Laboratory 4

ENGG4420: Real-Time Systems Design

Instructor:
Dr.Radu Muresan

Group 17: Wed-8:30 Section

Bilal Ayyache: 0988616
Robert Mackenzie Beggs: 0819747

Lab Start Date: November 11th, 2020
Lab End Date: December 4th, 2020

Contents

1 Introduction

1.1 Problem Description

Tasks can be categorized via use of the discrete labels of periodic or aperiodic; periodic tasks are interpreted as an infinite series of identical jobs with consistent deadlines, while aperiodic tasks are sporadic and possess no real deadline [5]. A challenge associated with real time operating systems is simultaneously executing different tasks with varying periods, as in addition to contending with the possibility of placing high computational load on the processor, the operating system must execute the given tasks at the proper intervals, ensuring that deadline constraints are met. By considering the period of given tasks, this work aims to accommodate and expand previous efforts via utilization of real time operating system kernel and objects, to the end of producing an intruder detection system.

1.2 System Requirements

Regulating periodic tasks is important to ensure system health as missing multiple deadlines can result in catastrophic failure of a given system. To ensure the periodicity of the given system, deadlines constraints must be adhered to. The deadline of a given task is characterized as hard, firm or soft by evaluating the impact of missing that task to overall system health. To successfully engage in scheduling multiple tasks to run with diverse temporal characteristics, priority must be determined and synchronization must occur. Priority is determined by evaluating the deadline characteristics of a given task. Synchronization is accomplished via real time objects such as queues, priority, mutex and semaphores, used to simultaneously update a given task with the corresponding task or interrupt service routine. These constraints are further complicated via use of periphery devices, as the temporal constraints of extraneous hardware also needs to be considered. A working example in FreeRTOS describing the process required to successfully implement display tasks was provided. A complete system, capable of running 3 main tasks, and 3 additional tasks, was required; both periodic and aperiodic tasks were included. The definition of these tasks is presented within Section 1.3. Futhermore, integration of periphery devices such as an LCD display, motion sensor and camera was required. Through implementation of an accurate model, knowledge of this hardware, FreeRTOS kernel and objects as well as scheduling was developed.

1.3 Tasks Description

In order to effectively implement the intruder detection system, the system was delineated into 3 major tasks:

- A motion sensor task which detects an intruder and triggers an interrupt
- A camera task which acquires a frame from streaming video at a given period
- An alert task which notifies the user given an interrupt

In addition to these required tasks, additional tasks were proposed:

- A button task which returns the system to the default state
- A GUI task is proposed, in which the LCD display is utilized to display pertinent status messages to the user
- A comm task is utilized in order to facilitate communication between the board and computer

2 Background

2.1 Evaluation Board

This system was implemented via use of the STM32F429I-DISC1 (STM32f4) as an ST evaluation board, which is characterized by the inclusion of the STMSTM32F429 microcontroller [1].

2.2 True Studio

True Studio is an RTOS aware IDE, explicit created for STM32 devices and development boards, that allows for the user to examine the condition of RTOS objects such as semaphores, and queues during control flow, easing the debugging and development process [8].

2.3 FreeRTOS

FreeRTOS is a flexibly real time operating system for microcontrollers that is developed in conjunction with chip manufacturers and is exhaustively documented [2]. It was chosen for implementation in this project due to its well documented, intuitive API [6].

2.4 PIR Motion Sensor

The PIR Directional Infrared Radial Sensor D203S (PIR Motion Sensor) was used to periodically collect data from the environment. The PIR motion sensor was selected due to its compact size, efficiency and durability [7].

2.5 Arducam

The ArduCAM-Mini-5MP-Plus OV5642 Camera Module is a small form factor high definition camera that integrates with an open source code library and a large array of hardware platforms that have SPI and I2C interfaces [4]. It was used in implementation for the camera task as introduced in Section 1.3.

2.6 Queues

In order to maintain mutual exclusion, queues categorized by priority as opposed to simpler first in first out (FIFO) are utilized within a Real Time paradigm [5].

2.7 Semaphores

Mutual exclusion can be summarized as the act of governing access to common resources to such a degree that no more than a single task can access a given resource at a given moment in time; this is achieved via synchronization. Semaphores are kernel objects in a real time environment utilized to inform tasks of the state of other tasks via alerts or ISRs; they are used to perform the sychonization. Binary semaphores are used to synchronize alerts to a given task, ensuring mutual exclusion. Counting semaphores are utilized to keep an accurate approximation of the amount of times a resource has been utilized in a given range of time [5].

2.8 Mutual Exclusion Semaphores

Given a shared resource, the sequence in which an element from a set of tasks is executed is referred to as priority. Under healthy operating conditions, a task with a higher priority then another should succeed it in terms of execution order; the case in which lower priority tasks precede task that were assigned a higher priority is known as priority inversion. Mutual exclusion semaphores (mutex) are a subset of binary semaphores that restrict unbounded priority inversion [5].

3 Implementation

3.1 Main

Main is responsible for initializing and configuring desired system settings. This was accomplished via invoking several functional calls from the associated header library file. Given that 'stm32f4xx_hal.h' has been included, calling 'HAL_Init' initialized the HAL library. HAL is a hardware abstraction layer, that allows for high-level APIs to be utilized and flash memory and data caches to be automatically configured [2, 6]. Following initialization of HAL, the system clock and all peripherals were configured via utilization of 'SystemClock_Config()' as well as the code block displayed in Figure 1 respectively. Finally, GPIO pins are configured via 'GPIO_InitStruct'.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA2D_Init();
MX_FMC_Init();
MX_LTDC_Init();
MX_SPI5_Init();
MX_SPI4_Init();
MX_I2C3_Init();

/* init code for USB_DEVICE */
MX_USB_DEVICE_Init();
```

Figure 1: Initialization of all required peripherals

All functions provided by STMicroelectronics for STM32 microcontrollers related to periphery initialization have the prefix 'MX', as depicted in Figure 1 [1, 3]. 'BSP_LCD' functions were then used to prepare the LCD to display the GUI. Finally, the FreeRTOS kernel was initialized and the scheduler was started via use of 'MX_FREERTOS_Init()' and 'osKernelStart()' respectively. 'MX_FREERTOS_Init()' is used to define threads for each of the tasks specified in Section 1.3. This is accomplished via use of 'osThreadDef' and 'osThreadCreate'. Given a name, priority, number of instances required and stack size, a thread matching those attributes is defined via use of 'osThreadDef'. Given a thread definition from 'osThreadDef' and a NULL pointer that acts as the start argument, 'osThreadCreate' generates a thread in READY state [6]. Similar process are repeated for the creation of queues and semaphores. 'osMutexDef' is used to define a mutex object, and 'osMutexCreate' is utilized to create the required object. Finally, 'osMessageQDef' and 'osMessageCreate' are analogous to 'osThreadDef' and 'osThreadCreate', with the purpose of defining and creating the Comm, Camera and Alert queues required for the specified tasks [6].

3.2 Motion Sensor Task

The purpose of the motion sensor task is to detect the presence of an intruder. Given the status of GPIO pin GPIOE and GPIO_PIN_4, an equality check is performed using 'HAL_GPIO_ReadPin'; a function that determines the status of the provided input port pin. In the case that the specified pin is equal to 0, a message is printed to console indicating that an intruder has been detected [6]. Additionally, given a queue id, an integer representing a status flag, and a timeout value, 'osMessagePut' is used to facilitate communication with Camera and Alert Queues; since an intruder has been detected the flag is set to true (value 1). Else, if an intruder is not detected, a message is printed in console indicating that no movement has been detected and 'osMessagePut' is used to share the status flag with Camera and Alert Queues; in this case the status remains false (value 0). The output to console is depicted in Figure 2.

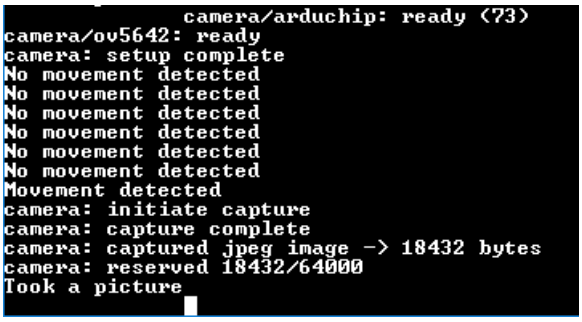

Figure 2: Demonstrating output when an intruder has and has not been detected.

This process is repeated indefinitely with a 1000 ms delay between cycles. The delay is accomplished via use of 'osDelay'.

3.3 Camera Task

The primary purpose of the camera task is to take a picture of movement if an intruder has been detected. In the variable declarations, the camera on flag was denoted to be off by default, an integer was set to indicate when photos should be taken is defined and the status of the camera was defined as an osEvent. osEvents are datatypes that allow for the status of RTOS functions to be easily described. An infinitely repeating cycle is then entered. If it is the first pass of the cycle, the camera is set up and a flag is set indicating that it is on. The status camera queue set in the motion sensor task is then evaluated, through use of 'osMessageGet'. Given a queue id as well as a timeout value, 'osMessageGet' returns the event information in the queue i.e. the status flag of whether an intruder has been detected generated via the Motion Sensor Task. This event information is converted to an integer, and stored as a flag that indicates whether or not a picture should be taken and then is evaluated. If an intruder has been detected, 'camera_initiate_capture' is called and a message is printed to console indicating that a picture has been taken; the flag that indicates a picture should be taken is reset. The process repeats indefinitely with a delay of 100 ms between cycles.

3.3.1 Camera Functions

Several provided functions were utilized to aid in making the camera task function effectively. Camera setup is accomplished via a three step process corresponding to control flow reaching 'arduchip_detect, 'arducam_exit_standby' and 'ov5642_detect'; these functions are predefined for the purpose of detecting and initializing the Arduchip interface, guaranteeing that the camera is powered on, and that the internal sensor has been detected respectively. In the case that all of these tasks successfully complete without error, a message is printed indicating that setup is complete. Otherwise a message is printed indicating the stage of the setup that failed.

In order to initiate the capture of a frame, the status of camera is checked; if the camera has not been properly configured a message is printed to console. If the capture has not begun, a different error message prints. While the capture is ongoing, status is checked to see if the capture has timed out; if it has an error message is displayed via console. Finally, a message indicating that the capture has completed is output on console. Additional functions entitled 'get_pixel' and 'display_bitmap' were utilized to capture images.

3.4 Alert Task

The purpose of the alert task is to indicate to the user that an intruder has been detected. Requisite to this function was the initialization of variables that serve as status flags. An osEvent, 'alertStatus', acts as a flag for event information received from a queue, 'sendAlert' represented the integer equivalent flag. Finally, 'blink' was initialized as a flag to indicate the status of an LED. Given the id to the alert queue as well as 'osWaitForever', a predefined macro function indicating that a function should never time out, 'osMessageGet' returns the event information regulating the status communicated by the motion sensor [6]. The event information is held in 'alertStatus', before being converted to an integer, and being stored in 'sendAlert'. 'sendAlert' is then compared to 1; in the case that it is equal to 1, the microcontroller displays the text "Intruder detected" on the lcd screen via 'BSP_LCD_DisplayStringAtLine'. Furthermore, the blinking of an LED light is triggered via 'HAL_GPIO_WritePin'. 'HAL_GPIO_WritePin' is a predefined

function that determines the value of a selected data port bit given an input; it is given the port address corresponding to the LED and an integer that indicates that the LED should blink [6]. This process is repeated indefinitely with a delay of 400 ms between cycles.

3.5 Button Task

Given the address of the peripherals corresponding to the button on the microcontroller, a comparison is performed using 'HAL_GPIO_ReadPin' to determine whether a button has been pressed by determining the value of the input port pin. In the case that the pin that value of the input pin is equivalent to that associated with the button, a message indicating that the button has been pressed, and the number of times that it has been pressed is printed to console and button counter is incremented. The in the GUI display indicating that an intruder has been detected is cleared via use of 'BSP_LCD_ClearStringLine'. Additionally, the status of the LED is reset to no longer blink via 'HAL_GPIO_WritePin'. This process is repeated indefinitely with a delay of 10 ms between cycles.

3.6 Comm Task

The purpose of the Comm Task is to communicate between the microcontroller and the computer. Given the appropriate queue id, 'osMessageGet' returns the value from the comm queue. Control flow is paused to wait for a mutex to become available via 'osMutexWait'. After one is available the message in the queue is copied to a string and the mutex is released via 'osMutexRelease'. Finally, a message indicating that the comm task was received as well as the message is printed to console. This cyclic process repeats indefinitely.

3.7 GUI Task

The primary purpose of the GUI task is to display persistent elements of the GUI on the LCD, through use of 'BSP_LCD' functions. The text "ENGG4420 Project" and a horizontal red line are continuously displayed, and their colour is configured. This process is persistent with a delay of 200 clock ticks between cycles. The GUI is visible in Figure 3

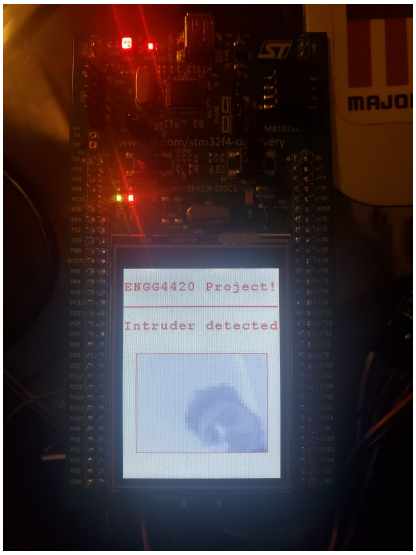

Figure 3: The GUI illustrating that an intruder has been detected

4 Conclusion

Contending with varying periodicity is a challenge in a real time environment. The purpose of this experiment was to demonstrate fluency in FreeRTOS and knowledge of real time objects; this was accomplished via implementation of an intruder detection system. In this system, the intruder was discerned via use of a motion sensor signal, that functioned equivalently to an interrupt service routine. Upon triggering of the ISR,streaming video frames were captured at a prescribed intervals and an alert was provided to the user via flashing LEDs, messaging on the LCD display and messaging to a mobile device. Finally, a button press is used to indicate that the alerts have been received; at this point the system returns to the 'wait' state. Implementation of this project highlighted the importance of task synchronization and communication in a real time environment. Further integration of real time systems with the Internet of Things is left for future experimentation.

References

[1] 2020. URL: http://www.disca.upv.es/aperles/arm_cortex_m3/llibre/st/STM32F439xx_User_Manual/group__gpio__exported__functions__group2.html.

[2] 2020. URL: https://www.freertos.org/RTOS.html.

[3] 2020. URL: https://www.st.com/resource/en/user_manual/dm00104712-stm32cubemx-for-stm32-configuration-and-initialization-c-code-generation-stmicroelectronics.pdf.

[4] *ArduCAM-Mini-5MP-Plus OV5642 Camera Module: 5MP SPI Camera User Guide*. 2016.

[5] Giorgio C Buttazzo. *Hard real-time computing systems*. 3rd ed. Springer, 2011.

[6] *CMSIS RTOS Documentation: Real-Time Operating System: API and RTX Reference Implementation*. 2020.

[7] *Phi Robotics: PIR Sensor Product Manual Version 1.0*. 2020.

[8] *True Studio Documentation*. 2020. URL: https://atollic.com/truestudio/.

A Appendix

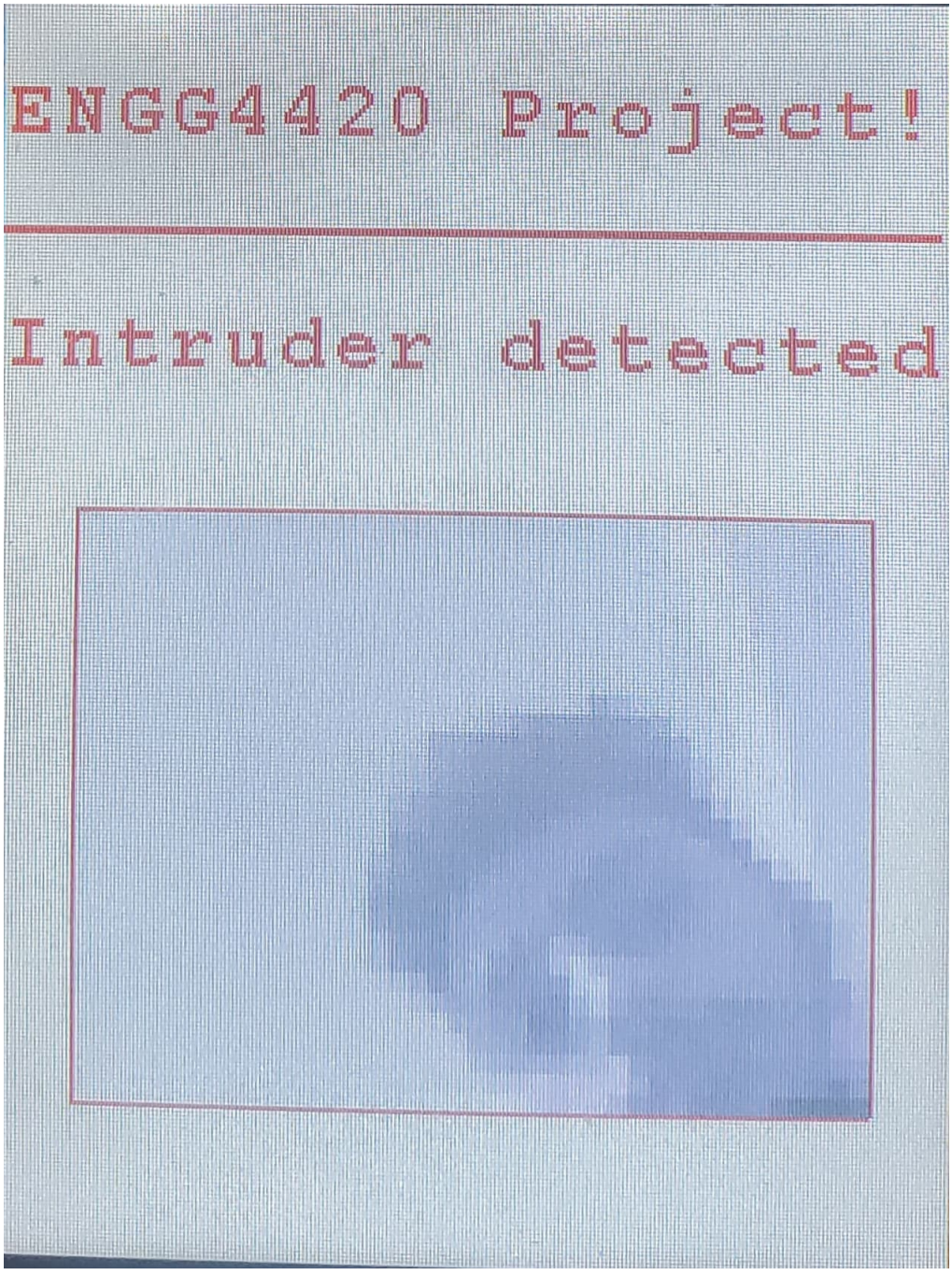

Figure 4:

A.1 main.c

```
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * Copyright (c) 2018 STMicroelectronics International N.V.
  * All rights reserved.
  *
  * Redistribution and use in source and binary forms, with or without
  * modification, are permitted, provided that the following conditions are met:
  *
  * 1. Redistribution of source code must retain the above copyright notice,
  *    this list of conditions and the following disclaimer.
  * 2. Redistributions in binary form must reproduce the above copyright notice,
  *    this list of conditions and the following disclaimer in the documentation
  *    and/or other materials provided with the distribution.
  * 3. Neither the name of STMicroelectronics nor the names of other
  *    contributors to this software may be used to endorse or promote products
  *    derived from this software without specific written permission.
  * 4. This software, including modifications and/or derivative works of this
  *    software, must execute solely and exclusively on microcontroller or
  *    microprocessor devices manufactured by or for STMicroelectronics.
  * 5. Redistribution and use of this software other than as permitted under
  *    this license is void and will automatically terminate your rights under
  *    this license.
  *
  * THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT
  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
  * PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY
  * RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT
  * SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
```

```
  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
  * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
  * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "cmsis_os.h"
#include "dma2d.h"
#include "i2c.h"
#include "ltdc.h"
#include "spi.h"
#include "usb_device.h"
#include "gpio.h"
#include "fmc.h"

#include "stm32f429i_discovery_lcd.h"
#include "usbd_cdc_if.h"


/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void MX_FREERTOS_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
#ifdef __GNUC__
/* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
int __io_putchar(int ch);
int _write(int file,char *ptr, int len)
{
 int DataIdx;
 for(DataIdx= 0; DataIdx< len; DataIdx++)
 {
 __io_putchar(*ptr++);
```

```
 }
return len;
}
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
PUTCHAR_PROTOTYPE
{
        while(CDC_Transmit_HS((uint8_t *)&ch, 1) != USBD_OK);
        return ch;
}


/**
  * @brief  The application entry point.
  *
  * @retval None
  */
int main(void)
{


  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();


  /* Configure the system clock */
  SystemClock_Config();


  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_LTDC_Init();
  MX_SPI5_Init();
  MX_SPI4_Init();
  MX_I2C3_Init();
```

```
  /* init code for USB_DEVICE */
  MX_USB_DEVICE_Init();

  /* USER CODE BEGIN 2 */
  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LCD_BACKGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_LayerDefaultInit(LCD_FOREGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_SelectLayer(LCD_FOREGROUND_LAYER);
  BSP_LCD_DisplayOn();
  BSP_LCD_Clear(LCD_COLOR_WHITE);
  /* USER CODE END 2 */

  /* Call init function for freertos objects (in freertos.c) */
  MX_FREERTOS_Init();

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {}


}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
```

```
__HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /**Initializes the CPU, AHB and APB busses clocks
  */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 4;
RCC_OscInitStruct.PLL.PLLN = 168;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

  /**Initializes the CPU, AHB and APB busses clocks
  */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_LTDC;
PeriphClkInitStruct.PLLSAI.PLLSAIN = 50;
PeriphClkInitStruct.PLLSAI.PLLSAIR = 2;
PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_2;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}
```

```
    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM1 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM1) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  file: The file name as string.
```

```
  * @param  line: The line in file as a number.
  * @retval None
  */
void _Error_Handler(char *file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

A.2 gpio.c

```
/**
  ******************************************************************************
  * File Name          : gpio.c
  * Description        : This file provides code for the configuration
  *                      of all used GPIO pins.
  ******************************************************************************
  * This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * Copyright (c) 2018 STMicroelectronics International N.V.
  * All rights reserved.
  *
  * Redistribution and use in source and binary forms, with or without
  * modification, are permitted, provided that the following conditions are met:
  *
  * 1. Redistribution of source code must retain the above copyright notice,
  *    this list of conditions and the following disclaimer.
  * 2. Redistributions in binary form must reproduce the above copyright notice,
  *    this list of conditions and the following disclaimer in the documentation
  *    and/or other materials provided with the distribution.
  * 3. Neither the name of STMicroelectronics nor the names of other
  *    contributors to this software may be used to endorse or promote products
  *    derived from this software without specific written permission.
  * 4. This software, including modifications and/or derivative works of this
  *    software, must execute solely and exclusively on microcontroller or
  *    microprocessor devices manufactured by or for STMicroelectronics.
  * 5. Redistribution and use of this software other than as permitted under
  *    this license is void and will automatically terminate your rights under
  *    this license.
  *
  * THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT
  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
  * PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY
  * RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT
  * SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
```

```
  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
  * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
  * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "gpio.h"


/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/*----------------------------------------------------------------------------*/
/* Configure GPIO                                                             */
/*----------------------------------------------------------------------------*/
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
     PA9   ------> USART1_TX
     PA10   ------> USART1_RX
*/
void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
```

```
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOF_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
__HAL_RCC_GPIOG_CLK_ENABLE();
__HAL_RCC_GPIOD_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(SPI4_SS_GPIO_Port, SPI4_SS_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOC, NCS_MEMS_SPI_Pin|CSX_Pin|OTG_FS_PSO_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(ACP_RST_GPIO_Port, ACP_RST_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOD, RDX_Pin|WRX_DCX_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOG, LD3_Pin|LD4_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin : PtPin */
GPIO_InitStruct.Pin = SPI4_SS_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(SPI4_SS_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : PCPin PCPin PCPin */
GPIO_InitStruct.Pin = NCS_MEMS_SPI_Pin|CSX_Pin|OTG_FS_PSO_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : PAPin PAPin PAPin PAPin */
GPIO_InitStruct.Pin = B1_Pin|MEMS_INT1_Pin|MEMS_INT2_Pin|TP_INT1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = ACP_RST_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(ACP_RST_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = OTG_FS_OC_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(OTG_FS_OC_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = BOOT1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = TE_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(TE_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PDPin PDPin */
  GPIO_InitStruct.Pin = RDX_Pin|WRX_DCX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : PAPin PAPin */
  GPIO_InitStruct.Pin = STLINK_RX_Pin|STLINK_TX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : PGPin PGPin */
  GPIO_InitStruct.Pin = LD3_Pin|LD4_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

  /*Configure GPIO pin : PF1 Pin */
  GPIO_InitStruct.Pin = GPIO_PIN_4;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

}

// TODO: set the interrupt handle (Optional)
//       hint: you can use flag event here.


/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

A.3 freertos.c

```
/* Includes ------------------------------------------------------------------*/
#include "FreeRTOS.h"
#include "task.h"
#include "cmsis_os.h"


/* USER CODE BEGIN Includes */
#include "usbd_cdc_if.h"
#include "stm32f429i_discovery.h"
#include "stm32f429i_discovery_lcd.h"
#include "spi.h"
#include "i2c.h"
#include "camera.h"
/* USER CODE END Includes */

/* Variables -----------------------------------------------------------------*/
osMessageQId  CameraQueueHandle;
osMessageQId  AlertQueueHandle;
osMessageQId  CommQueueHandle;
osMutexId     dataMutexHandle;


/* USER CODE BEGIN Variables */
uint8_t  RxData[256];
uint32_t data_received;
uint8_t  Str4Display[50];

osThreadId TaskGUIHandle;
osThreadId TaskCOMMHandle;
osThreadId TaskBTNHandle;
osThreadId TaskMotionHandle;
osThreadId TaskCameraHandle;
osThreadId TaskAlertHandle;



typedef struct
{
        uint8_t Value[10];
```

```
        uint8_t Source;
}data;

data DataToSend =  {"Hello\0", 1};
data DataVCP    =  {"VCP\0", 2};

/* USER CODE END Variables */



/* Function prototypes -------------------------------------------------------*/

extern void MX_USB_DEVICE_Init(void);
void MX_FREERTOS_Init(void); /* (MISRA C 2004 rule 8.1) */

/* USER CODE BEGIN FunctionPrototypes */
void StartTaskCOMM(void const * argument);
void StartTaskBTN(void const * argument);
void StartTaskGUI(void const * argument);
void sensor_task(void const* argument);
void camera_task(void const* argument);
void intruderAlert_task(void const* argument);
/* USER CODE END FunctionPrototypes */

/* Hook prototypes */

/* Init FreeRTOS */

void MX_FREERTOS_Init(void) {

        /************** TASK DEFINITIONS **************/

        osThreadDef(TaskCOMM, StartTaskCOMM, osPriorityHigh, 0, 128);
        TaskCOMMHandle = osThreadCreate(osThread(TaskCOMM), NULL);

        osThreadDef(TaskBTN, StartTaskBTN, osPriorityLow, 0, 128);
        TaskBTNHandle = osThreadCreate(osThread(TaskBTN), NULL);

        osThreadDef(TaskGUI, StartTaskGUI, osPriorityNormal, 0, 128);
        TaskGUIHandle = osThreadCreate(osThread(TaskGUI), NULL);
```

```
        osThreadDef(Task1, sensor_task, osPriorityHigh, 0, 128);
        TaskMotionHandle = osThreadCreate(osThread(Task1), NULL);

        osThreadDef(Task2, camera_task, osPriorityNormal, 0, 128);
        TaskCameraHandle = osThreadCreate(osThread(Task2), NULL);

        osThreadDef(Task3, intruderAlert_task, osPriorityAboveNormal, 0, 128);
        TaskAlertHandle = osThreadCreate(osThread(Task3), NULL);

        /************** QUEUES AND SEMAPHORES GIVEN TO US **************/

        osMutexDef(dataMutex);
        dataMutexHandle = osMutexCreate(osMutex(dataMutex));

        osMessageQDef(CommQueue, 1, &DataVCP);
        CommQueueHandle = osMessageCreate(osMessageQ(CommQueue), NULL);

        /********* OUR QUEUES AND SEMAPHORES *********/

        osMessageQDef(CameraQueue, 1, int);
        CameraQueueHandle = osMessageCreate(osMessageQ(CameraQueue), NULL);

        osMessageQDef(AlertQueue, 1, int);
        AlertQueueHandle = osMessageCreate(osMessageQ(AlertQueue), NULL);
}


void StartTaskCOMM(void const * argument)
{
  osEvent vcpValue;

  while(1)
  {
          vcpValue = osMessageGet(CommQueueHandle, osWaitForever);
          osMutexWait(dataMutexHandle, 0);
          memcpy(Str4Display,(char *)(((data *)vcpValue.value.p)->Value), data_received+1);
          osMutexRelease(dataMutexHandle);
          printf("CommTask received: %s\n\r", Str4Display);
  }
}
```

```
void sensor_task(void const* argument)
{
//        int counter = 0;

        while(1)
        {
                osDelay(1000);

                if(HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_4) == 0)                // read OUT pin status
                {

                        printf("Movement detected\n\r");
                        osMessagePut(CameraQueueHandle, 1, osWaitForever);
                        osMessagePut(AlertQueueHandle, 1, osWaitForever);

                }
                else
                {
                        printf("No movement detected\n\r");
                        osMessagePut(CameraQueueHandle, 0, osWaitForever);
                        osMessagePut(AlertQueueHandle, 0, osWaitForever);
                }
        }
}

void camera_task(void const* argument)
{
    osEvent cameraStatus;
    int cameraOn = 0;
        int takePicture;

        while(1)
        {
                if(!cameraOn)
                {
                        camera_setup();
                        cameraOn = 1;
                }

                cameraStatus = osMessageGet(CameraQueueHandle, osWaitForever);
                takePicture = (int)(cameraStatus.value.p);
```

```
                if(takePicture == 1)
                {
                        camera_initiate_capture();
                        printf("Took a picture\n\r");
                        takePicture = 0;
                }

                osDelay(100);
        }
}

void intruderAlert_task(void const* argument)
{
    osEvent alertStatus;
    int sendAlert;
    int blink = 0;

        while(1)
        {
                alertStatus = osMessageGet(AlertQueueHandle, osWaitForever);
                sendAlert = (int)(alertStatus.value.p);

                blink = !blink;

                if(sendAlert == 1)
                {
                        BSP_LCD_DisplayStringAtLine(4, (uint8_t *)"Intruder detected");
                        HAL_GPIO_WritePin(GPIOG,LD3_Pin|LD4_Pin, blink);
                }

                osDelay(400);
        }
}

void StartTaskGUI(void const * argument)
{


  while(1)
  {
```

```
        BSP_LCD_SetTextColor(LCD_COLOR_RED);
        BSP_LCD_DisplayStringAtLine(1, (uint8_t *)"ENG4420 Project");
        BSP_LCD_SetTextColor(LCD_COLOR_RED);
        BSP_LCD_DisplayStringAtLine(2, (uint8_t *)"_________________");
        BSP_LCD_SetTextColor(LCD_COLOR_RED);

        osDelay(200);
  }
}

void StartTaskBTN(void const * argument)
{
  uint8_t count_press = 0;

  while(1)
  {
        if(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN) == GPIO_PIN_SET)
        {
                printf("TaskBTN Button Pressed : %i times \n\r", count_press);
                count_press++;

                BSP_LCD_ClearStringLine(4);
                HAL_GPIO_WritePin(GPIOG,LD3_Pin|LD4_Pin, 0);

                while(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN)==GPIO_PIN_SET);
        }

        osDelay(10);
  }
}
```

A.4 camera.c

```
/*
 * camera.c
 *
 *  Created on: Nov 3, 2018
 *      Author: Graham Thoms
 */

#include "camera.h"
#include "arducam.h"

#include "picojpeg_test.h"
#include "util.h"
#include "stlogo.h"
#include "stm32f429i_discovery_lcd.h"
#include "usbd_cdc_if.h"


#define MAX_PIC_SIZE 64000
#define BITMAP_SIZE 3600


static uint32_t captureStart;
uint8_t fifoBuffer[BURST_READ_LENGTH];
uint8_t pic_buffer[MAX_PIC_SIZE];
uint8_t bitmap[BITMAP_SIZE];
int pic_index = 0;

static void camera_get_image();
BaseType_t write_fifo_to_buffer(uint32_t length);

void camera_setup(){

        cameraReady = pdFALSE;
        /**
         * Detect and initialize the Arduchip interface.
         * Ensure that the OV5642 is powered on.
         * Detect and initialize the OV5642 sensor chip.
         */
        if (   arduchip_detect()
```

```
                && arducam_exit_standby()
                && ov5642_detect()
        ) {

                osDelay(100);

                if (!ov5642_configure()) {
                        printf("camera_task: ov5642 configure failed\n\r");
                        return;
                } else {
                        printf("camera: setup complete\n\r");
                        cameraReady = pdTRUE;
                        osDelay(100);
                }
        } else {
                printf("camera: setup failed\n\r");
                cameraReady = pdTRUE;
        }
}

/**
 * Capture an image from the camera.
 */
void camera_initiate_capture(){

        uint8_t done = 0;

        printf("camera: initiate capture\n\r");

        if (!cameraReady) {
                printf("camera: set up camera before capture\n\r");
        }

        /* Initiate an image capture. */
        if (!arduchip_start_capture()) {
                printf("camera: initiate capture failed\n\r");
                return;
        }

        /* wait for capture to be done */
        captureStart = (uint32_t)xTaskGetTickCount();
```

```
        while(!arduchip_capture_done(&done) || !done){

                if ((xTaskGetTickCount() - captureStart) >= CAPTURE_TIMEOUT) {
                        printf("camera: capture timeout\n\r");
                        return;
                }
        }

        printf("camera: capture complete\n\r");

        camera_get_image();

        return;

}

void camera_get_image(){

        /* Determine the FIFO buffer length. */
        uint32_t length = 0;
        if (arduchip_fifo_length(&length) == pdTRUE) {
                printf("camera: captured jpeg image -> %lu bytes\n\r", length);
                write_fifo_to_buffer(length);
        } else {
                printf("camera: get fifo length failed\n\r");
        }

        return;
}



static void get_pixel(int* pDst, const uint8_t *pSrc, int luma_only, int num_comps)
{
   int r, g, b;
   if (num_comps == 1)
   {
      r = g = b = pSrc[0];
   }
   else if (luma_only)
   {
```

```
        const int YR = 19595, YG = 38470, YB = 7471;
        r = g = b = (pSrc[0] * YR + pSrc[1] * YG + pSrc[2] * YB + 32768) / 65536;
    }
    else
    {
        r = pSrc[0]; g = pSrc[1]; b = pSrc[2];
    }
    pDst[0] = r; pDst[1] = g; pDst[2] = b;
}


static void display_bitmap(const uint8_t * image, int width, int height, int comps, int start_x, i
{

        int a[3];

        for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                        get_pixel(a, image + (y * width + x) * comps, 0, comps);
                        uint32_t argb = (0xFF << 24) | (a[0] << 16) | (a[1] << 8) | (a[0]);
                        for (int yy = 0; yy < scale; yy++) {
                                for (int xx = 0; xx < scale; xx++) {
                                        BSP_LCD_DrawPixel(start_x + x * scale + xx, start_y + y *
                                }
                        }
                }
        }
        uint32_t old_col = BSP_LCD_GetTextColor();
        BSP_LCD_SetTextColor(LCD_COLOR_RED);
        BSP_LCD_DrawRect(start_x, start_y, width * scale, height * scale);
        BSP_LCD_SetTextColor(old_col);
}

BaseType_t
write_fifo_to_buffer(uint32_t length)
{
        int width, height, comp;
        uint16_t chunk = 0;
        unsigned int jpeg_size = length*sizeof(uint8_t);

        if (jpeg_size >= MAX_PIC_SIZE) {
```

```
                printf("camera: not enough memory (%d/%d)\n\r", jpeg_size, MAX_PIC_SIZE);
                return pdFALSE;
        } else {
                printf("camera: reserved %d/%d\n\r", jpeg_size, MAX_PIC_SIZE);
        }

        pic_index = 0;
        for (uint16_t i = 0; length > 0; ++i) {

                chunk = MIN(length, BURST_READ_LENGTH);
                arduchip_burst_read(fifoBuffer, chunk);
                for (uint16_t j = 0; j < chunk; j++) {
                        pic_buffer[pic_index] = fifoBuffer[j];
                        pic_index += 1;
                }
                length -= chunk;
        }

        FILE * pic_stream = fmemopen(pic_buffer, jpeg_size, "rb");        //jpg2tga will close thi

        picojpg_test(pic_stream, &width, &height, &comp, bitmap);

        if(width*height*comp == BITMAP_SIZE)
        {
                display_bitmap(bitmap, width, height, comp, 20, 130, 5);
        }

        return pdTRUE;
}
```