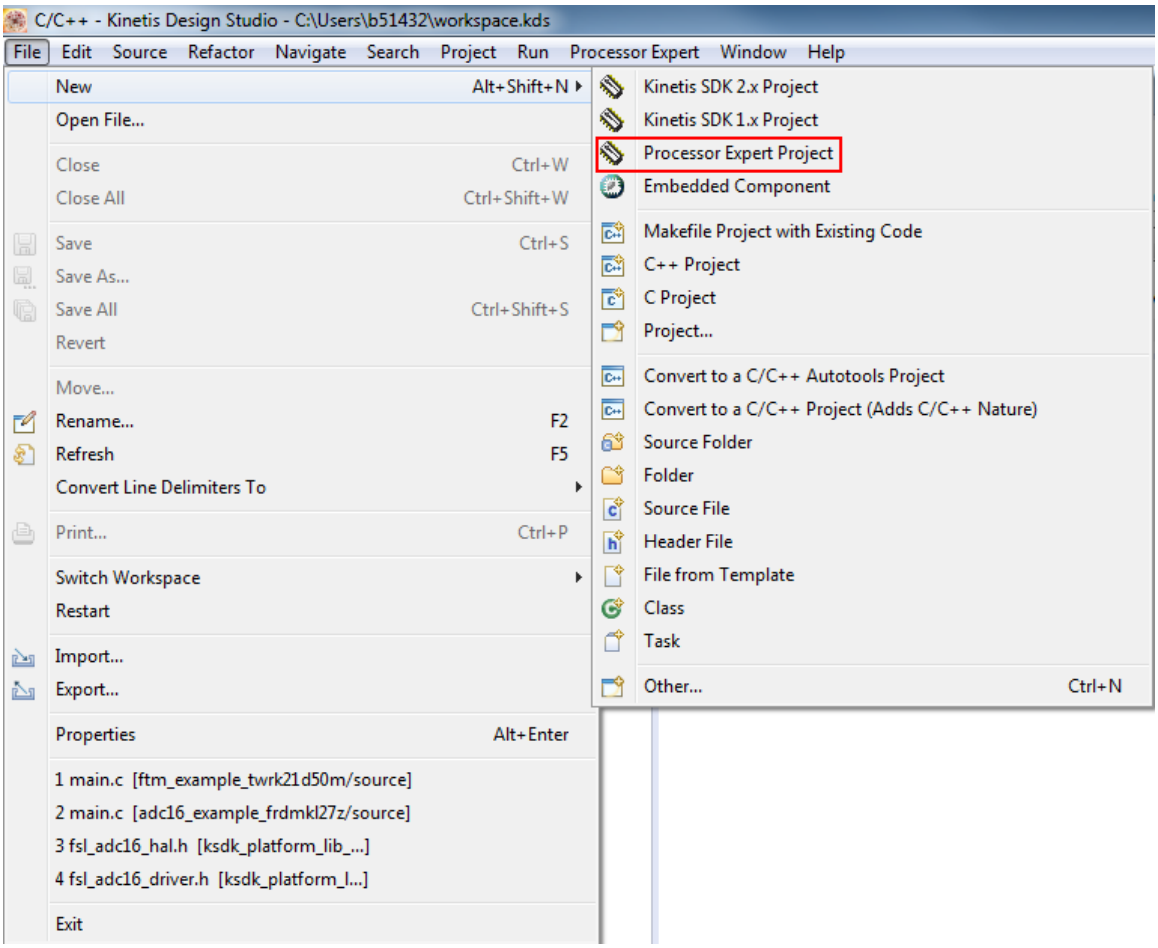
How to Start CAN Module Development on KDS v3.2.0 + Processor Expert

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".

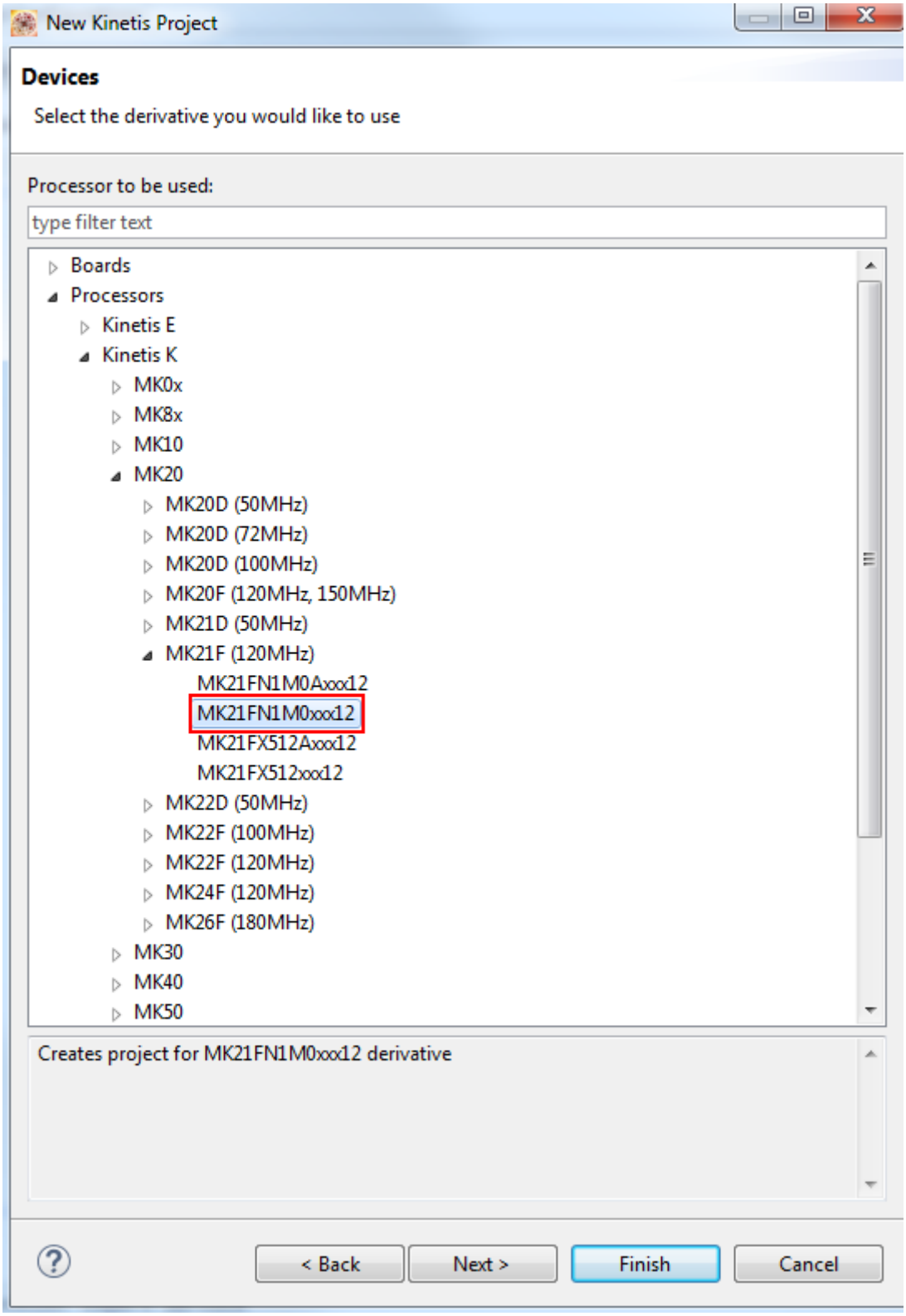
The application hardware is TWR-K21F120M board, software is KDS v3.2.0 .

1. Processor Expert Project

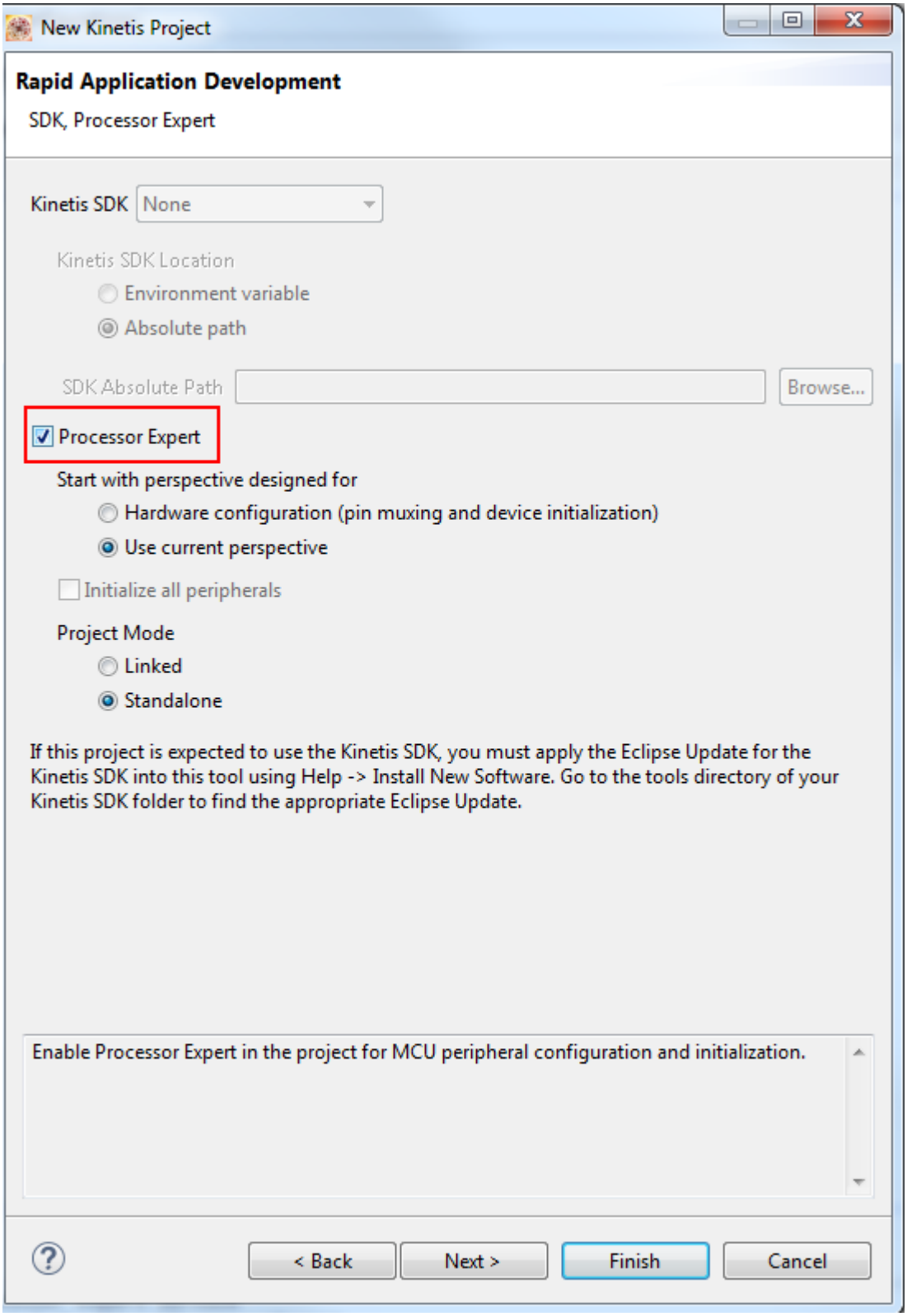
- Create a Processor Expert Project:



- Select the “MK21FN1M0xxx12” chip:



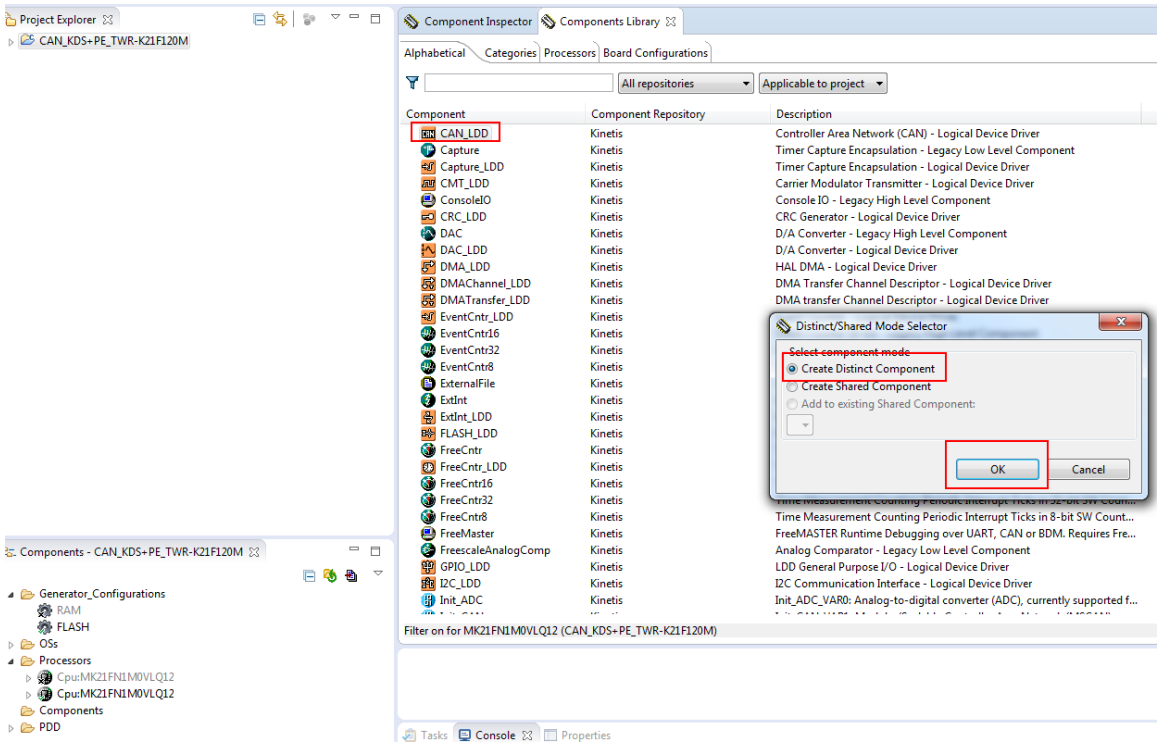
- Enable the “Processor Expert” option:



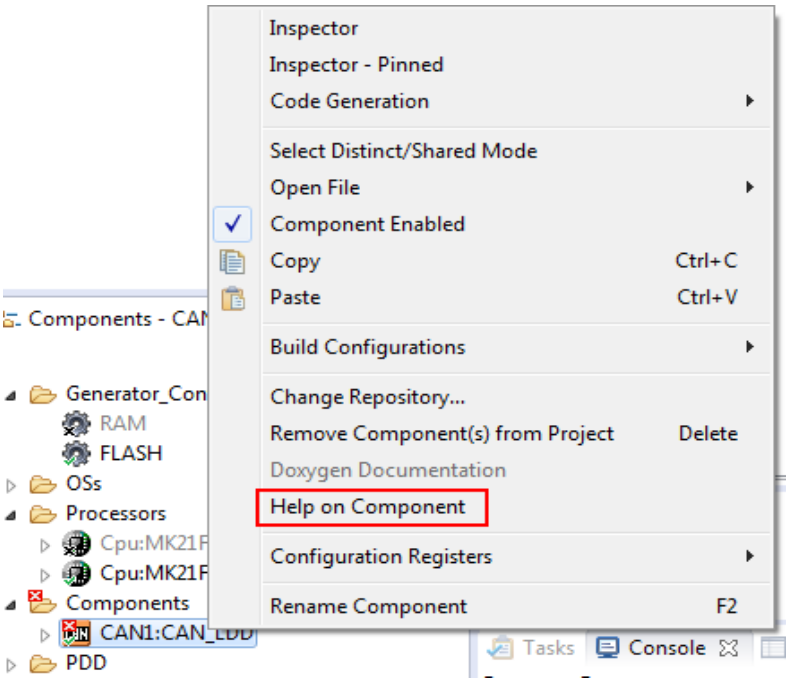
So for , one PE(Processor Expert) project is finished.

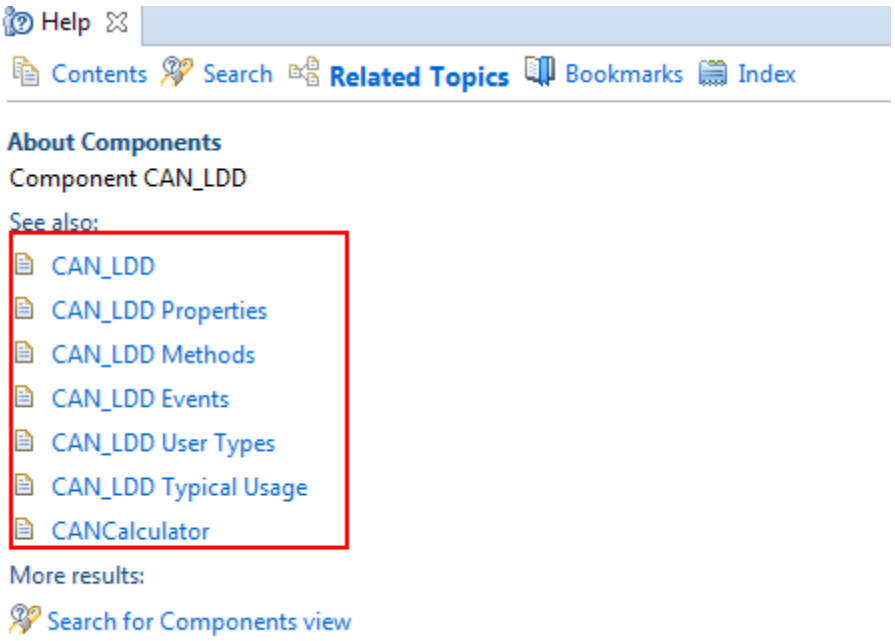
2. CAN_LDD

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

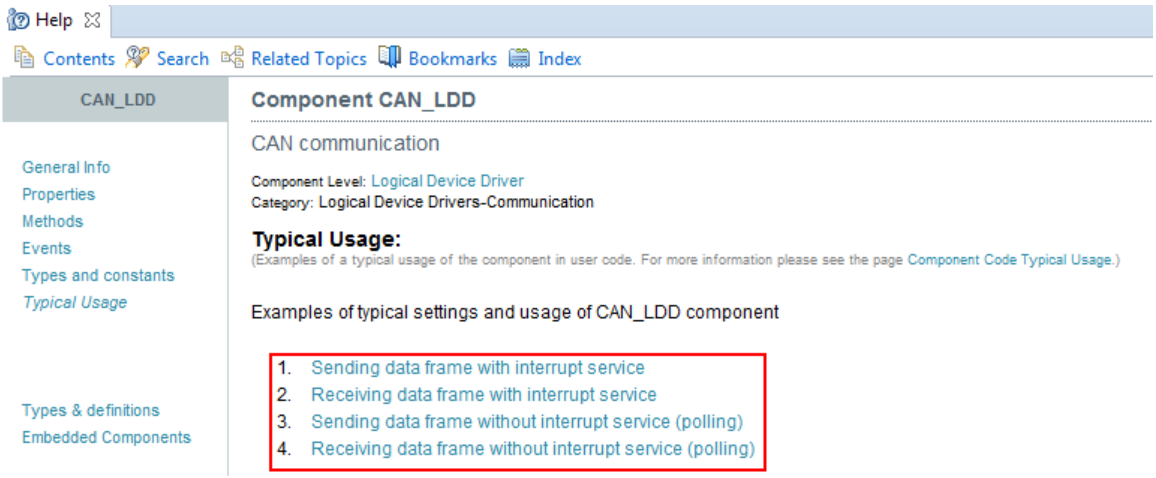


After add it , right click this component, then select “help on component”, we can find all of the information about this component, including the “Methods” “Events” “Typical Usage” and so on , this is an important reference when develop with the CAN_LDD.



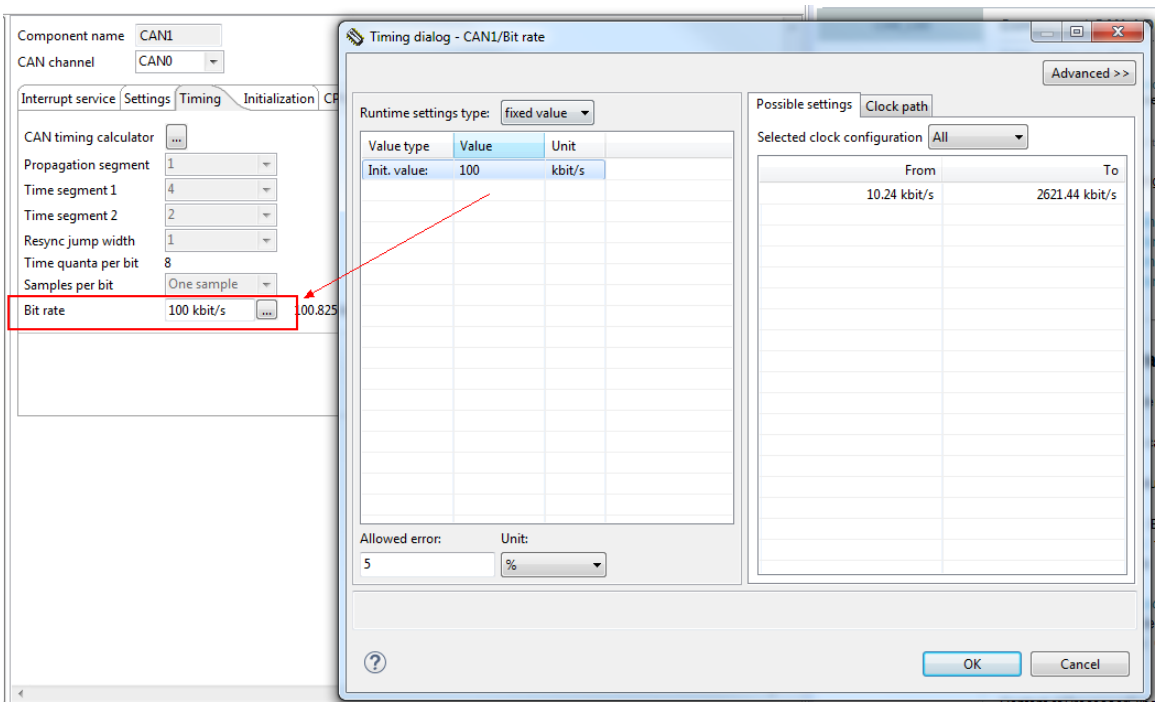


Especially pay attention to the “CAN_LDD Typical Usage”, there is some demo code about how to use this component .

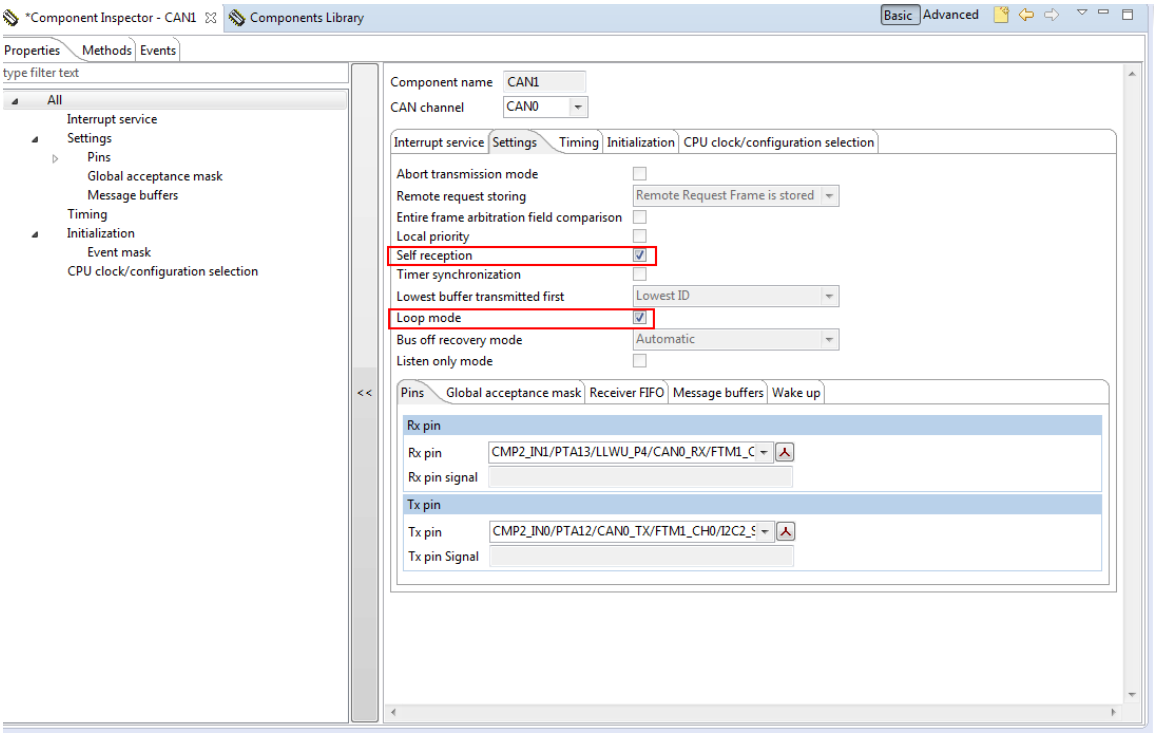


3. **Configure the CAN_LDD and write code**

- 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 “Recieving 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;                         /* Specyfying 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 :

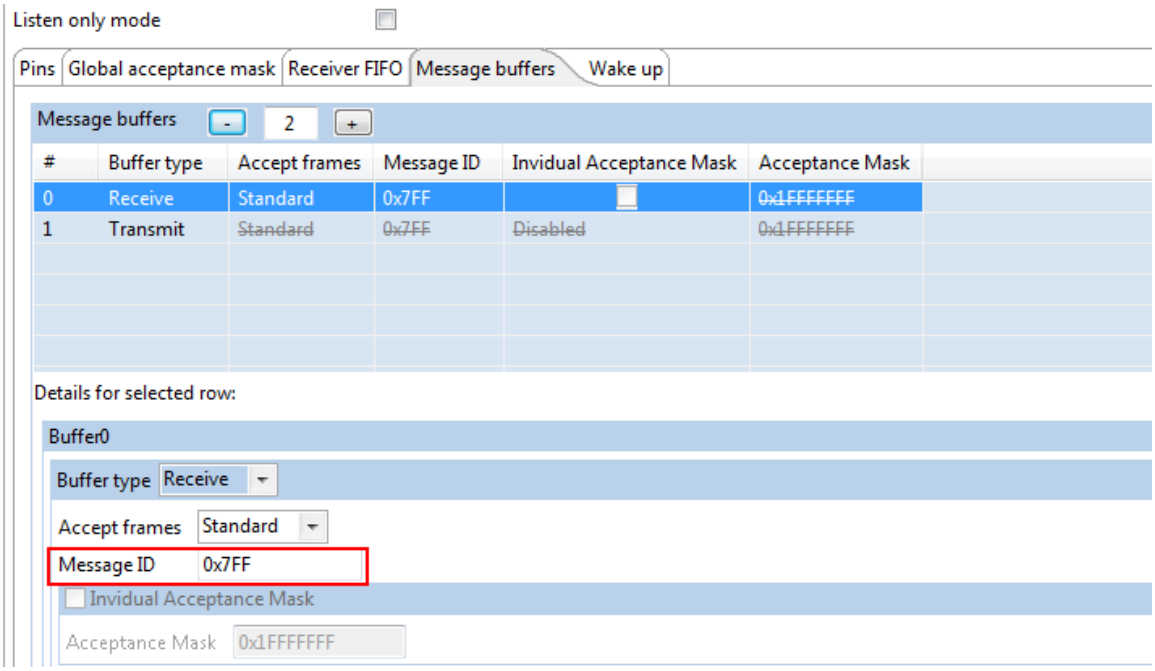
```
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
  int cnt = 0;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.   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

  /* 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;                      /* Specyfying 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  */
  }
```

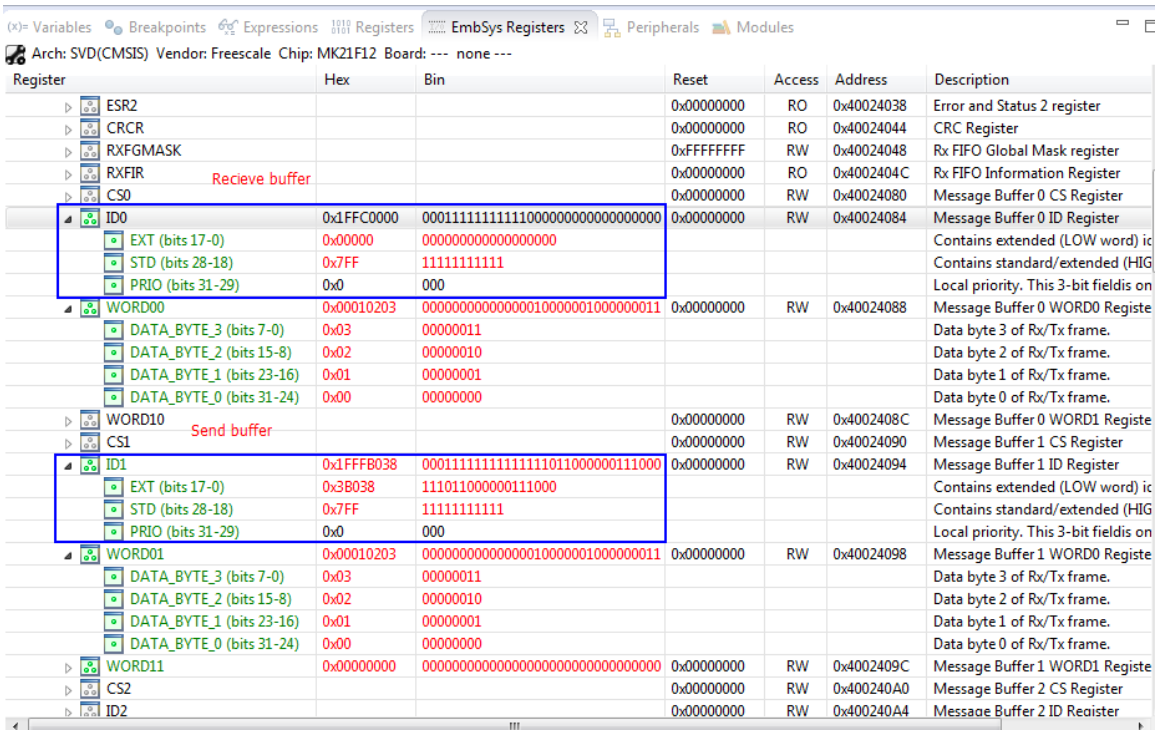
On the CAN_LDD component default configuration :



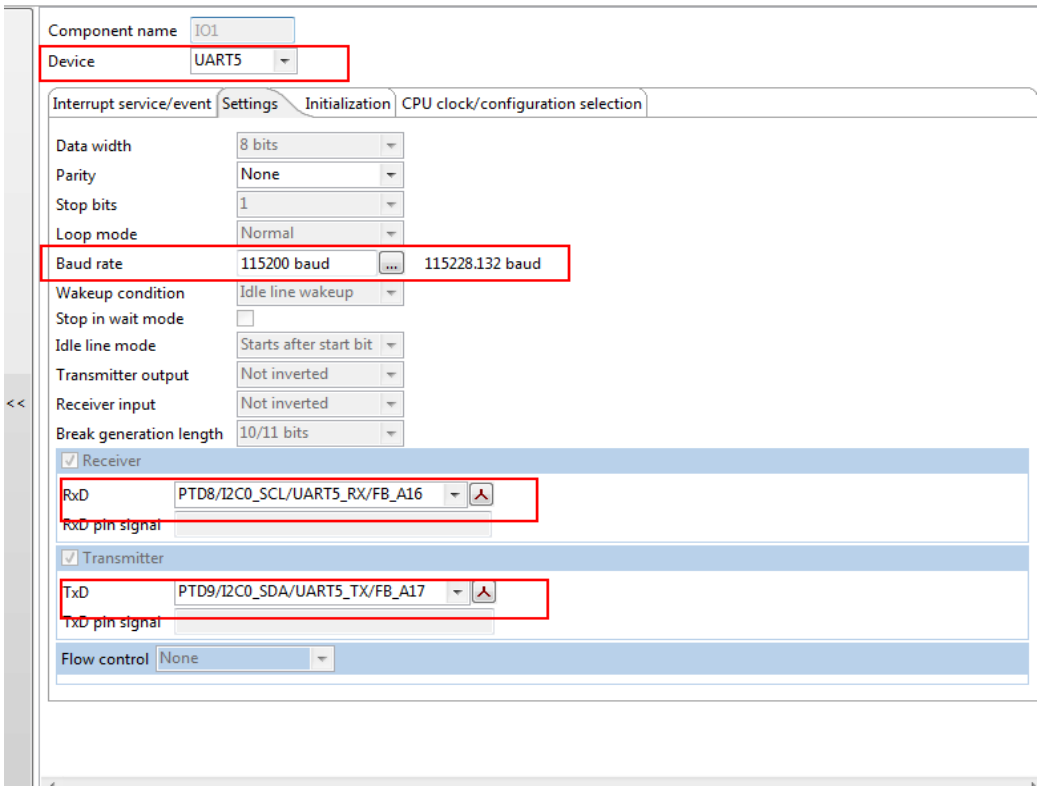
After generated code ,in the initialize function “CAN1_Init()” about the configuration part of receive buffer 0, we can see it set the Receive buffer ID to 0x07FFU:

```
/* 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*/
```

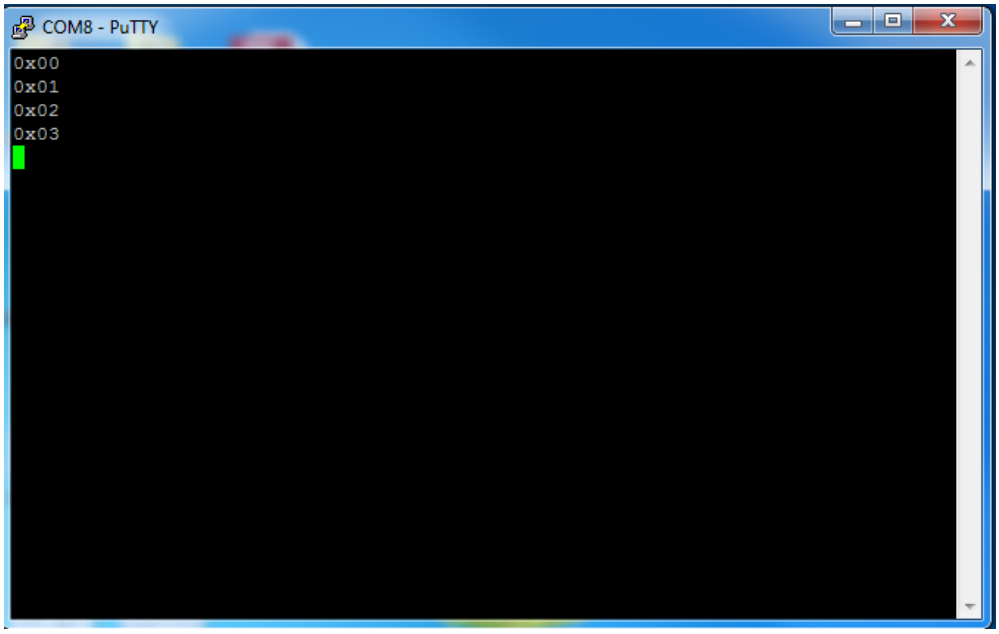
When debug , we can also check the Receive and Transfer ID from the register :



- 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.



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



Reference:

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