Refrigerator Temperature Monitor & Logger

The refrigerator temperature monitor and logger provides a means for monitoring how well temperature is regulated within a refrigerator. This is important for older refrigerators which may not regulate temperature very well or have leaks. Proper temperature control is especially important for storing expensive medication with strict temperature requirements. By monitoring the temperature carefully, one can determine the best places within the refrigerator for storage.

The temperature sensors for the logger will be daisy-chainable via I2C which can be placed throughout the refrigerator to monitor different regions. The temperatures will be displayed on an LCD and logged to a USB drive which can then be removed and processed on a PC.

Github repository: https://github.com/kqr2/TempMon
Video demos:

- Board demo
- Console demo

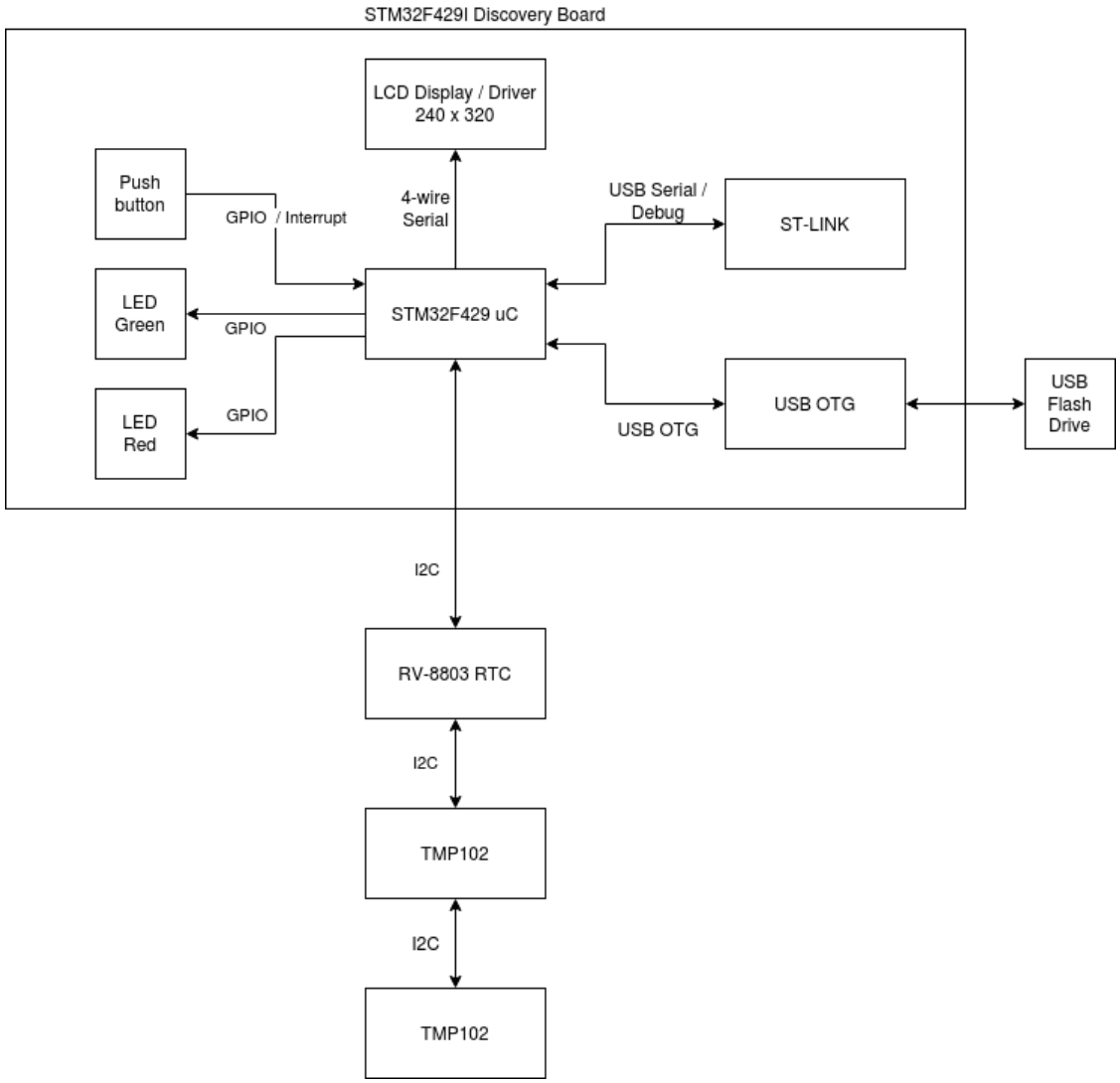
Hardware description

The design is based around a ST Micro 32F429IDISCOVERY board, the Sparkfun TMP102 I2C temperature sensor module, and the Sparkfun RV-8803 Real Time Clock (RTC) module. It also uses an external USB flash drive for storing the actual log data.

Hardware Block Diagram

The STM32F429I Discovery board provides the core hardware needed for this project other than the temperature sensors, RTC, cables and USB flash drive. It conveniently integrates an LCD and headers for the I2C interface. It also supports a USB serial port via ST-LINK which can be used for a command console as well as a pushbutton and two LEDs (green / red) for the user interface. Although the STM32F429 has a built-in RTC it requires an external crystal which was not easy to add to the board due to its tight location near the LCD. Consequently the RV-8803 RTC was used.

The main external peripheral bus is I2C. All the modules use the Sparkfun QWIIC connect system which makes it easy to connect the different modules.



The [TMP102 module] specification :

- I2C interface
- 12-bit, 0.0625°C resolution
- Typical temperature accuracy of ±0.5°CSupports up to four TMP102 sensors on the I^2C bus at a time with addresses 0x48, 0x49, 0x4A, or 0x4B

The [RV-8803] module specification

- I2C interface
- High Time Accuracy

    - ±1.5 ppm 0 to +50°C
    - ±3.0 ppm -40 to +85°C
- 240nA @ 3.3v Low-Power Consumption
- Battery backup
- I^2C Address: 0x32

For the USB serial port, the ST-LINK/V2-B on STM32F429I-DISC1 supports a Virtual COM port which is connected to the STM32 USART1 (PA9, PA10).

The USB OTG interface is used for interfacing to an external USB Flash Drive. To interface to a standard USB flash drive, you will need a USB 2.0 Micro USB Male to USB Female OTG Adapter.

The integrated QVGA LCD display (240 x 320) / driver (FRD240C48 / ILI9341) is configured for 4-wire serial interface.

Finally the Pushbutton and Green / Red LEDs are controlled via simple GPIOs.

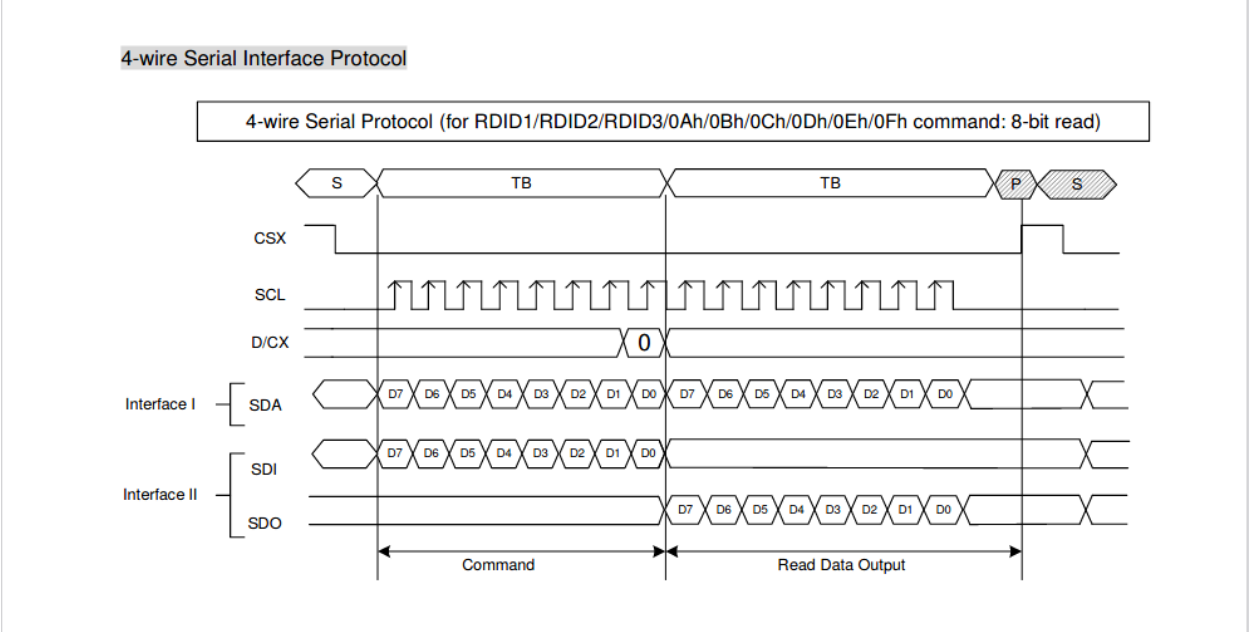
Hardware Software Interface

This information is useful for referring to the schematic, configuring the hardware via STM32CubeMX or while looking into the driver HAL or BSP code.

Interface / Peripheral	**Function**	**Pin**
UART1 / Serial1 / USB Serial Port via ST-LINK	TX	PA9
	RX	PA10
USB OTG	DM (data minus)	PB14
	DP (data plus)	PB15
	ID	PB12
	VBUS_FS	PB13
	FS_PSO (power switch on)	PC4
	FS_OC (overcurrent)	PC5
I2C3	I2C SCL (clock)	PA8
	I2C SCA (data)	PC9
Pushbutton / B1	GPIO	PA0

LED Green / LD3	GPIO	PG13
LED Red / LD4	GPIO	PG14

The integrated QVGA LCD display (240 x 320) / driver (FRD240C48 / ILI9341) is configured for 4-wire serial interface using mode IM[0..3] = [0110]
: 4-wire 8-bit serial Interface I using pins SCL,SDA,D/CX,CSX.



I2C Device	I2C Address (7-bit)
TMP102 Temp Sensor	0x48
	0x49
	0x4A
	0x4B
RV-8803 RTC	0x32

STM32F429 Board pinout

From https://os.mbed.com/platforms/ST-Discovery-F429ZI/

Labels usable in code

PX_Y — MCU pin without conflict	XXX — Arduino connector names (A0, D1, ...)
PX_Y — MCU pin connected to other components See **PeripheralPins.c** (link below) for more information	XXX — LEDs and Buttons (LED_1, USER_BUTTON, ...)

Labels not usable in code (for information only)

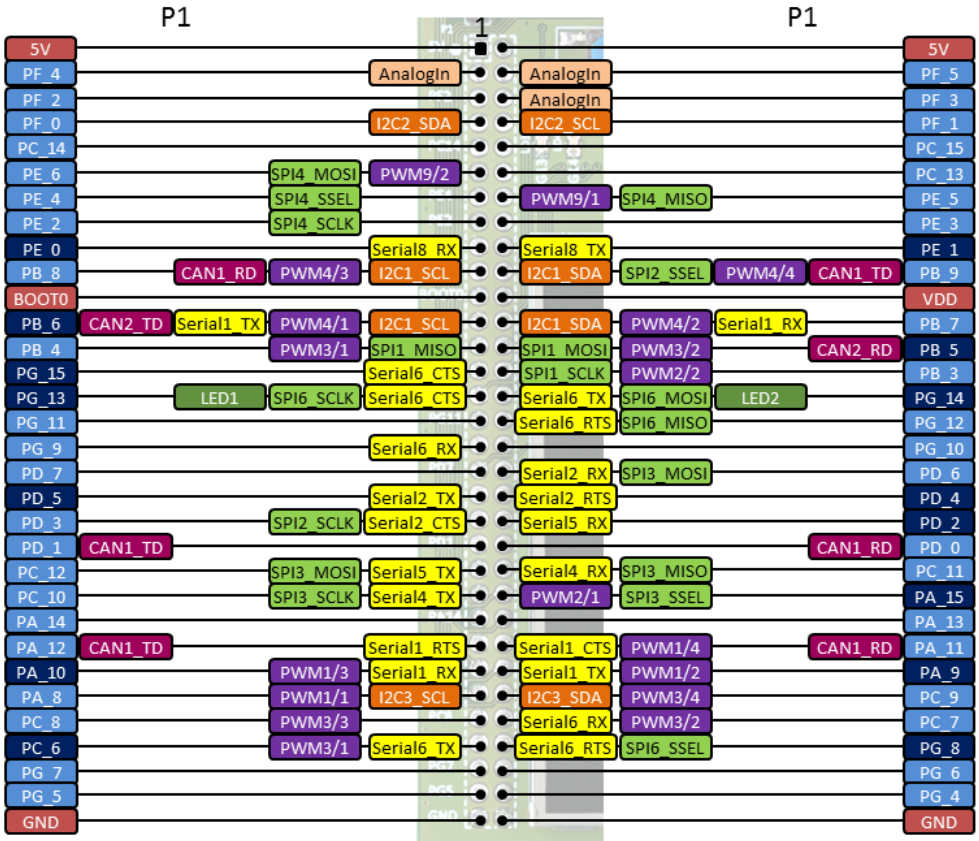
XXX — Serial pins (USART/UART)	XXX — AnalogIn (ADC) and AnalogOut pins (DAC)
XXX — SPI pins	XXX — CAN pins
XXX — I2C pins	
XXX — PWMOut pins (TIMER n/c[N]) n = Timer number c = Channel N = Inverted channel	XXX — Power and control pins (3V3, GND, RESET, ...)

life.augmented

DISCO-F429ZI

P1 HEADER

(top left side)

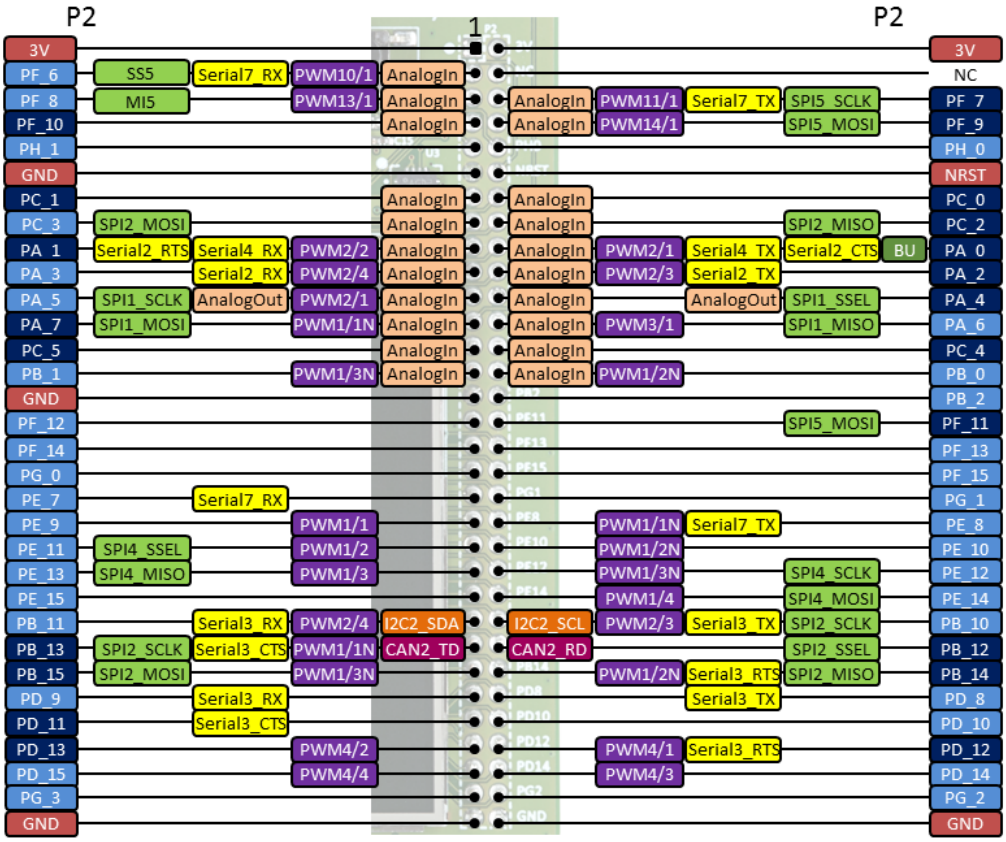


life.augmented

DISCO-F429ZI

P2 HEADER

(top left side)

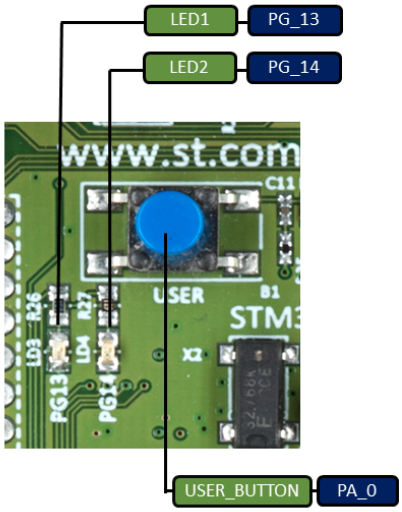


life.augmented

DISCO-F429ZI

P2 HEADER

(top left side)



Software Description

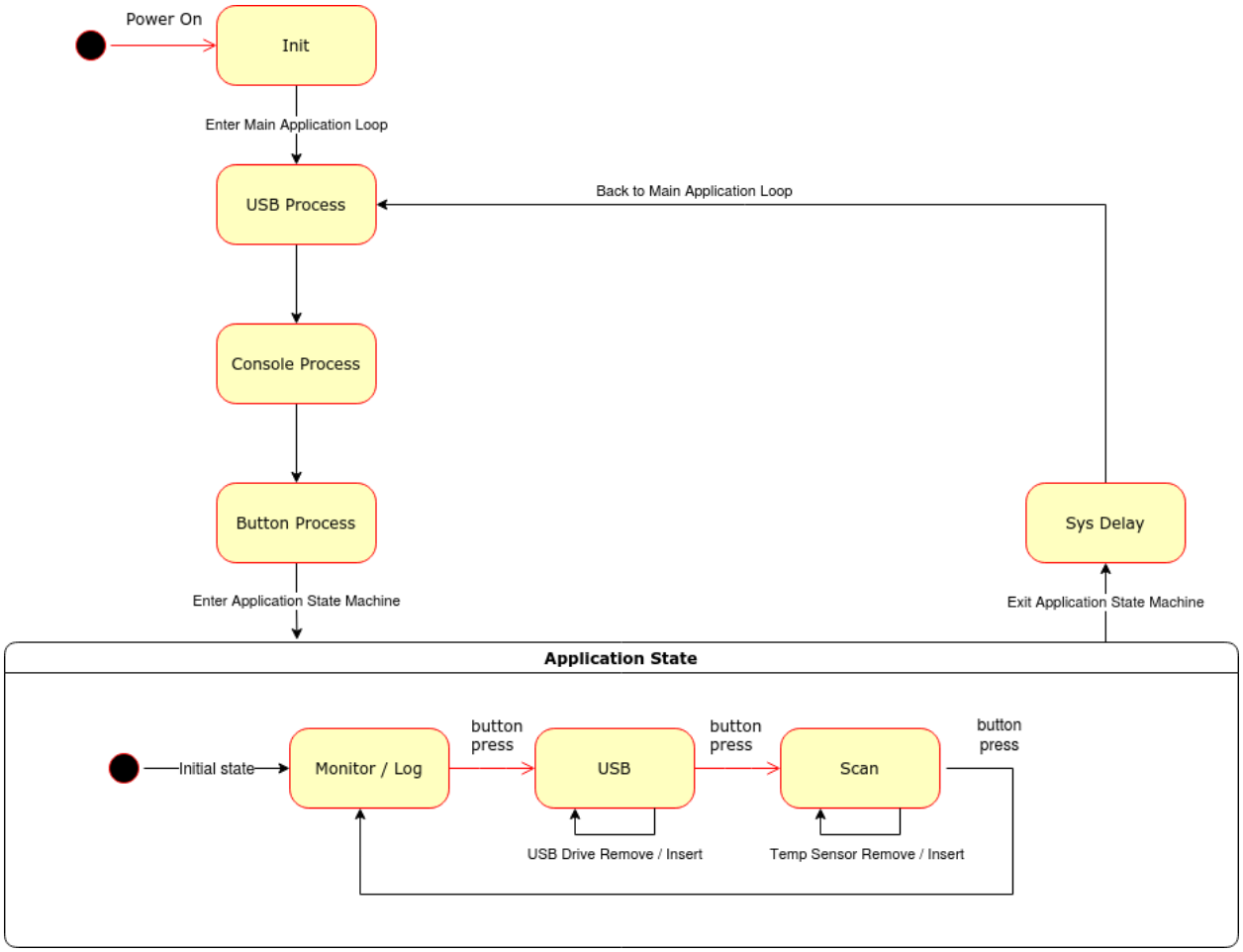
The application consists of a simple bare-metal main application loop with a state machine. The main application loop consists of :

- **USB Process** : process USB events including the insertion and removal of a USB flash drive
- **Console Process** : process console commands via the serial port
- **Button Process** : debounce push-button if a push-button interrupt occurs

Once these events and commands are processed, it enters the main application state machine which consists of three states / modes:

- **Monitor / Log** : update RTC, read the temperature sensors and log the data to flash
- **USB** : allows the user to safely remove / insert a USB flash drive
- **Scan** : allows the user to safely remove / insert new temperature sensors

Transitions are triggered by push-button events. After the state machine is processed, it goes back to the main application loop.



Function	Call reference	Code	Description
Initialization			
HAL init		main.cpp	Initialize the low level hardware abstraction layer drivers
BSP init		main.cpp	Initialize the board support package for peripherals including LCD
SYS init	main.cpp	sys.cpp	Initialize the system which includes I2C sensors (RTC, temp sensors)
Main Application Loop			
USB Process	main.cpp	usb_host.c	Process USB events
Console Process	main.cpp	console.c	Process the console commands
Button Process		main.cpp	Debounce the push-button
- Interrupt handler	stm32f4xx_it.c	main.cpp	Button interrupt handler
Application State Machine			
Next state logic		main.cpp	Based on button press, process next state transition
Monitor / Log mode		main.cpp	Main state where temperature sensors are read and logged to file on the USB flash drive
USB mode		main.cpp	Mode to safely remove / insert a new USB flash drive
Scan mode		main.cpp	Mode to safely remove / insert temperature sensors

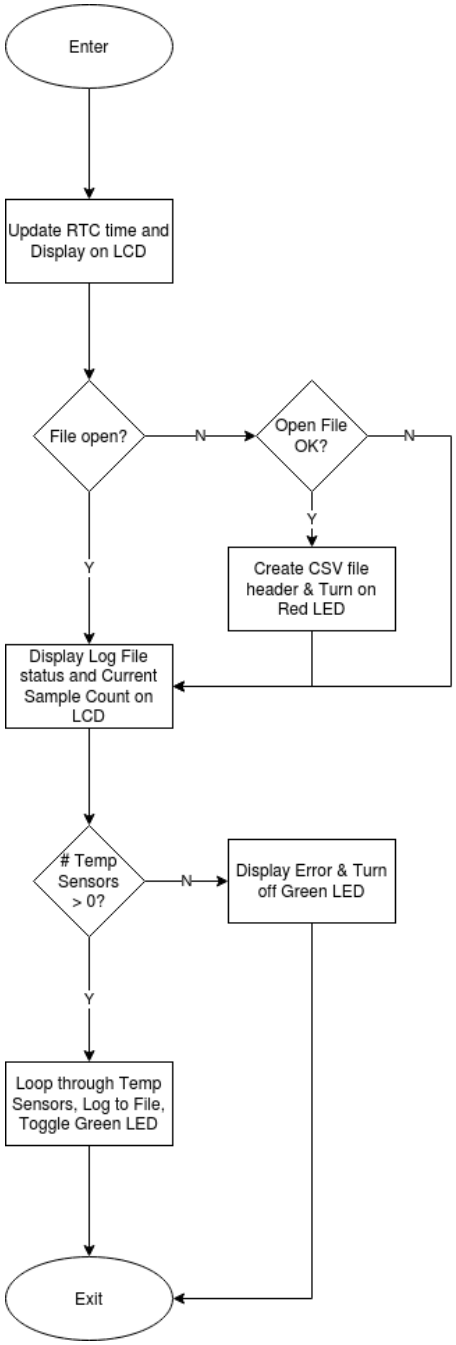
The USB and Console Processing functions are calls to third party libraries which have been configured for this board. The USB Processing code was generated by STM32CubeMX and the Console code was from https://wokwi.com/projects/324879108372693587 . Details on porting the Console code to this project can be found in https://github.com/kqr2/CommandConsoleDemo .

Button processing is triggered by the push-button interrupt. Once triggered, it will start debouncing. The button must be stable for at least 10 ms for it to be considered stable and pressed. This button press will then cause a transition in the application state machine.

The main files which were modified or added by me for this project are:

File	Description
main.cpp	Although the template is auto-generated by STM32CubeMX, I added the main application loop and application state machine to this file. I also added BSP calls in particular for controlling the LCD.
sys.cpp	This contains code to manage the I2C peripherals including RTC and temp sensors. • sys_init() : Initialize BSP I2C and initial scan for temp sensors • sys_tmp_rescan() : Rescan the temperature sensors • sys_set_timer_interval() : Change the monitor / logging interval
tmp102.c	This is a simple TMP102 driver. Although there are arduino libraries, I only needed a few functions and wrote them myself. Functions include: • tmp102_init() for initialization • tmp102_detect() for sensor detection • tmp102_read_temp() for fixed point temperature reads
consoleCommands.cpp	Several commands were added to the Console including: • **rtc_time** : get the current RTC date and time • **rtc_set** : set the current RTC time using unix time stamp • **sys_set_interval** : sets the effective monitor / logging loop interval in milliseconds (default 1000 msec)
stm32f4xx_it.c	Modified to call button interrupt handler
stm32f429i_discovery.c stm32f429i_discovery.h	Modified to export I2C functions. Added: • I2Cx_Detect() : Detect I2C device with specified address Broke up Transmit / Receive functions to make RV-8803 and TMP102 drivers easier to port and implement: • I2Cx_Master_Transmit(): Transmit only on master • I2Cx_Master_Receive() : Receive only on master
fatfs.c fatfs.h	Add code for better file handling and writing to files: • FatFS_open() : open file • FatFS_opened() : check if file is opened • FatFS_close() : close file • FatFS_write() : Write buffer • FatFS_writeln() : Write buffer with newline

Monitor / Log Mode



This is the main mode where the temperature is monitored and logged. If a USB flash drive is mounted, it will attempt to create a file with the following name:
TM_<YYYY><MM><DD>_<HH><mm>.TXT where:

- YYYY = Year
- MM = Month
- DD = Day
- HH = UTC Hour
- mm = UTC minutes

Data will be logged in a simple CSV format where the first line will be a header of the form:

- Date,Time,Sample,Temp_<address1>,Temp_<address2>, …

Likewise data will subsequently logged as:

- <UTC Current Data>,<UTC Current Time>,<Sample #>,<Temp address1 in deg C>, …

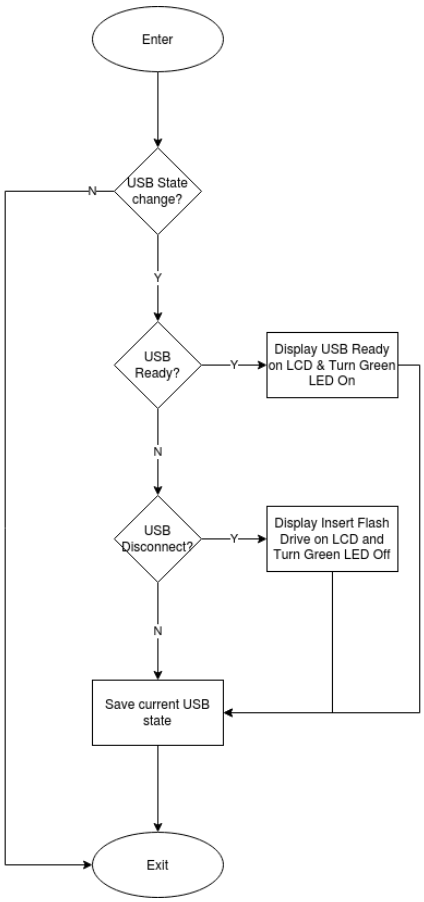
Example:

```
Filename: TM_20220612_0457.TXT

Date,Time,Sample,Temp_72,Temp_75
06/12/2022,04:57:52PM,12,27.0625,27.4375
06/12/2022,04:57:54PM,13,27.0625,27.4375
06/12/2022,04:57:55PM,14,27.0625,27.4375
06/12/2022,04:57:57PM,15,27.0625,27.3750
06/12/2022,04:57:58PM,16,27.0625,27.3750
06/12/2022,04:58:00PM,17,27.0625,27.3750
06/12/2022,04:58:01PM,18,27.0625,27.3750
.
.
.
```

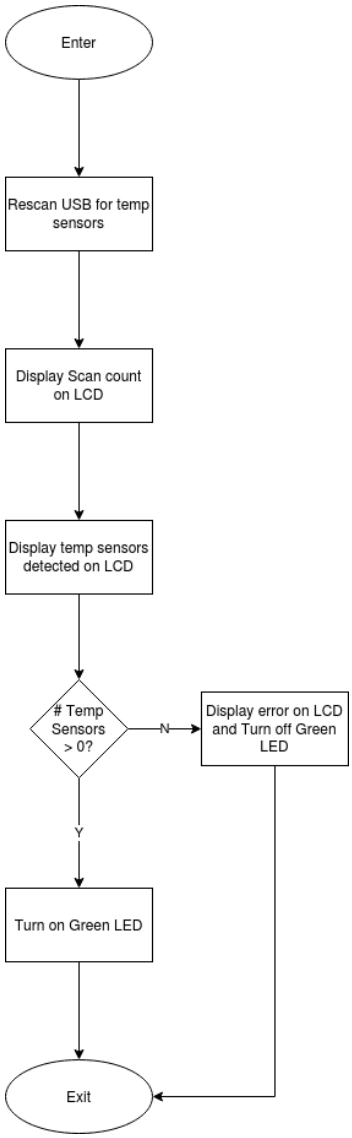
USB Mode

In this mode the user can safely eject the USB flash drive. The application will also check for any USB state changes and indicate that to the user.



Scan Mode

In this mode, the application scans for new temperature sensors and gives the user feedback on what has been detected. If nothing is found, then an error will be displayed.



Open Source and Third Party Libraries

This project would not be possible without the following libraries. The STM32 code can be found at https://github.com/STMicroelectronics/STM32CubeF4 and has a list of licenses that it uses too.

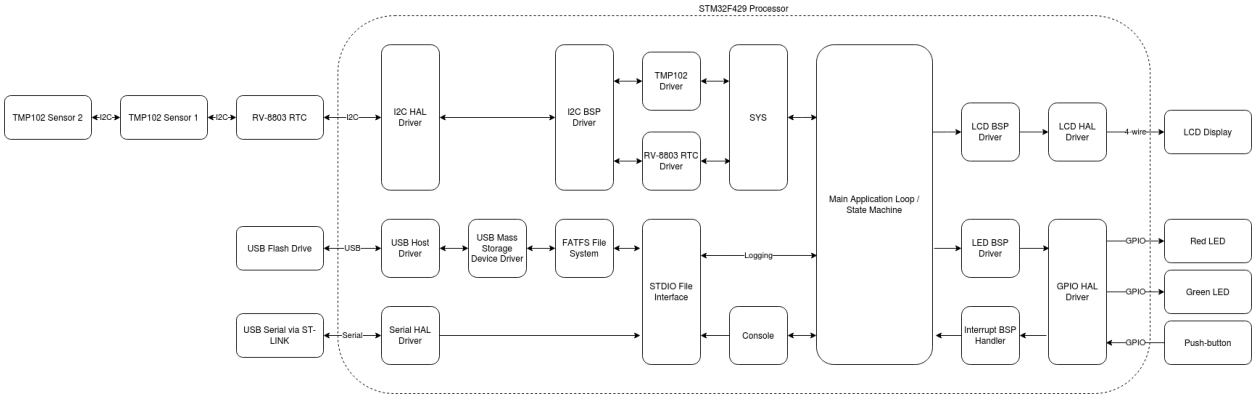
Software Library	**License**
STM32CubeMX Generated Code	BSD-3-Clause
STM32 HAL	
STM32F429I-Discovery BSP	

STM32 Middleware FATFS	
STM32 Middleware USB_HOST	SLA0044
CMSIS	Apache 2.0
Command Console Code	Unlicense
Sparkfun RV-8803	MIT License

This documentation should be included with all distributions of this project / product and citing of licenses serves as attribution of licenses for binary distribution purposes.

Software Block Diagram

This diagram shows how the software blocks interface with the hardware.

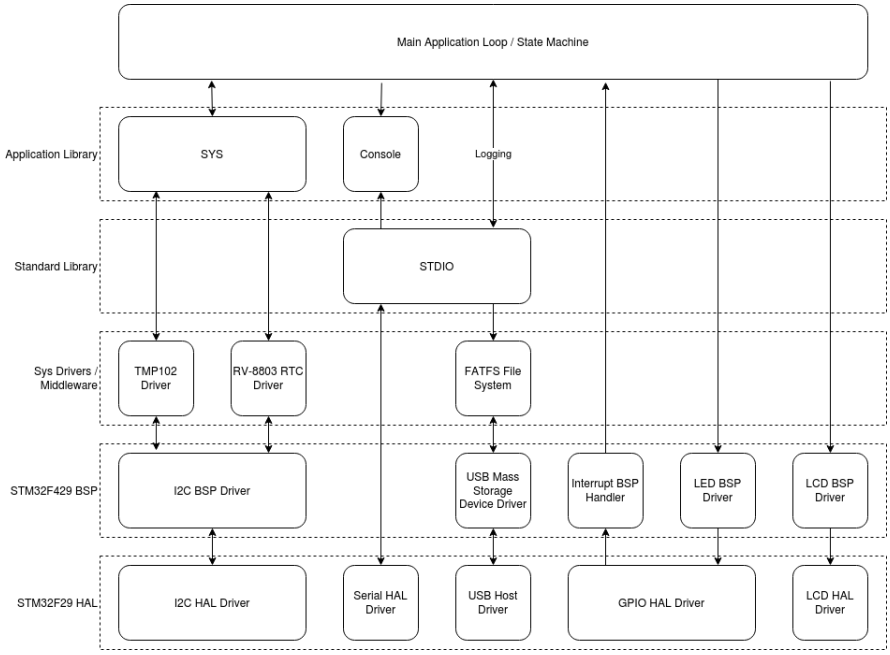


Hierarchy of Control Diagram / Layered Diagram

I combined these two types of diagrams into one. From top down the layers are essentially:

- Main Application Loop / State Machine
- Application Libraries
- Standard Libraries
- System Drivers / Middleware
- BSP Drivers
- HAL Drivers
-

The higher layer controls the next bottom layer.



Build instructions

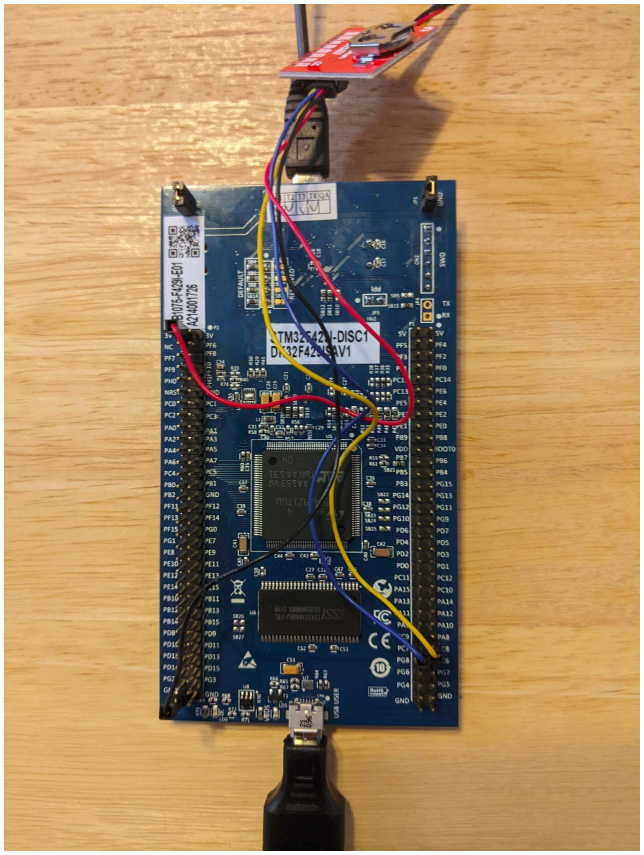
Hardware

Bill of Materials

Part Number	Description	Quantity
STM32F429I-DISC1	STM32F429 Discovery Board	1
KIT-15081	SparkFun Qwiic Cable Kit	1
SEN-16304	SparkFun Digital Temp Sensor TMP102 (Qwiic)	1-4
BOB-16281	SparkFun Real Time Clock Module RV-8803 (Qwiic)	1
BC62766 *	USB 2.0 Micro USB Male to Female OTG Adapter	1
P-FD128ATT03-GE *	PNY USB 2.0 128GB Flash Drive	1

* *Note that you can easily substitute flash drive and USB OTG adapter, however, these were the parts that were used for this project.*

I2C Connections



The QWIIC connect system makes it easy to add the RTC and temp sensor modules. Connect a QWIIC cable with female jumpers to the bottom of the STM32F429 discovery board using the P2 (left) and P1 (right) connectors.

All QWIIC cables have the following color scheme and arrangement:

- **Black = GND**
- **Red = 3.3V**
- **Blue = SDA**
- **Yellow = SCL**

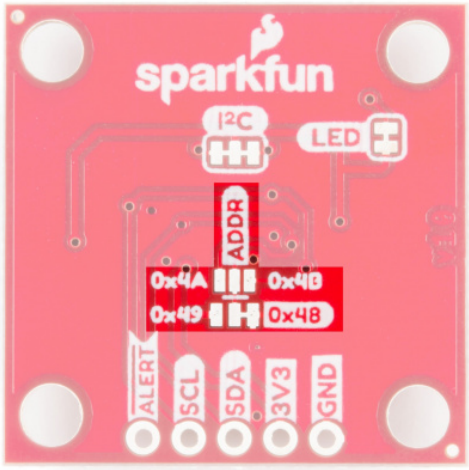
QWIIC Color	STM32F429 Connector / Pin	Description
Black	P2 / GND	Power GND
Red	P2 / 3V	Power 3V
Blue	P1 / PC9	I2C SDA (Data)
Yellow	P1 / PA8	I2C SCL (Clock)

The default address for the TMP102 temperature module is 0x48. If you wish to use more than one on a single I2C bus, you will need to modify the jumpers.

ADDR Jumpers

The default I²C address of the board is **0x48**. To change the address, cut the jumper connecting the two pads closest to the 0x48 label. Soldering one of the center pads to one of the outer most pads will change the boards address to the matching label. The TMP102's address is determined by connecting the address pin directly to one of the following:

ADDR	Address
GND	0x48
3V3	0x49
SDA	0x4A
SCL	0x4B



Although the i2c modules can be chained in any order, it's recommended that you put the RTC module first as it should not be disconnected during normal operation.

For this project demonstration, the I2C sensors were connected as follows:

I2C Chain Order	Part	I2C Address (7-bit)
1	RV-8803 RTC	0x32
2	TMP102	0x48
3	TMP102	0x4B

Software

Build environment and tools used :

- Lenovo ThinkPad
- Ubuntu 20.04
- STM32CubeIDE Version 1.9.0
- STM32CubeMX Version 6.5.0

Github repository: https://github.com/kqr2/TempMon

Although this project was developed and built using a Linux system, it should work on any system where STM32CubeIDE / MX and git is supported.

Build instructions

1. Clone the github project which contains the build and source files
    a. git clone https://github.com/kqr2/TempMon.git
2. Open the project in STM32CubeIDE
3. Build project
4. Attach USB cable to ST-LINK connector on STM32F429 discovery board
5. Run on target

STM32CubeMX Hardware Configuration

If you need to examine or change the hardware configuration you can open the TempMon.ioc file in STM32CubeMX. If you need to rebuild the configuration from scratch for some reason, you can use the following steps in STM32CubeMX:

- Create new project based on STM32F429i Discovery Board
    - This will configure most of the peripherals correctly for the board
- Add Middleware: See this video for helpful hints
    - USB Host
    - FATFS
- Remove Middleware:
    - FreeRTOS : RTOS is not needed as this is a bare-metal application
- In Project Manager Tab
    - Select STM32CubeIDE as Toolchain / IDE
- Generate Code

Download the STM32F429i BSP package and add to the project.

The BSP (board support package) is a higher level driver interface than the HAL (hardware abstraction layer). In general if you use the BSP for initialization you do not have to use the corresponding lower level HAL. For example in this case the BSP contains the higher level LCD configuration that works with the STM32F429 discovery board. You can comment out the lower level HAL calls for the LCD and just use the BSP which is more appropriate. If you choose to

not use the BSP and want to configure the LCD from STM32CubeMX from scratch, please see this video.

Debugging and testing the system

The project was debugged using STM32CubeIDE + ST-LINK debugger. Putty was used for serial console to the board. On a linux system, the Virtual COM / serial port will show up as /dev/ttyACM0 and connection speed is 115200 N81.

Debugging I2C will typically require a logic analyzer to view the bus signals, however, for this project it was not necessary. For each I2C device I used a register with default values (such as ID) to confirm that the I2C transactions were happening correctly. The most common mistake with I2C is usually the address. I2C addresses are usually given as 7-bits, however, the driver requires it as 8-bit. So a 0x48 device will need to use 0x48 <<1 in the driver call.

The QWICC connect system makes it easy to add / remove the I2C modules to test I2C scanning function and error cases. The USB flash drive uses a standard FATFS so it can be read from both the board as well as a laptop / PC which makes verifying the log data easy.

Power

For this project, I mostly powered the board directly from my laptop for ease of development, however, any USB battery pack can be used. Based on measurements using this USB power meter, the board consumes up to 240ma (with USB drive attached) and 200ma (without USB drive). Using a 30,800mAh battery pack such as https://www.amazon.com/Wireless-Portable-Charging-External-Compatible/dp/B0915T91JN/ should power the device for a little more than 5 days (30,800mAh / 240mA / 24 hours per day).

Future

In order to ready this project for production, a custom PCB to reduce size, cost as well as power consumption will be needed. Only required peripherals will be included. The LCD can also be smaller to further reduce cost and power consumption. I would also want to profile the amount of RAM, flash, and processing power used so I could potentially use a cheaper microcontroller.

A good mechanical enclosure will also be needed so that it can be more rugged and used in a refrigerator. The enclosure should support:

- QWICC connectors to temperature modules
- USB 2.0 connector for USB flash drive so no OTG adapter required
- Micro USB connector to allow connection to various USB batteries as well as debug

Each temperature module will also need its own rugged enclosure which supports daisy-chaining of the modules.

I would design the custom PCB most likely using KiCad and find a manufacturer that would be willing to build prototype PCBs and populate parts. For mechanical design, it's simple enough

to use FreeCAD and 3D print prototypes. Once the prototypes have been thoroughly tested for robustness I could look at manufacturers who could better help with production.

The project would also need to undergo any product certifications required such as EMC testing.

This project can be enhanced in many ways:

- Firmware upgrade – this can be done via the USB flash drive. If the firmware detects a valid upgrade, it can upgrade itself. This would allow future feature enhancements.
- Add light sensor : to conserve power, the LCD will automatically only turn-on when the refrigerator light turns on.
- Add built-in graphing capability to LCD to show temperature trends so the user does not have to plot the data themselves
- Improve the user interface to make it easier and more appealing
- Add support for time zones as only UTC is currently supported
- Use STM32 built-in RTC instead of RV-8803
- Include a battery power monitor
- Support other i2c temperature sensors that have more addresses. Although the TMP102 is easy to use, it only supports 4 per bus
- Add another I2C bus to make chaining of sensors even easier. By removing unused peripherals from the discovery board, this should free up additional I2C buses.

Self-assessment

Criteria	**Self-score**	**Comments**

Project meets minimum project goals	2.25	I gave myself 2.25 since I used 4 peripherals and the application allows users to dynamically add or remove sensors to the system which is interesting.
		● STM32F429 Cortex-M processor
		● Button causes interrupt and is debounced
		● **4 peripherals**
		  ○ LCD
		  ○ USB OTG
		  ○ RV-8803 RTC via I2C
		  ○ TMP102 Temp sensor via I2C
		● Serial port with command console
		● Application performs monitor & logging as intended. **Users can add or remove sensors dynamically.**
		● State machine with 3 distinct user controllable modes

Completeness of deliverables	2.75	I believe code is fairly readable even without the report. Also this report addresses each point pretty thoroughly and can be used as a developer reference. Video shows project features and overview of the hardware.
Clear intentions and working code	2.5	System performs as intended but the code could be more professionally polished.
Reusing code	2.5	Licensing of code identified for both 3rd party and project code. Confident that project can be rebuilt
Originality and scope of goals	2	Dynamic I2C scanning of sensors is interesting but was not awesome
Self-assessment	2.75	This seems recursive, but I feel I have done a fair assessment of my project.
Bonus: power analysis, firmware update, or system profiling	1	Measured power with meter
Bonus: version control was used	3	Almost 40 commits in a month. Project starts from the initial STM32CubeMX generation.