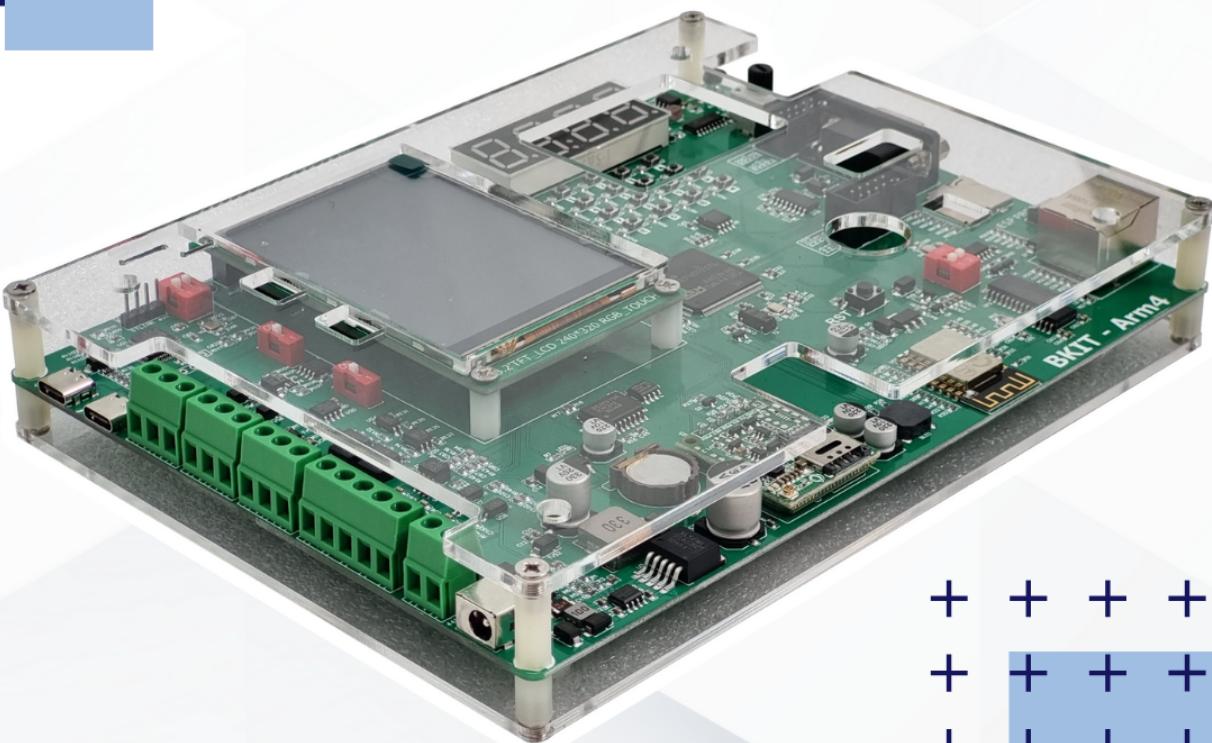


KIT THÍ NGHIỆM BKIT

ARM4



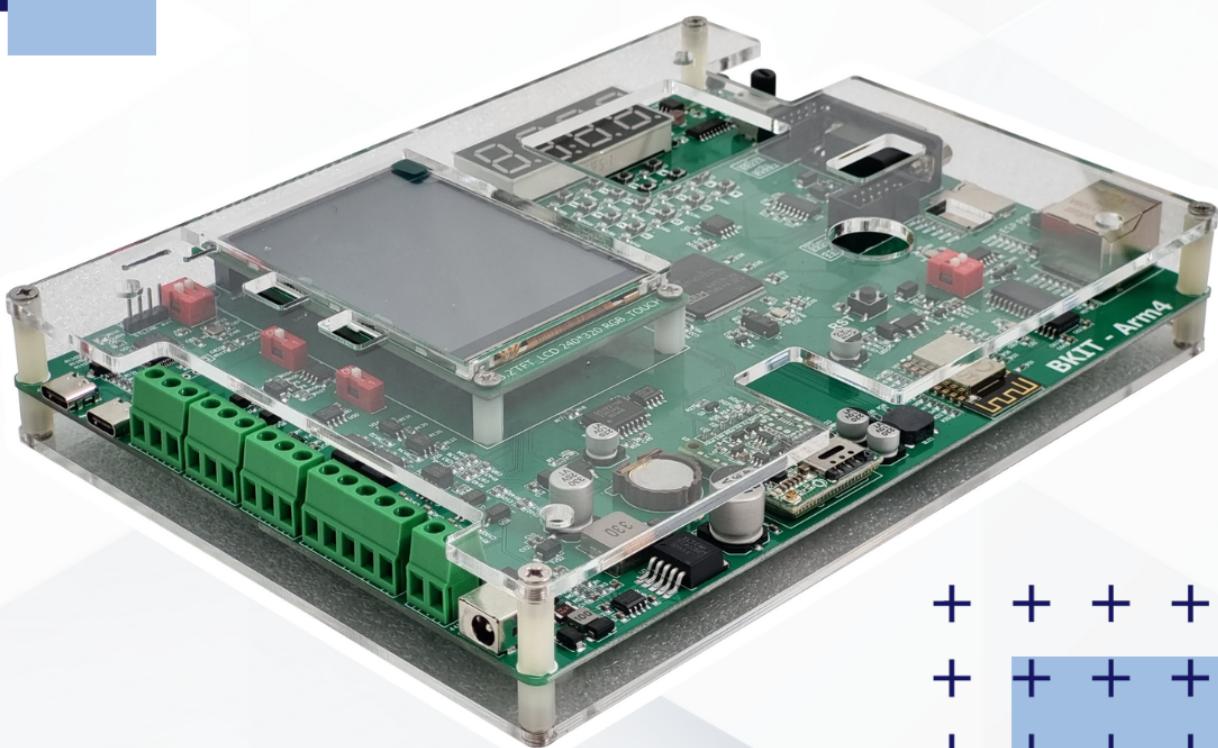
Mục lục

| | |
|--|-----------|
| Chapter 6. ADC-PWM | 4 |
| 1 Mục tiêu | 5 |
| 2 Giới thiệu | 5 |
| 3 Cơ sở lý thuyết | 6 |
| 3.1 Bộ chuyển đổi ADC | 6 |
| 3.2 Chế độ điều rộng xung - PWM | 9 |
| 4 Hướng dẫn cấu hình | 11 |
| 4.1 Cấu hình ADC: | 11 |
| 4.2 Cấu hình PWM: | 13 |
| 5 Hướng dẫn lập trình | 14 |
| 5.1 Thư viện sensor.h | 14 |
| 5.2 Thư viện buzzer.h | 15 |
| 6 Bài tập và báo cáo | 21 |
| 6.1 Bài tập 1 | 21 |
| 6.2 Bài tập 2 (Nâng cao) | 21 |
| Chapter 7. LCD Touch | 22 |
| 1 Mục tiêu | 23 |
| 2 Giới thiệu | 23 |
| 3 Cơ sở lý thuyết | 23 |
| 3.1 Touch sensor | 23 |
| 3.1.1 Cảm biến điện trở | 23 |
| 3.1.2 Cảm biến điện dung | 24 |

| | | |
|----------------------------------|--|-----------|
| 3.2 | Bộ nhớ EEPROM AT24C | 25 |
| 3.2.1 | Quá trình hình thành EEPROM | 25 |
| 3.2.2 | EEPROM AT24C | 26 |
| 4 | Hướng dẫn cấu hình | 26 |
| 4.1 | Cấu hình Touch: | 26 |
| 5 | Hướng dẫn lập trình | 29 |
| 5.1 | Thư viện touch.h | 29 |
| 5.2 | Thư viện software_timer.c | 30 |
| 5.3 | Thư viện at24c.h | 30 |
| 6 | Bài tập và báo cáo | 38 |
| Chapter 8. ESP8266 - WIFI | | 39 |
| 1 | Mục tiêu | 40 |
| 2 | Giới thiệu | 40 |
| 3 | Cơ sở lý thuyết | 41 |
| 3.1 | Giới thiệu về ESP8266 | 41 |
| 3.2 | Kiến trúc 5 lớp của IoT | 42 |
| 4 | Hướng dẫn cấu hình | 43 |
| 5 | Hướng dẫn lập trình | 44 |
| 5.1 | Hướng dẫn lập trình ESP8266 trên Arduino IDE | 45 |
| 5.2 | Hướng dẫn làm việc với Adafruit | 51 |
| 5.2.1 | Tạo tài khoản và kênh dữ liệu | 51 |
| 5.2.2 | Tạo dashboard và kết nối với feed | 53 |
| 5.3 | Hướng dẫn thư viện STM | 65 |
| 6 | Bài tập và báo cáo | 71 |

CHƯƠNG 6

ADC-PWM



1 Mục tiêu

- Tìm hiểu về module ADC và ứng dụng để đọc giá trị các cảm biến trên Kit thí nghiệm.
- Tìm hiểu về module PWM và ứng dụng để điều khiển buzzer.
- Biết cách sử dụng các thư viện liên quan đến đọc cảm biến và điều khiển buzzer.

2 Giới thiệu

ADC (Analog-to-Digital Converter) là một thành phần điện tử có chức năng chính là chuyển đổi tín hiệu analog (dạng liên tục) thành tín hiệu số (dạng rời rạc). ADC đóng vai trò quan trọng trong nhiều ứng dụng điện tử và hệ thống đo lường, giúp chúng ta thu thập và xử lý dữ liệu từ các cảm biến, tín hiệu điện áp, tín hiệu âm thanh và nhiều nguồn tín hiệu khác.

Tín hiệu số là loại tín hiệu được biểu thị bằng 0 hoặc 1, trong khi tín hiệu tương tự có phạm vi giá trị lớn hơn chỉ 0 hoặc 1. Cả hai đều được sử dụng trong các thiết bị điện tử xung quanh chúng ta, nhưng cách xử lý chúng khác nhau. Đối với đầu vào tương tự, chúng ta có thể thu thập dữ liệu thời gian thực từ cảm biến, sau đó sử dụng bộ chuyển đổi tương tự sang số (ADC) để chuyển đổi dữ liệu thành dạng số cho vi điều khiển. Nhưng làm thế nào nếu chúng ta muốn điều khiển thiết bị analog từ bộ vi điều khiển? Một số bộ vi điều khiển có bộ chuyển đổi kỹ thuật số sang tương tự (DAC) tích hợp, phát ra tín hiệu tương tự để điều khiển các thiết bị analog. Chúng ta cũng có thể sử dụng DAC bên ngoài. Tuy nhiên, việc sản xuất DAC khá đắt đỏ và có kích thước khá lớn. Để giải quyết những khó khăn này và thực hiện chức năng DAC một cách tiết kiệm chi phí hơn, chúng ta có thể sử dụng kỹ thuật điều xung (PWM).

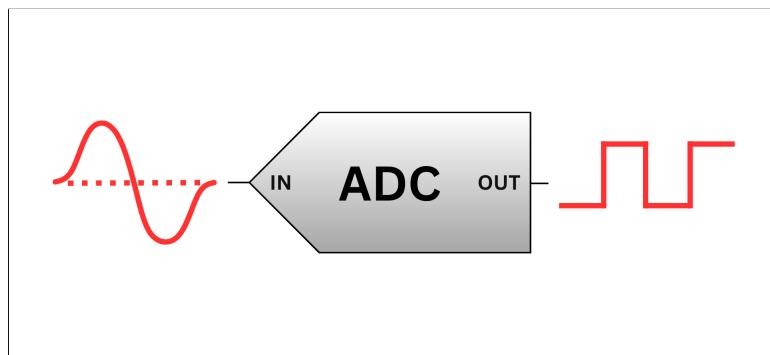
PWM (Pulse Width Modulation) là một kỹ thuật được sử dụng để điều khiển các thiết bị analog bằng cách sử dụng tín hiệu số. Kỹ thuật này có thể được sử dụng để xuất tín hiệu tương tự từ thiết bị kỹ thuật số (ví dụ như vi điều khiển).

3 Cơ sở lý thuyết

3.1 Bộ chuyển đổi ADC

Bộ chuyển đổi ADC hoạt động như thế nào:

Ta có thể coi hoạt động của ADC như một bộ chia tỷ lệ toán học. Tỷ lệ cơ bản là việc chuyển đổi các giá trị từ một dải này sang một dải khác, vì vậy ADC chuyển đổi một giá trị điện áp thành một số nhị phân. Dưới đây là một số tính năng chính của ADC, và qua đó, chúng ta sẽ tìm hiểu cách nó hoạt động.



Hình 6.1: Bộ chuyển đổi ADC

Phân loại bộ chuyển đổi ADC:

- **Bộ chuyển đổi Flash ADC:**

- Flash ADC thường rất nhanh vì nó thực hiện nhiều so sánh đồng thời. Vì vậy, nó thích hợp cho các ứng dụng yêu cầu tốc độ cao như trong các hệ thống điều khiển hoặc xử lý tín hiệu thời gian thực.
- Độ phân giải cố định: số bit đầu ra của Flash ADC được quyết định bởi số lượng so sánh. Ví dụ, nếu có 8 so sánh, ADC sẽ có độ phân giải cố định là 8-bit.
- Tính phức tạp cao: Flash ADC yêu cầu một số lượng so sánh rất lớn, và điều này làm tăng tính phức tạp của mạch. Điều này thường dẫn đến sự gia tăng về diện tích chip và tiêu thụ điện năng so với các loại ADC khác.
- Ứng dụng chính: Flash ADC thường được sử dụng trong các ứng dụng yêu cầu tốc độ cao như radar, truyền thông số hóa, và xử lý tín hiệu thời gian thực. Dù tiêu thụ điện năng cao và chi phí tổng cộng cao, tốc độ chuyển đổi cao và độ tin cậy làm Flash ADC được sử dụng nhiều.

- **Digital Ramp ADC:**

- Digital Ramp ADC là một dạng của ADC được sử dụng để chuyển đổi tín hiệu analog thành dạng số. Nó sử dụng một nguyên tắc hoạt động đặc biệt gọi là "kỹ thuật đường dốc số".
- Độ phân giải thấp: Digital Ramp ADC thường có độ phân giải thấp hơn so với nhiều loại ADC khác. Điều này có nghĩa là chúng có khả năng chuyển đổi tín hiệu với độ chính xác tương đối thấp.
- Tốc độ chuyển đổi chậm: Do cách hoạt động, tốc độ chuyển đổi của Digital Ramp ADC thường chậm hơn so với nhiều loại ADC khác. Điều này khiến chúng không thích hợp cho các ứng dụng yêu cầu tốc độ cao.
- Ứng dụng: Digital Ramp ADC thường được sử dụng trong các ứng dụng đòi hỏi độ chính xác thấp và tốc độ chuyển đổi không quan trọng, như hệ thống đo lường tự động hoặc ứng dụng kiểm tra. Tuy nhiên, loại ADC này vẫn có hạn chế về độ phân giải và tốc độ chuyển đổi.

- **Successive Approximation Register (SAR):**

- SAR là một loại ADC chuyển đổi tín hiệu analog thành số bằng phương pháp tiếp cận tương tự, thường được sử dụng trong các ứng dụng đòi hỏi độ chính xác cao và tốc độ chuyển đổi nhanh.
- Độ phân giải: SAR hỗ trợ độ phân giải từ 8-bit đến 18-bit hoặc cao hơn, cho phép chia thành nhiều mức giá trị khác nhau. Độ phân giải càng cao, độ chính xác càng tốt, điều này quan trọng trong các ứng dụng đo lường và điều khiển yêu cầu độ chính xác cao.
- Tốc độ: SAR thường có tốc độ chuyển đổi nhanh, được đo bằng số lần chuyển đổi mỗi giây. Tuy nhiên, tốc độ này có thể giảm khi chọn độ phân giải cao hoặc xử lý nhiều kênh đầu vào cùng một lúc.
- Ứng dụng: SAR thường được sử dụng trong các ứng dụng yêu cầu độ chính xác và hiệu suất cao, như hệ thống điều khiển, thiết bị y tế, các thiết bị công nghiệp, thiết bị đo lường và các ứng dụng audio, video hoặc các ứng dụng yêu cầu tốc độ chuyển đổi cao. Tuy nhiên, SAR có hai nhược điểm chính: tiêu thụ năng lượng cao và chi phí cho độ phân giải cao, thường cần nhiều bit trong bộ đếm, điều này có thể làm tăng chi phí và diện tích vật lý của vi mạch, làm phức tạp hóa thiết kế PCB và hao phí không gian.

- **Tracking ADC:**

- Tracking ADC là một loại ADC được sử dụng để chuyển đổi tín hiệu ADC theo thời gian thực, nghĩa là nó theo dõi và ghi lại giá trị tín hiệu đầu vào

liên tục, thay vì thực hiện chuyển đổi theo cách rời rạc. Một điểm quan trọng của Tracking ADC là nó cố gắng duy trì một giá trị kỹ thuật số ứng với tín hiệu analog đầu vào trong suốt thời gian hoạt động.

- Độ phân giải: Độ phân giải của Tracking ADC xác định số bit trong dữ liệu kỹ thuật số được ghi lại bởi nó. Tracking có khả năng chuyển đổi tín hiệu analog thành dạng số với độ chính xác tốt hơn. Độ phân giải thường được đo bằng số bit, ví dụ: 12-bit, 16-bit, 18-bit, vv. Độ phân giải cao đòi hỏi nhiều tài nguyên tính toán và băng thông dữ liệu lớn hơn.
- Tốc độ: Tốc độ của Tracking ADC là tốc độ lấy mẫu và chuyển đổi tín hiệu từ dạng analog thành dạng digital. Tốc độ cao cho phép Tracking ADC theo kịp và ghi lại biến động nhanh của tín hiệu. Tuy nhiên, tốc độ cao có thể đòi hỏi nhiều công suất và băng thông dữ liệu lớn.
- Ứng dụng: Tracking ADC thường được sử dụng trong các ứng dụng yêu cầu xử lý tín hiệu thời gian thực và theo dõi biến động nhanh của tín hiệu. Các ứng dụng tiêu biểu bao gồm: Hệ thống điều khiển, Truyền thông, Xử lý tín hiệu thời gian thực, Ghi âm và video... Tuy nhiên, Tracking ADC tiêu thụ năng lượng khá lớn, dễ bị nhiễu, có chi phí và kích thước khá lớn.

- **Slope (integrating) ADC:**

- Giống như Digital Ramp ADC, Slope cũng tạo ra sóng răng cưa. Tuy nhiên, Slope sử dụng một mạch opamp (gọi là bộ tích phân).
- Ưu điểm của Slope ADC bao gồm độ phân giải cao, độ chính xác tốt và không yêu cầu mạch tỷ lệ phân đoạn để chuyển đổi tín hiệu thành dạng số. Tuy nhiên, Slope ADC cần thời gian lấy mẫu đủ dài để chuyển đổi, và tốc độ chuyển đổi thấp hơn so với một số loại ADC khác. Slope ADC thường được sử dụng trong các ứng dụng yêu cầu độ chính xác cao, chẳng hạn như trong các hệ thống đo lường và kiểm tra. Tuy nhiên độ chính xác này sẽ giảm dần theo thời gian do hiện tượng nạp xả tụ.

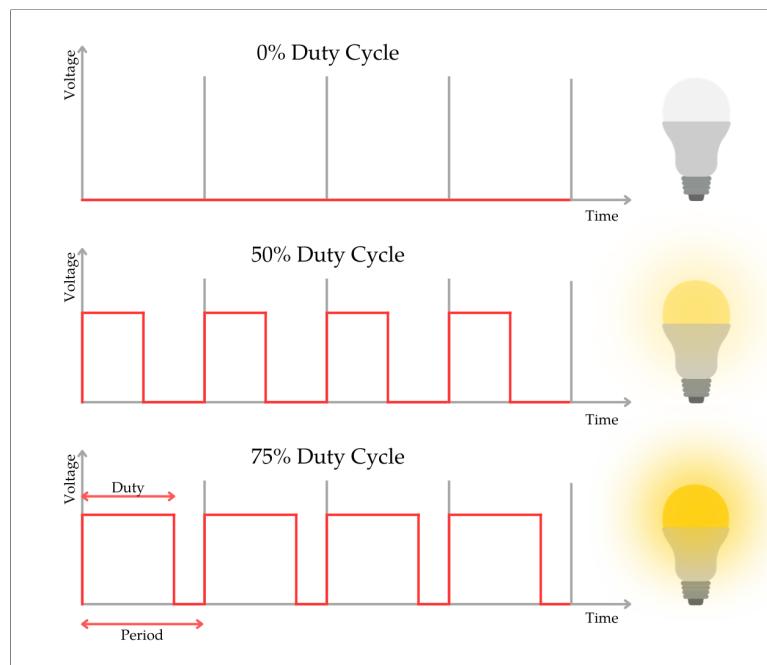
- **Một số loại ADC phức tạp khác:**

- Delta ADC
- Sigma-delta ADC
- Pipeline
- Time-interleaved
- ...

3.2 Chế độ điều rộng xung - PWM

Điều chỉnh độ rộng xung (PWM) (Pulse Width Modulation) áp dụng tín hiệu số để kiểm soát các ứng dụng nguồn, và có khả năng chuyển đổi trở lại tín hiệu tương tự mà không cần nhiều phần cứng. Các hệ thống analog thường tạo ra nhiệt do dòng điện chảy qua các điện trở trong hệ thống lớn, trong khi hệ thống số không tạo ra nhiệt độ như vậy.

- Duty Cycle: là một tham số quan trọng trong PWM và đo lường tỷ lệ thời gian mà tín hiệu PWM ở mức cao so với tổng thời gian chu kỳ. Nó thường được biểu thị dưới dạng phần trăm. Ví dụ, nếu chu kỳ là 1ms và duty cycle là 50%, thì tín hiệu PWM sẽ ở mức cao trong 0,5ms và mức thấp trong 0,5ms.



Hình 6.2: Ví dụ về Duty Cycle

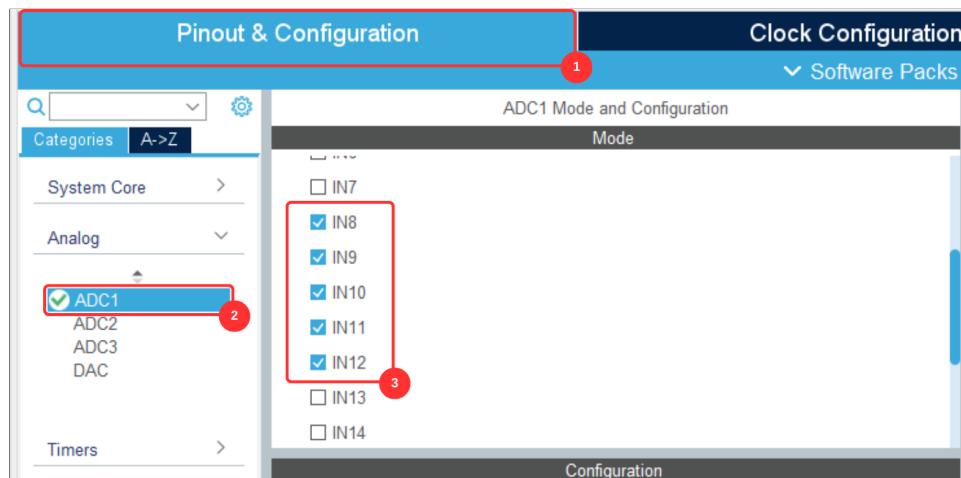
- Mạch cầu H: là một mạch điện tử thường được sử dụng để điều khiển động cơ DC hoặc tải điện áp cao. Nó bao gồm bốn công tắc điện tử (thường là transistor hoặc MOSFET) được kết hợp thành một cấu trúc dạng hình chữ "H". Bằng cách kiểm soát các công tắc, bạn có thể thay đổi hướng và tốc độ quay của động cơ hoặc điều khiển công suất đầu ra của tải.
- Switching Mode Power Supplies (SMPS): là một cách chuyển đổi hiệu suất cao sử dụng PWM để cung cấp năng lượng điện cho các thiết bị điện tử và hệ thống. SMPS thường sử dụng một cấu trúc chuyển đổi đặc biệt để biến đổi nguồn điện đầu vào thành nguồn điện đầu ra có điện áp và dòng điện

ổn định. Quá trình này tiết kiệm năng lượng và thường có hiệu suất cao hơn so với các nguồn cung cấp dựa trên cấu trúc tuyến tính. PWM được sử dụng trong SMPS để kiểm soát đầu ra và duy trì điện áp hoặc dòng điện ổn định bằng cách điều chỉnh duty cycle của tín hiệu PWM.

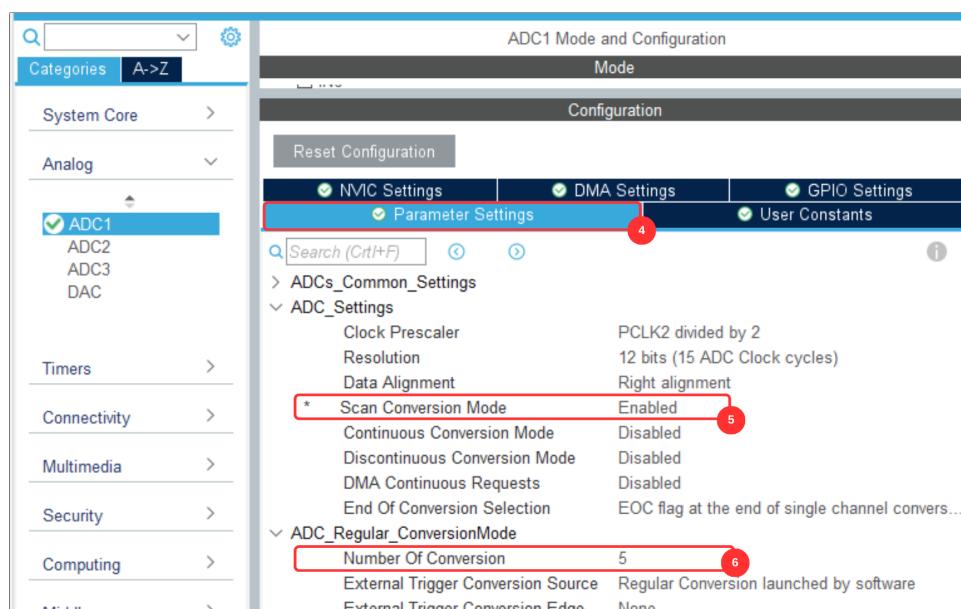
4 Hướng dẫn cấu hình

4.1 Cấu hình ADC:

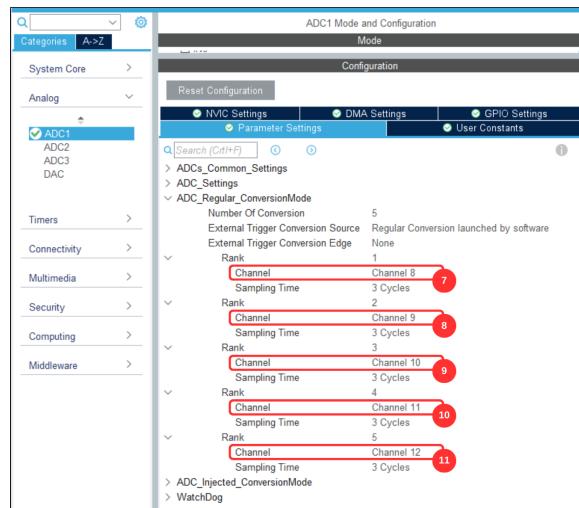
Trên Kit thí nghiệm, chúng ta sẽ giao tiếp với các ngoại vi thông qua ADC1 trên MCU. Trong file .ioc chúng ta sẽ config ADC1 như sau:



Hình 6.3: Config các kênh ADC

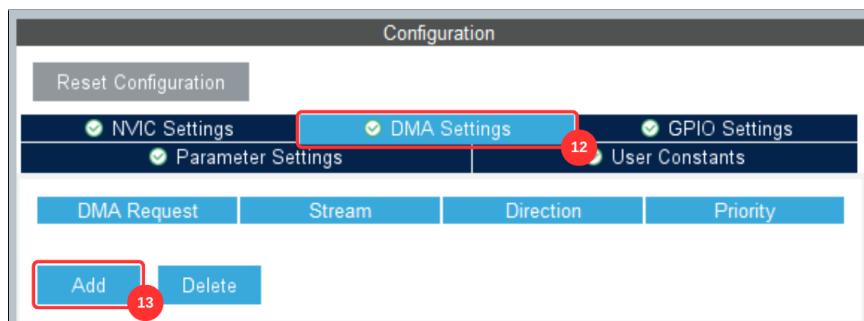


Hình 6.4: Config ADC

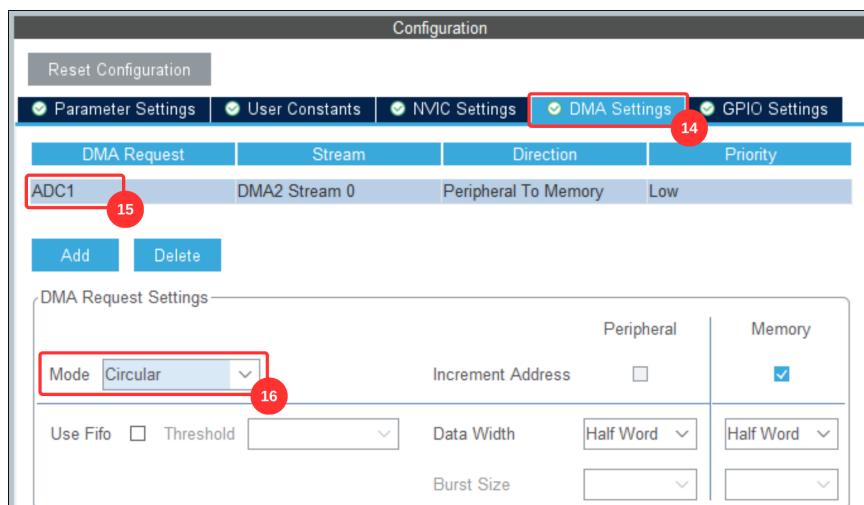


Hình 6.5: Config ADC

Trong thiết kế của Kit thí nghiệm, để có thể sử dụng đầy đủ các cảm biến trên bo mạch, ta cần phải sử dụng nhiều CHANNEL khác nhau của 1 module ADC. Để làm được việc này, ta bắt buộc phải sử dụng cơ chế DMA để hỗ trợ. Đây là cơ chế giúp trao đổi dữ liệu giữa bộ nhớ và ngoại vi mà không cần thông qua CPU.

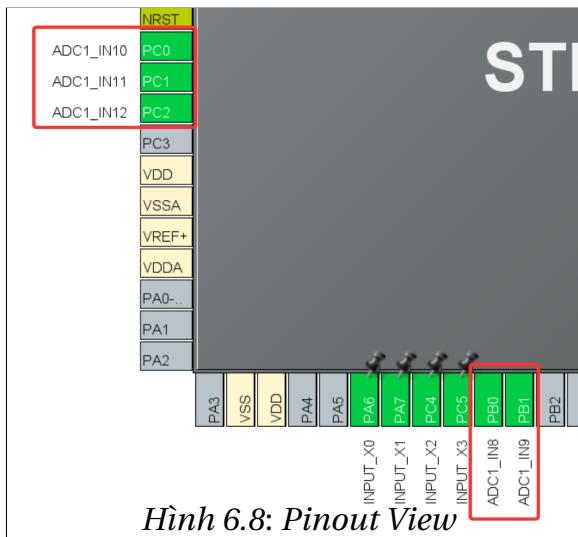


Hình 6.6: Config DMA



Hình 6.7: Config DMA

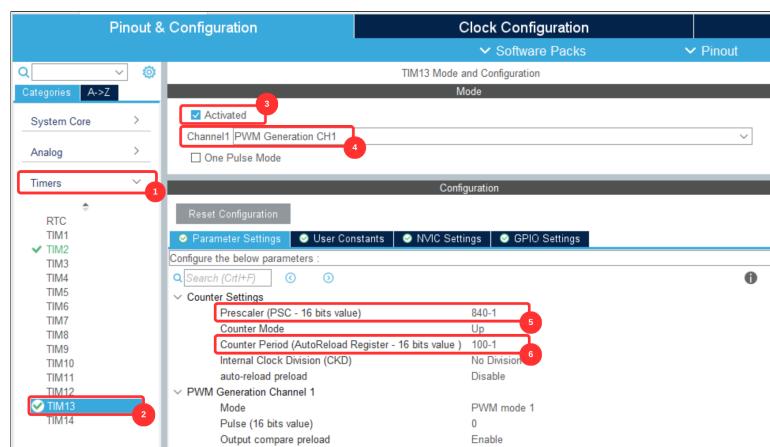
Sau khi cấu hình, ta có Pinout view như sau:



Hình 6.8: Pinout View

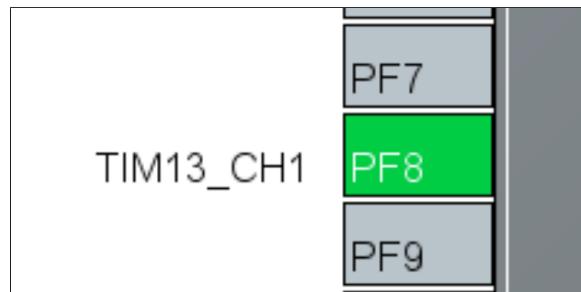
4.2 Cấu hình PWM:

Ta sẽ điều khiển buzzer bằng một xung có tần số 1000Hz. Dựa theo thiết kế của Kit thí nghiệm, buzzer sẽ được điều khiển bằng chân **CHANNEL1** của **TIM13**. Vậy nên ta sẽ cần config TIM13 với tần số 1000Hz. Việc config này đã được giải thích ở **LAB 2**. Với cách config dưới, duty_cycle tối đa của xung PWM sẽ tương ứng với giá trị của **Counter Period**. Trong quá trình lập trình, để thay đổi duty_cycle ta chỉ cần thay đổi giá trị tham số trong khoảng 0-99.



Hình 6.9: Config PWM

Sau khi cấu hình, ta có Pinout view như sau:



Hình 6.10: Pinout View

5 Hướng dẫn lập trình

5.1 Thư viện sensor.h

Các file sensor.h và sensor.c có thể được sao chép từ project mẫu **Bai6_ADC_PWM**.

`void sensor_init()`

- **Mô tả:** Khởi tạo các cảm biến.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

`void sensor_Read()`

- **Mô tả:** Đọc và lưu giá trị các cảm biến từ module ADC.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

`uint16_t sensor_GetLight()`

- **Mô tả:** Đọc giá trị cảm biến ánh sáng.
- **Tham số:** Không có.
- **Giá trị trả về:** Giá trị trả về từ cảm biến ánh sáng (0-4095).

`uint16_t sensor_GetPotentiometer()`

- **Mô tả:** Đọc giá trị biến trổ.
- **Tham số:** Không có.

- **Giá trị trả về:** Giá trị trả về từ biến trỏ (0-4095).

uint16_t sensor_GetVoltage()

- **Mô tả:** Đọc giá trị điện áp được cấp vào Kit thí nghiệm.
- **Tham số:** Không có.
- **Giá trị trả về:** Điện áp được cấp vào Kit thí nghiệm (V).

uint16_t sensor_GetCurrent()

- **Mô tả:** Đọc giá trị dòng điện tiêu thụ của Kit thí nghiệm.
- **Tham số:** Không có.
- **Giá trị trả về:** Dòng điện tiêu thụ (mA).

uint16_t sensor_GetTemperature()

- **Mô tả:** Đọc giá trị cảm biến nhiệt độ.
- **Tham số:** Không có.
- **Giá trị trả về:** Nhiệt độ cảm biến đo được (°C).

5.2 Thư viện buzzer.h

Các file buzzer.h và buzzer.c có thể được sao chép từ project mẫu **Bai6_ADC_PWM**.

void buzzer_init()

- **Mô tả:** Khởi tạo buzzer.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void buzzer_SetVolume(uint8_t dutyCycle)

- **Mô tả:** Thay đổi âm lượng buzzer bằng độ rộng xung.
- **Tham số:**
 - **dutyCycle:** Giá trị độ rộng xung (0-99).

- **Giá trị trả về:** Không có.

Dưới đây là chương trình mẫu giúp in các giá trị đọc được từ cảm biến và dùng 3 nút nhấn để điều khiển buzzer. Thông thường với giá trị duty cycle = 50%, buzzer sẽ kêu với cường độ lớn nhất. Chương trình mẫu sẽ điều khiển duty cycle = 50% nếu nhấn nút "mũi tên lên", duty cycle = 0% nếu nhấn nút "mũi tên xuống", duty cycle = 25% nếu nhấn nút "mũi tên phải".

```

1 // ...
2 /* USER CODE BEGIN Includes */
3 #include "software_timer.h"
4 #include "led_7seg.h"
5 #include "button.h"
6 #include "lcd.h"
7 #include "picture.h"
8 #include "ds3231.h"
9 #include "sensor.h"
10 #include "buzzer.h"
11 /* USER CODE END Includes */
12
13 // ...
14
15 /* USER CODE BEGIN PFP */
16 void system_init();
17 void test_LedDebug();
18 void test_Buzzer();
19 void test_AdC();
20 /* USER CODE END PFP */
21
22 // ...
23
24 int main(void)
25 {
26
27     // ...
28
29     /* USER CODE BEGIN 2 */
30     system_init();
31     /* USER CODE END 2 */
32
33     /* Infinite loop */

```

```

34  /* USER CODE BEGIN WHILE */
35  lcd_Clear(BLACK);
36  while (1)
37  {
38      while(!flag_timer2);
39      flag_timer2 = 0;
40      button_Scan();
41      test_LedDebug();
42      test_Adc();
43      test_Buzzer();
44  /* USER CODE END WHILE */

45
46  /* USER CODE BEGIN 3 */
47 }
48 /* USER CODE END 3 */
49 }

50
51 // ...
52
53 /* USER CODE BEGIN 4 */
54 void system_init(){
55     timer_init();
56     button_init();
57     lcd_init();
58     sensor_init();
59     buzzer_init();
60     setTimer2(50);
61 }
62
63 uint8_t count_led_debug = 0;
64
65 void test_LedDebug(){
66     count_led_debug = (count_led_debug + 1)%20;
67     if(count_led_debug == 0){
68         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port , DEBUG_LED_Pin);
69     }
70 }
71
72 uint8_t isButtonUp()

```

```

73 {
74     if (button_count[3] == 1)
75         return 1;
76     else
77         return 0;
78 }
79
80 uint8_t isButtonDown()
81 {
82     if (button_count[7] == 1)
83         return 1;
84     else
85         return 0;
86 }
87
88 uint8_t isButtonRight()
89 {
90     if (button_count[11] == 1)
91         return 1;
92     else
93         return 0;
94 }
95
96 uint8_t count_adc = 0;
97
98 void test_Adc(){
99     count_adc = (count_adc + 1)%20;
100    if(count_adc == 0){
101        sensor_Read();
102        lcd_ShowStr(10, 100, "Voltage:", RED, BLACK, 16, 0);
103        lcd_ShowFloatNum(130, 100,sensor_GetVoltage(), 4, RED,
104                         BLACK, 16);
105        lcd_ShowStr(10, 120, "Current:", RED, BLACK, 16, 0);
106        lcd_ShowFloatNum(130, 120,sensor_GetCurrent(), 4, RED,
107                         BLACK, 16);
108        lcd_ShowStr(10, 140, "Light:", RED, BLACK, 16, 0);
109        lcd_ShowIntNum(130, 140, sensor_GetLight(), 4, RED,
110                         BLACK, 16);
111        lcd_ShowStr(10, 160, "Potentiometer:", RED, BLACK, 16,
112

```

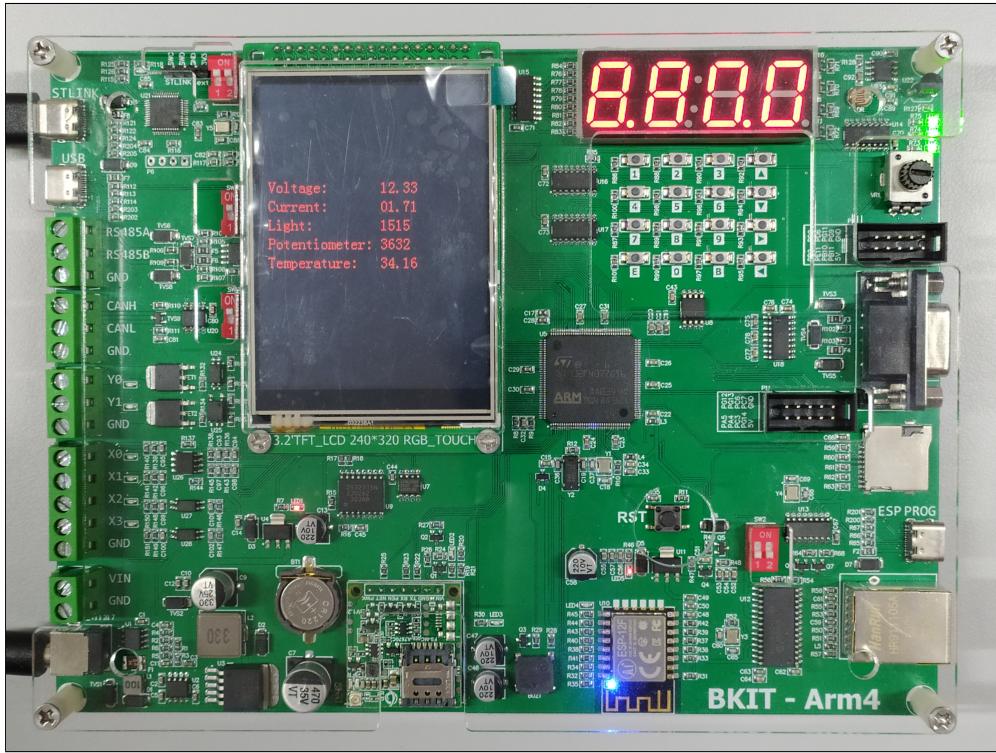
```

    0) ;
109     lcd_ShowIntNum(130, 160, sensor_GetPotentiometer(), 4,
110     RED, BLACK, 16);
111     lcd_ShowStr(10, 180, "Temperature:", RED, BLACK, 16, 0)
112     ;
113     lcd>ShowFloatNum(130, 180, sensor_GetTemperature(), 4,
114     RED, BLACK, 16);
115 }
116
117 void test_Buzzer(){
118     if(isButtonUp()){
119         buzzer_SetVolume(50);
120     }
121
122     if(isButtonDown()){
123         buzzer_SetVolume(0);
124     }
125
126     if(isButtonRight()){
127         buzzer_SetVolume(25);
128     }
129 }
130 /* USER CODE END 4 */

```

Program 6.1: Chương trình mẫu đọc cảm biến và điều khiển buzzer.

Hình ảnh kết quả trên Kit thí nghiệm:



Hình 6.11: Kết quả hiện thực trên Kit thí nghiệm

6 Bài tập và báo cáo

6.1 Bài tập 1

Giả lập hệ thống quan trắc môi trường. Hiển thị các thông số đo được lên màn hình gồm:

- Công suất điện tiêu thụ (mW).
- Ánh sáng (Strong hoặc Weak).
- Nhiệt độ ($^{\circ}\text{C}$).
- Độ ẩm (%), được giả lập bằng biến trỏ, giá từ từ 0-100% tương ứng với khoảng giá trị ADC từ nhỏ nhất tới lớn nhất đọc được từ biến trỏ).
- Led đồng hồ sẽ hiển thị thời gian của hệ thống.

Các dữ liệu đo được sẽ được gửi lên máy tính thông qua Uart-RS232. Khi độ ẩm vượt ngưỡng ($> 70\%$) cho phép hệ thống sau vào trạng thái báo động:

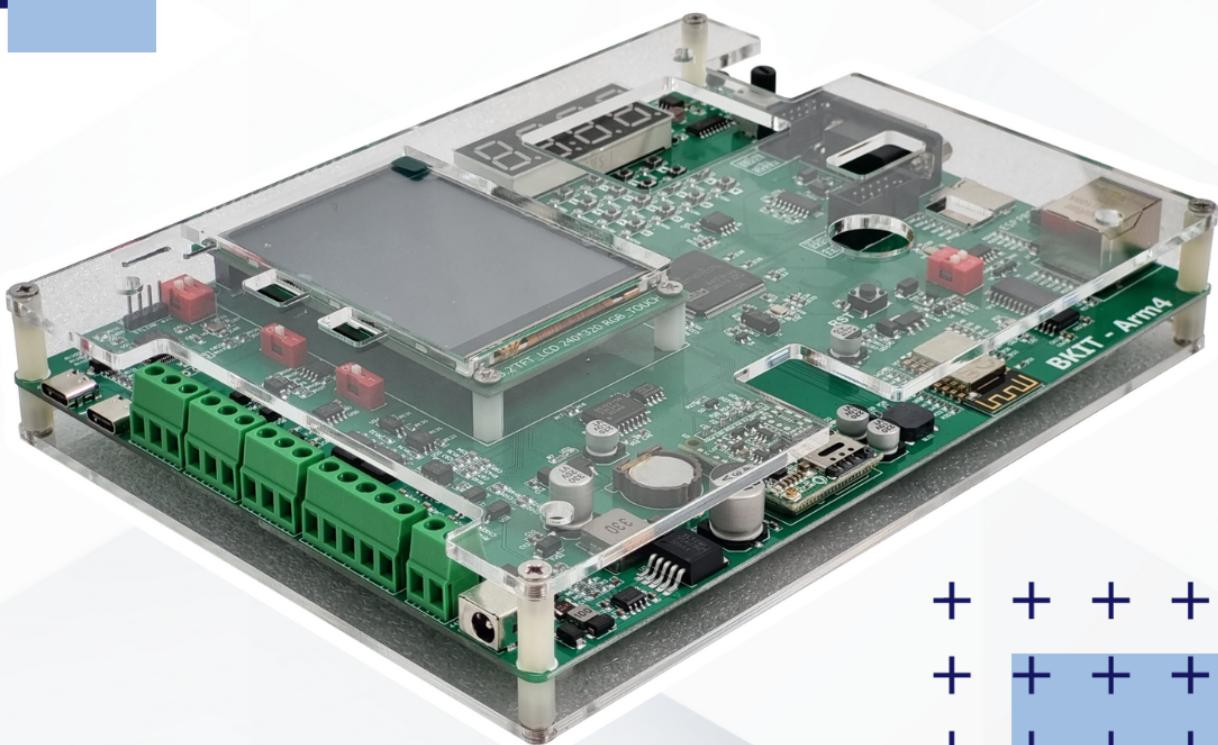
- Buzzer sẽ báo động mỗi 1s.
- Thông báo sẽ được gửi liên tục đến máy tính mỗi 1s.

6.2 Bài tập 2 (Nâng cao)

Trên màn hình LCD, vẽ biểu đồ đường hiển thị công suất tiêu thụ theo thời gian. Giả sử chu kì lấy mẫu là 15s.

CHƯƠNG 7

LCD Touch



1 Mục tiêu

- Tìm hiểu về 2 loại cảm biến touch.
- Tìm hiểu về bộ nhớ EEPROM AT24C.
- Hướng dẫn cách sử dụng màn hình LCD touch, từ cơ bản như hiển thị thông tin đến các tương tác cảm ứng.

2 Giới thiệu

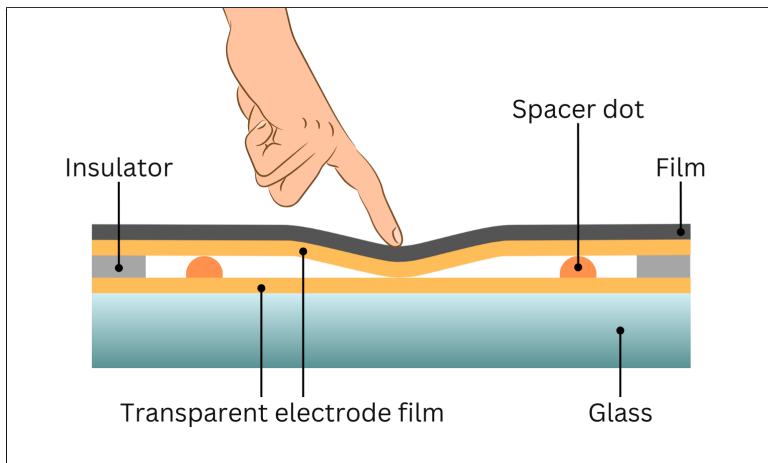
Touch sensor là một thành phần điện tử quan trọng, được sử dụng để phát hiện sự tương tác với môi trường xung quanh thông qua việc chạm hoặc áp lực. Cảm biến chạm được sử dụng rộng rãi từ điện thoại thông minh đến thiết bị gia dụng, giúp tạo ra giao diện người dùng thân thiện và hiệu quả. Cảm biến chạm hoạt động dựa trên nhiều nguyên tắc khác nhau, bao gồm cảm biến điện trở, cảm biến điện dung, cảm biến siêu âm và cảm biến hồng ngoại. Trong tương tác với màn hình, người ta thường quan tâm đến 2 loại cảm biến là cảm biến điện trở và cảm biến điện dung. Trong bài lab này, chúng ta sẽ tìm hiểu về LCD touch sử dụng cảm biến điện trở và bộ nhớ EEPROM AT24C512.

3 Cơ sở lý thuyết

3.1 Touch sensor

3.1.1 Cảm biến điện trở

- Cảm biến chạm điện trở hoạt động bằng cách sử dụng lớp mỏng của vật liệu dẫn điện nằm giữa hai lớp cách điện. Khi có chạm, sự tiếp xúc giữa hai lớp tạo ra một đường dẫn điện, làm thay đổi điện trở của hệ thống.

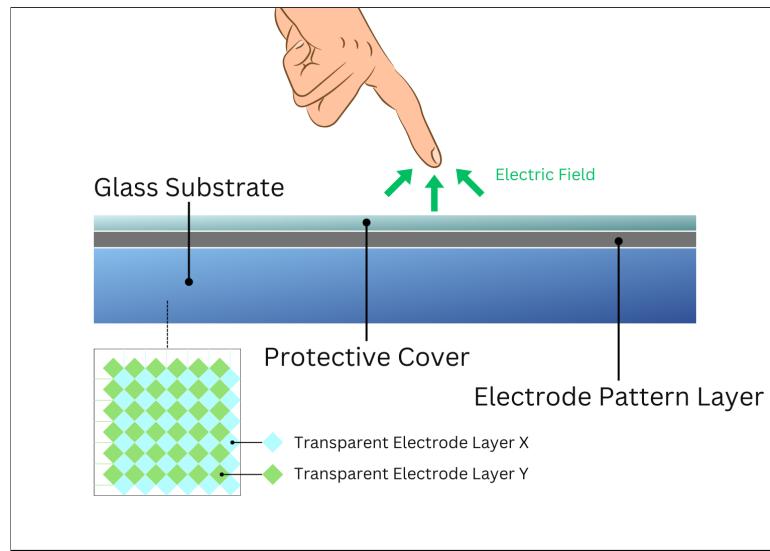


Hình 7.1: Minh họa cảm biến điện trở

- Cảm ứng điện trở xác định vị trí chạm thông qua việc đo điện trở tại các điểm khác nhau trên bề mặt, từ đó có thể xác định được vị trí chạm.
- Cảm biến điện trở thường được sử dụng trong các ứng dụng yêu cầu xác định chính xác vị trí chạm, như màn hình chạm trong các thiết bị đầu cuối công nghiệp.
- So với một số loại cảm biến chạm khác, cảm biến điện trở thường có chi phí thấp hơn.
- Bên cạnh đó, một số mô hình cảm biến điện trở có khả năng chịu ấn(cảm biến áp lực), nghĩa là chúng có thể phản ứng không chỉ với chạm mà còn với áp lực áp dụng lên bề mặt.

3.1.2 Cảm biến điện dung

- Cảm biến điện dung sử dụng nguyên tắc của điện dung để đo lường sự thay đổi trong dung tích của hệ thống khi có sự chạm. Bề mặt cảm biến có thể được làm từ các vật liệu dẫn điện và được cách điện từ môi trường bên ngoài. Nghĩa là khi có một vật dẫn điện (ngón tay hoặc bút cảm ứng) chạm vào cảm biến thì điện dung giữa hai bản kim loại giảm đi, điều đó giống như ngón tay đã “đánh cắp” một lượng điện trường.



Hình 7.2: Minh họa cảm biến điện dung

- Cảm ứng điện dung nhạy cảm và phản ứng nhanh. Cảm biến điện dung thường rất nhạy và có thể phản ứng với những chạm nhẹ từ ngón tay. Phản ứng nhanh giúp tạo ra trải nghiệm người dùng mượt mà và thú vị.
- Cảm ứng điện dung được sử dụng khá phổ biến: màn hình cảm ứng điện thoại thông minh, máy tính bảng; bảng điều khiển trong các thiết bị gia dụng như tủ lạnh thông minh.
- Có thể tạo ra cảm biến điện dung dựa trên dấu vết, nơi một tia xung được gửi qua bề mặt và các thay đổi trong dung tích được đo.

3.2 Bộ nhớ EEPROM AT24C

3.2.1 Quá trình hình thành EEPROM

Từ những năm 1940, bộ nhớ ROM (Read-Only Memory) xuất hiện trong máy tính đầu tiên, lưu trữ dữ liệu không thay đổi. Điều này có nghĩa là dữ liệu không thể thay đổi sau khi được ghi trong quá trình sản xuất. Những năm 1970, EPROM (Erasable Programmable Read-Only Memory) được giới thiệu có khả năng lập trình lại để giải quyết hạn chế của ROM, nhưng cần quá trình xóa bằng tia UV, điều đó làm cho việc ghi lại khá là khó khăn. Để dễ dàng ghi lại hơn, EEPROM (Electrically Erasable Programmable Read-Only Memory) được phát triển, loại bỏ sự cần thiết của tia UV và cho phép xóa và lập trình điện tử mà không cần tháo rời từ mạch điện tử. Từ lúc được giới thiệu cho đến thời điểm hiện tại, EEPROM và các công nghệ lưu trữ tương tự ngày càng phổ biến, liên tục cải tiến để đáp ứng nhu cầu ngày càng cao của công nghệ.

3.2.2 EEPROM AT24C

EEPROM AT24C là một dòng non-volatile memory được sản xuất bởi nhiều nhà sản xuất khác nhau, trong đó tập trung nhiều nhất là dòng sản phẩm của Microchip Technology. Trên Kit thí nghiệm này, chúng ta sẽ làm việc cụ thể với EEPROM AT24C.

- EEPROM AT24C được thiết kế để lưu trữ dữ liệu non-volatile, có khả năng xóa và lập trình lại mà không cần sự tháo rời từ mạch điện tử.
- Có nhiều phiên bản với các dung lượng khác nhau, từ vài kilobits đến vài nghìn kilobits, như AT24C32 (32 Kbits) hoặc AT24C256 (256 Kbits).
- AT24C hỗ trợ giao tiếp chuẩn I2C (Inter-Integrated Circuit), làm cho quá trình kết nối với các vi điều khiển và các thiết bị khác trở nên thuận tiện.
- Thường có thể vận hành ở điện áp thấp như 1.7V, giúp tiết kiệm năng lượng và phù hợp với nhiều ứng dụng di động.
- Sử dụng rộng rãi trong các ứng dụng yêu cầu lưu trữ cài đặt hệ thống, dữ liệu cảm biến, và thông tin cấu hình trong các thiết bị điện tử.

4 Hướng dẫn cấu hình

4.1 Cấu hình Touch:

Trong Kit thí nghiệm, vi điều khiển sẽ cần phải giao tiếp với IC Touch Controller XPT2046 trên bo mạch LCD để quản lý các tác vụ về màn hình cảm ứng. IC trên sẽ giao tiếp với vi điều khiển thông qua SPI và một chân khác để báo hiệu màn hình được chạm. Các chân được cấu hình như bảng sau:

| Ngoại vi | Chân vi điều khiển | Chức năng |
|----------|--------------------|-------------|
| T_CLK | PG8 | GPIO Output |
| T_CS | PG7 | GPIO Output |
| T_MISO | PC12 | GPIO Input |
| T_MOSI | PC9 | GPIO Output |
| T_PEN | PC8 | GPIO Input |

Bảng 7.1: Kết nối ngoại vi

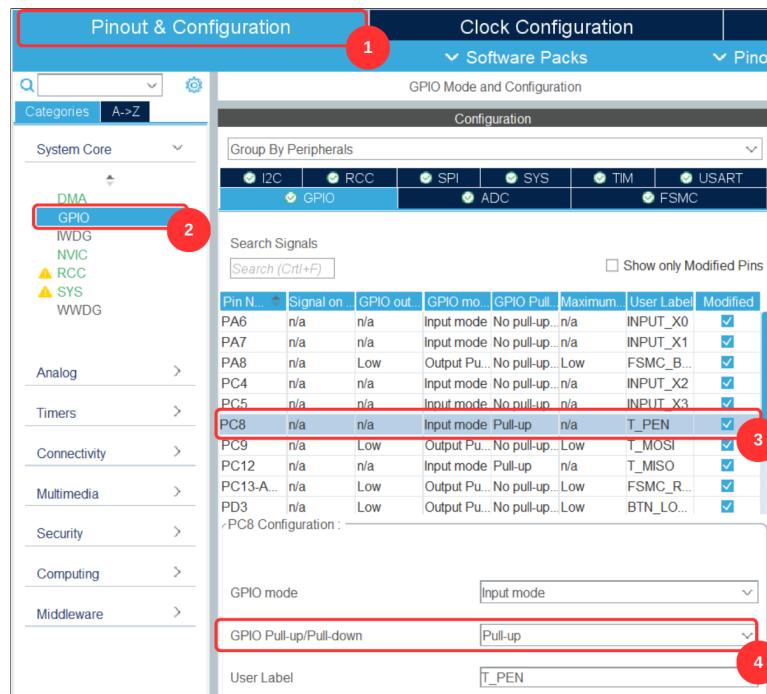
Các chân được dùng để giao tiếp SPI trong phần trên không có chức năng SPI sẵn trên vi điều khiển. Vậy nên ta sẽ config các chân trên là GPIO và sẽ lập trình các

chân này theo quy tắc của SPI. Cách làm này thường được gọi là SPI mềm do được điều khiển bởi phần mềm.



Hình 7.3: Config màn hình cảm ứng

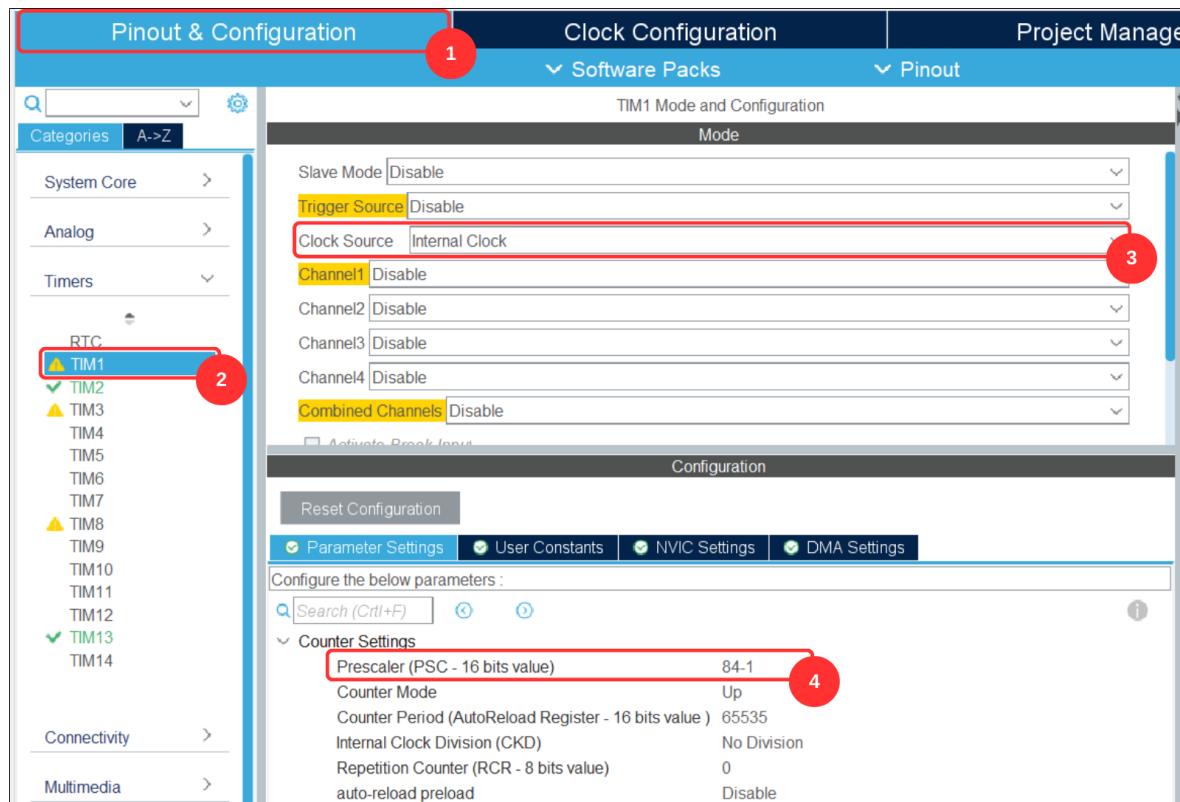
Lưu ý: Các chân Input sẽ phải được config điện trở kéo lên.



Hình 7.4: Config điện trở kéo lên

Để lập trình điều khiển SPI bằng phần mềm, ta sẽ cần hàm để tạo ra độ trễ với đơn vị micro giây. Vì vậy ta sẽ phải cấu hình thêm timer TIM1 cho công việc này. Để có

độ chia là micro giây, tức tần số là 1MHz, với xung clock đầu vào là 84 MHz (xem lại Lab 2) thì ta sẽ phải cấu hình **Prescaler** là **84-1**.



Hình 7.5: Config TIM1

5 Hướng dẫn lập trình

5.1 Thư viện touch.h

Các file touch.h và touch.c có thể được sao chép từ project mẫu **Bai7_TouchScreen**.

void touch_init()

- **Mô tả:** Khởi tạo màn hình cảm ứng.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void touch_Adjust()

- **Mô tả:** Điều chỉnh thông số màn hình cảm ứng.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void touch_Scan()

- **Mô tả:** Đọc màn hình cảm ứng.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

uint8_t touch_IsTouch()

- **Mô tả:** Kiểm tra xem màn hình cảm ứng có được chạm không.
- **Tham số:** Không có.
- **Giá trị trả về:**
 - 1: Được chạm.
 - 0: Không được chạm.

uint8_t touch_GetX0()

- **Mô tả:** Lấy tọa độ X điểm được chạm.

- **Tham số:** Không có.
- **Giá trị trả về:** Tọa độ X.

uint8_t touch_GetY()

- **Mô tả:** Lấy tọa độ Y điểm được chạm.
- **Tham số:** Không có.
- **Giá trị trả về:** Tọa độ Y.

5.2 Thư viện software_timer.c

Trong thư viện này sẽ được thêm các hàm hỗ trợ cho việc tạo delay.

void timer_EnableDelayUs()

- **Mô tả:** Cho phép tạo delay micro giây.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void delay_us(uint16_t us)

- **Mô tả:** Tạo delay micro giây.
- **Tham số:**
 - **us:** thời gian delay.
- **Giá trị trả về:** Không có.

5.3 Thư viện at24c.h

Các file at24c.h và at24c.c có thể được sao chép từ project mẫu **Bai7_TouchScreen**.

void at24c_init()

- **Mô tả:** Khởi tạo kết nối với eeprom.
- **Tham số:** Không có.
- **Giá trị trả về:** Không có.

void at24c_Read(uint16_t ReadAddr,uint8_t *pBuffer,uint16_t NumToRead)

- **Mô tả:** Đọc nhiều byte giá trị từ các thanh ghi liên tiếp.
- **Tham số:**
 - **ReadAddr:** Địa chỉ thanh ghi đầu tiên muốn đọc.
 - **pBuffer:** Mảng giá trị đọc được.
 - **NumToRead:** Số byte muốn đọc.
- **Giá trị trả về:** Không.

void at24c_Write(uint16_t WriteAddr,uint8_t *pBuffer,uint16_t NumToWrite)

- **Mô tả:** Ghi nhiều byte giá trị vào các thanh ghi liên tiếp.
- **Tham số:**
 - **WriteAddr:** Địa chỉ thanh ghi đầu tiên muốn ghi.
 - **pBuffer:** Mảng giá trị muốn ghi.
 - **NumToWrite:** Số byte muốn ghi.
- **Giá trị trả về:** Không.

Sau đây sẽ là phần hướng dẫn xây dựng một ứng dụng đơn giản để kiểm tra hoạt động của màn hình cảm ứng. Ta sẽ xây dựng một ứng dụng cho phép vẽ trên bảng đen và xóa bảng bằng cách chạm vào nút trên màn hình. Ứng dụng trên sẽ được xây dựng bởi máy trạng thái sau:

```
1 #define INIT 0
2 #define DRAW 1
3 #define CLEAR 2
4
5 int draw_Status = INIT;
6
7 uint8_t isButtonClear(){
8     if(!touch_IsTouched()) return 0;
9     return touch_GetX() > 60 && touch_GetX() < 180 &&
10    touch_GetY() > 10 && touch_GetY() < 60;
11}
12
13 void touchProcess(){
14     switch (draw_Status) {
```

```

14    case INIT:
15        // display blue button
16        lcd_Fill(60, 10, 180, 60, GBLUE);
17        lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
18        draw_Status = DRAW;
19        break;
20    case DRAW:
21        if(isButtonClear()){
22            draw_Status = CLEAR;
23            // clear board
24            lcd_Fill(0, 60, 240, 320, BLACK);
25            // display green button
26            lcd_Fill(60, 10, 180, 60, GREEN);
27            lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
28        }
29        break;
30    case CLEAR:
31        if(!touch_IsTouched()) draw_Status = INIT;
32        break;
33    default:
34        break;
35    }
36}

```

Program 7.1: Hiện thực máy trạng thái

Máy trạng thái trên sẽ được xử lí mỗi 50ms. Tuy nhiên, để sử dụng touch sensor, ta cần phải gọi hàm khởi tạo **touch_init**. Để đọc dữ liệu từ màn hình cảm ứng, ta sẽ kiểm tra xem màn hình có được chạm không và đọc giá trị vị trí được chạm. Để tăng độ nhạy của cảm biến, việc kiểm tra và đọc giá trị của màn hình cảm ứng sẽ được thực hiện liên tục. Điều này dẫn đến một số thay đổi cấu trúc trong vòng lặp chính. Lưu ý: hàm **touch_Adjust** chỉ cần gọi khi sử dụng màn hình cảm ứng lần đầu tiên. Mục tiêu của hàm này là để căn chỉnh, lưu lại các giá trị cảm ứng đọc được ở 4 góc màn hình vào eeprom. Cụ thể việc đọc màn hình cảm ứng được thể hiện dưới đây:

```

1 system_init();
2 touch_Adjust();
3 lcd_Clear(BLACK);
4 while (1)
5 {

```

```

6   //scan touch screen
7   touch_Scan();
8   //check if touch screen is touched
9   if(touch_IsTouched() && draw_Status == DRAW){
10       //draw a point at the touch position
11       lcd_DrawPoint(touch_GetX(), touch_GetY(), RED);
12   }
13   // 50ms task
14   if(flag_timer2 == 1){
15       flag_timer2 = 0;
16       touchProcess();
17       test_LedDebug();
18   }
19
20   /* USER CODE END WHILE */
21
22   /* USER CODE BEGIN 3 */
23 }
```

Program 7.2: Vòng lặp vô tận

Khi hiện thực xong chương trình, kết quả là ta có thể vẽ được trên LCD (có thể sử dụng vật cứng như đầu bút tương tác chạm) và có thể xóa màn hình đã vẽ bằng cách chạm vào nút **CLEAR** trên màn hình. Nếu có hàm **touch_Adjust** được gọi, thì ban đầu một màn hình sẽ hiện ra yêu cầu chạm vào các điểm được đánh dấu để lưu lại các thông số căn chỉnh. Dưới đây là toàn bộ source code của hàm file main.c.

```

1 // ...
2 /* USER CODE BEGIN Includes */
3 #include "software_timer.h"
4 #include "led_7seg.h"
5 #include "button.h"
6 #include "lcd.h"
7 #include "picture.h"
8 #include "ds3231.h"
9 #include "sensor.h"
10 #include "buzzer.h"
11 #include "touch.h"
12 /* USER CODE END Includes */
13
```

```

14 // Define statuses
15
16 #define INIT 0
17 #define DRAW 1
18 #define CLEAR 2
19
20 int draw_Status = INIT;
21 // ...
22 /* USER CODE BEGIN PFP */
23 void system_init();
24 void test_LedDebug();
25 void touchProcess();
26 /* USER CODE END PFP */
27
28 int main(void)
29 {
30
31 /* USER CODE BEGIN 2 */
32 system_init();
33 touch_Adjust();
34 lcd_Clear(BLACK);
35 /* USER CODE END 2 */
36
37 /* Infinite loop */
38 /* USER CODE BEGIN WHILE */
39
40 while (1)
41 {
42     //scan touch screen
43     touch_Scan();
44     //check if touch screen is touched
45     if(touch_IsTouched() && draw_Status == DRAW){
46         //draw a point at the touch position
47         lcd_DrawPoint(touch_GetX(), touch_GetY(), RED);
48     }
49     // 50ms task
50     if(flag_timer2 == 1){
51         flag_timer2 = 0;
52         touchProcess();

```

```

53     test_LedDebug();
54 }
55
56 /* USER CODE END WHILE */
57
58 /* USER CODE BEGIN 3 */
59 }
60 /* USER CODE END 3 */
61 }
62
63 // ...
64
65 /* USER CODE BEGIN 4 */
66 void system_init(){
67     timer_init();
68     button_init();
69     lcd_init();
70     touch_init();
71     setTimer2(50);
72 }
73
74 uint8_t count_led_debug = 0;
75
76 void test_LedDebug(){
77     count_led_debug = (count_led_debug + 1)%20;
78     if(count_led_debug == 0){
79         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port , DEBUG_LED_Pin);
80     }
81 }
82
83 uint8_t isButtonClear(){
84     if(!touch_IsTouched()) return 0;
85     return touch_GetX() > 60 && touch_GetX() < 180 &&
86     touch_GetY() > 10 && touch_GetY() < 60;
87 }
88
89 void touchProcess(){
90     switch (draw_Status) {
91         case INIT:

```

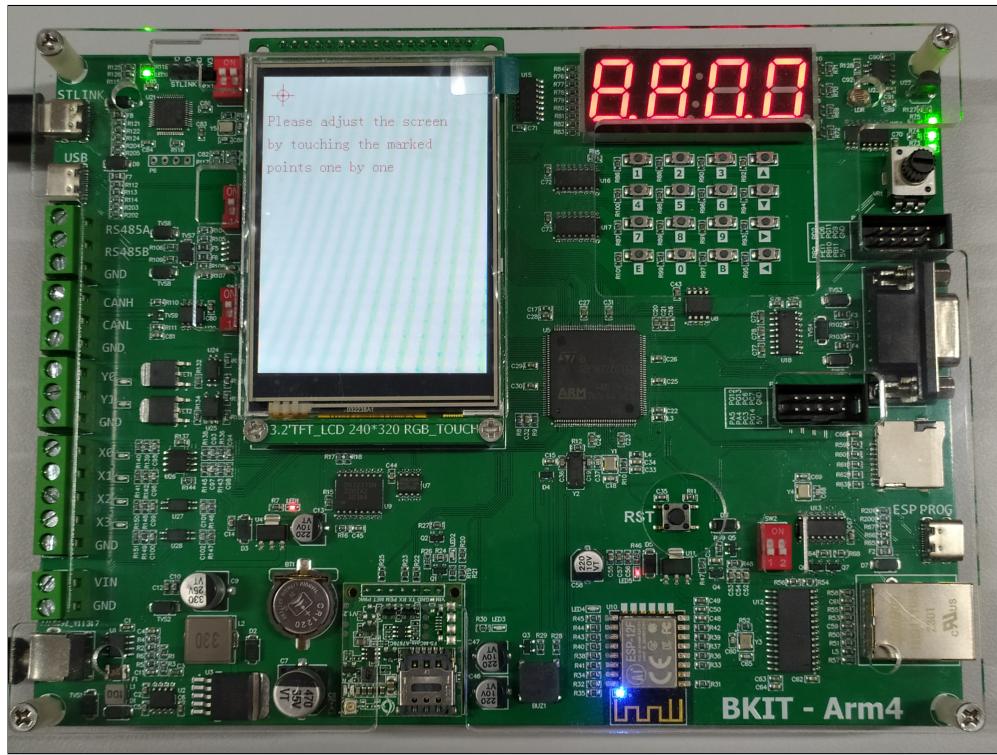
```

91                     // display blue button
92         lcd_Fill(60, 10, 180, 60, GBLUE);
93         lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
94         draw_Status = DRAW;
95         break;
96     case DRAW:
97         if(isButtonClear()){
98             draw_Status = CLEAR;
99                 // clear board
100            lcd_Fill(0, 60, 240, 320, BLACK);
101                 // display green button
102            lcd_Fill(60, 10, 180, 60, GREEN);
103            lcd_ShowStr(90, 20, "CLEAR", RED, BLACK, 24, 1);
104        }
105        break;
106    case CLEAR:
107        if(!touch_IsTouched()) draw_Status = INIT;
108        break;
109    default:
110        break;
111    }
112}
113 /* USER CODE END 4 */

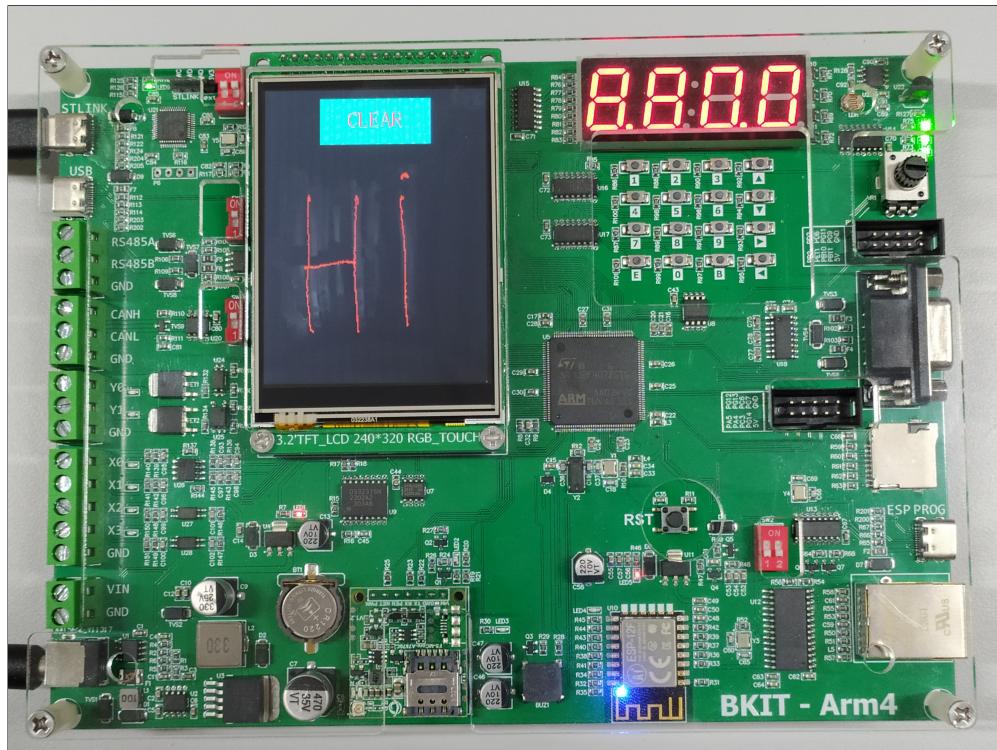
```

Program 7.3: Chương trình mẫu kiểm tra mà hình cảm ứng

Hình ảnh kết quả trên Kit thí nghiệm:



Hình 7.6: Kết quả hiển thị thực trên Kit thí nghiệm (1)



Hình 7.7: Kết quả hiển thị thực trên Kit thí nghiệm (2)

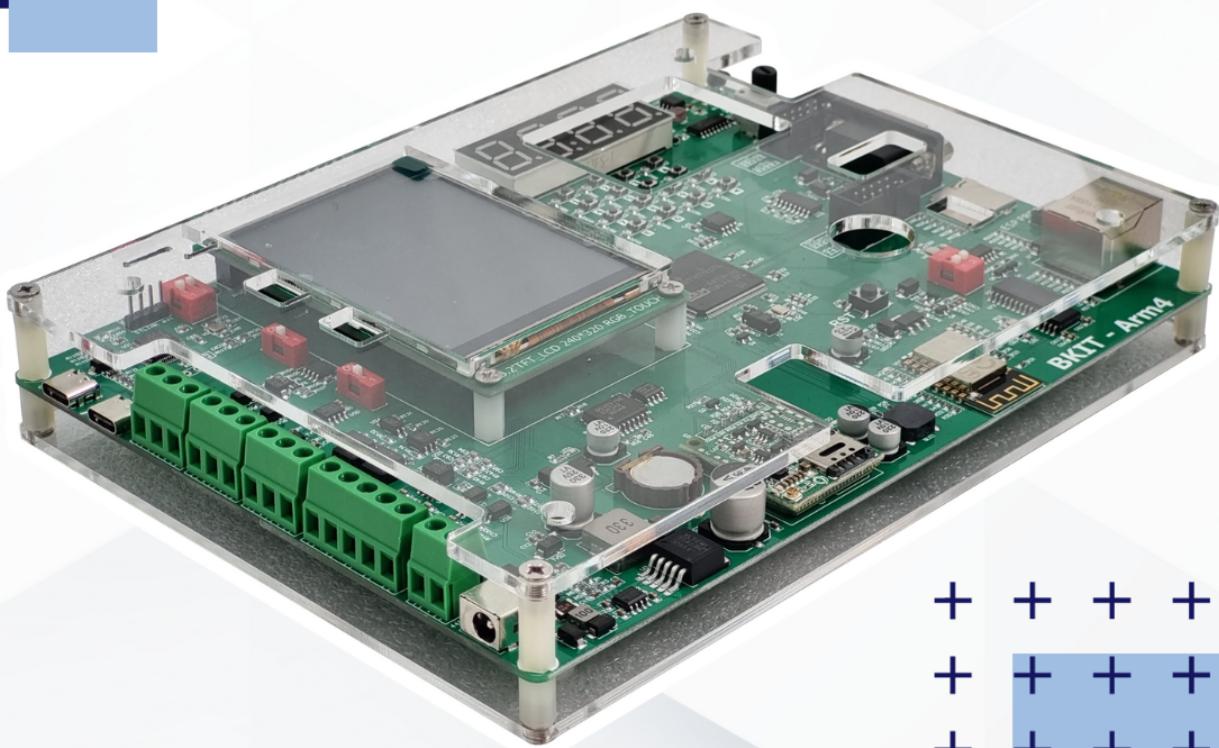
6 Bài tập và báo cáo

Hiện thực trò chơi rắn săn mồi cổ điển trên LCD. Trong đó, toàn bộ tương tác với người dùng sẽ được thao tác trên màn hình cảm ứng, ví dụ như:

- Chạm vào nút "Start" trên màn hình để bắt đầu trò chơi.
- Chạm vào các nút điều hướng trên màn hình để điều khiển con rắn.

CHƯƠNG 8

ESP8266 - WIFI



1 Mục tiêu

- Tìm hiểu về ESP8266 ESP-12 của Ai-Thinker.
- Hướng dẫn sử dụng ESP8266 trên mạch và biết cách giao tiếp với chip chính.
- Kết nối Wifi và điều khiển thông qua Adafruit bằng giao thức MQTT.

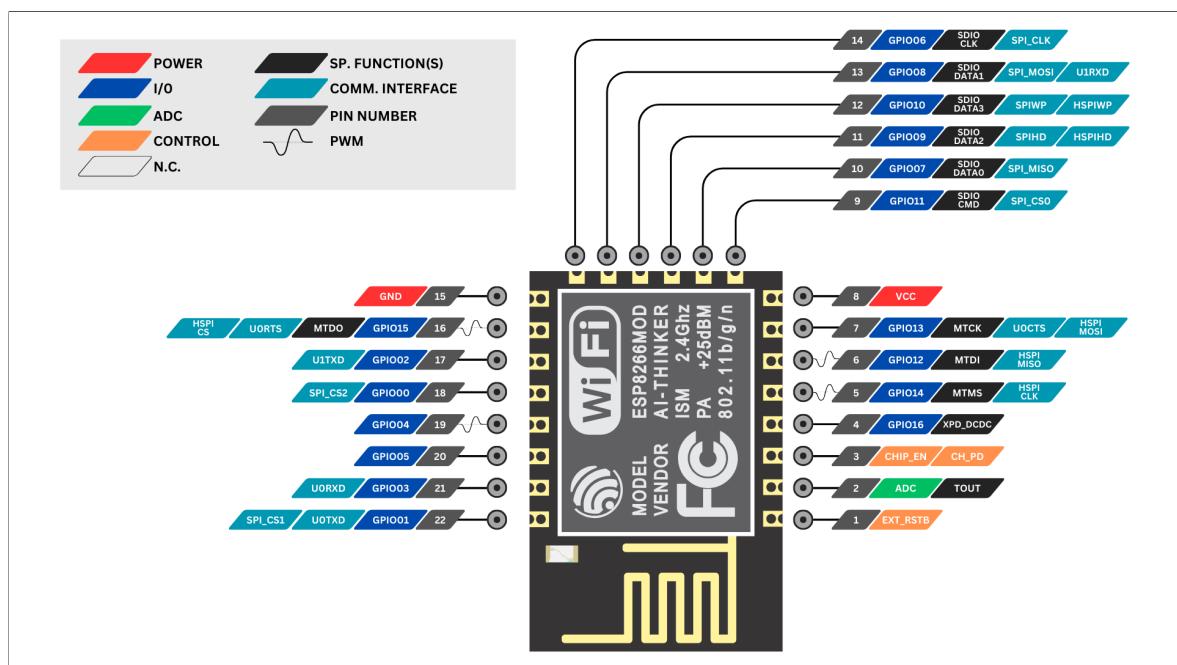
2 Giới thiệu

- Mạch thu phát Wifi ESP8266 ESP-12E của Ai-Thinker là một giải pháp Wifi tiết kiệm và hiệu quả cho các ứng dụng liên quan đến Internet và Wifi. Nó tích hợp thêm bộ khuếch đại công suất (PA) và bộ lọc tín hiệu RF (LNA), cải thiện khả năng thu phát sóng Wifi. Mạch có kích thước nhỏ gọn, chất lượng đáng tin cậy, vỏ kim loại chống nhiễu và anten Wifi PCB tích hợp. ESP8266 cũng hỗ trợ giao tiếp SPI, I2C, UART, ADC, PWM và các chức năng GPIO.
- Adafruit.io là một dịch vụ đám mây (cloud service) do Adafruit cung cấp, được thiết kế đặc biệt để hỗ trợ việc kết nối và quản lý dữ liệu từ các thiết bị IoT. Nó cung cấp giao diện đơn giản để thu thập, lưu trữ và truy xuất dữ liệu từ các cảm biến và thiết bị IoT, giúp người dùng dễ dàng tạo và theo dõi các ứng dụng IoT của mình. Trong bài lab này chúng ta sẽ sử dụng giao thức MQTT để giao tiếp giữa Adafruit.io và ESP8266. MQTT là một giao thức truyền thông nhẹ được sử dụng trong IoT, cho phép truyền dữ liệu giữa các thiết bị và máy chủ một cách hiệu quả thông qua cơ chế publish/subscribe.

3 Cơ sở lý thuyết

3.1 Giới thiệu về ESP8266

- Chip ESP8266EX sử dụng CPU 32-bit Tensilica Xtensa LX106, được tối ưu hóa cho các ứng dụng IoT. Bộ nhớ Flash tích hợp trực tiếp vào chip, giúp ESP8266EX lưu trữ chương trình và dữ liệu một cách hiệu quả. ESP-12E có 22 chân GPIO cho phép kết nối với nhiều thiết bị ngoại vi khác nhau. Mỗi chân có thể được cấu hình để thực hiện nhiều chức năng khác nhau, như GPIO, UART, SPI, I2C, ADC, PWM có thể kết nối với nhiều module khác.



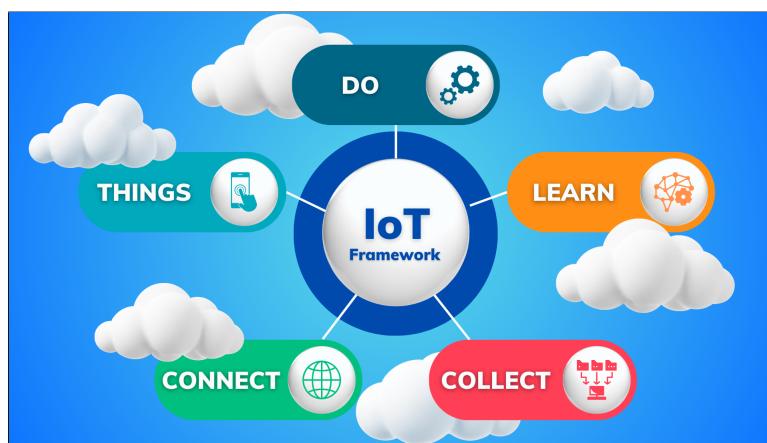
Hình 8.1: Sơ đồ chân của ESP8266 ESP-12E

- ESP-12E hỗ trợ các chuẩn Wi-Fi phổ biến, bao gồm 802.11 b/g/n, giúp nó kết nối với nhiều loại mạng Wifi. Ngoài ra, nó có thể hoạt động ở chế độ Station (kết nối với Access Point), Access Point (tạo mạng Wifi), hoặc cả hai. ESP-12E thường đi kèm với Flash có dung lượng từ 4MB đến 16MB, cung cấp không gian lưu trữ đủ cho chương trình và dữ liệu. Sử dụng RAM để lưu trữ dữ liệu trong quá trình chạy chương trình.
- Chế Độ Ngủ và Tiết Kiệm Năng Lượng:
 - Trong chế độ ngủ: ESP8266EX tiêu thụ rất ít năng lượng, thích hợp cho các ứng dụng yêu cầu tiết kiệm năng lượng cao.

- Trong chế độ tiết kiệm năng lượng: Cắt giảm một số chức năng để giảm tiêu thụ năng lượng, nhưng vẫn duy trì khả năng đánh thức nhanh. ESP8266 có thể được lập trình bằng Arduino IDE, môi trường lập trình phổ biến, giúp người dùng dễ dàng phát triển ứng dụng IoT. Bên cạnh đó, ESP8266 ESP-12E có sẵn nhiều thư viện hỗ trợ giúp lập trình viên tận dụng mọi khả năng của ESP8266.

3.2 Kiến trúc 5 lớp của IoT

- Theo tiến sĩ Timothy Chou, giảng viên Đại học Stanford, kiến trúc về ứng dụng thông minh dựa trên IoT được chia thành mô hình 5 lớp như sau:



Hình 8.2: Kiến trúc 5 tầng của IoT

- Chức năng chính của từng lớp trong kiến trúc này được khái quát như sau:
 - Things: Các thiết bị trong ứng dụng giám sát là rất đa dạng về chức năng và số lượng. Tùy vào ứng dụng, nhiều loại cảm biến khác nhau sẽ được sử dụng. Các nút cảm biến chủ yếu sử dụng giao tiếp không dây.
 - Connect: Chúng tôi thu thập dữ liệu từ các nút cảm biến. Đôi với mỗi loại ứng dụng, có nhiều tiêu chuẩn kết nối khác nhau, đòi hỏi lớp này phải hỗ trợ nhiều loại kết nối, từ Zigbee và Wifi trong các ứng dụng nhà thông minh, đến các giao tiếp rộng hơn như LoRa hay 3G/4G.
 - Collect: Dữ liệu thu thập được sẽ được gửi lên các máy chủ tập trung để lưu trữ. Tại đây, một lượng lớn dữ liệu sẽ được gửi về, đặt ra thách thức lớn cho máy chủ và cần áp dụng công nghệ Big Data.
 - Learn: Lớp này có nhiệm vụ lọc ra các thông tin đặc trưng, có ý nghĩa cụ thể cho từng loại ứng dụng. Công nghệ Machine Learning và hiện tại là Deep Learning sẽ được sử dụng ở đây.

- Do: Dựa vào các thông tin đặc trưng, hệ thống sẽ xây dựng các quy luật thích nghi với môi trường và đưa ra các quyết định cho hệ thống. Sự thực thi của mỗi quyết định sẽ được đo lường tự động, và sai lệch so với mục tiêu tối ưu sẽ được xem xét cho lần sau. Theo cách này, hệ thống sẽ tự học kinh nghiệm để ngày càng hoàn thiện hơn.

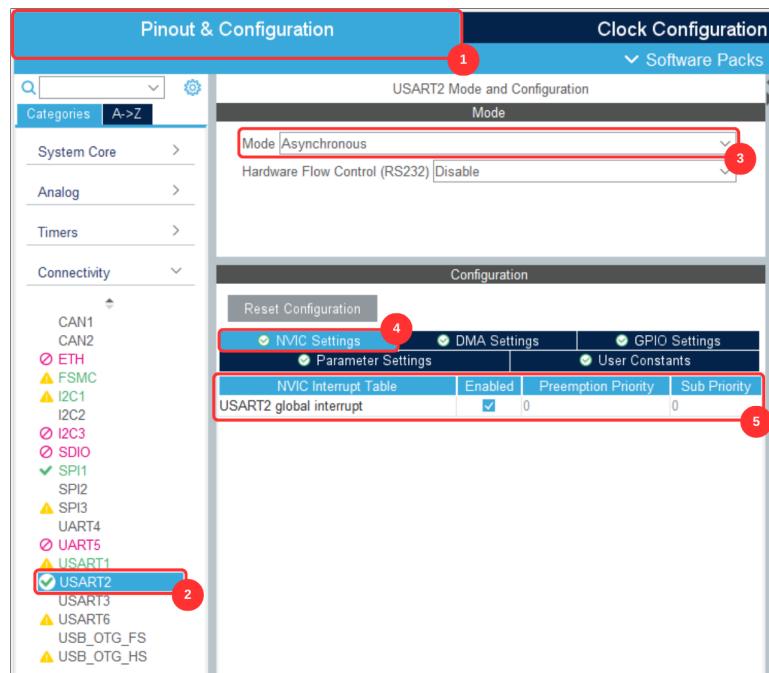
4 Hướng dẫn cấu hình

Trong Kit thí nghiệm, vi điều khiển sẽ cần phải giao tiếp với ESP8266 thông qua USART2 để truyền nhận dữ liệu. Các chân được cấu hình như bảng sau:

| Ngoại vi | Chân vi điều khiển | Chức năng |
|-----------|--------------------|-------------|
| ESP_POWER | PF10 | GPIO Output |
| ESP_BUSY | PF9 | GPIO Input |
| ESP_TX | PA3 | USART2_RX |
| ESP_RX | PA2 | USART2_TX |

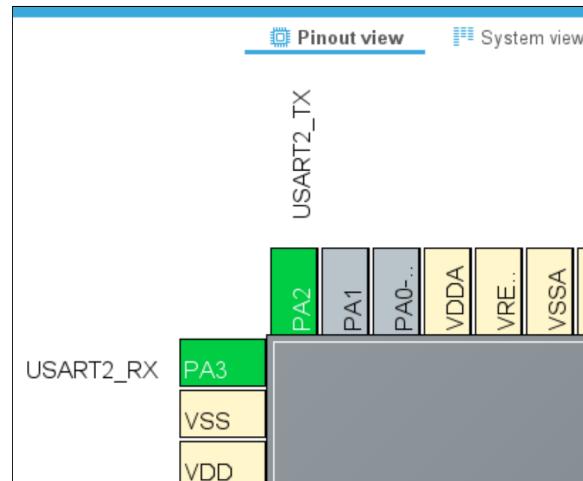
Bảng 8.1: Kết nối ngoại vi

Để chúng giao tiếp được với nhau, theo datasheet ta sẽ config USART2 như sau:



Hình 8.3: Config USART2

Sau khi config, ta có pinview out như hình sau:



Hình 8.4: Pinout view của USART2 sau khi config

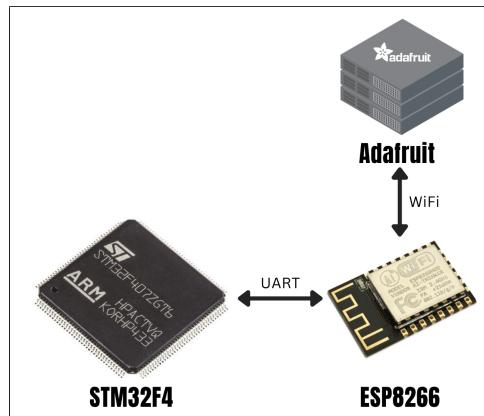
Hai chân ESP12_BUSY và ESP12_PWR sau khi được config theo bảng 8.1 sẽ có pin-view out như hình sau:



Hình 8.5: Config hai chân ESP12_BUSY và ESP12_PWR

5 Hướng dẫn lập trình

Trong bài lab này, chúng ta sẽ gửi tín hiệu chớp tắt đèn bằng nút nhấn hoặc từ server Adafruit theo mô hình sau:

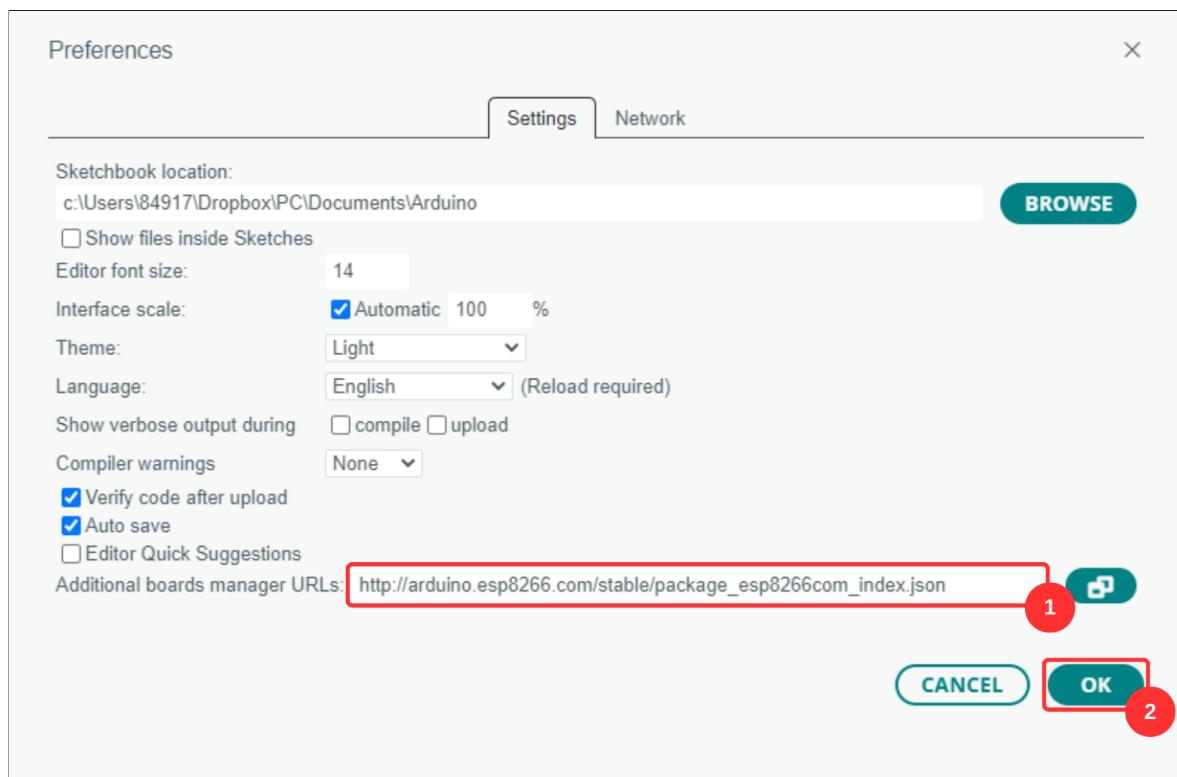


Hình 8.6: Mô hình của bài hướng dẫn

5.1 Hướng dẫn lập trình ESP8266 trên Arduino IDE

Trong phần này, chúng ta sẽ lập trình cho ESP8266 bằng Arduino IDE, bạn có thể tải IDE tại đây.

Ta sẽ chọn board **Generic ESP8266 Module**. Nếu không có sẵn board trên, bạn có thể thêm board bằng cách: File → Preferences và thêm url: http://arduino.esp8266.com/stable/package_esp8266com_index.json (hình 8.7).



Hình 8.7: Thêm board ESP8266

Dưới đây là cấu trúc chương trình khi lập trình trên Arduino IDE. Phần **setup** sẽ được dùng để thêm các đoạn code khởi tạo, chỉ chạy một lần. Phần **loop** sẽ chứa các đoạn code được lặp lại liên tục.

```
1 void setup() {  
2     // put your setup code here, to run once:  
3 }  
4  
5 void loop() {  
6     // put your main code here, to run repeatedly:  
7 }
```

Program 8.1: Cấu trúc chương trình



Hình 8.8: LED xanh được kết nối với chân GPIO2 trên ESP8266

Để kiểm tra hoạt động của ESP, ta sẽ thực hiện chương trình nháy LED. Trên ESP8266 có một đèn LED màu xanh được kết nối với chân GPIO2 (hình 8.8). Ta cần sử dụng 2 hàm cơ bản là **pinMode** để cấu hình chân và **digitalWrite** để ghi tín hiệu ra chân Output. Ngoài ra, ở cuối **loop** ta sẽ có câu lệnh **delay(10)**. Như vậy các câu lệnh ở trong loop sẽ được hiện thực mỗi 10ms. Tương tự như cách lập trình trên STM32, ta thêm biến đếm để thực hiện các công việc có chu kì lớn hơn. Đoạn code nhấp nháy LED mỗi 1s sẽ được minh họa dưới đây.

```

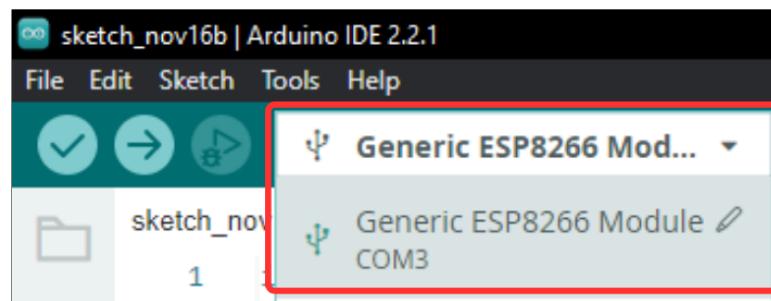
1 int led_counter = 0;
2 int led_status = HIGH;
3 void setup() {
4     // put your setup code here, to run once:
5     //set pin 2 as OUTPUT
6     pinMode(2, OUTPUT);
7 }
8
9 void loop() {
10    // put your main code here, to run repeatedly:
11    led_counter++;
12    if(led_counter == 100){
13        // every 1s
14        led_counter = 0;
15        //toggle LED
16        if(led_status == HIGH) led_status = LOW;
17        else led_status = HIGH;
18
19        digitalWrite(2, led_status);
20    }

```

```
21  
22     delay (10) ;  
23 }
```

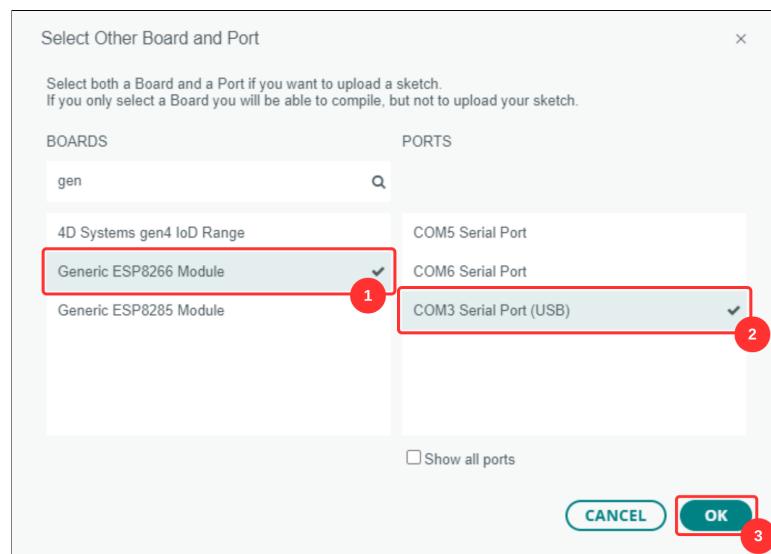
Program 8.2: Chương trình nháy LED

Để nạp chương trình cho ESP, ta cần kết nối cổng nạp **ESP PROG** với máy tính thông qua USB Type_C. Sau đó, ta điều chỉnh **SW2** trên Kit thí nghiệm ở trạng thái ON-ON. Tiếp theo, ta kết nối phần mềm Arduino IDE với ESP như hình sau:



Hình 8.9: Chọn board trong Arduino

Sau khi chọn board, cửa sổ chọn sau hiện ra để bạn biết máy tính của mình đang có thể kết nối với các thiết bị nào:



Hình 8.10: Chọn Board và Port cho ESP

Trường hợp không phát hiện được **COM Port** thì cần phải cài đặt Driver CH340 ở link sau: Driver CH340

Lưu ý: sau khi nạp chương trình thành công, ta phải điều chỉnh **SW2** trở lại trạng thái OFF-OFF thì ESP mới có thể giao tiếp với STM32.

ESP8266 ESP-12E được thiết kế trên Kit thí nghiệm với mục đích kết nối Internet. Trong bài lab này, ta sẽ kết nối Internet thông qua giao thức phổ biến nhất là Wifi.

```
1 #include <ESP8266WiFi.h>
2
3 //Wifi name
4 #define WLAN_SSID      "Your_Wifi_Name"
5 //Wifi password
6 #define WLAN_PASS      "Your_Wifi_Password"
7
8 int led_counter = 0;
9 int led_status = HIGH;
10
11 void setup() {
12     // put your setup code here, to run once:
13     //set pin 2,5 as OUTPUT
14     pinMode(2, OUTPUT);
15     pinMode(5, OUTPUT);
16     //set busy pin HIGH
17     digitalWrite(5, HIGH);
18
19     //connect Wifi
20     WiFi.begin(WLAN_SSID, WLAN_PASS);
21     while (WiFi.status() != WL_CONNECTED) {
22         delay(500);
23     }
24
25     //finish setup, set busy pin LOW
26     digitalWrite(5, LOW);
27
28 }
29
30 void loop() {
31     // put your main code here, to run repeatedly:
32     led_counter++;
33     if(led_counter == 100){
34         // every 1s
```

```

35     led_counter = 0;
36     //toggle LED
37     if(led_status == HIGH) led_status = LOW;
38     else led_status = HIGH;
39
40     digitalWrite(2, led_status);
41 }
42 delay(10);
43 }
```

Program 8.3: Kết nối Wifi

Chương trình trên là một ví dụ cho việc ESP8266 kết nối Internet bằng giao thức Wifi. Một số điểm cần lưu ý:

- Thay thế tên Wifi và password ở các dòng 4 và 6.
- Các dòng từ 20 → 23 được dùng để kết nối Wifi.
- Chân GPIO5 của ESP được cấu hình là Output và được sử dụng để báo cho STM32F4 biết ESP đang trong trạng thái BUSY hay không. Trong ví dụ này, tín hiệu từ GPIO5 ở mức HIGH thì nó đang trong trạng thái BUSY. Trên Kit thí nghiệm, trạng thái này được thể hiện ở **LED4**.
- Khi đoạn chương trình trên được thực hiện, **LED4** sẽ ở trạng thái sáng, sau khi Wifi được kết nối thành công thì **LED4** sẽ chuyển sang trạng thái tắt. Lúc này, LED xanh ở trên ESP sẽ chớp tắt.

Như đã đề cập ở phần trước, ESP và STM32 giao tiếp với nhau thông qua UART. Ngoài việc config USART2 trên STM32 như phần trên, ta cần thiết lập giao tiếp UART trên ESP trong đoạn code dưới đây:

```

1 void setup() {
2     // put your setup code here, to run once:
3
4     //set pin 2,5 as OUTPUT
5     pinMode(2, OUTPUT);
6     pinMode(5, OUTPUT);
7     //set busy pin HIGH
8     digitalWrite(5, HIGH);
9
10    Serial.begin(115200);
11 }
```

```

12 //connect Wifi
13 WiFi.begin(WLAN_SSID, WLAN_PASS);
14 while (WiFi.status() != WL_CONNECTED) {
15     delay(500);
16 }
17
18 //finish setup, set busy pin LOW
19 digitalWrite(5, LOW);
20
21 }
22
23 void loop() {
24     // put your main code here, to run repeatedly:
25
26     if(Serial.available()){
27         int msg = Serial.read();
28         if(msg == 'o') Serial.print('0');
29     }
30
31     led_counter++;
32     if(led_counter == 100){
33         // every 1s
34         led_counter = 0;
35         //toggle LED
36         if(led_status == HIGH) led_status = LOW;
37         else led_status = HIGH;
38
39         digitalWrite(2, led_status);
40     }
41     delay(10);
42 }

```

Program 8.4: Giao tiếp với STM32

Trong đoạn chương trình trên:

- Tại dòng 10, ta thiết lập giao tiếp UART với baudrate 115200, giá trị này phải thống nhất với giá trị đã được thiết lập trên STM32.
- Dòng 26 → 29, thực hiện việc nhận tín hiệu UART từ STM32 và xử lí. Trong ví dụ này, ta sẽ kiểm tra kết nối bằng cách gửi lại kí tự 'O' nếu nhận được kí tự

'o' từ STM32.

5.2 Hướng dẫn làm việc với Adafruit

5.2.1 Tạo tài khoản và kênh dữ liệu

Adafruit.io là một dịch vụ đám mây (cloud service) do Adafruit cung cấp, được thiết kế đặc biệt để hỗ trợ việc quản lý và truyền tải dữ liệu từ các thiết bị IoT. Server này chúng ta có thể sử dụng server miễn phí tại: <https://io.adafruit.com/>. Sau khi truy cập, ta chọn **Get Started for Free** để đăng ký. Nhập đầy đủ thông tin sau đó chọn **CREATE ACCOUNT** để tạo tài khoản.

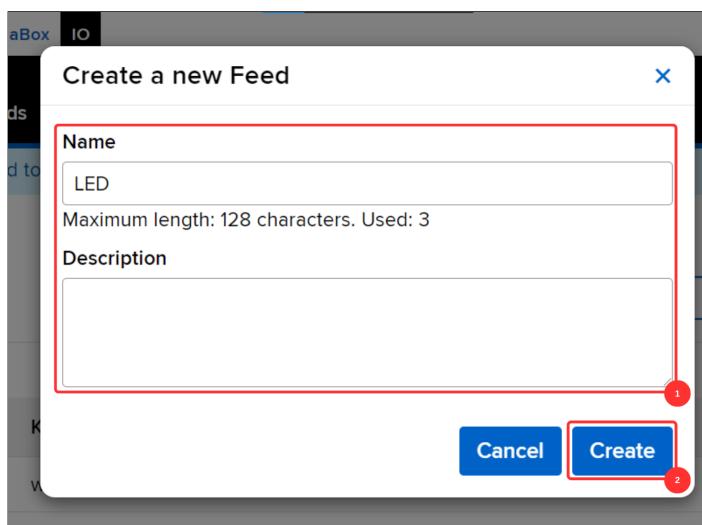
The screenshot shows the Adafruit sign-up interface. At the top, there is a navigation bar with links: Shop, Learn, Blog, Forums, LIVE!, AdaBox, and IO. Below the navigation bar is the Adafruit logo. The main section is titled "SIGN UP". On the left, there is a descriptive text: "The best way to shop with Adafruit is to create an account which allows you to shop faster, track the status of your current orders, review your previous orders and take advantage of our other member benefits." To the right of this text are five input fields labeled "FIRST NAME", "LAST NAME", "EMAIL", "USERNAME", and "PASSWORD". Below the "USERNAME" field, there is a note: "Username is viewable to the public on the forums, Adafruit IO, and elsewhere." At the bottom of the form are two buttons: a blue "CREATE ACCOUNT" button and a blue "SIGN IN" button below it.

Hình 8.11: Màn hình đăng ký Adafruit

Trong bài lab này, chúng ta sẽ làm việc ở trong IO, trước tiên là trong phần kênh dữ liệu (feed). Chúng ta tạo một feed như sau:

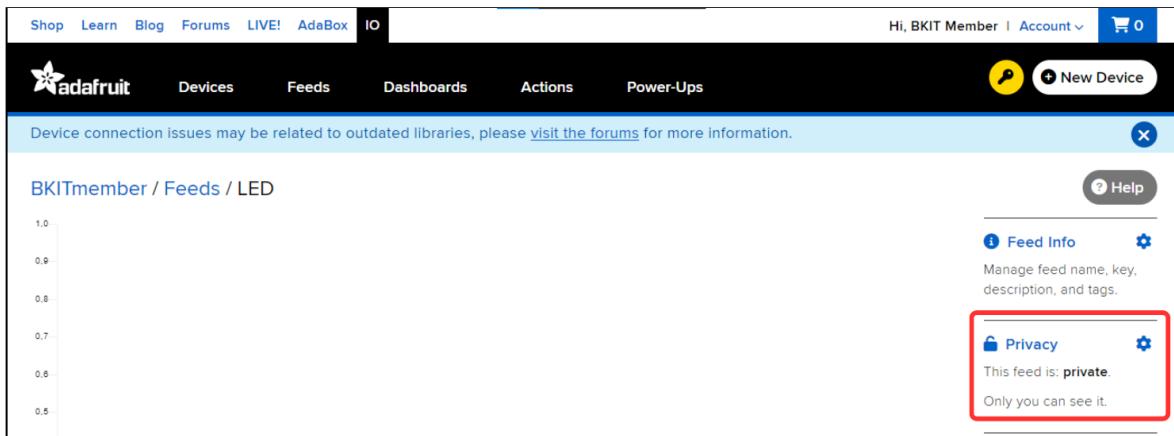
The screenshot shows the Adafruit IO web interface. At the top, there's a navigation bar with links for Shop, Learn, Blog, Forums, LIVE!, AdaBox, IO (highlighted with a red box and number 1), Devices, Feeds (highlighted with a red box and number 2), Dashboards, Actions, Power-Ups, and a user account section. Below the navigation is a message about device connection issues. The main area is titled 'BKITMember / Feeds' and shows two buttons: '+ New Feed' (highlighted with a red box and number 3) and '+ New Group'. A search bar and a help button are also present. The feed list table has columns for Feed Name, Key, Last value, and Recorded. One entry is shown: 'Welcome Feed' with key 'welcome-feed', last value 'welcome-feed', recorded 9 minutes ago, and a lock icon.

Hình 8.12: Tạo một feed mới

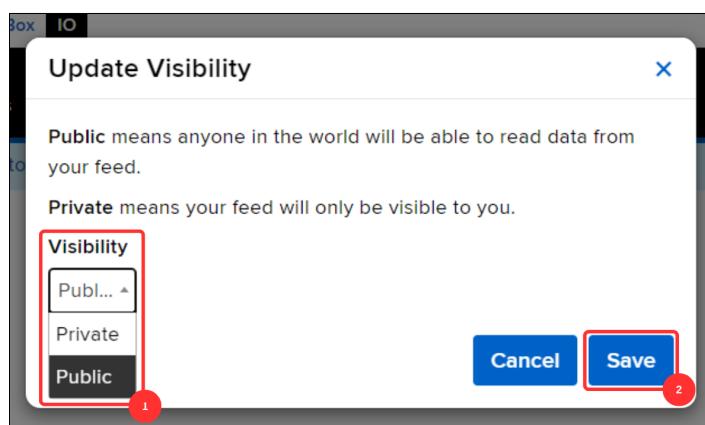


Hình 8.13: Đặt tên, mô tả và tạo feed

Sau khi tạo được một feed, chúng ta sẽ chọn feed và chuyển nó sang chế độ public (hình 8.14 và 8.15). Việc đổi chế độ này làm cho quá trình lập trình trở nên thuận tiện hơn.



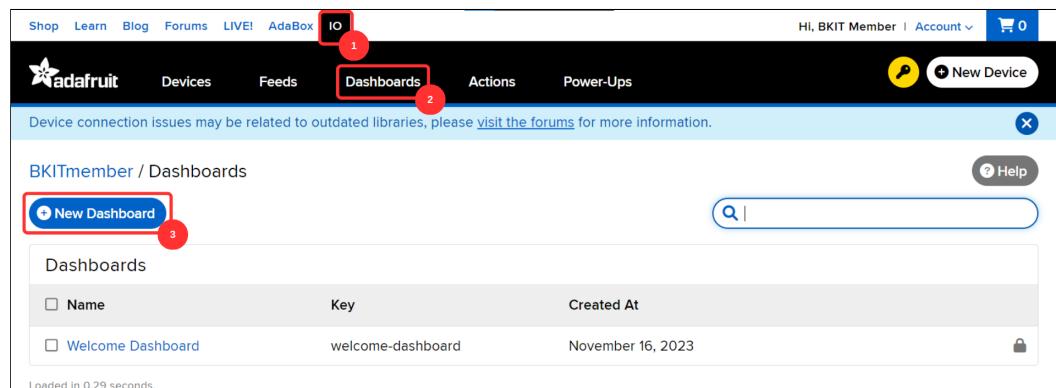
Hình 8.14: Chọn biểu tượng cài đặt tại mục Privacy



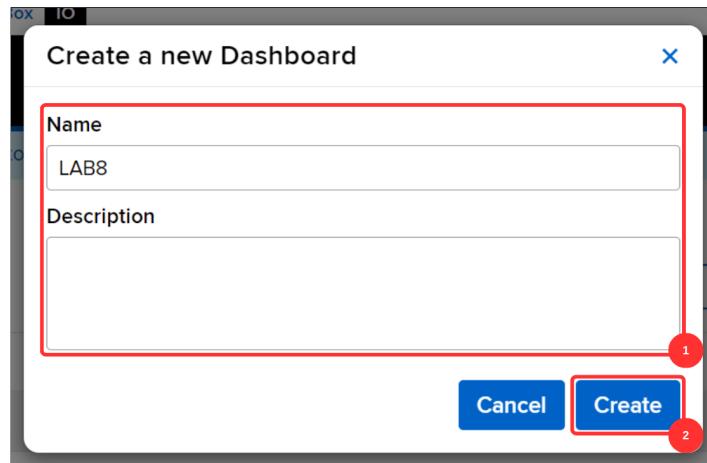
Hình 8.15: Chọn chế độ Public và Save

5.2.2 Tạo dashboard và kết nối với feed

Trong phần này, ta sẽ tạo một dashboard với một nút nhấn, sau đó liên kết dashboard và feed dữ liệu và kiểm tra tương tác giữa chúng. Trước tiên, chúng ta sẽ tạo một dashboard mới:

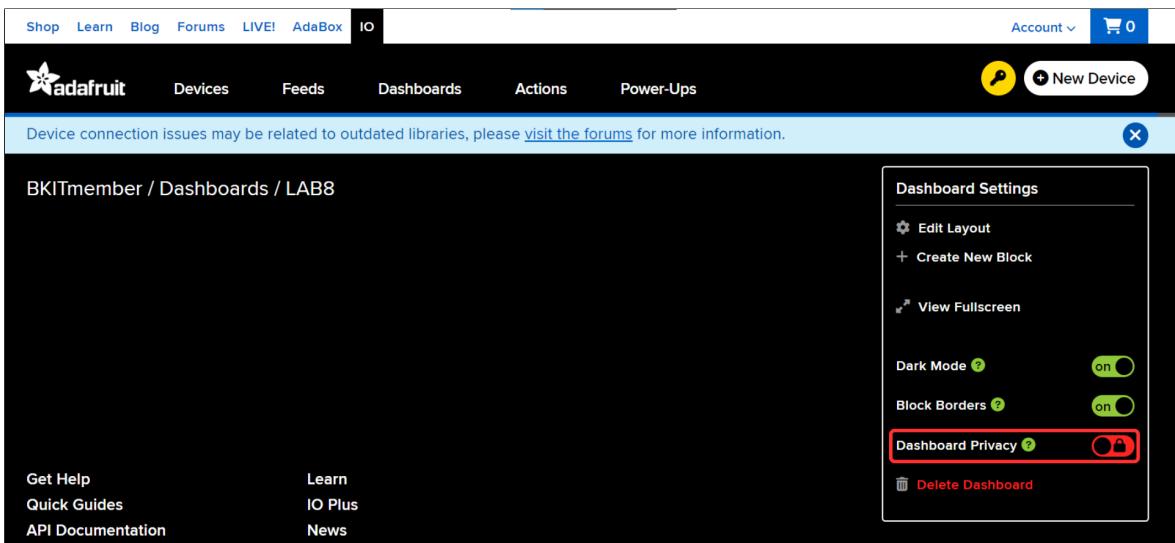


Hình 8.16: Tạo dashboard



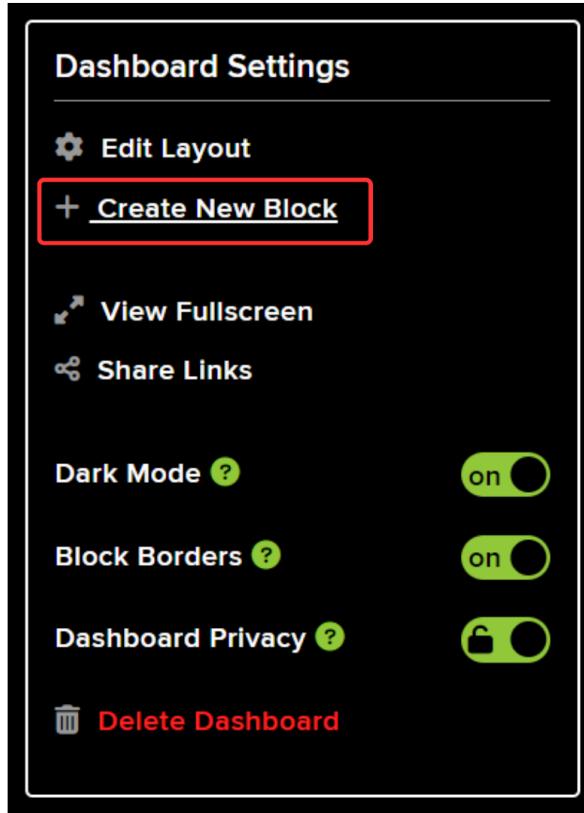
Hình 8.17: Đặt tên, mô tả và tạo dashboard

Sau khi tạo được một dashboard, chúng ta chọn dashboard và chuyển nó sang chế độ public (hình 8.18). Ta sẽ cần **Confirm** để xác nhận việc chuyển đổi. Việc này làm cho quá trình lập trình trở nên thuận tiện hơn.



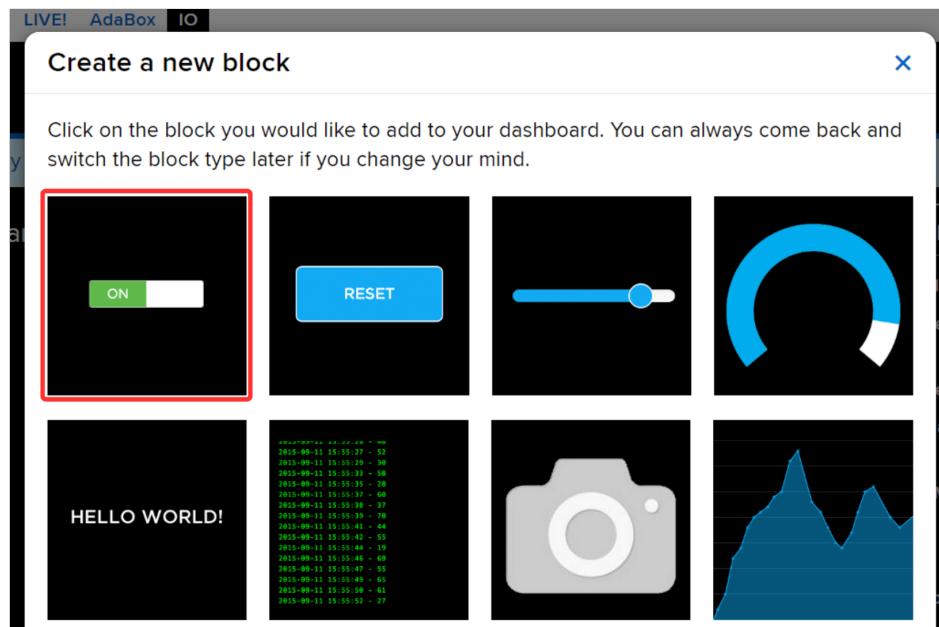
Hình 8.18: Chuyển dashboard sang public

Bây giờ, ta sẽ tiến hành tạo một block để thực hiện việc bật tắt đèn:



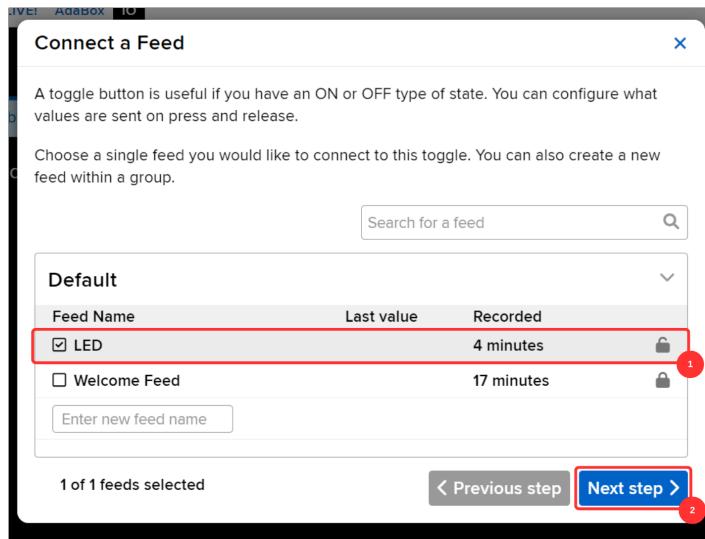
Hình 8.19: Tạo block mới

Sau khi chọn **Create New Block** thì hộp thoại như hình 8.20 sẽ xuất hiện, chúng ta có thể chọn những block phù hợp với project mà chúng ta đang làm. Trong bài lab này, chúng ta sẽ chọn Toggle button block, như hình dưới đây:



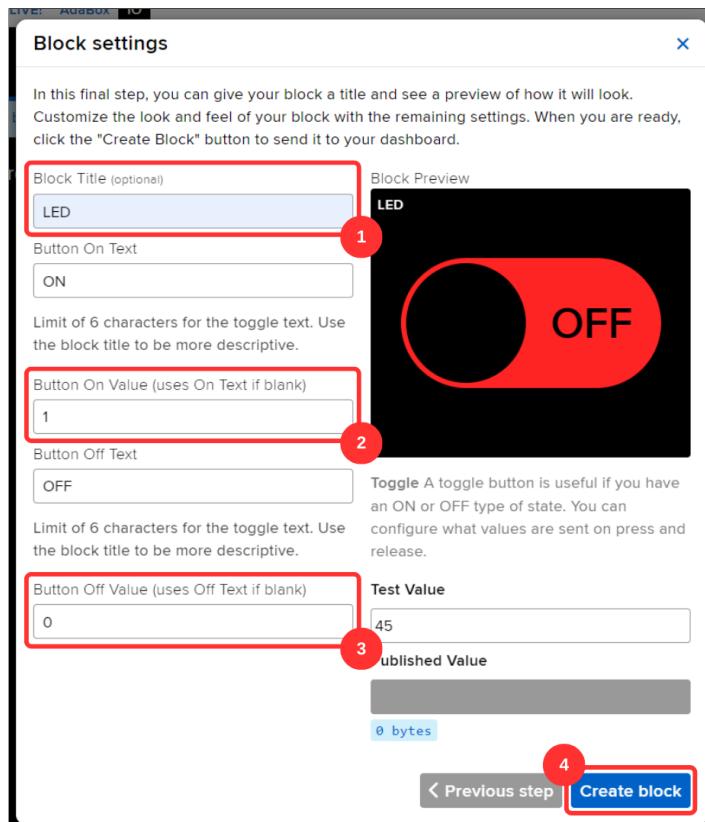
Hình 8.20: Hộp thoại chọn block

Sau khi chọn được block, ta sẽ liên kết block này với feed LED mà ta đã tạo lúc trước.



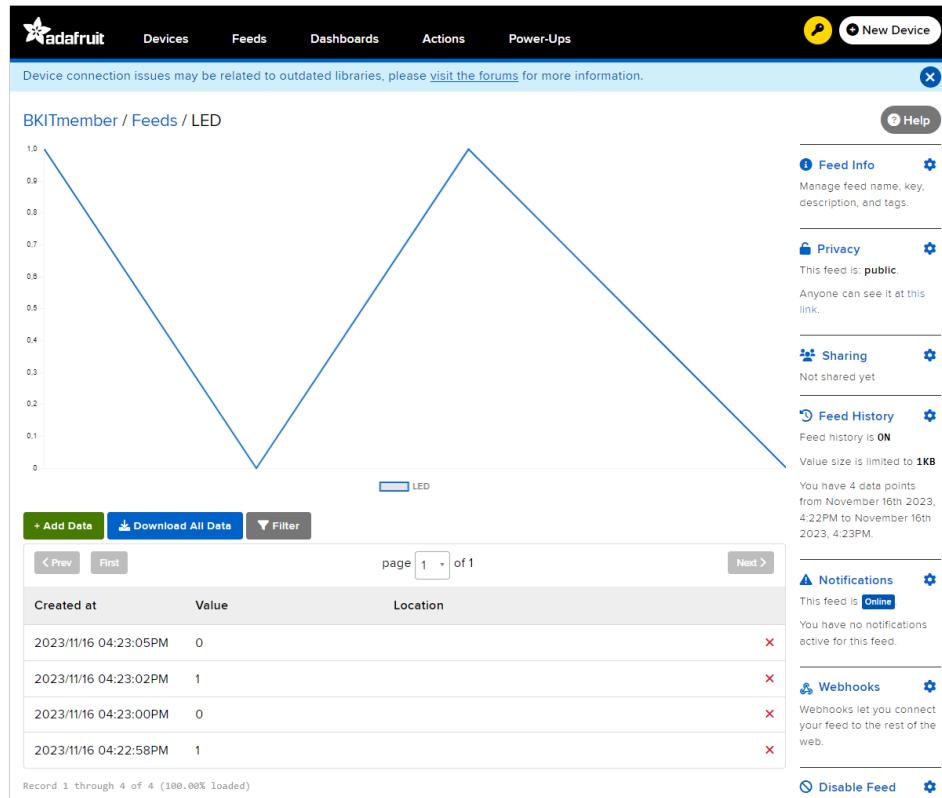
Hình 8.21: Kết nối block với feed

Định nghĩa các giá trị của toggle button, trong trường hợp này thì ta đang định nghĩa đèn bật thì dữ liệu gửi về là '1', đèn tắt thì dữ liệu gửi về là '0'. Sau đó chọn **Create block** để tạo block như hình 8.22.



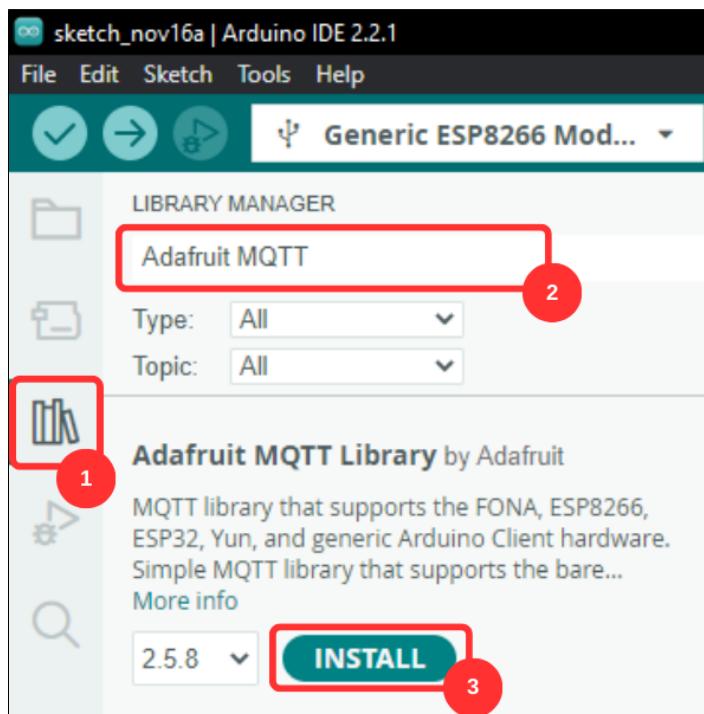
Hình 8.22: Định nghĩa các giá trị của block

Sau khi tạo xong dashboard, ta có thể chạy thử kết nối của dashboard với feed, biểu đồ giá trị trả về theo thời gian sẽ được thể hiện tại feed LED như hình 8.23.



Hình 8.23: Kiểm tra kết nối

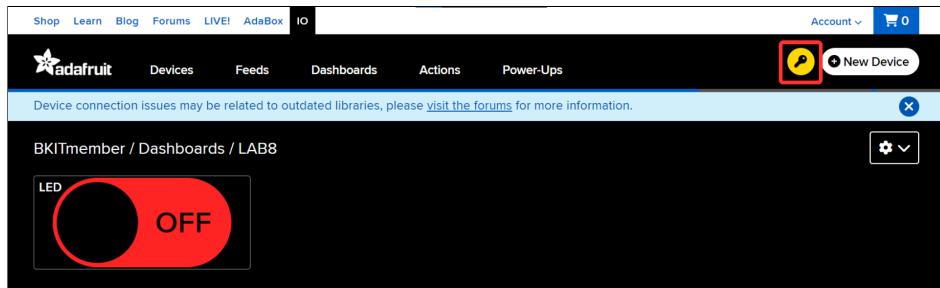
Để kết nối với server Adafruit, ta cần thêm thư viện Adafruit MQTT như sau:



Hình 8.24: Thêm thư viện trên Arduino

Sau đây là đoạn chương trình kết nối và đẩy dữ liệu lên Adafruit.

- Dòng 10 → 21 sẽ khai báo các tham số để kết nối với Adafruit thông qua MQTT. Để kết nối với server Adafruit ta cần lấy các thông tin **Username** và **Active Key** của tài khoản theo như hình 8.25 và 8.26.



Hình 8.25: Lấy key của Adafruit

YOUR ADAFRUIT IO KEY

Your Adafruit IO Key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your Adafruit IO Key can view all of your data, create new feeds for your account, and manipulate your active feeds.

If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.

Username: BKITmember

Active Key: aio_WqPO35igKT0CDtaUAr647rXFJxFs

REGENERATE KEY

[Hide Code Samples](#)

Arduino

```
#define IO_USERNAME "BKITmember"
#define IO_KEY "aio_WqPO35igKT0CDtaUAr647rXFJxFs"
```

Linux Shell

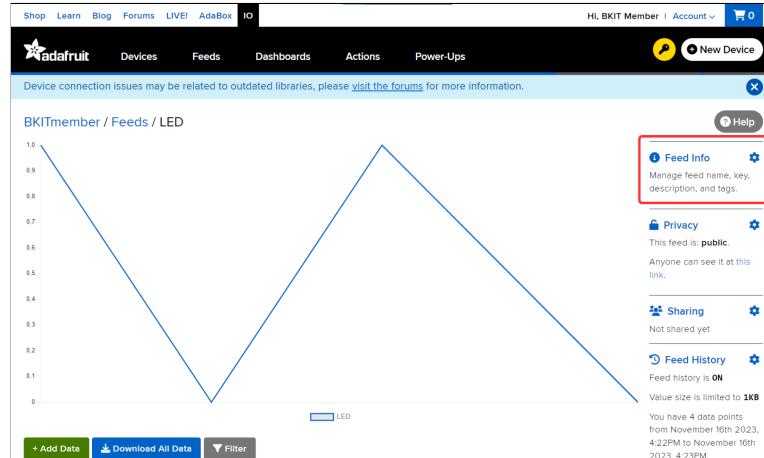
```
export IO_USERNAME="BKITmember"
export IO_KEY="aio_WqPO35igKT0CDtaUAr647rXFJxFs"
```

Scripting

```
ADAFRUIT_IO_USERNAME = "BKITmember"
ADAFRUIT_IO_KEY = "aio_WqPO35igKT0CDtaUAr647rXFJxFs"
```

Hình 8.26: Lấy key của Adafruit

- Dòng 23 sẽ khai báo biến nhằm đẩy dữ liệu lên một feed của Adafruit.



Hình 8.27: Lấy key của Feed

The screenshot shows a modal dialog titled "Create a new Feed". It has the following fields:

- Name**: LED
- Key**: led
- Current Endpoints**:
 - Web: <https://io.adafruit.com/BKITmember/feeds/led>
 - API: <https://io.adafruit.com/api/v2/BKITmember/feeds/led>
 - MQTT: <BKITmember/feeds/led> (highlighted with a red box)
- Description**: (empty)
- Show detailed feed JSON record**: (link)
- Buttons**: Cancel, Create

Hình 8.28: Lấy key của Feed

- Dòng 45 → 47 sẽ thiết lập kết nối với server Adafruit thông qua MQTT.
- Dòng 57 → 62 sẽ thêm đoạn chương trình giúp nhận dữ liệu thu thập từ STM32 và gửi lên server.
 - Nếu dữ liệu nhận được là 'a' thì ESP sẽ gửi giá trị 0 lên server nhằm báo hiệu đèn đã được tắt.
 - Nếu dữ liệu nhận được là 'A' thì ESP sẽ gửi giá trị 1 lên server nhằm báo hiệu đèn đã được bật.

```

1 #include <ESP8266WiFi.h>
2 #include "Adafruit_MQTT.h"
3 #include "Adafruit_MQTT_Client.h"
4
5 //Wifi name
6 #define WLAN_SSID          "Your_Wifi_Name"
7 //Wifi password
8 #define WLAN_PASS          "Your_Wifi_Password"
9
10 //setup Adafruit
11 #define AIO_SERVER          "io.adafruit.com"
12 #define AIO_SERVERPORT      1883
13 //fill your username
14 #define AIO_USERNAME        "Your_Adafruit_Name"
15 //fill your key
16 #define AIO_KEY              "Your_Adafruit_Password"
17
18 //setup MQTT
19 WiFiClient client;
20 Adafruit_MQTT_Client mqtt(&client, AIO_SERVER,
   AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
21
22 //set publish
23 Adafruit_MQTT_Publish light_pub = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/led");
24
25 int led_counter = 0;
26 int led_status = HIGH;
27
28 void setup() {
29   // put your setup code here, to run once:

```

```

30
31 //set pin 2,5 as OUTPUT
32 pinMode(2, OUTPUT);
33 pinMode(5, OUTPUT);
34 //set busy pin HIGH
35 digitalWrite(5, HIGH);

36
37 Serial.begin(115200);

38
39 //connect Wifi
40 WiFi.begin(WLAN_SSID, WLAN_PASS);
41 while (WiFi.status() != WL_CONNECTED) {
42     delay(500);
43 }

44
45 while (mqtt.connect() != 0) {
46     delay(500);
47 }

48
49 //finish setup, set busy pin LOW
50 digitalWrite(5, LOW);

51
52 }

53
54 void loop() {
55     // put your main code here, to run repeatedly:
56
57     if(Serial.available()){
58         int msg = Serial.read();
59         if(msg == 'o') Serial.print('0');
60         else if(msg == 'a') light_pub.publish(0);
61         else if(msg == 'A') light_pub.publish(1);
62     }
63
64     led_counter++;
65     if(led_counter == 100){
66         // every 1s
67         led_counter = 0;
68         //toggle LED

```

```

69     if(led_status == HIGH) led_status = LOW;
70     else led_status = HIGH;
71
72     digitalWrite(2, led_status);
73 }
74 delay(10);
75 }
```

Program 8.5: Gửi dữ liệu lên Adafruit

Sau khi gửi dữ liệu lên Adafruit thành công, ta cần hiện thực giao tiếp theo chiều ngược lại để ta có thể điều khiển đèn từ dashboard trên server. Khi ta thay đổi công tắc trên Adafruit dashboard, dữ liệu sẽ được gửi từ server xuống ESP, sau đó tùy vào dữ liệu nhận được mà ESP sẽ gửi dữ liệu đến STM32 để điều khiển đèn. Đoạn chương trình sau mô tả quá trình ESP nhận dữ liệu từ server. Trong đó:

- Dòng 26 được thêm vào để khởi tạo một biến giúp nhận dữ liệu từ một feed trên Adafruit.
- Dòng 31 → 34 là hàm sẽ được gọi mỗi khi nhận được dữ liệu điều khiển đèn từ Adafruit. Tùy theo dữ liệu nhận được mà ta sẽ gửi 'a' đến STM32 để tắt đèn, hoặc gửi 'A' để mở đèn.
- Dòng 53 → 54 nhằm mục đích đăng ký nhận tín hiệu mỗi khi có dữ liệu được gửi lên feed của Adafruit.
- Dòng 70 được thêm vào để liên tục cập nhật dữ liệu đã đăng ký từ server.

```

1 #include <ESP8266WiFi.h>
2 #include "Adafruit_MQTT.h"
3 #include "Adafruit_MQTT_Client.h"
4
5 //Wifi name
6 #define WLAN_SSID      "Your_Wifi_Name"
7 //Wifi password
8 #define WLAN_PASS      "Your_Wifi_Password"
9
10 //setup Adafruit
11 #define AIO_SERVER      "io.adafruit.com"
12 #define AIO_SERVERPORT  1883
13 //fill your username
14 #define AIO_USERNAME    "Your_Adafruit_Name"
```

```

15 //fill your key
16 #define AIO_KEY           "Your_Adafruit_Password"
17
18 //setup MQTT
19 WiFiClient client;
20 Adafruit_MQTT_Client mqtt(&client, AIO_SERVER,
   AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
21
22 //setup publish
23 Adafruit_MQTT_Publish light_pub = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/led");
24
25 //setup subscribe
26 Adafruit_MQTT_Subscribe light_sub = Adafruit_MQTT_Subscribe
   (&mqtt, AIO_USERNAME "/feeds/led", MQTT_QOS_1);
27
28 int led_counter = 0;
29 int led_status = HIGH;
30
31 void lightcallback(char* value, uint16_t len){
32   if(value[0] == '0') Serial.print('a');
33   if(value[0] == '1') Serial.print('A');
34 }
35
36 void setup() {
37   // put your setup code here, to run once:
38   //set pin 2,5 as OUTPUT
39   pinMode(2, OUTPUT);
40   pinMode(5, OUTPUT);
41   //set busy pin HIGH
42   digitalWrite(5, HIGH);
43
44   Serial.begin(115200);
45
46   //connect Wifi
47   WiFi.begin(WLAN_SSID, WLAN_PASS);
48   while (WiFi.status() != WL_CONNECTED) {
49     delay(500);
50   }

```

```

51
52 //subscribe light feed
53 light_sub.setCallback(lightcallback);
54 mqtt.subscribe(&light_sub);

55
56 //connect MQTT
57 while (mqtt.connect() != 0) {
58     mqtt.disconnect();
59     delay(500);
60 }

61
62 //finish setup, set busy pin LOW
63 digitalWrite(5, LOW);
64 }

65
66 void loop() {
67     // put your main code here, to run repeatedly:
68
69     //receive packet
70     mqtt.processPackets(1);

71
72     //read serial
73     if(Serial.available()){
74         int msg = Serial.read();
75         if(msg == 'o') Serial.print('0');
76         else if(msg == 'a') light_pub.publish(0);
77         else if(msg == 'A') light_pub.publish(1);
78     }

79
80     led_counter++;
81     if(led_counter == 100){
82         // every 1s
83         led_counter = 0;
84         //toggle LED
85         if(led_status == HIGH) led_status = LOW;
86         else led_status = HIGH;

87
88         digitalWrite(2, led_status);
89     }

```

```
90     delay(10);  
91 }
```

Program 8.6: Nhận dữ liệu từ Adafruit

5.3 Hướng dẫn thư viện STM

Vì STM32 và ESP giao tiếp với nhau thông qua UART nên các hàm liên quan đến ESP sẽ được thực thi bổ sung tại thư viện **uart.h**.

void uart_init_esp()

- **Mô tả:** Khởi tạo kết nối với ESP.
- **Tham số:** Không.
- **Giá trị trả về:** Không.

void uart_EspSendBytes(uint8_t* bytes, uint16_t size)

- **Mô tả:** Gửi các byte kí tự đến ESP.
- **Tham số:**
 - **bytes:** chuỗi dữ liệu.
 - **size:** độ dài chuỗi dữ liệu tính theo byte.
- **Giá trị trả về:** Không.

uint8_t uart_EspCheck()

- **Mô tả:** Kiểm tra kết nối đến ESP.
- **Tham số:** Không.
- **Giá trị trả về:**
 - **0:** Kết nối thất bại.
 - **1:** Kết nối thành công.

Để thuận tiện cho việc quản lý source code, ta sẽ thêm 2 file **light_control.h** và **light_control.c** để quản lý việc điều khiển đèn. Trong phần hướng dẫn này đèn tại **Output Y0** và nút nhấn "**0**" sẽ được dùng để mô phỏng.

```
1 #ifndef INC_LIGHT_CONTROL_H_
2 #define INC_LIGHT_CONTROL_H_
3
4 #include "gpio.h"
5 #include "uart.h"
6 #include "button.h"
7 #include "lcd.h"
8
9 extern uint8_t light_status;
10
11 void lightProcess();
12
13 void test_Esp();
```

Program 8.7: light_control.h

```
1 #include "light_control.h"
2
3 uint8_t light_status = 0;
4
5 void lightProcess(){
6     if(button_count[13] == 1){
7         light_status = 1 - light_status;
8         if(light_status == 1){
9             uart_EspSendBytes("A", 1);
10        } else {
11            uart_EspSendBytes("a", 1);
12        }
13    }
14    if(light_status == 1){
15        HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port, OUTPUT_Y0_Pin,
16        1);
17    } else {
18        HAL_GPIO_WritePin(OUTPUT_Y0_GPIO_Port, OUTPUT_Y0_Pin,
19        0);
20    }
21}
22
23 void test_Esp(){
24     if(uart_EspCheck() == 0) uart_EspSendBytes("o", 1);
```

```

23     else lcd_ShowStr(10, 50, "ESP Connect", GREEN, BLACK, 24,
24         0);
}

```

Program 8.8: light_control.c

Trong đó, hàm **lightProcess** sẽ điều khiển đèn dựa trên biến trạng thái và thay đổi biến trạng thái, gửi dữ liệu đến ESP khi có tín hiệu nút nhấn. Hàm **test_Esp** sẽ gửi kí tự 'o' đến ESP để kiểm tra kết nối và in lên màn hình thông báo nếu trạng thái kết nối là thành công. Ngoài ra, để xử lý dữ liệu nhận được từ UART, ta phải thêm các dòng lệnh vào hàm interrupt ở file **uart.c**

```

1 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
2     if(huart->Instance == USART1){
3         //rs232 isr
4         //...
5     }
6
7     if(huart->Instance == USART2){
8         //esp_isr
9         if(receive_buffer2 == '0') check_esp = 1;
10        else if(receive_buffer2 == 'a') light_status = 0;
11        else if(receive_buffer2 == 'A') light_status = 1;
12
13        //enable receive
14        HAL_UART_Receive_IT(&huart2, &receive_buffer2, 1);
15    }
16}

```

Program 8.9: Hàm intterupt Uart

Hàm main sẽ gọi các hàm điều khiển đèn được thực thi ở trên.

```

1 //...
2 /* USER CODE BEGIN Includes */
3 #include "software_timer.h"
4 #include "led_7seg.h"
5 #include "button.h"
6 #include "lcd.h"
7 #include "picture.h"
8 #include "ds3231.h"
9 #include "sensor.h"
10 #include "buzzer.h"

```

```

11 #include "touch.h"
12 #include "uart.h"
13 #include "light_control.h"
14 /* USER CODE END Includes */
15
16 // ...
17
18 /* USER CODE BEGIN PFP */
19 void system_init();
20 void test_LedDebug();
21 /* USER CODE END PFP */
22
23 int main(void)
24 {
25     // ...
26
27     /* USER CODE BEGIN 2 */
28     system_init();
29     /* USER CODE END 2 */
30
31     /* Infinite loop */
32     /* USER CODE BEGIN WHILE */
33     lcd_Clear(BLACK);
34     while (1)
35     {
36         // 50ms task
37         if(flag_timer2 == 1){
38             flag_timer2 = 0;
39             button_Scan();
40             test_Esp();
41             lightProcess();
42             test_LedDebug();
43         }
44
45         /* USER CODE END WHILE */
46
47         /* USER CODE BEGIN 3 */
48     }
49     /* USER CODE END 3 */

```

```

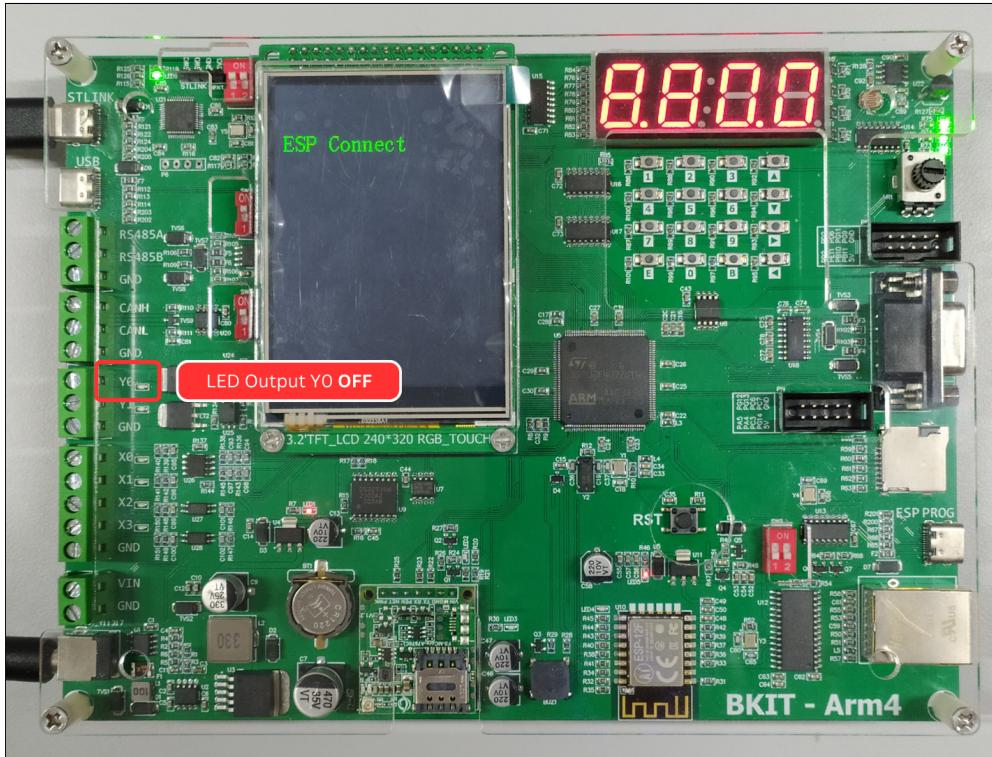
50 }
51
52 // ...
53
54 /* USER CODE BEGIN 4 */
55 void system_init(){
56     timer_init();
57     button_init();
58     lcd_init();
59     uart_init_esp();
60     setTimer2(50);
61 }
62
63 uint8_t count_led_debug = 0;
64
65 void test_LedDebug(){
66     count_led_debug = (count_led_debug + 1)%20;
67     if(count_led_debug == 0){
68         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin);
69     }
70 }
71
72 // ...

```

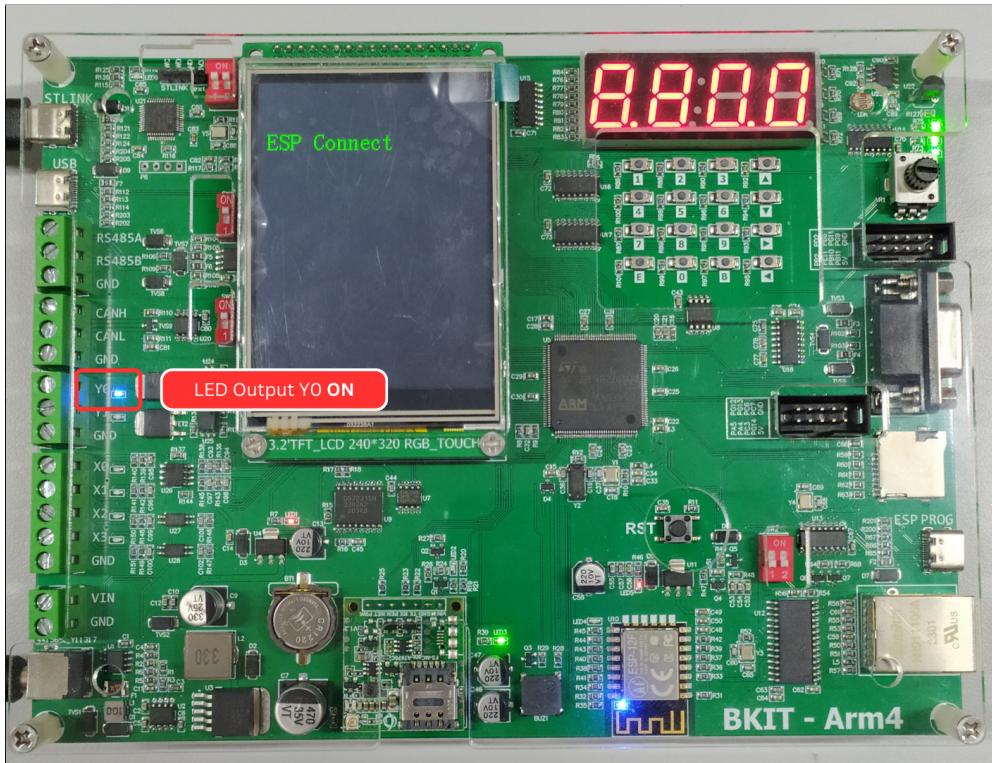
Program 8.10: File main.c

Khi hoàn thành chương trình trên, nếu kết nối ESP thành công, màn hình sẽ hiển thị chuỗi "ESP Connect". Tín hiệu Output Y0 sẽ có thể được điều khiển bởi nút nhấn "0" trên Kit thí nghiệm và bởi công tắc trên Adafruit dashboard. Trạng thái của Output Y0 sẽ được đồng bộ giữa Kit thí nghiệm và server.

Hình ảnh kết quả trên Kit thí nghiệm:



Hình 8.29: Kết quả hiện thực trên Kit thí nghiệm (1)



Hình 8.30: Kết quả hiện thực trên Kit thí nghiệm (2)

6 Bài tập và báo cáo

Thu thập dữ liệu từ cảm biến nhiệt độ trên Kit thí nghiệm và gửi lên server Adafruit mỗi 30s. Tạo dashboard trên Adafruit giúp hiển thị dữ liệu nhiệt độ theo dạng biểu đồ (Gợi ý: truyền dữ liệu nhiệt độ giữa STM và ESP dưới dạng "!TEMP:<Nhiệt độ>#").