

# UCSD Embedded RTOS

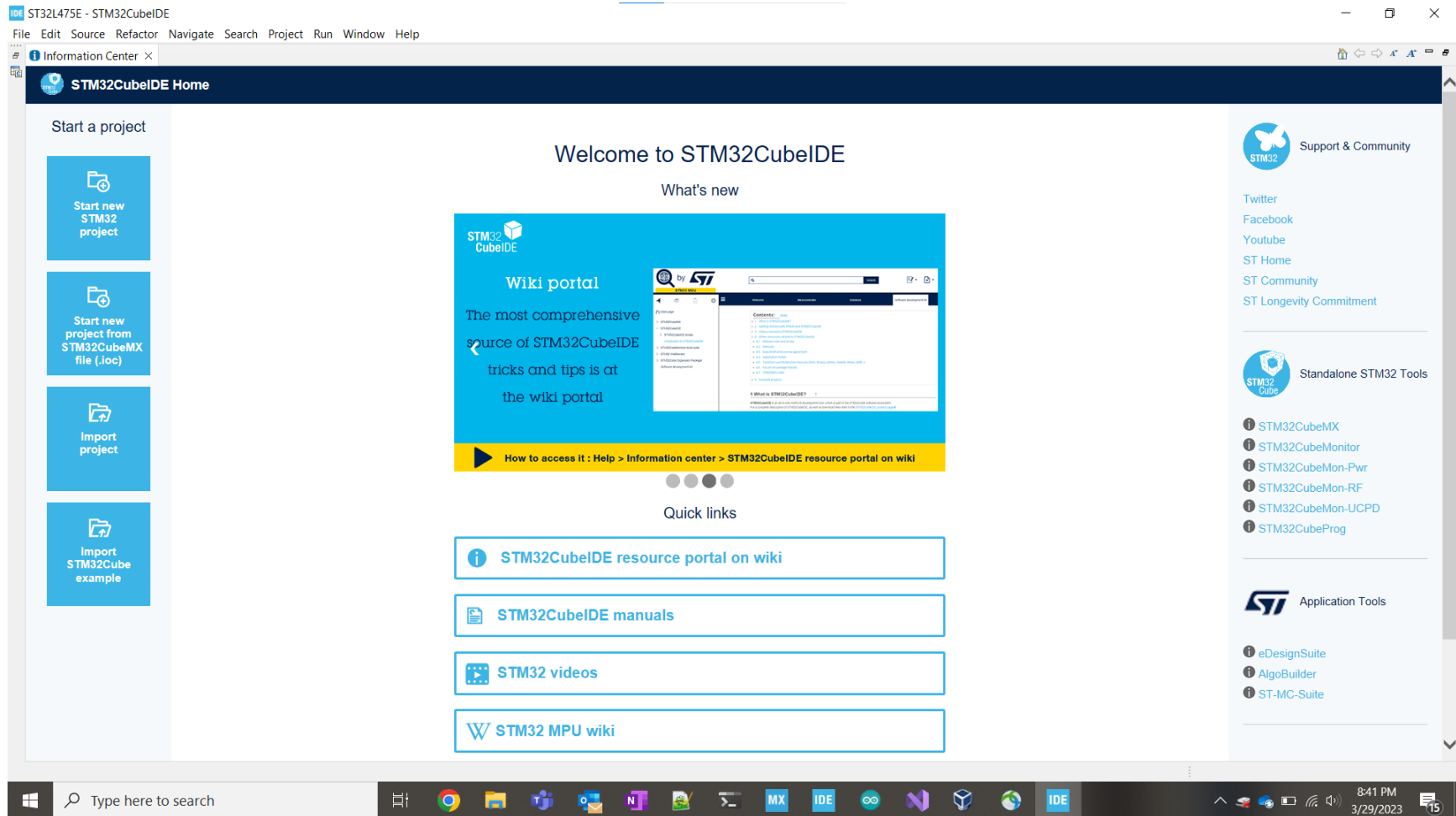
## Final Project

By

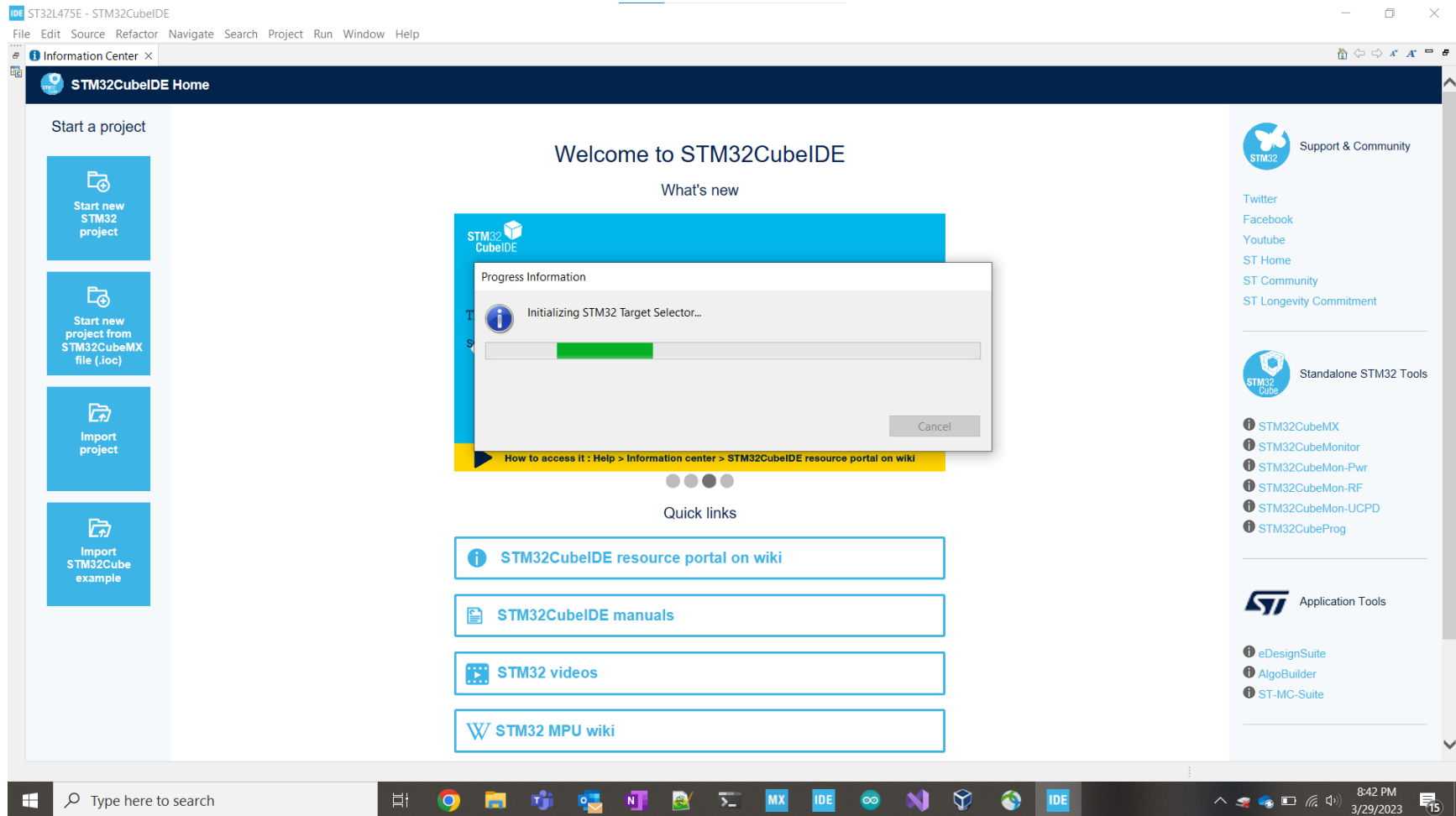
Fidel Quezada Guzman

[Fidel2993@gmail.com](mailto:Fidel2993@gmail.com)

# Step 1. Startup STM32CubeMX



## Step 2. Access Board Selector



## Step 3. Enter “B-L475-IOT01A” Board

The screenshot shows the STM32CubeIDE Target Selection dialog. The 'Board Selector' tab is active, displaying filters and a list of boards. The 'Commercial Part Number' filter is set to 'B-L475E-IOT01A'. The 'PRODUCT INFO' section shows details for the selected board. The 'MEMORY' section shows the configuration for the board. The 'Boards List' shows two items: 'B-L475E-IOT01A1' and 'B-L475E-IOT01A2'.

**Target Selection**  
STM32 target or STM32Cube example selection is required

**Board Filters**

Commercial Part Number: B-L475E-IOT01A

**PRODUCT INFO**

- Type
- Supplier
- MCU / MPU Series
- Marketing Status
- Price

**MEMORY**

- Ext. Flash = 64 (MBit)
- Ext. EEPROM = 0 (kBytes)
- Ext. RAM = 0 (MBit)

**Boards List: 2 items**

	Commercial Part No
★	B-L475E-IOT01A1
★	B-L475E-IOT01A2

**Support & Community**

- Twitter
- Facebook
- Youtube
- ST Home
- ST Community
- ST Longevity Commitment

**Standalone STM32 Tools**

- STM32CubeMX
- STM32CubeMonitor
- STM32CubeMon-Pwr
- STM32CubeMon-RF
- STM32CubeMon-UCPD
- STM32CubeProg

**Application Tools**

- eDesignSuite
- AlgoBuilder
- ST-MC-Suite

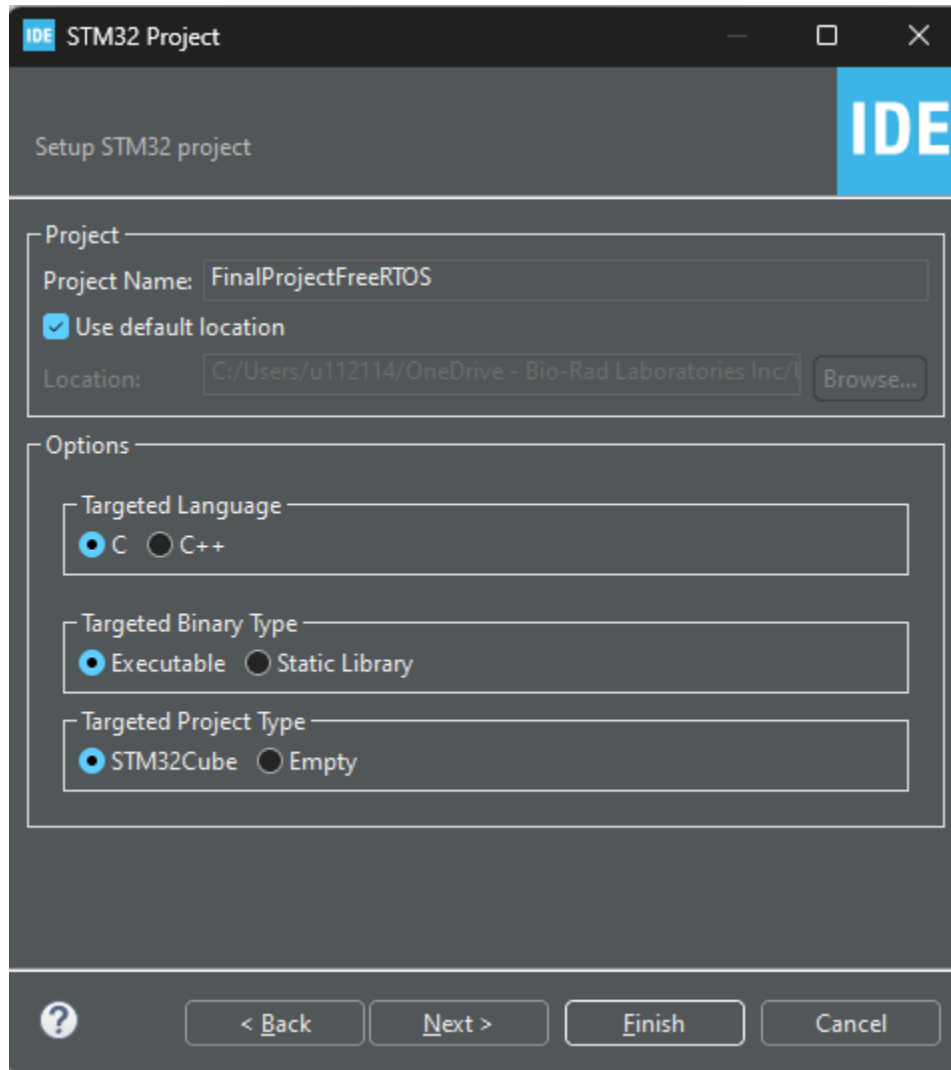
## Step 4. Select Board Photo

The screenshot shows the STM32CubeIDE interface with the 'Target Selection' window open. The window is titled 'STM32 Project' and has tabs for 'MCU/MPU Selector', 'Board Selector', 'Example Selector', and 'Cross Selector'. The 'Board Selector' tab is active, showing a search for 'B-L475E-IOT01A'. The 'PRODUCT INFO' section on the left lists details like Type, Supplier, MCU / MPU Series, Marketing Status, and Price. The 'MEMORY' section shows 'Ext. Flash = 64 (MBit)', 'Ext. EEPROM = 0 (kBytes)', and 'Ext. RAM = 0 (MBit)'. The 'Boards List' on the right shows two items: 'B-L475E-IOT01A1' and 'B-L475E-IOT01A2'. The 'B-L475E-IOT01A1' board is highlighted, showing its features: 'STM32L4 Discovery kit IoT node, low-power wireless, BLE, NFC, SubGHz, Wi-Fi'. The board is marked as 'ACTIVE' and 'Product is in mass production'. The 'Unit Price (US\$)' is \$53.0, and the 'Mounted Device' is 'STM32L475VGT6'. The 'Boards List' table has two items:

	Commercial Part No
★	B-L475E-IOT01A1
☆	B-L475E-IOT01A2

The 'Next >' button is highlighted, indicating the next step in the process. The background shows the STM32CubeIDE Home page with options to 'Start a project', 'Start new STM32 project', 'Start new project from STM32CubeMX file (.ioc)', 'Import project', and 'Import STM32Cube example'. The right sidebar shows 'Support & Community' links (Twitter, Facebook, Youtube, ST Home, ST Community, ST Longevity Commitment) and 'Standalone STM32 Tools' (STM32CubeMX, STM32CubeMonitor, STM32CubeMon-Pwr, STM32CubeMon-RF, STM32CubeMon-UCPD, STM32CubeProg). The bottom status bar shows the time as 8:43 PM on 3/29/2023.

## Step 5. Name Project & Select “Finish”



The image shows a 'Setup STM32 project' dialog box with a dark theme. The title bar says 'IDE STM32 Project'. The main area is divided into 'Project' and 'Options' sections. In the 'Project' section, the 'Project Name' is 'FinalProjectFreeRTOS', 'Use default location' is checked, and the 'Location' is 'C:/Users/u112114/OneDrive - Bio-Rad Laboratories Inc/'. In the 'Options' section, 'Targeted Language' is 'C', 'Targeted Binary Type' is 'Executable', and 'Targeted Project Type' is 'STM32Cube'. At the bottom are buttons for '?', '< Back', 'Next >', 'Finish', and 'Cancel'.

IDE STM32 Project

Setup STM32 project

Project

Project Name: FinalProjectFreeRTOS

☒ Use default location

Location: C:/Users/u112114/OneDrive - Bio-Rad Laboratories Inc/ Browse...

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

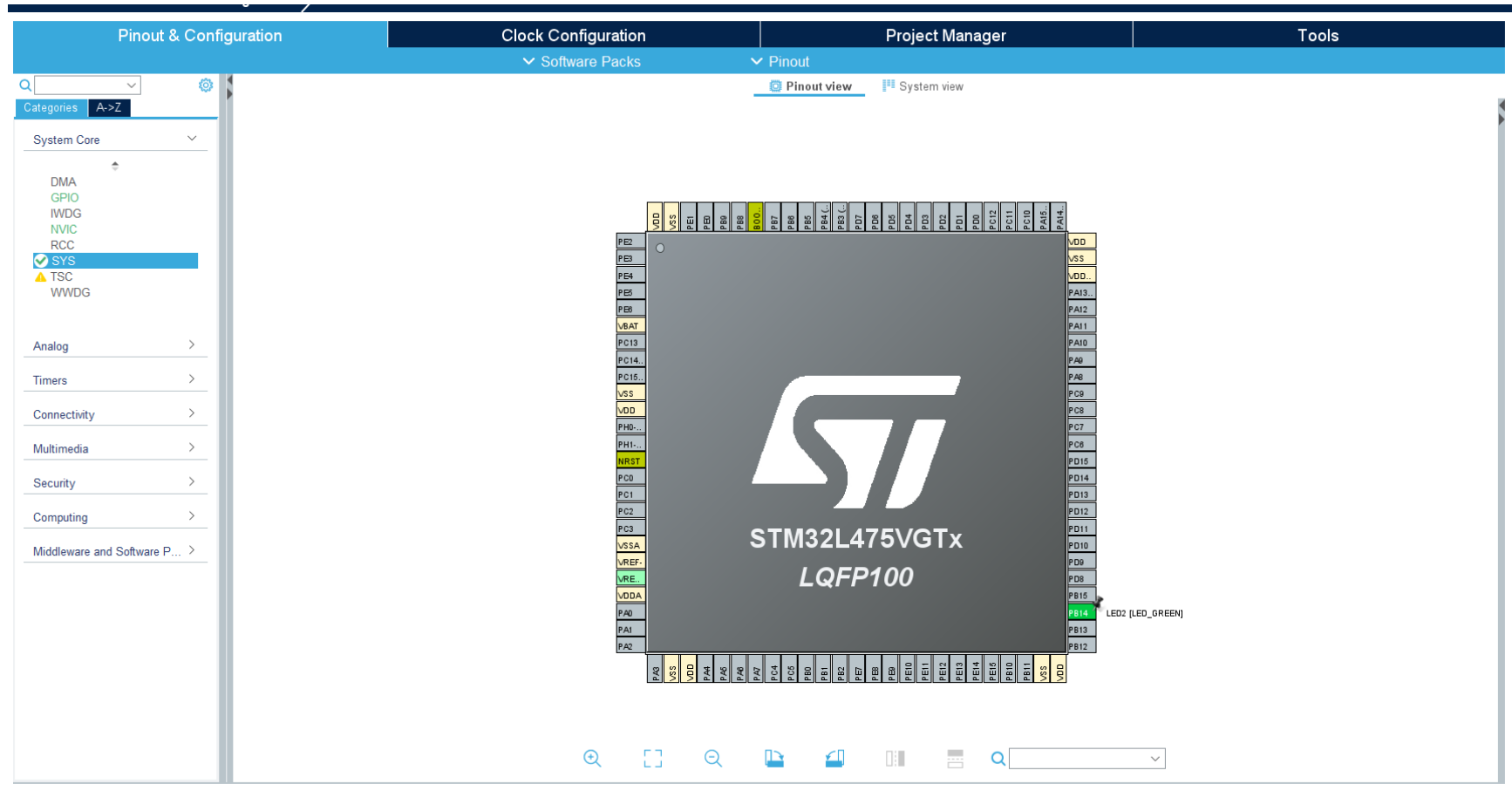
☒ Executable ☐ Static Library

Targeted Project Type

☒ STM32Cube ☐ Empty

? < Back Next > Finish Cancel

## Step 6. Observe Results (Pinout View)



Step 7. Enable UART1 TX/RX for Console Input/Output in Asynchronous Mode & enable the global interrupt for HAL\_UART\_TxCpltCallback functionality (non-blocking)

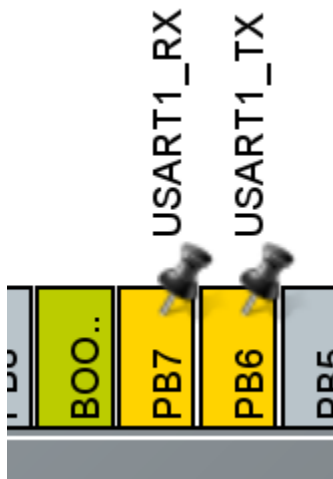


Diagram showing the connection of USART1 pins to the microcontroller pins:

- USART1\_RX is connected to PB7.
- USART1\_TX is connected to PB6.

The pins are labeled BOO., PB7, PB6, and DR5.

### Mode

Mode: Asynchronous

Hardware Flow Control (RS232): Disable

☐ Hardware Flow Control (RS485)

### Configuration

[Reset Configuration](#)

☒ Parameter Settings | ☒ User Constants | ☒ NVIC Settings | ☒ DMA Settings | ☒ GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART1 global interrupt	<input checked="" type="checkbox"/>	5	0



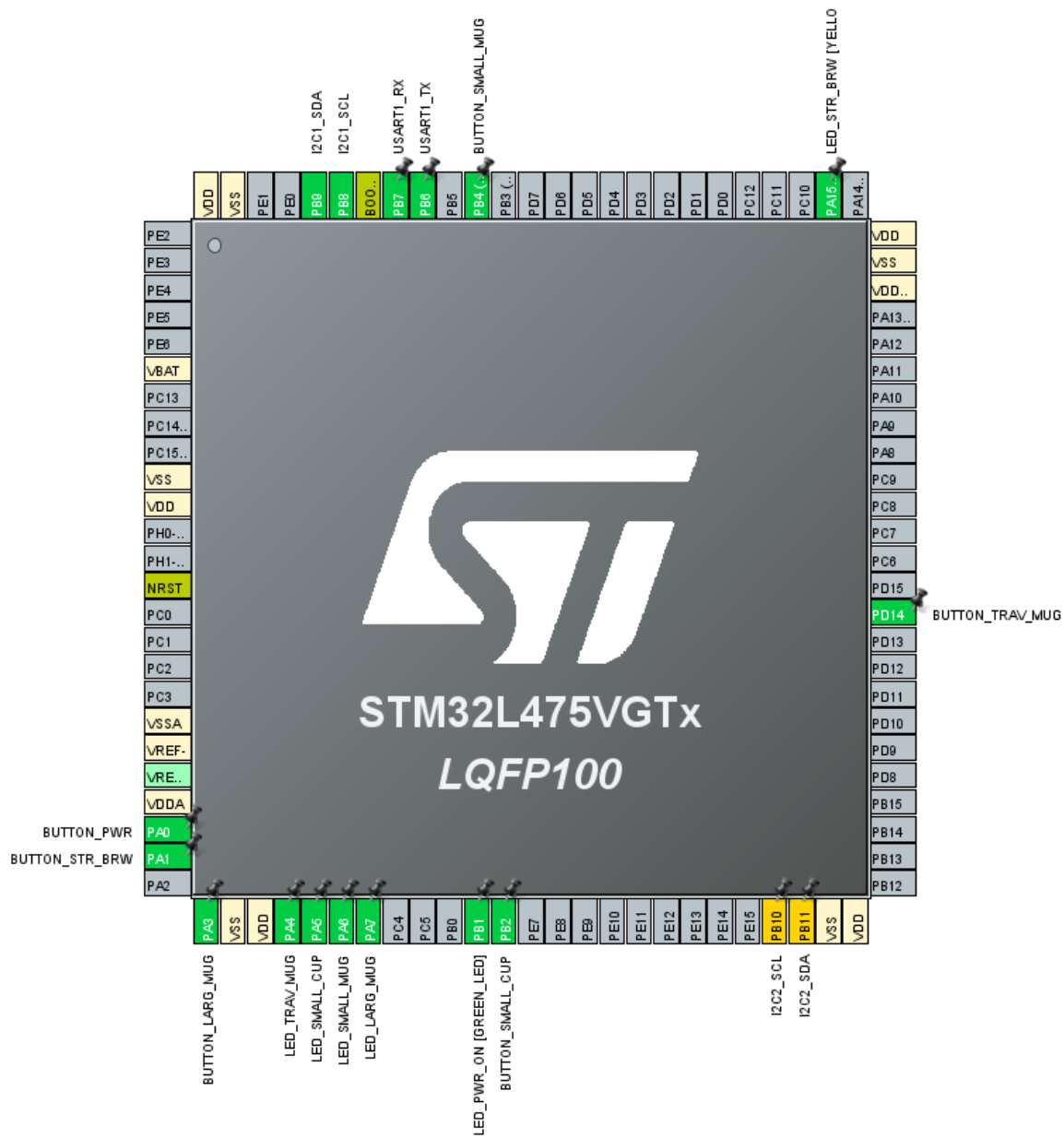
## Step 8. Enable Input GPIO Pins & respective Interrupts for Button usage

Pin Name	Signal on Pin	GPIO output I...	GPIO mode	...	...	Fas...	User Label	Modified
PA0	n/a	n/a	External Interr...	...	n/a	n/a	BUTTON_PWR	✓
PA1	n/a	n/a	External Interr...	...	n/a	n/a	BUTTON_STR_BRW	✓
PA3	n/a	n/a	External Interr...	...	n/a	n/a	BUTTON_LARGE_MUG	✓
PB2	n/a	n/a	External Interr...	...	n/a	n/a	BUTTON_SMALL_CUP	✓
PB4 (NJTRST)	n/a	n/a	External Interr...	...	n/a	n/a	BUTTON_SMALL_MUG	✓
PD14	n/a	n/a	External Interr...	...	n/a	n/a	BUTTON_TRAV_MUG	✓

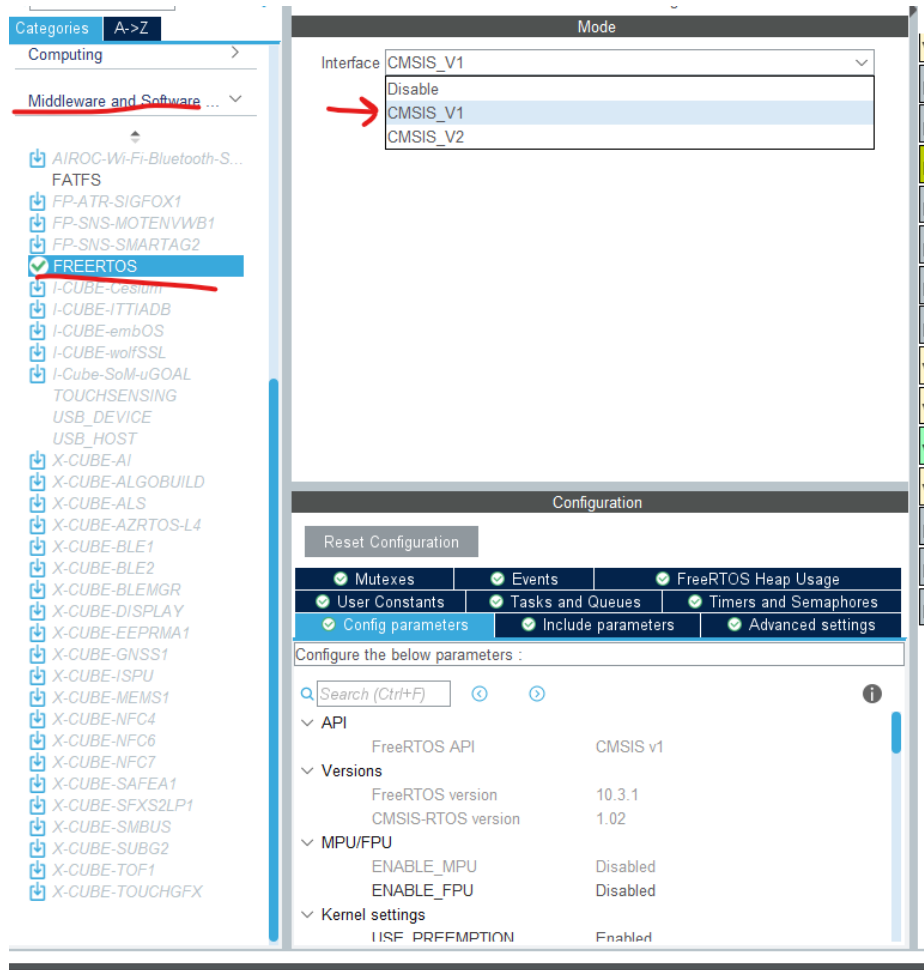
Configuration			
Group By Peripherals			
✓ GPIO	✓ Single Mapped Signals	✓ USART	✓ NVIC
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line0 interrupt	✓	0	0
EXTI line1 interrupt	✓	0	0
EXTI line2 interrupt	✓	0	0
EXTI line3 interrupt	✓	0	0
EXTI line[15:10] interrupts	✓	0	0

## Step 9. Enable Output GPIO Pins for LED usage

PA4	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_TRAV_MUG	<input checked="" type="checkbox"/>
PA5	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_SMALL_CUP	<input checked="" type="checkbox"/>
PA6	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_SMALL_MUG	<input checked="" type="checkbox"/>
PA7	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_LARGE_MUG	<input checked="" type="checkbox"/>
PA15 (JTDI)	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_STR_BRW [YELLOW_LED]	<input checked="" type="checkbox"/>
PB1	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_PWR_ON [GREEN_LED]	<input checked="" type="checkbox"/>



## Step 10. Enable FreeRTOS via Middleware and Software Tab using CMSIS\_v1.



## Step 11. We will use the following Tasks

Configuration

Reset Configuration

☒ User Constants

☒ Tasks and Queues

☒ Timers and Semaphores

☒ Mutexes

☒ Events

☒ FreeRTOS Heap Usage

☒ Config parameters

☒ Include parameters

☒ Advanced settings

Tasks

Task Name	Priority	Stack Size (Words)	Entry Function	Code Generation Op...	Parameter	Allocation	Buffer Name	Control Block Name
defaultTask	osPriorityAboveNormal	128	StartDefaultTask	Default	NULL	Dynamic	NULL	NULL
myTask02	osPriorityNormal	128	powerOnTask	Default	NULL	Dynamic	NULL	NULL
myTask03	osPriorityBelowNormal	128	strongBrewTask	Default	NULL	Dynamic	NULL	NULL
myTask04	osPriorityBelowNormal	128	brewTravelMug	Default	NULL	Dynamic	NULL	NULL
myTask05	osPriorityBelowNormal	128	brewLargeMug	Default	NULL	Dynamic	NULL	NULL
myTask06	osPriorityBelowNormal	128	brewSmallMug	Default	NULL	Dynamic	NULL	NULL
myTask07	osPriorityBelowNormal	128	brewSmallCup	Default	NULL	Dynamic	NULL	NULL

AddDelete

## Step 12. Create Binary Semaphores as well to manage Power & strongBrew features

Binary Semaphores		
Semaphore Name	Allocation	Control Block Name
powerOnSem	Dynamic	NULL
strongBrewSem	Dynamic	NULL
<div> <div>Add</div> <div>Delete</div> </div>		

## Step 13. For Group events: show enabling Timer & keep defaults, also Heap increase usage

FreeRTOS Configuration

<input checked="" type="checkbox"/> User Constants	<input checked="" type="checkbox"/> Tasks and Queues	<input checked="" type="checkbox"/> Timers and Semaphores	<input checked="" type="checkbox"/> Mutexes	<input checked="" type="checkbox"/> Events	<input checked="" type="checkbox"/> FreeRTOS Heap Usage
<input checked="" type="checkbox"/> Config parameters		<input checked="" type="checkbox"/> Include parameters		<input checked="" type="checkbox"/> Advanced settings	

Configure the below parameters :

RECORD_STACK_HIGH_ADDRESS	Disabled
Memory management settings	
Memory Allocation	Dynamic / Static
TOTAL_HEAP_SIZE	6000 Bytes
Memory Management scheme	heap_4
Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
USE_DAEMON_TASK_STARTUP_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
Run time and task stats gathering related definitions	
GENERATE_RUN_TIME_STATS	Disabled
USE_TRACE_FACILITY	Disabled
USE_STATS_FORMATTING_FUNCTIONS	Disabled
Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
Software timer definitions	
USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10
TIMER_TASK_STACK_DEPTH	256 Words
Interrupt nesting behaviour configuration	
LIBRARY_LOWEST_INTERRUPT_PRIORITY	15
LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	5
Added with 10.2.1 support	
MESSAGE_BUFFER_LENGTH_TYPE	size_t
USE_POSIX_ERRNO	Disabled

### Step 14. Enable a oneShot timer as well for the auto-shut OFF

Timers						
Timer Name	Callback	Type	Code Generation Option	Parameter	Allocation	Control Block Name
autoOffTimer	pvAutoOffTimerOneShot	osTimerOnce	Default	NULL	Dynamic	NULL



Step 15. Now we will also use Event Groups to manage the type of brews

Setting parameters

Include parameters

Advanced settings

Mutexes

Mutex Name	Allocation	Control Block Name
brewingMutex	Dynamic	NULL

Add

Delete

Recursive Mutexes

## Step 16. Confirm our Heap is good!

Configuration

Reset Configuration

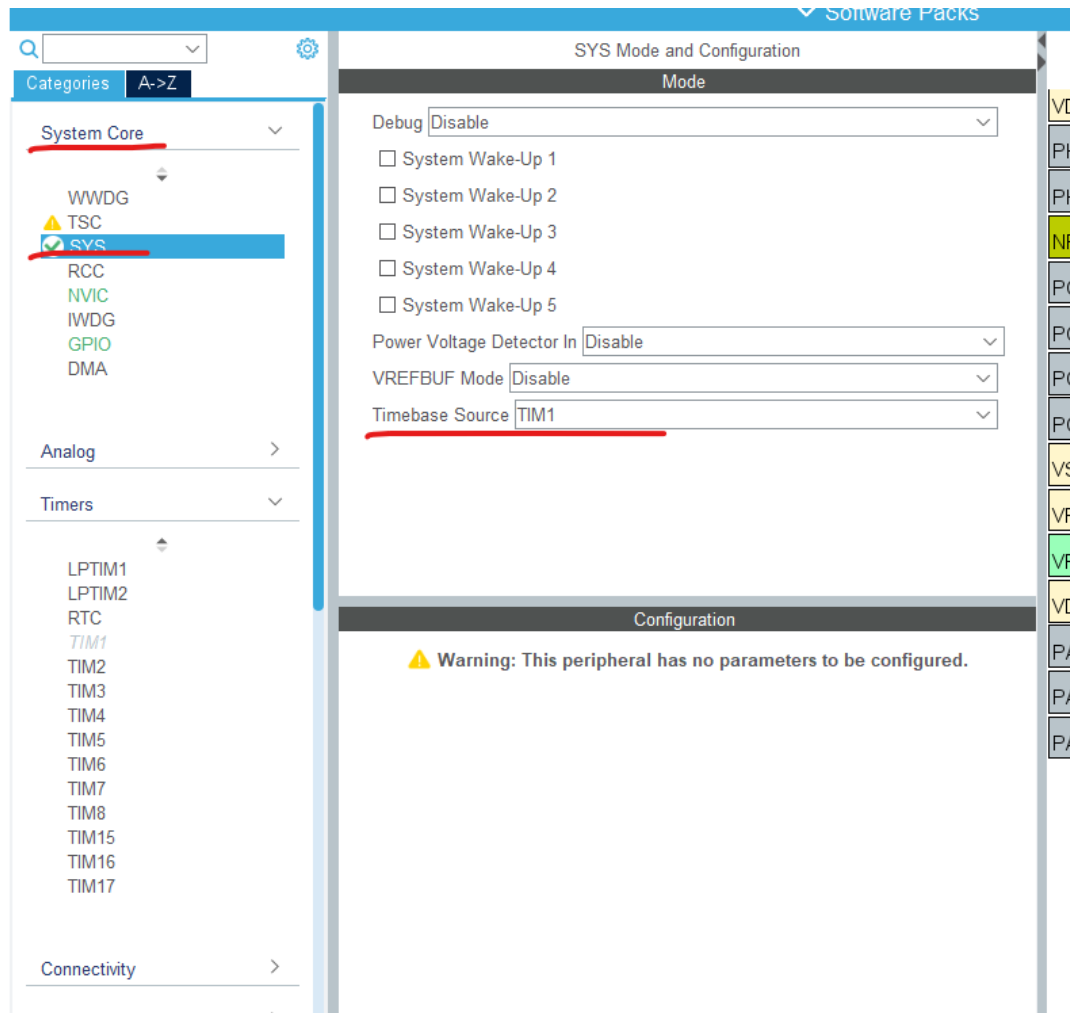
- ✓ User Constants
- ✓ Tasks and Queues
- ✓ Timers and Semaphores
- ✓ Mutexes
- ✓ Events
- ✓ FreeRTOS Heap Usage

- ✓ Config parameters
- ✓ Include parameters
- ✓ Advanced settings

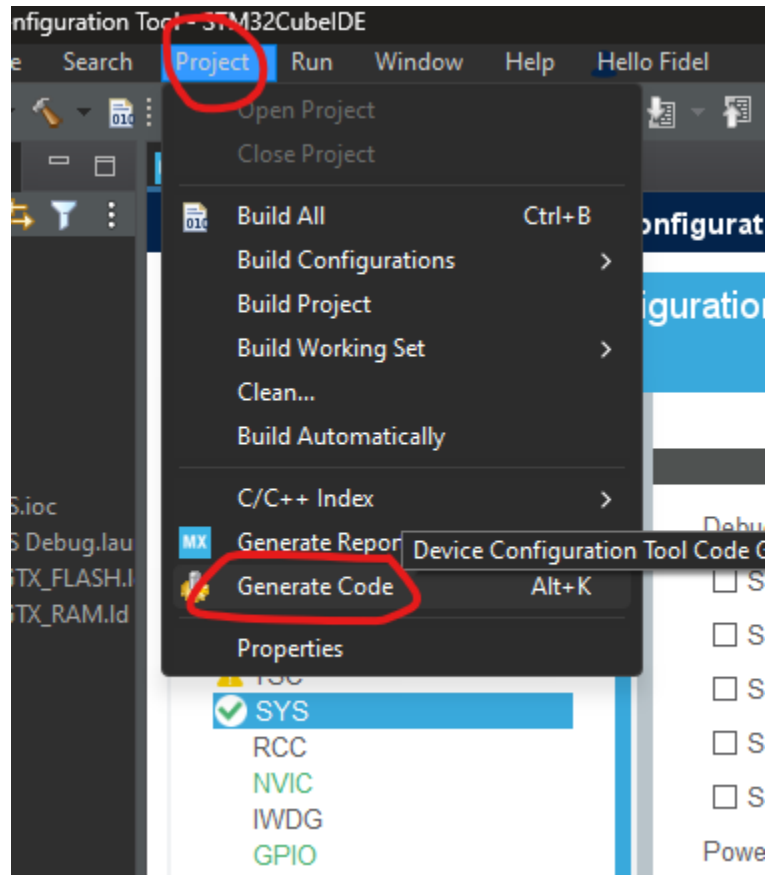
Summary

HEAP STILL AVAILABLE	1320 Bytes
TOTAL HEAP USED	4680 Bytes
Total amount for tasks	4368 Bytes
Total amount for queues	0 Bytes
Total amount for timers	48 Bytes
Total amount for mutexes and semaphores	264 Bytes
Total amount for events	0 Bytes
FreeRTOS tasks	
Idle task (FreeRTOS internal)	0 Bytes
Timer service task (FreeRTOS internal)	0 Bytes
defaultTask	624 Bytes
myTask02	624 Bytes
myTask03	624 Bytes
myTask04	624 Bytes
myTask05	624 Bytes
myTask06	624 Bytes
myTask07	624 Bytes
FreeRTOS timers	
autoOffTimer	48 Bytes
FreeRTOS mutexes and semaphores	
brewingMutex	88 Bytes
powerOnSem	88 Bytes
strongBrewSem	88 Bytes

## Step 17. Change System timebase from SysTick default/



## Step 18. Generate Code!!



## Step 19. Create function to facilitate IT UART transmission

```
113 // Transmit complete callback function
114 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
115 {
116     if (huart->Instance == USART1)
117     {
118         uart_tx_complete = 1; // Transmission complete
119     }
120 }
```

## Step 20. Fancy progress bar animation

```
122 // Progress Bar Animation
123 void updateProgressBar(uint8_t progress, uint8_t oz)
124 {
125     // Width constant/ variable
126     const uint8_t barWidth = 40;
127     uint8_t completedWidth = barWidth * progress / oz;
128
129     char progressBar[barWidth + 1]; // +1 for null terminator
130     memset(progressBar, ' ', barWidth);
131     progressBar[barWidth] = '\0';
132
133     // Fill Up Progress
134     for (uint8_t i = 0; i < completedWidth; ++i)
135     {
136         progressBar[i] = '*';
137     }
138
139     // Adjust size as needed
140     char output[50];
141     snprintf(output, sizeof(output), "[%s] %d oz\r\n", progressBar, progress);
142
143     // Copy the output to uart_buffer
144     snprintf(uart_buffer, sizeof(uart_buffer), "%s", output);
145
146     // Start UART transmission using interrupt-based function
147     uart_tx_complete = 0; // Transmission in progress. Callback function will set it to 1 when complete
148
149     // Start non-blocking UART transmission using HAL_UART_Transmit_IT
150     if (HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer)) != HAL_OK)
151     {
152         // Handle transmit error (optional)
153         snprintf(uart_buffer, sizeof(uart_buffer), "ERROR TRANSMITTING!\n");
154         HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer));
155     }
156     HAL_Delay(300);
157 }
```

## Step 21. Create function to facilitate printing

```
159 // Function to print value based on type
160 void printIT(void* value, ValueType type)
161 {
162     // Buffer message
163     switch (type)
164     {
165         case CHAR_TYPE:
166             snprintf(uart_buffer, sizeof(uart_buffer), "%s", (char*)value);
167             break;
168         case UINT8_TYPE:
169             snprintf(uart_buffer, sizeof(uart_buffer), "%u", *(uint8_t*)value);
170             break;
171         default:
172             snprintf(uart_buffer, sizeof(uart_buffer), "Unknown type\n");
173             break;
174     }
175
176     // Start UART transmission using interrupt-based function
177     uart_tx_complete = 0; // Transmission in progress. Callback function will set it to 1 when complete
178
179     // Start non-blocking UART transmission using HAL_UART_Transmit_IT
180     if (HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer)) != HAL_OK)
181     {
182         // Handle transmit error (optional)
183         snprintf(uart_buffer, sizeof(uart_buffer), "ERROR TRANSMITTING!\n");
184         HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer));
185     }
186 }
```

## Step 22. Create startup & Power messages

```
188 // Startup Prompt
189 void startupPrompt(void)
190 {
191     // Concatenate all lines into the buffer
192     snprintf(uart_buffer, sizeof(uart_buffer), "\n\n\n*****\n");
193     snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "UCSD Spring 2024 *\\n");
194     snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "Embedded RTOS *\\n");
195     snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "Fidel Quesada Guzman *\\n");
196     snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "*****\\n\\n\\n");
197     snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "\\tFidelicious Coffee Maker\\n");
198     snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "\\nPress the ON Button to Start the Revival Process\\n");
199
200     // Start UART transmission using interrupt-based function
201     uart_tx_complete = 0; // Transmission in progress. Callback function will set it to 1 when complete
202
203     // Start non-blocking UART transmission using HAL_UART_Transmit_IT
204     if (HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer)) != HAL_OK)
205     {
206         // Handle transmit error (optional)
207         snprintf(uart_buffer, sizeof(uart_buffer), "ERROR TRANSMITTING!");
208         HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer));
209     }
210 }
211
212 // Coffee Machine ON/Off Message
213 void powerMessage()
214 {
215     // print Messages
216     char message1[20] = "Coffee Maker ON\\n";
217     char message2[20] = "Coffee Maker OFF\\n";
218
219     if(pwr_status == TRUE)
220     {
221         printIT(message1, CHAR_TYPE);
222     }
223     else if(pwr_status == FALSE)
224     {
225         printIT(message2, CHAR_TYPE);
226     }
227 }
```



## Step 23. Create messages in charge of the autoOffTimer, StrongBrewing & Brew type

```
229 void autoOffTimerMessage()
230 {
231     // print Message
232     char message[35] = "Auto Turning OFF Coffee Machine\n";
233
234     printIT(message, CHAR_TYPE);
235 }
236
237 void strongBrewMessage()
238 {
239     // print Messages
240     char message1[20] = "Strong Brew\n";
241     char message2[20] = "Normal Brew\n";
242
243     if(strongBrewCoffee == FALSE)
244     {
245         printIT(message1, CHAR_TYPE);
246     }
247     else if(strongBrewCoffee == TRUE)
248     {
249         printIT(message2, CHAR_TYPE);
250     }
251 }
252
253 void brewTypeMessage(uint8_t* sizeBrew)
254 {
255     // print options
256     char strong[15] = "Strong Brew: ";
257     char normal[15] = "Normal Brew: ";
258
259     if (strongBrewCoffee == TRUE)
260     {
261         printIT(strong, CHAR_TYPE);
262         printIT((void*)&sizeBrew, UINT8_TYPE);
263         printIT("oz\n", CHAR_TYPE);
264     }
265     else
266     {
267         printIT(normal, CHAR_TYPE);
268         printIT((void*)&sizeBrew, UINT8_TYPE);
269         printIT("oz\n", CHAR_TYPE);
270     }
271 }
```

## Step 24. Create functions to manage water levels & auto-refill when needed

```
273 void waterLevelMessage(uint8_t* waterLevel)
274 {
275     // print options
276     char message[15] = "Water Level: ";
277     char newLine[3] = "\n";
278
279     // print
280     printIT(message, CHAR_TYPE);
281     printIT((void*)&waterLevel, UINT8_TYPE);
282     printIT(newLine, CHAR_TYPE);
283 }
284
285 // Check if we have enough water
286 bool enoughWaterCheck(uint8_t coffeeSize)
287 {
288     char message1[50] = "Water Level is too low!!\nAuto-Refilling now...\n";
289     char message2[55] = "Water Level is full now!\nPlease select Coffee Again\n";
290
291     // Check if we will have enough water after CoffeeSize selection
292     if(waterLevel < coffeeSize)
293     {
294         // Print Message to user
295         printIT(message1, CHAR_TYPE);
296
297         // Auto-Refill Water w/ animation
298         for (int i = 0; i <= 40; i++)
299         {
300             updateProgressBar(i, 40);
301         }
302         waterLevel = 40;
303
304         // Print Message to user
305         printIT(message2, CHAR_TYPE);
306
307         return FALSE;    // wont proceed w/ brew
308     }
309     else
310     {
311         return TRUE;     // proceed w/ brew
312     }
313 }
```

Step 25. Create Event Group manually & confirm other FreeRTOS objects are created properly

```
/* USER CODE BEGIN 1 */

// Event Group Creation
xEventGroup = xEventGroupCreate();

/* USER CODE END 1 */

/* MCU Configuration-----

/* Reset of all peripherals, Initializes the Flash int
HAL_Init();
```

```

364  /* Create the mutex(es) */
365  /* definition and creation of brewingMutex */
366  osMutexDef(brewingMutex);
367  brewingMutexHandle = osMutexCreate(osMutex(brewingMutex));
368
369  /* USER CODE BEGIN RTOS_MUTEX */
370  /* add mutexes, ... */
371  /* USER CODE END RTOS_MUTEX */
372
373  /* Create the semaphores(s) */
374  /* definition and creation of powerOnSem */
375  osSemaphoreDef(powerOnSem);
376  powerOnSemHandle = osSemaphoreCreate(osSemaphore(powerOnSem), 1);
377
378  /* definition and creation of strongBrewSem */
379  osSemaphoreDef(strongBrewSem);
380  strongBrewSemHandle = osSemaphoreCreate(osSemaphore(strongBrewSem), 1);
381
382  /* USER CODE BEGIN RTOS_SEMAPHORES */
383
384  // Initialize semaphores count to 0
385  if (powerOnSemHandle != NULL && strongBrewSemHandle != NULL)
386  {
387      osSemaphoreWait(powerOnSemHandle, 0);
388      osSemaphoreWait(strongBrewSemHandle, 0);
389  }
390
391
392  /* add semaphores, ... */
393  /* USER CODE END RTOS_SEMAPHORES */
394
395  /* Create the timer(s) */
396  /* definition and creation of autoOffTimer */
397  osTimerDef(autoOffTimer, pvAutoOffTimerOneShot);
398  autoOffTimerHandle = osTimerCreate(osTimer(autoOffTimer), osTimerOnce, NULL);
399

```

```
408  /* Create the thread(s) */
409  /* definition and creation of defaultTask */
410  osThreadDef(defaultTask, StartDefaultTask, osPriorityAboveNormal, 0, 128);
411  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
412
413  /* definition and creation of myTask02 */
414  osThreadDef(myTask02, powerOnTask, osPriorityNormal, 0, 128);
415  myTask02Handle = osThreadCreate(osThread(myTask02), NULL);
416
417  /* definition and creation of myTask03 */
418  osThreadDef(myTask03, strongBrewTask, osPriorityBelowNormal, 0, 128);
419  myTask03Handle = osThreadCreate(osThread(myTask03), NULL);
420
421  /* definition and creation of myTask04 */
422  osThreadDef(myTask04, brewTravelMug, osPriorityBelowNormal, 0, 128);
423  myTask04Handle = osThreadCreate(osThread(myTask04), NULL);
424
425  /* definition and creation of myTask05 */
426  osThreadDef(myTask05, brewLargeMug, osPriorityBelowNormal, 0, 128);
427  myTask05Handle = osThreadCreate(osThread(myTask05), NULL);
428
429  /* definition and creation of myTask06 */
430  osThreadDef(myTask06, brewSmallMug, osPriorityBelowNormal, 0, 128);
431  myTask06Handle = osThreadCreate(osThread(myTask06), NULL);
432
433  /* definition and creation of myTask07 */
434  osThreadDef(myTask07, brewSmallCup, osPriorityBelowNormal, 0, 128);
435  myTask07Handle = osThreadCreate(osThread(myTask07), NULL);
436
437  /* USER CODE BEGIN RTOS_THREADS */
438  /* add threads, ... */
439  /* USER CODE END RTOS_THREADS */
440
441  /* Start scheduler */
442  osKernelStart();
```

## Step 26. Implement default task to show startup prompt & Start LEDs OFF

```
684 /* USER CODE END Header_StartDefaultTask */
685 void StartDefaultTask(void const * argument)
686 {
687     /* USER CODE BEGIN 5 */
688
689     // Print Startup Prompt
690     startupPrompt();
691     osDelay(100);
692
693     // Start w/ PWR Green LED OFF, StrongBrewLED OFF
694     HAL_GPIO_WritePin(LED_STR_BRW_GPIO_Port, LED_STR_BRW_Pin, GPIO_PIN_RESET);
695     HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin, GPIO_PIN_RESET);
696     osDelay(100);
697     /* Infinite loop */
698     for(;;)
699     {
700         osDelay(1);
701     }
702     /* USER CODE END 5 */
703 }
```

Step 27. Implement Power task to show startup via message & LED indication, as well as display water levels (auto-refill to max at boot)

```
7128 void powerOnTask(void const * argument)
7129 {
7130     /* USER CODE BEGIN powerOnTask */
7131
7132     /* Infinite loop */
7133     for(;;)
7134     {
7135         // Wait for Semaphore indefinitely (till user hits power button)
7136         osSemaphoreWait(powerOnSemHandle, osWaitForever);
7137
7138         // Turn PWR ON
7139         if(pwr_status == FALSE)
7140         {
7141             pwr_status = TRUE;
7142             HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin, GPIO_PIN_SET);
7143             osDelay(100);
7144             powerMessage();
7145             waterLevel = 40;           // refill water every boot
7146             waterLevelMessage(waterLevel);
7147
7148             // Start Auto-Off Timer. Will trigger after 1m
7149             osTimerStart(autoOffTimerHandle, 60000);
7150         }
7151         // Turn OFF
7152         else if(pwr_status == TRUE)
7153         {
7154             // Stop Timer in case its ON
7155             osTimerStop(autoOffTimerHandle);
7156
7157             pwr_status = FALSE;
7158             HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin, GPIO_PIN_RESET);
7159             osDelay(100);
7160             powerMessage();
7161         }
7162         osDelay(300);
7163     }
7164     /* USER CODE END powerOnTask */
```

## Step 28. Strong Brew feature implementation

```
759 void strongBrewTask(void const * argument)
760 {
761     /* USER CODE BEGIN strongBrewTask */
762     /* Infinite loop */
763     for(;;)
764     {
765         // Only update StrongBrew status when Power ON
766         if(pwr_status == TRUE)
767         {
768             // Wait for Semaphore indefinitely (till user hits power button)
769             osSemaphoreWait(strongBrewSemHandle, osWaitForever);
770
771             // Toggle StrongBrew LED Toggle & strongBrewCoffee value
772             HAL_GPIO_TogglePin(LED_STR_BRW_GPIO_Port, LED_STR_BRW_Pin);
773             osDelay(100);
774             strongBrewMessage();
775             strongBrewCoffee = !strongBrewCoffee;
776         }
777         osDelay(300);
778     }
779 }
```



Step 29. Implement the different Brewing options via tasks that utilize Even Group capability for selecting. Auto-off timer is also re-set after every selection

```

789 void brewTravelMug(void const * argument)
790 {
791     /* USER CODE BEGIN brewTravelMug */
792     /* Infinite loop */
793     for(;;)
794     {
795         // Only start brewing when Power ON
796         if(pwr_status == TRUE)
797         {
798             // Wait on ISR. wait on l2ozBits, Clear on exit, don't wait on others, maxDelay
799             xEventGroupWaitBits(xEventGroup, travelMug_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);
800
801             // Check if we have enough water
802             continueBrew = enoughWaterCheck(travelMugSize);
803             if(continueBrew == TRUE)
804             {
805                 // LED ON
806                 HAL_GPIO_WritePin(LED_TRAV_MUG_GPIO_Port, LED_TRAV_MUG_Pin, GPIO_PIN_SET);
807                 osDelay(100);
808
809                 // Brew Strength & Size UART Print & Progress Bar Animation
810                 brewTypeMessage(travelMugSize);
811
812                 // Progress Bar animation
813                 for (int i = 0; i <= travelMugSize; i++)
814                 {
815                     updateProgressBar(i, travelMugSize);
816                     osDelay(50); // Add delay to simulate progress
817                 }
818
819                 // Decrement water level & print Water Level & Temp
820                 osDelay(100);
821                 waterLevel = waterLevel - travelMugSize;
822                 waterLevelMessage(waterLevel);
823
824                 // Reset continueBrew & auto-off Timer
825                 continueBrew = FALSE;
826                 osTimerStart(autoOffTimerHandle, 60000);
827
828                 // LED OFF
829                 HAL_GPIO_WritePin(LED_TRAV_MUG_GPIO_Port, LED_TRAV_MUG_Pin, GPIO_PIN_RESET);
830                 osDelay(100);
831             }
832             osDelay(300);
833         }
834     }
835     /* USER CODE END brewTravelMug */
836 }

```

```

845 void brewLargeMug(void const * argument)
846 {
847     /* USER CODE BEGIN brewLargeMug */
848     /* Infinite loop */
849     for(;;)
850     {
851         // Only start brewing when Power ON
852         if(pwr_status == TRUE)
853         {
854             // Wait on ISR. wait on l2ozBits, Clear on exit, don't wait on others, maxDelay
855             xEventGroupWaitBits(xEventGroup, largeMug_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);
856
857             // Check if we have enough water
858             continueBrew = enoughWaterCheck(largeMugSize);
859             if(continueBrew == TRUE)
860             {
861                 // LED ON
862                 HAL_GPIO_WritePin(LED_LARG_MUG_GPIO_Port, LED_LARG_MUG_Pin, GPIO_PIN_SET);
863                 osDelay(100);
864
865                 // Brew Strength & Size UART Print & Progress Bar Animation
866                 brewTypeMessage(largeMugSize);
867
868                 // Progress Bar animation
869                 for (int i = 0; i <= largeMugSize; i++)
870                 {
871                     updateProgressBar(i, largeMugSize);
872                     osDelay(50); // Add delay to simulate progress
873                 }
874
875                 // Decrement water level & print
876                 osDelay(100);
877                 waterLevel = waterLevel - largeMugSize;
878                 waterLevelMessage(waterLevel);
879
880                 // Reset continueBrew & auto-off Timer
881                 continueBrew = FALSE;
882                 osTimerStart(autoOffTimerHandle, 60000);
883
884                 // LED OFF
885                 HAL_GPIO_WritePin(LED_LARG_MUG_GPIO_Port, LED_LARG_MUG_Pin, GPIO_PIN_RESET);
886                 osDelay(100);
887             }
888             osDelay(300);
889         }
890     }
891     /* USER CODE END brewLargeMug */
892 }

```

```

900  /* USER CODE BEGIN brewSmallMug */
901 void brewSmallMug(void const * argument)
902 {
903     /* USER CODE BEGIN brewSmallMug */
904     /* Infinite loop */
905     for(;;)
906     {
907         // Only start brewing when Power ON
908         if(pwr_status == TRUE)
909         {
910             // Wait on ISR. wait on 12ozBits, Clear on exit, don't wait on others, maxDelay
911             xEventGroupWaitBits(xEventGroup, smallMug_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);
912
913             // Check if we have enough water
914             continueBrew = enoughWaterCheck(smallMugSize);
915             if(continueBrew == TRUE)
916             {
917                 // LED ON
918                 HAL_GPIO_WritePin(LED_SMALL_MUG_GPIO_Port, LED_SMALL_MUG_Pin, GPIO_PIN_SET);
919                 osDelay(100);
920
921                 // Brew Strength & Size UART Print & Progress Bar Animation
922                 brewTypeMessage(smallMugSize);
923
924                 // Progress Bar animation
925                 for (int i = 0; i <= smallMugSize; i++)
926                 {
927                     updateProgressBar(i, smallMugSize);
928                     osDelay(50); // Add delay to simulate progress
929                 }
930
931                 // Decrement water level & print
932                 osDelay(100);
933                 waterLevel = waterLevel - smallMugSize;
934                 waterLevelMessage(waterLevel);
935
936                 // Reset continueBrew & auto-off Timer
937                 continueBrew = FALSE;
938                 osTimerStart(autoOffTimerHandle, 60000);
939
940                 // LED OFF
941                 HAL_GPIO_WritePin(LED_SMALL_MUG_GPIO_Port, LED_SMALL_MUG_Pin, GPIO_PIN_RESET);
942                 osDelay(100);
943             }
944             osDelay(300);
945         }
946     }
947     /* USER CODE END brewSmallMug */
948 }

```

```

957 void brewSmallCup(void const * argument)
958 {
959     /* USER CODE BEGIN brewSmallCup */
960     /* Infinite loop */
961     for(;;)
962     {
963         // Only start brewing when Power ON
964         if(pwr_status == TRUE)
965         {
966             // Wait on ISR. wait on 12ozBits, Clear on exit, don't wait on others, maxDelay
967             xEventGroupWaitBits(xEventGroup, smallCup_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);
968
969             // Check if we have enough water
970             continueBrew = enoughWaterCheck(smallCupSize);
971             if(continueBrew == TRUE)
972             {
973                 // LED ON
974                 HAL_GPIO_WritePin(LED_SMALL_CUP_GPIO_Port, LED_SMALL_CUP_Pin, GPIO_PIN_SET);
975                 osDelay(100);
976
977                 // Brew Strength & Size UART Print & Progress Bar Animation
978                 brewTypeMessage(smallCupSize);
979
980                 // Progress Bar animation
981                 for (int i = 0; i <= smallCupSize; i++)
982                 {
983                     updateProgressBar(i, smallCupSize);
984                     osDelay(50); // Add delay to simulate progress
985                 }
986
987                 // Decrement water level & print
988                 osDelay(100);
989                 waterLevel = waterLevel - smallCupSize;
990                 waterLevelMessage(waterLevel);
991
992                 // Reset continueBrew & auto-off Timer
993                 continueBrew = FALSE;
994                 osTimerStart(autoOffTimerHandle, 60000);
995
996                 // LED OFF
997                 HAL_GPIO_WritePin(LED_SMALL_CUP_GPIO_Port, LED_SMALL_CUP_Pin, GPIO_PIN_RESET);
998                 osDelay(100);
999             }
1000             osDelay(300);
1001         }
1002     }
1003     /* USER CODE END brewSmallCup */
1004 }

```

## Step 30. Implement auto-shutoff callback

```
1002 /* pvAutoOffTimerOneShot function */
1003 void pvAutoOffTimerOneShot(void const * argument)
1004 {
1005     /* USER CODE BEGIN pvAutoOffTimerOneShot */
1006
1007     // Power OFF Coffee Machine
1008     pwr_status = FALSE;
1009
1010     // Green LED OFF
1011     HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin, GPIO_PIN_RESET);
1012     osDelay(100);
1013
1014     // Print Message
1015     autoOffTimerMessage();
1016
1017     /* USER CODE END pvAutoOffTimerOneShot */
```

Step 31. Configure ISR for all the button usages. Power, Strong brews, all different brewing options.

```
56
57 // Add Semaphore & Event Group Access
58 extern osSemaphoreId powerOnSemHandle;
59 extern osSemaphoreId strongBrewSemHandle;
60 extern EventGroupHandle_t xEventGroup;
61
62 /* USER CODE END 0 */
63
64 /* External variables -----
65 extern UART_HandleTypeDef huart1;
66 extern TIM_HandleTypeDef htim1;
67
68 /* USER CODE BEGIN EV */
69
70 /* USER CODE END EV */
```

```
172
173 void EXTI0_IRQHandler(void)
174 {
175     /* USER CODE BEGIN EXTI0_IRQn 0 */
176     /* USER CODE END EXTI0_IRQn 0 */
177     HAL_GPIO_EXTI_IRQHandler(BUTTON_PWR_Pin);
178     /* USER CODE BEGIN EXTI0_IRQn 1 */
179
180     // Release PowerOn Semaphore when interrupt is triggered
181     osSemaphoreRelease(powerOnSemHandle);
182
183     /* USER CODE END EXTI0_IRQn 1 */
184 }
185
```

```

188  /
189 void EXTI1_IRQHandler(void)
190 {
191     /* USER CODE BEGIN EXTI1_IRQn 0 */
192
193
194     /* USER CODE END EXTI1_IRQn 0 */
195     HAL_GPIO_EXTI_IRQHandler(BUTTON_STR_BRW_Pin);
196     /* USER CODE BEGIN EXTI1_IRQn 1 */
197
198     // Release strongBrew Semaphore when triggered
199     osSemaphoreRelease(strongBrewSemHandle);
200
201     /* USER CODE END EXTI1_IRQn 1 */
202 }
203

```



```

207 void EXTI2_IRQHandler(void)
208 {
209     /* USER CODE BEGIN EXTI2_IRQn 0 */
210
211     /* USER CODE END EXTI2_IRQn 0 */
212     HAL_GPIO_EXTI_IRQHandler(BUTTON_SMALL_CUP_Pin);
213     /* USER CODE BEGIN EXTI2_IRQn 1 */
214
215     // 6oz Large Cup Size
216
217     // Declare vars for FreeRTOS interrupt safe-operation
218     BaseType_t xHigherPriorityTaskWoken = pdFALSE;           // init false since bits not set to start
219     BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, smallCup_ISR_BIT, &xHigherPriorityTaskWoken);
220
221     // Check if Event Group Bits were successfully set
222     if (xResult == pdTRUE)
223     {
224         // request context switch when task unblocked
225         portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
226     }
227
228     /* USER CODE END EXTI2_IRQn 1 */
229 }

```

```

234 void EXTI3_IRQHandler(void)
235 {
236     /* USER CODE BEGIN EXTI3_IRQn 0 */
237
238     /* USER CODE END EXTI3_IRQn 0 */
239     HAL_GPIO_EXTI_IRQHandler(BUTTON_LARGE_MUG_Pin);
240     /* USER CODE BEGIN EXTI3_IRQn 1 */
241
242     // 10oz Large Mug Size
243
244     // Declare vars for FreeRTOS interrupt safe-operation
245     BaseType_t xHigherPriorityTaskWoken = pdFALSE;           // init false since bits not set to start
246     BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, largeMug_ISR_BIT, &xHigherPriorityTaskWoken);
247
248     // Check if Event Group Bits were successfully set
249     if (xResult == pdTRUE)
250     {
251         // request context switch when task unblocked
252         portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
253     }
254     /* USER CODE END EXTI3_IRQn 1 */
255 }
256

```

```

260 void EXTI4_IRQHandler(void)
261 {
262     /* USER CODE BEGIN EXTI4_IRQn 0 */
263
264
265     /* USER CODE END EXTI4_IRQn 0 */
266     HAL_GPIO_EXTI_IRQHandler(BUTTON_SMALL_MUG_Pin);
267     /* USER CODE BEGIN EXTI4_IRQn 1 */
268
269     // 8oz Small Mug Size
270
271     // Declare vars for FreeRTOS interrupt safe-operation
272     BaseType_t xHigherPriorityTaskWoken = pdFALSE;           // init false since bits not set to start
273     BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, smallMug_ISR_BIT, &xHigherPriorityTaskWoken);
274
275     // Check if Event Group Bits were successfully set
276     if (xResult == pdTRUE)
277     {
278         // request context switch when task unblocked
279         portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
280     }
281     /* USER CODE END EXTI4_IRQn 1 */
282 }
283

```

```

3178 void EXTI15_10_IRQHandler(void)
318 {
319     /* USER CODE BEGIN EXTI15_10_IRQn 0 */
320
321     /* USER CODE END EXTI15_10_IRQn 0 */
322     HAL_GPIO_EXTI_IRQHandler(BUTTON_TRAV_MUG_Pin);
323     /* USER CODE BEGIN EXTI15_10_IRQn 1 */
324
325     // 6oz Small Cup Size
326
327     // Declare vars for FreeRTOS interrupt safe-operation
328     BaseType_t xHigherPriorityTaskWoken = pdFALSE;           // init false since bits not set to start
329     BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, travelMug_ISR_BIT, &xHigherPriorityTaskWoken);
330
331     // Check if Event Group Bits were successfully set
332     if (xResult == pdTRUE)
333     {
334         // request context switch when task unblocked
335         portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
336     }
337     /* USER CODE END EXTI15_10_IRQn 1 */
338 }

```

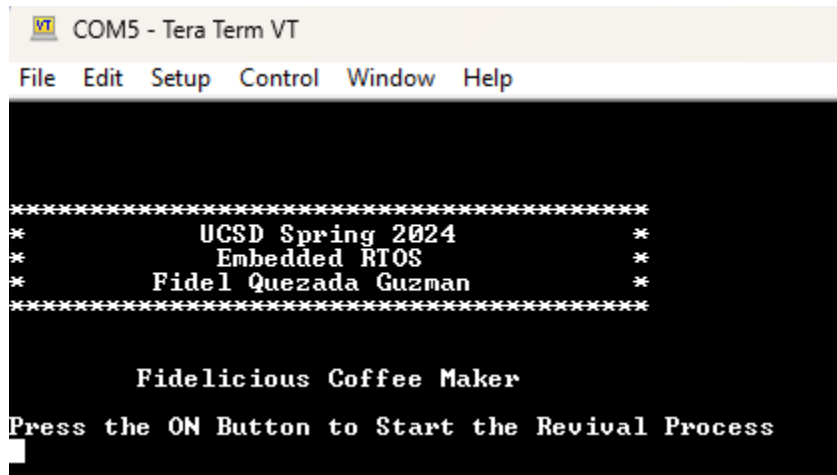
## Step 32. Configure Event Group bits & setBits functionality

```
34
35 // Define Event Group Bits
36 #define travelMug_ISR_BIT (1UL << 0UL)    // bit0
37 #define largeMug_ISR_BIT (1UL << 1UL)     // bit1
38 #define smallMug_ISR_BIT (1UL << 2UL)     // bit2
39 #define smallCup_ISR_BIT (1UL << 3UL)     // bit3
40
41 /* USER CODE END Includes */
42
43 /* Exported types -----
```

```
140
141 /* USER CODE BEGIN Defines */
142
143 // Add config option for xEventGroupSetBitsFromISR usage
144 #define INCLUDE_xTimerPendFunctionCall    1
145
146 /* Section where parameter definitions can be added (for instance, to override default ones in FreeRTOS.h) */
147 /* USER CODE END Defines */
148
149 #endif /* FREERTOS_CONFIG_H */
150
```

## Step 33. Run code!

### Startup prompt



The screenshot shows a Tera Term VT window titled "COM5 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main display area is black with white text. The text is as follows:

```
*****
*          UCSD Spring 2024          *
*          Embedded RTOS             *
*          Fidel Quezada Guzman      *
*****

      Fidelicious Coffee Maker

Press the ON Button to Start the Revival Process
```

Using Power button shall trigger the messages & turn on Power LED. Without this, other tasks wont run

```
*****
*          UCSD Spring 2024          *
*          Embedded RTOS             *
*          Fidel Quezada Guzman      *
*****

          Fidelicious Coffee Maker

Press the ON Button to Start the Revival Process
Coffee Maker ON
Water Level: 40
□
```

Using the StrongBrew button will toggle the LED & messages

```
*****
*          UCSD Spring 2024          *
*          Embedded RTOS             *
*          Fidel Quezada Guzman      *
*****

          Fidelicious Coffee Maker

Press the ON Button to Start the Revival Process
Coffee Maker ON
Water Level: 40
Strong Brew
Normal Brew
□
```

If nothing is done after 1m (limit adjustable) the coffee machine will shut off, turn power LED off & alert user

```
*****
*      UCSD Spring 2024      *
*      Embedded RTOS        *
*      Fidel Quezada Guzman  *
*****

      Fidelicious Coffee Maker

Press the ON Button to Start the Revival Process
Coffee Maker ON
Water Level: 40
Strong Brew
Normal Brew
Auto Turning OFF Coffee Machine
█
```

User can select a size, the program will recognize the option, brew & display the water leftover. LED for the selected brewing option will remain ON until its done!



```

*****
*          UCSD Spring 2024          *
*          Embedded RTOS            *
*          Fidel Quezada Guzman      *
*****

          Fidelicious Coffee Maker

Press the ON Button to Start the Revival Process
Coffee Maker ON
Water Level: 40
Normal Brew: 12oz
[                               1 0 oz
[***                             1 1 oz
[*****                          1 2 oz
[*****                           1 3 oz
[*****                           1 4 oz
[*****                           1 5 oz
[*****                           1 6 oz
[*****                           1 7 oz
[*****                           1 8 oz
[*****                           1 9 oz
[*****                          1 10 oz
[*****                          1 11 oz
[*****                          1 12 oz
Water Level: 28

```

If user selects an option but there is not enough water, water will auto-refill and prompt user to try again

Water Level: 4  
Water Level is too low!!  
Auto-Refilling now...

[	] 0	02
[*	] 1	02
[**	] 2	02
[***	] 3	02
[****	] 4	02
[*****	] 5	02
[*****	] 6	02
[*****	] 7	02
[*****	] 8	02
[*****	] 9	02
[*****	] 10	02
[*****	] 11	02
[*****	] 12	02
[*****	] 13	02
[*****	] 14	02
[*****	] 15	02
[*****	] 16	02
[*****	] 17	02
[*****	] 18	02
[*****	] 19	02
[*****	] 20	02
[*****	] 21	02
[*****	] 22	02
[*****	] 23	02
[*****	] 24	02
[*****	] 25	02
[*****	] 26	02
[*****	] 27	02
[*****	] 28	02
[*****	] 29	02
[*****	] 30	02
[*****	] 31	02
[*****	] 32	02
[*****	] 33	02
[*****	] 34	02
[*****	] 35	02
[*****	] 36	02
[*****	] 37	02
[*****	] 38	02
[*****	] 39	02
[*****	] 40	02

Water Level is full now!  
Please select Coffee Again

User can now try again!

```
normal Brew: 12oz  
***| 0 oz  
*****| 1 oz  
*****| 2 oz  
*****| 3 oz  
*****| 4 oz  
*****| 5 oz  
*****| 6 oz  
*****| 7 oz  
*****| 8 oz  
*****| 9 oz  
*****| 10 oz  
*****| 11 oz  
*****| 12 oz  
Water Level: 28
```

Other brewing options will follow this same path

```

Press the ON Button to Start the Revival Process
Coffee Maker ON
Water Level: 40
Normal Brew: 10oz

                                     1 0 oz
****                               1 1 oz
*****                             1 2 oz
*****                             1 3 oz
*****                             1 4 oz
*****                             1 5 oz
*****                             1 6 oz
*****                             1 7 oz
*****                             1 8 oz
*****                             1 9 oz
*****                             1 10 oz
Water Level: 30

```

```
Press the ON Button to Start the Revival Process
Coffee Maker ON
Water Level: 40
Normal Brew: 8oz
[                               1 0 oz
[*****                        1 1 oz
[*****                        1 2 oz
[*****                        1 3 oz
[*****                        1 4 oz
[*****                        1 5 oz
[*****                        1 6 oz
[*****                        1 7 oz
[*****                        1 8 oz
Water Level: 32
```