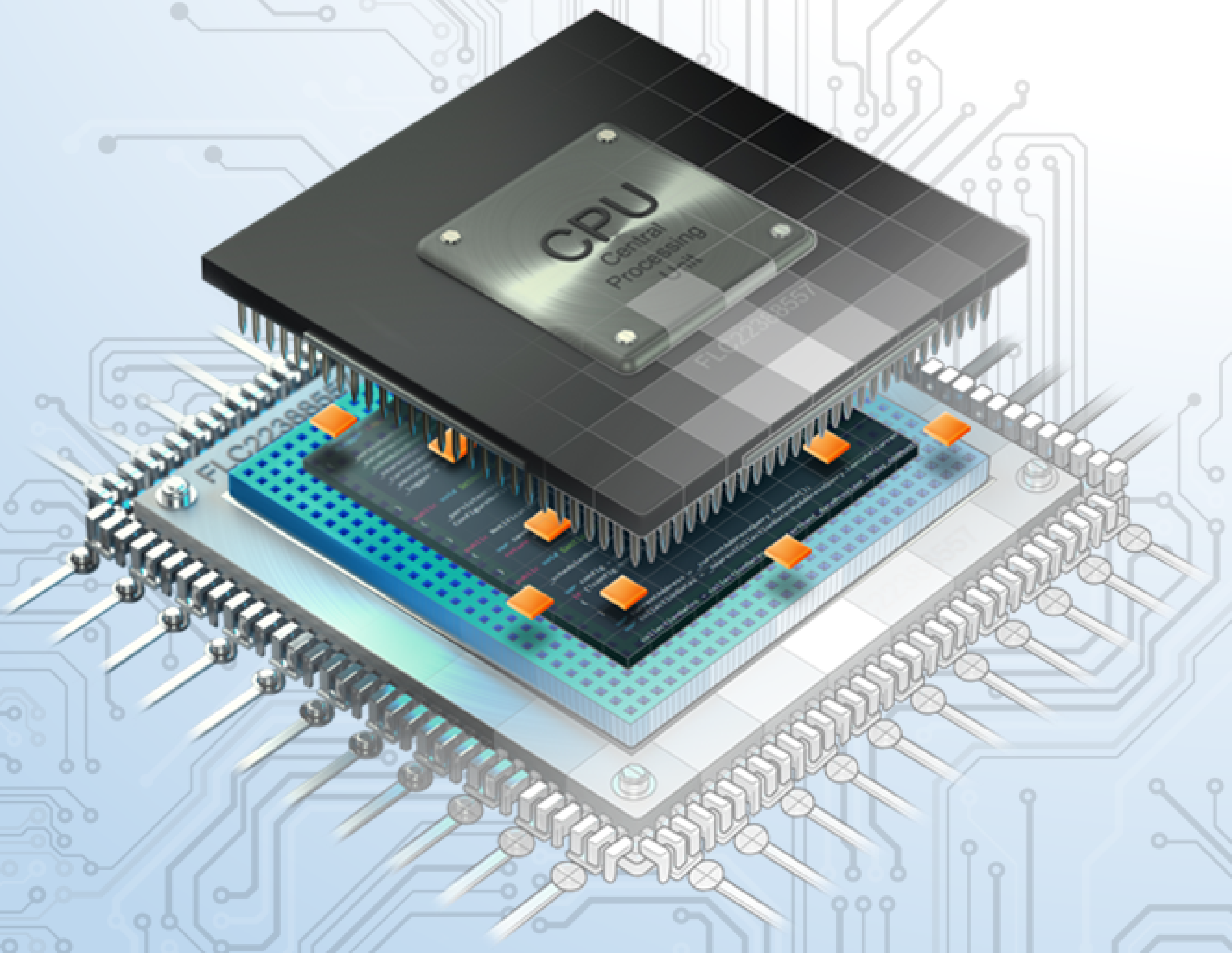
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

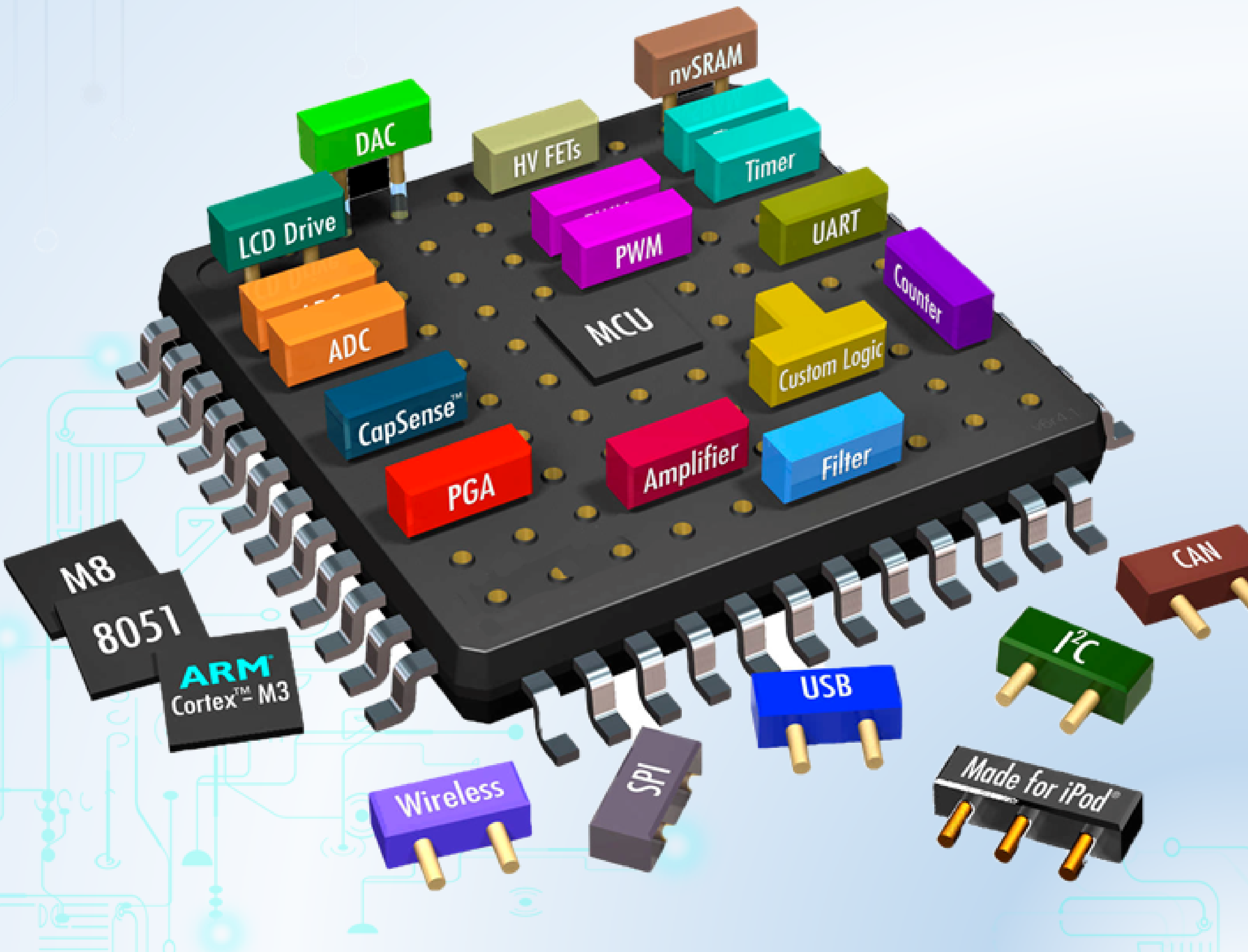


Dr. Le Trong Nhan

Mục lục

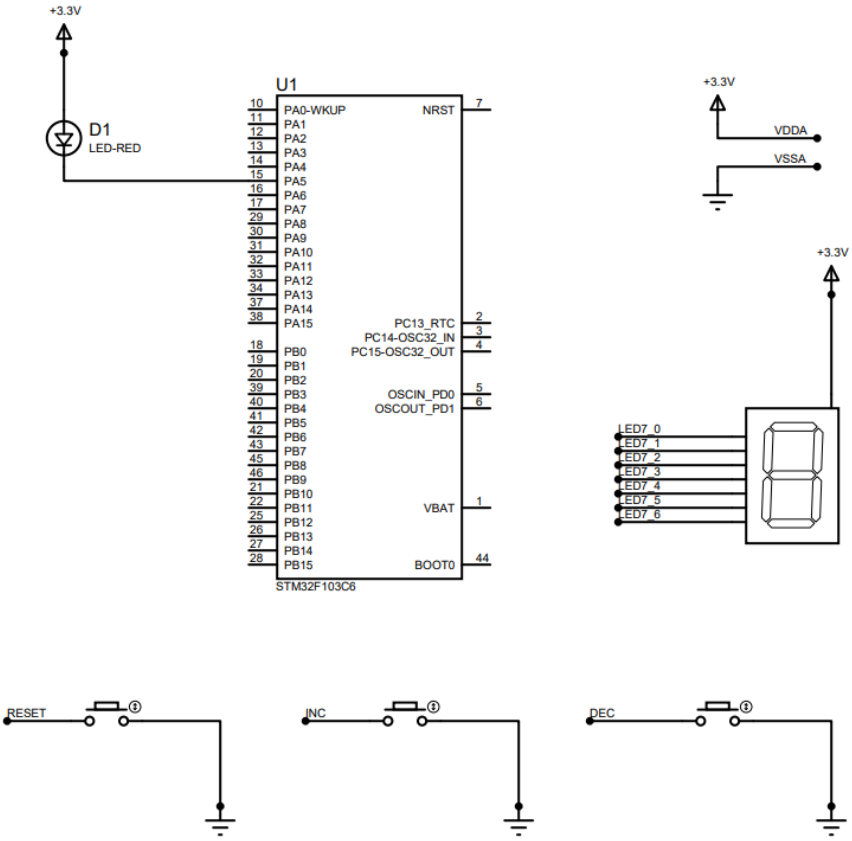
CHƯƠNG 1

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. 1.1, 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.

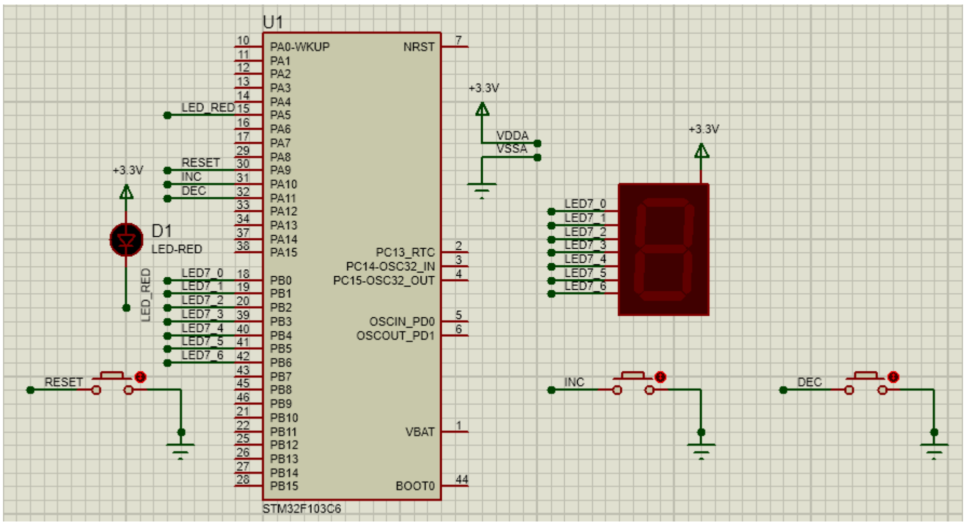
2 Implement and Report

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.

Schematics:



Hình 1.2: Proteus Schematics

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:

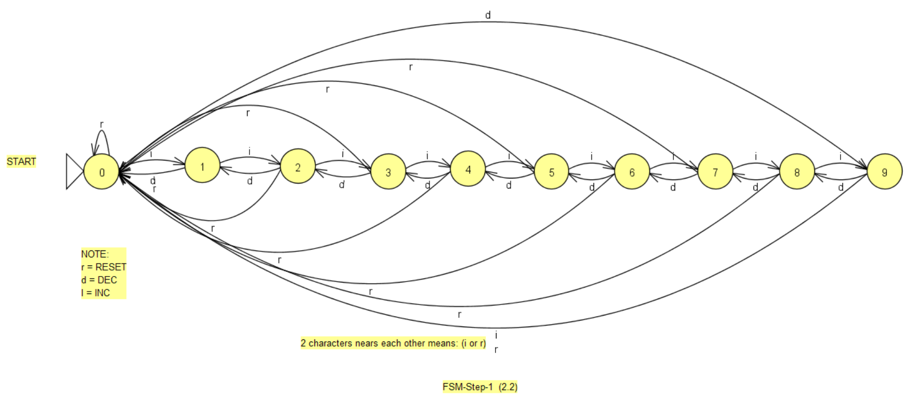
- Whenever the RESET is pressed, the counter value is 0.
- When INC is pressed, the counter is increased by 1. When counter is 9, it comes back to 0.

- 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.

FSM-Step-1:



Hình 1.3: FSM-Step-1 using JFLAP

The *State machine* is actually the state of the whole system. Coincidentally, each stage is represented by a number, which is the displayed output on the 7 segments LED. The system starts at 0, then depends on the input signal, the value is changed accordingly. Side note:

- The FSM is created via JFLAP, a FSM-building program with built in simulation.
- As noted within the figure, each letter reside on the arrow represents "the condition" for the system (machine) to change its state.
- If there are multiple characters near to each other, the logic of the condition is: <**a** *or* **b**>.

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

```
void fsm_simple_buttons_run() {
  switch(state) {
  case 0:
    display7SEG0(state);
```

```
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 1;
    else if (isButtonDECPressed() == 1) state = 9;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 1:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 2;
    else if (isButtonDECPressed() == 1) state = 0;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 2:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 3;
    else if (isButtonDECPressed() == 1) state = 1;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 3:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 4;
    else if (isButtonDECPressed() == 1) state = 2;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 4:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 5;
    else if (isButtonDECPressed() == 1) state = 3;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 5:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 6;
    else if (isButtonDECPressed() == 1) state = 4;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 6:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 7;
    else if (isButtonDECPressed() == 1) state = 5;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 7:
    display7SEG0(state);
```

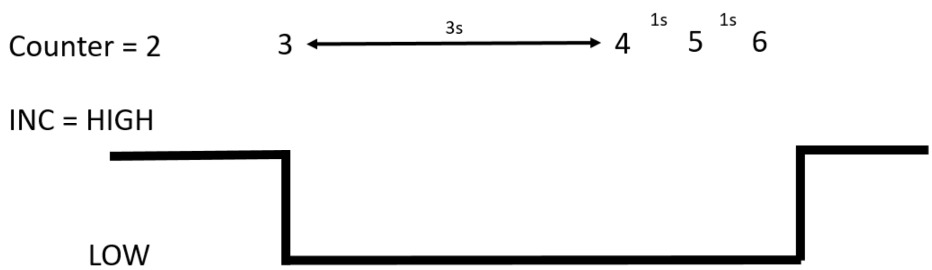
```
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 8;
    else if (isButtonDECPressed() == 1) state = 6;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 8:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 9;
    else if (isButtonDECPressed() == 1) state = 7;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  case 9:
    display7SEG0(state);
    setTimer1(1000);
    if (isButtonINCPressed() == 1) state = 0;
    else if (isButtonDECPressed() == 1) state = 8;
    else if (isButtonRESPressed() == 1) state = 0;
    break;
  default:
    break;
  }
}
```

Program 1.1: Implementation of the Simple 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.

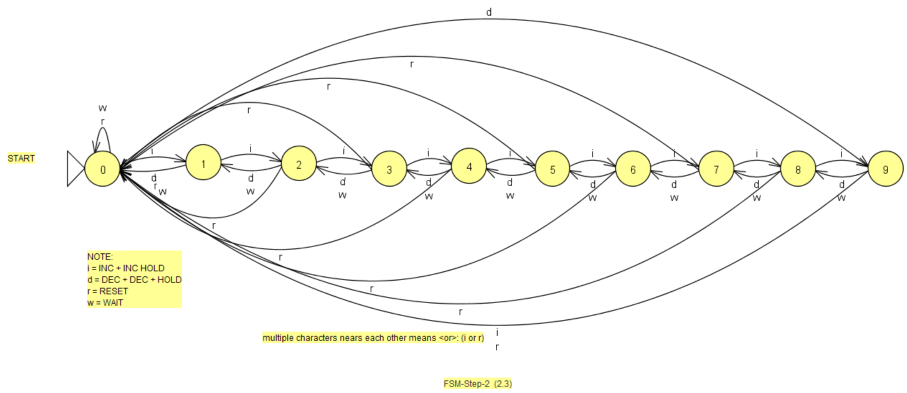
When a long-press event is detected, the value of counter keeps changing every 1 second until the button is released. For example, the current value of counter is 2 and the INC button is pressed. The value of counter immediately increased by 1, or counter = 3. The INC button keeps pressing for 3 seconds, 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 1.4: Long press behavior for INC button

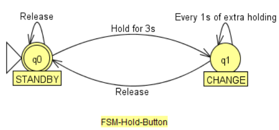
The behaviors of the DEC button are reversed to the INC button. The value of counter is also roll back if it reaches 0 or 9.

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



Hình 1.5: Full FSM for step 2

Due to my unorthodox implementation, I've create a separated *FSM* for holding buttons. Furthermore, the *FSM* has little different to the first one, since I merged the pairs of similar "conditions".



Hình 1.6: FSM for holding

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.

```
//This func do work
void subKeyProcess(int sw) {
```

```
  //HAL_GPIO_TogglePin(LED_TEST_GPIO_Port, LED_TEST_Pin);
  switch(sw) {
  case 0:
    button_inc_flag = 1;
    break;
  case 1:
    button_dec_flag = 1;
    break;
  case 2:
    button_reset_flag = 1;
    break;
  }
}

void holdKeyProcess(int sw) {
  //HAL_GPIO_TogglePin(LED_TEST_GPIO_Port, LED_TEST_Pin);
  switch(sw) {
  case 0:
    button_inc_flag = 1;
    break;
  case 1:
    button_dec_flag = 1;
    break;
  default:
    break;
  }
}

void getKeyInput(){
  for (int i = 0; i < 3; i++) {
    KeyReg0[i] = KeyReg1[i];
    KeyReg1[i] = KeyReg2[i];

    if (i == 0) {
      KeyReg2[i] = HAL_GPIO_ReadPin(INC_GPIO_Port, INC_Pin);
    }
    else if (i == 1) {
      KeyReg2[i] = HAL_GPIO_ReadPin(DEC_GPIO_Port, DEC_Pin);
    }
    else {
      KeyReg2[i] = HAL_GPIO_ReadPin(RESET_GPIO_Port, RESET_Pin);
    }

    if ((KeyReg0[i] == KeyReg1[i]) && (KeyReg1[i] == KeyReg2[i])) {
      if (KeyReg3[i] != KeyReg2[i]) {
```

```
        KeyReg3[i] = KeyReg2[i];
        if (KeyReg2[i] == PRESSED_STATE) {
          //TODO
          subKeyProcess(i);
          stop_hold = i;
          //TimerForKeyPress = 300;
        }
        //3s delay, or is it?

        //300 sometimes doesn't yeild the result of 3s delay on my machine.
        TimerForKeyPress = 300;
      }
      else {
        TimerForKeyPress--;
        if (TimerForKeyPress == 0) {
          if (KeyReg2[i] == PRESSED_STATE) {
              //TODO
            holdKeyProcess(stop_hold);
          }
          TimerForKeyPress = 100;
        }
      }
    }
  }
}
```

Note: Since I chose my *FSM* to represent the actual output of the 7-segments LED, just like the *FSM* for counters where each state is an output, the subsequences features' implementation are unorthodox, maybe even "out of the norms".

In this case, the "Hold button" feature is implemented in *button.c*,

2.4 State machine Step 3 - 2 points

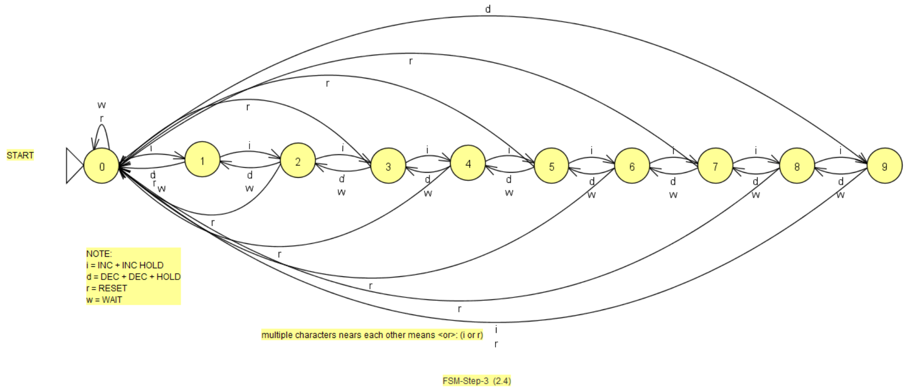
Finally, where there is no button event after 10 seconds, the value of of counter is counted down and stopped at 0. If the INC or DEC are pressed again, the status of the system comes back to previous state, which is designed in Subsection 2 or 3.

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

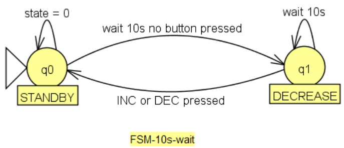
Again, since my abnormal implementation resulted in unorthodox *FSM*, here is the separate *FSM* for the automatic down counter.

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.

```
    setTimer1(100);
  }
  //LED-Blinking feature above
```



Hình 1.7: Full FSM for step 3



Hình 1.8: FSM for wait 10s

```c

  //10s countdown
  if (timer2_flag == 1) {
    if (state > 0) {
      state--;
    }
    //Decrease every 1s, for comfort
    setTimer2(100);
  }

    //Code continuation below
  switch(state) {
```

This feature ultilized one timer. Since the objective didn't clearly state in which interval the value gets decrements, so I chose 10 seconds as the interval between the changes.

Moreover, the down-counter clearly needs a "reset function" or "reset timer" while a button is pressed. This functionality is implemented in *button.c*, the example is of button Reset.

```c
int isButtonRESPressed() {
  if (button_reset_flag == 1) {
    button_reset_flag = 0;

    //RESET TIMER OF DECREMENTS
    setTimer2(1000);
    return 1;
  }
  return 0;
}
```

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.

```c
//Simple state
void fsm_simple_buttons_run() {
  //LED BLINKING
  if (timer1_flag == 1) {
    HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
    setTimer1(100);
  }
  //function continues below
```

I intergrated the blinking LED into the one-and-only FSM. The *INIT* timer was initialized towards the while loop in *main.c*.

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/notsotalented/mProcessor-Project24h

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

https://youtu.be/gOL3RSE-sxM

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) {
}
```

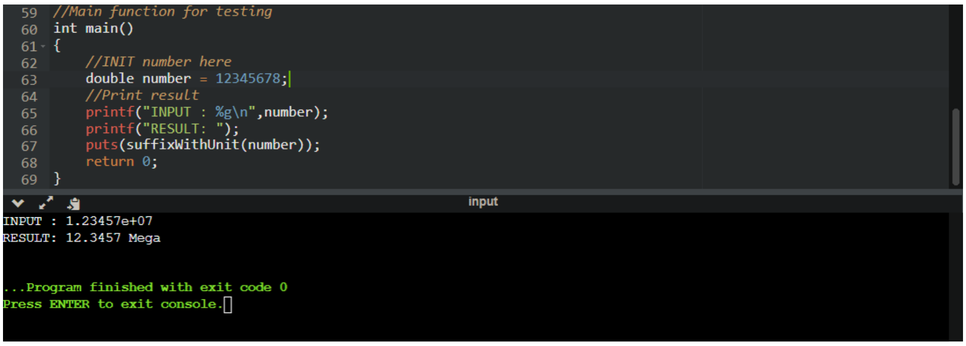
How would you solve them? Please share your thinking to solve this problem and provide your answer.

Answer:

```c
/*
    NguyenQuocAnh-1852238
    Programmed & Compiled on Online C Compiler
    https://www.onlinegdb.com/online_c_compiler
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Conversion function - the focus point
const char* suffixWithUnit(double number) {
    //INIT buffer
    static char buf[128];
    //INIT suffix table
    char suffix[8][10] = {" Kilo"," Mega"," Giga", " Tera",
   " Peta", " Exa", " Zetta", " Yotta"};
    //Intermediate to calculate number of power 10^unit =
  number
    double numberdiv = number;
    int unit = 0;
    while (numberdiv > 999) {
        numberdiv/=1000;
        unit++;
    }
    //Add suffix
    if (unit > 0) {
        if (unit < 9) {
            //Double to char array, %g copy without
  trailing zeros
            sprintf(buf, "%g", numberdiv);
            //Add suffix
            strncat(buf, suffix[unit - 1], 10);
        }
        else {
            //find 10^x 'tenx'
            int tenx = 0;
            double numberdiv10 = numberdiv;
            while (numberdiv10 > 9) {
            numberdiv10/=10;
            tenx++;
            }
            tenx+=unit*3;
            //Double to char array, %g copy without
  trailing zeros
            sprintf(buf, "%g", numberdiv10);
            //Add 'e+tenx'
            char esuffix[10];
            sprintf(esuffix, "%d", tenx - 24);
```

```c
                strncat(buf, "e+", 3);
                strncat(buf, esuffix, 10);
                //Add suffix
                strncat(buf, suffix[7], 10);
            }
        }
        else {
            //Double to char array, %g copy without trailing
   zeros
            sprintf(buf, "%g", numberdiv);
        }
        return buf;
}

//Main function for testing
int main()
{
        //INIT number here
        double number = 100e24;
        //Print result
        printf("INPUT : %g\n",number);
        printf("RESULT: ");
        puts(suffixWithUnit(number));
        return 0;
}
```

Summary:



Hình 1.9: Complied online

My perspective: one "unit conversion" program is needed, and since the subject is computer related, I chose the *Metric Prefix* syntax and rules `https://en.wikipedia.org/wiki/Metric_prefix#List_of_SI_prefixes` but they are implemented as *Suffixes*.

The program is tested and ran well on the Online C Compiler `https://www.onlinegdb.com/online_c_compiler`.

Background thoughts:

Provided that the *INPUT* follows "computer logic": non-negative (negative values are acceptable but not covered by the unit conversion), the *INPUT* will have a specific new suffix if the value surpasses 10^3. In other words, for every 10^3 increment of value, the suffix will change accordingly.

For example: $1000,000,000 = 1e+9 = 1\ Giga$. Until it reaches $1000\ Yotta$, then the syntax of suffix becomes $e+n\ Yotta$. For example: $1e26 = 100\ Yotta$; $1e28 = 1e+4\ Yotta$.

Finally, if the *INPUT* doesn't follow the above "computer logic", then the result will remain as is (no changes).

Quick recap of the implementation: The INPUT will be divided definitely by 1000 (to find the 10^3 increment). Appropriate suffixes will be attached accordingly.

If the value exceeds $999\ Yotta$, the leftover result will be divided definitely by 10, to find the "leftover 10^1 increments". Finally convert the result into the very last syntax as explained above.