Department of Electrical,
Computer, & Biomedical Engineering
Faculty of Engineering
& Architectural Science

Course Title:	Embedded Systems Design
Course Number:	COE718
Semester/Year:	Fall 2023

Instructor:	Dr. Gul N. Khan
TA:	Wenhao Wu

Assignment/Lab Number:	Project
Assignment/Lab Title:	Media Center

Submission Date:	2 December 2023
Due Date:	1 December 2023

LAST NAME	**FIRST NAME**	**Student Number**	**Section**	**Signature**	**Email**
Melegrito	Nyle Fernan	500974255	1	*NyleMelegrito*	nylefernan.melegrito@torontomu.ca

Abstract

This report showcases my Media Center project created using the knowledge learned from the COE718: Embedded Systems Design course. The media center includes a gallery, MP3 player, and a Space Invaders game. The media center is presented on an LCD screen and navigated using a joystick. I use a thread for processing joystick input. I created functions that display objects or a screen onto the LCD. The joystick thread and other threads (including timers) call these functions to change what is shown on the LCD display. The media center presents the user with menus to help with navigation. The gallery showcases bitmap images on the LCD. The MP3 player connects to the PC and uses a USB cable to stream audio from the PC to the onboard speaker. The Space Invaders game is based on the classic arcade game. It uses bitmap images to display sprites. The joystick is used to play the game and determines the player's movements in the game. The game uses timers to implement the 'A.I.' or behavior of NPCs. Another thread is used to represent bullet behavior in the game. I created utility functions that help with game logic. This includes collision detection between the bullet and aliens. Also a function that checks for a win or lose condition to end the game.
I used past projects and example projects from the school directory to help guide my implementation for the project. Past projects taught me how to interface with hardware, create interrupts service routines, use threads, schedule threads, use timers, and use mutexes.
The gallery worked as expected with no problems. The MP3 player works as well, but closing it caused the application to stop working. The game is fun and works with just a few bugs that do not ruin or break the game.
I could improve this project with more time and by utilizing threads more. It will require scheduling work but it would remove bugs and allow easy expansion on the project.

Introduction

This final project is the culmination of all the principles, techniques, and knowledge I learned about - embedded systems, the ARM Cortex-M3 processor, the NXP LPC1768 microcontroller, and the RTX Real-Time Operating System (RTOS).
The project itself is a media center that has three main components: a gallery, MP3 player, and one game.

Review

Hardware

I can access and configure peripheral devices by setting certain bits in certain registers either through masking or bitbanding. I was able to interact with the LCD screen by including GLCD support in the run-time environment. It gave me access to the *GLCD_SPI_LPC1700.c* and *GLCD.h* files that gave me functions that allowed me to control what is displayed on the screen. I included the *KBD.c* and *KBD.h* files given in lab 1 that gives me functions that help me get joystick input using a similar implementation in lab 1 that uses *if* statements that compare the current joystick value to set values representing the possible joystick inputs.

Threads

Lab 3 and up taught me how to create and initialize threads, as well as how to refer to them with their ID. Threads are streams of independent executions. Threads can be used to run multiple functionalities of a program a user would want to happen simultaneously. However, the kernel or

operating system only runs one thread at a time, but saves a threads context before switching one out. Therefore, with proper scheduling it gives the user the illusion of simultaneous execution of multiple functionalities the program has to offer. I also learned how to define and use a mutex when multiple threads want access to the same resource. This stops the resource from being corrupted or have weird values.

```
158  osMutexId mutex_id;
159  osMutexDef(screenMutex);
160  void CreateMutex (void)  {
161
162    mutex_id = osMutexCreate  (osMutex (screenMutex));
163    if (mutex_id != NULL)  {
164      // Mutex object created
165    }
166  }
167  void threadJS (void const *argument);
168  void threadB (void const *argument);
169
170  osThreadDef(threadJS, osPriorityNormal, 1, 0);
171  osThreadDef(threadB, osPriorityAboveNormal, 1, 0);
172  osThreadDef(play_MP3, osPriorityNormal, 1, 0);
173
174  osThreadId tidJS;
175  osThreadId tidB;
176  osThreadId tidMP3;
```

Timers

Lab 4 introduced timers on how to define and use them. They count down from a set amount of milliseconds and call a callback function when the timer hits zero. The timer would be periodic or just be used once. This gives the program a way to schedule one-time events that do not need to happen continuously unlike a thread.

```
185  void updateGame();
186  void showLose();
187  void showWin();
188  void checkWinLose();
189  void alien_dodge(void const *param);
190  void alien_approach(void const *param);
191  osTimerDef(timerY_handle, alien_dodge);
192  osTimerId alienY_timer;
193  osTimerDef(timerX_handle, alien_approach);
194  osTimerId alienX_timer;
```

Scheduling Techniques

Lab 3 taught me how to change a threads priority to allow certain threads to run instead of others when both threads are in a ready state. Lab 4 introduces Rate Monotonic Scheduling that schedules periodic tasks based on their deadline/period. When there is a group of repetitive tasks,

the programmer takes in their execution time and periods to design the program. The tasks should finish execution before the next time they are called to start again. Usually the task with the shortest deadline is prioritized and has a low chance of missing its deadline.

Methodology

General Design

I wanted to implement my project following a Rate Monotonic Scheduling (RMS) scheme. However, I did not completely follow this scheme for the whole project. This scheduling theme is mostly used for implementing the game. For most of the program I mostly just have 1 thread running that calls functions that change the LCD screen. This thread takes joystick input and updates the LCD screen by calling a chain of functions. I have a set of functions that show a specific screen, another set for updating a screen, and a few functions that decide which screen to show or update. The user navigates the media center by using the joystick to move his selection through the various menus, selected images in the gallery, and selected position in the game. I use variables to keep track of the user's selection or position in the screen and the state of the screen.

Reasoning Behind the One Thread

At first I was thinking about a foreground/background system where the foreground just handles interrupts and the background handles processing of data. Interrupts would take in input and send this data to a background process to handle the output, like deciding which screen to show or update then actually printing the screen on the LCD. I even considered using a queue to process inputs in order. Therefore, I'd have one thread to act as an ISR for joystick input, another thread that decides which context variables to change based on the input, then a last thread that determines which screen to print based on these variables and calls one of the functions that shows or updates a screen. I decided against this because at the time I began the project I was only considering the fact that there was one source of input (the joystick) and one output (the LCD screen). Though I knew about the MP3 player adding another pair of input and output, I ignored it and saved it for last. Thus, with only one input and output to consider using a queue and extra threads felt like extra steps and more work. Additionally, I wanted the screen to respond to joystick input immediately as well. Meaning if I were to queue a joystick input, it would be dequeued right away and processed. The flow of the program seemed very linear at the time (because I was only thinking about implementing the screens and none of the functionality yet) and one thread that did everything in the order I wanted made the most sense. I will discuss later how this was not the best way to implement the project and how I could improve on my methodology.

Gallery

The gallery outputs bitmap images onto the LCD screen. When the gallery is first selected from the menu it enters a list view of all the available images, showing the names of the image files. Upon selecting an image it will be displayed on the screen. This also enters a carousel-like view for the gallery, allowing the user to scroll through the images with the joystick without first having to select an image from the list view.

To load the images onto the board I first had to edit them to fit inside the LCD screen. I had to fix the dimensions of the image to fit my liking and based on the dimensions of the LCD. I also had to vertically flip some images because the board flips them when they are displayed. I then had to save the image as a source C file (.c) to get a bitmap array that described the image for the board.

MP3 Player

The MP3 player is based on the USB audio example project found in the course directory. For implementation I simply copied the same files in the example project into my own and altered a few things. The MP3 player connects to the computer using a USB and outputs audio from the computer out the speakers found on the board.

Game

The game is based on the classic arcade shooter game ‘Space Invaders’. The game involves a spaceship that the user controls and a group of aliens. The spaceship and aliens are placed on opposite ends of the screen. When the game begins the aliens slowly approach the spaceship while making perpendicular movements to make targeting them more challenging. The player must then control the spaceship to shoot bullets at the aliens to eliminate them. The goal of the game is to eliminate all aliens before they can reach the spaceship.

Design

Upon initialization, the joystick thread runs infinitely with a loop. It gets the joystick input value and uses *if* statements to determine which direction the user inputted. I designed the whole project so that movement or selection only occurs vertically. I use a *position* variable to keep track of the user’s current selection in the menus, image selection in the gallery’s carousel view, and the spaceship’s position in the game. Therefore, moving the joystick up or down increments or decrements this *position* variable. Both inputs would lead to calling a function that updates the screen. The two other inputs that matter to the program are selecting the joystick and moving it left. Both inputs lead to calling a function that shows a different type of screen. Moving the joystick left is used to go backwards in the application, while selecting is used to go forward as a menu item or image is selected.

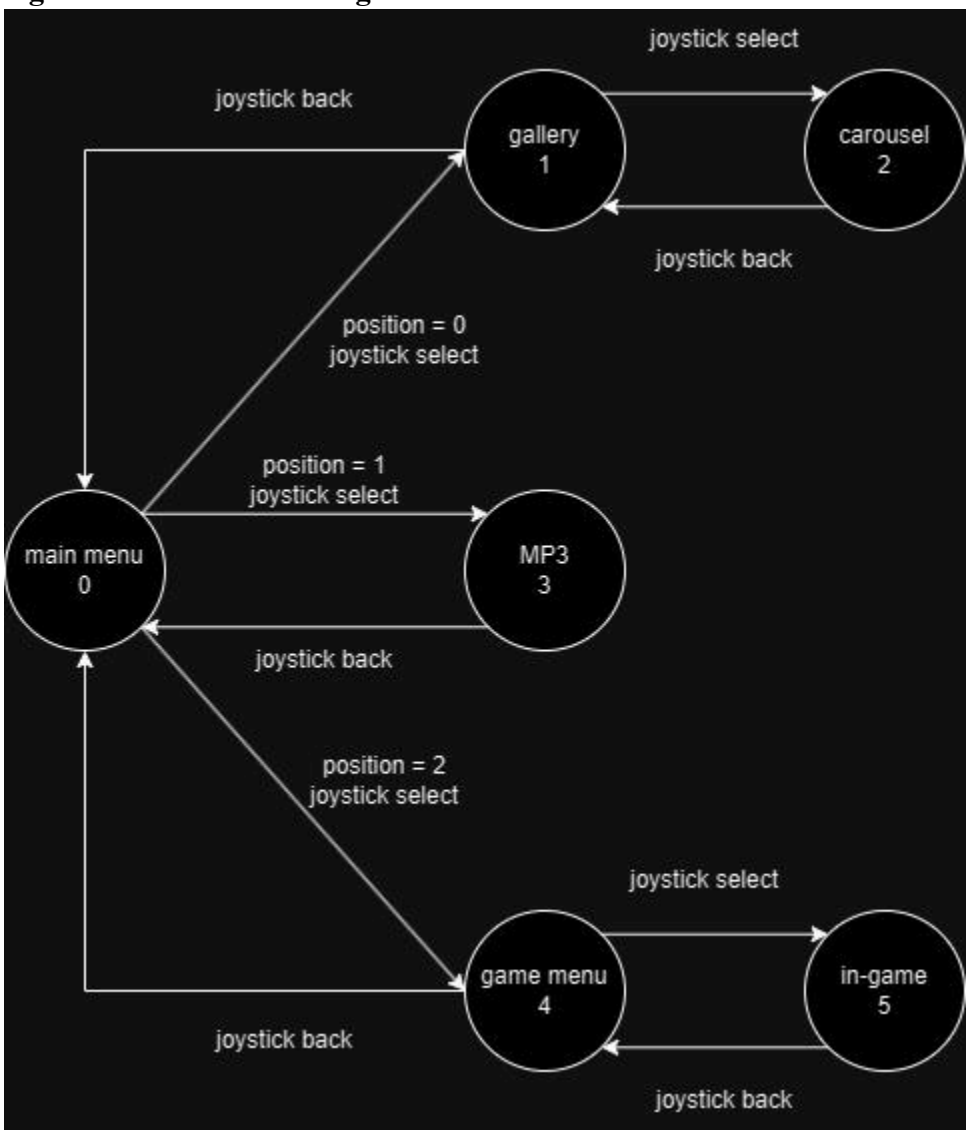
Figure 1. Joystick Thread

```
596 void threadJS (void const *argument) { //JOYSTICK THREAD
597   for(;;) {
598     jsVal = get_button();
599
600     if(jsVal == KBD_DOWN){
601       position++;
602       if(screenState == 5 && position > 4)
603         position = 4;
604     }
605     else if(jsVal == KBD_UP){
606       if(position)
607         position--;
608       else{
609         switch(screenState){
610           case 0:
611             position = 2;
612             break;
613           case 1:
614           case 2:
615             position = p-1;
616             break;
617           case 5:
618             position = 0;
619             break;
620         }//switch
621       }//if
622     }
623     else if(jsVal == KBD_SELECT) {
624       osMutexWait(mutex_id, osWaitForever);
625       select();
626       osMutexRelease(mutex_id);
627       continue;
628     }
629     else if(jsVal == KBD_LEFT) {
630       osMutexWait(mutex_id, osWaitForever);
631       goBack();
632       osMutexRelease(mutex_id);
633       continue;
634     }
635     else{continue;}
636     osMutexWait(mutex_id, osWaitForever);
637     updateScreen();
638     osMutexRelease(mutex_id);
639     osDelay(3000);
640   } //for end
641 }
```

The Different Screens

I have 6 screens in total and I use a *screenState* variable to keep track of which screen is currently being displayed. Screen state 0 would be the main menu and the first screen the user sees. Screen state 1, the Gallery, would be entered when the user selects it from the main menu. Screen state 2 would be the Carousel screen which is entered when the user selects any image from the Gallery. Screen state 3 is the MP3 screen which is entered when the user selects the MP3 player from the main menu. Screen state 4 is the game menu screen which is entered when the user selects the 'Space Invaders' game from the main menu. Screen state 5 is the in-game screen when the game is started by continuing from the game menu.

Figure 2. Screen State Diagram



Switching Screens

A change of screens happens when the user presses select on the joystick or swings the joystick left to go back to a previous screen. When either of those inputs are recorded the *select()* or *goBack()* function is called.

Selecting the joystick does different things depending on the screen state. I use *if* statements and the *screenState* variable to determine what to do with the select input (**Figure 3**). If *screenState=5* the select input is for shooting a bullet in the game. If *screenState=0* that means the user has selected something from the main menu. I use a *switch* statement and the *position* variable to determine whether the user wanted to enter the gallery, MP3 player, or game. The *switch* statement calls one of the three: *showGallery(), showMP3(), or showGameMenu()* (**Figure 5**). If *screenState=1* the user has selected to view an image from the gallery and the *updateCarousel()* function is called.

When moving the joystick left, the *goBack()* function (**Figure 4**) allows the user to make its way back towards the main menu. Similar to the *select()* function it uses the *screenState* variable and a *switch* statement to determine which screen to show on the LCD. If the current state is either the gallery, MP3 player, or game (*screenState = 1, 3, 4*) the *goBack()* function takes the user back to the main menu. If *screenState* = 2 then it takes the user from carousel view to the gallery list view. If *screenState* = 5 it takes the user from the game into the game menu.

Figure 3. Select Function

```
409 void select(){
410   if(screenState == 5){ //in-game
411     //shoot
412     if(!bullet_life){
413       bullet_life = 1;
414       bulletx = 0;
415       bullety = position;
416       updateGame();
417       tidB = osThreadCreate(osThread(threadB), NULL);
418     }
419   }
420   else if(screenState == 0) { //from main menu
421     GLCD_Clear(Black);
422     GLCD_SetBackColor(Black);
423     GLCD_SetTextColor(Yellow);
424     position = position % 3;
425     switch(position){
426       case 0:
427         screenState = 1;
428         position = 0;
429         showGallery(); //go to gallery
430         break;
431       case 1:
432         screenState = 3;
433         showMP3(); //go to MP3
434         tidMP3 = osThreadCreate(osThread(play_MP3), NULL);
435         break;
436       case 2:
437         screenState = 4;
438         showGameMenu(); //go to game
439     } //switch end
440   }
441   else if(screenState == 1) { //from gallery go to carousel view
442     screenState = 2;
443     updateCarousel();
444   }
445   else if(screenState == 4) { //from game menu start game
446     screenState = 5;
447     position = 2;
448     osTimerStart(alienY_timer, 40000);
449     osTimerStart(alienX_timer, 120000);
450     updateGame();
451   }
452 }
```

Figure 4. Go Back Function

```
453 void goBack() {
454   GLCD_Clear(Black);
455   GLCD_SetBackColor(Black);
456   GLCD_SetTextColor(Yellow);
457   switch(screenState) {
458     case 1: //from gallery go to main menu
459       screenState = 0;
460       position = 0;
461       showMainMenu();
462       break;
463     case 2: //from carousel view go to list view (gallery)
464       screenState = 1;
465       showGallery();
466       break;
467     case 3: //from MP3 player go to main menu
468       osThreadTerminate(tidMP3);
469       NVIC_EnableIRQ(TIMER0_IRQn);
470       USB_Connect(FALSE);
471       screenState = 0;
472       position = 1;
473       showMainMenu();
474       break;
475     case 4: //from game menu go to main menu
476       screenState = 0;
477       position = 2;
478       showMainMenu();
479       break;
480     case 5: //from in-game go to game menu
481       screenState = 4;
482       osTimerDelete(alienY_timer);
483       osTimerDelete(alienX_timer);
484       showGameMenu();
485       break;
486   } //switch end
487 }
```

Figure 5. Show Screen Functions

```
284   //SHOWING SCREENS
285 void showMainMenu() {
286     GLCD_DisplayString(0, 0, __FI, "Nyle's Media Center ");
287     GLCD_DisplayString(1, 0, __FI, "Gallery");
288     GLCD_DisplayString(2, 0, __FI, "MP3");
289     GLCD_DisplayString(3, 0, __FI, "Alien Invaders");
290     GLCD_DisplayString( position+1 , 16, __FI, "<--");
291   }
292 void showGallery() {
293     GLCD_DisplayString(0, 0, __FI, "       Gallery      ");
294     GLCD_DisplayString(1, 0, __FI, "ship.c              ");
295     GLCD_DisplayString(2, 0, __FI, "alien.c             ");
296     GLCD_DisplayString(3, 0, __FI, "alien2.c            ");
297     GLCD_DisplayString(4, 0, __FI, "bullet.c            ");
298     GLCD_DisplayString( (position%p)+1 , 16, __FI, "<--");
299   }
300 void showMP3() {
301     GLCD_DisplayString(0, 0, __FI, "         MP3        ");
302   }
303 void showGameMenu() {
304     GLCD_DisplayString(0, 0, __FI, "   Space Invaders   ");
305     GLCD_DisplayString(1, 0, __FI, "Defeat all invaders!");
306     GLCD_DisplayString(2, 0, __FI, "Use joystick to move");
307     GLCD_DisplayString(3, 0, __FI, "Use SELECT to shoot ");
308     GLCD_DisplayString(4, 0, __FI, "LEFT to leave game:(");
309
310     alien1_life = 1;
311     alien2_life = 1;
312     alien3_life = 1;
313     alien4_life = 2;
314     alien5_life = 2;
315     alien6_life = 2;
316     white_positiony = 2;
317     green_positiony = 2;
318     white_positionx = 5;
319     green_positionx = 6;
320     position_state = 1;
321     bullet_life = 0;
322   }
```

Updating Screens

Every time the joystick is moved up or down the *updateScreen()* function is called (**Figure 6**). The *updateScreen()* function uses the *screenState* variable to determine which kind of screen to update. The only screens that could be updated are the main menu screen, the gallery screens

with both the list view and carousel view, and finally the in-game screen (**Figure 7**). I will focus on the main menu and gallery list view screens in this section and expand on the carousel view and in-game screen in later sections. Both the main menu and gallery list view present a list of items that can be selected by the user. An arrow is printed to the right of the current selection to indicate to the user which item they are currently selecting. The *GLCD_DisplayString()* function allows me to print a line of text on the LCD screen. The screen has enough space for 10 lines and 20 characters on each line. The function allows me to choose which line to print on and even the starting horizontal position. I take advantage of this and the fact that the LCD screen saves its current state of pixels and only changes specific areas of the screen, depending on the LCD function called. Meaning I can print menu items and images names line by line. As long as I limit the menu items and image names within 17 characters, I can print this 3-character-wide arrow on the same line as the menu items without erasing the menu items. I set the *y* parameter of the *GLCD_DisplayString()* function to the *position* variable + 1 because the screen title is printed on line 0. The *x* parameter is set 17 so the arrow is printed on the last 3 character slots on a line on the right of the screen. As seen in **Figure 7** the functions apply modulus division on the *position* variable based on the amount of menu items a screen has. Even if the user moves the joystick down at the lowest point on the menu, it will reset back at the top of the menu. In **Figure 1** we see the *position* variable set to the last menu item's position when the user moves the joystick up when they are already at the top of a menu. It brings them to the bottom of the menu.

Figure 6. Update Screen Function

```
488 void updateScreen(){
489   switch(screenState){
490       case 0:
491         updateMainMenu();
492         break;
493       case 1:
494         updateGallery();
495         break;
496       case 2:
497         updateCarousel();
498         break;
499       case 3:
500         //MP3 - nothing rn
501         break;
502       case 4:
503         //game menu - nothing rn
504         break;
505       case 5:
506         updateGame();
507   } //switch
508 }
```

Figure 7. Screen Updating Functions

```
339  //UPDATING SCREENS
340 void updateMainMenu(){ //puts an arrow at currently selected menu item and clears the other two rows
341    position = position%3;
342
343    GLCD_DisplayString( position        +1 , 16, __FI, "<--");
344    GLCD_DisplayString( ((position+1)%3)+1 , 16, __FI, "   ");
345    GLCD_DisplayString( ((position+2)%3)+1 , 16, __FI, "   ");
346  }
347 void updateGallery(){
348    position = position%p;
349
350    GLCD_DisplayString( position        +1 , 16, __FI, "<--");
351    GLCD_DisplayString( ((position+1)%p)+1 , 16, __FI, "   ");
352    GLCD_DisplayString( ((position+2)%p)+1 , 16, __FI, "   ");
353    GLCD_DisplayString( ((position+3)%p)+1 , 16, __FI, "   ");
354  }
355 void updateCarousel(){
356    position = position%p;
357    GLCD_Clear(Black);
358    switch(position){
359      case 0:
360        GLCD_Bitmap(0, 0, 60, 48, SHIP_pixel_data);
361        break;
362      case 1:
363        GLCD_Bitmap(0, 0, 35, 48, ALIEN_pixel_data);
364        break;
365      case 2:
366        GLCD_Bitmap(0, 0, 36, 48, ALIEN2_pixel_data);
367        break;
368      case 3:
369        GLCD_Bitmap(0, 0, 48, 48, BULLET_pixel_data);
370        break;
371    } //switch
372    //GLCD_Bitmap(0, 0, 320, 240, pics[position%p]);
373  }
```

Gallery

To be able to use images in my program, I transfer the source C files of the bitmap images into my project. I make sure that the bitmap array is of type *unsigned char*. I then *extern* the name of the bitmap array into my *main.c* file which allows me to use and display the images using the *bitmap* function. I enter zero for both the *x* and *y* parameters of the function to initialize every image's position to the top left of the screen. I enter the width and height of each image into the parameters of the function. The dimensions of the images are taken from their source C file. An image is not displayed until the gallery is in carousel view when an image is selected from list view. Selecting an image calls the *updateCarousel()* function which doubles as a function that shows a new screen and updates the screen. It uses a *switch* statement that uses the *position* variable to determine which image was selected and should be displayed. The user can still move the joystick up and down to scroll through the images which changes the value of the *position* variable. Every joystick input changes or updates the screen. As long as the *screenState* variable is equal to 2, every up or down input calls the *updateCarousel()* function and displays the image corresponding to the value of the *position* variable. The way the *position* variable is calculated is the same as the gallery list view screen. This means when the user decides to go back to list view, the arrow will point to the image name that was last displayed on screen. As mentioned in the Updating Screens section, the *position* variable is modified and used in a way to allow the user to cycle through the images when they try to scroll past the end or even the beginning of list of images.

MP3

I copied everything in the *usbdmain.c* file in the USB example project and put it in a thread (**Figure 8**). I also changed the function that updates the clock and the variable that determines the clock frequency to the names found in the *system_LPC17xx.h* file to give them the clock values found in the hardware being used. When the user selects the MP3 player from the main menu, this thread is initialized. When the user moves the joystick left and the *goBack()* function is called the thread is terminated, the interrupt handler for the USB is disabled, and the USB connection is also stopped (**Figure 4**).

The example project connects to the USB cable by setting some signals in the peripheral register. It then uses an interrupt handler based on the system clock to output the audio stream coming from the USB cable. It also has another interrupt handler for when the potentiometer is turned on the board. It uses the potentiometer value to determine the speaker volume.

Figure 8. Play MP3 Thread

```
void play_MP3(void const *argument) {
  volatile uint32_t pclkdiv, pclk;
  SystemCoreClockUpdate();
  LPC_PINCON->PINSEL1 &=~((0x03<<18)|(0x03<<20));
  LPC_PINCON->PINSEL1 |= ((0x01<<18)|(0x02<<20));


  LPC_SC->PCONP |= (1 << 12);

  LPC_ADC->CR = 0x00200E04;
  LPC_DAC->CR = 0x00008000;
  pclkdiv = (LPC_SC->PCLKSEL0 >> 2) & 0x03;
  switch ( pclkdiv )
  {
  case 0x00:
  default:
    pclk = SystemCoreClock/4;
  break;
  case 0x01:
    pclk = SystemCoreClock;
  break;
  case 0x02:
    pclk = SystemCoreClock/2;
  break;
  case 0x03:
    pclk = SystemCoreClock/8;
  break;
  }
  LPC_TIM0->MR0 = pclk/DATA_FREQ - 1;
  LPC_TIM0->MCR = 3;
  LPC_TIM0->TCR = 1;
  NVIC_EnableIRQ(TIMER0_IRQn);

  USB_Init();
  USB_Connect(TRUE);

  while(1);
}
```

Space Invaders

The height of all sprites used in the game are 48 pixels because I wanted to split the screen in 5 horizontal sections (240/5 =48). The spaceship and aliens can have 5 possible positions vertically. I limited player movement to only being able to move vertically, consistent to how the user navigates the rest of the media center. While the spaceship can be moved with the joystick, I used two timers for the alien movement. One timer is used for the vertical movement (**Figure 9**) to make them harder to hit. I made them move as a group, separated into two groups. Thus, I used two variables, *white_positiony* and *green_positiony* , to track each group's vertical position on the screen because I wanted the groups to move in opposite directions. I also made a state variable to implement a finite state machine for their movement pattern. Another timer is used for their horizontal approach towards the spaceship (**Figure 10**). This timer takes 3 times as long as the previous timer because I did not want the aliens to approach the spaceship too quickly. The implementation is the same where I have two variables, *white_positionx* and *green_positionx* , to track the alien groups' horizontal position. The callback function for this timer is much simpler as it just decrements *white_positionx* and *green_positionx* . I partitioned the horizontal space to the right of the spaceship into 7 parts based on the width of the alien sprites. ((340-spaceship width)/alien width). Meaning the aliens have 7 possible horizontal positions.

Every time the timers are triggered and the joystick is moved, the LCD screen is cleared and updated. The spaceship and aliens are printed based on their position variables. Each alien has their own life variable that keeps track of their hit points. The white aliens start with 1 and the green aliens start with 2. An alien is only printed if their life variable is greater than 0.

A bullet is shot when the player selects the joystick. I implemented this by initializing a thread that controls bullet movement every time one is shot (**Figure 11**). Similar to the aliens, a bullet has variables to keep track of its horizontal and vertical position and its life. The bullet also has a height of 48 pixels and shares the same 5 possible vertical positions as the rest of the sprites. Its vertical position is set to the value of the *position* variable when the bullet is first shot. Then the thread increments its horizontal position variable. Only one bullet may be shot at a time and an *if* statement checks the *bullet_life* variable if a bullet already exists. See **Figure 12** below to see the variables used to implement the game. The *checkHit()* function (**Figure 13**) is called within the bullet thread to see when to stop displaying the bullet and terminate the bullet thread. The thread also checks if the bullet has reached the end of the screen and terminates it if it does.

The game ends when the aliens reach the spaceship or when the player eliminates all aliens. The *checkWinLose()* function is called at the end of the bullet thread every time there is a hit and it is also called at the end of the *alien_approach()* callback function. The *checkWinLose()* function checks if any aliens are alive and their horizontal position to determine whether the player has won or lost the game. It will then call either the *showLose()* or *showWin()* function to display an end-game screen (**Figure 15**).

I said that the game implements an RMS scheme because there are periodic tasks that occur. However, their deadlines are not that important, except for the player seeing their spaceship move. Even though I do not know how much time a tasks needs to fully execute, their deadlines or periods are much larger than their execution time. This means they have a high probability of completing their tasks before the next time they are called. The periodic tasks would be the joystick thread, the timers, and the bullet thread if a bullet exists.

The *select()* function resets the game variables to their initial states and initializes the timers. The *goBack()* function terminates the timers.

Figure 9. Alien Dodge Callback Function

```
195 void alien_dodge(void const *param){
196   switch(position_state){
197     case 1:
198       white_positiony = 3; //go down
199       green_positiony = 1; //go up
200       position_state = 2;
201       break;
202     case 2:
203       white_positiony = 2; //go middle
204       green_positiony = 2; //go middle
205       position_state = 3;
206       break;
207     case 3:
208       white_positiony = 1; //go up
209       green_positiony = 3; //go down
210       position_state = 4;
211       break;
212     case 4:
213       white_positiony = 2; //go middle
214       green_positiony = 2; //go middle
215       position_state = 1;
216       break;
217   }
218   osMutexWait(mutex_id, osWaitForever);
219   updateGame();
220   osMutexRelease(mutex_id);
221 }
```

Figure 10. Alien Approach Callback Function

```
222 void alien_approach(void const *param){
223   white_positionx--;
224   green_positionx--;
225   osMutexWait(mutex_id, osWaitForever);
226   updateGame();
227   osMutexRelease(mutex_id);
228   checkWinLose();
229 }
```

Figure 11. Bullet Thread

```
500 void threadB(void const *argument) { //BULLET THREAD
501   while(bullet_life) {
502     osMutexWait(mutex_id, osWaitForever);
503     updateGame();
504     osMutexRelease(mutex_id);
505     osDelay(8000);
506     bulletx++;
507     if(checkHit() || bulletx > 7){
508       bullet_life = 0;
509       checkWinLose();
510     }
511   }
512 
513 }
```

Figure 12. Space Invader Variables and Timers

```
178 //GAME
179 int8_t alien1_life = 1, alien2_life = 1, alien3_life = 1;//white
180 int8_t alien4_life = 2, alien5_life = 2, alien6_life = 2;//green
181 int8_t white_positiony = 2, green_positiony = 2, position_state = 1; //y movement
182 int8_t white_positionx = 5, green_positionx = 6; //x movement
183 int8_t bullet_life = 0, bulletx = 0, bullety;
184 
185 void updateGame();
186 void showLose();
187 void showWin();
188 void checkWinLose();
189 void alien_dodge(void const *param);
190 void alien_approach(void const *param);
191 osTimerDef(timerY_handle, alien_dodge);
192 osTimerId alienY_timer;
193 osTimerDef(timerX_handle, alien_approach);
194 osTimerId alienX_timer;
```

Figure 13. Check Hit Function

```
int checkHit(){
  if(alien1_life && bulletx == (white_positionx - 1) && bullety == (white_positiony-1)){
    alien1_life = 0;
    return 1;
  }
  if(alien2_life && bulletx == (white_positionx - 1) && bullety == (white_positiony)){
    alien2_life = 0;
    return 1;
  }
  if(alien3_life && bulletx == (white_positionx - 1) && bullety == (white_positiony+1)){
    alien3_life = 0;
    return 1;
  }
  if(alien4_life && bulletx == (green_positionx - 1) && bullety == (green_positiony-1)){
    alien4_life--;
    return 1;
  }
  if(alien5_life && bulletx == (green_positionx - 1) && bullety == (green_positiony)){
    alien5_life--;
    return 1;
  }
  if(alien6_life && bulletx == (green_positionx - 1) && bullety == (green_positiony+1)){
    alien6_life--;
    return 1;
  }
  return 0;
}
```

Figure 14. Check Win or Lose Function

```
void checkWinLose(){
  if((alien1_life || alien2_life || alien3_life) && (white_positionx == 0)){
    showLose();
    osTimerDelete(alienY_timer);
    osTimerDelete(alienX_timer);
  }
  else if((alien4_life || alien5_life || alien6_life) && (green_positionx == 0)){
    showLose();
    osTimerDelete(alienY_timer);
    osTimerDelete(alienX_timer);
  }
  else if(!(alien1_life || alien2_life || alien3_life || alien4_life || alien5_life || alien6_life)){
    showWin();
    osTimerDelete(alienY_timer);
    osTimerDelete(alienX_timer);
  }

}
```

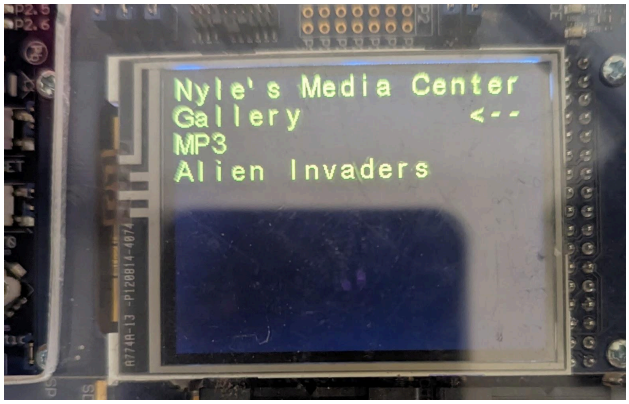
Figure 15. Showing End Game Screens

```
314 void showLose() {
315   GLCD_Clear(Black);
316   GLCD_DisplayString(0, 0, __FI, "   Space Invaders   ");
317   GLCD_DisplayString(1, 0, __FI, "   You're garbage   ");
318   GLCD_DisplayString(5, 0, __FI, "      YOU LOSE!     ");
319
320   GLCD_DisplayString(7, 0, __FI, " LEFT to Game Menu. ");
321 }
322 void showWin() {
323   GLCD_Clear(Black);
324   GLCD_DisplayString(0, 0, __FI, "   Space Invaders   ");
325   GLCD_DisplayString(1, 0, __FI, "   The aliens were  ");
326   GLCD_DisplayString(3, 0, __FI, "      deafeated.    ");
327   GLCD_DisplayString(5, 0, __FI, "       YOU WIN!     ");
328   GLCD_DisplayString(7, 0, __FI, " LEFT to Game Menu. ");
329 }
```

Results

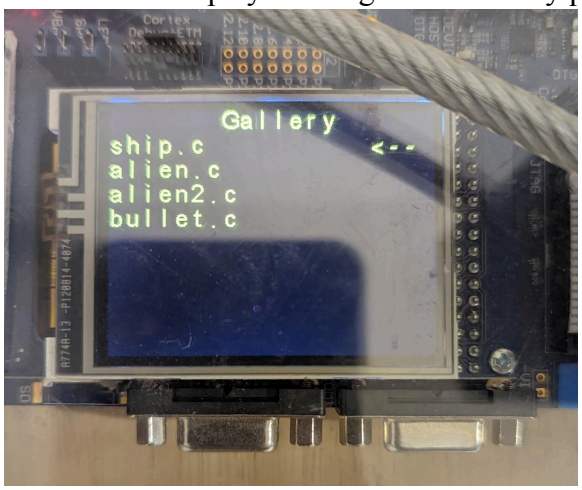
Overall Project

The project worked as intended. I was able to smoothly navigate the whole media center except leaving the MP3 player. See below the main menu and how the arrow changes based on the users selection.

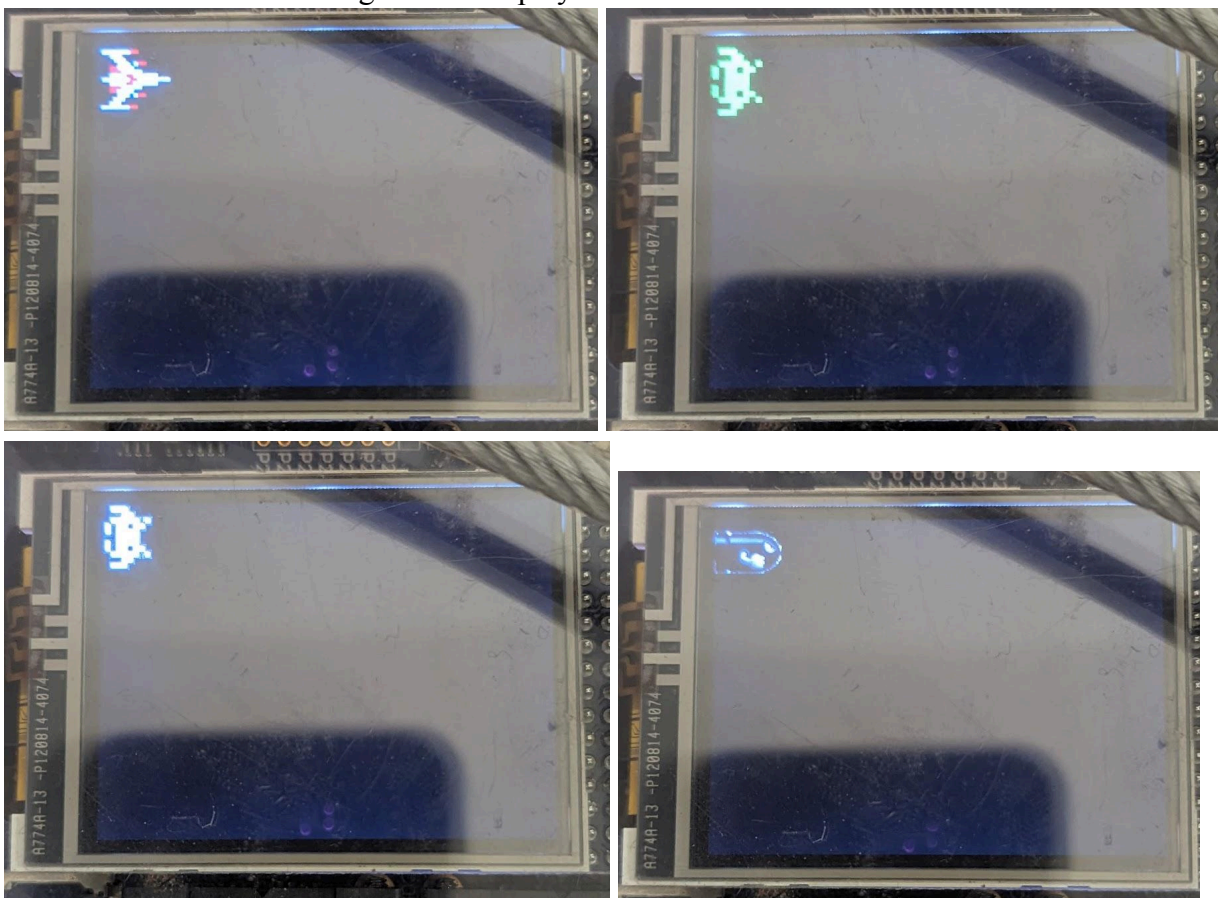


Gallery

I was able to display all images without any problems. See below the Gallery screen in list view.



See below the available images to be displayed.



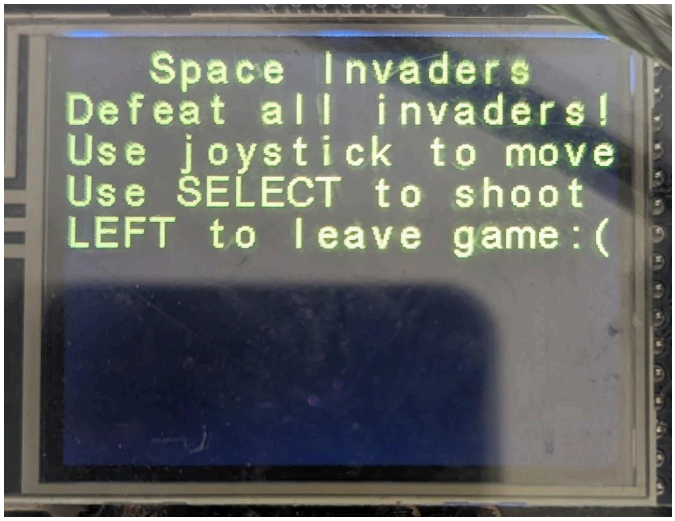
MP3

I was able to output audio from the PC onto the board using a USB cable. The only issue was turning this output off and being able to go back to the rest of the application. In **Figure 4**, I terminated the MP3 thread and disabled the USB connection. However, I had a typo where instead of disabling an interrupt I just re-enabled it instead. I think this is one issue that did not let me close the MP3 player without resetting the board. Another issue I had was that the audio that came out of the speakers was sped up. I probably had the system clock higher than usual. See MP3 screen below.

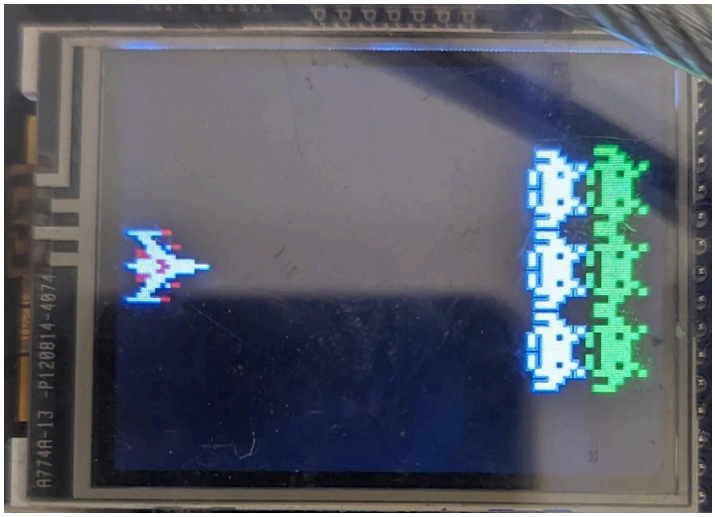
Game

I am very proud of the game. I was able to use bitmap images as sprites and develop game logic for a working game. Some issues that came up while playing were the game freezing and duplicating sprites. The game freezes when a stack overflow occurs. Sometimes multiple things call the *updateGame()* function at the same time in a short time period - like the joystick thread, bullet thread, or one of the timers. This updates the screen while some variables are changing. Some variables are used to determine where to display some of the sprites. A variable could be changed while the *updateGame()* function is still displaying images. I thought the problem was that multiple threads have access to the same resource (LCD screen) at the same time so I implemented a mutex for screen access. I failed to realize that I should have used a mutex for the variables used to display objects on the screen as well.

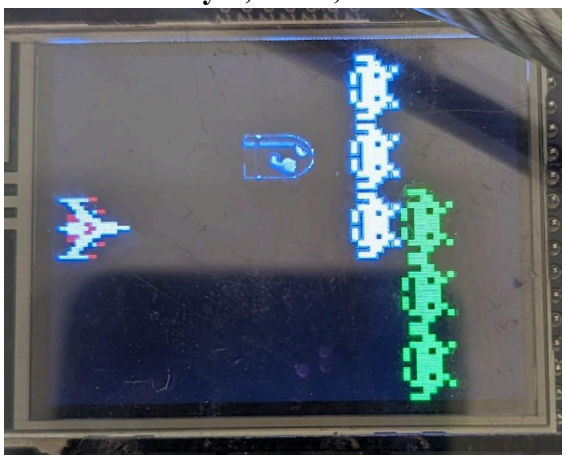
Game Menu



Initial Game Screen



Game with Player, Alien, and Bullet Movement



Conclusion

The project worked well and as intended for the most part. There were a few bugs but I at least have an idea of what could be causing those bugs and their solutions. I just needed more time. I do think I could have utilized threads more in the project as it would have helped greatly when implementing more complexity into the project. For example, I wanted to implement levels, pause screens, and level screens for the game. While I tried implementing this I found that I needed several variables to keep track of the state of the game in case the player wanted to pause and go back to the game. With threads I realized that the kernel automatically saves thread context before switching to a different thread. I could have used this context saving principle to save the game state instead of trying to keep track of it myself. I also should have implemented a thread that handled displaying things on the LCD screen. I could have set the refresh rate myself and wouldn't need to call a function that changes the LCD screen every time an event occurs like joystick input. This could have helped in the game when multiple things are trying to update the screen. Instead of those threads trying to update the screen independently, they just had to change the variables that determined what will be displayed. Once the variables have been set, the thread responsible for display will show the right things based on all these variables that other threads have changed. This would stop the game from duplicating sprites because if everything is scheduled properly, a variable cannot be changed while the LCD thread is displaying objects.

Note to David

Thanks for all the help during the semester and being a cool TA. Also sorry about putting everything in one file and making it messy. I just wanted to avoid any kind of importing problem.

Appendix
Main File (only file not given by course or Keil)

```
/*----------------------------------------------------------------------------
 * CMSIS-RTOS 'main' function template
 *---------------------------------------------------------------------------*/

#define osObjectsPublic                 // define objects in main module
#include "osObjects.h"                  // RTOS object definitions
#include "cmsis_os.h"                                   // CMSIS RTOS header file
#include <stdio.h>
#include <math.h>
#include "KBD.h"
#include "GLCD.h"
#include "LPC17xx.h"
#include "Board_LED.h"                  // ::Board Support:LED
#include "system_LPC17xx.h"

extern unsigned char BULLET_pixel_data[];
extern unsigned char ALIEN_pixel_data[];
extern unsigned char ALIEN2_pixel_data[];
extern unsigned char SHIP_pixel_data[];
#define __FI 1 /* Font index 16x24 */

//*USB STUFF
#include "type.h"
#include "usb.h"
#include "usbcfg.h"
#include "usbhw.h"
#include "usbcore.h"
#include "usbaudio.h"
extern  void SystemCoreClockUpdate(void);
extern uint32_t SystemCoreClock;
uint8_t  Mute;
uint32_t Volume;

#if USB_DMA
uint32_t *InfoBuf = (uint32_t *)(DMA_BUF_ADR);
short *DataBuf = (short *)(DMA_BUF_ADR + 4*P_C);
#else
uint32_t InfoBuf[P_C];
short DataBuf[B_S];
#endif

uint16_t  DataOut;
uint16_t  DataIn;
```

```
uint8_t   DataRun;
uint16_t  PotVal;
uint32_t  VUM;
uint32_t  Tick;


void get_potval (void) {
  uint32_t val;

  LPC_ADC->CR |= 0x01000000;
  do {
    val = LPC_ADC->GDR;
  } while ((val & 0x80000000) == 0);
  LPC_ADC->CR &= ~0x01000000;
  PotVal = ((val >> 8) & 0xF8) +
           ((val >> 7) & 0x08);
}

void TIMER0_IRQHandler(void)
{
  long  val;
  uint32_t cnt;

  if (DataRun) {
    val = DataBuf[DataOut];
    cnt = (DataIn - DataOut) & (B_S - 1);
    if (cnt == (B_S - P_C*P_S)) {
      DataOut++;
    }
    if (cnt > (P_C*P_S)) {
      DataOut++;
    }
    DataOut &= B_S - 1;
    if (val < 0) VUM -= val;
    else         VUM += val;
    val  *= Volume;
    val >>= 16;
    val  += 0x8000;
    val  &= 0xFFFF;
  } else {
    val = 0x8000;
  }

  if (Mute) {
    val = 0x8000;
  }
```

```
  LPC_DAC->CR = val & 0xFFC0;

  if ((Tick++ & 0x03FF) == 0) {
    get_potval();
    if (VolCur == 0x8000) {
      Volume = 0;
    } else {
      Volume = VolCur * PotVal;
    }
    val = VUM >> 20;
    VUM = 0;
    if (val > 7) val = 7;
  }

  LPC_TIM0->IR = 1;
}
void play_MP3(void const *argument) {
        volatile uint32_t pclkdiv, pclk;
  SystemCoreClockUpdate();
  LPC_PINCON->PINSEL1 &=~((0x03<<18)|(0x03<<20));
  LPC_PINCON->PINSEL1 |= ((0x01<<18)|(0x02<<20));


  LPC_SC->PCONP |= (1 << 12);

  LPC_ADC->CR = 0x00200E04;
  LPC_DAC->CR = 0x00008000;
  pclkdiv = (LPC_SC->PCLKSEL0 >> 2) & 0x03;
  switch ( pclkdiv )
  {
        case 0x00:
        default:
          pclk = SystemCoreClock/4;
        break;
        case 0x01:
          pclk = SystemCoreClock;
        break;
        case 0x02:
          pclk = SystemCoreClock/2;
        break;
        case 0x03:
          pclk = SystemCoreClock/8;
        break;
  }
  LPC_TIM0->MR0 = pclk/DATA_FREQ - 1;
```

```
  LPC_TIM0->MCR = 3;
  LPC_TIM0->TCR = 1;
  NVIC_EnableIRQ(TIMER0_IRQn);

  USB_Init();
  USB_Connect(TRUE);

        while(1);
}
//USB Stuff end */

uint32_t jsVal;
int8_t screenState = 0, position = 0, p = 4; //p = # of images
/* SCREEN STATES
        0 - main menu
        1 - gallery list view
        2 - gallery carousel view
        3 - MP3 player
        4 - game menu
        5 - game
*/
osMutexId mutex_id;
osMutexDef(screenMutex);
void CreateMutex (void)  {

  mutex_id = osMutexCreate  (osMutex (screenMutex));
  if (mutex_id != NULL)  {
    // Mutex object created
  }
}
void threadJS (void const *argument);
void threadB (void const *argument);

osThreadDef(threadJS, osPriorityNormal, 1, 0);
osThreadDef(threadB, osPriorityAboveNormal, 1, 0);
osThreadDef(play_MP3, osPriorityNormal, 1, 0);

osThreadId tidJS;
osThreadId tidB;
osThreadId tidMP3;

//GAME
int8_t alien1_life = 1, alien2_life = 1, alien3_life = 1;//white
int8_t alien4_life = 2, alien5_life = 2, alien6_life = 2;//green
int8_t white_positiony = 2, green_positiony = 2, position_state = 1; //y movement
int8_t white_positionx = 5, green_positionx = 6; //x movement
```

```
int8_t bullet_life = 0, bulletx = 0, bullety;

void updateGame();
void showLose();
void showWin();
void checkWinLose();
void alien_dodge(void const *param);
void alien_approach(void const *param);
osTimerDef(timerY_handle, alien_dodge);
osTimerId alienY_timer;
osTimerDef(timerX_handle, alien_approach);
osTimerId alienX_timer;
void alien_dodge(void const *param){
	switch(position_state){
		case 1:
			white_positiony = 3; //go down
			green_positiony = 1; //go up
			position_state = 2;
			break;
		case 2:
			white_positiony = 2; //go middle
			green_positiony = 2; //go middle
			position_state = 3;
			break;
		case 3:
			white_positiony = 1; //go up
			green_positiony = 3; //go down
			position_state = 4;
			break;
		case 4:
			white_positiony = 2; //go middle
			green_positiony = 2; //go middle
			position_state = 1;
			break;
	}
	osMutexWait(mutex_id, osWaitForever);
	updateGame();
	osMutexRelease(mutex_id);
}
void alien_approach(void const *param){
	white_positionx--;
	green_positionx--;
	osMutexWait(mutex_id, osWaitForever);
	updateGame();
	osMutexRelease(mutex_id);
	checkWinLose();
```

```
}
int checkHit(){
	if(alien1_life && bulletx == (white_positionx - 1) && bullety ==
(white_positiony-1)){
		alien1_life = 0;
		return 1;
	}
	if(alien2_life && bulletx == (white_positionx - 1) && bullety == (white_positiony)){
		alien2_life = 0;
		return 1;
	}
	if(alien3_life && bulletx == (white_positionx - 1) && bullety ==
(white_positiony+1)){
		alien3_life = 0;
		return 1;
	}
	if(alien4_life && bulletx == (green_positionx - 1) && bullety ==
(green_positiony-1)){
		alien4_life--;
		return 1;
	}
	if(alien5_life && bulletx == (green_positionx - 1) && bullety == (green_positiony)){
		alien5_life--;
		return 1;
	}
	if(alien6_life && bulletx == (green_positionx - 1) && bullety ==
(green_positiony+1)){
		alien6_life--;
		return 1;
	}
	return 0;
}
void checkWinLose(){
	if((alien1_life || alien2_life || alien3_life) && (white_positionx == 0)){
		showLose();
		osTimerDelete(alienY_timer);
		osTimerDelete(alienX_timer);
	}
	else if((alien4_life || alien5_life || alien6_life) && (green_positionx == 0)){
		showLose();
		osTimerDelete(alienY_timer);
		osTimerDelete(alienX_timer);
	}
	else if(!(alien1_life || alien2_life || alien3_life || alien4_life || alien5_life || alien6_life)){
		showWin();
		osTimerDelete(alienY_timer);
```

```
                osTimerDelete(alienX_timer);
        }

}
//SHOWING SCREENS
void showMainMenu() {
        GLCD_DisplayString(0, 0, __FI, "Nyle's Media Center ");
        GLCD_DisplayString(1, 0, __FI, "Gallery");
        GLCD_DisplayString(2, 0, __FI, "MP3");
        GLCD_DisplayString(3, 0, __FI, "Alien Invaders");
        GLCD_DisplayString( position+1 , 16, __FI, "<--");
}
void showGallery() {
        GLCD_DisplayString(0, 0, __FI, "       Gallery      ");
        GLCD_DisplayString(1, 0, __FI, "ship.c              ");
        GLCD_DisplayString(2, 0, __FI, "alien.c             ");
        GLCD_DisplayString(3, 0, __FI, "alien2.c            ");
        GLCD_DisplayString(4, 0, __FI, "bullet.c            ");
        GLCD_DisplayString( (position%p)+1 , 16, __FI, "<--");
}
void showMP3() {
        GLCD_DisplayString(0, 0, __FI, "        MP3        ");
}
void showGameMenu() {
        GLCD_DisplayString(0, 0, __FI, "   Space Invaders   ");
        GLCD_DisplayString(1, 0, __FI, "Defeat all invaders!");
        GLCD_DisplayString(2, 0, __FI, "Use joystick to move");
        GLCD_DisplayString(3, 0, __FI, "Use SELECT to shoot ");
        GLCD_DisplayString(4, 0, __FI, "LEFT to leave game:(");

        alien1_life = 1;
        alien2_life = 1;
        alien3_life = 1;
        alien4_life = 2;
        alien5_life = 2;
        alien6_life = 2;
        white_positiony = 2;
        green_positiony = 2;
        white_positionx = 5;
        green_positionx = 6;
        position_state = 1;
        bullet_life = 0;
}
void showLose() {
        GLCD_Clear(Black);
        GLCD_DisplayString(0, 0, __FI, "   Space Invaders   ");
```

```
        GLCD_DisplayString(1, 0, __FI, "   You're garbage   ");
        GLCD_DisplayString(5, 0, __FI, "      YOU LOSE!     ");

        GLCD_DisplayString(7, 0, __FI, " LEFT to Game Menu. ");
}
void showWin() {
        GLCD_Clear(Black);
        GLCD_DisplayString(0, 0, __FI, "   Space Invaders   ");
        GLCD_DisplayString(1, 0, __FI, "   The aliens were  ");
        GLCD_DisplayString(3, 0, __FI, "      deafeated.    ");
        GLCD_DisplayString(5, 0, __FI, "       YOU WIN!     ");
        GLCD_DisplayString(7, 0, __FI, " LEFT to Game Menu. ");
}
//UPDATING SCREENS
void updateMainMenu(){ //puts an arrow at currently selected menu item and clears the other two rows
        position = position%3;

        GLCD_DisplayString( position       +1 , 16, __FI, "<--");
        GLCD_DisplayString( ((position+1)%3)+1 , 16, __FI, "   ");
        GLCD_DisplayString( ((position+2)%3)+1 , 16, __FI, "   ");
}
void updateGallery(){
        position = position%p;

        GLCD_DisplayString( position       +1 , 16, __FI, "<--");
        GLCD_DisplayString( ((position+1)%p)+1 , 16, __FI, "   ");
        GLCD_DisplayString( ((position+2)%p)+1 , 16, __FI, "   ");
        GLCD_DisplayString( ((position+3)%p)+1 , 16, __FI, "   ");
}
void updateCarousel(){
        position = position%p;
        GLCD_Clear(Black);
        switch(position){
                case 0:
                        GLCD_Bitmap(0, 0, 60, 48, SHIP_pixel_data);
                        break;
                case 1:
                        GLCD_Bitmap(0, 0, 35, 48, ALIEN_pixel_data);
                        break;
                case 2:
                        GLCD_Bitmap(0, 0, 36, 48, ALIEN2_pixel_data);
                        break;
                case 3:
                        GLCD_Bitmap(0, 0, 48, 48, BULLET_pixel_data);
                        break;
```

```
        } //switch
        //GLCD_Bitmap(0, 0, 320, 240, pics[position%p]);
}
void updateGame() {
        GLCD_Clear(Black);
        GLCD_Bitmap(0, (position%5) * 48, 60, 48, SHIP_pixel_data); //shows ship based
on position

        if(bullet_life){
                GLCD_Bitmap((bulletx * 36) + 60, (bullety%5) * 48, 48, 48,
BULLET_pixel_data);
        }
        //WHITE ALIENS
        if(alien1_life){
                GLCD_Bitmap((white_positionx * 36) + 60, (white_positiony - 1) * 48, 36, 48,
ALIEN2_pixel_data);
        }
        if(alien2_life){
                GLCD_Bitmap((white_positionx * 36) + 60, (white_positiony) * 48, 36, 48,
ALIEN2_pixel_data);
        }
        if(alien3_life){
                GLCD_Bitmap((white_positionx * 36) + 60, (white_positiony + 1) * 48, 36,
48, ALIEN2_pixel_data);
        }
        //GREEN ALIENS
        if(alien4_life){
                GLCD_Bitmap((green_positionx * 36) + 60, (green_positiony - 1) * 48, 35, 48,
ALIEN_pixel_data);
        }
        if(alien5_life){
                GLCD_Bitmap((green_positionx * 36) + 60, (green_positiony) * 48, 35, 48,
ALIEN_pixel_data);
        }
        if(alien6_life){
                GLCD_Bitmap((green_positionx * 36) + 60, (green_positiony + 1) * 48, 35,
48, ALIEN_pixel_data);
        }

        //REPLACE SHIP IMAGE WITH ALL BLACK IMAGE
        /*
        GLCD_Bitmap(0, ((position+1)%5) * 48, 60, 48, SHIP_pixel_data);
        GLCD_Bitmap(0, ((position+2)%5) * 48, 60, 48, SHIP_pixel_data);
        GLCD_Bitmap(0, ((position+3)%5) * 48, 60, 48, SHIP_pixel_data);
        GLCD_Bitmap(0, ((position+4)%5) * 48, 60, 48, SHIP_pixel_data);*/
}
```

```
void select(){
	if(screenState == 5){ //in-game
		//shoot
		if(!bullet_life){
			bullet_life = 1;
			bulletx = 0;
			bullety = position;
			updateGame();
			tidB = osThreadCreate(osThread(threadB), NULL);
		}
	}
	else if(screenState == 0) { //from main menu
		GLCD_Clear(Black);
		GLCD_SetBackColor(Black);
		GLCD_SetTextColor(Yellow);
		position = position % 3;
		switch(position){
			case 0:
				screenState = 1;
				position = 0;
				showGallery(); //go to gallery
				break;
			case 1:
				screenState = 3;
				showMP3(); //go to MP3
				tidMP3 = osThreadCreate(osThread(play_MP3), NULL);
				break;
			case 2:
				screenState = 4;
				showGameMenu(); //go to game
		} //switch end
	}
	else if(screenState == 1) { //from gallery go to carousel view
		screenState = 2;
		updateCarousel();
	}
	else if(screenState == 4) { //from game menu start game
		screenState = 5;
		position = 2;
		osTimerStart(alienY_timer, 40000);
		osTimerStart(alienX_timer, 120000);
		updateGame();
	}
}
void goBack() {
	GLCD_Clear(Black);
```

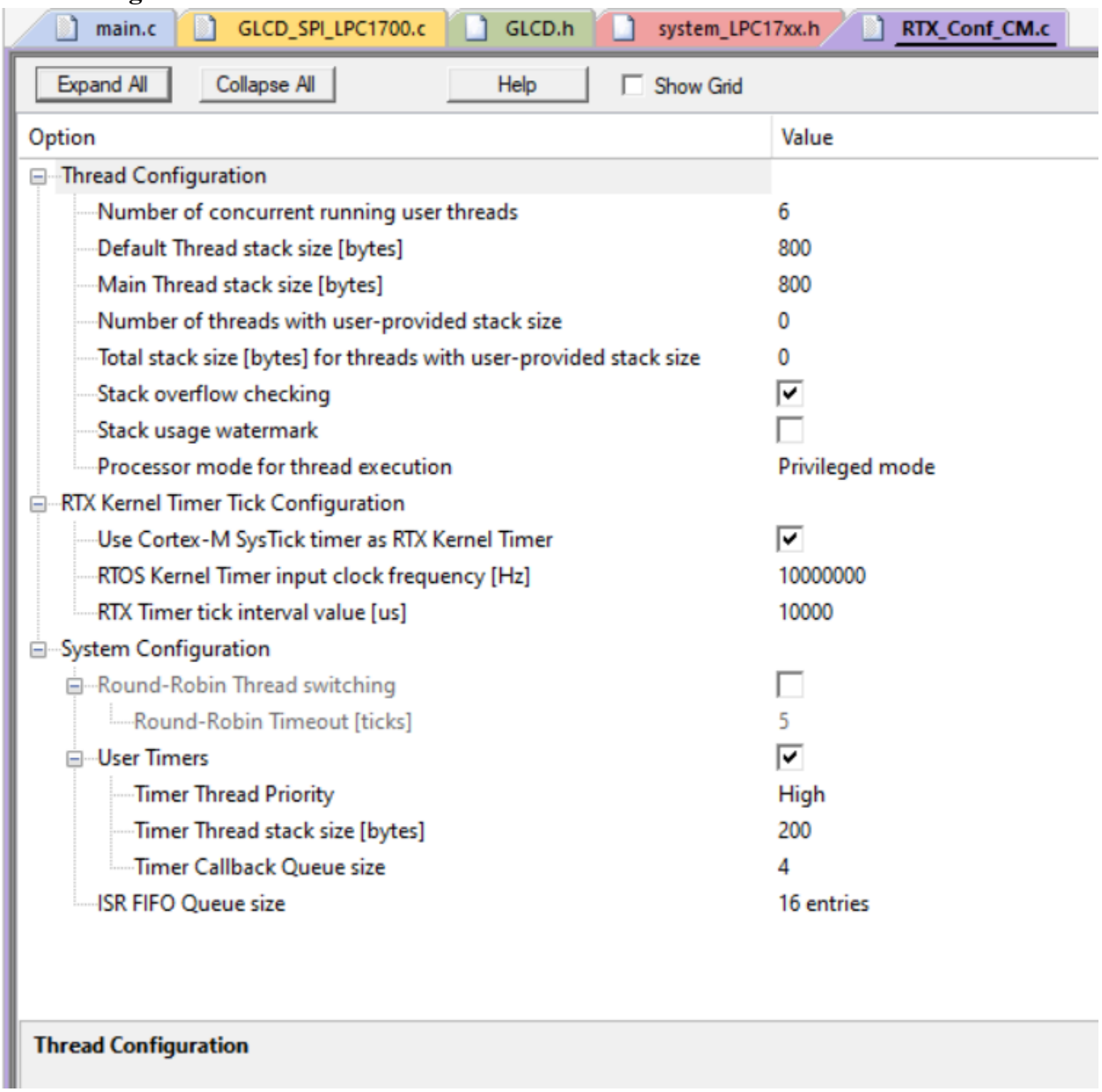
```
	GLCD_SetBackColor(Black);
	GLCD_SetTextColor(Yellow);
	switch(screenState) {
		case 1: //from gallery go to main menu
			screenState = 0;
			position = 0;
			showMainMenu();
			break;
		case 2: //from carousel view go to list view (gallery)
			screenState = 1;
			showGallery();
			break;
		case 3: //from MP3 player go to main menu
			osThreadTerminate(tidMP3);
			NVIC_EnableIRQ(TIMER0_IRQn);
			USB_Connect(FALSE);
			screenState = 0;
			position = 1;
			showMainMenu();
			break;
		case 4: //from game menu go to main menu
			screenState = 0;
			position = 2;
			showMainMenu();
			break;
		case 5: //from in-game go to game menu
			screenState = 4;
			osTimerDelete(alienY_timer);
			osTimerDelete(alienX_timer);
			showGameMenu();
			break;
	} //switch end
}
void updateScreen(){
	switch(screenState){
			case 0:
				updateMainMenu();
				break;
			case 1:
				updateGallery();
				break;
			case 2:
				updateCarousel();
				break;
			case 3:
				//MP3 - nothing rn
```

```
                        break;
                case 4:
                        //game menu - nothing rn
                        break;
                case 5:
                        updateGame();
        } //switch
}
void threadB(void const *argument) { //BULLET THREAD
        while(bullet_life) {
                osMutexWait(mutex_id, osWaitForever);
                updateGame();
                osMutexRelease(mutex_id);
                osDelay(8000);
                bulletx++;
                if(checkHit() || bulletx > 7){
                        bullet_life = 0;
                        checkWinLose();
                }
        }

}
void threadJS (void const *argument) { //JOYSTICK THREAD
        for(;;) {
                jsVal = get_button();

                if(jsVal == KBD_DOWN){
                        position++;
                        if(screenState == 5 && position > 4)
                                position = 4;
                }
                else if(jsVal == KBD_UP){
                        if(position)
                                position--;
                        else{
                                switch(screenState){
                                        case 0:
                                                position = 2;
                                                break;
                                        case 1:
                                        case 2:
                                                position = p-1;
                                                break;
                                        case 5:
                                                position = 0;
                                                break;
```

```
                        }//switch
                    }//if
                }
                else if(jsVal == KBD_SELECT) {
                    osMutexWait(mutex_id, osWaitForever);
                    select();
                    osMutexRelease(mutex_id);
                    continue;
                }
                else if(jsVal == KBD_LEFT) {
                    osMutexWait(mutex_id, osWaitForever);
                    goBack();
                    osMutexRelease(mutex_id);
                    continue;
                }
                else{continue;}
                osMutexWait(mutex_id, osWaitForever);
                updateScreen();
                osMutexRelease(mutex_id);
                osDelay(3000);
        }       //for end
}
/*
 * main: initialize and start the system
 */
int main (void) {
  osKernelInitialize ();                // initialize CMSIS-RTOS

  // initialize peripherals here
        KBD_Init();
        GLCD_Init();
        LED_Initialize();
        //initialize main menu

        GLCD_Clear(Black);
        GLCD_SetBackColor(Black);
        GLCD_SetTextColor(Yellow);
        GLCD_DisplayString(0, 0, __FI, "Nyle's Media Center ");
        GLCD_DisplayString(1, 0, __FI, "Gallery       <--");
        GLCD_DisplayString(2, 0, __FI, "MP3");
        GLCD_DisplayString(3, 0, __FI, "Alien Invaders");

        alienY_timer = osTimerCreate(osTimer(timerY_handle), osTimerPeriodic, (void
*)0);
        alienX_timer = osTimerCreate(osTimer(timerX_handle), osTimerPeriodic, (void
*)0);
```

```
  // create 'thread' functions that start executing,
  // example: tid_name = osThreadCreate (osThread(name), NULL);
        tidJS = osThreadCreate(osThread(threadJS), NULL);



  osKernelStart ();                     // start thread execution
        osDelay(osWaitForever);
        for(;;);
}
```

RTX Config Screenshot and Code



```
/*----------------------------------------------------------------------------
 *      CMSIS-RTOS  -  RTX
 *----------------------------------------------------------------------------
 *      Name:    RTX_Conf_CM.C
 *      Purpose: Configuration of CMSIS RTX Kernel for Cortex-M
 *      Rev.:    V4.70.1
 *----------------------------------------------------------------------------
 *
 * Copyright (c) 1999-2009 KEIL, 2009-2016 ARM Germany GmbH. All rights reserved.
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Licensed under the Apache License, Version 2.0 (the License); you may
 * not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an AS IS BASIS, WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *---------------------------------------------------------------------------*/

#include "cmsis_os.h"


/*----------------------------------------------------------------------------
 *      RTX User configuration part BEGIN
 *---------------------------------------------------------------------------*/

//-------- <<< Use Configuration Wizard in Context Menu >>> -----------------
//
// <h>Thread Configuration
// =======================
//
//   <o>Number of concurrent running user threads <1-250>
//   <i> Defines max. number of user threads that will run at the same time.
//   <i> Default: 6
#ifndef OS_TASKCNT
 #define OS_TASKCNT     6
#endif

//   <o>Default Thread stack size [bytes] <64-4096:8><#/4>
//   <i> Defines default stack size for threads with osThreadDef stacksz = 0
```

```
//   <i> Default: 200
#ifndef OS_STKSIZE
 #define OS_STKSIZE     200      // this stack size value is in words
#endif

//   <o>Main Thread stack size [bytes] <64-32768:8><#/4>
//   <i> Defines stack size for main thread.
//   <i> Default: 200
#ifndef OS_MAINSTKSIZE
 #define OS_MAINSTKSIZE 200      // this stack size value is in words
#endif

//   <o>Number of threads with user-provided stack size <0-250>
//   <i> Defines the number of threads with user-provided stack size.
//   <i> Default: 0
#ifndef OS_PRIVCNT
 #define OS_PRIVCNT     0
#endif

//   <o>Total stack size [bytes] for threads with user-provided stack size
<0-1048576:8><#/4>
//   <i> Defines the combined stack size for threads with user-provided stack size.
//   <i> Default: 0
#ifndef OS_PRIVSTKSIZE
 #define OS_PRIVSTKSIZE 0       // this stack size value is in words
#endif

//   <q>Stack overflow checking
//   <i> Enable stack overflow checks at thread switch.
//   <i> Enabling this option increases slightly the execution time of a thread switch.
#ifndef OS_STKCHECK
 #define OS_STKCHECK    1
#endif

//   <q>Stack usage watermark
//   <i> Initialize thread stack with watermark pattern for analyzing stack usage
(current/maximum) in System and Thread Viewer.
//   <i> Enabling this option increases significantly the execution time of osThreadCreate.
#ifndef OS_STKINIT
#define OS_STKINIT      0
#endif

//   <o>Processor mode for thread execution
//     <0=> Unprivileged mode
//     <1=> Privileged mode
//   <i> Default: Privileged mode
```

```
#ifndef OS_RUNPRIV
 #define OS_RUNPRIV     1
#endif

// </h>

// <h>RTX Kernel Timer Tick Configuration
// ======================================
//   <q> Use Cortex-M SysTick timer as RTX Kernel Timer
//   <i> Cortex-M processors provide in most cases a SysTick timer that can be used as
//   <i> as time-base for RTX.
#ifndef OS_SYSTICK
 #define OS_SYSTICK     1
#endif
//
//   <o>RTOS Kernel Timer input clock frequency [Hz] <1-1000000000>
//   <i> Defines the input frequency of the RTOS Kernel Timer.
//   <i> When the Cortex-M SysTick timer is used, the input clock
//   <i> is on most systems identical with the core clock.
#ifndef OS_CLOCK
 #define OS_CLOCK       10000000
#endif

//   <o>RTX Timer tick interval value [us] <1-1000000>
//   <i> The RTX Timer tick interval value is used to calculate timeout values.
//   <i> When the Cortex-M SysTick timer is enabled, the value also configures the SysTick
timer.
//   <i> Default: 1000  (1ms)
#ifndef OS_TICK
 #define OS_TICK        10000
#endif

// </h>

// <h>System Configuration
// =======================
//
// <e>Round-Robin Thread switching
// ===============================
//
// <i> Enables Round-Robin Thread switching.
#ifndef OS_ROBIN
 #define OS_ROBIN       0
#endif

//   <o>Round-Robin Timeout [ticks] <1-1000>
```

```
//   <i> Defines how long a thread will execute before a thread switch.
//   <i> Default: 5
#ifndef OS_ROBINTOUT
 #define OS_ROBINTOUT   5
#endif

// </e>

// <e>User Timers
// ==============
//   <i> Enables user Timers
#ifndef OS_TIMERS
 #define OS_TIMERS      1
#endif

//   <o>Timer Thread Priority
//                   <1=> Low
//    <2=> Below Normal  <3=> Normal  <4=> Above Normal
//                   <5=> High
//                   <6=> Realtime (highest)
//   <i> Defines priority for Timer Thread
//   <i> Default: High
#ifndef OS_TIMERPRIO
 #define OS_TIMERPRIO   5
#endif

//   <o>Timer Thread stack size [bytes] <64-4096:8><#/4>
//   <i> Defines stack size for Timer thread.
//   <i> Default: 200
#ifndef OS_TIMERSTKSZ
 #define OS_TIMERSTKSZ  50     // this stack size value is in words
#endif

//   <o>Timer Callback Queue size <1-32>
//   <i> Number of concurrent active timer callback functions.
//   <i> Default: 4
#ifndef OS_TIMERCBQS
 #define OS_TIMERCBQS   4
#endif

// </e>

//   <o>ISR FIFO Queue size<4=>   4 entries  <8=>   8 entries
//                   <12=> 12 entries  <16=> 16 entries
//                   <24=> 24 entries  <32=> 32 entries
//                   <48=> 48 entries  <64=> 64 entries
```

```
//                     <96=> 96 entries
//   <i> ISR functions store requests to this buffer,
//   <i> when they are called from the interrupt handler.
//   <i> Default: 16 entries
#ifndef OS_FIFOSZ
 #define OS_FIFOSZ      16
#endif

// </h>

//------------- <<< end of configuration section >>> ----------------------

// Standard library system mutexes
// ===============================
//  Define max. number system mutexes that are used to protect
//  the arm standard runtime library. For microlib they are not used.
#ifndef OS_MUTEXCNT
 #define OS_MUTEXCNT    8
#endif

/*----------------------------------------------------------------------------
 *      RTX User configuration part END
 *---------------------------------------------------------------------------*/

#define OS_TRV          ((uint32_t)(((double)OS_CLOCK*(double)OS_TICK)/1E6)-1)


/*----------------------------------------------------------------------------
 *      Global Functions
 *---------------------------------------------------------------------------*/

/*--------------------------- os_idle_demon ---------------------------------*/

/// \brief The idle demon is running when no other thread is ready to run
void os_idle_demon (void) {

  for (;;) {
    /* HERE: include optional user code to be executed when no thread runs.*/
  }
}

#if (OS_SYSTICK == 0)   // Functions for alternative timer as RTX kernel timer

/*--------------------------- os_tick_init ----------------------------------*/

/// \brief Initializes an alternative hardware timer as RTX kernel timer
```

```
/// \return                     IRQ number of the alternative hardware timer
int os_tick_init (void) {
  return (-1);  /* Return IRQ number of timer (0..239) */
}

/*---------------------------- os_tick_val ---------------------------------*/

/// \brief Get alternative hardware timer's current value (0 .. OS_TRV)
/// \return                     Current value of the alternative hardware timer
uint32_t os_tick_val (void) {
  return (0);
}

/*---------------------------- os_tick_ovf ---------------------------------*/

/// \brief Get alternative hardware timer's  overflow flag
/// \return                     Overflow flag\n
///                           - 1 : overflow
///                           - 0 : no overflow
uint32_t os_tick_ovf (void) {
  return (0);
}

/*---------------------------- os_tick_irqack ------------------------------*/

/// \brief Acknowledge alternative hardware timer interrupt
void os_tick_irqack (void) {
  /* ... */
}

#endif   // (OS_SYSTICK == 0)

/*---------------------------- os_error ------------------------------------*/

/* OS Error Codes */
#define OS_ERROR_STACK_OVF      1
#define OS_ERROR_FIFO_OVF       2
#define OS_ERROR_MBX_OVF        3
#define OS_ERROR_TIMER_OVF      4

extern osThreadId svcThreadGetId (void);

/// \brief Called when a runtime error is detected
/// \param[in]   error_code   actual error code that has been detected
void os_error (uint32_t error_code) {
```

```
  /* HERE: include optional code to be executed on runtime error. */
  switch (error_code) {
    case OS_ERROR_STACK_OVF:
      /* Stack overflow detected for the currently running task. */
      /* Thread can be identified by calling svcThreadGetId().   */
      break;
    case OS_ERROR_FIFO_OVF:
      /* ISR FIFO Queue buffer overflow detected. */
      break;
    case OS_ERROR_MBX_OVF:
      /* Mailbox overflow detected. */
      break;
    case OS_ERROR_TIMER_OVF:
      /* User Timer Callback Queue overflow detected. */
      break;
    default:
      break;
  }
  for (;;);
}


/*----------------------------------------------------------------------------
 *      RTX Configuration Functions
 *----------------------------------------------------------------------------*/

#include "RTX_CM_lib.h"

/*----------------------------------------------------------------------------
 * end of file
 *----------------------------------------------------------------------------*/
```