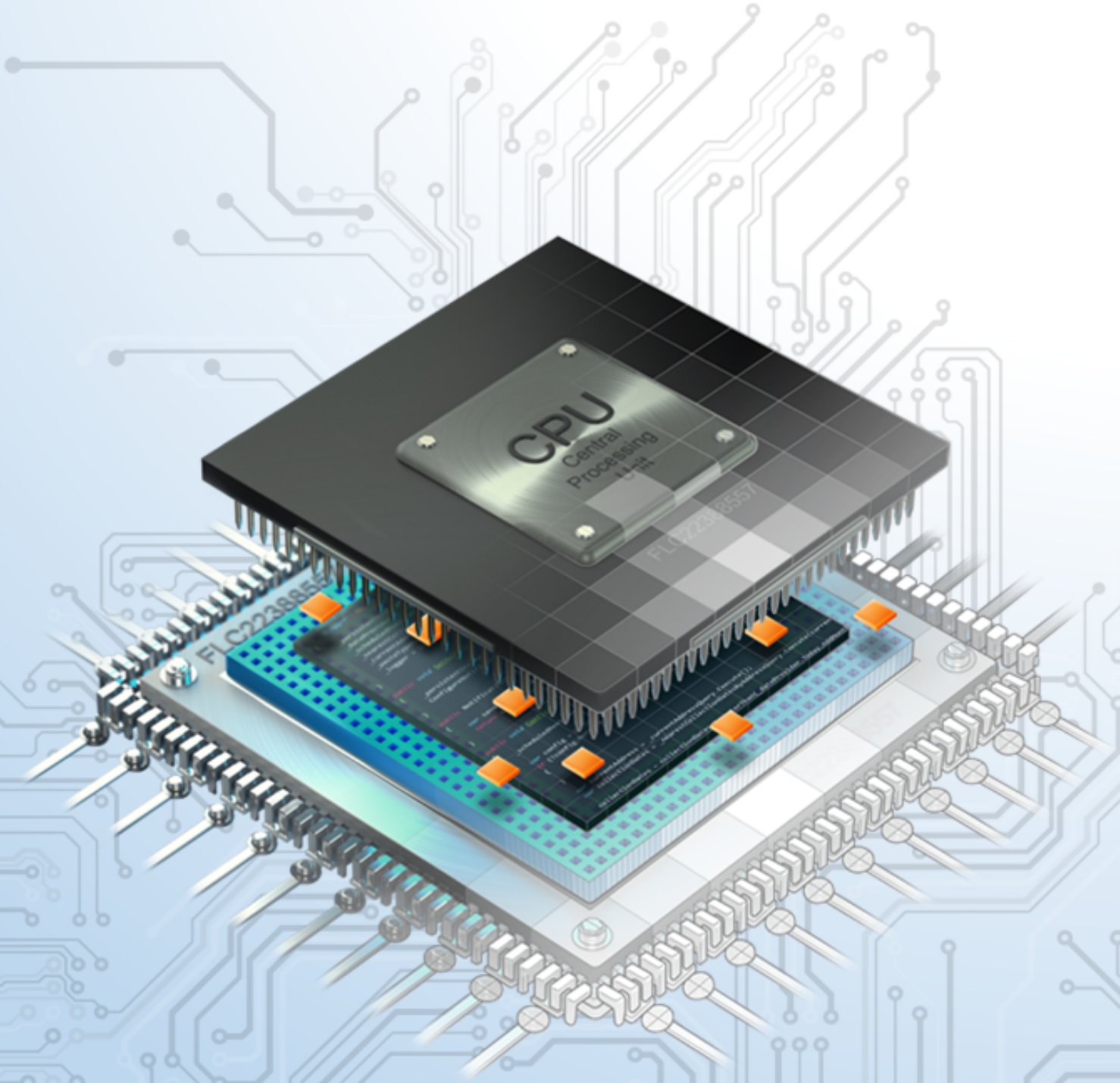
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

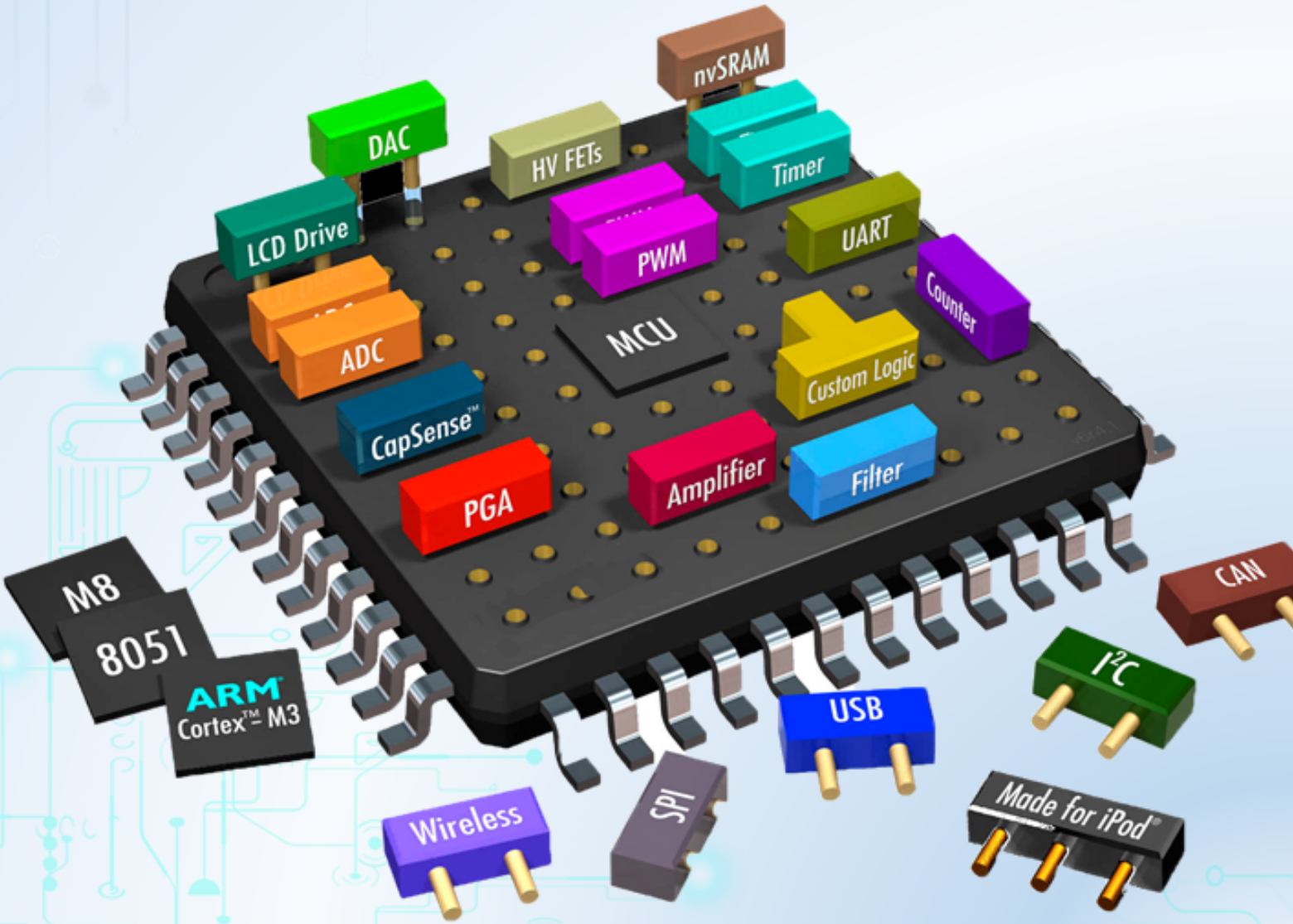


Dr. Le Trong Nhan

Mục lục

CHƯƠNG 1

Flow and Error Control in Communication



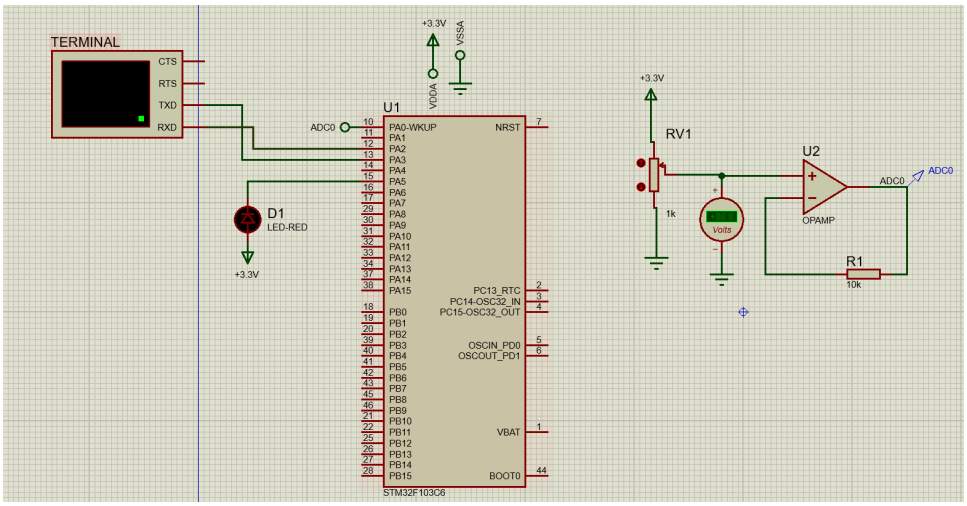
1 Project description

In this lab, a simple communication protocol is implemented as follows:

- From the console, user types **!RST#** to ask for a sensory data.
- The STM32 response the ADC_value, following a format **!ADC=1234#**, where 1234 presents for the value of ADC_value variable.
- The user ends the communication by sending **!OK#**

The timeout for waiting the **!OK#** at STM32 is 3 seconds. After this period, its packet is sent again. **The value is kept as the previous packet.**

1.1 Proteus Schematic



*Hình 1.1: Schematic for Scheduler test- **Lab6.pdsprj***

1.2 Soft timer

I've implemented a soft timer to use in uart_communication and in main.c for led blinking.

1.2.1 timer.h

```
/*
 * timer.h
 *
 *  Created on: Nov 20, 2021
 *      Author: ngocc
 */

#ifndef INC_TIMER_H_
#define INC_TIMER_H_

void initTimer(int timer_index);
```

```
12
13 void setTimer(int timer_index, int duration);
14
15 void timer_run(int timer_index);
16
17 int getTimerFlag(int timer_index);
18
19 void stopTimer(int index_timer);
20
21 #endif /* INC_TIMER_H_ */
```

1.2.2 timer.c

```
1  /*
2   * timer.c
3   *
4   *  Created on: Nov 20, 2021
5   *      Author: ngocc
6   */
7
8  #define NUM_OF_TIMER 3
9  #define PERIOD_DURATION 10
10
11 static int timerFlag[NUM_OF_TIMER];
12 static int timerCounter[NUM_OF_TIMER];
13
14 void initTimer(int timer_index) {
15   timerFlag[timer_index] = 1;
16 }
17
18 void setTimer(int timer_index, int duration) {
19   timerCounter[timer_index] = duration / PERIOD_DURATION;
20   timerFlag[timer_index] = 0;
21 }
22
23 void timer_run(int timer_index) {
24   if (timerCounter[timer_index] > 0){
25     timerCounter[timer_index]--;
26     if (timerCounter[timer_index] == 0) {
27       timerFlag[timer_index] = 1;
28     }
29   }
30 }
31
32 int getTimerFlag(int timer_index) {
33   if (timer_index >= NUM_OF_TIMER) return -1;
34   return timerFlag[timer_index];
35 }
36
37 void stopTimer(int index_timer) {
```

```
38   timerFlag[index_timer] = 0;
39   timerCounter[index_timer] = 0;
40 }
```

1.3 Command parser

This module is used to received a command from the console

For command parser module, we have 2 files as follow:

1.3.1 command_parser.h

```
1  /*
2   * command_parser.h
3   *
4   *  Created on: Nov 20, 2021
5   *      Author: ngocc
6   */
7
8  #ifndef INC_COMMAND_PARSER_H_
9  #define INC_COMMAND_PARSER_H_
10 #include "main.h"
11
12 extern unsigned char command_done;
13
14 void command_parser_fsm(uint8_t * buffer, unsigned int
       index);
15
16 char * getCommand();
17
18 void clear_command();
19
20 #endif /* INC_COMMAND_PARSER_H_ */
```

- command_done is a signal variable, will be set to 1 if a command is done (command read '#')
- command_parser_fsm() is a fsm to parse command input, and store that command inside command[30] array
- getCommand() is used to get the command from command[30] array
- clear_command() is used to clear the command[30] array

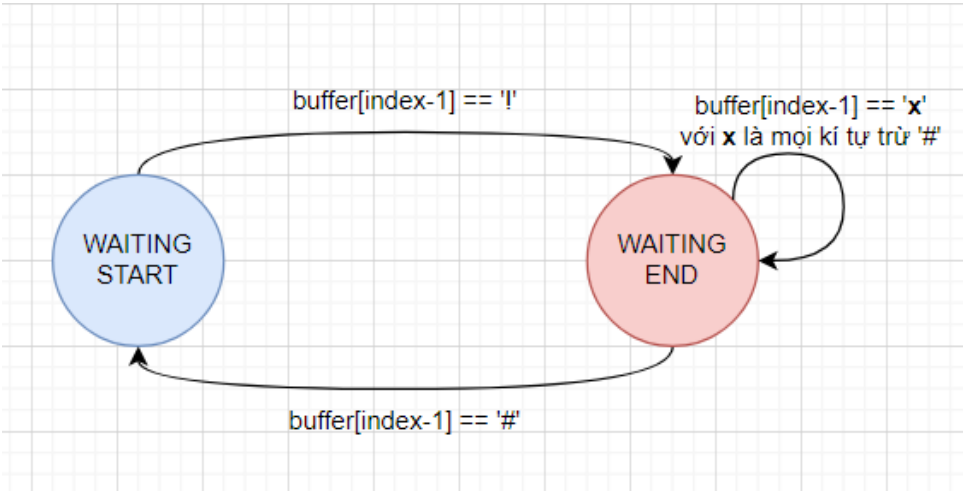
1.3.2 command_parser.c

```
1  /*
2   * command_parser.c
3   *
4   *  Created on: Nov 20, 2021
5   *      Author: ngocc
6   */
```

```c
#include "command_parser.h"
#include "main.h"


enum parser_state {WAITING_START, WAITING_END};

enum parser_state parser_State = WAITING_START;

char command[30];
unsigned char command_index = 0;
unsigned char command_done = 0;

UART_HandleTypeDef huart2;
//static uint8_t str[30] = "I was here!";

void command_parser_fsm(uint8_t * buffer, unsigned int
   index) {
  switch(parser_State) {
  case WAITING_START:
    if (buffer[index-1] == '!') {
//      HAL_UART_Transmit(&huart2, str, sizeof(str), sizeof
   (str) * 50);
      parser_State = WAITING_END;
      command_index = 0;
    }
    break;
  case WAITING_END:
    if (buffer[index-1] == '#') {
      parser_State = WAITING_START;
      command[command_index] = '\0';
      command_done = 1;
    } else {
      command[command_index++] = buffer[index-1];
      if (command_index == 30) command_index = 0;
    }
    break;
  }
}

char * getCommand() {
  return command;
}

void clear_command() {
  command[0] = '\0';
}
```

We have the FSM of the parser like this:

Function **command_parser_fsm** will read the buffer and index of a buffer. At WAIT-



Hình 1.2: command_parser_fsm()

ING_START state, it wait for character '!' at buffer[index-1]. If character '!' is enter, state is changed to WAITING_END and we also initialize the command buffer by assign command_index to 0.

In WATING_END state, we'll continuously read the input char, if input char is not '#' we put it inside command array next to others. If input char is '#', that mean the command is complete, we assign command_done to 1 to signal that command is done, and end the string with '\0'. Then, we'll go back to WAITING_START state, waiting for the next command.

1.4 uart_communication

In this module, we'll get all information needed from command parser (command_done signal and command array), get ADC value from hadc1 and print it to the terminal by UART if certain condition is met. **uart_communication_fsm** is the only function in this module.

1.4.1 uart_communication.h

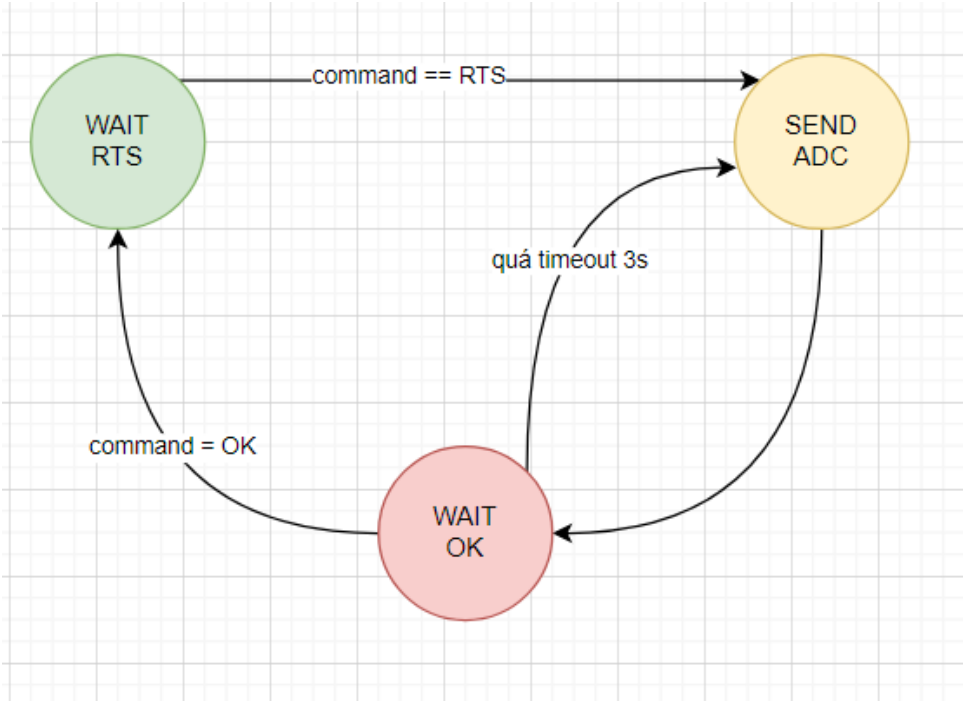
```
/*
 * uart_communication.h
 *
 *  Created on: Nov 20, 2021
 *      Author: ngocc
 */

#ifndef INC_UART_COMMUNICATION_H_
#define INC_UART_COMMUNICATION_H_


void uart_communication_fsm();
#endif /* INC_UART_COMMUNICATION_H_ */
```

1.4.2 uart_communication.c

```c
/*
 * uart_communication.c
 *
 *  Created on: Nov 20, 2021
 *      Author: ngocc
 */
#include "command_parser.h"
#include "main.h"
#include "uart_communication.h"
#include <string.h>
#include <stdio.h>
#include "timer.h"

ADC_HandleTypeDef hadc1;

UART_HandleTypeDef huart2;

enum commu {WAIT_RTS, WAIT_OK, SEND_ADC};
enum commu commu_state = WAIT_RTS;
int ADC_value = 0;
char str[20];

static uint8_t strs[30];

void uart_communication_fsm() {
  switch (commu_state) {
  case WAIT_RTS:
    if (command_done == 1) {
      command_done = 0;
      HAL_UART_Transmit(&huart2, strs, sprintf((char*)strs, "\r\ninput string is: %s \r\n", getCommand()), 1000);
      if (strcmp(getCommand(), "RTS") == 0) {
        ADC_value = HAL_ADC_GetValue(&hadc1);
//        HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!ADC=%d#\r\n", ADC_value), 1000);
        commu_state = SEND_ADC;
//        counter_uart = TIME_OUT;
        setTimer(2,3000);
      }
    }
    break;
  case SEND_ADC:
      HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!ADC=%d#\r\n", ADC_value), 1000);
      commu_state = WAIT_OK;
      break;
  case WAIT_OK:
    if (command_done == 1) {
      command_done = 0;
```

```
47
48        HAL_UART_Transmit(&huart2, strs, sprintf((char*)strs,
      "\r\ninput string is: %s \r\n", getCommand()), 1000);
49        if (strcmp(getCommand(), "OK") == 0) {
50          commu_state = WAIT_RTS;
51          stopTimer(2);
52        }
53      }
54      if (getTimerFlag(2) == 1) {
55        commu_state = SEND_ADC;
56        setTimer(2, 3000);
57      }
58
59      break;
60    }
61  }
```

The FSM for uart_communication consist of 3 state:



Hình 1.3: uart_communication_fsm()

- WAIT_RTS: wait for command RTS. If command RTS is get, we read ADC value and change state to SEND_ADC to print it to the terminal through UART. We use strcmp() to compare the command with "RTS", this function return 0 if 2 string match each other. We also use timer2 as a timeout, setTimer(2,3000) will raise the timerFlag[2] to 1 after 3000ms (3 second)
- SEND_ADC: a state to send ADC value to terminal. It will then immediately switch to WAIT_OK.
- WAIT_OK: this state wait for command OK. If command OK is get, we stop the timer(2) and change state to WAIT_RTS to wait for the next command. In

this state, if it does not receive command OK after 3 second, we'll resend the old ADC value and set timer2() to fire a flag after 3 second again.

1.5 main

In main.c, we have the following code to control uart and adc:

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_ADC1_Init();
  MX_USART2_UART_Init();

  HAL_TIM_Base_Start_IT(&htim2);
  HAL_ADC_Start(&hadc1);

  HAL_UART_Receive_IT(&huart2, &temp, 1);
  initTimer(1);
  initTimer(0);
  while (1)
  {

    if (getTimerFlag(1) == 1) {
      HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
      setTimer(1, 1000);
    }
    if (buffer_flag == 1) {
      command_parser_fsm(buffer, index_buffer);
      buffer_flag = 0;
    }

    uart_communication_fsm();
  }

}

void HAL_TIM_PeriodElapsvoid HAL_TIM_PeriodElapsedCallback(
   TIM_HandleTypeDef *htim) {
  if (htim->Instance == TIM2) {
    timer_run(0);
    timer_run(1);
    timer_run(2);
  }
}
```