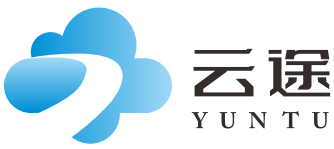
# 应用笔记

## AN0008

---

FlexCAN 基于 SDK 应用指南



# 目录

# 历史版本

| 版本号 | 日期 | 修改 |
| --- | --- | --- |
| 1.0 | 2022-10-28 | 初始版本 |

# 1 介绍

本文主要介绍了云途各系列 MCU 中 FlexCAN 模块基于 SDK 的应用方法及指南。FlexCAN 模块的功能特性及基本工作原理参考 AN0019：《FlexCAN 模块介绍》。

# 2 初始化

在 CAN 通道正常工作前需要对模块、消息邮箱进行相应的初始化。

## 2.1 CAN 模块初始化

CAN 模块初始化主要完成模块时钟初始化，CAN 波特率设定，CAN 工作模式，邮箱 payload 设定。通过 CAN_Init 实现。配置信息通过类型位 can_user_config_t 的指针传递。应用中可定义类型 can_user_config_t 的常量或变量作为初始化信息。

### 2.1.1 初始化函数 CAN_Init

```
status_t CAN_Init(const can_instance_t * const instance,
                  const can_user_config_t *config)
```

```
typedef struct
{
    uint32_t maxBuffNum;                /*!< Set maximum number of buffers */
    can_operation_modes_t mode;         /*!< Set operation mode */
    can_clk_source_t peClkSrc;          /*!< The clock source of the CAN Protocol Engine (PE). */
    bool enableFD;                      /*!< Enable flexible data rate */
    can_fd_payload_size_t payloadSize;  /*!< Set size of buffer payload */
    can_time_segment_t nominalBitrate;  /*!< Bit timing segments for nominal bitrate */
    can_time_segment_t dataBitrate;     /*!< Bit timing segments for data bitrate */
    void *extension;                    /*!< This field will be used to add extra settings to the
                                             basic configuration like FlexCAN Rx FIFO settings */
} can_user_config_t;
```

- maxBuffNum：最大邮箱数量
  - 可根据要使用的邮箱数量定义最大邮箱数量，如应用中使用到 n 个邮箱，则此值配置为 n，例如 YTM32B1LD 系列 CAN0 通道 n 的范围 1~32。
  - 当使能接收 FIFO 时，接收 FIFO 所占用的 mailbox 空间必须减掉。
- mode:CAN 模式
  - 运行模式。
  - 回环模式。
  - 停止模式。此模式下，初始化完成后，CAN 模块处于离线模式，不处理总线事物

```
typedef enum {
   CAN_NORMAL_MODE = 0U,        /*!< Normal mode or user mode */
   CAN_LOOPBACK_MODE = 2U,      /*!< Loop-back mode */
   CAN_DISABLE_MODE = 4U,       /*!< Module disable mode */
} can_operation_modes_t;
```

- peClkSrc
  - CAN_CLK_SOURCE_OSC：外部高频晶振 FXOSC 作为 CAN 模块始终
  - CAN_CLK_SOURCE_PERIPH：fastbus clock 作为 CAN 模块始终

```
   typedef enum {
      CAN_CLK_SOURCE_OSC    = 0U,  /*!< The CAN engine clock source is the oscillator clock. */
      CAN_CLK_SOURCE_PERIPH = 1U   /*!< The CAN engine clock source is the peripheral clock. */
   } can_clk_source_t;
```

注：当选用 *OSC* 作为时钟源时，*MCU* 的时钟模块初始化时必须使能 *FOSC*

- enableFD
  - ture: 使能 CANFD 模式，CAN 模块将工作在 CANFD 模式下
  - false：禁止 CANFD 模式，CAN 模块将工作经典 CAN 模式下 注：当使能 *CANFD* 模式时，*RX FIFO* 必须禁用
- payloadSize：邮箱负载模式选择
  - 当 CAN 被配置为工作在经典模式，次值必须配置为 8 负载
  - 当 CAN 被配置为工作在 CANFD 模式时，可配置为 8、16、32、64 负载

```
typedef enum {
   CAN_PAYLOAD_SIZE_8 = 0,  /*!< CAN message buffer payload size in bytes */
   CAN_PAYLOAD_SIZE_16 ,    /*!< CAN message buffer payload size in bytes */
   CAN_PAYLOAD_SIZE_32 ,    /*!< CAN message buffer payload size in bytes */
   CAN_PAYLOAD_SIZE_64      /*!< CAN message buffer payload size in bytes */
} can_fd_payload_size_t;
```

- nominalBitrate：CAN 波特率配置
- dataBitrate：CANFD 波特率配置
- extension：CAN FIFO 配置

```
{
  is_rx_fifo_needed,      /*!< Enable or disable RX FIFO. */
  num_id_filters,         /*!< The number of RX FIFO ID filters needed. */
  transfer_type,          /*!< Specifies if the Rx FIFO uses interrupts or DMA. */
  rxFifoDMAChannel,       /*!< Specifies the DMA channel number to be used for DMA transfers. */
}
```

- 当不开启接收 FIFO，可配置 extension = NULL
- 当开启 FIFO 时：
  - is_rx_fifo_needed = true
  - num_id_filters：配置 FIFO 滤波器组滤波器个数，此值必须为 8 的整数倍，所以必须为枚举类型 flexcan_rx_fifo_id_filter_num_t 中的值。
  - transfer_type：FIFO 接收处理过程为中断方式或 DMA 方式，必须为枚举类型 flexcan_rxfifo_transfer_type_t
  - rxFifoDMAChannel：当 transfer_type 为 DMA 方式时，此值为 DMA 通道号

### 2.1.2 默认配置

- 滤波器掩码采用为全局滤波器模式，掩码值为 0xFFFFFFFF
- 发送 Abort 使能
- Bussoff 自动恢复使能

### 2.1.3 配置示例：

- 经典 CAN 模式，禁用 RX FIFO，24M 外部 OSC 作为 CAN 时钟源，500K 波特率，采样点 81.25%

```
const can_user_config_t can_pal_Config = {
    .maxBuffNum = 32UL,                  /*!< Default config to CAN0 max buffer number: 32 */
    .mode = CAN_NORMAL_MODE,             /*!< Goto run mode after init. */
```

```
    .peClkSrc = CAN_CLK_SOURCE_OSC,       /*!< 24M OSC as CAN module cloce. */
    .enableFD = false,                    /*!< CAN module work in classical mode. */
    .payloadSize = CAN_PAYLOAD_SIZE_8,    /*!< 8 byte payload size per mailbox */
    .nominalBitrate =                     /*!< CAN bandrate config to : 500Kbps */
    {
        .propSeg = 3UL,
        .phaseSeg1 = 7UL,
        .phaseSeg2 = 2UL,
        .preDivider = 2UL,
        .rJumpwidth = 1UL
    },
    .extension = NULL                   /*!< Disable RX FIFO */
};
```

- 经典 CAN 模式；使能 RX FIFO，FIFO 采用终端方式接收数据，16 组滤波器，24M 外部 OSC 作为 CAN 时钟源，500K 波特率，采样点 81.25%

```
const can_user_config_t can_pal_Config = {
    .maxBuffNum = 32UL,                   /*!< Default config to CAN0 max buffer number: 32 */
    .mode = CAN_NORMAL_MODE,              /*!< Goto run mode after init. */
    .peClkSrc = CAN_CLK_SOURCE_OSC,       /*!< 24M OSC as CAN module cloce. */
    .enableFD = false,                    /*!< CAN module work in classical mode. */
    .payloadSize = CAN_PAYLOAD_SIZE_8,    /*!< 8 byte payload size per mailbox */
    .nominalBitrate =                     /*!< CAN bandrate config to : 500Kbps */
    {
       .propSeg = 3UL,
       .phaseSeg1 = 7UL,
       .phaseSeg2 = 2UL,
       .preDivider = 2UL,
       .rJumpwidth = 1UL
    },
    .extension =
    {
       .is_rx_fifo_needed = true,                       /*!< Enable rx fifo. */
       .num_id_filters = FLEXCAN_RX_FIFO_ID_FILTERS_8,  /*!< 8 rx filter */
       .transfer_type = FLEXCAN_RXFIFO_USING_INTERRUPTS,  /*!< using interrupt mode to receive message */
    }
};
```

- CANFD 模式；24M 外部 OSC 作为 CAN 时钟源，64 字节邮箱负载；波特率：500K，采样点 87.5%；数据段可变波特率：2000K，采样点 83.3%；

```
const can_user_config_t can_pal_Config = {
   .maxBuffNum = 32UL,                   /*!< Default config to CAN0 max buffer number: 32 */
   .mode = CAN_NORMAL_MODE,              /*!< Goto run mode after init. */
   .peClkSrc = CAN_CLK_SOURCE_OSC,       /*!< 24M OSC as CAN module cloce. */
   .enableFD = false,                    /*!< CAN module work in classical mode. */
   .payloadSize = CAN_PAYLOAD_SIZE_64,   /*!< 64 byte payload size per mailbox */
   .nominalBitrate =                     /*!< CAN bandrate config to : 500Kbps */
  {
      .propSeg = 12UL,
      .phaseSeg1 = 27UL,
      .phaseSeg2 = 5UL,
      .preDivider = 0UL,
      .rJumpwidth = 5UL
  },
  .dataBitrate =                        /*!< CANFD bandrate config to : 2000Kbps */
  {
      .propSeg = 7UL,
      .phaseSeg1 = 1UL,
      .phaseSeg2 = 1UL,
      .preDivider = 0UL,
      .rJumpwidth = 1UL
  },
.extension = NULL                   /*!< Must disable RX FIFO */
};
```

## 2.2 CAN 消息邮箱初始化

### 2.2.1 消息邮箱初始化

通过函数 CAN_ConfigRxBuff 和 CAN_ConfigTxBuff 配置接收消息邮箱和发送消息邮箱

```
status_t CAN_ConfigTxBuff(const can_instance_t * const instance,
                          uint32_t buffIdx,
                          const can_buff_config_t *config)

status_t CAN_ConfigRxBuff(const can_instance_t * const instance,
                          uint32_t buffIdx,
                          const can_buff_config_t *config,
                          uint32_t acceptedId)
```

- buffIdx：邮箱编号
  - 当 FIFO 禁用时，buffIdx 的取值范围为 0~n
  - 当 FIFO 使能时，buffIdx 的取值范围为 1~n 注：最大值 *n* 取决于 *FIFO* 是否使能和 *CAN* 初始化时设置的 *maxBuffNum* 值，当 *buffIdx* 超出最大值时，函数返回 *STATUS_CAN_BUFF_OUT_OF_RANGE*
- *config: 配置信息，参照以下结构体定义

```
typedef struct {
     bool enableFD;               /*!< Enable flexible data rate */
     bool enableBRS;              /*!< Enable bit rate switch inside a CAN FD frame */
     uint8_t fdPadding;           /*!< Value used for padding when the data length code (DLC)
                                     specifies a bigger payload size than the actual data length */
     can_msg_id_type_t idType;    /*!< Specifies whether the frame format is standard or extended */
     bool isRemote;               /*!< Specifies if the frame is standard or remote */
 } can_buff_config_t;
```

- acceptedId：接收邮箱滤波器接受码
  - 当被用作标准帧 (Standard Frame) 接收邮箱时，范围为 0~0x7FF；
  - 当被用作扩展帧 (Extended Frame) 接收邮箱时，范围为 0~0x1FFFFFFF；

### 2.2.2 接收 FIFO 初始化 Rx FIFO Init

当初始化使能 FIFO 后，通过函数 FLEXCAN_DRV_ConfigRxFifo(声明在 flexcan_driver.h 中) 配置接收 FIFO

```
void FLEXCAN_DRV_ConfigRxFifo(
    uint8_t instance,
    flexcan_rx_fifo_id_element_format_t id_format,
    const flexcan_id_table_t *id_filter_table)
```

- instance：CAN 控制器索引号，需引用 CAN 模块初始化结构体 can_instance_t 中 instIdx 元素值
- id_format：接收滤波器格式定义，详细参考芯片参考手册或《FlexCAN 模块介绍》

| 接收滤波器格式（id_format） | 标准帧滤波规则 | 扩展帧滤波规则 |
| --- | --- | --- |
| FLEXCAN_RX_FIFO_ID_FORMAT_A | 所有 ID 位比较 | 所有 ID 位比较 |
| FLEXCAN_RX_FIFO_ID_FORMAT_B | 所有 ID 位比较 | MSB14 位 ID 比较 |
| FLEXCAN_RX_FIFO_ID_FORMAT_C | MSB8 位 ID 比较 | MSB8 位 ID 比较 |

```
typedef enum {
     FLEXCAN_RX_FIFO_ID_FORMAT_A, /*!< One full ID (standard and extended) per ID Filter Table element.*/
     FLEXCAN_RX_FIFO_ID_FORMAT_B, /*!< Two full standard IDs or two partial 14-bit (standard and
                                     extended) IDs per ID Filter Table element.*/
     FLEXCAN_RX_FIFO_ID_FORMAT_C, /*!< Four partial 8-bit Standard IDs per ID Filter Table element.*/
```

```
} flexcan_rx_fifo_id_element_format_t;
```

- *id_filter_table：接收滤波器接受码表

```
typedef struct {
    bool isRemoteFrame;      /*!< Remote frame*/
    bool isExtendedFrame;    /*!< Extended frame*/
    uint32_t id;             /*!< Rx FIFO ID filter element*/
} flexcan_id_table_t;
```

注： - 此结构体中仅包含接收滤波器的接受码信息 - 接收滤波器接受码表必须为 8*(n+1)*(x+1)，其中：+n：寄存器中设置的滤波器数量信息，即结构体 can_user_config_t 中 extension 的 num_id_filters 元素值 + x：寄存器中设置的滤波器格式信息 - 调用 CAN_Init 初始化后，默认会采用全局掩码模式，并设置掩码位全为 1

#### 2.2.2.1 应用示例

- 接收滤波器格式为 FLEXCAN_RX_FIFO_ID_FORMAT_A，8 个滤波器，掩码采用初始化默认配置全局掩码模式，全局掩码值 0xFFFFFFFF。RxFIFO 将接收 ID 为与 rxFifoAccId 数组中的 ID 匹配的所有报文

```
#define CAN0_RX_FIFO_FILTER_COUNT (8 * (FLEXCAN_RX_FIFO_ID_FILTERS_8 + 1))
const uint32_t rxFifoAccId[CAN0_RX_FIFO_FILTER_COUNT] =
{
    0x0C1001F0U,
    0x081002F0U,
    0x0C0803A5U,
    0x0C1004F0U,
    0x0C1005E0U,
    0x0C1006F3U,
    0x0C1007F0U,
    0x0C10080FU,
};
const flexcan_id_table_t rxFifoFilterAccInfo[CAN0_RX_FIFO_FILTER_COUNT];

// Do something befor
/* Init rx fifo filter table acceptence code */
for(uint8_t i = 0;i < (uint8_t)CAN0_RX_FIFO_FILTER_COUNT;i++)
{
    rxFifoFilterAccInfo[i].isRemoteFrame = false;
    rxFifoFilterAccInfo[i].isExtendedFrame = true;
    rxFifoFilterAccInfo[i].id = rxFifoAccId[i];
}
// Do something after
```

- 接收滤波器格式为 FLEXCAN_RX_FIFO_ID_FORMAT_B，8 个滤波器，掩码采用初始化默认配置全局掩码模式，全局掩码值 0xFFFFFFFF。RxFIFO 将接收 ID 与 0x0C1Cxxxx ~ 0x0C2Bxxxx 匹配的所有报文
  - 如果为标准帧，所有 ID 位均比较
  - 如果为扩展帧，仅 ID 高 14 位比较

```
/* define filter table length
   filter table length must equal to : 8*(n+1)*(x+1)
      n : filter table length code configed in the register
      x : filter formate code configed in the register */
#define CAN0_RX_FIFO_FILTER_COUNT (8 * (FLEXCAN_RX_FIFO_ID_FILTERS_8 + 1) * 2)
/* Init rx fifo filter table acceptence code */
const flexcan_id_table_t rxFifoFilterAccInfo[CAN0_RX_FIFO_FILTER_COUNT] =
{
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C1C01F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C1D01F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C1E01F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C1F01F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2001F0U},
```

```
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2101F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2201F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2301F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2401F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2501F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2601F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2701F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2801F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2901F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2A01F0U},
    {.isRemoteFrame = false,.isExtendedFrame = true,.id = 0x0C2B01F0U},
};
```

附：*RxFIFO* 可接收混合帧，即又接收标准帧又接受扩展帧

### 2.2.3 掩码设置

接收滤波器掩码可配置为全局掩码模式和独立掩码模式。

# 3 报文发送

## 3.1 发送函数 CAN_Send

通过调用 CAN_Send 函数实现各个消息邮箱的报文发送。CAN_Send 函数通过中断方式发送 CAN 报文，当调用 CAN_Send 后，首先开启相应邮箱 CAN 发送中断，当消息发送完成后，关闭相应邮箱中断，发送完成。可通过调用 CAN_GetTransferStatus 检查邮箱是否处于空闲状态。

```
status_t CAN_Send(const can_instance_t * const instance,
                  uint32_t buffIdx,
                  const can_message_t *message)
```

- can_instance_t：CAN 实例索引，CAN_Init 中实例索引
- buffIdx：邮箱索引号
- message：CAN 消息指针

```
typedef struct {
    uint32_t cs;       /*!< Code and Status*/
    uint32_t id;       /*!< ID of the message */
    uint8_t data[64];  /*!< Data bytes of the CAN message*/
    uint8_t length;    /*!< Length of payload in bytes */
} can_message_t;
```

## 3.2 应用示例

```
can_message_t txMsg = {
    .cs = 0U,
    .id = 0x18FF0101U,
    .data = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07},
    .length = 8U,
};

/* Send CAN message via mailbox-0 */
if (CAN_GetTransferStatus(&can_pal_instance, 0) != STATUS_BUSY)
{
    /* Update data */
    txMsg.data[0] = 0x32;
    txMsg.data[0] = 0x24;
    txMsg.data[0] = 0x25;
```

```
    /* Send the message */
    status |= CAN_Send(&can_pal_instance, 0, &txMsg);
}
```

# 4 报文接收

## 4.1 接收函数 CAN_Receive

通过调用 CAN_Receive 函数实现各个消息邮箱的报文发送。CAN_Receive 函数通过中断方式发送 CAN 报文，当调用 CAN_Receive 后，首先开启相应邮箱 CAN 接收中断，当消息发送完成后，关闭相应邮箱中断，接收完成。可通过调用 CAN_GetTransferStatus 检查邮箱是否处于空闲状态。

## 4.2 应用示例: 查询的方式接收

```
can_message_t rxMsg;

/* Send CAN message via mailbox-1 */
if (CAN_GetTransferStatus(&can_pal_instance, 1) != STATUS_BUSY)
{
    status |= CAN_Receive(&can_pal_instance, 1, &rxMsg);
}
```

# 5 发送及接收中断

## 5.1 中断事件回调函数注册

可通过 CAN_InstallEventCallback 函数来注册接收及发送中断事件回调函数。

## 5.2 应用示例: 发送及接收中断处理程序注册

```
#define CAN0_RX_MSG_COUNT (3U)
#define CAN0_TX_MSG_COUNT (4U)
const can_buff_config_t can0_BuffCfg = {
    .idType = CAN_MSG_ID_STD,
};
can_message_t can0_TxMsgs[CAN0_TX_MSG_COUNT] = {
    {
        .id = 0x510,
        .length = 8,
    },
    {
        .id = 0x520,
        .length = 8,
    },
    {
        .id = 0x530,
        .length = 8,
    },
};
can_message_t can0_RxMsgs[CAN0_RX_MSG_COUNT] = {
    {
        .id = 0x110,
        .length = 8,
    },
    {
        .id = 0x120,
        .length = 8,
```

```
    },
    {
        .id = 0x130,
        .length = 8,
    },
};

void can0_mb_CallBack(uint32_t instance, can_event_t eventType, uint32_t objIdx, void *driverState)
{
    uint32_t index;
    (void)driverState;
    if(0 == instance)
    {
        switch (eventType)
        {
        case CAN_EVENT_RX_COMPLETE:
            /* Traverse all rx mailbox to find which mailbox has occured rx event */
            for (index = 0; index < CAN0_RX_MSG_COUNT; index++)
            {
                if (objIdx == index)
                {
                    /* Call upper layer to indication the rx event */
                    // Do something
                    /* Start mailbox to receive next message */
                    CAN_Receive(&can_pal_instance,
                                index,
                                (can_message_t *)&can0_RxMsgs[index]);
                }
            }
            break;
        case CAN_EVENT_TX_COMPLETE:
            /* Traverse all tx mailbox to find which mailbox has occured tx event */
            for (index = 0; index < CAN0_TX_MSG_COUNT; index++)
            {
                if (objIdx == index + CAN0_RX_MSG_COUNT)
                {
                    /* Call upper layer to confirm the tx event */
                    // Do something
                }
            }
            break;
        default:
            break;
    }
}

int main(void)
{
    // Do something befor
    status = CAN_Init(&can_pal_instance, &can_pal_Config);
    status = CAN_InstallEventCallback(&can_pal_instance, can0_mb_CallBack, NULL);
    /* Config rx messages mailbox, hardware mailbox 0~2 used as rx message mailbox */
    for (index = 0; index < CAN0_RX_MSG_COUNT; index++)
    {
        /* Config rx buffer */
        status = CAN_ConfigRxBuff(&can_pal_instance,
                                  index,
                                  &can0_BuffCfg,
                                  (uint32_t)can0_RxMsgs[index].id);
        /* start to receive message */
        status = CAN_Receive(&can_pal_instance,
                             index,
                             (can_message_t *)&can0_RxMsgs[index]);
    }
    /* Config tx messages mailbox, hardware mailbox 3~6 used as tx message mailbox */
    for (index = 0; index < CAN0_TX_MSG_COUNT; index++)
    {
```

```
        /* Config tx buffer */
        status = CAN_ConfigTxBuff(&can_pal_instance,
                                  index + CAN0_RX_MSG_COUNT,
                                  &can0_BuffCfg);

    }
    // Do something after

    for(;;)
    {
        // Do something befor
        Task_100ms()
        {
            uint32_t index = 0;
            /* Tx all messages in the can0_TxMsgs */
            for (index = 0; index < CAN0_TX_MSG_COUNT; index++)
            {
                if(CAN_GetTransferStatus(&can_pal_instance, index + CAN0_RX_MSG_COUNT) != STATUS_BUSY)
                {
                    /* Update Tx message data */
                    ......
                    /* Send message */
                    CAN_Send(&can_pal_instance, index + CAN0_RX_MSG_COUNT, &can0_TxMsgs[index]);
                }
            }
        }
        // Do something after
    }
}
```

# 6 总线错误处理

根据 ISO11898 定义的总线故障，当 CAN 总线通讯发生故障时，FlexCAN 控制器可检测到各种错误。当调用 CAN_Init 完成 CAN 模块初始化时，默认开启 CAN BusError 中断和 CAN Busoff 中断，并开启 BusOff 自动恢复功能。

## 6.1 错误中断事件回调函数注册

当产生 BusError 中断时，用户可通过 FLEXCAN_DRV_InstallErrorCallback 注册回调函数，处理 BusError 事务。同理 BusOff 事件处理方法相同。

## 6.2 应用示例：BusError 事件处理程序注册

```
void can0_error_CallBack(uint8_t instance, flexcan_event_type_t eventType, flexcan_state_t *flexcanState)
{
    uint32_t errCode = 0;
    (void)flexcanState;
    if(eventType == FLEXCAN_EVENT_ERROR)
    {
        errCode = FLEXCAN_DRV_GetErrorStatus(instance);
        if((bool)((errCode & CAN_ESR1_BOFFINT_MASK) >> CAN_ESR1_BOFFINT_SHIFT))
        {
            /* Add busoff process routine */
            // Do something
        }
        else if((bool)((errCode & CAN_ESR1_ERRINT_MASK) >> CAN_ESR1_ERRINT_SHIFT))
        {
            /* Add buserror process routine */
            // Do something
        }
        else
```

```
        {
        }
    }
}

int main(void)
{
    // Do something befor
    status |= CAN_Init(&can_pal_instance, &can_pal_Config);
    FLEXCAN_DRV_InstallErrorCallback(can_pal_instance.instIdx, can0_error_CallBack, NULL);
    // Do something after
}
```

# 版权与联系方式



**联系我们:**
**主页:** www.ytmicro.com