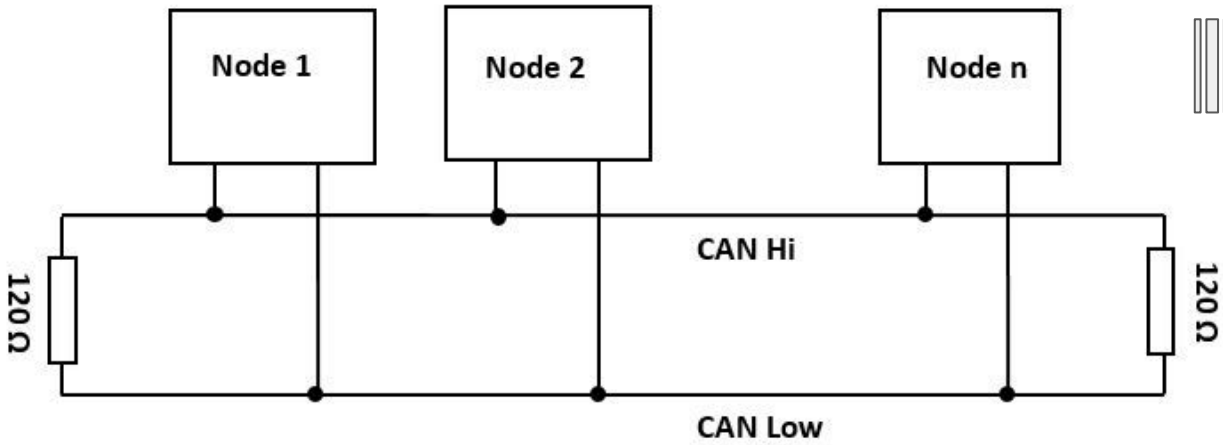
Bài 13: CAN

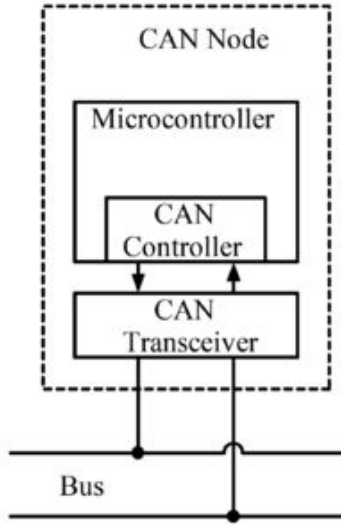
Phan Minh Thông

CAN Node - CAN Bus

Trong mạng CAN, các thiết bị được kết nối trên cùng 1 đường gồm 2 dây CAN_H và CAN_L, gọi là bus. Mỗi thiết bị trong mạng được gọi là 1 Node, gồm:

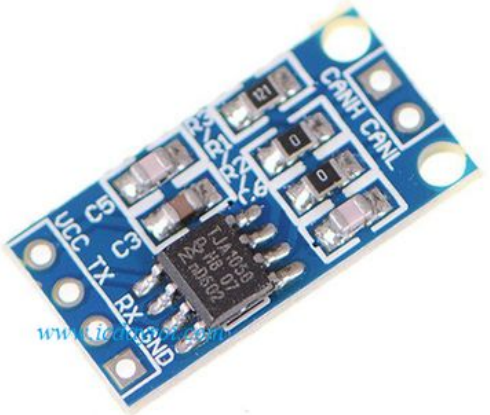
- Vi điều khiển: Chịu trách nhiệm truyền nhận xử lý data.
- CAN Controller: Thực hiện chức năng của giao thức CAN.
- CAN Transceiver: Giúp tạo điện áp cho Bus.



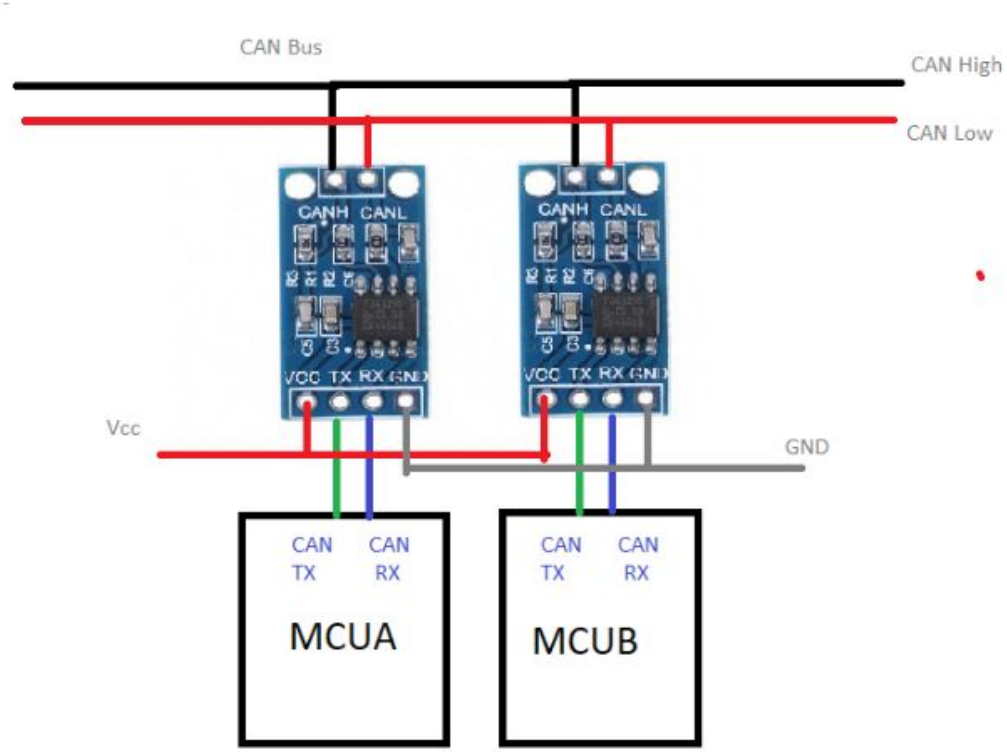


Chuẩn bị phần cứng.

Ở bài này ta sẽ thực hiện truyền nhận giữa 2 Node, nên cần 2 vi điều khiển: 1 stm32 và 1 vi điều khiển bất kì hỗ trợ CAN.
Ngoài ra, cần thêm 2 module CAN Transceiver: JTA1050(https://icdayroi.com/module-truyen-thong-can-bus-tja1050)

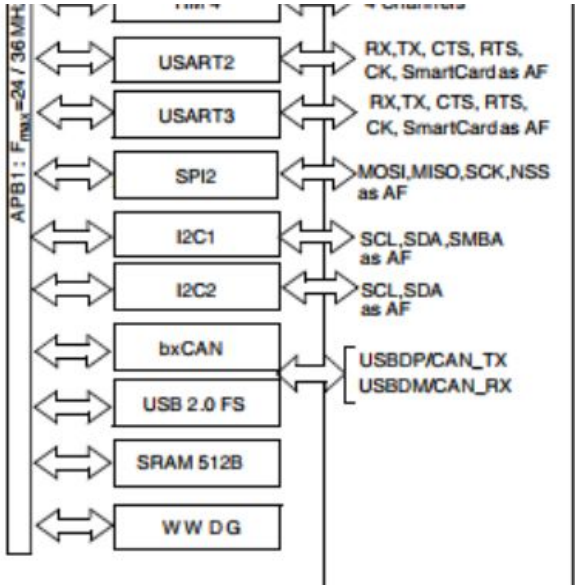


Sơ đồ nối dây.


CAN Bus
CAN High
CAN Low
CANH CANL
VCC TX RX GND
CANH CANL
VCC TX RX GND
Vcc
GND
CAN TX
CAN RX
MCUA
CAN TX
CAN RX
MCUB

Cấu hình CAN.

- Tương tự các ngoại vi, CAN được cấp xung từ APB1.
- Cấu hình cho 2 chân TX và RX của bộ CAN.
  - RX: Mode In_Floating.
  - TX: Mode AF_PP.



PA11	USART1_CTS/ CANRX(9)/ USBDM/ TIM1_CH4(9)	-
PA12	USART1_RTS/ CANTX(9) /USBDP TIM1_ETR(9)	-

Cấu hình CAN.

Các tham số cho CAN được cấu hình trong struct CAN_InitTypeDef. Gồm:

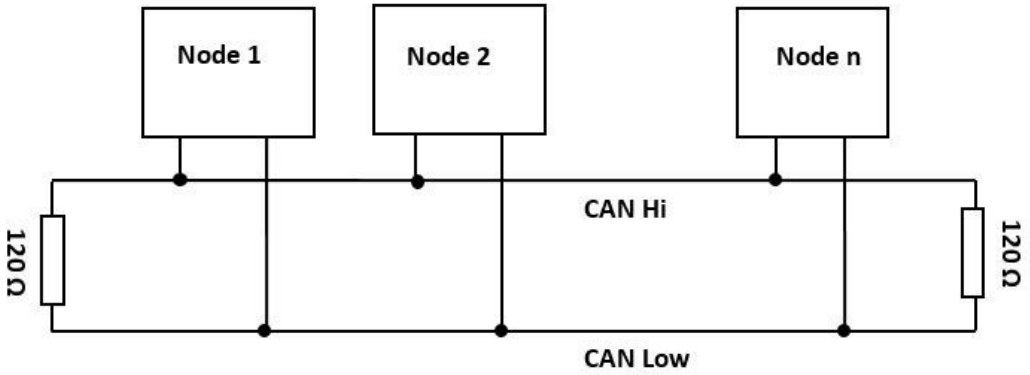
- CAN_TTCM: **Chế độ giao tiếp được kích hoạt theo thời gian** Ấn đinh khoảng thời gian khi truyền message.
- CAN_ABOM: **Quản lý ngắt bus tự động.** Nếu trong quá trình truyền xảy ra lỗi, bus sẽ được ngắt. Bit này quy định việc CAN có quay về trạng thái bình thường hay không.
- CAN_AWUM: **Chế độ đánh thức tự động.** Nếu CAN hoạt động ở SleepMode, Bit này quy định việc đánh thức CAN theo cách thủ công hay tự động.
- CAN_NART: **Không tự động truyền lại.** CAN sẽ thử lại để truyền tin nhắn nếu các lần thử trước đó không thành công. Nếu set bit này thì sẽ không truyền lại. Nên set bit khi sử dụng chung với CAN_TTCM, nếu không thì nên để =0;
- CAN_RFLM: **Chế độ khóa nhận FIFO.** Chế độ khóa bộ đệm khi đầy.
- CAN_TXFP: **Ưu tiên truyền FIFO.** Đặt bit này =1, ưu tiên truyền các gói có ID thấp hơn. Nếu đặt lên 0, CAN sẽ ưu tiên các gói theo thứ tự trong bộ đệm.
- CAN_Mode: **Chế độ CAN:**
    - CAN_Mode_Normal: Gửi message thông thường.
    - CAN_Mode_LoopBack: Các message gửi đi sẽ được lưu vào bộ nhớ đệm.
    - CAN_Mode_Silent: Chế độ chỉ nhận.
    - CAN_Mode_Silent_LoopBack: Kết hợp giữa 2 mode trên.

Cấu hình CAN.

Các tham số cho CAN được cấu hình trong struct CAN_InitTypeDef. Gồm:

- CAN_Prescaler: Cài đặt giá trị chia để tạo time quantum.
  - fCan = sysclk/CAN_Prescaler.
  - 1tq = 1/fCan.
- CAN_SJW: Thời gian trễ phần cứng, tính theo tq.
- CAN_BS1: Thời gian đồng bộ đầu frame truyền, tính theo tq.
- CAN_BS2: Thời gian đồng bộ cuối frame truyền, tính theo tq.

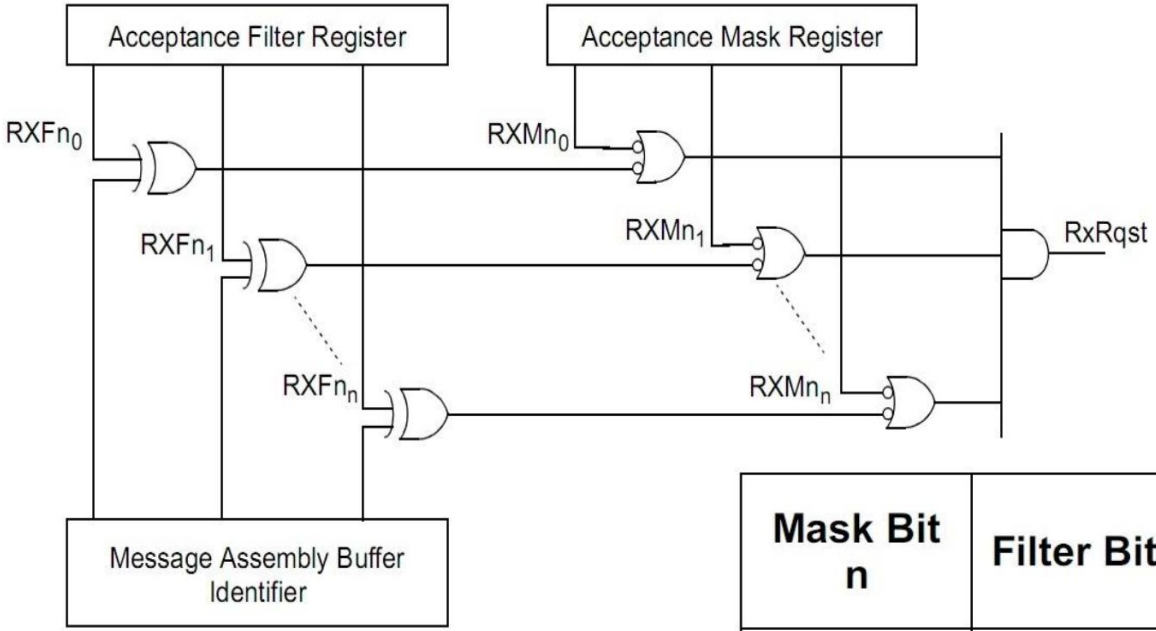
Tốc độ truyền CAN = 1/(SJW+BS1+BS2) .



Cấu hình CAN Mask & Filter.

CAN hỗ trợ bộ lọc ID, giúp các Node có thể lọc ra ID từ các message trên Bus để quyết định sẽ nhận massge nào. Các tham số cho bộ lọc được cấu hình trong CAN_FilterInitTypeDef:

- CAN_FilterNumber: Chọn bộ lọc để dùng, từ 0-13.
- CAN_FilterMode: Chế độ bộ lọc:
  - IdMask: Sử dụng mặt nạ bit để lọc ID.
  - IdList: Không sử dụng mặt nạ bit.
- CAN_FilterScale: Kích thước của bộ lọc, 32 hoặc 16 bit.
- CAN_FilterMaskIdHigh & CAN_FilterMaskIdLow: Giá trị cài đặt cho Mask, 32 bits.
- CAN_FilterIdHigh & CAN_FilterIdLow: Giá trị cài đặt cho bộ lọc, 32bits.
- CAN_FilterFIFOAssignment: .
- CAN_FilterActivation: Kích hoạt bộ lọc ID.



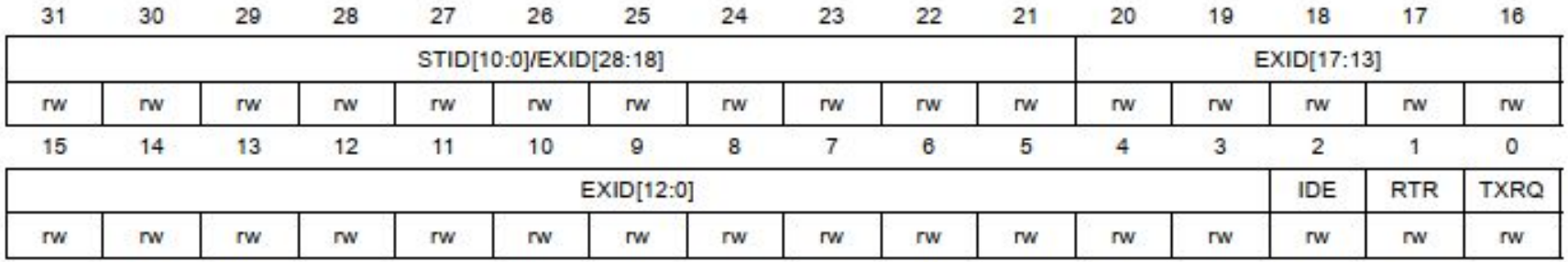
EX:
Mask: 0xFFF0
Filter: 0x005A
=>> ID được chấp nhận từ 0x0050-0x005F.

Mask Bit n	Filter Bit n	Message Identifier bit	Accept or Reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: X = don't care

Giá trị bộ lọc & giá trị ID trong massage.

Thanh ghi chứa giá trị ID của gói tin



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	TXRQ
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Để áp dụng được Mask và ID cho gói tin với ID là stdID, cần setup 11 bit cao của Mask cùng như của Filter.

Truyền - nhận CAN.

Để xác định được 1 gói tin, cần có **ID**, các bit **RTR**, **IDE**, **DLC** và tối đa **8** byte data như bài trước đã đề cập. Các thành phần này được tổ chức trong **CanTx/RxMsg**.

Hàm truyền: **uint8_t CAN_Transmit(CAN_TypeDef* CANx, CanTxMsg* TxMessage):**

- **CANx:** Bộ CAN cần dùng.
- **TxMessage**: Struct CanRxMsg cần truyền.

```
CanTxMsg TxMessage;

TxMessage.StdId = 0x123; // 11bit ID voi che do std
TxMessage.ExtId = 0x00;
TxMessage.RTR = CAN_RTR_DATA;
TxMessage.IDE = CAN_ID_STD;
TxMessage.DLC = len;

for (int i = 0; i < len; i++)
{
   TxMessage.Data[i] = data[i];
}

CAN_Transmit(CAN1, &TxMessage);
```

Truyền - nhận CAN.

Gói tin nhận được sẽ được lưu dưới dạng **CanRxMsg** struct. Gồm các thành phần tương tự **CanTxMsg** của gói tin nhận được.

Hàm **CAN_MessagePending(CAN_TypeDef* CANx, uint8_t FIFONumber)**: Trả về số lượng gói tin đang đợi trong FIFO của bộ CAN. Dùng hàm này để kiểm tra xem bộ CAN có đang truyền nhận hay không, nếu FIFO trống thì có thể nhận.

Hàm **CAN_Receive(CAN_TypeDef* CANx, uint8_t FIFONumber, CanRxMsg* RxMessage):** Nhận về 1 gói tin từ bộ **CANx**, lưu vào **RxMessage**.

Truyền - nhận CAN.

```
CanRxMsg RxMessage;
while (CAN_MessagePending(CAN1, CAN_FIFO0) <1 );
CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
      ID = RxMessage.StdId;
for (int i = 0; i < RxMessage.DLC; i++)
{
    TestArray[i] = RxMessage.Data[i];
}
```