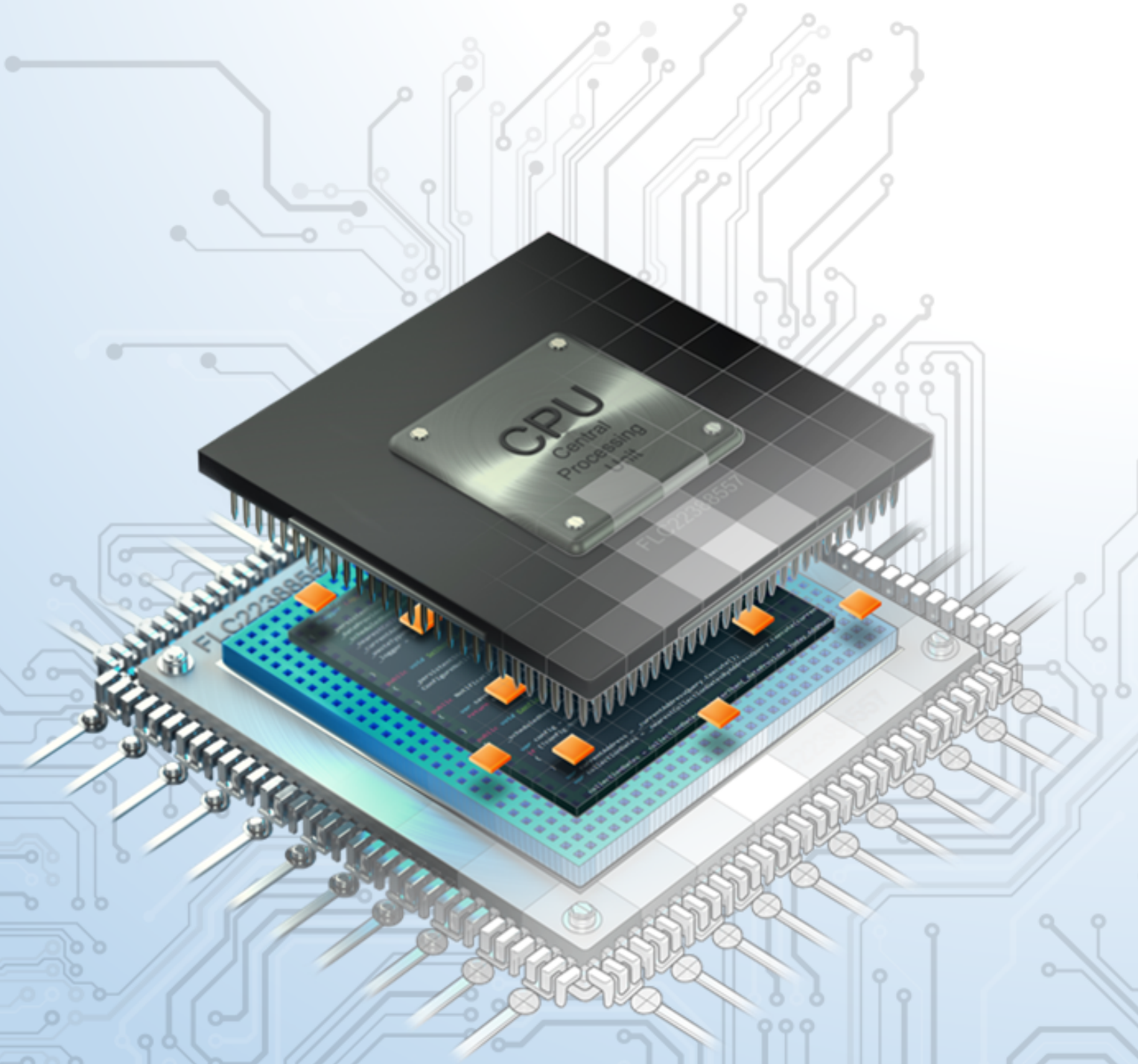
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

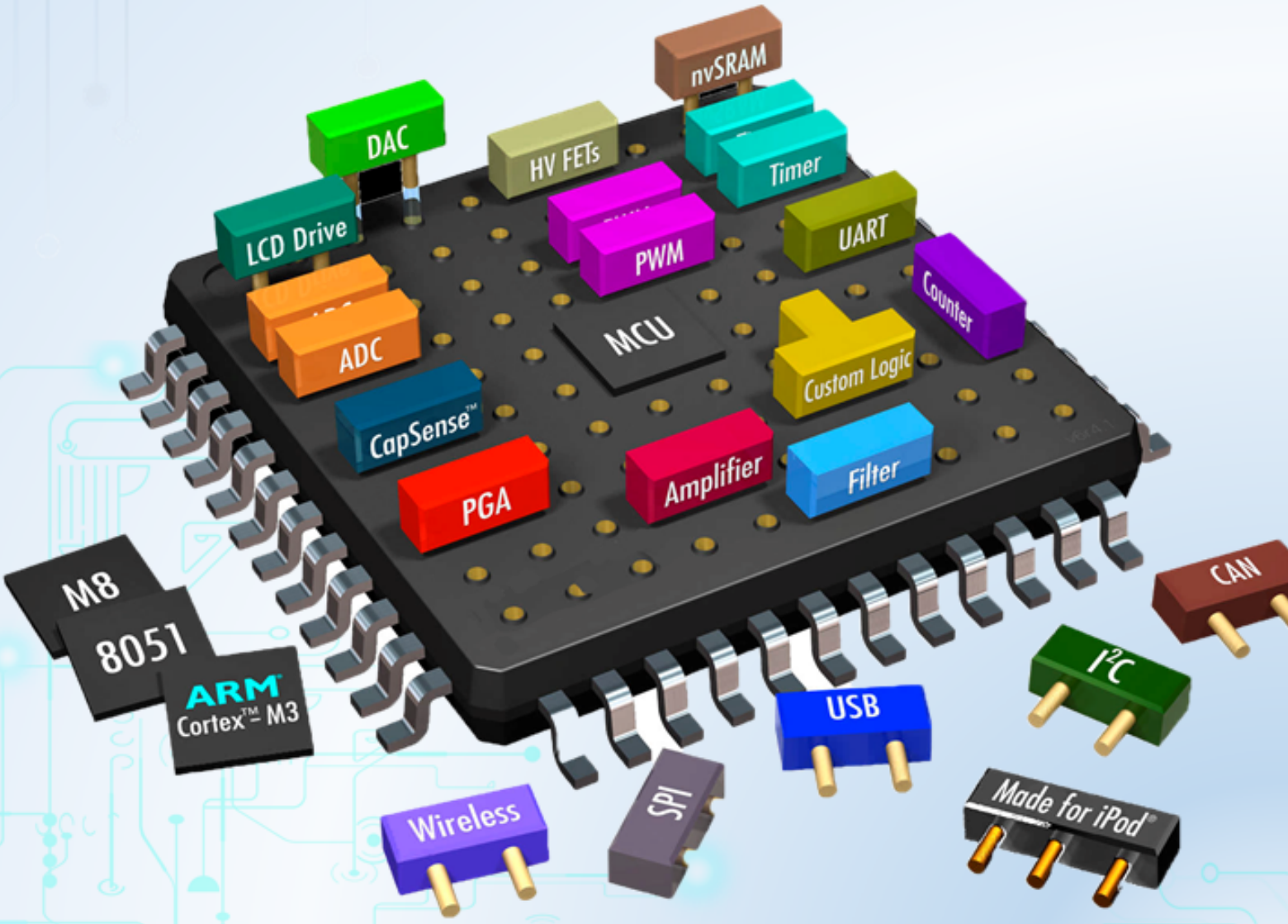


Dr. Le Trong Nhan

Mục lục

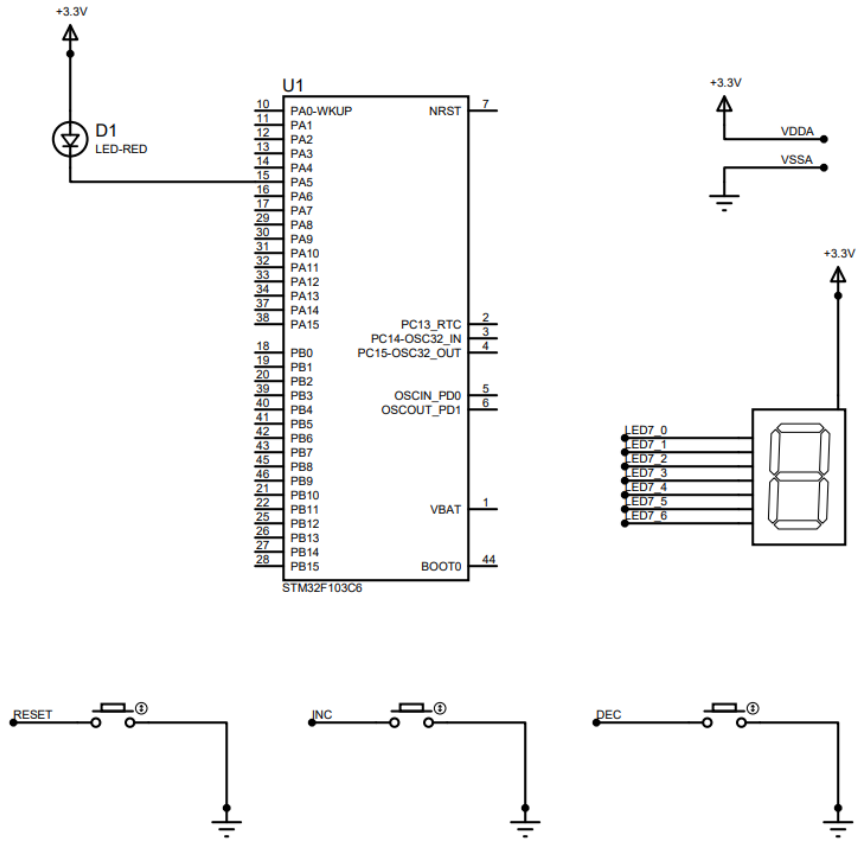
CHƯƠNG 1

MIDTERM 2022



1 Introduction

In this midterm project, a count-down system is designed and implemented in Proteus simulation. As it can be seen from Fig. 2.1, main components used in this project are the STM32F103C6, one LED, one LED7 segment and 3 different buttons.



Hình 1.1: *Proteus schematic for count-down system*

The main functions of the system are listed bellow:

- LED7 segment is used to display a counter ranging from 0 to 9.
- The **RESET** button is used to reset the counter value to 0. Meanwhile, the **INC** and **DEC** buttons are used to increase and decrease the counter value, respectively. There are two events need to handle for these buttons, including the normal-press and long-press.
- The D1 LED is blinking every second, which is normally used to monitor the execution of the system.

Students are supposed to following the section bellow, to finalize the project and fill in reports for their implementations. Some important notes for your midterm are listed bellow:

- The timer interrupt is 10ms. The value for counter is 9 (10 is also acceptable) when the pre-scaller is 7999.

- All the buttons must be DEBOUNCING by using a timer interrupt service routing. A timeout for long press event is 3 seconds.
- There is no HAL_Delay() function in your source code. All the delay behavior must be based on a software timer.
- This report must be submitted with your answer.
- GitHub link for the source code and demo video link must be public access.

2 Implement and Report

2.1 Proteus schematic - 1 point

In this part, students propose the connection of the LED7 segment and 3 buttons to the STM32F103C6.

Your report: The schematic of your system is presented here. The screen can be captured and present in this part.

2.2 State machine Step 1 - 2 points

A state machine is required in this step to perform just only the normal-press (or a button push) behavior of three buttons:

- Whenever the RESET is pressed, the counter value is 0.
- When INC is pressed, the counter is increased by 1. When counter is 9, it comes back to 0.
- When DEC is pressed, the counter is decreased by 1. When counter is 0, it rolls back to 9.

The value of the counter is displayed on the LED7 Segment.

Your report: Present your state machine in this part.

Your report: Present a main function, which is used to implement the state machine. This function should be invoked in main().

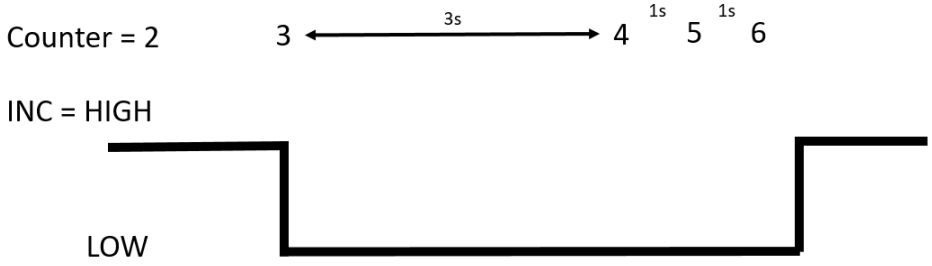
```
1 void fsm_simple_buttons_run(){
2     //TODO
3 }
```

Program 1.1: Implementation of the state machine

2.3 State machine Step 2 - 2 points

In this part, long-press events for INC and DEC buttons are added to the project. For a button, this event is raised after 3 seconds keep pressing the button.

When a long-press event is detected, the value of counter keeps changing every 1 second until the button is released. For example, the current value of counter is 2 and the INC button is pressed. The value of counter immediately increased by 1, or counter = 3. The INC button keeps pressing for 3 seconds, then the value of counter is 4. As long as the INC button is pressed, the value continues increasing **every 1 second**. This behavior is illustrated in the Figure bellow:



Hình 1.2: *Long press behavior for INC button*

The behaviors of the DEC button are reversed to the INC button. The value of counter is also roll back if it reaches 0 or 9.

Your report: Present your whole state machine when the long press events are added.

Your report: Present a main function, which is used to implement additional states. Minor changes in the previous source code are note required to present here.

2.4 State machine Step 3 - 2 points

Finally, where there is no button event after 10 seconds, the value of of counter is counted down and stopped at 0. If the INC or DEC are pressed again, the status of the system comes back to previous state, which is designed in Subsection 2 or 3.

Your report: Present your whole state machine for the 10s time-out event.

Your report: Present a main function, which is used to implement additional states. Minor changes in the previous source code are note required to present here.

2.5 Led Blinky for Debugging - 1 point

Finally, for many projects based on microcontroller, there is an LED keeps blinking every second. In this project, the LED connected to PA5 is used to perform this feature.

Your report: Present your solution and the source code for this feature. It can be very simple source code or a new state machine for this LED. If a state machine is used, please present it in the report.

2.6 Github and Demo

A link to your github presented the last commit of your project is provided in this section. This link contains all files in your STMCube project (configurations, header and source files)

https://github.com/manhcuong1502/Midterm_MC-MP_1852283

And a link for just one demo video is also needed to present here.

https://drive.google.com/file/d/1ROER_YIfvO7FklR9RGgFk-IIfAriVlvP/view?usp=sharing

3 Extra exercise - Engineer mindset -1 point

In this course, we encourage you to obtain an innovative mindset to solve daily problem. In this question, we would expect you to write a C program to solve the following problem.

Suffix with Unit

EXample:

1 suffixWithUnit(123) => 123

2 suffixWithUnit(1234) => 1.234 Kilo

3 suffixWithUnit(12345) => 12.345 Kilo

4 suffixWithUnit(1234567) => 1.234567 Mega

5 suffixWithUnit(12345678) => 12.345678 Mega

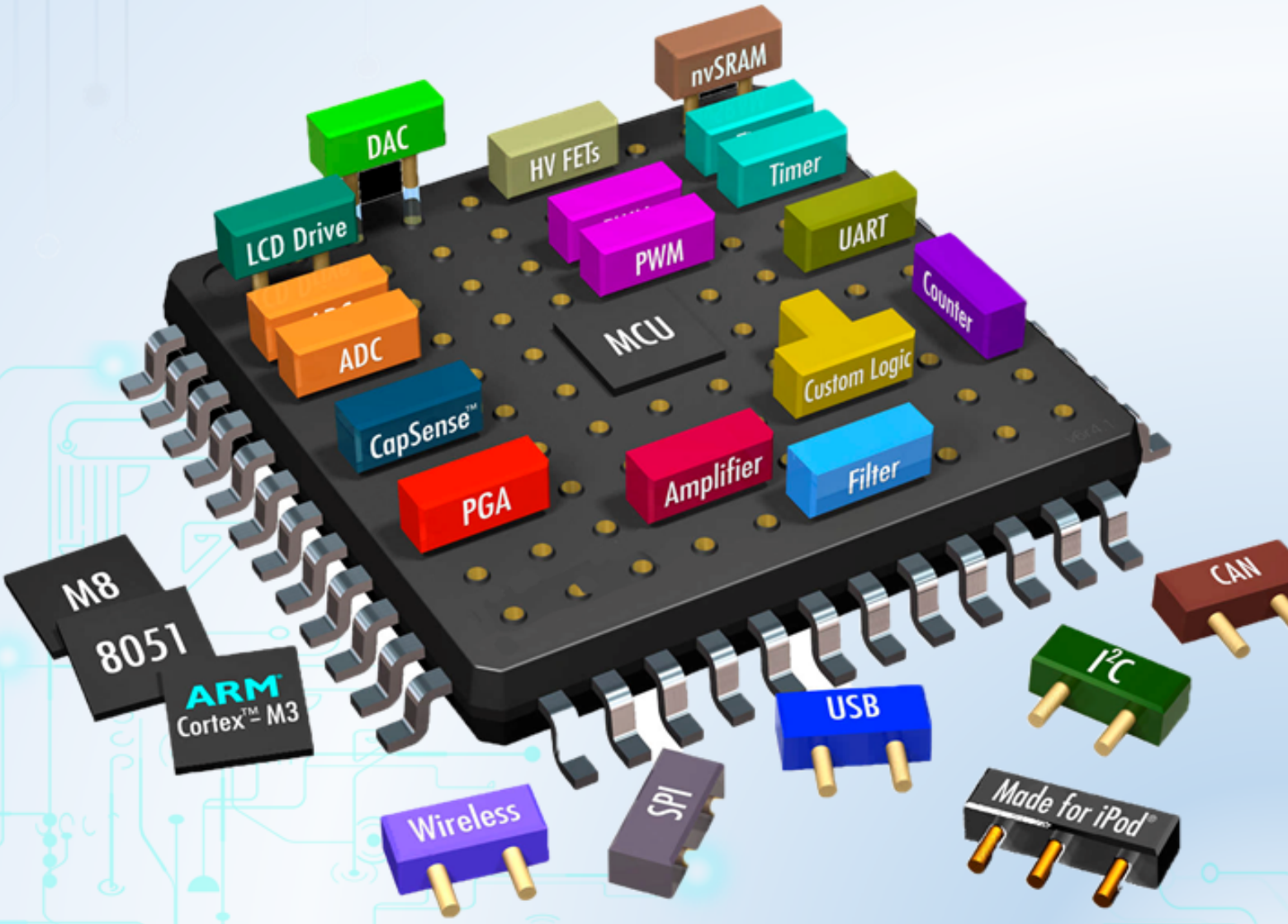
Prototype

```
string suffixWithUnit(double number) {
}
```

How would you solve them? Please share your thinking to solve this problem and provide your answer.

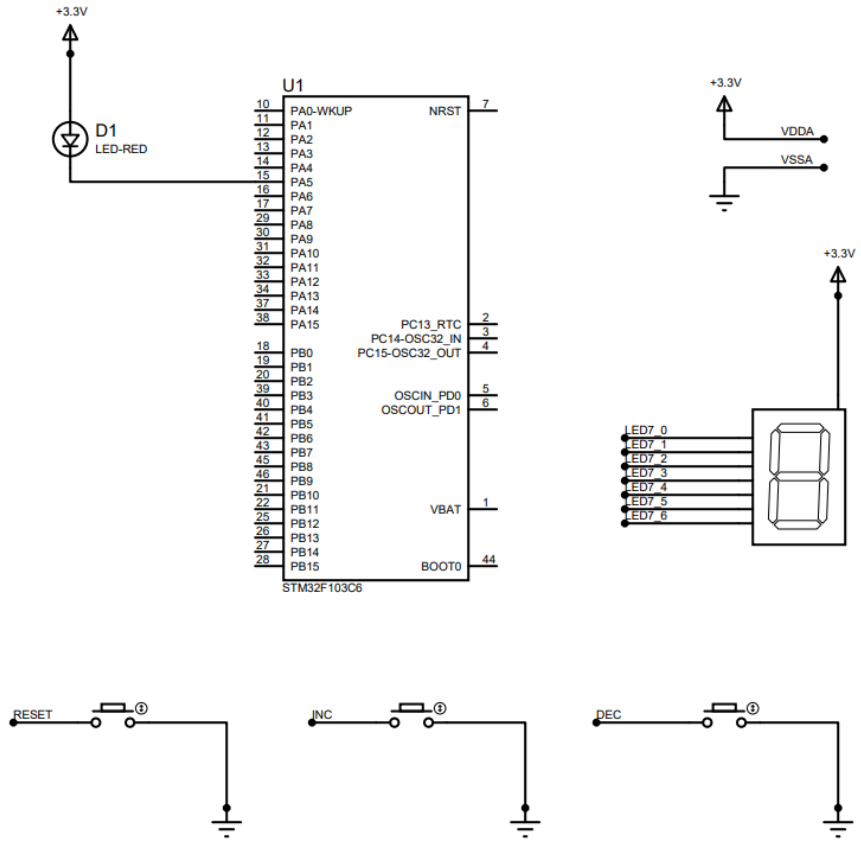
CHƯƠNG 2

GIỮA KÌ 2022



1 Giới thiệu

Trong project giữa kì này, một hệ thống đếm lùi sẽ được hiện thực trên phần mềm mô phỏng Proteus. Như được trình bày ở Hình 2.1, các thành phần chính của hệ thống bao gồm vi điều khiển STM32F103C6, một đèn LED, một LED 7 đoạn và 3 nút nhấn đơn.



Hình 2.1: *Proteus schematic for count-down system*

Một số tính năng chính của hệ thống được trình bày như sau:

- LED 7 đoạn dùng để hiển thị giá trị của counter, có giá trị từ 0 đến 9.
- Nút **RESET** được dùng để reset giá trị của counter về 0. Trong khi đó, nút nhấn **INC** và **DEC** được dùng để tăng hoặc giảm giá trị của counter. Có 2 sự kiện cần phải xử lý cho các nút nhấn, là nhấn thường và nhấn giữa. Trong dự án này, một nút nhấn được coi là nhấn giữ, nếu nó giữ nguyên trạng thái đó trong 3 giây liên tiếp.
- Đèn LED D1 được dùng để theo dõi hoạt động của hệ thống, nó sẽ luân phiên chớp tắt mỗi giây.

Sinh viên sẽ hiện thực dự án của mình theo từng bước yêu cầu ở phần bên dưới. Trong mỗi phần, sinh viên cần trình bày các yêu cầu về report. Một số lưu ý quan trọng cho phần hiện thực như sau:

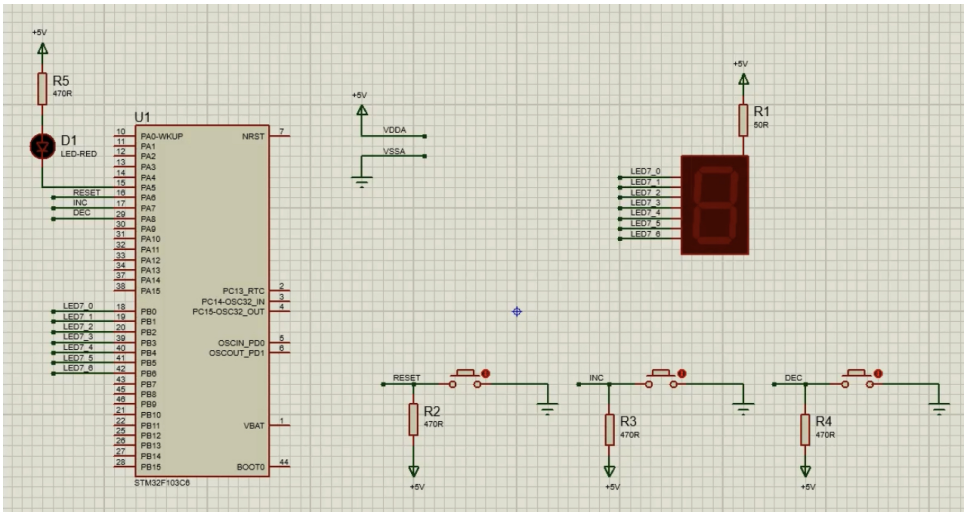
- Chu kì của ngắt timer là 10ms. Giá trị của counter khi cài đặt timer có thể là 9 hoặc 10 đều được chấp nhận (trong trường hợp này, prescaller là 7999).
- Tất cả các nút nhấn đều được xử lý chống rung bằng ngắt timer 10ms. Một nút nhấn sẽ được xem là nhấn đè nếu nó được giữ liên tục trong 3 giây.
- Không sử dụng HAL_Delay() trong việc hiện thực. Tất cả các hiệu ứng thời gian đều phải được hiện thực dựa trên software timer.
- Report này cần được nộp lại kèm theo các câu trả lời của sinh viên.
- Link github và video demo trong report này được chỉnh quyền truy cập public.

2 Hiện thực và Report

2.1 Sơ đồ nguyên lý trên Proteus - 1 điểm

Trong phần này, sinh viên đề xuất các kết nối của LED 7 đoạn và 3 nút nhấn với STM32F103C6.

Report: Trình bày sơ đồ nguyên lý tại đây. Sinh viên có thể chụp màn hình Proteus cho phần sơ đồ nguyên lý.



Hình 2.2: Sơ đồ nguyên lý

2.2 State machine Step 1 - 2 điểm

Một máy trạng thái sẽ được thiết kế để thực hiện các chức năng với trạng thái nhấn bình thường (normal-press) cho 3 nút nhấn, như sau:

- Khi nút RESET is được nhấn, giá trị counter sẽ là 0.
- Khi INC được nhấn, giá trị của counter tăng 1 đơn vị. Khi nó đến 9, giá trị tiếp theo là 0.
- Khi DC được nhấn, giá trị của counter giảm 1 đơn vị. Khi giá trị của nó là 0, giá trị tiếp theo là 9.

Giá trị của biến counter được hiển thị lên LED 7 đoạn.

Report: Trình bày máy trạng thái của hệ thống.
Ở bước này hệ thống có 2 trạng thái là **NORMAL** và **PRESSED**. Ở trạng thái **NORMAL**, hệ thống sẽ kiểm tra có nút nhấn nào được nhấn hay không. Nếu có, hệ thống sẽ chuyển sang trạng thái **PRESSED**, ngược lại hệ thống sẽ hoạt động bình thường (cách hoạt động được cập nhật ở step 3). Ở trạng thái **PRESSED**, hệ thống sẽ kiểm tra nút được nhấn và thực hiện chức năng tương ứng để cập nhật giá trị counter như mô tả trên.
Sau khi cập nhật trạng thái và biến counter, giá trị của counter sẽ được hiển thị ra LED 7 đoạn.

Your report: Trình bày hiện thực của hàm dùng để hiển thực máy trạng thái ở trên. Thông thường, hàm này sẽ được gọi trong main().

```
void fsm_simple_buttons_run() {
  switch(state){
  case NORMAL:
  {
    for (uint8_t i = 0; i < N_BUTTONS; i++){
      if (is_pressed(i)){
        button = i;
        break;
      }
    }
    if (button != -1){
      state = PRESSED;
    }
    // update for normal state at step 3 later
    break;
  }
  case PRESSED:
  {
    switch(button){
    case RST:
    {
      counter = 0;
      break;
    }
    case INC:
    {
      counter++;
```

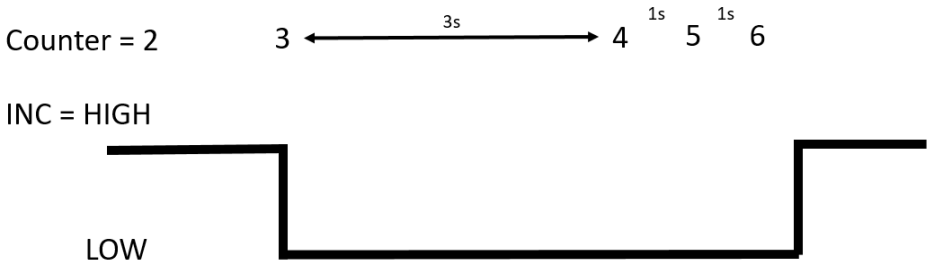
```
28        if (counter > 9) counter = 0;
29        break;
30      }
31      case DEC:
32      {
33        counter--;
34        if (counter < 0) counter = 9;
35        break;
36      }
37      }
38    }
39    }
40    // SHOW COUNTER VALUE INTO 7SEG_LED
41    led_7_seg_disp(counter);
42  }
```

Program 2.1: Hiện thực máy trạng thái

2.3 State machine Step 2 - 2 điểm

Trong phần này, việc nhấn giữ (long-press) cho nút INC và DEC được thêm vào trong dự án. Đối với nút nhấn, sự kiện long-press xảy ra khi nó được nhấn giữ liên tục sau 3 giây.

Khi một sự kiện nhấn giữ được phát hiện, giá trị của counter liên tục thay đổi mỗi 1 giây. Ví dụ, giá trị hiện tại của counter là 2. Khi nút INC được nhấn, ngay lập tức giá trị của nó sẽ tăng lên 3. Tuy nhiên, khi INC tiếp tục được giữ, 3 giây sau, sự kiện long-press xảy ra, giá trị của counter tăng lên 4. Từ lúc này, giá trị của counter sẽ tăng lên 1 mỗi giây, cho đến khi nút INC được thả ra. Giá trị của biến counter này được minh họa như hình bên dưới.



Hình 2.3: Long press behavior for INC button

Hành vi của nút DEC sẽ ngược lại với nút INC. Giá trị của biến counter sẽ xoay vòng khi nó đến 0 hoặc 9.

Report: Trình bày toàn bộ máy trạng thái cho đến bước này.
Ở bước này, hệ thống sẽ có thêm trạng thái **WAITING_3s** và **LONG_PRESSED**.
Trạng thái **WAITING_3s** xảy ra trong khoảng thời gian nút vừa được nhấn và giữ

cho đến khi đủ 3 giây (khoảng thời gian giữa giá trị 3 và 4 trong hình 2.3). Ở trạng thái này sẽ có 2 trường hợp, nút được nhấn giữ liên tục sau 3 giây và chuyển sang trạng thái **LONG_PRESSED** hoặc nút nhấn được buông ra và chuyển sang trạng thái **NORMAL**.
Ở trạng thái **LONG_PRESSED** giá trị của counter sẽ được thay đổi theo mô tả trên nếu nút nhấn được giữ liên tục sau 3 giây (nếu nút RST được nhấn giữ liên tục, counter sẽ luôn bằng 0). Việc đảm bảo giá trị counter thay đổi ở mỗi giây sẽ được hiện thực thông qua **software timer**.
Việc hiện thực trạng thái **NORMAL** của hệ thống sẽ được trình bày bổ sung ở step 3.
Report: Sinh viên trình bày một hàm chính dùng để hiện thực các trạng thái mới. Các thay đổi trên máy trạng thái cũ không cần phải trình bày ở đây.

```
void fsm_simple_buttons_run() {
  switch(state){
  case NORMAL:
  {
    for (uint8_t i = 0; i < N_BUTTONS; i++){
      if (is_pressed(i)){
        button = i;
        break;
      }
    }
    if (button != -1){
      state = PRESSED;
    }
    // update for normal state at step 3 later
    break;
  }
  case PRESSED:
  {
    switch(button){
    case RST:
    {
      counter = 0;
      break;
    }
    case INC:
    {
      counter++;
      if (counter > 9) counter = 0;
      break;
    }
    case DEC:
    {
      counter--;
      if (counter < 0) counter = 9;
      break;
    }
```

```
37      }
38
39      state = WAITING_3s;
40      break;
41    }
42    case WAITING_3s:
43    {
44      if (!is_pressed(button)){
45        state = NORMAL;
46      }
47      else if (is_pressed_3s(button)){
48        state = LONG_PRESSED;
49        timer_setting(COUNTER_CHANGE, 10);
50      }
51      break;
52    }
53    case LONG_PRESSED:
54    {
55      if (get_timer_flag(COUNTER_CHANGE)){
56        switch(button){
57        case RST:
58        {
59          counter = 0;
60          break;
61        }
62        case INC:
63        {
64          counter++;
65          if (counter > 9) counter = 0;
66          break;
67        }
68        case DEC:
69        {
70          counter--;
71          if (counter < 0) counter = 9;
72          break;
73        }
74        }
75        timer_setting(COUNTER_CHANGE, 1000);
76      }
77      else if (!is_pressed(button)){
78        state = NORMAL;
79        timer_setting(COUNTER_CHANGE, 0);
80      }
81      break;
82    }
83    }
84    // SHOW COUNTER VALUE INTO 7SEG_LED
85    led_7_seg_disp(counter);
```

```
}
```

2.4 State machine Step 3 - 2 points

Cuối cùng, khi không có nút nào được nhấn, hệ thống tự động đếm lùi biến counter và dừng lại khi nó bằng 0. Sau đó, nếu nút INC hoặc DEC được nhấn, trạng thái của hệ thống lại quay về một trong các trạng thái đã thiết kế trước đó.

Your report: Trình bày toàn bộ máy trạng thái, khi sự kiện time-out 10s được thêm vào.
Khi hệ thống chuyển từ trạng thái **WAITING_3s** hay **LONG_PRESSED** sang trạng thái **NORMAL**, **software timer** sẽ hoạt động để đảm bảo việc time-out 10s.
Report: Sinh viên trình bày một hàm chính dùng để hiện thực các trạng thái mới. Các thay đổi trên máy trạng thái cũ không cần phải trình bày ở đây.

```
void fsm_simple_buttons_run() {
  switch(state){
  case NORMAL:
  {
    for (uint8_t i = 0; i < N_BUTTONS; i++){
      if (is_pressed(i)){
        button = i;
        break;
      }
    }
    if (button != -1){
      state = PRESSED;
    }
    else if (get_timer_flag(SYSTEM_RUN)){
      if (counter > 0) counter--;
      timer_setting(SYSTEM_RUN, 1000);
    }
    // update for normal state at step 3 later
    break;
  }
  case PRESSED:
  {
    switch(button){
    case RST:
    {
      counter = 0;
      break;
    }
    case INC:
    {
      counter++;
      if (counter > 9) counter = 0;
      break;
```

```
    }
    case DEC:
    {
      counter--;
      if (counter < 0) counter = 9;
      break;
    }
    }

    state = WAITING_3s;
    break;
  }
  case WAITING_3s:
  {
    if (!is_pressed(button)){
      state = NORMAL;
      timer_setting(SYSTEM_RUN, 10000);
      button = -1;
    }
    else if (is_pressed_3s(button)){
      state = LONG_PRESSED;
      timer_setting(COUNTER_CHANGE, 10);
    }
    break;
  }
  case LONG_PRESSED:
  {
    if (get_timer_flag(COUNTER_CHANGE)){
      switch(button){
      case RST:
      {
        counter = 0;
        break;
      }
      case INC:
      {
        counter++;
        if (counter > 9) counter = 0;
        break;
      }
      case DEC:
      {
        counter--;
        if (counter < 0) counter = 9;
        break;
      }
      }
      timer_setting(COUNTER_CHANGE, 1000);
    }
```

```
    else if (!is_pressed(button)){
      state = NORMAL;
      timer_setting(COUNTER_CHANGE, 0);
      timer_setting(SYSTEM_RUN, 10000);
      button = -1;
    }
    break;
  }
  }
  // SHOW COUNTER VALUE INTO 7SEG_LED
  led_7_seg_disp(counter);
}
```

2.5 Led Blinky for Debugging - 1 điểm

Cuối cùng, trong nhiều dự án dựa trên vi điều khiển, có 1 LED luôn luôn chớp tắt mỗi giây, để giám sát hoạt động của hệ thống. Trong project này, LED nối với chân PA5 sẽ được dùng để hiện thực tính năng này.

Report: Sinh viên trình bày giải pháp của mình cho tính năng này. Giải pháp có thể rất đơn giản với vài dòng code hoặc thiết kế máy trạng thái và lập trình cho nó. Trong trường hợp thứ 2, sinh viên trình bày máy trạng thái trước khi trình bày phần hiện thực mã nguồn.
Ở phần này, hàm **led_blinking** được hiện thực cho việc chớp tắt LED. Thời gian giữa các lần chớp tắt được kiểm tra bằng **software timer**. Cứ sau mỗi 0.5 giây, flag của **software timer** phụ trách việc chớp tắt LED sẽ được bật. Khi flag này được bật, câu lệnh toggle sẽ được gọi để thay đổi tín hiệu đầu ra của chân PA5.

```
void led_blinking(){
  if (get_timer_flag(LED_BLINKING)){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    timer_setting(LED_BLINKING, 500);
  }
}
```

2.6 Github và Demo

Link github cho dự án của sinh viên được trình bày ở đây, bao gồm tất cả các files trong dự án (configurations, header and source files). Link này là lần commit sau cùng trong project giữa kì của sinh viên.

https://github.com/manhcuong1502/Midterm_MC-MP_1852283

And a link for just one demo video is also needed to present here.

https://drive.google.com/file/d/1ROER_YIfvO7FklR9RGgFk-IIfAriVlvP/view?usp=sharing

3 Bài tập thêm - Engineer mindset - 1 điểm

In this course, we encourage you to obtain an innovative mindset to solve daily problem. In this question, we would expect you to write a C program to solve the following problem.

Suffix with Unit

EXample:

1 suffixWithUnit(123) => 123

2 suffixWithUnit(1234) => 1.234 Kilo

3 suffixWithUnit(12345) => 12.345 Kilo

4 suffixWithUnit(1234567) => 1.234567 Mega

5 suffixWithUnit(12345678) => 12.345678 Mega

Prototype

```
string suffixWithUnit(double number) {
}
```

How would you solve them? Please share your thinking to solve this problem and provide your answer. **Report:**
Đây là bài toán chuyển đổi 1 số thực với độ chính xác kép được truyền vào hàm với đơn vị đo là **Byte** về đơn vị đo lớn nhất có thể và giá trị sau khi được chuyển đổi không nhỏ hơn 1.
Đầu tiên, trường hợp đặc biệt là giá trị truyền vào là 0 sẽ được kiểm tra và trả về kết quả là "0". Ngược lại, hàm sẽ tiếp tục thực hiện việc chuyển đổi như sau:
+ **Bước 1:** Vì giá trị truyền vào hàm là số thực, phần thập phân ban đầu sẽ được chuyển đổi về string trước và không bao gồm dấu chấm thập phân.
Ví dụ 1:

- 12.345, giá trị 345 sẽ được chuyển đổi về string.
- 12.76543, giá trị 76543 sẽ được chuyển đổi về string.
- 12, giá trị 000000 sẽ được chuyển đổi về string.

+ **Bước 2:** Sau đó tiến hành chuyển đổi về đơn vị đo lớn hơn cho đến khi không chuyển đổi được nữa, đồng thời đếm số lần chuyển đổi và nối phần thập phân vào chuỗi trước đó.
Ví dụ 2:

- 1000 có số lần chuyển đổi đơn vị đo là 1, chuỗi lưu trữ phần thập phân sau khi chuyển đổi là 000000000 và giá trị sau khi chuyển đổi đơn vị đo là 1.Trong đó, phần thập phân 000000 đã được chuyển đổi ở bước 1 và 000 được chuyển đổi ở bước 2.
- 1232345.67800 có số lần chuyển đổi đơn vị đo là 2, chuỗi lưu trữ phần thập phân sau khi chuyển đổi là 232345678000 và giá trị sau khi chuyển đổi đơn vị đo là 1. Trong đó phần thập phân 678000 đã được chuyển đổi ở bước 1 và 232232345 được chuyển đổi ở bước 2.

+ **Bước 3:** Loại bỏ các số 0 ở cuối của chuỗi lưu trữ phần thập phân. Nếu sau khi loại bỏ, độ dài chuỗi lưu trữ phần thập phân lớn hơn 0, ta thêm dấu "." ở đầu chuỗi, ngược lại chuỗi sẽ rỗng. Tiếp tục ví dụ 2, ta có kết quả như sau:

- 1000 sau khi chuyển đổi, chuỗi lưu trữ phần thập phân chỉ toàn là 0, nên toàn bộ chuỗi sẽ bị xóa.
- 12321232345.67800 sau khi chuyển đổi, chuỗi lưu trữ phần thập phân sẽ là ".232345678".

+ **Bước 4:** Thêm phần nguyên vào đầu chuỗi và thêm đơn vị đo vào cuối chuỗi tương ứng với số lần tăng đơn vị đo ở Bước 2.

Mã nguồn được hiện thực như sau:

```
string suffixWithUnit ( double number ) {
    if (number == 0.0)
        return "0";

    //STEP 1
    string res = to_string(number);
    res = res.substr(res.find('.') + 1);
    int n = 0;

    //STEP 2
    while ((long long)(number) / 1000 != 0){
        res = to_string((long long)number % 1000) + res;
        number = (long long)number / 1000;
        n++;
    }

    //STEP 3
    while (res.length() != 0 && res.back() == '0')
        res.pop_back();
    if (res.length() != 0){
        res = "." + res;
```

```
    }

    //STEP 4
    res = to_string(int(number)) + res;
    switch (n)
    {
    case 1:
        res += " Kilo";
        break;
    case 2:
        res += " Mega";
        break;
    case 3:
        res += " Giga";
        break;
    case 4:
        res += " Tera";
        break;
    case 5:
        res += " Peta";
        break;
    case 6:
        res += " Exa";
        break;
    case 7:
        res += " Zetta";
        break;
    default:
        break;
    }
    return res;
}
```