**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

-----------------🙢★🙠-----------------



**BÁO CÁO MINI PROJECT**

**MÔN THIẾT KẾ HỆ THỐNG NHÚNG**

***Lớp***: CE224.O12.1

***Giảng viên hướng dẫn***: ThS. Chung Quang Khánh

***Nhóm sinh viên thực hiện***:

1. Đào Phước Tài (50%) 21521391
2. Nguyễn Anh Khôi (50%) 21520301

***TP.Hồ Chí Minh, ngày 21 tháng 12 năm 2023***

1. **MÔ TẢ GAME**
2. Nguyên lý

* Đầu tiên, hệ thống ở trạng thái cân bằng khởi đầu.
* Người chơi sẽ hạ và nâng KIT để thực hiện động tác tâng bóng, khi đó trên màn hình LCD sẽ mô phỏng trạng thái trái bóng được nâng lên và rơi xuống.
* Mục tiêu của người chơi là phải nâng KIT đúng lúc trái bóng rơi xuống, đập vào "mặt vợt" và tâng lên lại, người chơi được tính điểm, đèn xanh chớp 1 lần.
* Nếu người chơi nâng đúng, tùy thuộc vào mức độ chính xác mà bóng sẽ nảy lên với tốc độ khác nhau.
* Nếu người tâng "hụt" thì bóng sẽ rớt và GAME OVER, đèn đỏ được bật và giữ cho tới khi reset.

1. Yêu cầu

A white paper with black text

Description automatically generated

1. **BÀI TẬP**
2. Bài tập 1: Thiết kế mô hình phần cứng cứng thiết cho yêu cầu trên: cần dùng những phần cứng nào, sử dụng các giao thức giao tiếp gì giữa các phần cứng? (15% số điểm).

* Các thành phần phần cứng cần thiết cho thiết kế này bao gồm:
  + Kit STM32F4 Discovery.
  + Màn hình hiển thị LCD.
  + Cảm biến (gyroscope) và nút nhấn (reset).
  + Cáp kết nối USB.
* Các giao thức giao tiếp giữa các phần cứng trên: SPI, I2C, UART.

1. Bài tập 2: Sử dụng RTOS, thiết mô hình phần mềm cho yêu cầu trên: nêu công việc của từng task, luồng xử lý dữ liệu như thế nào? (15% số điểm).

* Phần mềm trên được chia làm 2 task:
  + Task01 có nhiệm vụ xử lý hiển thị LCD, truyền dữ liệu độ cao qua máy tính thông qua USB và xử lý các logic của trò chơi.
  + Task02 có nhiệm vụ lấy dữ liệu gyroscope, lấy dữ liệu độ cao từ Task01 và xử lý logic việc tính điểm, tốc độ tâng bóng.
  + Task01 và Task02 giao tiếp với nhau qua Message Queue: Task01 chuyển thông số độ cao qua phương thức osMessagePut(), Task02 lấy thông số này thông qua osMessageGet() để phục vụ các nhiệm vụ xử lý.

1. Bài tập 3: Thiết kế lưu đồ giải thuật xử lý cho yêu cầu trên? (30% số điểm).

A screenshot of a computer

Description automatically generated

1. Bài tập 4: Hiện thực hệ thống và báo cáo kết quả? (40% số điểm).

* Code hệ thống

|  |
| --- |
| **#include** "main.h"  **#include** "cmsis\_os.h"  **#include** "usb\_device.h"  /\* Private includes ----------------------------------------------------------\*/  /\* USER CODE BEGIN Includes \*/  **#include** "stdio.h"  **#include** "stm32f429i\_discovery\_lcd.h"  **#include** "string.h"  **#include** "usbd\_cdc\_if.h"  **#include** "stm32f429i\_discovery\_gyroscope.h"  /\* USER CODE END Includes \*/  /\* Private variables ---------------------------------------------------------\*/  osThreadId TaskDisplayLCDHandle;  osThreadId TaskGyroScopeHandle;  osMessageQId myQueue01Handle;  /\* USER CODE BEGIN PV \*/  **static** **int** current\_radius = 50;  **static** **int** current\_speed = 10;  **static** **int** fallDown = 0;  **static** **int** gameRunning = 1;  **static** **int** score = 0;  **#define** MAX\_RADIUS\_OF\_HIGH\_SPEED 120  **#define** MAX\_RADIUS\_OF\_MEDIUM\_SPEED 110  **#define** MAX\_RADIUS\_OF\_LOW\_SPEED 100  **#define** MIN\_RADIUS 15  **#define** HIGH\_SPEED 15  **#define** MEDIUM\_SPEED 10  **#define** LOW\_SPEED 5  /\* USER CODE END PV \*/  /\* Private function prototypes -----------------------------------------------\*/  **void** **SystemClock\_Config**(**void**);  **static** **void** **MX\_GPIO\_Init**(**void**);  **void** **StartTask01**(**void** **const** \* argument);  **void** **StartTask02**(**void** **const** \* argument);  /\* USER CODE BEGIN PFP \*/  /\* USER CODE END PFP \*/  /\* Private user code ---------------------------------------------------------\*/  /\* USER CODE BEGIN 0 \*/  /\* USER CODE END 0 \*/  /\*\*  \* @brief The application entry point.  \* @retval int  \*/  **int** **main**(**void**)  {  /\* USER CODE BEGIN 1 \*/  /\* USER CODE END 1 \*/  /\* MCU Configuration--------------------------------------------------------\*/  /\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/  HAL\_Init();  /\* USER CODE BEGIN Init \*/  /\* USER CODE END Init \*/  /\* Configure the system clock \*/  SystemClock\_Config();  /\* USER CODE BEGIN SysInit \*/  /\* USER CODE END SysInit \*/  /\* Initialize all configured peripherals \*/  MX\_GPIO\_Init();  /\* USER CODE BEGIN 2 \*/  /\* USER CODE END 2 \*/  /\* USER CODE BEGIN RTOS\_MUTEX \*/  /\* add mutexes, ... \*/  /\* USER CODE END RTOS\_MUTEX \*/  /\* USER CODE BEGIN RTOS\_SEMAPHORES \*/  /\* add semaphores, ... \*/  /\* USER CODE END RTOS\_SEMAPHORES \*/  /\* USER CODE BEGIN RTOS\_TIMERS \*/  /\* start timers, add new ones, ... \*/  /\* USER CODE END RTOS\_TIMERS \*/  /\* Create the queue(s) \*/  /\* definition and creation of myQueue01 \*/  osMessageQDef(myQueue01, 16, uint8\_t);  myQueue01Handle = osMessageCreate(osMessageQ(myQueue01), NULL);  /\* USER CODE BEGIN RTOS\_QUEUES \*/  /\* add queues, ... \*/  /\* USER CODE END RTOS\_QUEUES \*/  /\* Create the thread(s) \*/  /\* definition and creation of TaskDisplayLCD \*/  osThreadDef(TaskDisplayLCD, StartTask01, *osPriorityNormal*, 0, 256);  TaskDisplayLCDHandle = osThreadCreate(osThread(TaskDisplayLCD), NULL);  /\* definition and creation of TaskGyroScope \*/  osThreadDef(TaskGyroScope, StartTask02, *osPriorityNormal*, 0, 256);  TaskGyroScopeHandle = osThreadCreate(osThread(TaskGyroScope), NULL);  /\* USER CODE BEGIN RTOS\_THREADS \*/  /\* add threads, ... \*/  /\* USER CODE END RTOS\_THREADS \*/  /\* Start scheduler \*/  osKernelStart();  /\* We should never get here as control is now taken by the scheduler \*/  /\* Infinite loop \*/  /\* USER CODE BEGIN WHILE \*/  **while** (1)  {  /\* USER CODE END WHILE \*/  /\* USER CODE BEGIN 3 \*/  }  /\* USER CODE END 3 \*/  }  /\*\*  \* @brief System Clock Configuration  \* @retval None  \*/  **void** **SystemClock\_Config**(**void**)  {  RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};  RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};  /\*\* Configure the main internal regulator output voltage  \*/  \_\_HAL\_RCC\_PWR\_CLK\_ENABLE();  \_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);  /\*\* Initializes the RCC Oscillators according to the specified parameters  \* in the RCC\_OscInitTypeDef structure.  \*/  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 = 8;  RCC\_OscInitStruct.PLL.PLLN = 336;  RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;  RCC\_OscInitStruct.PLL.PLLQ = 7;  **if** (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != *HAL\_OK*)  {  Error\_Handler();  }  /\*\* Initializes the CPU, AHB and APB buses 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\_DIV4;  **if** (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != *HAL\_OK*)  {  Error\_Handler();  }  }  /\*\*  \* @brief GPIO Initialization Function  \* @param None  \* @retval None  \*/  **static** **void** **MX\_GPIO\_Init**(**void**)  {  GPIO\_InitTypeDef GPIO\_InitStruct = {0};  /\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/  /\* USER CODE END MX\_GPIO\_Init\_1 \*/  /\* GPIO Ports Clock Enable \*/  \_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();  \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();  \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();  \_\_HAL\_RCC\_GPIOG\_CLK\_ENABLE();  /\*Configure GPIO pin Output Level \*/  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13|GPIO\_PIN\_14, *GPIO\_PIN\_RESET*);  /\*Configure GPIO pins : PG13 PG14 \*/  GPIO\_InitStruct.Pin = GPIO\_PIN\_13|GPIO\_PIN\_14;  GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;  GPIO\_InitStruct.Pull = GPIO\_NOPULL;  GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;  HAL\_GPIO\_Init(GPIOG, &GPIO\_InitStruct);  /\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/  /\* USER CODE END MX\_GPIO\_Init\_2 \*/  }  /\* USER CODE BEGIN 4 \*/  /\* USER CODE END 4 \*/  /\* USER CODE BEGIN Header\_StartTask01 \*/  /\*\*  \* @brief Function implementing the TaskDisplayLCD thread.  \* @param argument: Not used  \* @retval None  \*/  /\* USER CODE END Header\_StartTask01 \*/  **void** **StartTask01**(**void** **const** \* argument)  {  /\* init code for USB\_DEVICE \*/  MX\_USB\_DEVICE\_Init();  /\* USER CODE BEGIN 5 \*/  BSP\_LCD\_Init();  BSP\_LCD\_LayerDefaultInit(1, SDRAM\_DEVICE\_ADDR);  BSP\_LCD\_SelectLayer(1);  BSP\_LCD\_DisplayOn();  BSP\_LCD\_SetBackColor(LCD\_COLOR\_GREEN);  BSP\_LCD\_SetTextColor(LCD\_COLOR\_BLACK);  BSP\_LCD\_Clear(LCD\_COLOR\_GREEN);  **float** x\_angRate;  **char** str\_score [50];  **char** str\_currentHigh [50];  /\* Infinite loop \*/  **for**(;;)  {  **sprintf**(str\_score,"Score: %d", score);  **sprintf**(str\_currentHigh, "Current high: %d\n", current\_radius);  BSP\_LCD\_Clear(LCD\_COLOR\_GREEN);  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13, *GPIO\_PIN\_RESET*);  **if**(!fallDown && gameRunning){  CDC\_Transmit\_HS(str\_currentHigh, **strlen**(str\_currentHigh));  BSP\_LCD\_DisplayStringAtLine(1,str\_score);  BSP\_LCD\_FillCircle(120, 160, current\_radius);  **switch** (current\_speed) {  **case** HIGH\_SPEED:{  current\_radius += 8;  **if**(current\_radius >= MAX\_RADIUS\_OF\_HIGH\_SPEED)  fallDown = 1;  **break**;  }  **case** MEDIUM\_SPEED:{  current\_radius += 6;  **if**(current\_radius >= MAX\_RADIUS\_OF\_MEDIUM\_SPEED)  fallDown = 1;  **break**;  }  **case** LOW\_SPEED:{  current\_radius += 4;  **if**(current\_radius >= MAX\_RADIUS\_OF\_LOW\_SPEED)  fallDown = 1;  **break**;  }  **default**:  **break**;  }  }  **else**{  **if**(current\_radius > MIN\_RADIUS){  BSP\_LCD\_DisplayStringAtLine(1,str\_score);  BSP\_LCD\_FillCircle(120, 160, current\_radius);  current\_radius -= 5;  }  **else**{  gameRunning = 0;  BSP\_LCD\_Clear(LCD\_COLOR\_GREEN);  BSP\_LCD\_DisplayStringAtLine(5, " GAME OVER!");  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_14, *GPIO\_PIN\_SET*);  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13, *GPIO\_PIN\_RESET*);  CDC\_Transmit\_HS(str\_currentHigh, **strlen**(str\_currentHigh));  }  }  osMessagePut(myQueue01Handle, current\_radius, osWaitForever);  osDelay(100);  }  /\* USER CODE END 5 \*/  }  /\* USER CODE BEGIN Header\_StartTask02 \*/  /\*\*  \* @brief Function implementing the TaskGyroScope thread.  \* @param argument: Not used  \* @retval None  \*/  /\* USER CODE END Header\_StartTask02 \*/  **void** **StartTask02**(**void** **const** \* argument)  {  /\* USER CODE BEGIN StartTask02 \*/  BSP\_GYRO\_Init();  **float** L3GD20\_AngRate[3];  /\* Infinite loop \*/  **for**(;;){  osEvent event = osMessageGet(myQueue01Handle, osWaitForever);  **int** currentRadiusFromMessQ = event.value.v;  L3GD20\_ReadXYZAngRate(L3GD20\_AngRate);  **for**(**int** i = 0; i < 3; i++){  L3GD20\_AngRate[i] = L3GD20\_AngRate[i]/10000;  }  **if**(L3GD20\_AngRate[0] >= 15 && currentRadiusFromMessQ > 15 && currentRadiusFromMessQ <= 75 && gameRunning && fallDown){  fallDown = 0;  score++;  HAL\_GPIO\_WritePin(GPIOG, GPIO\_PIN\_13, *GPIO\_PIN\_SET*);  **if**(currentRadiusFromMessQ <= 35)  current\_speed = HIGH\_SPEED;  **else** **if** (currentRadiusFromMessQ > 35 && currentRadiusFromMessQ <= 45)  current\_speed = MEDIUM\_SPEED;  **else**  current\_speed = LOW\_SPEED;  }  osDelay(1);  }  /\* USER CODE END StartTask02 \*/  }  /\*\*  \* @brief Period elapsed callback in non blocking mode  \* @note This function is called when TIM6 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 == TIM6) {  HAL\_IncTick();  }  /\* USER CODE BEGIN Callback 1 \*/  /\* USER CODE END Callback 1 \*/  }  /\*\*  \* @brief This function is executed in case of error occurrence.  \* @retval None  \*/  **void** **Error\_Handler**(**void**)  {  /\* USER CODE BEGIN Error\_Handler\_Debug \*/  /\* User can add his own implementation to report the HAL error return state \*/  \_\_disable\_irq();  **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,  ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/  /\* USER CODE END 6 \*/  }  **#endif** /\* USE\_FULL\_ASSERT \*/ |

* Link báo cáo kết quả: *https://youtu.be/WsZNiXzZZtw*