Hello,

I give up. I wanted to make it all by myself but I need your help, guys. I have a problem with a CAN network- it works in loopback mode, reacts on IDs (filters are working), generates interrupts, everything is ok. But when I switch to normal mode... trasmitter works but receiver does not. I can see on osciloscope that there are normal CANH and CANL signals. And even if I send 8x'a' char- I can see sequences of the same combinations. I assume that trasmitter works.

Aaaand i would be the happiest man in the world, but receiver. This ugly receiver. It doesnt react. I dont know what that problem is connected to. I am using MCP2551 transceivers- I tried to use new ones- same situation. I tried to change resistor around RS pin (from 1k to 10k)- nothing. All the time there is 5V on RX pin in transceiver. In reference manual there is said that the bxCAN RX pin should be in input/pull up mode or floating- I treid both, nothing.

Can someone help me? Any idea would be helpful.

Btw, I used this website to count bititming with 72 MHz option:

http://www.bittiming.can-wiki.info/

It doesnt work, but I tried also other cominations, such as the one from Keil exemple. Nothing.

```
CAN_INIT.C

#include "CAN_init.h"
#include "stm32f10x.h"

CAN_message CAN_Rx, CAN_Tx;
volatile unsigned char CAN_Tx_0=1, CAN_Tx_1=1, CAN_Tx_2=1;
extern void USART_putchar(char ch);

void CAN_setup()
{
    CAN_clock_init();
    CAN_init_mode();

    CAN1->FMR |= CAN_FMR_FINIT;             // CAN initialization for filters ON, reception
deactivated
    CAN_filter_reset();                     // CAN filter reset- 13 filters has id=0
    CAN_filter_set(10,4);
    CAN1->FMR &= ~(CAN_FMR_FINIT);          // CAN initialization for filters OFF, reception
activated

    //CAN_test_mode();
    CAN_normal_mode();
}

void CAN_clock_init()
{
    RCC->APB1ENR |= RCC_APB1ENR_CAN1EN;     // Clock for CAN
    RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;     // Clock for alternate function

    AFIO->MAPR &= ~(AFIO_MAPR_CAN_REMAP);   // AFIO remap, TX->PA12, RX->PA11

    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;     // Clock for A Block

    GPIOA->CRH &= ~(GPIO_CRH_CNF11);        // PA11 CNF reset
    GPIOA->ODR |= GPIO_ODR_ODR11;           // PA11 Pull up
        GPIOA->CRH |= GPIO_CRH_CNF11_1;         // PA11 RX, input, pull up/pull down

    GPIOA->CRH &= ~(GPIO_CRH_CNF12);        // PA12 CNF reset
    GPIOA->CRH |= GPIO_CRH_MODE12;          // PA12 TX, 50 Mhz output
    GPIOA->CRH |= GPIO_CRH_CNF12_1;         // PA12 alternate function, push-pull
```

```
}

void CAN_init_mode()
{
    CAN1->MCR |= CAN_MCR_RESET;
    while (!(CAN1->MSR & CAN_MSR_SLAK));

    CAN1->MCR = CAN_MCR_INRQ;               // Enter initialization mode
    while (!(CAN1->MSR & CAN_MSR_INAK));    // Wait until CAN is in initialization mode


    CAN1->IER |= CAN_IER_TMEIE              // Transmit interrupt (when mailbox 0/1/2 gets empty)
              | CAN_IER_FMPIE0;             // Receive FIFO0 interrupt enable (when FMR0 isn't 0,
means new message)

    CAN1->BTR &= ~(((0x03) << 24) | ((0x07) << 20) | ((0x0F) << 16) | (0x1FF));     // BTR reset
  CAN1->BTR = 0x001c0008;

    NVIC_EnableIRQ(USB_HP_CAN1_TX_IRQn);    // Transmit interrupt declaration
    NVIC_EnableIRQ(USB_LP_CAN1_RX0_IRQn);   // Receive FIFO0 interrupt declaration
}

void CAN_filter_reset()
{
    unsigned int i;
    for (i=0;i<14;++i)
    {
        CAN_filter_set(0,i);
    }
}

void CAN_filter_set(unsigned int id, unsigned int filter_dec)
{
    unsigned int filter_bin;

    if (filter_dec > 13)
        return;

    filter_bin = (1 << filter_dec);             // Filter in decimal- transform into binary

    CAN1->FA1R &= ~filter_bin;                  // Deactive filter to configure

    id = (id << 21);
    CAN1->FS1R |= filter_bin;                   // Two 32-bit scale filter
    CAN1->FM1R |= filter_bin;                   // Identifier list mode
    CAN1->sFilterRegister[filter_dec].FR1 = id; // Identifier 1 set
    CAN1->sFilterRegister[filter_dec].FR2 = id; // Identifier 2 set
    CAN1->FFA1R &= ~(1 << filter_bin);

    CAN1->FA1R |= filter_bin;                   // Activate filter after configuration
}

void CAN_normal_mode()
{
    CAN1->MCR &= ~(CAN_MCR_INRQ);               // Leave initialization, CAN bus synchronization
ask
    if (!((CAN1->BTR & CAN_BTR_SILM) && (CAN1->BTR & CAN_BTR_LBKM)))
        while (CAN1->MSR & CAN_MSR_INAK);         // Waits until CAN bus is in idle state (receive
11 recessive bits)
}

void CAN_test_mode()
{
    CAN1->BTR |= CAN_BTR_SILM                                   // Silent mode
```

```
                        | CAN_BTR_LBKM;                                    // Loop back mode (combined
mode)
}

void CAN_send()
{
    CAN1->IER |= CAN_IER_TMEIE;                          // Transmit interrupt (when mailbox 0/1/2
gets empty)
    if (CAN_Tx_0 && (CAN1->TSR & CAN_TSR_TME0))         // Program uses transmit mailbox 0 when it
is empty (TME0) and it has sent message (interrupt)
    {
        CAN_message_write(&CAN_Tx, 0);                    // Message sending with mailbox 0
        CAN1->sTxMailBox[0].TIR |= CAN_TI0R_TXRQ;         // Send request with mailbox 0
        CAN_Tx_0=0;
    } else {
        if (CAN_Tx_1 && (CAN1->TSR & CAN_TSR_TME1))       // Program uses transmit mailbox 1 when
it is empty (TME1) and it has sent message (interrupt)
        {
            CAN_message_write(&CAN_Tx, 1);                  // Message sending with mailbox 1
            CAN1->sTxMailBox[1].TIR |= CAN_TI1R_TXRQ;       // Send request with mailbox 1
            CAN_Tx_1=0;
        } else {
            if (CAN_Tx_2 && (CAN1->TSR & CAN_TSR_TME2))     // Program uses transmit mailbox 2 when
it is empty (TME2) and it has sent message (interrupt)
            {
                CAN_message_write(&CAN_Tx, 2);                // Message sending with mailbox 2
                CAN1->sTxMailBox[2].TIR |= CAN_TI2R_TXRQ;     // Send request with mailbox 2
                CAN_Tx_2=0;
            }
        }
    }
}

void CAN_message_read(CAN_message *message)
{
    if (CAN1->RF0R & CAN_RF0R_FMP0)                                                 // Interrupt
if appeared new message in FIFO0
    {
        if (CAN1->sFIFOMailBox[0].RIR & CAN_RI0R_RTR)                                 // Getting
type of message (remote or data)
            message->type = REMOTE;
        else
            message->type = DATA;

        message->id = (CAN1->sFIFOMailBox[0].RIR >> 21);                                //
Getting identifier

        if (message->type == DATA)                                                    // If
message type is data, getting data length and data
        {
            message->len = (CAN1->sFIFOMailBox[0].RDTR & CAN_RDT0R_DLC);                  //
Getting data length

            switch (message->len-1)                                                     //
Getting data
            {
                case 7:
                    message->data[7] = ((CAN1->sFIFOMailBox[0].RDHR & CAN_RDH0R_DATA7) >> 24);
                case 6:
                    message->data[6] = ((CAN1->sFIFOMailBox[0].RDHR & CAN_RDH0R_DATA6) >> 16);
                case 5:
                    message->data[5] = ((CAN1->sFIFOMailBox[0].RDHR & CAN_RDH0R_DATA5) >> 8);
                case 4:
                    message->data[4] = (CAN1->sFIFOMailBox[0].RDHR & CAN_RDH0R_DATA4);
                case 3:
                    message->data[3] = ((CAN1->sFIFOMailBox[0].RDLR & CAN_RDL0R_DATA3) >> 24);
```

```
                case 2:
                    message->data[2] = ((CAN1->sFIFOMailBox[0].RDLR & CAN_RDL0R_DATA2) >> 16);
                case 1:
                    message->data[1] = ((CAN1->sFIFOMailBox[0].RDLR & CAN_RDL0R_DATA1) >> 8);
                case 0:
                    message->data[0] = (CAN1->sFIFOMailBox[0].RDLR & CAN_RDL0R_DATA0);
            }
        }
        CAN1->RF0R |= CAN_RF0R_RFOM0;  // Mailbox release
    }
    //USART_putchar('a');
}

void CAN_message_write(CAN_message *message, unsigned char mailbox)
{
    CAN1->sTxMailBox[mailbox].TDTR = 0;
    CAN1->sTxMailBox[mailbox].TIR = 0;
    CAN1->sTxMailBox[mailbox].TDHR = 0;
    CAN1->sTxMailBox[mailbox].TDLR = 0;                                          // Reset Transmit
registers

    CAN1->sTxMailBox[mailbox].TIR |= ((message->id << 21) & CAN_TI0R_STID);     // Standard
identifier input
    CAN1->sTxMailBox[mailbox].TIR &= ~(CAN_TI0R_IDE);                            // Standard
identifier set
    CAN1->sTxMailBox[mailbox].TIR |= ((message->type << 1) & CAN_TI0R_RTR);     // Message type set


    CAN1->sTxMailBox[mailbox].TDTR |= (message->len & CAN_TDT0R_DLC);         // Set data length

    switch (message->len-1)                                                     // Insert Data
    {
        case 7:
            CAN1->sTxMailBox[mailbox].TDHR |= (message->data[7] << 24);
        case 6:
            CAN1->sTxMailBox[mailbox].TDHR |= (message->data[6] << 16);
        case 5:
            CAN1->sTxMailBox[mailbox].TDHR |= (message->data[5] << 8);
        case 4:
            CAN1->sTxMailBox[mailbox].TDHR |= message->data[4];
        case 3:
            CAN1->sTxMailBox[mailbox].TDLR |= (message->data[3] << 24);
        case 2:
            CAN1->sTxMailBox[mailbox].TDLR |= (message->data[2] << 16);
        case 1:
            CAN1->sTxMailBox[mailbox].TDLR |= (message->data[1] << 8);
        case 0:
            CAN1->sTxMailBox[mailbox].TDLR |= message->data[0];
    }
}

void USB_LP_CAN1_RX0_IRQHandler(void)
{
    GPIOB->BSRR |= GPIO_BSRR_BR5;
    CAN_message_read(&CAN_Rx);
}

void USB_HP_CAN1_TX_IRQHandler(void)
{
    GPIOB->BSRR |= GPIO_BSRR_BR2;
    if (CAN1->TSR & CAN_TSR_RQCP0)      // Interrupt occures when transmit mailbox 0 gets empty
    {
        CAN1->TSR &= ~(CAN_TSR_RQCP0);
        CAN_Tx_0=1;
        CAN1->IER &= ~CAN_IER_TMEIE;
    }
```

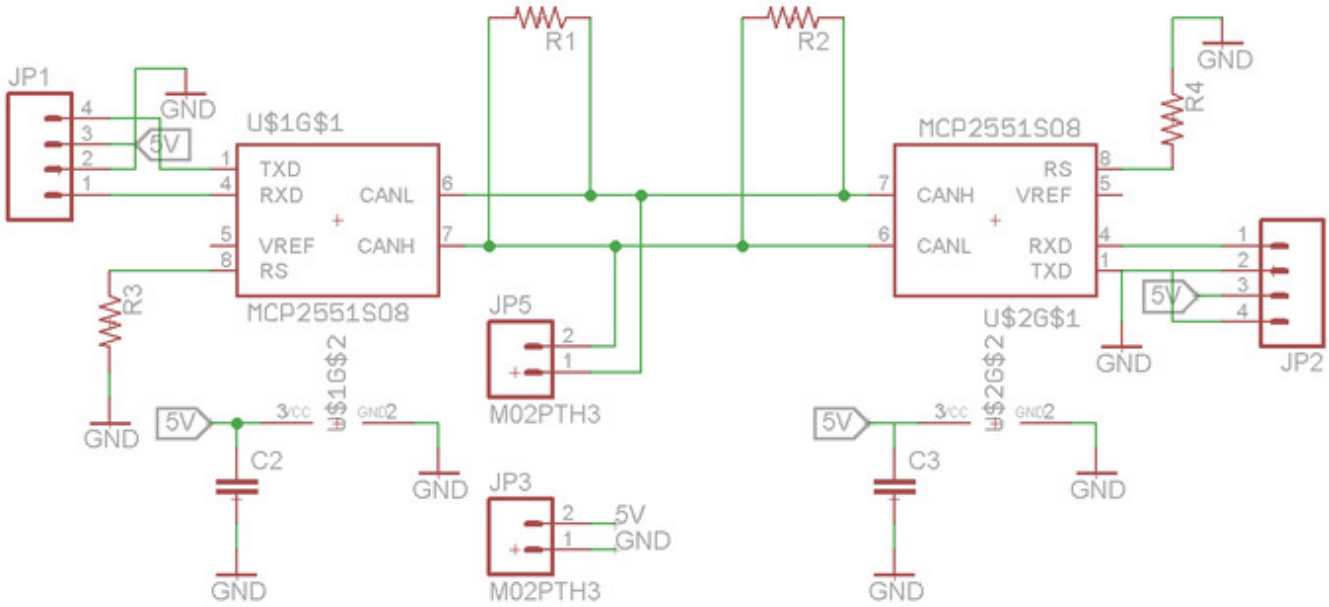
```
    if (CAN1->TSR & CAN_TSR_RQCP1)      // Interrupt occures when transmit mailbox 1 gets empty
    {
        CAN1->TSR &= ~(CAN_TSR_RQCP1);
        CAN_Tx_1=1;
        CAN1->IER &= ~CAN_IER_TMEIE;
    }
    if (CAN1->TSR & CAN_TSR_RQCP2)      // Interrupt occures when transmit mailbox 2 gets empty
    {
        CAN1->TSR &= ~(CAN_TSR_RQCP2);
        CAN_Tx_2=1;
        CAN1->IER &= ~CAN_IER_TMEIE;
    }
}
```

MAIN.C

```
#include "stm32f10x.h"
#include "CAN_init.h"

extern CAN_message CAN_Rx, CAN_Tx;
//volatile unsigned char CAN_Tx_0=1, CAN_Tx_1=1, CAN_Tx_2=1;

void LED_setup(void);
void USART_setup(void);
void USART_send(char * text);
void USART_putchar(char ch);

void delay(unsigned int nCount) {

  for(; nCount != 1; nCount--);
}

int main ()
{
    LED_setup();
    USART_setup();
    CAN_setup();

//  CAN_Tx.id = 10;
//  CAN_Tx.len = 8;
//  CAN_Tx.type = DATA;
//  CAN_Tx.data[0] = 'a';
//  CAN_Tx.data[1] = 'a';
//  CAN_Tx.data[2] = 'a';
//  CAN_Tx.data[3] = 'a';
//  CAN_Tx.data[4] = 'a';
//  CAN_Tx.data[5] = 'a';
//  CAN_Tx.data[6] = 'a';
//  CAN_Tx.data[7] = 'a';

    for(;;)
    {
//      CAN_send();
//      GPIOB->ODR &= ~GPIO_ODR_ODR5;
//      delay(100000);
//      GPIOB->ODR |= GPIO_ODR_ODR5;
//      delay(100000);
    }
}
```

The main file is written to wait for msg- he's receiver. The transmitter STM is without slashes in main file. Both microcontrollers are STM32F1. And both behave the same when I switch the roles.

And here is my connection:


JP1
GND
5V
U$1G$1
TXD
RXD
CANL
VREF
CANH
RS
MCP2551S08
R1
R2
R3
R4
JP5
M02PTH3
JP3
M02PTH3
5V
GND
C2
C3
MCP2551S08
U$2G$1
JP2