CAN Controller

CASE STUDY 1

By: Abdelrahman Matarawy

Overview of CAN Controller:

Tx mailboxes:

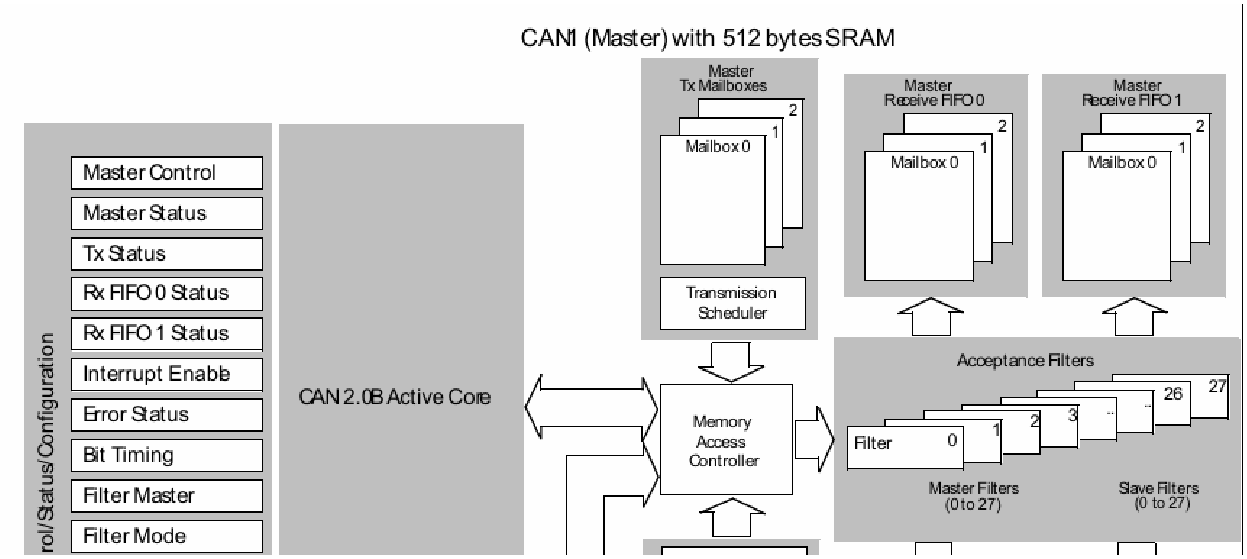
- Three transmit mailboxes are provided to the software for setting up messages. The transmission Scheduler decides which mailbox has to be transmitted first.

Acceptance filters:

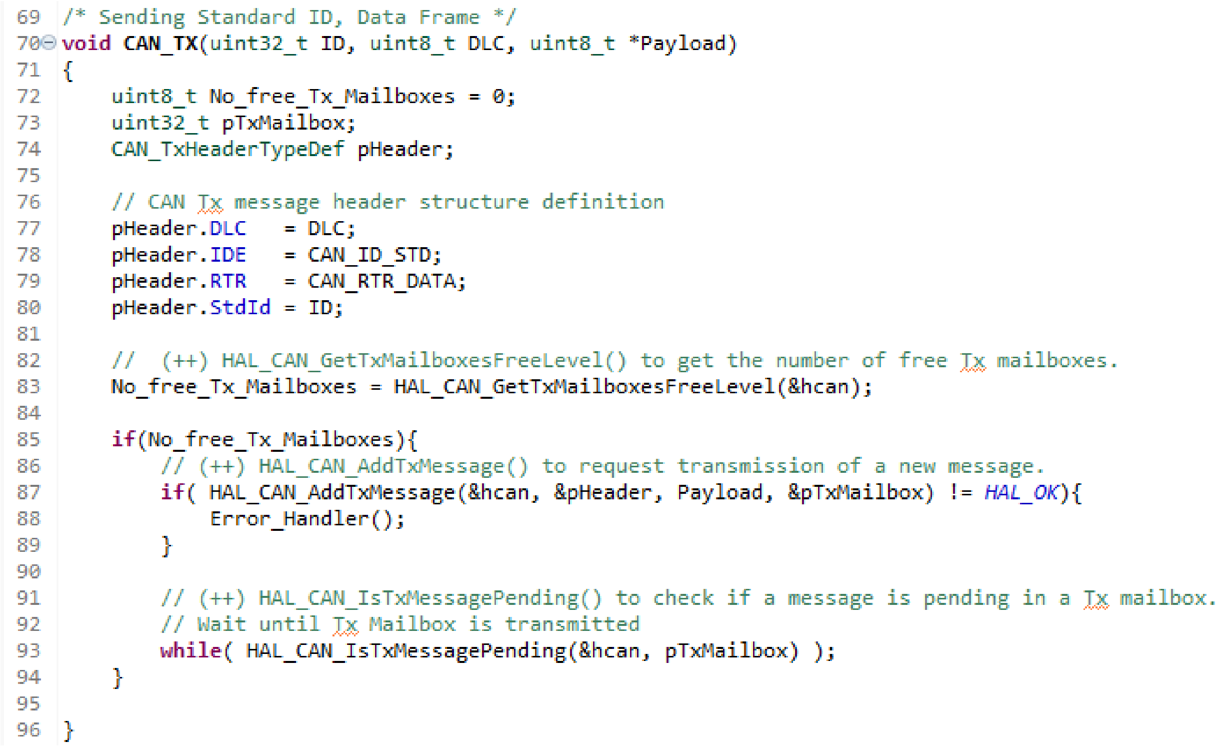
- The bxCAN provides 14 scalable/configurable identifier filter banks for selecting the incoming messages the software needs and discarding the others.

Receive FIFO:

- Two receive FIFOs are used by hardware to store the incoming messages. Three complete messages can be stored in each FIFO. The FIFOs are managed completely by hardware.

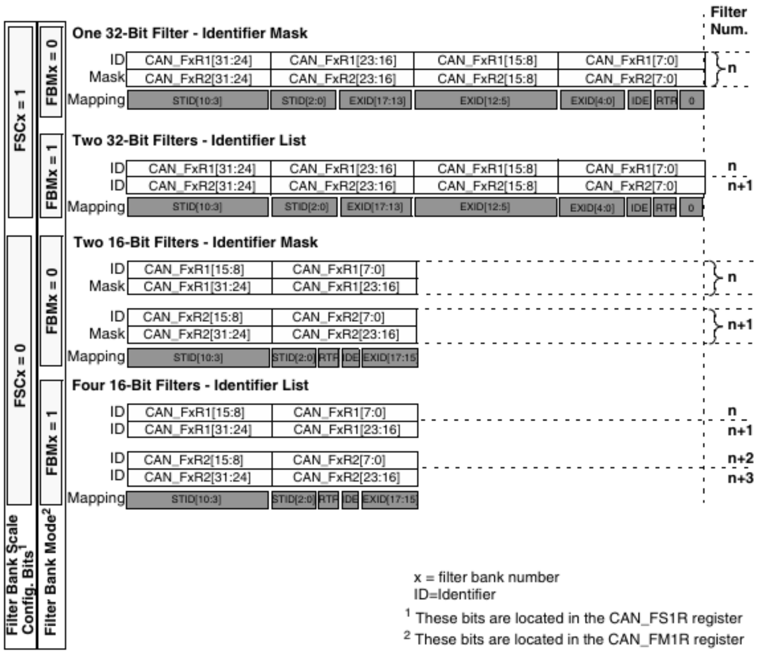


CAN Transmitter Code:



```c
/* Sending Standard ID, Data Frame */
void CAN_TX(uint32_t ID, uint8_t DLC, uint8_t *Payload)
{
    uint8_t No_free_Tx_Mailboxes = 0;
    uint32_t pTxMailbox;
    CAN_TxHeaderTypeDef pHeader;

    // CAN Tx message header structure definition
    pHeader.DLC   = DLC;
    pHeader.IDE   = CAN_ID_STD;
    pHeader.RTR   = CAN_RTR_DATA;
    pHeader.StdId = ID;

    //  (++) HAL_CAN_GetTxMailboxesFreeLevel() to get the number of free Tx mailboxes.
    No_free_Tx_Mailboxes = HAL_CAN_GetTxMailboxesFreeLevel(&hcan);

    if(No_free_Tx_Mailboxes){
        // (++) HAL_CAN_AddTxMessage() to request transmission of a new message.
        if( HAL_CAN_AddTxMessage(&hcan, &pHeader, Payload, &pTxMailbox) != HAL_OK){
            Error_Handler();
        }

        // (++) HAL_CAN_IsTxMessagePending() to check if a message is pending in a Tx mailbox.
        // Wait until Tx Mailbox is transmitted
        while( HAL_CAN_IsTxMessagePending(&hcan, pTxMailbox) );
    }

}
```

Receiver Filter:

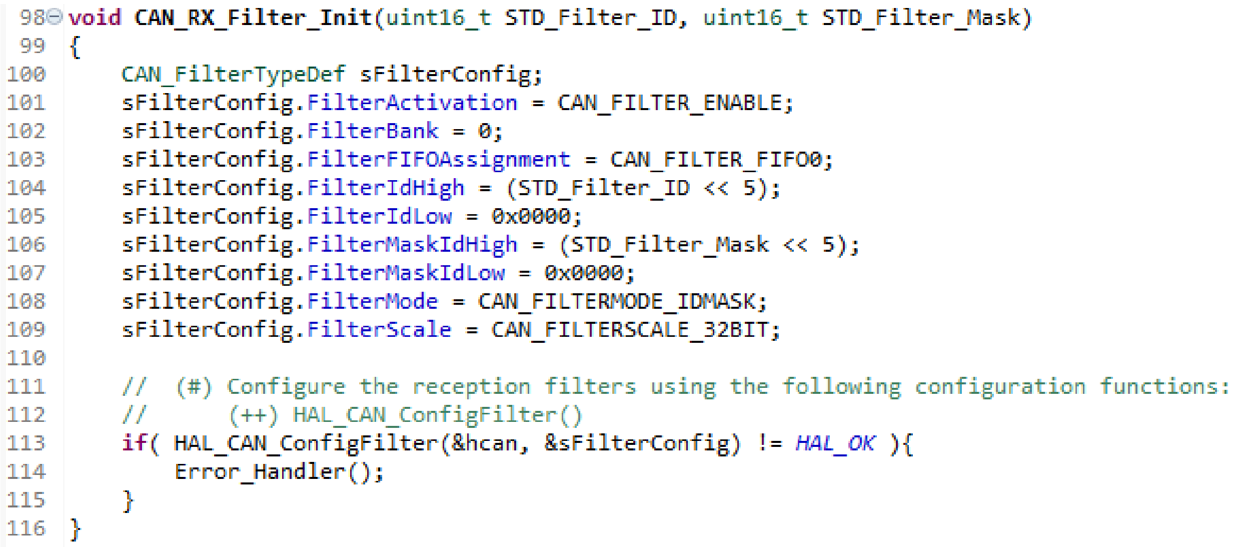


x = filter bank number
ID=Identifier

[1] These bits are located in the CAN_FS1R register

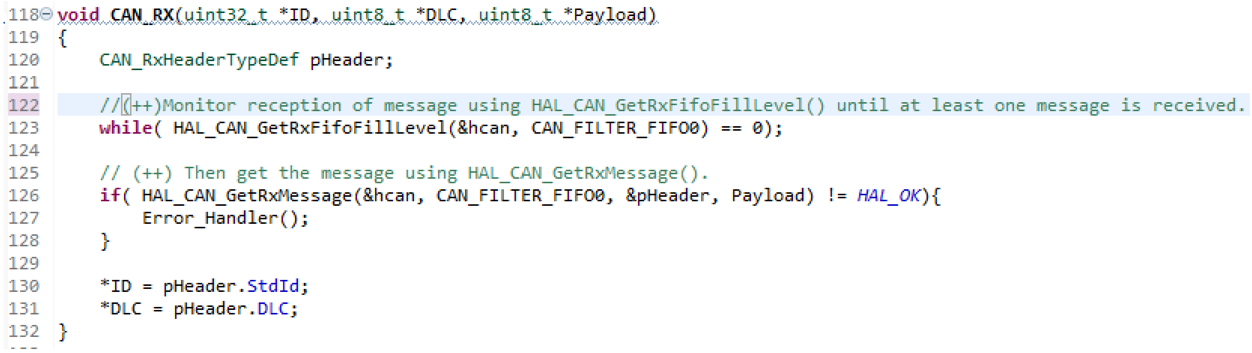
[2] These bits are located in the CAN_FM1R register

- I work For ID Mask as 1st Register Work for ID, while 2nd Work for Mask.
    - On Mask Register if I define bit as 1 so CAN Controller will Compare ID Register with ID Received from Tx.
- Also work on Register as 32 Bits.
- **Code:**



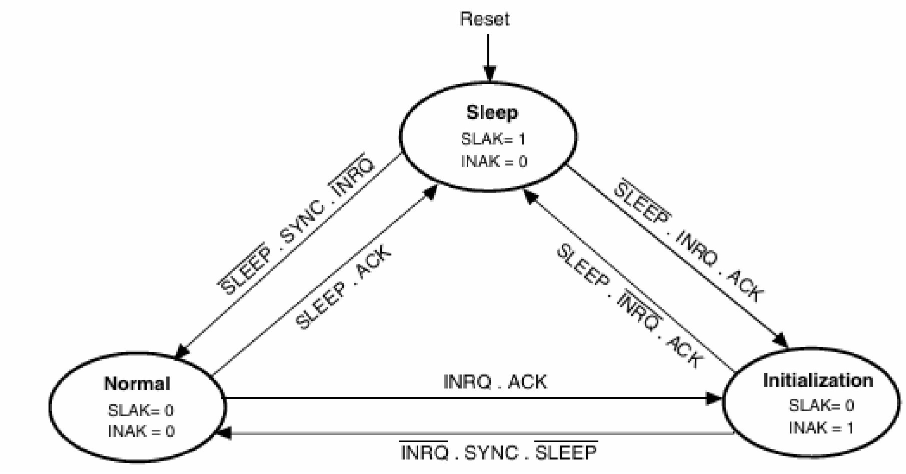
```c
void CAN_RX_Filter_Init(uint16_t STD_Filter_ID, uint16_t STD_Filter_Mask)
{
    CAN_FilterTypeDef sFilterConfig;
    sFilterConfig.FilterActivation = CAN_FILTER_ENABLE;
    sFilterConfig.FilterBank = 0;
    sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
    sFilterConfig.FilterIdHigh = (STD_Filter_ID << 5);
    sFilterConfig.FilterIdLow = 0x0000;
    sFilterConfig.FilterMaskIdHigh = (STD_Filter_Mask << 5);
    sFilterConfig.FilterMaskIdLow = 0x0000;
    sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
    sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;

    //  (#) Configure the reception filters using the following configuration functions:
    //      (++) HAL_CAN_ConfigFilter()
    if( HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK ){
        Error_Handler();
    }
}
```

CAN Receiver Code:

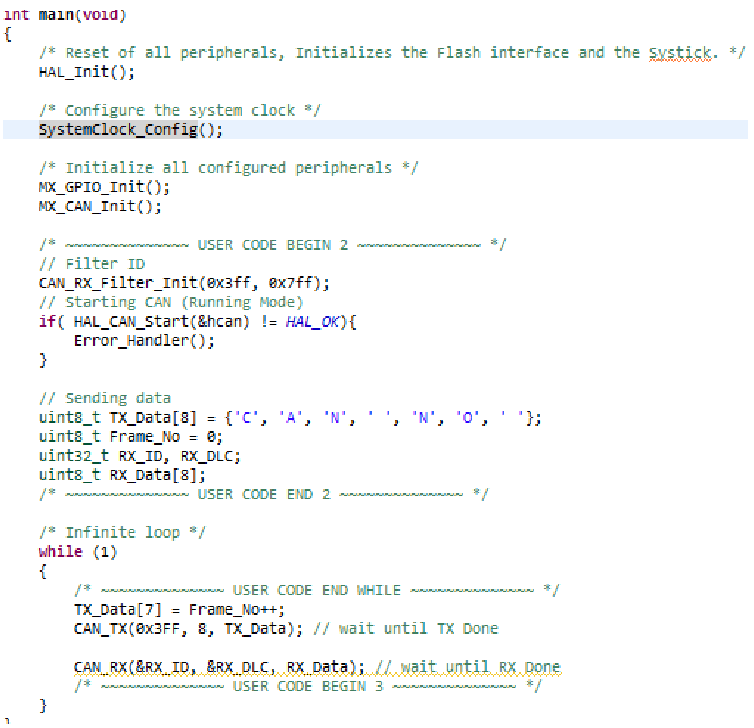


```c
void CAN_RX(uint32_t *ID, uint8_t *DLC, uint8_t *Payload)
{
    CAN_RxHeaderTypeDef pHeader;

    //(++)Monitor reception of message using HAL_CAN_GetRxFifoFillLevel() until at least one message is received.
    while( HAL_CAN_GetRxFifoFillLevel(&hcan, CAN_FILTER_FIFO0) == 0);

    // (++) Then get the message using HAL_CAN_GetRxMessage().
    if( HAL_CAN_GetRxMessage(&hcan, CAN_FILTER_FIFO0, &pHeader, Payload) != HAL_OK){
        Error_Handler();
    }

    *ID = pHeader.StdId;
    *DLC = pHeader.DLC;
}
```

Main Code:

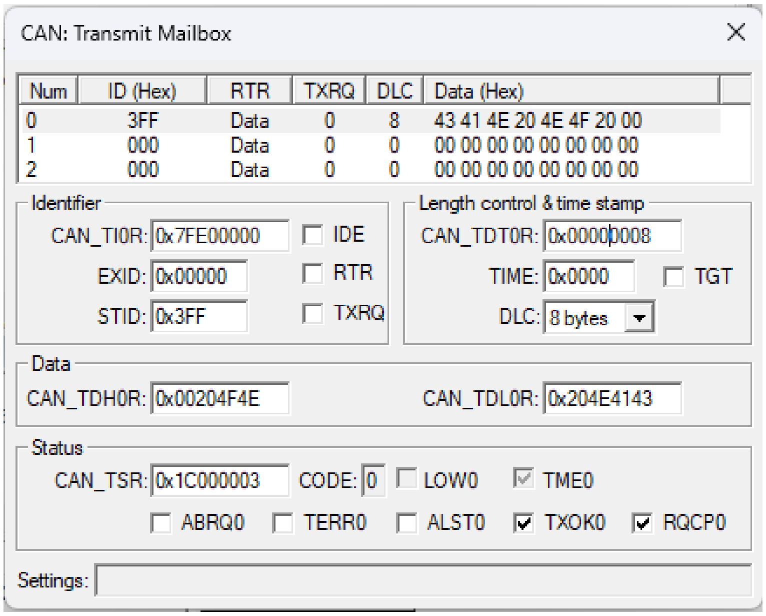


- 1st We Switch to Init Mode to define how CAN Controller Work.
- 2nd We Switch to Normal mode to start Running Tx and Rx Transmission Process.
- **Code:**



```c
int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_CAN_Init();

    /* ~~~~~~~~~~~~~~~ USER CODE BEGIN 2 ~~~~~~~~~~~~~~~ */
    // Filter ID
    CAN_RX_Filter_Init(0x3ff, 0x7ff);
    // Starting CAN (Running Mode)
    if( HAL_CAN_Start(&hcan) != HAL_OK){
        Error_Handler();
    }

    // Sending data
    uint8_t TX_Data[8] = {'C', 'A', 'N', ' ', 'N', 'O', ' '};
    uint8_t Frame_No = 0;
    uint32_t RX_ID, RX_DLC;
    uint8_t RX_Data[8];
    /* ~~~~~~~~~~~~~~~ USER CODE END 2 ~~~~~~~~~~~~~~~ */

    /* Infinite loop */
    while (1)
    {
        /* ~~~~~~~~~~~~~~~ USER CODE END WHILE ~~~~~~~~~~~~~~~ */
        TX_Data[7] = Frame_No++;
        CAN_TX(0x3FF, 8, TX_Data); // wait until TX Done

        CAN_RX(&RX_ID, &RX_DLC, RX_Data); // wait until RX Done
        /* ~~~~~~~~~~~~~~~ USER CODE BEGIN 3 ~~~~~~~~~~~~~~~ */
    }
}
```

Frame In Transmitter mailbox:



CAN: Transmit Mailbox

Num	ID (Hex)	RTR	TXRQ	DLC	Data (Hex)
0	3FF	Data	0	8	43 41 4E 20 4E 4F 20 00
1	000	Data	0	0	00 00 00 00 00 00 00 00
2	000	Data	0	0	00 00 00 00 00 00 00 00

Identifier
CAN_TI0R: 0x7FE00000 □ IDE
EXID: 0x00000 □ RTR
STID: 0x3FF □ TXRQ

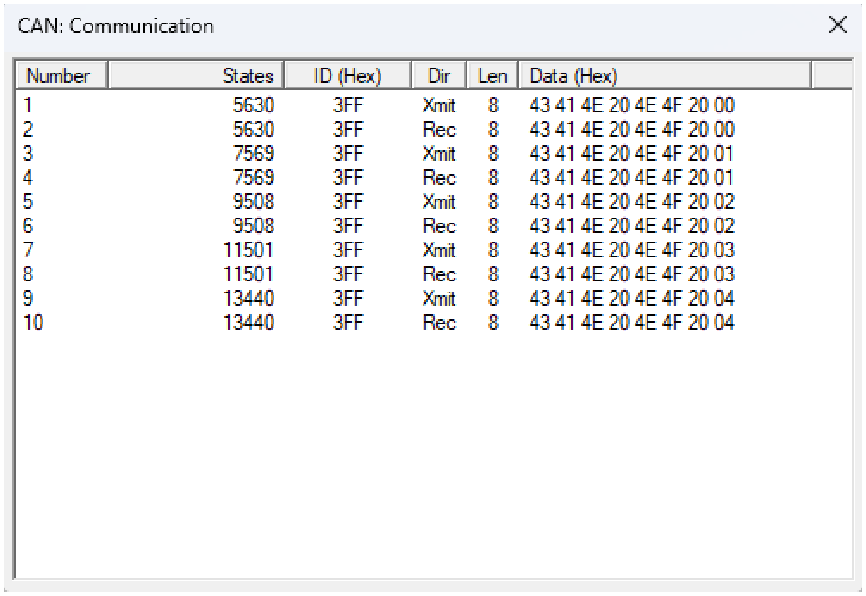
Length control & time stamp
CAN_TDT0R: 0x00000008
TIME: 0x0000 □ TGT
DLC: 8 bytes

Data
CAN_TDH0R: 0x00204F4E CAN_TDL0R: 0x204E4143

Status
CAN_TSR: 0x1C000003 CODE: 0 □ LOW0 ☑ TME0
□ ABRQ0 □ TERR0 □ ALST0 ☑ TXOK0 ☑ RQCP0

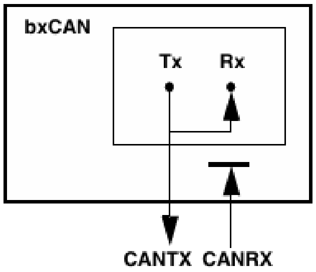
Settings:

Transmitting Data and Listen what I send using Loopback Mode: "In this mode, the bxCAN performs internal feedback from its Tx output to its Rx input."



CAN: Communication

Number	States	ID (Hex)	Dir	Len	Data (Hex)
1	5630	3FF	Xmit	8	43 41 4E 20 4E 4F 20 00
2	5630	3FF	Rec	8	43 41 4E 20 4E 4F 20 00
3	7569	3FF	Xmit	8	43 41 4E 20 4E 4F 20 01
4	7569	3FF	Rec	8	43 41 4E 20 4E 4F 20 01
5	9508	3FF	Xmit	8	43 41 4E 20 4E 4F 20 02
6	9508	3FF	Rec	8	43 41 4E 20 4E 4F 20 02
7	11501	3FF	Xmit	8	43 41 4E 20 4E 4F 20 03
8	11501	3FF	Rec	8	43 41 4E 20 4E 4F 20 03
9	13440	3FF	Xmit	8	43 41 4E 20 4E 4F 20 04
10	13440	3FF	Rec	8	43 41 4E 20 4E 4F 20 04


bxCAN
Tx
Rx
CANTX
CANRX