

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



ĐỒ ÁN MÔN HỌC THIẾT KẾ LUẬN LÝ

Smart Home Project
With Arduino

GVHD: Lê Trọng Nhân
SV thực hiện: Vũ Khánh Hưng – 1910232
Nguyễn Duy Khánh – 1913743

Tp. Hồ Chí Minh, Tháng 11/2021



Mục lục

1	Introduction	9
1.1	Giới thiệu về Arduino	9
1.1.1	Giới thiệu chung về Arduino	9
1.1.2	Dặc điểm kỹ thuật của Board Arduino Uno R3	9
1.1.3	Ưu và nhược điểm khi dùng Arduino	11
1.2	Giới thiệu về Proteus	11
1.2.1	Giới thiệu chung về Proteus	11
1.2.2	Những đặc điểm nổi bật của Proteus	12
1.2.3	Ưu và nhược điểm của Proteus	12
1.3	Thư viện Arduino trên Proteus	13
1.3.1	Dòng lực	13
1.3.2	Các loại Arduino hỗ trợ trong Proteus	13
1.3.3	Các bước thực hiện	13
1.4	Khởi tạo project trên Arduino	13
1.5	Ngắt thời gian (Timer interrupt)	16
1.5.1	Khái niệm về ngắt (Interrupt)	16
1.5.1.1	Ngắt phần cứng (Hardware interrupt)	16
1.5.1.2	Ngắt phần mềm (Software interrupt)	16
1.5.2	Timer ở mạch Arduino	17
1.5.3	Cài đặt	17
1.5.3.1	Nguyên lý	17
1.5.3.2	Thiết lập giá trị Prescaler và Compare match register	18
1.5.3.3	Thiết lập Timer	19
1.5.3.4	Vận dụng vào việc tạo timer interrupt <i>10ms</i>	20
1.6	Ứng dụng vào bài toán đọc nút nhấn	21
1.6.1	Giới thiệu	21
1.6.2	Một số kỹ thuật cho việc đọc từ chân cổng (port pin)	21
1.6.2.1	Sự cần thiết của điện trở kéo lên (Pull-up resistor)	21
1.6.2.2	Dối mặt với "nảy" nút bấm (switch bounce)	23
1.6.3	Cài đặt	24
1.6.3.1	Schematic của mạch điều khiển đèn bằng nút nhấn đơn giản	24
1.6.3.2	Cài đặt	25
2	Hệ thống giám sát không khí trong nhà (Indoor Air Monitoring)	26
2.1	Giới thiệu	26
2.2	Yêu cầu đặt ra	27
2.3	Giới thiệu về cảm biến đo nhiệt độ và độ ẩm DHT11	27
2.3.1	Giới thiệu sơ lược	27
2.3.2	Thông số kỹ thuật:	28
2.3.3	Chức năng các chân	28
2.3.4	Nguyên lý hoạt động	29
2.3.4.1	MCU gửi tín hiệu Start cho DHT11	29
2.3.4.2	MCU đọc giá trị trên DHT11	30
2.4	Giới thiệu về màn hình LED 16×2	31
2.4.1	Giới thiệu sơ lược	31
2.4.2	Thông số kỹ thuật	31
2.4.3	Chức năng các chân	32



2.4.4	Giới thiệu về DDRAM – Display Data RAM	32
2.4.5	Giới thiệu về CGROM – Character Generator ROM	33
2.4.6	Giới thiệu về CGRAM – Character Generator RAM	33
2.4.7	Các thanh ghi	33
2.4.8	Cờ báo bận (Busy flag - BF)	34
2.4.9	Tập lệnh (instruction) và bộ lệnh (command) của LCD	34
2.5	Giới thiệu mạch chuyển đổi giao tiếp I2C	37
2.5.1	Chuẩn giao tiếp I2C	37
2.5.1.1	Giới thiệu sơ lược	37
2.5.1.2	Đặc điểm giao tiếp I2C	37
2.5.1.3	Chế độ hoạt động (Tốc độ truyền)	38
2.5.1.4	Trình tự truyền bit trên đường truyền	38
2.5.2	Mạch chuyển đổi giao tiếp I2C	39
2.5.2.1	Giới thiệu sơ lược	39
2.5.2.2	Thông số kỹ thuật	40
2.5.2.3	Chức năng các chân	40
2.5.2.4	Sơ đồ khối (Block diagram)	41
2.5.2.5	Nguyên lý hoạt động	41
2.6	Kiến trúc hệ thống	42
2.7	Schematic của module trên proteus	42
2.8	Mô tả hệ thống	43
2.9	Hiện thực hệ thống	43
2.9.1	Đọc dữ liệu từ cảm biến	43
2.9.1.1	Thư viện:	43
2.9.1.2	Hiện thực module đọc dữ liệu từ cảm biến DHT11	44
2.9.2	Hiện thị thông tin nhiệt độ độ ẩm ra màn hình LCD	44
2.9.2.1	Thư viện	44
2.9.2.2	Hiện thực module LCD để hiển thị thông tin nhiệt độ độ ẩm ra màn hình	44
2.10	Demo	45
3	Hệ thống quạt làm mát dùng động cơ điện một chiều (Motor DC)	46
3.1	Giới thiệu	46
3.2	Yêu cầu đặt ra	47
3.3	Phương pháp điều khiển tốc độ động cơ (Motor)	47
3.3.1	Giới thiệu về PWM	47
3.3.2	Phương pháp điều chế PWM	47
3.4	Kiến trúc hệ thống	48
3.5	Schematic của module trên proteus	49
3.6	Mô tả hệ thống	49
3.7	Máy trạng thái	50
3.8	Hiện thực hệ thống	51
3.8.1	Hiện thực module FAN	51
3.8.2	Hiển thị thông tin trạng thái của quạt ra LCD	53
3.9	Demo	54



4 Hệ thống quản lý việc ra vào phòng bằng thẻ RFID (Radio Frequency Identification)	54
4.1 Giới thiệu	54
4.2 Yêu cầu đặt ra	54
4.3 Giới thiệu về Relay (Rờ le)	55
4.3.1 Giới thiệu sơ lược	55
4.3.2 Thông số kỹ thuật	55
4.3.3 Sơ đồ chân	56
4.3.4 Nguyên lý làm việc	56
4.4 Giới thiệu về RFID	56
4.4.1 Giới thiệu sơ lược về công nghệ RFID	56
4.4.2 Các thành phần chính của hệ thống RFID	57
4.4.2.1 Thẻ RFID	57
4.4.2.2 Các reader (đầu đọc) hoặc sensor (cảm biến) để truy vấn các thẻ.	59
4.4.2.3 Database	61
4.4.3 Phân loại RFID	61
4.4.4 Phương thức làm việc của RFID	62
4.5 Giới thiệu về chuẩn giao tiếp UART	63
4.5.1 Giới thiệu sơ lược	63
4.5.2 Các thông số cơ bản trong giao tiếp UART	64
4.6 Kiến trúc hệ thống	65
4.7 Schematic của module trên Proteus	65
4.8 Mô tả hệ thống	66
4.9 Máy trạng thái	67
4.10 Hiện thực hệ thống	68
4.10.1 Hiện thực module RFID	68
4.10.2 Hiển thị thông tin trạng thái của module RFID ra LCD	72
4.11 Demo	74
5 Thiết lập Cooperative scheduler cho hệ thống	74
5.1 Giới thiệu về Scheduler	74
5.1.1 Dánh giá về cấu trúc siêu vòng lặp (Super Loop)	74
5.1.2 Giới thiệu về Scheduler	76
5.2 Giới thiệu về Cooperative Scheduler	77
5.2.1 Giới thiệu sơ lược	77
5.2.2 Đặc điểm của Cooperative scheduler	77
5.3 Thiết lập Scheduler cho hệ thống	78
5.3.1 Giới thiệu sơ lược	78
5.3.2 Cài đặt	78
5.4 Triển khai	79
5.4.1 Ý tưởng	79
5.4.2 Flow chart	79
5.4.2.1 Hàm thêm tác vụ vào (SCH_Add_Task())	80
5.4.2.2 Hàm điều phối tác vụ (SCH_Dispatch_Tasks())	81
5.4.3 Hiện thực module scheduler	81



6 Gửi dữ liệu lên server IoT	85
6.1 Giới thiệu về IoT	85
6.1.1 Giới thiệu sơ lược	85
6.1.2 Kiến trúc 4 thành phần của ứng dụng IoT	86
6.1.3 Giới thiệu về Adafruit IO	86
6.2 Giới thiệu về giao thức MQTT (Message Queuing Telemetry Transport)	87
6.2.1 Giới thiệu sơ lược	87
6.2.2 Các khái niệm trong giao thức MQTT	87
6.2.3 Kiến trúc publish/subscribe	88
6.2.4 Cấu trúc giao thức MQTT	89
6.2.5 Cấu trúc gói tin MQTT	90
6.3 Thiết lập Dashboard thông qua server Adafruit IO	92
6.4 Hiện thực IoT Gateway bằng ngôn ngữ Python	93
6.4.1 Giao tiếp giữa gateway và hệ thống nhà thông minh	93
6.4.2 Giao tiếp giữa gateway và server Adafruit IO	93
6.4.3 Hiện thực Gateway IoT	94
6.5 Giao tiếp giữa gateway và Arduino	95
7 Một số module hỗ trợ cho các module chức năng chính của hệ thống	97
7.1 Module LED	97
7.2 Module Button	97
7.3 Module timer	101
8 Hoàn thiện hệ thống nhà thông minh	103
8.1 Schematic trên Proteus của toàn bộ hệ thống	103
8.2 File "common.h"	103
8.3 Chương trình chính (main)	104
9 Những khó khăn khi hiện thực project	106
10 Định hướng phát triển project trong tương lai	106
11 Tài liệu đính kèm của dự án	106
12 Kết luận	106
Tài liệu tham khảo	107



Danh sách hình

1	Hình ảnh về board Arduino Uno R3	9
2	Liệt kê các board của Arduino	13
3	Schematic của mạch chớp LED (Blink LED)	14
4	Giao diện kiểm tra biên dịch	15
5	Giao diện vùng biên dịch chương trình	15
6	Giao diện thay đổi Component	16
7	Kết nối button với một MCU	22
8	Sự cần thiết của pull-up resistor	22
9	Một cách tin cậy để kết nối 1 nút / công tắc với MCU	23
10	Hiện tượng switch bounce	23
11	Schematic của mạch điều khiển đèn bằng nút nhấn	25
12	Hình ảnh thực tế của cảm biến DHT11	27
13	Các chân của DHT11	29
14	Cách thức hoạt động	29
15	Nhận biết bit 0	30
16	Nhận biết bit 1	30
17	Hình ảnh thực tế của LED 16×2	31
18	Các chân của LED 16×2	32
19	Địa chỉ DDRAM với LCD 1 dòng	32
20	Địa chỉ DDRAM với LCD 2 dòng	33
21	Bản đồ mã ký tự LCD (LCD characters code map) cho 5×8 dot	33
22	Cách kết nối thiết bị khi dùng chung I2C bus	37
23	Truyền nhận dữ liệu giữa chủ/tớ (Master/Slave)	38
24	Trình tự truyền bit trên đường truyền	39
25	Hình ảnh thực tế của mạch I2C	40
26	Các chân của mạch chuyển đổi giao tiếp I2C	40
27	Sơ đồ khối của mạch chuyển đổi giao tiếp I2C	41
28	Sơ đồ kiến trúc hệ thống của module DHT11	42
29	Schematic của module Air Monitoring trên proteus	42
30	Bảng định địa chỉ LCD tương ứng với cách nối chân	43
31	Ứng dụng của motor một chiều (Motor DC)	46
32	Hình ảnh thực tế của Motor DC	47
33	Giản đồ thời gian xung PWM	48
34	Sơ đồ kiến trúc hệ thống của module FAN	48
35	Schematic của module FAN trên proteus	49
36	Sơ đồ máy trạng thái của hệ thống	50
37	Hình ảnh thực tế của Relay 5V	55
38	Các chân của Relay	56
39	Các thành phần của hệ thống RFID	57
40	Cấu tạo của thẻ RFID	57
41	Các thành phần của một Reader	60
42	Một số đầu đọc (reader) RFID phổ biến	61
43	Các dải tần số hoạt động của RFID	62
44	Hoạt động giữa tag và reader RFID	63
45	Ứng dụng của giao tiếp UART	64
46	Sơ đồ kiến trúc của hệ thống	65



47	Schematic của module RFID trên proteus	66
48	Sơ đồ máy trạng thái của module	67
49	Flow chart của hàm SCH_Add_Task()	80
50	Flow chart của hàm SCH_Dispatch_Tasks()	81
51	Kiến trúc 5 lớp của một ứng dụng Kết nối vạn vật (IOT)	85
52	Kiến trúc 4 thành phần trong ứng dụng IoT	86
53	Mô hình MQTT publish-subscribe cho việc truyền tín hiệu về nhiệt độ và độ ẩm	88
54	Cấu trúc gói tin MQTT	90
55	Dashboard tạo trên Adafruit IO	93
56	Hình ảnh cổng COM ảo trên Proteus	95
57	Hình ảnh thông số cấu hình của cổng COM ảo	96
58	Hình ảnh tạo kết nối ảo giữa COM1 và COM2	96
59	Máy trạng thái của nút nhấn có xử lý đè	98
60	Máy trạng thái của nút nhấn không có xử lý đè	98
61	Schematic của toàn bộ hệ thống	103



Danh sách bảng

1	Một số thông số của Arduino UNO R3	10
2	Mô tả Clock Select Bit cho timer 0 ở Arduino	18
3	Mô tả Clock Select Bit cho timer 1 ở Arduino	18
4	Mô tả Clock Select Bit cho timer 2 ở Arduino	19
5	Một số câu lệnh thiết lập Prescaler cho timer ở Arduino	19
6	Cách kích hoạt chế độ CTC ở timer trong Arduino	19
7	Cách kích hoạt timer ở Arduino	20
8	Ảnh hưởng của điểm sương đối với nhận thức của con người	27
9	Bảng khái quát chung các thông số của DHT11	28
10	Bảng chi tiết thông số về độ ẩm của DHT11	28
11	Bảng chi tiết thông số về nhiệt độ của DHT11	28
12	Chức năng các chân của DHT11	29
13	Đặc điểm về điện tử của LED 16×2	31
14	Đặc điểm về cơ khí của LED 16×2	31
15	Tên và chức năng của từng chân của LED 16×2	32
16	Chức năng chân RS và R/W theo mục đích sử dụng	34
17	Mã code của từng câu lệnh của LCD	35
18	Mô tả các tập lệnh của LCD	37
19	Thông số kỹ thuật của mạch I2C	40
20	Tên và chức năng của từng chân của mạch chuyển đổi I2C	41
21	Thông số kỹ thuật của Relay	55
22	Tên và chức năng của từng chân của Relay	56
23	Cấu trúc gói tin Connect do client gửi	91
24	Cấu trúc gói tin phản hồi Connect do server gửi	91
25	Cấu trúc gói tin Publish	92
26	Cấu trúc gói tin Subscribe	92



Danh sách đoạn chương trình

1	Chương trình chớp tắt đèn mỗi 500ms	14
2	Hàm tạo ngắt timer 10ms	20
3	Chương trình điều khiển đèn bằng nút bấm	25
4	Hiện thực module đọc dữ liệu từ cảm biến DHT11	44
5	Hiện thực module LCD để hiển thị thông tin nhiệt độ độ ẩm ra màn hình	44
6	Hiện thực module FAN	51
7	Tạo ra LCD thứ 2	53
8	Thêm biến hiển thị trạng thái của module FAN	53
9	Hàm khởi tạo LCD2	53
10	Triển khai hàm task_FAN_display()	53
11	Hiện thực module RFID	68
12	Tạo LCD thứ 3	72
13	Thêm biến để hiển thị theo trạng thái của module RFID	73
14	Hàm khởi tạo LCD3	73
15	Triển khai hàm task_RFID_display()	73
16	Cấu trúc siêu vòng lặp (Super Loop)	74
17	Cố gắng sử dụng kiến trúc siêu vòng lặp để thực thi các tác vụ có chu kỳ	75
18	Chạy 3 tác vụ định kỳ bằng cách sử dụng bộ lịch trình (scheduler)	76
19	Cấu trúc dữ liệu cho bộ lịch trình (scheduler)	78
20	Hàm interface của bộ lịch trình (Scheduler)	78
21	Hiện thực module của bộ lịch trình (Scheduler)	81
22	Hiện thực Gateway IoT bằng Python	94
23	Cấu hình data rate	96
24	Cấu hình chọn cổng Serial	96
25	Hiện thực module LED	97
26	Hiện thực module button	98
27	Hiện thực module timer	101
28	Hiện thực file common.h	103
29	Hiện thực file main	104

1 Introduction

1.1 Giới thiệu về Arduino

1.1.1 Giới thiệu chung về Arduino

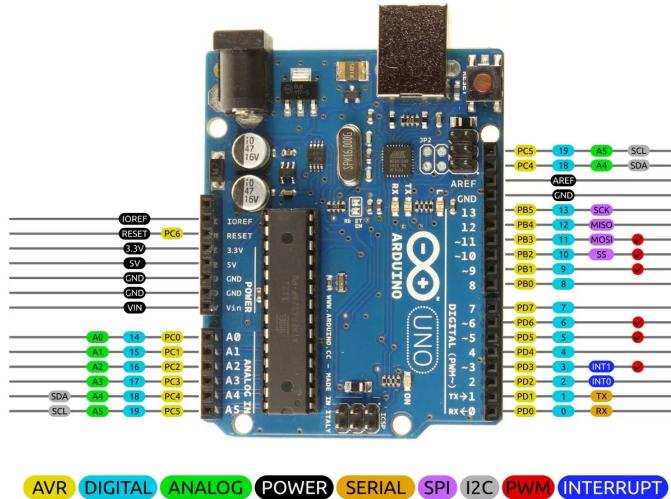
Arduino một nền tảng vi mạch thiết kế mở phần cứng (Open-source hardware) và phần mềm (Open-source software). Phần cứng Arduino là những bộ vi điều khiển bo mạch đơn (Single-board microcontroller) được khởi động vào năm 2005 như là một dự án dành cho sinh viên tại Interaction Design Institute Ivrea tại thị trấn Ivrea ở Ý, nhằm xây dựng các ứng dụng tương tác với nhau hoặc với môi trường được thuận lợi hơn.

Hiện nay Arduino được biết đến rất rộng rãi, từ học sinh trung học, đến sinh viên và người đi làm. Những dự án nhỏ và lớn được thực hiện một cách rất nhanh, các mã nguồn mở được chia sẻ nhiều trên diễn đàn trong nước và nước ngoài. Giúp ích rất nhiều cho những bạn theo đam mê nghiên cứu chế tạo những sản phẩm có ích cho xã hội. Trong những năm qua, Arduino là bộ não cho hàng ngàn dự án điện tử lớn nhỏ, từ những sản phẩm ra đời ứng dụng đơn giản trong cuộc sống đến những dự án khoa học phức tạp.

Hiện nay trên thị trường có rất nhiều phiên bản Arduino như Arduino Uno R3, Arduino Uno R3 CH340, Arduino Mega2560, Arduino Nano, Arduino Pro Mino, Arduino Lenadro, Arduino Industrial.... Trong dự án lần này nhóm sẽ sử dụng board Arduino uno R3 vì đây là loại Arduino phổ biến trên thị trường, giá thành hợp lý và đáp ứng đủ các yêu cầu của dự án.

1.1.2 Đặc điểm kỹ thuật của Board Arduino Uno R3

Arduino Uno R3 Pinout



2014 by Bouni
Photo by Arduino.cc

Hình 1: Hình ảnh về board Arduino Uno R3



Một vài thông số của Arduino UNO R3:

Vì điều khiển	ATmega328 họ 8 bit
Điện áp hoạt động	5V DC (chỉ được cấp qua cổng USB)
Tần số hoạt động	16MHz
Dòng tiêu thụ	Khoảng 30mA
Điện áp vào khuyên dùng	7-12V DC
Điện áp vào giới hạn	6-20V DC
Số chân Digital I/O	14 (6 chân hardware PWM)
Số chân Analog	6 (độ phân giải 10bit)
Dòng tối đa trên mỗi chân I/O	30 mA
Dòng ra tối đa (5V)	500 mA
Dòng ra tối đa (3.3V)	50 mA
Bộ nhớ flash	32 KB (ATmega328) với 0.5KB dùng bởi bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)

Bảng 1: Một số thông số của Arduino UNO R3

Các chân năng lượng:

- GND (Ground):** Cực âm của nguồn điện cấp cho Arduino UNO.
- 5V:** Cấp điện áp 5V đầu ra. Dòng tối đa cho phép ở chân này là 500mA.
- 3.3V:** Cấp điện áp 3.3V đầu ra. Dòng tối đa cho phép ở chân này là 50mA.
- IOREF:** Điện áp hoạt động của vi điều khiển trên Arduino UNO có thể được đo ở chân này. Và dĩ nhiên nó luôn là 5V. Mặc dù vậy ta không được lấy nguồn 5V từ chân này để sử dụng bởi chức năng của nó không phải là cấp nguồn.
- RESET:** Việc nhấn nút Reset trên board để reset vi điều khiển tương đương với việc chân RESET được nối với GND qua 1 điện trở **10KΩ**.

Các cổng vào/ra:

- 2 chân Serial:** 0 (RX) và 1 (TX): dùng để gửi (transmit – TX) và nhận (receive – RX) dữ liệu TTL Serial. Arduino Uno có thể giao tiếp với thiết bị khác thông qua 2 chân này.
- Chân PWM (~):** 3, 5, 6, 9, 10, và 11: cho phép bạn xuất ra xung PWM với độ phân giải 10 bit (giá trị từ **0 → 2¹⁰ – 1** tương ứng với **0 → 5V**) bằng hàm **analogWrite()**. Nói một cách đơn giản, bạn có thể điều chỉnh được điện áp ra ở chân này từ mức **0V** đến **5V** thay vì chỉ cố định ở mức **0V** và **5V** như những chân khác.
- Chân giao tiếp SPI:** 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ngoài các chức năng thông thường, 4 chân này còn dùng để truyền phát dữ liệu bằng giao thức SPI với các thiết bị khác.
- LED 13:** trên Arduino UNO có 1 đèn led màu cam (kí hiệu chữ L). Khi bấm nút Reset, bạn sẽ thấy đèn này nhấp nháy để báo hiệu. Nó được nối với chân số 13. Khi chân này được người dùng sử dụng, LED sẽ sáng.



- **AREF:** Chân này được gọi là tham chiếu tương tự, được sử dụng để cung cấp điện áp tham chiếu cho các đầu vào tương tự.
- **Giao tiếp I2C:** 2 chân A4 (SDA) và A5 (SCL) hỗ trợ giao tiếp I2C/TWI với các thiết bị khác.
- **USB:** Hỗ trợ điện áp khoảng 5V trong khi Vin và Power Jack hỗ trợ dải điện áp trong khoảng từ 7V đến 20V.
- **External Interrupts (Ngắt ngoài):** Chân 2 và 3 được sử dụng để cung cấp các ngắt ngoài.

1.1.3 Ưu và nhược điểm khi dùng Arduino

1. Ưu điểm:

- Cộng đồng hỗ trợ đông đảo, Khi có vấn đề xảy ra thì chúng ta dễ dàng có được câu trả lời.
- Hỗ trợ đa dạng các loại sensor, thiết bị phụ trợ, với các thư viện mã nguồn mở đầy đủ.
- Giá thành của board Arduino Uno R3 rẻ, phù hợp cho học sinh sinh viên.
- Dễ sử dụng đối với người mới bắt đầu, có hỗ trợ IDE, với ngôn ngữ C/C++.

2. Nhược điểm:

- Sức mạnh xử lí của nó không quá mạnh mẽ.
- Chỉ có thể chạy một Sketch(App) tại một thời điểm.
- Không có các cơ chế kiểm tra bộ nhớ, để đảm bảo an toàn.
- Chi phí cho CPU và memory so với các vi điều khiển khác thì tôn kém hơn.

1.2 Giới thiệu về Proteus

1.2.1 Giới thiệu chung về Proteus

Phần mềm vẽ Proteus là phần mềm vẽ mạch điện tử được phát triển bởi công ty Lancenter Electronics. Phần mềm có thể mô tả hầu hết các Linh Kiện Điện Tử thông dụng hiện nay, đặc biệt hỗ trợ cho cả các phần mềm như 8051, PIC, Motorola, AVR.

Proteus có khả năng mô phỏng hoạt động của các mạch điện tử bao gồm phần thiết kế mạch và viết trình điều khiển cho các loại vi điều khiển như MCS-51, AVR, PIC, ...

Có 2 chương trình trong phần mềm đó là ARES dùng trong vẽ mạch in và ISIS sử dụng cho mô phỏng mạch. Trong 2 chương trình này thì ISIS có phần nổi bật hơn so với ARES. ISIS đã được phát triển trong 12 năm và có tới hơn 12000 người dùng trên khắp thế giới (chắc chắn con số hiện tại đã tăng hơn rất nhiều). Điểm nổi bật của chúng đó là khả năng mô phỏng hoạt động của các vi điều khiển mà không cần dùng thêm bất kỳ một phần mềm phụ trợ nào khác. Từ phần mềm ISIS có thể dễ dàng chuyển sang ARES hoặc bất kỳ phần mềm vẽ mạch in khác.

Hình ảnh mạch điện được tạo bởi ISIS rất đẹp và dễ nhìn, chúng cho phép ta tùy chọn các đường nét, các màu sắc mạch điện hoặc các thiết kế theo các templates. Ngoài ra phần mềm mô phỏng mạch của Proteus có khả năng sắp xếp các đường mạch và vẽ điểm giao mạch tự động.



1.2.2 Những đặc điểm nổi bật của Proteus

- Có khả năng mô phỏng hầu hết trình điều khiển cho vi điều khiển.
- Chọn đối tượng và thiết lập thông số cho đối tượng dễ dàng.
- Xuất ra file Netlist tương thích với các chương trình làm mạch in thông dụng.
- Xuất file thống kê linh kiện cho mạch.
- ISIS tích hợp nhiều công cụ giúp cho việc quản lý mạch điện lớn, mạch điện có thể lên đến hàng ngàn linh kiện phục vụ cho thiết kế mạch chuyên nghiệp.
- Thiết kế theo cấu trúc (hierarchical design).
- Khả năng tự động đánh số linh kiện.

1.2.3 Ưu và nhược điểm của Proteus

1. Ưu điểm:

- Dễ dàng tạo ra một sơ đồ nguyên lý từ đơn giản đến các mạch có bộ lập trình vi xử lý.
- Dễ dàng chỉnh sửa các đặc tính của linh kiện trên sơ đồ nguyên lý.
- Hỗ trợ kiểm tra lỗi thiết kế trên sơ đồ nguyên lý. Có thể xem và lưu lại phần báo lỗi.
- Phần mềm chạy mô phỏng và phân tích các tính chất của một mạch điện một cách chính xác.
- Proteus cung cấp cho người sử dụng công cụ biên dịch cho các họ vi xử lý như MSC51, AVR, HC11, ... qua đó tạo ra các tập tin .hex dùng để nạp cho vi xử lý và tập tin .dsi dùng để xem và chạy kiểm tra từng bước trong quá trình mô phỏng.
- Phần mềm cung cấp rất nhiều mô hình linh kiện có chức năng mô phỏng, từ các vi điều khiển thông dụng đến các link kiện ngoại vi như LED, LCD, Keypad, cổng RS232... cho phép người sử dụng mô phỏng từ một hệ vi điều khiển hoàn chỉnh đến việc xây dựng phần mềm cho hệ thống đáp ứng các giao thức vật lý. Ngoài ra, Proteus còn cho phép bạn tự tạo link kiện tương tác động do đó bạn có thể thực hiện các mô phỏng có tương tác giống như hoạt động của một mạch thật.

2. Nhược điểm:

- Phần mềm do của công ty nước ngoài nên tính chất bản quyền khá cao và hầu như ít được biết nên rất khó kiểm ngoài thực tế.
- Trong khi thiết kế có nhiều phần trong Proteus chạy không theo một quy tắc nào làm người sử dụng đôi lúc gặp khó khăn.
- Sử dụng khá phức tạp nhất là đối với các mạch vi xử lý hay mạch cần chỉnh sửa tính chất của các sự kiện (do quá nhiều tính chất phải điều chỉnh).
- Hướng dẫn sử dụng trong Proteus hoàn toàn bằng Tiếng Anh nên đòi hỏi người sử dụng cũng phải có một nền tảng tiếng Anh cơ bản nếu muốn sử dụng nó một cách hiệu quả (nhất là tiếng Anh chuyên ngành về điện tử).



1.3 Thư viện Arduino trên Proteus

1.3.1 Động lực

Ta đã biết Arduino được sử dụng rất rộng rãi từ các dự án đơn giản cho đến các dự án phức tạp. Tuy nhiên chương trình Proteus không hỗ trợ sẵn mạch Arduino, do đó cần tải thêm thư viện Arduino cho Proteus.

1.3.2 Các loại Arduino hỗ trợ trong Proteus

Thư viện Arduino cho Proteus hỗ trợ các loại board sau:

- Arduino UNO.
- Arduino Mega 2560.
- Arduino Mega 1280.
- Arduino Nano.
- Arduino Mini.
- Arduino Pro Mini.

1.3.3 Các bước thực hiện

Để chạy mô phỏng Arduino trên Proteus, ta có thể lên trang web sau để tải thư viện Arduino:
<https://www.theengineeringprojects.com/2015/12/arduino-library-proteus-simulation.html>

Khi tải về sẽ có 2 file là ArduinoTEP.LIB và ArduinoTEP.IDX. Sau đó ta copy 2 file đó vào trong thư mục **LIBRARY** tương ứng của thư mục quản lý của **Proteus**.

Restart lại phần mềm **Proteus** và trong mục **Component**, ta tìm kiếm **ArduinoTEP** ta sẽ ra được hàng loạt các board Arduino được hỗ trợ như vừa liệt kê ở trên:

Results (6):		
Device	Library	Description
ARDUINO MEGA 2560	ArduinoTEP	Arduino MEGA 2560
ARDUINO MEGA1280	ArduinoTEP	Arduino MEGA (ATmega1280)
ARDUINO MINI	ArduinoTEP	Arduino Mini
ARDUINO NANO	ArduinoTEP	Arduino Nano
ARDUINO PRO MINI	ArduinoTEP	Arduino Pro Mini
ARDUINO UNO	ArduinoTEP	Arduino UNO R3 V1.0

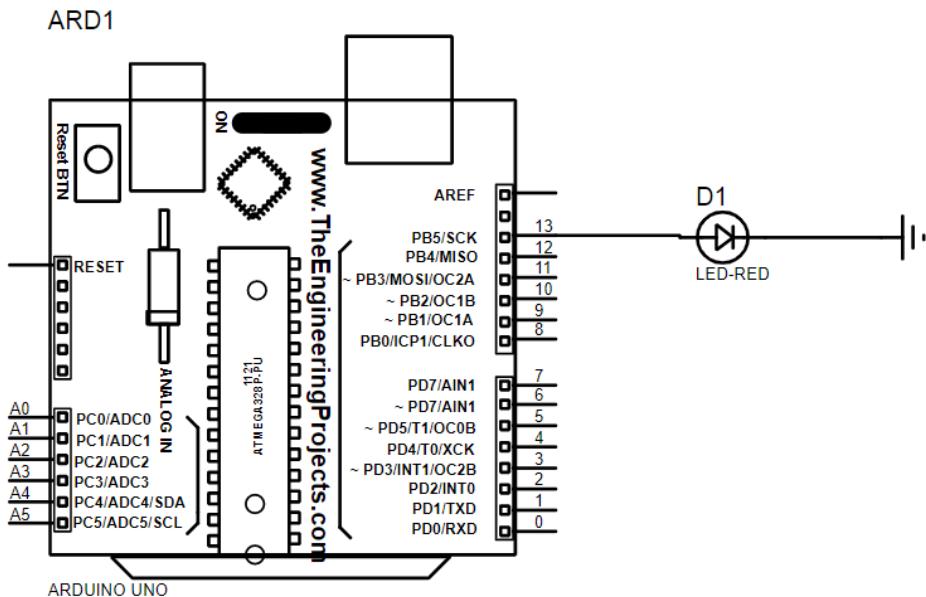
Hình 2: Liệt kê các board của Arduino

1.4 Khởi tạo project trên Arduino

Khi ta đã cài đặt được thư viện Arduino để sử dụng trong Proteus, ta có thể bắt đầu một project đầu tiên liên quan tới Arduino. Sau đây ta sẽ làm một dự án đầu tiên đó là chớp LED với thời gian luân chuyển (switch time) là **0.5s**.



Dầu tiên ta khởi động Proteus. Tạo một project mới. Sau đó bấm vào **Components** chọn **ARDUINO UNO** như **Hình 2** bên trên. Rồi ta xây dựng mạch như sau:



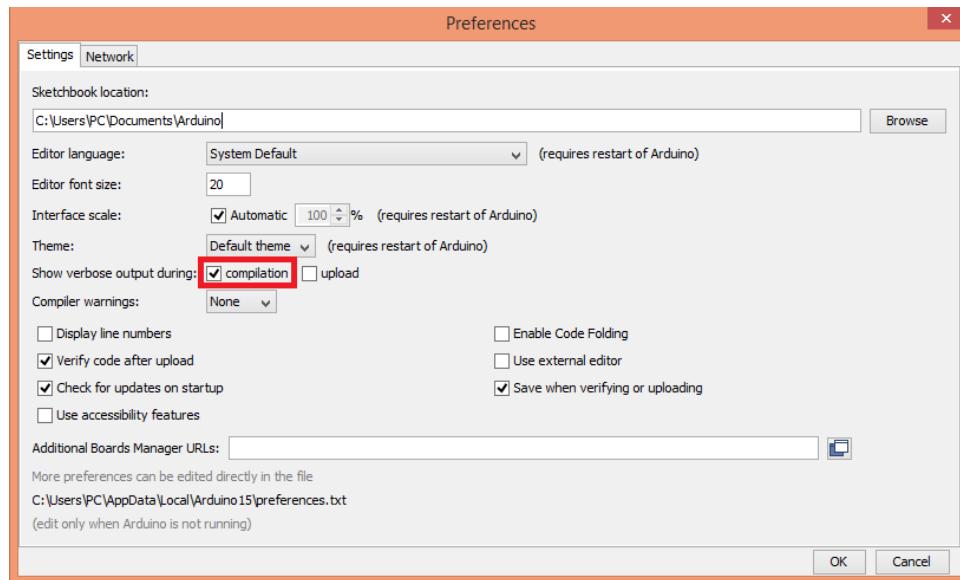
Hình 3: Schematic của mạch chớp LED (Blink LED)

Sau đó ta khởi động phần mềm Arduino và nhập đoạn chương trình sau:

```
1 void setup() {
2     // put your setup code here, to run once:
3     pinMode(13, OUTPUT);
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8     digitalWrite(13, HIGH);    // turn the LED on
9     delay(500);                // wait for half a second
10    digitalWrite(13, LOW);     // turn the LED off
11    delay(500);                // wait for half a second
12 }
```

Code 1: Chương trình chớp tắt đèn mỗi 500ms

Sau đó ta bấm vào **File → Reference** để kiểm tra xem tùy chọn biên dịch (compilation option) có sinh ra (generate) file .hex hay không. Nếu không thì ta tick vào phương án **compilation** như dưới đây:



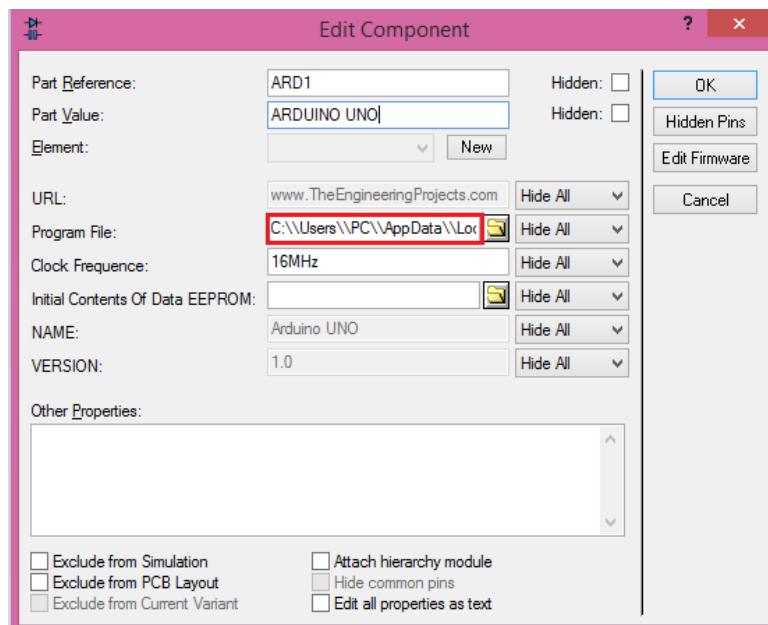
Hình 4: Giao diện kiểm tra biên dịch

Sau đó ta biên dịch chương trình và copy địa chỉ file .hex như bên dưới:

```
Done compiling
"E:\arduino\hardware\tools\avr\bin\avr-gcc-ar" rcs "C:\Users\PC\AppData\Local\Temp\arduino_build_772294\core\core.a"
Archiving built core (caching) in: C:\Users\PC\AppData\Local\Temp\arduino_cache_864363\core\core_arduino_avr_uno_09a5f148a1358c27e
Linking everything together...
"E:\arduino\hardware\tools\avr\bin\avr-gcc" -w -Os -g -flto -fuse-linker-plugin -Wl,--gc-sections -mmcu=atmega328p -o "C:\Users\PC\Blink.ino.elf"
"E:\arduino\hardware\tools\avr\bin\avr-objcopy" -O ihex -j .eprom --set-section-flags=.eprom=alloc,load --no-change-warnings
"E:\arduino\hardware\tools\avr\bin\avr-objcopy" -O hex -R .eprom "C:\Users\PC\AppData\Local\Temp\arduino_build_772294/Blink.elf"
"E:\arduino\hardware\tools\avr\bin\avr-size" -A "C:\Users\PC\AppData\Local\Temp\arduino_build_772294/Blink.elf"
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

Hình 5: Giao diện vùng biên dịch chương trình

Quay trở lại Proteus, bấm đúp chuột (double click) vào Arduino và copy địa chỉ file trên vào trong chõ **Program file** của khung **Edit Component** như sau đây:



Hình 6: Giao diện thay đổi Component

Sau đó bấm nút **Run**, ta đã thực hiện thành công dự án đầu tiên với đèn LED chớp mỗi **0.5s**.

1.5 Ngắt thời gian (Timer interrupt)

1.5.1 Khái niệm về ngắt (Interrupt)

Ngắt (Interrupt) là một tín hiệu (signal) đến bộ xử lý (processor) do phần cứng (hardware) hoặc phần mềm (software) tạo ra một sự kiện cần được thực hiện ngay lập tức. Bất cứ khi nào một ngắt xảy ra, bộ điều khiển (controller) sẽ hoàn thành việc thực hiện lệnh hiện tại và bắt đầu thực hiện một Quy trình Dịch vụ Ngắt (**Interrupt Service Route - ISR**) hoặc Trình xử lý Ngắt (**Interrupt Handler**). ISR cho bộ xử lý (processor) hoặc bộ điều khiển (controller) biết phải làm gì khi xảy ra ngắt. Các ngắt có thể là ngắt phần cứng (software interrupt) hoặc ngắt phần mềm (software interrupt).

1.5.1.1 Ngắt phần cứng (Hardware interrupt)

Ngắt phần cứng (Hardware interrupt) là một tín hiệu cảnh báo điện tử được gửi đến bộ xử lý (processor) từ một thiết bị bên ngoài, như bộ điều khiển dĩa hoặc thiết bị ngoại vi bên ngoài.

Ví dụ: Khi ta nhấn một phím trên bàn phím hoặc di chuyển chuột, chúng sẽ kích hoạt các ngắt phần cứng khiến bộ xử lý đọc vị trí tổ hợp phím hoặc chuột.

1.5.1.2 Ngắt phần mềm (Software interrupt)

Ngắt phần mềm (Software interrupt) là do một điều kiện ngoại lệ (exception) hoặc một lệnh đặc biệt trong tập lệnh gây ra ngắt khi nó được thực thi bởi bộ xử lý (processor).



Ví dụ: Nếu mạch logic số học của bộ xử lý chạy lệnh chia một số cho 0, để gây ra ngoại lệ chia cho 0, do đó khiến máy tính bỏ phép tính hoặc hiển thị thông báo lỗi. Các lệnh ngắt phần mềm hoạt động tương tự như các lệnh gọi chương trình con (subroutine calls).

1.5.2 Timer ở mạch Arduino

Bộ định thời (Timer) là một phần cấu tạo phần cứng của bộ điều khiển Arduino (các bộ điều khiển (controllers) khác cũng có phần cứng bộ đếm thời gian). Nó giống như một chiếc đồng hồ (clock), và có thể được sử dụng để đo các sự kiện thời gian.

Bộ định thời (Timer) có thể được lập trình bởi một số thanh ghi đặc biệt (special registers). Ta có thể cấu hình prescaler, chế độ hoạt động (mode operation) và một số thứ khác.

Bộ điều khiển (Controller) của **Arduino** là **Atmel AVR ATmega168** hoặc **ATmega328**. Các chip này tương thích với nhau và chỉ khác nhau về kích thước của bộ nhớ trong. Cả hai đều có 3 bộ timer, được gọi là **timer0**, **timer1** và **timer2**. **timer0** và **timer2** là bộ định thời 8 bits, trong đó **timer1** là bộ định thời 16 bits. Sự khác biệt quan trọng nhất giữa bộ định thời (timer) 8 bits và 16 bits là độ phân giải bộ định thời (timer resolution). 8 bits có nghĩa là $2^8 = 256$ giá trị trong đó 16 bits có nghĩa là $2^{16} = 65536$ giá trị cho độ phân giải cao hơn.

Bộ điều khiển (Controller) cho dòng **Arduino Mega** là **Atmel AVR ATmega1280** hoặc **ATmega2560**. Cũng giống hệt nhau chỉ khác nhau về kích thước bộ nhớ. Các bộ điều khiển này có 6 bộ định thời. **timer0**, **timer1** và **timer2** giống với **ATmega168/328**. **timer3**, **timer4** và **timer5** đều là các bộ định thời 16 bits, tương tự như **timer1**.

Tất cả các bộ định thời (timer) phụ thuộc vào đồng hồ hệ thống (system clock) của hệ thống Arduino. Thông thường tần số của hệ thống (system clock) là **16MHz**, nhưng đối với **Arduino Pro 3,3V** thì là **8MHz**.

Phần cứng bộ định thời (timer hardware) có thể được cấu hình với một số thanh ghi bộ định thời đặc biệt (special timer register). Trong phần Arduino firmware, tất cả các bộ định thời (timer) đã được định cấu hình ở tần số **1KHz** và các ngắt (interrupts) thường được kích hoạt.

1.5.3 Cài đặt

1.5.3.1 Nguyên lý

Ta đã biết trong **Arduino Uno** có ba bộ timer gọi là **timer0**, **timer1** và **timer2**. Mỗi bộ đếm thời gian có một bộ đếm được tăng lên sau mỗi lần tick của clock của bộ định thời (timer's clock). Các ngắt bộ định thời (timer interrupt) CTC (Clear Timer on Compare) được kích hoạt khi bộ đếm đạt đến một giá trị xác định, được lưu trữ trong thanh ghi đối chiếu so sánh (compare match register). Khi bộ đếm định thời (timer counter) đạt đến giá trị này, nó sẽ xóa (đặt lại về 0) vào lần đánh dấu tiếp theo của đồng hồ hẹn giờ, sau đó nó sẽ tiếp tục đếm đến giá trị đối chiếu so sánh (compare match value) một lần nữa. Bằng cách chọn giá trị đối chiếu so sánh (compare match value) và cài đặt tốc độ (speed) bộ định thời tăng bộ đếm, ta có thể kiểm soát tần suất ngắt bộ định thời (timer interrupt).

Ta có công thức tính tần số ngắt (timer interrupt) với công thức sau:

$$\text{Interrupt frequency (Hz)} = \frac{\text{Arduino clock speed}}{\text{Prescaler} \times (\text{compare match register} + 1)}$$

Lý do ta có **+1** bởi vì thanh ghi đối chiếu so sánh (compare match register) bắt đầu bởi **0**, điều này dẫn đến việc khởi tạo một thanh ghi định thời (timer interrupt) là **TCNTx = 0** với **x** là loại timer cần sử dụng. Với **Arduino Uno**, ta có **Arduino clock speed = 16MHz**, từ đây suy ra được:



$$\text{Interrupt frequency (Hz)} = \frac{16\text{MHz}}{\text{Prescaler} \times (\text{compare match register} + 1)} \quad (1)$$

Điều này tương đương với:

$$\text{compare match register} = \frac{16 \times 10^6}{\text{Prescaler} \times \text{Interrupt frequency}} - 1$$

Giá trị đối chiếu so sánh (compare match register) sẽ quyết định đến việc ta cần xài bộ định thời nào để tạo (`timer0`, `timer1` hay `timer2`?).

Tuy nhiên giá trị đối chiếu so sánh (compare match register) phụ thuộc vào giá trị **Prescaler** vì giá trị **Interrupt frequency** là hằng số - là giá trị mình mong muốn. Sau đây là cách thiết lập giá trị **Prescaler**.

1.5.3.2 Thiết lập giá trị Prescaler và Compare match register

Để thiết lập giá trị Prescaler, ta cần xác định thông qua các **Chip Select (CS)**. Cụ thể được thể hiện qua bảng sau:

Timer 0

CS02	CS01	CS00	Description
0	0	0	Không có clock (Bộ định thời / đếm dừng lại)
0	0	1	$clk_{I/O}/1$ (Không prescaling)
0	1	0	$clk_{I/O}/8$ (Từ prescaler)
0	1	1	$clk_{I/O}/64$ (Từ prescaler)
1	0	0	$clk_{I/O}/256$ (Từ prescaler)
1	0	1	$clk_{I/O}/1024$ (Từ prescaler)

Bảng 2: Mô tả Clock Select Bit cho timer 0 ở Arduino

Timer 1

CS12	CS11	CS10	Description
0	0	0	Không có clock (Bộ định thời / đếm dừng lại)
0	0	1	$clk_{I/O}/1$ (Không prescaling)
0	1	0	$clk_{I/O}/8$ (Từ prescaler)
0	1	1	$clk_{I/O}/64$ (Từ prescaler)
1	0	0	$clk_{I/O}/256$ (Từ prescaler)
1	0	1	$clk_{I/O}/1024$ (Từ prescaler)

Bảng 3: Mô tả Clock Select Bit cho timer 1 ở Arduino

Timer 2



CS22	CS21	CS20	Description
0	0	0	Không có clock (Bộ định thời / đếm dừng lại)
0	0	1	$clk_{I/O}/1$ (Không prescaling)
0	1	0	$clk_{I/O}/8$ (Từ prescaler)
0	1	1	$clk_{I/O}/32$ (Từ prescaler)
1	0	0	$clk_{I/O}/64$ (Từ prescaler)

Bảng 4: Mô tả Clock Select Bit cho timer 2 ở Arduino

Điều này rõ ràng ta thấy được là các giá trị prescaler có sẵn chỉ có **1, 8, 64, 256, 1024**. Để quy về prescaler mong muốn ta sẽ sử dụng thông qua một bộ ghi định thời (timer register) gọi là **TCCR_x** với **x** là timer cần được sử dụng. Cách sử dụng chúng được thể hiện qua 3 ví dụ sau:

Câu lệnh	Ý nghĩa
<code>TCCR2B = (1 << CS22)</code>	Kích hoạt bit CS2 cho ta timer 2 có prescaler = 64
<code>TCCR1B = (1 << CS11)</code>	Kích hoạt bit CS1 cho ta timer 1 có prescaler = 8
<code>TCCROB = (1 << CS02) (1 << CS00)</code>	Kích hoạt bit CS0 và CS2 cho ta timer 0 có prescaler = 1024

Bảng 5: Một số câu lệnh thiết lập Prescaler cho timer ở Arduino

Khi ta đã biết giá trị **Prescaler**, ta có thể biết được giá trị **Compare match register** là bao nhiêu và giá trị đó sẽ được ghi nhận thông qua thanh ghi bộ định thời (timer register) **OCR_x** với **x** là loại timer cần sử dụng.

1.5.3.3 Thiết lập Timer

Ta quan sát thì thấy rằng giữa 3 timer ở **Arduino Uno** có sự phân biệt về tên **Chip Select (CS)** và độ **prescaler** tương ứng. Do đó cần phải cho hệ thống biết rằng ta đang muốn xài timer nào trong 3 timer **timer0**, **timer1** và **timer2**. Để kích hoạt chế độ CTC tương ứng ta sử dụng các câu lệnh như sau đây:

Câu lệnh	Ý nghĩa
<code>TCCR0A = (1 << WGM01)</code>	Kích hoạt chế độ CTC ứng với timer0
<code>TCCR1B = (1 << WGM12)</code>	Kích hoạt chế độ CTC ứng với timer1
<code>TCCR2A = (1 << WGM21)</code>	Kích hoạt chế độ CTC ứng với timer2

Bảng 6: Cách kích hoạt chế độ CTC ở timer trong Arduino

Bên cạnh đó ta cần phải kích hoạt timer trước khi sử dụng. Cụ thể ta sẽ sử dụng tiếp hai thanh ghi định thời (timer register) tên là **TIMSK_x** và **OCIE_xA** với **x** là loại timer cần sử dụng. Cụ thể cú pháp và ý nghĩa các câu lệnh được thể hiện như sau:



Câu lệnh	Ý nghĩa
<code>TIMSK0 = (1 << OCIE0A)</code>	Kích hoạt timer0
<code>TIMSK1 = (1 << OCIE1A)</code>	Kích hoạt timer1
<code>TIMSK2 = (1 << OCIE2A)</code>	Kích hoạt timer2

Bảng 7: Cách kích hoạt timer ở Arduino

Ngoài ra ta còn một số hàm điều khiển việc ngắn, chẳng hạn `cli()` để dừng việc ngắn và `sei()` để tiếp tục việc ngắn.

1.5.3.4 Vận dụng vào việc tạo timer interrupt 10ms

Ta muốn tạo timer interrupt **10ms** bắt buộc ta phải tạo tần số ngắn là:

$$f = \frac{1}{T} = \frac{1}{10 \times 10^{-3}} = 100Hz$$

Mặt khác theo công thức (1) ta có được:

$$100 = \frac{16 \times 10^6}{\text{Prescaler} \times (\text{compare match register} + 1)}$$

Điều này suy ra được:

$$\text{Prescaler} \times (\text{compare match register} + 1) = 160000$$

Ta sẽ chọn phương án **prescaler = 256** và **Compare match register = 624**. Điều này kèm theo việc cần xài **timer1** làm bộ timer cần sử dụng dẫn đến việc ta cần phải kích hoạt **Chip Select (CS)** là **CS12**. Sau đây là đoạn chương trình thể hiện việc thiết lập **timer1** thỏa mãn điều trên:

```

1 void init_timer_interrupt(){
2     cli(); //stop interrupts
3     TCCR1A = 0; // set entire TCCR1A register to 0
4     TCCR1B = 0; // same for TCCR1B
5     TCNT1 = 0; //initialize counter value to 0
6     /*
7      We have initial frequency is 16 * 10^6 Hz
8      We need time = 10ms --> Frequency = 100Hz
9      We use prescaler = 256 and counter = 625.
10     Since counter begin at 0-based index so OCR1A = 624
11     Prove:
12         We have Freq_out = Freq_init / (prescaler * (counter + 1)) ←
13             = (16 * 10^6) / (256 * (624 + 1)) = 100Hz
14     */
15     OCR1A = 624; // = (16 * 10^6) / (100 * 256) - 1 (must be ←
16             <65536)
17     // turn on CTC mode
18     TCCR1B |= (1 << WGM12);
19     // Set CS12 bits for 256 prescaler
20     TCCR1B |= 1 << CS12;
21     // enable timer compare interrupt

```



```
20    TIMSK1 |= (1 << OCIE1A);  
21    sei(); //allow interrupts  
22 }
```

Code 2: Hàm tạo ngắt timer 10ms

1.6 Ứng dụng vào bài toán đọc nút nhấn

1.6.1 Giới thiệu

Hệ thống nhúng thường sử dụng các nút (hoặc phím, hoặc công tắc, hoặc bất kỳ dạng tiếp điểm cơ học nào) như một phần của giao diện người dùng của chúng. Quy tắc chung này được áp dụng từ hệ thống điều khiển từ xa cơ bản nhất để mở cửa nhà để xe, cho đến hệ thống lái tự động máy bay phức tạp nhất. Bất kể hệ thống bạn tạo là gì, bạn cần có khả năng tạo giao diện nút đáng tin cậy. Một nút thường được kết nối với MCU để tạo ra một mức logic nhất định khi được đẩy hoặc đóng hoặc "hoạt động" và mức logic ngược lại khi được bỏ đẩy hoặc mở hoặc "không hoạt động". Mức logic hoạt động có thể là 0 hoặc 1, nhưng vì lý do cả về lịch sử và điện, mức hoạt động là 0 phổ biến hơn. Chúng ta có thể sử dụng một nút nếu ta muốn thực hiện các thao tác như:

- Lái động cơ trong khi nhấn công tắc.
- Bật đèn khi nhấn công tắc.
- Kích hoạt máy bơm trong khi nhấn công tắc.

Các hoạt động này có thể được thực hiện bằng nút điện mà không cần sử dụng MCU. Tuy nhiên, việc sử dụng MCU cũng có thể phù hợp nếu ta yêu cầu các hành vi phức tạp hơn. Ví dụ:

- Lái động cơ trong khi nhấn công tắc.

Điều kiện: Nếu không có bộ phận bảo vệ an toàn, dừng quay động cơ. Thay vào đó, hãy phát ra tiếng còi trong 2 giây.

- Bật đèn khi nhấn công tắc.

Điều kiện: Để tiết kiệm điện năng, hãy bỏ qua yêu cầu bật đèn vào ban ngày.

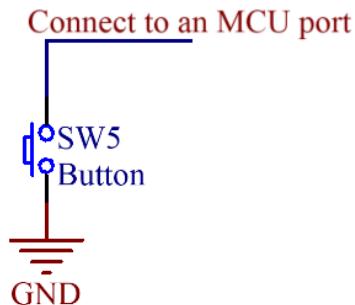
- Kích hoạt máy bơm trong khi nhấn công tắc.

Điều kiện: Nếu bể chữa nước chính dưới 300 lít, không khởi động máy bơm chính: thay vào đó, khởi động máy bơm dự trữ và rút nước từ bể chữa khẩn cấp.

Chính vì điều này nên việc xử lý nút nhấn trở nên vô cùng quan trọng.

1.6.2 Một số kỹ thuật cho việc đọc từ chân cổng (port pin)

1.6.2.1 Sự cần thiết của điện trở kéo lên (Pull-up resistor)



Hình 7: Kết nối button với một MCU

Hình 7 mô tả cách kết nối một nút (button) với MCU. Phần cứng này hoạt động như sau:

- Khi công tắc mở, nó không có tác động đến chân cổng. Một điện trở bên trong (internal resistor) trên cổng (port) "kéo" chân đến điện áp cung cấp của MCU. Nếu chúng ta đọc chân, chúng ta sẽ thấy giá trị 1.
- Khi nút nhấn đóng (nhấn), điện áp chân sẽ là 0V. Nếu chúng ta đọc mã pin, chúng ta sẽ thấy giá trị 0.

Tuy nhiên, nếu MCU không có điện trở kéo lên (pull-up resistor) bên trong, khi nhấn nút, giá trị đọc sẽ là 0, nhưng ngay cả khi chúng ta thả nút, giá trị đọc vẫn là 0 như **Hình 8** dưới đây:

With pull-ups:

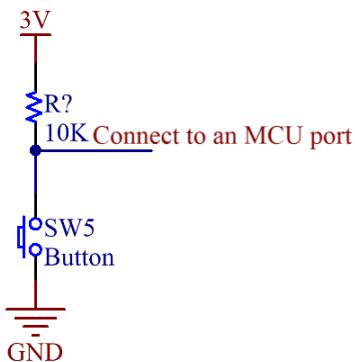


Without pull-ups:



Hình 8: Sự cần thiết của pull-up resistor

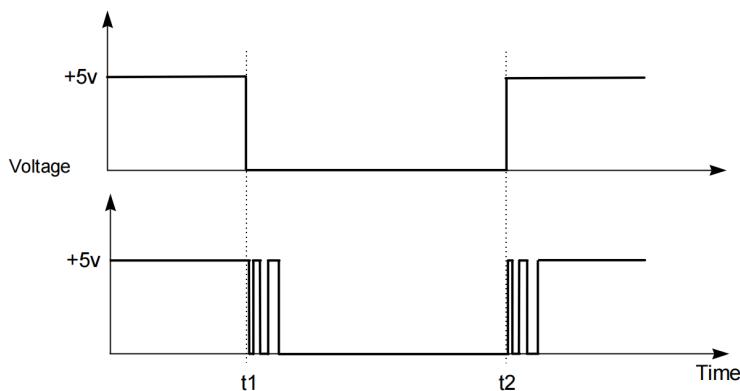
Vì vậy, một cách đáng tin cậy (reliable way) để kết nối một nút / công tắc (button/switch) với MCU là chúng ta sử dụng một cách rõ ràng một điện trở kéo lên (pull-up resistor) bên ngoài như trong **Hình 9**:



Hình 9: Một cách tin cậy để kết nối 1 nút / công tắc với MCU

1.6.2.2 Đối mặt với "nảy" nút bấm (switch bounce)

Ta biết rằng trong thực tế, tất cả các tiếp điểm của công tắc cơ sẽ bật lên (nghĩa là bật và tắt, liên tục, trong một khoảng thời gian ngắn) sau khi công tắc được đóng hoặc mở như thể hiện như hình sau:



Hình 10: Hiện tượng switch bounce

Phương pháp gõ lõi (debounce) đơn giản nhất là kiểm tra các phím (hoặc các nút hoặc công tắc) sau mỗi $N \text{ ms}$, trong đó $10 < N \leq 50$. Khi đó, ta có ba kết quả có thể xảy ra mỗi khi chúng ta đọc một nút:

- Ta đọc được trạng thái bền là 0.
- Ta đọc được trạng thái bền là 1.
- Ta đọc được giá trị khi nó đang "nảy", tức là có thể nhận giá trị 0 hoặc 1.

Đối với trường hợp 1 và 2 không có vấn đề gì, vì chúng là những gì chúng ta luôn muốn xảy ra. Trường hợp 3 cũng không có vấn đề gì vì trong thời gian "nảy", một trong hai trạng thái đều có thể chấp nhận được. Nếu ta vừa nhấn một nút hoạt động ở mức thấp và ta đọc số 1 khi nó bị "nảy", thì lần tiếp theo ta được đảm bảo đọc số 0 (hay nhớ rằng, lần tiếp theo thông qua tất cả các lần "nảy" sẽ không còn nữa), vì vậy ta sẽ chỉ phát hiện nút nhấn sau một chút. Ngược lại, nếu chúng ta đọc số 0 khi nút bị "nảy", nó sẽ vẫn là 0 vào lần tiếp theo sau khi tất cả



lần bật đã dừng lại, vì vậy ta chỉ phát hiện nút nhấn sớm hơn một chút. Điều tương tự cũng áp dụng cho việc phát hành một nút. Việc đọc một lần bị "nảy" (với tất cả số lần bị "nảy" vào thời điểm đọc tiếp theo) sẽ không bao giờ cho ta trạng thái nút không hợp lệ (invalid). Nó chỉ đọc nhiều lần "nảy" (nhiều lần đọc trong khi xảy ra hiện tượng "nảy") có thể đưa ra các trạng thái nút không hợp lệ, chẳng hạn như tín hiệu đẩy lặp lại từ một lần nhấn vật lý (physical push).

Tuy nhiên nếu tiếng ồn quá lớn, ta không thể lọc nó chỉ bằng phần mềm, mà sẽ cần phần cứng của một số loại (hoặc thậm chí thiết kế lại). Nhưng nếu tiếng ồn chỉ thỉnh thoảng, ta có thể lọc nó trong phần mềm mà không cần quá bận tâm. Bí quyết là thay vì liên quan đến một nút duy nhất **tạo** hoặc **ngắt** là hợp lệ, ta nhấn mạnh vào N sự kiện **tạo** hoặc **ngắt** liền kề để đánh dấu sự kiện nút hợp lệ. N sẽ là một yếu tố của tỷ lệ quét nút của bạn và số lượng bộ lọc bạn muốn thêm. N lớn hơn cho phép lọc nhiều hơn. Bộ lọc đơn giản nhất (nhưng vẫn là một cải tiến lớn so với không lọc) chỉ là N trên 2, có nghĩa là so sánh trạng thái nút hiện tại với trạng thái nút cuối cùng và chỉ khi cả hai đều giống nhau thì đầu ra mới hợp lệ.

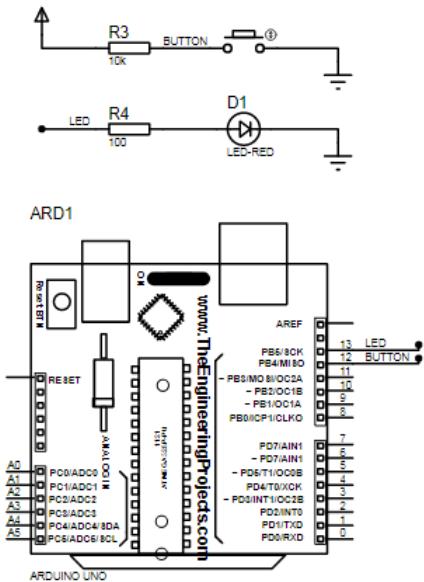
Lưu ý rằng bây giờ ta không có hai mà là ba trạng thái nút: hoạt động (hoặc được nhấn), không hoạt động (hoặc được thả) và không xác định hoặc không hợp lệ (đang lọc (filtering), chưa được lọc (filtered)). Trong hầu hết các trường hợp, ta có thể coi trạng thái không hợp lệ giống như trạng thái không hoạt động, vì trong hầu hết các trường hợp, ta chỉ quan tâm đến thời điểm hoạt động (từ bất kỳ trạng thái nào) và khi nào ngừng hoạt động (không hoạt động hoặc không hợp lệ). Với sự đơn giản hóa đó, ta có thể xem xét lọc (filtering) $N = 2$ đơn giản ở các phần sau.

Để mở rộng thành lọc (filtering) lớn hơn (N lớn hơn), hãy nhớ rằng kỹ thuật lọc về cơ bản liên quan đến việc đọc trạng thái nút hiện tại và sau đó đếm hoặc gửi lại bộ đếm. Ta sẽ đếm nếu trạng thái nút hiện tại giống với trạng thái nút cuối cùng và nếu số lượng đạt đến N thì sẽ báo cáo trạng thái nút mới hợp lệ. Ta đặt lại bộ đếm nếu trạng thái nút hiện tại khác với trạng thái nút cuối cùng và sau đó lưu trạng thái nút hiện tại làm trạng thái nút mới để so sánh với lần sau. Cũng lưu ý rằng giá trị N của ta càng lớn thì quy trình lọc của ta càng phải được gọi thường xuyên hơn, để ta nhận được phản hồi được lọc trong thời hạn **50ms** được chỉ định của ta. Vì vậy, ví dụ với $N = 8$, ta nên gọi quy trình lọc cứ sau **2 – 5ms**, cho thời gian phản hồi là **16 – 40ms** ($> 10ms$ và $< 50ms$).

1.6.3 Cài đặt

Để đơn giản hóa việc cài đặt, ta giả sử chỉ đọc nút bấm bằng cách đối mặt với switch debounce với $N = 2$ như trên với việc nhấn một nút nút nhấn làm thay đổi trạng thái của đèn.

1.6.3.1 Schematic của mạch điều khiển đèn bằng nút nhấn đơn giản



Hình 11: Schematic của mạch điều khiển đèn bằng nút nhấn

1.6.3.2 Cài đặt

```
1 #define LED_PIN 13
2 #define BUTTON_PIN 12
3 #define RELEASED HIGH
4 #define PRESSED LOW
5
6 int debounce1 = RELEASED, debounce2 = RELEASED;
7 int buttonFlag = 0;
8
9 void setup() {
10   pinMode(LED_PIN, OUTPUT);
11   pinMode(BUTTON_PIN, INPUT_PULLUP);
12
13   //initialization timer interrupt (10ms)
14   cli();
15   TCCR1A = 0;
16   TCCR1B = 0;
17   TCNT1 = 0;
18   OCR1A = 624;
19   TCCR1B |= (1 << WGM12);
20   TCCR1B |= 1 << CS12;
21   TIMSK1 |= (1 << OCIE1A);
22   sei();
23 }
24 void loop() {}
```



```
25 ISR(TIMER1_COMPA_vect){  
26     debounce2 = debounce1;  
27     debounce1 = digitalRead(BUTTON_PIN);  
28     if(debounce1 == debounce2){  
29         if(debounce2 == PRESSED){  
30             if(buttonFlag == 0) digitalWrite(LED_PIN, !digitalRead(↔  
31                             LED_PIN));  
32             buttonFlag = 1;  
33         }  
34         else buttonFlag = 0;  
35     }  
36 }
```

Code 3: Chương trình điều khiển đèn bằng nút bấm

2 Hệ thống giám sát không khí trong nhà (Indoor Air Monitoring)

2.1 Giới thiệu

Song song với quá trình công nghiệp hóa, đô thị hóa là vấn nạn ô nhiễm môi trường. Tác hại của sự thay đổi lớn trong thành phần của không khí, do khói, bụi, hơi hoặc các khí lạ, làm giảm tầm nhìn xa, biến đổi khí hậu, gây bệnh cho con người và cho sinh vật khác hoặc hủy hoại nhiều hệ sinh thái. Các thiết bị giám sát chất lượng không khí trong thực tế còn thủ công và hướng đến đối tượng công nghiệp. Do đó việc triển khai ứng dụng Internet Of Things vào việc xây dựng một hệ thống quản lý, giám sát chất lượng không khí nhằm mục đích khắc phục các hạn chế thực tại của các công cụ, thu thập liên tục tự động theo thời gian thực các giá trị về: Nhiệt độ, độ ẩm, bụi mìn 2,5, mức bức xạ tia tử ngoại UV, nồng độ khí các thải, và chỉ số AQI (Air Quality Index). Thiết bị có kích thước nhỏ, phù hợp với nhiều đối tượng sử dụng, trên nền tảng Website hoặc ứng dụng điện thoại thông minh (Android hoặc IOS) người dùng có thể theo dõi các thông số, để từ đó có thể phân tích dữ liệu, dự đoán sự biến đổi, nhận được cảnh báo, hoặc đưa ra các biện pháp xử lý, khắc phục.

Trong ngôi nhà thông minh thì nhiệt độ và độ ẩm đóng vai trò tiên quyết quyết định đến chất lượng cuộc sống của con người và kể cả trong nông nghiệp. Theo trang web được đề cập trong mục [7] ở **Tài liệu** có đưa ra về mối quan hệ giữa điểm sương (Dew point) tới nhận thức của con người được thể hiện như sau:

Điểm sương (Dew point)	Nhận thức của con người (Human perception)	Độ ẩm tương đối tại $32^{\circ}C(90^{\circ}F)$
$> 24^{\circ}C$	$> 75^{\circ}F$	Cực kỳ khó chịu, ngột ngạt
$21 - 24^{\circ}C$	$70 - 74^{\circ}F$	Rất ẩm, khá khó chịu
$18 - 21^{\circ}C$	$65 - 69^{\circ}F$	Hơi khó chịu đối với hầu hết mọi người ở mức trên (upper edge)
$16 - 18^{\circ}C$	$60 - 64^{\circ}F$	OK đối với hầu hết mọi người, nhưng tất cả đều nhận thấy độ ẩm ở mức trên (upper edge)
$13 - 16^{\circ}C$	$55 - 59^{\circ}F$	Thoải mái
$10 - 12^{\circ}C$	$50 - 54^{\circ}F$	Rất thoải mái
$< 10^{\circ}C$	$< 49^{\circ}F$	Hơi khô một chút

Bảng 8:Ảnh hưởng của điểm sương đối với nhận thức của con người

Trong phạm vi của đồ án ta chỉ quan tâm đến việc đo nhiệt độ và độ ẩm, hai thông số quan trọng nhất và phổ biến nhất được nhắc tới nhiều trong các thiết bị tự động hóa và trong cuộc sống hằng ngày của con người.

2.2 Yêu cầu đặt ra

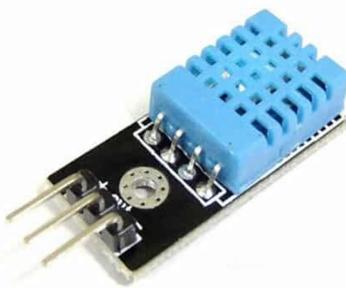
Trong khuôn khổ của việc giới hạn yêu cầu đo và xuất ra kết quả về nhiệt độ và độ ẩm, ta đặt ra một số yêu cầu và giới hạn như sau:

- Sử dụng cảm biến DHT11/DHT22 để đọc kết quả về nhiệt độ và độ ẩm.
- Hệ thống phải hiển thị được nhiệt độ và độ ẩm trên màn hình LCD.
- Hệ thống phải cập nhật lại kết quả đo **10s** 1 lần.
- Trong chương trình không được sử dụng vòng lặp **for** và **delay** với thời gian lớn.

2.3 Giới thiệu về cảm biến đo nhiệt độ và độ ẩm DHT11

2.3.1 Giới thiệu sơ lược

Cảm biến đo nhiệt độ và độ ẩm DHT11 là một cảm biến rất thông dụng hiện nay vì chi phí rẻ và dễ lấy dữ liệu thông qua giao tiếp 1 wire. Bộ tiền xử lý tín hiệu thích hợp trong cảm biến giúp có được dữ liệu chính xác mà không thông qua bất kỳ tính toán nào.



Hình 12: Hình ảnh thực tế của cảm biến DHT11



2.3.2 Thông số kỹ thuật:

Khái quát:

Item	Measurement range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20 – 90%RH, 0 – 50°C	±5%RH	±2°C	1	4 chân trên 1 hàng

Bảng 9: Bảng khái quát chung các thông số của DHT11

Chi tiết thông số về độ ẩm:

Thông số	Điều kiện	Min.	Typical	Max.	Đơn vị
Độ chia nhỏ nhất (Resolution)	–	1.0	1.0	1.0	%RH
		–	8 bit	–	
Độ lặp lại (Repeatability)	–	–	±1.0	–	%RH
		25°C	–	±4.0	
Độ chính xác (Accuracy)	–	0 – 50°C	–	–	%RH
		–	–	±5.0	
Khả năng thay đổi được (Interchangeability)	Hoàn toàn có thể thay đổi được (Fully interchangeable)				
Khoảng đo (Measurement range)	0°C	30.0	–	90.0	%RH
	25°C	20.0	–	90.0	
	50°C	20.0	–	80.0	
Thời gian phản hồi (Response time)	1/e (63%) 25°C, vận tốc không khí (air) 1m/s	6.0	10.0	15.0	s
Độ trễ (Hysteresis)	–	–	±1.0	–	%RH
Độ ổn định lâu dài (Long-term stability)	Typical	–	±1.0	–	%RH/year

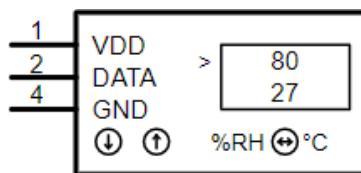
Bảng 10: Bảng chi tiết thông số về độ ẩm của DHT11

Chi tiết thông số về nhiệt độ:

Thông số	Điều kiện	Min.	Typical	Max.	Đơn vị
Độ chia nhỏ nhất (Resolution)	–	1.0	1.0	1.0	°C
		8 bit	8 bit	8 bit	
Độ lặp lại (Repeatability)	–	–	±1.0	–	°C
		–	–	–	
Độ chính xác (Accuracy)	–	±1.0	–	±2.0	°C
		–	–	–	
Khoảng đo (Measurement range)	–	0.0	–	50.0	°C
Thời gian phản hồi (Response time)	1/e (63%)	6.0	–	30.0	s

Bảng 11: Bảng chi tiết thông số về nhiệt độ của DHT11

2.3.3 Chức năng các chân



Hình 13: Các chân của DHT11

Ta có bảng thể hiện tên và chức năng của từng chân của DHT11 như sau:

Chân số	Kí hiệu	Mô tả
1	V_{cc}	Nguồn 3.5V đến 5.5V
2	Data	Dầu ra cả nhiệt độ và độ ẩm thông qua dữ liệu nối tiếp
3	NC	Không có kết nối và do đó không sử dụng
4	GND	Nối đất

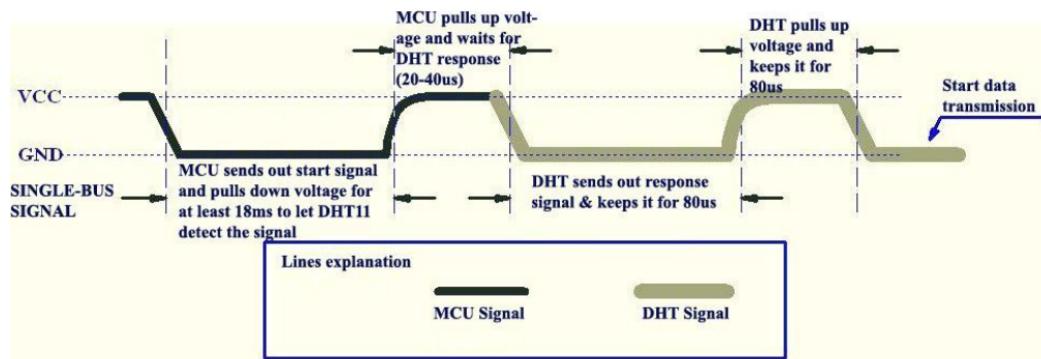
Bảng 12: Chức năng các chân của DHT11

2.3.4 Nguyên lý hoạt động

Để có thể giao tiếp với DHT11 theo chuẩn một chân vi xử lý cần thực hiện 2 bước:

- Gửi tín hiệu muôn đo (Start) tới DHT11, sau đó DHT11 xác nhận lại.
- Khi đã giao tiếp với DHT11, cảm biến sẽ gửi lại 5 byte dữ liệu và nhiệt độ đo được.

2.3.4.1 MCU gửi tín hiệu Start cho DHT11



Hình 14: Cách thức hoạt động

- MCU thiết lập chân DATA là Output, kéo chân DATA xuống 0 trong khoảng thời gian $> 18ms$. Khi đó DHT11 sẽ hiểu là MCU muốn đo nhiệt độ và độ ẩm.
- MCU đưa chân DATA lên 1, sau đó thiết lập lại là chân đầu vào.
- Sau khoảng thời gian $20 - 40\mu s$, DHT11 sẽ kéo chân DATA xuống thấp. Nếu $> 40\mu s$ mà chân DATA không được kéo xuống thấp nghĩa là không giao tiếp được với DHT11.

- Chân DATA sẽ ở mức thấp $80\mu s$ sau đó nó được DHT11 kéo lên cao trong $80\mu s$. Bằng việc giám sát chân DATA, MCU có thể biết được có giao tiếp với DHT11 hay không. Nếu tín hiệu do được DHT11 kéo cao, khi đó hoàn thiện quá trình giao tiếp giữa MCU và DHT11.

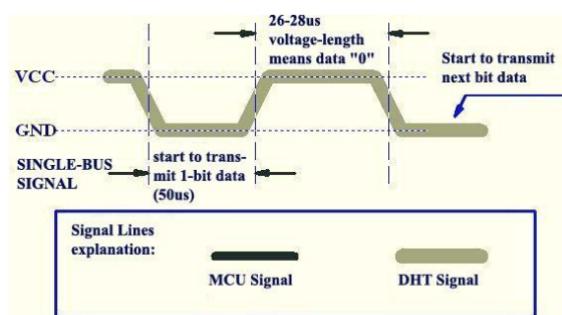
2.3.4.2 MCU đọc giá trị trên DHT11

DHT11 sẽ trả về nhiệt độ và độ ẩm về dưới dạng 5 bytes. Trong đó:

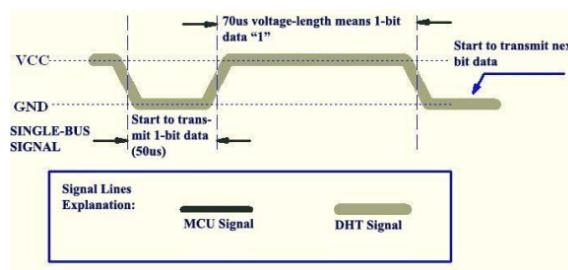
- Byte 1:** Giá trị phần nguyên của độ ẩm ($\%RH$)
- Byte 2:** Giá trị phần thập phân của độ ẩm ($\%RH$)
- Byte 3:** Giá trị phần nguyên của nhiệt độ ($^{\circ}C$)
- Byte 4:** Giá trị phần thập phân của nhiệt độ ($^{\circ}C$)
- Byte 5:** Kiểm tra tổng

Byte 5 dùng để kiểm tra tổng, tức là nếu $Byte 5 = Byte 1 + Byte 2 + Byte 3 + Byte 4$ thì giá trị nhiệt độ và độ ẩm là chính xác, nếu sai thì kết quả đo không có ý nghĩa.

Như vậy rõ ràng ta thấy sau khi giao tiếp được với DHT11 thì DHT11 sẽ gửi liên tiếp 40 bits 0 và 1 về MCU, tương ứng chia làm 5 bytes của nhiệt độ và độ ẩm. Cách nhận biết bit 0 hay 1 được thể hiện như sau:



Hình 15: Nhận biết bit 0



Hình 16: Nhận biết bit 1

Sau khi tín hiệu được đưa về 0, ta đợi chân DATA của DHT11 được MCU kéo lên 1. Nếu chân DATA là 1 trong khoảng $26 - 28\mu s$ thì là bit 0, còn nếu tồn tại $70\mu s$ thì là bit 1. Do đó trong lặp trình ta bắt sườn của chân DATA, sau đó delay $50\mu s$. Nếu giá trị đo được là 0 thì ta đọc được là 0, nếu giá trị đo được là 1 thì ta đọc được là 1. Cứ như thế ta đọc các bit tiếp theo.

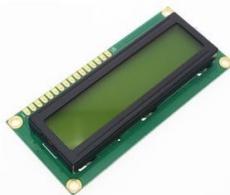
2.4 Giới thiệu về màn hình LED 16×2

2.4.1 Giới thiệu sơ lược

Ngày nay, thiết bị hiển thị **LCD (Liquid Crystal Display)** được sử dụng trong rất nhiều các ứng dụng của vi điều khiển. LCD có rất nhiều ưu điểm so với các dạng hiển thị khác:

- Có khả năng hiển thị kí tự đa dạng, trực quan (chữ, số và kí tự đồ họa).
- Dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau.
- Tốn rất ít tài nguyên hệ thống và giá thành rẻ, ...

LED 16×2 là một màn hình hiển thị bao gồm nhiều ma trận nhỏ, khi hoạt động thì LED 16×2 sẽ hiển thị các kí tự trong bảng mã ASCII. Vi điều khiển gửi các tín hiệu khởi tạo cho LED 16×2 , sau đó hiển thị các kí tự lên màn hình hiển thị.



Hình 17: Hình ảnh thực tế của LED 16×2

2.4.2 Thông số kỹ thuật

Dặc điểm về điện tử (Electrical characteristics):

Thông số	Điều kiện	Min.	Typical	Max.	Đơn vị
Điện thế cung cấp cho LED	$T_a = 25^{\circ}C$	0.0	4.4	13.5	V
Điện áp vào mức cao (V_{iH})	—	2.2	—	V_{DD}	V
Điện áp vào mức thấp (V_{iL})	—	—	—	0.6	V
Điện áp ra mức cao (V_{OH})	$-I_{OH} = 0.2mA$	2.4	—	—	V
Điện áp ra mức thấp (V_{OL})	$I_{OL} = 1.2mA$	—	—	0.4	V
Dòng điện cung cấp nguồn (I_{DD})	$V_{DD} = 5.0V$	—	1.0	3.0	mA

Bảng 13: Đặc điểm về điện tử của LED 16×2

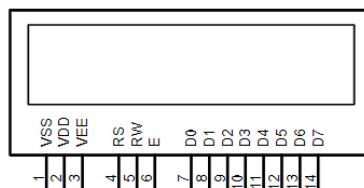
Dặc điểm về cơ khí (Mechanical characteristics):

Thông số	Kích thước	Đơn vị
Kích thước module	$84W \times 44H \times 12D$	mm
Vùng hiển thị hiệu quả	$61W \times 15.8H$	mm
Kích thước kí tự (5×7 dots)	$2.96W \times 4.86H$	mm
Kích thước pitch	3.55	mm
Kích thước dot	$0.56W \times 0.66H$	mm

Bảng 14: Đặc điểm về cơ khí của LED 16×2



2.4.3 Chức năng các chân



Hình 18: Các chân của LED 16×2

Ta có bảng thể hiện tên và chức năng của từng chân của LED 16×2 như sau:

Chân số	Kí hiệu	Mức (Level)	Chức năng
1	V_{SS}	–	Cung cấp nguồn
2	V_{DD}	–	
3	V_O	–	
4	RS	H/L	L: Đầu vào mã lệnh (instruction) H: Đầu vào dữ liệu (Data)
5	R/W	H/L	H: Đọc từ module LCD L: Viết vào module LCD
6	E	$H, H \rightarrow L$	Kích hoạt tín hiệu
7	$DB0$	H/L	8 đường của bus dữ liệu dùng để trao đổi thông tin với MPU. Có 2 chế độ sử dụng 8 đường bus này: + Chế độ 8 bit: Dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7. + Chế độ 4 bit: Dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7
8	$DB1$	H/L	
9	$DB2$	H/L	
10	$DB3$	H/L	
11	$DB4$	H/L	
12	$DB5$	H/L	
13	$DB6$	H/L	
14	$DB7$	H/L	

Bảng 15: Tên và chức năng của từng chân của LED 16×2

2.4.4 Giới thiệu về DDRAM – Display Data RAM

Dữ liệu hiển thị RAM (DDRAM) lưu trữ dữ liệu hiển thị được biểu thị bằng mã ký tự 8 bit. Dung lượng mở rộng của nó là 80×8 bits, hoặc 80 ký tự. Khu vực trong DDRAM không được sử dụng để hiển thị có thể được sử dụng làm RAM dữ liệu chung. Vì vậy, bất cứ thứ gì bạn gửi trên DDRAM thực sự được hiển thị trên màn hình LCD. Đối với màn hình LCD như 1×16 , chỉ có 16 ký tự được hiển thị, vì vậy bất kỳ thứ gì bạn viết sau 16 ký tự đều được viết bằng DDRAM nhưng người dùng sẽ không nhìn thấy được.

Hình dưới đây là địa chỉ DDRAM của LCD 1 dòng, 2 dòng:



Hình 19: Địa chỉ DDRAM với LCD 1 dòng



00 01 02 03 04 05 06 07	32 33 34 35 36 37 38 39	+ Character position (dec.)
00 01 02 03 04 05 06 07	20 21 22 23 24 25 26 27	+ Row0 DDRAM address (hex)
40 41 42 43 44 45 46 47	60 61 62 63 64 65 66 67	+ Row1 DDRAM address (hex)

Hình 20: Địa chỉ DDRAM với LCD 2 dòng

2.4.5 Giới thiệu về CGROM – Character Generator ROM

Bộ tạo ký tự ROM (CGROM) lưu trữ phông chữ được hiển thị trên màn hình LCD ký tự. Theo định nghĩa, (vì nó là ROM) không thể thay đổi phông chữ được lưu trữ trong CGROM. Vì CGROM hoàn toàn được xác định tại thời điểm sản xuất bộ điều khiển LCD, các nhà thiết kế cũng bao gồm một CGRAM, cho phép các bitmap của một vài ký tự được xác định lại tại thời điểm chạy.

CGROM tạo ra các mẫu ký tự 5×8 dot hoặc 5×10 dot từ mã ký tự 8 bits. Nó có thể tạo ra 208 mẫu ký tự 5×8 dot và 32 mẫu ký tự 5×10 dot. Các mẫu ký tự người dùng tự định nghĩa (user-defined) cũng có sẵn bằng ROM lập trình mặt nạ (mask-programming ROM).

Upper 4 bits	Lower 4 bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	(1)	0	0	P	^	P	-	タ	ミ	exp							
xxxx0001	(2)	!	1	A	Q	a	q	■	ア	チ	4	ä	q				
xxxx0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ				
xxxx0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	ω				
xxxx0100	(5)	\$	4	D	T	d	t	、	エ	ト	ト	μ	ο				
xxxx0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü				
xxxx0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ				
xxxx0111	(8)	*	7	G	W	g	w	ア	キ	ヌ	ラ	g	π				
xxxx1000	(1)	(8	H	X	h	x	イ	ク	ネ	リ	j	×				
xxxx1001	(2))	9	I	Y	i	y	ウ	ケ	ジ	ル	・	γ				
xxxx1010	(3)	*	:	J	Z	j	z	エ	コ	ハ	レ	j	≠				
xxxx1011	(4)	+	;	K	L	k	l	オ	サ	ヒ	ロ	^	フ				
xxxx1100	(5)	,	<	L	¥	1	l	ヤ	シ	フ	ワ	Φ	円				
xxxx1101	(6)	-	=	M	J	m	j	ュ	ス	ヘ	ン	£	÷				
xxxx1110	(7)	.	>	N	^	n	♪	ヨ	セ	ホ	^	ñ					
xxxx1111	(8)	/	?	O	_	o	†	ソ	ソ	マ	¶	ö	■				

Hình 21: Bản đồ mã ký tự LCD (LCD characters code map) cho 5×8 dot

2.4.6 Giới thiệu về CGRAM – Character Generator RAM

Vùng CGRAM được sử dụng để tạo các ký tự tùy chỉnh trong màn hình LCD. Trong RAM của bộ tạo ký tự, người dùng có thể viết lại các mẫu ký tự bằng chương trình. Đối với 5×8 dot, có thể viết 8 mẫu ký tự và đối với 5×10 dot, có thể viết 4 mẫu ký tự.

2.4.7 Các thanh ghi

Chip HD44780 có 2 thanh ghi 8 bit quan trọng:

- Thanh ghi lệnh IR (Instructor Register)



Để điều khiển LCD, người dùng phải "ra lệnh" thông qua 8 đường bus $DB0 - DB7$. Mỗi lệnh được nhà sản xuất LCD đánh địa chỉ rõ ràng. Người dùng chỉ việc cung cấp địa chỉ lệnh bằng cách nạp vào thanh ghi IR. Nghĩa là, khi ta nạp vào thanh ghi IR một chuỗi 8 bits, chip HD44780 sẽ tra bảng mã lệnh tại địa chỉ mà IR cung cấp và thực hiện lệnh đó.

- **Thanh ghi dữ liệu DR (Data Register)**

Thanh ghi DR dùng để chứa dữ liệu 8 bits để ghi vào vùng RAM DDRAM hoặc CGRAM (ở chế độ ghi) hoặc dùng để chứa dữ liệu từ 2 vùng RAM này gửi ra cho MPU (ở chế độ đọc). Nghĩa là, khi MPU ghi thông tin vào DR, mạch nội bộ trong chip sẽ tự động ghi thông tin này vào DDRAM hoặc CGRAM. Hoặc khi thông tin về địa chỉ được ghi vào IR, dữ liệu ở địa chỉ này trong vùng RAM nội của HD44780 sẽ được chuyển ra DR để truyền cho MPU.

Bằng cách điều khiển chân RS và R/W chúng ta có thể chuyển qua lại giữ 2 thanh ghi này khi giao tiếp với MPU. Bảng sau đây tóm tắt lại các thiết lập đối với hai chân RS và R/W theo mục đích giao tiếp:

RS	R/W	Chức năng
0	0	Ghi vào thanh ghi IR để ra lệnh cho LCD
0	1	Đọc cờ bận (Busy flag) ở DB7 và giá trị của bộ đếm địa chỉ ở $DB0 - DB6$
1	0	Ghi vào thanh ghi DR
1	1	Đọc dữ liệu từ DR

Bảng 16: Chức năng chân RS và R/W theo mục đích sử dụng

2.4.8 Cờ báo bận (Busy flag - BF)

Cờ bận (Busy flag) là cờ chỉ báo trạng thái cho LCD. Khi ta gửi một lệnh (command) hoặc dữ liệu (data) đến màn hình LCD để xử lý, cờ (flag) này được đặt (tức là $BF = 1$) và ngay sau khi lệnh được thực hiện thành công, cờ này sẽ bị xóa ($BF = 0$). Điều này rất hữu ích trong việc tạo ra và lượng độ trễ chính xác cho quá trình xử lý màn hình LCD.

Để đọc Cờ bận, điều kiện $RS = 0$ và $R/W = 1$ phải được thỏa mãn và MSB của bus dữ liệu LCD ($D7$) hoạt động như cờ bận. Khi $BF = 1$ nghĩa là LCD đang bận và sẽ không chấp nhận lệnh hoặc dữ liệu tiếp theo và $BF = 0$ nghĩa là LCD đã sẵn sàng cho lệnh hoặc dữ liệu tiếp theo để xử lý.

2.4.9 Tập lệnh (instruction) và bộ lệnh (command) của LCD

MCU chỉ có thể điều khiển thanh ghi lệnh (IR) và thanh ghi dữ liệu (DR) của màn hình LCD. Trước khi bắt đầu hoạt động bên trong của LCD, thông tin điều khiển được lưu tạm thời vào các thanh ghi này để cho phép giao tiếp với các MCU khác nhau, hoạt động ở các tốc độ khác nhau hoặc các thiết bị điều khiển ngoại vi khác nhau. Hoạt động bên trong của LCD được xác định bởi các tín hiệu được gửi từ MCU. Các tín hiệu này, bao gồm tín hiệu lựa chọn thanh ghi (RS), tín hiệu đọc/ghi (R/W) và bus dữ liệu ($DB0 \rightarrow DB7$), tạo nên các lệnh LCD. Có bốn loại lệnh như sau:

- Chỉ định các chức năng LCD, chẳng hạn như định dạng hiển thị, độ dài dữ liệu, v.v.
- Đặt địa chỉ RAM nội bộ.
- Thực hiện truyền dữ liệu với RAM nội.



- Thực hiện các chức năng khác.

Tên lệnh	Code															
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0						
Clear display	0	0	0	0	0	0	0	0	0	1						
Return home	0	0	0	0	0	0	0	0	1	*						
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S						
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B						
Cursor & Display Shift	0	0	0	0	0	1	S/C	R/L	*	*						
Function Set	0	0	0	0	1	DL	N	F	*	*						
Set CGRAM Address	0	0	0	1	A_{CG}											
Set DDRAM Address	0	0	1	A_{CG}												
Read BF & Address	0	1	BF	AC												
Write Data to CG or DDRAM	1	0	Write data													
Read Data to CG or DDRAM	1	1	Read data													

Bảng 17: Mã code của từng câu lệnh của LCD



Tên lệnh	Mô tả	Thời gian thực thi
Clear display	Xóa hiển thị (Clear Display) và đưa con trỏ trở về vị trí home (vị trí 0).	$82\mu s \rightarrow 1.64ms$
Return home	Lệnh Return home trả bộ đếm địa chỉ AC về 0, trả lại kiểu hiển thị gốc nếu nó bị thay đổi. Nội dung của DDRAM không thay đổi.	$40\mu s \rightarrow 1.64ms$
Entry Mode Set	Dặt hướng di chuyển con trỏ và kích hoạt/không kích hoạt hiển thị + I/D : Tăng ($I/D = 1$) hoặc giảm ($I/D = 0$) bộ đếm địa chỉ hiển thị AC 1 đơn vị mỗi khi có hành động ghi hoặc đọc vùng DDRAM. Vị trí con trỏ cũng di chuyển theo sự tăng giảm này. + S : Khi $S = 1$ toàn bộ nội dung hiển thị bị dịch sang phải ($I/D = 0$) hoặc sang trái ($I/D = 1$) mỗi khi có hành động ghi hoặc đọc vùng DDRAM. Khi $S = 0$: không dịch nội dung hiển thị. Nội dung hiển thị không dịch khi đọc DDRAM hoặc đọc/ghi vùng CGRAM.	$40\mu s$
Display ON/OFF Control	+ D : Hiển thị màn hình khi $D = 1$ và không hiển thị khi $D = 0$. Khi tắt hiển thị, nội dung DDRAM không thay đổi. + C : Hiển thị con trỏ khi $C = 1$ và không hiển thị khi $C = 0$. + B : Nhấp nháy kí tự tại vị trí con trỏ khi $B = 1$ và không nhấp nháy khi $B = 0$.	$40\mu s$
Cursor & Display Shift	Dịch chuyển con trỏ hay dữ liệu hiển thị sang trái mà không cần hành động ghi/đọc dữ liệu. Sau đây là nguyên lý hoạt động phụ thuộc vào giá trị của S/C và R/L : + $S/C = 0, R/L = 0$: Dịch vị trí con trỏ sang trái + $S/C = 0, R/L = 1$: Dịch vị trí con trỏ sang phải + $S/C = 1, R/L = 0$: Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo. + $S/C = 1, R/L = 1$: Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.	$40\mu s$
Function Set	+ DL : Thiết lập độ dài dữ liệu (data width). Khi $DL = 1$, LCD giao tiếp với MPU bằng giao thức 8 bits (từ $DB7 \rightarrow DB0$). Ngược lại, giao thức giao tiếp là 4 bits (từ $DB7 \rightarrow DB4$). Khi chọn giao thức 4 bits, dữ liệu được truyền/nhận 2 lần liên tiếp. với 4 bits cao gửi/nhận trước, 4 bits thấp gửi/nhận sau. + N : Thiết lập số hàng hiển thị. Khi $N = 0$: hiển thị 1 hàng, $N = 1$: hiển thị 2 hàng. + F : Thiết lập kiểu kí tự. Khi $F = 0$: kiểu kí tự 5×8 điểm ảnh, $F = 1$: kiểu kí tự 5×10 điểm ảnh.	$40\mu s$
Set CGRAM Address	Lệnh này ghi vào AC địa chỉ của CGRAM. Kí hiệu A_{CG} chỉ 1 bit của chuỗi dữ liệu 6 bits. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ CGRAM tại địa chỉ đã được chỉ định.	$40\mu s$

Tên lệnh	Mô tả	Thời gian thực thi
Set DDRAM Address	Lệnh này ghi vào AC địa chỉ của DDRAM, dùng khi cần thiết lập tọa độ hiển thị mong muốn. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ DDRAM tại địa chỉ đã được chỉ định.	$40\mu s$
Read BF & Address	Như đã đề cập trước đây, khi cờ BF bật, LCD đang làm việc và lệnh tiếp theo (nếu có) sẽ bị bỏ qua nếu cờ BF chưa về mức thấp. Cho nên, khi lập trình điều khiển, phải kiểm tra cờ BF trước khi ghi dữ liệu vào LCD. Khi đọc cờ BF, giá trị của AC cũng được xuất ra các bit AC. Nó là địa chỉ của CG hay DDRAM là tùy thuộc vào lệnh trước đó.	$1\mu s$
Write Data to CG or DDRAM	Viết dữ liệu vào CGRAM hoặc DDRAM.	$46\mu s$
Read Data to CG or DDRAM	Đọc dữ liệu từ CGRAM hoặc DDRAM.	$46\mu s$

Bảng 18: Mô tả các tập lệnh của LCD

2.5 Giới thiệu mạch chuyển đổi giao tiếp I2C

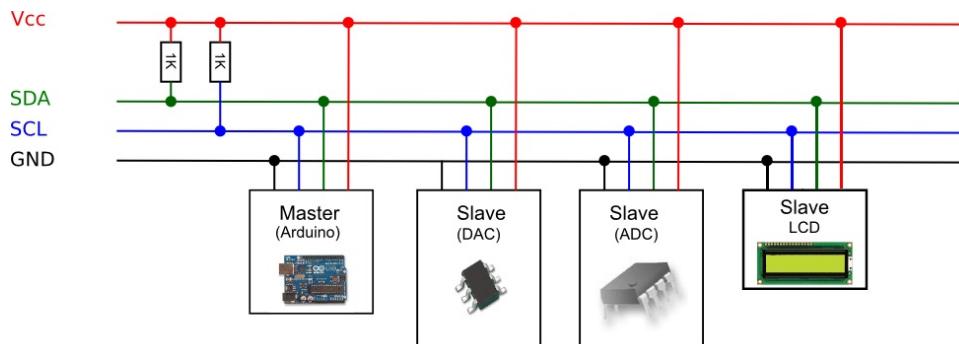
2.5.1 Chuẩn giao tiếp I2C

2.5.1.1 Giới thiệu sơ lược

Dầu năm 1980 Philips đã phát triển một chuẩn giao tiếp nối tiếp 2 dây được gọi là I2C. Đây là đường bus giao tiếp giữa các IC với nhau, I2C được phát triển bởi Philips, nhưng nó đã được rất nhiều nhà sản xuất IC sử dụng. I2C trở thành một chuẩn công nghiệp cho các giao tiếp điều khiển. Bus I2C được sử dụng làm bus giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như loại vi điều khiển 8051, PIC, AVR, ARM, Arduino, ... Chip nhớ như: RAM tĩnh (SRAM), EEPROM, bộ chuyển đổi tương tự sang số (ADC), số sang tương tự (DAC), IC điều khiển như LCD hay LED, ...

2.5.1.2 Đặc điểm giao tiếp I2C

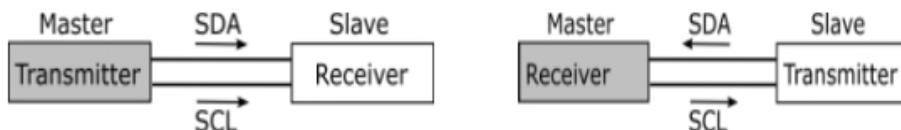
Một giao tiếp I2C gồm có hai dây: Serial Data (SDA) và Serial Clock (SCL). SDA là đường truyền dữ liệu hai hướng, còn SCL là đường truyền xung đồng hồ để đồng bộ và chỉ một hướng. Khi một thiết bị ngoại vi kết nối vào đường bus I2C thì chân SDA của nó nối với dây SDA của bus, chân SCL sẽ nối với dây SCL.



Hình 22: Cách kết nối thiết bị khi dùng chung I2C bus

Mỗi dây SCA hoặc SDL đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên (pull-up resistor) vì chân giao tiếp của I2C của các thiết bị ngoại vi thường là dạng cực máng hở (opendrain hay opencollector). Giá trị của điện trở này khác nhau tùy vào từng thiết bị và chuẩn giao tiếp, thường dao động trong khoảng $1k\Omega$ đến $4.7k\Omega$.

Trở lại với **Hình 20**, ta thấy có rất nhiều thiết bị cùng được kết nối vào một bus I2C, tuy nhiên sẽ không xảy ra chuyện nhầm lẫn giữa các thiết bị, bởi mỗi thiết bị sẽ được nhận ra bởi một địa chỉ duy nhất với một quan hệ chủ/tớ (Master/Slave) tồn tại trong suốt thời gian kết nối. Mỗi thiết bị có thể hoạt động như là thiết bị nhận dữ liệu hay có thể vừa truyền vừa nhận. Hoạt động truyền hay nhận còn tùy thuộc vào việc thiết bị đó là chủ (master) hay tớ (slave). Một thiết bị hay một IC khi kết nối với bus I2C, ngoài một địa chỉ (duy nhất) để phân biệt, nó còn được cấu hình là thiết bị chủ (master) hay tớ (slave). Khi giữa hai thiết bị chủ/tớ giao tiếp thì thiết bị chủ có nhiệm vụ tạo xung đồng hồ và quản lý địa chỉ của thiết bị tớ trong suốt quá trình giao tiếp. Thiết bị chủ giữ vai trò chủ động, còn thiết bị tớ giữ vai trò bị động trong việc giao tiếp.



Hình 23: Truyền nhận dữ liệu giữa chủ/tớ (Master/Slave)

Nhìn **Hình 21** trên ta thấy xung đồng hồ chỉ có một hướng từ chủ (Master) đến tớ (Slave), còn luồng dữ liệu có thể đi theo hai hướng, từ chủ (Master) đến tớ (Slave) hay ngược lại tớ (Slave) đến chủ (Master).

2.5.1.3 Chế độ hoạt động (Tốc độ truyền)

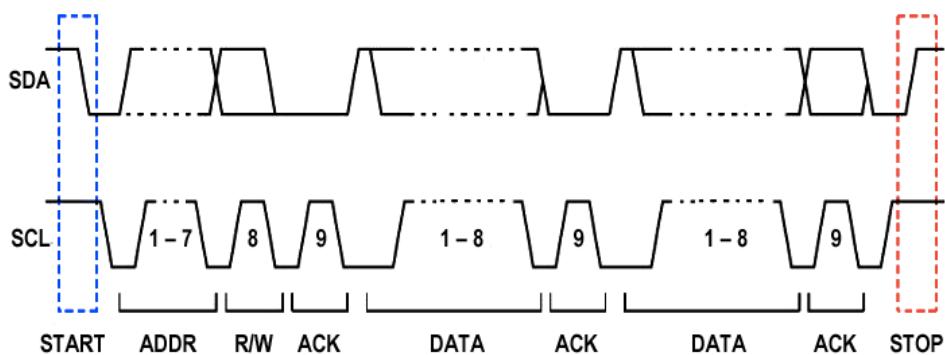
Các bus I2C có thể hoạt động ở ba chế độ khác nhau:

- Chế độ tiêu chuẩn (Standard mode).
- Chế độ nhanh (Fast mode).
- Chế độ cao tốc (High speed - HS mode).

Một bus I2C có thể hoạt động ở nhiều chế độ khác nhau:

- Một chủ - một tớ (One master - One slave).
- Một chủ - nhiều tớ (One master - Multi slave).
- Nhiều chủ - nhiều tớ (Multi master - Multi slave).

2.5.1.4 Trình tự truyền bit trên đường truyền



Hình 24: Trình tự truyền bit trên đường truyền

Bước 1: Thiết bị chủ tạo một điều kiện **START**. Điều kiện này thông báo cho tất cả các thiết bị tớ (Slave) chuẩn bị nhận dữ liệu từ đường truyền.

Bước 2: Thiết bị chủ (Master) gửi địa chỉ của thiết bị tớ (Slave) mà thiết bị chủ muốn giao tiếp và cờ (flag) đọc/ghi dữ liệu (nếu cờ thiết lập lên 1 byte tiếp theo được truyền từ thiết bị tớ đến thiết bị chủ, nếu cờ thiết lập xuống 0 thì byte tiếp theo truyền từ thiết bị chủ đến thiết bị tớ).

Bước 3: Khi thiết bị tớ trên bus I2C có địa chỉ đúng với địa chỉ mà thiết bị chủ gửi sẽ phản hồi lại bằng một xung **ACK**.

Bước 4: Giao tiếp giữa thiết bị chủ (Master) và thiết bị tớ (Slave) bắt đầu. Cả chủ (Master) và tớ (Slave) đều có thể nhận và truyền dữ liệu phụ thuộc vào việc truyền thông là đọc hay ghi. Bộ truyền gửi 8 bit dữ liệu tới bộ nhận. Bộ nhận phản hồi một xung **ACK**.

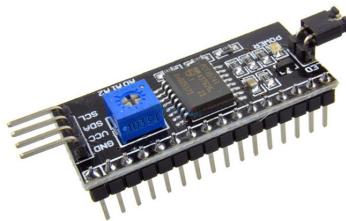
Bước 5: Để kết thúc quá trình giao tiếp, thiết bị chủ tạo ra một điều kiện **STOP**.

2.5.2 Mạch chuyển đổi giao tiếp I2C

2.5.2.1 Giới thiệu sơ lược

Thông thường để điều khiển và hiển thị các ký tự từ vi điều khiển (controller) xuất ra màn hình LCD ta cần **7 – 8** chân nối đến chân vi điều khiển (controller). Điều này gây ra nhiều phiền toái, dễ đứt dây, mạch rườm rà, khó viết chương trình. Những điều này có thể được khắc phục hoàn toàn bằng mạch chuyển đổi giao tiếp I2C vì số lượng dây tín hiệu giảm chỉ còn **2** dây. Bằng việc sử dụng giao tiếp I2C, việc giao tiếp màn hình được chuyển sang cho IC xử lý trên mạch. Ta chỉ cần gửi mã lệnh cùng với nội dung cần hiển thị, do vậy nó tạo điều kiện cho các vi điều khiển (controller) có nhiều thời gian xử lý các tiến trình phức tạp khác.

Để sử dụng các LCD có driver là HD44780 (LCD **16 × 2**, LCD **20 × 4**, ...) cần có ít nhất **6** chân của MCU kết nối với các chân **RS**, **EN**, **D7**, **D6**, **D5** và **D4** để có thể giao tiếp với LCD. Nhưng với module chuyển giao tiếp LCD sang I2C, ta chỉ cần **2** chân (SDA và SCL) của MCU kết nối với **2** chân (SDA và SCL) của hai module là đã có thể hiển thị thông tin lên LCD. Ngoài ra có thể điều chỉnh độ tương phản bằng biến trở ở trên module.



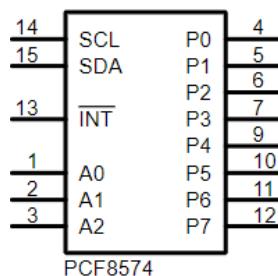
Hình 25: Hình ảnh thực tế của mạch I2C

2.5.2.2 Thông số kĩ thuật

Thông số	Điều kiện	Min.	Typical	Max.	Đơn vị
Điện thế cung cấp (V_{DD})	–	2.5	–	6.0	V
Dòng điện cấp nguồn (I_{DD})	+ Chế độ đang xử lý + $V_{DD} = 6V$ + Không tải + $V_I = V_{DD}$ hoặc V_{SS} + $f_{SCL} = 100kHz$	–	40.0	100.0	μA
Dòng tiêu thụ ở trạng thái chờ (I_{stb})	+ Chế độ đang chờ + $V_{DD} = 6V$ + Không tải + $V_I = V_{DD}$ hoặc V_{SS}	–	2.5	10	μA
Điện áp vào mức cao (V_{IH})	–	$0.7V_{DD}$	–	$V_{DD} + 0.5$	V
Điện áp vào mức thấp (V_{IL})	–	–0.5	–	$0.3V_{DD}$	V
Dòng điện ra mức cao (I_{OH})	$V_{OH} = V_{SS}$	10.0	25.0	–	μA
Dòng điện ra mức thấp (I_{OL})	$V_{OL} = 1V, V_{DD} = 5V$	10.0	25.0	–	μA
Tần số clock của SCL (f_{SCL})	–	–	–	100	kHz

Bảng 19: Thông số kĩ thuật của mạch I2C

2.5.2.3 Các chức năng các chân



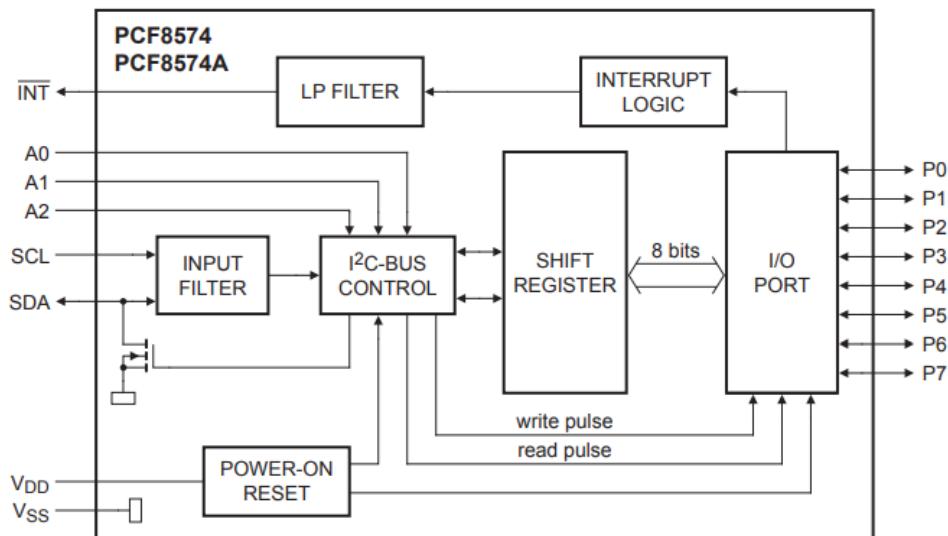
Hình 26: Các chân của mạch chuyển đổi giao tiếp I2C

Ta có bảng thể hiện tên và chức năng của từng chân của mạch chuyển đổi giao tiếp I2C như sau:

Chân số	Kí hiệu	Mô tả
1	A_0	Địa chỉ đầu vào 0
2	A_1	Địa chỉ đầu vào 1
3	A_2	Địa chỉ đầu vào 2
4	P_0	Quasi-bidirectional I/O 0
5	P_1	Quasi-bidirectional I/O 1
6	P_2	Quasi-bidirectional I/O 2
7	P_3	Quasi-bidirectional I/O 3
8	V_{SS}	Chân đất
9	P_4	Quasi-bidirectional I/O 4
10	P_5	Quasi-bidirectional I/O 5
11	P_6	Quasi-bidirectional I/O 6
12	P_7	Quasi-bidirectional I/O 7
13	INT	Gián đoạn tràn (Interrupt overflow) (tích cực mức THẤP)
14	SCL	Dường serial clock
15	SDA	Dường serial data

Bảng 20: Tên và chức năng của từng chân của mạch chuyển đổi I2C

2.5.2.4 Sơ đồ khối (Block diagram)



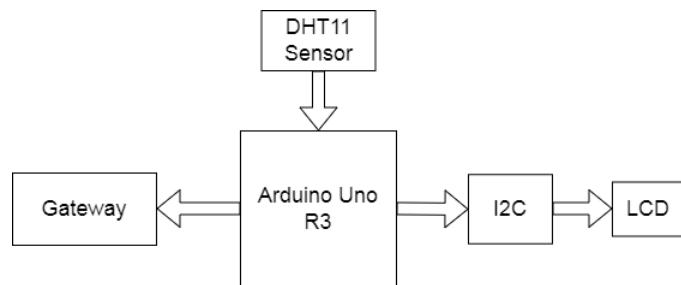
Hình 27: Sơ đồ khối của mạch chuyển đổi giao tiếp I2C

2.5.2.5 Nguyên lý hoạt động

Mỗi module I2C đều có một địa chỉ cho việc giao tiếp. Mặc định của nhà sản xuất cho module này là **0x27** tương ứng các chân **A₀**, **A₁**, **A₂** đều ở mức cao. Module giao tiếp với LCD theo chế

độ 4 bit với các chân $P_4 – P_7$ sẽ kết nối với $DB4 – DB7$. Khi cần gửi dữ liệu từ MCU sang hiển thị LCD, 1 byte dữ liệu sẽ được tách ra làm hai: 4 bits cao và 4 bits thấp. 4 bits cao được gửi trước và 4 bits thấp được gửi sau. Đồng thời 4 bit command cũng sẽ được đính kèm. Khi IC nhận được 1 byte dữ liệu, nó chuyển dữ liệu vào thanh ghi sau đó đưa ra các I/O để xuất ra LCD. Muốn chỉnh độ tương phản thì cần tùy chỉnh biến trở sao cho kết quả hiển thị tương phản trên LCD là dễ nhìn nhất.

2.6 Kiến trúc hệ thống

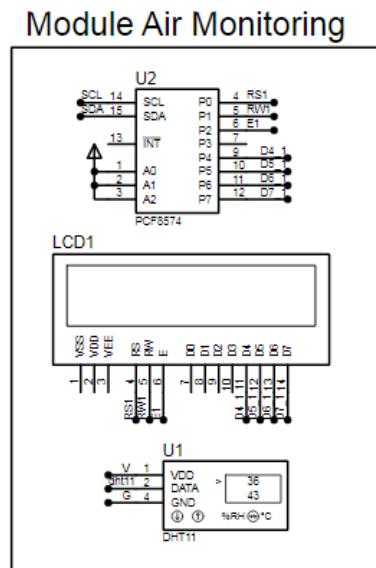


Hình 28: Sơ đồ kiến trúc hệ thống của module DHT11

Module gồm các thành phần: mạch Arduino Uno R3, mạch chuyển giao tiếp I2C, màn hình LCD 16×2 , cảm biến DHT11 và một gateway để giao tiếp với server.

Mạch Arduino sẽ có nhiệm vụ đọc dữ liệu từ cảm biến rồi hiển thị ra LCD thông qua mạch I2C và đồng thời truyền dữ liệu này lên gateway để đẩy lên server.

2.7 Schematic của module trên proteus





Chân DATA của cảm biến DHT11 sẽ được nối với chân 12 của Arduino và chân SCL, SDA của mạch I2C sẽ lần lượt được nối với chân SCL(A5), SDA(A4) của Arduino. Địa chỉ của LCD này sẽ là 0x27 nên 3 chân A0, A1, A2 của mạch I2C sẽ được đưa lên điện áp cao, được xác định theo bảng sau đây:

I2C Module Address Select

A0	A1	A2	HEX ADDRESS
0	0	0	0x27
1	0	0	0x26
0	1	0	0x25
1	1	0	0x24
0	0	1	0x23
1	0	1	0x22
0	1	1	0x21
1	1	1	0x20

Hình 30: Bảng định địa chỉ LCD tương ứng với cách nối chân

0 tương ứng với nối với điện áp cao, 1 tương ứng với không nối.

2.8 Mô tả hệ thống

Hệ thống sẽ định kì 2 giây sẽ đọc dữ liệu từ cảm biến và định kì 10 giây sẽ cập nhật thông tin nhiệt độ và độ ẩm của không khí trong nhà ra màn hình LCD.

2.9 Hiện thực hệ thống

2.9.1 Đọc dữ liệu từ cảm biến

2.9.1.1 Thư viện:

Có 2 thư viện phổ biến cho việc đọc dữ liệu từ cảm biến DHT11 là [DHT11.h](#) và [dht_nonblocking.h](#)

- **DHT11.h:** Đây là thư viện được phát triển bởi Adafruit, là một công ty, một cộng đồng khá đông đảo, với nhiều thư viện dành cho Arduino. Và là thư viện được sử dụng khá nhiều bởi người dùng Arduino trong việc đọc dữ liệu từ cảm biến DHT. Chính vì vậy độ ổn định, tối ưu của nó được đảm bảo tin cậy.
- **dht_nonblocking.h:** Đây là thư viện được phát triển bởi Ole Wolf, mục đích là để đọc dữ liệu từ cảm biến DHT mà không có blocking các hoạt động tính toán khác của vi điều khiển.

Trong quá trình sử dụng cả 2 thư viện để chọn một thư viện tối ưu hơn cho hệ thống thì ta thấy thư viện **dht_nonblocking.h** nó có một xác suất đọc dữ liệu thất bại cao hơn hẳn so với **DHT.h**. Ví dụ để có được dữ liệu chính xác từ cảm biến thì **dht_nonblocking.h** nó sẽ cho kết quả chính xác sau **4 – 5** lần đọc, trong khi đó phải hiếm khi lăm thì **DHT.h** nó mới đọc kết quả không chính xác.



2.9.1.2 Hiện thực module đọc dữ liệu từ cảm biến DHT11

```
1 #include <DHT.h>
2 #include "_DHT11.h"
3 #include "Arduino.h"
4
5 const int DHTPIN = 12;
6 const int DHTTYPE = DHT11;
7 DHT dht(DHTPIN, DHTTYPE);
8
9 // Save temperature value is read from DHT11 sensor
10 float temperature;
11 // Save humidity value is read from DHT11 sensor
12 float humidity;
13
14
15 // Initialization of module
16 void init_DHT11(){
17     dht.begin();
18 }
19
20 void task_measure_environment(){
21     float h = dht.readHumidity();
22     float t = dht.readTemperature();
23     // Check if reading from DHT11 successfully
24     if(!isnan(t) && !isnan(h)){
25         temperature = t;
26         humidity = h;
27     }
28 }
```

Code 4: Hiện thực module đọc dữ liệu từ cảm biến DHT11

2.9.2 Hiện thị thông tin nhiệt độ độ ẩm ra màn hình LCD

2.9.2.1 Thư viện

Dể sử dụng I2C bus trên Arduino, chúng ta sẽ cần sử dụng thư viện `Wire.h` (đây là built-in library của Arduino) và thư viện `LiquidCrystal_I2C` để hiện thị ra LCD.

2.9.2.2 Hiện thực module LCD để hiển thị thông tin nhiệt độ độ ẩm ra màn hình

```
1 #include <LiquidCrystal_I2C.h>
2 #include <Wire.h>
3 #include "_LCD.h"
4 #include "Arduino.h"
5
6 LiquidCrystal_I2C lcd1(0x27, 16, 2);
```



```
7 // Use for display degree symbol
8 byte degree[8] =
9 {
10     0B01110 ,
11     0B01010 ,
12     0B01110 ,
13     0B00000 ,
14     0B00000 ,
15     0B00000 ,
16     0B00000 ,
17     0B00000 ,
18     0B00000
19 };
20
21 // Extern from DHT11 module
22 extern float temperature;
23 extern float humidity;
24
25 // Initialization of module
26 void init_LCD(){
27     lcd1.begin();
28     lcd1.backlight();
29     lcd1.createChar(1, degree);
30 }
31
32 // LCD 1 display informations of module DHT11
33 void task_DHT11_display(){
34     lcd1.clear();
35     lcd1.setCursor(0,0);
36     lcd1.print("T = ");
37     lcd1.print(temperature);
38     lcd1.write(1);
39     lcd1.print("C");
40     lcd1.setCursor(0,1);
41     lcd1.print("H = ");
42     lcd1.print(humidity);
43     lcd1.print("%");
44 }
```

Code 5: Hiển thực module LCD để hiển thị thông tin nhiệt độ độ ẩm ra màn hình

2.10 Demo

Video demo của module: [Link video](#)

3 Hệ thống quạt làm mát dùng động cơ điện một chiều (Motor DC)

3.1 Giới thiệu

Các motor điều khiển đã sử dụng rộng rãi cho việc xây dựng kết cấu hạ tầng và các thiết bị gia dụng chính (home application) như quạt hay máy bơm với lưới cung cấp điện (grid connected power supply) cho phép hoạt động từ công suất thấp đến công suất cao.



Hình 31: Ứng dụng của motor một chiều (Motor DC)

Động cơ điện một chiều là động cơ điện hoạt động với dòng điện một chiều. Động cơ điện một chiều ứng dụng rộng rãi trong các ứng dụng dân dụng cũng như công nghiệp. Cấu tạo của động cơ gồm có 2 phần: stator đứng yên và rotor quay so với stator. Phần cảm (phần kích từ - thường đặt trên stator) tạo ra từ trường đi trong mạch từ, xuyên qua các vòng dây quấn của phần ứng (thường đặt trên rotor). Khi có dòng điện chạy trong mạch phần ứng, các thanh dẫn phần ứng sẽ chịu tác động bởi các lực điện từ theo phương tiếp tuyến với mặt trụ rotor, làm cho rotor quay. Tùy theo cách mắc cuộn dây rotor và stator mà người ta có các loại động cơ sau:

- Động cơ kích từ độc lập: Cuộn dây kích từ (cuộn dây stator) và cuộn dây phần ứng (rotor) mắc riêng rẽ nhau, có thể cấp nguồn riêng biệt.
- Động cơ kích từ nối tiếp: Cuộn dây kích từ mắc nối tiếp với cuộn dây phần ứng (rotor).

Đối với loại động cơ kích từ độc lập, người ta có thể thay thế cuộn dây kích từ bởi nam châm vĩnh cửu, khi đó ta có loại động cơ điện 1 chiều dùng nam châm vĩnh cửu. Đây là loại động cơ được sử dụng trong đồ án này.

Có nhiều phương pháp để điều khiển động cơ điện 1 chiều (DC Motor) bao gồm:

- Điều khiển bằng cách sử dụng điện trở.
- Điều khiển bằng cách điều khiển từ thông.



Hình 32: Hình ảnh thực tế của Motor DC

- Điều khiển bằng cách điều khiển điện áp phần ứng.

Sau đây ta sẽ giới thiệu một cách điều khiển tốc độ của động cơ điện 1 chiều bằng cách điều khiển điện áp phần ứng do khoảng điều chỉnh tốc độ rộng và giá thành rẻ bằng mạch Arduino ta đang xài ở đây. Cụ thể ta sẽ dùng nó để điều chỉnh tốc độ quay của quạt thông qua remote điều khiển quạt.

3.2 Yêu cầu đặt ra

Nhiệt độ của phòng không phải lúc nào cũng làm cho người ta thoải mái được, lúc nóng nực, lúc mát mẻ, chính vì vậy trong một ngôi nhà lúc nào cũng có một chiếc máy quạt để có thể thay đổi không khí làm cho ta cảm thấy thoải mái hơn. Sau đây là các yêu cầu đặt ra dành cho hệ thống quạt:

- Hệ thống hỗ trợ 3 chế độ quạt: chậm, nhanh vừa và nhanh.
- Hệ thống có chế độ auto tự động điều khiển quạt dựa vào nhiệt độ phòng.
- Hệ thống có thể ghi nhớ được chế độ hoạt động của điều hòa trước khi bị tắt để khi khởi động lại thì không phải mất công điều chỉnh lại.

3.3 Phương pháp điều khiển tốc độ động cơ (Motor)

Đối với loại động cơ kích từ độc lập dùng nam châm vĩnh cửu, để thay đổi tốc độ, ta thay đổi điện áp cung cấp cho rotor. Việc cấp áp 1 chiều thay đổi thường khó khăn, do vậy người ta dùng phương pháp điều xung (PWM).

3.3.1 Giới thiệu về PWM

PWM là viết tắt của **Pulse Width Modulation** (Điều chế độ rộng xung) và nó là một kỹ thuật phổ biến được sử dụng để thay đổi độ rộng của các xung trong một chuỗi xung. PWM có nhiều ứng dụng như điều khiển độ sáng của đèn LED, điều khiển tốc độ của động cơ DC, điều khiển động cơ servo hoặc nơi bạn phải lấy ngõ ra analog bằng các thiết bị kỹ thuật số.

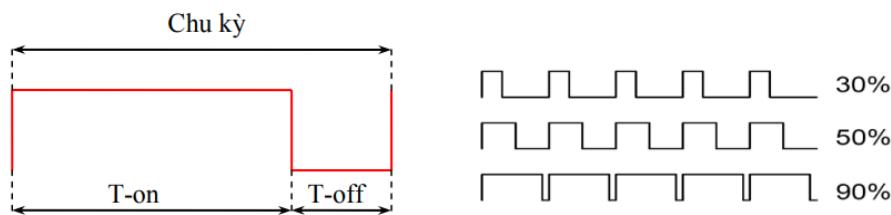
3.3.2 Phương pháp điều chế PWM

Tốc độ quay của động cơ DC tỷ lệ thuận với điện áp đặt vào nó. Do đó, cách đơn giản nhất để điều khiển tốc độ quay của rotor là thay đổi mức điện áp đặt vào động cơ. PWM là một phương pháp rất hiệu quả trong việc cung cấp ngay lập tức điện áp giữa mức cao và mức thấp

của nguồn điện giúp động cơ thay đổi tốc độ mượt hơn so với phương pháp cổ điển. Với công tắc đơn giản và một bộ nguồn thông dụng, chúng ta chỉ có thể cung cấp điện áp lớn nhất của bộ nguồn khi đóng công tắc, nghĩa là động cơ sẽ chạy với vận tốc tối đa. Và ngược lại khi hở công tắc động cơ sẽ tắt hẳn.

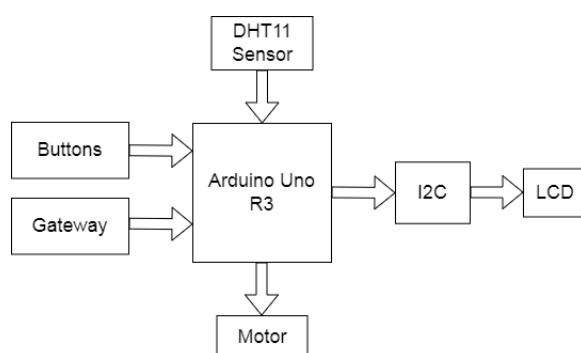
Nguyên tắc cơ bản của phương pháp điều khiển PWM là giữ nguyên giá trị điện áp và thay đổi thời gian đặt điện áp vào động cơ. Điều này có nghĩa, với tần số đóng ngắt công tắc đủ lớn, thời gian cấp điện áp vào động cơ càng lâu thì điện áp trung bình càng cao, ngược lại thời gian cấp điện vào động cơ ngắn điện áp trung bình giảm. Như vậy, PWM là một kỹ thuật so sánh tỷ lệ phần trăm điện áp nguồn bằng cách đóng ngắt nhanh nguồn điện cấp vào động cơ tạo ra một tín hiệu xung, với độ rộng xung (thời gian cấp điện áp) xác định sẽ tạo ra một điện áp trung bình xác định (được minh họa như **Hình 33**). Khi tần số đóng ngắt đủ lớn (thường từ $1 \rightarrow 20\text{kHz}$), động cơ sẽ chạy với một tốc độ ổn định nhờ moment quay. Đại lượng mô tả mối quan hệ giữa khoảng thời gian T_{ON} và T_{OFF} được gọi là độ rộng xung (duty cycle):

$$\text{Duty cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}} \times 100\%$$



Hình 33: Giải thích thời gian xung PWM

3.4 Kiến trúc hệ thống

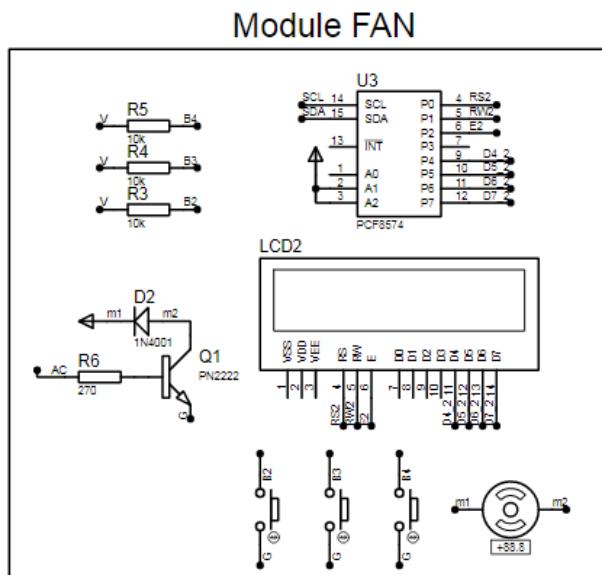


Hình 34: Sơ đồ kiến trúc hệ thống của module FAN

Module sẽ gồm các thành phần: Mạch Arduino Uno R3, Cảm biến DHT11, mạch chuyển giao tiếp I2C, LCD 16×2 , 3 nút nhấn và một gateway để giao tiếp với server.

Mạch Arduino sẽ có nhiệm vụ đọc tín hiệu từ nút nhấn, nhận lệnh điều khiển từ gateway, đọc giá trị nhiệt độ từ cảm biến DHT11 (Chỉ có ở mode auto) để điều khiển tốc độ quay của motor và hiển thị thông tin của quạt ra màn hình LCD.

3.5 Schematic của module trên proteus



Hình 35: Schematic của module FAN trên proteus

Các nút nhấn trong toàn bộ hệ thống điều sử dụng một điện trở để kéo lên. Trong module này sử dụng 3 nút nhấn **Button 2 (B2)**, **Button 3 (B3)** và **Button 4 (B4)** lần lượt được nối với các chân **9, 10, 11** của Arduino để nhận tín hiệu từ nút nhấn.

Việc điều khiển động cơ trong project này chủ yếu để tận dụng sức mạnh điều xung của Arduino nên mạch điều khiển động cơ của nhóm em chỉ đơn giản là dùng một người ngoài để cấp nguồn cho động cơ và điều xung thông qua một transistor để thay đổi tốc độ của động cơ. Chân điều khiển động cơ (AC) sẽ được nối với chân **5** (Có hỗ trợ PWM) của Arduino.

Chân SCL, SDA của mạch I2C này cũng được nối vào chân SCL và SDA của Arduino, địa chỉ của LCD trong module này là **0x26** nên chân A1, A2 sẽ được nối lên điện áp cao.

3.6 Mô tả hệ thống

Nhìn **Hình 34** ở trên ta thấy hệ thống của ta gồm các thiết bị ứng với chức năng như sau:

- **Button 2:** Bật / tắt quạt.
- **Button 3:** Thay đổi tốc độ quạt.
- **Button 4:** Chuyển sang mode tự động (AUTO).
- **Màn hình LCD:** Hiển thị thông tin về trạng thái của quạt.
- **Motor DC:** Mô phỏng động cơ của quạt.

Hệ thống có thể nhận được tín hiệu từ Web server **Adafruit IO** thông qua một **Gateway IoT** hiện thực bằng ngôn ngữ Python, với các tín hiệu **FAN_ON**, **FAN_INC**, **FAN_AUTO** tương ứng khi ấn các nút **Button 2**, **Button 3**, **Button 4**.

Mặc định hệ thống sẽ ở trạng thái tắt. khi ấn nút **POWER (Button 2)** thì hệ thống sẽ được bật. khi ấn lần nữa thì hệ thống sẽ tắt. Tương tự nếu ấn tiếp tục.

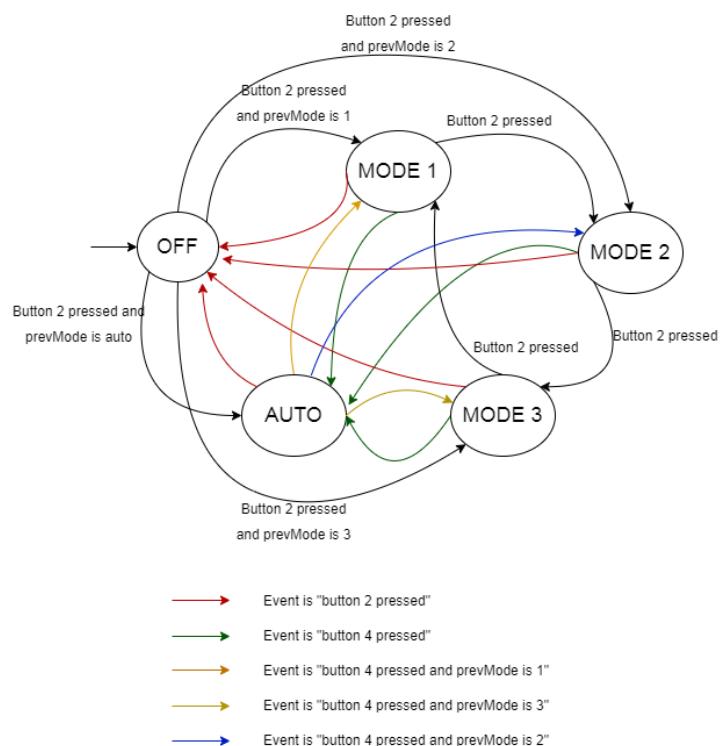
Để điều chỉnh tốc độ quạt thì ấn nút **INCREASE (Button 3)** để chuyển tốc độ nhanh dần. Hệ thống sẽ có 3 tốc độ: chậm, nhanh vừa và nhanh. Khi ấn nút Increase thì tốc độ sẽ tăng dần từ 1 đến 3 khi ở tốc độ số 3 nếu tiếp tục chuyển tốc độ thì hệ thống sẽ quay về tốc độ 1.

Hệ thống còn có một chế độ tự động, để chuyển sang chế độ tự động thì chúng ta nhấn nút **AUTO (Button 4)**. Khi ở chế độ tự động hệ thống sẽ dựa vào nhiệt độ trong phòng (định kì 10s sẽ cập nhật lại), đọc được từ cảm biến DHT11, để tự điều chỉnh tốc độ quạt cho phù hợp. Với sự điều chỉnh tốc độ quạt như sau:

- Khi nhiệt độ phòng nhỏ hơn $27^{\circ}C$ thì quạt sẽ không chạy.
- Khi nhiệt độ phòng trong khoảng $27 - 40^{\circ}C$ thì tốc độ quạt sẽ tăng tuyến tính theo nhiệt độ để đạt tốc độ maximum ở $40^{\circ}C$.
- Khi nhiệt độ phòng trên $40^{\circ}C$ thì quạt sẽ quay ở tốc độ tối đa.

Ngoài ra, hệ thống còn có hỗ trợ nhớ mode mà người dùng đã sử dụng trước khi tắt, để khi bật lại thì người dùng sẽ có một tốc độ quạt như ban đầu.

3.7 Máy trạng thái



Hình 36: Sơ đồ máy trạng thái của hệ thống

Ghi chú: prevMode là mode đang hoạt động trước khi quạt bị tắt nếu luồng sự kiện bắt đầu từ OFF, hoặc trước khi quạt chuyển sang auto nếu luồng sự kiện bắt đầu từ AUTO.



3.8 Hiện thực hệ thống

3.8.1 Hiện thực module FAN

```
1 #include "FAN.h"
2 #include "Arduino.h"
3 #include "_LCD.h"
4 #include "common.h"
5 #include "scheduler.h"
6
7 #define NO_OF_MODE 5
8
9 // Save state of FAN module
10 int mode = 0;
11 // Save state when turn off or change to auto mode
12 int prevMode = 1;
13 // Save speed value of motor
14 int FANSpeed = 0;
15 // Save ID of task task_FAN_update to delete it when turn off ←
16 // fan or out of auto mode
16 int ID_task_FAN_auto;
17
18 // use in auto mode to control speed of motor
19 extern float temperature;
20
21 // Initialization of module
22 void init_FAN(){
23     pinMode(FAN_PIN, OUTPUT);
24     // Display to lcd
25     task_FAN_display();
26     // Update speed of motor
27     task_FAN_update();
28 }
29
30 // Turn ON/OFF fan when button power(2) is pressed
31 void task_FAN_ON_OFF(){
32     if(mode > 0) {
33         if(mode == 4) SCH_Delete_Task(ID_task_FAN_auto);
34         prevMode = mode;
35         mode = 0;
36     }
37     else if(mode == 0) mode = prevMode;
38     task_FAN_display();
39     task_FAN_update();
40 }
41
42 // Change mode when button mode(3) is pressed
43 void task_FAN_increaseMode(){
44     if(mode != 0 && mode != 4){
```



```
45     mode++;
46     if(mode > NO_OF_MODE - 2){
47         mode = 1;
48     }
49     prevMode = mode;
50     task_FAN_display();
51     task_FAN_update();
52 }
53 }
54
55 // Change to auto mode when button auto(4) is pressed
56 void task_FAN_auto(){
57     if(mode > 0){
58         if(mode != 4){
59             mode = 4;
60             ID_task_FAN_auto = SCH_Add_Task(task_FAN_update, 20, ←
61                                         PERIOD_FAN_AUTO_UPDATE);
62         }
63         else{
64             if(prevMode != 4)mode = prevMode;
65             else mode = 1;
66             SCH_Delete_Task(ID_task_FAN_auto);
67         }
68         task_FAN_display();
69     }
70 }
71
72 // Update speed of motor base on mode value
73 void task_FAN_update(){
74     switch(mode){
75         case 0: // OFF
76             FANSpeed = 0;
77             break;
78         case 1: // MODE 0
79             FANSpeed = MODE_0_SPEED;
80             break;
81         case 2: // MODE 1
82             FANSpeed = MODE_1_SPEED;
83             break;
84         case 3: // MODE 2
85             FANSpeed = MODE_2_SPEED;
86             break;
87         case 4: // MODE AUTO
88             if(temperature < MIN_FAN_TEMPERATURE) FANSpeed = 0;
89             else if (temperature > MAX_FAN_TEMPERATURE) FANSpeed = ←
90                 255;
91             else FANSpeed = 255 - (215 * (temperature - ←
92                             MIN_FAN_TEMPERATURE)/(MAX_FAN_TEMPERATURE - ←
93                             MIN_FAN_TEMPERATURE));
```



```
90         break;
91     default:
92         break;
93     }
94     analogWrite(FAN_PIN, FANSpeed);
95 }
```

Code 6: Hiện thực module FAN

3.8.2 Hiển thị thông tin trạng thái của quạt ra LCD

Trong module LCD:

- Tạo ra LCD thứ 2:

```
1 LiquidCrystal_I2C lcd2(0x26, 16, 2);
```

Code 7: Tao ra LCD thứ 2

- Extern biến mode từ module FAN để hiển thị theo trạng thái của module FAN

```
1 extern int mode;
```

Code 8: Thêm biến hiển thị trạng thái của module FAN

- Thêm các hàm khởi tạo LCD vào trong hàm `init_LCD()`:

```
1 lcd2.begin();
2 lcd2.backlight();
```

Code 9: Hàm khởi tạo LCD2

- Thêm hàm `task_FAN_display()` để hiển thị thông tin quạt ra màn hình LCD 2.

```
1 void task_FAN_display(){
2     lcd2.clear();
3     lcd2.setCursor(0,0);
4     switch(mode){
5         case 0:
6             lcd2.print("FAN: OFF");
7             break;
8         case 1:
9             lcd2.print("FAN: MODE 1");
10            break;
11        case 2:
12            lcd2.print("FAN: MODE 2");
13            break;
14    }
15 }
```



```
14     case 3:  
15         lcd2.print("FAN: MODE 3");  
16         break;  
17     case 4:  
18         lcd2.print("FAN: AUTO");  
19         break;  
20     default:  
21         break;  
22     }  
23 }
```

Code 10: Triển khai hàm task_FAN_display()

3.9 Demo

Video demo của module: [Link video](#)

4 Hệ thống quản lý việc ra vào phòng bằng thẻ RFID (Radio Frequency Identification)

4.1 Giới thiệu

Ngày nay với sự phát triển của xã hội hiện đại ai trong số chúng ta cũng cần những thiết bị bảo vệ tài sản trong nhà như khoá cửa, thiết bị cảnh báo chống trộm hay camera nhưng có lẽ thiết bị được sử dụng nhiều nhất vẫn chính là khoá cửa. Hiện nay trên thị trường có rất nhiều loại khoá cửa nhưng hầu hết là khoá cơ khí, các khoá cơ khí này gặp vấn đề lớn đó là tính bảo mật của các loại khoá này là không cao, nên dễ dàng bị phá bởi các chìa khoá đa năng.

Đa số khoá kỹ thuật số đang có bán trên thị trường đều có giá bán khá cao và chủ yếu là loại khoá tay nắm ta thường thấy trong khách sạn hoặc các căn hộ chung cư. Khoá sử dụng phương pháp cài đặt mã số để khoá hoặc mở và người sử dụng có thể cài đặt số bất kỳ. Hệ thống số của khoá được thiết kế bằng các phím bấm số nên khá tiện lợi khi sử dụng. Bên cạnh loại chỉ có một chức năng khoá bằng mã số, còn có loại kèm theo chức năng khoá bằng thẻ. Nếu như bạn trót quên mã số thì có thể dùng thẻ để mở khoá.

Trong phạm vi của đồ án ta sẽ không làm hệ thống quản lý ra vào phòng bằng phím bấm số thay vào đó là làm chức năng quản lý bằng thẻ, mà cụ thể là ta sẽ sử dụng thẻ RFID (Radio Frequency Identification) như sau đây.

4.2 Yêu cầu đặt ra

Để tăng tính bảo mật cho việc quản lý ra vào phòng, hệ thống của ta bắt buộc phải có thêm yếu tố thời gian để tránh việc những người tấn công (attacker) họ quét gần các ID khi vào phòng. Cụ thể các yêu cầu được đặt ra như sau:

- Hệ thống sử dụng thẻ RFID để xác thực
- Hệ thống nhận diện được thẻ admin. Thẻ admin có toàn quyền trong việc quản lý, cụ thể bao gồm ra vào phòng, thêm một thẻ mới vào hệ thống.

- Hệ thống có khả năng báo cho người dùng về tình trạng nhập ID của mình. Cụ thể là hai trạng thái: **Hợp lệ (Valid)** và **Không hợp lệ (Invalid)**.
- Thời gian nhập ID là **10s**. Trong vòng **10s**, nếu người dùng nhập không đúng ID sẽ tự động quay về trạng thái **Locking (Khóa)**. Trong trường hợp người dùng nhập đúng ID thì sau **3s** hệ thống sẽ tự động quay về trạng thái **Locking (Khóa)**.
- Trong trường hợp thẻ đó là thẻ admin, khi thêm một thẻ ID mới, nếu trong vòng **3s** admin không nhập xong thẻ ID thì coi như việc thêm thẻ là bỏ và hệ thống sẽ quay về trạng thái **Locking (Khóa)**.
- Trong phạm vi mô phỏng, ta sẽ dùng **Virtual Terminal** của Proteus để minh họa dữ liệu từ RFID.

4.3 Giới thiệu về Relay (Rờ le)

4.3.1 Giới thiệu sơ lược

Relay là một công tắc chuyển đổi hoạt động bằng điện. Relay ở trạng thái ON hay OFF phụ thuộc vào dòng điện có đi qua Relay hay không.

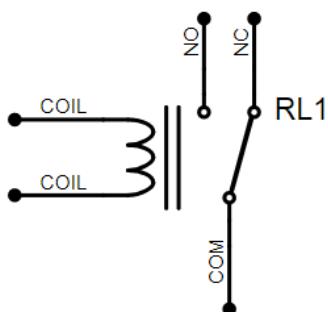


Hình 37: Hình ảnh thực tế của Relay 5V

4.3.2 Thông số kỹ thuật

Thông số	Giá trị
Điện áp kích hoạt	5V
Dòng điện kích hoạt	70mA
Dòng điện một chiều tối đa	10A @ 250/125V AC
Dòng điện xoay chiều tối đa	10A @ 30/28V DC
Thời gian xử lý	10ms
Thời gian nghỉ	5ms
Số lần chuyển mạch ON/OFF lớn nhất (cơ khí)	300 xử lý / phút
Số lần chuyển mạch ON/OFF lớn nhất (diện)	30 xử lý / phút

Bảng 21: Thông số kỹ thuật của Relay



Hình 38: Các chân của Relay

4.3.3 Sơ đồ chân

Ta có bảng thể hiện tên và chức năng của từng chân của Relay như sau:

Chân số	Tên gọi	Mô tả
1	Coil End 1	Được sử dụng để kích hoạt (Bật/Tắt) Relay. Thông thường một đầu được kết nối với 5V và đầu kia nối đất (Ground).
2	Coil End 2	
3	Common (COM)	Cổng chung (COM) được kết nối với một đầu của tải (load) sẽ được điều khiển.
4	Normally Close (NC)	Đầu kia của tải hoặc được kết nối với NO hoặc NC. Nếu được kết nối với NC, tải vẫn được kết nối trước khi kích hoạt.
5	Normally Open (NO)	Đầu kia của tải hoặc được kết nối với NO hoặc NC. Nếu kết nối với NO, tải vẫn bị ngắt kết nối trước khi kích hoạt.

Bảng 22: Tên và chức năng của từng chân của Relay

4.3.4 Nguyên lý làm việc

Khi có dòng điện đi qua Relay, dòng điện này sẽ đi qua cuộn dây bên trong và tạo một từ trường hút. Từ trường này tác động lên một đòn bẩy bên trong làm đóng hoặc mở các điểm tiếp điện và như vậy sẽ làm thay đổi trạng thái của Relay. Số tiếp điểm điện có thể là 1 hoặc nhiều, tùy vào thiết kế.

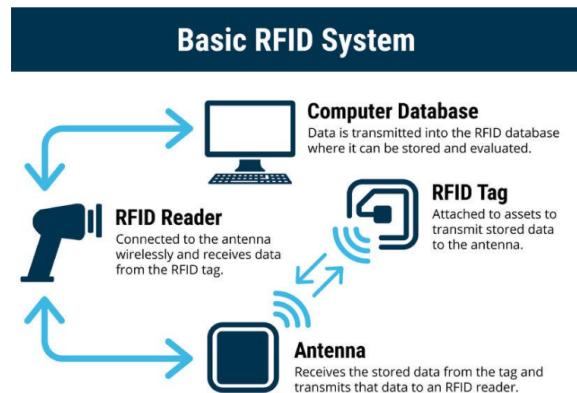
4.4 Giới thiệu về RFID

4.4.1 Giới thiệu sơ lược về công nghệ RFID

Công nghệ RFID (Radio Frequency Identification) là công nghệ nhận dạng đối tượng bằng sóng vô tuyến, cho phép một thiết bị đọc thông tin chứa trong chip ở khoảng cách xa, không cần tiếp xúc trực tiếp, không thực hiện bất kỳ giao tiếp vật lý nào giữa hai vật không nhìn thấy. Công nghệ RFID cho ta phương pháp truyền, nhận dữ liệu từ một điểm đến một điểm khác.

Kỹ thuật RFID sử dụng truyền thông không dây trong dải tần sóng vô tuyến để truyền dữ liệu từ các tag (thẻ) đến các reader (bộ đọc). Tag có thể được đính kèm hoặc gắn vào đối tượng được nhận dạng chẳng hạn sản phẩm, hộp hoặc giá kệ. Bộ đọc quét dữ liệu của thẻ (tag) và gửi thông tin đến cơ sở dữ liệu có lưu trữ dữ liệu của thẻ (tag). Chẳng hạn, các thẻ (tag) có thể được đặt trên kính chắn gió xe hơi để hệ thống thu phí đường có thể nhanh chóng nhận dạng và thu tiền trên các tuyến đường.

4.4.2 Các thành phần chính của hệ thống RFID

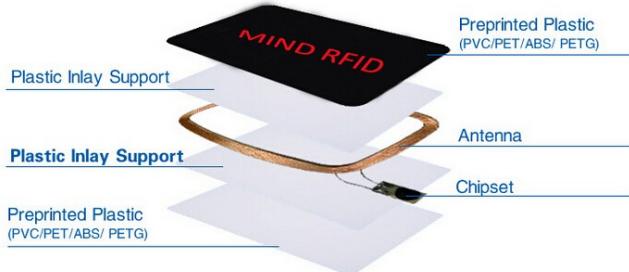


Hình 39: Các thành phần của hệ thống RFID

4.4.2.1 Thẻ RFID

Tag (thẻ) RFID là một thiết bị có thể lưu trữ và truyền dữ liệu đến một reader trong một môi trường không tiếp xúc bằng sóng vô tuyến. Tag RFID mang dữ liệu về một vật, một sản phẩm (item) nào đó và gắn lên sản phẩm đó. Mỗi tag có các bộ phận lưu trữ dữ liệu bên trong và cách giao tiếp với dữ liệu đó.

Thông thường mỗi tag RFID có một cuộn dây hoặc antenna nhưng không phải tất cả đều có vi chip và nguồn năng lượng riêng.



Hình 40: Cấu tạo của thẻ RFID

Các tag RFID có thể được phân loại theo hai phương pháp khác nhau. Danh sách sau trình bày việc phân loại thứ nhất, dựa trên việc tag có chứa nguồn cung cấp gắn bên trong hay là được cung cấp bởi reader:

Thụ động (Passive)

Loại tag này không có nguồn bên trong, sử dụng nguồn nhận được từ reader để hoạt động và truyền dữ liệu được lưu trữ trong nó cho reader. Tag thụ động có cấu trúc đơn giản và không có các thành phần động. Tag như thế có một thời gian sống dài và thường có sức chịu đựng với điều kiện môi trường khắc nghiệt.

Chẳng hạn, một số tag thụ động có thể chịu đựng các hóa chất găm mòn như acid, nhiệt độ lên tới $400^{\circ}F$ (xấp xỉ $204^{\circ}C$) và nhiệt độ cao hơn nữa. Đối với loại tag này, khi tag và reader truyền thông với nhau thì reader luôn truyền trước rồi mới đến tag. Cho nên bắt buộc phải có reader để tag có thể truyền dữ liệu của nó.



Tag thụ động nhỏ hơn tag tích cực hoặc tag bán tích cực. Nó có nhiều phạm vi đọc, ít hơn 1 inch đến khoảng 30 feet (xấp xỉ 9m). Tag thụ động cũng rẻ hơn tag tích cực hoặc bán tích cực. Thẻ thông minh (smart card) là một loại tag RFID thụ động, ngày nay nó được sử dụng rộng rãi trong các lĩnh vực khác nhau (chẳng hạn như huy hiệu ID). Dữ liệu trên tag này được đọc khi nó gần reader. Tag này không cần phải tiếp xúc với reader trong quá trình đọc.

Tag thụ động bao gồm những thành phần chính sau:

- Vi mạch (microchip)
- Antenna

Tích cực (Active)

Tag tích cực có một nguồn năng lượng bên trong (chẳng hạn một bộ pin hoặc có thể là những nguồn năng lượng khác như sử dụng nguồn năng lượng mặt trời) và điện tử học để thực thi những nhiệm vụ chuyên dụng. Tag tích cực sử dụng nguồn năng lượng bên trong để truyền dữ liệu cho reader. Nó không cần nguồn năng lượng từ reader để truyền dữ liệu. Thành phần bên trong gồm bộ vi mạch, cảm biến và các cổng vào/ra được cấp nguồn bởi nguồn năng lượng bên trong nó. Vì vậy, những thành phần này có thể đo được nhiệt độ xung quanh và phát ra dữ liệu nhiệt độ chuẩn. Những thành phần này có thể sử dụng dữ liệu này để xác định các tham số khác như hạn sử dụng của item được gắn tag. Tag có thể truyền thông tin này cho reader (cùng với từ định danh duy nhất của nó). Ta có thể xem tag tích cực như một máy tính không dây với những đặc tính thêm vào (chẳng hạn như một cảm biến hoặc một bộ cảm biến).

Đối với loại tag này, trong quá trình truyền giữa tag và reader, tag luôn truyền trước, rồi mới đến reader. Vì sự hiện diện của reader không cần thiết cho việc truyền dữ liệu nên tag tích cực có thể phát dữ liệu của nó cho những vùng lân cận nó thậm chí trong cả trường hợp reader không có ở nơi đó. Loại tag tích cực này (truyền dữ liệu liên tục khi có cũng như không có reader hiện diện) cũng được gọi là máy phát (transmitter).

Loại tag tích cực khác ở trạng thái ngủ hoặc nguồn yếu khi không có reader. Reader đánh thức tag này khỏi trạng thái ngủ bằng cách phát một lệnh thích hợp. Trạng thái này tiết kiệm nguồn năng lượng, vì vậy loại tag này có thời gian sống dài hơn tag tích cực được gọi là máy phát kể trên. Thêm nữa là vì tag chỉ truyền khi được thảm vấn nên số nhiễu RF trong môi trường cũng bị giảm xuống. Loại tag tích cực này được gọi là một máy phát/máy thu hoặc một bộ tách sóng-tag có thể hoạt động ở chế độ máy phát và máy thu. Tag này chỉ truyền khi được reader thảm vấn. Tag ở trạng thái ngủ hoặc nguồn giảm khi không được reader thảm vấn. Vì vậy tất cả tag này có thể được gọi là transponder. Khoảng cách đọc của tag tích cực là 100 feet (xấp xỉ 30.5m) hoặc hơn nữa khi máy phát tích cực của loại tag này được dùng đến.

Tag tích cực bao gồm những thành phần chính sau:

- Vi mạch (microchip)
- Anten
- Cung cấp nguồn bên trong
- Điện tử học bên trong

Bán tích cực (Semi – active, cũng như bán thụ động semi – passive)

Tag bán tích cực có một nguồn năng lượng bên trong (chẳng hạn là bộ pin) và điện tử học bên trong để thực thi những nhiệm vụ chuyên dụng. Nguồn bên trong cung cấp sinh lực cho tag hoạt động. Tuy nhiên trong quá trình truyền dữ liệu, tag bán tích cực sử dụng nguồn từ reader. Tag bán tích cực được gọi là tag có hỗ trợ pin (battery-assisted tag).



Dối với loại tag này, trong quá trình truyền giữa tag và reader thì reader luôn truyền trước rồi đến tag. Tag bán tích cực cũng cho phép đọc tốt hơn ngay cả khi gắn tag bằng những vật liệu chấn tần số vô tuyến (RF-opaque và RF-absorbent). Sự có mặt của những vật liệu này có thể ngăn không cho tag thụ động hoạt động đúng dẫn đến việc truyền dữ liệu không thành công. Tuy nhiên, đây không phải là vấn đề khó khăn đối với tag bán tích cực.

Phạm vi đọc của tag bán tích cực có thể lên đến 100 feet (xấp xỉ 30.5m) với điều kiện lý tưởng bằng cách sử dụng mô hình tán xạ đã được điều chế (modulated back scatter) trong UHF và sóng viba.

Việc phân loại tag còn dựa trên khả năng hỗ trợ ghi chép dữ liệu:

- Chỉ đọc (Read only)
- Ghi 1 lần, đọc nhiều lần (Write once, read many)
- Đọc – Ghi (Read – Write)

Ngoài ra còn có một số loại tag khác như là:

- Tag SAW (Surface Acoustic Wave)
- Tag Non – RFID
- Tag một bit EAS (Electronic Article Surveillance – Giám sát điện tử)

4.4.2.2 Các reader (đầu đọc) hoặc sensor (cảm biến) để truy vấn các thẻ.

Reader RFID được gọi là vật tra hỏi (interrogator), là một thiết bị đọc và ghi dữ liệu lên tag RFID tương thích. Hoạt động ghi dữ liệu lên tag bằng reader được gọi là tạo tag. Quá trình tạo tag và kết hợp tag với một đối tượng được gọi là đưa tag vào hoạt động (commissioning the tag). Decommissioning tag có nghĩa là tách tag ra khỏi đối tượng được gắn tag và tùy ý làm mất hiệu lực hoạt động của tag. Thời gian mà reader có thể phát năng lượng RF để đọc tag được gọi là chu kỳ làm việc của reader.

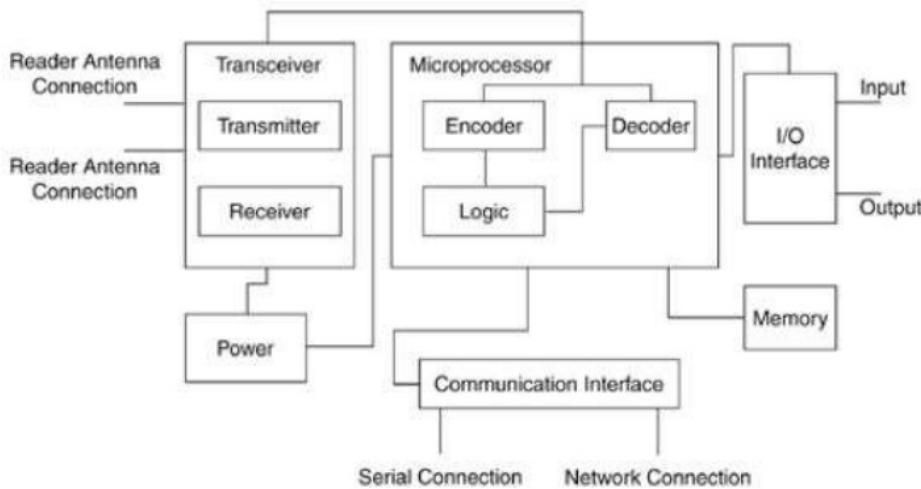
Reader là hệ thàn kinh trung ương của toàn hệ thống phần cứng RFID thiết lập việc truyền với thành phần này và điều khiển nó, là thao tác quan trọng nhất của bất kỳ thực thể nào muốn liên kết với thực thể phần cứng này.

Tag thụ động (passive tag) được kích thích nguồn năng lượng bằng quá trình truyền sóng radio và bộ phận thu sẽ lắng nghe quá trình truyền này. Các tag tích cực cũng cần có giao tiếp với bộ phận thu được gắn vào hệ thống.

Trong quy trình RFID, điểm cuối của thiết bị truyền/hệ thống được gọi là bộ đọc (reader). Reader được đặt giữa tag và bộ lọc sự kiện (event filter) trong một hệ thống RFID. Reader đóng vai trò giao tiếp với tag, tạo ra các sự kiện mức năng lượng thấp từ quá trình đọc và gửi những sự kiện này đến bộ lọc sự kiện.

Một reader có các thành phần chính sau:

- **Máy phát (Transmitter):** Máy phát của reader truyền nguồn AC và chu kỳ xung đồng hồ qua anten của nó đến tag trong phạm vi đọc cho phép. Đây là một phần của máy thu phát, thành phần chịu trách nhiệm gửi tín hiệu của reader đến môi trường xung quanh và nhận lại đáp ứng của tag qua anten của reader. Antenna của reader được kết nối với thành phần thu phát của nó. Anten của reader có thể được gắn với mỗi cổng antenna. Hiện tại thì một số reader có thể hỗ trợ đến 4 cổng anten.



Hình 41: Các thành phần của một Reader

- **Máy thu (Receiver):** Thành phần này cũng là một phần của máy thu phát. Nó nhận tín hiệu tương tự từ tag qua anten của reader. Sau đó nó gửi những tín hiệu này cho vi mạch của reader, tại nơi này nó được chuyển thành tín hiệu số tương đương (có nghĩa là dữ liệu mà tag đã truyền cho reader được biểu diễn ở dạng số).
- **Vi mạch (Microprocessor):** Thành phần này chịu trách nhiệm cung cấp giao thức cho reader để nó truyền thông với tag tương thích với nó. Nó thực hiện việc giải mã và kiểm tra lỗi tín hiệu tương tự nhận từ máy thu.Thêm nữa là vi mạch có thể chứa luận lý để thực hiện việc lọc và xử lý dữ liệu đọc được từ tag.
- **Bộ nhớ (Memory):** Bộ nhớ dùng lưu trữ dữ liệu như các tham số cấu hình reader và một bản kê khai các lần đọc tag. Vì vậy nếu việc kết nối giữa reader và hệ thống mạch điều khiển/phần mềm bị hỏng thì tất cả dữ liệu tag đã được đọc không bị mất. Tuy nhiên, dung lượng của bộ nhớ sẽ giới hạn số lượng tag đọc được trong một khoảng thời gian. Nếu trong quá trình đọc mà việc kết nối bị hỏng thì một phần dữ liệu đã lưu sẽ bị mất (có nghĩa là bị ghi đè bởi các tag khác được đọc sau đó).
- **Kênh vào/ra đối với các cảm biến, cơ cấu chấp hành, bảng tín hiệu điện báo bên ngoài (I/O Interface):** Các reader không cần bật suốt. Các tag có thể chỉ xuất hiện lúc nào đó và rời khỏi reader mãi mãi cho nên việc bật reader suốt sẽ gây lãng phí năng lượng. Thêm nữa là giới hạn vừa đề cập ở trên cũng ảnh hưởng đến chu kỳ làm việc của reader. Thành phần này cung cấp một cơ chế bật và tắt reader tùy thuộc vào các sự kiện bên ngoài. Có một số loại cảm biến như cảm biến về ánh sáng hoặc chuyển động để phát hiện các đối tượng được gắn tag trong phạm vi đọc của reader. Cảm biến này cho phép reader bật lên để đọc tag. Thành phần cảm biến này cũng cho phép reader xuất tín hiệu điều khiển cục bộ tùy thuộc vào một số điều kiện qua một bảng tín hiệu điện báo (chẳng hạn báo bằng âm thanh) hoặc cơ cấu chấp hành (ví dụ mở hoặc đóng van an toàn, di chuyển một cánh tay robot, ...).
- **Mạch điều khiển (có thể nó được đặt ở bên ngoài):** Mạch điều khiển là một thực thể cho phép thành phần bên ngoài là con người hoặc chương trình máy tính giao tiếp, điều khiển các chức năng của reader, điều khiển bảng tín hiệu điện báo và cơ cấu chấp hành kết hợp với reader này. Thường thì các nhà sản xuất hợp nhất thành phần này vào reader.

(như phần mềm hệ thống (firmware) chẳng hạn). Tuy nhiên, có thể đóng gói nó thành một thành phần phần cứng/phần mềm riêng phải mua chung với reader.

- **Mạch truyền thông:** Thành phần giao diện truyền thông cung cấp các lệnh truyền đến reader, nó cho phép tương tác với các thành phần bên ngoài qua mạch điều khiển, để truyền dữ liệu của nó, nhận lệnh và gửi lại đáp ứng. Thành phần giao diện này cũng có thể xem là một phần của mạch điều khiển hoặc là phương tiện truyền giữa mạch điều khiển và các thực thể bên ngoài. Thực thể này có những đặc điểm quan trọng cần xem nó như một thành phần độc lập. Reader có thể có một giao diện tuần tự. Giao diện tuần tự là loại giao diện phổ biến nhất nhưng các reader thế hệ sau sẽ được phát triển giao diện mạng thành một tính năng chuẩn. Các reader phức tạp có các tính năng như tự phát hiện bằng chương trình ứng dụng, có gắn các Web server cho phép reader nhận lệnh và trình bày kết quả dùng một trình duyệt Web chuẩn, ...
- **Nguồn năng lượng (Power):** Thành phần này cung cấp nguồn năng lượng cho các thành phần của reader. Nguồn năng lượng được cung cấp cho các thành phần này qua một dây dẫn điện được kết nối với một ngõ ra bên ngoài thích hợp.

Dưới đây là một số đầu đọc (reader) RFID phổ biến:



Hình 42: Một số đầu đọc (reader) RFID phổ biến

4.4.2.3 Database

Là hệ thống thông tin phụ trợ để theo dõi và chứa thông tin về item có đính thẻ. Thông tin được lưu trong database bao gồm định danh item, phần mô tả nhà sản xuất hoạt động của item, vị trí. Kiểu thông tin chứa trong database sẽ biến đổi tùy theo ứng dụng. Các database cũng có thể kết nối đến các mạng khác như mạng LAN để kết nối database qua Internet. Việc kết nối này cho phép dữ liệu chia sẻ với một database cục bộ mà thông tin được thu thập trước tiên từ nó.

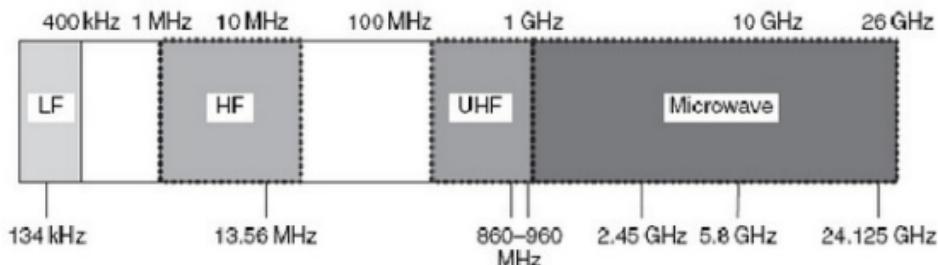
4.4.3 Phân loại RFID

Hệ thống RFID có thể phân loại dựa trên tần số hoạt động, nguồn cung cấp cho thẻ và giao thức truyền dữ liệu giữa thẻ và đầu đọc. Dựa trên phương pháp cấp nguồn cho thẻ thì có thể phân loại thành hệ thống RFID thụ động, tích cực. Dựa trên giao thức truyền dữ liệu giữa thẻ



và đầu đọc thì có thể phân thành "thẻ nói trước – tag talksfirst" và "đầu đọc nói trước – Reader talks first".

Dựa theo tần số hoạt động thì ta có hình ảnh sau thể hiện dải tần số hoạt động của một hệ thống RFID:



Hình 43: Các dải tần số hoạt động của RFID

Tần số làm việc là tần số của sóng điện từ phát ra từ các anten trên đầu đọc và thẻ. Tần số ảnh hưởng đến tốc độ truyền thông và khoảng cách giao tiếp giữa các anten. Tần số cao cho biết phạm vi đọc xa hơn, tốc độ nhanh nhưng tốn năng lượng hơn. Mỗi ứng dụng phù hợp với một kiểu tần số cụ thể do đặc điểm của sóng điện từ ở mỗi dải tần là khác nhau, ví dụ sóng có tần số thấp có khả năng truyền xuyên qua tường tốt hơn sóng có tần số cao hơn nó, nhưng tần số cao có tốc độ đọc nhanh. RFID sử dụng sóng từ 30 KHz đến 5,8GHz. Có thể thấy 4 dải tần chính được sử dụng ở các hệ thống RFID.

- Low Frequency: **125 – 134 KHz**, phù hợp với các ứng dụng phạm vi ngắn như hệ thống chống trộm, khóa tự động.
- High Frequency: Băng tần **13,65 MHz**, với phạm vi quét khoảng 1m, thích hợp cho các ứng dụng đọc các item như quản lý hiệu sách, theo dõi hành lý trên máy bay.
- Ultrahigh Frequency: Tần số khoảng từ **860 – 960 MHz**, cho phép khoảng cách đọc lên đến vài mét, thích ứng cho các ứng dụng dây chuyền vì tốc độ và phạm vi của nó, được ứng dụng trong quản lý hàng tại các nhà kho, các dây chuyền sản xuất tự động.
- Microware Frequency: Tần số trong khoảng **2.45 – 5.8 GHz**. Cho phép khoảng cách truyền xa hơn, lên đến vài chục mét.

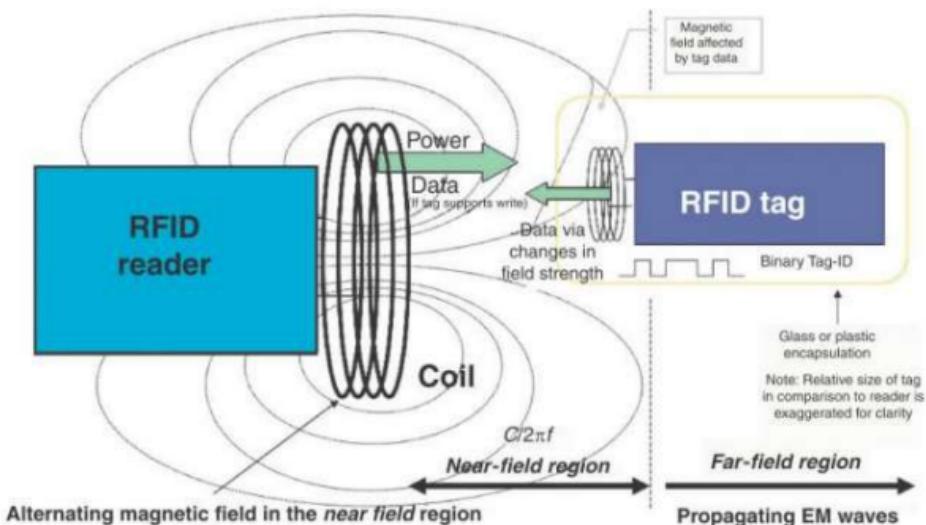
Việc phân chia các dải tần là tương đối, lưu ý là ở mỗi quốc gia, tần số hoạt động của hệ thống cần được sự cho phép của nhà nước. Ví dụ, đối với các thẻ RFID UHF, ở Hoa Kỳ, chúng làm việc từ **902 – 928 MHz**. Ở Ấn Độ chấp nhận phạm vi tần số từ **865 – 867 MHz**.

4.4.4 Phương thức làm việc của RFID

Một hệ thống RFID có ba thành phần cơ bản: thẻ, đầu đọc, và một host computer. Thẻ RFID gồm chip bán dẫn nhỏ và anten được thu nhỏ trong một số hình thức đóng gói. Vài thẻ RFID giống như những nhãn giấy và được ứng dụng để bỏ vào hộp và đóng gói. Một số khác được sáp nhập thành các vách của các thùng chứa plastic được đúc. Còn một số khác được xây dựng thành miếng da bao cổ tay. Mỗi thẻ được lập trình với một nhận dạng duy nhất cho phép theo dõi không dây đối tượng hoặc con người đang gắn thẻ đó. Bởi vì các chip được sử dụng trong thẻ RFID có thể giữ một số lượng lớn dữ liệu, chúng có thể chứa thông tin như chuỗi số, thời dấu, hướng dẫn cấu hình, dữ liệu kỹ thuật, sổ sách y học, và lịch trình. Cũng như phát sóng

tivi hay radio, hệ thốngs RFID cũng sử dụng bốn băng thông tần số chính: tần số thấp (LF), tần số cao (HF), siêu cao tần (UHF) hoặc sóng cực ngắn (viba). Các hệ thống trong siêu thị ngày nay hoạt động ở băng thông UHF, trong khi các hệ thống RFID cũ sử dụng băng thông LF và HF. Băng thông vibra đang được để dành cho các ứng dụng trong tương lai.

Các thẻ RFID có thể được cấp nguồn bởi một bộ pin thu nhỏ trong thẻ (các thẻ active) hoặc bởi một RFID reader mà nó “wake up” thẻ để yêu cầu trả lời khi thẻ đang trong phạm vi (thẻ passive).



Hình 44: Hoạt động giữa tag và reader RFID

Thẻ active RFID có thể được đọc xa 100 feet từ RFID reader và có thể là thẻ “thông minh” (với bộ nhớ được viết lên và xóa như một ổ cứng máy tính) hoặc là thẻ chỉ đọc. Thẻ passive RFID có thể được đọc xa RFID reader 20 feet và nói chung là bộ nhớ chỉ đọc. Kích thước thẻ và giá cả, dải đọc, độ chính xác đọc/ghi, tốc độ dữ liệu và chức năng hệ thống thay đổi theo đặc điểm nêu ra trong thiết kế và dải tần hệ thống FRID sử dụng.

RFID reader gồm một anten liên lạc với thẻ RFID và một đơn vị đo điện tử học đã được nối mạng với host computer. Đơn vị đo tiếp sóng giữa host computer và tất cả các thẻ trong phạm vi đọc của anten, cho phép một đầu đọc liên lạc với hàng trăm thẻ đồng thời. Nó cũng thực thi các chức năng bảo mật như mã hóa/ giải mã và xác thực người dùng. Đầu đọc RFID có thể phát hiện thẻ ngay cả khi không nhìn thấy chúng. Hầu hết các mạng RFID gồm nhiều thẻ và nhiều đầu đọc được nối mạng với nhau bởi một máy tính trung tâm, hầu như thường là một trạm làm việc gọn để bàn. Host xử lý dữ liệu mà các đầu đọc thu thập từ các thẻ và dịch nó giữa mạng RFID và các hệ thống kỹ thuật thông tin lớn hơn, mà nơi đó quản lý dây chuyền hoặc cơ sở dữ liệu quản lý có thể thực thi. “Middleware” phần mềm nối hệ thống RFID với một hệ thống IT (Information Technology) quản lý luồng dữ liệu.

4.5 Giới thiệu về chuẩn giao tiếp UART

4.5.1 Giới thiệu sơ lược

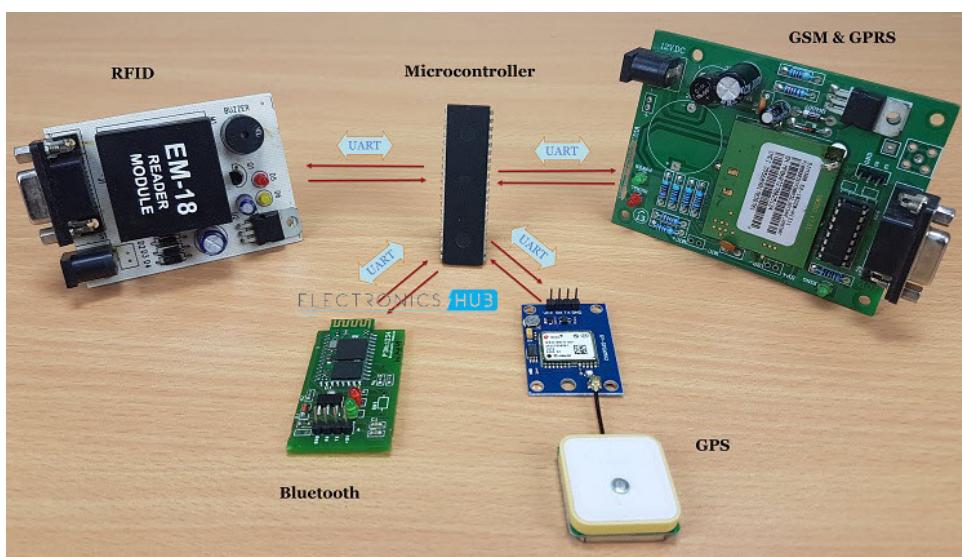
UART - Universal synchronous asynchronous receiver transmitter là một ngoại vi cơ bản và thường dùng trong các quá trình giao tiếp với các module như: Xbee, Wifi, Bluetooth,

RFID, Khi giao tiếp UART kết hợp với các IC giao tiếp như MAX232CP, SP485EEN, ... thì sẽ tạo thành các chuẩn giao tiếp RS232, RS485. Đây là các chuẩn giao tiếp thông dụng và phổ biến trong công nghiệp từ trước đến nay.

Ưu điểm của giao tiếp UART không đồng bộ: tiết kiệm chân vi điều khiển (2 chân), là ngoại vi mà bất kì 1 vi điều khiển nào cũng có, có khá nhiều module, cảm biến dùng UART để truyền nhận dữ liệu với vi điều khiển.

Nhược điểm của loại ngoại vi này là tốc độ khá chậm, tốc độ tối đa tùy thuộc vào từng dòng; quá trình truyền nhận dễ xảy ra lỗi nên trong quá trình truyền nhận cần có các phương pháp để kiểm tra (thông thường là truyền thêm bit hoặc byte kiểm tra lỗi).

Khi muốn truyền dữ liệu đi xa, chúng ta cần phải sử dụng các IC thông dụng để tạo thành các chuẩn giao tiếp đáng tin cậy như RS485 hay RS232.



Hình 45: Ứng dụng của giao tiếp UART

UART có chức năng chính là truyền dữ liệu nối tiếp. Trong UART, giao tiếp giữa hai thiết bị có thể được thực hiện theo hai phương thức là giao tiếp dữ liệu nối tiếp và giao tiếp dữ liệu song song. Giao tiếp dữ liệu nối tiếp có nghĩa là dữ liệu có thể được truyền qua một cáp hoặc một đường dây ở dạng bit-bit và nó chỉ cần hai cáp. Nó yêu cầu số lượng mạch hay dây rất ít. Giao tiếp này rất hữu ích trong các mạch ghép hơn giao tiếp song song. Giao tiếp dữ liệu song song có nghĩa là dữ liệu có thể được truyền qua nhiều cáp cùng một lúc. Truyền dữ liệu song song yêu số lượng mạch và dây nhiều. Vì vậy, giao tiếp song song tốn kém nhưng đổi lại rất nhanh, nó đòi hỏi phần cứng và cáp bổ sung.

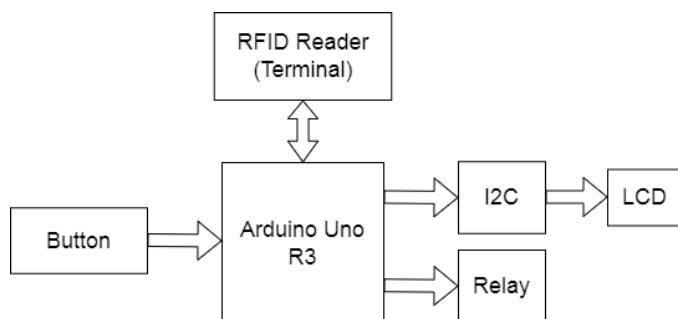
4.5.2 Các thông số cơ bản trong giao tiếp UART

- Baud rate (tốc độ Baud):** Khi truyền nhận không đồng bộ để hai modul hiểu được nhau thì cần quy định một khoảng thời gian cho 1 bit truyền nhận, nghĩa là trước khi truyền thì tốc độ phải được cài đặt đầu tiên. Theo định nghĩa thì tốc độ baud là số bit truyền trong một giây.
- Frame (khung truyền):** Do kiểu truyền thông nối tiếp này rất dễ mất dữ liệu nên ngoài tốc độ, khung truyền cũng được cài đặt từ ban đầu để tránh bớt sự mất mát dữ liệu này.

Khung truyền quy định số bit trong mỗi lần truyền, các bit báo như start, stop, các bit kiểm tra như parity, và số bit trong một data.

- **Bit Start:** Là bit bắt đầu trong khung truyền. Bit này nhằm mục đích báo cho thiết bị nhận biết quá trình truyền bắt đầu.
- **Data:** Dữ liệu cần truyền Data không nhất thiết phải 8 bit. Có thể là 5, 6, 7, 8, 9. Trong UART bit LSB được truyền đi trước, bit MSB được truyền đi sau.
- **Parity bit:** Là bit kiểm tra dữ liệu đúng không. Có 2 loại parity: chẵn (even parity), lẻ (odd parity). Parity chẵn nghĩa là số lượng số 1 trong dữ liệu bao gồm bit parity luôn là số chẵn. Ngược lại tổng số lượng các số 1 trong parity lẻ luôn là số lẻ. Bit Parity là không bắt buộc nên có thể dùng hoặc không.
- **Stop:** là bit báo cáo kết thúc khung truyền. Thường là mức 5V. Và có thể có 1 hoặc 2 bits stop.

4.6 Kiến trúc hệ thống

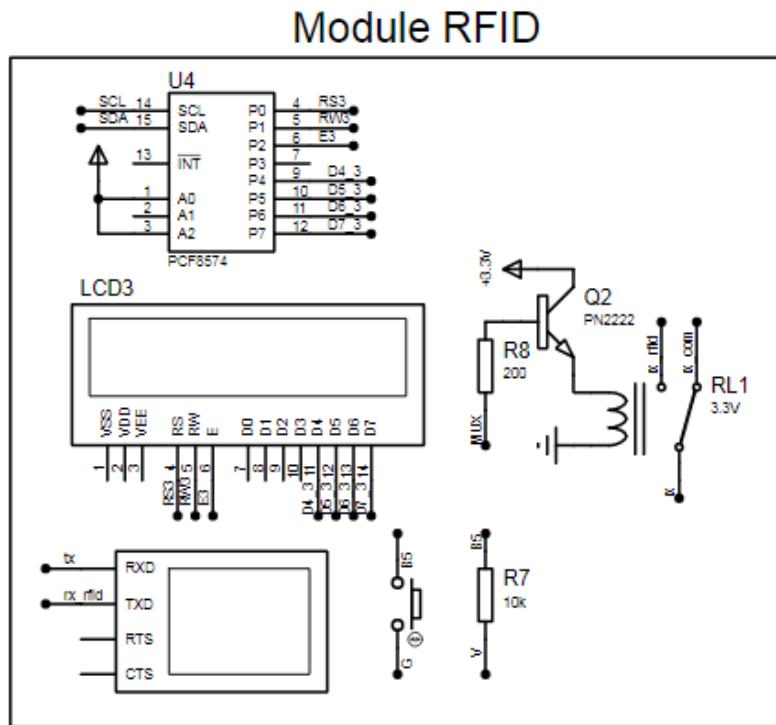


Hình 46: Sơ đồ kiến trúc của hệ thống

Module này gồm: Mạch Arduino Uno R3, mạch đọc thẻ RFID (Trong mô phỏng là terminal), một button, mạch chuyển giao tiếp I2C, màn hình LCD và relay.

Mạch Arduino sẽ có nhiệm vụ nhận tín hiệu từ nút nhấn để điều khiển relay kích hoạt module RFID hoạt động. Khi module hoạt động Arduino sẽ nhận tín hiệu từ nút nhấn cũng như terminal để thực hiện quét thẻ hoặc thêm thẻ mới và hiện thị trạng thái của module ra LCD và cả terminal.

4.7 Schematic của module trên Proteus



Hình 47: Schematic của module RFID trên proteus

Button trong module này cũng dùng một điện trở kéo lên, chân nhận tín hiệu từ button(B5) sẽ được nối với chân 7 của Arduino.

Cách lăm LCD như các module trước, trong module này LCD có địa chỉ là 0x25 nên chân A0 và A2 của mạch I2C sẽ được nối lên điện áp cao.

Relay được dùng để chuyển chân Rx của Arduino có thể giao tiếp qua lại giữa 2 module RFID và gateway, nên chân Rx sẽ được nối với chân chung (COM) của Relay. Bình thường khi khởi động hệ thống sẽ không kích hoạt module RFID hoạt động mà cho gateway hoạt động nên chân Rx của terminal sẽ được nối với chân thường mở(NO) của Relay còn chân Rx của cổng COM ảo sẽ được nối với chân thường đóng(NC) của Relay. Chân điều khiển Mux sẽ được nối vào chân 4 của Arduino.

4.8 Mô tả hệ thống

Để mô phỏng hoạt động quét thẻ RFID trên proteus thì nhóm em dùng một terminal, giao tiếp với Arduino qua UART, để nhập thông tin của thẻ, là một dãy 10 kí tự có thẻ có cả chữ và số, không chứa các kí tự không phải chữ và số.

Do giới hạn của Arduino Uno R3 chỉ có một giao tiếp UART, mà trong project lần này ta có sử dụng 2 module liên quan đến UART, nên chính vì vậy ta quyết định sử dụng một Relay để chuyển đổi chân Rx của Arduino qua lại giữa hai module là RFID và Gateway.

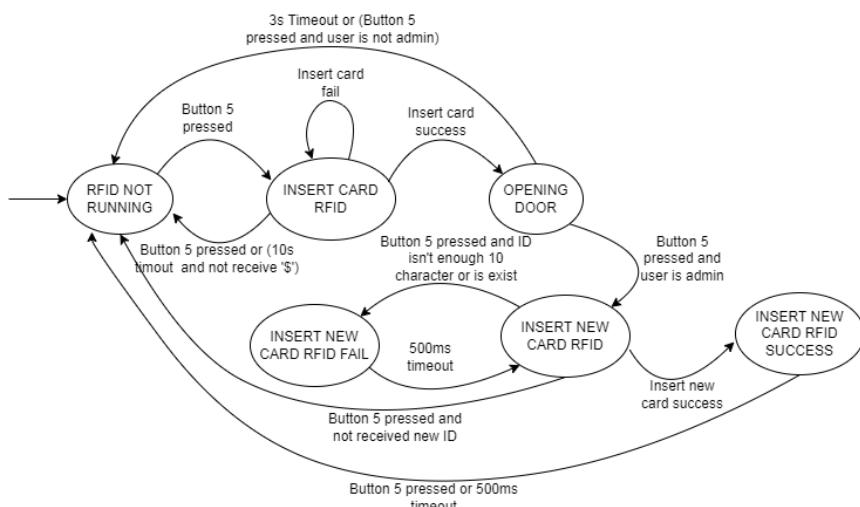
Ban đầu module RFID sẽ ở trạng thái không hoạt động, lúc này chân Rx của Arduino sẽ được sử dụng cho việc giao tiếp với Gateway. Trên LCD sẽ hiển thị Locking.

Để tiến hành quét thẻ RFID thì ta ấn Button 5. Lúc này LCD vẫn hiển thị Locking, trên terminal sẽ hiển thị Please insert card !!! và timeout sẽ được đặt là 10s.

Để nhập thông tin của thẻ, ta tiến hành nhập kí tự \$, để báo bắt đầu nhập thẻ, sau đó là một dãy 10 kí tự của thẻ RFID. Nếu sau 10s kể từ lúc đặt timeout mà hệ thống không nhận được kí tự \$ thì hệ thống sẽ dừng hoạt động của module RFID, khi này trên terminal sẽ hiển thị Locking door, trên LCD vẫn hiển thị Locking, và chân Rx của Arduino lúc này sẽ dùng cho giao tiếp với gateway. Nếu trong vòng 10s hệ thống nhận được kí tự '\$' thì hệ thống sẽ reset timeout và chờ người dùng nhập thông tin vào. Nếu nhận được đủ 10 kí tự thì hệ thống sẽ tự động kiểm tra xem có khớp với ID đã có trong hệ thống không, mặc định trong hệ thống sẽ có 2 ID là 12345admin (ID của admin, khi đăng nhập với ID này người dùng có thể thêm một ID mới vào hệ thống và tối đa hệ thống có 5 ID) và 12345aaaaa (normal user). Nếu người dùng không nhập đủ 10 kí tự hoặc không nhận kí tự nào mà ấn **Button 5** thì hệ thống cung tiến hành dừng hoạt động của module RFID. Khi đăng nhập thành công, trên LCD sẽ hiện thị Opening, trên terminal sẽ hiện thị Valid ID, nếu là admin thì sẽ thêm dòng Start admin session và đặt timeout là 3s, sau 3s hệ thống sẽ dừng hoạt động của module RFID.

Để tiến hành thêm thẻ RFID mới thì ta phải đăng nhập vào hệ thống với ID của admin, sau khi đăng nhập thành công, trong 3s người dùng phải tiến hành ấn **Button 5**, để hệ thống tiến hành thêm ID mới, lúc này LCD sẽ hiện thị Add new card, trên terminal sẽ hiện thị Please insert new card ID !!!. Để tiến hành nhập ID mới thì người dùng phải nhập kí tự \$ để báo bắt đầu nhập, và nhập 10 kí tự vào, nếu hệ thống nhận đủ 10 kí tự hệ thống sẽ không nhận nữa, người dùng phải ấn **Button 5** để tiến hành thêm vào hệ thống, hệ thống sẽ kiểm tra xem ID mới có bị trùng với ID cũ không, nếu bị trùng thì trên LCD sẽ thông báo Card is exist và trên terminal sẽ hiển thị ID card is exist sau 500ms hệ thống sẽ yêu cầu nhập lại ID, nếu người dùng nhập ID chưa đủ 10 kí tự mà vẫn ấn **Button 5** để đăng ký thì LCD sẽ thông báo Card is wrong và trên terminal hiển thị ID card is wrong!!! và sau 500ms sẽ yêu cầu người dùng nhập lại, lúc này LCD sẽ hiện thị Add new card và terminal hiển thị Please insert new card ID !!!. Người dùng sẽ phải nhập lại như lúc đầu với ID mới hoặc tiếp tục ấn **Button 5** để dừng hoạt động của module RFID. Nếu thêm ID mới thành công thì LCD sẽ thông báo Successfully và trên terminal sẽ hiển thị Insert new ID card successfully !!! sau 500ms hệ thống sẽ tự động dừng hoạt động của module RFID.

4.9 Máy trạng thái



Hình 48: Sơ đồ máy trạng thái của module



4.10 Hiện thực hệ thống

4.10.1 Hiện thực module RFID

```
1 #include "_RFID.h"
2 #include "Arduino.h"
3 #include "timer.h"
4 #include "common.h"
5 #include "_LCD.h"
6 #include "scheduler.h"
7 #include "gateway.h"
8 #include <stdint.h>
9
10
11 #define MAX_OF_IDS 5
12 #define NO_OF_CHAR_ID 10
13
14 // Array save ID of cards in system, default have two card
15 String IDs[MAX_OF_IDS] = {"12345admin", "12345aaaaa"};
16 // Number of current cards in system
17 int curNoOfIDs = 2;
18 // Save state of RFID module, default is -1 (RFID is not ready ←
19 // to use)
20 int stateRFID = -1;
21 // If in admin session isAdmin will be 1, else will be 0, ←
22 // default is 0
23 bool isAdmin = 0;
24
25 // Save ID read from terminal
26 String id;
27 // Save character read from terminal
28 char rcvChar;
29 // If receive '$'(character before starting to input ID) ←
30 // character it's value will be 1, else will be 0
31 bool detect_receive_id = 0;
32 // Use for display information of module in each state
33 int display_code = 0;
34
35 // Save ID of task task_gateway_sending in scheduler
36 extern uint32_t ID_task_gateway;
37
38 // Function end operation of module RFID
39 void exitRFID(){
40     ID_task_gateway = SCH_Add_Task(task_gateway_sending, 0, ←
41         PERIOD_GATEWAY_SENDING);
42     detect_receive_id = 0;
43     isAdmin = 0;
44     stateRFID = -1;
45     digitalWrite(MUX_PIN, LOW);
```



```
42     Serial.println(F("\r\nLocking door"));
43 }
44
45 // Initialization of module
46 void init_RFID(){
47     pinMode(MUX_PIN, OUTPUT);
48     digitalWrite(MUX_PIN, LOW);
49     Serial.begin(9600);
50     display_code = 0;
51     task_RFID_display();
52 }
53
54 // Function change state of RFID
55 void changeStateRFID(){
56     // Change to mode 0
57     if(stateRFID == -1){
58         // Change rx PIN for using RFID module
59         digitalWrite(MUX_PIN, HIGH);
60         SCH_Delete_Task(ID_task_gateway);
61         stateRFID = 0;
62         Serial.println(F("Please insert card ID !!!"));
63         // Set time out waiting for insert card
64         setTimer(0, TIMEOUT_WAITING_INPUT);
65     }
66     // Change to mode 1
67     else if(stateRFID == 2 && isAdmin){
68         resetTimer(0);
69         stateRFID = 1;
70         Serial.println(F("Please insert new card ID !!!"));
71         display_code = 2;
72         task_RFID_display();
73     }
74     else if(stateRFID == 1){
75         // Change to mode -1(Stop RFID running)
76         if(id.length() == 0){
77             Serial.println(F("End admin session"));
78             exitRFID();
79             display_code = 0;
80             task_RFID_display();
81         }
82         else if(id.length() == NO_OF_CHAR_ID){
83             int i;
84             for(i = 0; i < curNoOfIDs; i++){
85                 if(id == IDs[i]){
86                     break;
87                 }
88             }
89             // Change to mode 3
90             if(i < curNoOfIDs){
```



```
91         Serial.println(F("\r\nID card is exist"));
92         stateRFID = 3;
93         display_code = 5;
94         task_RFID_display();
95         setTimer(0, TIMEOUT_LCD_ANNOUNCEMENT);
96     }
97     // Change to mode 4
98     else{
99         IDs[curNoOfIDs++] = id;
100        Serial.println(F("\r\nInsert new ID card successfully ←
101                      !!!"));
102        stateRFID = 4;
103        display_code = 4;
104        task_RFID_display();
105        setTimer(0, TIMEOUT_LCD_ANNOUNCEMENT);
106    }
107    // Change to mode 3
108    else{
109        Serial.println(F("\r\nID card is wrong!!!"));
110        stateRFID = 3;
111        display_code = 6;
112        task_RFID_display();
113        setTimer(0, TIMEOUT_LCD_ANNOUNCEMENT);
114    }
115}
116// Change to mode -1(Stop RFID running)
117else if(stateRFID != 3){
118    exitRFID();
119    display_code = 0;
120    task_RFID_display();
121}
122}
123
124// RFID running base on state of module
125void RFID_run(){
126    switch(stateRFID){
127        case 0: // Waiting insert RFID card
128            if(getTimerFlag(0) == 1){
129                exitRFID();
130                resetTimer(0);
131            }
132            if(Serial.available() > 0){
133                rcvChar = Serial.read();
134                if(rcvChar == '$'){
135                    id = "";
136                    detect_receive_id = 1;
137                    resetTimer(0);
138                    Serial.print(rcvChar);
```



```
139 }
140 else if(detect_receive_id){
141     if(id.length() < NO_OF_CHAR_ID){
142         Serial.print(rcvChar);
143         id += rcvChar;
144     }
145     if(id.length() == NO_OF_CHAR_ID){
146         int i;
147         for(i = 0; i < curNoOfIDs; i++){
148             if(id == IDs[i]){
149                 Serial.print(F("\r\nValid ID\r\n"));
150                 display_code = 1;
151                 task_RFID_display();
152                 if(i == 0){
153                     Serial.println(F("Start admin session"));
154                     isAdmin = 1;
155                 }
156                 setTimer(0, TIMEOUT_FOR_RFID_SESSION);
157                 stateRFID = 2;
158                 break;
159             }
160         }
161         if(i >= curNoOfIDs){
162             Serial.print(F("\r\nInvalid ID\r\n"));
163         }
164         detect_receive_id = 0;
165     }
166 }
167 }
168 break;
169 case 1: // Waiting insert new RFID card
170 if(Serial.available() > 0){
171     rcvChar = Serial.read();
172     if(rcvChar == '$'){
173         id = "";
174         detect_receive_id = 1;
175         Serial.print(rcvChar);
176     }
177     else if(detect_receive_id){
178         if(id.length() < NO_OF_CHAR_ID){
179             Serial.print(rcvChar);
180             id += rcvChar;
181         }
182     }
183 }
184 break;
185 case 2: // Time out waiting for state 0, 3s
186 if(getTimerFlag(0) == 1){
187     display_code = 0;
```



```
188     task_RFID_display();
189     if(isAdmin == 1){
190         Serial.println(F("End admin session"));
191     }
192     exitRFID();
193     resetTimer(0);
194 }
195 break;
196 case 3: // Time out waiting for announcing fail to insert new card
197     if(getTimerFlag(0) == 1){
198         id = "";
199         display_code = 2;
200         task_RFID_display();
201         detect_receive_id = 0;
202         stateRFID = 1;
203         Serial.println(F("Please insert new card ID !!!"));
204         resetTimer(0);
205     }
206     break;
207 case 4: // Time out waiting for announcing insert new card successfully
208     if(getTimerFlag(0) == 1){
209         display_code = 0;
210         task_RFID_display();
211         exitRFID();
212         resetTimer(0);
213     }
214     break;
215 default:
216     break;
217 }
218 }
```

Code 11: Hiện thực module RFID

4.10.2 Hiển thị thông tin trạng thái của module RFID ra LCD

Trong module LCD:

- Tao ra LCD thứ 3:

```
1 LiquidCrystal_I2C lcd3(0x25, 16, 2);
```

Code 12: Tao LCD thứ 3

- Extern biến display_code từ module RFID để hiển thị theo trạng thái của module RFID:



```
1 extern int display_code;
```

Code 13: Thêm biến để hiển thị theo trạng thái của module RFID

- Thêm các hàm khởi tạo LCD vào trong hàm init_LCD():

```
1 lcd3.begin();
2 lcd3.backlight();
```

Code 14: Hàm khởi tạo LCD3

- Thêm hàm task_RFID_display() để hiển thị thông tin của module RFID ra màn hình LCD 3:

```
1 void task_RFID_display(){
2     switch(display_code){
3         case 0:
4             lcd3.clear();
5             lcd3.setCursor(0,0);
6             lcd3.print("Locking");
7             break;
8         case 1:
9             lcd3.clear();
10            lcd3.setCursor(0,0);
11            lcd3.print("Opening");
12            break;
13        case 2:
14            lcd3.clear();
15            lcd3.setCursor(0,0);
16            lcd3.print("Add new card");
17            break;
18        case 4:
19            lcd3.clear();
20            lcd3.setCursor(0,0);
21            lcd3.print("Successfully");
22            break;
23        case 5:
24            lcd3.clear();
25            lcd3.setCursor(0,0);
26            lcd3.print("Card is exist");
27            break;
28        case 6:
29            lcd3.clear();
30            lcd3.setCursor(0,0);
31            lcd3.print("Card is wrong");
32            break;
33    default:
```



```
34 }         break;  
35 }  
36 }
```

Code 15: Triển khai hàm task_RFID_display()

4.11 Demo

Video demo của module: [Link video](#)

5 Thiết lập Cooperative scheduler cho hệ thống

5.1 Giới thiệu về Scheduler

5.1.1 Đánh giá về cấu trúc siêu vòng lặp (Super Loop)

Cấu trúc siêu vòng lặp (Super Loop) có dạng như sau:

```
1 void setup() {  
2     // put your setup code here, to run once:  
3     X_Init(); // Prepare for task X  
4 }  
5  
6 void loop() {  
7     // put your main code here, to run repeatedly:  
8     X(); // Perform the task  
9 }
```

Code 16: Cấu trúc siêu vòng lặp (Super Loop)

Nhìn đoạn chương trình trên ta thấy được 2 ưu điểm của cấu trúc siêu vòng lặp (Super Loop) đó là:

- Khá đơn giản và dễ hiểu.
- Gần như không xài tới bộ nhớ hệ thống (System memory) hay tài nguyên CPU (CPU Resources).

Tuy nhiên một nhược điểm khá lớn của cấu trúc siêu vòng lặp này là khó thực hiện tác vụ (task) trong một thời gian chính xác. Trong thực tế, các tác vụ hầu như được gắn liền với một thời gian chính xác cho trước hay số lần thực hiện trong một thời gian chính xác cho trước vì nếu như không thỏa mãn được điều đó, hệ thống chạy sẽ không chính xác hay dẫn đến các trạng thái không mong muốn, đó cũng là yêu cầu của các hệ thống thời gian thực (Real time systems). Cụ thể ta xét một tập hợp các yêu cầu được tập hợp từ một loạt các dự án nhúng khác nhau (không theo thứ tự cụ thể):

- Tốc độ hiện tại của xe phải được đo trong khoảng thời gian 0.5 giây.
- Màn hình phải được refresh 40 lần mỗi giây.
- Một biến đổi tần số thời gian phải được thực hiện 20 lần mỗi giây.



- Nếu báo thức kêu, nó phải được tắt (vì lý do hợp lý) sau 20 phút.
- Nếu cửa trước bị mở, chuông báo động phải kêu sau 30 giây nếu không nhập đúng mật khẩu trong thời gian này.
- Dữ liệu rung động cơ phải được lấy mẫu 1000 lần mỗi giây.
- Dữ liệu miền tần số phải được phân loại 20 lần mỗi giây.
- Bàn phím phải được quét sau mỗi 200 ms.
- Nút chính (diều khiển) phải giao tiếp với tất cả các nút khác (nút cảm biến và nút âm thanh) một lần mỗi giây.
- Các cảm biến phải được lấy mẫu một lần mỗi giây.

Ta có thể tóm tắt danh sách này bằng cách nói rằng nhiều hệ thống nhúng phải thực hiện các tác vụ tại các thời điểm cụ thể. Cụ thể hơn, ta có hai loại hoạt động để thực hiện:

- Các tác vụ định kỳ (làm theo chu kỳ), được thực hiện (giả sử) cứ sau **100ms**.
- Các tác vụ chỉ làm 1 lần (one-shot task), sẽ được thực hiện sau khoảng thời gian trễ (giả sử) là **50ms**.

Điều này rất khó đạt được với kiến trúc đơn giản như thể hiện như trên. Tuy nhiên ta có thể "cố gắng" thực hiện việc trên bằng cách ta tìm ra thời gian thực hiện tác vụ đấy là bao nhiêu. Sau đó ta sẽ viết chương trình bao gồm việc thực hiện tác vụ và một hàm delay với thời gian bằng hiệu của thời gian cần đạt và thời gian thực hiện tác vụ. Ví dụ, giả sử rằng chúng ta cần bắt đầu tác vụ (task) X sau mỗi **200ms** và tác vụ cần **10ms** cần hoàn thành. Chương trình dưới đây minh họa một cách điều chỉnh chương trình gốc để đạt được ý muốn của mình:

```
1 void setup() {
2     // put your setup code here, to run once:
3     X_Init();      // Prepare for task X
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8     X();           // Perform the task
9     delay(190);   // Time delay is 200 - 10 = 190ms
10}
```

Code 17: Cố gắng sử dụng kiến trúc siêu vòng lặp để thực thi các tác vụ có chu kỳ

Nhìn chung, cách tiếp cận này không đầy đủ và khó thực hiện, bởi vì nó sẽ chỉ hoạt động nếu các điều kiện sau được thỏa mãn:

- Biết được thời gian thực hiện tác vụ (task X) chính xác là bao nhiêu.
- Thời gian đó không được phép thay đổi.

Trong các ứng dụng thực tế, việc xác định thời lượng tác vụ chính xác hiếm khi đơn giản. Giả sử chúng ta có một nhiệm vụ rất đơn giản không tương tác với thế giới bên ngoài mà thay vào đó, thực hiện một số tính toán bên trong. Ngay cả trong những trường hợp khá hạn chế



này, những thay đổi đối với cài đặt tối ưu hóa trình biên dịch - thậm chí là những thay đổi đối với một phần đường như không liên quan của chương trình - có thể thay đổi tốc độ thực thi tác vụ. Điều này có thể làm cho việc tinh chỉnh thời gian trở nên rất tẻ nhạt và dễ xảy ra lỗi. Điều kiện thứ hai thậm chí còn nhiều vấn đề hơn. Thông thường trong một hệ thống nhúng, nhiệm vụ sẽ được yêu cầu để tương tác với thế giới bên ngoài một cách phức tạp. Trong những trường hợp này, thời lượng tác vụ sẽ thay đổi tùy theo các hoạt động bên ngoài theo cách mà người lập trình có rất ít quyền kiểm soát.

Để thực hiện được việc trên ta sẽ cần có sự can thiệp của ngắt (interrupt), cụ thể ở đây là ngắt dựa vào thời gian (time-based interrupt). Khi một ngắt (interrupt) được tạo ra, bộ xử lý jump đến một địa chỉ ở dưới cùng của vùng bộ nhớ CODE (CODE memory area). Các vị trí này phải chứa mã phù hợp mà vì điều khiển có thể đáp ứng với ngắt hoặc thông thường hơn, các vị trí sẽ bao gồm một lệnh jump khác, cung cấp địa chỉ của "quy trình dịch vụ ngắt" (ISR) phù hợp nằm ở nơi khác trong (CODE) memory.

5.1.2 Giới thiệu về Scheduler

Có hai góc nhìn về Scheduler:

- Dưới góc nhìn đơn giản, scheduler có thể được xem như một hệ điều hành đơn giản cho phép các tác vụ (task) được gọi định kỳ hoặc các tác vụ chỉ dùng một lần (one-shot tasks).
- Dưới góc nhìn sâu hơn, scheduler có thể được xem như một quy trình dịch vụ ngắt bộ định thời (ISR) đơn lẻ được chia sẻ giữa nhiều tác vụ (task) khác nhau. Do đó, chỉ một bộ định thời (timer) cần được khởi tạo và mọi thay đổi đối với bộ định thời (timing) thường chỉ yêu cầu một chức năng được thay đổi. Hơn nữa, chúng ta thường có thể sử dụng cùng một scheduler cho dù chúng ta cần thực hiện 1, 10 hay 100 tác vụ khác nhau.

Dưới đây là một chương trình mẫu việc thêm 3 tác vụ (tasks) bằng cách sử dụng Scheduler:

```
1 void setup() {
2     // put your setup code here, to run once:
3     SCH_Init();      // Set up the scheduler
4     // Add the tasks (1ms tick interval)
5     SCH_Add_Task(Func_A, 0, 2); // Func_A will run every 2ms
6     SCH_Add_Task(Func_B, 1, 10); // Func_B will run every 10ms
7     SCH_Add_Task(Func_C, 3, 15); // Func_C will run every 15ms
8 }
9
10 void loop() {
11     // put your main code here, to run repeatedly:
12     SCH_Dispatch_Tasks();
13 }
```

Code 18: Chạy 3 tác vụ định kỳ bằng cách sử dụng bộ lịch trình (scheduler)

Dựa trên phương diện của hệ thời gian thực (RTOS), thì 2 loại bộ lịch trình (Scheduler) hệ thời gian thực (RTOS) thường được sử dụng nhất là: Bộ lịch trình hợp tác (**Cooperative scheduler**) và Bộ lịch trình có sự tranh chấp (**Preemptive scheduler**). Trong phạm vi của đồ án, ta chỉ đề cập đến bộ lịch trình hợp tác (**Cooperative scheduler**) như sau đây.



5.2 Giới thiệu về Cooperative Scheduler

5.2.1 Giới thiệu sơ lược

Bộ lịch trình hợp tác (Cooperative Scheduler) vẫn cho phép các tác vụ được lên lịch thông qua việc sử dụng bộ định thời định kỳ (periodic timer) nền tạo ra một tick hệ thống (system tick) giống như trong RTOS. Sự khác biệt là thay vì có các ưu tiên và tranh chấp, bộ lịch trình hợp tác (cooperative scheduler) chỉ thực hiện các nhiệm vụ xảy ra trong một khoảng thời gian định kỳ. Nếu hai tác vụ phải chạy cùng một lúc, tác vụ (task) có độ ưu tiên cao hơn trong danh sách tác vụ sẽ chạy (task) đầu tiên, sau đó là tác vụ (task) thứ hai, Cooperative Scheduler cho phép thực hiện hành vi thời gian thực mềm (soft real-time) nhưng thông qua việc sử dụng các ngắt (interrupt) và các cơ chế khác cũng có thể đáp ứng nhu cầu thời gian thực cứng (hard real-time).

5.2.2 Đặc điểm của Cooperative scheduler

Bộ lập lịch hợp tác (Cooperative scheduler) cung cấp kiến trúc hệ thống đơn tác vụ (single-tasking system architecture).

Nguyên lý hoạt động:

- Các tác vụ (task) được lên lịch thực thi vào những thời điểm cụ thể (định kỳ - period task hoặc một lần - one-shot task).
- Khi một tác vụ (task) được lên lịch thực thi, nó sẽ được thêm vào danh sách chờ (waiting list).
- Khi CPU rảnh (free), tác vụ chờ (waiting task) tiếp theo (nếu có) sẽ được thực hiện.
- Tác vụ thực thi đến khi hoàn thành, sau đó trả lại quyền kiểm soát cho bộ lịch trình (scheduler).

Mức độ triển khai chương trình:

- Bộ lịch trình (Scheduler) đơn giản và có thể được triển khai với một lượng nhỏ đoạn code.
- Bộ lịch trình (Scheduler) chỉ phải cấp phát bộ nhớ cho một tác vụ duy nhất tại một thời điểm.
- Bộ lịch trình (Scheduler) nói chung sẽ được viết hoàn toàn bằng ngôn ngữ cấp cao (chẳng hạn như ngôn ngữ C).
- Bộ lịch trình (Scheduler) không phải là một ứng dụng riêng biệt (seperate application), nó trở thành một phần trong đoạn chương trình của nhà phát triển.

Hiệu suất:

Để có thời gian phản hồi nhanh với các sự kiện (event) bên ngoài đòi hỏi sự cẩn thận ở giai đoạn thiết kế.

Dữ tin cậy và an toàn:

Lịch trình hợp tác (Cooperative scheduling) đơn giản, dễ đoán, đáng tin cậy và an toàn. Lý do chính tại sao các bộ lịch trình hợp tác (Cooperative scheduler) đều đáng tin cậy và có thể dự đoán được là chỉ có một tác vụ (task) hoạt động tại bất kỳ thời điểm nào: tác vụ (task) này chạy đến khi hoàn thành (Run to complete - RTC) và sau đó trả lại quyền kiểm soát cho bộ lịch



trình (Scheduler). Ngược lại điều này với tình huống trong một hệ thống tranh chấp (Preemptive system) hoàn toàn với nhiều hơn một tác vụ (task) đang hoạt động. Giả sử một tác vụ (task) trong hệ thống đang đọc từ một cổng (port) và bộ lịch trình (Scheduler) thực hiện "chuyển đổi ngữ cảnh" (context-switch), khiến một tác vụ (task) khác truy cập vào cùng một cổng (port): trong những trường hợp này, nếu ta không thực hiện hành động để ngăn chặn, dữ liệu có thể bị mất (lost) hoặc bị hỏng (corrupted).

5.3 Thiết lập Scheduler cho hệ thống

5.3.1 Giới thiệu sơ lược

Một bộ lịch trình (Scheduler) thường có các thành phần chính (key component) như sau:

- Một cấu trúc dữ liệu (data structure) cho bộ lịch trình (Scheduler).
- Một hàm khởi tạo (Init function).
- Một quy trình dịch vụ ngắn đơn (ISR), được sử dụng để cập nhật bộ lịch trình theo các khoảng thời gian đều đặn.
- Một hàm thêm tác vụ (task) vào bộ lịch trình (Scheduler).
- Một hàm điều phối tác vụ (dispatch task) với mục đích một tác vụ (task) được thực thi khi chúng đến hạn phải chạy.
- Một hàm để xóa các tác vụ (task) khỏi bộ lịch trình (Scheduler) (không bắt buộc trong tất cả các ứng dụng).

5.3.2 Cài đặt

Trước hết ta sẽ cài đặt cấu trúc dữ liệu (data structure) cho bộ lịch trình (Scheduler) như sau đây:

```
1 #define MAX_NUMBER_OF_TASK 15
2
3 typedef struct{
4     void (*pTask)(void);
5     uint32_t Delay;
6     uint32_t Period;
7     uint32_t TaskID;
8 } structTask;
9
10 // Array save task in scheduler
11 structTask SCH_task_array[MAX_NUMBER_OF_TASK];
```

Code 19: Cấu trúc dữ liệu cho bộ lịch trình (scheduler)

Sau đó ta sẽ viết các hàm interface theo các thành phần chính (key component) được giới thiệu ở **Phần 5.3.1** như sau đây:

```
1 // Initialization of module
2 void init_SCH();
```



```
3 // Update time of scheduler
4 void SCH_Update();
5
6 // Add task to scheduler
7 uint32_t SCH_Add_Task(void(*pFunction)(), uint32_t DELAY, ←
8     uint32_t PERIOD);
9
10 // Delete task from scheduler
11 uint32_t SCH_Delete_Task(uint32_t taskID);
12
13 // Execute task
14 void SCH_Dispatch_Tasks();
```

Code 20: Hàm interface của bộ lịch trình (Scheduler)

Việc triển khai chi tiết các hàm trên sẽ được thể hiện ở phần sau đây.

5.4 Triển khai

5.4.1 Ý tưởng

Như giới thiệu ở trên thì hàm `SCH_Update()`, sẽ được gọi trong hàm ngắt timer, chính vì vậy việc tối ưu để giảm tối đa độ phức tạp của nó là điều cần thiết. Trong dự án lần này nhóm đã tạo ra một bộ scheduler với độ phức tạp là $O(1)$ ở hàm `SCH_Update()`.

Ta thấy nhiệm vụ chính của hàm `SCH_Update` là lặp qua mỗi task và giảm delay của nó. Giả sử trong hàng đợi tất cả các task đều có delay như nhau thì việc lặp qua các task và giảm delay là việc làm không được tối ưu, giải pháp tốt hơn là chỉ dùng một biến delay thôi, nó sẽ là delay cho tất cả các task, lúc này nhiệm vụ của hàm `SCH_Update()` chỉ là giảm delay của một biến, độ phức tạp là $O(1)$.

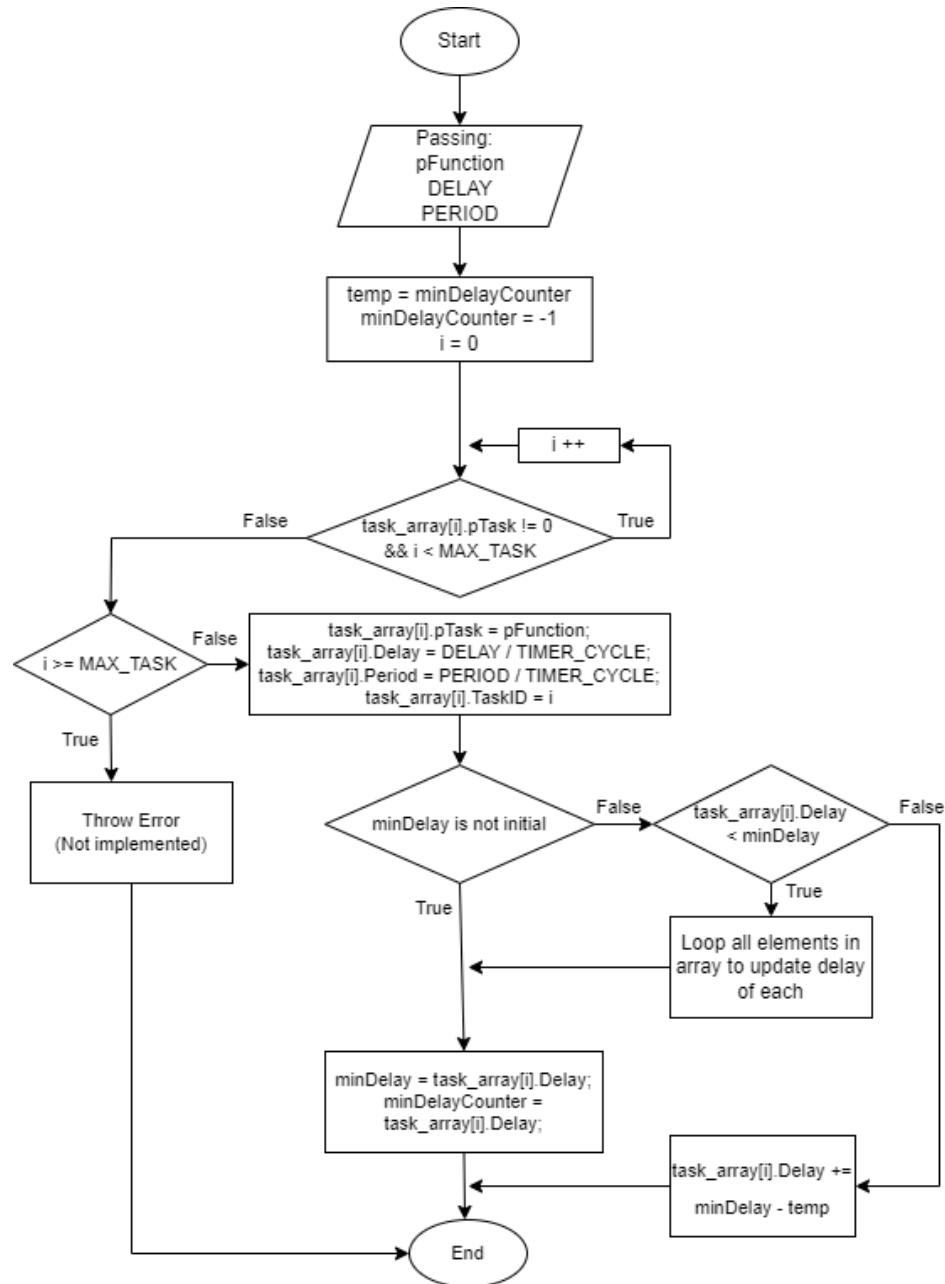
Trường hợp trong hàng đợi, các task đều có delay khác nhau thì sao? Trong trường hợp này ta sẽ tiến hành chọn trong hàng đợi task nào hiện đang có delay nhỏ nhất và gán delay của task đó cho một biến, `minDelayCounter`, để hàm `SCH_Update()` sẽ thực hiện trừ dần biến đó và biến `minDelay` để lưu giá trị delay đó. khi `minDelayCounter` bằng 0 thì ta sẽ thực hiện task đó và cập nhật delay cho các task khác bằng cách trừ delay của mỗi task cho `minDelay`, và cập nhật delay cho cả task vừa mới hiện thực, sau đó ta tiến hành tìm `minDelay`, `minDelayCounter` mới sau khi delay của các task đã được cập nhật.

Trường hợp có nhiều hơn 1 task có cùng `minDelay` thì sao? Thì lúc này ở hàm `SCH_Dispatch_Tasks()` ta sẽ lặp qua các task và trừ delay của nó cho `minDelay`, task nào có kết quả là 0, ta sẽ thực hiện task đó, không thì ta sẽ cập nhật delay cho task đó, và sau đó cập nhật `minDelay`, `minDelayCounter`. Ta thấy khi `minDelayCounter` chưa bằng 0 thì việc gọi liên tục hàm `SCH_Dispatch_Tasks()` rất lãng phí, lúc này ta thêm điều kiện khi `minDelayCounter` bằng 0 mới gọi `SCH_Dispatch_Tasks()`.

Trường hợp tại thời điểm thêm một task mới, thì delay của task đó nhỏ hơn `minDelayCounter` thì sao? Lúc này ta sẽ cập nhật delay cho các task cũ bằng cách trừ delay của nó cho `minDelay - minDelayCounter`, đây là khoảng thời gian mà nó đã chờ được, và cập nhật `minDelay`, `minDelayCounter`.

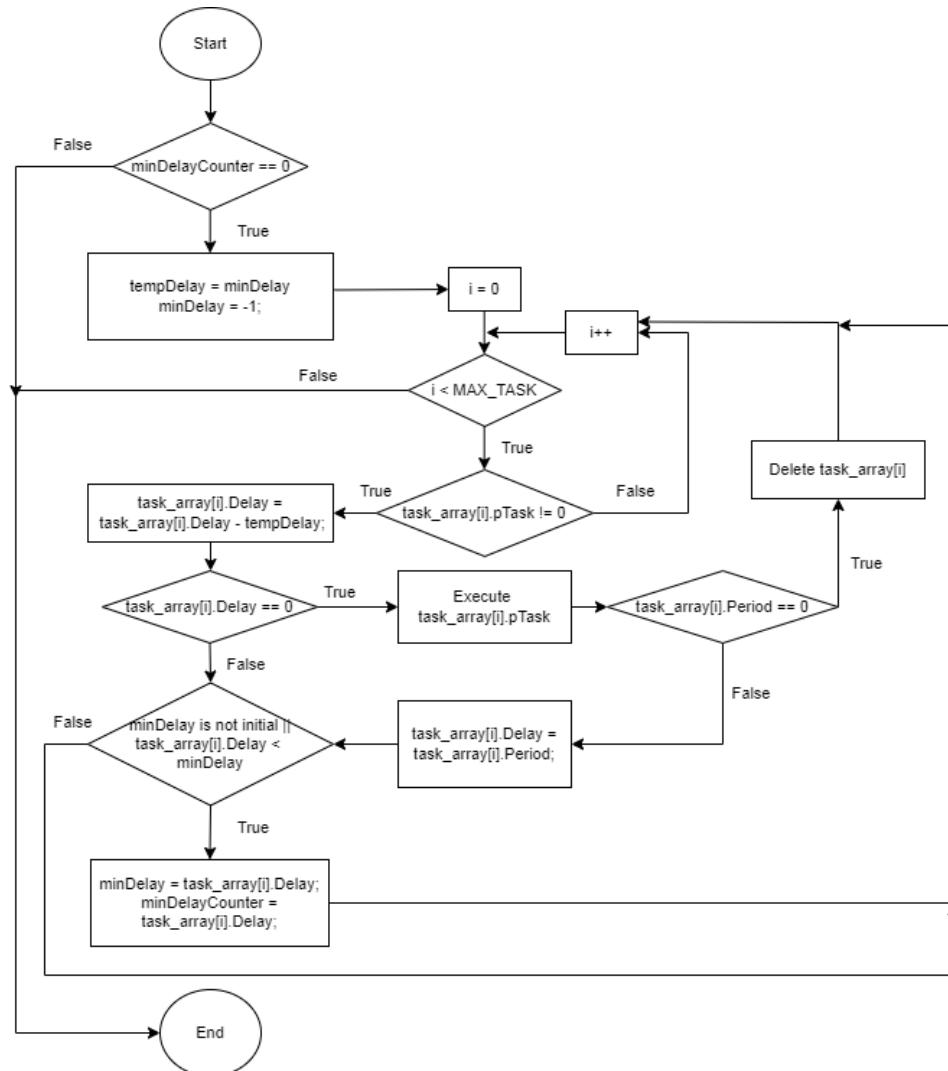
5.4.2 Flow chart

5.4.2.1 Hàm thêm tác vụ vào (SCH_Add_Task())



Hình 49: Flow chart của hàm SCH_Add_Task()

5.4.2.2 Hàm điều phối tác vụ (SCH_Dispatch_Tasks())



Hình 50: Flow chart của hàm SCH_Dispatch_Tasks()

5.4.3 Hiện thực module scheduler

```

1 #include "scheduler.h"
2 #include "common.h"
3 #include <stdint.h>
4 #include "Arduino.h"
5
6 #define MAX_NUMBER_OF_TASK 15
7

```



```
8  typedef struct{
9      void (*pTask)(void);
10     uint32_t Delay;
11     uint32_t Period;
12     uint32_t TaskID;
13 } structTask;
14
15 // Array save task in scheduler
16 structTask SCH_task_array[MAX_NUMBER_OF_TASK];
17 static int minDelayCounter = -1;
18 static int minDelay = -1;
19
20 // Initialization of module
21 void init_SCH(){
22     unsigned char i = 0;
23     for(; i<MAX_NUMBER_OF_TASK; i++){
24         SCH_task_array[i].pTask = 0;
25         SCH_task_array[i].Delay = 0;
26         SCH_task_array[i].Period = 0;
27         SCH_task_array[i].TaskID = 0;
28     }
29 }
30
31 // Update time of scheduler
32 void SCH_Update(){
33     if(minDelayCounter > 0) minDelayCounter--;
34 }
35
36 // Add task to scheduler
37 uint32_t SCH_Add_Task(void(*pFunction)(), uint32_t DELAY, ←
38     uint32_t PERIOD){
39     // Save minDelayCounter
40     int tempMinDelayCounter = minDelayCounter;
41     // Reset minDelayCounter to prevent SCH_Update update it's ←
42     // value
43     minDelayCounter = -1;
44     // Find empty element
45     unsigned char i = 0;
46     for(; i < MAX_NUMBER_OF_TASK; i++){
47         if(SCH_task_array[i].pTask == 0) break;
48     }
49     // Array is full, so I just end function
50     if(i >= MAX_NUMBER_OF_TASK){
51         return MAX_NUMBER_OF_TASK;
52     }
53     // Add task
54     SCH_task_array[i].pTask = pFunction ;
55     SCH_task_array[i].Delay = DELAY / TIMER_CYCLE;
56     SCH_task_array[i].Period = PERIOD / TIMER_CYCLE;
```



```
55     SCH_task_array[i].TaskID = i;
56     // minDelay is not initiated
57     if(minDelay < 0){
58         minDelay = SCH_task_array[i].Delay;
59     }
60     // minDelay is initiated and new task's delay is smaller than ←
61     // minDelayCounter
62     else if(minDelay >= 0 && SCH_task_array[i].Delay < ←
63             tempMinDelayCounter){
64         minDelay = SCH_task_array[i].Delay;
65         // delayed is time the old tasks have waited
66         int delayed = minDelay - tempMinDelayCounter;
67         // This if use for passing case delayed == 0
68         if(delayed > 0){
69             for(int j = 0; j < MAX_NUMBER_OF_TASK; j++){
70                 if(j != i && SCH_task_array[i].pTask != 0){
71                     SCH_task_array[i].Delay -= delayed;
72                 }
73             }
74         }
75         else if(minDelay >= 0 && SCH_task_array[i].Delay > ←
76                  tempMinDelayCounter){
77             // Make start time of new task is the same other old tasks
78             SCH_task_array[i].Delay += (minDelay - tempMinDelayCounter);
79         }
80         // Update minDelay, minDelayCounter
81         minDelayCounter = minDelay;
82         return i;
83     }
84     // Delete task from scheduler
85     uint32_t SCH_Delete_Task(uint32_t taskID){
86         if(taskID < 0 || taskID >= MAX_NUMBER_OF_TASK || ←
87             SCH_task_array[taskID].pTask == 0){
88             return 99999;
89         }
90         SCH_task_array[taskID].pTask = 0;
91         SCH_task_array[taskID].Delay = 0;
92         SCH_task_array[taskID].Period = 0;
93         SCH_task_array[taskID].TaskID = 0;
94         return taskID;
95     }
96     // Execute task
97     void SCH_Dispatch_Tasks(){
98         if(minDelayCounter == 0){
99             // Save minDelay
             int tempDelay = minDelay;
```



```
100 // Reset minDelay
101 minDelay = -1;
102 minDelayCounter = -1;
103
104 for(int i = 0; i < MAX_NUMBER_OF_TASK; i++){
105     if(SCH_task_array[i].pTask != 0){
106         // Update delay of task
107         SCH_task_array[i].Delay = SCH_task_array[i].Delay - ↫
108             tempDelay;
109         // Execute task
110         if(SCH_task_array[i].Delay == 0){
111             (*SCH_task_array[i].pTask)();
112             // Task is execute one time
113             if(SCH_task_array[i].Period == 0) {
114                 SCH_Delete_Task(i);
115                 continue;
116             }
117             // Task has period
118             else {
119                 SCH_task_array[i].Delay = SCH_task_array[i].Period;
120             }
121
122         // Update minDelayTask
123         // minDelay is not initiated
124         if(minDelay < 0) {
125             minDelay = SCH_task_array[i].Delay;
126         }
127         // minDelay is initiated
128         else{
129             if(SCH_task_array[i].Delay < minDelay) {
130                 minDelay = SCH_task_array[i].Delay;
131             }
132         }
133     }
134     minDelayCounter = minDelay;
135 }
136 }
137 }
```

Code 21: Hiện thực module của bộ lịch trình (Scheduler)

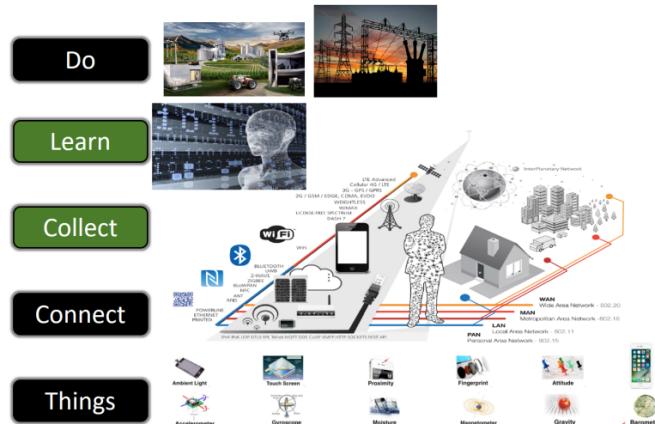
6 Gửi dữ liệu lên server IoT

6.1 Giới thiệu về IoT

6.1.1 Giới thiệu sơ lược

Internet vạn vật, hay còn gọi là Internet of Things – IoTs, là một cuộc cách mạng trong việc kết nối giữa các thiết bị không dây với nhau. Ban đầu, chúng ta có mạng Internet, một thành tựu của cuộc cách mạng khoa học công nghệ lần thứ 3, cho phép các máy tính có thể kết nối và trao đổi thông tin toàn cầu. Tuy nhiên, với sự phát triển nhanh chóng của ngành vi cơ điện tử (Micro Electro Mechanical System), không chỉ máy tính, giờ đây rất nhiều các thiết bị có khả năng kết nối vào mạng Internet. Thông dụng nhất trong cuộc sống mà chúng ta có thể kể đến như các điện thoại thông minh, máy tính bảng, các loại thẻ thông minh (Smart cards) hay như các nốt trong mạng cảm biến không dây (Wireless Sensor Networks). Theo ước tính của cộng đồng khoa học, đến năm 2025 sẽ có 75 tỉ thiết bị có thể kết nối Internet với nhau. Với những đặc tính đó, một thế hệ mạng mới đã được hình thành, và là sản phẩm đặc trưng cho cuộc cách mạng khoa học công nghệ lần thứ 4, mạng Internet vạn vật, hay còn gọi là IoT - Internet of Things.

Dựa trên mạng Internet vạn vật, các ứng dụng không còn ở khái niệm thông minh nữa, mà sẽ tiến lên một bước cao hơn, gọi là tự hành (autonomous), chẳng hạn như các ứng dụng giám sát và tự động thích nghi trong việc điều khiển như các dịch vụ trong nhà, bãi giữ xe, hay các hệ thống quan trắc trong nông nghiệp, thủy hải sản. Theo diễn giả nổi tiếng Timothy Chou, kiến trúc về ứng dụng thông minh dựa trên Internet vạn vật của ông được chia thành mô hình 5 lớp, như mô tả ở hình bên dưới.



Hình 51: Kiến trúc 5 lớp của một ứng dụng Kết nối vạn vật (IOT)

Chức năng chính của từng lớp trong kiến trúc này được khái quát như sau:

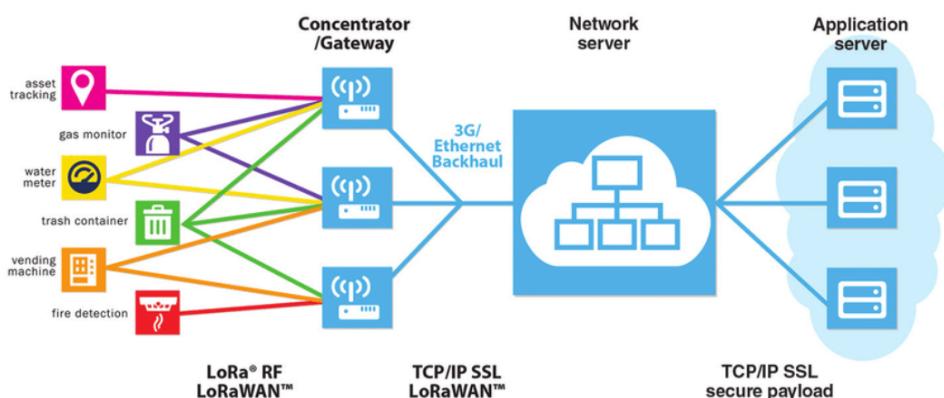
- **Things:** Các thiết bị trong ứng dụng giám sát. Chúng ta có thể thấy, đây là lớp rất phong phú về mặt số lượng và đa dạng về chức năng. Rất nhiều các loại cảm biến sẽ được dùng, tùy vào các ứng dụng giám sát. Bên cạnh đó, các nốt cảm biến sẽ chủ yếu dựa vào giao tiếp không dây.
- **Connect:** Thu thập dữ liệu từ các nốt cảm biến. Do có rất nhiều tiêu chuẩn kết nối tùy theo từng loại ứng dụng, lớp này phải hỗ trợ nhiều loại kết nối, từ giao tiếp Zigbee và WiFi

trong các ứng dụng nhà thông minh, với khoảng cách giao tiếp ngắn cho đến các giao trên không gian rộng như LoRa hay 3G/4G.

- **Collect:** Sau khi dữ liệu được thu thập, chúng sẽ được gửi lên các server tập trung để lưu trữ dữ liệu. Tại đây, một lượng lớn dữ liệu sẽ được đẩy về, tạo ra một thách thức không nhỏ cho các server và phải ứng dụng các công nghệ về Big Data (dữ liệu lớn) để xử lý.
- **Learn:** Nhiệm vụ của lớp này là lọc ra các thông tin đặc trưng, có ngữ nghĩa đặc thù cho từng loại ứng dụng. Các công nghệ về Học Máy và hiện tại là Học Sâu (Deep Learning) sẽ được áp dụng ở đây.
- **Do:** Dựa vào các thông tin đặc trưng, hệ thống sẽ xây dựng nên những quy luật thích nghi theo ngoại cảnh, và đề xuất các quyết định cho hệ thống. Với mỗi quyết định, việc thực thi sẽ được đo đạc một cách tự động, và sai lệnh của quyết định đó so với mục tiêu tối ưu sẽ được xem xét lại cho lần sau. Theo cách này, hệ thống sẽ tự động tích lũy “kinh nghiệm” trong một thời gian dài, để ngày càng trở nên thông minh và hoàn thiện hơn.

6.1.2 Kiến trúc 4 thành phần của ứng dụng IoT

Dựa trên mô hình 5 lớp của kiến trúc IoTs, chúng ta sẽ phân ứng dụng thành 4 thành phần cơ bản, bao gồm cảm biến, gateway trung tâm, server và các thiết bị để theo dõi dữ liệu và điều khiển từ xa, như trình bày ở hình bên dưới:



Hình 52: Kiến trúc 4 thành phần trong ứng dụng IoT

Trong phạm vi đồ án ta sẽ sử dụng **Adafruit IO** làm server và dùng ngôn ngữ **Python** làm ngôn ngữ lập trình Gateway IoT ta đề cập dưới đây.

6.1.3 Giới thiệu về Adafruit IO

Adafruit IO là một trong những nhà cung cấp đám mây tập trung nhiều hơn vào việc triển khai IoT trên đám mây. Adafruit IO hỗ trợ các phần cứng khác nhau như Raspberry PI, ESP2866 và Arduino. Các nhà phát triển IoT thích Adafruit IO hơn các nhà cung cấp đám mây IoT khác vì những lý do sau:

- **API mạnh mẽ:** Cung cấp các thư viện cho các ngôn ngữ lập trình khác nhau, cũng cung cấp hỗ trợ giao diện người dùng tích hợp sẵn.



- **Trang tổng quan (Dashboard):** Hiểu dữ liệu thông qua biểu đồ và đồ thị cho phép chúng tôi đưa ra quyết định tốt hơn.
- **Tính riêng tư (Privacy):** Dữ liệu được bảo mật trên nền tảng đám mây với các thuật toán mã hóa tốt hơn.
- **Tài liệu & Cộng đồng:** Nhiều blog với sự hỗ trợ của cộng đồng đáng kinh ngạc cho phép các sản phẩm phát triển liên tục.
- **Chi phí rẻ:** Mọi thứ có thể làm miễn phí trên Adafruit IO.
- **Thời gian phản hồi dữ liệu nhanh:** Khi người dùng muốn bật một bóng đèn với server ThingSpeak, tối thiểu 30 giây sau Gateway IoT mới có thể nhận ra lệnh này và thực hiện. Tuy nhiên, với server Adafruit IO, độ trễ này là rất thấp, và thông thường là dưới 1 giây.

6.2 Giới thiệu về giao thức MQTT (Message Queueing Telemetry Transport)

6.2.1 Giới thiệu sơ lược

Giao thức Message Queue Telemetry Transport (MQTT) là giao thức truyền thông điệp theo mô hình publish/subscribe ra đời năm 1999 bởi Andy Stanfor-Clark và Arlen Nipper tại IBM để giải quyết những hạn chế cụ thể của việc kết nối các đường ống dẫn dầu và khí đốt từ xa qua vệ tinh. Một số yêu cầu được đặt ra cho MQTT:

- Thực hiện đơn giản.
- Cung cấp một dạng chất lượng dịch vụ.
- Nhẹ để xử lý và hiệu quả về băng thông.
- Là dữ liệu dạng bất khả tri.
- Có để ý đến sự liên tục của phiên (session).
- Có tính bảo mật.

MQTT là giao thức nội bộ và độc quyền của IBM trong nhiều năm cho đến khi được phát hành trong phiên bản 3.1 năm 2010 dưới dạng một sản phẩm miễn phí bản quyền. Năm 2013, MQTT đã được tiêu chuẩn hóa và được chấp nhận vào tổ hợp OASIS (tổ chức tiêu chuẩn hóa các dự án phi lợi nhuận). Vào năm 2014, OASIS đã phát hành nó dưới dạng phiên bản MQTT 3.1.1. MQTT cũng là một tiêu chuẩn ISO (ISO/IEC PRF 20922).

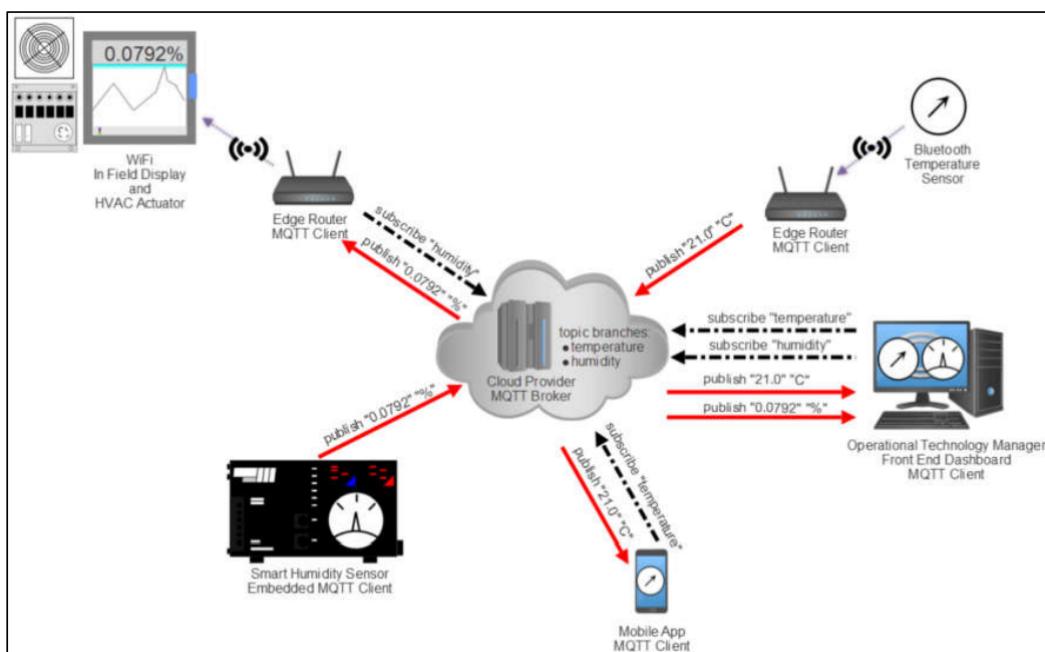
6.2.2 Các khái niệm trong giao thức MQTT

- **Subscriber:** Client ở đóng vai trò là nơi nhận thông điệp.
- **Subscribe:** Hành động đăng ký nhận thông điệp từ một chủ đề.
- **Publisher:** Client ở đóng vai trò là nơi phát thông điệp.
- **Publish:** Hành động gửi thông điệp lên một chủ đề.
- **Broker:** Server vai trò trung gian, nhiệm vụ chính của broker là nhận và chuyển các thông điệp đến từ publisher và đi đến các subscriber tương ứng. Có thể có thêm các chức năng khác tùy nền tảng MQTT.

- **Topic:** Chủ đề, khi subscriber muốn nhận thông điệp, nó phải nêu rõ là nhận thông điệp từ topic nào. Tương tự, với publisher, khi muốn đăng thông điệp sẽ phải nêu rõ thông điệp muốn đăng lên topic nào. Một publisher có thể đăng thông điệp lên bất kỳ topic nào, và nếu có subscriber đang theo dõi ở topic đó thì broker sẽ chuyển thông điệp đến cho subscriber.

6.2.3 Kiến trúc publish/subscribe

Kiến trúc Publish/Subscribe là một cách để tách một ứng dụng máy khách (client) đang truyền thông điệp từ một ứng dụng client khác đang nhận thư. Không giống như mô hình client-server truyền thống, client không được nhận dạng bởi đặc tính vật lý nào như địa chỉ IP, điều này hữu ích vì trong việc triển khai IoT vì danh tính vật lý có thể không xác định hoặc phổ biến. MQTT là giao thức có kiến trúc publish/subscribe, nhưng không phải là một hàng đợi tin nhắn. Hàng đợi tin nhắn theo bản chất lưu trữ tin nhắn trong khi MQTT thì không. Trong MQTT, nếu không có ai đăng ký (hoặc nghe) một chủ đề, chủ đề đó sẽ bị bỏ qua và mất đi.



Hình 53: Mô hình MQTT publish-subscribe cho việc truyền tín hiệu về nhiệt độ và độ ẩm

Hình 52 là một ví dụ minh họa về một cấu trúc liên kết MQTT publish-subscribe. Trong đó, các client chạy ở cạnh biên thực hiện publish hoặc subscribe các topic được quản lý bởi MQTT broker. Ở đây, có 2 topics đó là "độ ẩm" và "nhiệt độ". Hình vẽ cũng cho thấy một cảm biến thông minh có đủ tài nguyên để trở thành một MQTT client trong khi đó các cảm biến khác phải cần đến các bộ định tuyến biên (edge router) để thực hiện chức năng này.

MQTT là dữ liệu dạng bất khả tri, bất kỳ kiểu dữ liệu nào cũng có thể chứa trong nội dung của thông điệp, điều này cho phép chuyển đi các cá dữ liệu như âm thanh, hình ảnh, nhị phân, ... miễn là publisher và subscriber đã thống nhất với nhau và hiểu được định dạng dữ liệu đó.



6.2.4 Cấu trúc giao thức MQTT

MQTT được vận chuyển dựa trên giao thức TCP, điều đó giúp đảm bảo một phần gói tin sẽ được chuyển đi đến đích.

MQTT có thể lưu giữ thông điệp trên broker vô thời hạn. Chế độ này được điều khiển bởi một cờ gửi kèm thông điệp. Khi một thông điệp được lưu giữ trên broker, nó sẽ được gửi đến bất subcriber nào đăng ký vào nhánh chủ đề đó. Điều này cho phép subcriber mới nhận được thông điệp ở chủ đề đăng ký mà không cần đợi.

Mặc dù MQTT dựa trên TCP, các kết nối vẫn có thể bị mất, đặc biệt là trong môi trường cảm biến không dây hay thiết bị mất nguồn điện. Lúc này, kết nối sẽ ở trạng thái nửa mở (half-open) và máy chủ tin rằng kết nối còn duy trì và vẫn chờ dữ liệu đến. Để khắc phục tình trạng này, MQTT sử dụng một cơ chế gọi là keep-alive. Trong đó, cả broker và client đều đảm bảo rằng kết nối vẫn còn ngay cả khi không có thông điệp nào được truyền đi trong một khoảng thời gian. Cụ thể, khi không có thông điệp nào được truyền đi trong một khoảng thời gian định trước, cơ chế keep-alive kích hoạt client gửi một gói PINGREQ đến broker, gói này sẽ được xác nhận bằng một PINGRESP. Bộ định thời keep-alive được thiết lập ở cả client và broker. Cả gói tin keep-alive và thông điệp đều làm bộ định thời keep-alive bắt đầu đếm lại. Nếu bộ định thời keep-alive đếm vượt quá giá trị ngưỡng, broker sẽ đóng kết nối lại và gửi đi gói LWT tới các client.

Trong trường hợp không nhận được bản lưu cùn tồn tại và bộ đếm thời gian hết hạn, nhà môi giới sẽ đóng kết nối và gửi gói LWT đến tất cả các máy khách. Sau đó, khách hàng có thể cố gắng kết nối lại. Trong trường hợp đó, một kết nối nửa mở sẽ được nhà môi giới đóng lại và mở ra một kết nối mới cho khách hàng.

Cơ chế keep-alive hỗ trợ các kết nối bị gián đoạn, việc thiết lập lại các thông tin chủ đề theo dõi của client và các thông số QoS (Quality of Service) có thể dẫn đến việc tiêu tốn tài nguyên không cần thiết. Để hạn chế điều này, MQTT cho phép các dạng kết nối liên tục. Một kết nối liên tục sẽ lưu các thông tin trên broker như sau:

- Tất cả các thông tin theo dõi chủ đề của client.
- Tất cả các thông điệp QoS chưa được xác nhận bởi client.
- Tất cả các thông điệp QoS mới mà client chưa nhận được.

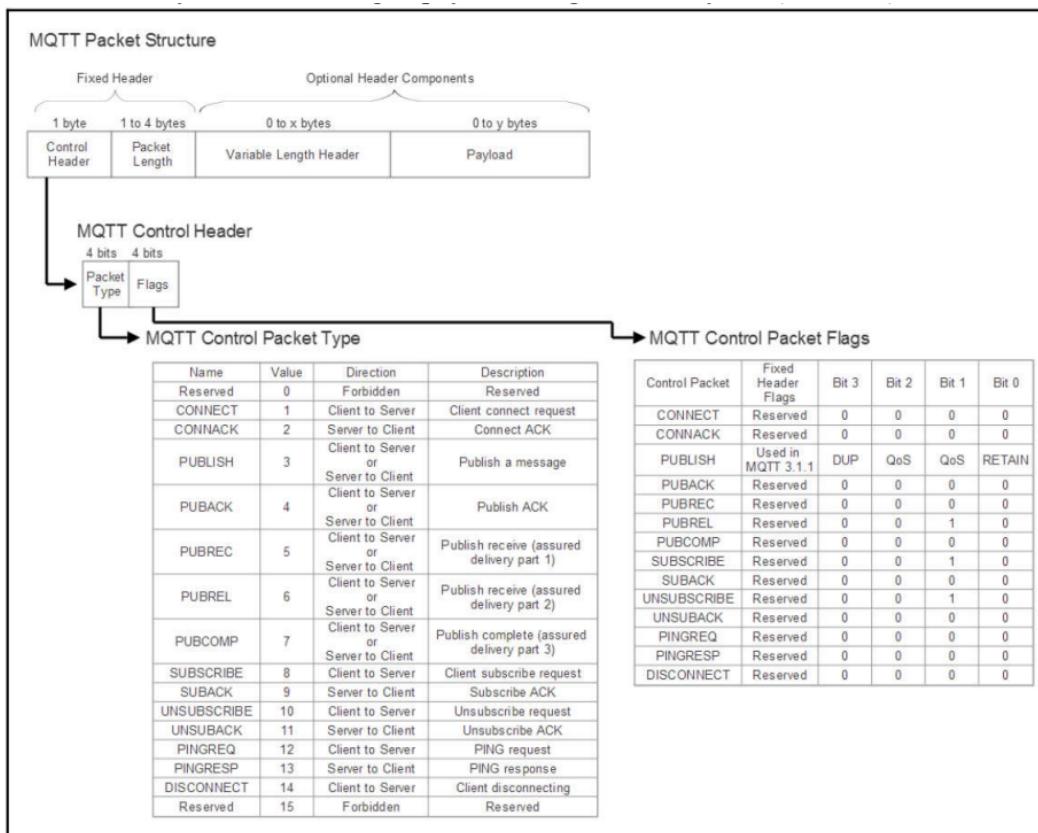
Broker cũng có thể thiết lập cấu hình không cho phép sử dụng các kết nối liên tục. Các client nào không muốn thất lạc bất kỳ thông điệp nào thì nên thiết lập kết nối liên tục, ngược lại đối với client chỉ thực hiện phát hành các thông điệp.

Có ba cấp độ QoS trong MQTT:

- **QoS-0:** Đây là mức chất lượng thấp nhất, khi thông điệp gửi với QoS0 thì không cần người nhận xác nhận hay người gửi gửi lại thông điệp.
- **QoS-1:** Chế độ này sẽ đảm bảo gửi thông điệp ít nhất một lần đến người nhận. Nó có thể được gửi nhiều lần và người nhận sẽ gửi lại xác nhận kèm theo phản hồi PUBACK.
- **QoS-2:** Đây là mức chất lượng cao nhất, đảm bảo và thông báo cho cả người gửi và người nhận rằng một thông điệp đã được truyền đi một cách chính xác. Chế độ này tạo ra nhiều lưu lượng truy cập hơn với khi thực hiện bắt tay nhiều bước giữa người gửi và người nhận. Nếu người nhận nhận được một thông điệp với QoS-2, nó sẽ trả lời bằng gói PUBREC cho người gửi để xác nhận và người gửi sẽ trả lời bằng gói PUBREL. PUBREL cho phép người nhận loại bỏ mọi lần truyền lại thông điệp một cách an toàn. PUBREL sau đó được xác nhận bởi người nhận bằng PUBCOMP. Cho đến khi tin nhắn PUBCOMP được gửi đi, người nhận sẽ lưu thông điệp gốc vào bộ nhớ cache cho an toàn.

6.2.5 Cấu trúc gói tin MQTT

Gói MQTT bao gồm 2 mào đầu, một mào đầu 2 byte cố định, một mào đầu có kích thước thay đổi và sau cùng là payload cũng có thể thay đổi.



Hình 54: Cấu trúc gói tin MQTT

Một liên kết truyền thông bằng giao thức MQTT bắt đầu với client gửi một thông điệp CONNECT đến một broker. Broker sẽ luôn trả lời một gói CONNECT bằng một gói CONNACK kèm mã trạng thái. Một khi kết nối được thiết lập, nó sẽ duy trì trạng thái mở. Dưới đây là các thông điệp MQTT và định dạng cấu trúc của nó:

CONNECT (client gửi cho server)

Client sẽ gửi gói tin Connect đến Server để yêu cầu kết nối có cấu trúc như sau:



Trường	Yêu cầu	Mô tả
clientID	Bắt buộc	Xác định client cho server. Mỗi client có một ID, chiều dài từ 1 – 23 byte UTF-8
cleanSession	Tùy chọn	0: Server phải quay lại các liên kết với client. Client và server phải lưu lại trạng thái phiên sau khi ngắt kết nối. 1: Client và server phải hủy các trạng thái của phiên liên kết trước và bắt đầu một phiên mới.
username	Tùy chọn	Tên để server xác thực client.
password	Tùy chọn	Mật khẩu độ dài từ 0 – 65536 byte kèm 2 byte cố định ở đầu.
lastWillTopic	Tùy chọn	Chủ đề sẽ đẩy thông điệp khi mất kết nối
lastWillQoS	Tùy chọn	2 bit cho biết mức QoS khi xuất bản thông điệp
lastWillMessage	Tùy chọn	Xác định loại tải trọng thông điệp gói Will
lastWillRetain	Tùy chọn	Cho biết gói Will có được duy trì trên broker sau khi đã được xuất bản
keepAlive	Tùy chọn	Client sẽ phải gửi thông điệp hoặc gói PINGREQ trước khi bộ định thời keep-alive hết hạn. Server sẽ đóng kết nối sau 1.5 thời gian thiết lập của keep-alive. Giá trị 0 sẽ vô hiệu hóa cơ chế này (không sử dụng).

Bảng 23: Cấu trúc gói tin Connect do client gửi

Gói trả lời yêu cầu CONNECT (server gửi cho client)

Không phải broker sẽ chấp nhận tất cả các yêu cầu kết nối. Broker sẽ phản hồi các yêu cầu kết nối có thể, gói tin trả lời của server có cấu trúc như sau:

Mã trả về	Mô tả
0	Kết nối thành công
1	Kết nối bị từ chối – Định danh của client đúng chuẩn UTF-8, nhưng không được server chấp nhận
2	Kết nối bị từ chối – Server không khả dụng.
3	Kết nối bị từ chối – Server không khả dụng.
4	Kết nối bị từ chối – Xác thực sai.
5	Kết nối bị từ chối – Client không được phép kết nối.

Bảng 24: Cấu trúc gói tin phản hồi Connect do server gửi

PUBLISH (client gửi cho server)

Một client muốn xuất bản dữ liệu đến một topic. Mỗi gói tin PUBLISH phải có một topic có cấu trúc như sau:



Trường	Yêu cầu	Mô tả
packetID	Bắt buộc	Dịnh danh cho gói là duy nhất nằm trong phần mào đầu thay đổi. Đối với QoS-0 thì giá trị packetID luôn bằng 0.
topicName	Tùy chọn	Nhánh chủ đề để thông điệp được đưa lên
qos	Tùy chọn	Cấp QoS 0,1 hay 2.
retainFlag	Tùy chọn	Cờ cho biết có duy trì thông điệp trên chủ đề.
payload	Tùy chọn	Tùy chọn Nội dung thông điệp
dupFlag	Tùy chọn	Có phải là thông điệp trùng và được gửi lại

Bảng 25: Cấu trúc gói tin Publish

SUBSCRIBE (client gửi cho server)

Nội dung của một gói đăng ký chủ đề bao gồm tối thiểu thông tin về topic và mức QoS. Một gói đăng ký có thể có nhiều cặp topic và QoS nếu client có nhu cầu để giảm thiểu số lần gửi đăng ký, cấu trúc như sau:

Trường	Yêu cầu	Mô tả
packetID	Bắt buộc	Dịnh danh cho gói là duy nhất nằm trong phần mào đầu thay đổi.
topic_1	Bắt buộc	Đường dẫn chủ đề 1 muốn lắng nghe
qos_1	Bắt buộc	Mức độ QoS mà thông điệp được gửi đến topic_1.
topic_2	Tùy chọn	Đường dẫn chủ đề 2 muốn lắng nghe.
qos_2	Tùy chọn	Mức độ QoS mà thông điệp được gửi đến topic_2.

Bảng 26: Cấu trúc gói tin Subscribe

Tại đây cho phép thực hiện đăng ký nhiều topic sử dụng các ký tự đại diện. Chẳng hạn, ta có đường dẫn chủ đề {country}/{city}/{temperature,humidity}.

- Ký tự đại diện một cấp +: Thay thế một cấp trong đường dẫn tên chủ đề. Ví dụ khi đăng ký chủ đề VN/+/{temperature} có nghĩa là client đăng ký nhận thông điệp của tất cả các thành phố của Việt Nam.
- Ký tự đại diện nhiều cấp *: Thay thế nhiều cấp trong đường dẫn tên chủ đề. Luôn là ký tự cuối cùng trong đường dẫn. Ví dụ khi đăng ký chủ đề VN/*, có nghĩa là client đăng ký nhận tất cả các thông điệp của nước Việt Nam.
- Ký tự đại diện cho các chủ đề đặc biệt \$: Đây là chế độ thống kê đặc biệt dành cho các MQTT server. Client không được xuất bản thông điệp vào các chủ đề \$.

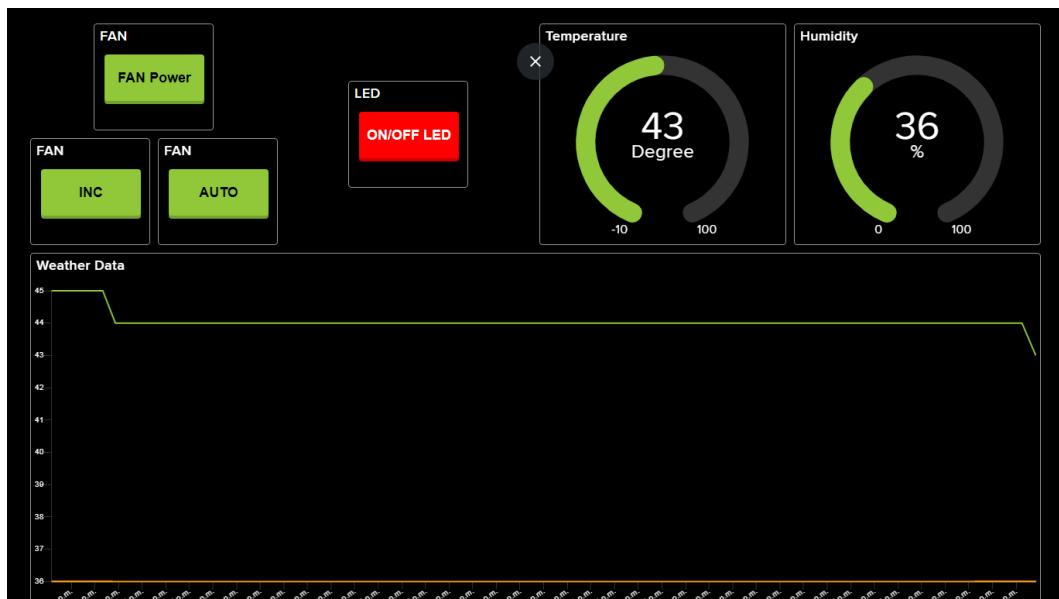
6.3 Thiết lập Dashboard thông qua server Adafruit IO

Để tạo được Dashboard trên Adafruit IO thì trước tiên ta phải tạo một tài khoản thành công, sau đó đăng nhập vào. Tạo các feed cần thiết cho các luồng dữ liệu giữa gateway và Dashboard. Sau khi tạo các feed thì ta tiến hành tạo block tương ứng với các feed và tùy vào mục đích của mình.

Dashboard gồm 7 block: 4 block button, 2 block gauge, 1 block line chart. Trong đó:

- Trong 4 block button có 3 block button dùng cho điều khiển quạt(FAN Power, INC, AUTO tương ứng với các chức năng của Button 2, Button3, Button 4 trên Proteus) và 1 block button dùng cho việc tắt bật LED.

- 2 block gauge dùng để hiển thị thông tin nhiệt độ và độ ẩm hiện thời của không khí ở trong nhà.
- 1 block line chart dùng để hiện thị sự thay đổi của nhiệt độ và độ ẩm của không khí ở trong nhà theo thời gian.



Hình 55: Dashboard tạo trên Adafruit IO

6.4 Hiện thực IoT Gateway bằng ngôn ngữ Python

6.4.1 Giao tiếp giữa gateway và hệ thống nhà thông minh

Cấu trúc câu lệnh giữa gateway và hệ thống:

- Từ hệ thống gửi lên gateway sẽ có định dạng là: *<command>#
command là thông tin về nhiệt độ và độ ẩm dạng float, được cách nhau bởi khoảng trắng.
Ví dụ: *27.0 70.0# thì có nghĩa là nhiệt độ là 27°C và độ ẩm là 70%.
- Từ gateway gửi xuống hệ thống sẽ có định dạng là: #<command>*
command là các lệnh mà server yêu cầu hệ thống để điều khiển các thiết bị. command có thể là các lệnh sau đây: FAN_ON, FAN_INC, FAN_AUTO và LED_ON.

6.4.2 Giao tiếp giữa gateway và server Adafruit IO

- **Từ gateway gửi lên server:** Khi nhận được thông tin về nhiệt độ và độ ẩm gửi lên từ hệ thống, gateway sẽ phân tích data nhận được và gửi thông tin lên feed thích hợp ở server.
- **Từ server xuông gateway:** Message từ server xuông gateway gồm: FAN, INC, AUTO và LED sau khi nhận được các message này thì gateway ngay lập tức sẽ tạo các lệnh tương ứng FAN_ON, FAN_INC, FAN_AUTO và LED_ON để gửi xuống hệ thống.



6.4.3 Hiện thực Gateway IoT

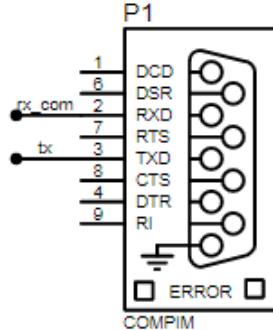
```
1 import serial
2 from time import sleep
3 import sys
4 from Adafruit_IO import MQTTClient
5
6 AIO_USERNAME = "KhanhNguyen"
7 AIO_KEY = "aio_xeqf11kxn9Lcqtc3eiiuQjdBhbJy"
8 AIO_SMARTHOME = ["AC_Power", "AC_Increase", "AC_Decrease", "LED", "Temperature", "Humidity"]
9
10 arduino = serial.Serial(port='COM2', baudrate=9600, timeout=.1)
11
12 def connected(client):
13     print("Connected...")
14     for feed in AIO_SMARTHOME:
15         client.subscribe(feed)
16
17 def subscribe(client , userdata , mid , granted_qos):
18     print("Subscribe successfully...")
19
20 def disconnected(client):
21     print("Disconnected...")
22     sys.exit (1)
23
24 def message(client , feed_id , payload):
25     print("Received data: " + payload)
26     if(payload == "FAN"):
27         arduino.write(bytes("#FAN_ON*", 'UTF-8'))
28     elif(payload == "AUTO"):
29         arduino.write(bytes("#FAN_AUTO*", 'UTF-8'))
30     elif(payload == "INC"):
31         arduino.write(bytes("#FAN_INC*", 'UTF-8'))
32     elif(payload == "LED"):
33         arduino.write(bytes("#LED_ON*", 'UTF-8'))
34
35 client = MQTTClient(AIO_USERNAME , AIO_KEY)
36 client.on_connect = connected
37 client.on_disconnect = disconnected
38 client.on_message = message
39 client.on_subscribe = subscribe
40 client.connect()
41 client.loop_background()
42
43 def data_processing(data):
44     if data.endswith("\r\n") & data.startswith("*"):
45         data = data.replace ("\r\n", "")
46         data = data.replace ("*", "")
47     return data.split()
48
49
50 def send_data(data):
```

```
51     if(len(data)):
52         data = data.replace ("b'", "")
53         data = data.replace("'", "")
54         d = data_processing(data)
55         if (d != None):
56             t = float(d[0])
57             h = float(d[1])
58             print("Update: temperature =", t, " humidity =", h)
59             client.publish(AIO_SMARTHOMER[4], t)
60             client.publish(AIO_SMARTHOMER[5], h)
61
62     while True:
63         if(arduino.in_waiting):
64             data = str(arduino.readline())
65             send_data(data)
66             sleep(1)
```

Code 22: Hiện thực Gateway IoT bằng Python

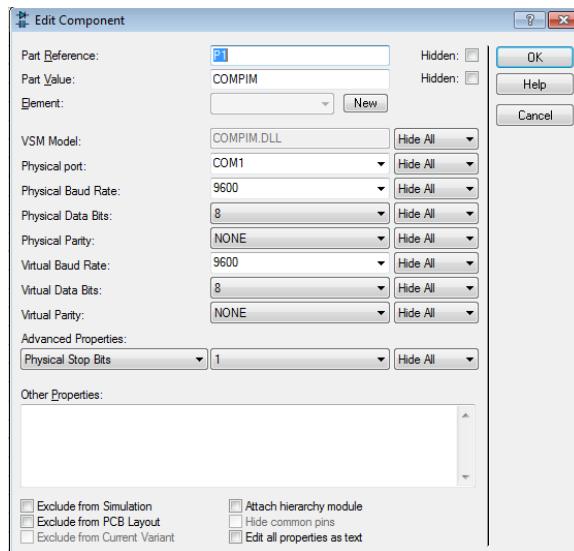
6.5 Giao tiếp giữa gateway và Arduino

- Arduino giao tiếp với gateway qua một cổng COM ảo trên Proteus.
Trên Proteus cổng COM ảo có mã thiết bị là **COMPIM**.



Hình 56: Hình ảnh cổng COM ảo trên Proteus

Sau khi nối dây hoàn tất ta tiến hành chỉnh cấu hình cho cổng COM này như sau:



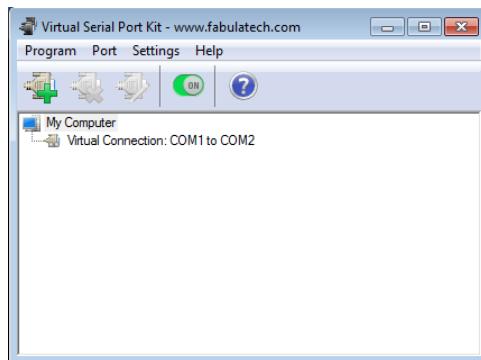
Hình 57: Hình ảnh thông số cấu hình của cổng COM ảo

Trên phần khởi tạo Serial của Arduino ta cũng config data rate là 9600.

```
1 Serial.begin(9600);
```

Code 23: Cấu hình data rate

- Gateway giao tiếp với Arduino thông qua cổng COM trên máy tính, chính vì vậy ta cần một phần mềm để tạo kết nối ảo giữa cổng COM ảo trên proteus và cổng COM trên máy tính. Nhóm em dùng phần mềm "Virtual Serial Port Kit". Trên phần mềm ta tiến hành tạo kết nối ảo từ cổng COM1(Proteus) đến cổng COM2(Máy tính), ta được kết quả như sau:



Hình 58: Hình ảnh tạo kết nối ảo giữa COM1 và COM2

Trong gateway, phần cấu hình chọn cổng serial ta cũng cấu hình cho thích hợp:

```
1 arduino = serial.Serial(port='COM2', baudrate=9600, timeout=.1)
```

Code 24: Cấu hình chọn cổng Serial



7 Một số module hỗ trợ cho các module chức năng chính của hệ thống

7.1 Module LED

Chứa các hàm dùng để khởi tạo các chân điều khiển LED và các hàm điều khiển các LED sử dụng trong hệ thống.

Các hàm trong module:

- `init_LED()`: Dùng để khởi tạo chân điều khiển LED.
- `toggleLED()`: Dùng để đảo trạng thái của LED từ bật sang tắt và ngược lại.

Code hiện thực module LED:

```
1 #include "_LED.h"
2 #include "Arduino.h"
3 #include "common.h"
4
5 // Initialization of module
6 void init_LED(){
7     pinMode(LED_PIN, OUTPUT);
8 }
9
10 // Toggle state of LED
11 void toggleLED(){
12     digitalWrite(LED_PIN, !digitalRead(LED_PIN));
13 }
```

Code 25: Hiện thực module LED

7.2 Module Button

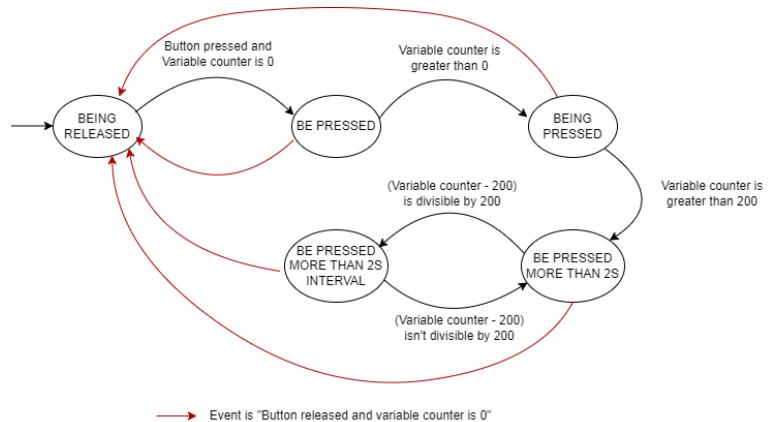
Chứa các hàm dùng để đọc tín hiệu từ nút nhấn và xử lý tín hiệu nhận được để điều khiển các thiết bị.

Đối với hàm đọc tín hiệu từ nút nhấn được xử lý chống rung 3 lần.

Các hàm trong module:

- `init_button_reading()`: Dùng để khởi tạo giá trị cho các phần tử của mảng trong module và khởi tạo các chân tín hiệu của button.
- `button_reading()`: Dùng để đọc tín hiệu từ nút nhấn.
- `fsm_button_reading()`: Dùng để xử lý tín hiệu nhận được từ việc đọc nút nhấn.

Máy trạng thái của một nút nhấn có xử lý nhấn đè 2s, sau khi nhấn đè 2s thì định kì mỗi 2s sẽ điều khiển thiết bị:

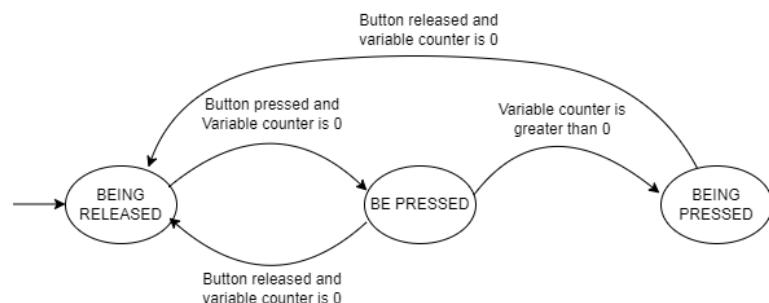


Hình 59: Máy trạng thái của nút nhấn có xử lý đè

⇒ Máy trạng thái này được áp dụng cho button 1, dùng để điều khiển LED. Khi ấn button LED sẽ đảo trạng thái của LED, nếu ấn đè hơn 2s mà vẫn còn ấn đè thì mỗi 2s nó sẽ tự động đảo trạng thái của LED.

⇒ Video demo sử dụng nút nhấn có xử lý đè để điều khiển LED: [Link video](#)

Máy trạng thái của một nút nhấn không có xử lý đè:



Hình 60: Máy trạng thái của nút nhấn không có xử lý đè

⇒ Máy trạng thái này áp dụng cho button 2, 3, 4 và 5.

Code hiện thực của module button:

```

1 #include "Arduino.h"
2 #include "_button.h"
3 #include "FAN.h"
4 #include "common.h"
5 #include "_RFID.h"
6 #include "_LED.h"
7 #include "scheduler.h"

```



```
8
9 #define PRESSED LOW
10#define RELEASED HIGH
11
12#define NUMBER_OF_BUTTONS 5
13
14enum STATE{
15    BEING_RELEASED ,
16    BE_PRESSED ,
17    BEING_PRESSED ,
18    BEING_PRESSED_MORE_1 ,
19    BE_PRESSED_MORE_1_INTERVAL ,
20};
21
22// Array save PIN value of buttons
23int buttonPin[NUMBER_OF_BUTTONS] = {BUTTON_1_PIN , BUTTON_2_PIN , ↵
24    BUTTON_3_PIN , BUTTON_4_PIN , BUTTON_5_PIN};
25
26// Array save first debounce of buttons
27int buttonDebounce1[NUMBER_OF_BUTTONS];
28// Array save second debounce of buttons
29int buttonDebounce2[NUMBER_OF_BUTTONS];
30// Array save third debounce of buttons
31int buttonDebounce3[NUMBER_OF_BUTTONS];
32// Array save counter of buttons
33int buttonCounter[NUMBER_OF_BUTTONS];
34// Array save flag of buttons
35STATE buttonFlag[NUMBER_OF_BUTTONS];
36
37// Initialization of module
38void init_button_reading(){
39    pinMode(BUTTON_1_PIN , INPUT_PULLUP);
40    pinMode(BUTTON_2_PIN , INPUT_PULLUP);
41    pinMode(BUTTON_3_PIN , INPUT_PULLUP);
42    pinMode(BUTTON_4_PIN , INPUT_PULLUP);
43    pinMode(BUTTON_5_PIN , INPUT_PULLUP);
44    for(int i = 0; i < NUMBER_OF_BUTTONS; i++){
45        buttonDebounce1[i] = RELEASED;
46        buttonDebounce2[i] = RELEASED;
47        buttonDebounce3[i] = RELEASED;
48        buttonCounter[i] = 0;
49        buttonFlag[i] = BEING_RELEASED;
50    }
51}
52
53// Function get signal from button
54void button_reading(){
55    for(int i = 0; i < NUMBER_OF_BUTTONS; i++){
56        buttonDebounce1[i] = buttonDebounce2[i];
```



```
56     buttonDebounce2[i] = buttonDebounce3[i];
57     buttonDebounce3[i] = digitalRead(buttonPin[i]);
58     if(buttonDebounce1[i] == buttonDebounce2[i] && ←
59         buttonDebounce2[i] == buttonDebounce3[i]){
60         if(buttonDebounce1[i] == PRESSED){
61             buttonCounter[i]++;
62             if(buttonCounter[i] == 1) buttonFlag[i] = BE_PRESSED;
63             else buttonFlag[i] = BEING_PRESSED;
64             // For button 1
65             if(i == 0){
66                 if((buttonCounter[i] + 1) >= (←
67                     DURATION_FOR_AUTO_OPERATION_1 / TIMER_CYCLE)){
68                     buttonFlag[i] = BEING_PRESSED_MORE_1;
69                     if(((buttonCounter[i] + 1) * TIMER_CYCLE - ←
70                         DURATION_FOR_AUTO_OPERATION_1) % ←
71                         DURATION_FOR_INTERVAL_OPERATION_1) == 0){
72                         buttonFlag[i] = BE_PRESSED_MORE_1_INTERVAL;
73                         buttonCounter[i] = (DURATION_FOR_AUTO_OPERATION_1 ←
74                             / TIMER_CYCLE) - 1;
75                     }
76                 }
77             }
78             else{
79                 buttonCounter[i] = 2;
80             }
81         }
82     }
83 }
84
85 // function call other functions when have button event
86 void fsm_button_reading(){
87     for(int i = 0; i < NUMBER_OF_BUTTONS; i++){
88         switch(buttonFlag[i]){
89             case BEING_RELEASED:
90                 break;
91             case BEING_PRESSED:
92                 break;
93             case BE_PRESSED:
94                 if(i == 0) toggleLED();
95                 if(i == 1) task_FAN_ON_OFF();
96                 if(i == 2) task_FAN_increaseMode();
97                 if(i == 3) task_FAN_auto();
98                 if(i == 4) changeStateRFID();
```



```
100     buttonFlag[i] = BEING_PRESSED;
101     break;
102 case BEING_PRESSED_MORE_1:
103     break;
104 case BE_PRESSED_MORE_1_INTERVAL:
105     toggleLED();
106     buttonFlag[i] = BEING_PRESSED_MORE_1;
107     break;
108 default:
109     break;
110 }
111 }
112 }
```

Code 26: Hiện thực module button

7.3 Module timer

Chứa các hàm liên quan đến thời gian của hệ thống.

Các hàm trong module:

- `setTimer(int index, int duration)`: dùng để set software timer thứ `index` với thời gian là `duration`.
- `getTimerFlag(int index)`: dùng để lấy giá trị của cờ software timer thứ `index`.
- `resetTimer(int index)`: dùng để reset software timer thứ `index`, sẽ set cờ và biến counter của nó là 0.
- `timer_run()`: dùng để cập nhật thời gian của software timer.
- `init_timer_software()`: dùng để khởi tạo giá trị cho các array của software timer.
- `init_timer_interrupt()`: dùng để tạo ngắt timer cho Arduino.

Code hiện thực của module timer:

```
1 #include "common.h"
2 #include "timer.h"
3 #include "Arduino.h"
4
5 #define NUMBER_OF_SOFTWARE_TIMER 5
6
7 int timerCounter[NUMBER_OF_SOFTWARE_TIMER];
8 int timerFlag[NUMBER_OF_SOFTWARE_TIMER];
9
10 void setTimer(int index, int duration){
11     if(index >= NUMBER_OF_SOFTWARE_TIMER) return;
12     timerCounter[index] = duration / TIMER_CYCLE;
13     timerFlag[index] = 0;
14 }
```

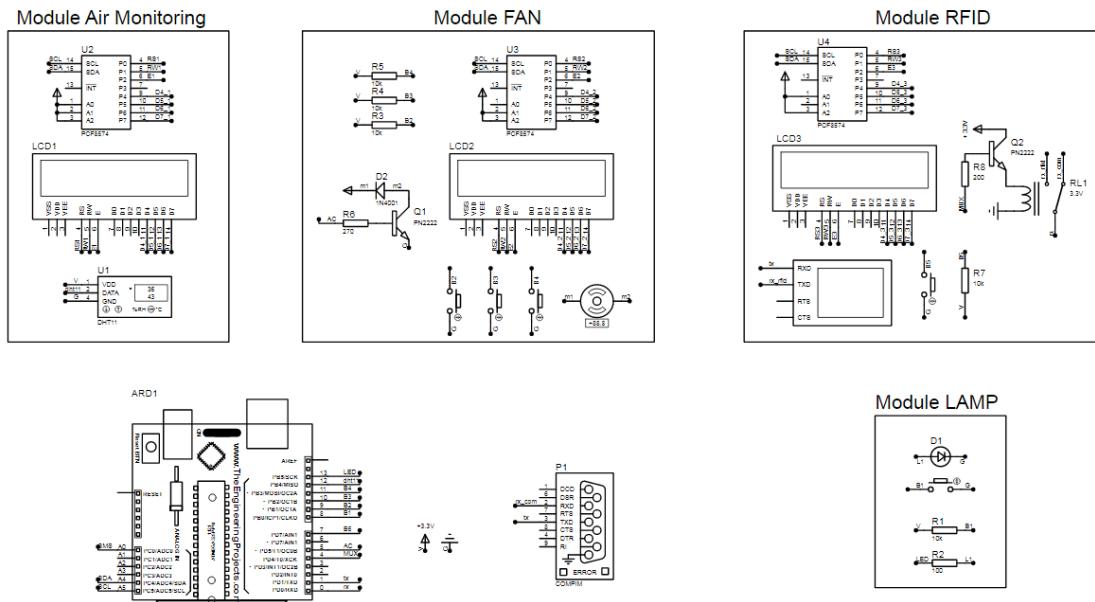


```
15
16 int getTimerFlag(int index){
17     if(index >= NUMBER_OF_SOFTWARE_TIMER) return -1;
18     return timerFlag[index];
19 }
20
21 void resetTimer(int index){
22     if(index >= NUMBER_OF_SOFTWARE_TIMER) return;
23     timerCounter[index] = 0;
24     timerFlag[index] = 0;
25 }
26
27 void timer_run(){
28     for(int i = 0; i < NUMBER_OF_SOFTWARE_TIMER; i++){
29         if(timerCounter[i] > 0){
30             timerCounter[i]--;
31             if(timerCounter[i] == 0) timerFlag[i] = 1;
32         }
33     }
34 }
35
36 void init_timer_software(){
37     for(int i = 0; i < NUMBER_OF_SOFTWARE_TIMER; i++){
38         setTimer(i, TIMER_CYCLE);
39     }
40 }
41
42 void init_timer_interrupt(){
43     cli(); //stop interrupts
44     TCCR1A = 0; // set entire TCCR1A register to 0
45     TCCR1B = 0; // same for TCCR1B
46     TCNT1 = 0; //initialize counter value to 0
47     OCR1A = 624; // = (16 * 10^6) / (100 * 256) - 1 (must be ←
48     // <65536)
49     // turn on CTC mode
50     TCCR1B |= (1 << WGM12);
51     // Set CS12 bits for 256 prescaler
52     TCCR1B |= 1 << CS12;
53     // enable timer compare interrupt
54     TIMSK1 |= (1 << OCIE1A);
55     sei(); //allow interrupts
}
```

Code 27: Hiện thực module timer

8 Hoàn thiện hệ thống nhà thông minh

8.1 Schematic trên Proteus của toàn bộ hệ thống



Hình 61: Schematic của toàn bộ hệ thống

8.2 File "common.h"

Sẽ định nghĩa các thông số để cấu hình hệ thống theo ý muốn của người sử dụng và các thông số dùng chung cho các module trong hệ thống.

Hiện thực của file common.h:

```

1 // For system, cannot config
2 #define TIMER_CYCLE 10
3
4 // For user config
5 // Scheduling
6 #define PERIOD_GATEWAY_SENDING 10000
7 #define PERIOD_DHT11_DISPLAY 10000
8 #define PERIOD_DHT11_MEASURE 2000
9 #define PERIOD_FAN_AUTO_UPDATE 10000
10
11 // LED module
12 #define LED_PIN 13
13
14 // Button module
15 #define BUTTON_1_PIN 8

```



```
16 #define BUTTON_2_PIN 9
17 #define BUTTON_3_PIN 10
18 #define BUTTON_4_PIN 11
19 #define BUTTON_5_PIN 7
20 #define DURATION_FOR_AUTO_OPERATION_1 2000
21 #define DURATION_FOR_INTERVAL_OPERATION_1 2000
22
23 // FAN module
24 #define FAN_PIN 5
25 #define MAX_FAN_TEMPERATURE 40
26 #define MIN_FAN_TEMPERATURE 27
27 #define MODE_0_SPEED 50
28 #define MODE_1_SPEED 130
29 #define MODE_2_SPEED 255
30
31 // RFID module
32 #define TIMEOUT_FOR_RFID_SESSION 3000
33 #define TIMEOUT_WAITING_INPUT 10000
34 #define TIMEOUT_LCD_ANNOUNCEMENT 500
35 #define MUX_PIN 4
```

Code 28: Hiệu thực file common.h

8.3 Chương trình chính (main)

Chương trình chính sẽ có nhiệm vụ gọi các hàm đã hiện thực ở các module để tiến hành điều khiển hoạt động của tất cả các thiết bị trong các module khác nhau.

Code hiện thực chương trình chính:

```
1 #include "FAN.h"
2 #include "timer.h"
3 #include "_DHT11.h"
4 #include "_LCD.h"
5 #include "_button.h"
6 #include "_RFID.h"
7 #include "_LED.h"
8 #include "gateway.h"
9 #include "scheduler.h"
10 #include "common.h"
11 #include <stdint.h>
12
13 // Save ID of task task_gateway_sending in scheduler
14 uint32_t ID_task_gateway;
15
16 void setup() {
17     // Call initialization function of other modules
18     init_LCD();
19     init_button_reading();
```



```
20     init_timer_interrupt();
21     init_timer_software();
22     init_LED();
23     init_DHT11();
24     init_SCH();
25     init_FAN();
26     init_RFID();
27     init_gateway();
28
29     // Scheduling tasks have period
30     SCH_Add_Task(task_measure_environment, 0, PERIOD_DHT11_MEASURE);
31     SCH_Add_Task(task_DHT11_display, 10, PERIOD_DHT11_DISPLAY);
32     ID_task_gateway = SCH_Add_Task(task_gateway_sending, 20,
33                                     PERIOD_GATEWAY_SENDING);
33 }
34
35 void loop(){
36     // Execute task if out of delay
37     SCH_Dispatch_Tasks();
38     // Call other functions when have button event
39     fsm_button_reading();
40     // Finite state machine for RFID module
41     RFID_run();
42     // Wait server requirement and call functions to do it
43     gateway_command_received_parse();
44 }
45
46 // Interrupt service routine function
47 ISR(TIMER1_COMPA_vect){
48     // Read signal from buttons
49     button_reading();
50     // Update time of software timer
51     timer_run();
52     // Update time of scheduler
53     SCH_Update();
54 }
```

Code 29: Hiện thực file main



9 Những khó khăn khi hiện thực project

- Việc dùng Proteus trong mô phỏng làm cho quá trình debug một số lỗi trở nên khó khăn, do một phần nó sẽ chạy không đúng theo thời gian thực.
- Một số module không phải mặc định của proteus gây khó khăn trong việc tìm kiếm lỗi không biết là do code hiện thực hay do thiết bị mô phỏng hoạt động sai.
- Do khoảng cách địa lý xa nên gây khó khăn trong việc trao đổi của nhóm.

10 Định hướng phát triển project trong tương lai

- Trong tương lai nếu có điều kiện thì nhóm sẽ hiện thực một hệ thống với các thiết bị thật.
- Bổ sung thêm một module wifi để có thể sử dụng độc lập hệ thống, không cần kết nối với máy tính.
- Thêm một số chức năng có tính thực tế cao hơn.
- Tự hiện thực một trang web server/một ứng dụng di động để có thể tạo ra các chức năng theo mong muốn.

11 Tài liệu đính kèm của dự án

- Toàn bộ source code của dự án: [Github](#) or [Google Drive](#)
- Toàn bộ video giới thiệu về dự án: [Click here](#)
- Project trên Proteus: [Click here](#)

12 Kết luận

Mặc dù trong thời kì giãn cách khó khăn trong việc trao đổi làm việc của nhóm, những khó khăn khi dùng phần mềm mô phỏng để hiện thực dự án nhưng nhóm cũng đã hoàn thành tốt các chức năng theo yêu cầu cũng như tự thêm một vài tính năng tự sáng tạo của nhóm. Qua dự án lần này nhóm cũng đã có thêm những kiến thức về Arduino, hiện thực một ứng dụng IOT thực thụ.

Nhóm em xin chân thành cảm ơn thầy Lê Trọng Nhân đã hỗ trợ nhiệt tình để nhóm em có thể hoàn thành được dự án lần này.



Tài liệu

- [1] Wikipedia. *Nhà thông minh*. Truy cập từ:
https://vi.wikipedia.org/wiki/Nh%C3%A0_th%C3%B4ng_minh
- [2] *Arduino Timer Interrupts*. Truy cập từ:
<https://www.instructables.com/Arduino-Timer-Interrupts/>
- [3] *Arduino: Pushing the Limits // Arduino Timers*. Truy cập từ:
<https://www.instructables.com/Arduino-Pushing-the-Limits-Arduino-Timers/>
- [4] *Arduino 101: Timers and Interrupts*. Truy cập từ:
<https://www.robotshop.com/community/forum/t/arduino-101-timers-and-interrupts/13072>
- [5] *Embedded Systems - Interrupts*. Truy cập từ:
https://www.tutorialspoint.com/embedded_systems/es_interrupts.htm
- [6] Mike Silva. *Introduction to Microcontrollers - Buttons and Bouncing*. Truy cập từ:
<https://www.embeddedrelated.com/showarticle/505.php>
- [7] Wikidoc. *Dew point*. Truy cập từ:
https://www.wikidoc.org/index.php/Dew_point#cite_note-1
- [8] *DHT11 Humidity & Temperature Sensor*. Truy cập từ:
<https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>
- [9] Hitachi. *LM016L*. Truy cập từ:
<https://datasheetspdf.com/pdf-file/1462370/Hitachi/LM016L/1>
- [10] *LCD 16 × 2 (LM016L)*. Truy cập từ:
<https://embeddedcenter.wordpress.com/ece-study-centre/display-module/lcd-16x2-lm016l/>
- [11] *Giới thiệu cơ bản về LCD 16 × 2*. Truy cập từ:
<http://thutemplate2115.blogspot.com/2015/08/d.html>
- [12] Nguyễn Chí Linh. *Chuẩn giao tiếp I2C - I2C trong PIC*. Truy cập từ:
<https://tailieumienphi.vn/doc/chuan-giao-tiep-i2c-0zmwtq.html>
- [13] Philips. *Datasheet PCF8574 - Remote 8-bit I/O expander for I2C-bus*. Truy cập từ:
<https://www.aurel32.net/elec/pcf8574.pdf>
- [14] *RFID: Công nghệ làm cho các ngành công nghiệp thông minh hơn*. Truy cập từ:
<https://smartfactoryvn.com/technology/internet-of-things/rfid-cong-nghe-lam-cho-cac-nganh-cong-nghiep-thong-minh-hon/>



-
- [15] *GIỚI THIỆU CẤU TẠO VÀ PHƯƠNG THỨC LÀM VIỆC CỦA RFID.* Truy cập từ:
<http://tueminh.tech/vn/giai-phap-RFID/gioi-thieu-cau-tao-phuong-thuc-lam-viec-RFID.html>
 - [16] *5V Relay, Pinout, Description, Working & Datasheet.* Truy cập từ:
<https://components.monofindia.com/5v-relay-pinout-and-working-sheet/>
 - [17] *Songle Relay.* Truy cập từ:
https://components101.com/sites/default/files/component_datasheet/5V%20Relay%20Datasheet.pdf
 - [18] *Công nghệ RFID là gì? Ứng dụng của RFID.* Truy cập từ:
<https://plctech.com.vn/cong-nghe-rfid-la-gi-ung-dung-cua-rfid/>
 - [19] *Kiến thức cơ bản về giao tiếp UART (Serial) trong Arduino.* Truy cập từ:
<https://espace.edu.vn/tu-hoc-arduino/kien-thuc-co-ban-ve-giao-tiep-uart-serial-trong-arduino/>
 - [20] Michael J Pont. *Patterns for Time-Triggered Embedded Systems*, Pearson Education.
 - [21] Jacob Beningo. *7 steps to writing a simple cooperative scheduler.*, Truy cập từ:
<https://www.edn.com/7-steps-to-writing-a-simple-cooperative-scheduler/>
 - [22] Perry Lea. *Internet of Things for Architects*, Packt, ISBN 978-1-78847-059-9.