Programación del CAN BUS

- Introducción al CAN BUS
- CAN BUS en STM32Fxx
- Configuración y programación de CAN BUS en STM32F407
- Programación de una tarea que transmite por el bus
- Programación de la interrupción para recibir el dato
- Envío de mensajes con distintos IDs desde un mismo nodo. Máscaras para recepción

Conexiones

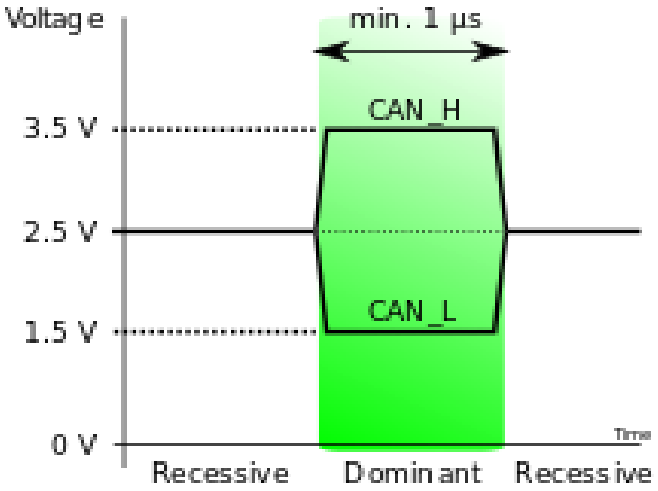
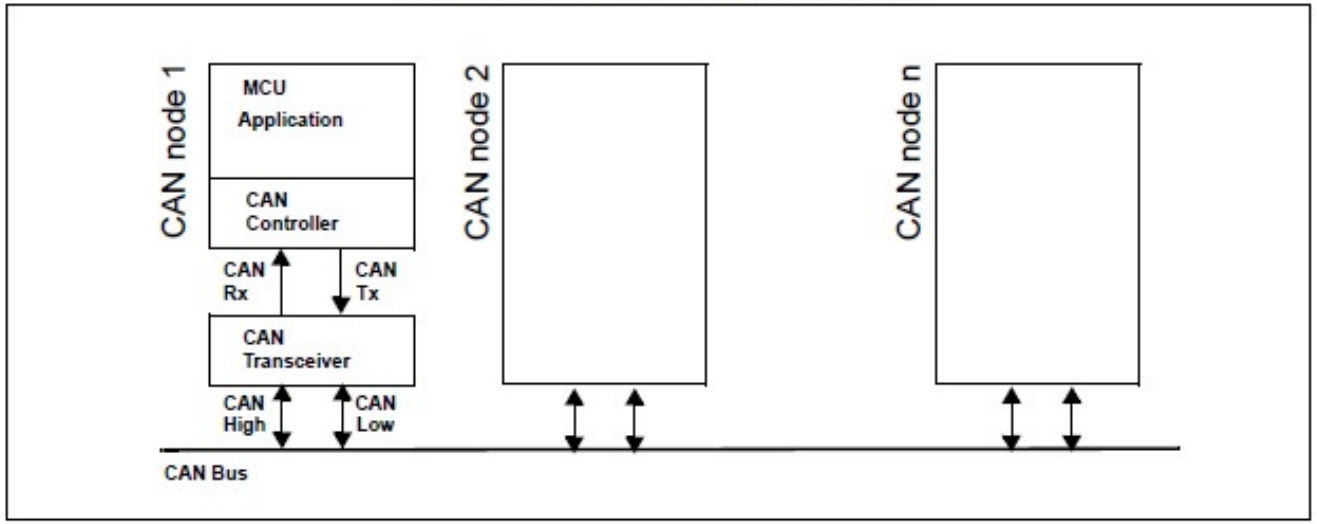
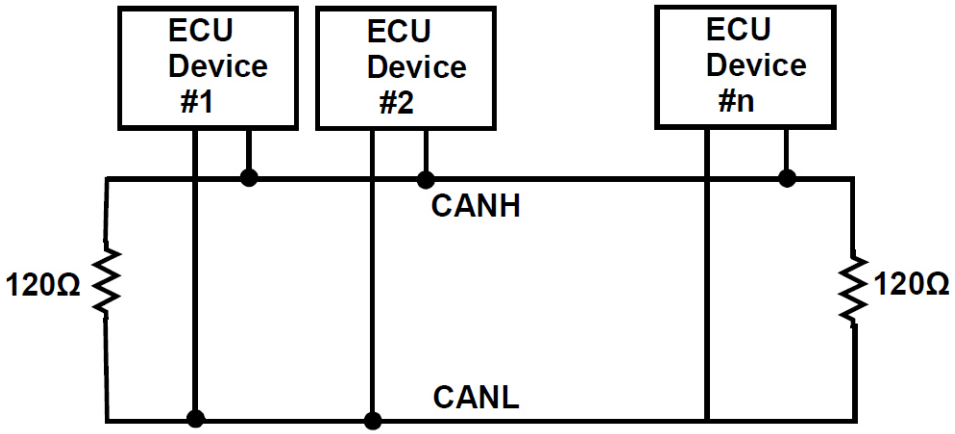
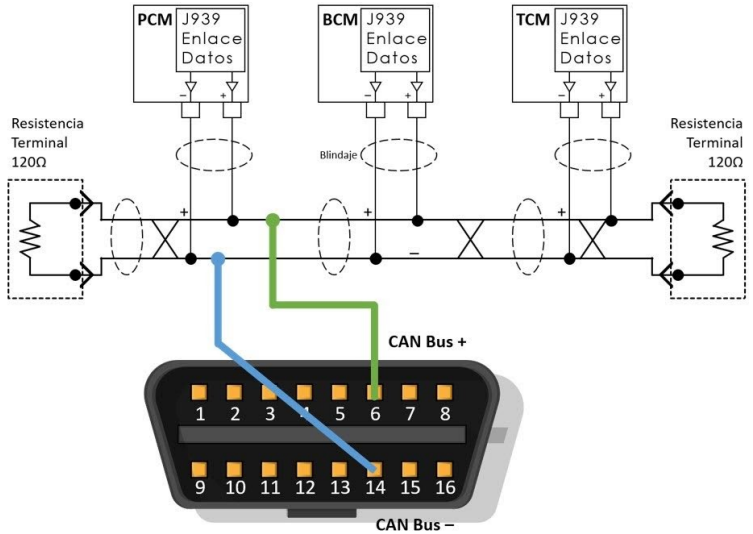


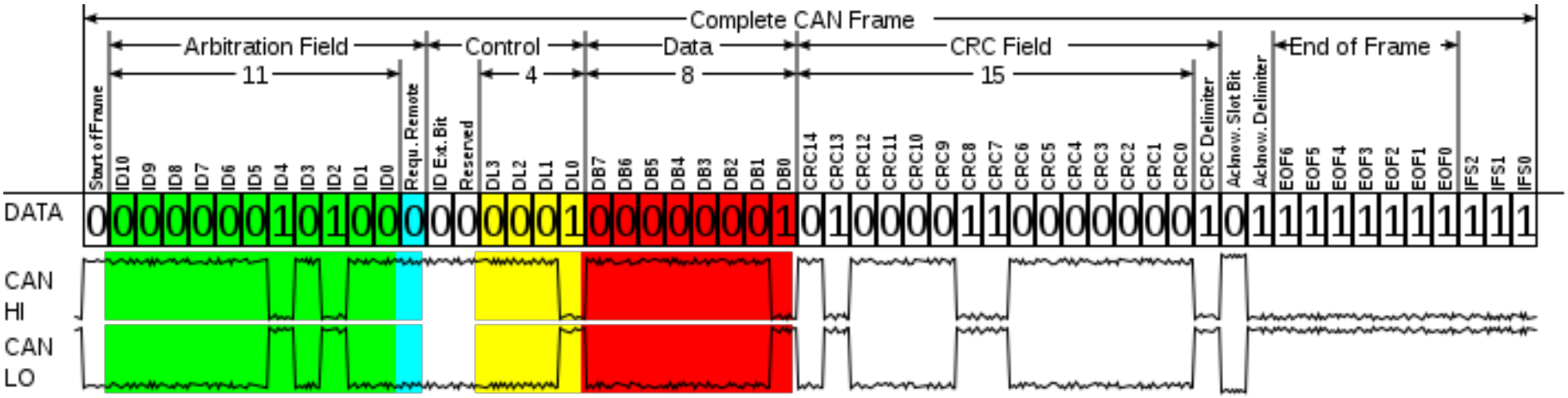
Figure 334. CAN network topology

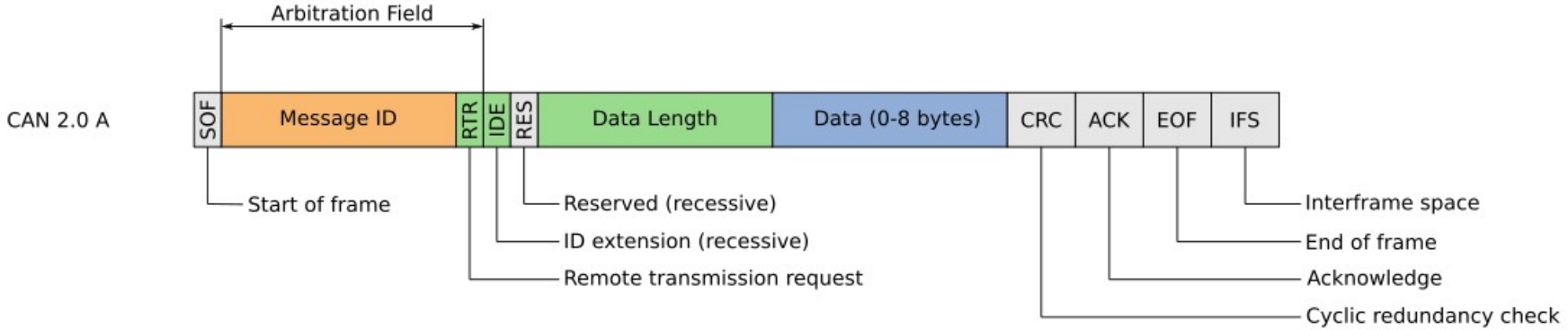


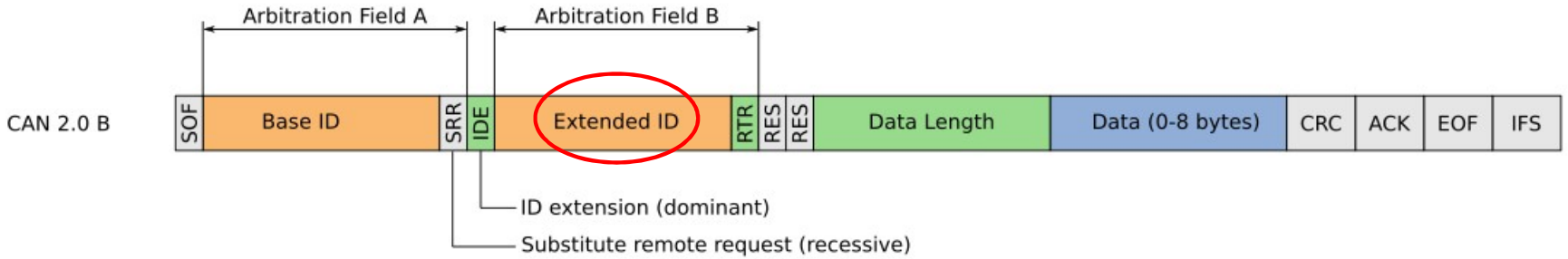




Trama de mensaje

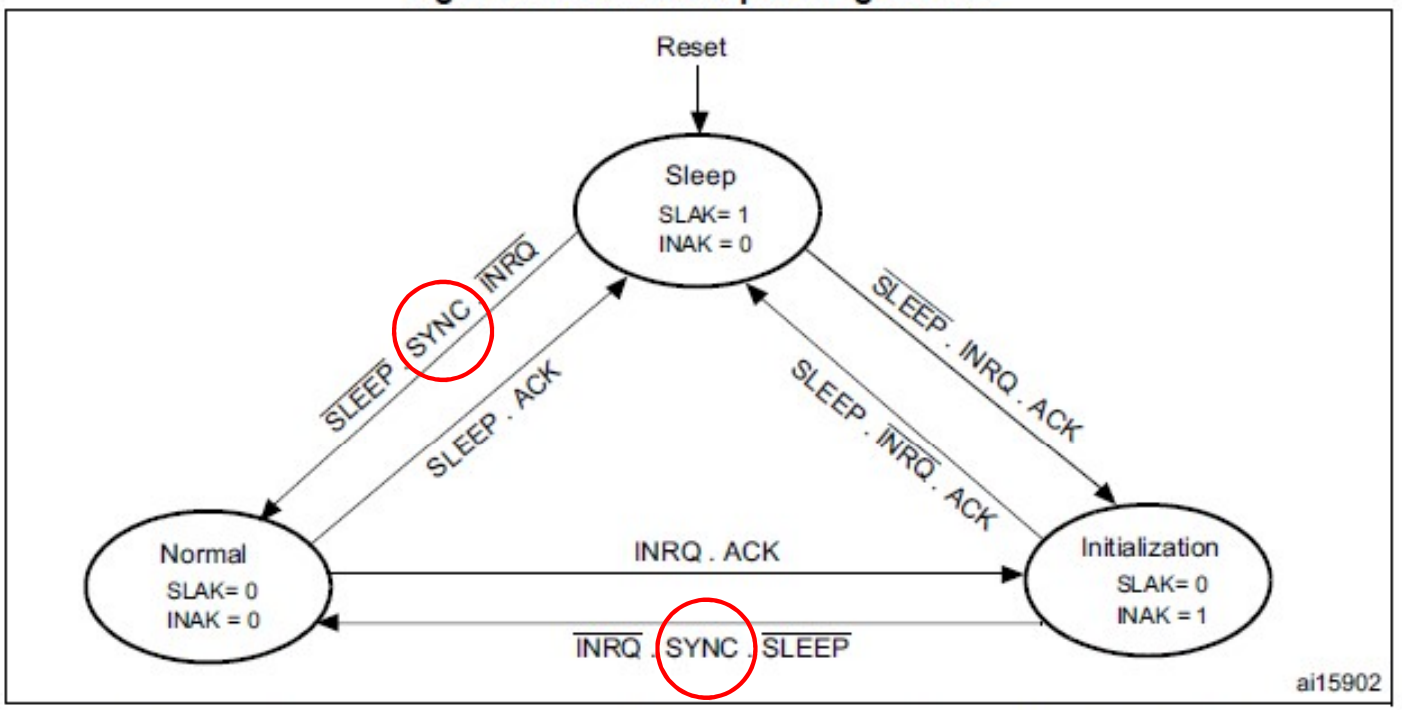






Modos de funcionamiento

Figure 336. bxCAN operating modes

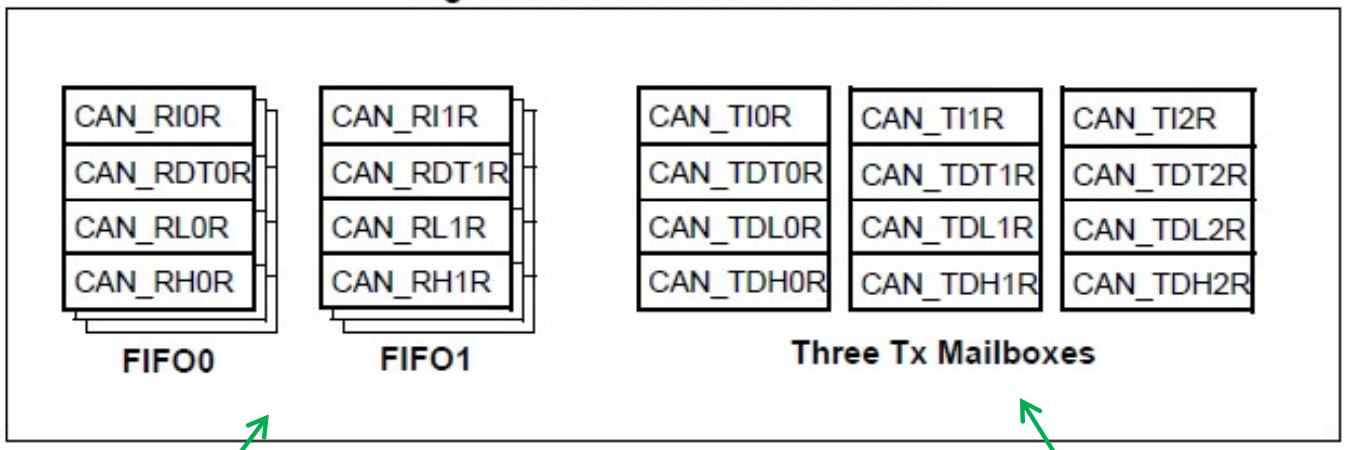


1. ACK = The wait state during which hardware confirms a request by setting the INAK or SLAK bits in the CAN_MSR register
2. SYNC = The state during which bxCAN waits until the CAN bus is idle, meaning 11 consecutive recessive bits have been monitored on CANRX

Buzones (mailboxes)

- There are three TX Mailboxes and two RX Mailboxes, as shown in *Figure 349.*
- *Each RX* Mailbox allows access to a 3-level depth FIFO, the access being offered only to the oldest received message in the FIFO.
- Each mailbox consist of four registers.

Figure 349. RX and TX mailboxes



Indicamos de qué *receive mailbox* leemos

Podemos elegir el orden en que se envían los mensajes de las *transmit mailboxes*:

- FIFO

- Por prioridades

Afecta al tiempo máximo de transmisión de un mensaje

Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.

To fulfill this requirement, the bxCAN Controller provides 28 configurable and scalable filter banks (27-0) to the application. This hardware filtering saves CPU resources which would be otherwise needed to perform filtering by software. Each filter bank x consists of two 32-bit registers, CAN_FxR0 and CAN_FxR1.

Scalable width

To optimize and adapt the filters to the application needs, each filter bank can be scaled independently. Depending on the filter scale a filter bank provides:

- One 32-bit filter for the STDID[10:0], EXTID[17:0], IDE and RTR bits.
- Two 16-bit filters for the STDID[10:0], RTR, IDE and EXTID[17:15] bits.

Furthermore, the filters can be configured in mask mode or in identifier list mode.

Mask mode

In **mask mode the identifier registers are associated with mask registers specifying which** bits of the identifier are handled as “must match” or as “don’t care”.

Identifier list mode

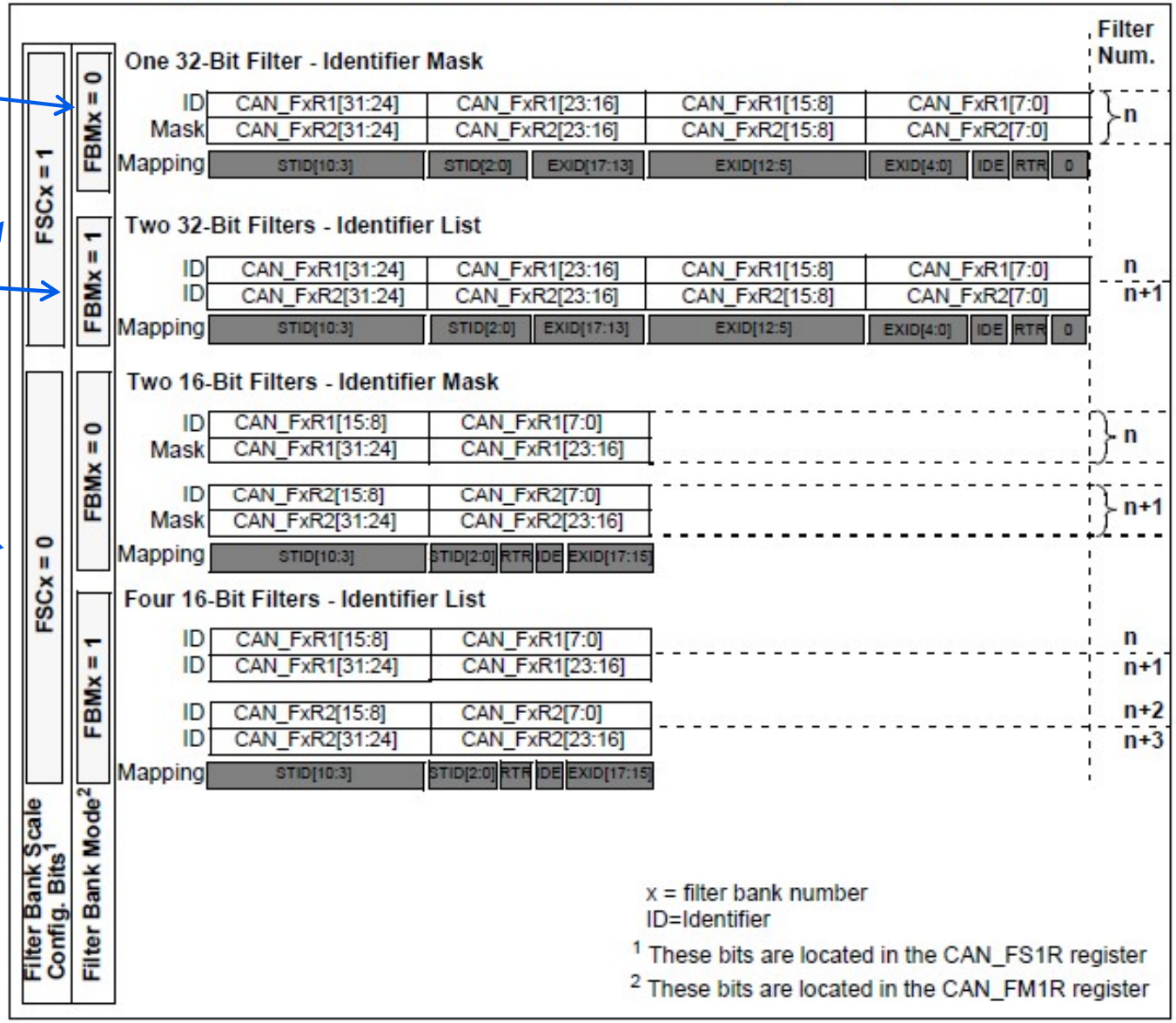
In **identifier list mode, the mask registers are used as identifier registers. Thus instead of** defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.

Mask mode

Identifier list mode

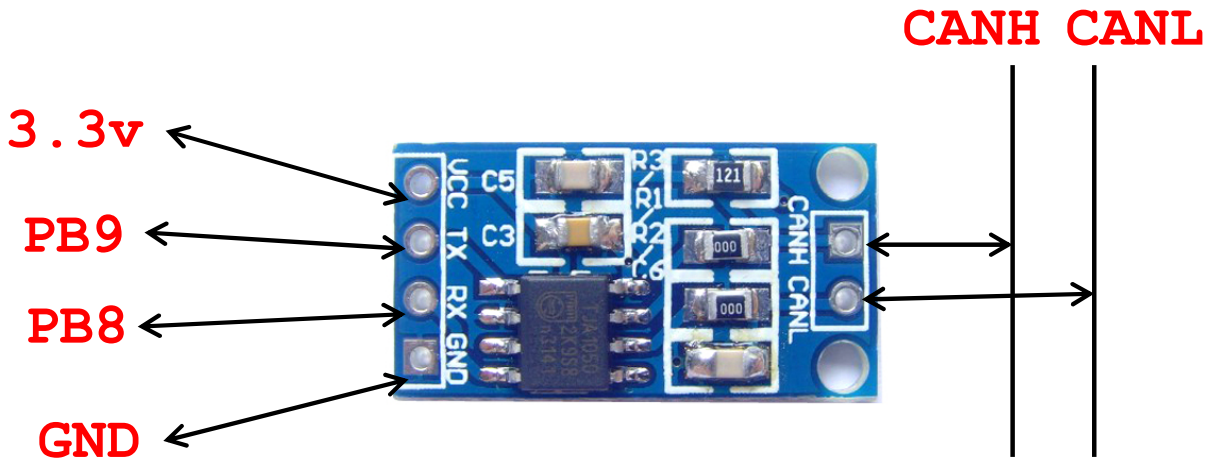
Scalable width

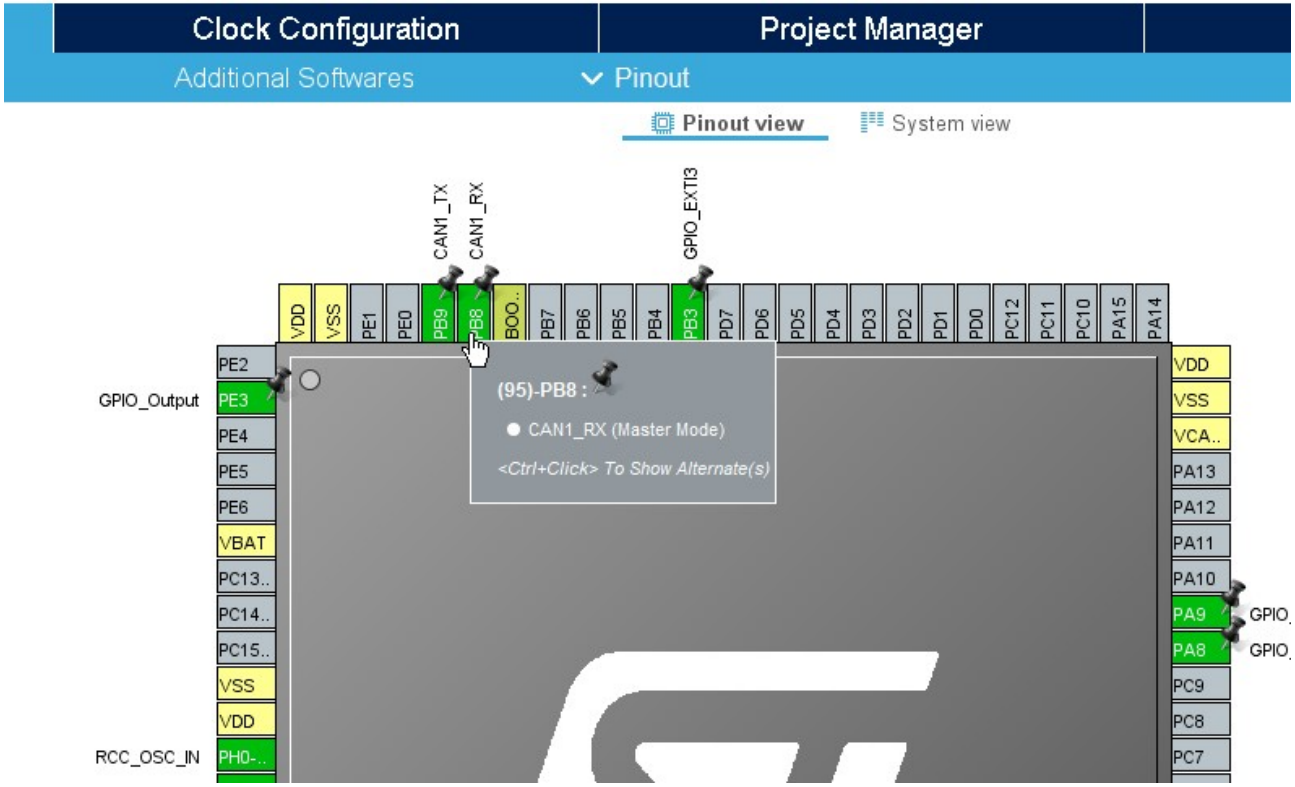
Figure 342. Filter bank scale configuration - register organization



CAN BUS en STM32Fxx

Conexiones





Librería para programación de CAN BUS:

stm32f4xx_hal_can.h

C:\PruebasSTM\Pinicio**Drivers\STM32F4xx_HAL_Driver\Inc**\

stm32f4xx_hal_can.c

C:\PruebasSTM\Pinicio**Drivers\STM32F4xx_HAL_Driver\Src**\

Algunos ejemplos en youtube:

https://www.youtube.com/watch?v=ymD3F0h-iIE&t=906s (en inglés)

https://www.youtube.com/watch?v=jdakCAkZyTs (en francés)

Fichero main.c

```
/* Declaraciones para tranmisiones en CAN BUS -------------------------*/

CAN_HandleTypeDef hcan1;

//declare a specific header for message transmittions
CAN_TxHeaderTypeDef pHeader;

//declare header for message reception
CAN_RxHeaderTypeDef pRxHeader;
uint32_t TxMailbox;

//declare byte to be transmitted
uint8_t ByteSent = 0;

//declare a receive byte
uint8_t ByteReceived = 0;

//declare CAN filter structure
CAN_FilterTypeDef sFilterConfig;
```

```
/* Funcion para inicializar la transmision */

void InicializaTransmisionesCAN(void);

int main(void)
{ …
  MX_CAN1_Init();  // incluida por MXCube

  InicializaTransmisionesCAN();  // llamamos a la función
  …
}
```

Configuración y programación de CAN BUS en STM32F407

```
void InicializaTransmisionesCAN() {

pHeader.DLC=1; //give message size of 1 byte
pHeader.IDE=CAN_ID_STD; //set identifier to standard
pHeader.RTR=CAN_RTR_DATA; //set data type to remote transmission request
//define a standard identifier, used for message identification by filters
pHeader.StdId=0x2FF; //(##switch this for the other microcontroller##)

//filter one (stack light blink)
sFilterConfig.FilterFIFOAssignment=CAN_FILTER_FIFO0; //set fifo assignment

//the ID that the filter looks for
sFilterConfig.FilterIdHigh=0x2F4<<5; //(##switch this for the other microcontroller##)
sFilterConfig.FilterIdLow=0;

sFilterConfig.FilterMaskIdHigh=0;
sFilterConfig.FilterMaskIdLow=0;

sFilterConfig.FilterScale=CAN_FILTERSCALE_32BIT; //set filter scale
sFilterConfig.FilterActivation=ENABLE;

HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig); //configure CAN filter

HAL_CAN_Start(&hcan1); //start CAN
HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING); //enable interrupts
};
```

CAN TX mailbox identifier register (CAN_TIxR) (x=0..2)

Address offsets: 0x180, 0x190, 0x1A0
Reset value: 0xXXXX XXXX (except bit 0, TXRQ = 0)

All TX registers are write protected when the mailbox is pending transmission (TMEx reset).

This register also implements the TX request control (bit 0) - reset value 0.

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

Bits 31:21 **STID[10:0]/EXID[28:18]**: Standard identifier or extended identifier
The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bits 20:3 **EXID[17:0]**: Extended identifier
The LSBs of the extended identifier.

Transmisión de un dato desde una tarea

```
void Tarea2 (void const * argument){

  TickType_t lastWakeTime;
  lastWakeTime = xTaskGetTickCount();
  …

  for(;;){
    …
    ByteSent = xxxx;
    HAL_CAN_AddTxMessage(&hcan1, &pHeader, &ByteSent, &TxMailbox);
    …
    vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(T_TAREA2));
  {
}
```

Recepción de un dato en el manejador de interrupción

Fichero stm32fxx_it.c

```
/* External variables ---------------------------------------------*/
extern CAN_HandleTypeDef hcan1;
extern TIM_HandleTypeDef htim1;

// Variables externas para transmisiones en CAN BUS
extern CAN_TxHeaderTypeDef pHeader;
extern CAN_RxHeaderTypeDef pRxHeader;
extern uint32_t TxMailbox;
extern uint8_t ByteSent, ByteReceived;


void CAN1_RX0_IRQHandler(void)
{
  HAL_CAN_IRQHandler(&hcan1);
  /* USER CODE BEGIN CAN1_RX0_IRQn 1 */
  HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &pRxHeader, &ByteReceived);
  /* USER CODE END CAN1_RX0_IRQn 1 */
}
```

Ejercicio para el filtrado de varios identificadores

Configurar el en Nodo 2 el modo IDMASK en la estructura sFilterConfig. Para ello consultar las estructuras de datos en las librerias:

```
stm32f4xx_hal_can.h //C:\PruebasSTM\Pinicio\Drivers\STM32F4xx_HAL_Driver\Inc\
stm32f4xx_hal_can.c //C:\PruebasSTM\Pinicio\Drivers\STM32F4xx_HAL_Driver\Src\
```

Dar el valor apropiado a la máscara para filtrar las direcciones deseadas, teniendo en cuenta las especificaciones de la máscara

Mask

Each bit of the register specifies whether the bit of the associated identifier register must match with the corresponding bit of the expected identifier or not.

0: Don't care, the bit is not used for the comparison

1: Must match, the bit of the incoming identifier must have the same level has specified in the corresponding identifier register of the filter.

Probar las transmisiones desde el Nodo 1 con diferentes Identificadores, unos que cuplan la máscara y otros que no la cumplan

Envío de mensajes con distintos IDs desde un mismo nodo. Máscaras para recepción

Nodo de Envío: configuramos dos cabeceras con distintos IDs

```
CAN_TxHeaderTypeDef pHeader, pHeader2; //declare a specific header transmittions

void InicializaTransmisionesCAN() {
        pHeader.DLC=1; //give message size of 1 byte
        pHeader.IDE=CAN_ID_STD; //set identifier to standard
        pHeader.RTR=CAN_RTR_DATA; //set data type to remote transmission request
        pHeader.StdId=0x2FA; //define a standard identifier,

        pHeader2.DLC=1; //give message size of 1 byte
        pHeader2.IDE=CAN_ID_STD; //set identifier to standard
        pHeader2.RTR=CAN_RTR_DATA; //set data type to remote transmission request
        pHeader2.StdId=0x2FB; //define a standard identifier

        sFilterConfig.FilterFIFOAssignment=CAN_FILTER_FIFO0; //set fifo assignment
        sFilterConfig.FilterIdHigh=0x2F4<<5; //the ID that the filter looks for
        sFilterConfig.FilterIdLow=0;
        sFilterConfig.FilterMaskIdHigh=0;
        sFilterConfig.FilterMaskIdLow=0;
        sFilterConfig.FilterScale=CAN_FILTERSCALE_32BIT; //set filter scale
        sFilterConfig.FilterActivation=ENABLE;

        HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig); //configure CAN filter

        HAL_CAN_Start(&hcan1); //start CAN
        HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
};
```

Envío de mensajes con distintos IDs desde un mismo nodo. Máscaras para recepción

Nodo de Envío: Cada tarea realiza el envío con una cabecera distinta

```
void Tarea1(void const * argument) {
...
  for(;;) {
    ...
    ByteSent = XXX;
    HAL_CAN_AddTxMessage(&hcan1, &pHeader, &ByteSent, &TxMailbox);  // Envio dato
    ...
  vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(T_TAREA1));
  }
}

void Tarea2 (void const * argument){
...
  for(;;){
    ...
    ByteSent2 = YYY;
    HAL_CAN_AddTxMessage(&hcan1, &pHeader2, &ByteSent2, &TxMailbox);  // Envio dato
    ...
  vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(T_TAREA2));
  }
}
```

Envío de mensajes con distintos IDs desde un mismo nodo. Máscaras para recepción

Nodo de Recepción: Programamos la máscara

```
// FICHERO main.c  en el Nodo de Recepción

void InicializaTransmisionesCAN() {
  pHeader.DLC=1; //give message size of 1 byte
  pHeader.IDE=CAN_ID_STD; //set identifier to standard
  pHeader.RTR=CAN_RTR_DATA; //set data type to remote transmission request?
  pHeader.StdId=0x2F4; //define a standard identifier,

  sFilterConfig.FilterFIFOAssignment=CAN_FILTER_FIFO0; //set fifo assignment
  sFilterConfig.FilterIdHigh=0x2FF<<5; //the ID:10 1111 1111 that the filter looks for
  sFilterConfig.FilterIdLow=0;
  sFilterConfig.FilterMaskIdHigh=0x3F0<<5;  //Mask: 11 1111 0000
  sFilterConfig.FilterMaskIdLow=0;
  sFilterConfig.FilterScale=CAN_FILTERSCALE_32BIT; //set filter scale

  sFilterConfig.FilterMode=CAN_FILTERMODE_IDMASK;  //set identifier mask mode

  sFilterConfig.FilterActivation=ENABLE;

  HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig); //configure CAN filter
  HAL_CAN_Start(&hcan1); //start CAN
  HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING); //enable interr
};
```

Nodo de Recepción: Consultamos el ID de la cabecera

```
// FICHERO stm32f4xx_it.c  en el Nodo de Recepción

extern uint8_t ByteReceived, ByteReceived1, ByteReceived2;

void CAN1_RX0_IRQHandler(void)
{
  HAL_CAN_IRQHandler(&hcan1);

  HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &pRxHeader, &ByteReceived);
  if (pRxHeader.StdId == 0x2FA) ByteReceived1 = ByteReceived;
  if (pRxHeader.StdId == 0x2FB) ByteReceived2 = ByteReceived;
}
```

Envío de mensajes con distintos IDs desde un mismo nodo. Máscaras para recepción

Las estructuras de datos

```
CAN_HandleTypeDef     hcan1;
CAN_TxHeaderTypeDef   pHeader,  pHeader2;
CAN_RxHeaderTypeDef   pRxHeader;
```

y las funciones

```
HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig);
HAL_CAN_Start(&hcan1);
HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
HAL_CAN_AddTxMessage (&hcan1, &pHeader2, &ByteSent2, &TxMailbox);
```

se consultan en las librerías:

```
stm32f4xx_hal_can.h //C:\PruebasSTM\Pinicio\Drivers\STM32F4xx_HAL_Driver\Inc\
stm32f4xx_hal_can.c //C:\PruebasSTM\Pinicio\Drivers\STM32F4xx_HAL_Driver\Src\
```