```
/**
 1
 2
 3
        @copyright © 2010 - 2020, Fraunhofer-Gesellschaft zur Foerderung der
 4
      * angewandten Forschung e.V. All rights reserved.
 5
 6
      * BSD 3-Clause License
 7
      * Redistribution and use in source and binary forms, with or without
 8
      * modification, are permitted provided that the following conditions are met:
 9
      * 1. Redistributions of source code must retain the above copyright notice,
10
            this list of conditions and the following disclaimer.
11
       2. Redistributions in binary form must reproduce the above copyright
12
            notice, this list of conditions and the following disclaimer in the
13
            documentation and/or other materials provided with the distribution.
14
      * 3. Neither the name of the copyright holder nor the names of its
            contributors may be used to endorse or promote products derived from
15
16
            this software without specific prior written permission.
17
18
      * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
19
      * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
20
      * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
21
      * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
22
      * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
23
      * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
24
      * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
25
      * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
26
      * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27
      * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
28
      * POSSIBILITY OF SUCH DAMAGE.
29
30
      * We kindly request you to use one or more of the following phrases to refer
31
      * to foxBMS in your hardware, software, documentation or advertising
32
      * materials:
33
34
      * ″ This product uses parts of foxBMS&req; ″
35
36
      * ″ This product includes parts of foxBMS&req; ″
37
38
      * ″ This product is derived from foxBMS&req; ″
39
40
      */
41
     /**
42
43
     * @file
                can.c
      * @author foxBMS Team
44
      * @date 12.07.2015 (date of creation)
45
      * @ingroup DRIVERS
46
47
      * @prefix CAN
48
49
      * @brief Driver for the CAN module
50
51
      * Implementation of the CAN Interrupts, initialization, buffers,
      * receive and transmit interfaces.
52
```

```
53
54
     * /
55
56
     /*-----Includes -----*/
57
     #include "can.h"
58
59
     /*====== Macros and Definitions ============*/
     #define ID 16BIT FIF00
                                  (UU)
61
     #define ID 16BIT FIF01
                                  (1U)
     #define ID_32BIT_FIF00
 62
                                  (2U)
     #define ID_32BIT_FIF01
                                  (3U)
 #define MSK 16BIT FIF00
                                  (4U)
#define MSK 16BIT FIF01
                                  (5U)
     #define MSK 32BIT
66
                                  (6U)
 67
     /*======= Constant and Variable Definitions ==========*/
 68
     uint8 t canNode0 listenonly mode = 0;
 69
70
     uint8 t canNode1 listenonly mode = 0;
71
72
     #if CAN USE CAN NODEO
73 #if CANO_USE_TX_BUFFER
74
     CAN_TX_BUFFERELEMENT_s can0_txbufferelements[CAN0_TX_BUFFER_LENGTH];
75
     CAN TX BUFFER s can0 txbuffer = {
76
         .length = CANO_TX_BUFFER_LENGTH,
77
         .buffer = &can0_txbufferelements[0],
78
     };
79
     #endif /* CANO USE TX BUFFER */
80
 81
     #if CANO USE RX BUFFER
     CAN RX BUFFERELEMENT s can0 rxbufferelements [CAN0 RX BUFFER LENGTH];
 82
83
     CAN_RX_BUFFER_s can0_rxbuffer = {
         .length = CANO_RX_BUFFER_LENGTH,
84
         .buffer = &can0 rxbufferelements[0],
85
86
     };
87
     #endif /* CANO_USE_RX_BUFFER */
88
     #if CANO BUFFER BYPASS NUMBER OF IDs > 0 This is for Rx.
     uint8 t can0 fastLinkIndex[CAN0 BUFFER BYPASS NUMBER OF IDs]; /* Link Table for bufferBypassing */
90
     #endif
91
 92
93
     CAN ERROR s CANO errorStruct = {
94
         .canError = HAL_CAN_ERROR_NONE,
         95
96
     };
     #endif /* CAN USE CAN NODEO */
97
98
99
     #if CAN USE CAN NODE1
     #if CAN1 USE TX BUFFER
100
     CAN_TX_BUFFERELEMENT_s can1_txbufferelements[CAN1_TX_BUFFER_LENGTH];
101
102
     CAN_TX_BUFFER_s can1_txbuffer = {
103
            .length = CAN1_TX_BUFFER_LENGTH,
104
            .buffer = &can1 txbufferelements[0],
```

```
105
      };
106
      #endif /* CAN1 USE TX BUFFER */
107
108
      #if CAN1 USE RX BUFFER
109
      CAN RX BUFFERELEMENT s can1 rxbufferelements[CAN1 RX BUFFER LENGTH];
110
      CAN_RX_BUFFER_s can1_rxbuffer = {
111
              .length = CAN1_RX_BUFFER_LENGTH,
112
              .buffer = &can1 rxbufferelements[0],
113
      };
114
      #endif /* CAN1 USE RX BUFFER */
115
      #if CAN1 BUFFER BYPASS NUMBER OF IDs > 0
116
117
      uint8 t can1 fastLinkIndex[CAN1 BUFFER BYPASS NUMBER OF IDs]; /* Link Table for bufferBypassing */
118
      #endif
119
                                                                      cansignal_cfg.h
                                                                                  C stm32f4xx_hal_can.h × C cansignal_cfg.c
                                                                                                             C can_cfq.c
120
      CAN ERROR s CAN1 errorStruct = {
                                                                      mcu-hal > STM32F4xx_HAL_Driver > Inc > C stm32f4xx_hal_can.h > •○ CAN_FilterTypeDef
121
          .canError = HAL CAN ERROR NONE,
                                                                            uint32_t FilterScale;
          122
                                                                      151
123
      };
                                                                      152
124
      #endif /* CAN USE CAN NODE1 */
                                                                      153
                                                                            uint32 t FilterActivation;
125
                                                                      154
      /* ******************
126
                                                                      155
                                                                      156
                                                                            uint32_t SlaveStartFilterBank; /*!< Select the start filter bank
127
       * Dummies for filter initialization and message reception
                                                                      157
128
                                                                      158
129
                                                                      159
130
      CAN FilterTypeDef sFilterConfig = {
                                                                      160
131
             /* No need to insert here something */
                                                                      161
132
              .FilterActivation = ENABLE, /* enable the filter */
                                                                      162
133
     };
                                                                      163
                                                                            CAN_FilterTypeDef;
134
135
      /*======= Function Prototypes =========*/
136
      /* Inits */
137
      static uint8 t CAN GetNextID (CAN MSG RX TYPE s* can RxMsgs, uint8 t numberOfRxIDs, uint8 t startIndex,
138
             uint8 t filterCase);
139
      static uint8_t CAN_NumberOfNeededFilters(CAN_MSG_RX_TYPE_s* can_RxMsgs, uint8_t* numberOfDifferentIDs, uint32_t*
140
      static uint32_t CAN_InitFilter (CAN_HandleTypeDef* ptrHcan, CAN_MSG_RX_TYPE_s* can_RxMsgs, uint8_t numberOfRxMsgs);
141
142
      /* Interrupts */
143
      static void CAN TxCpltCallback (CAN Node TypeDef e canNode);
144
      static void CAN RxMsq(CAN NodeTypeDef e canNode, CAN HandleTypeDef* ptrHcan, uint8 t FIFONumber);
145
146
      /* Buffer/Interpreter */
147
      static STD RETURN TYPE e CAN BufferBypass (CAN NodeType Def e canNode, uint32 t msqID, uint8 t* rxData, uint8 t DLC,
148
             uint8 t RTR);
149
      static STD_RETURN_TYPE_e CAN_InterpretReceivedMsg(CAN_NodeTypeDef_e canNode, uint32_t msgID, uint8_t* data, uint8_t
      DLC,
150
             uint8 t RTR);
151
152
      153
      /* ***********
154
```

```
C cansignal_cfg.h C can_cfg.h C stm32f4xx_hal_can.h ×
                                                                                                              C cansignal_cfq.h C can_cfq.h X C stm32f4xx_hal_can.h
                                                                                                                                                                         C cansignal_cfg.c
155
           * Initialization
                                                                       mcu-hal > STM32F4xx_HAL_Driver > Inc > C stm32f4xx_hal_can.h > • C
           **********
156
                                                                                                              mcu-primary > src > driver > config > C can_cfg.h > •○ CAN_MSG_RX_TYPE_s
                                                                       225 typedef struct __CAN_HandleTypeDef
157
                                                                           CAN_TypeDef
                                                                                                                     typedef struct CAN MSG RX TYPE {
158
         uint32_t CAN_Init(void) {
                                                                                                                         uint32_t ID; /*!< message ID */</pre>
                                                                       229 CAN InitTypeDef
159
               uint32 t retval = 0;
                                                                                                               299
                                                                                                                         uint32_t mask; /*!< mask or 0x0000 to select list mode */</pre>
                                                                           __IO HAL_CAN_StateTypeDef State;
160
                                                                                                               300
                                                                                                                         uint8_t DLC; /*!< data length */</pre>
                                                                                                                         uint8_t RTR; /*!< rtr bit */</pre>
161
         #if CAN_USE_CAN_NODEO
                                                                                                              301
                                                                                                              302
                                                                                                                         uint32_t fifo; /*!< selected CAN hardware (CAN FILTER FIF00 or CAN F
162
               /* DeInit CANO handle */
                                                                       236 } CAN HandleTypeDef;
                                                                                                                         STD_RETURN_TYPE_e (*func)(uint32_t ID, uint8_t*, uint8_t, uint8_t);
163
               if (HAL CAN DeInit(&hcan0) != HAL OK) {
                                                                                                               304 } CAN MSG RX TYPE s:
164
                     /* Error deintializing handle -> set error bit */
165
                     retval |= STD_ERR_BIT_0;
                                                                                                                             C cansignal_cfg.h C can_cfg.c × C can_cfg.h C stm32f4xx_hal_can.h C cansignal_cfg.c C can.c
166
                                                                                                                            mcu-primary > src > driver > config > C can_cfg.c > (e) can0_RxMsgs
167
                                                                                                                             453 /* Bypassed messages are --- ALSO --- to be configured here. See further down for bypass ID setting! */
                                                                                                                             454 CAN_MSG_RX_TYPE_s can0_RxMsgs[] = {
168
               /* Init CANO-handle */
                                                                                                                                     { 0x120, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< state request
               if (HAL_CAN_Init(&hcan0) != HAL OK) {
169
                                                                                                                                     { CAN_ID_SOFTWARE_RESET_MSG, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< software reset
                     /* Error intializing handle -> set error bit */
170
                                                                                                                             459 #ifdef CURRENT SENSOR ISABELLENHUETTE TRIGGERED
                                                                                                                                     { 0x35C, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL }, /*!< current sensor I */ { 0x35D, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL }, /*!< current sensor U1 */
171
                     retval |= STD ERR BIT 1;
                                                                                                                                     { 0x35E, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< current sensor U2
172
               }
                                                                                          This should have been called
                                                                                                                                     { 0x35F, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL },
173
                                                                                                                                     { 0x525, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL },
                                                                                          can0_RxMsgHeader since it is
                                                                                                                                     { 0x526, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL },
174
               /* Configure CANO hardware filter */
                                                                                        not the true message.
                                                                                                                            466
                                                                                                                                     { 0x527, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL }
                                                                                                                                      { 0x528, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL },
175
               retval |= CAN_InitFilter(&hcan0, &can0_RxMsgs[0], can_CAN0_rx_length);
                                                                                                                            468 #else /* CURRENT SENSOR ISABELLENHUETTE CYCLIC */
                                                                                                                                     { 0x521, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL },
                                                                                                                                                                    /*!< current sensor I in cyclic mode */
176
                                                                                                                                      { 0x522, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< current sensor U1 in cyclic mode */
                                                                                                                                      { 0x523, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL },
                                                                                                                                                                     /*!< current sensor U2 in cyclic mode */
177
               /* Check if more rx messages are bypassed than received */
                                                                                                                                      { 0x524, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL },
                                                                                                                                                                     /*!< current sensor U3 in cyclic mode */
                                                                                                                                      { 0x525, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL },
                                                                                                                                                                     /*!< current sensor T in cyclic mode */
178
         #pragma GCC diagnostic push
                                                                                                                                      { 0x526, 0xFFFF, 8, 0, CAN_FILTER_FIF00, NULL },
                                                                                                                                                                    /*!< current sensor Power in cyclic mode */
                                                                                                                            475
                                                                                                                                      { 0x527, 0xFFFF, 8, 0, CAN FILTER FIFO0, NULL },
                                                                                                                                                                    /*/< current sensor C-C in cyclic mode */
179
               /* configurations might exist that use this comparison */
                                                                                                                                      { 0x528, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*!< current sensor E-C in cyclic mode */
                                                                                                                                      * CURRENT SENSOR_ISABELLENHUETTE
         #pragma GCC diagnostic ignored "-Wtype-limits"
180
                                                                                                                                      { 0x100, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< debug message
                                                                                                                                      { 0x777, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< request SW version */
181
               if (CANO BUFFER BYPASS NUMBER OF IDs > can CANO rx length) {
                                                                                                                                      { 0x121, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL }, /*/< engine request */
182
         #pragma GCC diagnostic pop
183
                     retval |= STD_ERR_BIT_7;
                                                                                                                     C cansignal_cfg.h C can_cfg.c × C can_cfg.h C stm32f4xx_hal_can.h
                                                                                                                                                                                     C cansignal_cfg.c
184
               }
                                                                                                                      mcu-primary > src > driver > config > C can_cfg.c > [6] can_CAN0_rx_length
185
                                                                                                                      488 const uint8_t can_CANO_rx_length = sizeof(can0_RxMsgs)/sizeof(can0_RxMsgs[0]);
186
               /* Enable CANO message receive interrupt FIFOO */
187
               if (HAL CAN ActivateNotification(&hcan0, CAN IT RX FIFO0 MSG PENDING) != HAL OK) {
188
                     retval |= STD ERR BIT 8;
189
190
               hcan0.State = HAL_CAN_STATE_READY;
               /* Enable CANO message receive interrupt FIFO1 */
191
                                                                                                 It seems that FIFO 1 is never used.
192
               if (HAL CAN ActivateNotification(&hcan0, CAN IT RX FIF01 MSG PENDING) != HAL OK) {
193
                     retval |= STD_ERR_BIT_9;
194
195
               /* Enable CANO Transmit mailbox empty interrupt */
196
               if (HAL_CAN_ActivateNotification(&hcan0, CAN_IT_TX_MAILBOX_EMPTY) != HAL_OK) {
197
                      retval |= STD_ERR_BIT_10;
198
               }
199
                          Debug freeze
200
               /* set DBF bit to 0 for CAN activity while in debug mode */ DBF = 0 ==> CAN works during debug.
               CLEAR BIT (hcan0.Instance->MCR, CAN MCR DBF);
201
202
203
         #if CAN_USE_STANDBY_CONTROL == 1
204
               IO_WritePin(CAN_0_TRANS_STANDBY_CONTROL, IO_PIN_SET);
205
         #endif /* CAN USE STANDBY CONTROL == 1 */
206
```

```
207
          /* Start CAN */
208
          HAL CAN Start (&hcan0);
209
      #endif /* CAN USE CAN NODE0 */
210
211
212
      #if CAN_USE_CAN_NODE1
213
          /* DeInit CAN1 handle */
214
          if (HAL CAN DeInit(&hcan1) != HAL OK) {
215
              /* Error deintializing handle -> set error bit */
216
              retval |= STD_ERR_BIT_11;
217
         }
218
219
          /* Init CAN1-handle */
220
          if (HAL CAN Init(&hcan1) != HAL OK) {
221
              /* Error intializing handle -> set error bit */
222
              retval |= STD ERR BIT 12;
223
          }
224
225
          /* Configure CAN1 hardware filter */
226
          retval |= CAN InitFilter(&hcan1, &can1 RxMsqs[0], can CAN1 rx length);
227
228
          /* Check if more RX messages are bypassed than received */
229
      #pragma GCC diagnostic push
230
          /* configurations might exist that use this comparison */
      #pragma GCC diagnostic ignored "-Wtype-limits"
231
232
          if (CAN1 BUFFER BYPASS NUMBER OF IDs > can CAN1 rx length) {
233
      #pragma GCC diagnostic pop
234
              retval |= STD_ERR_BIT_13;
235
          }
236
237
          /* Enable CAN1 message receive interrupt FIF00 */
238
          if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIF00_MSG_PENDING) != HAL_OK) {
239
              retval |= STD ERR BIT 14;
240
241
          /* Enable CAN1 message receive interrupt FIF01 */
          hcan1.State = HAL_CAN_STATE_READY;
242
243
          if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIF01_MSG_PENDING) != HAL_OK) {
244
              retval |= STD ERR BIT 15;
245
246
          /* Enable CAN1 Transmit mailbox empty interrupt */
247
          if (HAL CAN ActivateNotification (&hcan1, CAN IT TX MAILBOX EMPTY) != HAL OK) {
248
              retval |= STD_ERR_BIT_16;
249
          }
250
251
          /* set DBF bit to 0 for CAN activity while in debug mode */
252
          CLEAR BIT (hcan1.Instance->MCR, CAN MCR DBF);
253
254
      #if CAN USE STANDBY CONTROL == 1
255
          IO_WritePin(CAN_1_TRANS_STANDBY_CONTROL, IO_PIN_SET);
256
      #endif /* CAN_USE_STANDBY_CONTROL == 1 */
257
2.58
          /* Start CAN */
```

```
259
          HAL CAN Start (&hcan1);
      #endif /* CAN USE CAN NODE1 */
260
261
          return retval;
262
      }
263
      /**
264
265
       * @brief Initializes message filtering
266
       * @retval 0: if no error occurred, otherwise error code
267
268
      static uint32_t CAN_InitFilter(CAN_HandleTypeDef* ptrHcan, CAN_MSG_RX_TYPE_s* can_RxMsqs, uint8_t numberOfRxMsqs) {
269
          /* Contains the occurrence of of the different filter cases
                                                                                 * (Used for numberOfDifferentIDs)
270
            * [0] - ID List mode 16bit routed on FIF00
            * [1] - ID List mode 16bit routed on FIF01
271
272
            * [2] - ID List mode 32bit routed on FIF00
273
            * [3] - ID List mode 32bit routed on FIF01
274
           * [4] - Mask mode 16bit routed on FIFO0
275
           * [5] - Mask mode 16bit routed on FIF01
           * [6] - Mask mode 32bit
276
277
           uint8 t numberOfDifferentIDs[7] = \{0, 0, 0, 0, 0, 0, 0, 0, \};
278
           static uint8 t filterNumber = 0; /* Number of the filter to be initialized */
279
          uint32_t retval = 0;
280
                                                                See Lines 459 to 557
281
           /* Calculate number of needed filter banks */
2.82
          uint8_t numberNeededFilters = CAN_NumberOfNeededFilters (can_RxMsqs, &numberOfDifferentIDs[0], &retval);
          numberNeededFilters--; /* Decrement by one because IS_CAN_FILTER_BANK_DUAL checks filter bank numbers starting
283
          with 0 */
                                                                     mcu-hal\STM32F4xx HAL Driver\Inc\stm32f4xx hal can.h:
284
          if (IS_CAN_FILTER_BANK_DUAL(numberNeededFilters)) { 735: #define IS_CAN_FILTER_BANK_DUAL(BANK) ((BANK) <= 270)
285
               uint8_t j = 0; /* Counts the number of initialized instances per case */
286
               uint8 t posRxMsqs = 0; /* Iterator for can RxMsqs[] */
               uint8 t numberRegistersUsed = 0; /* Counts how many register space is already used in each filter bank (max.
287
               64bit) */
288
               uint8_t caseID = 0; /* indicates the actual filter mode that will be initialized */
289
                                          True for hcan0
                                                                           stm32f4xx_hal_can.h C:\bms\phxbms\embedded-software\mcu-hal\STM32F4xx_HAL_Driver\lnc - Definitions (1)
290
               if (ptrHcan->Instance == CAN2) {
291
                   /* Set start slave bank filter */
                                                                             #define IS_CAN_PRESCALER(PRESCALER) (((PRESCALER) >= 1U) && ((PRESCALER) <= 1024U))
                                                                             #lefine IS_CAN_FILTER_ID_HALFWORD(HALFWORD) ((HALFWORD) <= 0xFFFFU)
292
                   sFilterConfig.FilterBank = filterNumber;
                                                                             #define IS CAN FILTER BANK DUAL(BANK) ((BANK) <= 27U)
293
                                                                             #define IS_CAN_FILTER_BANK_SINGLE(BANK) ((BANK) <= 13U)
294
               for (caseID = 0; caseID < 2u; caseID++) {</pre>
295
                   /* ID List mode 16bit routed on FIFO0 or FIFO1 */
                                                                                      Dual filter bank--28 filter banks.
296
297
                   if (numberOfDifferentIDs[caseID] > 0U) {
298
                        i = 0;
299
                        while (j < numberOfDifferentIDs[caseID]) {</pre>
300
                             /* Until all IDs in that filter case are treated */
301
                                          See Lines 569 to 599
302
                            posRxMsqs = CAN_GetNextID (can_RxMsqs, numberOfRxMsqs, posRxMsqs, caseID); /* Get array position
                            of next ID */
303
304
                            switch (numberRegistersUsed) {
305
                                 case 0: /* 1st ID per filter bank */
306
                                     sFilterConfig.FilterIdHigh = ((can_RxMsgs[posRxMsgs].ID << 5)</pre>
307
                                              | can RxMsqs[posRxMsqs].RTR << 4);</pre>
```

```
308
                                  j++;
309
                                  break:
310
311
                              case 1: /* 2nd ID */
312
                                  sFilterConfig.FilterIdLow = ((can RxMsqs[posRxMsqs].ID << 5)
313
                                          can_RxMsqs[posRxMsqs].RTR << 4);</pre>
314
                                  j++;
315
                                  break;
316
317
                              case 2: /* 3rd ID */
318
                                  sFilterConfig.FilterMaskIdHigh = ((can_RxMsgs[posRxMsgs].ID << 5)</pre>
319
                                          can RxMsqs[posRxMsqs].RTR << 4);
320
                                  j++;
321
                                  break;
322
323
                              case 3: /* 4th ID */
324
                                  sFilterConfig.FilterMaskIdLow = ((can_RxMsgs[posRxMsgs].ID << 5)
325
                                          can_RxMsqs[posRxMsqs].RTR << 4);</pre>
326
                                  j++;
327
                                  break;
328
                          numberRegistersUsed = j % 4U;  /* space for 4 IDs a 16 bit in one filter bank */
329
330
                          if ((numberRegistersUsed == 0 && j > 1U) || (j == numberOfDifferentIDs[caseID])) {
331
                              /* all registers in filter bank used OR no more IDs in that case */
332
                              sFilterConfig.FilterMode = CAN_FILTERMODE_IDLIST;
                              sFilterConfig.FilterScale = CAN FILTERSCALE 16BIT;
333
334
                              if (caseID == ID 16BIT FIF00) {
335
                                  sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
336
                              } else if (caseID == ID_16BIT_FIF01) {
                                  sFilterConfig.FilterFIFOAssignment = CAN FILTER FIFO1;
337
338
339
                              sFilterConfig.FilterBank = filterNumber;
340
                              HAL CAN ConfigFilter (ptrHcan, &sFilterConfig); /* initialize filter bank */
341
                              filterNumber++; /* increment filter number */
342
                          }
343
                          posRxMsqs++; /* increment array position to find next valid ID */
344
                      }
345
                      posRxMsqs = 0;  /* reset variables for next case */
346
                      numberRegistersUsed = 0;
347
                  }
348
349
              for (caseID = 2; caseID < 6U; caseID++) {</pre>
350
                  /* ID List mode 32bit routed on FIF00 or FIF01; Mask mode 16bit routed on FIF00 or FIF01 */
351
                  j = 0;
352
                  if (numberOfDifferentIDs[caseID] > OU) {
353
                      while (j < numberOfDifferentIDs[caseID]) {</pre>
                          /* Until all IDs in that filter case are treated */
354
355
356
                          posRxMsqs = CAN_GetNextID(can_RxMsqs, numberOfRxMsqs, posRxMsqs, caseID); /* Get array position
                          of next ID */
357
358
                          switch (numberRegistersUsed) {
```

```
case 0: /* first 32bit per filter bank */
359
360
                                  if (caseID == ID_32BIT_FIF00 || caseID == ID_32BIT_FIF01) { /* list mode 32bit */
361
                                      sFilterConfig.FilterIdHigh = ((can_RxMsqs[posRxMsqs].ID << 3) >> 16); /* 1 << 2 is
                                      for setting IDE bit to receive extended identifiers */
362
                                      sFilterConfig.FilterIdLow = (uint16 t)((can RxMsqs[posRxMsqs].ID << 3) | 1 << 2
363
                                              can_RxMsqs[posRxMsqs].RTR << 1);</pre>
364
                                  } else if (caseID == MSK_16BIT_FIF00 || caseID == MSK_16BIT_FIF01) { /* mask mode
365
                                      sFilterConfig.FilterIdHigh = ((can RxMsqs[posRxMsqs].ID << 5)
366
                                              can_RxMsqs[posRxMsqs].RTR << 4);</pre>
367
                                      sFilterConfig.FilterMaskIdHigh = can_RxMsqs[posRxMsqs].mask;
                                      sFilterConfig.FilterIdLow = 0 \times 0000; /* set second register to 0 \times FFFF, */
368
                                      sFilterConfig.FilterMaskIdLow = 0xFFFF; /* otherwise all messages would be received */
369
370
371
                                  j++;
372
                                  break;
373
                              case 1: /* second 32bit per filter bank */
374
375
                                  if (caseID == ID 32BIT FIF00 || caseID == ID 32BIT FIF01) { /* list mode 32bit */
376
                                      sFilterConfig.FilterMaskIdHigh = ((can RxMsqs[posRxMsqs].ID << 3) >> 16); /* 1 << 2
                                      is for setting IDE bit to receive extended identifiers */
377
                                      sFilterConfig.FilterMaskIdLow = (uint16_t)((can_RxMsgs[posRxMsgs].ID << 3) | 1 << 2
378
                                              can RxMsqs[posRxMsqs].RTR << 1);</pre>
                                  } else if (caseID == MSK_16BIT_FIF00 || caseID == MSK_16BIT_FIF01) { /* mask mode
379
                                  16bit */
                                      sFilterConfig.FilterIdLow = ((can_RxMsgs[posRxMsgs].ID << 5)</pre>
381
                                              can RxMsqs[posRxMsqs].RTR << 4);</pre>
382
                                      sFilterConfig.FilterMaskIdLow = can_RxMsqs[posRxMsqs].mask;
383
                                  }
384
                                  j++;
385
                                  break;
386
387
                          numberRegistersUsed = j % 2;  /* Space for two IDs a 32bit or two mask a 16bit */
388
                          if ((numberRegistersUsed == 0 && j > 1U) || (j == numberOfDifferentIDs[caseID])) {
389
                              /* all registers in filter bank used OR no more IDs in that case */
390
                              if (caseID == ID_32BIT_FIF00 || caseID == ID_32BIT_FIF01) {
                                  sFilterConfig.FilterMode = CAN_FILTERMODE_IDLIST;
391
392
                                  sFilterConfig.FilterScale = CAN FILTERSCALE 32BIT;
393
                                  if (caseID == ID_32BIT FIF00)
                                      sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFOO;
394
395
                                  else
396
                                      sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO1;
397
                              } else if (caseID == MSK 16BIT FIF00 || caseID == MSK 16BIT FIF01) {
398
                                  sFilterConfig.FilterMode = CAN FILTERMODE IDMASK;
399
                                  sFilterConfig.FilterScale = CAN FILTERSCALE 16BIT;
400
                                  if (caseID == MSK 16BIT FIF00)
401
                                      sFilterConfig.FilterFIFOAssignment = CAN FILTER FIFOO;
402
                                  else
403
                                      sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO1;
404
405
                              sFilterConfig.FilterBank = filterNumber;
                              HAL CAN ConfigFilter (ptrHcan, &sFilterConfig); /* initialize filter bank */
406
```

```
/* increment filter number */
407
                              filterNumber++;
408
409
                                          /* increment array position to find next valid ID */
                          posRxMsqs++;
410
411
                      posRxMsqs = 0;
                                          /* reset variables for next case */
412
                      numberRegistersUsed = 0;
413
                  }
414
              }
415
              j = 0;
416
              if (numberOfDifferentIDs[MSK_32BIT] > 0U) {
417
                  /* Mask mode 32bit */
418
                  while (j < numberOfDifferentIDs[MSK_32BIT]) { /* Get array position of next ID */</pre>
419
420
                      /* Until all IDs in that filter case are treated */
                      posRxMsqs = CAN GetNextID(can RxMsqs, numberOfRxMsqs, posRxMsqs, MSK 32BIT);
421
422
423
                      sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
424
                      sFilterConfig.FilterScale = CAN FILTERSCALE 32BIT;
425
                      sFilterConfig.FilterIdHigh = ((can RxMsgs[posRxMsgs].ID << 3) >> 16); /* 1 << 2 is for setting IDE
                      bit to receive extended identifiers */
426
                      sFilterConfig.FilterIdLow = (uint16_t)((can_RxMsgs[posRxMsgs].ID << 3) | 1 << 2</pre>
427
                              | can_RxMsgs[posRxMsgs].RTR << 1);</pre>
428
                      sFilterConfig.FilterMaskIdHigh = can RxMsqs[posRxMsqs].mask >> 16;
429
                      sFilterConfig.FilterMaskIdLow = (uint16_t)(can_RxMsgs[posRxMsgs].mask);
430
                      sFilterConfiq.FilterFIFOAssignment = can_RxMsqs[posRxMsqs].fifo;
431
                      sFilterConfig.FilterBank = filterNumber;
432
                      HAL CAN ConfigFilter (ptrHcan, &sFilterConfig);
433
                      filterNumber++;
434
                      posRxMsqs++;
435
                      j++;
436
                  }
437
              }
438
          } else {
439
              /* Too many filterbanks needed! Check the value of CAN NUMBER OF FILTERBANKS */
440
              /* If correct, try to reduce the IDs through masks or optimize used filter bank space. */
441
              /* Number of different filter cases can be evaluated in numberOfDifferentIDs[]. One */
              /* filter bank can filter as many messages as followed: */
442
              /* 4 IDs in list mode 16bit */
443
              /* 2 IDs in list mode 32bit and mask mode 16bit */
444
445
              /* 1 ID in 32bit mask mode */
446
              retval |= STD ERR BIT 6;
447
          }
448
          return retval;
449
      }
450
451
      /**
       * @brief Returns the number of filters that have to be initialized
452
453
454
       * @param can_RxMsgs:
                                         pointer to receive message struct
       * @param numberOfDifferentIDs: pointer to array, where to store the specific number of different IDs
455
456
457
       * @retval number of needed filters
```

```
458
459
      static uint8 t CAN NumberOfNeededFilters (CAN MSG RX TYPE s* can RxMsgs, uint8 t* numberOfDifferentIDs, uint32 t*
      error) {
460
          static uint8_t retVal = 0;
                                              /* static so save the number of filters from CANO and add to the ones from CANO */
461
           uint16 t can rx length = 0;
462
463
           if (can RxMsgs == &can0 RxMsgs[0]) {
464
               can rx length = can CANO rx length;
465
           } else if (can RxMsqs == &can1 RxMsqs[0]) {
466
               can_rx_length = can_CAN1_rx_length;
467
           } else {
468
               can rx length = 0;
469
          }
470
471
           for (uint8_t i = 0; i < can_rx_length; i++) {</pre>
      #if (CANO BUFFER BYPASS NUMBER_OF_IDs > 0) && (CAN_USE_CAN_NODE0 == 1)
472
                    (This is defined as 1)
473
474
               if (can_RxMsqs == &can0_RxMsqs[0]) {
                    /* Set buffer bypass IDs link table */
475
                    for (uint8 t k = 0; k < CANO BUFFER BYPASS NUMBER OF IDs; k++) {
476
477
                        if (can_RxMsqs[i].ID == can0_bufferBypass_RxMsqs[k]) {
478
                             /* bypass ID == ID in message receive struct */
479
480
                             can0_fastLinkIndex[k] = i; /* set for can_bufferBypass_RxMsqs[k] link to array index */
481
                            break;
                                                            C can_cfq.c X C can_cfq.h C stm32f4xx_hal_can.h C cansignal_cfq.c C can.c C can.h
482
                        }
                                                            mcu-primary > src > driver > config > C can_cfg.c > (a) can0_bufferBypass_RxMsgs
                                                            495 /* These ID: have to be included in the configuration for the filters in can RxMsgs[]! */
483
                   }
                                                            496 uint32_t can0_bufferBypass_RxMsgs[CAN0_BUFFER_BYPASS_NUMBER_OF_IDs] = { CAN_ID_SOFTWARE_RESET_MSG };
484
485
      #endif /* (CAN0_BUFFER_BYPASS_NUMBER_OF_IDs > 0) && (CAN_USE_CAN_NODE0 == 1) */
486
      #if (CAN1 BUFFER BYPASS NUMBER OF IDs > 0) && (CAN USE CAN NODE1 == 1)
487
488
               if (can_RxMsqs == &can1_RxMsqs[0]) {
                    /* Set buffer bypass IDs link table */
489
                    for (int k = 0; k < CAN1 BUFFER BYPASS NUMBER OF IDs; k++) {
490
491
                        if (can1_RxMsqs[i].ID == can1_bufferBypass_RxMsqs[k]) {
492
                             /* bypass ID == ID in message receive struct */
493
494
                             can1 fastLinkIndex[k] = i; /* set for can1 bufferBypass RxMsqs[k] link to array index */
495
                            break;
496
                        }
497
                   }
498
               }
499
      #endif /* (CAN1_BUFFER_BYPASS_NUMBER_OF_IDs > 0) && (CAN_USE_CAN_NODE1 == 1) */
500
               if (can_RxMsgs[i].mask == 0 && IS_CAN_STDID(can_RxMsgs[i].ID)) {
                                                                                          mcu-hal\STM32F4xx_HAL_Driver\Inc\stm32f4xx_hal_can.h:
                   /* ID List mode 16bit */
501
                                                                                           747: #define IS_CAN_STDID(STDID) ((STDID) <= 0x7FFU)
502
503
                    if (can RxMsqs[i].fifo == CAN FILTER FIF00) {
                        numberOfDifferentIDs[ID 16BIT FIFO0]++;
504
505
                    } else if (can_RxMsqs[i].fifo == CAN_FILTER_FIFO1) {
506
                        numberOfDifferentIDs[ID_16BIT_FIF01]++;
507
                    } else {
508
                        /* Invalid FIFO selection; check can RxMsqs[i].fifo value */
```

```
509
                       *error |= STD ERR BIT 2;
510
                  1
511
              } else if ((can_RxMsqs[i].mask == 0) && IS_CAN_EXTID(can_RxMsqs[i].ID)) {
512
                  /* ID List mode 32bit */
513
514
                   if (can_RxMsqs[i].fifo == CAN_FILTER_FIF00) {
515
                       numberOfDifferentIDs[ID_32BIT_FIFO0]++;
516
                   } else if (can RxMsqs[i].fifo == CAN FILTER FIFO1) {
517
                       numberOfDifferentIDs[ID 32BIT FIFO1]++;
518
                  } else {
                                                                                                  C can.c X C can_cfg.h
                                                                                           can_cfg.c
                                                                                                                           C stm32f4
519
                       /* Invalid FIFO selection; check can_RxMsqs[i].fifo value */
                                                                                           cu-common > src > driver > can > C can.c >  MSK_16BIT_FIFO0
520
                       *error |= STD ERR BIT 3;
                                                                                           59 /*======== Macros and Definitions ===
521
                                                                                           60 #define ID_16BIT_FIF00
                                                                                                                      (0U)
522
              } else if (can_RxMsqs[i].mask > 0 && IS_CAN_STDID(can_RxMsqs[i].ID)) {
                                                                                           61 #define ID 16BIT FIF01
                                                                                                                      (1U)
                  /* Mask mode 16bit */
523
                                                                                           62 #define ID_32BIT_FIF00
                                                                                                                      (2U)
524
                                                                                           63 #define ID_32BIT_FIF01
                                                                                                                      (3U)
525
                   if (can_RxMsgs[i].fifo == CAN_FILTER_FIFO0) {
                                                                                           64 #define MSK_16BIT_FIF00
                                                                                                                      (4U)
526
                       numberOfDifferentIDs[MSK_16BIT_FIFO0]++;
                                                                                           65 #define MSK_16BIT_FIF01
                                                                                                                      (5U)
527
                   } else if (can RxMsqs[i].fifo == CAN FILTER FIFO1) {
                                                                                          66 #define MSK_32BIT
                                                                                                                     (6U)
528
                       numberOfDifferentIDs[MSK 16BIT FIFO1]++;
529
                   } else {
530
                       /* Invalid FIFO selection; check can RxMsqs[i].fifo value */
531
                       *error |= STD ERR BIT 4;
532
533
              } else if ((can_RxMsqs[i].mask > 0U) && IS_CAN_EXTID(can_RxMsqs[i].ID)) {
                  /* Mask mode 32bit */
534
535
536
                   numberOfDifferentIDs[MSK_32BIT]++;
537
              } else {
538
                   /* Invalid ID > IS CAN EXTID; check can RxMsqs[i].ID value */
539
                   *error |= STD ERR BIT 5;
540
                   break;
541
              }
542
543
          for (uint8_t i = 0; i < 2U; i++) {</pre>
544
              if (numberOfDifferentIDs[i] > OU) {
545
                   retVal += (numberOfDifferentIDs[i] + 2) / 4; /* 4 IDs per filter; rounding up */
546
              }
547
          }
548
          for (uint8 t i = 2; i < 6U; i++) {</pre>
549
              if (numberOfDifferentIDs[i] > OU) {
550
                   retVal += (numberOfDifferentIDs[i] + 1) / 2; /* 2 IDs per filter; rounding up */
551
              }
552
          }
553
          if (numberOfDifferentIDs[MSK 32BIT] > 0U) {
554
                                                                   /* 1 ID per filter */
              retVal += numberOfDifferentIDs[6];
555
          }
556
          return retVal;
557
      }
558
559
       * @brief Returns the next index of wished filter ID setting in CAN_MSG_RX_TYPE_t can_RxMsgs[CAN_NUMBER_OF_RX_IDs]
560
```

```
561
562
       * @param can RxMsgs:
                                pointer to receive message struct
563
       * @param numberOfRxIDs: count of that type of receive message in can RxMsgs struct
564
       * @param startIndex:
                                index where to start searching
565
       * # @param filterCase:
                                specifies the object what will be found
566
567
       * @retval returns index
568
569
      static uint8 t CAN GetNextID (CAN MSG RX TYPE s* can RxMsgs, uint8 t numberOfRxIDs, uint8 t startIndex,
570
              uint8 t filterCase) {
571
          uint8 t retVal = 0;
                                         There can be a hidden bug—if startIndex = 0, there is risk that
572
          uint8 t i = startIndex;
                                         retVal = 0 is ambiguous.
573
          while (i < numberOfRxIDs) {</pre>
574
              if ((filterCase == ID_16BIT_FIF00 && can_RxMsgs[i].mask == 0U) && IS_CAN_STDID(can_RxMsgs[i].ID) &&
              (can_RxMsgs[i].fifo == CAN_FILTER FIF00)) {
575
                  retVal = i;
                                                   All the else if branches use the same operations; these branches
576
                  break;
                                                   can be combined using OR.
577
              } else if ((filterCase == ID_16BIT_FIF01) && (can_RxMsgs[i].mask == 0U) && IS_CAN_STDID(can_RxMsgs[i].ID) &&
              (can RxMsqs[i].fifo == CAN FILTER FIFO1)) {
578
                  retVal = i;
579
                  break;
580
              } else if ((filterCase == ID 32BIT FIF00) && (can RxMsqs[i].mask == 0U) && !IS CAN STDID(can RxMsqs[i].ID) &&
              IS CAN EXTID (can RxMsqs[i].ID) && (can RxMsqs[i].fifo == CAN FILTER FIFOO)) {
581
                  retVal = i;
582
                  break;
              } else if ((filterCase == ID 32BIT FIFO1) && (can RxMsqs[i].mask == 0U) && !IS CAN STDID(can RxMsqs[i].ID) &&
583
              IS CAN EXTID (can RxMsqs[i].ID) && (can RxMsqs[i].fifo == CAN FILTER FIFO1)) {
584
                  retVal = i;
585
                  break;
              } else if ((filterCase == MSK 16BIT FIFO0) && (can RxMsqs[i].mask > 0U) && IS CAN STDID(can RxMsqs[i].ID) &&
586
              (can RxMsqs[i].fifo == CAN FILTER FIF00)) {
587
                  retVal = i;
588
                  break;
589
              } else if ((filterCase == MSK 16BIT FIFO1) && (can RxMsqs[i].mask > 0U) && IS CAN STDID(can RxMsqs[i].ID) &&
              (can_RxMsqs[i].fifo == CAN_FILTER_FIFO1)) {
590
                  retVal = i;
591
                  break:
              } else if ((filterCase == MSK 32BIT) && (can RxMsqs[i].mask > 0U) && !IS CAN STDID(can RxMsqs[i].ID) &&
592
              IS_CAN_EXTID(can_RxMsqs[i].ID)) {
593
                  retVal = i;
594
                  break:
595
              }
596
              i++;
597
          }
598
          return retVal;
599
      1
600
      /* **********
601
602
       * Interrupt handling
       ***********
603
604
      /**
605
```

```
* @brief Transmission Mailbox 0 complete callback.
606
        * @param hcan pointer to a CAN HandleTypeDef structure that contains
607
608
                  the configuration information for the specified CAN.
609
        * @retval None
610
611
     void HAL_CAN_TxMailbox0CompleteCallback(CAN_HandleTypeDef *hcan) {
612
     #if CANO USE TX BUFFER
613
         if (hcan->Instance == CAN2) {
614
              CAN TxCpltCallback (CAN NODE0);
615
616
     #endif /* CANO_USE_TX_BUFFER */
617
     #if CAN1 USE TX BUFFER
          /* No need for callback, if no buffer is used */
618
619
         if (hcan->Instance == CAN1) {
620
             /* Transmission complete callback */
              CAN TxCpltCallback (CAN NODE1);
621
622
623
     #endif /* CAN1_USE_TX_BUFFER */
624
     }
625
626
     /**
627
       * @brief Transmission Mailbox 1 complete callback.
628
        * @param hcan pointer to a CAN HandleTypeDef structure that contains
629
                  the configuration information for the specified CAN.
630
        * @retval None
631
632
     void HAL_CAN_TxMailbox1CompleteCallback(CAN_HandleTypeDef *hcan) {
633
     #if CANO_USE_TX_BUFFER
634
          if (hcan->Instance == CAN2) {
635
              CAN_TxCpltCallback (CAN_NODE0);
636
     #endif /* CANO_USE_TX_BUFFER */
637
638
     #if CAN1 USE TX BUFFER
639
          /* No need for callback, if no buffer is used */
640
         if (hcan->Instance == CAN1) {
641
              /* Transmission complete callback */
642
              CAN_TxCpltCallback (CAN_NODE1);
643
     #endif /* CAN1_USE_TX_BUFFER */
644
645
     }
646
647
     /**
648
       * @brief Transmission Mailbox 2 complete callback.
        * @param hcan pointer to a CAN HandleTypeDef structure that contains
649
                  the configuration information for the specified CAN.
650
651
        * @retval None
652
     void HAL CAN TxMailbox2CompleteCallback(CAN HandleTypeDef *hcan) {
653
654
     #if CANO_USE_TX_BUFFER
655
          if (hcan->Instance == CAN2) {
656
              CAN_TxCpltCallback (CAN_NODE0);
657
         }
```

The next three TxMailboxnCompleteCallback functions can be wrappers of the same common function.

```
658
     #endif /* CANO USE TX BUFFER */
#if CAN1 USE TX BUFFER
660
         /* No need for callback, if no buffer is used */
661
         if (hcan->Instance == CAN1) {
662
             /* Transmission complete callback */
663
             CAN_TxCpltCallback (CAN_NODE1);
664
665
     #endif /* CAN1 USE TX BUFFER */
666
667
668
     /**
669
       * @brief Rx FIFO 0 message pending callback.
        * @param hcan pointer to a CAN_HandleTypeDef structure that contains
670
671
                 the configuration information for the specified CAN.
672
       * @retval None
673
       * /
674
     void HAL_CAN_RxFifo0MsqPendingCallback(CAN_HandleTypeDef *hcan) {
675
         /* Call callback function to interpret or save RX message in buffer */
676
         if (hcan->Instance == CAN2) {
677
             CAN RxMsq(CAN NODEO, hcan, CAN RX FIFOO); /* change towards HAL CAN IRQHandler */
678
679
         if (hcan->Instance == CAN1) {
680
             CAN RxMsq(CAN NODE1, hcan, CAN RX FIFO0); /* change towards HAL CAN IRQHandler */
681
         }
682
    }
683
     /**
684
685
       * @brief Rx FIFO 1 message pending callback.
686
        * @param hcan pointer to a CAN HandleTypeDef structure that contains
                 the configuration information for the specified CAN.
687
688
       * @retval None
689
       * /
690 void HAL CAN RxFifo1MsqPendingCallback(CAN HandleTypeDef *hcan) {
691
         /* Call callback function to interpret or save RX message in buffer */
692
         if (hcan->Instance == CAN2) {
693
             CAN_RxMsq(CAN_NODEO, hcan, CAN_RX_FIFO1);
                                                             /* change towards HAL_CAN_IRQHandler */
694
695
         if (hcan->Instance == CAN1) {
696
             CAN_RxMsq(CAN_NODE1, hcan, CAN_RX_FIFO1); /* change towards HAL_CAN_IRQHandler */
697
         }
698
     }
699
701
        * @brief Error CAN callback.
702
        * @param hcan pointer to a CAN HandleTypeDef structure that contains
703
                 the configuration information for the specified CAN.
        * @retval None
704
705
       * /
706
     void HAL_CAN_ErrorCallback(CAN_HandleTypeDef *hcan) {
707
         CAN_ERROR_s* errorStruct = NULL_PTR;
708
709
         if (hcan->Instance == CAN1) {
```

```
710
      #if CAN USE CAN NODE1 == 1
711
              errorStruct = &CAN1 errorStruct;
712
      #endif /* CAN USE CAN NODE1 == 1 */
713
         } else {
714
      #if CAN USE CAN NODE0 == 1
715
              errorStruct = &CANO_errorStruct;
716
      #endif /* CAN USE CAN NODE0 == 1 */
717
         }
718
719
          /* Check Error Warning flag and set error codes */
720
          if (errorStruct != NULL_PTR) {
721
              /* Check Error Warning Flag */
722
              if ((hcan->ErrorCode & HAL CAN ERROR EWG) != 0) {
723
                  /* This bit is set by hardware when the warning limit has been
724
                   * reached (Receive Error Counter or Transmit Error Counter>=96
725
                   * until error counter 127 write error frames dominant on can bus
                   * increment error occurrence of error warning state */
726
727
                  errorStruct->canErrorCounter[0]++;
728
             }
729
730
              /* Check Error Passive Flag */
731
              if ((hcan->ErrorCode & HAL CAN ERROR EPV) != 0) {
                  /* This bit is set by hardware when the Error Passive limit has
732
733
                   * been reached (Receive Error Counter or Transmit Error Counter
                   * > 127) write error frames recessive on can bus increment error
734
735
                   * occurrence of error passive state */
                  errorStruct->canErrorCounter[1]++;
736
737
             }
738
              /* Check Bus-Off Flag */
739
              if ((hcan->ErrorCode & HAL CAN ERROR BOF) != 0) {
740
                  /* This bit is set by hardware when it enters the bus-off state.
741
                   * The bus-off state is entered on TEC overflow, greater than 255
742
                   * increment error occurrence of bus-off state */
                  errorStruct->canErrorCounter[2]++;
743
744
              }
745
              /* Check stuff error flag */
746
              if ((hcan->ErrorCode & HAL CAN ERROR STF) != 0) {
747
                  /* When five consecutive bits of the same level have been
748
                   * transmitted by a node, it will add a sixth bit of the opposite
749
                   * level to the outgoing bit stream. The receivers will remove this
750
                   * extra bit. This is done to avoid excessive DC components on the
                   * bus, but it also gives the receivers an extra opportunity to
751
752
                   * detect errors: if more than five consecutive bits of the same
753
                   * level occurs on the bus, a Stuff Error is signaled. */
754
                  errorStruct->canErrorCounter[3]++;
755
756
              if ((hcan->ErrorCode & HAL CAN ERROR FOR) != 0) {
757
                  /* FORM ERROR --- Some parts of the CAN message have a fixed format,
758
                   * i.e. the standard defines exactly what levels must occur and
759
                   * when. (Those parts are the CRC Delimiter, ACK Delimiter, End of
760
                   * Frame, and also the Intermission, but there are some extra
                   * special error checking rules for that.) If a CAN controller
761
```

```
762
                   * detects an invalid value in one of these fixed fields, a Form
763
                   * Error is signaled. */
764
                  errorStruct->canErrorCounter[4]++;
765
766
              if ((hcan->ErrorCode & HAL CAN ERROR ACK) != 0) {
767
                  /* ACKNOWLEDGMENT ERROR --- All nodes on the bus that correctly
768
                   * receives a message (regardless of their being interested of its
                   ^{\star} contents or not) are expected to send a dominant level in the
769
                   * so-called Acknowledgement Slot in the message. The transmitter
770
771
                   * will transmit a recessive level here. If the transmitter can
772
                   * detect a dominant level in the ACK slot, an Acknowledgement
773
                   * Error is signaled. */
                  errorStruct->canErrorCounter[5]++;
774
775
776
              if ((hcan->ErrorCode & HAL CAN ERROR BR) != 0) {
777
                  /* BIT RECESSIVE ERROR --- Each transmitter on the CAN bus monitors
778
                   * (i.e. reads back) the transmitted signal level. If the bit level
                   * actually read differs from the one transmitted, a Bit Error (No
779
780
                   * bit error is raised during the arbitration process.) */
781
                  errorStruct->canErrorCounter[6]++;
782
783
              if ((hcan->ErrorCode & HAL CAN ERROR BD) != 0) {
784
                  /* BIT DOMINANT ERROR --- Each transmitter on the CAN bus monitors
785
                   * (i.e. reads back) the transmitted signal level. If the bit level
786
                   * actually read differs from the one transmitted, a Bit Error (No
                   * bit error is raised during the arbitration process.) */
787
788
                  errorStruct->canErrorCounter[7]++;
789
              1
790
              if ((hcan->ErrorCode & HAL CAN ERROR CRC) != 0) {
                  /* CRC ERROR --- Each message features a 15-bit Cyclic Redundancy
791
792
                   * Checksum (CRC), and any node that detects a different CRC in the
                   * message than what it has calculated itself will signal an CRC
793
794
                   * Error. */
795
                  errorStruct->canErrorCounter[8]++;
796
              }
797
              if ((hcan->ErrorCode & HAL_CAN_ERROR_RX_FOV0) != 0) {
798
                  /* Rx FIFOO overrun error */
799
                  errorStruct->canErrorCounter[9]++;
800
801
              if ((hcan->ErrorCode & HAL CAN ERROR RX FOV1) != 0) {
802
                  /* Rx FIFO1 overrun error */
803
                  errorStruct->canErrorCounter[10]++;
804
805
              if ((hcan->ErrorCode & HAL CAN ERROR TX ALSTO) != 0) {
                  /* TxMailbox 0 transmit failure due to arbitration lost */
806
807
                  errorStruct->canErrorCounter[11]++;
808
              }
809
              if ((hcan->ErrorCode & HAL CAN ERROR TX TERRO) != 0) {
810
                  /* TxMailbox 1 transmit failure due to transmit error */
811
                  errorStruct->canErrorCounter[12]++;
812
              }
813
              if ((hcan->ErrorCode & HAL_CAN_ERROR_TX_ALST1) != 0) {
```

```
814
                  /* TxMailbox 0 transmit failure due to arbitration lost */
815
                  errorStruct->canErrorCounter[13]++;
816
              }
817
              if ((hcan->ErrorCode & HAL CAN ERROR TX TERR1) != 0) {
818
                  /* TxMailbox 1 transmit failure due to transmit error */
819
                  errorStruct->canErrorCounter[14]++;
820
821
              if ((hcan->ErrorCode & HAL CAN ERROR TX ALST2) != 0) {
822
                  /* TxMailbox 0 transmit failure due to arbitration lost */
823
                  errorStruct->canErrorCounter[15]++;
824
825
              if ((hcan->ErrorCode & HAL CAN ERROR TX TERR2) != 0) {
                   /* TxMailbox 1 transmit failure due to transmit error */
826
827
                  errorStruct->canErrorCounter[16]++;
828
              1
              if ((hcan->ErrorCode & HAL CAN ERROR TIMEOUT) != 0) {
829
830
                  /* Timeout error */
831
                  errorStruct->canErrorCounter[17]++;
832
              }
833
              if ((hcan->ErrorCode & HAL CAN ERROR NOT INITIALIZED) != 0) {
834
                  /* Peripheral not initialized */
835
                  errorStruct->canErrorCounter[18]++;
836
837
              if ((hcan->ErrorCode & HAL_CAN_ERROR_NOT_READY) != 0) {
838
                  /* Peripheral not ready */
839
                  errorStruct->canErrorCounter[19]++;
840
841
              if ((hcan->ErrorCode & HAL_CAN_ERROR_NOT_STARTED) != 0) {
842
                  /* Peripheral not started */
843
                  errorStruct->canErrorCounter[20]++;
844
845
              if ((hcan->ErrorCode & HAL_CAN_ERROR_PARAM) != 0) {
                  /* Parameter error */
846
                  errorStruct->canErrorCounter[21]++;
847
848
              }
849
850
      }
851
852
853
       * @brief Transmission complete callback in non blocking mode
854
855
       * @param canNode: canNode that transmitted a message
856
857
       * @retval none (void)
858
                              Transmit from a mailbox is complete
859
      static void CAN TxCpltCallback (CAN NodeTypeDef e canNode) {
860
          STD RETURN TYPE e retVal = E NOT OK;
861
          CAN TX BUFFER s* can txbuffer = NULL;
862
863
          if (canNode == CAN_NODE0) {
                                                            Lines 863 to 871 can be written in a function to hide the details. This is alos
864
      #if CAN_USE_CAN_NODE0 == 1
                                                            good for code reuse.
865
              can txbuffer = &can0 txbuffer;
```

```
#endif /* #if CAN_USE_CAN_NODE0 == 1 */
866
         } else if (canNode == CAN NODE1) {
867
868
     #if CAN_USE_CAN_NODE1 == 1
869
             can txbuffer = &can1 txbuffer;
870
     #endif /* CAN USE CAN NODE1 == 1 */
871
872
         /* Transmit buffer existing, check if message is ready for transmission */
873
         if (can txbuffer != NULL) {
             /* No Error during start of transmission */
874
875
             if ((can_txbuffer->ptrWrite == can_txbuffer->ptrRead)
876
                     && (can_txbuffer->buffer[can_txbuffer->ptrRead].newMsg == 0)) {
877
                 /* nothing to transmit, buffer is empty */
                 retVal = E NOT OK;
878
879
             } else {
880
                 retVal = CAN TxMsqBuffer(canNode);
                 if (retVal != E OK) {
881
                     /* Error during transmission, retransmit message later */
882
883
                     retVal = E_NOT_OK;
884
                 }
885
             }
886
         } else {
887
             /* no transmit buffer active */
888
             retVal = E NOT OK;
889
         }
890
     }
891
892
893
      /* ***********
894
895
      * Transmit message
896
      **************
897
898
     STD RETURN TYPE e CAN TxMsq(CAN NodeTypeDef e canNode, uint32 t msqID, uint8 t* ptrMsqData, uint32 t msqLength,
899
             uint32 t RTR) {
900
         STD_RETURN_TYPE_e retVal = E_NOT_OK;
901
         CAN_TxHeaderTypeDef canMessage;
902
         CAN_HandleTypeDef *ptrHcan;
903
         uint32 t freeMailboxes = 0;
904
         uint32_t *ptrMailbox = NULL_PTR;
905
906
         if (canNode == CAN NODE0) {
907
             if (canNode0_listenonly_mode) {
908
                 ptrHcan = NULL;
             } else {
909
910
                 /* Check if at least one mailbox is free */
911
                 freeMailboxes = HAL CAN GetTxMailboxesFreeLevel(&hcan0);
912
                 if (freeMailboxes != 0) {
913
                     ptrHcan = &hcan0;
914
                 } else {
915
                     ptrHcan = NULL;
916
917
             }
```

```
918
          } else if (canNode == CAN NODE1) {
919
              if (canNode1_listenonly_mode) {
920
                   ptrHcan = NULL;
921
              } else {
922
                   /* Check if at least one mailbox is free */
923
                   freeMailboxes = HAL_CAN_GetTxMailboxesFreeLevel(&hcan1);
924
                   if (freeMailboxes != 0) {
925
                      ptrHcan = &hcan1;
926
                  } else {
927
                       ptrHcan = NULL;
928
                  }
929
              }
930
          } else {
931
              ptrHcan = NULL;
932
          1
933
934
          if ((IS_CAN_STDID(msqID) || IS_CAN_EXTID(msqID)) && IS_CAN_DLC(msqLength) && ptrHcan != NULL) {
935
              if (IS CAN STDID(msqID)) {
936
                   canMessage.StdId = msqID;
                   canMessage.IDE = CAN ID STD;
937
938
              } else {
939
                   canMessage.ExtId = msqID;
940
                   canMessage.IDE = CAN ID EXT;
941
942
              canMessage.DLC = msqLength;
943
              canMessage.RTR = RTR;
944
              canMessage.TransmitGlobalTime = DISABLE;
945
946
              /* Copy message in TX mailbox and transmit it */
              HAL CAN AddTxMessage (ptrHcan, &canMessage, ptrMsqData, ptrMailbox);
947
948
          } else {
                                                               A better way is to write another wrapper function to hide ptrMailbox, which
949
              retVal = E_NOT_OK;
                                                               is not assigned a specific value.
950
951
          return retVal;
952
      }
953
                              Send to buffer
954
955
      STD RETURN TYPE e CAN Send (CAN NodeTypeDef e canNode, uint32 t msqID, uint8 t* ptrMsqData, uint32 t msqLength,
956
              uint32 t RTR) {
957
          STD RETURN TYPE e retVal = E NOT OK;
958
          uint8 t tmptxbuffer wr = 0;
959
          CAN_TX_BUFFER_s* can_txbuffer = NULL_PTR;
960
961
          if (canNode == CAN_NODE0) {
962
      #if CAN USE CAN NODE0 == 1
963
              can txbuffer = &can0 txbuffer;
      #endif /* #if CAN USE CAN NODE0 == 1 */
964
965
          } else if (canNode == CAN NODE1) {
966
      #if CAN_USE_CAN_NODE1 == 1
967
              can_txbuffer = &can1_txbuffer;
      #endif /* #if CAN USE CAN NODE1 == 1 */
968
969
          }
```

```
970
 971
           /* Transmit buffer exisiting */
 972
          if (can txbuffer != NULL PTR) {
 973
               tmptxbuffer wr = can txbuffer->ptrWrite;
 974
 975
               if (tmptxbuffer_wr == can_txbuffer->ptrRead) {
 976
                   if (can txbuffer->buffer[tmptxbuffer wr].newMsg == 0) {
 977
                       /* free buffer space for message */
 978
 979
                       can_txbuffer->ptrWrite++;
 980
                      can_txbuffer->ptrWrite = can_txbuffer->ptrWrite % can_txbuffer->length;
 981
                      retVal = E OK;
 982
                  } else {
                      /* buffer full */
 983
 984
                      retVal = E NOT OK;
 985
                  }
 986
               } else {
 987
                  can txbuffer->ptrWrite++;
 988
                   can txbuffer->ptrWrite = can txbuffer->ptrWrite % can txbuffer->length;
 989
                   retVal = E OK;
 990
              }
 991
          } else {
 992
              retVal = E NOT OK;
 993
          }
 994
 995
          if ((can txbuffer != NULL PTR) &&
 996
               (retVal == E OK) &&
 997
               998
               (IS CAN DLC (msqLength))) {
999
               /* if buffer free and valid CAN identifier */
1000
1001
               if (IS CAN STDID(msqID)) {
1002
                   can txbuffer->buffer[tmptxbuffer wr].msq.StdId = msqID;
1003
                   can txbuffer->buffer[tmptxbuffer wr].msg.IDE = 0;
1004
               } else {
                  can txbuffer->buffer[tmptxbuffer_wr].msg.ExtId = msgID;
1005
1006
                   can txbuffer->buffer[tmptxbuffer wr].msq.IDE = 1;
1007
              }
               can txbuffer->buffer[tmptxbuffer wr].newMsg = 1;
1008
1009
               can txbuffer->buffer[tmptxbuffer wr].msq.RTR = RTR;
1010
               can txbuffer->buffer[tmptxbuffer wr].msq.DLC = msqLength;
                                                                         /* Data length of the frame that will be
               transmitted */
1011
               can_txbuffer->buffer[tmptxbuffer_wr].msg.TransmitGlobalTime = DISABLE;
1012
1013
               /* copy message data in handle transmit structure */
1014
               can txbuffer->buffer[tmptxbuffer wr].data[0] = ptrMsqData[0];
                                                                               A for loop works better
1015
               can txbuffer->buffer[tmptxbuffer wr].data[1] = ptrMsqData[1];
1016
               can txbuffer->buffer[tmptxbuffer wr].data[2] = ptrMsqData[2];
1017
               can_txbuffer->buffer[tmptxbuffer_wr].data[3] = ptrMsqData[3];
1018
               can_txbuffer->buffer[tmptxbuffer_wr].data[4] = ptrMsqData[4];
1019
               can_txbuffer->buffer[tmptxbuffer_wr].data[5] = ptrMsgData[5];
1020
               can txbuffer->buffer[tmptxbuffer wr].data[6] = ptrMsqData[6];
```

```
1021
               can txbuffer->buffer[tmptxbuffer wr].data[7] = ptrMsqData[7];
1022
1023
              retVal = E OK;
1024
           } else {
1025
               retVal = E NOT OK;
1026
1027
1028
           return retVal;
1029
      }
1030
1031
1032
       STD RETURN TYPE e CAN TxMsqBuffer (CAN NodeTypeDef e canNode) {
           STD RETURN TYPE e retVal = E NOT OK;
1033
1034
           HAL StatusTypeDef retHal = HAL ERROR;
           CAN TX BUFFER s* can txbuffer = NULL;
1035
1036
           CAN HandleTypeDef* ptrHcan = NULL;
1037
           uint32 t freeMailboxes = 0;
           uint32 t *ptrMailbox = NULL PTR;
1038
1039
1040
           if (canNode == CAN NODE0) {
1041
       #if CAN_USE_CAN_NODE0 == 1
1042
               if (!canNode0 listenonly mode) {
1043
                   can txbuffer = &can0 txbuffer;
1044
                   ptrHcan = &hcan0;
1045
       #endif /* #if CAN USE CAN NODE0 == 1 */
1046
           } else if (canNode == CAN NODE1) {
1047
1048
       #if CAN USE CAN NODE1 == 1
1049
               if (!canNodel listenonly mode) {
1050
                   can txbuffer = &can1 txbuffer;
1051
                   ptrHcan = &hcan1;
1052
1053
       #endif /* CAN USE CAN NODE1 == 1 */
1054
1055
           /* Check if at least one mailbox is free */
1056
           freeMailboxes = HAL_CAN_GetTxMailboxesFreeLevel(&hcan0);
1057
1058
           if ((can txbuffer != NULL) && (freeMailboxes != 0)) {
               if ((can txbuffer->ptrWrite == can txbuffer->ptrRead)
1059
1060
                       && (can txbuffer->buffer[can txbuffer->ptrRead].newMsg == 0)) {
                   /* nothing to transmit, buffer is empty */
1061
                   retVal = E NOT OK;
1062
1063
               } else {
1064
                   /* Copy message into TX mailbox and transmit it */
1065
                   retHal = HAL CAN AddTxMessage(ptrHcan, &can txbuffer->buffer[can txbuffer->ptrRead].msq,
                   can txbuffer->puffer[can txbuffer->ptrRead].data, ptrMailbox);
1066
1067
                   if (retHal == HAL OK) {
1068
                       /* No Error during start of transmission */
                       can txbuffer->buffer[can_txbuffer->ptrRead].newMsg = 0; /* Msg is sent, set newMsg to 0, to allow
1069
                       writing of new data in buffer space */
                       can txbuffer->ptrRead++;
1070
```

```
1071
                          can txbuffer->ptrRead = can txbuffer->ptrRead % can txbuffer->length;
1072
                          retVal = E OK:
1073
                     } else {
1074
                          retVal = E NOT OK;
                                                        /* Error during transmission, retransmit message later */
1075
1076
                 }
1077
            } else {
                 /* no transmit buffer active or TX mailboxes full */
1078
1079
                 retVal = E NOT OK;
1080
1081
            return retVal;
1082
        }
1083
1084
1085
            Receive message
         ************
1086
1087
1088
        /**
1089
         * @brief Receives CAN messages and stores them either in RxBuffer or in hcan
1090
1091
         * @param canNode:
                                  canNode which received the message
1092
         * @param ptrHcan:
                                  pointer to a CAN HandleTypeDef structure that contains
1093
                                  the message information of the specified CAN.
1094
         * @param FIFONumber: FIFO in which the message has been received
1095
         * @retval none (void)
1096
1097
        static void CAN RxMsq(CAN NodeTypeDef e canNode, CAN HandleTypeDef* ptrHcan, uint8 t FIFONumber) {
1098
            uint8_t bypassLinkIndex = 0;
                                                           C can_cfg.c
                                                                       C can.h
                                                                               X C can.c
                                                                                               C can_cfg.h
                                                                                                       C can_cfq.c
                                                                                                                   C can.h X C can.c
                                                                                                                                           C can_cfg.h
            CAN_RX_BUFFERELEMENT_s tmpMsqBuffer;
1099
1100
            uint32 t msqID = 0;
                                                           mcu-common > src > driver > can > C can.h > •○ CAN_RX_BUFFER_s
                                                                                                       mcu-common > src > driver > can > C can.h > •○ CAN_RX_BUFFER_s
1101
                                                                                                            typedef struct CAN_RX_BUFFER {
                                                                typedef struct CAN_RX_BUFFERELEMENT {
        #if CANO USE_RX_BUFFER || CAN1_USE_RX_BUFFER 101
1102
                                                                                                        113
                                                                                                                uint8_t ptrRead;
                                                                    CAN RxHeaderTypeDef msg;
1103
            uint32_t* can_bufferbypass_rxmsqs = NULL; 102
                                                                                                        114
                                                                                                                uint8_t ptrWrite;
                                                                    uint8_t data[8];
                                                                                                        115
                                                                                                                uint8_t length;
1104
            uint32 t bufferbypasslength = 0;
                                                            103
                                                                    uint8_t newMsg;
                                                                                                        116
                                                                                                                CAN_RX_BUFFERELEMENT_s* buffer;
                                                            104 } CAN_RX_BUFFERELEMENT_s;
1105
            CAN_RX_BUFFER_s* can_rxbuffer = NULL;
                                                                                                        117
                                                                                                             } CAN RX BUFFER s;
1106
            CAN_MSG_RX_TYPE_s* can_rxmsqs = NULL;
1107
            uint8 t* can fastLinkIndex = NULL;
                                                                                   C cansignal_cfq.h
                                                                                                  C can_cfq.h × C stm32f4xx_hal_can.h
                                                                                                                               C cansignal_cfg.c
                                                                                                                                              C
        #endif /* CANO USE RX BUFFER || CAN1 USE RX BUFFER */
1108
                                                                                    mcu-primary > src > driver > config > C can_cfg.h > •○ CAN_MSG_RX_TYPE_s
1109
1110
            /* Set pointer on respective RxBuffer */
                                                                                    297
                                                                                         typedef struct CAN_MSG_RX_TYPE {
1111
            if (canNode == CAN NODE1) {
                                                                                    298
                                                                                            uint32_t ID; /*!< message ID */</pre>
                                                                                    299
                                                                                            uint32_t mask; /*!< mask or 0x0000 to select list mode */</pre>
1112
        #if CAN1_USE_RX_BUFFER && CAN_USE_CAN_NODE1 == 1
                                                                                    300
                                                                                            uint8 t DLC; /*!< data length */</pre>
1113
                 can rxbuffer = &can1 rxbuffer;
                                                                                    301
                                                                                            uint8 t RTR; /*!< rtr bit */
1114
        #if (CAN1 BUFFER BYPASS NUMBER OF IDs > 0u)
                                                                                    302
                                                                                            uint32_t fifo; /*!< selected CAN hardware (CAN FILTER FIF00 or CAN F
1115
                 can rxmsqs = &can1 RxMsqs[0];
                                                                                    303
                                                                                            STD_RETURN_TYPE_e (*func)(uint32_t ID, uint8_t*, uint8_t, uint8_t);
1116
                 can bufferbypass rxmsqs = &can1 bufferBypass RxMsqs[0];
                                                                                        } CAN MSG RX TYPE s:
1117
                 bufferbypasslength = CAN1 BUFFER BYPASS NUMBER OF IDs;
1118
                 can fastLinkIndex = &can1 fastLinkIndex[0];
1119
        #endif /* (CAN1_BUFFER_BYPASS_NUMBER_OF_IDs > 0u) */
1120
        #endif /* CAN1_USE_RX_BUFFER && CAN_USE_CAN_NODE1 == 1 */
1121
            } else if (canNode == CAN NODE0) {
1122
        #if CANO USE RX BUFFER && CAN USE CAN NODEO == 1
```

```
can rxbuffer = &can0 rxbuffer;
1123
1124
       #if (CANO BUFFER BYPASS NUMBER OF IDs > 0U)
1125
               can rxmsqs = &can0 RxMsqs[0];
                                                         See Line 175.
1126
                can bufferbypass rxmsqs = &can0 bufferBypass RxMsqs[0];
1127
               bufferbypasslength = CANO BUFFER BYPASS NUMBER OF IDs;
1128
                can_fastLinkIndex = &can0_fastLinkIndex[0]; See Line 90.
1129
       #endif /* (CANO BUFFER BYPASS NUMBER OF IDs > 0u) */
       #endif /* CANO USE RX BUFFER && CAN USE CAN NODE0 == 1 */
1130
1131
           }
1132
1133
           /* Get message ID */
1134
           HAL CAN GetRxMessage (ptrHcan, FIFONumber, &tmpMsqBuffer.msg , &tmpMsqBuffer.data[0]);
1135
1136
       #if (CAN1 BUFFER BYPASS NUMBER OF IDs > 0) || (CAN0 BUFFER BYPASS NUMBER OF IDs > 0)
1137
           if (can bufferbypass rxmsqs != NULL) {
1138
                /* only needed when messages are bypassed */
1139
                for (bypassLinkIndex = 0; bypassLinkIndex < bufferbypasslength; bypassLinkIndex++) {</pre>
1140
1141
                    if ((tmpMsqBuffer.msq.StdId == can bufferbypass rxmsqs[bypassLinkIndex]) ||
1142
                        (tmpMsqBuffer.msq.ExtId == can bufferbypass rxmsqs[bypassLinkIndex])) {
1143
                        break;
                                       Only match the ID in the received message with the ID in the Buffer Bypass array.
1144
                    }
1145
               }
                                      No match found in the previous step, which means that we need to put the data from
                                      tmpMsgBuffer to the buffer.
1146
       #endif /* #if (CAN1_BUFFER_BYPASS_NUMBER_OF_IDs > 0) || (CAN0_BUFFER_BYPASS_NUMBER_OF_IDs > 0) */
1147
1148
           if (bypassLinkIndex >= bufferbypasslength && can rxbuffer != NULL) {
1149
                /* ##### Use buffer / Copy data in buffer ##### */
1150
1151
       #if CANO USE RX BUFFER || CAN1 USE RX BUFFER
1152
                /* NO NEED TO DISABLE INTERRUPTS, BECAUSE FUNCTION IS CALLED FROM ISR */
1153
1154
                /* Set to 1 to mark message as new received. Set to 0 when reading message from buffer */
1155
                can rxbuffer->buffer[can rxbuffer->ptrWrite].newMsq = 1;
1156
                                                                   Note that msg is a struct and we can copy a struct in C directly.
1157
                /* Get message header */
                can_rxbuffer->buffer[can_rxbuffer->ptrWrite].msg = tmpMsqBuffer.msg;
1158
1159
1160
               /* Get the data field */
                for (uint8 t i = 0; i < can rxbuffer->buffer[can rxbuffer->ptrWrite].msq.DLC; i++) {
1161
1162
                    can rxbuffer->buffer[can rxbuffer->ptrWrite].data[i] = tmpMsqBuffer.data[i];
1163
               }
1164
               /* Increment write pointer */           Bypass the buffer
1165
1166
               can rxbuffer->ptrWrite++;
               can_rxbuffer->ptrWrite = can_rxbuffer->ptrWrite % can rxbuffer->length;
1167
1168
       #endif /* CANO USE RX BUFFER || CAN1 USE RX BUFFER */
           } else if (bypassLinkIndex <√bufferbypasslength && can rxmsgs != NULL && can fastLinkIndex != NULL) {
1169
1170
                /* ##### Buffer active but bypassed ##### */
1171
1172
       #if ((CAN1_BUFFER_BYPASS_NUMBER_OF_IDs > 0) || (CAN0_BUFFER_BYPASS_NUMBER_OF_IDs > 0)) && ((CAN_USE_CAN_NODE0 == 1)
       | (CAN USE CAN NODE1 == 1)) This #if is buggy.
                /* call buffer bypass callback function */
1173
```

```
if (tmpMsqBuffer.msq.IDE == 0U) {
1174
1175
                   if (can rxmsqs[can fastLinkIndex[bypassLinkIndex]].func != NULL) {
1176
                        can_rxmsqs[can_fastLinkIndex[bypassLinkIndex]].func(tmpMsqBuffer.msq.StdId,
1177
                                                                             tmpMsqBuffer.data,
1178
                                                                              tmpMsqBuffer.msq.DLC,
1179
                                                                              tmpMsqBuffer.msq.RTR);
1180
                   } else {
1181
                        /* No callback function defined */
1182
                       CAN BufferBypass (canNode, tmpMsqBuffer.msq.StdId, tmpMsqBuffer.data, tmpMsqBuffer.msq.DLC,
                       tmpMsqBuffer.msq.RTR);
                                                    See Line 1295
1183
                   }
               } else {
1184
                   if (can rxmsqs[can fastLinkIndex[bypassLinkIndex]].func != NULL) {
1185
1186
                       can_rxmsqs[can_fastLinkIndex[bypassLinkIndex]].func(tmpMsqBuffer.msq.ExtId,
1187
                                                                              tmpMsqBuffer.data,
1188
                                                                              tmpMsqBuffer.msq.DLC,
1189
                                                                              tmpMsqBuffer.msq.RTR);
1190
                   } else {
1191
                        /* No callback function defined */
1192
                       CAN BufferBypass (canNode, tmpMsqBuffer.msq.ExtId, tmpMsqBuffer.data, tmpMsqBuffer.msq.DLC,
                       tmpMsqBuffer.msq.RTR);
1193
                   }
1194
               }
1195
       #endif /* ((CAN1_BUFFER_BYPASS_NUMBER_OF_IDs > 0) || (CAN0_BUFFER_BYPASS_NUMBER_OF_IDs > 0)) && ((CAN_USE_CAN_NODE0)
       == 1) || (CAN_USE_CAN_NODE1 == 1)) */
1196
           } else {
                        This is the case never happens for this project.
1197
               /* ##### Buffer not active ##### */
1198
1199
               CAN_MSG_RX_TYPE_s* msqRXstruct;
1200
               uint8 t length;
1201
               if (canNode == CAN NODE0) {
1202
                   msgRXstruct = &can0 RxMsgs[0];
1203
                   length = can CANO rx length;
1204
               } else {
1205
                   msgRXstruct = &can1_RxMsgs[0];
1206
                   length = can_CAN1_rx_length;
1207
               }
1208
1209
               if (tmpMsqBuffer.msq.IDE == 0U) {
1210
                  msqID = tmpMsqBuffer.msq.StdId;
1211
               } else {
1212
                   msqID = tmpMsqBuffer.msq.ExtId;
1213
               }
1214
1215
               /* Search for correct message in RX message array to check for callback */
1216
               uint8 t rxMsq = 0;
1217
               for (; rxMsq < length; rxMsq++) {</pre>
1218
                   if (msqRXstruct[rxMsq].ID == msqID) {
1219
                       break;
1220
                   }
1221
               }
1222
```

```
/* Interpret received message */
1223
                if (msqRXstruct[rxMsql.func != NULL) {
1224
1225
                     msgRXstruct[rxMsg].func(msgID, &tmpMsgBuffer.data[0], tmpMsgBuffer.msg.DLC, tmpMsgBuffer.msg.RTR);
1226
                } else {
1227
                     CAN_InterpretReceivedMsg(canNode, msgID, &tmpMsgBuffer.data[0], tmpMsgBuffer.msg.DLC,
                     tmpMsgBuffer.msg.RTR);
                                                  See Line 1351
                                                                                                                           C can_cfg.h X
                                                                                                                 C can.c
1228
                }
                                                                                                        mcu-primary > src > driver > config > C can_cfg.h > • O C
1229
            }
                                                                                                        325 typedef struct CanPdu {
1230
       }
                                                                                                              uint8_t sdu[8];
1231
                                                                                                              uint32_t id;
1232
                            CAN ReadReceiveBuffer
                                                                                                              uint8_t dlc;
                                                                                                        329 } Can_PduType;
1233
       STD RETURN TYPE e CAN ReceiveBuffer (CAN NodeTypeDef e canNode, Can PduType* msq) {
            /* E OK is returned, if buffer is empty and interpret function is called successful */
1234
1235
            STD RETURN TYPE e retVal = E NOT OK;
1236
                                                                                                 C can.h × C can.c
                                                                                     C can_cfg.c
                                                                                                                        C can cfa.h
1237
       #if CANO USE RX BUFFER || CAN1 USE RX BUFFER
            CAN RX BUFFER s* can rxbuffer = NULL;
1238
                                                                                      mcu-common > src > driver > can > C can.h > •○ CAN_RX_BUFFER_s
1239
                                                                                          typedef struct CAN_RX_BUFFER {
1240
       #if CANO USE RX BUFFER && CAN USE CAN NODEO == 1
                                                                                      113
                                                                                              uint8 t ptrRead;
                                                                                      114
                                                                                              uint8_t ptrWrite;
1241
            if (canNode == CAN NODE0) {
                                                                                      115
                                                                                              uint8_t length;
1242
                can_rxbuffer = &can0_rxbuffer;
                                                                                      116
                                                                                              CAN_RX_BUFFERELEMENT_s* buffer;
1243
            1
                                                                                      117
                                                                                          } CAN_RX_BUFFER_s;
1244
       #endif /* CANO USE RX BUFFER && CAN USE CAN NODE0 == 1 */
1245
       #if CAN1_USE_RX_BUFFER && CAN_USE_CAN_NODE1 == 1
1246
            if (canNode == CAN NODE1) {
                                                                                      C can_cfq.c
                                                                                                 C can.h
                                                                                                         X C can.c
                                                                                                                        C can_cfq.h
1247
                can rxbuffer = &can1 rxbuffer;
                                                                                      mcu-common > src > driver > can > C can.h > •○ CAN_RX_BUFFER_s
1248
                                                                                          typedef struct CAN RX BUFFERELEMENT {
1249
       #endif /* CAN1 USE RX BUFFER && CAN USE CAN NODE1 == 1 */
                                                                                      101
                                                                                             CAN RxHeaderTypeDef msg;
1250
                                                                                      102
                                                                                              uint8_t data[8];
1251
            if (msq == NULL) {
                                                                                      103
                                                                                             uint8_t newMsg;
1252
                 /* null pointer to message data struct */
                                                                                      104 } CAN_RX_BUFFERELEMENT_s;
1253
                can rxbuffer = NULL;
1254
            }
1255
1256
            if (dcan_rxbuffer->ptrWrite != can_rxbuffer->ptrRead) &&
1257
                     (can_rxbuffer->buffer[can_rxbuffer->ptrRead].newMsg == 1) &&
1258
                     /(can rxbuffer != NULL)) {
                 /* buffer not empty -> read message */
1259
                if (can_rxbuffer->buffer[can_rxbuffer->ptrRead].msq.IDE == 1) {
1260
1261
                     /* Extended ID used */
1262
                     msq->id = can rxbuffer->buffer[can_rxbuffer->ptrRead].msq.ExtId;
1263
                } else {
1264
                     msg->id = can_rxbuffer->buffer[can_rxbuffer->ptrRead].msg.StdId;
1265
                }
1266
                msq->dlc = can rxbuffer->buffer[can rxbuffer->ptrRead].msq.DLC;
1267
1268
                for (uint8 t i = 0; i < 8U; i++) {</pre>
1269
                     msq->sdu[i] = can rxbuffer->buffer[can rxbuffer->ptrRead].data[i];
1270
                }
1271
1272
                 /* Set to 0 to mark buffer entry as read. Set to 1 when writing message into buffer */
1273
                can rxbuffer->buffer[can rxbuffer->ptrRead].newMsq = 0;
```

```
1274
               /* Move to next buffer element */
1275
1276
               can rxbuffer->ptrRead++;
1277
               can rxbuffer->ptrRead = can rxbuffer->ptrRead % can rxbuffer->length;
1278
               retVal = E OK;
1279
           }
1280
       #endif /* CANO_USE_RX_BUFFER || CAN1_USE_RX_BUFFER */
1281
           return retVal;
1282
       }
1283
1284
1285
        * @brief Receives a bypassed CAN message and interprets it
1286
        * @param canNode: canNode on which the message has been received
1287
1288
        * @param msqID:
                          message ID
        * @param data:
                          pointer to the message data
1289
1290
        * @param DLC:
                          length of received data
1291
        * @param RTR:
                          RTR bit of received message
1292
1293
        * @retval E OK if interpreting was successful, otherwise E NOT OK
1294
1295
       static STD RETURN TYPE e CAN BufferBypass (CAN NodeTypeDef e canNode, uint32 t msqID, uint8 t* rxData, uint8 t DLC,
1296
               uint8 t RTR) {
1297
           STD_RETURN_TYPE_e retVal = E_OK;
1298
           /* ********************
1299
1300
               Implement wished functionality of received messages here,
1301
1302
            * if no callback function in CAN MSG RX TYPE s struct is defined
            *******************************
1303
1304
1305
           /* Perform SW reset */
1306
           if (msqID == CAN ID SOFTWARE RESET MSG && DLC == 8) {
1307
               uint8 t reset = 0;
1308
1309
               /* CAN data = FF FF FF FF FF FF FF */
1310
               for (uint8 t i = 0; i < DLC; i++) {
                   if (rxData[i] != 0xFF) {
1311
                                              C can_cfq.c
                                                        C can.c
                                                                   C can_cfq.h × C stm32f4xx_l
1312
                       reset = 1;
1313
                                              mcu-primary > src > driver > config > C can_cfg.h >  C CAN_SW_RESET_W
                   }
                                              294 /↑ #dejine CAN_SW_KESEI_WITH_DEVICE_ID
1314
                                              295 #define CAN_SW_RESET_WITH_DEVICE_ID
1315
       #if CAN_SW_RESET_WITH_DEVICE_ID == 1
1316
       /*
                  CAN data = MCU Device ID Byte [0] [1] [2] [3] [4] [5] [6] [7] */
       /*
                  if (rxData[0] == (uint8_t)mcu_unique_deviceID.off0 && data[1] == (uint8_t)(mcu_unique_deviceID.off0 >> 8) &&
1317
                       rxData[2] == (uint8 t) (mcu unique deviceID.off0 >> 16) && rxData[3] ==
1318
                       (uint8 t) (mcu unique deviceID.off0 >> 24) &&
1319
                       rxData[4] == (uint8 t) mcu unique deviceID.off32 && rxData[5] == (uint8 t) (mcu unique deviceID.off32
                       >> 8) &&
1320
                       rxData[6] == (uint8_t) (mcu_unique_deviceID.off32 >> 16) && rxData[7] ==
                       (uint8_t) (mcu_unique_deviceID.off32 >> 24)) {
1321
                   reset = 1:
1322
```

```
1323
      * /
1324
1325
              if (rxData[0] == 0) {
1326
                  if ((CAN_CheckNodeID(&data[5]) == E_OK) || (CAN_CheckUniqueDeviceID(&data[1]) == E_OK) ||
                  (CAN CheckBroadcastID(&data[5]) == E OK)) {
1327
                     reset = 1;
1328
1329
              }
1330
      #else /* CAN SW RESET WITH DEVICE ID != 1 */
1331
              reset = 1;
                                  This line makes the operation in line 1312 useless.
1332
      #endif /* CAN_SW_RESET_WITH_DEVICE_ID == 1 */
1333
1334
              if (reset == 1)
1335
                 HAL_NVIC_SystemReset();
1336
1337
          return retVal;
1338
      }
1339
      /**
1340
1341
       * @brief Interprets the received message
1342
1343
       * @param canNode: canNode on which the message has been received
1344
       * @param msqID: message ID
       * @param data: pointer to the message data
1345
1346
       * @param DLC: length of received data
1347
       * @param RTR: RTR bit of received message
1348
1349
       * @return E_OK if interpretation successful, otherwise E_NOT_OK
1350
       * /
1351
      static STD RETURN TYPE e CAN InterpretReceivedMsg (CAN NodeTypeDef e canNode, uint32 t msgID, uint8 t* data, uint8 t
                              Called in Line 1227.
1352
              uint8_t RTR) {
1353
          STD RETURN TYPE e retVal = E NOT OK;
1354
1355
          /* **********************
1356
1357
              Implement wished functionality of received messages here,
1358
1359
           * if no callback function in CAN_MSG_RX_TYPE_s struct is defined
1360
                      ******************
1361
          return retVal;
1362
      }
1363
      /* ***********
1364
1365
       * Sleep mode
       ************
1366
1367
1368
      void CAN SetSleepMode(CAN NodeTypeDef e canNode) {
          if (canNode == CAN_NODE0) {
1369
1370
              HAL_CAN_RequestSleep(&hcan0);
1371
          } else if (canNode == CAN_NODE1) {
1372
              HAL CAN RequestSleep (&hcan1);
```

```
1373
1374
          return;
1375
     }
1376
1377
1378
      void CAN_WakeUp(CAN_NodeTypeDef_e canNode) {
1379
          if (canNode == CAN_NODE0) {
1380
              HAL_CAN_WakeUp(&hcan0);
          } else if (canNode == CAN_NODE1) {
1381
1382
              HAL_CAN_WakeUp(&hcan1);
1383
1384
          return;
1385
      }
1386
```