BÁO CÁO THỰC HÀNH BÀI Lab5_VL53L01_Flash

Môn học: **Chuyên đề thiết kế hệ thống nhúng 1** - Mã lớp: **CE437.P11**
Giảng viên hướng dẫn thực hành: **Phạm Minh Quân**

Thông tin sinh viên	**Mã số sinh viên**	**Họ và tên**
	22521472	**Phạm Quốc Tiến**
	22521570	**Trịnh Thành Trung**
	22521564	**Nguyễn Đức Trung**
Link các tài liệu tham khảo *(nếu có)*		
Đánh giá của giảng viên: *+ Nhận xét* *+ Các lỗi trong chương trình* *+ Gợi ý*		

[Báo cáo chi tiết các thao tác, quy trình sinh viên đã thực hiện trong quá trình làm bài thực hành. Chụp lại hình ảnh màn hình hoặc hình ảnh kết quả chạy trên sản phẩm. Mô tả và giải thích chương trình tương ứng để cho ra kết quả như hình ảnh đã trình bày.]

Mục lục

1) Giao tiếp với bộ nhớ Flash

Trong bộ nhớ flash của stm32f103c8t6 được phân bổ thành 127 page, mỗi page có kích thước là 1Kbyte. Do trong bộ nhớ flash có lưu trữ cả code nạp xuống vậy nên nhóm sẽ thực hiện đọc ghi ở page 127 với địa chỉ bắt đầu là 0x08001000

Table 3. Flash module organization (medium-density devices)

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 Kbyte
	Page 1	0x0800 0400 - 0x0800 07FF	1 Kbyte
	Page 2	0x0800 0800 - 0x0800 0BFF	1 Kbyte
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 Kbyte
	Page 4	0x0800 1000 - 0x0800 13FF	1 Kbyte
	. . .	. . .	. . .
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 Kbyte
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16

Thêm thư viện và khai báo

```
24 #include <string.h>
25 #include <stdbool.h>
26 #include <stdio.h>
```

```
36 #define START_PAGE_ADRR 0x0801FC00
37 #define WORD_TO_BYTES 4
38 #define OFF_SET 3
39 #define PAGE_NUM_USE 1
```

```
55 const char* write_data = "CEEC";
56 size_t data_len;
57 uint32_t read_data[50];
```

Trong hàm FLASH_WritePage có chức năng viết dữ liệu xuống vùng nhớ FLASH thì nhóm sẽ thực hiện gán data_len độ dài của chuỗi cần ghi cộng thêm 1 ký tự kết thúc '\0'. Sau đó nhóm tiến hành mở khóa vùng nhớ FLASH

```
75 void FLASH_WritePage(void)
76 {
77     data_len = strlen(write_data) + 1;
78     HAL_FLASH_Unlock();
```

Do đặc điểm của bộ nhớ FLASH nên nhóm sẽ xóa dữ liệu ở Page 127 trước khi tiến hành ghi dữ liệu. Biến PageError sẽ có địa chỉ cũa vùng nhớ bị lỗi (nếu có) khi đang xóa và phần này nhóm sẽ mặc định rằng không có vùng nhớ lỗi

```
    FLASH_EraseInitTypeDef EraseInit =
            {
                    .TypeErase = FLASH_TYPEERASE_PAGES,
                    .PageAddress = START_PAGE_ADRR,
                    .NbPages = PAGE_NUM_USE
            };
    uint32_t PageError = 0;
    HAL_FLASHEx_Erase(&EraseInit, &PageError);
```

Tiếp theo nhóm sẽ tiến hành covert data từ chuỗi ký tự sang dạng uint32_t, sau đó tiến hành ghi xuống vùng nhớ FLASH

```
    uint32_t data = 0;
    for(uint32_t position = 0; position < data_len; position += WORD_TO_BYTES)
    {
        data = 0;
        memcpy(&data, &write_data[position],
                (data_len - position >= WORD_TO_BYTES) ? WORD_TO_BYTES : (data_len - position));
        HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, START_PAGE_ADRR + position, data);
    }
```

Cụ thể do chương trình sẽ ghi xuống vùng nhớ một word (4 bytes) mỗi lần ghi. Tuy nhiên, dữ liệu tron write_data có thể không chia hết cho 4 bytes. Vậy nên điều kiện trong memcpy đảm bảo không ghi quá mức dữ liệu sẵn có trong write_data và dữ liệu cuối cùng nếu nhỏ hơn 4 bytes thì vẫn được ghi đầy đủ mà không gây lỗi.

Sau đó nhóm sẽ khóa bộ nhớ FLASH lại

```
    HAL_FLASH_Lock();
}
```

Trong hàm FLASH_ReadPage() có chức năng đọc dữ liệu trong bộ nhớ FLASH thì nhóm sẽ tiến hành gán địa chỉ ban đầu, reset biến read_data

```
void FLASH_ReadPage(void)
{
    uint32_t address = START_PAGE_ADRR;
    memset(read_data, 0, sizeof(read_data));
```

Sau đó nhóm sẽ tiến hành đọc giá trị từ bộ nhớ FLASH

```
    for (uint32_t position = 0; position < data_len; position += WORD_TO_BYTES)
    {
        read_data[position / WORD_TO_BYTES] = *(__IO uint32_t *)(address + position);
    }
}
```

Trong hàm FLASH_CompareData() có chức năng so sánh chuỗi ghi và chuỗi đọc được có giống nhau không thì nhóm sẽ tiến hành chuyển đổi dữ liệu đọc được từ uint32_t sang chuỗi char

```
bool FLASH_CompareData(void)
{
    char buffer[50] = {0};

    for (uint32_t position = 0; position < (data_len + OFF_SET) / WORD_TO_BYTES; position++)
    {
        memcpy(&buffer[position * WORD_TO_BYTES], &read_data[position], WORD_TO_BYTES);
    }
```

Lý đo phải cộng thêm OFF_SET là để đảm bảo lấy các byte cuối cùng (không đủ 4 bytes). Lý do lấy OFF_SET = 3 là do phần byte dư tối đa 3 bytes.

Sau đó nhóm sẽ return kết quả so sánh write_data và buffer

```
    return (strcmp(buffer, write_data) == 0);
}
```

Trong hàm main nhóm sẽ bật led green khi hai chuỗi giống nhau và bật led red khi hai chuỗi khác nhau

```
163   HAL_Delay(500);
164   FLASH_WritePage();
165   FLASH_ReadPage();
166   if(FLASH_CompareData())
167   {
168       HAL_GPIO_WritePin(LEDG_GPIO_Port, LEDG_Pin, 0);
169   }
170   else
171   {
172       HAL_GPIO_WritePin(LEDR_GPIO_Port, LEDR_Pin, 0);
173   }
```

2) Giao tiếp I2C

Trong phần code này nhóm sẽ sử dụng thư viện để giao tiếp với VL53L1X
Thêm thư viện vào main.h

```
33 /* USER CODE BEGIN Includes */
34 #include "VL53L1X_api.h"
35 #include "vl53l1_platform.h"
36 /* USER CODE END Includes */
```

Khai báo trong hàm main

```
154   uint16_t Distance_0 = 0;
155   uint16_t Distance_1 = 0;
156   char Data[20] = "";
157   uint16_t Data_len = 20;
```

Tiếp theo nhóm sẽ kéo chân Xshut của hai cảm biến về 0 để hai cảm biến quay về chế độ standby. Sau đó nhóm sẽ gọi hàm khởi tạo

```
158   HAL_GPIO_WritePin(XShut0.Port, XShut0.Pin, 0);
159   HAL_GPIO_WritePin(XShut1.Port, XShut1.Pin, 0);
160   while(VL53L1__Init(VL53L1_0_ADDR, XShut0) != 0){}
161   while(VL53L1__Init(VL53L1_1_ADDR, XShut1) != 0){}
```

Trong hàm khởi tạo nhóm sẽ kéo chân XShut của cảm biến khởi tạo để cảm biến chạy chế độ boot. Khi đó nhóm có thể lập trình cho cảm biến được

```
154 uint8_t VL53L1__Init(uint8_t VL53L1_NEW_ADDR, GPIO_HandlerStruct XShut){
155     uint8_t refRegs[4] = {0,0,0,0};
156     uint8_t status =0;
157 
158     // Enable VL53L1 sensor waiting for a complete boot sequence
159 #ifdef  VL53L1__USING_XSHUT
160     status |= VL53L1__Xshut(XShut, 1);
161     if (status)
162         return (status);
163 #endif
164     HAL_Delay(4);
```

Sau đó nhóm sẽ cài đặt các giá trị để cảm biến hoạt động

```
    //check if VL53L1X is alive and kicking. Remove MASKREV if VL53L1
    VL53L1_ReadMulti(VL53L1__ADDR, VL53L1__MODELID_INDEX, refRegs, 4);
    if ((refRegs[0]!=VL53L1__MODELID_VALUE) || (refRegs[1]!=VL53L1__MODULETYPE_VALUE) || (refRegs[2]!=VL53L1__MASKREV_VALUE))
        return (1);


    // VL53L1X sensor is available
    /* initializing: default setting  */
    status |= VL53L1X_SensorInit(VL53L1__ADDR);
    /* initializing: device calibration settings*/
    status |= VL53L1X_SetOffset(VL53L1__ADDR, VL53L1__CALIB_OFFSET);
    status |= VL53L1X_SetXtalk(VL53L1__ADDR, VL53L1__CALIB_XTALK);
    /* initializing: project settings */
    status |= VL53L1X_SetDistanceMode(VL53L1__ADDR, VL53L1__DISTANCE_MODE);
    status |= VL53L1X_SetTimingBudgetInMs(VL53L1__ADDR, VL53L1__TIMING_BUDGET);
    status |= VL53L1X_SetInterMeasurementInMs(VL53L1__ADDR, VL53L1__INTERMEASUREMENT);
    status |= VL53L1X_SetDistanceThreshold(VL53L1__ADDR,VL53L1__LOWER_THRESHOLD, VL53L1__UPPER_THRESHOLD, VL53L1__WINDOW_MODE, 0);

```

Sau cùng nhóm sẽ set lại địa chỉ của cảm biến (do khi mới khởi động lại thì cảm biến sẽ chạy giá trị mặc định) và return lại lỗi. Nếu không có lỗi thì status = 0

```
    status |= VL53L1X_SetI2CAddress(VL53L1__ADDR, VL53L1_NEW_ADDR);
    return (status);
};
```

Tiếp đến trong hàm main nhóm sẽ tiến hành điều khiển cảm biến bắt đầu đo

```
  VL53L1X_StartRanging(VL53L1_0_ADDR);
  VL53L1X_StartRanging(VL53L1_1_ADDR);
```

Trong vòng while nhóm sẽ tiến hành đọc và xuất data qua UART

```
  while (1)
  {
      VL53L1__GetDistance(VL53L1_0_ADDR, GPIO1_0, &Distance_0);
      VL53L1__GetDistance(VL53L1_1_ADDR, GPIO1_1, &Distance_1);
      Data_len = sprintf(Data, "\n\r%u %u", Distance_0, Distance_1);
      HAL_UART_Transmit(&huart1, (uint8_t*)Data, Data_len, 100);
      HAL_Delay(100);
```

Trong hàm VL53L1_GetDistance() thì hàm sẽ thực hiện kiểm tra chân GPIO1 đã được kéo lên chưa (kéo lên tương đương dữ liệu sẵn sàng để đọc). Chương trình sẽ chở đến khi hết thời gian đo mà chương trình đã lập trình cho cảm biến

```
uint8_t VL53L1__GetDistance(uint8_t __ADDR, GPIO_HandlerStruct GPIO1, uint16_t *Distance){
    uint8_t RangingStatus;
    uint8_t status =0;
    uint32_t testingTime=HAL_GetTick();
    static uint16_t PrevDistance=0;

#ifdef  VL53L1__USING_GPIO
    // VL53L1X data available test if TOF_GPIO pin is available
    while ((!HAL_GPIO_ReadPin(GPIO1.Port, GPIO1.Pin)) && ((HAL_GetTick()-testingTime)<=VL53L1__INTERMEASUREMENT)) {};
    if (HAL_GPIO_ReadPin(GPIO1.Port, GPIO1.Pin)) {
#else
```

Nếu mà chân GPIO1 không kéo lên thì hàm sẽ set distance bằng giá trị trước đó đo được và trả về status = 1 (tương đương lỗi). Còn không thì chương trình sẽ thực hiện lấy data.

```
218         status |= VL53L1X_GetRangeStatus(__ADDR, &RangingStatus);
219         status |= VL53L1X_GetDistance(__ADDR, Distance);
220         status |= VL53L1X_ClearInterrupt(__ADDR);
221         if ((status==0) && (RangingStatus<=VL53L1__RANGE_STATUS_THRESH)) {
222             PrevDistance=*Distance;
223         } else {
224             *Distance=PrevDistance;
225             status=1;
226         }
227     } else{
228         *Distance=PrevDistance;
229         status=1;
230     }
231     return status;
232 }
```

Nếu giá trị mà GetRangeStatus mà trả về lớn hơn 2 thì chương trình sẽ trả về lỗi còn không thì sẽ phụ thuộc vào status. Lý do mà nhóm lấy giá trị 2 thì trong datasheet họ nói rằng số 2 tương ứng với việc warning: low return signal level hay là mức tín hiệu phản xạ trở lại từ đối tượng được đo là quá thấp để đảm bảo phép đo chính xác tức là nhóm chấp nhận lấy dữ liệu từ trường hợp này đến trường hợp tốt nhất.

3) DEMO

Code
Video demo