UCSD Embedded RTOS

Course Number: ECE-40290
Section ID: 146369

Final Project

Date: 5/26/2020

Chris Isabelle
christopher.j.isabelle@gmail.com
SID: U01136665

Summary:

- Coffee maker controller using an STM IoT discovery board.
- User interface is a VT100 ANSI escape sequence console for output and a joystick for user input.
- The joystick is sampled by and an ADC.
- A Nucleo-32 is used as a coffee simulator. This simulator outputs DC voltages (using an on-board DAC) that correspond to simulated coffee level (or volume) and temperature.
- The user selects a temperature and a volume using the joystick.
- In brew mode:
- The controller reads coffee level and temperature (from the simulator) and outputs real-time measurements to the VT100 terminal.
- The controller controls level or volume by disabling Brew mode GPIO to the simulator.
- The controller controls coffee temperature by enabling and disabling the simulator heater function keeping the coffee within 2 degC of the user selected temperature.

Link to 5 minute video of project overview and demo:

https://drive.google.com/open?id=14_quspUf1-eXi1L9w1KWlhGDyQ9vCscJ

RTOS Components:

Tasks:

prvTaskConsoleOutput

Waits for a user defined data element to become available on a **queue**. If **queue** is not empty:

- Pulls data element from the **queue**.
- Formats the output char array.
- Output the char array to a VT100 compatible terminal.

prvTaskProcessUserInput

Check **event group** for setup brew.

- A **mutex** controls access to ADC1.
- Read ADC to acquire Joystick input.

prvTaskProcessTemperature

Check **event group** for OK to start brew. If start brew is enabled:

- A **mutex** controls access to ADC1.
- Reads ADC to acquire coffee temperature.
- Sends output message to pvTaskConsoleOutput by placing a user defined data element on a **queue**.
- Enables coffee warmer if coffee temperature below set point.
- Disables coffee warmer if coffee temperature above the set point.

prvTaskProcessLevel

Check **event group** for OK to start brew. If start brew is enabled:

- A **mutex** controls access to ADC1.
- Reads ADC to acquire coffee level.
- Sends output message to pvTaskConsoleOutput by placing a user defined data element on a **queue**.
- Enables brew mode if coffee level is below set point.
- Disables brew mode if coffee level above the set point.

Semaphore:

User button clicked event sent from **ISR** to user input **task**.

Mutex:

Provides ADC1 access control.

When in Brew mode both the Temperature and the Brew volume tasks need to read ADC1.

Timers:

pvClickEventOneShotTimer

When the User button is depressed this one shot timer will light turn the Blue LED off after 500msec.

This is used to toggle off the Blue LED, enabled by the User Button ISR.

pvBrewEnabledPeriodicTimer

When in brew mode this timer will add put a Brew time message to the console.

Queue:

Queue that holds output user defined data for VT100 console output.

Interrupt Handler Routine:

Handler for User Button Click

Send semaphore for User Button Clicked.

Enabled Blue LED (LD4)

Sets one-shot timer the shut off LED after 500msec.

Event group:

Used to determine is all brew conditions have been met prior to starting up Brew related tasks.

Critical Section:

Provides resource management for GPIO access to the Blue LED (LD4).

This is required because both the interrupt and an RTOS timer handler access this resource.

I/O Interconnect Table

Function	Simulator MCU Port	Simulator-pin	IoT Board-pin	IoT Board MCU Port	Jumper Color
.3 VDC			CN2-4	3V3	WHT
5.0 VDC	VIN	CN4-1	CN2-5	5V	WHT
GND	GND	CN4-2	CN2-6	GND	BLK
Simulator ON	PB3 (GPIO Output)	LD3 (Green LED)			
Enable Warmer	PA3 (GPIO Input)	CN4-10	CN4-2	PC4 (GPIO Output)	RED
Enable Brew	PA4 (GPIO Input)	CN4-9	CN4-1	PC5 (GPIO Output)	BRN
Coffee Level	PA5 (DAC1 CH2)	CN4-8	CN4-5	PC1 (ADC1 CH2)	BLU
Coffee Temperature	PA6 (DAC2 CH1)	CN4-7	CN4-4	PC2 (ADC1 CH3)	GRN
Joy Stick VRY			CN4-6	PC0 (ADC1 CH1)	PUR
User Select			B2 (Blue User Button)	PC13 (EXTI13)	
User Ack			LD4 (Blue LED)	PC9	

Code Overview

Macros

```
//ANSI Escape Sequqnces for VT100 terminal emulation
#define RED_ON_BLK  "31;40"
#define WHT_ON_BLK  "37;40"
#define YEL_ON_BLK  "33;40"
#define GRN_ON_BLK  "32;40"
#define BLU_ON_BLK  "34;40"

#define BLK_ON_RED  "30;41"
#define BLK_ON_WHT  "30;47"
#define BLK_ON_YEL  "30;43"
#define BLK_ON_GRN  "30;42"

#define NORMAL WHT_ON_BLK

#define EVENT_GROUP_TEMPERATURE_SET (1<<0)
#define EVENT_GROUP_VOLUME_SET (1<<1)
#define EVENT_GROUP_BREW_ENABLED (1<<2)

#define BLUE_LED_Pin GPIO_PIN_9
#define BLUE_LED_GPIO_Port GPIOC
#define SET_BLUE_LED(enable) HAL_GPIO_WritePin(BLUE_LED_GPIO_Port, BLUE_LED_Pin, !(enable))

#define SET_WARM_MODE(enable) HAL_GPIO_WritePin(ENABLE_HEATER_GPIO_Port, ENABLE_HEATER_Pin, !enable)
#define SET_BREW_MODE(enable) HAL_GPIO_WritePin(ENABLE_BREW_GPIO_Port, ENABLE_BREW_Pin, !enable)

#define STATE_INIT             0
#define STATE_SET_MODE         1
#define STATE_SET_VOLUME       2
#define STATE_SET_TEMPERATURE  3
#define STATE_BREW             4

/* USER CODE END PD */
```

Utility function to format queue messages.

```
uint32_t outputMsgCounter = 0;
static void updateUserInterface(int y, int x, char * clr, char * txt)
{
  queueCfg_t queueMsg;
  queueMsg.yPosition=y;
  queueMsg.xPosition=x;
  queueMsg.textColor=clr;
  queueMsg.textStr=txt;
  xQueueSendToBack(xMsgQueue, &queueMsg, portMAX_DELAY);
  //this delay intentionally slow queue TX processing
  //20msec is a 50Hz update rate
  vTaskDelay(pdMS_TO_TICKS(20));
  outputMsgCounter++;
}
```

One shot timer function that turns off LED .5 second after is was turned on by the ISR.

```
static void prvUserInputFlashTimerOneShot(TimerHandle_t xTimer)
{
  //turn LED OFF
  //since this GPIO is also controlled by an interrupt handler
  //need to designate this as a critical section for GPIO resourse management.
  taskENTER_CRITICAL();
  SET_BLUE_LED(0);
  taskEXIT_CRITICAL();
  (void)xTimer;
}
```

Periodic timer that increments and output a brew time in seconds.

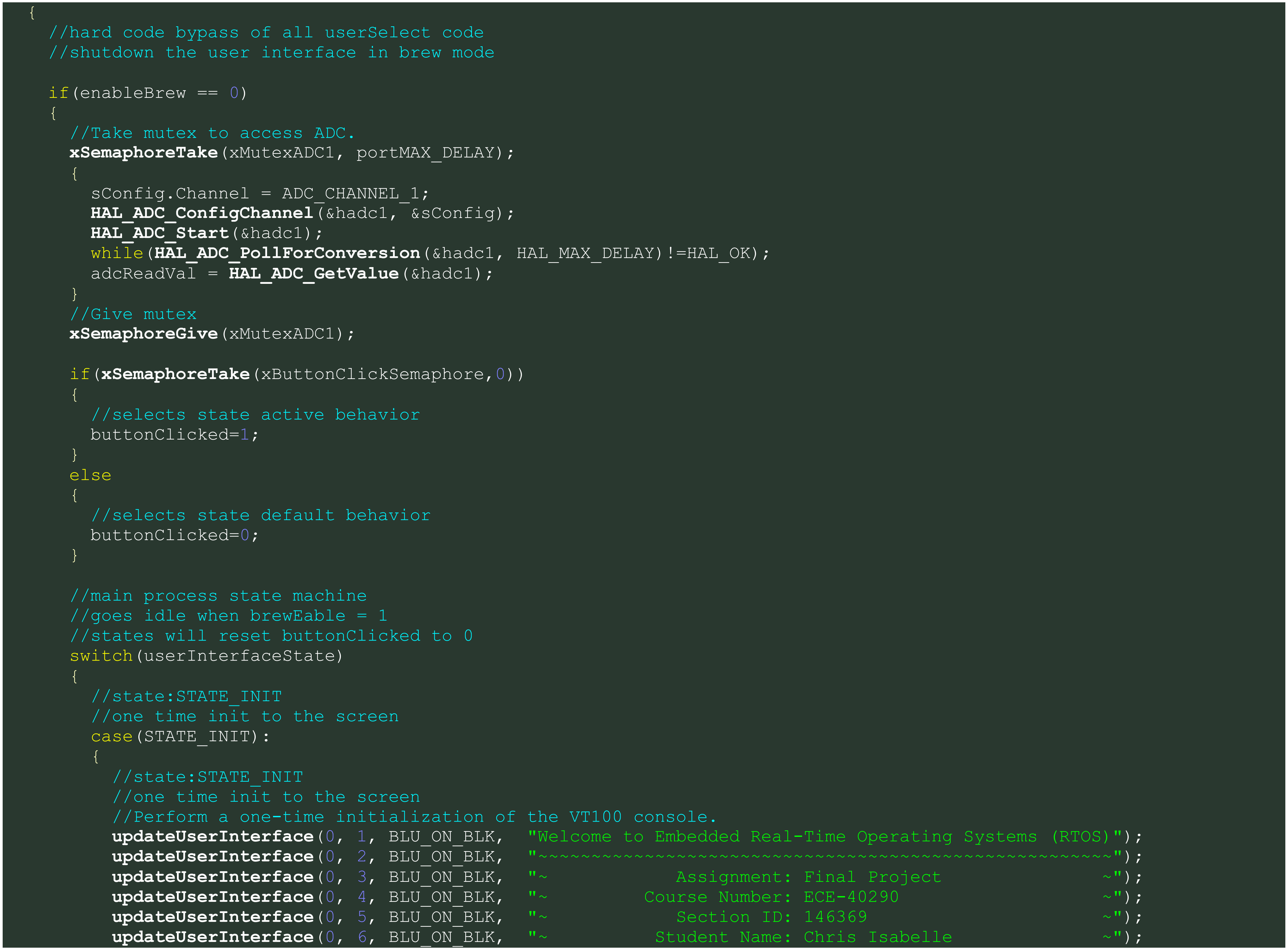
```
uint32_t uptime=0;
static void prvBrewTimerAutoReload(TimerHandle_t xTimer)
{
  char buf[80];
  sprintf(buf, "%li secs", uptime);
  updateUserInterface(25, 27, YEL_ON_BLK,  buf);
  uptime++;
  (void)xTimer;
}
```

Task that manages the output to the UART.

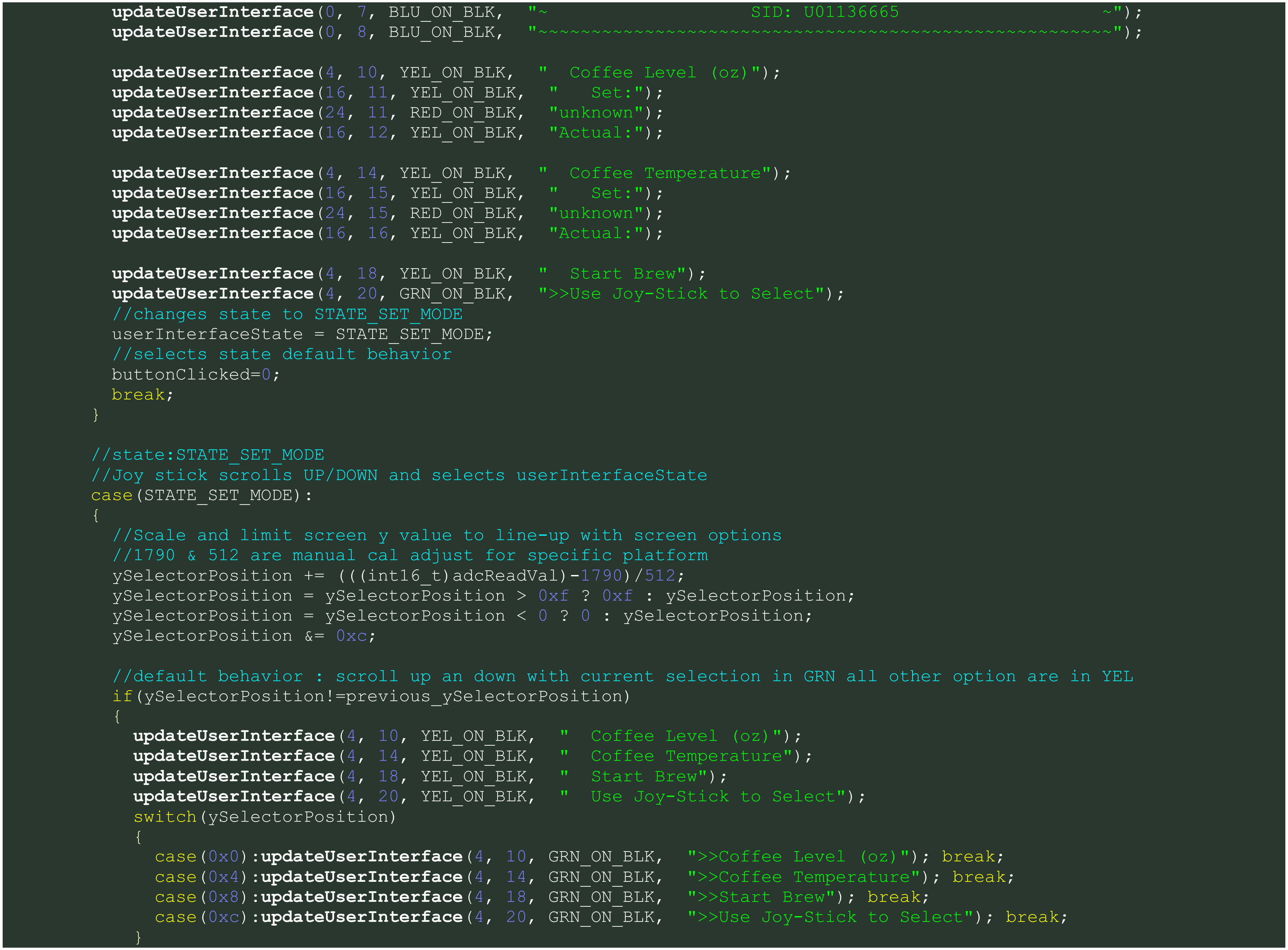
```
static void prvTaskConsoleOutput(void* pvParameters)
{
  char buf[80];
  queueCfg_t queueMsg;

  //init output
  snprintf(buf, sizeof(buf), "\033[2J");
  HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);

  for (;; )
  {
    //Wait for a user defined data element to become available on a queue.  If queue is not empty:
    uxQueueMessagesWaiting(xMsgQueue);
    while (uxQueueMessagesWaiting(xMsgQueue))
    {
      //Pull data element from the queue.
      xQueueReceive(xMsgQueue, &queueMsg, portMAX_DELAY);
      //Format the output char array
      sprintf(buf, "\033[%d;%dH\033[%sm%s", queueMsg.xPosition, queueMsg.yPosition, queueMsg.textColor, 
queueMsg.textStr);
      HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
      sprintf(buf, "\033[1D");
      HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
    }
  }
}
```

Task that manages user input. This is the primary state machine for the application

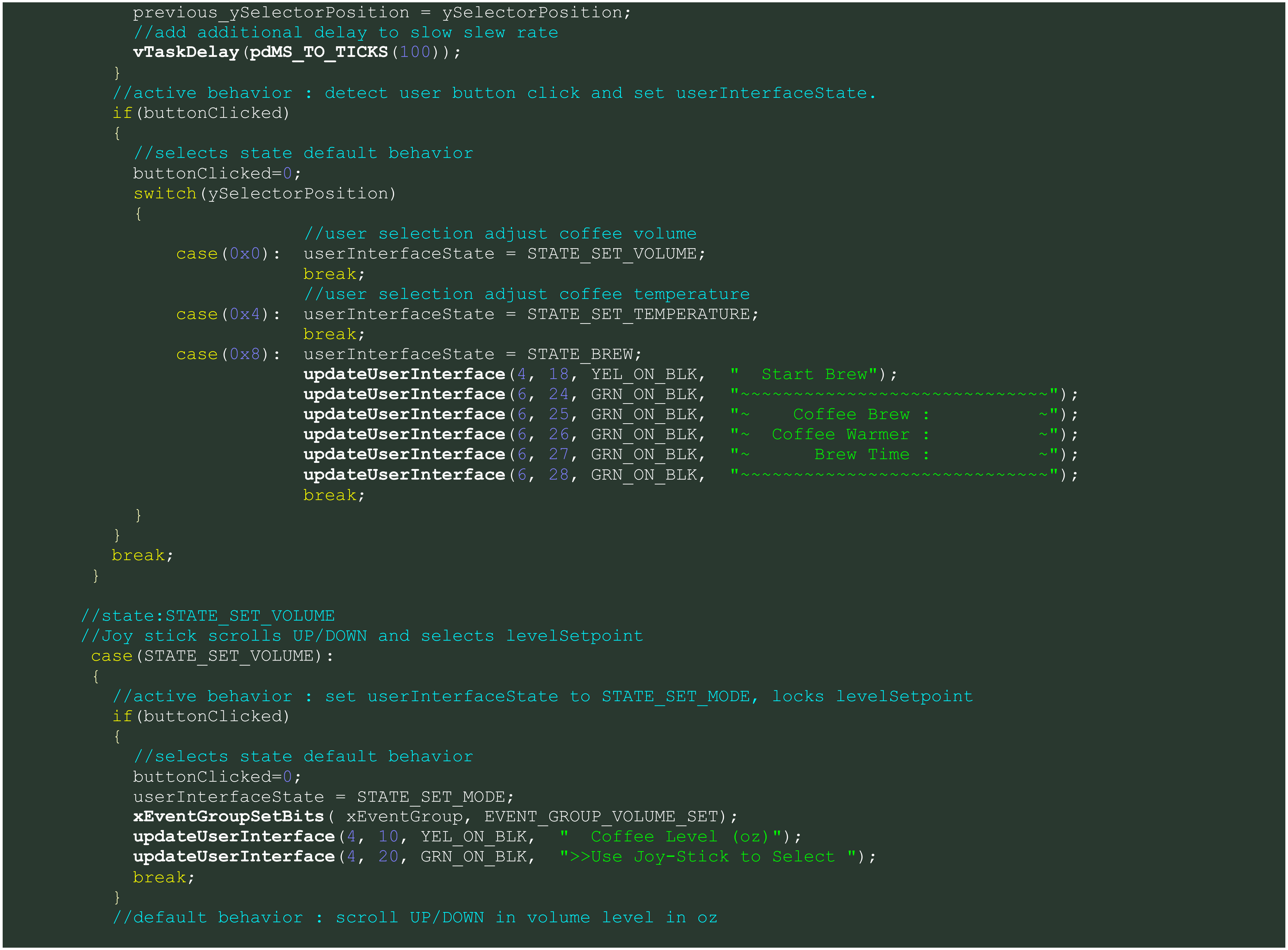
```
static void prvTaskProcessUserInput(void* pvParameters)
{
  uint8_t enableBrew = 0;
  uint8_t ySelectorPosition;
  uint8_t previous_ySelectorPosition;
  uint32_t adcReadVal;
  uint8_t buttonClicked = 0;

  char buf[80];

  //init state mechine to STATE_INIT
  uint8_t userInterfaceState = STATE_INIT;

  for (;;)
```



```
{
  //hard code bypass of all userSelect code
  //shutdown the user interface in brew mode

  if(enableBrew == 0)
  {
    //Take mutex to access ADC.
    xSemaphoreTake(xMutexADC1, portMAX_DELAY);
    {
      sConfig.Channel = ADC_CHANNEL_1;
      HAL_ADC_ConfigChannel(&hadc1, &sConfig);
      HAL_ADC_Start(&hadc1);
      while(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)!=HAL_OK);
      adcReadVal = HAL_ADC_GetValue(&hadc1);
    }
    //Give mutex
    xSemaphoreGive(xMutexADC1);

    if(xSemaphoreTake(xButtonClickSemaphore,0))
    {
      //selects state active behavior
      buttonClicked=1;
    }
    else
    {
      //selects state default behavior
      buttonClicked=0;
    }

    //main process state machine
    //goes idle when brewEable = 1
    //states will reset buttonClicked to 0
    switch(userInterfaceState)
    {
      //state:STATE_INIT
      //one time init to the screen
      case(STATE_INIT):
      {
        //state:STATE_INIT
        //one time init to the screen
        //Perform a one-time initialization of the VT100 console.
        updateUserInterface(0, 1, BLU_ON_BLK,  "Welcome to Embedded Real-Time Operating Systems (RTOS)");
        updateUserInterface(0, 2, BLU_ON_BLK,  "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        updateUserInterface(0, 3, BLU_ON_BLK,  "~            Assignment: Final Project               ~");
        updateUserInterface(0, 4, BLU_ON_BLK,  "~         Course Number: ECE-40290                   ~");
        updateUserInterface(0, 5, BLU_ON_BLK,  "~            Section ID: 146369                      ~");
        updateUserInterface(0, 6, BLU_ON_BLK,  "~          Student Name: Chris Isabelle              ~");
```

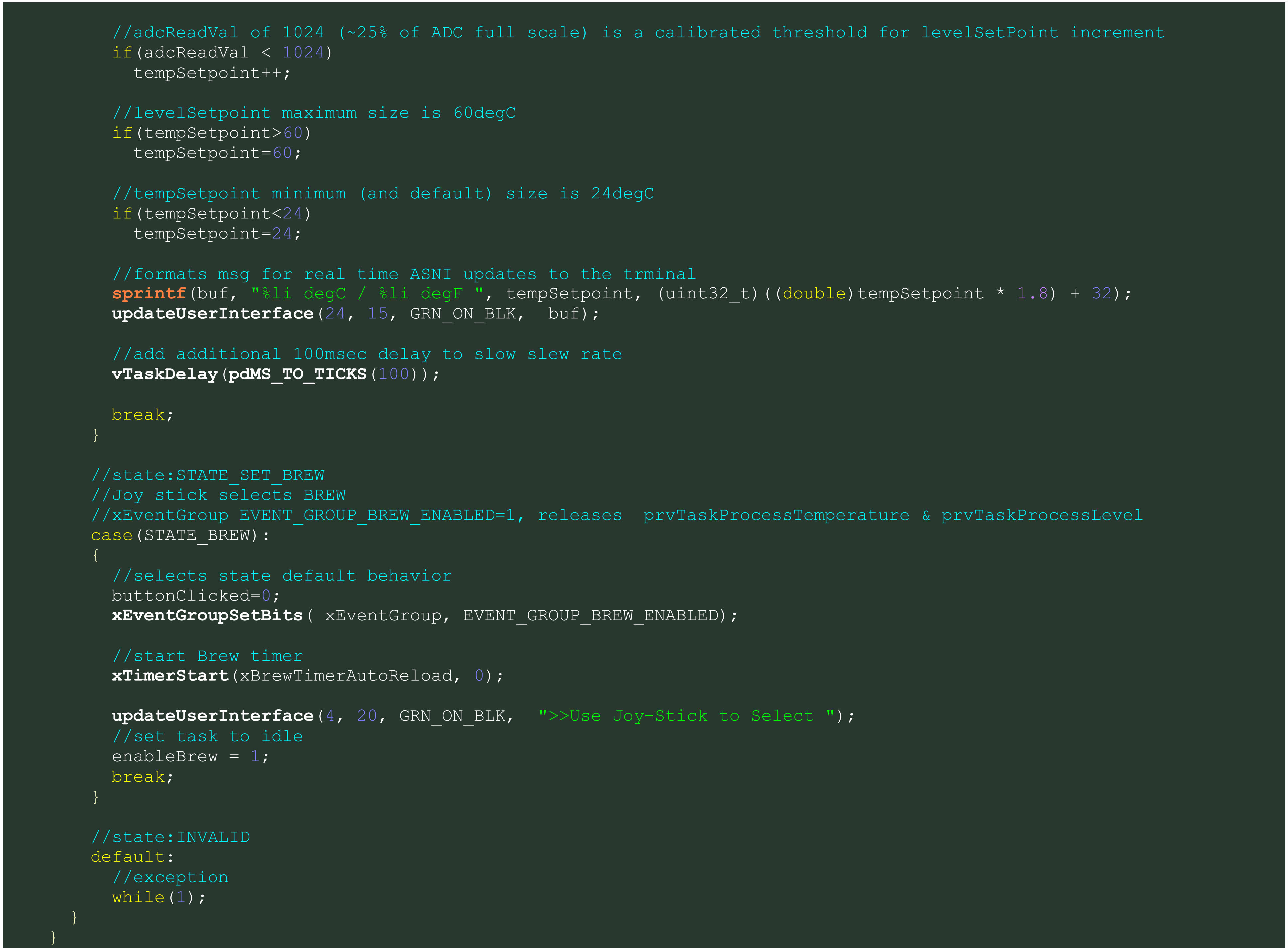


```
      updateUserInterface(0, 7, BLU_ON_BLK,  "~                   SID: U01136665                   ~");
      updateUserInterface(0, 8, BLU_ON_BLK,  "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

      updateUserInterface(4, 10, YEL_ON_BLK,  "  Coffee Level (oz)");
      updateUserInterface(16, 11, YEL_ON_BLK,  "   Set:");
      updateUserInterface(24, 11, RED_ON_BLK,  "unknown");
      updateUserInterface(16, 12, YEL_ON_BLK,  "Actual:");

      updateUserInterface(4, 14, YEL_ON_BLK,  "  Coffee Temperature");
      updateUserInterface(16, 15, YEL_ON_BLK,  "   Set:");
      updateUserInterface(24, 15, RED_ON_BLK,  "unknown");
      updateUserInterface(16, 16, YEL_ON_BLK,  "Actual:");

      updateUserInterface(4, 18, YEL_ON_BLK,  "  Start Brew");
      updateUserInterface(4, 20, GRN_ON_BLK,  ">>Use Joy-Stick to Select");
      //changes state to STATE_SET_MODE
      userInterfaceState = STATE_SET_MODE;
      //selects state default behavior
      buttonClicked=0;
      break;
    }

    //state:STATE_SET_MODE
    //Joy stick scrolls UP/DOWN and selects userInterfaceState
    case(STATE_SET_MODE):
    {
      //Scale and limit screen y value to line-up with screen options
      //1790 & 512 are manual cal adjust for specific platform
      ySelectorPosition += (((int16_t)adcReadVal)-1790)/512;
      ySelectorPosition = ySelectorPosition > 0xf ? 0xf : ySelectorPosition;
      ySelectorPosition = ySelectorPosition < 0 ? 0 : ySelectorPosition;
      ySelectorPosition &= 0xc;

      //default behavior : scroll up an down with current selection in GRN all other option are in YEL
      if(ySelectorPosition!=previous_ySelectorPosition)
      {
        updateUserInterface(4, 10, YEL_ON_BLK,  "  Coffee Level (oz)");
        updateUserInterface(4, 14, YEL_ON_BLK,  "  Coffee Temperature");
        updateUserInterface(4, 18, YEL_ON_BLK,  "  Start Brew");
        updateUserInterface(4, 20, YEL_ON_BLK,  "  Use Joy-Stick to Select");
        switch(ySelectorPosition)
        {
          case(0x0):updateUserInterface(4, 10, GRN_ON_BLK,  ">>Coffee Level (oz)"); break;
          case(0x4):updateUserInterface(4, 14, GRN_ON_BLK,  ">>Coffee Temperature"); break;
          case(0x8):updateUserInterface(4, 18, GRN_ON_BLK,  ">>Start Brew"); break;
          case(0xc):updateUserInterface(4, 20, GRN_ON_BLK,  ">>Use Joy-Stick to Select"); break;
        }
```



```
     previous_ySelectorPosition = ySelectorPosition;
     //add additional delay to slow slew rate
     vTaskDelay(pdMS_TO_TICKS(100));
   }
   //active behavior : detect user button click and set userInterfaceState.
   if(buttonClicked)
   {
     //selects state default behavior
     buttonClicked=0;
     switch(ySelectorPosition)
     {
                     //user selection adjust coffee volume
         case(0x0):  userInterfaceState = STATE_SET_VOLUME;
                     break;
                     //user selection adjust coffee temperature
         case(0x4):  userInterfaceState = STATE_SET_TEMPERATURE;
                     break;
         case(0x8):  userInterfaceState = STATE_BREW;
                     updateUserInterface(4, 18, YEL_ON_BLK,  "  Start Brew");
                     updateUserInterface(6, 24, GRN_ON_BLK,  "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
                     updateUserInterface(6, 25, GRN_ON_BLK,  "~    Coffee Brew :           ~");
                     updateUserInterface(6, 26, GRN_ON_BLK,  "~  Coffee Warmer :           ~");
                     updateUserInterface(6, 27, GRN_ON_BLK,  "~      Brew Time :           ~");
                     updateUserInterface(6, 28, GRN_ON_BLK,  "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
                     break;
     }
   }
   break;
 }

//state:STATE_SET_VOLUME
//Joy stick scrolls UP/DOWN and selects levelSetpoint
 case(STATE_SET_VOLUME):
 {
   //active behavior : set userInterfaceState to STATE_SET_MODE, locks levelSetpoint
   if(buttonClicked)
   {
     //selects state default behavior
     buttonClicked=0;
     userInterfaceState = STATE_SET_MODE;
     xEventGroupSetBits( xEventGroup, EVENT_GROUP_VOLUME_SET);
     updateUserInterface(4, 10, YEL_ON_BLK,  "  Coffee Level (oz)");
     updateUserInterface(4, 20, GRN_ON_BLK,  ">>Use Joy-Stick to Select ");
     break;
   }
   //default behavior : scroll UP/DOWN in volume level in oz
```

```
          //adcReadVal of 3096 (~75% of ADC full scale) is a calibrated threshold for levelSetPoint decrement
          if(adcReadVal > 3096)
            levelSetpoint--;

          //adcReadVal of 1024 (~25% of ADC full scale) is a calibrated threshold for levelSetPoint increment
          if(adcReadVal < 1024)
            levelSetpoint++;

          //levelSetpoint maximum size is 20oz
          if(levelSetpoint > 20)
            levelSetpoint=20;

          //levelSetpoint minimum (and default) size is 4oz
          if(levelSetpoint < 4)
            levelSetpoint=4;

          //formats msg for real time ASNI updates to the trminal
          sprintf(buf, "%li oz    ", levelSetpoint);
          updateUserInterface(24, 11, GRN_ON_BLK,  buf);

          //add additional 100ms delay to slow slew rate
          vTaskDelay(pdMS_TO_TICKS(100));

          break;
        }

        //state:STATE_SET_TEMPERATURE
        //Joy stick scrolls UP/DOWN and selects tempSetpoint
        case(STATE_SET_TEMPERATURE):
        {
          //active behavior : set userInterfaceState to STATE_SET_MODE, locks tempSetpoint
          if(buttonClicked)
          {
            //selects state default behavior
            buttonClicked=0;
            userInterfaceState = STATE_SET_MODE;
            xEventGroupSetBits( xEventGroup, EVENT_GROUP_TEMPERATURE_SET);
            updateUserInterface(4, 14, YEL_ON_BLK,  "  Coffee Temperature (degC)");
            updateUserInterface(4, 20, GRN_ON_BLK,  ">>Use Joy-Stick to Select ");
            break;
          }

          //default behavior : scroll UP/DOWN in volume level in oz

          //adcReadVal of 3096 (~75% of ADC full scale) is a calibrated threshold for levelSetPoint decrement
          if(adcReadVal > 3096)
            tempSetpoint--;
```



```
      //adcReadVal of 1024 (~25% of ADC full scale) is a calibrated threshold for levelSetPoint increment
      if(adcReadVal < 1024)
        tempSetpoint++;

      //levelSetpoint maximum size is 60degC
      if(tempSetpoint>60)
        tempSetpoint=60;

      //tempSetpoint minimum (and default) size is 24degC
      if(tempSetpoint<24)
        tempSetpoint=24;

      //formats msg for real time ASNI updates to the trminal
      sprintf(buf, "%li degC / %li degF ", tempSetpoint, (uint32_t)((double)tempSetpoint * 1.8) + 32);
      updateUserInterface(24, 15, GRN_ON_BLK,  buf);

      //add additional 100msec delay to slow slew rate
      vTaskDelay(pdMS_TO_TICKS(100));

      break;
    }

    //state:STATE_SET_BREW
    //Joy stick selects BREW
    //xEventGroup EVENT_GROUP_BREW_ENABLED=1, releases  prvTaskProcessTemperature & prvTaskProcessLevel
    case(STATE_BREW):
    {
      //selects state default behavior
      buttonClicked=0;
      xEventGroupSetBits( xEventGroup, EVENT_GROUP_BREW_ENABLED);

      //start Brew timer
      xTimerStart(xBrewTimerAutoReload, 0);

      updateUserInterface(4, 20, GRN_ON_BLK,  ">>Use Joy-Stick to Select ");
      //set task to idle
      enableBrew = 1;
      break;
    }

    //state:INVALID
    default:
      //exception
      while(1);
  }
}
```

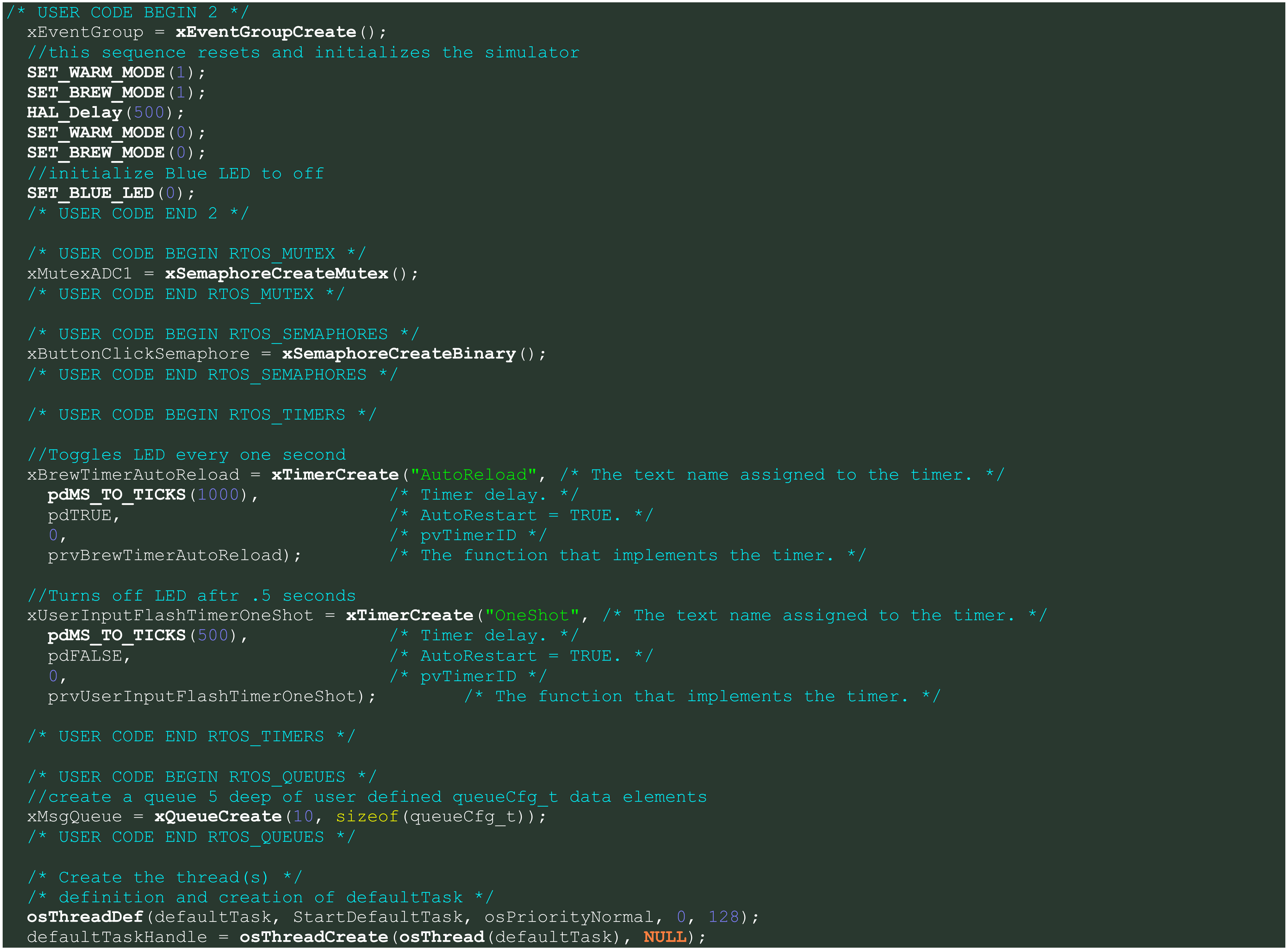
```
    //process user interface ~ 10HZ
    //100msec is a 10Hz update rate
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
```

Real-time temperature monitor/controller task

```c
static void prvTaskProcessTemperature(void* pvParameters)
{
  char buf[80];
  uint32_t adcTemperature;
  for (;;)
  {
    //Check event group for OK to monitor & control temperature.
    xEventGroupWaitBits(xEventGroup,
        EVENT_GROUP_TEMPERATURE_SET | EVENT_GROUP_BREW_ENABLED,
        pdFALSE, //do not clear bits
        pdTRUE,  //wait for all bits
        portMAX_DELAY);

    //Take mutex to access ADC.
    xSemaphoreTake(xMutexADC1, portMAX_DELAY);
    {
      //Read ADC to acquire coffee temperature.
      sConfig.Channel = ADC_CHANNEL_3;
      HAL_ADC_ConfigChannel(&hadc1, &sConfig);
      HAL_ADC_Start(&hadc1);
      while(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)!=HAL_OK);
      adcTemperature = (uint32_t)( ((float)HAL_ADC_GetValue(&hadc1)/4096) * 100);
    }
    //Give mutex
    xSemaphoreGive(xMutexADC1);

    //Send output message to pvTaskConsoleOutput by placing a user defined data element on a queue.
    sprintf(buf, "%li degC / %li degF ", adcTemperature, (uint32_t)((double)adcTemperature * 1.8) + 32);
    updateUserInterface(24, 16, GRN_ON_BLK, buf);

    //Enable coffee warmer if coffee temperature below set point.
    if(adcTemperature > (tempSetpoint+1))
    {
      SET_WARM_MODE(0);
      updateUserInterface(25, 26, YEL_ON_BLK, "OFF");
    }

    //Disable coffee warmer if coffee temperature above the set point.
    if(adcTemperature < (tempSetpoint-1))
    {
      SET_WARM_MODE(1);
      updateUserInterface(25, 26, RED_ON_BLK, "ON ");
    }

    //update temperature at  1hz rate
    //delay 1000msec
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
}
```

Real-time level/volume monitor/controller task

```
static void prvTaskProcessLevel(void* pvParameters)
{
  char buf[80];
  uint32_t adcLevel;
  for (;;)
  {
    //Check event group for OK to monitor & control level.
    xEventGroupWaitBits(xEventGroup,
        EVENT_GROUP_VOLUME_SET | EVENT_GROUP_BREW_ENABLED,
        pdFALSE, //do not clear bits
        pdTRUE,  //wait for all bits
        portMAX_DELAY);

    //Take mutex to access ADC.
    xSemaphoreTake(xMutexADC1, portMAX_DELAY);
    {
      //Read ADC to acquire coffee level.
      sConfig.Channel = ADC_CHANNEL_2;
      HAL_ADC_ConfigChannel(&hadc1, &sConfig);
      HAL_ADC_Start(&hadc1);
      while(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)!=HAL_OK);
      adcLevel = (uint32_t)( ((float)HAL_ADC_GetValue(&hadc1)/4096) * 24);
    }
    //Give mutex
    xSemaphoreGive(xMutexADC1);

    //Send output message to pvTaskConsoleOutput by placing a user defined data element on a queue.
    sprintf(buf, "%li oz  ", adcLevel);
    updateUserInterface(24, 12, GRN_ON_BLK, buf);

    if(adcLevel >= levelSetpoint)
    {
      SET_BREW_MODE(0);
      updateUserInterface(25, 25, RED_ON_BLK, "READY");
//      //this terminates the coffee level processing until a reset/restart
//      xEventGroupSetBits( xEventGroup, EVENT_GROUP_VOLUME_SET);
    }
    else
    {
      SET_BREW_MODE(1);
      updateUserInterface(25, 25, GRN_ON_BLK, "ON   ");
    }
    //update volume at 1hz rate
    //delay 1000msec
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
}
//C callback function to trap interrupt for Blue <USER> button press
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  xHigherPriorityTaskWoken = pdFALSE;
  UBaseType_t unSavedInterruptStatus;

  if(GPIO_Pin == BUTTON_EXTI13_Pin)
  {
    //turn LED ON
    //since this GPIO is also controlled by a task
    //need to designate this as a critical section for GPIO resourse management.
    unSavedInterruptStatus = taskENTER_CRITICAL_FROM_ISR();
    SET_BLUE_LED(1);
    taskEXIT_CRITICAL_FROM_ISR(unSavedInterruptStatus);

    //start one shot timer that turns LED off.
    xTimerStartFromISR(xUserInputFlashTimerOneShot, 0);
    xSemaphoreGiveFromISR(xButtonClickSemaphore, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
}
```



```
/* USER CODE BEGIN 2 */
  xEventGroup = xEventGroupCreate();
  //this sequence resets and initializes the simulator
  SET_WARM_MODE(1);
  SET_BREW_MODE(1);
  HAL_Delay(500);
  SET_WARM_MODE(0);
  SET_BREW_MODE(0);
  //initialize Blue LED to off
  SET_BLUE_LED(0);
  /* USER CODE END 2 */

  /* USER CODE BEGIN RTOS_MUTEX */
  xMutexADC1 = xSemaphoreCreateMutex();
  /* USER CODE END RTOS_MUTEX */

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  xButtonClickSemaphore = xSemaphoreCreateBinary();
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */

  //Toggles LED every one second
  xBrewTimerAutoReload = xTimerCreate("AutoReload", /* The text name assigned to the timer. */
    pdMS_TO_TICKS(1000),            /* Timer delay. */
    pdTRUE,                         /* AutoRestart = TRUE. */
    0,                              /* pvTimerID */
    prvBrewTimerAutoReload);        /* The function that implements the timer. */

  //Turns off LED aftr .5 seconds
  xUserInputFlashTimerOneShot = xTimerCreate("OneShot", /* The text name assigned to the timer. */
    pdMS_TO_TICKS(500),             /* Timer delay. */
    pdFALSE,                        /* AutoRestart = TRUE. */
    0,                              /* pvTimerID */
    prvUserInputFlashTimerOneShot);        /* The function that implements the timer. */

  /* USER CODE END RTOS_TIMERS */

  /* USER CODE BEGIN RTOS_QUEUES */
  //create a queue 5 deep of user defined queueCfg_t data elements
  xMsgQueue = xQueueCreate(10, sizeof(queueCfg_t));
  /* USER CODE END RTOS_QUEUES */

  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
```

```
/* USER CODE BEGIN RTOS_THREADS */
BaseType_t rtnVal;
rtnVal = xTaskCreate(prvTaskConsoleOutput, "cons-out task", configMINIMAL_STACK_SIZE, NULL, 2, NULL);
if(rtnVal == -1)  //exception
  while(1);

rtnVal = xTaskCreate(prvTaskProcessUserInput, "proc-user task", configMINIMAL_STACK_SIZE, NULL, 3, NULL);
if(rtnVal == -1)  //exception
  while(1);

rtnVal = xTaskCreate(prvTaskProcessTemperature, "pro-temp task", configMINIMAL_STACK_SIZE, NULL, 4, NULL);
if(rtnVal == -1)  //exception
  while(1);

rtnVal = xTaskCreate(prvTaskProcessLevel, "proc-vol task", configMINIMAL_STACK_SIZE, NULL, 4, NULL);
if(rtnVal == -1)  //exception
  while(1);

/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* Start scheduler */
osKernelStart();
```