ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO BÀI TẬP LỚN

LẬP TRÌNH HỆ THỐNG NHÚNG

Giảng viên hướng dẫn:
Nguyễn Phan Hải Phú

Sinh viên thực hiện:
Tăng Nguyễn Nhật Quỳnh - 2212896
Trần Quốc Bảo – MSSV: 2110802
Danh Bình – MSSV: 2112895

TP. Hồ Chí Minh, Tháng 12 năm 2025

Mục lục

1 Usart

1.1 Giới thiệu tổng quan

USART (Universal Synchronous/ Asynchronous Receiver Transmitter) là một trong những thành phần ngoài vi truyền thông nối tiếp phổ biến trong hệ thống vi điều khiển. Ngoại vi này được thiết kế nhằm thực hiện việc truyền và nhận dữ liệu nối tiếp giữa vi điều khiển (MCU) và các thiết bị bên ngoài, giúp cho việc trao đổi thông tin trở nên dễ dàng, tiết kiệm chân kết nối và đảm bảo độ tin cậy cao. USART thường được sử dụng để giao tiếp giữa MCU với máy tính, module truyền thông (Wi-Fi, Bluetooth,...), các cảm biến hoặc giữa các vi điều khiển với nhau. Trong hệ thống truyền thông nối tiếp, dữ liệu được truyền từng bit một theo thứ tự thời gian thay vì truyền song song nhiều bit cùng lúc. Cách truyền này giúp làm giảm số lượng dây kết nối, đặc biệt hữu ích đối với các hệ thống nhúng có kích thước nhỏ hoặc yêu cầu tiết kiệm tài nguyên phần cứng. Để đảm bảo cho việc truyền dữ liệu chính xác, hai thiết bị giao tiếp với nhau cần thống nhất các thông số như tốc độ truyền, số liệu bit dữ liệu, bit kiểm tra và bit dừng,.... Trên kit REnesas CK-RA6M5, ngoại vi truyền thông USART được tích hợp trong khối SCI (Serial Communication Interface)- một module truyền thông đa năng được thiết kế riêng dành cho dòng vi điều khiển Renesas RA series. Khối SCI có khả năng cấu hình linh hoạt để hoạt động ở nhiều dạng giao tiếp khác nhau như UART/USART, Simple SPI hay Simple I2C,... tùy theo yêu cầu của người lập trình. Mỗi kênh SCI có thể hoạt động độc lập cho phép MCU giao tiếp cùng lúc với nhiều thiết bị khác nhau. Ngoại vi USART được sử dụng trong hầu hết các hệ thống nhúng bởi tính đơn giản, sự ổn định cũng như giá thành thấp. Việc truyền dữ liệu giữa MCU và máy tính sẽ được kiểm tra, hiển thị và ghi lại thông tin qua cổng terminal. Phần ngoại vi này còn cho phép vừa giao tiếp với các module truyền thông ở các thiết bị khác thông qua giao thức nối tiếp vừa cho phép giao tiếp và trao đổi dữ liệu giữa các vi. Tổng kết lại, USART là phần ngoại vi truyền thông cơ bản nhưng lại có vai trò lớn trong hệ thống nhúng. Trên kit Renesas CKRA6M5, ngoại vi này được triển khai thông qua khối SCI có khả năng cấu hình linh hoạt, tốc độ cao, dễ sử dụng và tương thích với nhiều loại thiết bị ngoại vi khác nhau. Việc nắm vững nguyên lý hoạt động và cách cấu hình USART trên bộ kit là nền tảng quan trọng để xây dựng các ứng dụng truyền thông nối tiếp trong lập trình hệ thống nhúng.

1.2 Mô tả về các thanh ghi liên quan đến ngoại vi

Như đã trình bày từ trước, ngoại vi USART được triển khai thông qua khối SCI. Khối này được cấu thành bởi nhiều thanh ghi điều khiển và trạng thái góp phần cấu hình vận hành và giám sát quá trình truyền - nhận dữ liệu một cách hiệu quả. Việc hiểu rõ chức năng của các thanh ghi này là cơ sở để lập trình giao tiếp USART. Dưới đây là mô tả về các thanh ghi quan trọng liên quan đến hoạt động của ngoại vi USART trên RA6M5.

1.2.1 SCR (Serial Control Register)

Đây là thanh ghi điều khiển chính của USART, cho phép việc điều khiển các chức năng cơ bản như:

- TE (Transmit Enable): Cho phép khối bật chức năng truyền. Khi TE = 1, module cho phpes ghi TDR và bắt đầu gửi.

- RE (Receive Enable): Cho phép khối bật chức năng nhận hoạt động. RE = 1, module chấp nhận khung vào RDR.
- RIE (Receive Interrupt Enable): Cho phép ngắt khi RDRF=1 (dữ liệu nhận xong).
- TEIE, REIE: Cho phép ngắt khi đã truyền hoàn tất hoặc nếu có lỗi (overrun, framing, parity).

Thanh ghi SCR là nơi đầu tiên được cấu hình khi khởi tạo USART.

1.2.2 SMR (Serial Mode Register).

Thanh ghi này xác định chế độ hoạt động cũng như định dạng khung truyền cho ngoại vi USART.

- CM (Communication Mode): Lựa chọn giữa chế độ đồng bộ (1) hoặc bất đồng bộ (0).
- CHR (Character Length): Chọn độ dài dữ liệu (7 hoặc 8 bit).
- PE, PM (Parity Enable/Mode): Bật/tắt và chọn chế độ chẵn/lẻ của bit chẵn lẻ (parity bit).
- STOP: Chọn số lượng bit dừng (1 hoặc 2 bit).
- CKS (Clock Select): Chọn nguồn clock nội hoặc ngoại, đồng thời xác định tốc độ baud.

Thanh ghi SMR thường sẽ được cấu hình ngay sau thanh ghi chính SCR để xác lập định dạng truyền.

1.2.3 SEMR (Serial Extended Mode Register).

Thanh ghi SEMR mở rộng cấu hình tốc độ baud và tăng chất lượng truyền nhận và được dùng khi cần baud rate cao hoặc để chống nhiễu.

- ABCS (Asynchronous Base Clock Select): chọn tỉ lệ phân mảnh clock divisor cho BRR (ví dụ chia 1 hoặc 16), từ đó cho phép khối hoạt động ổn định hơn ở baud rate cao.
- BGDM (Baud Rate Generator Double-speed Mode): bật double speed để kích hoạt chế độ nhân đôi tốc độ. Khi BGDM=1, công thức tính BRR sẽ xảy ra thay đổi theo hướng làm giảm giá trị BRR để làm tăng baud rate.
- BRME (Baud Rate Modulation Enable): bật modulation để tinh chỉnh baud chính xác hơn.
- NFEN (Noise filter enable): bật bộ lọc nhiễu cho RX (hữu ích cho môi trường nhiễu cao).

1.2.4 BRR (Bit Rate Register).

Thanh ghi này được dùng để thiết lập tốc độ baud của đường truyền nối tiếp. Giá trị trong BRR được tính toán dựa trên tần số clock hệ thống và tốc độ baud mong muốn theo công thức:

$$\text{BRR} = \left(\frac{PCLK}{divisor \times Baudrate}\right) - 1$$

Giá trị BRR càng nhỏ, tốc độ truyền càng cao.

1.2.5 SSR (Serial Status Register).

Đây là thanh ghi trạng thái phản ánh tình trạng hiện tại của USART với các thành phần như sau:

- TDRE (Transmit Data Register Empty): Cho biết bộ đệm truyền đã sẵn sàng cho byte tiếp theo.
- RDRF (Receive Data Register Full): Cho biết có dữ liệu mới đã nhận xong và sẵn sàng đọc.
- ORER (Overrun Error): Báo lỗi tràn bộ đệm nhận (khi dữ liệu mới đến nhưng chưa đọc dữ liệu cũ).
- PER, FER (Parity Error/Framing Error): Báo lỗi parity hoặc framing trong quá trình truyền nhận.
- MPB / LBD (multi-processor/address mark): Nếu hỗ trợ chế độ multi-processor.

Thanh ghi SSR thược được kiểm tra trong vòng lặp của chương trình hoặc trong các hàm ngắt để có thể xử lý dữ liệu đúng thời điểm.

1.2.6 TDR (Transmit Data Register) và RDR (Receive Data Register).

Hai thanh ghi này là cổng dữ liệu chính của USART.Trong đó:

- TDR: Nơi ghi dữ liệu cần truyền đi. Mỗi lần ghi vào TDR, phần cứng sẽ tự động chuyển dữ liệu ra đường truyền TX.
- RDR: Nơi chứa dữ liệu vừa nhận được. Khi bit RDRF trong SSR bật, chương trình có thể đọc giá trị từ RDR.

1.2.7 SCMR (Smart Card Mode Register).

Thanh ghi này được dùng trong các ứng dụng đặc biệt như giao tiếp với thẻ thông minh. Cho phép mở rộng cấu hình truyền, bit stop và bit parity đặc thù. Cần được bật trước khi sử dụng và tắt khi không dùng đến.

1.2.8 SEMR (Serial Extended Mode Register).

Cung cấp các tùy chọn nâng cao cho cấu hình USART:

- ABCS (Clock Select): Điều chỉnh tỷ lệ chia clock (8x hoặc 16x baud rate).
- BRME (Bit Rate Modulation Enable): Cho phép hiệu chỉnh chính xác tốc độ baud.
- NFEN (Noise Filter Enable): Bật bộ lọc nhiễu tín hiệu đầu vào RX.

1.3 Mô tả các chế độ của ngoại vi USART.

Ngoại vi USART trong vi điều khiển Renesas RA6M5 là một khối truyền thông nối tiếp linh hoạt, có khả năng hoạt động ở nhiều chế độ khác nhau tùy theo yêu cầu của ứng dụng, Việc lựa chọn chế độ phù hợp với hoạt động sẽ tối ưu tốc đồ truyền tải dữ liệu, đảm bảo sự ổn định và khả năng tương thích của hệ thống. Dưới đây là các chế dộ hoạt động của USART trong RA6M5:

1.3.1 Chế độ bất đồng bộ (Asynchronous Mode).

Đây là chế độ phổ biến nhất và cũng là chế độ mặc định của USART. Trong chế độ này, việc truyền và nhận dữ liệu không sử dụng tín hiệu xung nhịp chung (clock) giữa hai thiết bị. Cả hai bên chỉ cần thống nhất trước tốc độ baud (baud rate) và định dạng khung truyền. Dữ liệu được truyền từng khung gồm:

1 bit start + 7/8 bit dữ liệu + (tùy chọn) 1 bit parity + 1 hoặc 2 bit stop

Tốc độ baud được xác định bởi thanh ghi BRR. Chế độ này rát tiện dụng khi giao tiếp với các thiết bị phần cứng phổ biến như máy tính (qua USB-Serial), Bluetooth hay Wi-FI. Ưu điểm của chế độ này là dễ triển khai mà không cần đồng hồ chung, chỉ cần hai dây truyền thông là TX và RX và có độ tương thích cao với nhiều chuẩn giao tiếp thông dụng. Ngược lại, chế độ này có một số hạn chế như việc dễ bị sai lệch bit nếu hai thiết bị giao tiếp bị lệch tốc độ baud quá lớn, không đạt được tốc độ cao như chế độ đồng bộ.

1.3.2 Chế độ đồng bộ (Synchronous Mode).

Ở chế độ này, USART sử dụng thêm một tín hiệu clock đồng bộ (SCK) để đảm bảo dữ liệu được truyền và nhận theo cùng một xung nhịp. Điều này giúp cải thiện độ chính xác và tốc độ truyền dữ liệu. Chế độ này sẽ sử dụng ba đường truyền là TX, RX và SCK (xung clock). Dữ liệu truyền đi sẽ được lấy mẫu theo biên của tín hiệu clock, một thiết bị sẽ giữ vai trò Master (phát clock còn thiết bị kia sẽ là Slave (nhận clock). Ưu điểm chính của chế độ này là tốc độ truyền tải và độ tin cậy cao hơn so với chế độ bất đồng bộ vì dữ liệu ở đây được đồng bộ chính xác theo clock. Điều này đặc biệt hữu ích trong các ứng dụng yêu cầu tốc độ truyền nhanh và độ chính xác cao. Tuy nhiên, chế độ này cũng có một số hạn chế như cần phải có thêm dây dẫn clock làm tăng số lượng chân giao tiếp, khoảng cách truyền dữ liệu bị giới hạn và thiết kế mạch sẽ phức tạp hơn.

1.3.3 Chế độ tốc độ cao (High-Speed Baud Rate Mode).

Chế độ này cho phép tăng tốc đáng kể tốc độ truyền bằng cách giảm hệ số chia clock (divisor). Việc này được thực hiện thông qua các bit cấu hình trong thanh ghi SEMR, giúp cho ngoại vi USART có thể hoạt động ở các mức baud cao như 115200, 230400,.. hoặc trên 1 Mbps. Ở chế độ này, hệ thông thường sẽ kích hoạt thêm các tính năng lọc nhiễu và điều chỉnh độ chính xác baud. Đây là chức năng quan trọng trong những ứng dụng truyền tải dữ liệu tốc độ cao như trong cảm biến tốc độ cao hay module WiFi.

1.3.4 Chế độ kiểm tra và xử lý lỗi.

Trong quá trình truyền và nhận, việc xảy ra lỗi là điều khó tránh khỏi nên việc trang bị thêm chế độ này với khả năng phát hiện các lỗi phổ biến như lỗi parity, lỗi framing hay lỗi tràn bộ đệm nhận. Các lỗi này sẽ được ghi nhận trong thanh ghi trạng thái (SSR) và có thể kích hoạt ngắt để hệ thống xử lý kịp thời. Cơ chế này đảm bảo chất lượng dữ liệu, nhất là trong các môi trường gây nhiễu hay khi truyền trên khoảng cách dài.

1.3.5 Chế độ truyền một dây (Single-Wire/Half-Duplex).

Cho phép sử dụng chung một chân cho cả truyền và nhận. Mặc dù không thể truyền và nhận cùng lúc nhưng chế độ này lại hữu ích khi các thiết bị có số chân hạn chế hoặc các giao tiếp đơn giản chỉ cần một dây dẫn.

1.3.6 Chế độ truyền nhiều điểm (Multiprocessor Mode).

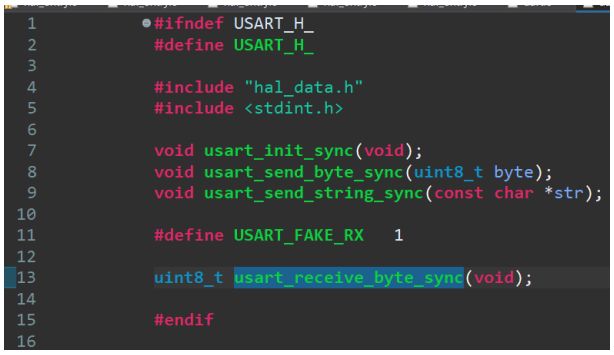
Nơi nhiều thiết bị có thể cùng chia sẻ một đường UART. Trong chế độ này, mỗi khung dữ liệu đều có thể chứa thông tin địa chỉ, giúp các thiết bị không liên quan có thể bỏ qua dữ liệu không dành cho mình. Chế độ này đặc biệt hiệu quả trong các mạng nhiều nút (peer).

1.4 Chương trình ví dụ về USART.

Mục tiêu của phần này là trình bày cách triển khai ngoại vi USART trên mạch CK-RA6M5 thông qua hai cách là FSP và lập trình thanh ghi.

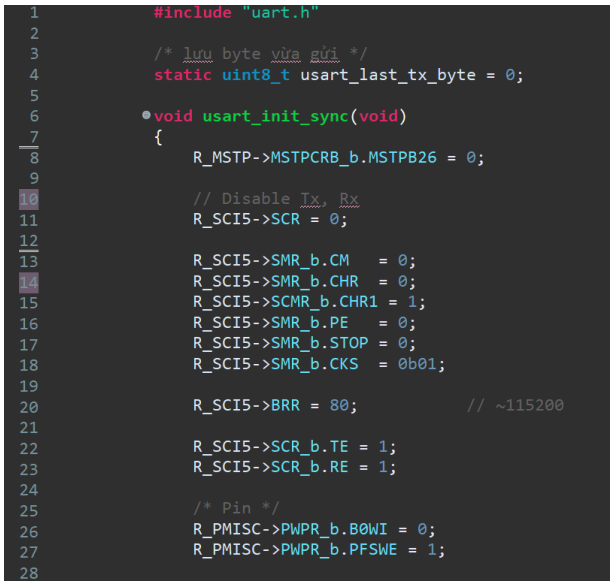
1.4.1 Lập trình thanh ghi.

Đối với lập trình thanh ghi, hệ thống được tổ chức theo hướng module hóa với hai thành phần chính. Module thứ nhất là uart.c và uart.h, đóng vai trò như một lớp trừu tượng cung cấp các hàm khởi tạo và giao tiếp USART đồng bộ theo phương pháp polling. Module thứ hai là hal_entry.c, nơi chứa hàm hal_entry() – điểm bắt đầu của chương trình được dùng để kiểm thử hoạt động của USART thông qua việc gửi và nhận dữ liệu. Cách tổ chức này giúp chương trình dễ đọc, dễ bảo trì và có khả năng mở rộng trong các phiên bản sau.



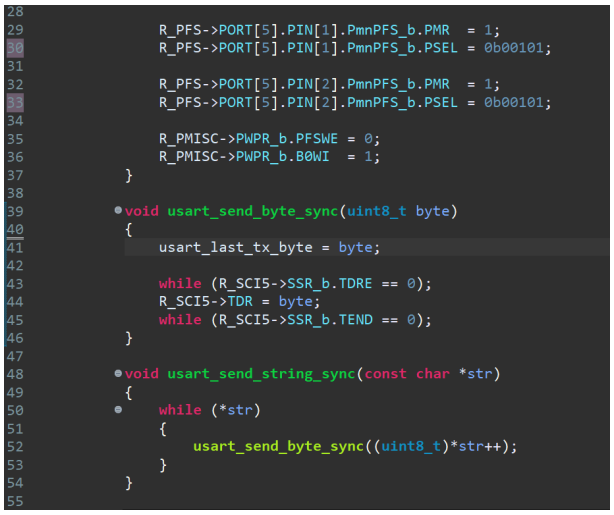
```c
#ifndef USART_H_
#define USART_H_

#include "hal_data.h"
#include <stdint.h>

void usart_init_sync(void);
void usart_send_byte_sync(uint8_t byte);
void usart_send_string_sync(const char *str);

#define USART_FAKE_RX   1

uint8_t usart_receive_byte_sync(void);

#endif

```

Hình 1: Usart_thanh ghi.h



```c
#include "uart.h"

/* lưu byte vừa gửi */
static uint8_t usart_last_tx_byte = 0;

void usart_init_sync(void)
{
    R_MSTP->MSTPCRB_b.MSTPB26 = 0;

    // Disable Tx, Rx
    R_SCI5->SCR = 0;

    R_SCI5->SMR_b.CM   = 0;
    R_SCI5->SMR_b.CHR  = 0;
    R_SCI5->SCMR_b.CHR1 = 1;
    R_SCI5->SMR_b.PE   = 0;
    R_SCI5->SMR_b.STOP = 0;
    R_SCI5->SMR_b.CKS  = 0b01;

    R_SCI5->BRR = 80;           // ~115200

    R_SCI5->SCR_b.TE = 1;
    R_SCI5->SCR_b.RE = 1;

    /* Pin */
    R_PMISC->PWPR_b.B0WI = 0;
    R_PMISC->PWPR_b.PFSWE = 1;

```

Hình 2: Usart_thanh ghi.c_P1



```c
    R_PFS->PORT[5].PIN[1].PmnPFS_b.PMR  = 1;
    R_PFS->PORT[5].PIN[1].PmnPFS_b.PSEL = 0b00101;

    R_PFS->PORT[5].PIN[2].PmnPFS_b.PMR  = 1;
    R_PFS->PORT[5].PIN[2].PmnPFS_b.PSEL = 0b00101;

    R_PMISC->PWPR_b.PFSWE = 0;
    R_PMISC->PWPR_b.B0WI  = 1;
}

void usart_send_byte_sync(uint8_t byte)
{
    usart_last_tx_byte = byte;

    while (R_SCI5->SSR_b.TDRE == 0);
    R_SCI5->TDR = byte;
    while (R_SCI5->SSR_b.TEND == 0);
}

void usart_send_string_sync(const char *str)
{
    while (*str)
    {
        usart_send_byte_sync((uint8_t)*str++);
    }
}

```

Hình 3: Usart_thanh ghi.c_P2

Trong quá trình khởi tạo USART, chương trình trước hết thực hiện cấp xung clock cho module SCI5 bằng cách đưa bit tương ứng trong thanh ghi Module Stop Control về trạng thái cho phép hoạt động. Sau đó, bộ truyền và bộ nhận được tắt tạm thời nhằm đảm bảo quá trình cấu hình không gây ra lỗi truyền thông. Module SCI5 được cấu hình làm việc ở chế độ Asynchronous USART, sử dụng khung truyền 8 bit dữ liệu, không bit parity, 1 bit stop và không đảo mức logic. Bộ chia clock được thiết lập phù hợp để tạo ra tốc độ baud xấp xỉ 115200 bps thông qua thanh ghi BRR, đáp ứng yêu cầu truyền thông tốc độ cao trong các hệ thống nhúng phổ biến. Sau khi hoàn tất cấu hình thông số truyền, chương trình kích hoạt lại bộ phát và bộ thu USART. Song song đó, các chân GPIO liên quan đến SCI5 cũng được cấu hình thông qua thanh ghi PFS. Cụ thể, chân P5.1 được gán chức năng TXD5 và chân P5.2 được gán chức năng RXD5, cho phép vi điều khiển truyền và nhận dữ liệu USART qua các chân vật lý tương ứng. Việc mở và khóa quyền ghi PFS được thực hiện đúng trình tự nhằm đảm bảo an toàn cấu hình phần cứng.Quá trình truyền dữ liệu USART được triển khai theo phương pháp polling, tức là CPU liên tục kiểm tra trạng thái của các cờ trong thanh ghi trạng thái của SCI5.

Khi gửi một byte dữ liệu, chương trình chờ đến khi thanh ghi truyền trống, sau đó ghi dữ liệu vào thanh ghi TDR và tiếp tục chờ cho đến khi quá trình truyền hoàn tất. Việc gửi chuỗi ký tự được thực hiện bằng cách lặp qua từng ký tự trong chuỗi và gửi lần lượt từng byte, đảm bảo dữ liệu được truyền đúng thứ tự. Ở chiều nhận dữ liệu, chương trình hiện tại sử dụng cơ chế nhận giả lập nhằm phục vụ mục đích kiểm thử logic hệ thống khi chưa triển khai phần nhận USART thực tế. Byte dữ liệu vừa được truyền đi sẽ được lưu lại và trả về khi hàm nhận được gọi, tạo ra cơ chế echo đơn giản. Cách tiếp cận này giúp kiểm tra nhanh luồng xử lý của chương trình và có thể dễ dàng thay thế bằng cơ chế nhận thực thông qua thanh ghi RDR và cờ RDRF trong các bước phát triển tiếp theo. Trong hàm hal_entry, sau khi USART được khởi tạo thì chương trình sẽ gửi một chuỗi thông báo xác nhận rằng giao tiếp USART bất đồng bộ đã sẵn sàng hoạt động. Tiếp theo, một ký tự kiểm tra được gửi đi để đảm bảo đường truyền hoạt động ổn định. Trong vòng lặp vô hạn, chương trình liên tục thực hiện thao tác nhận dữ liệu và gửi lại chính dữ liệu đó, tạo thành chức năng echo. Cơ chế này thường được sử dụng để kiểm tra nhanh tính đúng đắn của cấu hình USART và đường truyền vật lý.

```c
#include "hal_data.h"
#include "uart.h"

void hal_entry(void)
{
    usart_init_sync();

    usart_send_string_sync("USART ASYN OK\r\n");
    usart_send_byte_sync('A');
    while (1)
    {

        uint8_t c = usart_receive_byte_sync();
        usart_send_byte_sync(c);
    }
}
```

Hình 4: Usart_thanh ghi_hal entry

1.4.2 Lập trình FSP

Chương trình được xây dựng nhằm mục tiêu triển khai và kiểm chứng hoạt động của USART ở chế độ bất đồng bộ (Asynchronous USART) trên vi điều khiển Renesas thông qua driver UART của FSP. Khác với cách tiếp cận truy cập thanh ghi trực tiếp, giải pháp này sử dụng các API do FSP cung cấp, cho phép giảm độ phức tạp khi cấu hình phần cứng, đồng thời tăng tính ổn định và khả năng tái sử dụng của chương trình trong các hệ thống nhúng thực tế. Việc khởi tạo USART được thực hiện thông qua hàm usart_init(), trong đó API R_SCI_UART_Open() được gọi với các tham số đã được cấu hình sẵn trong FSP Configuration Tool, bao gồm tốc độ baud, số bit dữ liệu, parity và stop bit. Sau khi khởi tạo, module USART sẵn sàng cho việc truyền và nhận dữ liệu bất đồng bộ. Cơ chế truyền và nhận dữ liệu được xây dựng dựa trên ngắt và callback. Khi nhận được dữ liệu, driver FSP phát sinh sự kiện UART_EVENT_RX_CHAR, dữ liệu được lưu lại và cờ trạng thái được thiết lập để thông báo cho chương trình chính. Khi quá trình truyền hoàn tất, sự kiện UART_EVENT_TX_COMPLETE được sử dụng để xác nhận việc gửi dữ liệu thành công. Các hàm gửi và nhận trong chương trình sử dụng cơ chế chờ cờ để đảm bảo tính đồng bộ trong xử lý.

```c
#include "uart.h"

static volatile uint8_t rx_data;
static volatile bool rx_flag = false;
static volatile bool tx_done = false;

void uart_callback(uart_callback_args_t *p_args)
{
    if (p_args->event == UART_EVENT_RX_CHAR)
    {
        rx_data = (uint8_t)p_args->data;
        rx_flag = true;
    }
    else if (p_args->event == UART_EVENT_TX_COMPLETE)
    {
        tx_done = true;
    }

}

void usart_init(void)
{
    R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
}

void usart_send_byte(uint8_t data)
{
```

Hình 5: Usart_FSP_c

```c
void usart_init(void)
{
    R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
}

void usart_send_byte(uint8_t data)
{
    tx_done = false;
    R_SCI_UART_Write(&g_uart0_ctrl, &data, 1);
    while (!tx_done);
}

void usart_send_string(const char *str)
{
    while (*str)
    {
        usart_send_byte((uint8_t)*str);
        str++;
    }
}

uint8_t usart_receive_byte(void)
{
    while (!rx_flag);
    rx_flag = false;
    return rx_data;
}
```

Hình 6: Usart_FSP_c_P2

Trong hàm hal_entry(), chương trình gửi một ký tự kiểm tra, nhận lại dữ liệu và thực hiện gửi echo để kiểm chứng đường truyền hai chiều. Dựa trên kết quả nhận được, chương trình gửi thông báo xác nhận thành công hoặc thất bại của quá trình truyền nhận.

Hình 7: Usart_Fsp_Hal entryċ

2 Timer

2.1 Giới thiệu tổng quan.

2.1.1 Timer

Trong các hệ thống vi điều khiển hiện đại ngày nay, Timer là một trong những ngoại vi cơ bản nhưng quan trọng được dùng để quản lý thời gian và thực hiện các tác vụ theo chu kỳ. Về bản chất, Timer là một bộ đếm nội tại (counter) được xung nhịp từ clock hệ thống hoặc clock ngoài, có thể đếm lên hoặc đếm xuống tùy theo sự điều chỉnh của người dùng. Timer cho phép vi điều khiển tạo ra các delay chính xác, điều khiển các sự kiện định kỳ, tạo ngắt theo thời gian hoặc xuất tín hiệu điều chế xung (PWM) để điều khiển động cơ, LED, buzzer,.... Có thể hiểu đơn giản rằng Timer được lập trình để hoạt động như một bộ đếm thời gian cơ bản, hoặc nâng cao hơn như bộ đếm đa năng (general timer) với các chế độ so sánh, capture hay hoạt động bất đồng bộ để tiết kiệm năng lượng. Nhờ tính linh hoạt và khả năng điều khiển thời gian chính xác, Timer là ngoại vi không thể thiếu trong các ứng dụng nhúng, từ việc điều khiển thời gian trong các hệ thống RTOS cho đến các ứng dụng điều khiển thiết bị ngoại vi. Hiện này, hai loại timer phổ biến được sử dụng rộng rãi trong các vi điều khiển hiện đại là Systick và AGT.

2.1.2 Systick.

Systick là một timer hệ thống được tích hợp sẵn trong lõi ARM Cortex-M được hoạt động như một bộ đếm giảm dần nội tại với độ chính xác cao. Nó chủ yếu được sử dụng để tạo tick cơ bản cho hệ thống, phục vụ cho lập lịch trong các hệ điều hành thời gian thực (RTOS) hay thực hiện các delay ngắn trong các ứng dụng nhúng. Systick có cấu trúc đơn giản với các thanh ghi chính bao gồm:

- LOAD: lưu giá trị nạp ban đầu của bộ đếm.

- VAL: lưu giá trị hiện tại của bộ đếm, giảm dần theo xung nhịp.
- CTRL: điều khiển hoạt động của timer, bao gồm bật/tắt, chọn nguồn xung nhịp, bật ngắt khi bộ đếm về 0.

Nguyên lý hoạt động của Systick rất trực quan: khi nạp giá trị vào LOAD thì bộ đếm VAL sẽ giảm dần theo xung nhịp hệ thống. Khi VAL bằng 0, Systick có thể phát sinh ngắt và đồng thời tự nạp lại giá trị LOAD nếu chọn chế độ lặp lại. Nhờ nguyên lý hoạt động trên, Systick thường được dùng để thực hiện các tác vụ định kỳ, tạo sự delay chính xác hoặc cung cấp xung tick cho các RTOS giúp quản lý đa nhiệm và đồng bộ hóa các quá trình trong hệ thống.

2.1.3 AGT (Advanced/Asynchronous General Timer)

Ngược lại với Systick, AGT là một timer tổng quát, mạnh mẽ và linh hoạt hơn Systick thích hợp cho các ứng dụng yêu cầu điều khiển phức tạp hoặc tạo tín hiệu PWM. AGT có khả năng hoạt động đồng bộ hoặc bất đồng bộ với CPU, điều này giúp giảm tiêu thụ năng lượng khi cần. Ngoài ra, AGT hỗ trợ nhiều chế độ đếm như: đếm lên, đếm xuống, hoặc đếm lên-xuống, đồng thời tích hợp các chức năng capture (ghi lại thời điểm tín hiệu đầu vào), compare (so sánh bộ đếm với giá trị tham chiếu) và phát sinh ngắt khi đạt giá trị mong muốn. Cấu trúc cơ bản của AGT bao gồm:

- CNT: thanh ghi bộ đếm hiện tại.
- PR (Period Register): giá trị tham chiếu hoặc chu kỳ bộ đếm.
- TCSR/TIER: thanh ghi điều khiển timer và bật ngắt.
- TIO/Output Pin: dùng để xuất tín hiệu PWM hoặc tín hiệu đồng bộ với bộ đếm.

Nguyên lý hoạt động của AGT là: khi nạp giá trị PR, CNT sẽ tăng hoặc giảm theo xung nhịp được tùy chỉnh. Khi CNT đạt đến PR, AGT sẽ phát sinh ngắt, đảo tín hiệu PWM hoặc reset bộ đếm tùy theo chế độ cấu hình. Nhờ vào tính linh hoạt trên, AGT có thể được dùng để tạo delay chính xác, điều khiển động cơ, LED, buzzer hoặc đo tần số và độ rộng xung tín hiệu đầu vào.

2.2 Mô tả thanh ghi của ngoại vi.

2.2.1 Systick.

Như đã đề cập ở trên, Systick là một timer được cấu thành từ ba thanh ghi chính điều khiển toàn bộ hoạt động là: CTRL, LOAD và Val cùng với một thanh ghi phụ là CALIB.

- Systick Control and Status Register (CTRL)

  Thanh ghi CTRL dùng để điều khiển hoạt động của Systick và trạng thái ngắt. Các bit quan trọng gồm:

  - ENABLE (bit 0): Bật/tắt bộ đếm. Khi ENABLE = 1, Systick bắt đầu giảm từ giá trị LOAD xuống 0.

- TICKINT (bit 1): Bật/tắt ngắt khi bộ đếm VAL về 0. Khi TICKINT = 1, Systick phát sinh ngắt cho CPU.
- CKSOURCE (bit 2): Chọn nguồn xung nhịp cho Systick. Có thể chọn clock hệ thống (CPU) hoặc clock bên ngoài.
- COUNTFLAG (bit 16): Cờ trạng thái cho biết bộ đếm đã tràn về 0 trong chu kỳ vừa qua. Bit này được đọc tự động khi cần kiểm tra trạng thái.

- Systick Reload Value Register (LOAD). Thanh ghi LOAD của Systick là một thành phần cốt lõi quyết định chu kỳ đếm của timer và ảnh hưởng trực tiếp đến thời gian phát sinh ngắt hoặc tick hệ thống. Khi Systick được bật hoặc khi bộ đếm hiện tại (VAL) giảm xuống 0 trong chế độ lặp lại, giá trị trong LOAD sẽ được nạp vào VAL, khởi tạo một chu kỳ mới. Giá trị trong LOAD được tính theo số xung nhịp của clock nguồn, và chu kỳ thời gian thực được xác định theo công thức:

$$\mathrm{T}_{\mathrm{tick}} = \frac{LOAD+1}{f_{\mathrm{CLK}}}$$

  Trong đó fCLKT là tần số xung nhịp của timer. Ví dụ, nếu sử dụng clock 48 MHz và muốn tạo tick 1 ms, giá trị LOAD sẽ là 47.999, cho phép Systick phát sinh ngắt đều đặn mỗi 1 ms. LOAD có độ rộng 24 bit, cho phép tạo chu kỳ rất dài, đồng thời giá trị này chỉ nạp vào VAL khi bắt đầu hoặc khi VAL về 0, nên việc thay đổi LOAD trong quá trình chạy sẽ điều chỉnh chu kỳ ngắt mà không cần tắt timer. Nhờ đó, LOAD đóng vai trò quyết định thời gian định kỳ của Systick, giúp thực hiện delay chính xác, quản lý tick hệ thống và lập lịch trong các ứng dụng RTOS

- Systick Current Value Register (VAL) Thanh ghi VAL của Systick là thanh ghi lưu giá trị hiện tại của bộ đếm và phản ánh trạng thái thời gian còn lại trong chu kỳ hiện tại. Khi Systick được bật, VAL sẽ giảm dần từ giá trị nạp từ thanh ghi LOAD xuống 0 theo từng xung nhịp của clock nguồn. Khi VAL đạt 0, Systick có thể phát sinh ngắt nếu bit TICKINT trong thanh ghi CTRL được bật, đồng thời VAL sẽ được nạp lại từ LOAD nếu chế độ lặp lại được kích hoạt tạo nên một chu kỳ định thời liên tục. Thanh ghi VAL có thể được đọc bất cứ lúc nào để kiểm tra thời gian còn lại trong chu kỳ hiện tại tuy nhiên việc ghi trực tiếp vào VAL sẽ reset bộ đếm về giá trị mới có thể làm ảnh hưởng đến vi điều khiển. Nhờ vai trò này, VAL cho phép lập trình viên giám sát và quản lý chính xác quá trình đếm của Systick, đảm bảo các tác vụ định kỳ, delay hoặc tick hệ thống được thực hiện đúng thời gian yêu cầu.

- Systick Calibration Register (CALIB). Thanh ghi CALIB của Systick là thanh ghi dùng để hiệu chuẩn độ chính xác của bộ đếm Systick so với xung nhịp chuẩn. Nó cho phép lập trình viên biết được chu kỳ định thời thực tế của Systick, đảm bảo các tick hoặc delay được tạo ra đúng thời gian mong muốn. Thanh ghi CALIB bao gồm các bit quan trọng như TENMS - thể hiện giá trị bộ đếm tương ứng với 10 ms khi sử dụng clock hệ thống, SKEW - cho biết bộ đếm có bị lệch so với giá trị chuẩn hay không và NOREF - cho biết không có giá trị tham chiếu chuẩn. Nhờ CALIB, Systick có thể tạo ra các chu kỳ thời gian chính xác hơn, đặc biệt hữu ích

khi sử dụng trong các hệ thống RTOS hoặc các ứng dụng đòi hỏi đồng bộ thời gian cao. Thanh ghi này thường được đọc để tính toán và hiệu chỉnh các giá trị nạp vào LOAD, giúp lập lịch và delay trong hệ thống nhúng đạt độ chính xác mong muốn.

2.2.2 AGT.

AGT là một timer tổng quát, mạnh mẽ và linh hoạt, thường được sử dụng cho các ứng dụng tạo delay chính xác, PWM, capture/compare hay ngắt theo sự kiện. Hoạt động của AGT được điều khiển bởi một số thanh ghi cơ bản, cho phép lập trình viên cấu hình đầy đủ các chế độ đếm và chức năng ngoại vi

- Timer Counter Register (CNT) Thanh ghi CNT của AGT là thanh ghi lưu giá trị hiện tại của bộ đếm giúp phản ánh trạng thái thời gian hoặc xung nhịp còn lại trong chu kỳ hoạt động của timer. Giá trị trong CNT sẽ tăng hoặc giảm tùy theo chế độ đếm đã được cấu hình có thể là đếm lên, đếm xuống hoặc là đếm lên-xuống. Khi CNT đạt giá trị tham chiếu trong thanh ghi PR hoặc về 0, AGT có thể phát sinh ngắt, đảo tín hiệu ra chân TIO hoặc reset bộ đếm tùy theo chế độ đã thiết lập. Thanh ghi CNT có thể được đọc bất cứ lúc nào để giám sát trạng thái timer hoặc đo khoảng thời gian giữa các sự kiện, giúp lập trình viên quản lý chính xác các tác vụ định thời, tạo PWM hoặc đo tần số tín hiệu trong các ứng dụng nhúng.

- Period Register (PR). Thanh ghi PR (Period Register) của AGT là thanh ghi xác định giá trị tham chiếu hoặc chu kỳ hoạt động của bộ đếm. Khi giá trị trong CNT đạt PR, AGT sẽ thực hiện các hành động đã được cấu hình chẳng hạn như phát sinh ngắt, đảo tín hiệu ra chân TIO hoặc reset bộ đếm tùy theo chế độ hoạt động. Giá trị PR quyết định chu kỳ định thời, độ rộng xung PWM hoặc tần số xuất ngắt, giúp lập trình viên điều khiển chính xác các sự kiện theo thời gian trong hệ thống nhúng. Nhờ PR, AGT có thể tạo ra các tín hiệu định kỳ hoặc PWM với chu kỳ và độ rộng xung được điều chỉnh theo yêu cầu của ứng dụng.

- Timer Control/Status Register (TCSR). Thanh ghi TCSR (Timer Control/Status Register) của AGT là thanh ghi quan trọng dùng để cấu hình và điều khiển toàn bộ hoạt động của timer đồng thời giám sát trạng thái hiện tại của bộ đếm. Thông qua TCSR, lập trình viên có thể chọn chế độ đếm của AGT bao gồm đếm lên (up-count), đếm xuống (down-count) hoặc đếm lên-xuống (up-down count), phù hợp với yêu cầu ứng dụng như tạo PWM hoặc đếm thời gian. Thanh ghi này cũng cho phép bật hoặc tắt timer, giúp quản lý việc vận hành bộ đếm mà không cần thay đổi cấu hình khác. Ngoài ra, TCSR cung cấp khả năng lựa chọn nguồn xung nhịp cho timer, có thể là đồng bộ với CPU hoặc bất đồng bộ để tiết kiệm năng lượng từ đó kiểm soát tốc độ đếm và chu kỳ hoạt động của AGT. Thanh ghi cũng cho phép cấu hình tín hiệu ra chân TIO, dùng để xuất PWM hoặc đồng bộ các tín hiệu với sự kiện ngoại vi. Bên cạnh đó, TCSR chứa các cờ trạng thái phản ánh tình trạng hiện tại của timer, chẳng hạn như báo hiệu timer đang chạy, đã đạt giá trị PR, hoặc xảy ra tràn bộ đếm, giúp lập trình viên giám sát và xử lý kịp thời các sự kiện. Nhờ tính năng đa dạng này, TCSR không chỉ là công cụ để điều khiển AGT mà còn cung cấp thông tin trạng thái quan trọng, cho phép lập trình viên điều khiển chính xác chu kỳ, xung PWM và các sự kiện ngắt trong các ứng dụng nhúng phức tạp.

- Timer Interrupt Enable Register (TIER) Thanh ghi TIER (Timer Interrupt Enable Register) của AGT là thanh ghi dùng để quản lý và bật/tắt việc ngắt của timer điều có thể cho phép lập trình viên kiểm soát các sự kiện cần được xử lý bởi CPU. Thông qua TIER, ta có thể cấu hình để AGT phát sinh ngắt khi bộ đếm CNT đạt giá trị 0 hoặc khi CNT đạt giá trị tham chiếu PR cũng như các ngắt liên quan đến chức năng capture/compare. Việc bật các ngắt này giúp AGT phản hồi kịp thời các sự kiện định thời, tạo ra các chu kỳ tín hiệu hoặc đo tần số/xung tín hiệu đầu vào. Khi một ngắt được kích hoạt, AGT sẽ gửi tín hiệu đến CPU để thực hiện hàm xử lý ngắt, đảm bảo các tác vụ thời gian thực được thực hiện chính xác. Nhờ TIER, lập trình viên có thể linh hoạt chọn lựa loại ngắt cần thiết, đồng thời quản lý hiệu quả các sự kiện trong hệ thống nhúng, từ việc tạo PWM đến các ứng dụng điều khiển ngoại vi phức tạp.

- Timer I/O Register (TIO). Thanh ghi TIO (Timer I/O Register) của AGT là thanh ghi dùng để xuất tín hiệu ra chân ngoài của vi điều khiển, cho phép AGT tương tác trực tiếp với các thiết bị ngoại vi hoặc tạo ra các tín hiệu điều khiển như PWM. Giá trị của TIO có thể được lập trình để xuất logic cao hoặc thấp theo từng sự kiện của bộ đếm hoặc có thể được tự động đảo khi bộ đếm CNT đạt giá trị tham chiếu PR hoặc về 0 và từ đó tạo ra các xung PWM với độ rộng và chu kỳ xác định. Thanh ghi này cũng hỗ trợ các chế độ đồng bộ với CNT giúp tín hiệu ra chân TIO phản ánh chính xác trạng thái timer tại mỗi thời điểm. Nhờ TIO, lập trình viên có thể sử dụng AGT để điều khiển động cơ, LED, buzzer hoặc các thiết bị ngoại vi khác mà không cần phải can thiệp trực tiếp bằng phần mềm trong từng xung nhịp, giúp giảm tải CPU và đảm bảo tín hiệu ra ổn định, chính xác theo thời gian thực.

2.3 Mô tả các chế độ của ngoại vi timer

2.3.1 Systick.

Systick tuy là một timer hệ thống đơn giản song vẫn cung cấp nhiều chế độ cơ bản nhằm phục vụ các nhu cầu định thời của người sử dụng. Các chế độ chính và quan trọng của Systick bao gồm:

- Chế độ bật/tắt timer (Enable/Disable). Chế độ này được điều khiển bởi bit ENABLE trong thanh ghi CTRL, cho phép lập trình viên quản lý trực tiếp việc khởi động hoặc dừng bộ đếm của Systick. Khi bit ENABLE được bật, bộ đếm bắt đầu giảm giá trị từ thanh ghi LOAD xuống 0 theo từng xung clock. Khi đạt đến 0, Systick có thể phát sinh ngắt nếu được cấu hình cho phép và sau đó tự động nạp lại giá trị từ LOAD để bắt đầu chu kỳ mới. Ngược lại, khi bit ENABLE bị tắt, bộ đếm dừng lại ngay lập tức và giữ nguyên giá trị hiện tại trong thanh ghi VAL mà không bị reset. Nhờ chế độ này, người lập trình có thể linh hoạt điều khiển thời điểm hoạt động của timer, đặc biệt hữu ích trong các ứng dụng yêu cầu bắt đầu hoặc tạm dừng định thời chính xác.

- Chế độ chọn nguồn xung nhịp (Clock Source). Nguồn xung nhịp của Systick được chọn thông qua bit CLKSOURCE trong thanh ghi CTRL và đây là yếu tố ảnh hưởng trực tiếp đến tốc độ đếm và độ chính xác của timer. Nếu CLKSOURCE được đặt là 1, Systick sử dụng clock hệ thống (Processor Clock) – tức là đồng hồ chính

của CPU, mang lại độ chính xác cao và phù hợp cho các ứng dụng cần định thời chính xác hoặc đồng bộ với hoạt động của bộ xử lý. Ngược lại, nếu CLKSOURCE bằng 0, Systick sẽ sử dụng nguồn clock bên ngoài (External Reference Clock) cho phép timer hoạt động độc lập với CPU đặc biệt hữu ích trong các chế độ tiết kiệm năng lượng hoặc khi CPU ở trạng thái ngủ (sleep mode).

- Chế độ ngắt (Interrupt Mode). thanh ghi CTRL. Khi bit này được bật, Systick sẽ phát sinh một ngắt mỗi khi bộ đếm giảm về 0 và cho phép vi điều khiển thực thi các hàm xử lý ngắt (ISR) được định nghĩa sẵn. Cơ chế này đặc biệt quan trọng trong các hệ thống Real-Time Operating System (RTOS), nơi Systick được dùng để tạo tick thời gian hệ thống, giúp lập lịch và quản lý các tác vụ định kỳ. Nếu bit TICKINT bị tắt, Systick vẫn hoạt động bình thường nhưng không tạo ngắt, thích hợp cho các ứng dụng cần đo hoặc tạo trễ mà không cần can thiệp từ CPU. Nhờ chế độ này, Systick có thể hoạt động như một bộ định thời chủ động hoặc thụ động, tùy vào nhu cầu của hệ thống.

2.3.2 AGT.

Ngoại vi timer AGT hỗ trợ nhiều chế độ hoạt động giúp cho lập trình viên dễ dàng cấu hình và tối ưu hệ thống.

- Chế độ đếm (Counting Mode). Đây là chế độ cơ bản và quan trọng nhất của AGT, cho phép bộ định thời thực hiện đếm xung theo thời gian. AGT có thể hoạt động ở ba kiểu đếm chính: đếm lên (Up-count mode), đếm xuống (Down-count mode) và đếm lên-xuống (Up/Down-count mode).
  - Ở chế độ đếm lên, bộ đếm CNT bắt đầu từ 0 và tăng dần cho đến khi đạt đến giá trị định sẵn trong thanh ghi PR (Period Register). Khi đạt giá trị này, timer có thể tự động phát sinh ngắt hoặc tín hiệu ra chân TIO, sau đó quay lại 0 để bắt đầu chu kỳ mới.
  - Ở chế độ đếm xuống, bộ đếm bắt đầu từ giá trị PR và giảm dần về 0. Khi chạm mốc 0, timer có thể phát sinh ngắt và tự động nạp lại giá trị PR, thích hợp cho các ứng dụng cần chu kỳ thời gian cố định.
  - Cuối cùng, chế độ đếm lên-xuống thường được dùng trong tạo xung PWM đối xứng, giúp giảm méo dạng tín hiệu và tạo ra tần số dao động ổn định, đặc biệt hữu ích trong điều khiển động cơ hoặc nguồn xung.
- Chế độ xuất tín hiệu (Output Mode). Trong chế độ này, AGT có khả năng tự động tạo và xuất tín hiệu ra chân TIO mà không cần CPU can thiệp liên tục. Thông qua thanh ghi TIO (Timer I/O Register), người dùng có thể cấu hình tín hiệu đầu ra để chuyển mức (toggle) giữ mức cao/thấp cố định hoặc tạo dạng sóng PWM dựa trên giá trị đếm trong CNT và PR. Khi CNT đạt đến các mốc nhất định (0 hoặc PR), AGT sẽ tự động thay đổi trạng thái logic của chân TIO, giúp tạo ra tín hiệu định kỳ có chu kỳ và độ rộng xung chính xác. Chế độ này thường được sử dụng để điều khiển LED, buzzer, động cơ servo hoặc phát xung đồng bộ hóa với ngoại vi khác. Ưu điểm của chế độ này là giảm tải cho CPU vì AGT có thể duy trì hoạt động định kỳ độc lập, đảm bảo tín hiệu ổn định theo thời gian thực.

- Chế độ ngắt và sự kiện (Interrupt/Event Mode). Đây là một trong những chế độ giúp AGT tích hợp chặt chẽ với hệ thống điều khiển. Khi các điều kiện xác định xảy ra ví dụ như CNT đạt 0, CNT đạt giá trị PR hoặc có tín hiệu capture — AGT có thể phát sinh ngắt hoặc sự kiện (event) thông qua thanh ghi TIER (Timer Interrupt Enable Register). Khi chế độ ngắt được bật, AGT gửi tín hiệu đến CPU để thực hiện hàm xử lý ngắt (ISR), cho phép phản ứng nhanh và chính xác với các sự kiện thời gian thực. Ngoài ra, AGT còn có thể phát sự kiện nội bộ (internal event) để kích hoạt các ngoại vi khác như ADC, DAC hoặc UART, giúp đồng bộ hóa hoạt động mà không cần CPU can thiệp. Nhờ khả năng sinh ngắt và sự kiện, AGT đóng vai trò quan trọng trong các ứng dụng thời gian thực, hệ thống tiết kiệm năng lượng và điều khiển tự động nơi yêu cầu hệ thống phải phản hồi nhanh và chính xác với các thay đổi của tín hiệu.

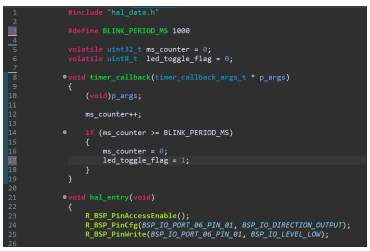
2.4 Chương trình ví dụ về Timer

2.4.1 Systick

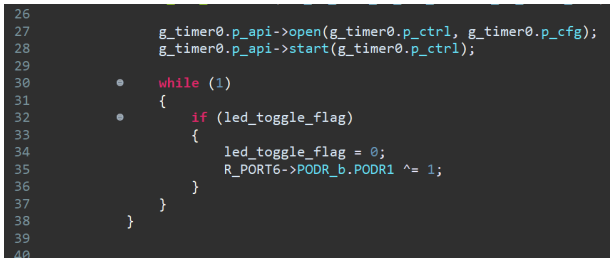
Trong các hệ thống nhúng, việc tạo các tác vụ định kỳ theo thời gian (time-based task) là yêu cầu rất phổ biến, điển hình như điều khiển LED nhấp nháy, tạo xung clock, hay lập lịch các tiến trình đơn giản. Chương này trình bày và phân tích hai chương trình điều khiển LED nhấp nháy trên vi điều khiển RA, sử dụng hai cơ chế Systick timer khác nhau là FSP và thanh ghi

Trong chương trình thứ nhất, GPT Timer được cấu hình thông qua FSP để tạo ngắt định kỳ 1 ms. Với cách cấu hình này, GPT đóng vai trò như một system tick timer, mô phỏng chức năng của SysTick trong việc đếm thời gian và kích hoạt các tác vụ theo chu kỳ. Mục đích của chương trình là tạo một ngắt định kỳ với độ phân giải 1 ms. Mỗi lần ngắt xảy ra, hàm callback timer_callback() được gọi và biến đếm mili-giây ms_counter được tăng lên. Khi biến này đạt đến giá trị 1000 ms (tương ứng 1 giây), chương trình chỉ đặt một cờ logic (led_toggle_flag) để báo cho vòng lặp chính biết rằng đã đến thời điểm đổi trạng thái LED.

Trong hàm hal_entry(), chân P6.1 được cấu hình làm ngõ ra để điều khiển LED. Timer GPT được khởi tạo và kích hoạt bằng các hàm open() và start() của FSP. Vòng lặp vô hạn while(1) liên tục kiểm tra cờ led_toggle_flag; khi cờ được set, LED sẽ được đảo trạng thái và cờ được xóa. Cách tiếp cận này đảm bảo rằng phần xử lý chính không nằm trong ngắt, giúp hệ thống ổn định và dễ mở rộng.



Hình 8: Systick_fsp_p1



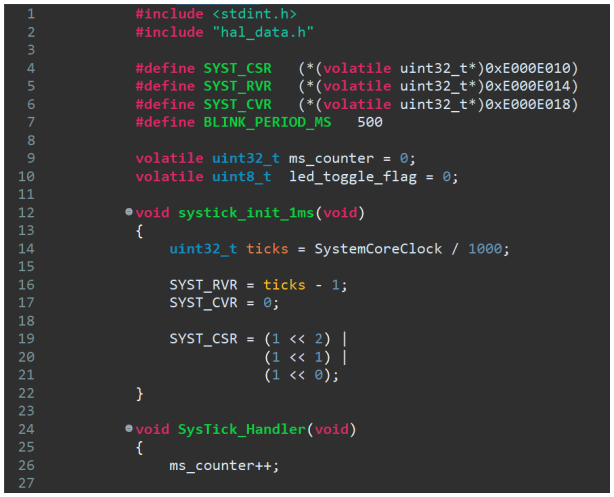
```
26
27              g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
28              g_timer0.p_api->start(g_timer0.p_ctrl);
29
30          •   while (1)
31              {
32          •       if (led_toggle_flag)
33                  {
34                      led_toggle_flag = 0;
35                      R_PORT6->PODR_b.PODR1 ^= 1;
36                  }
37              }
38          }
39
40
```

Hình 9: Systick_fsp_p2

Chương trình thứ hai sử dụng SysTick Timer, là bộ định thời tích hợp sẵn trong lõi ARM Cortex-M, hoạt động độc lập với FSP. Việc cấu hình SysTick trong chương trình này được thực hiện trực tiếp thông qua các thanh ghi phần cứng như SYST_CSR, SYST_RVR và SYST_CVR, thay vì thông qua API trừu tượng.

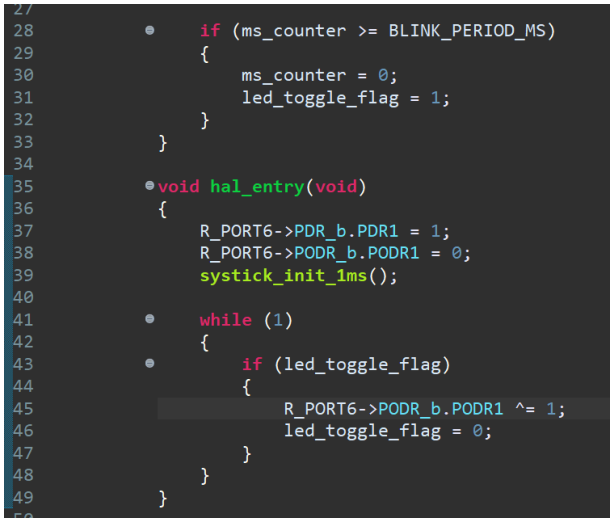
SysTick được cấu hình để tạo ngắt mỗi 1 ms bằng cách nạp giá trị phù hợp dựa trên tần số SystemCoreClock. Mỗi khi ngắt SysTick xảy ra, hàm phục vụ ngắt SysTick_Handler() sẽ được gọi, trong đó biến ms_counter được tăng lên. Khi đủ 500 ms, chương trình đặt cờ led_toggle_flag để báo cho vòng lặp chính thực hiện việc đảo trạng thái LED.

Trong hàm hal_entry(), chân P6.1 được cấu hình trực tiếp thông qua các thanh ghi PORT để làm ngõ ra. Vòng lặp chính chỉ đảm nhiệm việc kiểm tra cờ và toggle LED khi cần thiết. Cách lập trình này cho phép người lập trình kiểm soát trực tiếp phần cứng, nhưng đòi hỏi hiểu rõ kiến trúc ARM và cơ chế ngắt.



```
1           #include <stdint.h>
2           #include "hal_data.h"
3
4           #define SYST_CSR   (*(volatile uint32_t*)0xE000E010)
5           #define SYST_RVR   (*(volatile uint32_t*)0xE000E014)
6           #define SYST_CVR   (*(volatile uint32_t*)0xE000E018)
7           #define BLINK_PERIOD_MS   500
8
9           volatile uint32_t ms_counter = 0;
10          volatile uint8_t  led_toggle_flag = 0;
11
12        • void systick_init_1ms(void)
13          {
14              uint32_t ticks = SystemCoreClock / 1000;
15
16              SYST_RVR = ticks - 1;
17              SYST_CVR = 0;
18
19              SYST_CSR = (1 << 2) |
20                         (1 << 1) |
21                         (1 << 0);
22          }
23
24        • void SysTick_Handler(void)
25          {
26              ms_counter++;
27
```

Hình 10: Systick_thanh_ghi_p1



```
28          if (ms_counter >= BLINK_PERIOD_MS)
29          {
30              ms_counter = 0;
31              led_toggle_flag = 1;
32          }
33      }
34
35      void hal_entry(void)
36      {
37          R_PORT6->PDR_b.PDR1 = 1;
38          R_PORT6->PODR_b.PODR1 = 0;
39          systick_init_1ms();
40
41          while (1)
42          {
43              if (led_toggle_flag)
44              {
45                  R_PORT6->PODR_b.PODR1 ^= 1;
46                  led_toggle_flag = 0;
47              }
48          }
49      }
```

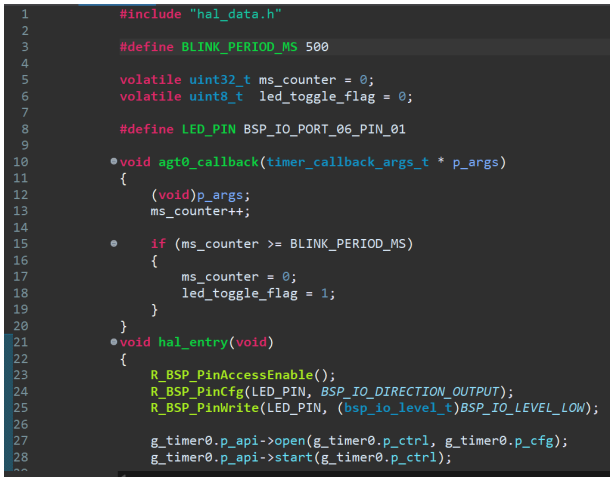
Hình 11: Systick_thanh_ghi_p2

Hai chương trình đều thực hiện thành công chức năng điều khiển LED nhấp nháy dựa trên timer. Mô phỏng Systick Timer thông qua GPT (FSP) phù hợp cho các hệ thống thực tế, yêu cầu tính ổn định và khả năng mở rộng. Trong khi lập trình thanh ghi cho SysTick Timer phù hợp cho mục đích học tập, nghiên cứu kiến trúc ARM và cơ chế ngắt cơ bản.

2.4.2 AGT

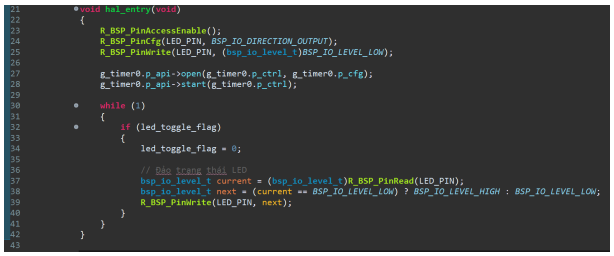
Chương này sẽ sử dụng AGT (Asynchronous General Purpose Timer) để điều khiển LED nhấp nháy và được triển khai theo hai cách tiếp cận khác nhau: sử dụng FSP API và lập trình thuần thanh ghi. AGT là một timer ngoại vi của vi điều khiển RA, có ưu điểm là tiêu thụ năng lượng thấp và vẫn hoạt động trong các chế độ tiết kiệm năng lượng.

Ở phiên bản thứ nhất, AGT0 được cấu hình thông qua FSP. Timer được thiết lập để tạo ngắt định kỳ với chu kỳ 1 ms (hoặc giá trị tương đương theo cấu hình). Mỗi khi xảy ra sự kiện so khớp (compare/match), hàm callback agt0_callback() được gọi và biến ms_counter được tăng lên. Khi số mili-giây đạt đến giá trị định trước (500 ms), chương trình chỉ đặt cờ led_toggle_flag. Việc đảo trạng thái LED được thực hiện trong vòng lặp chính while(1). Cách tổ chức này tương tự như mô hình system tick, giúp giảm tải cho ISR và đảm bảo tính ổn định của hệ thống.



```c
#include "hal_data.h"

#define BLINK_PERIOD_MS 500

volatile uint32_t ms_counter = 0;
volatile uint8_t  led_toggle_flag = 0;

#define LED_PIN BSP_IO_PORT_06_PIN_01

void agt0_callback(timer_callback_args_t * p_args)
{
    (void)p_args;
    ms_counter++;

    if (ms_counter >= BLINK_PERIOD_MS)
    {
        ms_counter = 0;
        led_toggle_flag = 1;
    }
}
void hal_entry(void)
{
    R_BSP_PinAccessEnable();
    R_BSP_PinCfg(LED_PIN, BSP_IO_DIRECTION_OUTPUT);
    R_BSP_PinWrite(LED_PIN, (bsp_io_level_t)BSP_IO_LEVEL_LOW);

    g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
    g_timer0.p_api->start(g_timer0.p_ctrl);
```

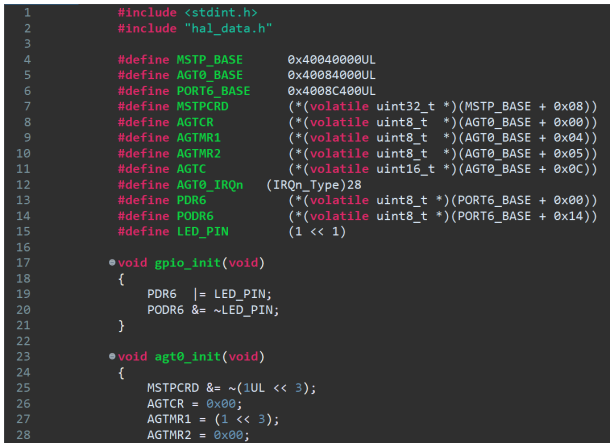
Hình 12: AGT_FSP_P1



```c
void hal_entry(void)
{
    R_BSP_PinAccessEnable();
    R_BSP_PinCfg(LED_PIN, BSP_IO_DIRECTION_OUTPUT);
    R_BSP_PinWrite(LED_PIN, (bsp_io_level_t)BSP_IO_LEVEL_LOW);

    g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
    g_timer0.p_api->start(g_timer0.p_ctrl);

    while (1)
    {
        if (led_toggle_flag)
        {
            led_toggle_flag = 0;

            // Đảo trạng thái LED
            bsp_io_level_t current = (bsp_io_level_t)R_BSP_PinRead(LED_PIN);
            bsp_io_level_t next = (current == BSP_IO_LEVEL_LOW) ? BSP_IO_LEVEL_HIGH : BSP_IO_LEVEL_LOW;
            R_BSP_PinWrite(LED_PIN, next);
        }
    }
}
```

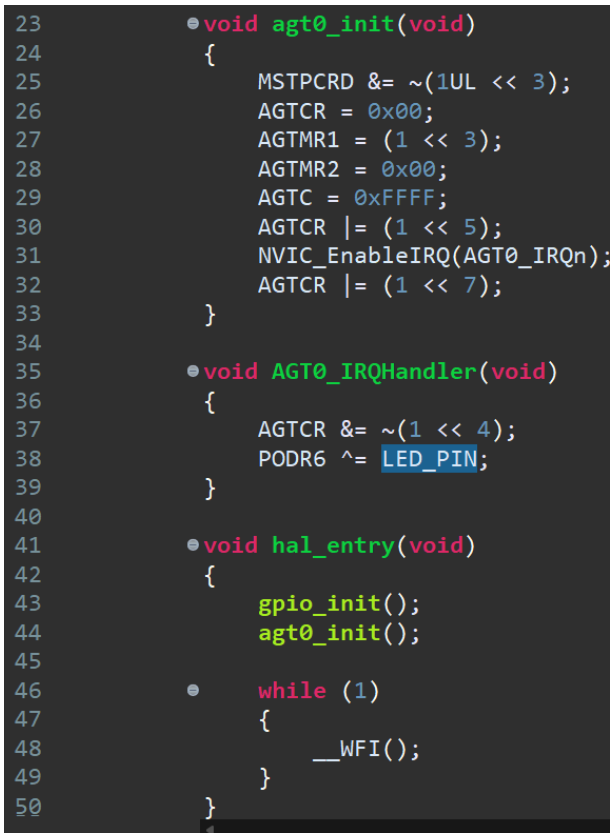
Hình 13: AGT_FSP_P2

Ở phiên bản thứ hai, AGT0 được cấu hình trực tiếp bằng cách truy cập các thanh ghi phần cứng dựa trên datasheet của RA6M5. Module AGT0 trước hết được bật thông qua thanh ghi MSTP để cấp xung clock. Sau đó, bộ đếm được reset, prescaler được thiết lập (chia 1024), và giá trị so khớp (AGTCMP) được nạp tương ứng với chu kỳ mong muốn. Khi AGT0 đạt giá trị so khớp, ngắt được kích hoạt và hàm phục vụ ngắt AGT0_IRQHandler() sẽ được gọi để đảo trực tiếp trạng thái LED.



```c
#include <stdint.h>
#include "hal_data.h"

#define MSTP_BASE       0x40040000UL
#define AGT0_BASE       0x40084000UL
#define PORT6_BASE      0x4008C400UL
#define MSTPCRD         (*(volatile uint32_t *)(MSTP_BASE + 0x08))
#define AGTCR           (*(volatile uint8_t  *)(AGT0_BASE + 0x00))
#define AGTMR1          (*(volatile uint8_t  *)(AGT0_BASE + 0x04))
#define AGTMR2          (*(volatile uint8_t  *)(AGT0_BASE + 0x05))
#define AGTC            (*(volatile uint16_t *)(AGT0_BASE + 0x0C))
#define AGT0_IRQn   (IRQn_Type)28
#define PDR6            (*(volatile uint8_t *)(PORT6_BASE + 0x00))
#define PODR6           (*(volatile uint8_t *)(PORT6_BASE + 0x14))
#define LED_PIN         (1 << 1)

void gpio_init(void)
{
    PDR6  |= LED_PIN;
    PODR6 &= ~LED_PIN;
}

void agt0_init(void)
{
    MSTPCRD &= ~(1UL << 3);
    AGTCR = 0x00;
    AGTMR1 = (1 << 3);
    AGTMR2 = 0x00;
```

Hình 14: AGT_thanh_ghi_P1



```c
void agt0_init(void)
{
    MSTPCRD &= ~(1UL << 3);
    AGTCR = 0x00;
    AGTMR1 = (1 << 3);
    AGTMR2 = 0x00;
    AGTC = 0xFFFF;
    AGTCR |= (1 << 5);
    NVIC_EnableIRQ(AGT0_IRQn);
    AGTCR |= (1 << 7);
}

void AGT0_IRQHandler(void)
{
    AGTCR &= ~(1 << 4);
    PODR6 ^= LED_PIN;
}

void hal_entry(void)
{
    gpio_init();
    agt0_init();

    while (1)
    {
        __WFI();
    }
}
```

Hình 15: AGT_thanh_ghi_P2

So với cách dùng FSP, lập trình AGT ở mức thanh ghi cho phép kiểm soát hoàn toàn phần cứng và giúp người học hiểu rõ hơn về cấu trúc timer, NVIC và cơ chế ngắt của vi điều khiển. Tuy nhiên, phương pháp này đòi hỏi độ chính xác cao trong việc cấu hình địa chỉ thanh ghi, bit điều khiển và thứ tự khởi tạo, nếu không sẽ dễ dẫn đến lỗi hệ thống

3 Bài tập lớn 2

3.1 Cảm biến HS3001

3.1.1 Giới thiệu chung.

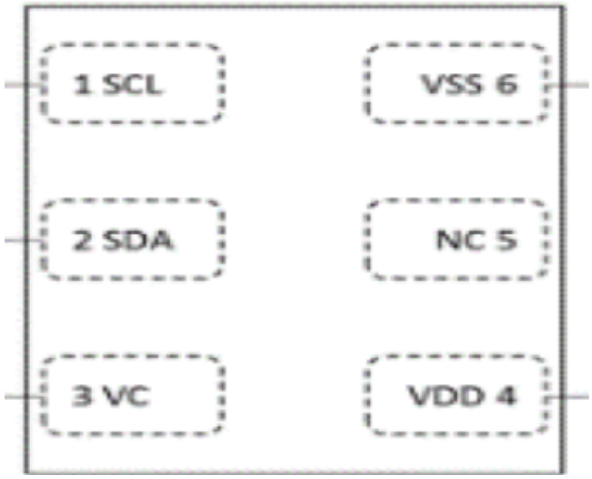
HS3001 là cảm biến độ ẩm và nhiệt độ kỹ thuật số do Renesas Electronics phát triển, được tích hợp trực tiếp trên kit CK-RA6M5. Cảm biến được thiết kế nhằm cung cấp các thông số môi trường với độ chính xác cao, mức tiêu thụ năng lượng thấp, phù hợp cho các hệ thống nhúng và ứng dụng IoT. Trong hệ thống đang xét, HS3001 đảm nhiệm chức năng thu thập các thông số môi trường cơ bản như nhiệt độ và độ ẩm không khí, từ đó cung cấp dữ liệu đầu vào cho quá trình giám sát, phân tích và đánh giá điều kiện môi trường của hệ thống.

3.1.2 Các đặc tính kỹ thuật cảm biến HS3001.

- Độ chính xác độ ẩm (RH) : ±1,5% RH (điển hình).
- Thời gian phản hồi độ ẩm (RH) : nhanh (thường là 6 giây).
- Độ chính xác cảm biến nhiệt độ: ±0,2 °C (điển hình, trong dải -10 °C đến +80 °C)
- Độ phân giải : 14 bit, tương đương 0,01% RH (điển hình)
- Mức tiêu thụ điện năng thấp: trung bình 1,0µA (đối với một lần đo RH + T mỗi giây, ở độ phân giải 8 bit, cấp nguồn 1,8V)
- Phạm vi điện áp cung cấp mở rộng: 2.3V đến 5.5V

3.1.3 Cấu tạo của cảm biến HS3001.

Phần tử cảm biến độ ẩm (Humidity Sensing Element): Thường là một lớp polymer nhạy ẩm, trong đó điện dung thay đổi theo độ ẩm của không khí, cho phép xác định giá trị độ ẩm tương đối của môi trường. Cảm biến nhiệt độ tích hợp (Integrated Temperature Sensor) : Có nhiệm vụ đo nhiệt độ môi trường xung quanh và được sử dụng để bù sai số nhiệt cho phép đo độ ẩm, nhằm nâng cao độ chính xác của kết quả đo. Khối xử lý tín hiệu (ASIC - Application-Specific Integrated Circuit): Thực hiện thu nhận và chuyển đổi tín hiệu analog từ các phần tử cảm biến sang dữ liệu số, đồng thời xử lý và hiệu chỉnh tín hiệu theo các thông số đã được hiệu chuẩn sẵn. Giao tiếp số I^2C: Cho phép vi điều khiển đọc trực tiếp dữ liệu nhiệt độ và độ ẩm từ cảm biến mà không cần sử dụng bộ chuyển đổi ADC ngoài, giúp đơn giản hóa thiết kế phần cứng của hệ thống.



Hình 16: Sơ đồ chân

3.1.4 Sơ đồ chân

- Chân 1 (SCL) : Chân xung clock của giao tiếp I²C
- Chân 2 (SDA) : Chân dữ liệu của giao tiếp I²C
- Chân 3 (VC) : Chân nguồn nội / lọc nguồn (kết nối tụ decoupling).
- Chân 4 (VDD) : Chân cấp nguồn cho cảm biến
- Chân 5 (NC) : Không kết nối (Not Connected).
- Chân 6 (VSS) : Chân mass (GND)

3.1.5 Nguyên lí hoạt động

Đo độ ẩm : HS3001 đo độ ẩm dựa trên cơ chế cảm biến điện dung. Khi độ ẩm không khí thay đổi, lượng hơi nước hấp thụ vào lớp polymer nhạy ẩm thay đổi, làm điện dung của phần tử cảm biến biến đổi. Sự thay đổi này được mạch xử lý bên trong chuyển đổi và số hóa bằng ADC độ phân giải 14-bit, sau đó được hiệu chuẩn, bù sai số nhiệt và mã hóa thành giá trị độ ẩm tương đối (%RH) để truyền qua giao tiếp I²C. Đo nhiệt độ : HS3001 tích hợp cảm biến nhiệt độ để đo nhiệt độ môi trường. Giá trị nhiệt độ thu được vừa được xuất ra như một thông số độc lập, vừa được sử dụng để bù sai số nhiệt cho phép đo độ ẩm, giúp nâng cao độ chính xác và độ tin cậy của kết quả đo.

3.1.6 Giao tiếp và xử lý trong hệ thống

Trong hệ thống, cảm biến HS3001 được sử dụng để thu thập dữ liệu nhiệt độ và độ ẩm môi trường, đóng vai trò là nguồn dữ liệu môi trường đầu vào cho quá trình giám sát và phân tích. HS3001 được kết nối với vi điều khiển RA6M5 thông qua giao tiếp I²C. Trong giai đoạn khởi động (Hal_entry), hệ thống tiến hành cấu hình bus I²C bằng hàm g_comms_i2c_bus0_quick_setup(), sau đó khởi tạo cảm biến HS3001 bằng hàm g_hs300x_sensor0_quick_setup() thông qua thư viện FSP (Flexible Software Package), đảm bảo cảm biến sẵn sàng hoạt động. Về phần cứng, HS3001 được nối với RA6M5 qua hai

chân SCL và SDA, sử dụng điện trở kéo lên (Rp) cho bus I²C và tụ tách nhiễu 0.1 µF tại chân VDD nhằm đảm bảo tín hiệu ổn định. Trong vòng lặp chính, vi điều khiển định kỳ đọc dữ liệu từ HS3001 bằng hàm g_hs300x_sensor0_quick_getting_humidity_and_temperature(). Kết quả đo được lưu vào biến data (kiểu rm_hs300x_data_t), trong đó giá trị nhiệt độ và độ ẩm được tách thành phần nguyên và phần thập phân, thuận tiện cho việc hiển thị và truyền dữ liệu. Sau khi thu thập, dữ liệu từ HS3001 được đóng gói thành gói dữ liệu (sensor_packet_t) và đưa vào hàng đợi sensor_data_queue để tránh mất dữ liệu khi kết nối tạm thời gián đoạn. Khi kết nối với PC Server được xác nhận thông qua giao tiếp UART (cờ isConnectedFlag = True), các gói dữ liệu trong hàng đợi sẽ lần lượt được gửi lên máy tính. Tại phía server, dữ liệu từ HS3001 được giải mã, hiển thị, lưu trữ và phục vụ cho việc phân tích điều kiện môi trường, ví dụ: "Temperature: 39.40". Trong hệ thống, HS3001 được sử dụng để giám sát nhiệt độ và độ ẩm môi trường, đóng vai trò là nguồn dữ liệu nền. Các thông số này được dùng để đánh giá điều kiện không khí, đồng thời kết hợp với các chỉ số chất lượng không khí thu thập từ cảm biến ZMOD4410, nhằm phục vụ cho việc phân tích và đánh giá tổng thể chất lượng môi trường.

3.2 Cảm biến ZMOD4410.

3.2.1 Giới thiệu chung

ZMOD4410 là cảm biến chất lượng không khí trong nhà (Indoor Air Quality – IAQ) do Renesas Electronics phát triển, sử dụng công nghệ Metal-Oxide (MOx) để phát hiện các hợp chất hữu cơ bay hơi (Volatile Organic Compounds – VOCs).

Trong hệ thống, ZMOD4410 đảm nhiệm vai trò đánh giá mức độ ô nhiễm không khí, cung cấp thông tin chất lượng không khí chuyên sâu, từ đó bổ sung và hoàn thiện dữ liệu môi trường so với các thông số nhiệt độ và độ ẩm thu thập từ cảm biến HS3001.

3.2.2 Các đặc tính kỹ thuật cảm biến ZMOD4410.

- Đạt chứng nhận JEDEC JESD47, đảm bảo tuổi thọ hoạt động trên 10 năm.
- Firmware có thể cấu hình, với đầu ra dựa trên thuật toán học máy AI, cho phép:
  - Đo nồng độ tổng các hợp chất hữu cơ bay hơi (TVOCs) một cách tuyệt đối và chất lượng không khí trong nhà (IAQ).
  - Ước tính nồng độ carbon dioxide tương đương (eCO_2).
  - Cung cấp tín hiệu điều khiển tương đối để kích hoạt hành động bên ngoài dựa trên IAQ và sự thay đổi mùi.
  - Hỗ trợ thuật toán phân biệt mùi dựa trên lưu huỳnh.
- Tích hợp bộ nhớ không bay hơi (NVM) để lưu trữ dữ liệu riêng của module (cấu hình và hiệu chuẩn).
- Tiêu thụ năng lượng trung bình cực thấp, chỉ khoảng 160 µW, phù hợp cho các ứng dụng chạy bằng pin.
- Khả năng chống siloxane, giúp bảo vệ cảm biến MOx khỏi các tác nhân gây suy giảm tuổi thọ.

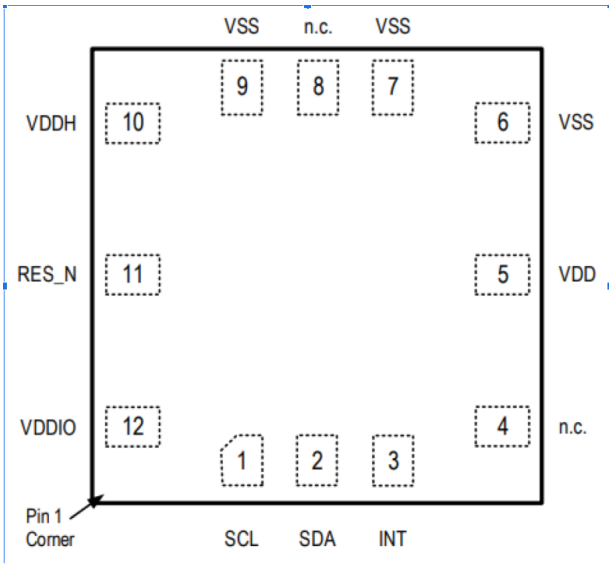
- Điện áp cung cấp: từ 1.7 V đến 3.6 V.
- Tuân thủ tiêu chuẩn RoHS.
- Có phiên bản chống nước và chống bụi, đạt chứng nhận IP67.

3.2.3 Cấu tạo của cảm biến ZMOD4410.

Cảm biến ZMOD4410 bao gồm các khối chức năng chính sau:

- Phần tử cảm biến khí MOx (Metal Oxide): Là phần tử nhạy khí có điện trở thay đổi khi tiếp xúc với các hợp chất hữu cơ bay hơi (VOCs), phản ánh mức độ ô nhiễm của không khí.
- Bộ gia nhiệt (Heater): Có nhiệm vụ gia nhiệt phần tử cảm biến MOx đến nhiệt độ làm việc thích hợp, đảm bảo độ nhạy và tính ổn định của phép đo.
- Mạch đo và xử lý tín hiệu: Thực hiện đo sự thay đổi điện trở của phần tử cảm biến, sau đó xử lý và chuyển đổi tín hiệu analog thành dữ liệu số.
- Giao tiếp số I^2C: Cho phép kết nối trực tiếp với vi điều khiển, giúp truyền dữ liệu đo và cấu hình cảm biến một cách thuận tiện.
- Thuật toán IAQ tích hợp (Firmware/Library): Dữ liệu thô thu được từ cảm biến được xử lý thông qua thuật toán IAQ tích hợp, chuyển đổi thành các chỉ số chất lượng không khí có ý nghĩa như IAQ, TVOCs, eCO_2.

3.2.4 Sơ đồ chân ZMOD4410



Hình 17: Sơ đồ chân ZMOD4410

Nhóm chân giao tiếp:

- Chân 1 – SCL: Chân xung nhịp của giao tiếp I^2C, dùng để đồng bộ truyền dữ liệu với vi điều khiển
- Chân 2 – SDA: Chân dữ liệu của giao tiếp I^2C, truyền dữ liệu hai chiều giữa ZMOD4410 và vi điều khiển.
- Chân 3 – INT: Chân ngắt (Interrupt), dùng để thông báo trạng thái hoặc dữ liệu sẵn sàng cho vi điều khiển (tùy cấu hình firmware).

Nhóm chân nguồn:

- Chân 5 – VDD: Nguồn cấp chính cho mạch đo và xử lý tín hiệu của cảm biến
- Chân 10 – VDDH: Nguồn cấp cho bộ gia nhiệt (Heater) của phần tử cảm biến MOx.
- Chân 12 – VDDIO: Nguồn cấp cho khối giao tiếp số (I^2C), cho phép tương thích mức logic với vi điều khiển.

Nhóm chân mass :Chân 6, 7, 9 – VSS: Các chân mass (GND), cần được nối chung để đảm bảo ổn định điện áp và giảm nhiễu. Nhóm chân điều khiển : Chân 11 – RES_N: Chân reset mức thấp, dùng để khởi động lại cảm biến khi cần thiết. Chân không kết nối : Chân 4, 8 – n.c. (No Connect): Không sử dụng, để hở theo khuyến nghị của nhà sản xuất.

3.2.5 Nguyên lí hoạt động

ZMOD4410 là cảm biến chất lượng không khí trong nhà (Indoor Air Quality – IAQ) sử dụng công nghệ cảm biến khí Metal-Oxide (MOx) để phát hiện các hợp chất hữu cơ bay hơi (VOC) trong môi trường không khí.

Phần tử cảm biến MOx có điện trở thay đổi khi tiếp xúc với các khí VOC. Khi các phân tử VOC hấp phụ lên bề mặt vật liệu oxide kim loại, các phản ứng oxy hóa–khử xảy ra, làm thay đổi mật độ điện tích bề mặt và dẫn đến sự thay đổi điện trở của cảm biến. Mức thay đổi điện trở này tỷ lệ với nồng độ tổng các khí VOC trong môi trường.

ZMOD4410 tích hợp bộ gia nhiệt vi mô (micro-heater) nhằm duy trì phần tử cảm biến ở nhiệt độ làm việc tối ưu, giúp kích hoạt phản ứng hóa học, tăng độ nhạy, cải thiện độ ổn định và giảm ảnh hưởng của độ ẩm. Bộ gia nhiệt được điều khiển theo chu kỳ để đảm bảo độ chính xác cao với mức tiêu thụ năng lượng thấp.

Sự thay đổi điện trở của phần tử MOx được mạch đo bên trong cảm biến chuyển đổi thành tín hiệu điện và số hóa. Do cảm biến MOx nhạy với nhiều loại khí khác nhau nhưng không phân biệt từng loại khí cụ thể, dữ liệu thô thu được không phản ánh trực tiếp nồng độ của từng khí riêng lẻ. Vì vậy, ZMOD4410 sử dụng thuật toán IAQ do Renesas cung cấp, dựa trên các mô hình học máy (AI/Machine Learning), để:

- Ước lượng nồng độ TVOCs
- Tính toán chỉ số chất lượng không khí trong nhà (IAQ).
- Ước tính mức CO_2 tương đương (eCO_2).
- Phân biệt một số mùi đặc trưng (ví dụ mùi lưu huỳnh)

Cuối cùng, các giá trị IAQ, TVOCs và eCO_2được truyền đến vi điều khiển thông qua giao tiếp I_2C, phục vụ cho việc hiển thị, lưu trữ và điều khiển các hệ thống như thông gió, cảnh báo hoặc HVAC.

3.2.6 Giao tiếp và xử lý trong hệ thống

Cảm biến chất lượng không khí ZMOD4410 được tích hợp vào hệ thống Node (kit CK-RA6M5) nhằm giám sát mức độ ô nhiễm không khí trong nhà. Dữ liệu đo được được đóng gói kèm theo thời gian và truyền về PC Server thông qua giao tiếp UART. Quá trình này gồm ba giai đoạn chính: khởi tạo, thu thập dữ liệu và truyền tải.

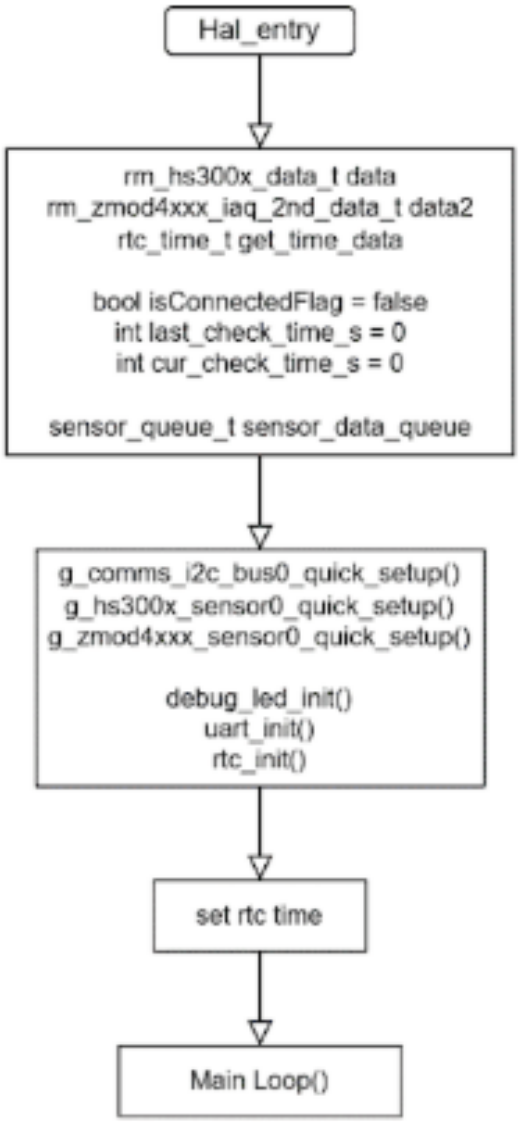
Giai đoạn khởi tạo (Initialization) Trong hàm khởi tạo chính (Hal_entry), hệ thống tiến hành cấu hình các tài nguyên cho ZMOD4410. Bus I^2C được thiết lập bằng hàm g_comms_i2c_bus0_quick_setup() để tạo kênh giao tiếp giữa vi điều khiển RA6M5 và cảm biến. Sau đó, cảm biến ZMOD4410 được khởi tạo và đưa vào trạng thái hoạt động thông qua hàm g_zmod4xxx_sensor0_quick_setup(). Dữ liệu đo từ cảm biến được lưu trữ trong biến data2 (kiểu rm_zmod4xxx_iaq_2nd_data_t). Thu thập dữ liệu trong vòng lặp chính (Main Loop) Trong mỗi vòng lặp chính, hệ thống gọi hàm g_zmod4xxx_sensor0_quick_getting_iaq_2nd_gen_data() để đọc các thông số chất lượng không khí như IAQ, TVOCs, eCO_2. Dữ liệu thu được được lưu vào biến data2. Sau khi hoàn tất việc đọc dữ liệu từ ZMOD4410, HS3001 và thời gian từ RTC, hệ thống tạo một gói dữ liệu (sensor_packet_t), gắn dấu thời gian và đưa vào hàng đợi sensor_data_queue nhằm tránh mất dữ liệu khi kết nối bị gián đoạn. Truyền dữ liệu về PC Server Hệ thống kiểm tra trạng thái kết nối thông qua hàm check_connect(). Khi cờ isConnectedFlag được xác nhận, hàm send_avaiable_data() sẽ lấy các gói dữ liệu trong hàng đợi và gửi về PC Server qua UART. Tại phía máy tính, dữ liệu từ ZMOD4410 được xử lý và hiển thị cùng với dữ liệu nhiệt độ, độ ẩm từ HS3001 và thời gian đo từ RTC, phục vụ cho việc giám sát và phân tích chất lượng không khí.

Trong hệ thống, cảm biến ZMOD4410 được sử dụng để đánh giá chất lượng không khí trong nhà, từ đó phát hiện sớm các tình trạng không khí ô nhiễm và môi trường thiếu thông thoáng, làm cơ sở cho việc giám sát, cảnh báo và đề xuất các biện pháp cải thiện điều kiện không khí.

3.3 a

Thực hiện hệ thống: Node (kit CK-RA6M5) đọc dữ liệu cảm biến nhiệt độ và độ ẩm HS3001 và ZMOD4410 trên kit, gửi gói dữ liệu cùng với thời gian đến server PC thông qua UART.

3.3.1 Lưu đồ giải thuật.



Hình 18: Lưu đồ giải thuật chính

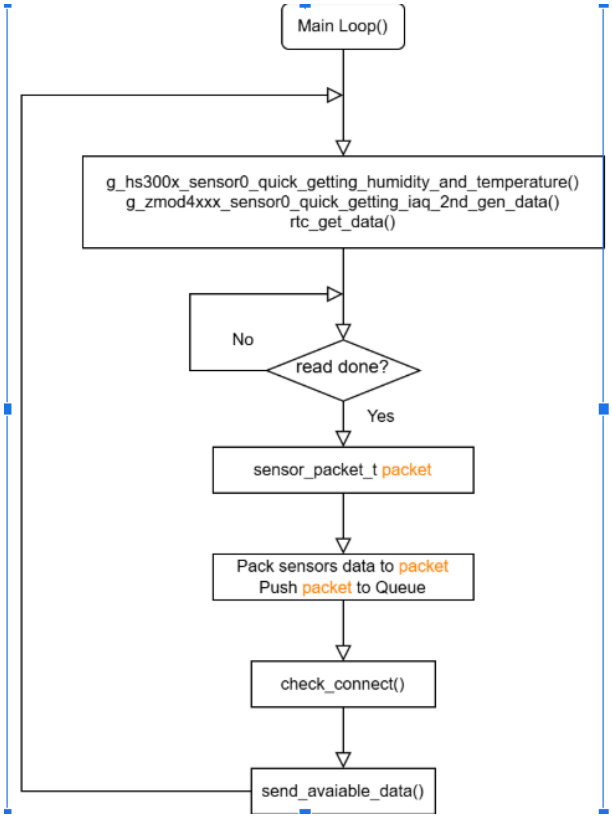
Với các biến sử dụng trong chương trình:

- data, data2 và get_timer_data: lưu trữ dữ liệu đọc từ các cảm biến.
- isConnected, last_check_time_s và cur_check_time_s: phục vụ kiểm tra trạng thái kết nối.
- sensor_data_queue: hàng đợi dùng để lưu các gói dữ liệu đã đọc được.

Khi khởi động, hệ thống tiến hành cấu hình các ngoại vi và cảm biến cần thiết, bao gồm:

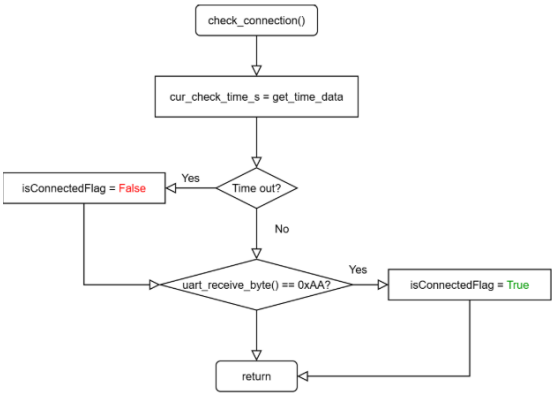
- UART TX và RX và đèn LED đỏ.
- Các cảm biến được thiết lập thông qua các hàm của FSP: g_comms_i2c_bus0_quick_setup(), g_hs300x_sensor0_quick_setup(), rtc_init(), và g_zmod4xxx_sensor0_quick_setup().

Trong lần chạy đầu tiên khi mới cấp nguồn, thời gian của bộ RTC cần được thiết lập lại bằng hàm rtc_set_time().



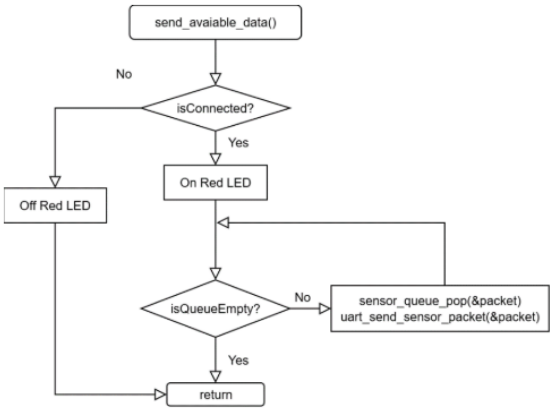
Hình 19: Lưu đồ giải thuật vòng lặp chính.

Ở mỗi vòng lặp, hệ thống đọc giá trị từ các cảm biến thông qua các hàm của thư viện FSP và lưu kết quả vào các biến data, data2 và get_timer_data. Sau khi quá trình đọc hoàn tất, một gói tin mới được tạo và chứa các dữ liệu vừa thu thập và được đẩy vào hàng đợi sensor_data_queue. Sau đó chương trình kiểm tra trạng thái kết nối giữa kit và máy tính bằng hàm check_connect(), sau đó gửi các gói tin đang có thông qua hàm send_avaiable_data().



Hình 20: Lưu_đồ_giải_thuật_của_check_connect()

Hàm sẽ kiểm tra khoảng thời gian đã trôi qua bằng cách sử dụng biến cur_check_time_s từ bộ RTC. Nếu thời gian này vượt quá CHECK_CONNECT_INTERVAL, hệ thống xem như đã hết hạn (timeout) và đặt cờ isConnectedFlag về False. Cờ isConnectedFlag sẽ giữ trạng thái False cho đến khi hệ thống nhận được byte 0xAA qua UART RX gửi từ máy tính, lúc đó cờ sẽ được đặt lại thành True.



Hình 21: Lưu đồ giải thuật của send_avaiable_data()

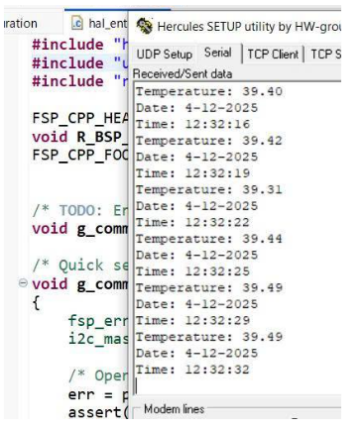
Hàm sẽ kiểm tra trạng thái của cờ isConnected:

- Nếu cờ ở trạng thái False, hệ thống tắt đèn LED đỏ và không thực hiện gửi dữ liệu.
- Nếu cờ ở trạng thái True, chương trình sẽ gửi các gói tin trong hàng đợi qua UART TX cho đến khi hàng đợi trống hoàn toàn.

Expression	Type	Value
data	rm_hs300x...	{...}
humidity	rm_hs300x...	{...}
integer_part	int16_t	49
decimal_part	int16_t	74
temperature	rm_hs300x...	{...}
integer_part	int16_t	39
decimal_part	int16_t	40
data2	rm_zmod4...	{...}
rmox	float [13]	0x2000054c <data2>
compensation_rmox	float	0
log_rcda	float	4.60470009
log_nonlog_rcda	float [3]	0x20000588 <data2+60>
iaq	float	1
tvoc	float	0.0416546427
etoh	float	0.0221567247
eco2	float	400
sample_id	uint8_t	0 '\0'
rel_iaq	float	99.4753265
get_time_data	rtc_time_t	{...}
tm_sec	int	28
tm_min	int	33
tm_hour	int	12
tm_mday	int	4
tm_mon	int	12
tm_year	int	125
tm_wday	int	3
tm_yday	int	0
tm_isdst	int	0

Hình 22: Kết quả đọc các cảm biến

Các kết quả của cảm biến có thể được đọc trực tiếp thông qua cửa sổ Expression trên E2 Studio (Window > Show View > Expressions).



Hình 23: Kết quả thu được qua UART TX