```
1 // NX: Updated 2011.03.17
2 // NX: Updated 2011.03.30
3
4 /*
5  * testCAN1.c
6  *
7  *  Created on: 16 Öåâ 2010
8  *      Author: ÔÇÁÍÏÓ
9  */
10
11 #include "main.h"
12 #include "CAN1.h"
13 #include "stm32f10x_can.h"
14 #include "stm32f10x_rcc.h"
15
16 #ifdef CAN1_H_
17                                  // STM32 CAN adaption layer
18 /*-----------------------------------------------------------------------------
19   initialize CAN interface
20  *-----------------------------------------------------------------------------*/
21
22     CAN_InitTypeDef CAN1_InitStruct;
23     CAN_FilterInitTypeDef  CAN_FilterInitStructure;
24     CanTxMsg TxMessage;
25     CanRxMsg RxMessage;
26     uint32_t i = 0;
27     uint8_t TransmitMailbox = 0;
28     uint32_t tmp = 0x00;                                    // ???
29
30 //*****************************************************************************
   ****************************************
31 void CAN1_setup (void) {
32
33     //  enable clock for Alternate Function & for GPIO A
34     RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN, ENABLE);
35     //GPIO_PinRemapConfig(GPIO_Remap1_CAN1, ENABLE);
36     // nx: GPIO_PinRemapConfig(GPIO_CAN1_onPA11_12, ENABLE);
37
38     /* CAN register init */
39     CAN_StructInit(&CAN1_InitStruct);
40     CAN_DeInit(CAN1);                                       // Reset CAN1
41
42     /* CAN cell init */
43     CAN1_InitStruct.CAN_TTCM=DISABLE;
44     CAN1_InitStruct.CAN_ABOM=DISABLE;
45     CAN1_InitStruct.CAN_AWUM=DISABLE;
46     CAN1_InitStruct.CAN_NART=ENABLE;    // NX: if NART is 0 then auto retransmits. Has
   to be 1(=ENABLE) to transmit once
47     CAN1_InitStruct.CAN_RFLM=DISABLE;
48     CAN1_InitStruct.CAN_TXFP=DISABLE;
49     CAN1_InitStruct.CAN_Mode=CAN_Mode_Normal;       //NX:
50     CAN1_InitStruct.CAN_SJW=CAN_SJW_1tq;
51     CAN1_InitStruct.CAN_BS1=CAN_BS1_5tq;
52     CAN1_InitStruct.CAN_BS2=CAN_BS2_2tq;            //NX: I changed to 2 from 3 to
   have total 8us in OSC seems correct 8us-->125KHz
53     CAN1_InitStruct.CAN_Prescaler = 35;             // 125Kbps
54     CAN_Init(CAN1, &CAN1_InitStruct);               // <---------- Write the Register
55
```

```
56      /*  Initialize the CAN_Prescaler member
57                                    1
58              BaudRate    = ------------------                    = 1/8us = 125KHz
59                            NominalBitTime
60
61              NominalBitTime = 1 x tq + tBS1 + tBS2                 = 1us + 5usec +2usec =
    8us
62              with:
63                  tBS1 = tq x (TS1[3:0] + 1),                     = 5usec
64                  tBS2 = tq x (TS2[2:0] + 1),                     = 3usec
65                  tq = (BRP[9:0] + 1) x tPCLK                     = 27.77777nsec x 36 =
    1usec
66                  tPCLK = time period of the APB1 clock, 36MHz    = 27.77777nsec
67                  BRP[9:0], TS1[3:0] and TS2[2:0] are defined in the CAN_BTR Register.
68                  where tq refers to the Time quantum
69      */
70
71
72      /* CAN filter init
     ************************************************************************************
    ********************/
73  //  CAN_FilterInitStructure.CAN_FilterNumber=0;
74      CAN_FilterInitStructure.CAN_FilterNumber=1;
75      CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
76      CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;
77      CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;
78      CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
79      CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
80      CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;
81  //  CAN_FilterInitStructure.CAN_FilterFIFOAssignment=0;
82      CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_FIFO0;
83      CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;
84      CAN_FilterInit(&CAN_FilterInitStructure);
85
86  }
87
88  /************************************************************************************
    ************************************
89  * @brief  NX: Prepare the Receive structure with the "default" values, Update the
    receive structure from FIFO
90  * @param  NX: None
91  * @retval NX: None
92  */
93  void CanRx(void) {
94                                      // NX: Prepare the Receive structure with the
    "default" values
95                                      // NX: RxMessage = pointer to a structure receive
    message which  contains CAN Id, CAN DLC,CAN datas,FMI
96                                      // NX: CAN receive FIFO mailbox identifier register
    (CAN_RIxR) (x=0..1)
97      RxMessage.StdId=0x00;           // NX: Bits 31:21 STID[10:0]/EXID[28:18]: The standard
    identifier or the MSBs of the extended identifier
98                                      // NX: (depending on the IDE bit value).
99      RxMessage.IDE=CAN_ID_STD;       // NX: IDE=CAN_RIxR[bit2]: Identifier
    extension:identifier type of message in the mailbox
100                                     // NX: 0: Standard, 1: Extended
101     RxMessage.DLC=0;                // NX: DLC=CAN_TDTxR[3:0]:
102                                     // NX: Data Length Code No# of bytes data frame
```

```
   contains.
103                                 // NX: A message can contain from 0 to 8 data bytes,
   depending on the value in the DLC field
104     RxMessage.Data[0]=0x00;     // NX: CAN receive FIFO 0 register (CAN_RF0R)
105     RxMessage.Data[1]=0x00;     // NX: CAN receive FIFO 1 register (CAN_RF1R)
106
107     CAN_Receive(CAN1, CAN_FIFO0, &RxMessage); // NX: Update the receive structure from
   FIFO
108 }
109
110 int CanRxValid(int i, int l) {
111     // NX: Check for valid message
112     if (RxMessage.StdId != i) {
113         return FAILED;
114     }
115     if (RxMessage.IDE != CAN_ID_STD) {
116         return FAILED;
117     }
118     if (RxMessage.DLC != l) {
119         return FAILED;
120     }
121 //  if (RxMessage.Data[0]  != d0) {
122 //      return FAILED;
123 //  }
124 //  if (RxMessage.Data[1]  != d1) {
125 //      return FAILED;
126 //  }
127 //  if (RxMessage.Data[2]  != d2) {
128 //      return FAILED;
129 //  }
130 //  if (RxMessage.Data[3]  != d3) {
131 //      return FAILED;
132 //  }
133     return PASSED; /* Test Passed */
134 }
135
136 //*****************************************************************************
   ***************************************
137
138 void CanTx(int i,int l,int d0,int d1, int d2,int d3) {      // NX: Identifier=0x11,
   Identifier-Type=Standard, Message-size=2bytes, Data[0]=0xCA, Data[1]=0xFE
139     uint8_t TransmitMailbox = 0;
140     TxMessage.StdId=i;
141     TxMessage.RTR=CAN_RTR_DATA;
142     TxMessage.IDE=CAN_ID_STD;
143     TxMessage.DLC=l;
144     TxMessage.Data[0]=d0;
145     TxMessage.Data[1]=d1;
146     TxMessage.Data[0]=d2;
147     TxMessage.Data[1]=d3;
148     TransmitMailbox=CAN_Transmit(CAN1, &TxMessage);                                 //
   NX: Transmit the Message
149 }
150
151 void CanTxWait(void){
152     uint32_t i = 0;
153     while ((CAN_TransmitStatus(CAN1, TransmitMailbox) != CANTXOK)               // NX:
   Wait Transmission to complete
```

```
154             && (i != 0xFF))
155     {
156         i++;
157     }
158 
159     i = 0;
160     while ((CAN_MessagePending(CAN1, CAN_FIFO0) < 1)                          // NX:
    Wait TX FIFO to empty
161             && (i != 0xFF))
162     {
163         i++;
164     }
165 }
166 
167 #endif
168 
```