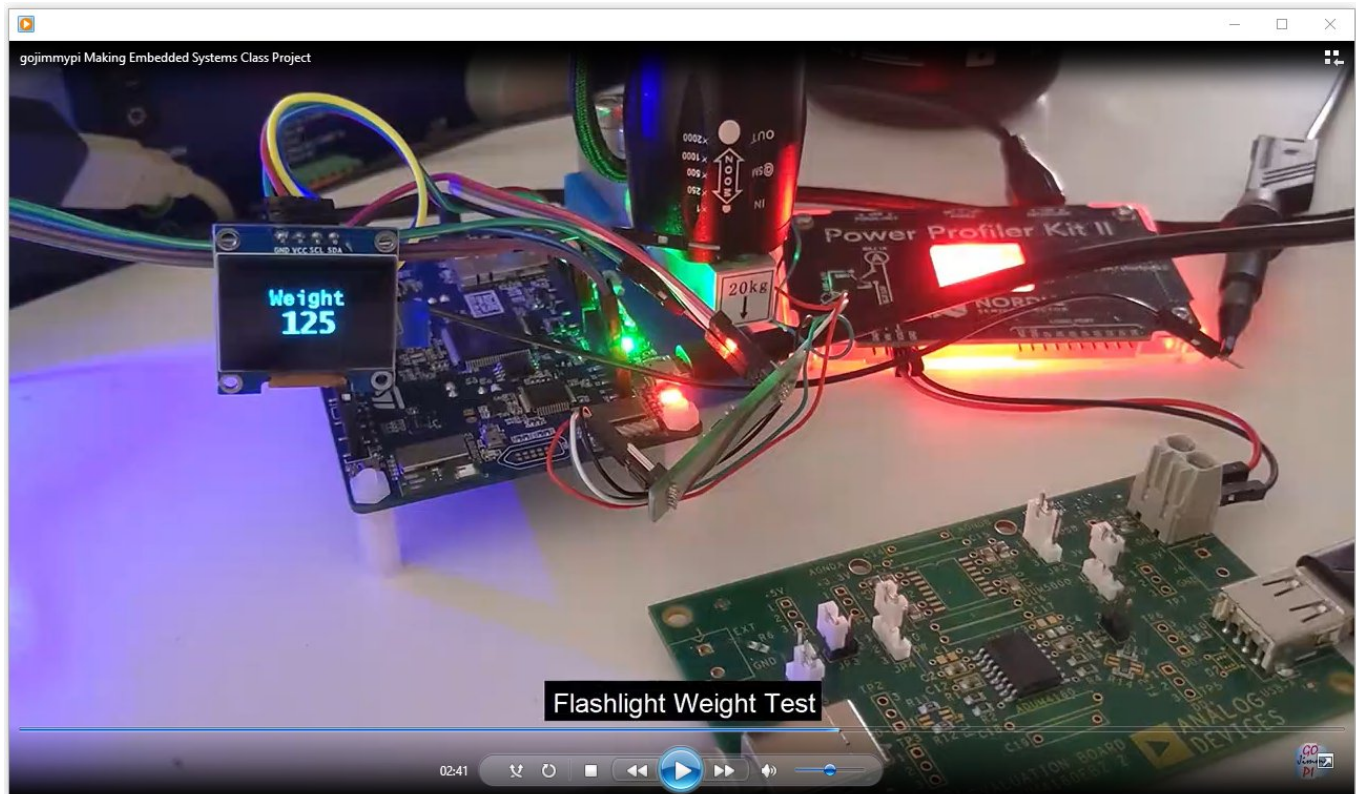


# Final Project Report

---

This Final Project Report is a supplement to the [Final Project Minimum Project Requirement Detail](#). An embedded microcontroller was used to create a propane tank weight measurement system. [Source code](#) is available on GitHub.

See also the [YouTube Video](#):



## Application Description

This application is an embedded controller to monitor grill propane tank weight to replace the existing mechanical spring scale:



## Hardware Description

This project uses the [STM32L475VG](#), part of the [STM32 Ultra Low Power](#) Arm Cortex-M4 32-bit MCU+FPU series found on the [B-L475E-IOT01A Discovery Board](#).

## Software description

The application is written in C/C++ and targets the STM32 ARM Platform.

### Describe the code in general

The code is a multi-threaded FreeRTOS embedded application. There are 5 threads names LED1, LED2, UART1, DISPLAY, and PWM.

### LED1 Thread

The [LED1 Thread](#) is a simple state machine and controls LED1 (blinky, on, or off). This thread also sends pressure reading and scale weight text to the UART.

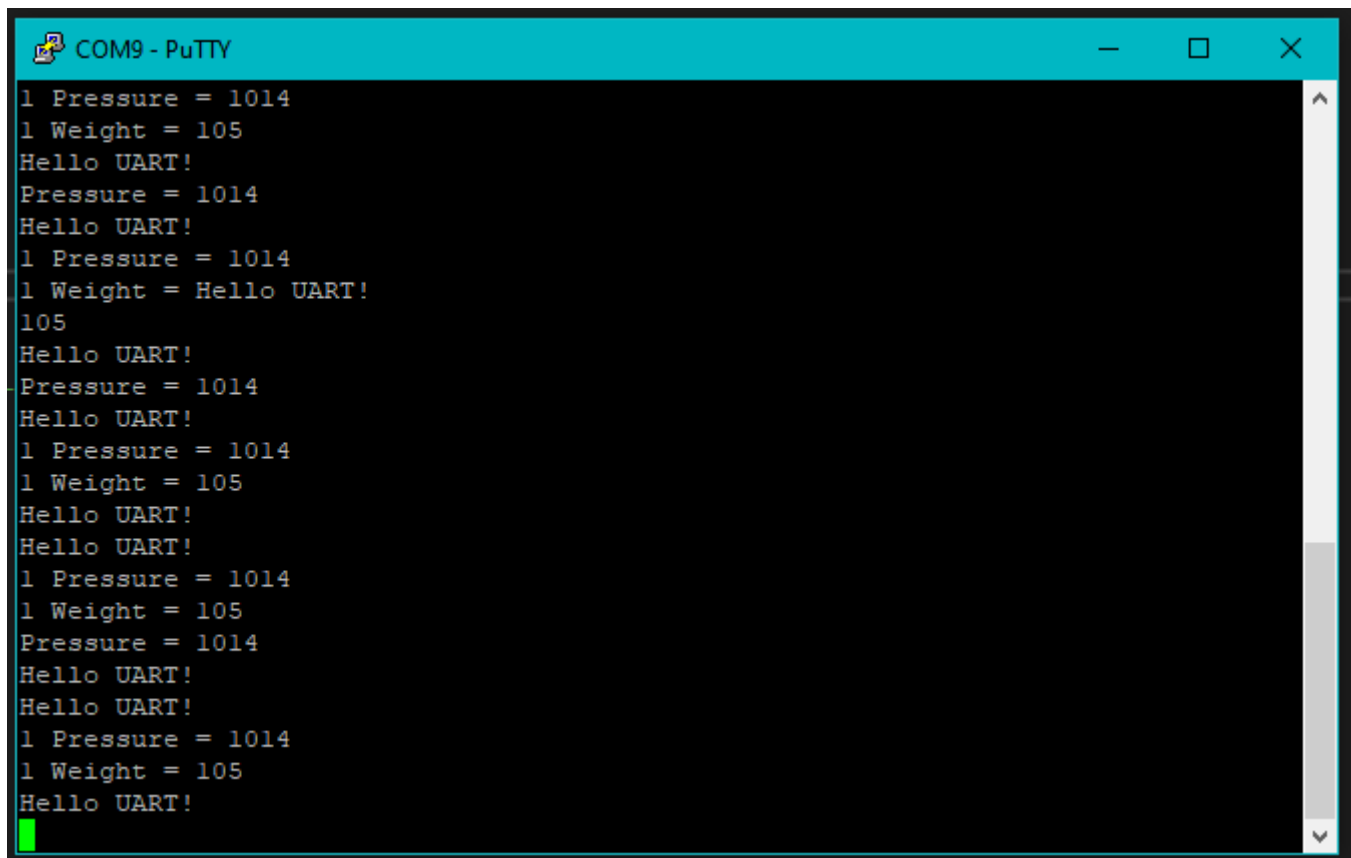
## LED2 Thread

The [LED1 Thread](#) monitors the state machine and blinks LED2 depending on the current state:

- [IsBlinking](#) - one blink
- [AlwaysOn](#) - two blinks
- [AlwaysOff](#) - three blinks

## UART1 Thread

Just to test RTOS concurrency (and hard fault avoidance!) the [UART1 Thread](#) sends a "Hello UART1" message to the UART about every 1.35 seconds. An odd number was chosen to ensure pseudo-random conflicts in timing with other messages.



```
COM9 - PuTTY
1 Pressure = 1014
1 Weight = 105
Hello UART!
Pressure = 1014
Hello UART!
1 Pressure = 1014
1 Weight = Hello UART!
105
Hello UART!
Pressure = 1014
Hello UART!
1 Pressure = 1014
1 Weight = 105
Hello UART!
Hello UART!
1 Pressure = 1014
1 Weight = 105
Pressure = 1014
Hello UART!
Hello UART!
1 Pressure = 1014
1 Weight = 105
Hello UART!
```

This particular screen snip (above) is interesting. Note the seemingly stray "105" value. Apparently this is where the thread tried to grab a mutex to use the UART, but it was not available on a timely basis, so it skipped the entire "1 weight" part of the message. Recall this occurs in [two steps](#):

```
UART_TxMessageIntValue(WeightMessage, sizeof(WeightMessage),
CurrentTankWeight);
// (RTOS Thread switch occurred around here)
UART_TxMessage(CrLf, sizeof(CrLf));
```

## DISPLAY Thread

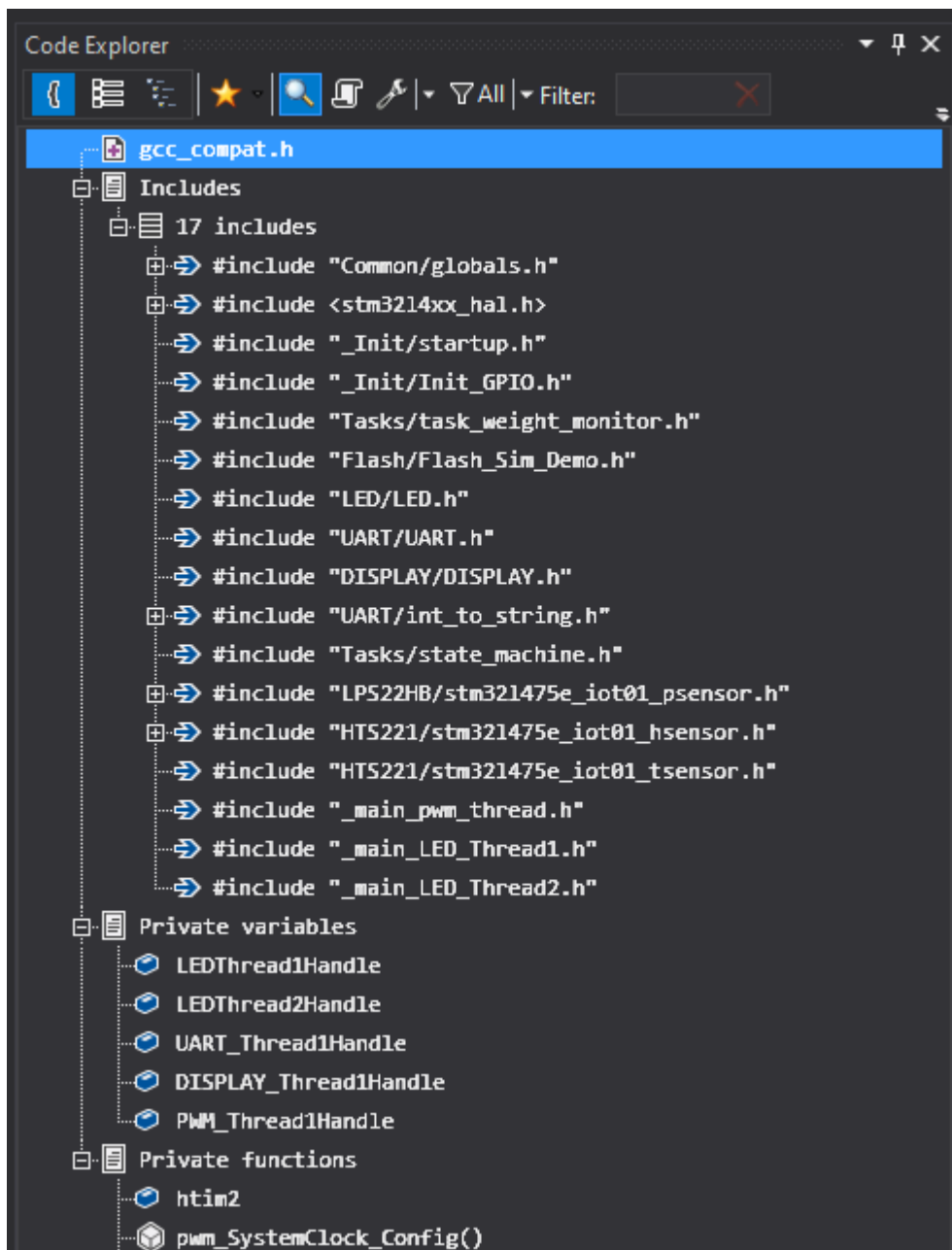
The [Display Thread](#) is where the scale is monitored and the value displayed on the screen. There's also a small state machine monitoring for a button long press to then tare the scale and save the offset value to flash.

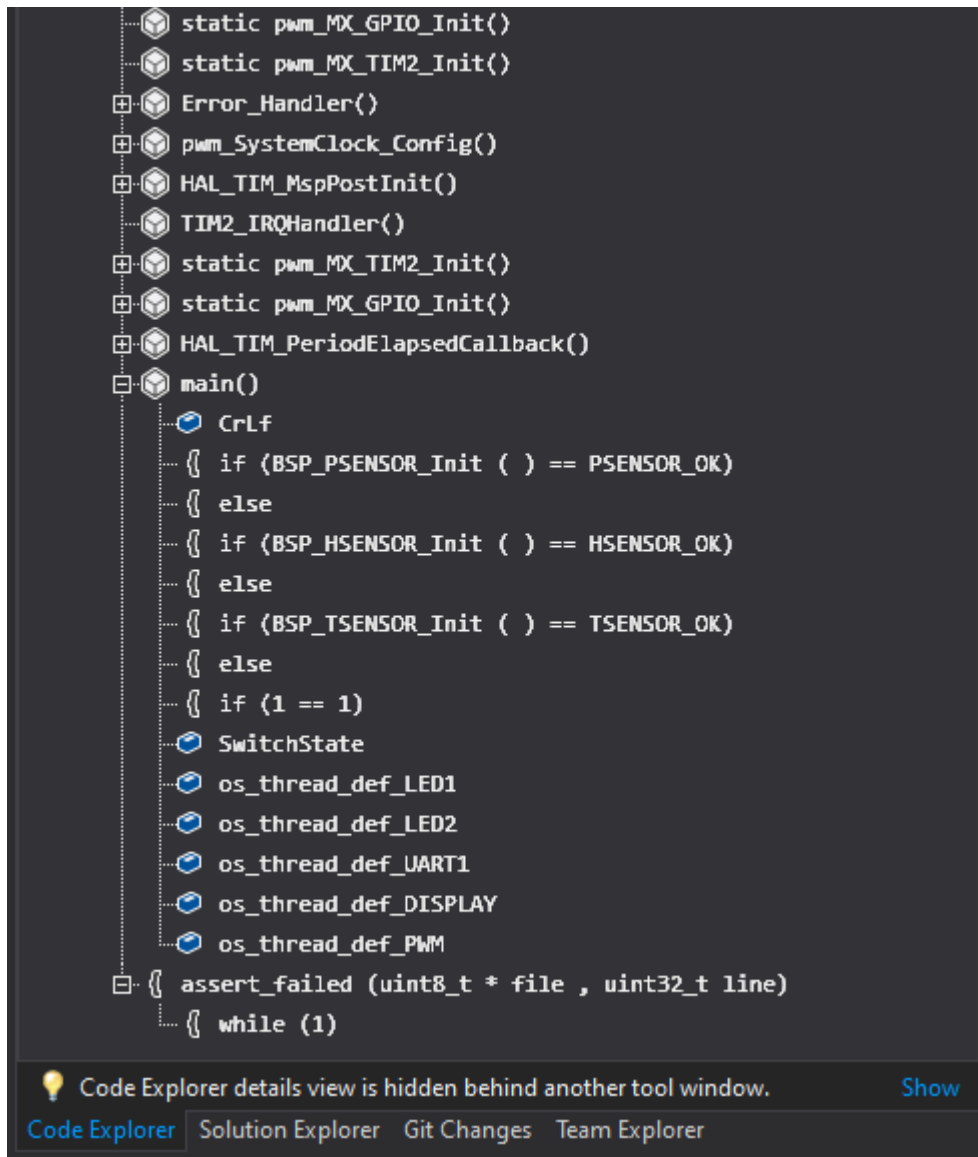
## PWM Thread

The [PWM thread](#) is included only for "interesting" experiments and is not required for basic functionality.

After some initial code experiments, the thread runs in a loop sending barometric pressure reading to the UART every 2 seconds.

Code Explorer View of [main\(\)](#)





```

static pwm_MX_GPIO_Init()
static pwm_MX_TIM2_Init()
Error_Handler()
pwm_SystemClock_Config()
HAL_TIM_MspPostInit()
TIM2_IRQHandler()
static pwm_MX_TIM2_Init()
static pwm_MX_GPIO_Init()
HAL_TIM_PeriodElapsedCallback()
main()
    CrLf
    { if (BSP_PSENSOR_Init ( ) == PSENSOR_OK)
      { else
        { if (BSP_HSENSOR_Init ( ) == HSENSOR_OK)
          { else
            { if (BSP_TSENSOR_Init ( ) == TSENSOR_OK)
              { else
                { if (1 == 1)
                  SwitchState
                  os_thread_def_LED1
                  os_thread_def_LED2
                  os_thread_def_UART1
                  os_thread_def_DISPLAY
                  os_thread_def_PWM
                { assert_failed (uint8_t * file , uint32_t line)
                  { while (1)

```

Code Explorer details view is hidden behind another tool window. [Show](#)

[Code Explorer](#) [Solution Explorer](#) [Git Changes](#) [Team Explorer](#)

Describe the parts you wrote in some detail (maybe 3-5 sentences per module)

The main code repository can be found at [github.com/gojimmypi/loT\\_BBQ/tree/main/loT\\_BBQ\\_STM32](https://github.com/gojimmypi/loT_BBQ/tree/main/loT_BBQ_STM32). MIT License.

- [Display Code](#) was written specifically for this project. MIT License.
- [Flash Config](#) was created from scratch for this project. MIT license.
- [LED Blinky and State Machine](#) created by gojimmypi for this project. MIT license.
- [UART](#) created for this project, MIT license. Uses [stm32l4xx\\_hal.h](#) assumed to still be [BSD 3-Clause license](#)

Describe code you re-used from other sources, including the licenses for those

This project was created using the [Sysprogs VisualGDB Extension](#) for [Visual Studio 2019](#). A tiny amount of initialization code was generated using the [STM32CubeIDE](#).

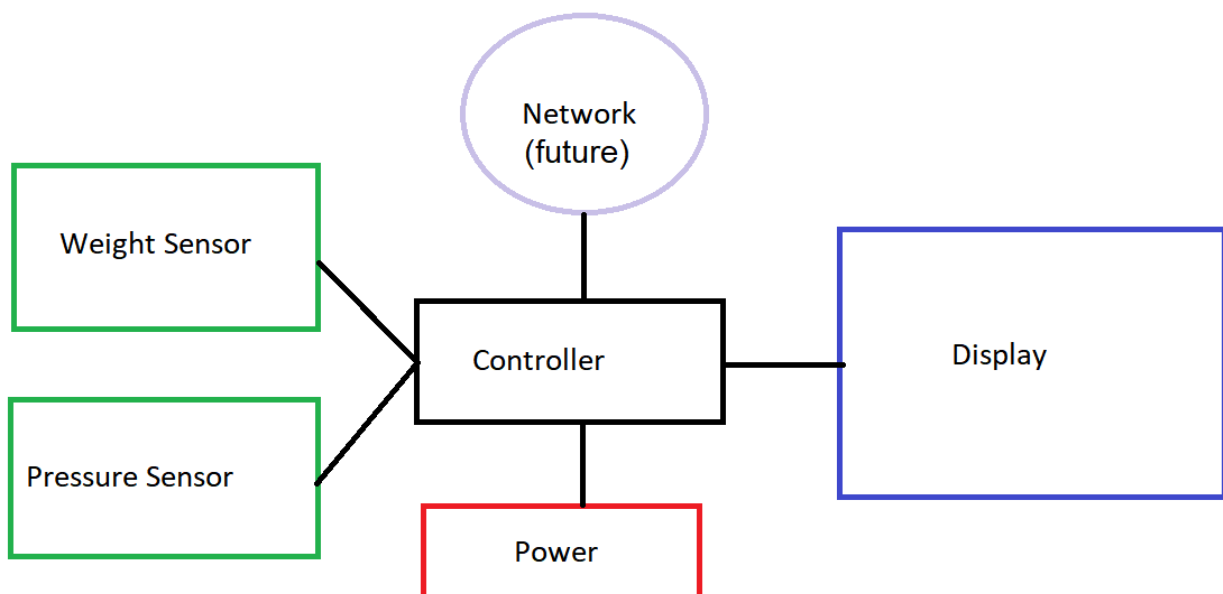
- [HX711 Arduino Library](#) was ported to this STM32 HAL environment with FreeRTOS support. MIT License.

- [STM hsensor HAL](#) BSD 3-Clause license.
- [STM Pressure Sensor HAL](#) BSD 3-Clause license
- [ssd1306 display driver](#) was created by Aleksander Alekseev and [modified](#) for this project. MIT License
- [Flash HAL](#). Assumed to still be Licensed by ST under BSD 3-Clause license, but the [license File is now blank](#).

Additional thanks to Elecia White (the Making Embedded Systems Instructor) for the code review, in particular to help find the cause of those nasty Hard Faults. Also thanks to the class mentors Erin, Daniel, Thomas, and Jeff... as well as all my classmates on the discord channel. A lot of smart and creative people with all sorts of interesting ideas!

## Diagram(s) of the architecture

Show below is the basic diagram of the architecture. The network feature has not yet been implemented:

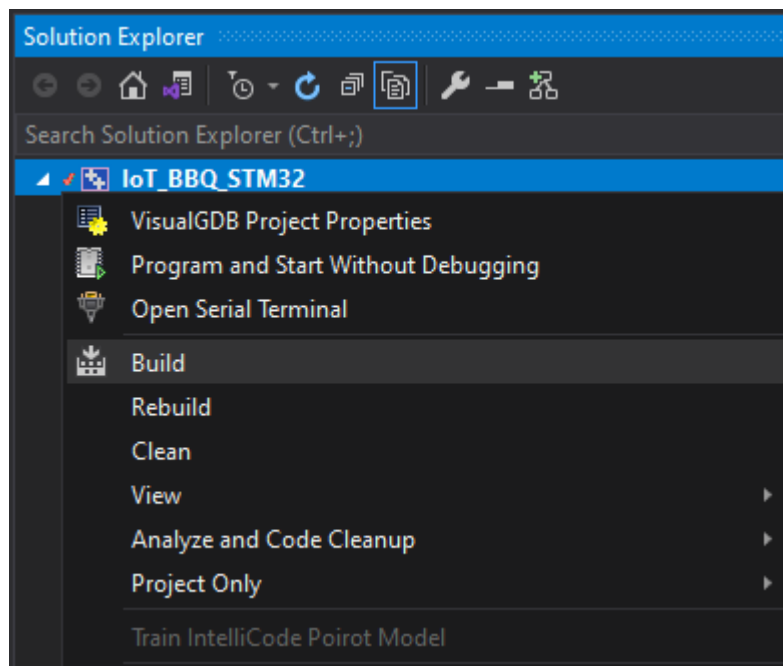


## Build Instructions

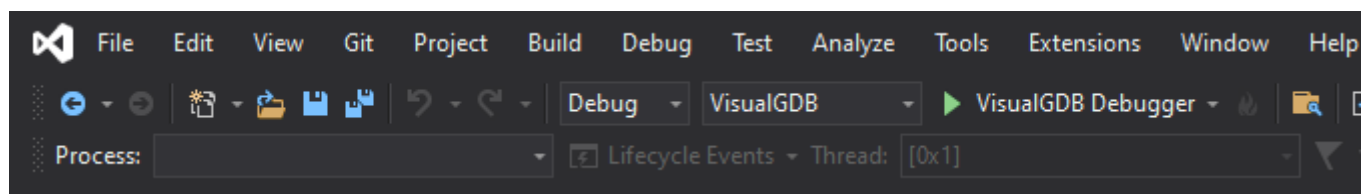
There are two options for building: the GUI from Visual Studio and a command-line batch file, both on Windows.

### Build in Visual Studio

Open the Visual Studio Solution File called [IoT\\_BBQ\\_STM32.sln](#). Find the [IoT\\_BBQ\\_STM32](#) Project. Right-click and select "Build...":



To build and upload code, simply click the green **VisualGDB Debugger** button:



If the build was successful, the code will be sent to the board:



```

C:\Users\gojimmy\pi\AppData\Local\VisualGDB\EmbeddedDebugPackages\com.sysprogs.arm.openocd.st\bin\openocd.exe -c "gdb_port 64567" -c "telnet_port 64565" --set "CHIPNAME STM32L475VGXX" --set "CONNECT_UNDER_RESET 1" -f interface/stlink-dap.cfg -f target/stm32l4x.cfg -c init -c "reset init" -c "echo VisualGDB_OpenOCD_Ready"
Open On-Chip Debugger 0.10.0 (2021-03-26)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
STM32L475VGXX
1
dapdirect_swd
Info : STLINK V2J39M27 (API v2) VID:PID 0483:3748
Info : Target voltage: 3.217859
Info : clock speed 4000 kHz
Info : stlink_dap_op_connect(connect)
Info : SWD DPIDR 0x2ba01477
Info : STM32L475VGXX.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : STM32L475VGXX.cpu: external reset detected
Info : starting gdb server for STM32L475VGXX.cpu on 64567
Info : Listening on port 64567 for gdb connections
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08004620 msp: 0x20018000
VisualGDB_OpenOCD_Ready
Info : Listening on port 6666 for tcl connections
Info : Listening on port 64565 for telnet connections
Info : accepting 'gdb' connection on tcp/64567
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08004620 msp: 0x20018000
Info : device idcode = 0x10076415 (STM32L47/L48xx - Rev 4 : 0x1007)
Info : RDP level 0 (0xAA)
Info : flash size = 1024kbytes
Info : flash mode : dual-bank
Warn : Prefer GDB command "target extended-remote 64567" instead of "target remote 64567"
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08004620 msp: 0x20018000
Info : device idcode = 0x10076415 (STM32L47/L48xx - Rev 4 : 0x1007)
Info : RDP level 0 (0xAA)
Info : flash size = 1024kbytes
Info : flash mode : dual-bank
flash_bank_summary:0x8000000|0x100000|STM32L475VGXX.flash
FLASH progress reporting is now on

target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08004620 msp: 0x20018000
Info : Erasing FLASH: 0x08000000-0x08010800...
Info : Programming FLASH (2 sections, 66236 bytes)...
Info : Programming FLASH section 1/2 (392 bytes) at 0x08000000...
Info : Padding image section 0 at 0x08000188 with 8 bytes
Info : Padding image section 1 at 0x080102c4 with 4 bytes (bank write end alignment)
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08004620 msp: 0x20018000

```

## Build from Command-Line

If the STM32 toolchain is installed (for Windows) there's also an auto-generated [batch file](#) to build from command-line.

See also [Project Readme Build Instructions](#)

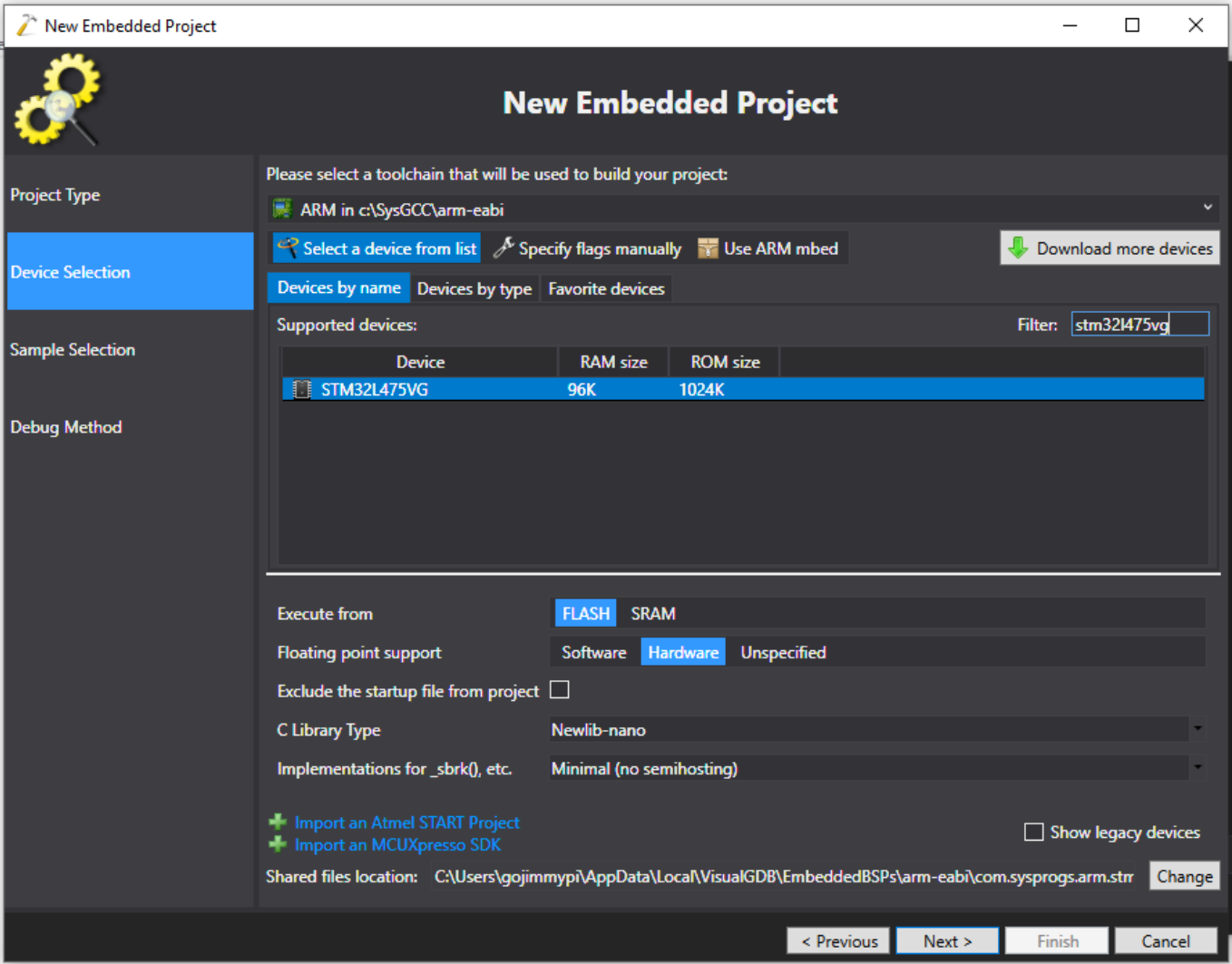


How to build the system (including the toolchain(s))

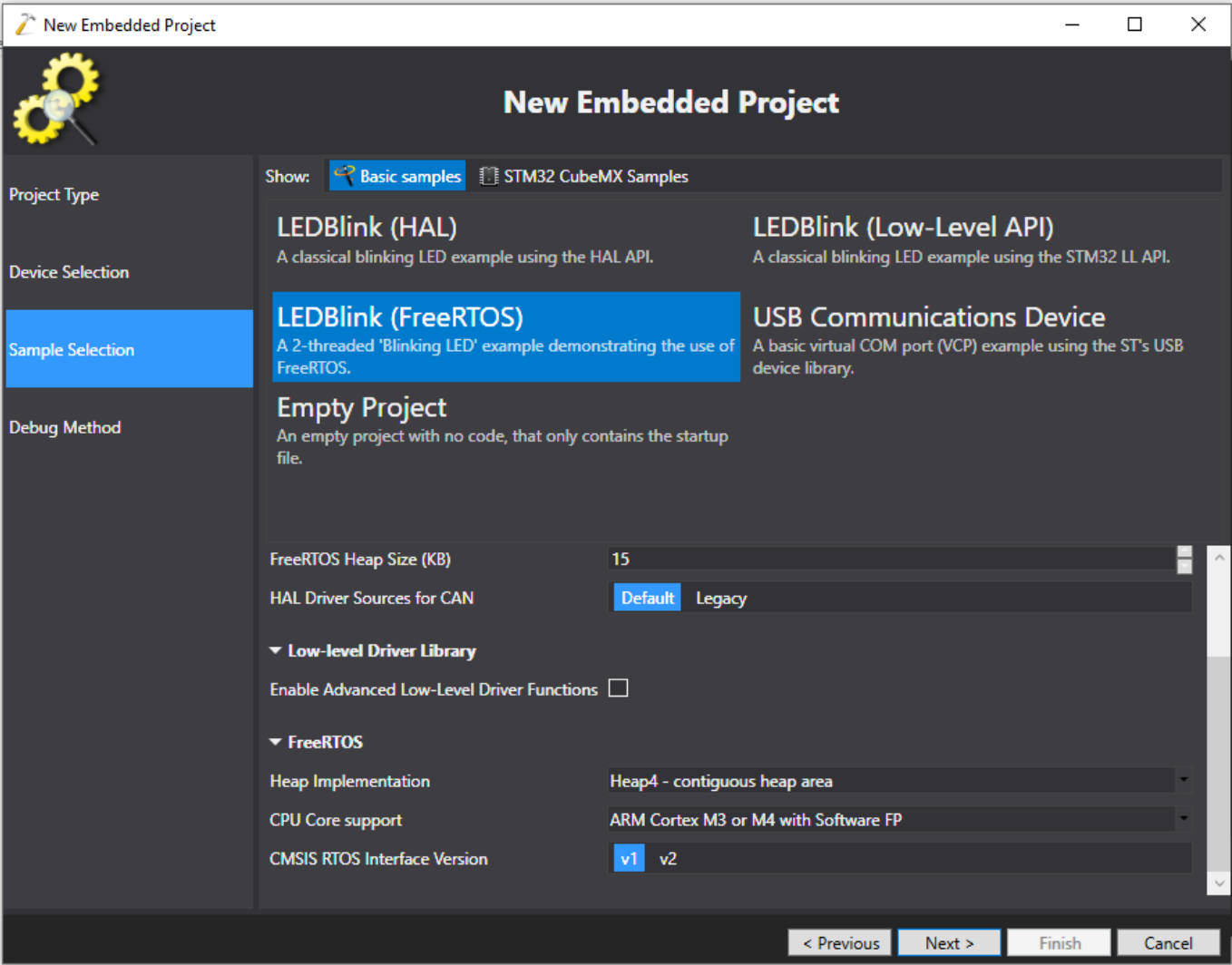
The only toolchain required for this project is the [VisualGDB Extension](#). See the [Developing STM32 projects with Visual Studio](#) walk-through.

These settings are included the project solution file but included here for reference:

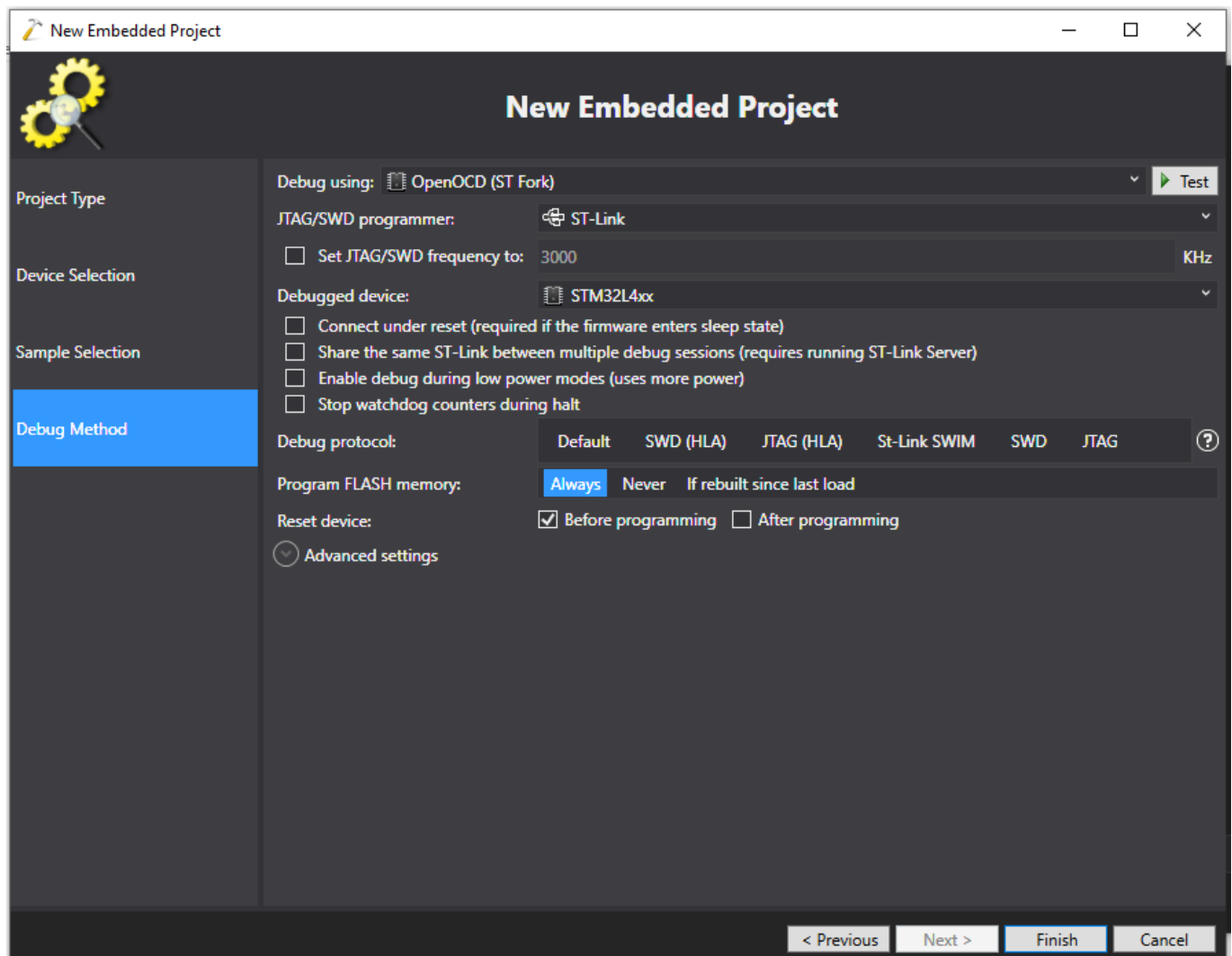
Project Config Step 1:



Project Config Step 2:



Project Config Step 3:

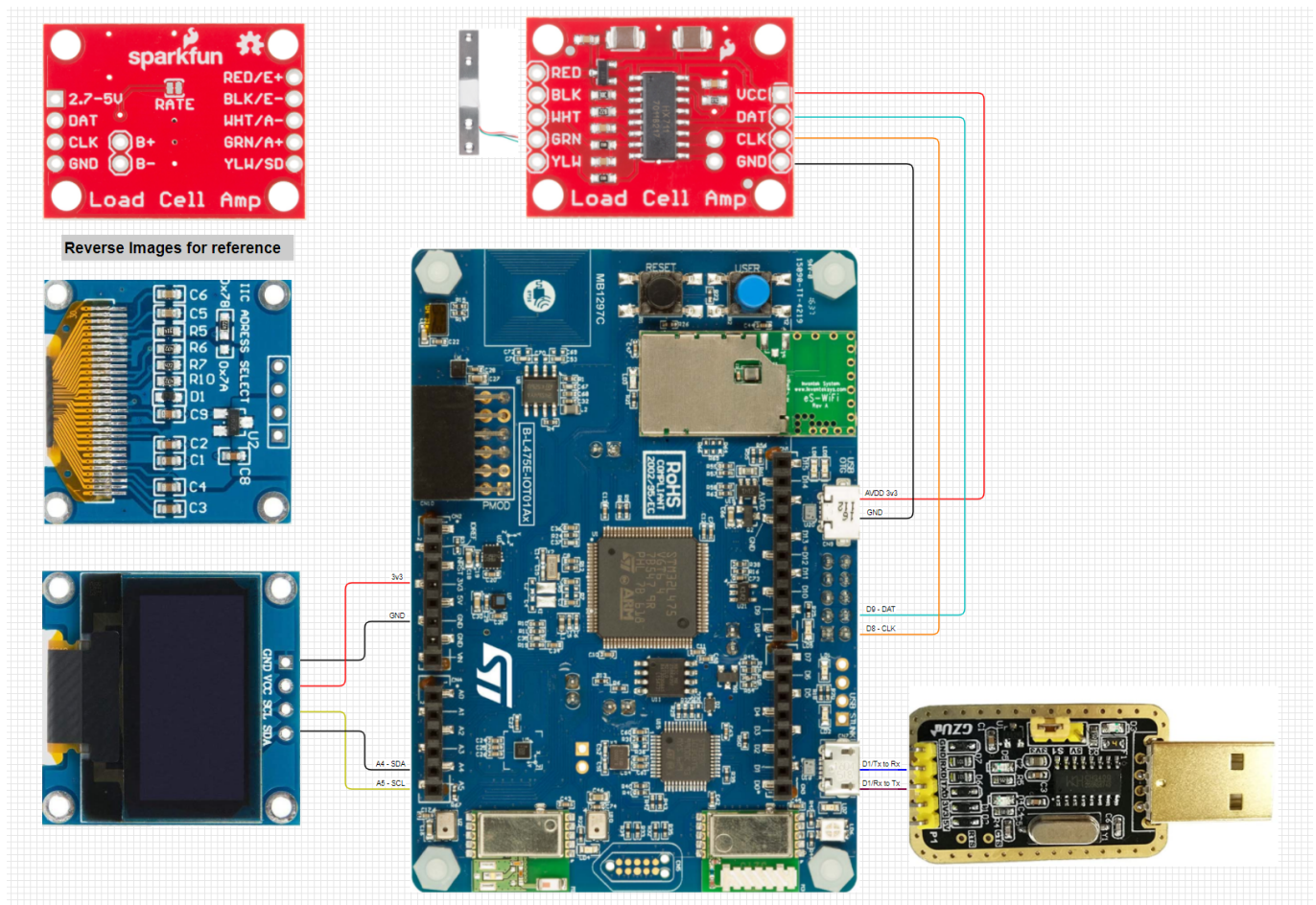


See above for the [build instructions](#).

## Hardware

- Mouser [ST B-L475E-IOT01A1](#)
- Amazon [SSD1306 I2C Serial](#)
- Sparkfun [Load Cell Amplifier HX711](#)
- Adafruit [Strain Gauge Load Cell - 4 Wires - 20Kg](#)

Hardware wiring diagram:



## Software

The source code can be downloaded from [GitHub gojimmypi/loT\\_BBQ](https://github.com/gojimmypi/loT_BBQ). The only other software needed is Visual Studio and VisualGDB (and an internet connection, as VisualGDB will download and install toolchains and libraries as needed.)

## How you debugged and tested the system

All of the development and debugging was completed in Visual Studio using breakpoints and the single-step-debugging features available from the VisualGDB extension.

Single-step inspection:

```
/**
 * @brief Read the specified input port pin.
 * @param GPIOx where x can be (A..H) to select the GPIO peripheral for STM32L4 family
 * @param GPIO_Pin specifies the port bit to read.
 *         This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
 * @retval The input port pin value.
 */
3 references (outdated)
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    GPIO_PinState bitstatus;

    /* Check the parameters */
    assert_param(IS_GPIO_PIN(GPIO_Pin));

    if ((GPIOx->IDR & GPIO_Pin) != 0x00u)
    {
        bitstatus = GPIO_PIN_SET;
    }
    else
    {
        bitstatus = GPIO_PIN_RESET;
    }
    return bitstatus;
}
```

Heap pointer check:

FileEditViewGitProjectBuildDebugTestAnalyzeToolsExtensionsWindowHelp

Search (Ctrl+Q)

IoT\_BBQ\_STM32

DebugVisualGDBContinue

heap\_4.cint\_to\_string.htemplate.cppint\_to\_string.cUART.hmain.cUART.cppSTM32L475VG\_flash.Ids

low\_power\_tick\_management\_LPTIM.c

(\*) (global scope)

PWM\_Thread1(const void \*)PWM\_Thread1(const void \*)::myHeapPointer

Go

```
volatile long myHeapPointer = pvPortMalloc(0); // this returns a NULL pointer
myHeapPointer = (long)pvPortMalloc(1); // a non-zero param returns heap pointer

UART_init();

// Print global variable buffer
UART_TxMessage(buffer, bufferLength);
UART_TxMessage(CrLf, bufferLength);

// show decimal value of Stack pointer
UART_TxMessageIntValue(StackMessage, bufferLength, myStackPointer);
UART_TxMessage(CrLf, bufferLength);

// show hex value of current stack pointer
UART_TxMessageIntValueHex(StackMessageHex, bufferLength, myStackPointer);
UART_TxMessage(CrLf, bufferLength);

// show hex value of current heap pointer (xstart is static, not available)
UART_TxMessageIntValueHex(HeapMessageHex, bufferLength, myHeapPointer);
UART_TxMessage(CrLf, bufferLength);

// show hex value of current heap size
UART_TxMessageIntValueHex(HeapMessageSizeHex, bufferLength, myHeapSize);
UART_TxMessage(CrLf, bufferLength);

// show free heap free size
UART_TxMessageIntValueHex(HeapMessageFreeHex, bufferLength, myHeapFreeSize);
UART_TxMessage(CrLf, bufferLength);
```

myHeapPointer0x20001ea0

100%

01

gojimmypi, 20 hours ago | 1 author, 20 changes

Ln: 428Ch: 73MIXEDCRLF

Live Watch

Mode:GlobalsWatchFreeRTOS

Expression	Address	Type	Value	Plot
Threads				
Synchronization Primitives				
Heap				
10144 bytes available				
- Allocated Bytes5056				
- Allocated Blocks17				
- Max. Allocation Size1024				
- Free Bytes10144				
- Free Blocks1				
- Max. Free Block Size10144				
[Heap Blocks] [18 blocks]				
[0x20000a68]	0x20000a68	80 bytes	00 00 00 00 68 0a ...	
[0x20000ac0]	0x20000ac0	512 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x20000cc8]	0x20000cc8	96 bytes	38 0c 00 20 2c 02 ...	
[0x20000d30]	0x20000d30	512 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x20000f38]	0x20000f38	96 bytes	a0 0e 00 20 c9 00 ...	
[0x20000fa0]	0x20000fa0	512 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x200011a8]	0x200011a8	96 bytes	00 11 00 20 00 00 ...	
[0x20001210]	0x20001210	512 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x20001418]	0x20001418	96 bytes	78 13 00 20 00 00 ...	
[0x20001480]	0x20001480	512 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x20001688]	0x20001688	96 bytes	78 15 00 20 00 00 ...	
[0x200016f0]	0x200016f0	512 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x200018f8]	0x200018f8	96 bytes	a8 18 00 20 00 00 ...	
[0x20001960]	0x20001960	200 bytes	b0 19 00 20 b0 19 ...	
[0x20001a30]	0x20001a30	1024 bytes	a5 a5 a5 a5 a5 a5 ...	
[0x20001e38]	0x20001e38	96 bytes	e8 1d 00 20 00 00 ...	
[0x20001ea0]	0x20001ea0	8 bytes	00 00 00 00 00 00 ...	
[0x20001eb0]	0x20001eb0	10144 bytes	unallocated	

0 active variables (0 bytes, 0 requests). Updating 9x per second (up to 31x/second possible).

Memory 1Find Symb...VisualGDB...Live WatchVisual Wat...VisualGDB...Hardware...GDB SessionDeveloper...Call StackException...Immediate...Error ListOutputFind Symb...BreakpointsLocalsWatch 1

Readygojimmypi / IoT\_BBQ

Memory address inspection:

The code editor shows the following code:

```
// show Initialized Variable Address
UART_TxMessageIntValueHex(InitMessageHex, bufferLenth, (long)&myInitializedVariable);
UART_TxMessage(CrLf, bufferLenth);

// show Uninitialized Variable Address
UART_TxMessageIntValueHex(NotInitMessageHex, bufferLenth, (long)&myNotInitializedVariable);
UART_TxMessage(CrLf, bufferLenth);

// static char numStr[32];
// int to_strine(numStr, 32, 42);
```

The Live Watch window shows the following variables:

Expression	Address	Type	Value	Plot
hi2c3	0x20000438	_I2C_HandleTypeDef	{...}	
htim2	0x20000588		{...}	
InitMessageHex	0x20004624	unsigned char[80]	{...}	
IsInitialized	0x20000558	8-bit unsigned (hex)	0x00	
IsSimInitialized	0x20000020	32-bit signed (dec)	1	
LEDThread1Handle	0x20000570	tskTaskControlBlock *	0x20000a00	

Breakpoints, code explorer, and other system monitoring features were quite helpful:

The screenshot shows several key features of the development environment:

- Barebones TIM2 handler:** A code snippet showing the initialization of the TIM2 interrupt handler.
- Startup:** A code snippet showing the initialization of the UART and the start of the scheduler.
- Exception Thrown:** A message indicating that an exception was thrown, specifically a SIGTRAP (trace/breakpoint trap).
- Call Stack at Fault Time:** A stack trace showing the sequence of function calls leading to the fault, including `HardFault_Handler`, `PendSV_Handler`, and `prvProcessTimerOrBlockTask`.
- Throughlist corrupted:** A message indicating that the throughlist was corrupted.
- Shortly before fault:** A Live Watch window showing the state of the system just before the fault, including the status of threads (Kernel, DISPLAY, LED1, LED2, PWM, Ter Svc, UART1) and the state of synchronization primitives and heap memory.

## Future

Although the discovery board was convenient from a *getting started* perspective, it is also massive and power hungry. The next steps are to port this project to a **Blue Pill**.



## What would be needed to get this project ready for production?

Key to a production deployment would be an *enclosure* and some way to have them mass-produced. For large-scale production, some means of mass-programming the devices would be needed. It would not be practical to manually plug in a million boards and program them each one at a time.

## How would you extend this project to do something more? Are there other features you'd like? How would you go about adding them?

I chose the this particular IoT Discovery board for the WiFi capabilities. As it turns out, the power consumption is simply way too high for battery operation. Our grill is located a bit far from the house; I certainly don't want to run an extension cord. I have some [LoRa experience](#) and plan to see if I can get that working with an MQTT server.

There are also plans for including a firebox temperature sensor, specifically the MAX6675 Module + K Type Thermocouple Temperature as described in the [original README](#) from a few years back when I created something for the ESP8266.

## Grading

Self assessment of the project: for each criteria, choose a score (1, 2, 3) and explain your reason for the score in 1-2 sentences. Have you gone beyond the base requirements? How so?

Criteria	1 - Needs Improvement	2 - Meets Expectations	3 - Exceeds Expectations
Project meets minimum project goals		2+ <a href="#">Basic State Machine</a> , additional sensors <a href="#">LPS22HB</a> , <a href="#">HX711</a> , well documented. No serial commands	
Completeness of deliverables			Readable code, <a href="#">each point addressed</a> , <a href="#">video</a>
Clear intentions and working code		2+ System works as decribed	(docs not as professionally polished as I'd like)
Reusing code			<a href="#">License summary</a> and <a href="#">Build Info</a>
Originality and scope of goals			This is clearly an awesome Propane Tank Weight Project! 😊 The scope of goals was appropriately challenging yet reasonable for the class.

Criteria	1 - Needs Improvement	2 - Meets Expectations	3 - Exceeds Expectations
Self-assessment (Mentor category only)		n/a	
Bonus: Power analysis, firmware update, or system profiling			<a href="#">Power Analysis</a> also <a href="#">shared on Twitter</a>
! Bonus: Version control was used.			<a href="#">hundreds of commits</a>
<ul style="list-style-type: none"><li>I created the video footage of the project in action on my phone and took all the pictures, but the final consolidated video was assembled and subtitled by someone else. Thank you!!!</li></ul>			

See the Class Assignment [additional links](#) for other resources.