**Freescale Semiconductor**
Application Note

Document Number: AN4962
Rev. 1.0, 07/2014

# 针对Kinetis系列MCU 编写和调试CAN总线驱动程序

作者：IMM FAE 何此昂

## 1 简介

控制器局域网络（CAN），是一种串行多主站的局域网总线，具有高可靠，实时，适应于对环境温度恶劣、电磁干扰严重以及震动大的环境。其总线特点如下：

- 总线的通信介质是双绞线、同轴电缆或者光纤等；
- CAN总线为多主站总线，各节点可在任意时刻向网络上的其他节点发送信息，且不分主从；
- CAN总线采用独特的非破坏性总线仲裁技术，高优先级节点优先传输数据，故实时性好；
- CAN总线具有点对点、点对多点以及全局广播数据传输的功能；
- CAN总线上某一节点出现严重错误时，可自动脱离总线，而总线上的其他操作不受影响；
- CAN总线系统扩充时，可直接将新节点挂接在总线上，因此走线少，系统扩充容易；
- CAN总线最大传输速率可达1 Mbit/s，直接通信距离最远可达到10 km（通信速率在5 kbit/s）；
- CAN总线上的节点数和帧格式有关，在标准帧（11位报文标识符）时可达到110个，而扩展帧（29位标识符）格式时，个数不受限制。CAN总线采用短帧结构，每帧有效字节数最多为8个，数据传输时间短，并有CRC校验，因此数据出错率低。

目录

## 1.1 目的

本文档的目的是为了便于开发者了解如何编写和调试针对Kinetis Cortex M4内核MCU的CAN驱动程序，提供了如何使用Freescale的处理器专家软件Processor Expert以及MQX4.0实时操作系统来快速生成CAN的驱动代码的实例以及调试步骤。为了兼容更多的平台，特选取K60DN512VMD10芯片和TWR-K60D100M作为实验平台进行验证。

## 2 硬件设计

选取K60DN512 塔式系统，主板TWR-K60D100M，版本B。通信板卡TWR-SER，版本G。

针对不同的TWR板卡，其连线需要注意如下。

主板TWR-K60D100M的时钟部分如图 1所示。

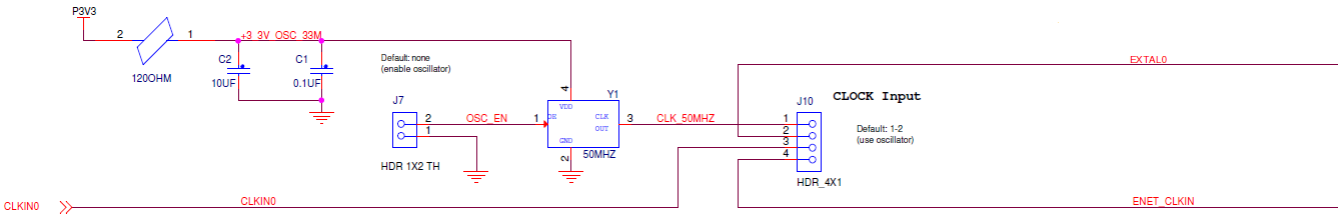


**图 1 TWR-K60D100M的时钟部分**

由于采用CLKIN0 25MHz的时钟，需要将J10的2、3引脚短接。

TWR-SER的时钟部分如图 2所示。

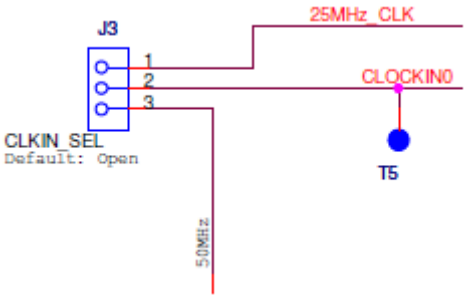


**图 2 TWR-SER的时钟部分**

时钟部分需要将J3的1、2引脚短接，为TWR-K60D100M主板提供时钟。

针对CAN调试部分如图 3所示，需要将J5的5、6引脚短接，7、8引脚短接，9、10引脚短接引入120欧的匹配阻抗电阻。

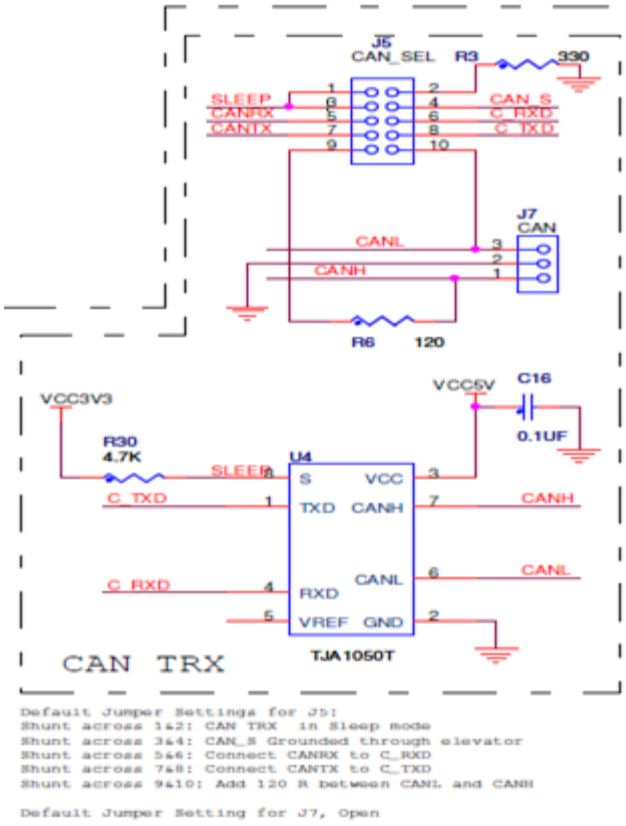


**图 3 TWR-SER的CAN部分电路**

为了作为通用设计调试的参考，将主要的系统时钟，调试口以及CAN总线通信端口的硬件原理设计参考如下。

## 2.1 JTAG 调试口电路

使用标准的JTAG口连接电路，如图 4所示，JTAG RESET引脚连接K60芯片的复位引脚。

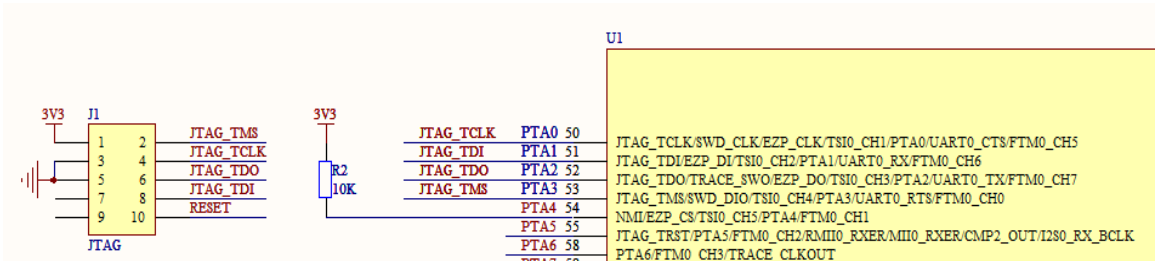


**图 4 JTAG调试电路**

## 2.2 时钟电路

选择外部无源的25M时钟连接到K60的时钟引脚上，如图 5所示。

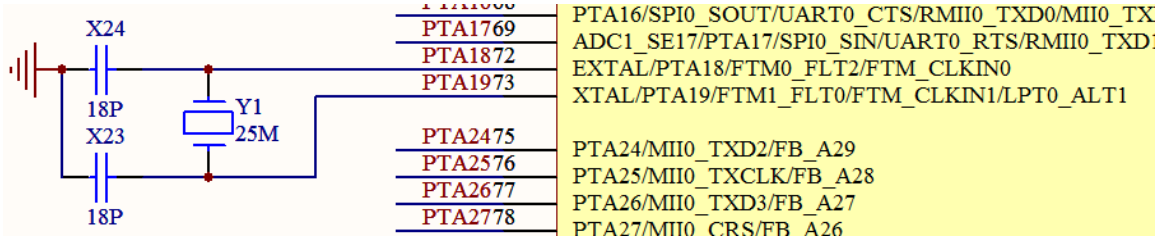


**图 5 外部时钟电路**

## 2.3 CAN总线驱动电路

提供Freescale的MC33901 CAN驱动芯片的原理设计图，如图 6所示。

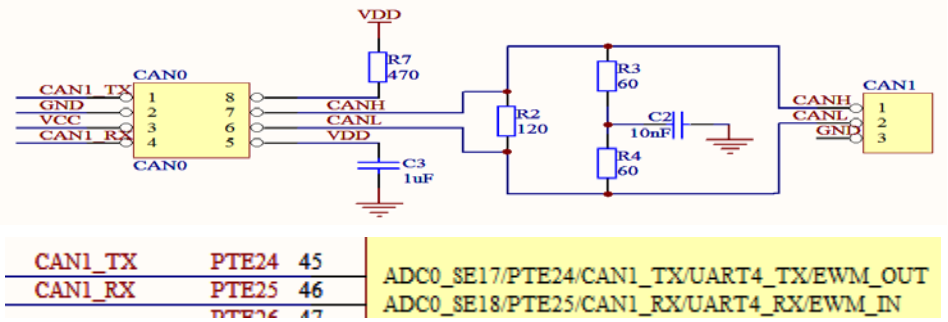


**图 6 CAN硬件驱动设计电路**

K60的CAN1的通道有两个复用通道：PET24（RX接收）/PTE25（TX发送）和PTC16（RX接收）/PTC17（TX发送），这里选择PTE口作为CAN1的通信口。和其他厂家的CAN驱动芯片不同的地方（如TJA1040，TWR-SER板使用该器件）是第5脚，TJA1040可以悬空，但是MC33901这个引脚需要接电源。MC33901的结构如图 7所示，第5脚VIO作为输入，手册要求电压输入范围2.8V－5.5V。

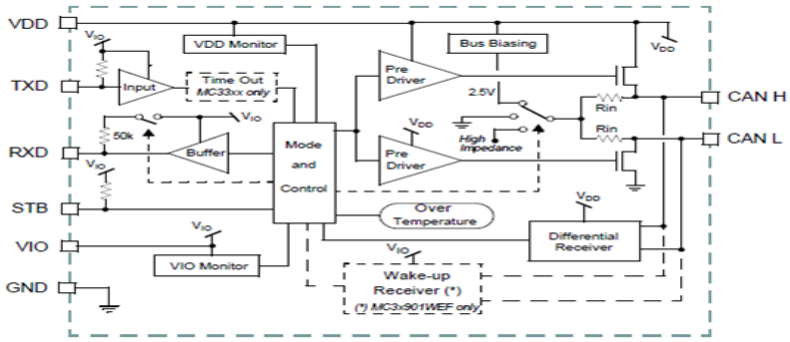


**图 7 MC33901结构图**

**注意**

硬件设计部分需要考虑到CAN的自测试功能。一般都会使用Loop模式来进行测试，这是检测CAN控制器部分是否正常的最常用的办法，不需要额外焊接CAN的驱动芯片。但是，如果在产品调试时，涉及到外围CAN驱动器的电路调试，如果焊接了CAN的驱动器芯片，而通信数据又不正常；或者在没有焊接CAN的驱动器芯片，这时又没有启用CAN的Loop（回环）测试模式时，在这两种情况下可以简单地将CAN发送和CAN接收短接进行波形的观测，来判断是否是因为CAN的驱动电路导致通信异常的问题。因为在Loop模式下CAN的发送和接受是短接在一起的，而非Loop模式下则没有。另外，需要注意的是，使用非Loop模式下测试的时候，如果没有连接CAN的驱动器芯片，使用CAN的分析仪或者示波器就不会在CAN总线上监测到有持续的CAN波形。

# 3 软件设计

## 3.1 创建基于PE的CAN总线工程

Freescale提供免费的处理器专家软件PE，可以方便地利用图形化工具生成各种各样的驱动代码，而且所生成的代码可以直接在IAR或者KEIL等不同平台下使用。例如用图 8所示的界面对时钟模块进行配置后，就可以生成相应的时钟模块的初始化和控制代码。

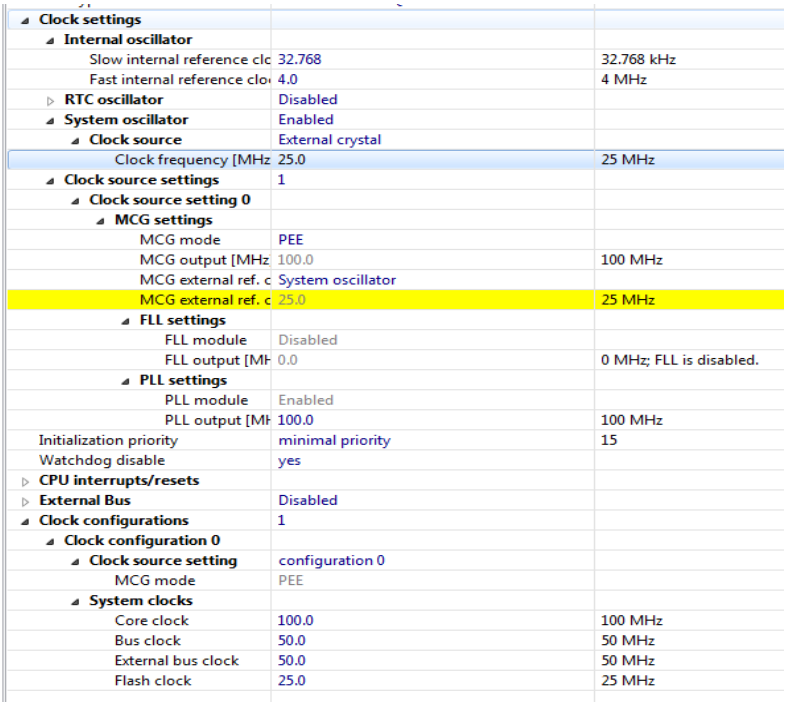


**图 8 PE下的时钟配置**

按照上述的时钟配置，在Generated Code下看到cpu.c文件中关于系统时钟配置的代码如图 9所示。

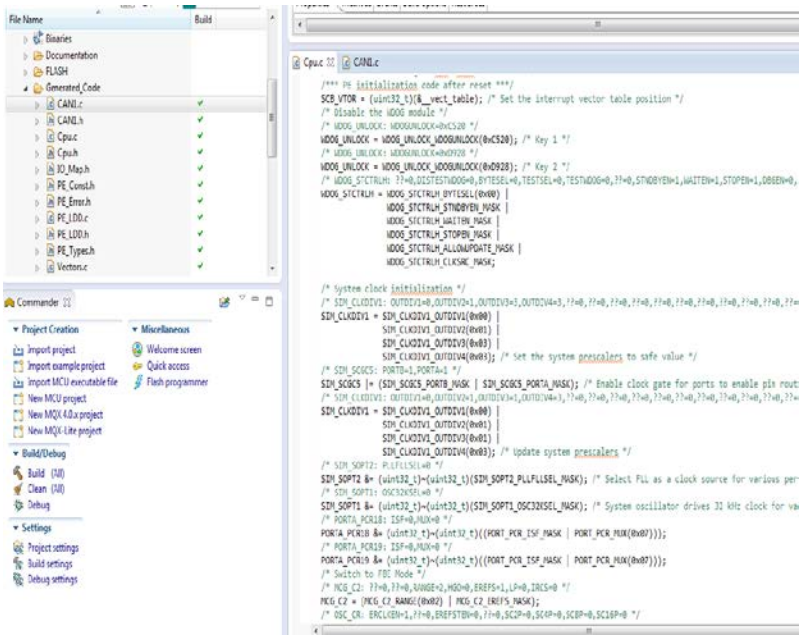

图 9 CPU.c代码

下面使用Processor Expert中的CAN_LDD组件来设置基于中断和轮询模式的CAN发送和接收例程。LDD是逻辑设备驱动（Logic Device Driver）。针对每个不同的LDD组件，都包括Init()方法来初始化对应外设和驱动，Deinit()方法去初始化对应外设和驱动。

### 3.1.1 创建基于中断的CAN总线收发例程

#### 3.1.1.1 使用中断服务程序发送数据帧

下面的例子演示了使用标准帧ID和扩展帧ID来发送数据帧。当数据帧被成功的发送后，OnFreeTxBuffer 事件将会被调用。要求CAN组件配置如下：

- Interrupt service: Enabled
- Message buffers: 1, Buffer0 – Buffer type: Transmit
- Bit rate: 100 kbit/s
- Loop mode:  no
- Enabled in init. code: yes
- Methods to enable: SendFrame
- Events to enable: OnFreeTxBuffer

处理器专家根据以上配置自动生成的代码在文件ProcessorExpert.c中的相关内容如下所示：

```
volatile bool DataFrameTxFlg;
LDD_TDeviceData *MyCANPtr;
LDD_TError Error;
LDD_CAN_TFrame Frame;
uint8_t OutData[4] = {0x00U, 0x01U, 0x02U, 0x03U}; /* 初始化输出数据缓存 */
void main(void)
{
MyCANPtr = CAN1_Init(NULL);      /* 初始化 CAN1 组件 */
Frame.MessageID = 0x123U;        /* 设置发送 Tx ID 数值为标准帧格式 */
Frame.FrameType = LDD_CAN_DATA_FRAME; /* 指定数据帧类型为发送数据帧 Tx */
Frame.Length = sizeof(OutData);  /* 设置数据帧字节个数，为 4 个 */
Frame.Data = OutData;            /* 设置输出数据 OutData 缓存区的指针 */
DataFrameTxFlg = FALSE;          /* 初始化 DataFrameTxFlg 标志 */
Error = CAN1_SendFrame(MyCANPtr, 0U, &Frame); /* 通过缓冲区 Buffer 0 来发送数据帧*/
while (!DataFrameTxFlg) {        /* 等待，直到数据帧传送完成 */
}
Frame.MessageID = (0x123456U | LDD_CAN_MESSAGE_ID_EXT);
                                          /* 设置发送 Tx ID 数值为扩展帧格式 */
Frame.FrameType = LDD_CAN_DATA_FRAME;     /* 指定数据帧类型为发送数据帧 Tx */
Frame.Length = sizeof(OutData);           /* 设置数据帧字节个数，为 4 个 */
Frame.Data = OutData;                     /* 设置输出缓存区 OutData 的指针*/
DataFrameTxFlg = FALSE;                   /* 清除 DataFrameTxFlg 标志 */
Error = CAN1_SendFrame(MyCANPtr, 0U, &Frame); /* 通过缓冲区 Buffer 0 来发送数据帧*/
while (!DataFrameTxFlg) {                 /* 等待，直到数据帧传送完成 */
}
for(;;) {}
}
```

文件 Event.c 中的内容如下:

```
extern volatile bool DataFrameTxFlg;
void CAN1_OnFreeTxBuffer(LDD_TUserData *UserDataPtr, LDD_CAN_TMBIndex BufferIdx)
{
DataFrameTxFlg = TRUE; /* 置位 DataFrameTxFlg 标志 */
}
```

### 3.1.1.2 使用中断服务程序接收数据帧

下面的例子演示了使用中断方式接收标准帧ID和扩展帧ID数据帧。当数据帧被成功的接收后，OnFullRxBuffer 事件将会被调用。要求CAN组件配置如下：

- Interrupt service: Enabled
- Global Acceptance Mask: yes
- Acceptance mask for buffer 0..n: 1FFFFFFF

- Message buffers: 1, Buffer0 - Buffer type: Receive
- Accept frames: Standard
- Message ID: 7FF
- Bit rate: 100 kbit/s
- Loop mode: no
- Enabled in init. code: yes
- Methods to enable: ReadFrame,SetRxBufferID
- Events to enable: OnFullRxBuffer

处理器专家根据以上配置自动生成的代码在文件ProcessorExpert.c中的相关内容如下所示：

```
volatile bool DataFrameRxFlg = FALSE;
LDD_TDeviceData *MyCANPtr;
LDD_TError Error;
LDD_CAN_TFrame Frame;
uint8_t InpData[8];
void main(void)
{

MyCANPtr = CAN1_Init(NULL);       /* 初始化 CAN1 组件*/
while (!DataFrameRxFlg) {         /*等待，直到数据帧被接收*/
}
Frame.Data = InpData;              /* 设置输入数据 InpData 缓存的指针*/
Error = CAN1_ReadFrame(MyCANPtr, 0U, &Frame); /* 从数据缓存区 0 中读取数据帧 */
/*
Frame.MessageID 包含了 CAN ID 节点的数值。如果 if((Frame.MessageID &
LDD_CAN_MESSAGE_ID_EXT)!=0) ，则为扩展帧格式(extended ID,)，否则为标准的 ID(standard ID)。
Frame.FrameType 表示接收帧类型，如 LDD_CAN_DATA_FRAME，Frame.Length 表示接收帧 Rx 中接收的字
节的个数，InpData[] 包含了接收 Rx 数据字节。
*/

DataFrameRxFlg = FALSE;    /* 清除 DataFrameRxFlg 标志 */
TError = CAN1_SetRxBufferID(MyCANPtr, 0U, (0x123456U|LDD_CAN_MESSAGE_ID_EXT));
                           /* 设置缓存区 0 中的接收 ID   Rx 为扩展帧类型*/
while (!DataFrameRxFlg) { /*等待，直到数据帧被接收 */
}
Frame.Data = InpData;
Error = CAN1_ReadFrame(MyCANPtr, 0U, &Frame); /* 从数据缓存区 0 中读取数据帧 */

/*
```

```
Frame.MessageID 包含了 CAN ID 节点的数值。如果 if((Frame.MessageID &
LDD_CAN_MESSAGE_ID_EXT)!=0) ，则为扩展帧格式(extended ID,)，否则为标准的 ID(standard ID)。
Frame.FrameType 表示接收帧类型，如 LDD_CAN_DATA_FRAME，Frame.Length 表示接收帧中接收的字节的
个数，InpData[] 包含了接收 Rx 数据字节。
*/

for(;;) {}
}
```

程序 Event.c 中的内容如下：

```
extern volatile bool DataFrameRxFlg;
void CAN1_OnFullRxBuffer(LDD_TUserData *UserDataPtr, LDD_CAN_TMBIndex BufferIdx)
{
DataFrameRxFlg = TRUE; /* 置位标志 DataFrameRxFlg */
}
```

### 3.1.1.3 扩展帧的设置

有些场景下，采用某些配置后发现两个节点可以正常地发送和接收标准数据帧，但是在处理扩展帧的时候却出现异常。例如使用可以正常收发扩展帧格式的CAN诊断工具作为一个节点A，和另外一个节点B通信，发现B节点却只能发送，不能接收。在下面的示例中，设置B节点发送数据帧的ID为0xC001，CAN_LDD组件的配置如图 10所示：

| Name | Value | Details |
|---|---|---|
| Component name | CAN1 | |
| CAN channel | CAN0 | CAN0 |
| **Interrupt service** | Enabled | |
| **Settings** | | |
| **Pins** | | |
| **Global acceptance mask** | no | |
| **Receiver FIFO** | Disabled | |
| **Message buffers** | 2 | |
| **Buffer0** | | |
| **Buffer type** | Receive | |
| Accept frames | Extended | |
| Message ID | C001 H | |
| **Invidual Acceptance Mask** | Enabled | |
| Acceptance Mask | 1FFFFFFF H | Std: 0x07FF, Ext: 0x1FFFFFFF |
| **Buffer1** | | |
| **Buffer type** | Transmit | |
| Accept frames | Extended | |
| Message ID | 7FF H | |
| **Invidual Acceptance Mask** | Enabled | |
| Acceptance Mask | 1FFFFFFF H | Std: 0x07FF, Ext: 0x1FFFFFFF |
| Abort transmission mode | no | |
| Remote request storing | Remote Request Frame is stored | |
| Entire frame arbitration field comparison | no | |
| Local priority | no | |
| Self reception | Disabled | |
| Timer synchronization | Disabled | |
| Lowest buffer transmitted first | Lowest ID | |
| Loop mode | no | |
| Bus off recovery mode | Automatic | |
| Listen only mode | no | |
| **Wake up** | Disabled | |
| **Timing** | | |
| CAN timing calculator | *click to run ->* | |

**图 10 CAN扩展帧设置**

通过该设置，该节点可以接收到任何总线上扩展帧的内容，而不仅仅是0xC001。因为在验收滤波器（Individual acceptance mask filters）的作用下，只有当接收报文中的标识位和验收滤波器预定

义的位值相同时，CAN控制器才会将数据报文存到接收缓存中。如果验收滤波器中的位设置为0，缓存区接收总线上所有的信息。如果验收滤波器中的位设置为1，则接收ID和缓存器中的ID（Message ID）的设置一致，数据才被接收。

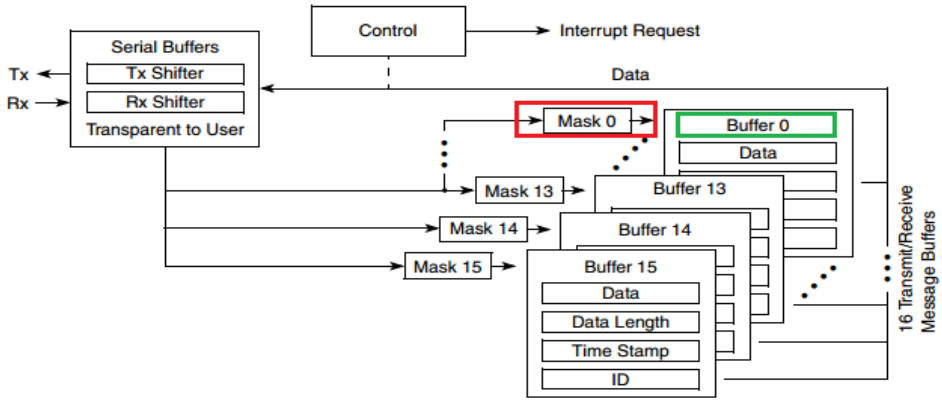


**图 11 16个信息缓存框图**

如果B节点全部验收的mask（一共16个，如图 11所示）为1FFFFFFF，所有的位都会被诊断，只有C001的信息被接收，为了接收所有的信息，这里mask要设置为0。

## 3.1.2 创建基于轮询的CAN总线收发例程

### 3.1.2.1 使用轮询方式发送数据帧

下面的例子演示了使用轮询方式来发送标准帧ID和扩展帧ID数据帧。当数据帧被成功的发送后，OnFreeTxBuffer 事件将会被调用。要求CAN组件配置如下：

- Interrupt service: Disabled
- Message buffers: 1, Buffer0 - Buffer type: Transmit
- Bit rate: 100 kbit/s
- Loop mode: no
- Enabled in init. code: yes
- Methods to enable: SendFrame
- Events to enable: OnFreeTxBuffer

根据以上配置，处理器专家自动生成的代码在文件ProcessorExpert.c中的相关内容如下所示：

```
volatile bool DataFrameTxFlg;
LDD_TDeviceData *MyCANPtr;
LDD_TError Error;
LDD_CAN_TFrame Frame;
```

```
uint8_t OutData[4] = {0x00U, 0x01U, 0x02U, 0x03U}; /* 初始化输出数据缓存*/
void main(void)
{
. . .
MyCANPtr = CAN1_Init(NULL);           /* 初始化 CAN1 组件*/
Frame.MessageID = 0x123U;             /* 设置发送 Tx ID 为标准帧格式 */
Frame.FrameType = LDD_CAN_DATA_FRAME; /* 指定数据帧类型为发送帧 Tx */
Frame.Length = sizeof(OutData);       /* 设置数据帧的字节个数为 4 */
Frame.Data = OutData;                 /* 设置输出数据 OutData 缓存的指针*/
DataFrameTxFlg = FALSE;               /* 初始化 DataFrameTxFlg 标志 */
Error = CAN1_SendFrame(MyCANPtr, 0U, &Frame); /* 通过缓冲区 0 发送数据帧 */
while (!DataFrameTxFlg) {                     /* 等待，直到数据帧发送完成 */
CAN1_Main(MyCANPtr);
}
. . .
Frame.MessageID = (0x123456U | LDD_CAN_MESSAGE_ID_EXT); /* 设置 Tx ID 为扩展帧格式 */
Frame.FrameType = LDD_CAN_DATA_FRAME; /* 指定数据帧类型为发送帧 Tx */
Frame.Length = sizeof(OutData);       /* 设置数据帧的字节个数为 4 */
Frame.Data = OutData;                 /* 设置输出数据 OutData 缓存的指针*/
DataFrameTxFlg = FALSE;               /* 清除 DataFrameTxFlg 标志 */
Error = CAN1_SendFrame(MyCANPtr, 0U, &Frame); /* 通过缓存区 0 发送数据帧*/
while (!DataFrameTxFlg) {                     /* 等待，直到数据帧发送完成 */
CAN1_Main(MyCANPtr);
}
. . .
for(;;) {}
}
```

程序 Event.c 中的内容如下:

```
extern volatile bool DataFrameTxFlg;
void CAN1_OnFreeTxBuffer(LDD_TUserData *UserDataPtr, LDD_CAN_TMBIndex BufferIdx)
{
DataFrameTxFlg = TRUE;              /* 置位 DataFrameTxFlg 标志 */
}
```

### 3.1.2.2 使用轮询方式接收数据帧

下面的例子演示了使用轮询方式来标准帧ID和扩展帧ID来接收数据帧。当数据帧被成功的接收后，OnFullRxBuffer 事件将会被调用。

要求组件配置如下:

- Interrupt service: Disabled
- Global Acceptance Mask: yes

- Acceptance mask for buffer 0..n: 1FFFFFFF
- Message buffers: 1, Buffer0 - Buffer type: Receive
- Accept frames: Standard
- Message ID: 7FF
- Bit rate: 100 kbit/s
- Loop mode : no
- Enabled in init. code: yes
- Methods to enable: ReadFrame,SetRxBufferID
- Events to enable: OnFullRxBuffer

根据以上配置，处理器专家自动生成的代码在文件ProcessorExpert.c中的相关内容如下所示：

```
volatile bool DataFrameRxFlg = FALSE;
LDD_TDeviceData *MyCANPtr;
LDD_TError Error;
LDD_CAN_TFrame Frame;
uint8_t InpData[8];
void main(void)
{
MyCANPtr = CAN1_Init(NULL); /* 初始化 CAN1 组件*/
while (!DataFrameRxFlg) {   /* 等待，直到数据帧被接收 */
CAN1_Main(MyCANPtr);
}
Frame.Data = InpData;                          /* 设置输入数据 InpData 缓存区指针 */
Error = CAN1_ReadFrame(MyCANPtr, 0U, &Frame); /* 从数据缓存区 0 中读取数据帧 */
/*
Frame.MessageID 包含了 CAN ID 节点的数值。如果 if((Frame.MessageID &
LDD_CAN_MESSAGE_ID_EXT)!=0) ，则为扩展帧格式(extended ID,)，否则为标准的 ID(standard ID)。
Frame.FrameType 表示接收帧类型，如 LDD_CAN_DATA_FRAME，Frame.Length 表示接收帧中接收的字节的
个数，InpData[] 包含了接收 Rx 数据字节。
*/
DataFrameRxFlg = FALSE;   /* 清除 DataFrameRxFlg 标志 */
Error = CAN1_SetRxBufferID(MyCANPtr, 0U, (0x123456U|LDD_CAN_MESSAGE_ID_EXT));
                          /* 设置缓存区 0 的接收 Rx ID 为扩展帧类型*/
while (!DataFrameRxFlg) { /* 等待数据帧接收完成 */
CAN1_Main(MyCANPtr);
}
Frame.Data = InpData; /* 设置输入数据 InpData 缓存区指针 */
Error = CAN1_ReadFrame(MyCANPtr, 0U, &Frame); /* 从数据缓存区 0 中读取数据帧 */
/*
```

```
Frame.MessageID 包含了 CAN ID 节点的数值。如果 if((Frame.MessageID &
LDD_CAN_MESSAGE_ID_EXT)!=0) ，则为扩展帧格式(extended ID,)，否则为标准的 ID(standard ID)。
Frame.FrameType 表示接收帧类型，如 LDD_CAN_DATA_FRAME，Frame.Length 表示接收帧中接收的字节的
个数，InpData[] 包含了接收 Rx 数据字节。
*/
for(;;) {}
}
文件 Event.c 设置数据帧标志如下:
extern volatile bool DataFrameRxFlg;
void CAN1_OnFullRxBuffer(LDD_TUserData *UserDataPtr, LDD_CAN_TMBIndex BufferIdx)
{
DataFrameRxFlg = TRUE; /* 设置数据帧 DataFrameRxFlg 标志*/
}
```

测试结果波形：

- 在K60的CAN TX引脚检测到的正常的波形图如图 12所示。

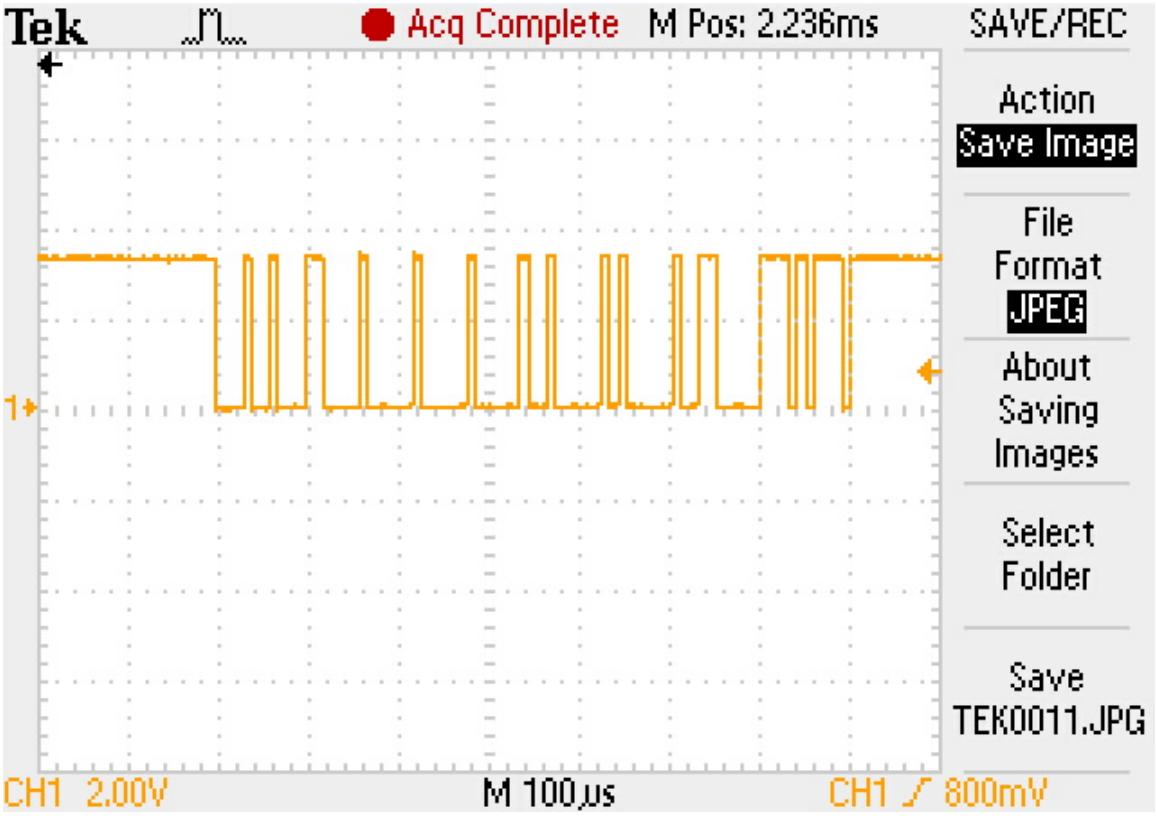


**图 12 K60 CAN发送引脚波形图**

- 由于CAN驱动芯片外围电路的错误设计（实际焊接使用的是K欧级的匹配阻抗）引起的异常的波形图如图 13所示，系统进入了Standby（待机）模式，正确的匹配阻抗请参照图 3中的波形设置。

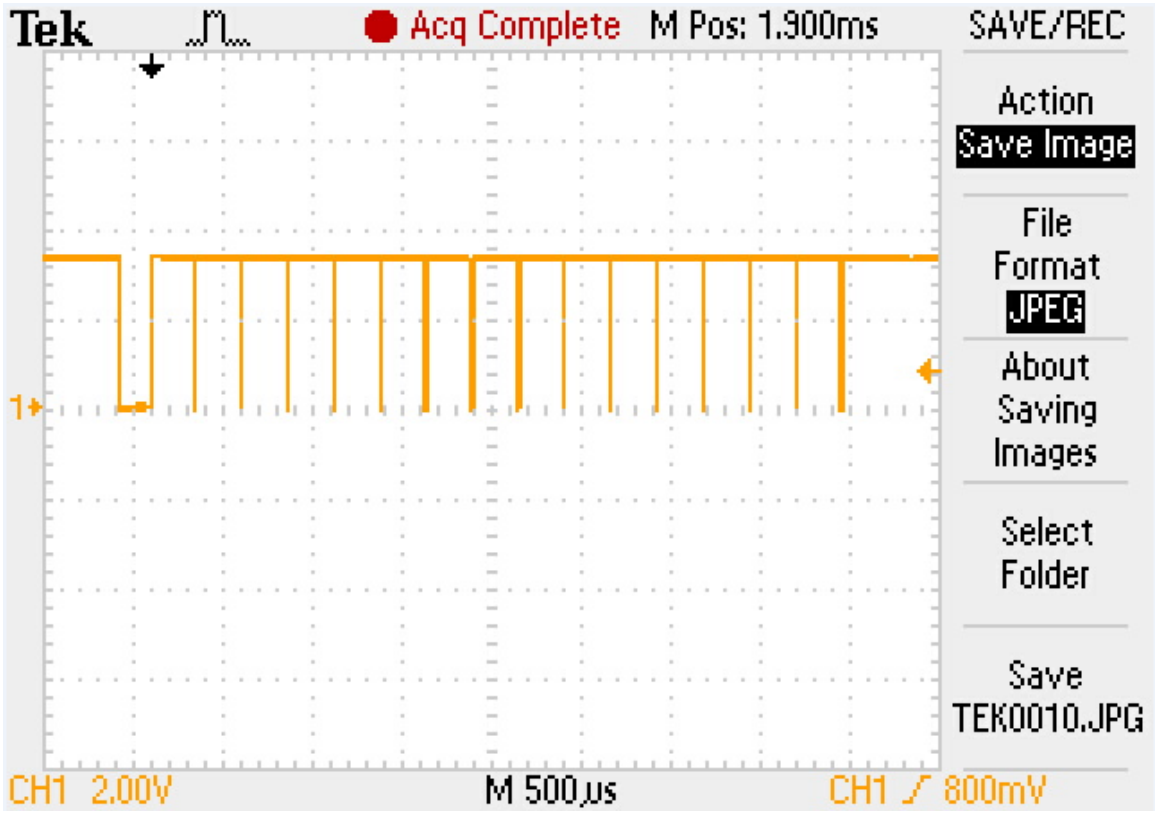


**图 13 K60 CAN发送引脚异常的波形图**

## 3.2 调试基于MQX 4.0的CAN总线工程

由于现有的很多应用都涉及到实时调度，多任务处理，CAN总线作为一个通讯外设，也需要加载到实时操作系统中。Freescale提供了免费的MQX操作系统，最新的MQX4.0版可以从Freescale的官网下载，下载链接如下：

www.freescale.com/webapp/sps/download/license.jsp?colCode=DL-MQX-CX&prodCode=MQX&appType=file2&location=null&DOWNLOAD_ID=null

下载并安装之后，在安装目录中有一个基础的CAN总线的例程，分别基于Codewarrior IDE、 Keil IDE以及IAR IDE编写的工程实例。这里以IAR IDE为例进行阐述。打开如下安装目录下的示例程序：

Freescale_MQX_4_0\mqx\examples\can\flexcan\iar\flexcan_twrk60d100m

如图 14所示。

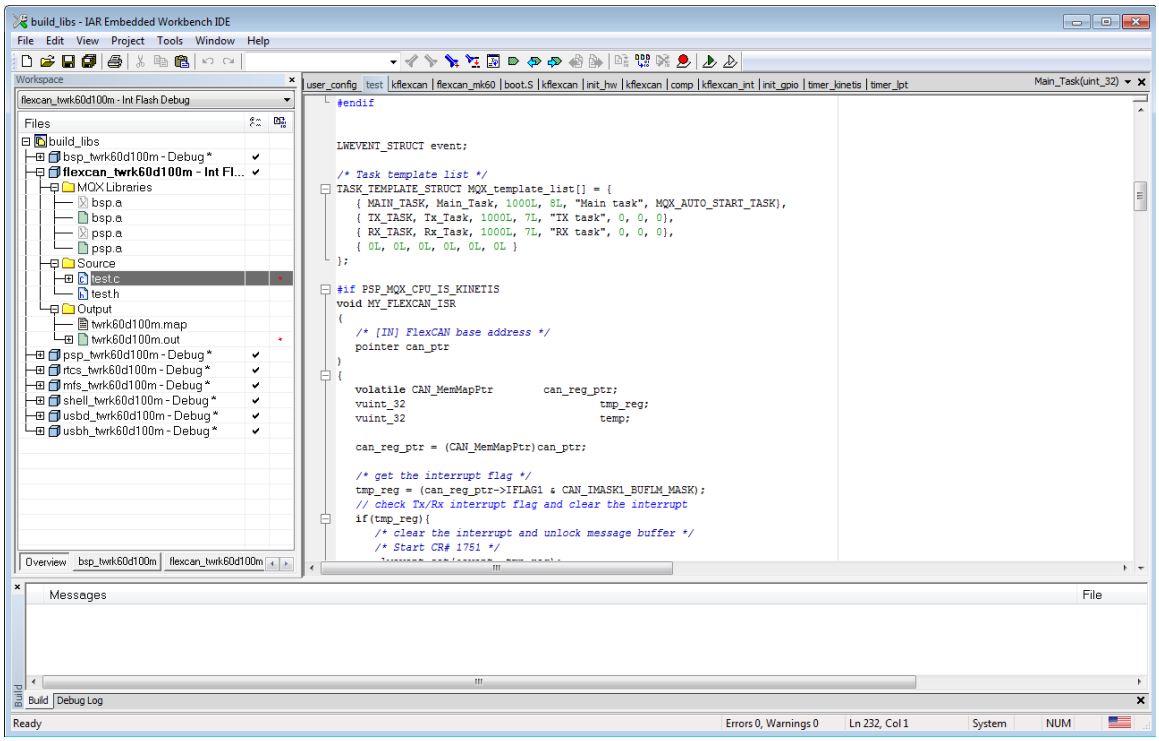


**图 14 IAR CAN调试例程**

这里由于使用了MQX下的BSP，因此在Flexcan_twrk60d100m的工程中加载了MQX的Libraries文件，需要首先编译twrk60d100m的bsp板级支持包，其路径在\Freescale_MQX_4_0\config\twrk60d100m\iar下的build_libs 工程中，直接打开就可以看到上述图 9中的bsp、psp、rtcs、mfs、shell、usbd等目录，由于和CAN驱动相关的MQX RTOS部分只有bsp.a和psp.a的库，因此在工程中只加载这两个部分。该Flexcan的工程主要包括三个任务：

- MAIN_TASK用于初始化建立CAN的寄存器配置以及相应的功能使能；
- TX_TASK用于CAN的发送；
- RX_TASK用于CAN的接收。

相应的CAN底层驱动的代码在BSP包中。

在任务启动后，在MAIN_TASK中会调用_bsp_flexcan_io_init函数来初始化CAN的配置。该函数在bsp Files下的init_gpio.c文件中，在这个文件中指定了相应的CAN总线对应的引脚。例如这里对应的是CAN1口，相应的GPIO口PORTE24，PORTE25引脚就配置成 CAN总线功能口，如果要使用其他的CAN功能引脚，可以在这里修改配置，如图 15所示。

```
/* 配置 GPIO 口最为 FlexCAN1 的外设功能 */
   pctl = (PORT_MemMapPtr)PORTE_BASE_PTR;
   pctl->PCR[24] = (PORT_PCR_MUX(2)|PORT_PCR_DSE_MASK);     /* CAN1_TX.E24  */
   pctl->PCR[25] = (PORT_PCR_MUX(2)|PORT_PCR_DSE_MASK);     /* CAN1_RX.E25  */
```

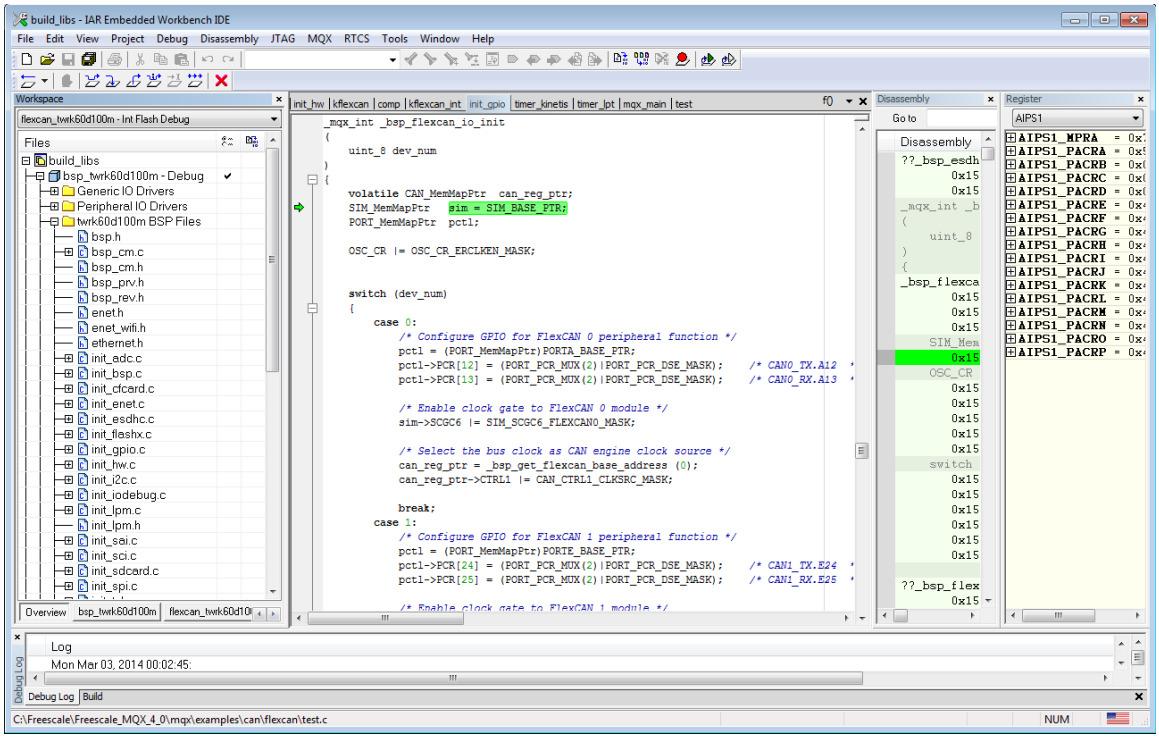


**图 15 FlexCAN的GPIO引脚设置**

接下来配置消息发送的格式，这里选择FLEXCAN_STANDARD标准帧格式，然后选择收发的邮箱号：

```
RX_mailbox_num = 0;
TX_mailbox_num = 1;
RX_remote_mailbox_num = 2;
TX_remote_mailbox_num = 3;
```

接着，选择收发节点ID号：

```
RX_identifier = 0x321;
TX_identifier = 0x123;
RX_remote_identifier = 0x00F;
TX_remote_identifier = 0x0F0;
```

涉及到配置CAN总线的波特率以及通信时序同步段、相位缓冲段等，在bsp文件下的Peripheral IO Drivers中的flexcan文件中，其包括kflexcan.c文件中的函数FLEXCAN_Initialize定义了这些配置参数，如图 16所示。

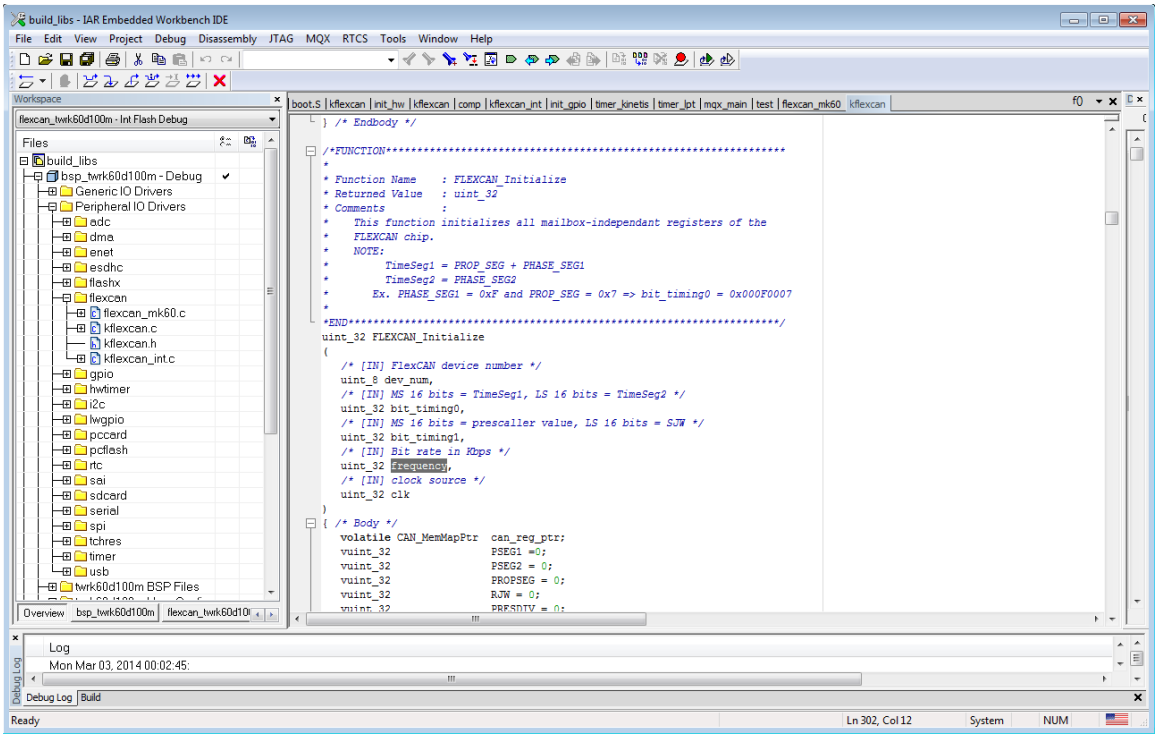


**图 16 FlexCAN的配置**

由于CAN总线是一个多路传输系统，当某一个节点出现故障时，不影响其他节点通信。针对这一总线脱离的功能，当节点有故障时该节点自动脱离总线，当节点正常时候，总线自动恢复。相关的配置代码如下。

在kflexcan.c程序中选择FlexCAN的模式：

```
uint_32 FLEXCAN_Select_mode
(
   /* [IN] FlexCAN device number */
   uint_8 dev_num,
   /* [IN] operation Mode */
   uint_32 mode
)
```

除了正常的CAN模式、回环模式等，还有总线脱离模式：

```
case (FLEXCAN_BOFFREC_MODE):
   /* Bus Off Recovery mode (according to CAN 2.0b) */
   can_reg_ptr->CTRL1 &= ~(CAN_CTRL1_BOFFMSK_MASK);
break;
```

在K60DN10.h头文件中定义了CAN0和CAN1的总线脱离中断号：

```
INT_CAN0_Bus_Off =46;
INT_CAN1_Bus_Off =54;
```

在主程序test.c中有针对总线脱离的中断服务程序配置代码如下：

```
/* Enable error interrupts */
if(flexcan_error_interrupt == 1)
{
   result = FLEXCAN_Install_isr_err_int( CAN_DEVICE, MY_FLEXCAN_ISR );
   printf("\nFLEXCAN Error ISR install, result: 0x%lx", result);
   result = FLEXCAN_Install_isr_boff_int( CAN_DEVICE, MY_FLEXCAN_ISR  );
   printf("\nFLEXCAN Bus off ISR install, result: 0x%lx", result);
   result = FLEXCAN_Error_int_enable(CAN_DEVICE);
   printf("\nFLEXCAN error interrupt enable. result: 0x%lx", result);
}
```

# 4 总结

上述裸机的PE代码和基于实时操作系统MQX4.0版本的代码在K60DN512VLQ10、K10DN512VLK10等不同的器件平台上验证通过，可以快捷的移植到其他的Cortex M4内核的Kinetis芯片中。

# 5 参考文献


- 现场总线CAN原理与应用技术　北航出版社　饶运涛
- *Freescale CAN_LDD Embedded Component User Guide* Rev. 0, 05/2011

**How to Reach Us:**

**Home Page:**
www.freescale.com

**Web Support:**
www.freescale.com/support



Document Number: AN4962
Rev. 1.0
07/2014