Sự khác nhau giữa Macro, Inline và Function

1. Macro:

- Được xử lý bởi preprocessor
- Thay thế đoạn code được khai báo macro vào bất cứ chỗ nào xuất hiện macro đó
- VD: #define SUM(a,b) (a+b)
  - Preprocessor khi gặp bất kỳ lời gọi SUM(first+last) nào thì thay ngay bằng (first+last)

2. Inline

- Được xử lý bởi compiler
- Được khai báo với từ khóa inline
- Khi compiler thấy bất kỳ chỗ nào xuất hiện inline function, nó sẽ thay thế chỗ đó bởi định nghĩa của hàm đã được compile tương ứng. –> Phần được thay thế không phải code mà là đoạn code đã được compile
-

3. Hàm bình thường

- Khi thấy hàm được gọi, compiler sẽ phải lưu con trỏ chương trình PC hiện tại vào stack; chuyển PC tới hàm được gọi, thực hiện hàm đó xong và lấy kết quả trả về; sau đó quay lại vị trí ban đầu trong stack trước khi gọi hàm và tiếp tục thực hiện chương trình.
- Như có thể thấy, các này khiến chương trình tốn thời gian hơn là chỉ cần thay thế đoạn code đã được compile (cách của inline function)

4. So sánh

- Macro đơn giản là chỉ thay thế đoạn code macro vào chỗ được gọi trước khi được biên dịch
- Inline thay thế đoạn mã code đã được biên dịch vào chỗ được gọi
- Hàm bình thường phải tạo một function call, lưu địa chỉ trước khi gọi hàm vào stack sau đó mới thực hiện hàm và sau cùng là quay trở về địa chỉ trên stack trước khi gọi hàm và thực hiện tiếp chương trình
- Macro khiến code trở nên dài hơn rất nhiều so với bình thường nhưng thời gian chạy nhanh.
- Hàm inline cũng khiến code dài hơn, tuy nhiên nó làm giảm thời gian chạy chương trình

- Hàm bình thường sẽ phải gọi function call nên tốn thời gian hơn inline function nhưng code ngắn gọn hơn.

Biến static cục bộ

Khi 1 biến cục bộ được khai báo với từ khóa static. Biến sẽ chỉ được khởi tạo 1 lần duy nhất và tồn tại suốt thời gian chạy chương trình. Giá trị của nó không bị mất đi ngay cả khi kết thúc hàm. Tuy nhiên khác với biến toàn cục có thể gọi trong tất cả mọi nơi trong chương trình, thì biến cục bộ static chỉ có thể được gọi trong nội bộ hàm khởi tạo ra nó. Mỗi lần hàm được gọi, giá trị của biến chính bằng giá trị tại lần gần nhất hàm được gọi.

Biến static toàn cục

- Biến toàn cục static sẽ chỉ có thể được truy cập và sử dụng trong File khai báo nó, các File khác không có cách nào truy cập được.

So sánh struct và union

Về mặt ý nghĩa, struct và union cơ bản giống nhau. Tuy nhiên, về mặt lưu trữ trong bộ nhớ, chúng có sự khác biệt rõ rệt như sau:

- struct: Dữ liệu của các thành viên của struct được lưu trữ ở những vùng nhớ khác nhau. Do đó kích thước của 1 struct tối thiểu bằng kích thước các thành viên cộng lại tại vì còn phụ thuộc vào bộ nhớ đệm (struct padding)
- Union : Dữ liệu các thành viên sẽ dùng chung 1 vùng nhớ. Kích thước của union được tính là kích thước lớn nhất của kiểu dữ liệu trong union. Việc thay đổi nội dung của 1 thành viên sẽ dẫn đến thay đổi nội dung của các thành viên khác.

Sự khác nhau giữa bộ nhớ Heap và bộ nhớ Stack

- Bộ nhớ Heap và bộ nhớ Stack bản chất đều cùng là vùng nhớ được tạo ra và lưu trữ trong RAM khi chương trình được thực thi.

- Bộ nhớ Stack được dùng để lưu trữ các biến cục bộ trong hàm, tham số truyền vào... Truy cập vào bộ nhớ này rất nhanh và được thực thi khi chương trình được biên dịch.

- Vùng nhớ Stack được quản lý bởi hệ điều hành, dữ liệu được lưu trong Stack sẽ tự động hủy khi hàm thực hiện xong công việc của mình.

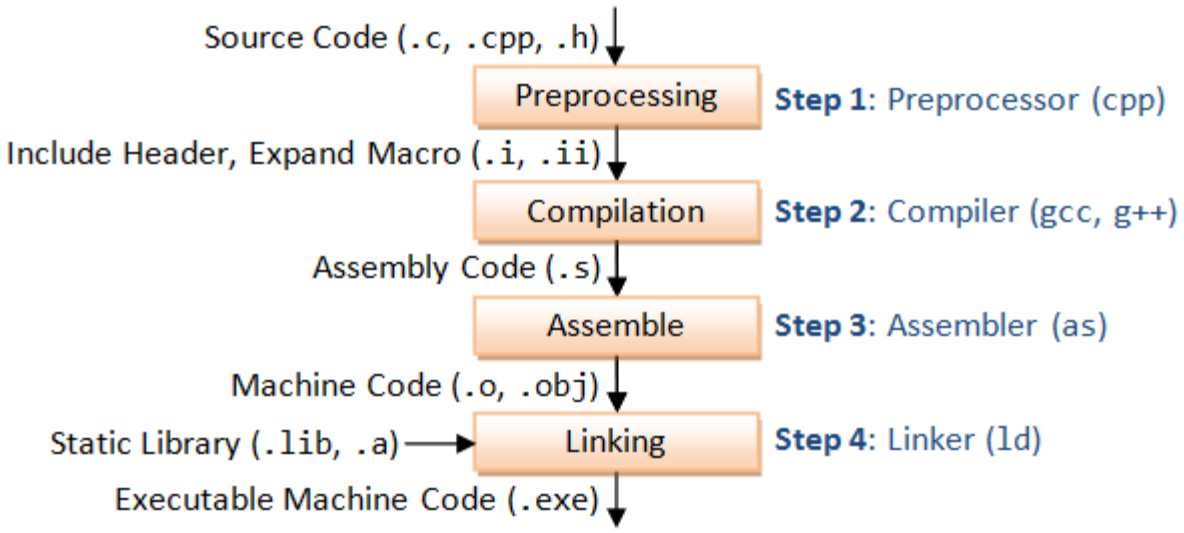
- Bộ nhớ Heap được dùng để lưu trữ vùng nhớ cho những biến con trỏ được cấp phát động bởi các hàm *malloc - calloc - realloc* (trong C).

- Nếu bạn liên tục cấp phát vùng nhớ mà không giải phóng thì sẽ bị lỗi tràn vùng nhớ Heap (Heap overflow).

Compiler

Quy trình dịch là quá trình chuyển đổi từ ngôn ngữ bậc cao (NNBC) (C/C++, Pascal, Java, C#…) sang ngôn ngữ đích (ngôn ngữ máy) để máy tính có thể hiểu và thực thi. Ngôn ngữ lập trình C là một ngôn ngữ dạng biên dịch. Chương trình được viết bằng C muốn chạy được trên máy tính phải trải qua một quá trình biên dịch để chuyển đổi từ dạng mã nguồn sang chương trình dạng mã thực thi. Quá trình được chia ra làm 4 giai đoạn chính:

- Giai đoàn tiền xử lý (Pre-processor)
- Giai đoạn dịch NNBC sang Asembly (Compiler)
- Giai đoạn dịch asembly sang ngôn ngữ máy (Asember)
- Giai đoạn liên kết (Linker)



1. Giai đoạn tiền xử lý – Preprocessor

Giai đoạn này sẽ thực hiện:

- Nhận mã nguồn
- Xóa bỏ tất cả chú thích, comments của chương trình
- Chỉ thị tiền xử lý (bắt đầu bằng #) cũng được xử lý

Ví dụ: chỉ thị #include cho phép ghép thêm mã chương trình của một tệp tiêu đề vào mã nguồn cần dịch. Các hằng số được định nghĩa bằng #define sẽ được thay thế bằng giá trị cụ thể tại mỗi nơi sử dụng trong chương trình.

2. Cộng đoạn dịch Ngôn Ngữ Bậc Cao sang Assembly

- Phân tích cú pháp (syntax) của mã nguồn NNBC
- Chuyển chúng sang dạng mã Assembly là một ngôn ngữ bậc thấp (hợp ngữ) gần với tập lệnh của bộ vi xử lý.

3. Công đoạn dịch Assembly

- Dich chương trình => Sang mã máy 0 và 1
- Một tệp mã máy (.obj) sinh ra trong hệ thống sau đó.

4. Giai đoạn Linker

- Trong giai đoạn này mã máy của một chương trình dịch từ nhiều nguồn (file .c hoặc file thư viện .lib) được liên kết lại với nhau để tạo thành chương trình đích duy nhất
- Mã máy của các hàm thư viện gọi trong chương trình cũng được đưa vào chương trình cuối trong giai đoạn này.

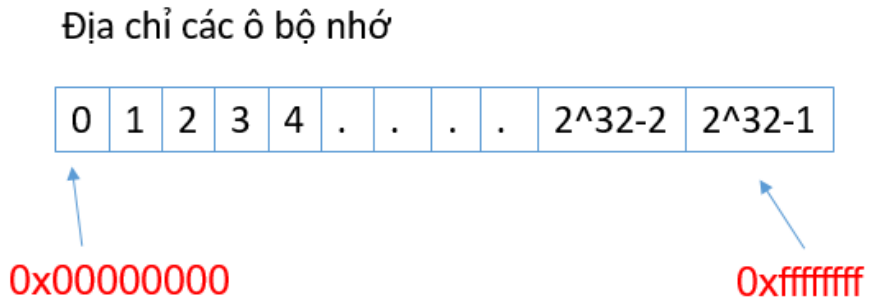
- Chính vì vậy mà các lỗi liên quan đến việc gọi hàm hay sử dụng biến tổng thể mà không tồn tại sẽ bị phát hiện. Kể cả lỗi viết chương trình chính không có hàm main() cũng được phát hiện trong liên kết.

Kết thúc quá trình tất cả các đối tượng được liên kết lại với nhau thành một chương trình có thể thực thi được (executable hay .exe) thống nhất.

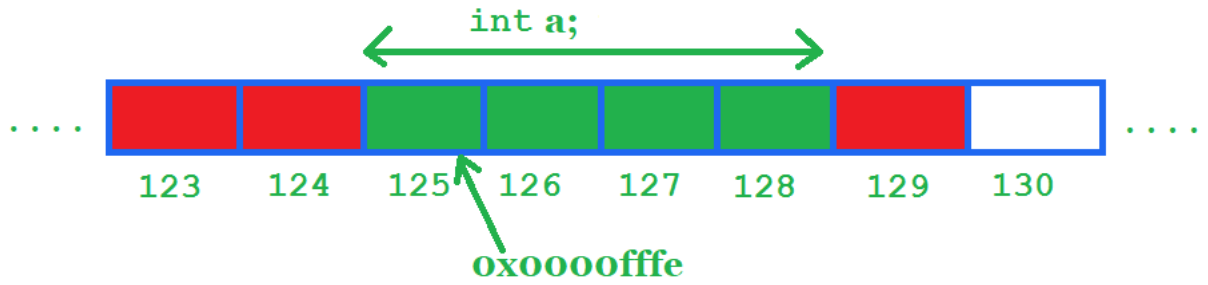
Khái niệm con trỏ

Bộ nhớ RAM chứa rất nhiều **ô nhớ**, **mỗi ô nhớ** có **kích thước 1 byte**.

Mỗi ô nhớ có **địa chỉ duy nhất** và địa chỉ này được đánh số từ **0 trở đi**. Nếu CPU 32 bit thì có 2^32 địa chỉ có thể đánh cho các ô nhớ trong RAM.



Khi khai báo biến, trình biên dịch dành riêng một vùng nhớ với địa chỉ duy nhất để lưu biến. Trình biên dịch có nhiệm vụ **liên kết** địa chỉ ô nhớ đó với tên biến. Khi gọi tên biến, nó sẽ truy xuất tự động đến ô nhớ đã liên kết với tên biến để lấy dữ liệu. Các bạn phải **luôn phân biệt** giữa **địa chỉ bộ nhớ** và **dữ liệu được lưu trong đó**.



Địa chỉ của biến bản chất cũng là **một con số** thường được biểu diễn ở hệ cơ số 16. Ta có thể sử dụng **con trỏ (pointer)** để lưu địa chỉ của các biến.

Con trỏ là gì?

Trong ngôn ngữ C/C++, **con trỏ (pointer)** là những biến lưu trữ địa chỉ bộ nhớ của những biến khác.

Trong hình trên, biến **var** lưu **giá trị 5** có địa chỉ là **0x61ff08**.
Biến **pointVar** là **biến con trỏ**, **lưu địa chỉ** của biến **var** (**trỏ đến vùng nhớ** của biến **var**), tức là nó lưu giá trị **0x61ff08**.

Con trỏ NULL

Con trỏ **NULL** là con trỏ lưu địa chỉ **0x00000000**. Tức địa chỉ bộ nhớ **0**, có ý nghĩa đặc biệt, cho biết con trỏ không trỏ vào đâu cả.

```
int *p2;//con trỏ chưa khởi tạo, vẫn trỏ đến một vùng nhớ nào đó không xác định
int *p3 = NULL;//con trỏ null không trỏ đến vùng nhớ nào
```

Kích thước của con trỏ

Ví dụ các khai báo con trỏ sau:

```
char *p1;
int *p2;
float *p3;
double *p4;
```

Kích thước của các biến con trỏ **có khác nhau không**? Con trỏ chỉ lưu địa chỉ nên **kích thước của mọi con trỏ là như nhau**. Kích thước này phụ thuộc vào môi trường hệ thống máy tính:

Môi trường **Windows 32 bit**: 4 bytes

- Môi trường **Windows 64 bit**: 8 bytes

Một số lưu ý khi sử dụng con trỏ

Khi khởi tạo con trỏ NULL

Chữ **NULL** phải viết hoa, viết thường **null** sẽ bị lỗi.

```
int *p1 = NULL;//đúng
int *p2 = null;//lỗi
```

Không nên sử dụng con trỏ khi chưa được khởi tạo

Kết quả tính toán có thể sẽ phát sinh những lỗi không lường trước được nếu chưa khởi tạo con trỏ.

Sử dụng biến con trỏ sai cách

==========******==========

Giao thức SPI

SPI (Serial Peripheral Interface) là một chuẩn truyền thông nối tiếp tốc độ cao do Motorola đề xuất.

- Các bit dữ liệu được truyền nối tiếp nhau và có xung clock đồng bộ.
- Giao tiếp song công, có thể truyền và nhận cùng một thời điểm.
- Khoảng cách truyền ngắn, được sử dụng để trao đổi dữ liệu với nhau giữa các chip trên cùng một bo mạch.
- Tốc độ truyền khoảng vài Mb/s.
- Các dòng vi điều khiển thường được tích hợp module giao tiếp SPI dùng để giao tiếp truyền dữ liệu với các vi điều khiển khác, hoặc giao tiếp với các ngoại vi bên ngoài như cảm biến, EEPROM, ADC, LCD, SD Card,…

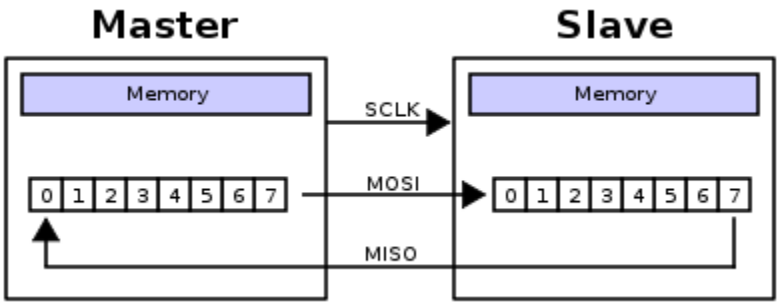
Giao tiếp 1 Master với 1 Slave

Bus SPI gồm có 4 đường tín hiệu:

- **SCLK**: Serial Clock
- **MOSI**: Master Out, Slave In

- **MISO**: Master In, Slave Out
- **SS**: Slave Select

Cách truyền và nhận dữ liệu

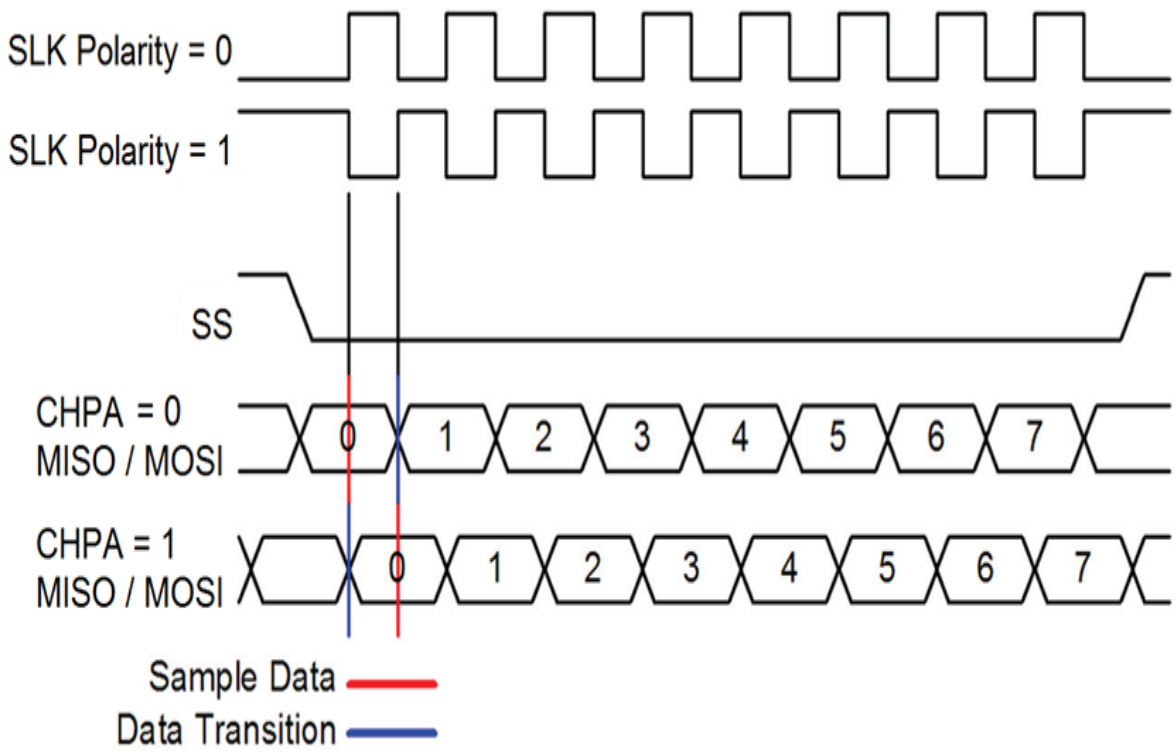


- Mỗi chip Master hay Slave sẽ có một thanh ghi dữ liệu 8 bit chứa dữ liệu cần gửi đi hoặc dữ liệu nhận về.
- Cứ mỗi xung nhịp do Master tạo ra trên chân SCLK, một bit trong thanh ghi dữ liệu của Master được truyền qua Slave trên đường MOSI, đồng thời một bit trong thanh ghi dữ liệu của Slave cũng được truyền qua cho Master trên đường MISO.

Các chế độ hoạt động

- Có 4 chế độ truyền nhận khác nhau

Mode	CPOL	CPHA
1	0	0
2	0	1
3	1	0
4	1	1



Giao Thức I2C
I2C chỉ sử dụng hai dây để truyền dữ liệu giữa các thiết bị:

SDA (Serial Data) - đường truyền cho master và slave để gửi và nhận dữ liệu.

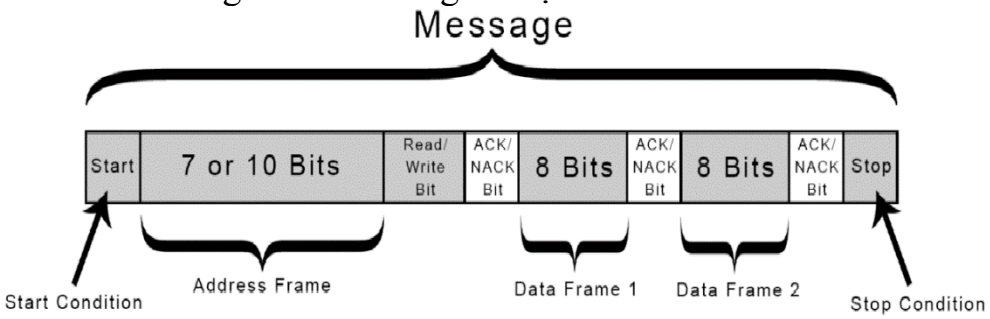
SCL (Serial Clock) - đường mang tín hiệu xung nhịp.

I2C là một giao thức truyền thông nối tiếp, vì vậy dữ liệu được truyền từng bit dọc theo một đường duy nhất (đường SDA).

I2C là đồng bộ, do đó đầu ra của các bit được đồng bộ hóa với việc lấy mẫu các bit bởi một tín hiệu xung nhịp được chia sẻ giữa master và slave. Tín hiệu xung nhịp luôn được điều khiển bởi master.

Cách hoạt động của I2C
Với I2C, dữ liệu được truyền trong các tin nhắn. Tin nhắn được chia thành các khung dữ liệu. Mỗi tin nhắn có một khung địa chỉ chứa địa chỉ nhị phân của địa chỉ

slave và một hoặc nhiều khung dữ liệu chứa dữ liệu đang được truyền. Thông điệp cũng bao gồm điều kiện khởi động và điều kiện dừng, các bit đọc / ghi và các bit ACK / NACK giữa mỗi khung dữ liệu:



Điều kiện khởi động: Đường SDA chuyển từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ mức cao xuống mức thấp.

Điều kiện dừng: Đường SDA chuyển từ mức điện áp thấp sang mức điện áp cao sau khi đường SCL chuyển từ mức thấp lên mức cao.

Khung địa chỉ: Một chuỗi 7 hoặc 10 bit duy nhất cho mỗi slave để xác định slave khi master muốn giao tiếp với nó.

Bit Đọc / Ghi: Một bit duy nhất chỉ định master đang gửi dữ liệu đến slave (mức điện áp thấp) hay yêu cầu dữ liệu từ nó (mức điện áp cao).

Bit ACK / NACK: Mỗi khung trong một tin nhắn được theo sau bởi một bit xác nhận / không xác nhận. Nếu một khung địa chỉ hoặc khung dữ liệu được nhận thành công, một bit ACK sẽ được trả lại cho thiết bị gửi từ thiết bị nhận.

Địa chỉ
I2C không có các đường Slave Select như SPI, vì vậy cần một cách khác để cho slave biết rằng dữ liệu đang được gửi đến slave này chứ không phải slave khác. Nó thực hiện điều này bằng cách định địa chỉ. Khung địa chỉ luôn là khung đầu tiên sau bit khởi động trong một tin nhắn mới.

Master gửi địa chỉ của slave mà nó muốn giao tiếp với mọi slave được kết nối với nó. Sau đó, mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ phù hợp, nó sẽ gửi lại một bit ACK điện áp thấp cho master. Nếu địa chỉ không khớp, slave không làm gì cả và đường SDA vẫn ở mức cao.

Bit đọc / ghi

Khung địa chỉ bao gồm một bit duy nhất ở cuối tin nhắn cho slave biết master muốn ghi dữ liệu vào nó hay nhận dữ liệu từ nó. Nếu master muốn gửi dữ liệu đến slave, bit đọc / ghi ở mức điện áp thấp. Nếu master đang yêu cầu dữ liệu từ slave, thì bit ở mức điện áp cao.

Data
Sau khi master phát hiện bit ACK từ slave, data đã sẵn sàng được gửi.

Khung dữ liệu luôn có độ dài 8 bit và được gửi với bit quan trọng nhất trước. Mỗi khung dữ liệu ngay sau đó là một bit ACK / NACK để xác minh rằng khung đã được nhận thành công. Bit ACK phải được nhận bởi master hoặc slave (tùy thuộc vào cái nào đang gửi dữ liệu) trước khi khung dữ liệu tiếp theo có thể được gửi.

Sau khi tất cả các khung dữ liệu đã được gửi, master có thể gửi một điều kiện dừng cho slave để tạm dừng quá trình truyền. Điều kiện dừng là sự chuyển đổi điện áp từ thấp lên cao trên đường SDA sau khi chuyển tiếp từ thấp lên cao trên đường SCL , với đường SCL vẫn ở mức cao.

Các bước truyền dữ liệu I2C

1. Master gửi điều kiện khởi động đến mọi slave được kết nối bằng cách chuyển đường SDA từ mức điện áp cao sang mức điện áp thấp trước khi chuyển đường SCL từ mức cao xuống mức thấp.

2. Master gửi cho mỗi slave địa chỉ 7 hoặc 10 bit của slave mà nó muốn giao tiếp, cùng với bit đọc / ghi.

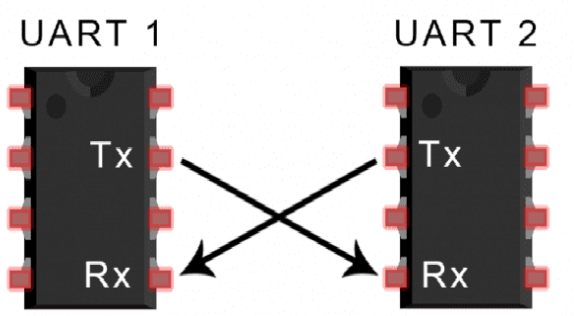
3. Mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ trùng khớp, slave sẽ trả về một bit ACK bằng cách kéo dòng SDA xuống thấp cho một bit. Nếu địa chỉ từ master không khớp với địa chỉ của slave, slave rời khỏi đường SDA cao.
4. Master gửi hoặc nhận khung dữ liệu.

5. Sau khi mỗi khung dữ liệu được chuyển, thiết bị nhận trả về một bit ACK khác cho thiết bị gửi để xác nhận đã nhận thành công khung.

6. Để dừng truyền dữ liệu, master gửi điều kiện dừng đến slave bằng cách chuyển đổi mức cao SCL trước khi chuyển mức cao SDA.

Giao thức UART

UART hay bộ thu-phát không đồng bộ đa năng là một trong những hình thức giao tiếp kỹ thuật số giữa thiết bị với thiết bị đơn giản và lâu đời nhất. Bạn có thể tìm thấy các thiết bị UART trong một phần của mạch tích hợp (IC) hoặc dưới dạng các thành phần riêng lẻ. Các UART giao tiếp giữa hai nút riêng biệt bằng cách sử dụng một cặp dẫn và một nối đất chung.



Hướng dẫn giao tiếp UART

Vì nó là thiết lập phổ quát nên chúng ta có thể định cấu hình UART để hoạt động với nhiều loại giao thức nối tiếp khác nhau. UART đã được điều chỉnh thành các đơn vị chip đơn vào đầu những năm 1970, bắt đầu với Western Digital's WD1402A.

Trong một sơ đồ giao tiếp UART:

1. Chân Tx (truyền) của một chip kết nối trực tiếp với chân Rx (nhận) của chip kia và ngược lại. Thông thường, quá trình truyền sẽ diễn ra ở 3.3V hoặc 5V. UART là một giao thức một master, một slave, trong đó một thiết bị được thiết lập để giao tiếp với duy nhất một thiết bị khác.

2. Dữ liệu truyền đến và đi từ UART song song với thiết bị điều khiển (ví dụ: CPU).

3. Khi gửi trên chân Tx, UART đầu tiên sẽ dịch thông tin song song này thành nối tiếp và truyền đến thiết bị nhận.

4. UART thứ hai nhận dữ liệu này trên chân Rx của nó và biến đổi nó trở lại thành song song để giao tiếp với thiết bị điều khiển của nó.

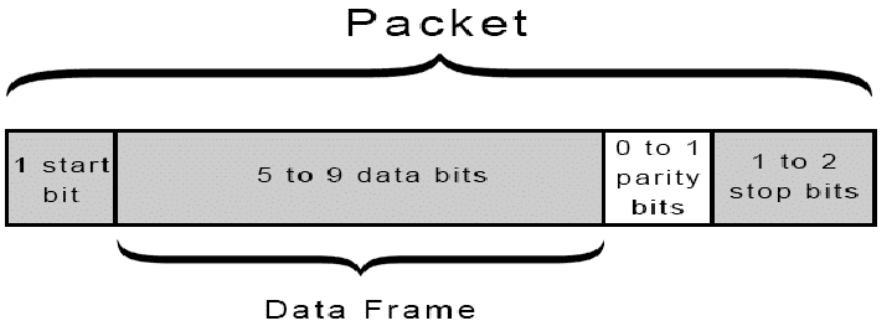
UART truyền dữ liệu nối tiếp, theo một trong ba chế độ:

- Full duplex: Giao tiếp đồng thời đến và đi từ mỗi master và slave

- Half duplex: Dữ liệu đi theo một hướng tại một thời điểm

- Simplex: Chỉ giao tiếp một chiều

Dữ liệu truyền qua UART được tổ chức thành các gói. Mỗi gói chứa 1 bit bắt đầu, 5 đến 9 bit dữ liệu (tùy thuộc vào UART), một bit chẵn lẻ tùy chọn và 1 hoặc 2 bit dừng.



Bit bắt đầu
Đường truyền dữ liệu UART thường được giữ ở mức điện áp cao khi không truyền dữ liệu. Để bắt đầu truyền dữ liệu, UART truyền sẽ kéo đường truyền từ mức cao xuống mức thấp trong một chu kỳ clock. Khi UART nhận phát hiện sự chuyển đổi điện áp cao xuống thấp, nó bắt đầu đọc các bit trong khung dữ liệu ở tần số của tốc độ truyền.

Khung dữ liệu
Khung dữ liệu chứa dữ liệu thực tế được chuyển. Nó có thể dài từ 5 bit đến 8 bit nếu sử dụng bit chẵn lẻ. Nếu không sử dụng bit chẵn lẻ, khung dữ liệu có thể dài 9 bit. Trong hầu hết các trường hợp, dữ liệu được gửi với bit ít quan trọng nhất trước tiên.

Bit chẵn lẻ
Bit chẵn lẻ là một cách để UART nhận cho biết liệu có bất kỳ dữ liệu nào đã thay đổi trong quá trình truyền hay không. Bit có thể bị thay đổi bởi bức xạ điện từ, tốc độ truyền không khớp hoặc truyền dữ liệu khoảng cách xa. Sau khi UART nhận đọc khung dữ liệu, nó sẽ đếm số bit có giá trị là 1 và kiểm tra xem tổng số là số chẵn hay lẻ. Nếu bit chẵn lẻ là 0 (tính chẵn), thì tổng các bit 1 trong khung dữ liệu phải là một số chẵn. Nếu bit chẵn lẻ là 1 (tính lẻ), các bit 1 trong khung dữ liệu sẽ tổng thành một số lẻ. Khi bit chẵn lẻ khớp với dữ liệu, UART sẽ biết rằng quá trình truyền không có lỗi. Nhưng nếu bit chẵn lẻ là 0 và tổng là số lẻ; hoặc bit chẵn lẻ là 1 và tổng số là chẵn, UART sẽ biết rằng các bit trong khung dữ liệu đã thay đổi.

Bit dừng
Để báo hiệu sự kết thúc của gói dữ liệu, UART gửi sẽ điều khiển đường truyền dữ liệu từ điện áp thấp đến điện áp cao trong ít nhất khoảng 2 bit.

Có thể tóm tắt lại như sau. Quá trình truyền dữ liệu diễn ra dưới dạng các gói dữ liệu, bắt đầu bằng một bit bắt đầu, đường mức cao được kéo xuống đất. Sau bit bắt đầu, năm đến chín bit dữ liệu truyền trong khung dữ liệu của gói, theo sau là bit chẵn lẻ tùy chọn để xác minh việc truyền dữ liệu thích hợp. Cuối cùng, một hoặc nhiều bit dừng được truyền ở nơi đường đặt ở mức cao. Như vậy là kết thúc một gói.

UART là giao thức không đồng bộ, do đó không có đường clock nào điều chỉnh tốc độ truyền dữ liệu. Người dùng phải đặt cả hai thiết bị để giao tiếp ở cùng tốc độ. Tốc độ này được gọi là tốc độ truyền, được biểu thị bằng bit trên giây hoặc bps. Tốc độ truyền thay đổi đáng kể, từ 9600 baud đến 115200 và hơn nữa. Tốc độ truyền giữa UART truyền và nhận chỉ có thể chênh lệch khoảng 10% trước khi thời gian của các bit bị lệch quá xa.

Mặc dù UART là giao thức cũ và chỉ có thể giao tiếp giữa một master và slave duy nhất, nhưng nó dễ thiết lập và cực kỳ linh hoạt. Do đó, bạn có thể gặp nó khi làm việc với các dự án vi điều khiển. UART có thể là một phần của hệ thống mà bạn sử dụng hàng ngày, mà có thể bạn không nhận ra.

Ngắt (Interrupt)

là một số sự kiện khẩn cấp bên trong hoặc bên ngoài bộ vi điều khiển xảy ra, buộc vi điều khiển tạm dừng thực hiện chương trình hiện tại, phục vụ ngay lập tức nhiệm vụ mà ngắt yêu cầu – nhiệm vụ này gọi là trình phục vụ ngắt (**ISR:** Interrupt Service Routine).

Trình phục vụ ngắt

Đối với mỗi ngắt thì phải có một **trình phục vụ ngắt** (**ISR**) hay trình quản lý ngắt để đưa ra nhiệm vụ cho bộ vi điều khiển khi được gọi ngắt. Khi một ngắt được gọi thì bộ vi điều khiển sẽ chạy trình phục vụ ngắt. Đối với mỗi ngắt thì có một vị trí cố định trong bộ nhớ để giữ địa chỉ **ISR** của nó. Nhóm vị trí bộ nhớ được dành riêng để lưu giữ địa chỉ của các **ISR** được gọi là **bảng vector ngắt**. Xem **Hình 1**.

Ngắt	Cờ ngắt	Địa chỉ trình phục vụ ngắt	Số thứ tự ngắt
Reset	-	0000h	-
Ngắt ngoài 0	**IE0**	0003h	**0**
Timer 0	**TF0**	000Bh	**1**
Ngắt ngoài 1	**IE1**	0013h	**2**
Timer 1	**TF1**	001Bh	**3**
Ngắt truyền thông	**RI/TI**	0023h	**4**

Trong lập trình **C** trên **Keil c** cho 8051, chúng ta khai báo **trình phục vụ ngắt** theo cấu trúc sau:

```
Void  Name (void) interrupt X                //( X: là số thứ tự của ngắt )
{
      // chương trình phục vụ ngắt
}
```

Khi đó địa chỉ ngắt sẽ được tự động tính bằng:

Interrupt Address = (X * 8) + 3

1.4 Quy trình khi thực hiện một ngắt

Khi kích hoạt một ngắt bộ vi điều khiển thực hiện các bước sau:

Ø Nó hoàn thành nốt lệnh đang thực hiện và lưu địa chỉ của **lệnh kế tiếp** vào ngăn xếp.
Ø Nó cũng **lưu tình trạng hiện tại** của tất cả các ngắt.
Ø Nó nhảy đến một vị trí cố định trong bộ nhớ được gọi là **bảng vector ngắt**, nơi lưu giữ địa chỉ của một **trình phục vụ ngắt**.
Ø Bộ vi điều khiển nhận địa chỉ **ISR** từ bảng vector ngắt và nhảy tới đó. Nó bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của **ISR** và trở về chương trình chính từ ngắt.
Ø Khi bộ vi điều khiển quay trở về nơi nó đã bị ngắt. Trước hết nó nhận địa chỉ của bộ đếm chương trình PC từ ngăn xếp bằng cách kéo 02 byte trên đỉnh của ngăn xếp vào PC. Sau đó bắt đầu thực hiện tiếp các lệnh từ địa chỉ đó.

1.5 Các bước cho phép và cấm ngắt

Khi bật lại nguồn thì tất cả mọi ngắt đều bị cấm (bị che), có nghĩa là không có ngắt nào được bộ vi điều khiển đáp ứng trừ khi chúng được kích hoạt.

Các ngắt phải được kích hoạt bằng phần mềm để bộ vi điều khiển đáp ứng chúng. Có một thanh ghi được gọi là thanh ghi cho phép ngắt **IE** (Interrupt Enable) – ở địa chỉ A8H chịu trách nhiệm về việc cho phép và cấm các ngắt. **Hình 2** trình bày chi tiết về thanh ghi **IE**.

Bit	Tên	Địa chỉ	Chức năng
7	**EA**	AFh	Cho phép/cấm hoạt động của cả thanh ghi
6	-	AEh	Chưa sử dụng
5	-	ADh	Chưa sử dụng
4	**ES**	ACh	Cho phép ngắt cổng truyền thông nối tiếp
3	**ET1**	ABh	Cho phép ngắt Timer 1
2	**EX1**	AAh	Cho phép ngắt ngoài 1
1	**ET0**	A9h	Cho phép ngắt Timer 0
0	**EX0**	A8h	Cho phép ngắt ngoài 0

Hình 2: Thanh ghi cho phép ngắt **IE**.

Để cho phép một ngắt ta phải thực hiện các bước sau:

Ø Nếu **EA = 0** thì **không** có ngắt nào được đáp ứng cho dù bit tương ứng của nó trong **IE**có giá trị cao. **Bit D7 - EA** của thanh ghi **IE** phải được bật lên cao để cho phép các bit còn lại của thanh ghi hoạt động được.
Ø Nếu **EA = 1** thì tất cả mọi ngắt đều được phép và sẽ được đáp ứng nếu các bit tương ứng của chúng trong **IE** có **mức cao**.

Để hiểu rõ điểm quan trọng này ta hãy xét **ví dụ 1.**

Ví dụ 1:
Hãy lập trình cho 8051:
a) cho phép **ngắt nối tiếp**, **ngắt Timer0** và **ngắt phần cứng ngoài 1** (EX1)
Lời giải:

```
#include<at89x51.h>
main()
{
        //a)
        IE=0x96;        //1001 0110: lệnh này tương đương với 4 lệnh phía dưới

        EA=1;           //Cho phép sử dụng ngắt
        ES=1;           //Cho phép ngắt cổng nối tiếp
        ET0=1;          //Cho phép ngắt timer0
        EX1=1;          //Cho phép ngắt ngoài 1
```

```
        //b)
        ET0=0;          //Cấm ngắt timer0

        //c)
       EA=0;           //Cấm tất cả các ngắt

        while(1)
        {
                //Chương trình chính
                //…
        }
}
```

Timer

Bộ đếm/Bộ định thời: Đây là các ngoại vi được thiết kế để thực hiện một nhiệm vụ đơn giản: đếm các xung nhịp. Mỗi khi có thêm một xung nhịp tại đầu vào đếm thì giá trị của bộ đếm sẽ được tăng lên 01 đơn vị (trong chế độ đếm tiến/đếm lên) hay giảm đi 01 đơn vị (trong chế độ đếm lùi/đếm xuống).

Xung nhịp đưa vào đếm có thể là một trong hai loại:

Ø Xung nhịp bên trong IC: Đó là xung nhịp được tạo ra nhờ kết hợp mạch dao động bên trong IC và các linh kiện phụ bên ngoài nối với IC. Trong trường hợp sử dụng xung nhịp loại này, người ta gọi là các **bộ định thời** (**timers**). Do xung nhịp bên loại này thường đều đặn nên ta có thể dùng để **đếm thời gian** một cách khá chính xác.
Ø Xung nhịp bên ngoài IC: Đó là các tín hiệu logic thay đổi liên tục giữa 02 mức 0-1 và không nhất thiết phải là đều đặn. Trong trường hợp này người ta gọi là các **bộ đếm** (**counters**). Ứng dụng phổ biến của các bộ đếm là **đếm các sự kiện bên ngoài** như đếm các sản phẩm chạy trên băng chuyền, đếm xe ra/vào kho bãi…

Một khái niệm quan trọng cần phải nói đến là sự kiện "**tràn**" (**overflow**). Nó được hiểu là sự kiện bộ đếm đếm vượt quá giá trị tối đa mà nó có thể biểu diễn và quay trở về giá trị 0. Với bộ đếm 8 bit, giá trị tối đa là 255 (tương đương với FF trong hệ Hexa) và là 65535 (FFFFH) với bộ đếm 16 bit.

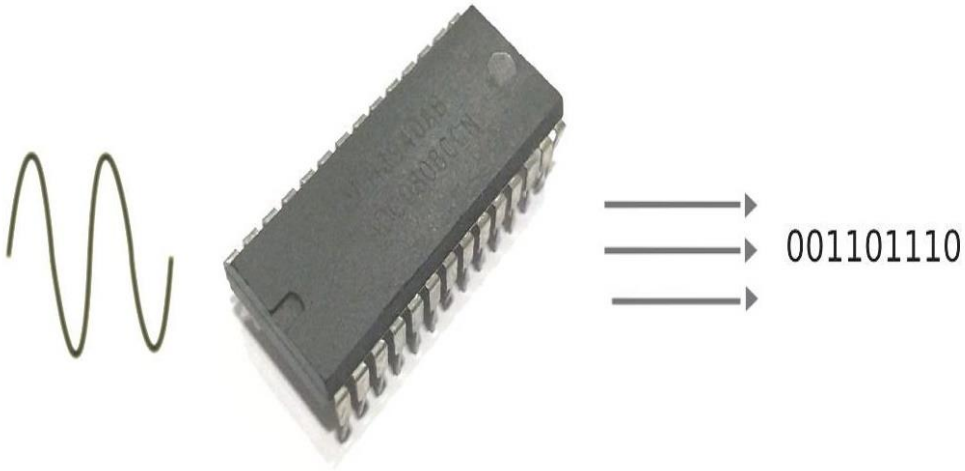
```
static void TIM4_Config(void)

{

  /* TIM4 configuration:

   - TIM4CLK is set to 16 MHz, the TIM4 Prescaler is equal to 128 so the TIM1 counter

   clock used is 16 MHz / 128 = 125 000 Hz

  - With 125 000 Hz we can generate time base:

      max time base is 2.048 ms if TIM4_PERIOD = 255 --> (255 + 1) / 125000 = 2.048 ms

      min time base is 0.016 ms if TIM4_PERIOD = 1   --> (  1 + 1) / 125000 = 0.016 ms

  - In this example we need to generate a time base equal to 1 ms

   so TIM4_PERIOD = (0.001  125000 - 1) = 124 /


  / Time base configuration /

  TIM4_TimeBaseInit(TIM4_PRESCALER_128, 16);

  / Clear TIM4 update flag /

  TIM4_ClearFlag(TIM4_FLAG_UPDATE);

  / Enable update interrupt /

  TIM4_ITConfig(TIM4_IT_UPDATE, ENABLE);


  / enable interrupts /

  enableInterrupts();
```

```
  / Enable TIM4 /
  TIM4_Cmd(ENABLE);
}

INTERRUPT_HANDLER(TIM4_UPD_OVF_IRQHandler, 23)
{
  if(TIM2_GetITStatus(TIM2_IT_UPDATE) == 1){
   TimingDelay_Decrement();
  / Cleat Interrupt Pending bit /
  TIM4_ClearITPendingBit(TIM4_IT_UPDATE);
  }

}
```

ADC

Bộ chuyển đổi ADC là gì

ADC là từ viết tắt của Analog to Digital Converter hay bộ chuyển đổi analog sang kỹ thuật số là một mạch chuyển đổi giá trị điện áp liên tục (analog) sang giá trị nhị phân (kỹ thuật số) mà thiết bị kỹ thuật số có thể hiểu được sau đó có thể được sử dụng để tính toán kỹ thuật số. Mạch ADC này có thể là vi mạch ADC hoặc được nhúng vào một bộ vi điều khiển.



Tại sao phải chuyển đổi analog sang kỹ thuật số

Thiết bị điện tử ngày nay hoàn toàn là kỹ thuật số, không còn là thời kỳ của máy tính analog. Thật không may cho các hệ thống kỹ thuật số, thế giới chúng ta đang sống vẫn là analog và đầy màu sắc, không chỉ đen và trắng.

Ví dụ, một cảm biến nhiệt độ như LM35 tạo ra điện áp phụ thuộc vào nhiệt độ, trong trường hợp của thiết bị cụ thể nó sẽ tăng 10mV khi nhiệt độ tăng lên mỗi độ. Nếu chúng ta kết nối trực tiếp thiết bị này với đầu vào kỹ thuật số, nó sẽ ghi là cao hoặc thấp tùy thuộc vào các ngưỡng đầu vào, điều này là hoàn toàn vô dụng.

Thay vào đó, chúng ta sử dụng một bộ ADC để chuyển đổi đầu vào điện áp analog thành một chuỗi các bit có thể được kết nối trực tiếp với bus dữ liệu của bộ vi xử lý và được sử dụng để tính toán.

ADC hoạt động như thế nào

Một cách rất hay để xem xét hoạt động của ADC là tưởng tượng nó như một bộ chia tỷ lệ toán học. Tỷ lệ về cơ bản là ánh xạ các giá trị từ dải này sang dải khác, vì vậy ADC ánh xạ một giá trị điện áp sang một số nhị phân.

Những gì chúng ta cần là một thứ có thể chuyển đổi điện áp thành một loạt các mức logic, ví dụ như trong một thanh ghi. Tất nhiên, các thanh ghi chỉ có thể chấp nhận các mức logic làm đầu vào, vì vậy nếu bạn kết nối tín hiệu trực tiếp với đầu vào logic, kết quả sẽ không tốt. Vì vậy cần có một giao diện ở giữa logic và điện áp đầu vào analog.

Dưới đây là một số tính năng quan trọng của ADC, trong khi xem qua, chúng ta sẽ tìm hiểu cách nó hoạt động.

Điện áp tham chiếu

Tất nhiên, không có ADC nào là tuyệt đối, vì vậy điện áp được ánh xạ tới giá trị nhị phân lớn nhất được gọi là điện áp tham chiếu. Ví dụ: trong bộ chuyển đổi 10 bit với 5V làm điện áp tham chiếu, 1111111111 (tất cả các bit một, số nhị phân 10 bit cao nhất có thể) tương ứng với 5V và 0000000000 (số thấp nhất tương ứng với 0V). Vì vậy, mỗi bước nhị phân lên đại diện cho khoảng 4,9mV, vì có thể có 1024 chữ số trong 10 bit. Số đo điện áp trên mỗi bit này được gọi là độ phân giải của ADC.