channels 10-15 are not available for NUCLEO-L031K6.
Source: datasheets

3. Select the timer peripheral (TIM_X) and corresponding external ADC trigger (ADC1_TRGO):

MCU class	TIM_X	ADC1_TRGO
F0 [3, Table 43]	TIM3	0x3
	TIM15	0x4
L0 [4, Table 58]	TIM22	0x4

Table 4.6: External ADC trigger mappings for each TIM peripheral [3]

4.7.3 Results

The LCD text display should display the words “7: 2 Potentiometers” on the 2nd line.

- Once per TIM22 update interrupt, the ADC peripheral samples the analog value of a channel (ADC_INz), and a DMA channel transfers the value to a 100 * 2 array.
- After every ADC channel sample, the ADC reads from the next available channel (2 channels).
- After 100 samples from each ADC channel, the average for each is displayed on the text display.

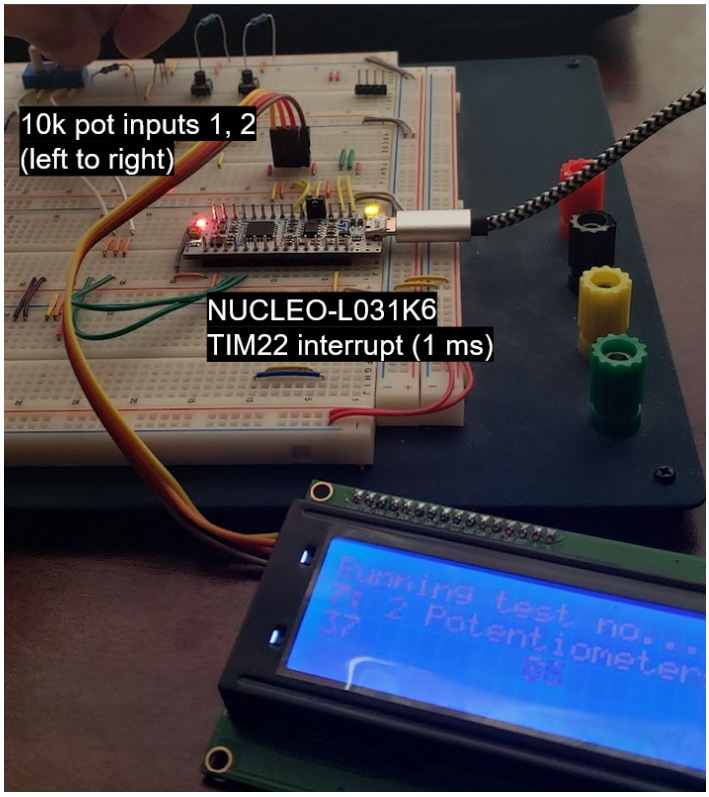


Figure 4.14: Breadboard setup of Test 7

Bibliography

[1] STMicroelectronics. *STM32F051x4 STM32F051x6 STM32F051x8*, rev. 6 (2015). Accessed: February 23, 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f051r8.pdf

[2] STMicroelectronics. *STM32L031x4 STM32L031x6*, rev. 6 (2018). Accessed: February 23, 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l031k6.pdf

[3] STMicroelectronics. *RM0091 Reference Manual*, rev. 9 (2017). Accessed: February 23, 2022. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[4] STMicroelectronics. *RM0037 Reference Manual*, rev. 9 (2021). Accessed: February 23, 2022. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0377-ultralowpower-stm32l0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[5] johnrickman. "LiquidCrystal_I2C." GitHub. https://github.com/johnrickman/LiquidCrystal_I2C (accessed February 23, 2022).