Use M480 CAN in RTX

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This code demos CAN interface in RTX
BSP Version	M480 Series BSP CMSIS V3.03.000
Hardware	NuMaker-ETM-M487 v1.2

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Function Description

1.1 Introduction

When a microcontroller application has to handle many tasks at the same time, we can use RTOS (real-time operating system) to help us manage these tasks. In RTOS system, CPU processing time is divided into a number of time slots. Each time slot processes different task and the switching between these tasks can be hundred times per second. It means that the application tasks are running simultaneously. When the application becomes more complicated, we can use RTOS to help us manage these tasks.

CMSIS-RTOS in Keil MDK (Microcontroller Development Kit) provides a standard API for RTOS and application layer, includes the Keil free RTOS, RTX (Real-Time eXecutive) kernel. Once others RTOS can support CMSIS-RTOS, we can also port our application to others RTOS platform easily. This sample code uses RTX kernel, which is provided by Keil CMSIS-MDK, to create a RTX kernel system to execute CAN task.

1.2 Principle

In the example code, it demos how to communicate the messages between two CAN interfaces on different M480 target board, respectively. For different boards, two software Keil projects named by RTX_CAN0 and RTX_CAN1 are created for message communication. The following figure describes the related hardware connection.

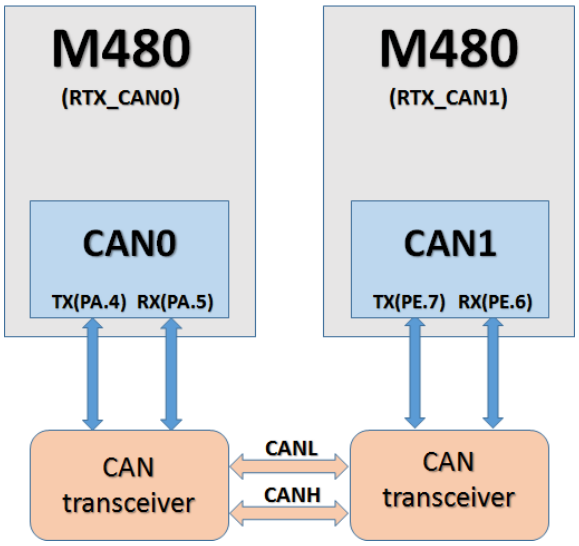
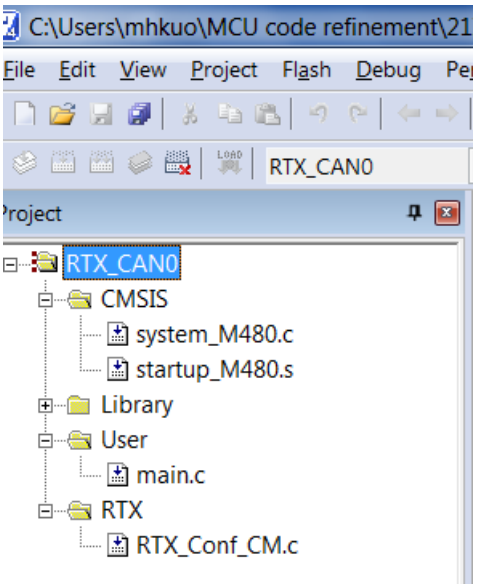


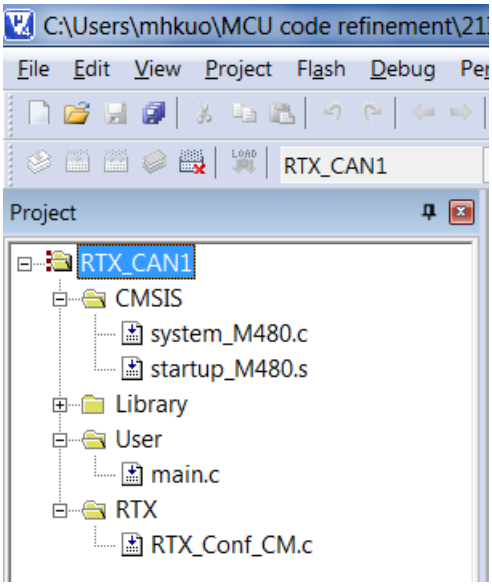
Fig. 1 Hardware connection between RTX_CAN0 and RTX_CAN1

In the beginning of program running, RTX_CAN0 will send the message to RTX_CAN1, and then waits for the response from RTX_CAN1. Therefore, we shall let RTX_CAN1 be executed firstly.

- CAN0 software project



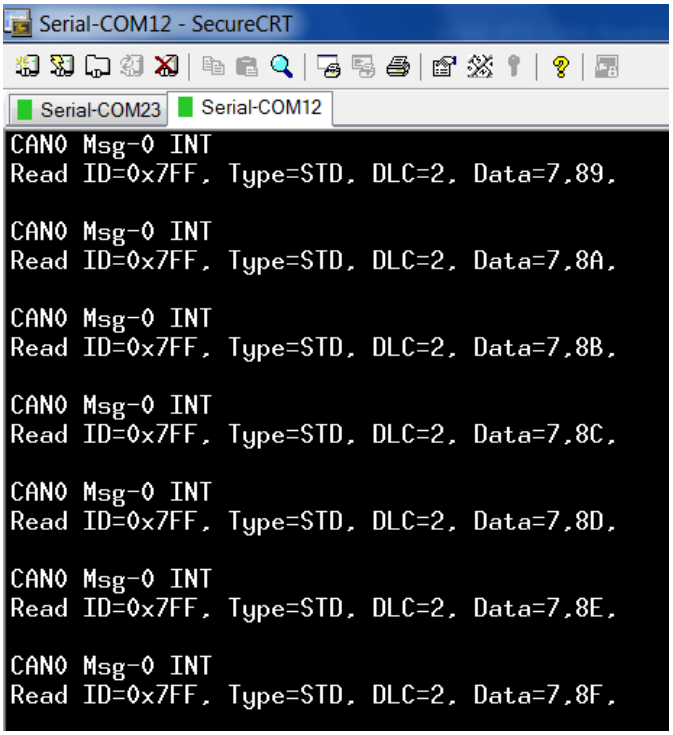
- CAN1 software project



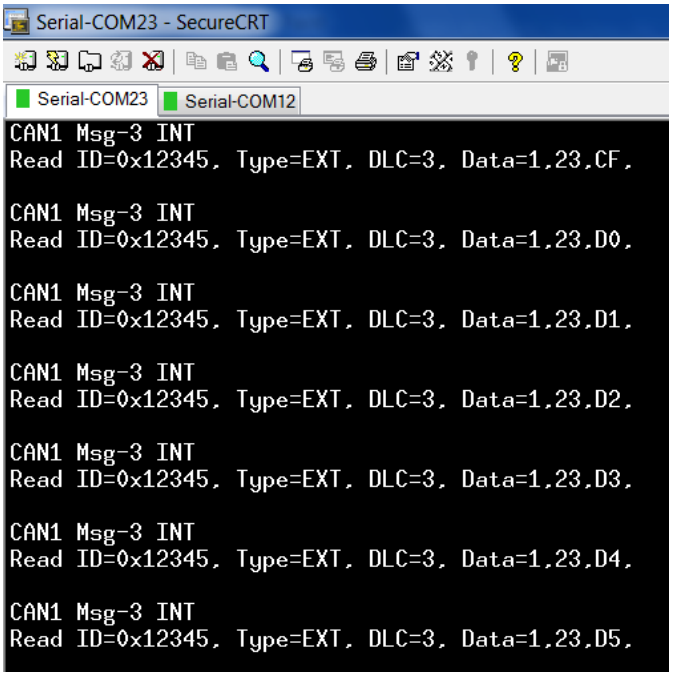
1.3 Demo Result

After connecting two CAN interfaces and running the corresponding sample codes in RTX_CAN0 and RTX_CAN1, we can get the following communication message.

- RTX_CAN0 received message



- RTX_CAN1 received message



2 Code Description

2.1 RTX

- Thread creation

This sample code has two tasks for CAN0 and CAN1. The taskCAN0 is used for RTX_CAN0, and taskCAN1 is for RTX_CAN1, respectively.

```
__task void os_init (void)
{
     OS_TID taskCAN0, taskCAN1;
 #ifdef OPT_RTX_TX     // for RTX_CAN0 project
     taskCAN0 = os_tsk_create(NormalMode_CAN0, 1);
#endif
#ifdef OPT_RTX_RX     // for RTX_CAN1 project
     taskCAN1 = os_tsk_create(NormalMode_CAN1, 2);
#endif
     os_tsk_delete_self();
}
```

2.2 CAN

- RTX_CAN0

In RTX_CAN0, the CAN0 interface is used to communicate with RTX_CAN1. Before to receive the message from CAN1, the message ID (CAN1_NODE_ID) is given in CAN_SetRxMsg() function to filter the received message.

```
    /* set received message */
    if(CAN_SetRxMsg(psCAN, MSG(CAN0_RECEIVE_MSG_NUM), CAN_STD_ID, CAN1_NODE_ID) == FALSE)
        printf("ser RX message object failed\n");
```

Before to send message, the CAN0_NODE_ID is given for RTX_CAN1 to receive it.

```
    /* setup transmitted message */
    sMsg.FrameType = CAN_DATA_FRAME;
    sMsg.IdType    = CAN_EXT_ID;
    sMsg.Id        = CAN0_NODE_ID;
    sMsg.DLC       = CAN0_TRANSMIT_MSG_LENGTH;
    sMsg.Data[0]   = CAN0_TRANSMIT_PATTERN0;
    sMsg.Data[1]   = CAN0_TRANSMIT_PATTERN1;
    sMsg.Data[2]   = CAN0_TRANSMIT_PATTERN2;
```

- RTX_CAN1

In RTX_CAN1, the CAN0 interface is used to communicate with RTX_CAN0. Before to receive the message from CAN0, the message ID (CAN0_NODE_ID) is given in CAN_SetRxMsg() function to filter the received message.

```
    /* setup received message */
    if(CAN_SetRxMsg(psCAN, MSG(CAN1_RECEIVE_MSG_NUM), CAN_EXT_ID, CAN0_NODE_ID) == FALSE)
        printf("ser RX message object failed\n");
```

Before to send message, the CAN1_NODE_ID is given for RTX_CAN0 to receive it.

```
    /* setup transmitted message */
    sMsg.FrameType = CAN_DATA_FRAME;
    sMsg.IdType    = CAN_STD_ID;
    sMsg.Id        = CAN1_NODE_ID;
    sMsg.DLC       = CAN1_TRANSMIT_MSG_LENGTH;
    sMsg.Data[0]   = CAN1_TRANSMIT_PATTERN0;
    sMsg.Data[1]   = CAN1_TRANSMIT_PATTERN1;
```

3 Software and Hardware Environment

Software Environment

- BSP version
  - M480 Series BSP CMSIS V3.04.000
- IDE version
  - Keil uVersion 4.54

Hardware Environment

- Circuit components
  - NuMaker-ETM-M487 v1.2
  - CAN transceiver (SN65HVD230)
- Diagram

  With the hardware connection, please to the above Fig.1.

4 Directory Information

EC_M480_RTX_CAN_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and can get both RTX_CAN0.uvproj and RTX_CAN1.uvproj. Use two IDE to open these two projects, respectively.
2. Enter Keil compile mode
   a. Build two projects separately.
   b. Firstly, download RTX_CAN1 to the target board with CAN1 interface connected.
   c. Then, download RTX_CAN0 to another board with CAN0 interface connected.
   d. Start/Stop debug session
3. Enter debug mode
   a. Run RTX_CAN1 before RTX_CAN0.

6 Revision History

Date	Revision	Description
Sep. 27, 2019	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.