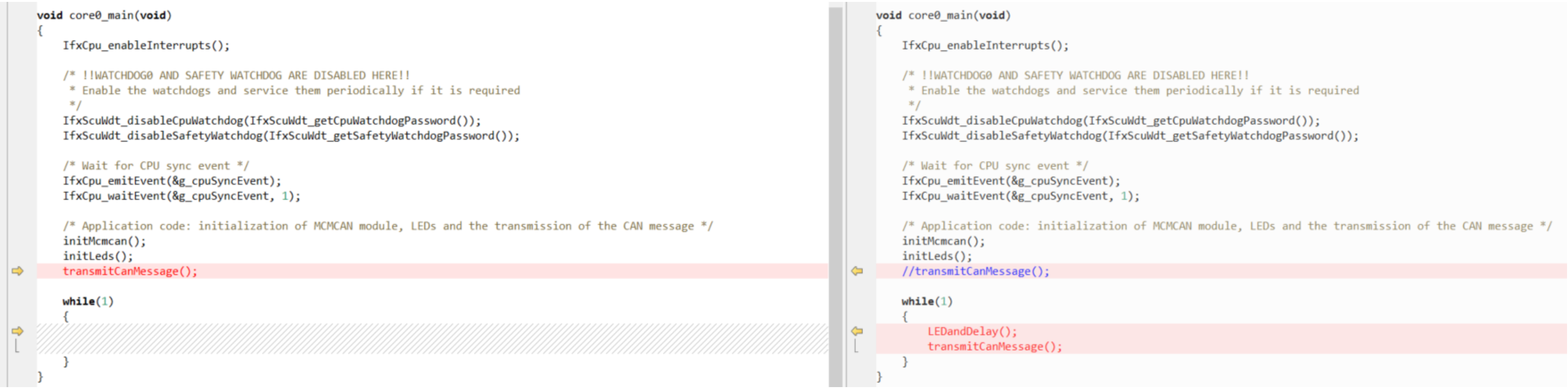
Step 1:

- cpu0_main.c
  - LED blinking ~3 seconds
  - Transmit a CAN packet out

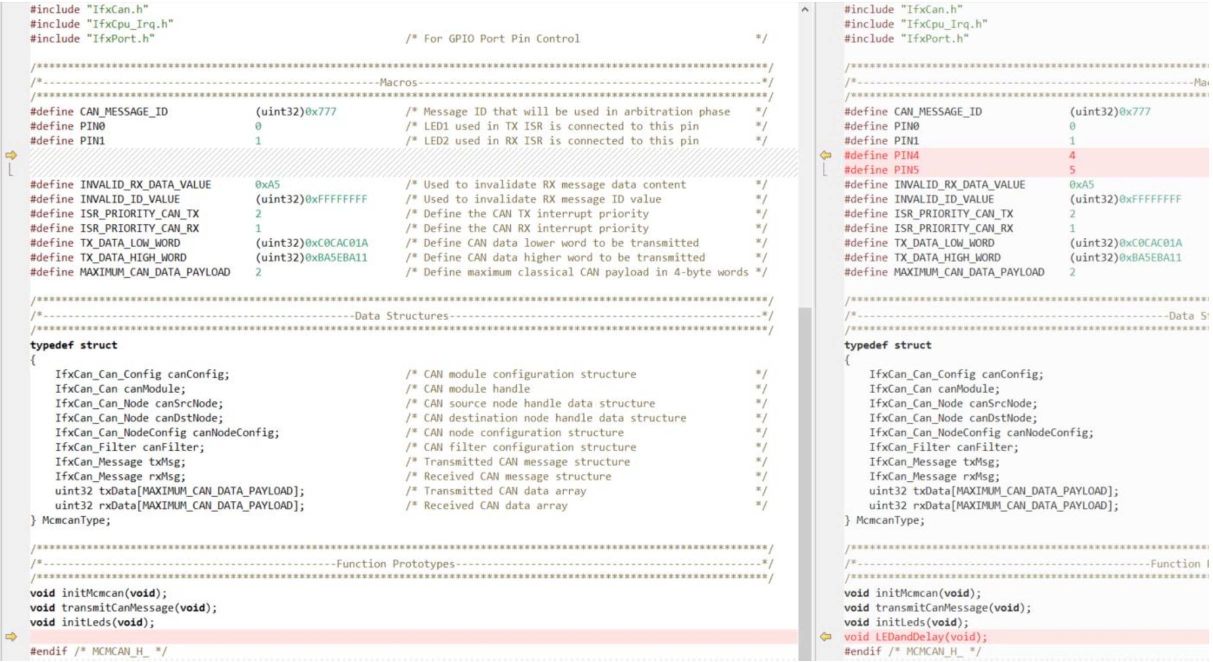


```
void core0_main(void)
{
    IfxCpu_enableInterrupts();

    /* !!WATCHDOG0 AND SAFETY WATCHDOG ARE DISABLED HERE!!
     * Enable the watchdogs and service them periodically if it is required
     */
    IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());
    IfxScuWdt_disableSafetyWatchdog(IfxScuWdt_getSafetyWatchdogPassword());

    /* Wait for CPU sync event */
    IfxCpu_emitEvent(&g_cpuSyncEvent);
    IfxCpu_waitEvent(&g_cpuSyncEvent, 1);

    /* Application code: initialization of MCMCAN module, LEDs and the transmission of the CAN message */
    initMcmcan();
    initLeds();
    transmitCanMessage();

    while(1)
    {


    }
}
```

```
void core0_main(void)
{
    IfxCpu_enableInterrupts();

    /* !!WATCHDOG0 AND SAFETY WATCHDOG ARE DISABLED HERE!!
     * Enable the watchdogs and service them periodically if it is required
     */
    IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());
    IfxScuWdt_disableSafetyWatchdog(IfxScuWdt_getSafetyWatchdogPassword());

    /* Wait for CPU sync event */
    IfxCpu_emitEvent(&g_cpuSyncEvent);
    IfxCpu_waitEvent(&g_cpuSyncEvent, 1);

    /* Application code: initialization of MCMCAN module, LEDs and the transmission of the CAN message */
    initMcmcan();
    initLeds();
    //transmitCanMessage();

    while(1)
    {
        LEDandDelay();
        transmitCanMessage();
    }
}
```

Step 2:

- Mcmcan.h
  - Add pin names
  - Add function names

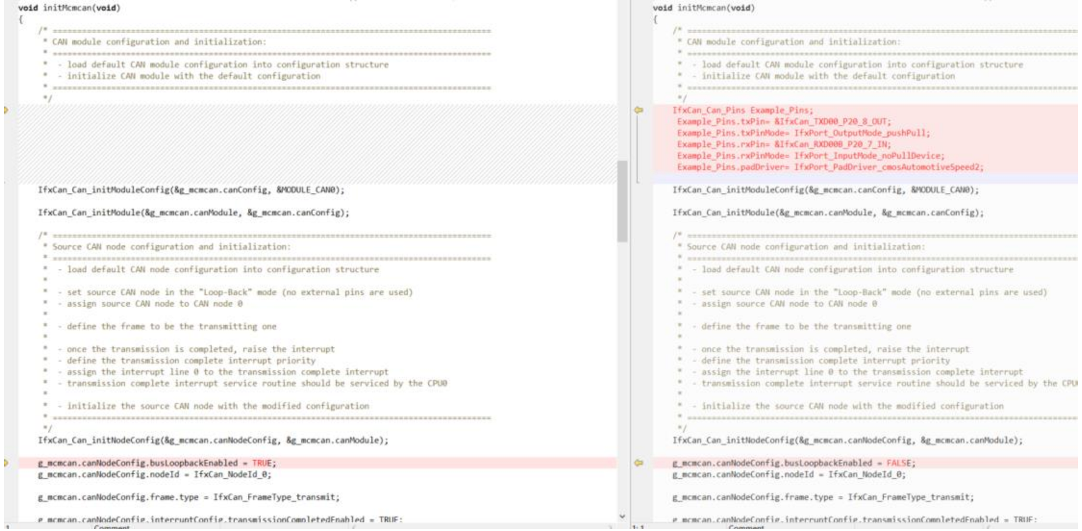


```
#include "IfxCan.h"
#include "IfxCpu_Irq.h"
#include "IfxPort.h"                                         /* For GPIO Port Pin Control                          */

/*********************************************************************************************************************/
/*-----------------------------------------------------Macros--------------------------------------------------------*/
/*********************************************************************************************************************/
#define CAN_MESSAGE_ID              (uint32)0x777           /* Message ID that will be used in arbitration phase   */
#define PIN0                        0                       /* LED1 used in TX ISR is connected to this pin        */
#define PIN1                        1                       /* LED2 used in RX ISR is connected to this pin        */


#define INVALID_RX_DATA_VALUE       0xA5                    /* Used to invalidate RX message data content          */
#define INVALID_ID_VALUE            (uint32)0xFFFFFFFF      /* Used to invalidate RX message ID value              */
#define ISR_PRIORITY_CAN_TX         2                       /* Define the CAN TX interrupt priority                 */
#define ISR_PRIORITY_CAN_RX         1                       /* Define the CAN RX interrupt priority                 */
#define TX_DATA_LOW_WORD            (uint32)0xC0CAC01A      /* Define CAN data lower word to be transmitted        */
#define TX_DATA_HIGH_WORD           (uint32)0xBA5EBA11      /* Define CAN data higher word to be transmitted       */
#define MAXIMUM_CAN_DATA_PAYLOAD    2                       /* Define maximum classical CAN payload in 4-byte words */

/*********************************************************************************************************************/
/*-------------------------------------------------Data Structures---------------------------------------------------*/
/*********************************************************************************************************************/
typedef struct
{
    IfxCan_Can_Config canConfig;                            /* CAN module configuration structure                  */
    IfxCan_Can canModule;                                   /* CAN module handle                                   */
    IfxCan_Can_Node canSrcNode;                             /* CAN source node handle data structure               */
    IfxCan_Can_Node canDstNode;                             /* CAN destination node handle data structure          */
    IfxCan_Can_NodeConfig canNodeConfig;                    /* CAN node configuration structure                    */
    IfxCan_Filter canFilter;                                /* CAN filter configuration structure                  */
    IfxCan_Message txMsg;                                   /* Transmitted CAN message structure                   */
    IfxCan_Message rxMsg;                                   /* Received CAN message structure                      */
    uint32 txData[MAXIMUM_CAN_DATA_PAYLOAD];                /* Transmitted CAN data array                          */
    uint32 rxData[MAXIMUM_CAN_DATA_PAYLOAD];                /* Received CAN data array                             */
} McmcanType;

/*********************************************************************************************************************/
/*-----------------------------------------------Function Prototypes-------------------------------------------------*/
/*********************************************************************************************************************/
void initMcmcan(void);
void transmitCanMessage(void);
void initLeds(void);

#endif /* MCMCAN_H_ */
```

```
#include "IfxCan.h"
#include "IfxCpu_Irq.h"
#include "IfxPort.h"

/*****************************************************************
/*-----------------------------------------------------Ma
/*****************************************************************
#define CAN_MESSAGE_ID              (uint32)0x777
#define PIN0                        0
#define PIN1                        1
#define PIN4                        4
#define PIN5                        5
#define INVALID_RX_DATA_VALUE       0xA5
#define INVALID_ID_VALUE            (uint32)0xFFFFFFFF
#define ISR_PRIORITY_CAN_TX         2
#define ISR_PRIORITY_CAN_RX         1
#define TX_DATA_LOW_WORD            (uint32)0xC0CAC01A
#define TX_DATA_HIGH_WORD           (uint32)0xBA5EBA11
#define MAXIMUM_CAN_DATA_PAYLOAD    2

/*****************************************************************
/*-------------------------------------------------Data S
/*****************************************************************
typedef struct
{
    IfxCan_Can_Config canConfig;
    IfxCan_Can canModule;
    IfxCan_Can_Node canSrcNode;
    IfxCan_Can_Node canDstNode;
    IfxCan_Can_NodeConfig canNodeConfig;
    IfxCan_Filter canFilter;
    IfxCan_Message txMsg;
    IfxCan_Message rxMsg;
    uint32 txData[MAXIMUM_CAN_DATA_PAYLOAD];
    uint32 rxData[MAXIMUM_CAN_DATA_PAYLOAD];
} McmcanType;

/*****************************************************************
/*-----------------------------------------------Function 
/*****************************************************************
void initMcmcan(void);
void transmitCanMessage(void);
void initLeds(void);
void LEDandDelay(void);
#endif /* MCMCAN_H_ */
```

Step 3:

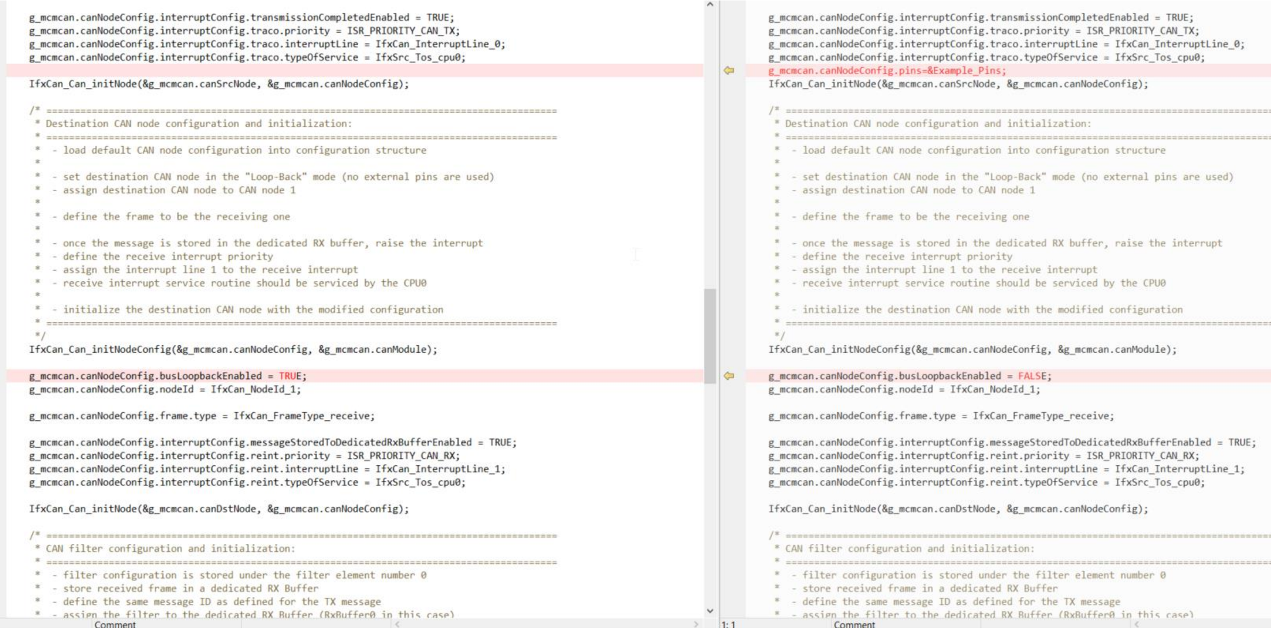
- Mcmcan.c
  - Add pin struct
  - Disable bus loopback for node 0 and node 1
  - canNodeConfig assigned pin struct member



```c
void initMcmcan(void)
{
    /* ==========================================================================================
     * CAN module configuration and initialization:
     * ==========================================================================================
     *  - load default CAN module configuration into configuration structure
     *  - initialize CAN module with the default configuration
     * ==========================================================================================
     */







    IfxCan_Can_initModuleConfig(&g_mcmcan.canConfig, &MODULE_CAN0);

    IfxCan_Can_initModule(&g_mcmcan.canModule, &g_mcmcan.canConfig);

    /* ==========================================================================================
     * Source CAN node configuration and initialization:
     * ==========================================================================================
     *  - load default CAN node configuration into configuration structure
     *
     *  - set source CAN node in the "Loop-Back" mode (no external pins are used)
     *  - assign source CAN node to CAN node 0
     *
     *  - define the frame to be the transmitting one
     *
     *  - once the transmission is completed, raise the interrupt
     *  - define the transmission complete interrupt priority
     *  - assign the interrupt line 0 to the transmission complete interrupt
     *  - transmission complete interrupt service routine should be serviced by the CPU0
     *
     *  - initialize the source CAN node with the modified configuration
     * ==========================================================================================
     */
    IfxCan_Can_initNodeConfig(&g_mcmcan.canNodeConfig, &g_mcmcan.canModule);

    g_mcmcan.canNodeConfig.busLoopbackEnabled = TRUE;
    g_mcmcan.canNodeConfig.nodeId = IfxCan_NodeId_0;

    g_mcmcan.canNodeConfig.frame.type = IfxCan_FrameType_transmit;

    g_mcmcan.canNodeConfig.interruptConfig.transmissionCompletedEnabled = TRUE;
```

```c
void initMcmcan(void)
{
    /* ==========================================================================================
     * CAN module configuration and initialization:
     * ==========================================================================================
     *  - load default CAN module configuration into configuration structure
     *  - initialize CAN module with the default configuration
     * ==========================================================================================
     */
    IfxCan_Can_Pins Example_Pins;
     Example_Pins.txPin= &IfxCan_TXD00_P20_8_OUT;
     Example_Pins.txPinMode= IfxPort_OutputMode_pushPull;
     Example_Pins.rxPin= &IfxCan_RXD00B_P20_7_IN;
     Example_Pins.rxPinMode= IfxPort_InputMode_noPullDevice;
     Example_Pins.padDriver= IfxPort_PadDriver_cmosAutomotiveSpeed2;

    IfxCan_Can_initModuleConfig(&g_mcmcan.canConfig, &MODULE_CAN0);

    IfxCan_Can_initModule(&g_mcmcan.canModule, &g_mcmcan.canConfig);

    /* ==========================================================================================
     * Source CAN node configuration and initialization:
     * ==========================================================================================
     *  - load default CAN node configuration into configuration structure
     *
     *  - set source CAN node in the "Loop-Back" mode (no external pins are used)
     *  - assign source CAN node to CAN node 0
     *
     *  - define the frame to be the transmitting one
     *
     *  - once the transmission is completed, raise the interrupt
     *  - define the transmission complete interrupt priority
     *  - assign the interrupt line 0 to the transmission complete interrupt
     *  - transmission complete interrupt service routine should be serviced by the CPU
     *
     *  - initialize the source CAN node with the modified configuration
     * ==========================================================================================
     */
    IfxCan_Can_initNodeConfig(&g_mcmcan.canNodeConfig, &g_mcmcan.canModule);

    g_mcmcan.canNodeConfig.busLoopbackEnabled = FALSE;
    g_mcmcan.canNodeConfig.nodeId = IfxCan_NodeId_0;

    g_mcmcan.canNodeConfig.frame.type = IfxCan_FrameType_transmit;

    g_mcmcan.canNodeConfig.interruptConfig.transmissionCompletedEnabled = TRUE;
```

Step 3:

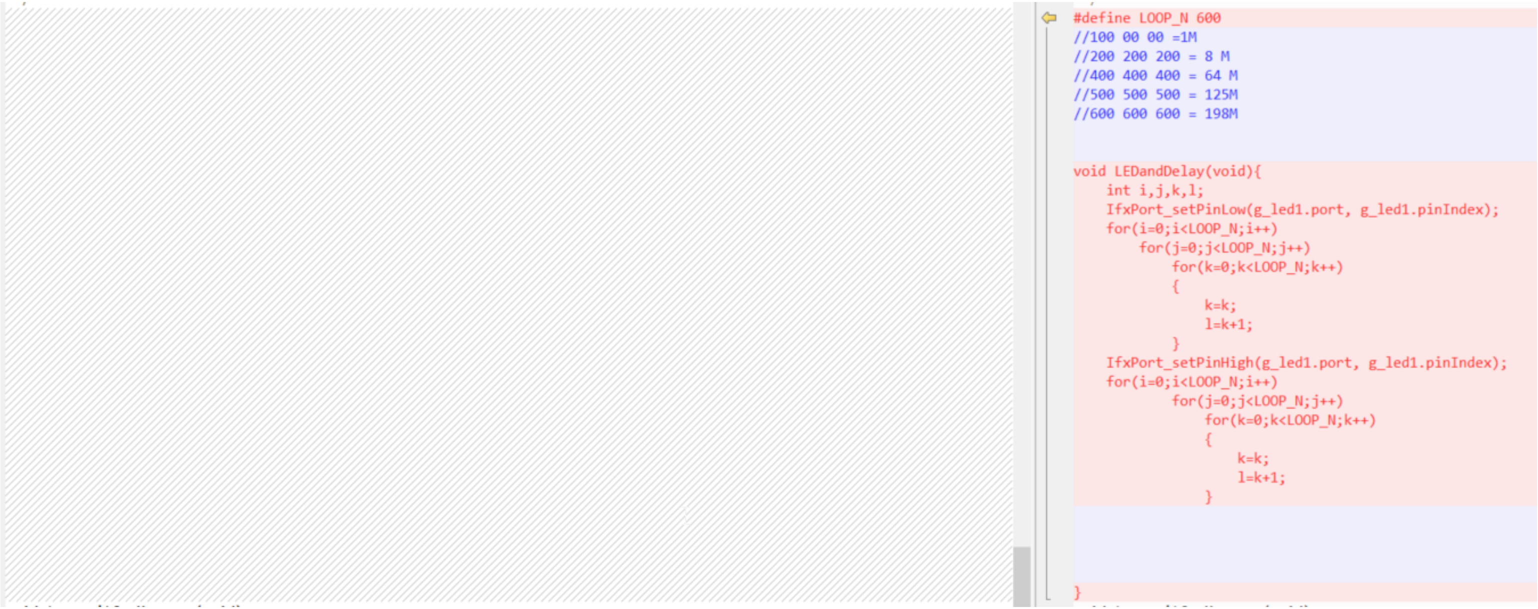


```c
g_mcmcan.canNodeConfig.interruptConfig.transmissionCompletedEnabled = TRUE;
g_mcmcan.canNodeConfig.interruptConfig.traco.priority = ISR_PRIORITY_CAN_TX;
g_mcmcan.canNodeConfig.interruptConfig.traco.interruptLine = IfxCan_InterruptLine_0;
g_mcmcan.canNodeConfig.interruptConfig.traco.typeOfService = IfxSrc_Tos_cpu0;

IfxCan_Can_initNode(&g_mcmcan.canSrcNode, &g_mcmcan.canNodeConfig);

/* ==========================================================================================
 * Destination CAN node configuration and initialization:
 * ==========================================================================================
 *  - load default CAN node configuration into configuration structure
 *
 *  - set destination CAN node in the "Loop-Back" mode (no external pins are used)
 *  - assign destination CAN node to CAN node 1
 *
 *  - define the frame to be the receiving one
 *
 *  - once the message is stored in the dedicated RX buffer, raise the interrupt
 *  - define the receive interrupt priority
 *  - assign the interrupt line 1 to the receive interrupt
 *  - receive interrupt service routine should be serviced by the CPU0
 *
 *  - initialize the destination CAN node with the modified configuration
 * ==========================================================================================
 */
IfxCan_Can_initNodeConfig(&g_mcmcan.canNodeConfig, &g_mcmcan.canModule);

g_mcmcan.canNodeConfig.busLoopbackEnabled = TRUE;
g_mcmcan.canNodeConfig.nodeId = IfxCan_NodeId_1;

g_mcmcan.canNodeConfig.frame.type = IfxCan_FrameType_receive;

g_mcmcan.canNodeConfig.interruptConfig.messageStoredToDedicatedRxBufferEnabled = TRUE;
g_mcmcan.canNodeConfig.interruptConfig.reint.priority = ISR_PRIORITY_CAN_RX;
g_mcmcan.canNodeConfig.interruptConfig.reint.interruptLine = IfxCan_InterruptLine_1;
g_mcmcan.canNodeConfig.interruptConfig.reint.typeOfService = IfxSrc_Tos_cpu0;

IfxCan_Can_initNode(&g_mcmcan.canDstNode, &g_mcmcan.canNodeConfig);

/* ==========================================================================================
 * CAN filter configuration and initialization:
 * ==========================================================================================
 *  - filter configuration is stored under the filter element number 0
 *  - store received frame in a dedicated RX Buffer
 *  - define the same message ID as defined for the TX message
 *  - assign the filter to the dedicated RX Buffer (RxBuffer0 in this case)
```

```c
g_mcmcan.canNodeConfig.interruptConfig.transmissionCompletedEnabled = TRUE;
g_mcmcan.canNodeConfig.interruptConfig.traco.priority = ISR_PRIORITY_CAN_TX;
g_mcmcan.canNodeConfig.interruptConfig.traco.interruptLine = IfxCan_InterruptLine_0;
g_mcmcan.canNodeConfig.interruptConfig.traco.typeOfService = IfxSrc_Tos_cpu0;
g_mcmcan.canNodeConfig.pins=&Example_Pins;
IfxCan_Can_initNode(&g_mcmcan.canSrcNode, &g_mcmcan.canNodeConfig);

/* ==========================================================================================
 * Destination CAN node configuration and initialization:
 * ==========================================================================================
 *  - load default CAN node configuration into configuration structure
 *
 *  - set destination CAN node in the "Loop-Back" mode (no external pins are used)
 *  - assign destination CAN node to CAN node 1
 *
 *  - define the frame to be the receiving one
 *
 *  - once the message is stored in the dedicated RX buffer, raise the interrupt
 *  - define the receive interrupt priority
 *  - assign the interrupt line 1 to the receive interrupt
 *  - receive interrupt service routine should be serviced by the CPU0
 *
 *  - initialize the destination CAN node with the modified configuration
 * ==========================================================================================
 */
IfxCan_Can_initNodeConfig(&g_mcmcan.canNodeConfig, &g_mcmcan.canModule);

g_mcmcan.canNodeConfig.busLoopbackEnabled = FALSE;
g_mcmcan.canNodeConfig.nodeId = IfxCan_NodeId_1;

g_mcmcan.canNodeConfig.frame.type = IfxCan_FrameType_receive;

g_mcmcan.canNodeConfig.interruptConfig.messageStoredToDedicatedRxBufferEnabled = TRUE;
g_mcmcan.canNodeConfig.interruptConfig.reint.priority = ISR_PRIORITY_CAN_RX;
g_mcmcan.canNodeConfig.interruptConfig.reint.interruptLine = IfxCan_InterruptLine_1;
g_mcmcan.canNodeConfig.interruptConfig.reint.typeOfService = IfxSrc_Tos_cpu0;

IfxCan_Can_initNode(&g_mcmcan.canDstNode, &g_mcmcan.canNodeConfig);

/* ==========================================================================================
 * CAN filter configuration and initialization:
 * ==========================================================================================
 *  - filter configuration is stored under the filter element number 0
 *  - store received frame in a dedicated RX Buffer
 *  - define the same message ID as defined for the TX message
 *  - assign the filter to the dedicated RX Buffer (RxBuffer0 in this case)
```

Step 4:

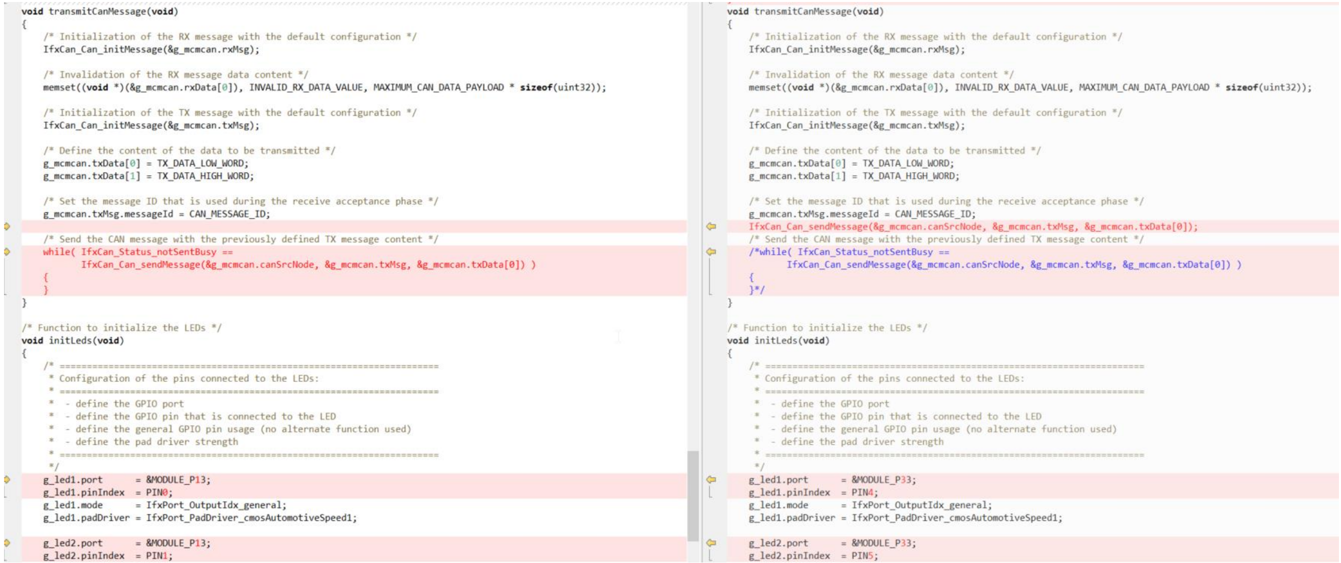
- Add delay function and LED blinking



```
#define LOOP_N 600
//100 00 00 =1M
//200 200 200 = 8 M
//400 400 400 = 64 M
//500 500 500 = 125M
//600 600 600 = 198M


void LEDandDelay(void){
    int i,j,k,l;
    IfxPort_setPinLow(g_led1.port, g_led1.pinIndex);
    for(i=0;i<LOOP_N;i++)
        for(j=0;j<LOOP_N;j++)
            for(k=0;k<LOOP_N;k++)
            {
                k=k;
                l=k+1;
            }
    IfxPort_setPinHigh(g_led1.port, g_led1.pinIndex);
    for(i=0;i<LOOP_N;i++)
            for(j=0;j<LOOP_N;j++)
                for(k=0;k<LOOP_N;k++)
                {
                    k=k;
                    l=k+1;
                }




}
```

Step 5:

- Modify transmitCanMessage()
- Set LED pin definitions to P33.4 and P33.5

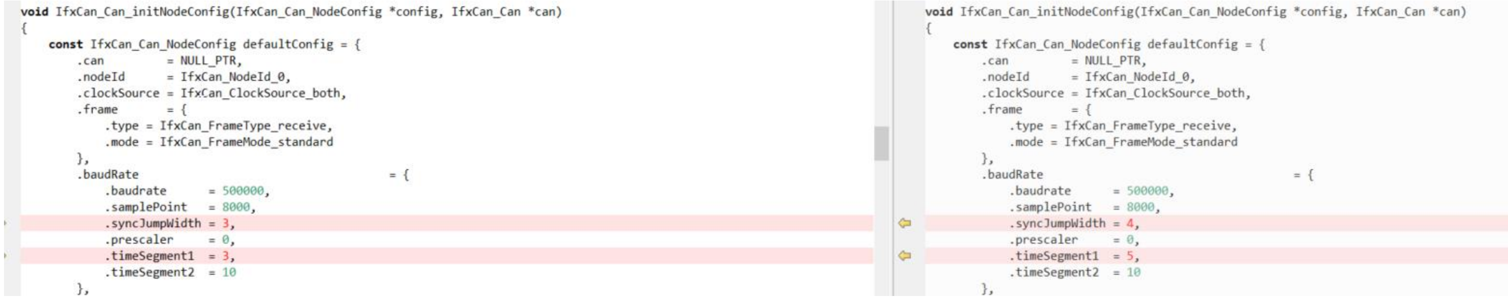


```c
void transmitCanMessage(void)
{
    /* Initialization of the RX message with the default configuration */
    IfxCan_Can_initMessage(&g_mcmcan.rxMsg);

    /* Invalidation of the RX message data content */
    memset((void *)(&g_mcmcan.rxData[0]), INVALID_RX_DATA_VALUE, MAXIMUM_CAN_DATA_PAYLOAD * sizeof(uint32));

    /* Initialization of the TX message with the default configuration */
    IfxCan_Can_initMessage(&g_mcmcan.txMsg);

    /* Define the content of the data to be transmitted */
    g_mcmcan.txData[0] = TX_DATA_LOW_WORD;
    g_mcmcan.txData[1] = TX_DATA_HIGH_WORD;

    /* Set the message ID that is used during the receive acceptance phase */
    g_mcmcan.txMsg.messageId = CAN_MESSAGE_ID;

    /* Send the CAN message with the previously defined TX message content */
    while( IfxCan_Status_notSentBusy ==
           IfxCan_Can_sendMessage(&g_mcmcan.canSrcNode, &g_mcmcan.txMsg, &g_mcmcan.txData[0]) )
    {
    }
}

/* Function to initialize the LEDs */
void initLeds(void)
{
    /* ======================================================================
     * Configuration of the pins connected to the LEDs:
     * ======================================================================
     *  - define the GPIO port
     *  - define the GPIO pin that is connected to the LED
     *  - define the general GPIO pin usage (no alternate function used)
     *  - define the pad driver strength
     * ======================================================================
     */
    g_led1.port      = &MODULE_P13;
    g_led1.pinIndex  = PIN0;
    g_led1.mode      = IfxPort_OutputIdx_general;
    g_led1.padDriver = IfxPort_PadDriver_cmosAutomotiveSpeed1;

    g_led2.port      = &MODULE_P13;
    g_led2.pinIndex  = PIN1;
```

```c
void transmitCanMessage(void)
{
    /* Initialization of the RX message with the default configuration */
    IfxCan_Can_initMessage(&g_mcmcan.rxMsg);

    /* Invalidation of the RX message data content */
    memset((void *)(&g_mcmcan.rxData[0]), INVALID_RX_DATA_VALUE, MAXIMUM_CAN_DATA_PAYLOAD * sizeof(uint32));

    /* Initialization of the TX message with the default configuration */
    IfxCan_Can_initMessage(&g_mcmcan.txMsg);

    /* Define the content of the data to be transmitted */
    g_mcmcan.txData[0] = TX_DATA_LOW_WORD;
    g_mcmcan.txData[1] = TX_DATA_HIGH_WORD;

    /* Set the message ID that is used during the receive acceptance phase */
    g_mcmcan.txMsg.messageId = CAN_MESSAGE_ID;
    IfxCan_Can_sendMessage(&g_mcmcan.canSrcNode, &g_mcmcan.txMsg, &g_mcmcan.txData[0]);
    /* Send the CAN message with the previously defined TX message content */
    /*while( IfxCan_Status_notSentBusy ==
           IfxCan_Can_sendMessage(&g_mcmcan.canSrcNode, &g_mcmcan.txMsg, &g_mcmcan.txData[0]) )
    {
    }*/
}

/* Function to initialize the LEDs */
void initLeds(void)
{
    /* ======================================================================
     * Configuration of the pins connected to the LEDs:
     * ======================================================================
     *  - define the GPIO port
     *  - define the GPIO pin that is connected to the LED
     *  - define the general GPIO pin usage (no alternate function used)
     *  - define the pad driver strength
     * ======================================================================
     */
    g_led1.port      = &MODULE_P33;
    g_led1.pinIndex  = PIN4;
    g_led1.mode      = IfxPort_OutputIdx_general;
    g_led1.padDriver = IfxPort_PadDriver_cmosAutomotiveSpeed1;

    g_led2.port      = &MODULE_P33;
    g_led2.pinIndex  = PIN5;
```

Step 6:

- Ifxcan_can.c
  - Change ifxCan_Can_initNodeConfig() for .baudRate

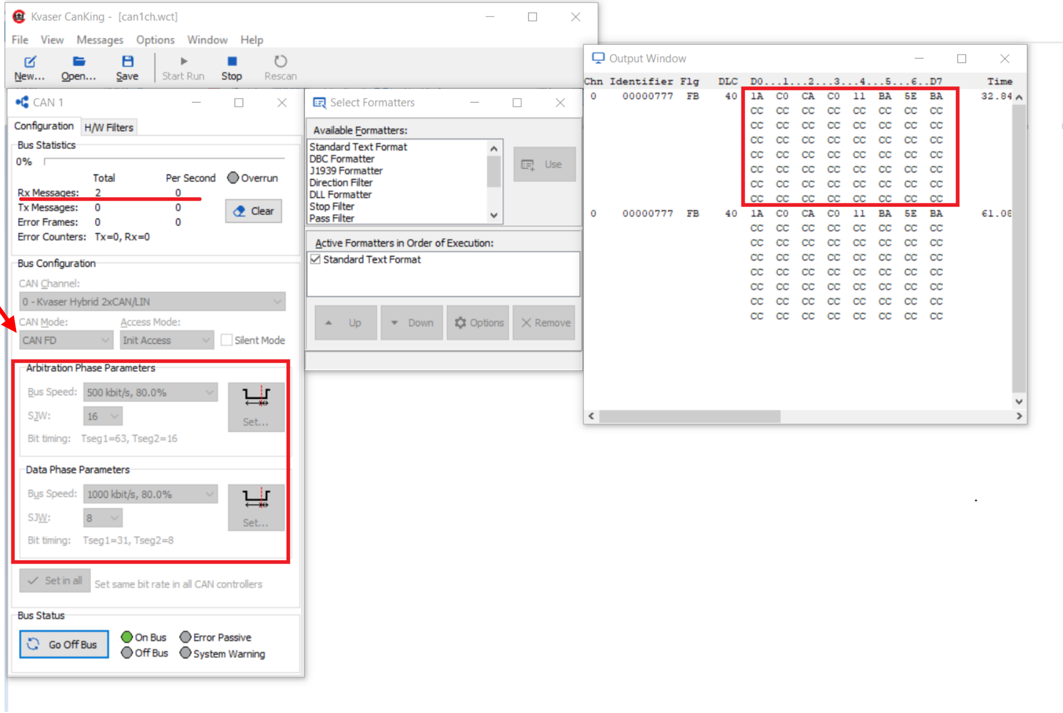


```
void IfxCan_Can_initNodeConfig(IfxCan_Can_NodeConfig *config, IfxCan_Can *can)
{
    const IfxCan_Can_NodeConfig defaultConfig = {
        .can         = NULL_PTR,
        .nodeId      = IfxCan_NodeId_0,
        .clockSource = IfxCan_ClockSource_both,
        .frame       = {
            .type = IfxCan_FrameType_receive,
            .mode = IfxCan_FrameMode_standard
        },
        .baudRate                                  = {
            .baudrate      = 500000,
            .samplePoint   = 8000,
            .syncJumpWidth = 3,
            .prescaler     = 0,
            .timeSegment1  = 3,
            .timeSegment2  = 10
        },
```

```
void IfxCan_Can_initNodeConfig(IfxCan_Can_NodeConfig *config, IfxCan_Can *can)
{
    const IfxCan_Can_NodeConfig defaultConfig = {
        .can         = NULL_PTR,
        .nodeId      = IfxCan_NodeId_0,
        .clockSource = IfxCan_ClockSource_both,
        .frame       = {
            .type = IfxCan_FrameType_receive,
            .mode = IfxCan_FrameMode_standard
        },
        .baudRate                                  = {
            .baudrate      = 500000,
            .samplePoint   = 8000,
            .syncJumpWidth = 4,
            .prescaler     = 0,
            .timeSegment1  = 5,
            .timeSegment2  = 10
        },
```

*CAN FD related

- Kvaser CanKing for CAN FD



Bus Statistics

0%

	Total	Per Second
Rx Messages:	2	0
Tx Messages:	0	0
Error Frames:	0	0
Error Counters:	Tx=0, Rx=0	

○ Overrun

Bus Configuration

CAN Channel: 0 - Kvaser Hybrid 2xCAN/LIN

CAN Mode: CAN FD

Access Mode: Init Access

□ Silent Mode

Arbitration Phase Parameters

Bus Speed: 500 kbit/s, 80.0%

SJW: 16

Bit timing: Tseg1=63, Tseg2=16

Data Phase Parameters

Bus Speed: 1000 kbit/s, 80.0%

SJW: 8

Bit timing: Tseg1=31, Tseg2=8

Set same bit rate in all CAN controllers

Bus Status

● On Bus
○ Off Bus
○ Error Passive
○ System Warning

Available Formatters:

Standard Text Format
DBC Formatter
J1939 Formatter
Direction Filter
DLL Formatter
Stop Filter
Pass Filter

Active Formatters in Order of Execution:

☑ Standard Text Format

```
Chn Identifier Flg   DLC  D0...1...2...3...4...5...6..D7       Time
 0    00000777  FB    40  1A  C0  CA  C0  11  BA  5E  BA      32.84
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
 0    00000777  FB    40  1A  C0  CA  C0  11  BA  5E  BA      61.08
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
                          CC  CC  CC  CC  CC  CC  CC  CC
```