ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

MICROPROCESSORS-MICROCONTROLLERS (CO3009)

TRAFFIC LIGHT ASSIGNMENT

GVHD: Lê Trọng Nhân

Sinh viên: Trần Viết Bình – 2112909
Nguyễn Hoàng Vương – 2112677
Đỗ Nguyên An – 2112726

Ho Chi Minh, Tháng 12/2023

Mục lục

1 Giới thiệu đề tài

Traffic Light System

Trong bối cảnh đô thị ngày càng phát triển, việc quản lý giao thông trở thành một thách thức đối với cộng đồng. Để giải quyết vấn đề này, nhóm chúng em đã thực hiện dự án môn học "Vi xử lý-vi điều khiển" với mục tiêu xây dựng một hệ thống đèn giao thông sử dụng vi điều khiển STM32F103C6RB và các thiết bị ngoại vi tương ứng.

Dự án của chúng em bao gồm ba chế độ hoạt động chính: **Chế độ Bình thường**, **Chế độ Điều chỉnh thời gian**, và **Chế độ điều khiển bằng tay**. Trong **Chế độ Bình thường**, hệ thống sẽ hoạt động theo quy tắc thông thường của đèn giao thông. Trong **Chế độ Điều chỉnh thời gian**, chúng em cung cấp khả năng điều chỉnh thời gian sáng/tắt cho từng đèn giao thông, tối ưu hóa luồng giao thông theo điều kiện thực tế, trong **Chế độ điều khiển bằng tay**, chúng em cung cấp khả năng điều khiển chuyển trạng thái sáng của từng đèn giao thông bằng tay. Cuối cùng, **Chế độ Người đi bộ** được thiết kế để tối ưu hóa an toàn cho người đi bộ, điều chỉnh đèn giao thông dựa trên sự xuất hiện của họ.

Về mã nguồn dự án sẽ được đặt ở link sau: https://github.com/AlmoedAli/PROJECT_MCU/tree/master

2 Tổng quan kiến thức

2.1 Ngắt Timer

Trong lập trình vi điều khiển, việc sử dụng ngắt timer là một phần quan trọng để đạt được các chức năng đồng bộ và thời gian chính xác. Vi điều khiển STM32, với khả năng tích hợp nhiều timer, cung cấp một nền tảng mạnh mẽ cho việc thực hiện các chức năng đa nhiệm và theo dõi thời gian.

Ngắt Timer trong Vi Điều Khiển STM32

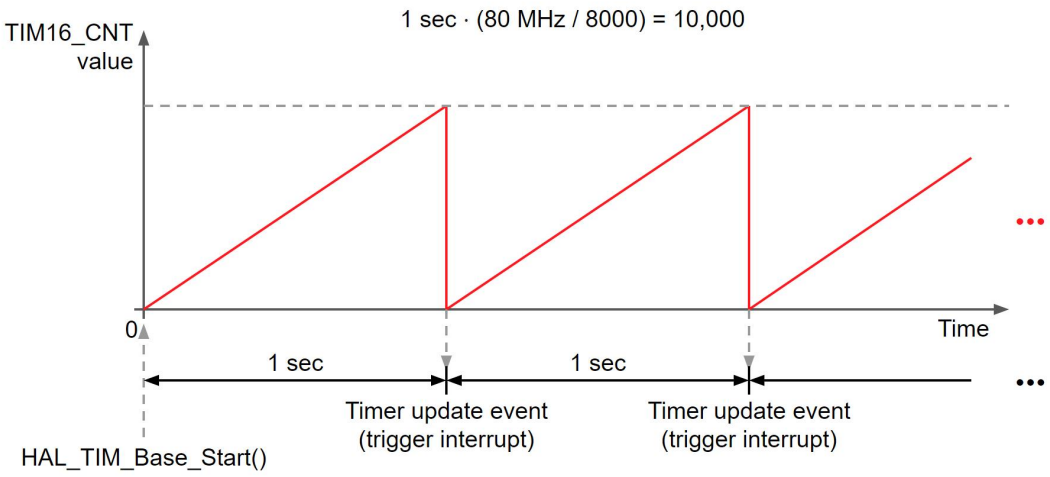
Ngắt timer là một cơ chế quan trọng giúp chúng ta thực hiện các nhiệm vụ theo chu kỳ hoặc theo thời gian cố định. Trong vi điều khiển STM32, các timer như Timer2, Timer3, v.v., có khả năng tạo ra ngắt định kỳ khi đến một giá trị được cấu hình trước đó. Các ngắt timer giúp chúng ta tự động thực hiện các hàm xử lý ngắt vào thời điểm xác định, tăng tính chính xác và hiệu suất của hệ thống.

Sử Dụng Clock Nội của STM32

Vi điều khiển STM32 thường được tích hợp với các nguồn clock nội, giúp cung cấp đồng bộ cho các thành phần của hệ thống. Sử dụng clock nội giúp tiết kiệm năng lượng và giảm độ phức tạp của hệ thống, đồng thời cung cấp khả năng đồng bộ cao giữa các thành phần.

Đối với dự án của chúng em, việc sử dụng ngắt timer và clock nội của STM32 là quan trọng để đảm bảo chính xác trong việc điều khiển thời gian và quy trình của hệ thống đèn giao thông thông minh. Thông qua việc lập trình ngắt timer và sử dụng clock nội, chúng tôi có thể linh hoạt và hiệu quả quản lý các chế độ hoạt động khác nhau của hệ thống.

Dưới là biểu đồ hoạt động của một Timer:



2.2 Nút nhấn

Cơ Chế Hoạt Động của Nút Nhấn

Nút nhấn, hay còn được gọi là nút bấm, là một thành phần cơ bản trong các hệ thống điều khiển. Khi được nhấn, nút sẽ tạo ra một tín hiệu điện mức thấp hoặc thay đổi trạng thái điện mức của mạch, thông báo về sự tương tác từ người sử dụng.

Chống Rung cho Nút Nhấn

Chống rung là một vấn đề thường gặp khi sử dụng nút nhấn. Do độ cơ học của nút, tín hiệu có thể chuyển động giữa hai trạng thái cao và thấp một số lần trước khi ổn định. Điều này có thể dẫn đến sự nhầm lẫn trong việc đọc trạng thái thực tế của nút.

Phương Án Phần Cứng Chống Rung

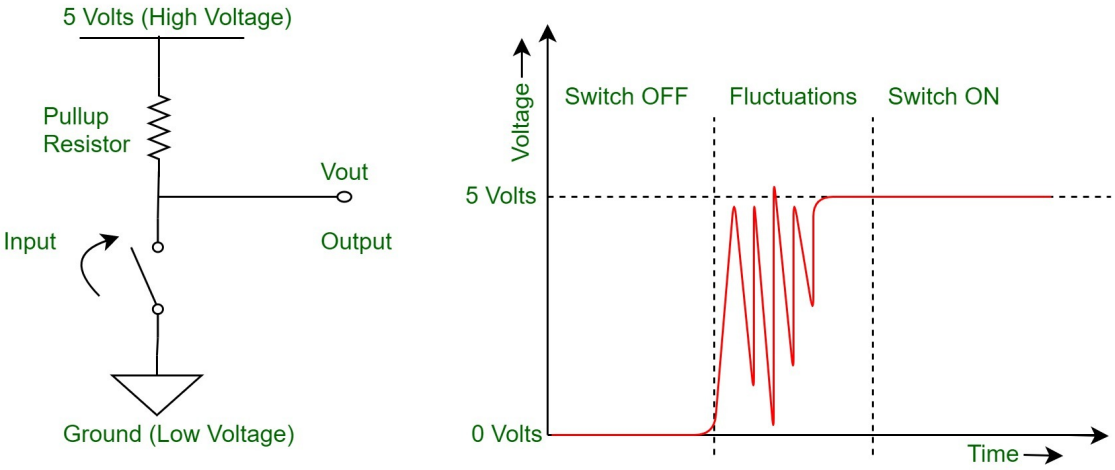
Phương án phần cứng thường bao gồm việc sử dụng các linh kiện như tụ điện và trở kéo lên (pull-up resistor) để làm giảm tác động của sự chuyển động cơ học. Tuy nhiên, các phương án này có thể phức tạp và không linh hoạt khi cần điều chỉnh.

Phương Án Phần Mềm Chống Rung

Phương án phần mềm chống rung được thực hiện thông qua việc lập trình để xử lý sự chuyển động không ổn định của nút. Các thuật toán chống rung thông dụng bao gồm sử dụng hàm thời gian chờ (debouncing) để đảm bảo rằng chỉ có một tín hiệu ổn định được đọc sau một khoảng thời gian nhất định kể từ lần nhấn đầu tiên.

Tổng Kết Lựa Chọn Phương Án Phần Mềm Chống Rung

Trong dự án của chúng em đã quyết định sử dụng phương án phần mềm để chống rung cho nút nhấn. Lựa chọn này mang lại sự linh hoạt cao và dễ dàng điều chỉnh để đáp ứng yêu cầu cụ thể của hệ thống. Bằng cách sử dụng thuật toán chống rung trong phần mềm, chúng em đảm bảo tính ổn định và chính xác của thông tin đọc từ nút nhấn, giảm thiểu nhầm lẫn và tăng trải nghiệm người sử dụng.



2.3 Cooperative Scheduling

Khái Niệm về Cooperative Scheduling

Cooperative Scheduling là một mô hình lập lịch (scheduling) trong hệ thống nhúng, nơi mà các tiến trình hoặc nhiệm vụ (tasks) hoạt động cùng một lịch trình và tự chủ động chia sẻ tài nguyên. Trái với mô hình lập lịch preemptive, Cooperative Scheduling không buộc chấp nhận tác động gián đoạn từ hệ điều hành, mà thay vào đó yêu cầu sự hợp tác giữa các tiến trình.

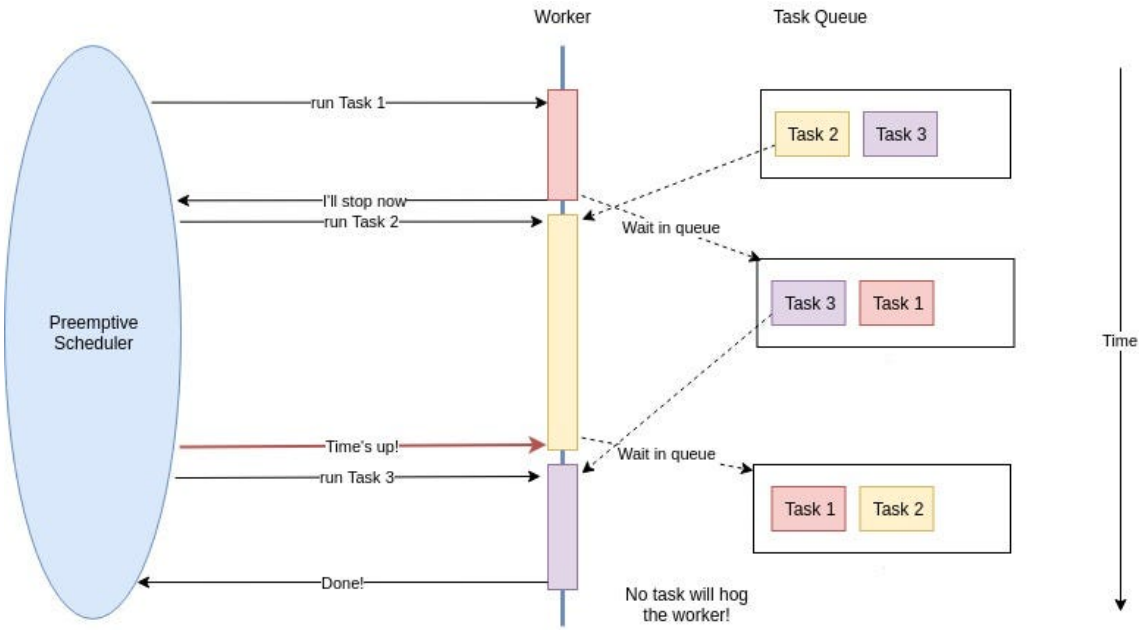
Áp Dụng Cooperative Scheduling Cho Hệ Thống Đèn Giao Thông

Trong hệ thống đèn giao thông, việc sử dụng Cooperative Scheduling có thể mang lại nhiều lợi ích. Các chế độ hoạt động khác nhau của đèn, như chế độ bình thường, chế độ điều chỉnh thời gian, và chế độ cho người đi bộ, có thể được coi là các nhiệm vụ riêng biệt. Mỗi nhiệm vụ này được thiết kế để hoạt động trong một khoảng thời gian xác định, và chúng có thể chia sẻ tài nguyên mà không cần đến sự can thiệp gián đoạn từ hệ điều hành.

Trong mô hình này, mỗi chế độ hoạt động sẽ chủ động chia sẻ tài nguyên hệ thống, như quản lý thời gian đèn, đảm bảo rằng các chế độ hoạt động khác nhau có thể thực hiện công việc mà không gặp xung đột. Việc này giúp cải thiện độ tin cậy và ổn định của hệ thống đèn giao thông.

Lợi Ích Từ Sự Hợp Tác trong Cooperative Scheduling

Sự hợp tác giữa các tiến trình trong Cooperative Scheduling giúp giảm thiểu tình trạng cạnh tranh và cải thiện hiệu suất của hệ thống. Đồng thời, nó cũng tạo điều kiện thuận lợi để quản lý tài nguyên và đồng bộ hóa các nhiệm vụ, đặc biệt quan trọng trong các ứng dụng như hệ thống đèn giao thông nơi độ chính xác và đồng đều của các chế độ hoạt động là quan trọng.



2.4 Giao tiếp bất đồng bộ - UART

Khái Niệm về Giao Tiếp Bất Đồng Bộ

Giao tiếp bất đồng bộ (UART) là một phương pháp truyền thông giữa các thiết bị điện tử, trong đó dữ liệu được truyền đi theo kiểu không đồng bộ, tức là không có đồng hồ chung giữa bộ truyền và bộ nhận. Thay vào đó, mỗi ký tự được truyền đi với một bit đồng hồ riêng biệt.

Ứng Dụng của UART trong Hệ Thống Đèn Giao Thông

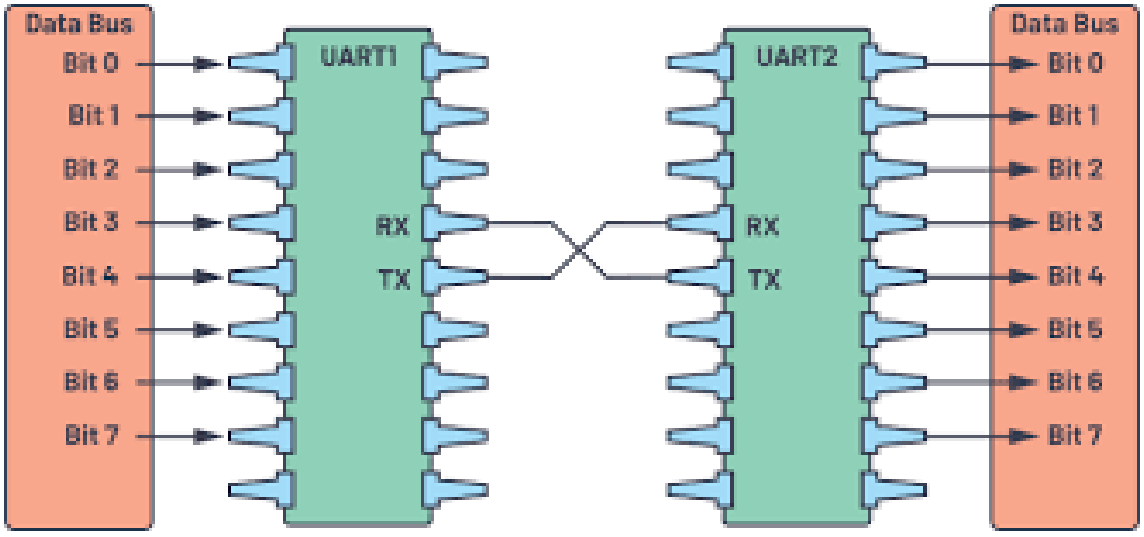
Trong hệ thống đèn giao thông, UART có thể được sử dụng để thiết lập giao tiếp giữa các thành phần khác nhau của hệ thống. Ví dụ, vi điều khiển chính có thể sử dụng UART để liên lạc với các đèn giao thông cụ thể hoặc các thiết bị đo lường như cảm biến. Giao tiếp bất đồng bộ này giúp truyền thông dữ liệu một cách linh hoạt và hiệu quả.

Ưu Điểm của Giao Tiếp Bất Đồng Bộ

Giao tiếp bất đồng bộ mang lại nhiều ưu điểm cho hệ thống nhúng. Đầu tiên, nó cho phép truyền dữ liệu trong thời gian thực mà không cần sự đồng bộ chặt chẽ. Thứ hai, vì không cần một tín hiệu đồng hồ chung, việc kết nối giữa các thiết bị có thể thực hiện dễ dàng và không đòi hỏi nhiều tài nguyên.

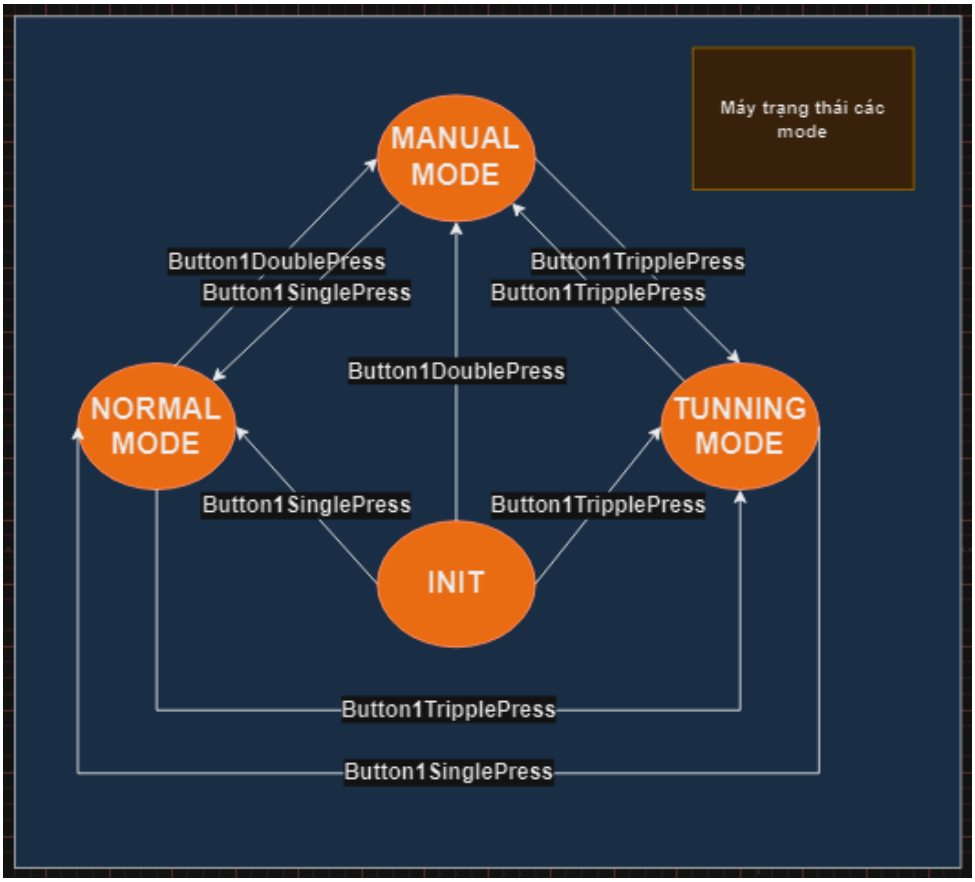
Lợi Ích Trong Quản Lý Hệ Thống Đèn Giao Thông

Trong hệ thống đèn giao thông của chúng em, việc sử dụng UART cho phép các thành phần khác nhau của hệ thống truyền thông dữ liệu một cách hiệu quả. Điều này có thể bao gồm việc chuyển đổi giữa các chế độ hoạt động, nhận dữ liệu từ các cảm biến, hoặc thậm chí định cấu hình hệ thống từ xa. Bằng cách này, UART chơi một vai trò quan trọng trong việc đảm bảo tính linh hoạt và kết nối tốt giữa các thành phần của hệ thống đèn giao thông.



3 Hiện thực dự án

3.1 Tổng quan



Hình 1: *Các mode của toàn bộ hệ thống*

Tổng quan dự án: hệ thống đèn giao thông sẽ có 4 trạng thái:

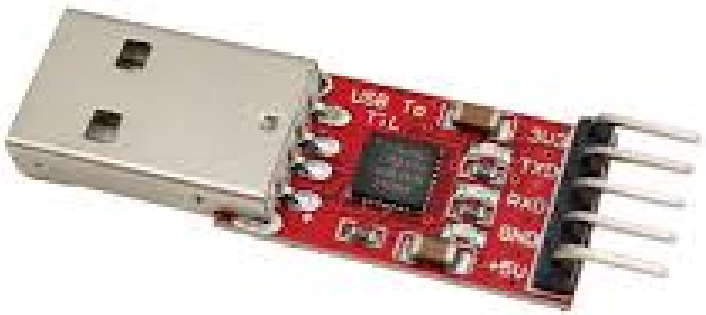
- **INIT**: Trạng thái khởi tạo của hệ thống.
- **NORMAL MODE**: Trạng thái hoạt động bình thường của đèn giao thông với thời gian cài đặt cho 3 đèn là đỏ, vàng, xanh lần lượt là 5, 2, 3 (giây)
- **MANUAL MODE**: Trạng thái điều chỉnh hệ thống đèn bằng tay cho người điều khiển.
- **TUNNING MODE**: Trạng thái điều chỉnh thời gian cho từng đèn và phải thoả mãn điều kiện thời gian đèn xanh cộng đèn vàng phải bằng với đèn đỏ.

Tất cả các trạng thái của hệ thống được mô tả theo link sau: https://app.diagrams.net/#G1bsWY0a2mD0NHFUBmuCIxr4l5El7R_Pc9

Vi điều khiển được sử dụng trong mạch là STM32F103RB

Hình 2: *Hình ảnh minh hoạ*

Ngoài ra các thiết bị ngoại vi và thiết bị chuyển đổi được dùng trong dự án: LED đơn, Buzzer, mạch chuyển USB sang TTL CP2102.



Hình 3: *Mạch chuyển USB sang TTL CP2102*

3.2 Hiện thực các hàm chức năng của dự án

3.2.1 Liệt kê các file mã nguồn trong dự án:

- button.c: xử lý nút nhấn (chống run, nhấn đúp, ...)
- deviceDriverSingleLed.c: điều khiển các led đơn bằng cách gọi các function của lớp physical.
- physicalSingleLed.c: lớp physical điều khiển các led đơn.
- ledWalk.c : xử lý các chức năng của đèn dành cho người đi bộ.
- pwn.c: tạo xung analog với kỹ thuật PWN, trong file này còn có cấu hình của TIMER để xử lý cho còi (buzzer).
- manualMode.c: xử lý cho trạng thái MANUAL
- normalMode.c: xử lý cho trạng thái NORMAL
- tunningMode.c: xử lý cho trạng thái TUNNING MODE
- uart.c: xử lý tác vụ giao tiếp bất đồng bộ với terminal (sử dụng Hercules)
- scho1.c: hiện thực cooperative schedulling.

3.2.2 NORMAL STATE

Trước hết ta nói đến trạng thái NORMAL, đây là trạng thái hệ thống đèn sẽ hoạt động bình thường theo chu trình lần lượt: 2 hệ thống đèn giao thông, 1 bên nếu đỏ trước thì bên kia sẽ là xanh và ngược lại, đỏ sẽ đếm trong 5 giây và xanh trong 3 giây, giữa đỏ và xanh, vàng sẽ đếm trong 2 giây.

3.2.3 MANUAL STATE

Trạng thái này có thể điều chỉnh bằng tay để chuyển từ cặp trạng thái của hai hướng đèn giao thông khác nhau. Điều khiển bằng nút bấm, khi bấm có thể chuyển trạng thái của hai hệ thống.

3.2.4 TUNNING STATE

Trạng thái này giúp điều chỉnh lại thời lượng của đèn đỏ, xanh, vàng cho cả hai hệ thống đèn.

3.2.5 Về Cooperative Scheduling trong hệ thống.

```
#include "scho1.h"

#include <deviceDriver7Segment.h>
#include "normalMode.h"
#include <main.h>
#include "deviceDriverSingleLed.h"
```

```
#include "softwareTimer.h"
#include "ledWalk.h"
#include "uart.h"
#include "scho1.h"
#include "button.h"
#include "tuningMode.h"
#include "pwm.h"

typedef struct
{
  // Pointer to the task
  void (*pTask)(void);
  // Delay (ticks) until the function will (next) be run
  uint32_t Delay;
  // Interval (ticks) between subsequent runs
  uint32_t Period;
  // Incremented (by scheduler) when task is due to execute
  uint8_t RunMe;
  // This is a hint to solve the question below
  uint32_t TaskID;
} sTask;

uint8_t nTask = 0;
sTask SCH_tasks_G[SCH_MAX_TASKS];

// Ham giam dan thoi gian delay va set RunMe len de execute
void SCH_Update()
{
  if (SCH_tasks_G[0].pTask)
  {
    if (SCH_tasks_G[0].Delay == 0)
    {
      SCH_tasks_G[0].RunMe += 1;
    }
    else
    {
      SCH_tasks_G[0].Delay--;
    }
  }
}

// Ham them task vao mang
void SCH_Add_Task(void(*pFunction), const uint32_t Delay, const uint32_t
    Period, uint8_t taskID)
{
  int index = 0;
  int total_delay = Delay / 10;

  for (int i = 0; i < SCH_MAX_TASKS; i++)
  {
    if (SCH_tasks_G[i].pTask)
    {
      if (SCH_tasks_G[i].Delay <= total_delay)
      {
        total_delay = total_delay - SCH_tasks_G[i].Delay;
      }
```

```
      else
      {
        index = i;
        break;
      }
    }
    else
    {
      index = i;
      break;
    }
  }
  for (int i = SCH_MAX_TASKS; i > index; i--)
  {
    if (SCH_tasks_G[i - 1].pTask)
    {
      SCH_tasks_G[i].pTask = SCH_tasks_G[i - 1].pTask;
      SCH_tasks_G[i].Delay = SCH_tasks_G[i - 1].Delay;
      SCH_tasks_G[i].Period = SCH_tasks_G[i - 1].Period;
      SCH_tasks_G[i].RunMe = SCH_tasks_G[i - 1].RunMe;
      SCH_tasks_G[i].TaskID = SCH_tasks_G[i - 1].TaskID;
    }
  }
  SCH_tasks_G[index].pTask = pFunction;
  SCH_tasks_G[index].Delay = total_delay;
  SCH_tasks_G[index].Period = Period;
  SCH_tasks_G[index].RunMe = 0;
  SCH_tasks_G[index].TaskID = taskID;
  if (SCH_tasks_G[index + 1].pTask)
  {
    SCH_tasks_G[index + 1].Delay = SCH_tasks_G[index + 1].Delay -
   total_delay;
  }
}

void SCH_Delete(uint8_t index)
{
  SCH_tasks_G[index].Delay = 0;
  SCH_tasks_G[index].Period = 0;
  SCH_tasks_G[index].RunMe = 0;
  SCH_tasks_G[index].pTask = 0x0000;
  SCH_tasks_G[index].TaskID = 0;
}

void SCH_Delete_ID(uint8_t ID)
{
  uint8_t index = 0;
  uint8_t final = 0;

  for(int i = 0; i < SCH_MAX_TASKS; i++)
  {
    if( SCH_tasks_G[i].TaskID == ID){
      SCH_Delete(i);
      index = i;
      break;
    }
```

```
  }
  for (int i = index; i < SCH_MAX_TASKS; i++)
  {
    SCH_tasks_G[i].pTask = SCH_tasks_G[i + 1].pTask;
    SCH_tasks_G[i].Delay = SCH_tasks_G[i + 1].Delay;
    SCH_tasks_G[i].Period = SCH_tasks_G[i + 1].Period;
    SCH_tasks_G[i].RunMe = SCH_tasks_G[i + 1].RunMe;
    SCH_tasks_G[i].TaskID = SCH_tasks_G[i + 1].TaskID;
    if (SCH_tasks_G[i].pTask == 0x0000 && i != index)
    {
      final = i;
      break;
    }
  }
  SCH_Delete(final);
}

// Xoa task dau tien
void SCH_Delete_Task(uint8_t index)
{
  int final = 0;
  SCH_Delete(index);
  for (int i = index; i < SCH_MAX_TASKS; i++)
  {
    SCH_tasks_G[i].pTask = SCH_tasks_G[i + 1].pTask;
    SCH_tasks_G[i].Delay = SCH_tasks_G[i + 1].Delay;
    SCH_tasks_G[i].Period = SCH_tasks_G[i + 1].Period;
    SCH_tasks_G[i].RunMe = SCH_tasks_G[i + 1].RunMe;
    SCH_tasks_G[i].TaskID = SCH_tasks_G[i + 1].TaskID;
    if (SCH_tasks_G[i].pTask == 0x0000)
    {
      final = i;
      break;
    }
  }
  SCH_Delete(final);
}

// Ham check xem task0 da can execute chua
void SCH_Dispatch_Tasks(void)
{
  if (SCH_tasks_G[0].pTask)
  {
    if (SCH_tasks_G[0].RunMe > 0)
    {
      (*SCH_tasks_G[0].pTask)();
      SCH_tasks_G[0].RunMe--;

      if (SCH_tasks_G[0].Period)
      {
        SCH_Add_Task(SCH_tasks_G[0].pTask, SCH_tasks_G[0].Period,
            SCH_tasks_G[0].Period, SCH_tasks_G[0].TaskID);
      }
      SCH_Delete_Task(0);
    }
  }
```

```
}
// Ham xoa tat ca cac task trong array, khien array nhu vua duoc khoi
    tao
void SCH_Init(void)
{
  uint8_t i;
  for (i = 0; i < SCH_MAX_TASKS; i++)
  {
    SCH_tasks_G[i].Delay = 0;
    SCH_tasks_G[i].Period = 0;
    SCH_tasks_G[i].RunMe = 0;
    SCH_tasks_G[i].pTask = 0x0000;
    SCH_tasks_G[i].TaskID = 0;
  }
}

//function for run normal mode every 1s
void task1()
{
  if (modeStatus == NORMALMODE)
  {
    runNormalMode();
    ledWalkOperationNormalMode();
  }
}

//function for run tuning mode every 0.5s
void task2()
{
  if (modeStatus == TUNINGMODE)
  {
    animationTuningMode();
  }
}

//function for get button every 10ms
void task3()
{
  getInputButton();
}

void task4()
{
  buzzerRun();
}
```

SCH_Dispatch_Task() sẽ được gọi ở while(1) và SCH_Update() sẽ được gọi trong hàm HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)

3.2.6 Buzzer

Buzzer ở đây được hiện thực như sau bằng kỹ thuật PWN:

```
void buzzerSetup()
{
  pre = 399;
  com = 400;
}

void buzzerRun()
{
  if(flagBuzzer == 1)
  {
    __HAL_TIM_SET_PRESCALER(&htim3, pre);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, com);
  }
  if(com < 9000)
    com = com + 400;
  if(pre > 0)
    pre-= 25;
}

void buzzerBegin()
{
  flagBuzzer = 1;
}


void buzzerOff()
{
  flagBuzzer=0;
  __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
}
```

4 Tham khảo

Link dự án của nhóm: https://github.com/AlmoedAli/PROJECT_MCU/tree/master
Link state machine: https://app.diagrams.net/#G1bsWY0a2mD0NHFUBmuCIxr4l5El7R_Pc9
Tài liệu tham khảo: MCU Labs HK231

——————— FINISH ———————