Anexo A. Circuitos Impresos

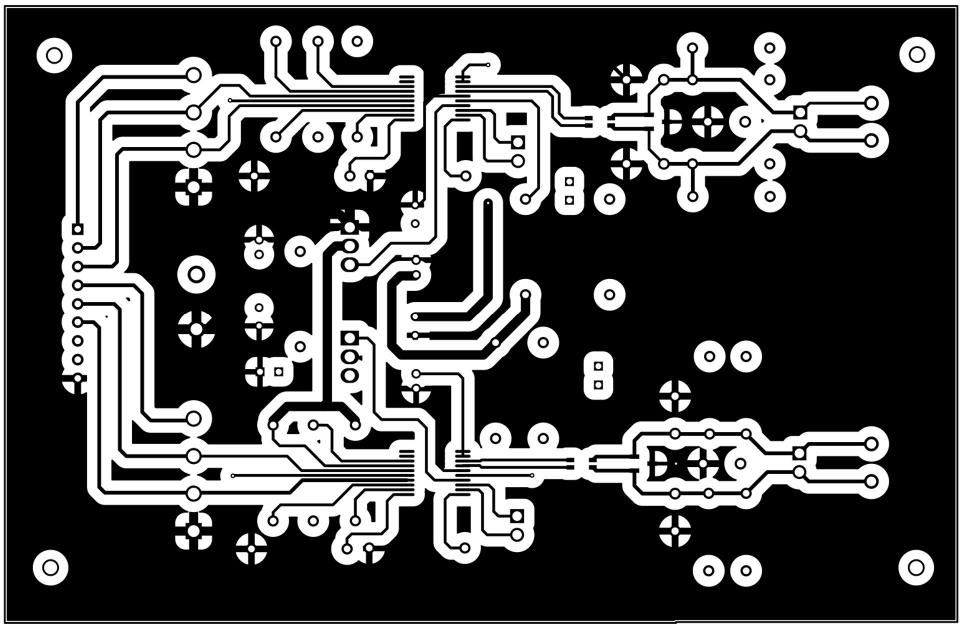

Figura A: 1. BD capa superior.

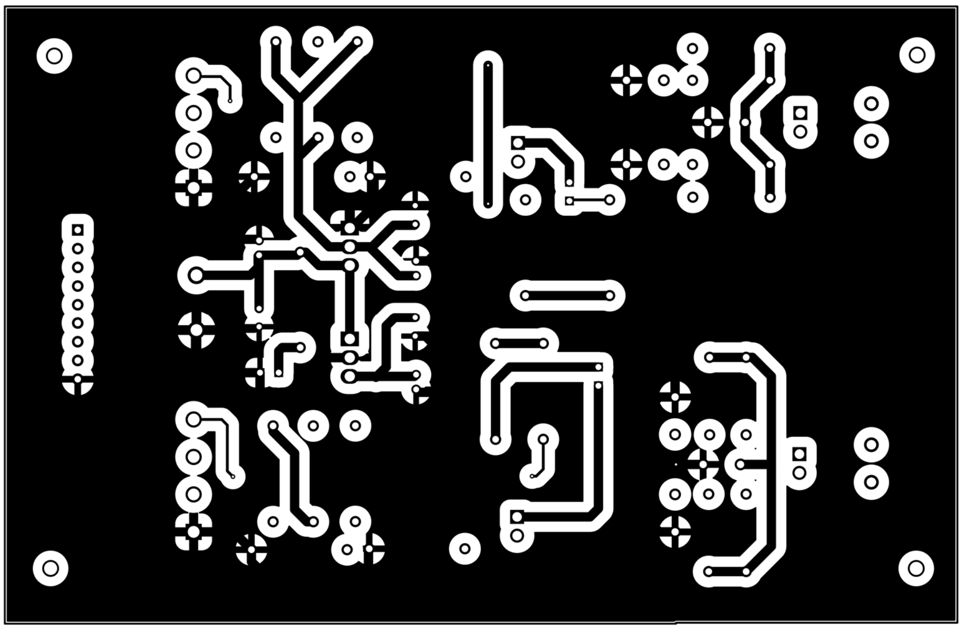

Figura A: 2. BD capa inferior.

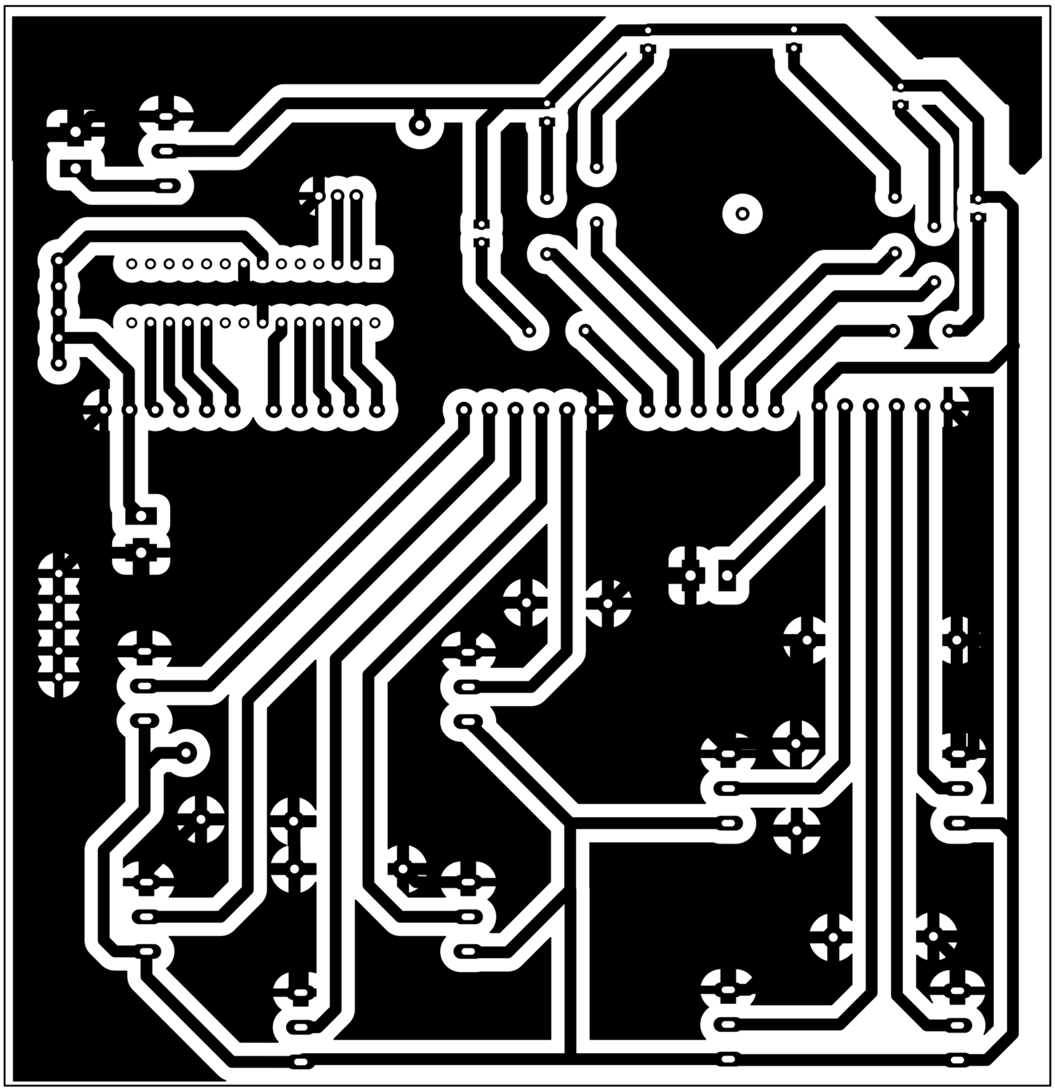

Figura A: 3. Panel de sensores.

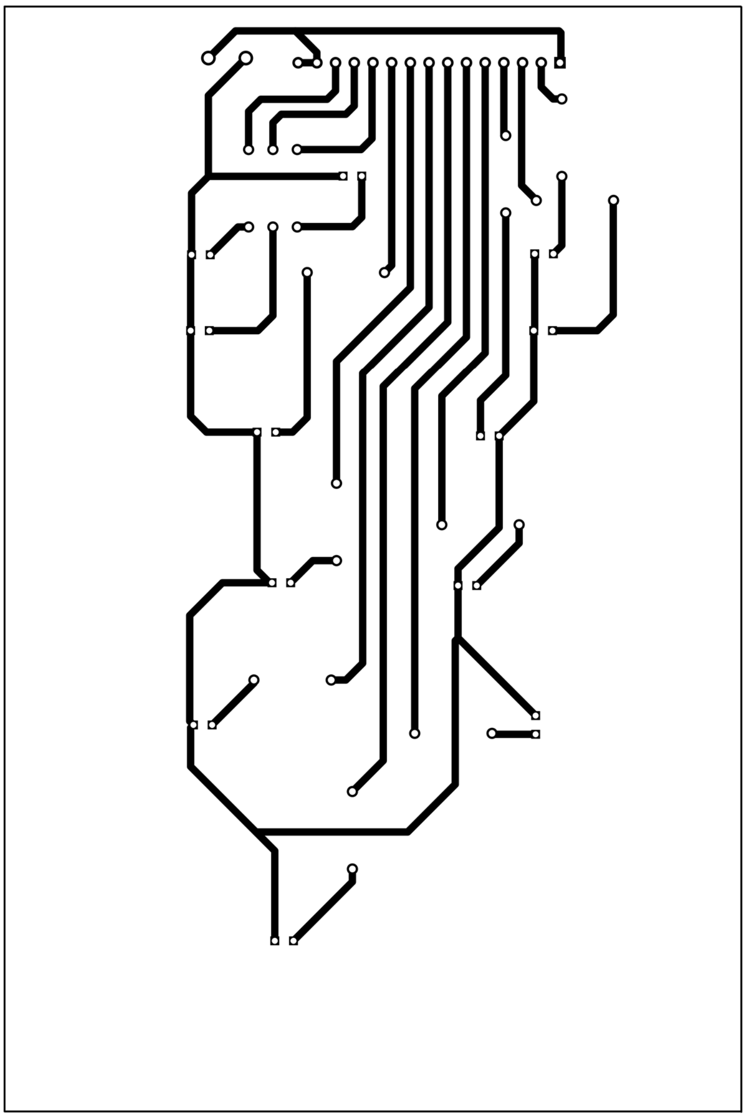

Figura A: 4. Panel de indicadores.

Anexo B. Código biblioteca FlexRay(fray.h)

```
#ifndef __FRAY_H__
#define __FRAY_H__

// CMD constants (SUCC1)

#define CMD_command_not_accepted            0x0
#define CMD_CONFIG                          0x1
#define CMD_READY                           0x2
#define CMD_WAKEUP                          0x3
#define CMD_RUN                             0x4
#define CMD_ALL_SLOTS                       0x5
#define CMD_HALT                            0x6
#define CMD_FREEZE                          0x7
#define CMD_SEND_MTS                        0x8
#define CMD_ALLOW_COLDSTART                 0x9
#define CMD_RESET_STATUS_INDICATORS         0xA
#define CMD_MONITOR_MODE                    0xB
#define CMD_CLEAR_RAMS                      0xC
#define CMD_LOOPBACK MODE                   0xF

#include <stdint.h>
#define FLEX_PAYLOAD 64

#define slot_Stup 0;
#define slot1 1;
#define slot2 2;
#define slot3 3;
#define slot4 4;
#define mslot10 10;
#define mslot11 11;
#define mslot12 12;
#define mslot13 13;
#define mslot14 14;

typedef volatile struct fray_registers //Communication Controller Registers
{
   /* ----------------------------------------------------------------------- */
   /* FRAY_ST                                                                  */
   /*  Definition of the FLEXRAY register map                                  */

   /* ECC Control Register */
   /* 0x0 */
   union ECC_CTRL
   {
      unsigned long ECC_CTRL_UL;
      struct
      {
         unsigned :4;
         unsigned SBE_EVT_EN :4; //ECC Single-Bit Error Indication.
```

```
        unsigned :4;
        unsigned SBEL_B4 :4; //ECC Single-Bit Error Lock
        unsigned :12;
        unsigned DIAGSEL_B4 :4; //Diagnostic Mode select Key

    } ECC_CTRL_ST;
} ECC_CTRL_UN;

/* ECC Diagnostic Status Register */
/* 0x4 */
union ECCDSTAT
{
    unsigned long ECCDSTAT_UL;
    struct
    {
        unsigned :8;
        unsigned DEFH_B1 :1;
        unsigned DEFG_B1 :1;
        unsigned DEFF_B1 :1;
        unsigned DEFE_B1 :1;
        unsigned DEFD_B1 :1;
        unsigned DEFC_B1 :1;
        unsigned DEFB_B1 :1;
        unsigned DEFA_B1 :1;
        unsigned :8;
        unsigned SEFH_B1 :1;
        unsigned SEFG_B1 :1;
        unsigned SEFF_B1 :1;
        unsigned SEFE_B1 :1;
        unsigned SEFD_B1 :1;
        unsigned SEFC_B1 :1;
        unsigned SEFB_B1 :1;
        unsigned SEFA_B1 :1;

    } ECCDSTAT_ST;
} ECCDSTAT_UN;

/* ECC Test Register */
/* 0x8 */
union ECCTEST
{
    unsigned long ECCTEST_UL;
    struct
    {
        unsigned :9;
        unsigned RDECC_B7 :7;
        unsigned :9;
        unsigned WRECC_B7 :7;

    } ECCTEST_ST;
} ECCTEST_UN;

/* Single-Bit Error Status Register */
/* 0xC */
union SBESTAT
{
   unsigned long SBESTAT_UL;
   struct
   {
      unsigned SBE :1;
      unsigned :16;
      unsigned FMB :7;
      unsigned :1;
      unsigned MFMB :1;
      unsigned FMBD :1;
      unsigned STBF2 :1;
      unsigned STBF1 :1;
      unsigned SMR :1;
      unsigned SOBF :1;
      unsigned SIBF :1;
   } SBESTAT_ST;
} SBESTAT_UN;

/*  Special Registers                                        */
/* Test Register 1                                           */
/* 0x10 */
union test1
{
   unsigned long TEST1_UL;
   struct
   {
      unsigned CERB_B4 :4;   //Coding Error Report Channel B.
      unsigned CERA_B4 :4;   //Coding Error Report Channel A
      unsigned :2;
      unsigned txenb_B1 :1; /* Control of Channel B Transmit Enable Pin    */
      unsigned txena_B1 :1; /* Control of Channel A Transmit Enable Pin    */
      unsigned txb_B1 :1; /* Control of Channel B Transmit Pin          */
      unsigned txa_B1 :1; /* Control of Channel A Transmit Pin          */
      unsigned rxb_B1 :1; /* Monitor Channel B Receive Pin              */
      unsigned rxa_B1 :1; /* Monitor Channel A Receive Pin              */
      unsigned :6;
      unsigned AOB_B1 :1;   //Activity on B
      unsigned AOA_B1 :1;   //Activity on A
      unsigned :2;
      unsigned tmc_B2 :2; /* Test Mode Control                          */
      unsigned :2;
      unsigned ELBE_B1 :1;   //External Loop Back Enable.
      unsigned wrten_B1 :1; /* Write Test Register Enable                 */
   } TEST1_ST;
} TEST1_UN;

/* Test Register 2                                           */
/* 0x14 */
union test2
{
   unsigned long TEST2_UL;
```

```
        struct
        {
            unsigned :16;
            unsigned rdpb_B1 :1; //When ECC mode is enabled, this bit is always read
as 0
            unsigned wrpb_B1 :1; //When ECC mode is enabled, this bit has no effect.
            unsigned :7;
            unsigned ssel_B3 :3;   //Segment select.
            unsigned :1;
            unsigned rs_B3 :3;   //RAM select
        } TEST2_ST;
    } TEST2_UN;

    unsigned :32;

    /* Lock Register                                                    */
    /* 0x1C */
    union lck
    {
        unsigned long LCK_UL;
        struct
        {
            unsigned :16;  //Reserved
            unsigned tmk_B8 :8;   //Test mode key.
            unsigned clk_B8 :8;   //Configuration lock key
        } LCK_ST;
    } LCK_UN;

    /*  Interrupt Registers                                              */
    /* Error Input Register                                              */
    /* 0x20 */
    union eir
    {
        unsigned long EIR_UL;
        struct
        {
            unsigned :5;
            unsigned tabb_B1 :1;
            unsigned ltvb_B1 :1;
            unsigned edb_B1 :1;
            unsigned :5;
            unsigned taba_B1 :1;
            unsigned ltva_B1 :1;
            unsigned eda_B1 :1;
            unsigned :4;
            unsigned mhf_B1 :1;
            unsigned ioba_B1 :1;
            unsigned iiba_B1 :1;
            unsigned efa_B1 :1;
            unsigned rfo_B1 :1;
            unsigned perr_B1 :1;
            unsigned ccl_B1 :1;
            unsigned ccf_B1 :1;
        unsigned sfo_B1 :1;
        unsigned sfbm_B1 :1;
        unsigned cna_B1 :1;
        unsigned pemc_B1 :1;
     } EIR_ST;
  } EIR_UN;

  /* Status Interrupt Register                                   */
  /* 0x24 */
  union sir
  {
     unsigned long SIR_UL;
     struct
     {
        unsigned :6;
        unsigned mtsb_B1 :1;
        unsigned wupb_B1 :1;
        unsigned :6;
        unsigned mtsa_B1 :1;
        unsigned wupa_B1 :1;
        unsigned sds_B1 :1;
        unsigned mbsi_B1 :1;
        unsigned sucs_B1 :1;
        unsigned swe_B1 :1;
        unsigned tobc_B1 :1;
        unsigned tibc_B1 :1;
        unsigned ti1_B1 :1;
        unsigned ti0_B1 :1;
        unsigned nmvc_B1 :1;
        unsigned rfcl_B1 :1;
        unsigned rfne_B1 :1;
        unsigned rxi_B1 :1;
        unsigned txi_B1 :1;
        unsigned cycs_B1 :1;
        unsigned cas_B1 :1;
        unsigned wst_B1 :1;
     } SIR_ST;
  } SIR_UN;

  /* Error Interrupt Line Select                                 */
  /* 0x28 */
  union eils
  {
     unsigned long EILS_UL;
     struct
     {
        unsigned :5;
        unsigned tabbl_B1 :1;
        unsigned ltvbl_B1 :1;
        unsigned edbl_B1 :1;
        unsigned :5;
        unsigned tabal_B1 :1;
        unsigned ltval_B1 :1;
```

```
            unsigned edal_B1 :1;
            unsigned :4;
            unsigned mhfl_B1 :1;
            unsigned iobal_B1 :1;
            unsigned iibal_B1 :1;
            unsigned efal_B1 :1;
            unsigned rfol_B1 :1;
            unsigned perrl_B1 :1;
            unsigned ccll_B1 :1;
            unsigned ccfl_B1 :1;
            unsigned sfol_B1 :1;
            unsigned sfbml_B1 :1;
            unsigned cnal_B1 :1;
            unsigned pemcl_B1 :1;
        } EILS_ST;
    } EILS_UN;

    /* Status Interrupt Line Select                                   */
    /* 0x2C */
    union sils
    {
        unsigned long SILS_UL;
        struct
        {
            unsigned :6;
            unsigned mtsbl_B1 :1;
            unsigned wupbl_B1 :1;
            unsigned :6;
            unsigned mtsal_B1 :1;
            unsigned wupal_B1 :1;
            unsigned sdsl_B1 :1;
            unsigned mbsil_B1 :1;
            unsigned sucsl_B1 :1;
            unsigned swel_B1 :1;
            unsigned tobcl_B1 :1;
            unsigned tibcl_B1 :1;
            unsigned ti1l_B1 :1;
            unsigned ti0l_B1 :1;
            unsigned nmvcl_B1 :1;
            unsigned rffl_B1 :1;
            unsigned rfnel_B1 :1;
            unsigned rxil_B1 :1;
            unsigned txil_B1 :1;
            unsigned cycsl_B1 :1;
            unsigned casl_B1 :1;
            unsigned wstl_B1 :1;
        } SILS_ST;
    } SILS_UN;

    /* Error Interrupt Enable Set                                     */
    /* 0x30 */
    union eies
    {
        unsigned long EIES_UL;
        struct
        {
            unsigned :5;
            unsigned tabbe_B1 :1;
            unsigned ltvbe_B1 :1;
            unsigned edbe_B1 :1;
            unsigned :5;
            unsigned tabae_B1 :1;
            unsigned ltvae_B1 :1;
            unsigned edae_B1 :1;
            unsigned :4;
            unsigned mhfe_B1 :1;
            unsigned iobae_B1 :1;
            unsigned iibae_B1 :1;
            unsigned efae_B1 :1;
            unsigned rfoe_B1 :1;
            unsigned perre_B1 :1;
            unsigned ccle_B1 :1;
            unsigned ccfe_B1 :1;
            unsigned sfoe_B1 :1;
            unsigned sfbme_B1 :1;
            unsigned cnae_B1 :1;
            unsigned pemce_B1 :1;
        } EIES_ST;
    } EIES_UN;

    /* Error Interrupt Enable Reset                                   */
    /* 0x34 */
    union eier
    {
        unsigned long EIER_UL;
        struct
        {
            unsigned :5;
            unsigned tabbe_B1 :1;
            unsigned ltvbe_B1 :1;
            unsigned edbe_B1 :1;
            unsigned :5;
            unsigned tabae_B1 :1;
            unsigned ltvae_B1 :1;
            unsigned edae_B1 :1;
            unsigned :4;
            unsigned mhfe_B1 :1;
            unsigned iobae_B1 :1;
            unsigned iibae_B1 :1;
            unsigned efae_B1 :1;
            unsigned rfoe_B1 :1;
            unsigned perre_B1 :1;
            unsigned ccle_B1 :1;
            unsigned ccfe_B1 :1;
            unsigned sfoe_B1 :1;
            unsigned sfbme_B1 :1;
        unsigned cnae_B1 :1;
        unsigned pemce_B1 :1;
    } EIER_ST;
} EIER_UN;

/* Status Interrupt Enable Set                                    */
/* 0x38 */
union sies
{
    unsigned long SIES_UL;
    struct
    {
        unsigned :6;
        unsigned mtsbe_B1 :1;
        unsigned wupbe_B1 :1;
        unsigned :6;
        unsigned mtsae_B1 :1;
        unsigned wupae_B1 :1;
        unsigned sdse_B1 :1;
        unsigned mbsie_B1 :1;
        unsigned sucse_B1 :1;
        unsigned swee_B1 :1;
        unsigned tobce_B1 :1;
        unsigned tibce_B1 :1;
        unsigned ti1e_B1 :1;
        unsigned ti0e_B1 :1;
        unsigned nmvce_B1 :1;
        unsigned rffe_B1 :1;
        unsigned rfnee_B1 :1;
        unsigned rxie_B1 :1;
        unsigned txie_B1 :1;
        unsigned cycse_B1 :1;
        unsigned case_B1 :1;
        unsigned wste_B1 :1;
    } SIES_ST;
} SIES_UN;

/* Status Interrupt Enable Reset                                  */
/* 0x3C */
union sier
{
    unsigned long SIER_UL;
    struct
    {
        unsigned :6;
        unsigned mtsbe_B1 :1;
        unsigned wupbe_B1 :1;
        unsigned :6;
        unsigned mtsae_B1 :1;
        unsigned wupae_B1 :1;
        unsigned sdse_B1 :1;
        unsigned mbsie_B1 :1;
        unsigned sucse_B1 :1;
        unsigned swee_B1 :1;
        unsigned tobce_B1 :1;
        unsigned tibce_B1 :1;
        unsigned ti1e_B1 :1;
        unsigned ti0e_B1 :1;
        unsigned nmvce_B1 :1;
        unsigned rffe_B1 :1;
        unsigned rfnee_B1 :1;
        unsigned rxie_B1 :1;
        unsigned txie_B1 :1;
        unsigned cycse_B1 :1;
        unsigned case_B1 :1;
        unsigned wste_B1 :1;
    } SIER_ST;
} SIER_UN;

/* Interrupt Line Enable                                        */
/* 0x40 */
union ile
{
    unsigned long ILE_UL;
    struct
    {
        unsigned :30;
        unsigned eint1_B1 :1;
        unsigned eint0_B1 :1;
    } ILE_ST;
} ILE_UN;

/* Timer 0 Configuration                                        */
/* 0x44 */
union t0c
{
    unsigned long T0C_UL;
    struct
    {
        unsigned :2;
        unsigned t0mo_B14 :14;
        unsigned :1;
        unsigned t0cc_B7 :7;
        unsigned :6;
        unsigned t0ms_B1 :1;
        unsigned t0rc_B1 :1;
    } T0C_ST;
} T0C_UN;

/* Timer 1 Configuration                                        */
/* 0x48 */
union t1c
{
    unsigned long T1C_UL;
    struct
    {
        unsigned :2;
        unsigned t1mc_B14 :14;
        unsigned :14;
        unsigned t1ms_B1 :1;
        unsigned t1rc_B1 :1;
    } T1C_ST;
} T1C_UN;

/* Stop Watch Register 1                                          */
/* 0x4C */
union stpw1
{
    unsigned long STPW1_UL;
    struct
    {
        unsigned :2;
        unsigned smtv_B14 :14;
        unsigned :2;
        unsigned sccv_B6 :6;
        unsigned :1;
        unsigned eint1_B1 :1;
        unsigned eint0_B1 :1;
        unsigned eetp_B1 :1;
        unsigned sswt_B1 :1;
        unsigned edge_B1 :1;
        unsigned swms_B1 :1;
        unsigned eswt_B1 :1;
    } STPW1_ST;
} STPW1_UN;

/* Stop Watch Register 2                                          */
/* 0x50 */
union stpw2
{
    unsigned long STPW2_UL;
    struct
    {
        unsigned :5;
        unsigned SSCVB :11;
        unsigned :5;
        unsigned SSCVA :11;
    } STPW2_ST;
} STPW2_UN;

unsigned long RES1[11]; //Reserved Interrupt Registers

/*  CC Control Registers                                         */
/* SUC Configuration Register 1                                    */
/* 0x80 */
union succ1
{
    unsigned long SUCC1_UL;
    struct
    {
        unsigned :4; //MSB    Reserved
        unsigned cchb_B1 :1; //Connected to channel B
        unsigned ccha_B1 :1; //Connected to channel A
        unsigned mtsb_B1 :1; //Select channel A for MTS Transmission B
        unsigned mtsa_B1 :1; //Select channel A for MTS Transmission A
        unsigned hcse_B1 :1; //Halt due to clock sync error
        unsigned tsm_B1 :1; //Transmission slot mode
        unsigned wucs_B1 :1; //Wakeup channel select
        unsigned pta_B5 :5; //Passive to active
        unsigned csa_B5 :5; //Cold start attempts
        unsigned :1; //Reserved
        unsigned txsy_B1 :1; //Transmit sync frame in key slot
        unsigned txst_B1 :1; //Transmit startup frame in key slot
        unsigned pbsy_B1 :1; //POC busy
        unsigned :3; //Reserved
        unsigned cmd_B4 :4; //LSB The controller host interface command vector
    } SUCC1_ST;
} SUCC1_UN;

/* SUC Configuration Register 2                                    */
/* 0x84 */
union succ2
{
    unsigned long SUCC2_UL;
    struct
    {
        unsigned :4;
        unsigned ltn_B4 :4;
        unsigned :3;
        unsigned lt_B21 :21;
    } SUCC2_ST;
} SUCC2_UN;

/* SUC Configuration Register 3                                    */
/* 0x88 */
union succ3
{
    unsigned long SUCC3_UL;
    struct
    {
        unsigned :24;
        unsigned wcf_B4 :4;
        unsigned wcp_B4 :4;
    } SUCC3_ST;
} SUCC3_UN;

/* NEM Configuration Register                                      */
/* 0x8C */
union nemc
{
    unsigned long NEMC_UL;
    struct
```

```
      {
         unsigned :28;
         unsigned nml_B4 :4;
      } NEMC_ST;
   } NEMC_UN;

   /* PRT Configuration Register 1                                   */
   /* 0x90 */
   union prtc1
   {
      unsigned long PRTC1_UL;
      struct
      {
         unsigned rwp_B6 :6;
         unsigned :1;
         unsigned rxw_B9 :9;
         unsigned brp_B2 :2;
         unsigned spp_B2 :2;
         unsigned :1;
         unsigned casm_B7 :7;
         unsigned tsst_B4 :4;
      } PRTC1_ST;
   } PRTC1_UN;

   /* PRT Configuration Register 2                                   */
   /* 0x94 */
   union prtc2
   {
      unsigned long PRTC2_UL;
      struct
      {
         unsigned :2;
         unsigned txl_B6 :6;
         unsigned txi_B8 :8;
         unsigned :2;
         unsigned rxl_B6 :6;
         unsigned :2;
         unsigned rxi_B6 :6;
      } PRTC2_ST;
   } PRTC2_UN;

   /* MHD Configuration Register                                      */
   /* 0x98 */
   union mhdc
   {
      unsigned long MHDC_UL;
      struct
      {
         unsigned :3;
         unsigned slt_B13 :13;
         unsigned :9;
         unsigned sfdl_B7 :7;
      } MHDC_ST;
```

```
} MHDC_UN;

unsigned :32;

/* GTU Configuration Register 1                                */
/* 0xA0 */
union gtuc1
{
   unsigned long GTUC1_UL;
   struct
   {
      unsigned :12;
      unsigned ut_B20 :20;
   } GTUC1_ST;
} GTUC1_UN;

/* GTU Configuration Register 2                                */
/* 0xA4 */
union gtuc2
{
   unsigned long GTUC2_UL;
   struct
   {
      unsigned :12;
      unsigned snm_B4 :4; //Sync node max (in frames).
      unsigned :2;
      unsigned mpc_B14 :14; //Macrotick per cycle (in macroticks).
   } GTUC2_ST;
} GTUC2_UN;

/* GTU Configuration Register 3                                */
/* 0xA8 */
union gtuc3
{
   unsigned long GTUC3_UL;
   struct
   {
      unsigned :1;
      unsigned miob_B7 :7;
      unsigned :1;
      unsigned mioa_B7 :7;
      unsigned uiob_B8 :8;
      unsigned uioa_B8 :8;
   } GTUC3_ST;
} GTUC3_UN;

/* GTU Configuration Register 4                                */
/* 0xAC */
union gtuc4
{
   unsigned long GTUC4_UL;
   struct
   {
        unsigned :2;
        unsigned ocs_B14 :14;
        unsigned :2;
        unsigned nit_B14 :14;
    } GTUC4_ST;
} GTUC4_UN;

/* GTU Configuration Register 5                                    */
/* 0xB0 */
union gtuc5
{
    unsigned long GTUC5_UL;
    struct
    {
        unsigned dec_B8 :8;
        unsigned :3;
        unsigned cdd_B5 :5;
        unsigned dcb_B8 :8;
        unsigned dca_B8 :8;
    } GTUC5_ST;
} GTUC5_UN;

/* GTU Configuration Register 6                                    */
/* 0xB4 */
union gtuc6
{
    unsigned long GTUC6_UL;
    struct
    {
        unsigned :5;
        unsigned mod_B11 :11;
        unsigned :5;
        unsigned asr_B11 :11;
    } GTUC6_ST;
} GTUC6_UN;

/* GTU Configuration Register 7                                    */
/* 0xB8 */
union gtuc7
{
    unsigned long GTUC7_UL;
    struct
    {
        unsigned :6;
        unsigned nss_B10 :10;
        unsigned :6;
        unsigned ssl_B10 :10;
    } GTUC7_ST;
} GTUC7_UN;

/* GTU Configuration Register 8                                    */
/* 0xBC */
union gtuc8
```

```
{
   unsigned long GTUC8_UL;
   struct
   {
      unsigned :3;
      unsigned nms_B13 :13;
      unsigned :10;
      unsigned msl_B6 :6;
   } GTUC8_ST;
} GTUC8_UN;

/* GTU Configuration Register 9                                    */
/* 0xC0 */
union gtuc9
{
   unsigned long GTUC9_UL;
   struct
   {
      unsigned :14;
      unsigned dsi_B2 :2;
      unsigned :3;
      unsigned mapo_B5 :5;
      unsigned :2;
      unsigned apo_B6 :6;
   } GTUC9_ST;
} GTUC9_UN;

/* GTU Configuration Register 10                                   */
/* 0xC4 */
union gtuc10
{
   unsigned long GTUC10_UL;
   struct
   {
      unsigned :5;
      unsigned mrc_B11 :11;
      unsigned :2;
      unsigned moc_B14 :14;
   } GTUC10_ST;
} GTUC10_UN;

/* GTU Configuration Register 11                                   */
/* 0xC8 */
union gtuc11
{
   unsigned long GTUC11_UL;
   struct
   {
      unsigned :5;
      unsigned erc_B3 :3;
      unsigned :5;
      unsigned eoc_B3 :3;
      unsigned :6;
```

```
        unsigned ercc_B2 :2;
        unsigned :6;
        unsigned ecc_B2 :2;
      } GTUC11_ST;
   } GTUC11_UN;

   /*    // BGS Configuration Register
    // 0xCC
    union bgs
    {
    unsigned long BGS_UL;
    struct
    {
    unsigned :22;
    unsigned dse_B1 :1;
    unsigned bgd_B1 :1;
    unsigned :2;
    unsigned bgt_B6 :6;
    } BGS_ST;
    } BGS_UN;

    unsigned long RES2[12]; // Reserved
    */
   unsigned long RES2[13]; // Reserved Communication Controller Status
Registers

   /* CC Status Registers                                          */
   /* CC Status Vector                                            */
   /* 0x100 */
   union ccsv
   {
      unsigned long CCSV_UL;
      struct
      {
         unsigned :2;
         unsigned psl_B6 :6;
         unsigned rca_B5 :5;
         unsigned wsv_B3 :3;
         unsigned :1;
         unsigned csi_B1 :1;
         unsigned csai_B1 :1;
         unsigned csni_B1 :1;
         unsigned :2;
         unsigned slm_B2 :2;
         unsigned hrq_B1 :1;
         unsigned fsi_B1 :1;
         unsigned pocs_B6 :6;
      } CCSV_ST;
   } CCSV_UN;

   /* CC Error Vector                                            */
   /* 0x104 */
   union ccev
```

```
{
   unsigned long CCEV_UL;
   struct
   {
      unsigned :19;
      unsigned ptac_B5 :5;
      unsigned errm_B2 :2;
      unsigned :2;
      unsigned ccfc_B4 :4;
   } CCEV_ST;
} CCEV_UN;

unsigned :32;
unsigned :32;

/* Slot Counter Value                                          */
/* 0x110 */
union scv
{
   unsigned long SCV_UL;
   struct
   {
      unsigned :5;
      unsigned sccb_B11 :11;
      unsigned :5;
      unsigned scca_B11 :11;
   } SCV_ST;
} SCV_UN;

/* Macrotick and Cycle Counter Value                                */
/* 0x114 */
union mtccv
{
   unsigned long MTCCV_UL;
   struct
   {
      unsigned :10;
      unsigned ccv_B6 :6;
      unsigned :2;
      unsigned mtv_B14 :14;
   } MTCCV_ST;
} MTCCV_UN;

/* Rate Correction Value                                       */
/* 0x118 */
union rcv
{
   unsigned long RCV_UL;
   struct
   {
      unsigned :20;
      unsigned rcv_B12 :12;
   } RCV_ST;
} RCV_UN;

/* Offset Correction Value                                  */
/* 0x11C */
union ocv
{
   unsigned long OCV_UL;
   struct
   {
      unsigned :12;
      unsigned ocv_B14 :20;
   } OCV_ST;
} OCV_UN;

/* Sync Frame Status                                        */
/* 0x120 */
union sfs
{
   unsigned long SFS_UL;
   struct
   {
      unsigned :12;
      unsigned rclr_B1 :1;
      unsigned mrcs_B1 :1;
      unsigned olcr_B1 :1;
      unsigned mocs_B1 :1;
      unsigned vsbo_B4 :4;
      unsigned vsbe_B4 :4;
      unsigned vsao_B4 :4;
      unsigned vsae_B4 :4;
   } SFS_ST;
} SFS_UN;

/* Symbol Windows and NIT Status                                */
/* 0x124 */
union swnit
{
   unsigned long SWNIT_UL;
   struct
   {
      unsigned :20;
      unsigned sbnb_B1 :1;
      unsigned senb_B1 :1;
      unsigned sbna_B1 :1;
      unsigned sena_B1 :1;
      unsigned mtsb_B1 :1;
      unsigned mtsa_B1 :1;
      unsigned tcsb_B1 :1;
      unsigned sbsb_B1 :1;
      unsigned sesb_B1 :1;
      unsigned tcsa_B1 :1;
      unsigned sbsa_B1 :1;
      unsigned sesa_B1 :1;
```

```
    } SWNIT_ST;
} SWNIT_UN;

/* Aggregated Channel Status                                */
/* 0x128 */
union acs
{
    unsigned long ACS_UL;
    struct
    {
        unsigned :19;
        unsigned sbvb_B1 :1;
        unsigned cib_B1 :1;
        unsigned cedb_B1 :1;
        unsigned sedb_B1 :1;
        unsigned vfrb_B1 :1;
        unsigned :3;
        unsigned sbva_B1 :1;
        unsigned cia_B1 :1;
        unsigned ceda_B1 :1;
        unsigned seda_B1 :1;
        unsigned vfra_B1 :1;
    } ACS_ST;
} ACS_UN;

unsigned :32;

/* Even Sync ID [1..15]                                     */
/* 0x130 .. 0x168 */
unsigned long ESID_UL[15];

unsigned :32;

/* Odd Sync ID [1..15]                                      */
/* 0x170 .. 0x1A8 */
unsigned long OSID_UL[15];

unsigned :32;

/* Network Management Vector 1                              */
/* 0x230 */
union nmv1
{
    unsigned long NMV1_UL;
} NMV1_UN;

/* Network Management Vector 2                              */
/* 0x234 */
union nmv2
{
    unsigned long NMV2_UL;
} NMV2_UN;
```

```
/* Network Management Vector 3                                          */
/* 0x238 */
unsigned long NMV3_UL;

unsigned long RES3[81]; /* Reserved                                    */

/* Message Buffer Control Registers                                     */
/* Message RAM Configuration                                            */
/* 0x300 */
union mrc
{
   unsigned long MRC_UL;
   struct
   {
      unsigned :5;
      unsigned splm_B1 :1;
      unsigned sec_B2 :2;
      unsigned lcb_B8 :8;
      unsigned ffb_B8 :8;
      unsigned fdb_B8 :8;
   } MRC_ST;
} MRC_UN;

/* FIFO Rejection Filter                                        */
/* 0x304 */
union frf
{
   unsigned long FRF_UL;
   struct
   {
      unsigned :7;
      unsigned rnf :1;
      unsigned rss :1;
      unsigned cyf_B7 :7;
      unsigned :3;
      unsigned fid_B11 :11;
      unsigned ch_B2 :2;
   } FRF_ST;
} FRF_UN;

/* FIFO Rejection Filter Mask                                       */
/* 0x308 */
union frfm
{
   unsigned long FRFM_UL;
   struct
   {
      unsigned :19;
      unsigned mfid_B11 :11;
      unsigned :2;
   } FRFM_ST;
} FRFM_UN;
```

```
unsigned long FCL;

/* Message Buffer Status Registers                                    */

/* Message Handler Status                                             */
/* 0x310 */
union mhds
{
   unsigned long MHDS_UL;
   struct
   {
      unsigned :1;
      unsigned mbu_B7 :7;
      unsigned :1;
      unsigned mbt_B7 :7;
      unsigned :1;
      unsigned fmb_B7 :7;
      unsigned cram_B1 :1;
      unsigned mfmb_B1 :1;
      unsigned fmbd_B1 :1;
      unsigned ptbf2_B1 :1;
      unsigned ptbf1_B1 :1;
      unsigned pmr_B1 :1;
      unsigned pobf_B1 :1;
      unsigned pibf_B1 :1;
   } MHDS_ST;
} MHDS_UN;

unsigned :32;

/* FIFO Status Register                                    */
/* 0x318 */
union fsr
{
   unsigned long FSR_UL;
   struct
   {
      unsigned :16;
      unsigned rffl_B8 :8;
      unsigned :5;
      unsigned rfo_B1 :1;
      unsigned rfcl_B1 :1;
      unsigned rfne_B1 :1;
   } FSR_ST;
} FSR_UN;

/* Message Handler Constraints Flags                                  */
/* 0x31C */
union mhdf
{
   unsigned long MHDF_UL;
   struct
   {
```

```
            unsigned :23;
            unsigned wahp_B1 :1;
            unsigned :2;
            unsigned tbfb_B1 :1;
            unsigned tbfa_B1 :1;
            unsigned fnfb_B1 :1;
            unsigned fnfa_B1 :1;
            unsigned snub_B1 :1;
            unsigned snua_B1 :1;
        } MHDF_ST;
    } MHDF_UN;

    /* Transmission Request Register 1                                 */
    /* 0x320 */
    union txrq1
    {
        unsigned long TXRQ1_UL;
    } TXRQ1_UN;

    /* Transmission Request Register 2                                 */
    /* 0x324 */
    union txrq2
    {
        unsigned long TXRQ2_UL;
    } TXRQ2_UN;

    /* Transmission Request Register 3                                 */
    /* 0x328 */
    union txrq3
    {
        unsigned long TXRQ3_UL;
    } TXRQ3_UN;

    /* Transmission Request Register 4                                 */
    /* 0x32C */
    union txrq4
    {
        unsigned long TXRQ4_UL;
    } TXRQ4_UN;

    /* New Data Register 1                                           */
    /* 0x330 */
    union ndat1
    {
        unsigned long NDAT1_UL;
    } NDAT1_UN;

    /* New Data Register 2                                           */
    /* 0x334 */
    union ndat2
    {
        unsigned long NDAT2_UL;
    } NDAT2_UN;
```

```
/* New Data Register 3                                        */
/* 0x338 */
union ndat3
{
   unsigned long NDAT3_UL;
} NDAT3_UN;

/* New Data Register 4                                        */
/* 0x33c */
union ndat4
{
   unsigned long NDAT4_UL;
} NDAT4_UN;

/* Message Buffer Status Changed 1                                 */
/* 0x340 */
union mbsc1
{
   unsigned long MBSC1_UL;
} MBSC1_UN;

/* Message Buffer Status Changed 2                                 */
/* 0x344 */
union mbsc2
{
   unsigned long MBSC2_UL;
} MBSC2_UN;

/* Message Buffer Status Changed 3                                 */
/* 0x348 */
union mbsc3
{
   unsigned long MBSC3_UL;
} MBSC3_UN;

/* Message Buffer Status Changed 4                                 */
/* 0x34C */
union mbsc4
{
   unsigned long MBSC4_UL;
} MBSC4_UN;

unsigned long RES4[44]; /* Reserved                           */

/* Input Buffer                                          */
/* Write Data Section [1..64]                                  */
/* 0x400 .. 0x4FC */
unsigned long WRDS[64];

/* Write Header Section 1                                     */
/* 0x500 */
union wrhs1
```

```
{
   unsigned long WRHS1_UL;
   struct
   {
      unsigned :2;
      unsigned mbi_B1 :1;
      unsigned txm_B1 :1;
      unsigned ppit_B1 :1;
      unsigned cfg_B1 :1;
      unsigned chb_B1 :1;
      unsigned cha_B1 :1;
      unsigned :1;
      unsigned cyc_B7 :7;
      unsigned :5;
      unsigned fid_B11 :11;
   } WRHS1_ST;
} WRHS1_UN;

/* Write Header Section 2                                 */
/* 0x504 */
union wrhs2
{
   unsigned long WRHS2_UL;
   struct
   {
      unsigned :9;
      unsigned pl_B7 :7;
      unsigned :5;
      unsigned crc_B11 :11;
   } WRHS2_ST;
} WRHS2_UN;

/* Write Header Section 3                                 */
/* 0x508 */
union wrhs3
{
   unsigned long WRHS3_UL;
   struct
   {
      unsigned :21;
      unsigned dp_B11 :11;
   } WRHS3_ST;
} WRHS3_UN;

unsigned :32;

/* Input Buffer Command Mask                                  */
/* 0x510 */
union ibcm
{
   unsigned long IBCM_UL;
   struct
   {
        unsigned :13;
        unsigned stxrs_B1 :1;
        unsigned ldss_B1 :1;
        unsigned lhss_B1 :1;
        unsigned :13;
        unsigned stxrh_B1 :1;
        unsigned ldsh_B1 :1;
        unsigned lhsh_B1 :1;
    } IBCM_ST;
} IBCM_UN;

/* Input Buffer Command Request                                    */
/* 0x514 */
union ibcr
{
    unsigned long IBCR_UL;
    struct
    {
        unsigned ibsys_B1 :1;
        unsigned :8;
        unsigned ibrs_B7 :7;
        unsigned ibsyh_B1 :1;
        unsigned :8;
        unsigned ibrh_B7 :7;
    } IBCR_ST;
} IBCR_UN;

unsigned long RES5[58]; /* Reserved                          */

/* Output Buffer                                             */
/* Read Data Section [1..64]                                      */
/* 0x600 .. 0x6FC */
unsigned long RDDS[64];

/* Read Header Section 1                                          */
/* 0x700 */
union rdhs1
{
    unsigned long RDHS1_UL;
    struct
    {
        unsigned :2;
        unsigned mbi_B1 :1;
        unsigned txm_B1 :1;
        unsigned ppit_B1 :1;
        unsigned cfg_B1 :1;
        unsigned chb_B1 :1;
        unsigned cha_B1 :1;
        unsigned :1;
        unsigned cyc_B7 :7;
        unsigned :5;
        unsigned fid_B11 :11;
    } RDHS1_ST;
} RDHS1_UN;

/* Read Header Section 2                                    */
/* 0x704 */
union rdhs2
{
   unsigned long RDHS2_UL;
   struct
   {
      unsigned :1;
      unsigned plr_B7 :7;
      unsigned :1;
      unsigned plc_B7 :7;
      unsigned :5;
      unsigned crc_B11 :11;
   } RDHS2_ST;
} RDHS2_UN;

/* Read Header Section 3                                    */
/* 0x708 */
union rdhs3
{
   unsigned long RDHS3_UL;
   struct
   {
      unsigned :2;
      unsigned res_B1 :1;
      unsigned ppi_B1 :1;
      unsigned nfi_B1 :1;
      unsigned syn_B1 :1;
      unsigned sfi_B1 :1;
      unsigned rci_B1 :1;
      unsigned :2;
      unsigned rcc_B6 :6;
      unsigned :5;
      unsigned dp_B11 :11;
   } RDHS3_ST;
} RDHS3_UN;

/* Message Buffer Status                                    */
/* 0x70C */
union mbs
{
   unsigned long MBS_UL;
   struct
   {
      unsigned :2;
      unsigned ress_B1 :1;
      unsigned ppis_B1 :1;
      unsigned nfis_B1 :1;
      unsigned syns_B1 :1;
      unsigned sfis_B1 :1;
      unsigned rcis_B1 :1;
```

```
         unsigned :2;
         unsigned ccs_B6 :6;
         unsigned ftb_B1 :1;
         unsigned fta_B1 :1;
         unsigned :1;
         unsigned mlst_B1 :1;
         unsigned esb_B1 :1;
         unsigned esa_B1 :1;
         unsigned tcib_B1 :1;
         unsigned tcia_B1 :1;
         unsigned svob_B1 :1;
         unsigned svoa_B1 :1;
         unsigned ceob_B1 :1;
         unsigned ceoa_B1 :1;
         unsigned seob_B1 :1;
         unsigned seoa_B1 :1;
         unsigned vfrb_B1 :1;
         unsigned vfra_B1 :1;
      } MBS_ST;
   } MBS_UN;

   /* Output Buffer Command Mask                                   */
   /* 0x710 */
   union obcm
   {
      unsigned long OBCM_UL;
      struct
      {
         unsigned :14;
         unsigned rdsh_B1 :1;
         unsigned rhsh_B1 :1;
         unsigned :14;
         unsigned rdss_B1 :1;
         unsigned rhss_B1 :1;
      } OBCM_ST;
   } OBCM_UN;

   /* Output Buffer Command Request                                */
   /* 0x714 */
   union obcr
   {
      unsigned long OBCR_UL;
      struct
      {
         unsigned :9;
         unsigned obrh_B7 :7;
         unsigned obsys_B1 :1;
         unsigned :5;
         unsigned req_B1 :1;
         unsigned view_B1 :1;
         unsigned :1;
         unsigned obrs_B7 :7;
      } OBCR_ST;
   } OBCR_UN;
} FRAY_ST;

//*****************************************************
// Structure for initializing CC - Fr_Init - Fr_ConfigPtr
typedef volatile struct cfg
{
   int mrc;
   int prtc1;
   int prtc2;
   int mhdc;
   int gtu1;
   int gtu2;
   int gtu3;
   int gtu4;
   int gtu5;
   int gtu6;
   int gtu7;
   int gtu8;
   int gtu9;
   int gtu10;
   int gtu11;
   int succ2;
   int succ3;
} cfg;

// Structure for configuring buffer
typedef volatile struct wrhs
{
   //Header 1
   unsigned mbi;
   unsigned txm;
   unsigned ppit;
   unsigned cfg;
   unsigned chb;
   unsigned cha;
   unsigned cyc;
   unsigned fid;

   //Header 2
   unsigned plr;   //RDHS2
   unsigned plc;
   unsigned crc;

   //Header 3
   unsigned res;   //RDHS3
   unsigned ppi;   //RDHS3
   unsigned nfi;   //RDHS3
   unsigned syn;   //RDHS3
   unsigned sfi;   //RDHS3
   unsigned rci;   //RDHS3
   unsigned rcc;   //RDHS3
   unsigned dp;        //R/W
   //Message Buffer Status MBS (all R)
   unsigned long mbs_UL;

   //unsigned long Data[FLEX_PAYLOAD];

} wrhs;

// Structure for initializing buffer
typedef volatile struct bc
{
   //IBCM
   unsigned stxrs; //R
   unsigned ldss;  //R
   unsigned lhss;  //R
   unsigned stxrh;
   unsigned ldsh;
   unsigned lhsh;

   //IBCR
   unsigned ibsys; //R
   unsigned ibrs;  //R
   unsigned ibsyh; //R
   unsigned ibrh;

   // OBCM
   unsigned rdsh;  //R
   unsigned rhsh;  //R
   unsigned rdss;
   unsigned rhss;

   //OBCR
   unsigned obrh;
   unsigned obsys; //R
   unsigned req;
   unsigned view;
   unsigned rsvd;  //R
   unsigned obrs;
} bc; //Buffer command

//=====================================================
================
typedef volatile struct FRAY_BUFF
{
   struct Buff_Headers
   {
      union h1
      {
         unsigned long HEADER1_UL;
         struct
         {
            unsigned :2;
            unsigned Mess_buff_int :1;
            unsigned Transmi_mode :1;
            unsigned Payload_Pre_Indi :1; //ppit_B1 :1;  Payload preamble
indicator transmit
            unsigned Message_buf_config_bit :1;
            unsigned CHB_filt_contr :1;
            unsigned CHA_filt_contr :1;
            unsigned :1;
            unsigned Cycle_Code :7;
            unsigned :5;
            unsigned FrameID :11;
         } HEADER1_ST;
      } HEADER1_UN;
      union h2
      {
         unsigned long HEADER2_UL;
         struct
         {
            unsigned :9;
            unsigned Payload_length_config :7;
            unsigned :5;
            unsigned Header_CRC :11;
         } HEADER2_ST;
      } HEADER2_UN;
      union h3
      {
         unsigned long HEADER3_UL;
         struct
         {
            unsigned :21;
            unsigned Data_pointer :11;
         } HEADER3_ST;
      } HEADER3_UN;
      union h4
      {
         unsigned long BUFFER_STATUS_UL;
         struct
         {
            unsigned :2;
            unsigned Reserved_bit_status :1;
            unsigned Payload_preamble_indic :1;
            unsigned Null_frame_indic :1;
            unsigned Sync_frame_indicator :1;
            unsigned Startup_frame_indic :1;
            unsigned Received_ch_indic :1;
            unsigned :2;
            unsigned Cycle_count_status :6;
            unsigned Frame_transmitted_chB :1;
            unsigned Frame_transmitted_chA :1;
            unsigned :1; ///////////
            unsigned Message_lost :1;
            unsigned Empty_slot_chB :1;
            unsigned Empty_slot_chA :1;
            unsigned Transm_conflict_indication_chB :1;
            unsigned Transm_conflict_indication_chA :1;
            unsigned violation_limit_slot_chB :1;
            unsigned violation_limit_slot_chA :1;
            unsigned Content_error_chB :1;
            unsigned Content_error_chA :1;
            unsigned Syntax_error_chB :1;
            unsigned Syntax_error_chA :1;
            unsigned Valid_frame_received_chB :1;
            unsigned Valid_frame_received_chA :1;
         } BUFFER_STATUS_ST;
      } BUFFER_STATUS_UN;
   } Buff_Headers;
   unsigned long Buff_Data[FLEX_PAYLOAD];
   unsigned ranura :7;
} fray_buff;

typedef volatile struct trama
{
   unsigned ranura :1;
   union segmento_startup
   {
      unsigned seg_startup_UL :5;
      struct
      {
         unsigned reserva :1;
         unsigned pre_payload :1;
         unsigned nula :1;
         unsigned sincrona :1;
         unsigned startup :1;
      } seg_startup_ST;
   } seg_startup_UN;

   unsigned ID_trama :11;
   unsigned longt_payload :7;
   unsigned CRC_cabecera :11;
   unsigned cont_cicl :6;
   unsigned long dato[FLEX_PAYLOAD];
   unsigned long trailer_CRC :24;

} trama_t; //Buffer command

//*******************************************************

#define frayREG     (FRAY_ST *)     0xFFF7C800 //FlexRay Communication
Controller

// Functions
void Fray_buffConfig(FRAY_ST *Fray_PST, wrhs *Fr_LPduPtr);
int header_crc_calc(wrhs *Fr_LPduPtr);
void Fr_Init(FRAY_ST *Fray_PST, cfg *Fr_ConfigPtr);
int Fr_ControllerInit(FRAY_ST *Fray_PST);
int Fr_AllowColdStart(FRAY_ST *Fray_PST);
int Fr_StartCommunication(FRAY_ST *Fray_PST);
int Fr_ReStartCommunication(FRAY_ST *Fray_PST);

void Fray_Transfer_Input(FRAY_ST *Fray_PST, bc *Fr_LSduPtr);
void Fray_Transfer_Output(FRAY_ST *Fray_PST, bc *Fr_LSduPtr);

void Config_CC_Nodo_A(FRAY_ST *Fray_PST);
void Config_Custom_mSlot(FRAY_ST *Fray_PST,trama_t* frame);
void configure_initialize_node_b(FRAY_ST *Fray_PST);

void Escritura_trama(trama_t* frame, unsigned seg_startup, unsigned fid,
unsigned plc, unsigned cont_cycl);
void Lectura_trama(trama_t* frame, fray_buff* bufer);

#endif
```

Anexo C. Código biblioteca FlexRay(fray.c)

```
#include <Codigos/fray.h>

void Fray_buffConfig(FRAY_ST *Fray_PST, wrhs *Fr_LPduPtr)
/*****************************************************************
****
 The function Fr_PrepareLPdu shall perform the following tasks on FlexRay
 CC Fr_CtrIdx:
 1. Figure out the physical resource (e.g., a buffer) mapped to the processing of
 the FlexRay frame identified by Fr_LPduIdx.
 2. Configure the physical resource (a buffer) appropriate for Fr_LPduPtr
 operation (SlotId, Cycle filter, payload length, header CRC, etc.) if the MCG
 uses the reconfiguration feature.

*****************************************************************
****/
{
   int wrhs1;
   int wrhs2;
   wrhs1 = (Fr_LPduPtr->mbi) & 0x1) << 29;
   wrhs1 |= (Fr_LPduPtr->txm & 0x1) << 28;
   wrhs1 |= (Fr_LPduPtr->ppit & 0x1) << 27;
   wrhs1 |= (Fr_LPduPtr->cfg & 0x1) << 26;
   wrhs1 |= (Fr_LPduPtr->chb & 0x1) << 25;
   wrhs1 |= (Fr_LPduPtr->cha & 0x1) << 24;
   wrhs1 |= (Fr_LPduPtr->cyc & 0x7F) << 16;
   wrhs1 |= (Fr_LPduPtr->fid & 0x7FF);
   Fray_PST->WRHS1_UN.WRHS1_UL = wrhs1;

   wrhs2 = (Fr_LPduPtr->plc & 0x7F) << 16) | (Fr_LPduPtr->crc & 0x7FF);
   Fray_PST->WRHS2_UN.WRHS2_UL = wrhs2;

   Fray_PST->WRHS3_UN.WRHS3_UL = (Fr_LPduPtr->dp & 0x7FF);
}

void Fr_Init(FRAY_ST *Fray_PST, cfg *Fr_ConfigPtr)
/*****************************************************************
****
 The function Fr_Init shall internally store the configuration address to enable
 subsequent API calls to access the configuration.

*****************************************************************
****/
{
   Fray_PST->MRC_UN.MRC_UL = Fr_ConfigPtr->mrc;
   Fray_PST->PRTC1_UN.PRTC1_UL = Fr_ConfigPtr->prtc1;
   Fray_PST->PRTC2_UN.PRTC2_UL = Fr_ConfigPtr->prtc2;
   Fray_PST->MHDC_UN.MHDC_UL = Fr_ConfigPtr->mhdc;
   Fray_PST->GTUC1_UN.GTUC1_UL = Fr_ConfigPtr->gtu1;
```

```
    Fray_PST->GTUC2_UN.GTUC2_UL = Fr_ConfigPtr->gtu2;
    Fray_PST->GTUC3_UN.GTUC3_UL = Fr_ConfigPtr->gtu3;
    Fray_PST->GTUC4_UN.GTUC4_UL = Fr_ConfigPtr->gtu4;
    Fray_PST->GTUC5_UN.GTUC5_UL = Fr_ConfigPtr->gtu5;
    Fray_PST->GTUC6_UN.GTUC6_UL = Fr_ConfigPtr->gtu6;
    Fray_PST->GTUC7_UN.GTUC7_UL = Fr_ConfigPtr->gtu7;
    Fray_PST->GTUC8_UN.GTUC8_UL = Fr_ConfigPtr->gtu8;
    Fray_PST->GTUC9_UN.GTUC9_UL = Fr_ConfigPtr->gtu9;
    Fray_PST->GTUC10_UN.GTUC10_UL = Fr_ConfigPtr->gtu10;
    Fray_PST->GTUC11_UN.GTUC11_UL = Fr_ConfigPtr->gtu11;
    Fray_PST->SUCC2_UN.SUCC2_UL = Fr_ConfigPtr->succ2;
    Fray_PST->SUCC3_UN.SUCC3_UL = Fr_ConfigPtr->succ3;
}

void Fray_Transfer_Input(FRAY_ST *Fray_PST, bc *Fr_LSduPtr)
/*****************************************************************
****
 Transferencia de datos desde el búfer de entrada a la RAM de mensajes

*****************************************************************
****/
{
    //Espere hasta que se complete la transferencia en curso
    while (Fray_PST->IBCR_UN.IBCR_ST.ibsyh_B1 != 0)
        ;    // wait for completion on host registers
    while (Fray_PST->IBCR_UN.IBCR_ST.ibsys_B1 != 0)
        ;    // wait for completion on shadow registers

    //Escribir los datos en buffer (WRDSn)
    //Echo en la funcion CargaBuffer

    //Escriba la sección de encabezado en WRHS1,2,3
    //Echo en la funcion Fray_buffConfig

    //Máscara de comando de escritura
    Fray_PST->IBCM_UN.IBCM_ST.stxrh_B1 = Fr_LSduPtr->stxrh;
    Fray_PST->IBCM_UN.IBCM_ST.lhsh_B1 = Fr_LSduPtr->lhsh;
    Fray_PST->IBCM_UN.IBCM_ST.ldsh_B1 = Fr_LSduPtr->ldsh;

    //Solicitar transferencia de datos al búfer de mensajes de destino (ibrh = numero
de buffer)
    Fray_PST->IBCR_UN.IBCR_ST.ibrh_B7 = Fr_LSduPtr->ibrh; //Fr_LSduPtr->ibrh &
0x3F;

    while (Fr_LSduPtr->ibsyh != 0 && Fray_PST->IBCR_UN.IBCR_ST.ibsyh_B1 != 0)
        ;    // wait for completion on host registers
    while (Fr_LSduPtr->ibsys != 0 && Fray_PST->IBCR_UN.IBCR_ST.ibsys_B1 != 0)
        ;    // wait for completion on shadow registers
}

void Fray_Transfer_Output(FRAY_ST *Fray_PST, bc *Fr_LSduPtr)
/*****************************************************************
****
```

```
 Transferencia de datos desde la RAM de mensajes al búfer de salida

*****************************************************************
****/
{
   // Garantizar que no haya transferencias en curso
   while (Fray_PST->OBCR_UN.OBCR_ST.obsys_B1 != 0)
      ;

   //Escribir máscara de comando de búfer de salida
   Fray_PST->OBCM_UN.OBCM_ST.rhss_B1 = Fr_LSduPtr->rhss & 0x1;
   Fray_PST->OBCM_UN.OBCM_ST.rdss_B1 = Fr_LSduPtr->rdss & 0x1;

   //Solicitud de la transferencia del búfer de mensajes
   Fray_PST->OBCR_UN.OBCR_ST.obrs_B7 = Fr_LSduPtr->obrs; //Fr_LSduPtr-
>obrs & 0x3F; //req=1, view=0
   Fray_PST->OBCR_UN.OBCR_ST.req_B1 = 1;

   // wait for completion on shadow registers
   while (Fray_PST->OBCR_UN.OBCR_ST.obsys_B1 != 0)
      ;

   Fray_PST->OBCR_UN.OBCR_ST.view_B1 = 1;

   //Máscara de comando de búfe salida para segundo mensaje
   Fray_PST->OBCM_UN.OBCM_ST.rhss_B1 = Fr_LSduPtr->rhss & 0x1;
   Fray_PST->OBCM_UN.OBCM_ST.rdss_B1 = Fr_LSduPtr->rdss & 0x1;

   //Alterna OBF Shadow y OBF Host e inicia la transferencia del segundo búfer de
mensaje
   Fray_PST->OBCR_UN.OBCR_ST.obrs_B7 = Fr_LSduPtr->obrs; //req=0, view=1
   Fray_PST->OBCR_UN.OBCR_ST.req_B1 = 1;

   // wait for completion on shadow registers
   while (Fray_PST->OBCR_UN.OBCR_ST.obsys_B1 != 0)
      ;

   Fray_PST->OBCR_UN.OBCR_ST.view_B1 = 1;
}

int Fr_ControllerInit(FRAY_ST *Fray_PST)
/*****************************************************************
****
 Fr_ControllerInit

*****************************************************************
****/
{
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_UL = 0x0F1FFB00;
   if (Fray_PST->CCSV_UN.CCSV_ST.pocs_B6 != 0xF) //CONFIG state
      Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_CONFIG;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   // unlock CONFIG and enter READY state
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0xCE;
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0x31;
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_READY;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   return 1;
}

int Fr_AllowColdStart(FRAY_ST *Fray_PST)
/*****************************************************************
****
 Fr_AllowColdStart

*****************************************************************
****/
{
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_ALLOW_COLDSTART;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   return 1;
}

int Fr_StartCommunication(FRAY_ST *Fray_PST)
/*****************************************************************
****
 Fr_StartCommunication

*****************************************************************
****/
{
   // write SUCC1 configuration
   (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_RUN);
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   return 1;
}
```

```
int Fr_ReStartCommunication(FRAY_ST *Fray_PST)
{

   // write SUCC1 configuration
   (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_FREEZE);
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   Fray_PST->SUCC1_UN.SUCC1_UL = 0x0F1FFB00;
   if (Fray_PST->CCSV_UN.CCSV_ST.pocs_B6 != 0xF) //CONFIG state
      Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_CONFIG;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;
   // unlock CONFIG and enter READY state
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0xCE;
   Fray_PST->LCK_UN.LCK_ST.clk_B8 = 0x31;
   // write SUCC1 configuration
   Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_READY;
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   // Wait for PBSY bit to clear - POC not busy
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0x0)
      ;

   // write SUCC1 configuration
   (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 = CMD_RUN);
   // Check if POC has accepted last command
   if (Fray_PST->SUCC1_UN.SUCC1_ST.cmd_B4 == 0x0)
      return -1;
   return 1;
}

int header_crc_calc(wrhs *Fr_LPduPtr)
/******************************************************************
****
 This function calculates the header CRC.

******************************************************************
****/
{
   unsigned int header;

   int CrcInit = 0x1A;
   int length = 20;
   int CrcNext;
   unsigned long CrcPoly = 0x385;
   unsigned long CrcReg_X = CrcInit;
```

```
    unsigned long header_temp, reg_temp;

    header = (Fr_LPduPtr->syn & 0x1) << 19) | (Fr_LPduPtr->sfi & 0x1) << 18);
    header |= (Fr_LPduPtr->fid & 0x7FF) << 7) | (Fr_LPduPtr->plc & 0x7F);

    header <<= 11;
    CrcReg_X <<= 21;
    CrcPoly <<= 21;

    while (length--)
    {
        header <<= 1;
        header_temp = header & 0x80000000;
        reg_temp = CrcReg_X & 0x80000000;

        if (header_temp ^ reg_temp)
        {  // Step 1
            CrcNext = 1;
        }
        else
        {
            CrcNext = 0;
        }

        CrcReg_X <<= 1;             // Step 2

        if (CrcNext)
        {
            CrcReg_X ^= CrcPoly;        // Step 3
        }
    }

    CrcReg_X >>= 21;

    return CrcReg_X;
}

void Escritura_trama(trama_t* frame, unsigned seg_startup, unsigned fid,
                     unsigned plc, unsigned cont_cycl)
/*****************************************************************
****
 frame
 segment startup => (ppi)(nfi)(syn)(sfi)
 frame ID
 payload length configured (0xFF <=> 1 byte)
 Receive cycle count
 Datos

*****************************************************************
****/
{
    frame->seg_startup_UN.seg_startup_UL = seg_startup;
    frame->ID_trama = fid;                  //frame ID
```

```
   frame->longt_payload = plc;    //payload length configured (0xFF <=> 1 byte)

   wrhs header; //Calculo del CRC
   header.syn = frame->seg_startup_UN.seg_startup_ST.sincrona;
   header.sfi = frame->seg_startup_UN.seg_startup_ST.startup;
   header.fid = frame->ID_trama;
   header.plc = frame->longt_payload;
   frame->CRC_cabecera = header_crc_calc(&header); // calculated  by the host

   frame->cont_cicl = cont_cycl;                  //Receive cycle count
   frame->trailer_CRC = 0;
}

void Lectura_trama(trama_t* frame, fray_buff* bufer)
/*******************************************************************
****
 Leé trama Flexray

*******************************************************************
****/
{
   frame->ranura = bufer->ranura;
   frame->seg_startup_UN.seg_startup_ST.reserva =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Reserved_bit_status;
   frame->seg_startup_UN.seg_startup_ST.pre_payload =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Payload_preamble_indic;
   frame->seg_startup_UN.seg_startup_ST.nula =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Null_frame_indic;
   frame->seg_startup_UN.seg_startup_ST.sincrona =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Sync_frame_indicator;
   frame->seg_startup_UN.seg_startup_ST.startup =
         bufer-
>Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_ST.Startup_frame_indic;

   frame->ID_trama = bufer->Buff_Headers.HEADER1_UN.HEADER1_ST.FrameID;
   frame->longt_payload =
         bufer->Buff_Headers.HEADER2_UN.HEADER2_ST.Payload_length_config;
   frame->CRC_cabecera = bufer-
>Buff_Headers.HEADER2_UN.HEADER2_ST.Header_CRC;
   frame->cont_cicl = bufer-
>Buff_Headers.HEADER1_UN.HEADER1_ST.Cycle_Code;
   int j;
   for (j = 0; j < frame->longt_payload; j++)
   {
      frame->dato[j] = bufer->Buff_Data[j];
   }
}
```

Anexo D. Código biblioteca MRSIII (MRSIII.h)

```
/*
 * MRSIII.h
 *
 *  Created on: 24/06/2021
 *      Author: Luis Rey
 */

#ifndef CODIGOS_MRSIII_H_
#define CODIGOS_MRSIII_H_

#include <Codigos/Panel_Airbag.h>
#include <Codigos/fray.h>

typedef volatile struct MRS_III
{
    unsigned Activacion    : 1;
    unsigned Estado;
    unsigned Umbral_pretensor   : 5;
    unsigned Umbral_activacion_frontal   : 5;
    unsigned Umbral_choque_trasero   : 5;
    unsigned Umbral_BST        : 5;
    unsigned Umbral_ITS        : 5;
    unsigned Umbral_airbag_1   : 5;
    unsigned Umbral_airbag_2   : 5;
    unsigned Umbral_airbag_3   : 5;
    unsigned Umbral_airbag_4   : 5;
} MRS_t;

void config_Umrales(trama_t* frame);
void set_Umbrales(MRS_t* umbrales);
void get_Umbrales(MRS_t* umbrales);
int autocomprobacion_panel_Indica();
int autocomprobacion_sensores();
void reset_Indicadores();
void set_Indicadores();
void ejecucion_MRSIII();
void ActivarIndica(panelact_t* panel, uint8_t retraso);
unsigned ActAirbag_Cond(uint8_t prioridad);
unsigned ActiAirbag_Pasj(uint8_t prioridad);
void whait(unsigned long value);
int verifica_Vel();
int desacelerador();
void initUmbrales();

#endif /* CODIGOS_MRSIII_H_ */
```

Anexo E. Código biblioteca MRSIII (MRSIII.c)

```c
/*
 * MRSIII.c
 *
 *  Created on: 16/11/2020
 *      Author: Luis Rey
 */

#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>

unsigned Umbral_pretensor = 0x02;
unsigned Umbral_activacion_frontal = 0x05;
unsigned Umbral_choque_trasero = 0x07;
unsigned Umbral_BST_FRONT;
unsigned Umbral_BST_LAT;
unsigned Umbral_BST_TRA;
unsigned Umbral_ITS;

unsigned Umbral_airbag_1;
unsigned Umbral_airbag_2;
unsigned Umbral_airbag_3;
unsigned Umbral_airbag_4;

unsigned Vel_ant;

int dxlay = 400;
hetBASE_t* het1_debug = hetREG1;

extern panelsen_t panel_sensores;
extern panelact_t panel_indicadores;
extern unsigned modo_config;
extern FRAY_ST* FlexRay_CC;
extern MRS_t MRSIII;
extern volatile boolean dinaseg_1;
extern volatile boolean dinaseg_2;
extern volatile boolean dinaseg_3;
extern volatile boolean dinaseg_4;
extern volatile boolean dinaseg_5;

void initUmbrales()
{
    Umbral_airbag_1 = Umbral_activacion_frontal;
    Umbral_airbag_2 = Umbral_activacion_frontal * 2;
    Umbral_airbag_3 = Umbral_activacion_frontal * 3;
    Umbral_airbag_4 = Umbral_activacion_frontal * 4;

    Umbral_BST_FRONT = Umbral_airbag_2;
    Umbral_BST_LAT = Umbral_airbag_1;
    Umbral_BST_TRA = Umbral_airbag_2;
    Umbral_ITS = Umbral_airbag_2;


}

void config_Umrales(trama_t* frame)
{
    Umbral_pretensor = frame->dato[0];
    Umbral_activacion_frontal = frame->dato[1];
    Umbral_choque_trasero = frame->dato[2];

    initUmbrales();

    modo_config = 0;
}

void set_Umbrales(MRS_t* umbrales)
{
    Umbral_pretensor = umbrales->Umbral_pretensor;
    Umbral_activacion_frontal = umbrales->Umbral_activacion_frontal;
    Umbral_choque_trasero = umbrales->Umbral_choque_trasero;
    initUmbrales();
}

void get_Umbrales(MRS_t* umbrales)
{
    umbrales->Umbral_pretensor = Umbral_pretensor;
    umbrales->Umbral_activacion_frontal = Umbral_activacion_frontal;
    umbrales->Umbral_choque_trasero = Umbral_choque_trasero;
    umbrales->Umbral_BST = Umbral_BST_FRONT;
    umbrales->Umbral_ITS = Umbral_ITS;
    umbrales->Umbral_airbag_1 = Umbral_airbag_1;
    umbrales->Umbral_airbag_2 = Umbral_airbag_2;
    umbrales->Umbral_airbag_3 = Umbral_airbag_3;
    umbrales->Umbral_airbag_4 = Umbral_airbag_4;
}

int autocomprobacion_panel_Indica()
{
    panelact_t* indicadores = &panel_indicadores;
    int i;
    for (i = 0; i < 0x3FFF; i = (i << 1) + 1)
    {
        indicadores->Indicadores_UN.Indicadores_UL = i;
        whait(250);
        ActivarIndica(indicadores, 0);
    }
    whait(400);
    indicadores->Indicadores_UN.Indicadores_UL = 0;
    ActivarIndica(indicadores, 0);
    whait(400);
    indicadores->Indicadores_UN.Indicadores_UL = 0x3FFF;
    ActivarIndica(indicadores, 0);
    whait(300);
    indicadores->Indicadores_UN.Indicadores_UL = 0;
    ActivarIndica(indicadores, 0);
    whait(300);
    indicadores->Indicadores_UN.Indicadores_UL = 0x3FFF;
    ActivarIndica(indicadores, 0);
    whait(700);
    indicadores->Indicadores_UN.Indicadores_UL = 0;
    ActivarIndica(indicadores, 0);
    return 3;
}

int autocomprobacion_sensores()
{
    panelsen_t* sensores = &panel_sensores;
    panelact_t* indicadores = &panel_indicadores;

    if (sensores->encendido)
    {
        if (sensores->seg_Front_Cond & sensores->seg_Front_Pasj
                & sensores->seg_Late_Cond & sensores->seg_Late_Pasj
                & sensores->seg_Trasero)
        {
            if (sensores->SBE_Cond & sensores->SBE_Pasj)
            {
                if (sensores->scint_Cond & sensores->scint_Pasj)
                {
                    indicadores->Indicadores_UN.Indicadores_UL = 0;
                    ActivarIndica(indicadores, 0);
                    return 1;
                }
            }
        }
        indicadores->Indicadores_UN.Indicadores_UL = 0;
        indicadores->Indicadores_UN.Indicadores_ST.Testigo_Lum = 1;
        ActivarIndica(indicadores, 0);
        return 0;
    }
    indicadores->Indicadores_UN.Indicadores_UL = 0;
    ActivarIndica(indicadores, 0);
    return 2;
}

void reset_Indicadores()
{
    panelact_t* indicadores = &panel_indicadores;
    int x;
    x=150;
    indicadores->Indicadores_UN.Indicadores_UL = 0x1;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0xA;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0x14;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0x60;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0x180;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0x600;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0xFFF;
    ActivarIndica(indicadores, 0);
    whait(x);
    indicadores->Indicadores_UN.Indicadores_UL = 0;
    ActivarIndica(indicadores, 0);
}

void set_Indicadores()
{
    panelact_t* indicadores = &panel_indicadores;
    indicadores->Indicadores_UN.Indicadores_UL = 0x1FFF;
    ActivarIndica(indicadores, 0);
}

int verifica_Vel()
{
    if (panel_sensores.velocidad > 1)
        return 1;
    return 0;
}

int desacelerador()
{
    if (Vel_ant - panel_sensores.velocidad >= Umbral_pretensor
            && Vel_ant - panel_sensores.velocidad <= 6)
        return 1;
    return 0;
}

void ejecucion_MRSIII()
{
    panelsen_t* sensores = &panel_sensores;
    panelact_t* indicadores = &panel_indicadores;
    unsigned modo = 0x00;

    //Control de pretensores del cinturon
    if (desacelerador() == 1)
    {
        if (sensores->SBE_Cond == 1)
            if (sensores->scint_Cond == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
        if (sensores->SBE_Pasj == 1)
            if (sensores->scint_Pasj == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
        MRSIII.Activacion = 1;
    }

    //Vel_ant=panel_sensores.velocidad;

    if (verifica_Vel() == 1)
    {

        //control activacion frontal del airbag conductor
        if (sensores->seg_Front_Cond == 1)
        {
            if (sensores->imp_Front_Cond >= Umbral_airbag_4)
            {
                modo = ActAirbag_Cond(4);
                MRSIII.Activacion = 1;
            }
            else if (sensores->imp_Front_Cond >= Umbral_airbag_3)
            {
                modo = ActAirbag_Cond(3);
                MRSIII.Activacion = 1;
            }
            else if (sensores->imp_Front_Cond >= Umbral_airbag_2)
            {
                modo = ActAirbag_Cond(2);
                MRSIII.Activacion = 1;
            }
            else if (sensores->imp_Front_Cond >= Umbral_airbag_1)
            {
                modo = ActAirbag_Cond(1);
                MRSIII.Activacion = 1;
            }
        }

        //control activacion frontal del airbag pasajero
        if (sensores->seg_Front_Pasj == 1)
        {
            if (sensores->imp_Front_Pasj >= Umbral_airbag_4)
            {
                modo = (modo & 0xC) + ActiAirbag_Pasj(4);
                MRSIII.Activacion = 1;
            }
            else if (sensores->imp_Front_Pasj >= Umbral_airbag_3)
            {
                modo = (modo & 0xC) + ActiAirbag_Pasj(3);
                MRSIII.Activacion = 1;
            }
            else if (sensores->imp_Front_Pasj >= Umbral_airbag_2)
        {
           modo = (modo & 0xC) + ActiAirbag_Pasj(2);
           MRSIII.Activacion = 1;
        }
        else if (sensores->imp_Front_Pasj >= Umbral_airbag_1)
        {
           modo = (modo & 0xC) + ActiAirbag_Pasj(1);
           MRSIII.Activacion = 1;
        }
     }

     //control activacion trasera del airbag
     if (Umbral_choque_trasero < sensores->imp_Trasero
           && sensores->seg_Trasero == 1)
     {
        modo = 0;
        if (sensores->imp_Trasero > Umbral_airbag_4)
        {
           modo = ActAirbag_Cond(4) + ActiAirbag_Pasj(4);
           if (sensores->scint_Cond == 1)
              indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
           if (sensores->scint_Pasj == 1)
              indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
           MRSIII.Activacion = 1;
        }
        else if (sensores->imp_Trasero > Umbral_airbag_3)
        {
           modo = ActAirbag_Cond(3) + ActiAirbag_Pasj(3);
           if (sensores->scint_Cond == 1)
              indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
           if (sensores->scint_Pasj == 1)
              indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
           MRSIII.Activacion = 1;
        }
     }

     //Control del BST
     if (Umbral_BST_FRONT <= sensores->imp_Front_Cond
           && sensores->seg_Front_Cond == 1)
           || (Umbral_BST_FRONT <= sensores->imp_Front_Pasj
                 && sensores->seg_Front_Pasj == 1)
           || (Umbral_BST_TRA <= sensores->imp_Trasero
                 && sensores->seg_Trasero == 1)
           || (Umbral_BST_LAT <= sensores->imp_Late_Cond
                 && sensores->seg_Late_Cond == 1)
           || (Umbral_BST_LAT <= sensores->imp_Late_Pasj
                 && sensores->seg_Late_Pasj == 1)
     {
        indicadores->Indicadores_UN.Indicadores_ST.BST = 1;
        if (sensores->SBE_Cond == 1)
           if (sensores->scint_Cond == 1)
              indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
        if (sensores->SBE_Pasj == 1)
```

```
            if (sensores->scint_Pasj == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
         MRSIII.Activacion = 1;
      }

      //Control de ITS
      if (Umbral_ITS < sensores->imp_Late_Cond
            || Umbral_ITS < sensores->imp_Late_Pasj)
            && (sensores->seg_Late_Cond == 1 || sensores->seg_Late_Pasj == 1)
      {
         if (sensores->SBE_Cond == 1)
         {
            indicadores->Indicadores_UN.Indicadores_ST.Acti_ITS_1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Thorax_cond = 1;
            if (sensores->scint_Cond == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
         }
         if (sensores->SBE_Pasj == 1)
         {
            indicadores->Indicadores_UN.Indicadores_ST.Acti_ITS_2 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Thorax_pasj = 1;
            if (sensores->scint_Pasj == 1)
                indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
         }

         MRSIII.Activacion = 1;
      }
   }
   if (MRSIII.Activacion==1)
   {
      rtiStartCounter(rtiREG1, 1);
      ActivarIndica(indicadores, modo);
      dinaseg_5=true;
   }
}

void ActivarIndica(panelact_t* panel, uint8_t convinacion)
{
   HET1_digitalWrite(PIN_HET_2,
                  panel->Indicadores_UN.Indicadores_ST.Testigo_Lum);
   HET1_digitalWrite(PIN_HET_18,
                  panel->Indicadores_UN.Indicadores_ST.Desp_cond_E1);
   HET1_digitalWrite(PIN_HET_30,
                  panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E1);
   HET1_digitalWrite(PIN_HET_22,
                  panel->Indicadores_UN.Indicadores_ST.Acti_ITS_1);
   HET1_digitalWrite(PIN_HET_12,
                  panel->Indicadores_UN.Indicadores_ST.Acti_ITS_2);
   HET1_digitalWrite(PIN_HET_25,
                  panel->Indicadores_UN.Indicadores_ST.Thorax_cond);
   HET1_digitalWrite(PIN_HET_27,
                  panel->Indicadores_UN.Indicadores_ST.Thorax_pasj);
   HET1_digitalWrite(PIN_HET_29,
                 panel->Indicadores_UN.Indicadores_ST.Preten_cond);
    HET1_digitalWrite(PIN_HET_10,
                 panel->Indicadores_UN.Indicadores_ST.Preten_pasj);
    HET1_digitalWrite(PIN_HET_28, panel->Indicadores_UN.Indicadores_ST.BST);
    switch (convinacion)
    {
    case 0:
    {
       HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       break;
    }
    case 1:
    {
       HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);

       break;
    }
    case 2:
    {
       HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       whait(dxlay);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       break;
    }
    case 4:
    {
       HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);

       break;
    }
    case 8:
    {
       HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
       whait(dxlay);
       whait(dxlay);
       HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
       break;
```

```
   }
   case 5:
   {
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      break;
   }
   case 9:
   {
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      break;
   }
   case 6:
   {
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      break;
   }
   case 10:
   {
      whait(dxlay);
      whait(dxlay);
      HET1_digitalWrite(PIN_HET_16,
                     panel->Indicadores_UN.Indicadores_ST.Desp_cond_E2);
      HET1_digitalWrite(PIN_HET_14,
                     panel->Indicadores_UN.Indicadores_ST.Desp_pasj_E2);
      break;
   }
   }
}

unsigned ActAirbag_Cond(uint8_t prioridad)
//prioridad 4->maxima 1->minima,
{
   panelsen_t* sensores = &panel_sensores;
   panelact_t* indicadores = &panel_indicadores;

   if (sensores->SBE_Cond == 1)
   {
      if (sensores->scint_Cond == 1)
      {
```

```
      indicadores->Indicadores_UN.Indicadores_ST.Preten_cond = 1;
      switch (prioridad)
      {
      case 1:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 0;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 0;
         return 0;
      }
      case 2:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 0;
         return 0;
      }
      case 3:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
         return 4;
      }
      case 4:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
         return 0;
      }
      }
   }
   else
   {
      switch (prioridad)
      {
      case 1:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 0;
         return 0;
      }
      case 2:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
         return 8;
      }
      case 3:
      {
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
         indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
         return 4;
      }
      case 4:
      {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_cond_E2 = 1;
            return 0;
         }
         }
      }
   }
   return 0;
}

unsigned ActiAirbag_Pasj(uint8_t prioridad)
//prioridad 4->maxima 1->minima,
{
   panelsen_t* sensores = &panel_sensores;
   panelact_t* indicadores = &panel_indicadores;

   if (sensores->SBE_Pasj == 1)
   {
      if (sensores->scint_Pasj == 1)
      {
         indicadores->Indicadores_UN.Indicadores_ST.Preten_pasj = 1;
         switch (prioridad)
         {
         case 1:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 0;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 0;
            return 0;
         }
         case 2:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 0;
            return 0;
         }
         case 3:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
            return 1;
         }
         case 4:
         {
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
            indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
            return 0;
         }
         }
      }
      else
      {
         switch (prioridad)
         {
            case 1:
            {
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 0;
               return 0;
            }
            case 2:
            {
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
               return 2;
            }
            case 3:
            {
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
               return 1;
            }
            case 4:
            {
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E1 = 1;
               indicadores->Indicadores_UN.Indicadores_ST.Desp_pasj_E2 = 1;
               return 0;
            }
            }
         }
      }
   return 0;
}

void whait(unsigned long value)
{
   while (value)
   {
      long x = value * 180;
      while (x)
      {
         x--;
      }
      value--;
   }
}

void HET1_digitalWrite(unsigned HET_pin, unsigned value)
{
   uint32 mask = 0xFFFFFFFF - (value << HET_pin);
   het1_debug->DSET = 1 << HET_pin;
   het1_debug->DOUT = (uint32) hetREG1->DOUT & mask;
}
```

Anexo F. Código módulo FlexRay A

```
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>

wrhs header_reg; //Variables globales
cfg Fray_Config;
bc Fray_buffcomand;
int uh;
int custon12 = 0;

extern panelsen_t panel_sensores;
extern panelact_t panel_indicadores;
extern MRS_t MRSIII;
extern unsigned datos_carga_util[64];
extern volatile boolean dinaseg_1;
extern volatile boolean dinaseg_2;
extern volatile boolean dinaseg_3;
extern volatile boolean dinaseg_4;
extern volatile boolean dinaseg_5;
extern unsigned modo_config;
extern unsigned frameMOD;
extern trama_t interfaz_trama_in;

void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,
                   unsigned plc, unsigned long* datos);

void Config_CC_Nodo_A(FRAY_ST *Fray_PST)
{
   //Variables locales apuntan a las varibles globales
   cfg *Fr_ConfigPtr = &Fray_Config; // Structure for initializing CC - Fr_Init -
Fr_ConfigPtr

   // GTUs (Global Time Unit ), PRTC configuration
   Fr_ConfigPtr->gtu1 = 0x00036B00; // pMicroPerCycle = 224000d = 36B00h (has
to be x40 of MacroPerCyle)
                              // [19:0]: These bits configure the duration of the
communication cycle in microticks

   Fr_ConfigPtr->gtu2 = 0x000F15E0; // gSyncodeMax = Fh, gMacroPerCycle =
5600d = 15E0h  (cycle period, 5.6us)
                              //[13:0]: Macrotick per cycle (in macroticks). These bits
configure the duration of one communication cycle
                              //     in macroticks. The cycle length must be identical in all
nodes of a cluster.
                              //[19:16]: Sync node max (in frames). These bits
configure the maximum number of frames within a cluster
                              //      with sync frame indicator bit SYN set. The number of
frames must be identical in all nodes of a cluster.
```

```
   Fr_ConfigPtr->gtu3 = 0x00061818; // gMacroInitialOffset = 6h,
pMicroInitialOffset = 24d = 18h
   Fr_ConfigPtr->gtu4 = 0x0AE40AE3; // gOffsetCorrectionStart - 1 = 2788d = AE4h
(OCS)  , gMacroPerCycle - gdNIT - 1 = 2787d = AE3h (NIT)
   Fr_ConfigPtr->gtu5 = 0x33010303; // pDecodingCorrection = 51d = 33h,
pClusterDriftDamping = 1h, pDelayCompensation = 3h
   Fr_ConfigPtr->gtu6 = 0x01510081; // pdMaxDrift = 337d = 151h,
pdAcceptedStartupRange = 129d = 81h

   Fr_ConfigPtr->gtu7 = 0x00080056; // gNumberOfStaticSlots = 8h, gdStaticSlot =
86d = 56h
                              // [25:16]: These bits configure the number of static slots
in a cycle.
                              // [9:0]: These bits configure the duration of a static slot
(macroticks).

   Fr_ConfigPtr->gtu8 = 0x015A0004; // gNumberOfMinislots = 346d = 15Ah,
gdMinislot = 4h
                              // [28:16]:These bits configure the number of minislots in
the dynamic segment of a cycle
                              // [5:0]: These bits configure the duration of a minislot

   Fr_ConfigPtr->gtu9 = 0x00010204; // gdDynamicSlotIdlePhase = 1,
gdMinislotActionPointOffset = 2, gdActionPointOffset = 4h
   Fr_ConfigPtr->gtu10 = 0x015100CD; // pRateCorrectionOut = 337d = 151h,
pOffsetCorrectionOut = 205d = CDh
   Fr_ConfigPtr->gtu11 = 0x00000000; // pExternRateCorrection = 0,
pExternOffsetCorrection = 0, no ext. clk. corr.

   Fr_ConfigPtr->succ2 = 0x0F036DA2; // gListenNoise = Fh, pdListenTimeout =
224674d = 36DA2h
                               //LTN [27:24]: Listen timeout noise. Configures the upper
limit for the startup and wakeup listen timeout in the
                               //presence of noise. Must be identical in all nodes of a
cluster.
                               //The wakeup / startup noise timeout is calculated as
follows: LT[20:0] · (LTN[3:0] + 1)
                               // LT[20:0]: Listen timeout. Configures the upper limit of
the startup and wakeup listen timeout.

   Fr_ConfigPtr->succ3 = 0x000000FF; // gMaxWithoutClockCorrectionFatal = Fh ,
passive = Fh
                               //WCF[7:4]: Maximum without clock correction fatal.
These bits define the number of consecutive even/odd
                               //cycle pairs with missing clock correction terms that will
cause a transition from
                               //NORMAL_ACTIVE or NORMAL_PASSIVE state.

   //WCP[3:0]: Maximum without clock correction passive. These bits define the
number of consecutive
   //even/odd cycle pairs with missing clock correction terms that will cause a
transition from
```

```
   //NORMAL_ACTIVE to NORMAL_PASSIVE to HALT state.

   Fr_ConfigPtr->prtc1 = 0x084C000A; // pWakeupPattern = 2h,
gdWakeupSymbolRxWindow = 76d, BRP = 0, gdTSSTransmitter = Ah
                              //BRP[15:14]; Baud rate prescaler. These bits configure
the baud rate on the FlexRay bus. The baud rates
                              //listed below are valid with a sample clock of 80 MHz.
One bit time always consists of 8 samples
                              //independent of the configured baud rate.  =0 ->10Mb/s

   Fr_ConfigPtr->prtc2 = 0x3CB41212; // gdWakeupSymbolTxLow = 60d,
gdWakeupSymbolTxIdle = 180d, gdWakeupSymbolRxLow = 18d,
gdWakeupSymbolRxIdle = 18d

   Fr_ConfigPtr->mhdc = 0x010D0009; // pLatestTransmit = 269d = 010Dh,
gPayloadLengthStatic = 9h
                              //Start of latest transmit (in minislots). These bits
configure the maximum minislot value allowed
                              //minislots before inhibiting new frame transmissions in the
Dynamic Segment of the cycle.
                              //[7:0]: Static frame data length.

   Fr_ConfigPtr->mrc = 0x00174006; // LCB=23d, FFB=64d, FDB=4d (0..3 static,
4..23 dyn., 0 fifo)

   // Wait for PBSY bit to clear - POC not busy.
   // 1: Signals that the POC is busy and cannot accept a command from the host.
CMD(3-0) is locked against write accesses.
   while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0)
      ;

   // Initialize
   Fr_Init(Fray_PST, Fr_ConfigPtr);

   //Configuración por defecto slots

   wrhs *Fray_header = &header_reg; // Structure for configuring buffers -
Fray_buffConfig - Fray_header
   bc *Fray_buff_comand = &Fray_buffcomand; // Structure for initializing buffer -
Fray_Transfer_Input, Fr_ReceiveRxLPdu - Fray_buff_comand

   //Header Section Global Config
   //Header Section Register 1 (WRHS1)
   Fray_header->mbi = 1;   // message buffer interrupt disabled
   Fray_header->txm = 1;   // transmission mode - continuous mode
   Fray_header->ppit = 0;  // Payload Preamble Indicator is not set
   Fray_header->cfg = 0; // message buffer configuration bit (0= receive buffer, 1 =
transmit buffer)
   Fray_header->chb = 1;   // Ch B
   Fray_header->cha = 1;   // Ch A
   Fray_header->cyc = 0;   // Cycle Filtering Code (no cycle filtering)
   Fray_header->fid = 0;   // Frame ID
   //Header Section 2 (WRHS2)
   Fray_header->plc = 0;   // Payload Length configured
   Fray_header->crc = 0;

   //Header Section 3 (WRHS3)
   Fray_header->dp = 0;   // Pointer to start of data in message RAM

   //Buffer Command Global Config
   //Output Buffer
   Fray_buff_comand->rdss = 0;  // read data section                OBCM
   Fray_buff_comand->rhss = 0;  // read header section              OBCM
   Fray_buff_comand->obrs = 0;  // output buffer number             OBCR

   //Input Buffer
   Fray_buff_comand->stxrh = 0; // set transmission request         IBCM  0
   Fray_buff_comand->ldsh = 0; // load data section                 IBCM  0
   Fray_buff_comand->lhsh = 1; // load header section               IBCM  1
   Fray_buff_comand->ibrh = 0; // input buffer number               IBCR  0
   Fray_buff_comand->ibsyh = 1; // check for input buffer busy host      IBCR
   Fray_buff_comand->ibsys = 1; // check for input buffer busy shadow    IBCR

   // Message buffers Particular Config

//=====================================================
===Static segment
   // Buffer #1 Transmit buffer
   Fray_header->fid = 1;       // frame ID
   Fray_header->cfg = 1;       // Transmit buffer
   Fray_header->plc = 9; // 18 byte payload length configured (0xFF <=> 1 byte)
   Fray_header->dp = 0x80;     // Pointer to start of data in message RAM
   Fray_header->rcc = 0;      //Receive cycle count
   Fray_header->rci = 0;      //Received on channel indicator
   Fray_header->sfi = 1;      //Startup frame indicator
   Fray_header->syn = 1;      //Sync frame indicator
   Fray_header->nfi = 0;      //Null frame indicator
   Fray_header->ppi = 0;      //Payload preamble indicator
   Fray_header->res = 0;      //Reserved bit
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 0;  // input buffer number              IBCR
   Fray_buffConfig(Fray_PST, Fray_header);
   Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

   // buffer #2 Transmit buffer
   Fray_header->fid = 2;   // frame ID
   Fray_header->dp = 0x90; // Pointer to start of data in message RAM
   Fray_header->cfg = 1;    // Transmit buffer
   Fray_header->syn = 0;    // sync frame indicator
   Fray_header->sfi = 0;    // startup frame indicator
   Fray_header->cha = 1;    // Transmission on Ch A
   Fray_header->chb = 1;    // Transmission on Ch B
   Fray_header->plc = 9;  // 254 byte payload
   Fray_header->crc = header_crc_calc(Fray_header);
   Fray_buff_comand->ibrh = 1;  // input buffer number
```

```
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #3 Transmit buffer
    Fray_header->fid = 3;   // frame ID
    Fray_header->dp = 0x100; // Pointer to start of data in message RAM
    Fray_header->cfg = 1;    // Transmit buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Transmission on Ch A
    Fray_header->chb = 1;    // Transmission on Ch B
    Fray_header->plc = 9;  // 254 byte payload
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 2;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #4 Receive buffer
    Fray_header->fid = 4;    // frame ID
    Fray_header->dp = 0x200; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 1;    // Receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 3;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);


//===================================================
===Dynamic segment
    // buffer #10 Transmit buffer
    Fray_header->fid = 90;   // frame ID
    Fray_header->dp = 0x110; // Pointer to start of data in message RAM
    Fray_header->cfg = 1;    // Transmit buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Transmission on Ch A
    Fray_header->chb = 0;    // No transmission on Ch B
    Fray_header->plc = 9;  // 254 byte payload
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 10;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #11 Transmit buffer
    Fray_header->fid = 91;   // frame ID
    Fray_header->dp = 0x120; // Pointer to start of data in message RAM
    Fray_header->cfg = 1;    // Transmit buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Transmission on Ch A
    Fray_header->chb = 0;    // No transmission on Ch B
    Fray_header->plc = 9;  // 254 byte payload
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 11;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #12 Transmit buffer
    Fray_header->fid = 92;   // frame ID
    Fray_header->dp = 0x130; // Pointer to start of data in message RAM
    Fray_header->cfg = 1;    // Transmit buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Transmission on Ch A
    Fray_header->chb = 0;    // No transmission on Ch B
    Fray_header->plc = 9;  // 254 byte payload
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 12;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #13 Receive buffer
    Fray_header->fid = 93;    // frame ID
    Fray_header->dp = 0x210; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 13;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #14 Receive buffer
    Fray_header->fid = 94;    // frame ID
    Fray_header->dp = 0x220; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 14;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    uh = Fr_ControllerInit(Fray_PST);
```

```
   // Initialize Interrupts
   Fray_PST->EIR_UN.EIR_UL = 0xFFFFFFFF; // Clear Error Interrupt Register
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF; // Clear Status Interrupt Register
   Fray_PST->SILS_UN.SILS_UL = 0x00000000; // all Status Interrupt Line Select
Register Interrupt is assigned to interrupt line CC_int0.
   Fray_PST->SIER_UN.SIER_UL = 0xFFFFFFFF; // Disable all Status Int.
   Fray_PST->SIES_UN.SIES_UL = 0x00000004; // Enable Cycle start interrupt is
enabled (CYCSE)
   Fray_PST->ILE_UN.ILE_ST.eint1_B1 = 1; // enable eray_int1

   Fr_AllowColdStart(Fray_PST);
}


void comunicacion_Nodo_A(FRAY_ST *Fray_PST)
{
   panelsen_t* psensores = &panel_sensores;
   panelact_t* pindicadores = &panel_indicadores;
   trama_t sistema;
   int i, plr, plc;
   bc buff_comand_output;
   unsigned long ndat1;
   fray_buff bufer;
   unsigned long* datos = (unsigned long*) Fray_PST->WRDS;

   bc buff_comand_input;

   buff_comand_input.stxrh = 1;  // set transmission request          IBCM
   buff_comand_input.ldsh = 1;  // load data section                 IBCM
   buff_comand_input.lhsh = 0;  // load header section               IBCM
   buff_comand_input.ibrh = 0;  // input buffer number               IBCR
   buff_comand_input.ibsys = 0; // check for input buffer busy shadow  IBCR
   buff_comand_input.ibsyh = 1; // check for input buffer busy host    IBCR

   //wait for cycle start interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;         // clear all status int. flags
   while (Fray_PST->SIR_UN.SIR_ST.cycs_B1 == 0x0)
      ;    // wait for CYCS (Cycle Start Interrupt) interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;         // clear all status int. flags

   buff_comand_input.ibrh = slot_Stup
   ;
   datos[0] = psensores->imp_Front_Cond;
   datos[1] = psensores->imp_Front_Pasj;
   datos[2] = psensores->imp_Late_Cond;
   datos[3] = psensores->imp_Late_Pasj;
   datos[4] = psensores->imp_Trasero;
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);

   buff_comand_input.ibrh = slot1
   ;
   datos[0] = psensores->seg_Front_Cond;
   datos[1] = psensores->seg_Front_Pasj;
```

```
datos[2] = psensores->seg_Late_Cond;
datos[3] = psensores->seg_Late_Pasj;
datos[4] = psensores->seg_Trasero;
Fray_Transfer_Input(Fray_PST, &buff_comand_input);

buff_comand_input.ibrh = slot2
;
datos[0] = psensores->velocidad;
datos[1] = psensores->encendido;
Fray_Transfer_Input(Fray_PST, &buff_comand_input);


if (dinaseg_1==true)
{
   buff_comand_input.ibrh = mslot10
   ;
   datos[0] = psensores->SBE_Cond;
   datos[1] = psensores->SBE_Pasj;
   Escritura_Trama_Dinamica(0x0, 90, 2, datos);    //Seg Dinamico 1
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);
   dinaseg_1 = false;
}

if (dinaseg_2==true)
{
   buff_comand_input.ibrh = mslot11
   ;
   datos[0] = psensores->scint_Cond;
   datos[1] = psensores->scint_Pasj;
   Escritura_Trama_Dinamica(0x0, 91, 2, datos);    //Seg Dinamico 2
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);
   dinaseg_2=false;
}

if (dinaseg_3 == true)
   {
      buff_comand_input.ldsh = 1; // load data section                  IBCM  0
      buff_comand_input.lhsh = 1; // load header section
      buff_comand_input.ibrh = mslot12;
      get_Umbrales(&MRSIII);
      datos[0] = MRSIII.Estado;
      datos[1] = MRSIII.Umbral_pretensor;
      datos[2] = MRSIII.Umbral_activacion_frontal;
      datos[3] = MRSIII.Umbral_choque_trasero;
      Escritura_Trama_Dinamica(0x0, 92, 4, datos);    //Seg Dinamico 3
      Fray_Transfer_Input(Fray_PST, &buff_comand_input);
      dinaseg_3 = false;
   }
else if (frameMOD == 0x1)
{
   buff_comand_input.ldsh = 1; // load data section                  IBCM  0
   buff_comand_input.lhsh = 1; // load header section
   buff_comand_input.ibrh = mslot12;
```

```
        int i;
        for (i = 0; i < interfaz_trama_in.longt_payload; i++)
        {
           datos[i] = interfaz_trama_in.dato[i];
        }
        Escritura_Trama_Dinamica(interfaz_trama_in.seg_startup_UN.seg_startup_UL,
                       interfaz_trama_in.ID_trama,
                       interfaz_trama_in.longt_payload, datos); //Seg Dinamico 3
        Fray_Transfer_Input(Fray_PST, &buff_comand_input);
        frameMOD=0;
    }


// check received frames
    ndat1 = Fray_PST->NDAT1_UN.NDAT1_UL;

    if (ndat1 != 0)
    {
        buff_comand_output.rdss = 1;  // read data section
        buff_comand_output.rhss = 1;  // read header section

        if (ndat1 & 0x8)  //buffer #4 Estado del MRS III
        {
           buff_comand_output.obrs = slot3; // output buffer number
           Fray_Transfer_Output(Fray_PST, &buff_comand_output);
           //Leer el búfer de mensajes
           bufer.Buff_Headers.HEADER1_UN.HEADER1_UL =
                  Fray_PST->RDHS1_UN.RDHS1_UL;
           bufer.Buff_Headers.HEADER2_UN.HEADER2_UL =
                  Fray_PST->RDHS2_UN.RDHS2_UL;
           bufer.Buff_Headers.HEADER3_UN.HEADER3_UL =
                  Fray_PST->RDHS3_UN.RDHS3_UL;
           bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL =
                  Fray_PST->MBS_UN.MBS_UL;
           bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

           plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
           plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
           if (plr == plc)
              for (i = 0; i < plr; i++)
              {
                  bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
              }
           Lectura_trama(&sistema, &bufer);

           MRSIII.Activacion = sistema.dato[0];
           MRSIII.Estado = sistema.dato[1];

        }
        if (ndat1 & 0x2000)    //buffer #13 Valores de configuración MRS III
        {
           buff_comand_output.obrs = mslot13; // output buffer number
```

```
            Fray_Transfer_Output(Fray_PST, &buff_comand_output); //Leer el búfer de
mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL =
                    Fray_PST->RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL =
                    Fray_PST->RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL =
                    Fray_PST->RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL =
                    Fray_PST->MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
                    bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
                }
            Lectura_trama(&sistema, &bufer);

            MRSIII.Estado = sistema.dato[0];
            MRSIII.Umbral_pretensor = sistema.dato[1];
            MRSIII.Umbral_activacion_frontal = sistema.dato[2];
            MRSIII.Umbral_choque_trasero = sistema.dato[3];
//            MRSIII.Umbral_BST = sistema.dato[4];
//            MRSIII.Umbral_ITS = sistema.dato[5];
            set_Umbrales(&MRSIII);

        }
        if (ndat1 & 0x4000)    //buffer #14 Estado del panel indicadores
        {
            buff_comand_output.obrs = mslot14;  // output buffer number
            Fray_Transfer_Output(Fray_PST, &buff_comand_output);
            //Leer el búfer de mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL =
                    Fray_PST->RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL =
                    Fray_PST->RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL =
                    Fray_PST->RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL =
                    Fray_PST->MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
                    bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
                }
            Lectura_trama(&sistema, &bufer);
            pindicadores->Indicadores_UN.Indicadores_UL = (sistema.dato[1] << 8)
                    + sistema.dato[0];
        }
    }
}

void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,
                    unsigned plc, unsigned long* datos)
/*******************************************************************
****
 segment startup => (ppi)(nfi)(syn)(sfi)
 frame ID
 payload length configured (0xFF <=> 1 byte)
 Datos

*******************************************************************
****/
{
    trama_t sistema;
    sistema.seg_startup_UN.seg_startup_UL = seg_startup;

    wrhs header;
    header.cfg = 1;    // Transmit buffer
    header.syn = sistema.seg_startup_UN.seg_startup_ST.sincrona;
    header.sfi = sistema.seg_startup_UN.seg_startup_ST.startup;
    header.fid = fid;
    header.plc = plc;
    header.rcc = 0;      //Receive cycle count
    header.cha = 1;    // Transmission on Ch A
    header.chb = 0;    // No transmission on Ch B

    switch(fid){
    case 90:
        header.dp = 0x110; // Pointer to start of data in message RAM
        break;
    case 91:
        header.dp = 0x120; // Pointer to start of data in message RAM
        break;
    case 92:
        header.dp = 0x130; // Pointer to start of data in message RAM
        break;
    case 93:
        header.dp = 0x210; // Pointer to start of data in message RAM
        break;
    case 94:
        header.dp = 0x220; // Pointer to start of data in message RAM
        break;
    default:
        header.dp = 0x240; // Pointer to start of data in message RAM
    }
    sistema.CRC_cabecera = header_crc_calc(&header); // calculated  by the host
    Fray_buffConfig(frayREG, &header);
}
```

Anexo G. Código módulo FlexRay B

```
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>

wrhs header_reg; //Variables globales
cfg Fray_Config;
bc Fray_buffcomand1;
bc Fray_buffcomand2;
int uh;

extern panelsen_t panel_sensores;
extern panelact_t panel_indicadores;
extern MRS_t MRSIII;
extern unsigned datos_carga_util[64];
extern volatile boolean dinaseg_1;
extern volatile boolean dinaseg_2;
extern volatile boolean dinaseg_3;
extern volatile boolean dinaseg_4;
extern volatile boolean dinaseg_5;
extern volatile unsigned modo_config;
extern volatile unsigned frameMOD;
extern trama_t interfaz_trama_in;
extern unsigned EnableMRSIII;


void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,unsigned
plc, unsigned long* datos);

void configure_initialize_node_b(FRAY_ST *Fray_PST)
{
   wrhs *Fray_header = &header_reg;
   cfg *Fr_ConfigPtr = &Fray_Config;
   bc *Fray_buff_comand = &Fray_buffcomand1;

   // GTUs, PRTC configuration
   Fr_ConfigPtr->gtu1 = 0x00036B00; // pMicroPerCycle = 224000d = 36B00h
   Fr_ConfigPtr->gtu2 = 0x000F15E0; // gSyncodeMax = Fh, gMacroPerCycle =
5600d = 15E0h
   Fr_ConfigPtr->gtu3 = 0x00061818; // gMacroInitialOffset = 6h,
pMicroInitialOffset = 24d = 18h
   Fr_ConfigPtr->gtu4 = 0x0AE40AE3; // gOffsetCorrectionStart - 1 = 2788d =
AE4h, gMacroPerCycle - gdNIT - 1 = 2787d = AE3h
   Fr_ConfigPtr->gtu5 = 0x33010303; // pDecodingCorrection = 51d = 33h,
pClusterDriftDamping = 1h, pDelayCompensation = 3h
   Fr_ConfigPtr->gtu6 = 0x01510081; // pdMaxDrift = 337d = 151h,
pdAcceptedStartupRange = 129d = 81h
   Fr_ConfigPtr->gtu7 = 0x00080056; // gNumberOfStaticSlots = 8h, gdStaticSlot =
86d = 56h
```

```
    Fr_ConfigPtr->gtu8 = 0x015A0004; // gNumberOfMinislots = 346d = 15Ah,
gdMinislot = 4h
    Fr_ConfigPtr->gtu9 = 0x00010204; // gdDynamicSlotIdlePhase = 1,
gdMinislotActionPointOffset = 2, gdActionPointOffset = 4h
    Fr_ConfigPtr->gtu10 = 0x015100CD; // pRateCorrectionOut = 337d = 151h,
pOffsetCorrectionOut = 205d = CDh
    Fr_ConfigPtr->gtu11 = 0x00000000; // pExternRateCorrection = 0,
pExternOffsetCorrection = 0, no ext. clk. corr.
    Fr_ConfigPtr->succ2 = 0x0F036DA2; // gListenNoise = Fh, pdListenTimeout =
224674d = 36DA2h
    Fr_ConfigPtr->succ3 = 0x000000FF; // gMaxWithoutClockCorrectionFatal = Fh ,
passive = Fh
    Fr_ConfigPtr->prtc1 = 0x084C000A; // pWakeupPattern = 2h,
gdWakeupSymbolRxWindow = 76d, BRP = 0, gdTSSTransmitter = Ah
    Fr_ConfigPtr->prtc2 = 0x3CB41212; // gdWakeupSymbolTxLow = 60d,
gdWakeupSymbolTxIdle = 180d, gdWakeupSymbolRxLow = 18d,
gdWakeupSymbolRxIdle = 18d
    Fr_ConfigPtr->mhdc = 0x010D0009; // pLatestTransmit = 269d = 010Dh,
gPayloadLengthStatic = 9h
    Fr_ConfigPtr->mrc = 0x00174006; // LCB=23d, FFB=64d, FDB=4d (0..3 static,
4..23 dyn., 0 fifo)

    // Wait for PBSY bit to clear - POC not busy
    while (Fray_PST->SUCC1_UN.SUCC1_ST.pbsy_B1 != 0)
        ;

    // Initialize
    Fr_Init(Fray_PST, Fr_ConfigPtr);

    //Header Section Global Config
    //Header Section Register 1 (WRHS1)
    Fray_header->mbi = 1;   // message buffer interrupt disabled
    Fray_header->txm = 1;   // transmission mode - continuous mode
    Fray_header->ppit = 0;  // Payload Preamble Indicator is not set
    Fray_header->cfg = 0; // message buffer configuration bit (0= receive buffer, 1 =
transmit buffer)
    Fray_header->chb = 1;   // Ch B
    Fray_header->cha = 1;   // Ch A
    Fray_header->cyc = 0;   // Cycle Filtering Code (no cycle filtering)
    Fray_header->fid = 0;   // Frame ID

    //Header Section 2 (WRHS2)
    Fray_header->plc = 0;   // Payload Length configured
    Fray_header->crc = 0;  // calculated  by the host

    //Header Section 3 (WRHS3)
    Fray_header->dp = 0;   // Pointer to start of data in message RAM
    Fray_header->sfi = 0;   // startup frame indicator
    Fray_header->syn = 0;   // sync frame indicator

    //Buffer Command Global Config
    //Output Buffer
    Fray_buff_comand->rdss = 0;  // read data section                    OBCM
    Fray_buff_comand->rhss = 0;  // read header section                  OBCM
    Fray_buff_comand->obrs = 0;  // output buffer number                  OBCR

    //Input Buffer
    Fray_buff_comand->stxrh = 0; // set transmission request             IBCM
    Fray_buff_comand->ldsh = 0; // load data section                   IBCM
    Fray_buff_comand->lhsh = 1; // load header section                   IBCM
    Fray_buff_comand->ibrh = 0; // input buffer number                   IBCR
    Fray_buff_comand->ibsyh = 1; // check for input buffer busy host      IBCR
    Fray_buff_comand->ibsys = 1; // check for input buffer busy shadow    IBCR

    // Message buffers Particular Config
//=====================================================
===Static segment
    // Buffer #4 Transmit buffer
    Fray_header->fid = 4;       // frame ID
    Fray_header->cfg = 1;       // Transmit buffer
    Fray_header->plc = 9; // 18 byte payload length configured (0xFF <=> 1 byte)
    Fray_header->dp = 0x200;     // Pointer to start of data in message RAM
    Fray_header->rcc = 0;      //Receive cycle count
    Fray_header->rci = 0;      //Received on channel indicator
    Fray_header->sfi = 1;      //Startup frame indicator
    Fray_header->syn = 1;      //Sync frame indicator
    Fray_header->nfi = 0;      //Null frame indicator
    Fray_header->ppi = 0;      //Payload preamble indicator
    Fray_header->res = 0;      //Reserved bit
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 0;  // input buffer number                  IBCR
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #1 Receive buffer
    Fray_header->fid = 1;    // frame ID
    Fray_header->dp = 0x80; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 1;    // Receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 1;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #2 Receive buffer
    Fray_header->fid = 2;    // frame ID
    Fray_header->dp = 0x90; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 1;    // Receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 2;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #3 Receive buffer
    Fray_header->fid = 3;    // frame ID
    Fray_header->dp = 0x100; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 1;    // Receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 3;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);


//=====================================================
===Dynamic segment
    // buffer #13 Transmit buffer
    Fray_header->fid = 93;   // frame ID
    Fray_header->dp = 0x210; // Pointer to start of data in message RAM
    Fray_header->cfg = 1;    // Transmit buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Transmission on Ch A
    Fray_header->chb = 0;    // No transmission on Ch B
    Fray_header->plc = 9;  // 254 byte payload
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 13;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #14 Transmit buffer
    Fray_header->fid = 94;   // frame ID
    Fray_header->dp = 0x220; // Pointer to start of data in message RAM
    Fray_header->cfg = 1;    // Transmit buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Transmission on Ch A
    Fray_header->chb = 0;    // No transmission on Ch B
    Fray_header->plc = 9;  // 254 byte payload
    Fray_header->crc = header_crc_calc(Fray_header);
    Fray_buff_comand->ibrh = 14;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #10 Receive buffer
    Fray_header->fid = 90;    // frame ID
    Fray_header->dp = 0x110; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 10;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #11 Receive buffer
    Fray_header->fid = 91;    // frame ID
    Fray_header->dp = 0x120; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 11;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    // buffer #12 Receive buffer
    Fray_header->fid = 92;    // frame ID
    Fray_header->dp = 0x130; // Pointer to start of data in message RAM
    Fray_header->cfg = 0;    // Receive buffer
    Fray_header->syn = 0;    // sync frame indicator
    Fray_header->sfi = 0;    // startup frame indicator
    Fray_header->cha = 1;    // Receive on Ch A
    Fray_header->chb = 0;    // No receive on Ch B
    Fray_header->plc = 9;    // 18 byte payload
    Fray_header->crc = 0;
    Fray_buff_comand->ibrh = 12;  // input buffer number
    Fray_buffConfig(Fray_PST, Fray_header);
    Fray_Transfer_Input(Fray_PST, Fray_buff_comand);

    Fr_ControllerInit(Fray_PST);

    // Initialize Interrupts
    Fray_PST->EIR_UN.EIR_UL = 0xFFFFFFFF; // Clear Error Int.
    Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF; // Clear Status Int.
    Fray_PST->SILS_UN.SILS_UL = 0x00000000; // all Status Int. to eray_int0
    Fray_PST->SIER_UN.SIER_UL = 0xFFFFFFFF; // Disable all Status Int.
    Fray_PST->SIES_UN.SIES_UL = 0x00000004; // Enable CYCSE Int.
    Fray_PST->ILE_UN.ILE_UL = 0x00000002; // enable eray_int1

    Fr_AllowColdStart(Fray_PST);
}
```

```
void comunicacion_Nodo_B(FRAY_ST *Fray_PST)
{
   panelsen_t* psensores = &panel_sensores;
   panelact_t* pindicadores = &panel_indicadores;
   trama_t sistema;
   int i, plr, plc;
   bc buff_comand_output;
   unsigned long ndat1;
   fray_buff bufer;
   unsigned long* datos = (unsigned long*) Fray_PST->WRDS;

   bc buff_comand_input;
   buff_comand_input.stxrh = 1;  // set transmission request           IBCM
   buff_comand_input.ldsh = 1;  // load data section                  IBCM
   buff_comand_input.lhsh = 0;  // load header section                 IBCM
   buff_comand_input.ibrh = 0;  // input buffer number                 IBCR
   buff_comand_input.ibsys = 0; // check for input buffer busy shadow  IBCR
   buff_comand_input.ibsyh = 1; // check for input buffer busy host    IBCR

   //wait for cycle start interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;         // clear all status int. flags
   while (Fray_PST->SIR_UN.SIR_ST.cycs_B1 == 0x0)
      ;    // wait for CYCS (Cycle Start Interrupt) interrupt flag
   Fray_PST->SIR_UN.SIR_UL = 0xFFFFFFFF;         // clear all status int. flags

   buff_comand_input.ibrh = slot_Stup;
   datos[0] = MRSIII.Activacion;
   datos[1] = MRSIII.Estado;
   Fray_Transfer_Input(Fray_PST, &buff_comand_input);

   if (dinaseg_4==true) //Configuracion MRS III
   {
      buff_comand_input.ibrh = mslot13
      ;
      get_Umbrales(&MRSIII);
      datos[0] = MRSIII.Estado;
      datos[1] = MRSIII.Umbral_pretensor;
      datos[2] = MRSIII.Umbral_activacion_frontal;
      datos[3] = MRSIII.Umbral_choque_trasero;
      datos[4] = MRSIII.Umbral_BST;
      datos[5] = MRSIII.Umbral_ITS;
      Escritura_Trama_Dinamica(0x0, 93, 6, datos);    //Seg Dinamico 4
      Fray_Transfer_Input(Fray_PST, &buff_comand_input);
      dinaseg_4=false;
   }

   if (dinaseg_5==true)    //Estado panel
   {
      buff_comand_input.ibrh = mslot14
      ;
      datos[0] = pindicadores->Indicadores_UN.Indicadores_UL & 0xFF;
      datos[1] = (pindicadores->Indicadores_UN.Indicadores_UL >> 8) & 0xFF;
```

```
        Escritura_Trama_Dinamica(0x0, 94, 2, datos);    //Seg Dynamic 5
        Fray_Transfer_Input(Fray_PST, &buff_comand_input);
        dinaseg_5=false;
    }

// check received frames
    ndat1 = Fray_PST->NDAT1_UN.NDAT1_UL;

    if (ndat1 != 0)
    {
        buff_comand_output.rdss = 1;  // read data section
        buff_comand_output.rhss = 1;  // read header section

        if (ndat1 & 0x2)    //buffer #1
        {
            buff_comand_output.obrs = slot1
            ;  // output buffer number
            Fray_Transfer_Output(Fray_PST, &buff_comand_output);
            //Leer el búfer de mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
                    bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
                }
            Lectura_trama(&sistema, &bufer);

            psensores->imp_Front_Cond = sistema.dato[0];
            psensores->imp_Front_Pasj = sistema.dato[1];
            psensores->imp_Late_Cond = sistema.dato[2];
            psensores->imp_Late_Pasj = sistema.dato[3];
            psensores->imp_Trasero = sistema.dato[4];

        }
        if (ndat1 & 0x04)    //buffer #2
        {
            buff_comand_output.obrs = slot2
            ;
            // output buffer number
            Fray_Transfer_Output(Fray_PST, &buff_comand_output);
            //Leer el búfer de mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
                    bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
                }
            Lectura_trama(&sistema, &bufer);

            psensores->seg_Front_Cond = sistema.dato[0];
            psensores->seg_Front_Pasj = sistema.dato[1];
            psensores->seg_Late_Cond = sistema.dato[2];
            psensores->seg_Late_Pasj = sistema.dato[3];
            psensores->seg_Trasero = sistema.dato[4];

        }
        if (ndat1 & 0x08)    //buffer #3
        {
            buff_comand_output.obrs = slot3
            ;
            // output buffer number
            Fray_Transfer_Output(Fray_PST, &buff_comand_output);
            //Leer el búfer de mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
                    bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
                }
            Lectura_trama(&sistema, &bufer);

            psensores->velocidad = sistema.dato[0];
            psensores->encendido = sistema.dato[1];

        }
        if (ndat1 & 0x400)    //buffer #10
        {
            buff_comand_output.obrs = mslot10
            ;  // output buffer number
            Fray_Transfer_Output(Fray_PST, &buff_comand_output);
            //Leer el búfer de mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
                    bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
                }
            Lectura_trama(&sistema, &bufer);

            psensores->SBE_Cond = sistema.dato[0];
            psensores->SBE_Pasj = sistema.dato[1];

        }
        if (ndat1 & 0x0800)    //buffer #11
        {
            buff_comand_output.obrs = mslot11
            ;  // output buffer number
            Fray_Transfer_Output(Fray_PST, &buff_comand_output);
            //Leer el búfer de mensajes
            bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
            bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
            bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
            bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
            bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

            plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
            plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
            if (plr == plc)
                for (i = 0; i < plr; i++)
                {
```

```
                bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
            }
        Lectura_trama(&sistema, &bufer);

        psensores->scint_Cond = sistema.dato[0];
        psensores->scint_Pasj = sistema.dato[1];
    }
    if (ndat1 & 0x1000)    //buffer #12
    {
        buff_comand_output.obrs = mslot12
        ;  // output buffer number
        Fray_Transfer_Output(Fray_PST, &buff_comand_output);
        //Leer el búfer de mensajes
        bufer.Buff_Headers.HEADER1_UN.HEADER1_UL = Fray_PST-
>RDHS1_UN.RDHS1_UL;
        bufer.Buff_Headers.HEADER2_UN.HEADER2_UL = Fray_PST-
>RDHS2_UN.RDHS2_UL;
        bufer.Buff_Headers.HEADER3_UN.HEADER3_UL = Fray_PST-
>RDHS3_UN.RDHS3_UL;
        bufer.Buff_Headers.BUFFER_STATUS_UN.BUFFER_STATUS_UL = Fray_PST-
>MBS_UN.MBS_UL;
        bufer.ranura = Fray_PST->OBCR_UN.OBCR_ST.obrh_B7;

        plc = Fray_PST->RDHS2_UN.RDHS2_ST.plc_B7;
        plr = Fray_PST->RDHS2_UN.RDHS2_ST.plr_B7;
        if (plr == plc)
            for (i = 0; i < plr; i++)
            {
                bufer.Buff_Data[i] = Fray_PST->RDDS[i] & 0xFF;
            }
        Lectura_trama(&sistema, &bufer);

        if(sistema.ID_trama==92){
            modo_config = 1;
            MRSIII.Estado = sistema.dato[0];
            MRSIII.Umbral_pretensor = sistema.dato[1];
            MRSIII.Umbral_activacion_frontal = sistema.dato[2];
            MRSIII.Umbral_choque_trasero = sistema.dato[3];
            set_Umbrales(&MRSIII);
        }
    }
  }
}

void Escritura_Trama_Dinamica(unsigned seg_startup, unsigned fid,
                 unsigned plc, unsigned long* datos)
/*****************************************************************
****
 segment startup => (ppi)(nfi)(syn)(sfi)
 frame ID
 payload length configured (0xFF <=> 1 byte)
 Datos
```

```
*******************************************************************
****/
{
   trama_t sistema;
   sistema.seg_startup_UN.seg_startup_UL = seg_startup;

   wrhs header;
   header.cfg = 1;    // Transmit buffer
   header.syn = sistema.seg_startup_UN.seg_startup_ST.sincrona;
   header.sfi = sistema.seg_startup_UN.seg_startup_ST.startup;
   header.fid = fid;
   header.plc = plc;
   header.rcc = 0;      //Receive cycle count
   header.cha = 1;    // Transmission on Ch A
   header.chb = 0;    // No transmission on Ch B

   switch(fid){
   case 90:
      header.dp = 0x110; // Pointer to start of data in message RAM
      break;
   case 91:
      header.dp = 0x120; // Pointer to start of data in message RAM
      break;
   case 92:
      header.dp = 0x130; // Pointer to start of data in message RAM
      break;
   case 93:
      header.dp = 0x210; // Pointer to start of data in message RAM
      break;
   case 94:
      header.dp = 0x220; // Pointer to start of data in message RAM
      break;
   }
   sistema.CRC_cabecera = header_crc_calc(&header); // calculated  by the host
   Fray_buffConfig(frayREG, &header);
}
```

Anexo H. Código proyecto nodo A (HL_sys_main.c)

```
/** @file HL_sys_main.c
*   @brief Application main file
*   @date 11-Dec-2018
*   @version 04.07.01
*
*   This file contains an empty main function,
*   which can be used for the application.
*/

/*
* Copyright (C) 2009-2018 Texas Instruments Incorporated - www.ti.com
*
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*    Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
*    Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the
*    distribution.
*
*    Neither the name of Texas Instruments Incorporated nor the names of
*    its contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
*  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
*  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*/
```

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include "HL_sys_common.h"

/* USER CODE BEGIN (1) */
#include "HL_sys_core.h"
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>
#include "HL_sci.h"
/* USER CODE END */

/** @fn void main(void)
*   @brief Application main function
*   @note This function is empty by default.
*
*   This function is called after startup.
*   The user can use this function to implement the application.
*/

/* USER CODE BEGIN (2) */
void comunicacion_Nodo_A(FRAY_ST *Fray_PST);
int lectura_Interfaz(trama_t* frame);
void envio_Interfaz(trama_t* frame);
void trama_Interfaz(trama_t* frame);
void response_Interfaz(trama_t* frame);


volatile int taskSist = 0;
volatile unsigned modo_lectura;
volatile boolean dinaseg_1;
volatile boolean dinaseg_2;
volatile boolean dinaseg_3;
volatile boolean dinaseg_4;
volatile boolean dinaseg_5;
int posicion;

trama_t interfaz_trama_in;
trama_t interfaz_trama_out;
panelsen_t panel_sensores;
panelact_t panel_indicadores;
MRS_t MRSIII;
FRAY_ST* FlexRay_CC = frayREG;
adcData_t adc_data[7];

unsigned datos_carga_util[64];
unsigned modo_config;
unsigned interfaz_in;
unsigned interfaz_out;
unsigned band_Vib;
unsigned frameMOD;
int response=0;

/* USER CODE END */

int main(void)
{
/* USER CODE BEGIN (3) */
   _enable_IRQ_interrupt_();

//===================================================
==========GPIO
   gioInit();

//===================================================
==========RTI
   rtiInit();
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE1);
   rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);

//===================================================
==========FlexRay
   Config_CC_Nodo_A(FlexRay_CC);
   Fr_StartCommunication(FlexRay_CC);

//===================================================
==========ADC
   adcInit();
   adcStartConversion(adcREG1, adcGROUP1);

//===================================================
==========HET
   hetInit();

//===================================================
==========UART
   sciInit();

//===================================================
==========Inicialización
   modo_lectura = 0;
   posicion = 0;
   dinaseg_1 = false;
   dinaseg_2 = false;
   dinaseg_3 = false;
   dinaseg_4 = false;
   dinaseg_5 = false;
   interfaz_in = 0;

   while (1)
   {
        switch (taskSist) //Planificador
        {
        case 1:
        {
            if (modo_lectura == 0)
                lectura_sensores(&panel_sensores);
            else if(interfaz_in == 0)
                interfaz_sensores(&panel_sensores);
            while(taskSist==1);
            break;
        }
        case 2:
        {
            _disable_IRQ_interrupt_();
            comunicacion_Nodo_A(FlexRay_CC);
            _enable_IRQ_interrupt_();
            while(taskSist==2);
            break;
        }
        case 3:
        {
            if (interfaz_in)
            {
                response = lectura_Interfaz(&interfaz_trama_in);
                if(response==1){
                    response_Interfaz(&interfaz_trama_out);
                    envio_Interfaz(&interfaz_trama_out);
                    response=0;
                }

                if (interfaz_trama_in.ID_trama == 0x9)
                    modo_lectura = interfaz_trama_in.dato[0] & 0x1;

                if (interfaz_trama_in.ID_trama == 92)
                {
                    dinaseg_3=true;
                    config_Umrales(&interfaz_trama_in);
                }
                else if(interfaz_trama_in.ID_trama > 94 && interfaz_trama_in.ID_trama
 < 101)
                    frameMOD = 0x1;
                interfaz_in = 0;
            }
            if (interfaz_out){
                trama_Interfaz(&interfaz_trama_out);
                envio_Interfaz(&interfaz_trama_out);
                interfaz_out=0;
            }
            while(taskSist==3);
            break;
        }
        }
    }
    /* USER CODE END */

    //return 0;
}


/* USER CODE BEGIN (4) */
/* USER CODE END */
```

Anexo I. Código proyecto nodo B (HL_sys_main.c)

```
/** @file HL_sys_main.c
*   @brief Application main file
*   @date 11-Dec-2018
*   @version 04.07.01
*
*   This file contains an empty main function,
*   which can be used for the application.
*/

/*
* Copyright (C) 2009-2018 Texas Instruments Incorporated - www.ti.com
*
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*    Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
*    Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the
*    distribution.
*
*    Neither the name of Texas Instruments Incorporated nor the names of
*    its contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
*  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
*  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*/
```

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include "HL_sys_common.h"

/* USER CODE BEGIN (1) */
#include "HL_sys_core.h"
#include <Codigos/fray.h>
#include <Codigos/Panel_Airbag.h>
#include <Codigos/MRSIII.h>
/* USER CODE END */

/** @fn void main(void)
*   @brief Application main function
*   @note This function is empty by default.
*
*   This function is called after startup.
*   The user can use this function to implement the application.
*/

/* USER CODE BEGIN (2) */
void comunicacion_Nodo_B(FRAY_ST *Fray_PST);
void lectura_estados();
void Operacion_MRSIII();


volatile int taskSist = 0;
volatile boolean dinaseg_1;
volatile boolean dinaseg_2;
volatile boolean dinaseg_3;
volatile boolean dinaseg_4;
volatile boolean dinaseg_5;
volatile unsigned modo_config;
volatile unsigned modo_lectura;
int posicion;

panelsen_t panel_sensores;
panelact_t panel_indicadores;
MRS_t MRSIII;
FRAY_ST* FlexRay_CC = frayREG;


/* USER CODE END */

int main(void)
{
/* USER CODE BEGIN (3) */
    _enable_IRQ_interrupt_();
```

```
//===========================================================
==========GPIO
   gioInit();

//===========================================================
==========RTI
   rtiInit();
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE1);
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE2);
   rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
   rtiStopCounter(rtiREG1, 1);

//===========================================================
==========FlexRay
   configure_initialize_node_b(FlexRay_CC);
   Fr_StartCommunication(FlexRay_CC);

//===========================================================
==========HET
   hetInit();

//===========================================================
==========Inicialización
   modo_lectura = 0;
   posicion = 0;
   dinaseg_1 = false;
   dinaseg_2 = false;
   dinaseg_3 = false;
   dinaseg_4 = false;
   dinaseg_5 = false;
   modo_lectura = 0;
   MRSIII.Estado = 0;
   MRSIII.Activacion = 0;
   posicion = 0;
   initUmbrales();
   reset_Indicadores();
   MRSIII.Estado = autocomprobacion_panel_Indica();
   dinaseg_4 = true;
   dinaseg_5 = true;

   while (1)
   {
      switch (taskSist) //Planificador
      {
      case 1:
      {
         if (dinaseg_4==true)
            get_Umbrales(&MRSIII);

         if (MRSIII.Activacion == 0)
         {
            if (MRSIII.Estado == 4)
            {
               MRSIII.Estado = autocomprobacion_panel_Indica();
               dinaseg_5=true;
            }
            else if (MRSIII.Estado != 1)
            {
               MRSIII.Estado = autocomprobacion_sensores();
               dinaseg_5=true;
            }
         }
         while (taskSist == 1);
         break;
      }
      case 2:
      {
         _disable_IRQ_interrupt_();
         comunicacion_Nodo_B(FlexRay_CC);
         _enable_IRQ_interrupt_();
         while (taskSist == 2);
         break;
      }
      case 3:
      {
         if (modo_config == 1)
         {
            set_Umbrales(&MRSIII);
            dinaseg_4=true;
            modo_config = 0;
         }


         if (MRSIII.Estado == 1)
         {
            ejecucion_MRSIII();
            dinaseg_5=true;
         }
         while (taskSist == 3);
         break;
      }
      }
   }
   /* USER CODE END */

   //return 0;
}


/* USER CODE BEGIN (4) */
/* USER CODE END */
```

Anexo J. Script Matlab del procesamiento digital de la trama FlexRay muestreada.

```
%% Import data from spreadsheet
% Script for importing data from the following spreadsheet:
%
%    Workbook: D:\Tesis\imagenes\tramas\Trama FlexRay_v2.xlsm
%    Worksheet: F0002CH1
%
% To extend the code for use with different selected data or a different
% spreadsheet, generate a function instead of a script.

% Auto-generated by MATLAB on 2021/10/17 15:28:33

%% Import the data
clear
clc
[~, ~, raw0_0] = xlsread('D:\Tesis\imagenes\tramas\Trama
FlexRay_v2.xlsm','F0002CH1','D1:E1400');
[~, ~, raw0_1] = xlsread('D:\Tesis\imagenes\tramas\Trama
FlexRay_v2.xlsm','F0002CH1','G1:G1400');
raw = [raw0_0,raw0_1];
raw(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw)) = {''};

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Create output variable
data = reshape([raw{:}],size(raw);

%% Allocate imported array to column variable names
BPH = data(:,1);
BMH = data(:,2);
%Tiempo = data(:,3);

N=size(BMH)*2;
Tiempo=zeros();
BP=zeros();
BM=zeros();
uBus=zeros();
Reloj=zeros();
Tiempo(1)=1e-8;
for i=2:1:N-2
    Tiempo(i,1)=Tiempo(i-1,1)+1e-8;
end

x=1;
band=1;
offset =0;
```

```
for i=1:1:N-2
   if(mod(i,2)==0)
      BP(i,1)=BPH(x,1)+offset;
      BM(i,1)=BMH(x,1)+offset;

   else
      BP(i,1)=(BPH(x,1)+BPH(x+1,1)/2+offset;
      BM(i,1)=(BMH(x,1)+BMH(x+1,1)/2+offset;
      Reloj(i,1)=-1;
      x=x+1;
   end
   if(mod(i,5)==0)
      band=band*-1;
   end
   if(band==1)
      Reloj(i,1)=.5;
   else
      Reloj(i,1)=-.5;
   end
   uBus(i,1)=BP(i,1)-BM(i,1);

end

figure

subplot(2,1,1)
plot(Tiempo,BP,Tiempo,BM,[0],[3.6])

title('Trama Diferencial FlexRay')
xlabel('Tiempo')
ylabel('Voltaje')
legend('BP','BM')
%set(gca,'xtick',[],'ytick',[])

subplot(2,1,2)
grid off;
plot(Tiempo,uBus,Tiempo,Reloj,[0],[2])

title('Codificación FlexRay')
xlabel('Tiempo')

ylabel('Amplitud')
legend('uBus','Reloj')
set(gca,'ytick',[])

%% Clear temporary variables
clearvars data raw raw0_0 raw0_1 R;
```