

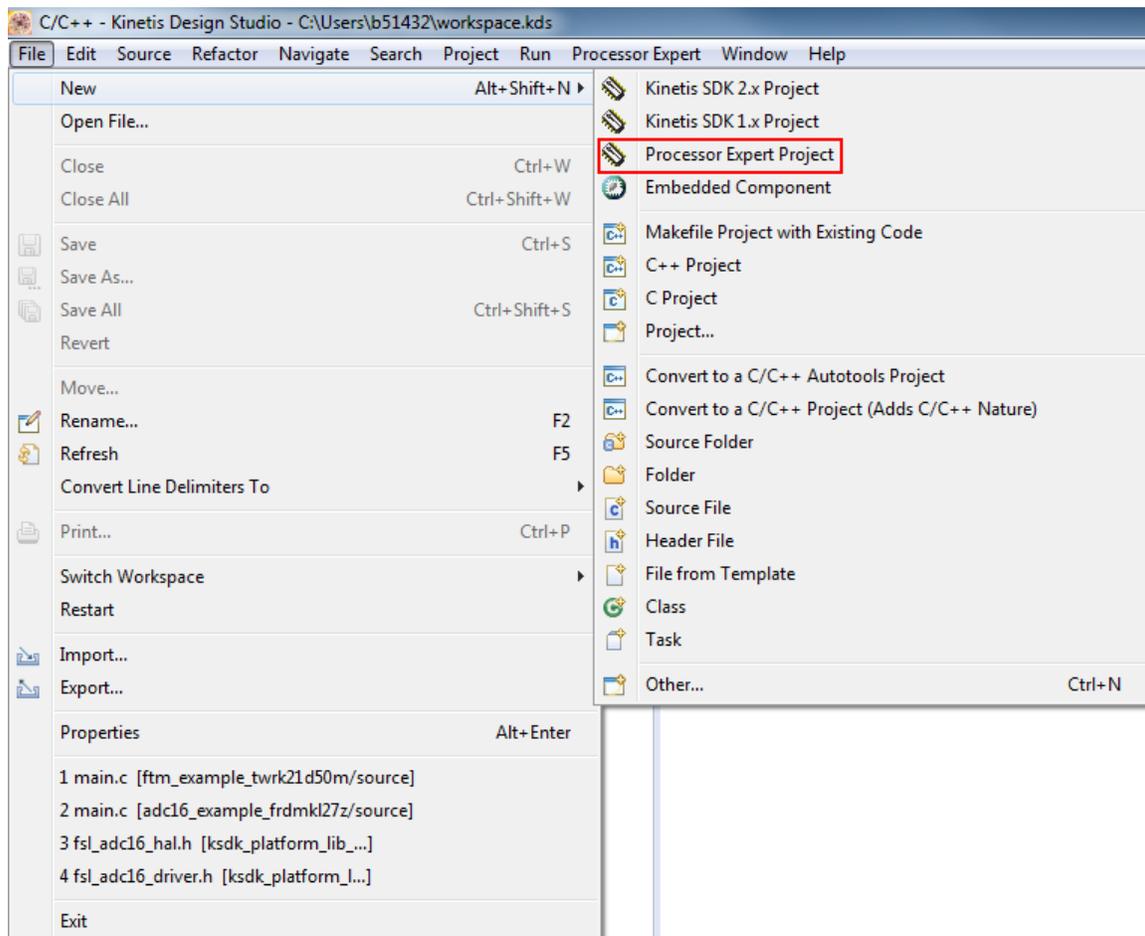
How to Start CAN Module Development on KDS v3.2.0 + Processor Expert

This document introduces how to develop a simple CAN module application on KDS, it is suitable for the beginners to start using the KDS and Processor Expert, and it mainly focus on how to developing a CAN Loopback application, how to use the “Typical Usage” of “help on component”.

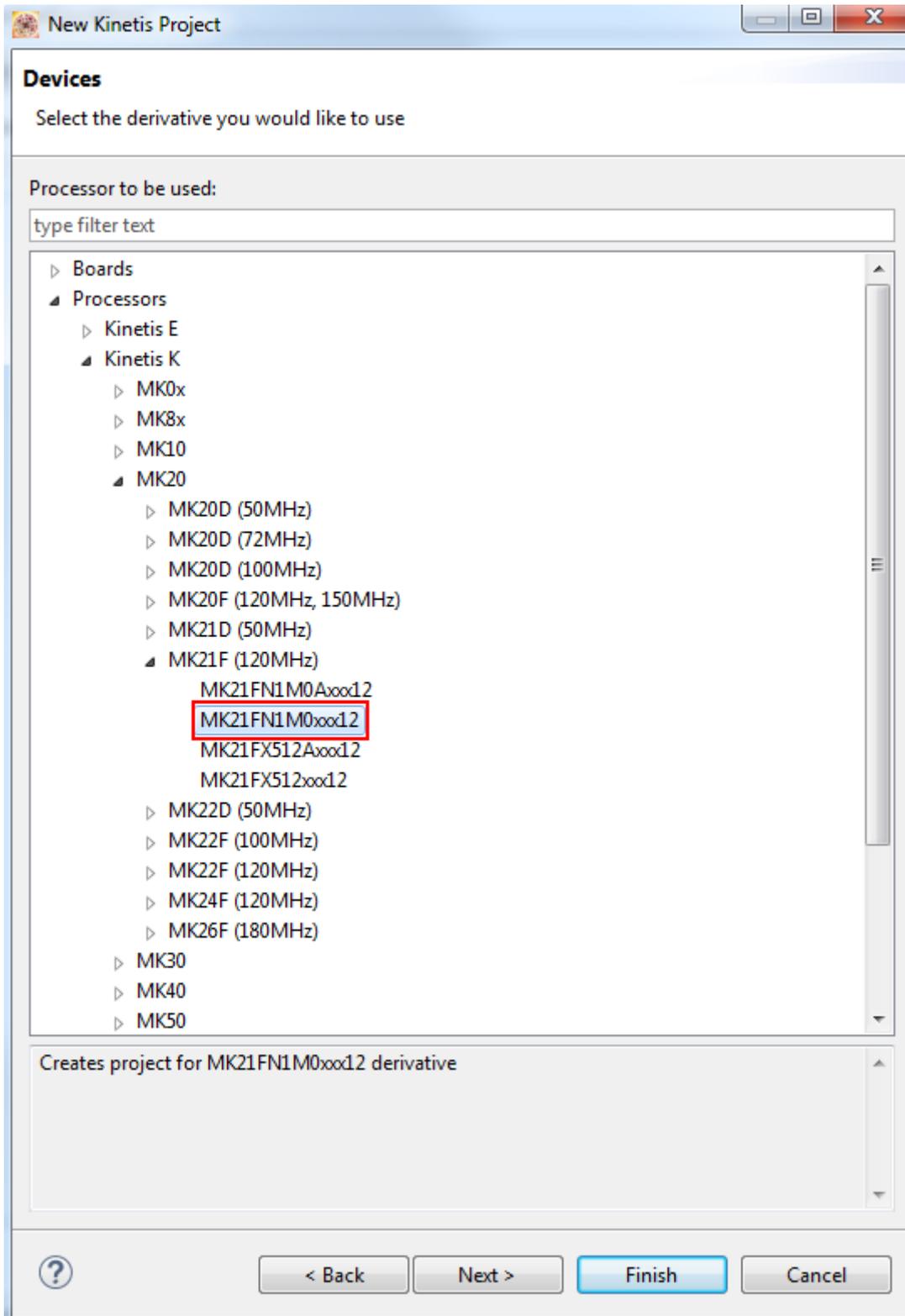
The application hardware is TWR-K21F120M board, software is KDS v3.2.0 .

1. Processor Expert Project

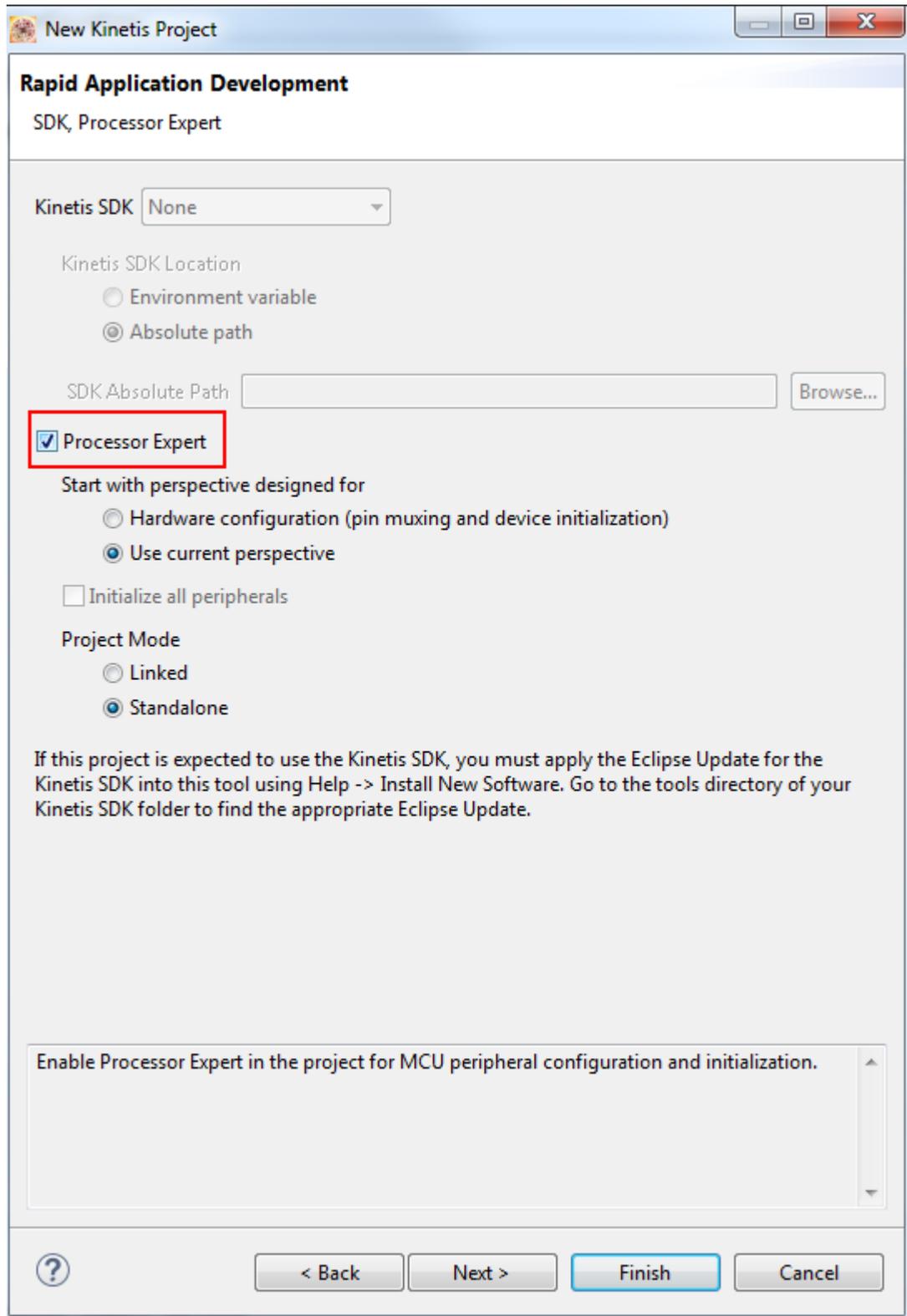
- Create a Processor Expert Project:



- Select the "MK21FN1M0xxx12" chip:



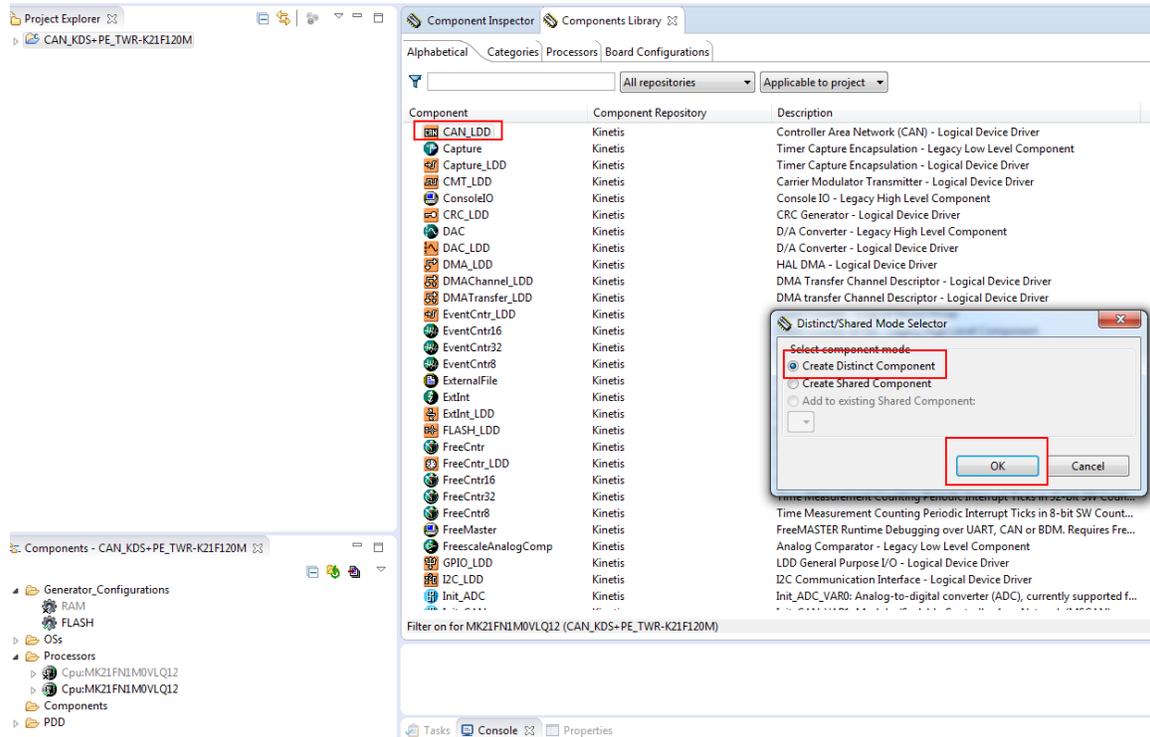
- Enable the "Processor Expert" option:



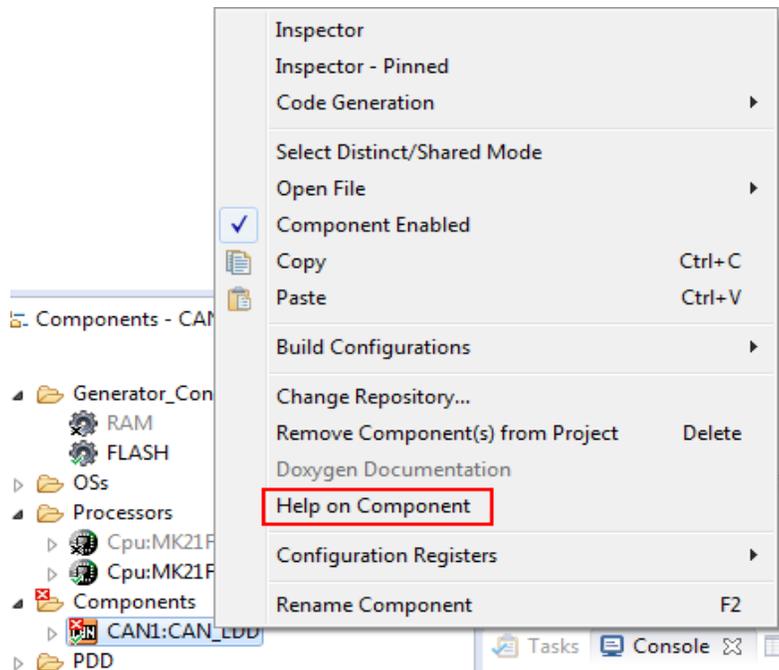
So for , one PE(Processor Expert) project is finished.

2. CAN_LDD

Double click the CAN_LDD component can add it to the project :



After add it , right click this component, then select “help on component”, we can find all of the information about this component, including the “Methods” “Events” “Typical Usage” and so on , this is an important reference when develop with the CAN_LDD.



Help Contents Search Related Topics Bookmarks Index

About Components

Component CAN_LDD

See also:

- CAN_LDD
- CAN_LDD Properties
- CAN_LDD Methods
- CAN_LDD Events
- CAN_LDD User Types
- CAN_LDD Typical Usage
- CANCalculator

More results:

Search for Components view

Especially pay attention to the “CAN_LDD Typical Usage”, there is some demo code about how to use this component .

Help Contents Search Related Topics Bookmarks Index

CAN_LDD

Component CAN_LDD

CAN communication

Component Level: Logical Device Driver
Category: Logical Device Drivers-Communication

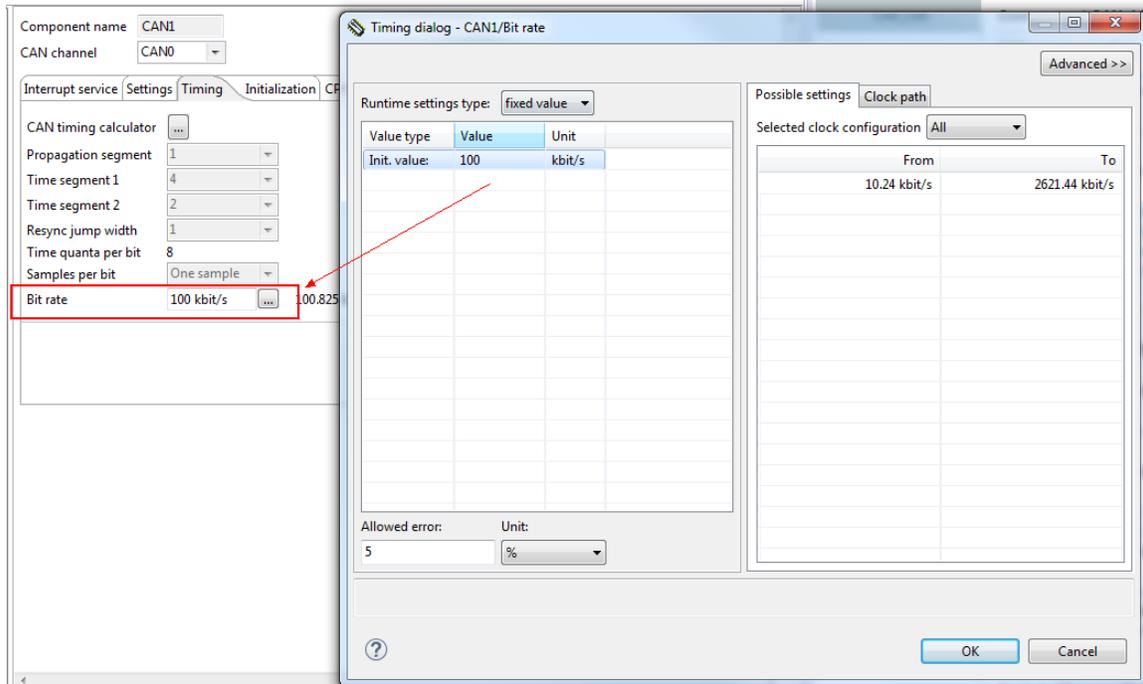
Typical Usage:
(Examples of a typical usage of the component in user code. For more information please see the page Component Code Typical Usage.)

Examples of typical settings and usage of CAN_LDD component

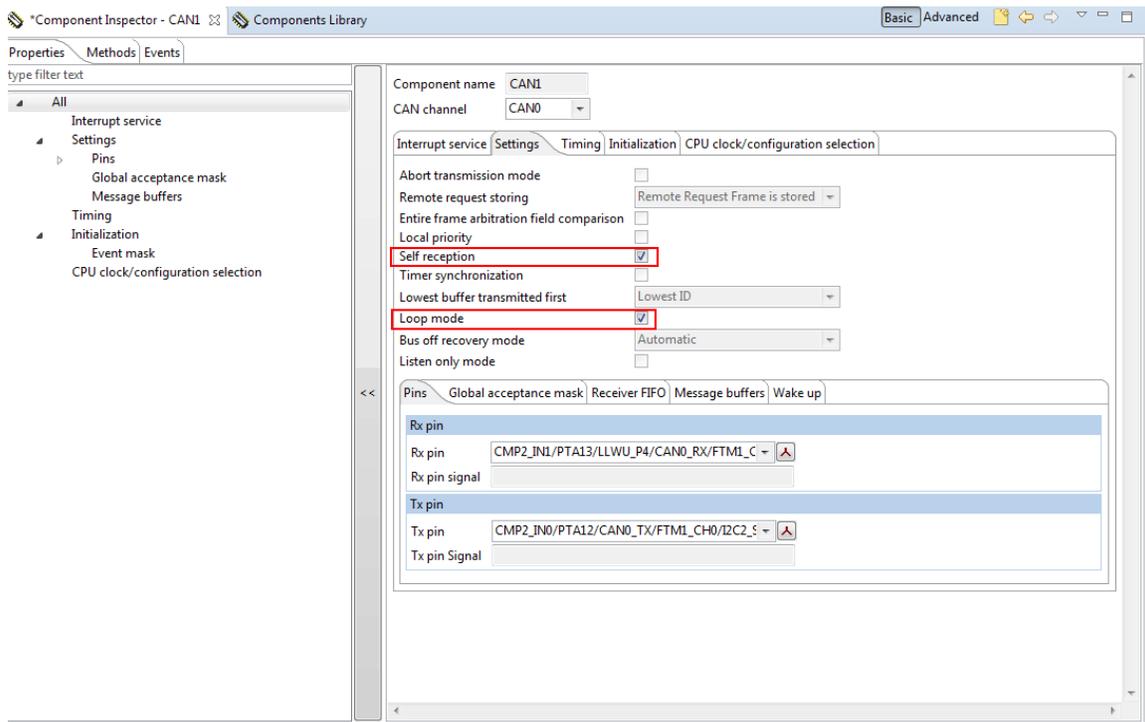
1. Sending data frame with interrupt service
2. Receiving data frame with interrupt service
3. Sending data frame without interrupt service (polling)
4. Receiving data frame without interrupt service (polling)

3. Configure the CAN_LDD and write code

- Configure the “Bit rate” to “100kbit/s”



- Enable the "Self reception" and "Loop mode":



- When write the code , we can refer to the "Typical Usage" about the "Sending data frame with interrupt service" demo and "Receiving data frame with interrupt service".

Pay attention that , in the Typical Usage code , the Transfer Message ID is 0x123u, while in the CAN_LDD default configuration , the Receive Message ID is 0x7FF, they are should be same . So we should change one of them (In standard frame format, the admissible range is from 0x00 to 0x7FF; in extended format, the range is from 0x00 to 0x1FFFFFFF), on my project , I change the TX ID from 0x123u to 0x7FFu on the code:

On the Typical Usage code :

```
volatile bool DataFrameTxFlg;
LDD_TDeviceData *MyCANPtr;
LDD_TError Error;
LDD_CAN_TFrame Frame;
uint8_t OutData[4] = {0x00U, 0x01U, 0x02U, 0x03U};          /* Initialization of output data buffer */

void main(void)
{
    . . .
    MyCANPtr = CAN2_Init(NULL);                             /* Initialization of CAN2 component */

    Frame.MessageID = 0x123U;                               /* Set Tx ID value - standard */
    Frame.FrameType = LDD_CAN_DATA_FRAME;                  /* Specyfing type of Tx frame - Data frame */
    Frame.Length = sizeof(OutData);                        /* Set number of bytes in data frame - 4B */
    Frame.Data = OutData;                                  /* Set pointer to OutData buffer */
    DataFrameTxFlg = FALSE;                                 /* Initialization of DataFrameTxFlg */
    Error = CAN2_SendFrame(MyCANPtr, 0U, &Frame);          /* Sends the data frame over buffer 0 */
    while (!DataFrameTxFlg) {                               /* Wait until data frame is transmitted */
    }
}
```

On my project :

```
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
    /* Write your local variable definition here */
    int cnt = 0;
    /** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */
    PE_low_level_init();
    /** End of Processor Expert internal initialization.          */

    /* Write your code here */
    //Send
    MyCANPtr = CAN1_Init(NULL);                             /* Initialization of CAN1 component */
    Frame_TX.MessageID = 0x7FFU;                            /* Set Tx ID value - standard */
    Frame_TX.FrameType = LDD_CAN_DATA_FRAME;                /* Specyfing type of Tx frame - Data frame */
    Frame_TX.Length = sizeof(TX_Data);                      /* Set number of bytes in data frame - 4B */
    Frame_TX.Data = TX_Data;                                /* Set pointer to OutData buffer */
    DataFrameTxFlg = FALSE;
    DataFrameRxFlg = FALSE;

    Error = CAN1_SendFrame(MyCANPtr, 1U, &Frame_TX);        /* Sends the data frame over buffer 0 */

    while (!DataFrameRxFlg) {                               /* Wait until data frame is received */
    }
    //Recieve
    Frame_RX.Data = RX_Data;
    Error = CAN1_ReadFrame(MyCANPtr, 0U, &Frame_RX);        /* Reads a data frame from buffer 0 and fills Frame structure */
    for(cnt=0;cnt<4;cnt++)
    {
        printf("0x%02x \r\n",RX_Data[cnt]);                /* print the received data */
    }
}
```

On the CAN_LDD component default configuration :

Listen only mode

Pins Global acceptance mask Receiver FIFO Message buffers Wake up

Message buffers - 2 +

#	Buffer type	Accept frames	Message ID	Individual Acceptance Mask	Acceptance Mask
0	Receive	Standard	0x7FF	<input type="checkbox"/>	0x1FFFFFFF
1	Transmit	Standard	0x7FF	Disabled	0x1FFFFFFF

Details for selected row:

Buffer0

Buffer type Receive

Accept frames Standard

Message ID 0x7FF

Individual Acceptance Mask

Acceptance Mask 0x1FFFFFFF

After generated code ,in the initialize function “CAN1_Init()” about the configuration part of receive buffer 0, we can see it set the Receive buffer ID to 0x07FFU:

```

/* Initialize the message buffer 0 - Rx */
CAN_PDD_SetMessageBufferCode(CAN0_BASE_PTR, 0U, CAN_PDD_MB_RX_NOT_ACTIVE);
CAN_PDD_EnableMessageBufferIDExt(CAN0_BASE_PTR, 0U, PDD_DISABLE); /* Extended Frame bit IDE clear*/
CAN_PDD_SetMessageBufferID(CAN0_BASE_PTR, 0U, CAN_PDD_BUFFER_ID_STD, 0x07FFU); /* Set standard buffer ID */
CAN_PDD_SetMessageBufferCode(CAN0_BASE_PTR, 0U, CAN_PDD_MB_RX_EMPTY); /* Empty Frame*/
CAN_PDD_EnableMessageBufferSRR(CAN0_BASE_PTR, 0U, PDD_DISABLE); /* SRR set to 0 */
CAN_PDD_EnableMessageBufferRTR(CAN0_BASE_PTR, 0U, PDD_DISABLE); /* RTR set to 0*/
CAN_PDD_SetMessageBufferWORD0(CAN0_BASE_PTR, 0U, 0x00U); /*Clear Data field*/
CAN_PDD_SetMessageBufferWORD1(CAN0_BASE_PTR, 0U, 0x00U); /*Clear Data field*/
CAN_PDD_SetMessageBufferTimeStamp(CAN0_BASE_PTR, 0U, 0x00U); /* Empty Frame*/

```

When debug , we can also check the Receive and Transfer ID from the register :

Arch: SVD(CMSIS) Vendor: Freescale Chip: MK21F12 Board: --- none ---

Register	Hex	Bin	Reset	Access	Address	Description
ESR2			0x00000000	RO	0x40024038	Error and Status 2 register
CRCR			0x00000000	RO	0x40024044	CRC Register
RXFGMASK			0xFFFFFFFF	RW	0x40024048	Rx FIFO Global Mask register
RXFIR			0x00000000	RO	0x4002404C	Rx FIFO Information Register
CS0			0x00000000	RW	0x40024080	Message Buffer 0 CS Register
ID0	0x1FFC0000	00011111111111000000000000000000	0x00000000	RW	0x40024084	Message Buffer 0 ID Register
EXT (bits 17-0)	0x000000	000000000000000000000000				Contains extended (LOW word) ic
STD (bits 28-18)	0x7FF	111111111111				Contains standard/extended (HIG
PRIO (bits 31-29)	0x0	000				Local priority. This 3-bit fields on
WORD00	0x00010203	0000000000000000010000001000000011	0x00000000	RW	0x40024088	Message Buffer 0 WORD0 Registe
DATA_BYTE_3 (bits 7-0)	0x03	00000011				Data byte 3 of Rx/Tx frame.
DATA_BYTE_2 (bits 15-8)	0x02	00000010				Data byte 2 of Rx/Tx frame.
DATA_BYTE_1 (bits 23-16)	0x01	00000001				Data byte 1 of Rx/Tx frame.
DATA_BYTE_0 (bits 31-24)	0x00	00000000				Data byte 0 of Rx/Tx frame.
WORD10			0x00000000	RW	0x4002408C	Message Buffer 0 WORD1 Register
CS1			0x00000000	RW	0x40024090	Message Buffer 1 CS Register
ID1	0x1FFF038	0001111111111111011000000111000	0x00000000	RW	0x40024094	Message Buffer 1 ID Register
EXT (bits 17-0)	0x3B038	111011000000111000				Contains extended (LOW word) ic
STD (bits 28-18)	0x7FF	111111111111				Contains standard/extended (HIG
PRIO (bits 31-29)	0x0	000				Local priority. This 3-bit fields on
WORD01	0x00010203	0000000000000000010000001000000011	0x00000000	RW	0x40024098	Message Buffer 1 WORD0 Register
DATA_BYTE_3 (bits 7-0)	0x03	00000011				Data byte 3 of Rx/Tx frame.
DATA_BYTE_2 (bits 15-8)	0x02	00000010				Data byte 2 of Rx/Tx frame.
DATA_BYTE_1 (bits 23-16)	0x01	00000001				Data byte 1 of Rx/Tx frame.
DATA_BYTE_0 (bits 31-24)	0x00	00000000				Data byte 0 of Rx/Tx frame.
WORD11	0x00000000	00000000000000000000000000000000	0x00000000	RW	0x4002409C	Message Buffer 1 WORD1 Register
CS2			0x00000000	RW	0x400240A0	Message Buffer 2 CS Register
ID2			0x00000000	RW	0x400240A4	Message Buffer 2 ID Register

- At last add the component of "ConsoleIO" to print the data that CAN module received:
 Configure the UART module refer to your board, for the TWR-K21F120M, I use the UART5.

Component name: IO1

Device: UART5

Interrupt service/event: Settings Initialization CPU clock/configuration selection

Data width: 8 bits

Parity: None

Stop bits: 1

Loop mode: Normal

Baud rate: 115200 baud (115228.132 baud)

Wakeup condition: Idle line wakeup

Stop in wait mode:

Idle line mode: Starts after start bit

Transmitter output: Not inverted

Receiver input: Not inverted

Break generation length: 10/11 bits

Receiver

RxD: PTD8/I2C0_SCL/UART5_RX/FB_A16

RxD pin signal:

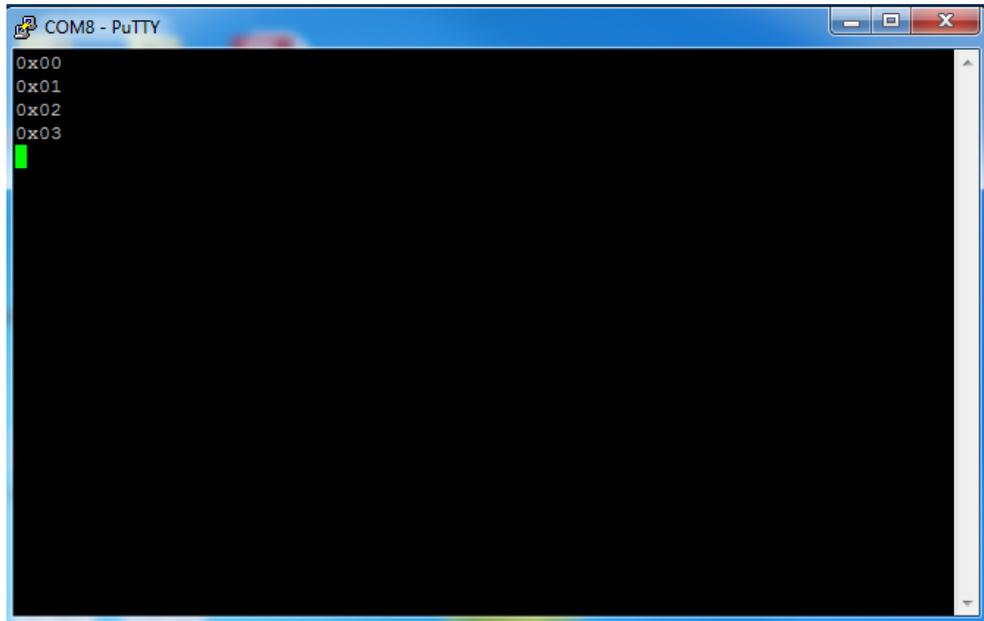
Transmitter

TxD: PTD9/I2C0_SDA/UART5_TX/FB_A17

TxD pin signal:

Flow control: None

- When run the project , we can see the result on the Termianl.



A screenshot of a PuTTY terminal window titled "COM8 - PuTTY". The terminal displays four lines of memory addresses: 0x00, 0x01, 0x02, and 0x03. A green cursor is positioned at the beginning of the line containing 0x03. The terminal background is black, and the text is white. The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

Reference:

- (1) K21 Sub-Family Reference Manual
- (2) TWR-K21F120M_SCH