UCSD Embedded RTOS

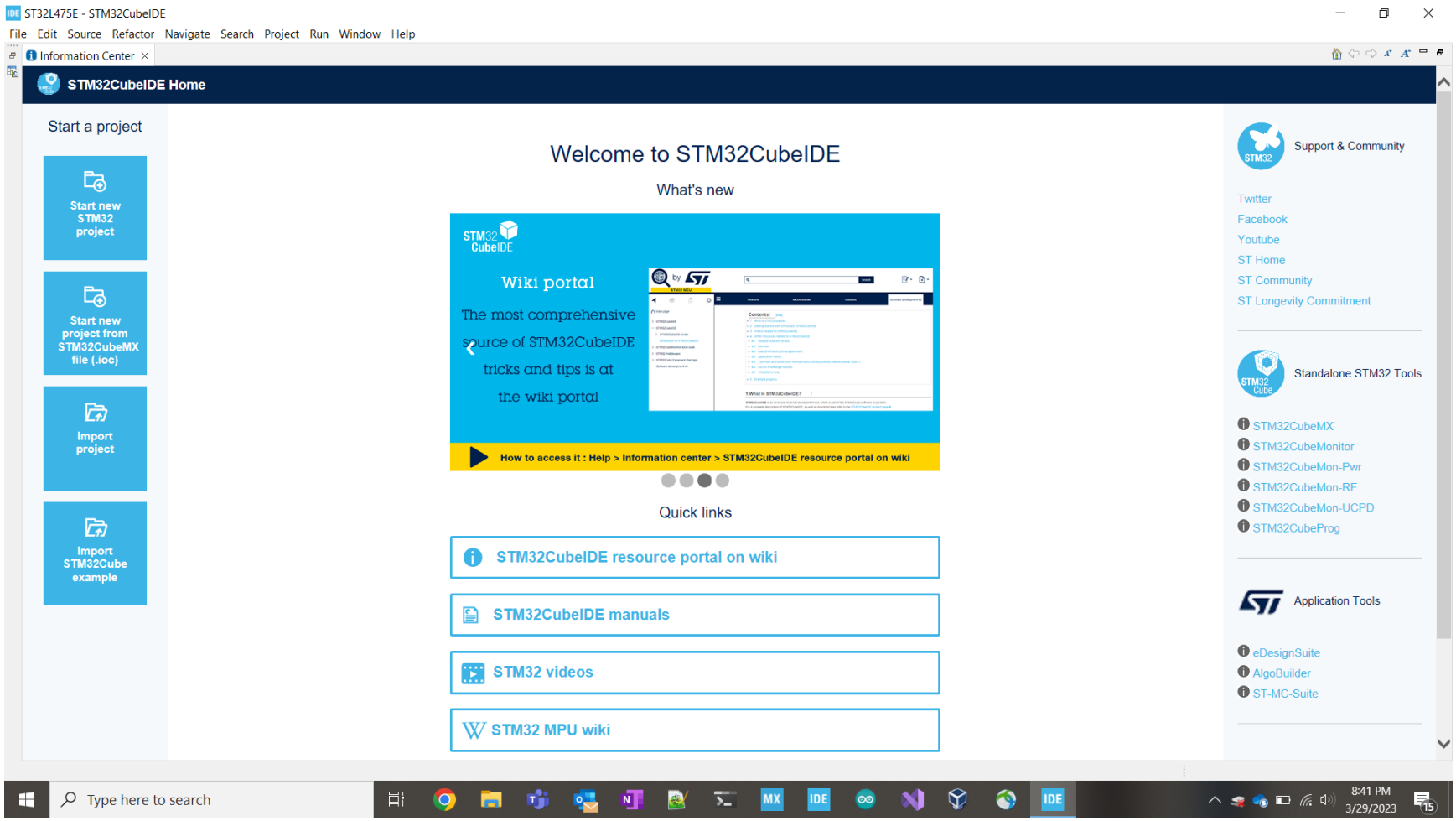
Final Project

By

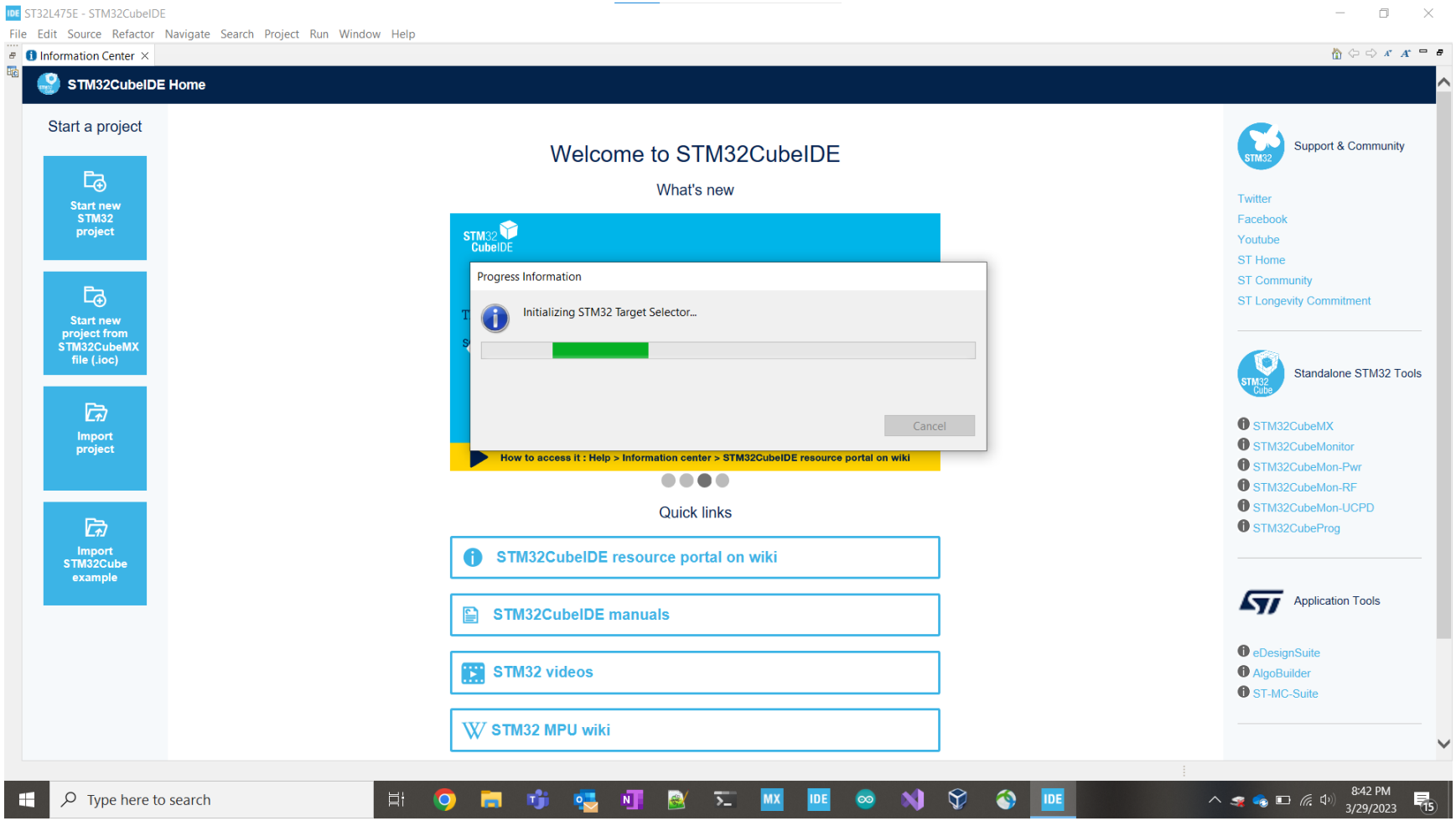
Fidel Quezada Guzman

Fidel2993@gmail.com

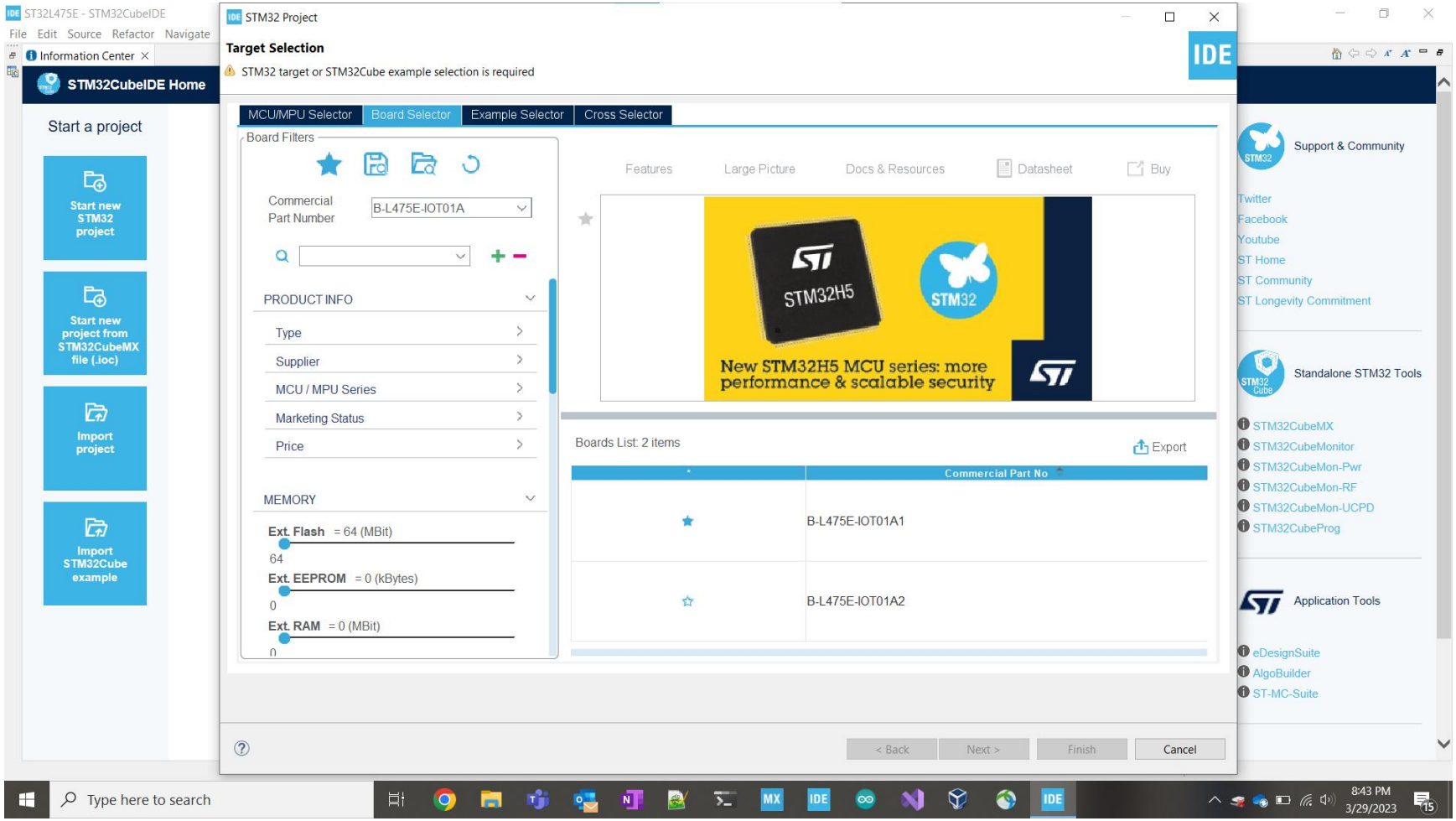
Step 1. Startup STM32CubeMX



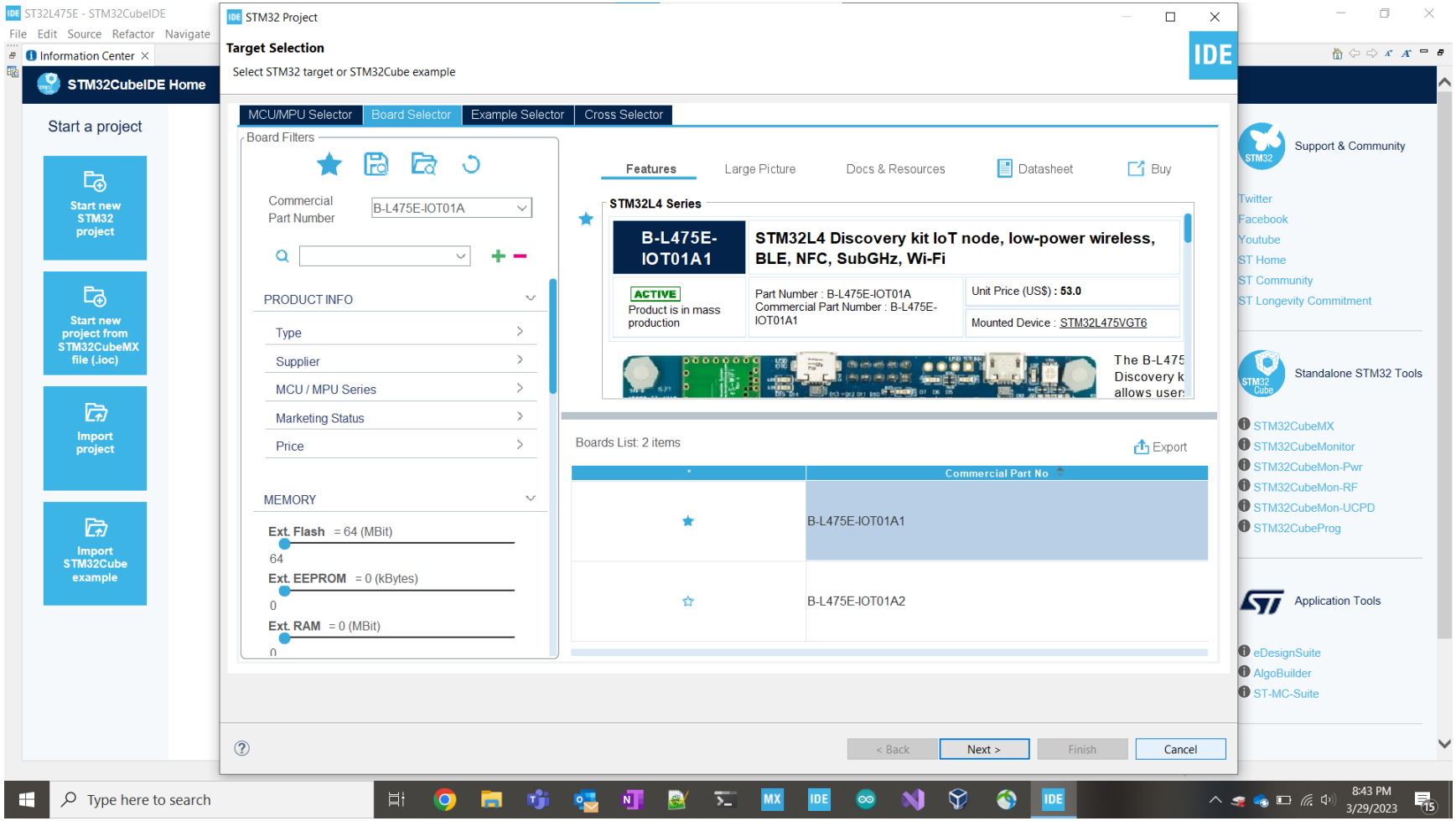
Step 2. Access Board Selector



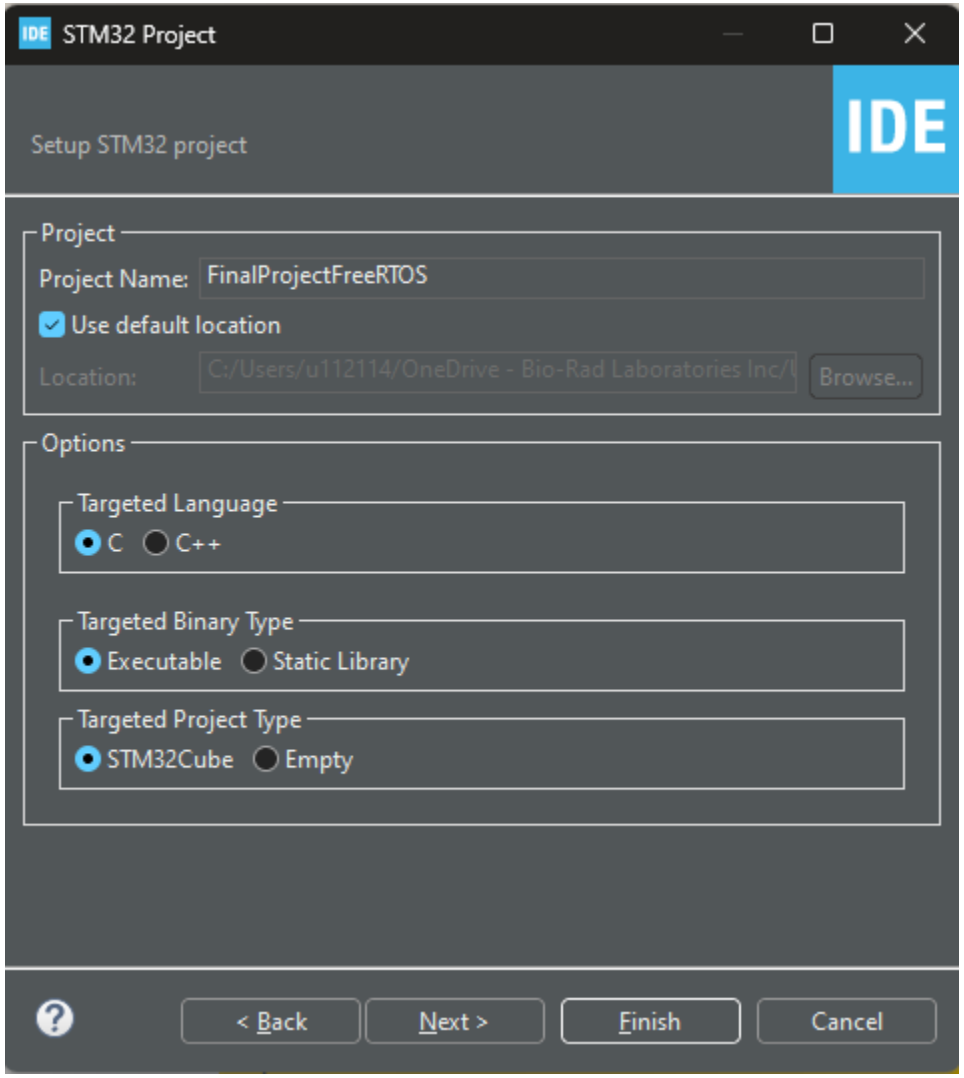
Step 3. Enter “B-L475-IOT01A” Board



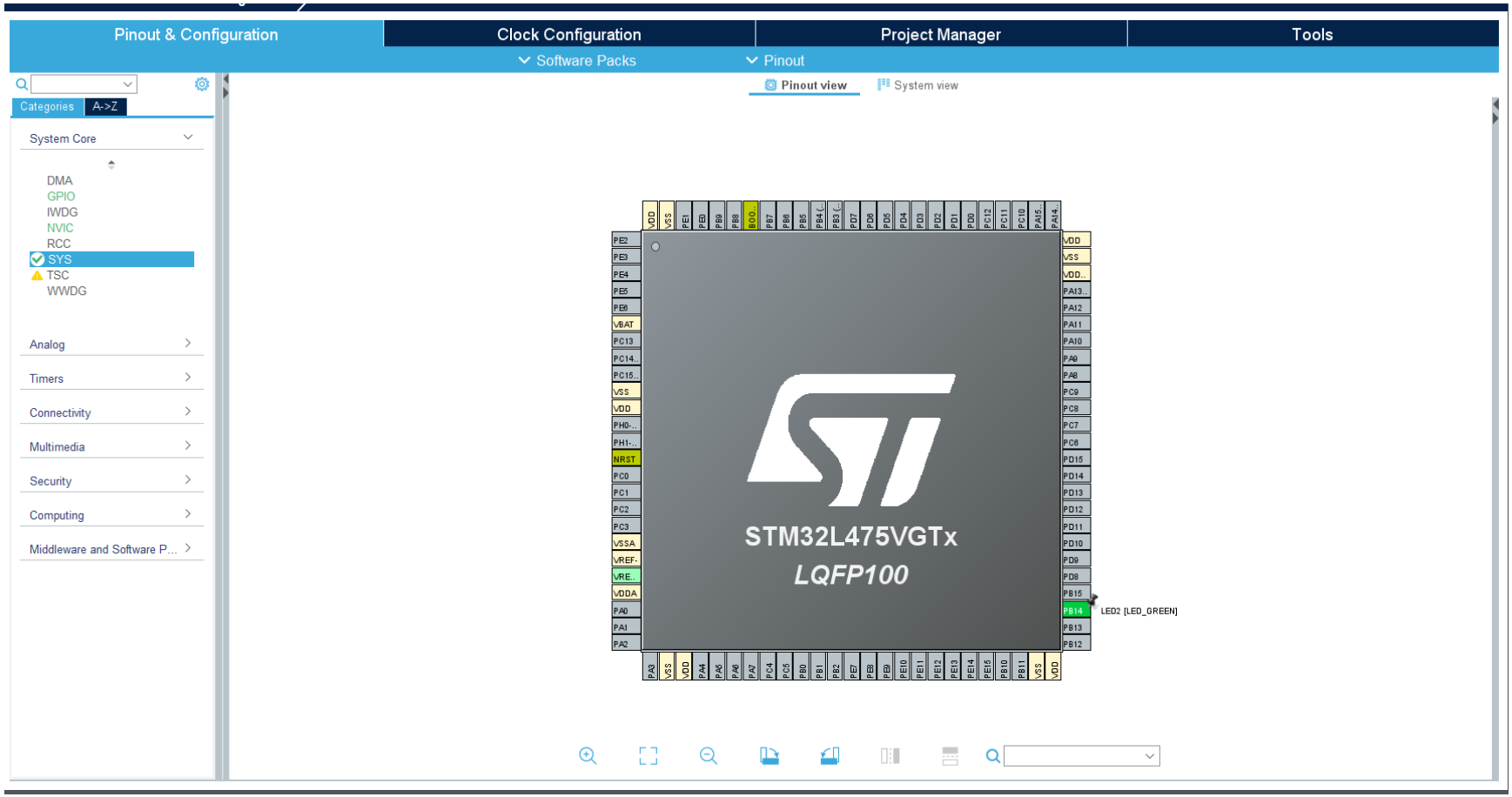
Step 4. Select Board Photo



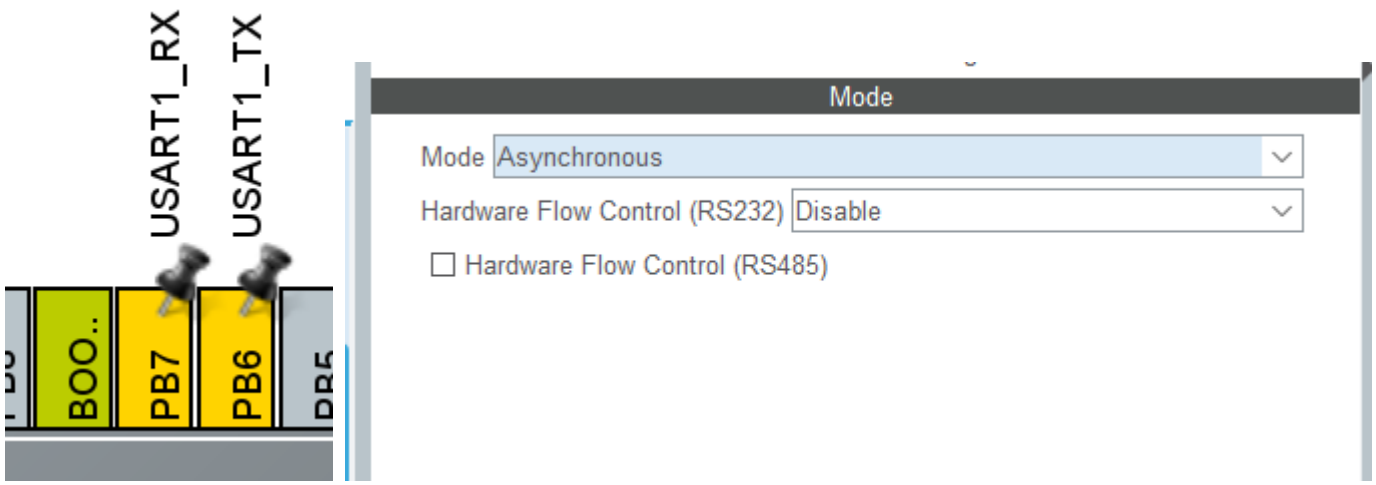
Step 5. Name Project & Select “Finish”

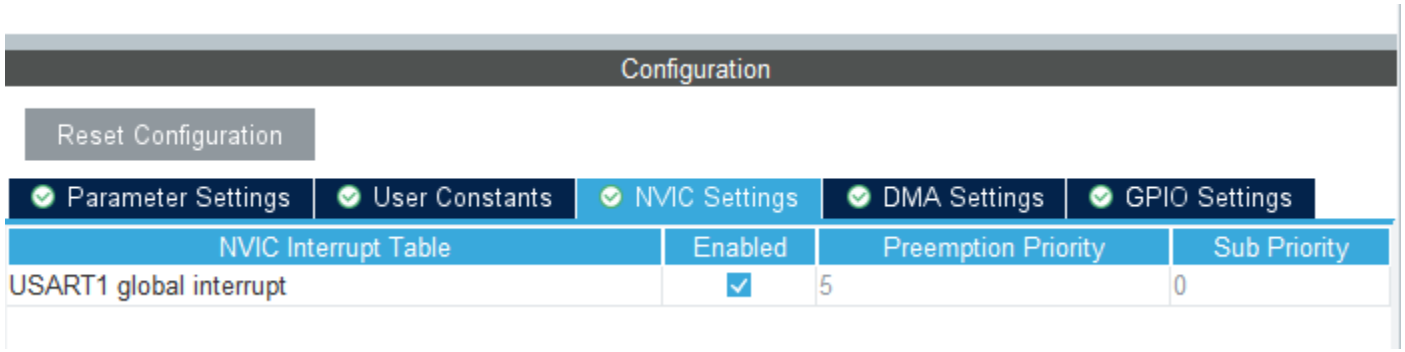


Step 6. Observe Results (Pinout View)



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





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

Pin Name	Signal on Pin	GPIO output l...	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_LARG_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

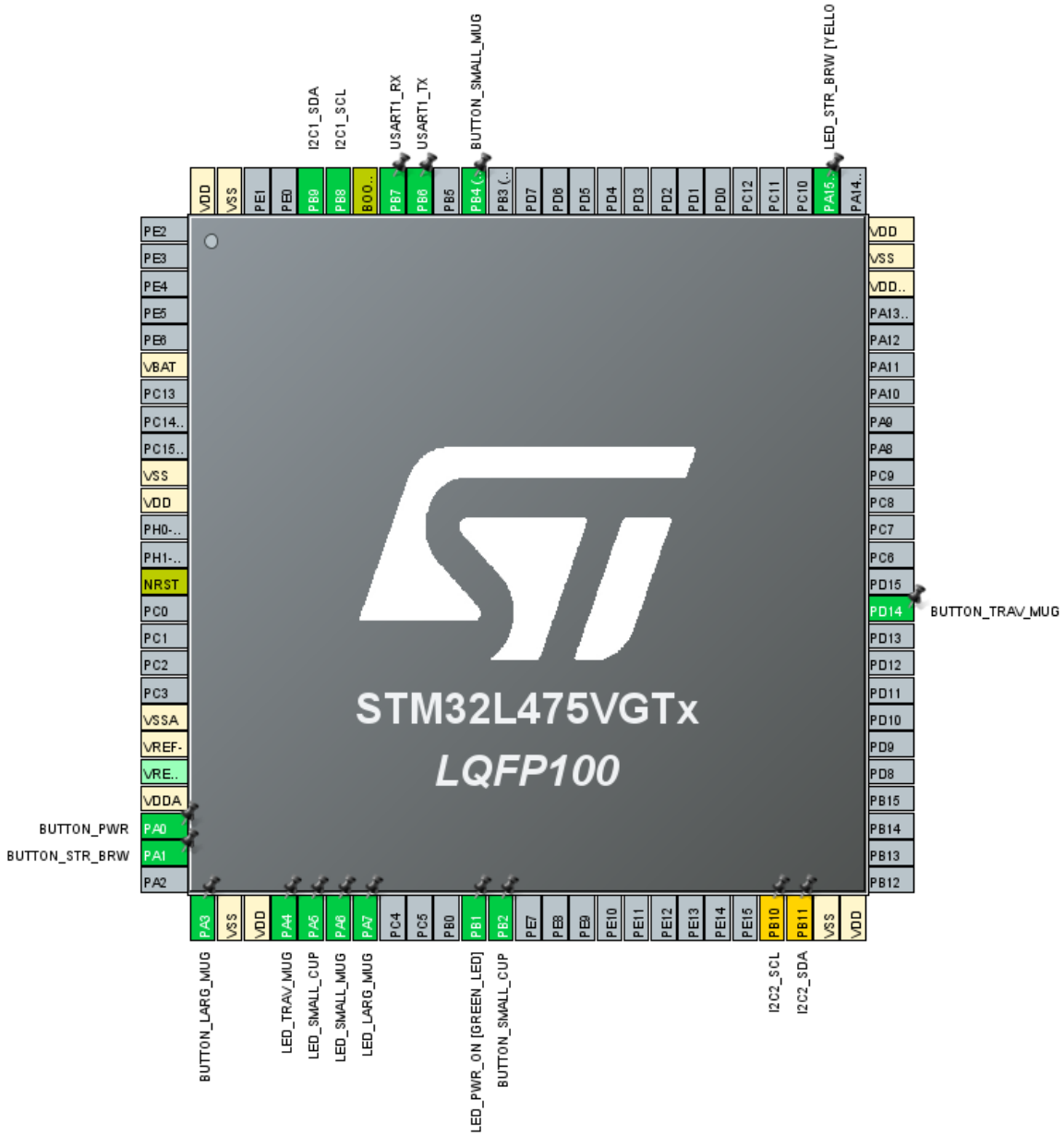
Group By Peripherals

GPIO | Single Mapped Signals | USART | NVIC

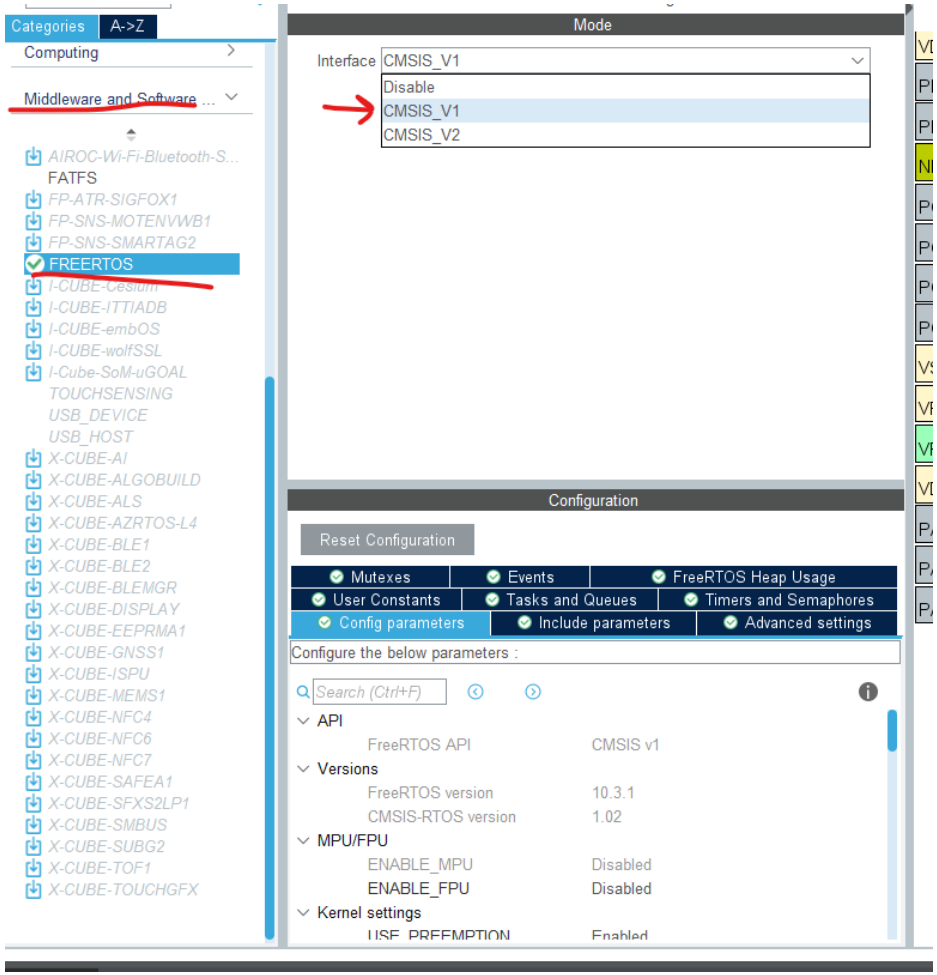
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line0 interrupt	☑	0	0
EXTI line1 interrupt	☑	0	0
EXTI line2 interrupt	☑	0	0
EXTI line3 interrupt	☑	0	0
EXTI line[15:10] interrupts	☑	0	0

Step 9. Enable Output GPIO Pins for LED usage

PA4	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_TRAV_MUG	☑
PA5	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_SMALL_CUP	☑
PA6	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_SMALL_MUG	☑
PA7	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_LARG_MUG	☑
PA15 (JTDI)	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_STR_BRW [YELLOW_LED]	☑
PB1	n/a	Low	Output Push Pull	No pu...	Low	n/a	LED_PWR_ON [GREEN_LED]	☑


STM32L475VGTx
LQFP100
VDD
VSS
PE1
PE0
PB9
PB8
BOO..
PB7
PB6
PB5
PB4 (..
PB3 (..
PD7
PD6
PD5
PD4
PD3
PD2
PD1
PD0
PC12
PC11
PC10
PA15..
PA14..
I2C1_SDA
I2C1_SCL
USART1_RX
USART1_TX
BUTTON_SMALL_MUG
LED_STR_BRW [YELLO
PE2
PE3
PE4
PE5
PE6
VBAT
PC13
PC14..
PC15..
VSS
VDD
PH0-..
PH1-..
NRST
PC0
PC1
PC2
PC3
VSSA
VREF-
VRE..
VDDA
PA0
PA1
PA2
BUTTON_PWR
BUTTON_STR_BRW
VDD
VSS
VDD..
PA13..
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PD15
PD14
PD13
PD12
PD11
PD10
PD9
PD8
PB15
PB14
PB13
PB12
BUTTON_TRAV_MUG
PA3
VSS
VDD
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PE7
PE8
PE9
PE10
PE11
PE12
PE13
PE14
PE15
PB10
PB11
VSS
VDD
BUTTON_LARG_MUG
LED_TRAV_MUG
LED_SMALL_CUP
LED_SMALL_MUG
LED_LARG_MUG
LED_PWR_ON [GREEN_LED]
BUTTON_SMALL_CUP
I2C2_SCL
I2C2_SDA

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

Add | Delete

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

Add Delete

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

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

Config parameters | Include parameters | Advanced settings

Configure the below parameters :

Search (Ctrl+F)

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

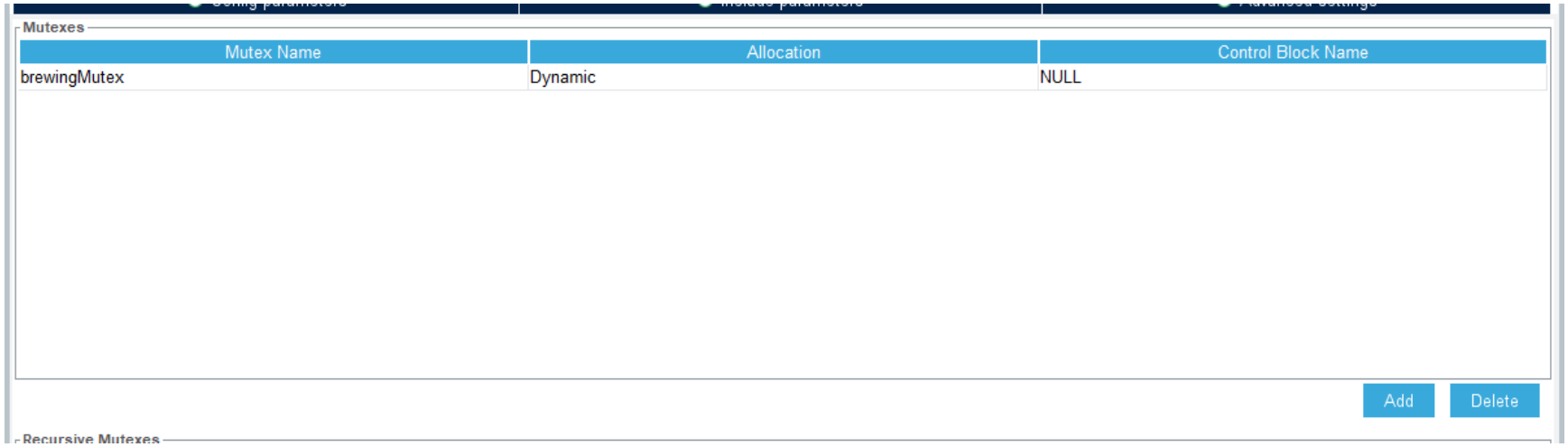
Step 14. Enable a oneShot timer as well for the auto-shut OFF

Timers

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

Add Delete

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



Step 16. Confirm our Heap is good!

Configuration

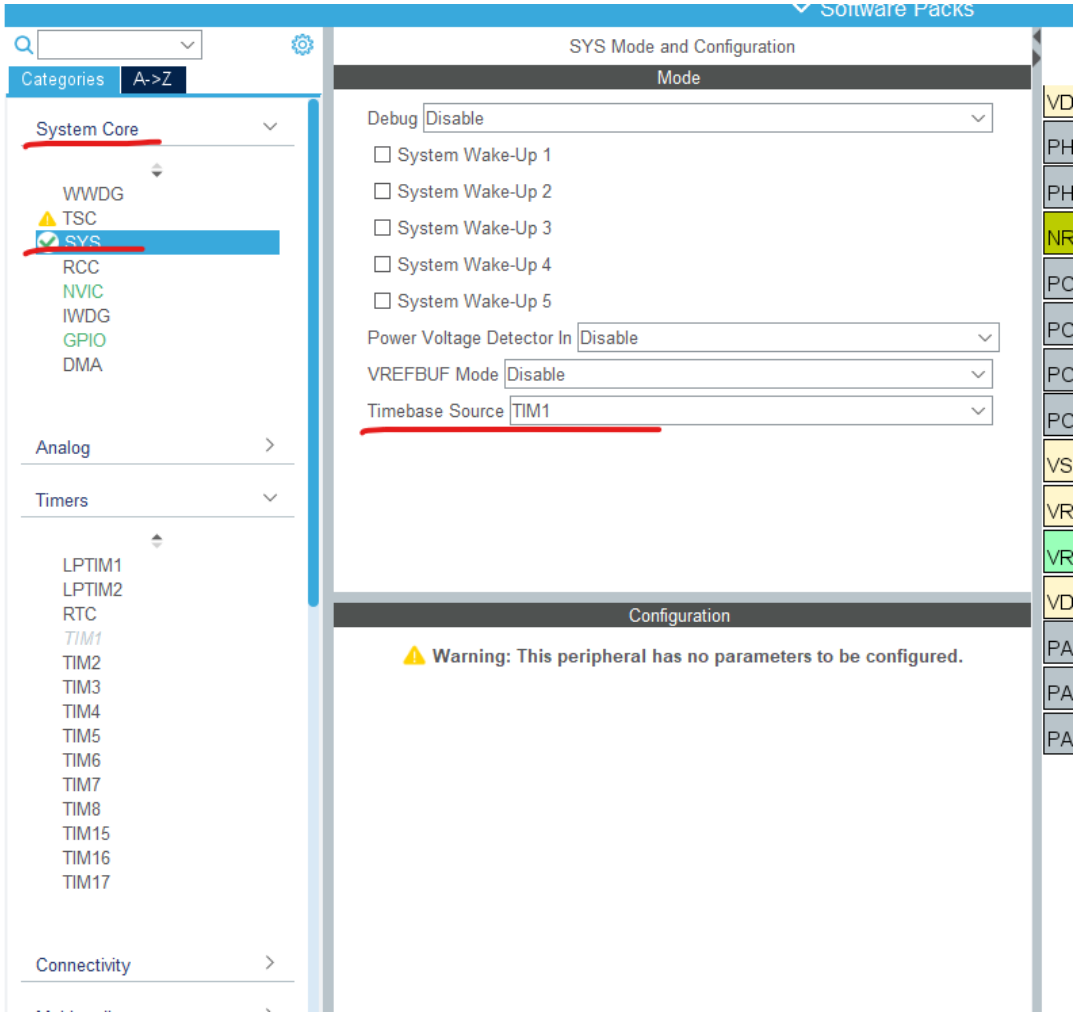
Reset Configuration

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

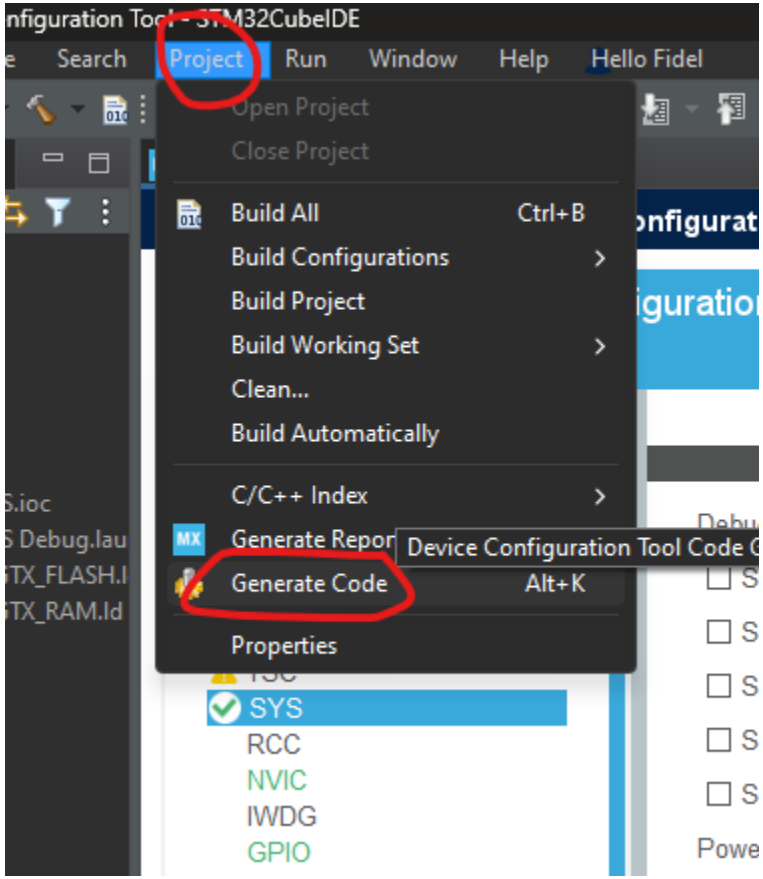
Config parameters | Include parameters | Advanced settings

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 ans 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

```
// Transmit complete callback function
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART1)
    {
        uart_tx_complete = 1; // Transmission complete
    }
}
```

Step 20. Fancy progress bar animation

```c
// Progress Bar Animation
void updateProgressBar(uint8_t progress, uint8_t oz)
{
    // Width constant/ variable
    const uint8_t barWidth = 40;
    uint8_t completedWidth = barWidth * progress / oz;

    char progressBar[barWidth + 1]; // +1 for null terminator
    memset(progressBar, ' ', barWidth);
    progressBar[barWidth] = '\0';

    // Fill Up Progress
    for (uint8_t i = 0; i < completedWidth; ++i)
    {
        progressBar[i] = '*';
    }

    // Adjust size as needed
    char output[50];
    snprintf(output, sizeof(output), "[%s] %d oz\r\n", progressBar, progress);

    // Copy the output to uart_buffer
    snprintf(uart_buffer, sizeof(uart_buffer), "%s", output);

    // Start UART transmission using interrupt-based function
    uart_tx_complete = 0; // Transmission in progress. Callback function will set it to 1 when complete

    // Start non-blocking UART transmission using HAL_UART_Transmit_IT
    if (HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer)) != HAL_OK)
    {
        // Handle transmit error (optional)
        snprintf(uart_buffer, sizeof(uart_buffer), "ERROR TRANSMITTING!\n");
        HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer));
    }
    HAL_Delay(300);
}
```

Step 21. Create function to facilitate printing

```c
// Function to print value based on type
void printIT(void* value, ValueType type)
{
    // Buffer message
    switch (type)
    {
        case CHAR_TYPE:
            snprintf(uart_buffer, sizeof(uart_buffer), "%s", (char*)value);
            break;
        case UINT8_TYPE:
            snprintf(uart_buffer, sizeof(uart_buffer), "%u", *(uint8_t*)value);
            break;
        default:
            snprintf(uart_buffer, sizeof(uart_buffer), "Unknown type\n");
            break;
    }

    // Start UART transmission using interrupt-based function
    uart_tx_complete = 0; // Transmission in progress. Callback function will set it to 1 when complete

    // Start non-blocking UART transmission using HAL_UART_Transmit_IT
    if (HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer)) != HAL_OK)
    {
        // Handle transmit error (optional)
        snprintf(uart_buffer, sizeof(uart_buffer), "ERROR TRANSMITTING!\n");
        HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer));
    }
}
```

Step 22. Create startup & Power messages

```c
// Startup Prompt
void startupPrompt(void)
{
    // Concatenate all lines into the buffer
    snprintf(uart_buffer, sizeof(uart_buffer), "\n\n\n\n*****************************************\n");
    snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "*           UCSD Spring 2024           *\n");
    snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "*            Embedded RTOS             *\n");
    snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "*        Fidel Quezada Guzman          *\n");
    snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "*****************************************\n\n\n");
    snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "\tFidelicious Coffee Maker\n");
    snprintf(uart_buffer + strlen(uart_buffer), sizeof(uart_buffer) - strlen(uart_buffer), "\nPress the ON Button to Start the Revival Process\n");

    // Start UART transmission using interrupt-based function
    uart_tx_complete = 0; // Transmission in progress. Callback function will set it to 1 when complete

    // Start non-blocking UART transmission using HAL_UART_Transmit_IT
    if (HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer)) != HAL_OK)
    {
        // Handle transmit error (optional)
        snprintf(uart_buffer, sizeof(uart_buffer), "ERROR TRANSMITTING!");
        HAL_UART_Transmit_IT(&huart1, (uint8_t *)uart_buffer, strlen(uart_buffer));
    }
}

// Coffee Machine ON/Off Message
void powerMessage()
{
    // print Messages
    char message1[20] = "Coffee Maker ON\n";
    char message2[20] = "Coffee Maker OFF\n";

      if(pwr_status == TRUE)
      {
          printIT(message1, CHAR_TYPE);
      }
      else if(pwr_status == FALSE)
      {
          printIT(message2, CHAR_TYPE);
      }
}
```

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

```
void autoOffTimerMessage()
{
    // print Message
    char message[35] = "Auto Turning OFF Coffee Machine\n";

    printIT(message, CHAR_TYPE);
}

void strongBrewMessage()
{
    // print Messages
    char message1[20] = "Strong Brew\n";
    char message2[20] = "Normal Brew\n";

      if(strongBrewCoffee == FALSE)
      {
          printIT(message1, CHAR_TYPE);
      }
      else if(strongBrewCoffee == TRUE)
      {
          printIT(message2, CHAR_TYPE);
      }
}

void brewTypeMessage(uint8_t* sizeBrew)
{
    // print options
    char strong[15] = "Strong Brew: ";
    char normal[15] = "Normal Brew: ";

    if (strongBrewCoffee == TRUE)
    {
        printIT(strong, CHAR_TYPE);
        printIT((void*)&sizeBrew, UINT8_TYPE);
        printIT("oz\n", CHAR_TYPE);
    }
    else
    {
        printIT(normal, CHAR_TYPE);
        printIT((void*)&sizeBrew, UINT8_TYPE);
        printIT("oz\n", CHAR_TYPE);
    }
}
```

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

```
void waterLevelMessage(uint8_t* waterLevel)
{
    // print options
    char message[15] = "Water Level: ";
    char newLine[3] = "\n";

    // print
    printIT(message, CHAR_TYPE);
    printIT((void*)&waterLevel, UINT8_TYPE);
    printIT(newLine, CHAR_TYPE);
}

// Check if we have enough water
bool enoughWaterCheck(uint8_t coffeeSize)
{
    char message1[50] = "Water Level is too low!!\nAuto-Refilling now...\n";
    char message2[55] = "Water Level is full now!\nPlease select Coffee Again\n";

    // Check if we will have enough water after CoffeeSize selection
    if(waterLevel < coffeeSize)
    {
        // Print Message to user
        printIT(message1, CHAR_TYPE);

        // Auto-Refill Water w/ animation
        for (int i = 0; i <= 40; i++)
        {
          updateProgressBar(i, 40);
        }
        waterLevel = 40;

        // Print Message to user
        printIT(message2, CHAR_TYPE);

        return FALSE;   // wont proceed w/ brew
    }
    else
    {
        return TRUE;    // proceed w/ brew
    }
}
```

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();
```

```
/* Create the mutex(es) */
/* definition and creation of brewingMutex */
osMutexDef(brewingMutex);
brewingMutexHandle = osMutexCreate(osMutex(brewingMutex));

/* USER CODE BEGIN RTOS_MUTEX */
/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* Create the semaphores(s) */
/* definition and creation of powerOnSem */
osSemaphoreDef(powerOnSem);
powerOnSemHandle = osSemaphoreCreate(osSemaphore(powerOnSem), 1);

/* definition and creation of strongBrewSem */
osSemaphoreDef(strongBrewSem);
strongBrewSemHandle = osSemaphoreCreate(osSemaphore(strongBrewSem), 1);

/* USER CODE BEGIN RTOS_SEMAPHORES */

// Initialize semaphores count to 0
if (powerOnSemHandle != NULL && strongBrewSemHandle != NULL)
{
    osSemaphoreWait(powerOnSemHandle, 0);
    osSemaphoreWait(strongBrewSemHandle, 0);
}


/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* Create the timer(s) */
/* definition and creation of autoOffTimer */
osTimerDef(autoOffTimer, pvAutoOffTimerOneShot);
autoOffTimerHandle = osTimerCreate(osTimer(autoOffTimer), osTimerOnce, NULL);
```

```
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

```
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */

    // Print Startup Prompt
    startupPrompt();
    osDelay(100);

    // Start w/ PWR Green LED OFF, StrongBrewLED OFF
    HAL_GPIO_WritePin(LED_STR_BRW_GPIO_Port, LED_STR_BRW_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin,GPIO_PIN_RESET);
    osDelay(100);
  /* Infinite loop */
  for(;;)
  {
      osDelay(1);
  }
  /* USER CODE END 5 */
}
```

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

```
void powerOnTask(void const * argument)
{
  /* USER CODE BEGIN powerOnTask */

  /* Infinite loop */
  for(;;)
  {
      // Wait for Semaphore indefinitely (till user hits power button)
      osSemaphoreWait(powerOnSemHandle, osWaitForever);

      // Turn PWR ON
      if(pwr_status == FALSE)
      {
          pwr_status = TRUE;
          HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin, GPIO_PIN_SET);
          osDelay(100);
          powerMessage();
          waterLevel = 40;          // refill water every boot
          waterLevelMessage(waterLevel);


          // Start Auto-Off Timer. Will trigger after 1m
          osTimerStart(autoOffTimerHandle, 60000);
      }
      // Turn OFF
      else if(pwr_status == TRUE)
      {
          // Stop Timer in case its ON
          osTimerStop(autoOffTimerHandle);

          pwr_status = FALSE;
          HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin, GPIO_PIN_RESET);
          osDelay(100);
          powerMessage();
      }
      osDelay(300);
  }
  /* USER CODE END powerOnTask */
```

Step 28. Strong Brew feature implementation

```c
void strongBrewTask(void const * argument)
{
  /* USER CODE BEGIN strongBrewTask */
  /* Infinite loop */
  for(;;)
  {
      // Only update StrongBrew status when Power ON
      if(pwr_status == TRUE)
      {
          // Wait for Semaphore indefinitely (till user hits power button)
          osSemaphoreWait(strongBrewSemHandle, osWaitForever);

          // Toggle StrongBrew LED Toggle & strongBrewCoffee value
          HAL_GPIO_TogglePin(LED_STR_BRW_GPIO_Port, LED_STR_BRW_Pin);
          osDelay(100);
          strongBrewMessage();
          strongBrewCoffee = !strongBrewCoffee;
      }
      osDelay(300);
  }
```

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

```c
void brewTravelMug(void const * argument)
{
  /* USER CODE BEGIN brewTravelMug */
  /* Infinite loop */
  for(;;)
  {
      // Only start brewing when Power ON
      if(pwr_status == TRUE)
      {
          // Wait on ISR. wait on 12ozBits, Clear on exit, don't wait on others, maxDelay
          xEventGroupWaitBits(xEventGroup, travelMug_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);

          // Check if we have enough water
          continueBrew = enoughWaterCheck(travelMugSize);
          if(continueBrew == TRUE)
          {
              // LED ON
              HAL_GPIO_WritePin(LED_TRAV_MUG_GPIO_Port, LED_TRAV_MUG_Pin, GPIO_PIN_SET);
              osDelay(100);

              // Brew Strength & Size UART Print & Progress Bar Animation
              brewTypeMessage(travelMugSize);

              // Progress Bar animation
              for (int i = 0; i <= travelMugSize; i++)
              {
                  updateProgressBar(i, travelMugSize);
                  osDelay(50);  // Add delay to simulate progress
              }

              // Decrement water level & print Water Level & Temp
              osDelay(100);
              waterLevel = waterLevel - travelMugSize;
              waterLevelMessage(waterLevel);

              // Reset continueBrew & auto-off Timer
              continueBrew = FALSE;
              osTimerStart(autoOffTimerHandle, 60000);

              // LED OFF
              HAL_GPIO_WritePin(LED_TRAV_MUG_GPIO_Port, LED_TRAV_MUG_Pin, GPIO_PIN_RESET);
              osDelay(100);
          }
          osDelay(300);
      }
  }
  /* USER CODE END brewTravelMug */
}
```

```c
void brewLargeMug(void const * argument)
{
  /* USER CODE BEGIN brewLargeMug */
  /* Infinite loop */
  for(;;)
  {
      // Only start brewing when Power ON
      if(pwr_status == TRUE)
      {
          // Wait on ISR. wait on 12ozBits, Clear on exit, don't wait on others, maxDelay
          xEventGroupWaitBits(xEventGroup, largeMug_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);

          // Check if we have enough water
          continueBrew = enoughWaterCheck(largeMugSize);
          if(continueBrew == TRUE)
          {
              // LED ON
              HAL_GPIO_WritePin(LED_LARG_MUG_GPIO_Port, LED_LARG_MUG_Pin, GPIO_PIN_SET);
              osDelay(100);

              // Brew Strength & Size UART Print & Progress Bar Animation
              brewTypeMessage(largeMugSize);

              // Progress Bar animation
              for (int i = 0; i <= largeMugSize; i++)
              {
                  updateProgressBar(i, largeMugSize);
                  osDelay(50);  // Add delay to simulate progress
              }

              // Decrement water level & print
              osDelay(100);
              waterLevel = waterLevel - largeMugSize;
              waterLevelMessage(waterLevel);

              // Reset continueBrew & auto-off Timer
              continueBrew = FALSE;
              osTimerStart(autoOffTimerHandle, 60000);

              // LED OFF
              HAL_GPIO_WritePin(LED_LARG_MUG_GPIO_Port, LED_LARG_MUG_Pin, GPIO_PIN_RESET);
              osDelay(100);
          }
          osDelay(300);
      }
  }
  /* USER CODE END brewLargeMug */
}
```

```c
void brewSmallMug(void const * argument)
{
  /* USER CODE BEGIN brewSmallMug */
  /* Infinite loop */
  for(;;)
  {
      // Only start brewing when Power ON
      if(pwr_status == TRUE)
      {
          // Wait on ISR. wait on 12ozBits, Clear on exit, don't wait on others, maxDelay
          xEventGroupWaitBits(xEventGroup, smallMug_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);

          // Check if we have enough water
          continueBrew = enoughWaterCheck(smallMugSize);
          if(continueBrew == TRUE)
          {
              // LED ON
              HAL_GPIO_WritePin(LED_SMALL_MUG_GPIO_Port, LED_SMALL_MUG_Pin, GPIO_PIN_SET);
              osDelay(100);

              // Brew Strength & Size UART Print & Progress Bar Animation
              brewTypeMessage(smallMugSize);

              // Progress Bar animation
              for (int i = 0; i <= smallMugSize; i++)
              {
                  updateProgressBar(i, smallMugSize);
                  osDelay(50);  // Add delay to simulate progress
              }

              // Decrement water level & print
              osDelay(100);
              waterLevel = waterLevel - smallMugSize;
              waterLevelMessage(waterLevel);

              // Reset continueBrew & auto-off Timer
              continueBrew = FALSE;
              osTimerStart(autoOffTimerHandle, 60000);

              // LED OFF
              HAL_GPIO_WritePin(LED_SMALL_MUG_GPIO_Port, LED_SMALL_MUG_Pin, GPIO_PIN_RESET);
              osDelay(100);
          }
          osDelay(300);
      }
  }
  /* USER CODE END brewSmallMug */
}
```

```c
/* USER CODE END Header_brewSmallCup */
void brewSmallCup(void const * argument)
{
  /* USER CODE BEGIN brewSmallCup */
  /* Infinite loop */
  for(;;)
  {
      // Only start brewing when Power ON
      if(pwr_status == TRUE)
      {
          // Wait on ISR. wait on 12ozBits, Clear on exit, don't wait on others, maxDelay
          xEventGroupWaitBits(xEventGroup, smallCup_ISR_BIT, pdTRUE, pdFALSE, portMAX_DELAY);

          // Check if we have enough water
          continueBrew = enoughWaterCheck(smallCupSize);
          if(continueBrew == TRUE)
          {
              // LED ON
              HAL_GPIO_WritePin(LED_SMALL_CUP_GPIO_Port, LED_SMALL_CUP_Pin, GPIO_PIN_SET);
              osDelay(100);

              // Brew Strength & Size UART Print & Progress Bar Animation
              brewTypeMessage(smallCupSize);

              // Progress Bar animation
              for (int i = 0; i <= smallCupSize; i++)
              {
                  updateProgressBar(i, smallCupSize);
                  osDelay(50);  // Add delay to simulate progress
              }

              // Decrement water level & print
              osDelay(100);
              waterLevel = waterLevel - smallCupSize;
              waterLevelMessage(waterLevel);

              // Reset continueBrew & auto-off Timer
              continueBrew = FALSE;
              osTimerStart(autoOffTimerHandle, 60000);

              // LED OFF
              HAL_GPIO_WritePin(LED_SMALL_CUP_GPIO_Port, LED_SMALL_CUP_Pin, GPIO_PIN_RESET);
              osDelay(100);
          }
          osDelay(300);
      }
  }
  /* USER CODE END brewSmallCup */
}
```

Step 30. Implement auto-shutoff callback

```
/* pvAutoOffTimerOneShot function */
void pvAutoOffTimerOneShot(void const * argument)
{
  /* USER CODE BEGIN pvAutoOffTimerOneShot */

    // Power OFF Coffee Machine
    pwr_status = FALSE;

    // Green LED OFF
    HAL_GPIO_WritePin(LED_PWR_ON_GPIO_Port, LED_PWR_ON_Pin,GPIO_PIN_RESET);
    osDelay(100);

    // Print Message
    autoOffTimerMessage();

  /* USER CODE END pvAutoOffTimerOneShot */
```

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

```c
// Add Semaphore & Event Group Access
extern osSemaphoreId powerOnSemHandle;
extern osSemaphoreId strongBrewSemHandle;
extern EventGroupHandle_t xEventGroup;

/* USER CODE END 0 */

/* External variables ---------------------------------
extern UART_HandleTypeDef huart1;
extern TIM_HandleTypeDef htim1;

/* USER CODE BEGIN EV */
```

```c
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(BUTTON_PWR_Pin);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

    // Release PowerOn Semaphore when interrupt is triggered
    osSemaphoreRelease(powerOnSemHandle);

  /* USER CODE END EXTI0_IRQn 1 */
}
```

```c
void EXTI1_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI1_IRQn 0 */


  /* USER CODE END EXTI1_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(BUTTON_STR_BRW_Pin);
  /* USER CODE BEGIN EXTI1_IRQn 1 */

  // Release strongBrew Semaphore when triggered
  osSemaphoreRelease(strongBrewSemHandle);

  /* USER CODE END EXTI1_IRQn 1 */
}
```

```c
void EXTI2_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI2_IRQn 0 */

  /* USER CODE END EXTI2_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(BUTTON_SMALL_CUP_Pin);
  /* USER CODE BEGIN EXTI2_IRQn 1 */

  // 6oz Large  Cup Size

  // Declare vars for FreeRTOS interrupt safe-operation
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;                // init false since bits not set to start
  BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, smallCup_ISR_BIT, &xHigherPriorityTaskWoken);

  // Check if Event Group Bits were successfully set
  if (xResult == pdTRUE)
  {
      // request context switch when task unblocked
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }

  /* USER CODE END EXTI2_IRQn 1 */
}
```

```c
void EXTI3_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI3_IRQn 0 */

  /* USER CODE END EXTI3_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(BUTTON_LARG_MUG_Pin);
  /* USER CODE BEGIN EXTI3_IRQn 1 */

  // 10oz Large  Mug Size

  // Declare vars for FreeRTOS interrupt safe-operation
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;                // init false since bits not set to start
  BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, largeMug_ISR_BIT, &xHigherPriorityTaskWoken);

  // Check if Event Group Bits were successfully set
  if (xResult == pdTRUE)
  {
      // request context switch when task unblocked
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
  /* USER CODE END EXTI3_IRQn 1 */
}
```

```c
void EXTI4_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI4_IRQn 0 */


  /* USER CODE END EXTI4_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(BUTTON_SMALL_MUG_Pin);
  /* USER CODE BEGIN EXTI4_IRQn 1 */

  // 8oz Small Mug Size

  // Declare vars for FreeRTOS interrupt safe-operation
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;                // init false since bits not set to start
  BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, smallMug_ISR_BIT, &xHigherPriorityTaskWoken);

  // Check if Event Group Bits were successfully set
  if (xResult == pdTRUE)
  {
      // request context switch when task unblocked
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
  /* USER CODE END EXTI4_IRQn 1 */
}
```

```c
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(BUTTON_TRAV_MUG_Pin);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  // 6oz Small Cup Size

  // Declare vars for FreeRTOS interrupt safe-operation
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;                // init false since bits not set to start
  BaseType_t xResult = xEventGroupSetBitsFromISR(xEventGroup, travelMug_ISR_BIT, &xHigherPriorityTaskWoken);

  // Check if Event Group Bits were successfully set
  if (xResult == pdTRUE)
  {
      // request context switch when task unblocked
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

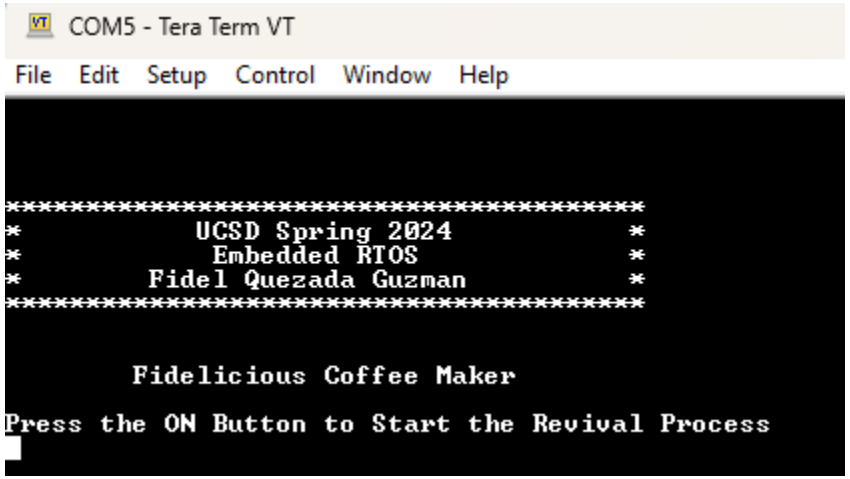
Step 32. Configure Event Group bits & setBits functionality

```
// Define Event Group Bits
#define travelMug_ISR_BIT (1UL << 0UL)      // bit0
#define largeMug_ISR_BIT (1UL << 1UL)       // bit1
#define smallMug_ISR_BIT (1UL << 2UL)       // bit2
#define smallCup_ISR_BIT (1UL << 3UL)       // bit3

/* USER CODE END Includes */

/* Exported types ------------------------------------------
```

```
/* USER CODE BEGIN Defines */

  // Add config option for xEventGroupSetBitsFromISR usage
#define INCLUDE_xTimerPendFunctionCall          1

/* Section where parameter definitions can be added (for instance, to override default ones in FreeRTOS.h) */
/* USER CODE END Defines */

#endif /* FREERTOS_CONFIG_H */
```

Step 33. Run code!

Startup prompt



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
[                                      ] 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
```

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 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
[*************                           ] 13 oz
[**************                          ] 14 oz
[***************                         ] 15 oz
[****************                        ] 16 oz
[*****************                       ] 17 oz
[******************                      ] 18 oz
[*******************                     ] 19 oz
[********************                    ] 20 oz
[*********************                   ] 21 oz
[**********************                  ] 22 oz
[***********************                 ] 23 oz
[************************                ] 24 oz
[*************************               ] 25 oz
[**************************              ] 26 oz
[***************************             ] 27 oz
[****************************            ] 28 oz
[*****************************           ] 29 oz
[******************************          ] 30 oz
[*******************************         ] 31 oz
[********************************        ] 32 oz
[*********************************       ] 33 oz
[**********************************      ] 34 oz
[***********************************     ] 35 oz
[************************************    ] 36 oz
[*************************************   ] 37 oz
[**************************************  ] 38 oz
[*************************************** ] 39 oz
[****************************************] 40 oz
Water Level is full now!
Please select Coffee Again
```

User can now try again!

```
ormal 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
ater Level: 28
```

Other brewing options will follow this same path

```
ress the ON Button to Start the Revival Process
offee Maker ON
ater Level: 40
ormal Brew: 10oz
                                        ] 0 oz
****                                    ] 1 oz
********                                ] 2 oz
************                            ] 3 oz
****************                        ] 4 oz
********************                    ] 5 oz
************************                ] 6 oz
****************************            ] 7 oz
********************************        ] 8 oz
************************************    ] 9 oz
****************************************] 10 oz
ater Level: 30
```

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