KEA128 CAN Bootloader

1. Introduction

In most cases, there is no debug interface for user to update the application or fix bugs , so user can use Bootloader. Bootloader is a small program put into a device that allows user using communication interfaces to upgrade application , such as use UART , I2C , USB, CAN and so on.

This application describe a bootloader based on the TRK-KEA128 board with CAN interface. The bootlaoder and user application code are written in separate projects , user can program the bootloader with tools such as J-Link ,Multilink, then boot the application file through CAN. The bootloader and application code are all provided , user can porting it to their own board of KEA/KE chips.

2. Bootloader Frame Protocol

This bootloader uses the XMODEM protocol , XMODEM is a simple file transfer protocol , it allowed user to transmit files between their devices that both sides used XMODEM . (About the detail introduce of XMODEM , you can check here : https://en.wikipedia.org/wiki/XMODEM)

2.1 Start Transfer

The transfer starts from the receiver(on this project , the receiver is TRK-KEA128 board) sending ‘C ’ to transmitter(the transmitter is PC) . Refer to the XMODEM protocol , sending ‘C’ means the receiver wants to use CRC to check . Pay attention that, the receiver(TRK-KEA128 board) need to keep sending ‘C’ , not only send once. As to when PC side receives ‘C’, if it not send APP(.bin file) to board at once, the communication immediately disconnect . When PC send APP to board, board will not send ‘C’ automatic.

The following is the function sending ‘C’ on the booltoader project :

```
while(1)
{
        xmodem_putchar('C');

        if(xmodem_is_active())
        {
          xmodem_download();
         break;
        }
}
```

If the PC receives the characters ‘C’, it meaning the board and PC successful communication, also tell PC using CRC check .

The following Figure 1 is the Tera Term view when board sending ‘C’ :

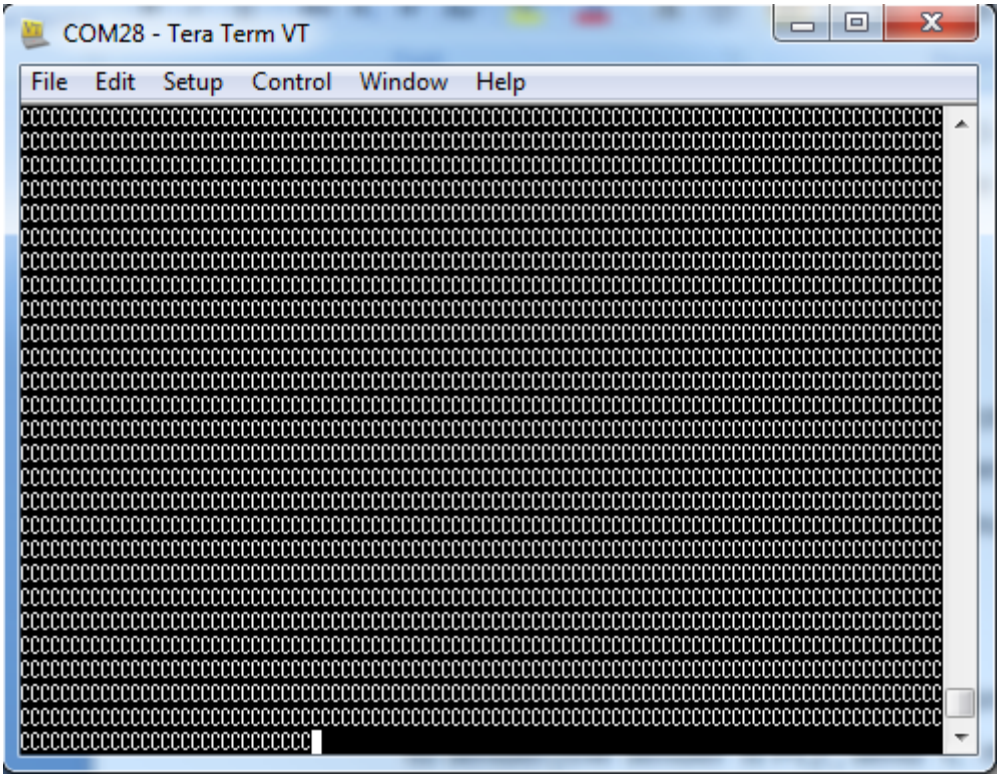

Figure 1. Tera Term view when connected with board

2.2 Transfer and End

The XMODEM used a 133-byte data packet, The packet was prefixed by a simple 3-byte header, containing a <SOH> character, a "block number" from 0-255, and the "inverse" block number—255 minus the block number. The packet was also suffixed with 2-byte checksum of the data bytes.

Table 1. Frame Package Format

Byte1	Byte2	Byte3	Byte4-131	Byte132-133
Start Of Header	Packet Number	~(Packet Number)	Packet Data	16-Bit CRC

About the start header character Terminology :

Table 2. The Meaning of Start Header Characters

Name	Decimal	Hexadecimal	Description
SOH	01	H001	Start Of Header
EOT	04	H004	End Of Transmission
ACK	06	H006	Acknowledge (positive)
DLE	16	H010	Data Link Escape
NAK	21	H015	Negative Acknowledge
CAN	24	H018	Characters Abort Now

Files were transferred one packet at a time. When received, the packet's checksum was calculated by the receiver and compared to the one received from the sender at the end of the packet. If the two matched, the receiver sent an <ACK>message back to the sender, which then sent the next packet in sequence. If there was a problem with the checksum, the receiver instead sent a <NAK>. If a <NAK> was received, the sender would re-send the packet, and continued to try several times, normally ten, before aborting the transfer.

A <NAK> was also sent if the receiver did not receive a valid packet within ten seconds while still expecting data due to the lack of a <EOT> character. A seven-second timeout was also used within a packet, guarding against dropped connections in mid-packet.

The block numbers were also examined in a simple way to check for errors. After receiving a packet successfully, the next packet should have a one-higher number. If it instead received the same block number this was not considered serious, it was implied that the <ACK> had not been received by the sender, which had then re-sent the packet.

When the sender complete transfer data , it will send a block with the <EOT> . After the receiver get it , it will send <ACK> to confirm. For some time it was suggested that sending a <CAN> character instead of an <ACK> or <NAK> should be supported in order to easily abort the transfer from the receiving end. Likewise, a <CAN> received in place of the <SOH> indicated the sender wished to cancel the transfer.

In the bootloader project , using C language implement this protocol as below :

```
while(done == 0)
 {
   read_bytes(&ch,1);
   switch(ch)
    {
      case SOH:
          done = read_packet(buffer, idx);
          if (done == 0)
```

```
        {
            idx++;
            size += PKTLEN_128;
            xmodem_write_image((uint8_t*)buffer, PKTLEN_128);
            xmodem_putchar(ACK);
        }
        else
        {
            done = 0;
            xmodem_putchar(NAK);
        }
        break;
    case EOT:
        xmodem_putchar(ACK);
        done = size;
        break;
    case CAN:
        done = -1;
        break;
    default:
        xmodem_putchar(NAK);
        break;
    }
}
```

3. Memory relocation and code implementation

Figure 2 is the memory map of KEAZ128 chip, the left side of the figure is the default memory map , and the right side of the figure is relocated flash memory and vectors. This chip has 128k flash, from 0x00000000 to 0x0001FFFF. The memory from 0x00000000 to 0x000000C0 is the original vectors area, from 0x00000400 to 0x00004FFF is the bootloader area. The memory from 0x00004000 to 0x0001FFFF is for the user application code .

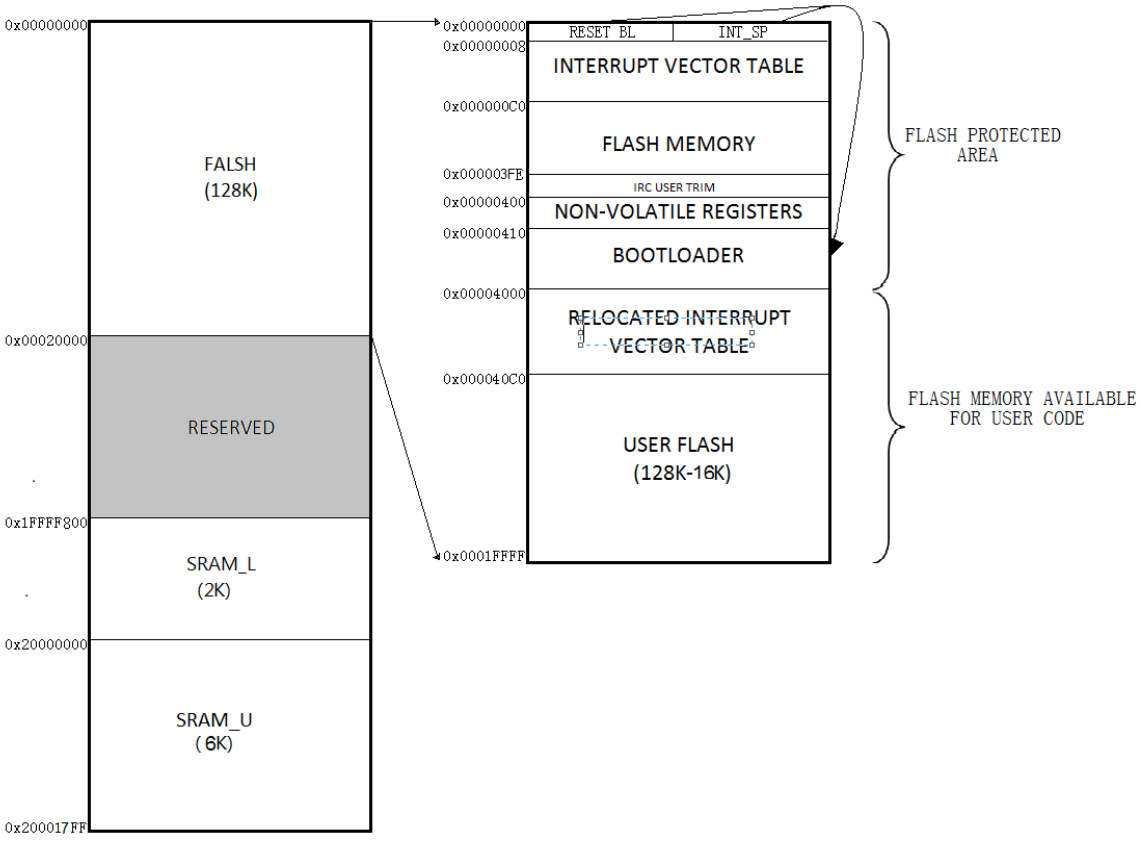

Figure 2. Memory allocation

When mcu reset , it will run to the reset vector 0x00000000. Then check the work mode, whether it in boot mode or user code mode. In this project , by checking the level of an external GPIO, if the GPIO pin is high , it will enter into boot mode to run bootloader , after update user code , then run to application. And if it is low, it will directly enter into user application code .

In the bootloader project , using C language implement this process as below :

```
if(is_enter_menu_triggered())   // if PTH4 is high
{
    printf("enter bootlaoder mode...\r\n");
    download_app();
 }
 // if PTH4 is low , it will directly run to APP
appStack = *(uint32_t*)APP_IMAGE_START;
appEntry = *(uint32_t*)(APP_IMAGE_START + 4);

JumpToUserApplication(*((unsigned long*) APP_IMAGE_START), *((unsigned
long*)( APP_IMAGE_START+4)));
```

4. User application code

In order to use this bootloader update application code , we need revise the application linker file for vector relocation :

```
MEMORY {
  m_interrupts (RX)   : ORIGIN = 0x00004000, LENGTH = 0x000000C0
  m_text       (RX)   : ORIGIN = 0x00004410, LENGTH = 0x0001FBF0-0x4000
  m_data       (RW)   : ORIGIN = 0x1FFFE000, LENGTH = 0x00004000
  m_cfmprotrom  (RX)  : ORIGIN = 0x00004400, LENGTH = 0x00000010
}
```

5. Run the demo

5.1 Hardware

For the PC has no CAN interface , so a convert board is required to transfer the UART signal into the CAN signal(also can transfer CAN signal into UART signal) , it is cheap and convenient . Therefore, the PC connected with convert board, the convert board connected with target board. Here the convert board is FRDM-KE06 board , the target board is TRK-KEA128 board , the physical connection diagram as Figure3.

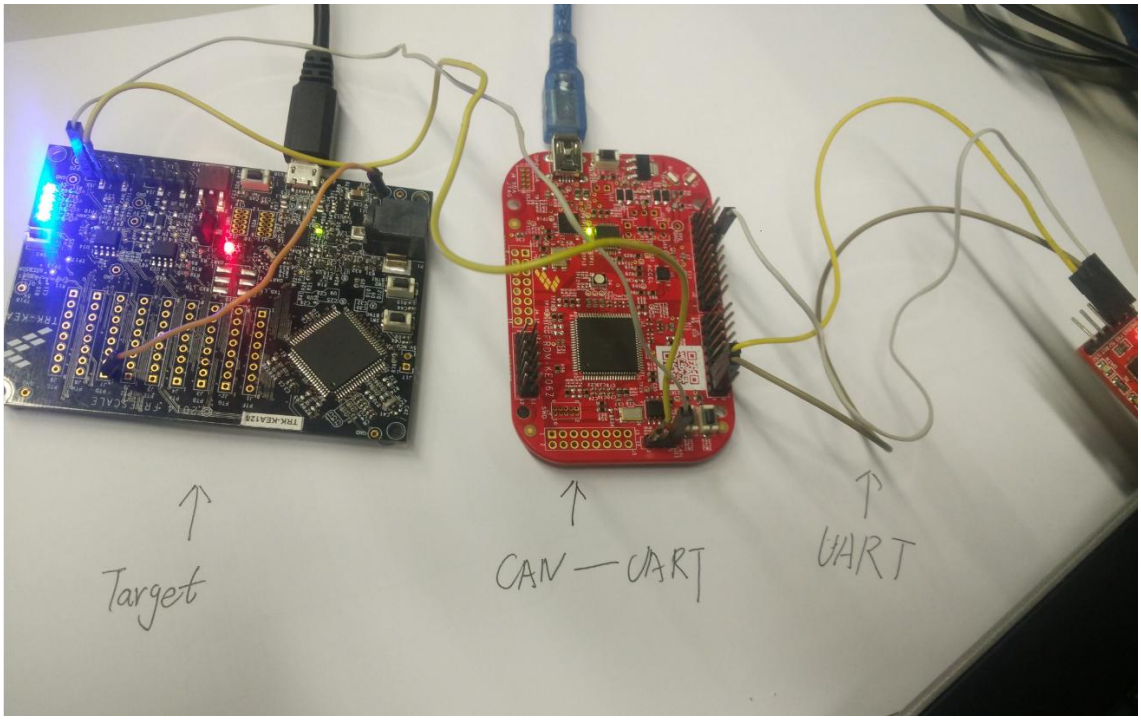

Figure 3. Hardware connection

1). Connect the UART of FRDM-KE06 to PC, here I use the UART1, PTC7(J1- pin1 on board) as TX , PTC6(J1-pin 3 on board) as RX. The UART-CAN adapter board requires some reworks because the on-board K20 OpenSDA debugger cannot meet the boot loader transmit speed requirements. One external USB-UART board is required in order to receive data from the PC. To change the default UART port connected on an external UART-USB board, cut out the R53 and R52 and connect R78 and R79.

2). Connect the CAN port between TRK-KEA128 and FRDM-KE06 refer to Table 3.

Table 3. CAN Configuration

TRK-KEA128 Board		Connects To	FRDM-KE06 Board	
Pin Name	Board Location		Pin Name	Board Location
CANH	J15-pin1	->	CAN_H	J11-pin1
CANL	J15-pin2	->	CAN_L	J11-pin3

3). Connect J7-pin1(PTD0 pin)to High, when rest, this will put the bootloader to boot mode .

5.2 Software

There are three projects , the project named ”Bootloader_KEA-128” is the bootloader program , the project named “UART-CAN” is the convert program, the “APP” folder includes the application program(“LED-D3-demo” and “LED-D8-demo”) and the .bin files. These projects all be developed on KDS(Kinetis Design Studio)v3.2.

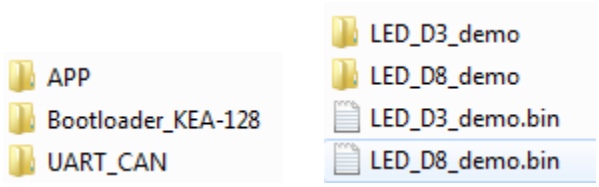

Figure 4. Projects

1). Use KDS open the project “Bootloader_KEA-128”, build ,then download it to the TRK-KEA128 board.

2). Also use KDS download the project of “UART-CAN” to the convert board of “FRDM-KE06”.

3). Open Tera Term, create a new section, and select the UART-CAN COM port open as: *115200-8- none-1-none*. You can refer to the Figure 5, Figure 6, Figure 7.

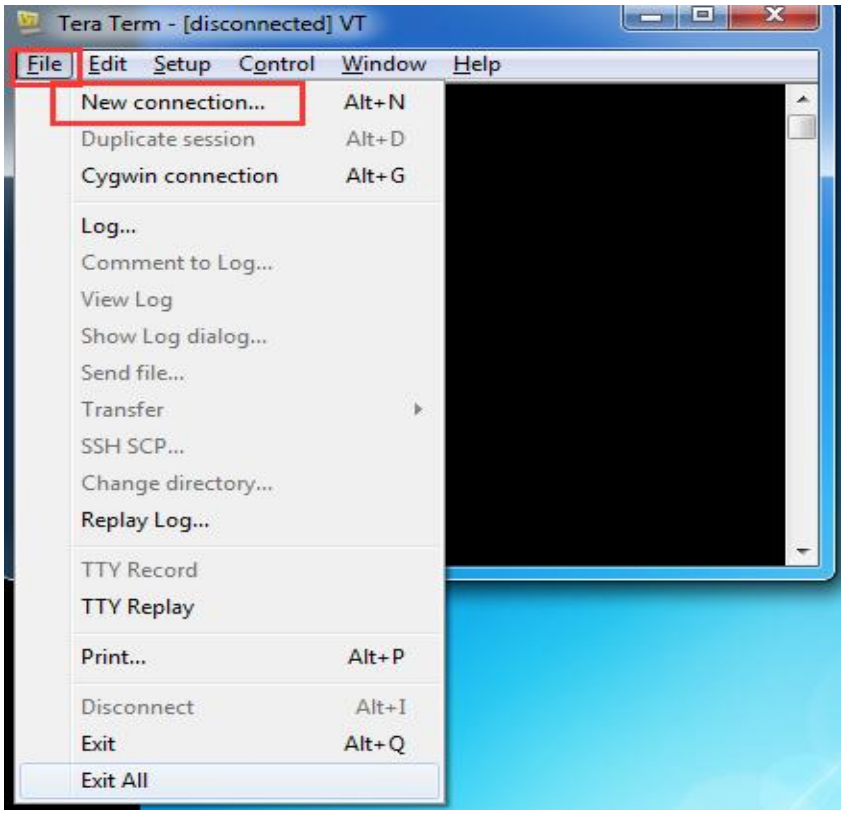

Figure 5. Configure Tera Term-1

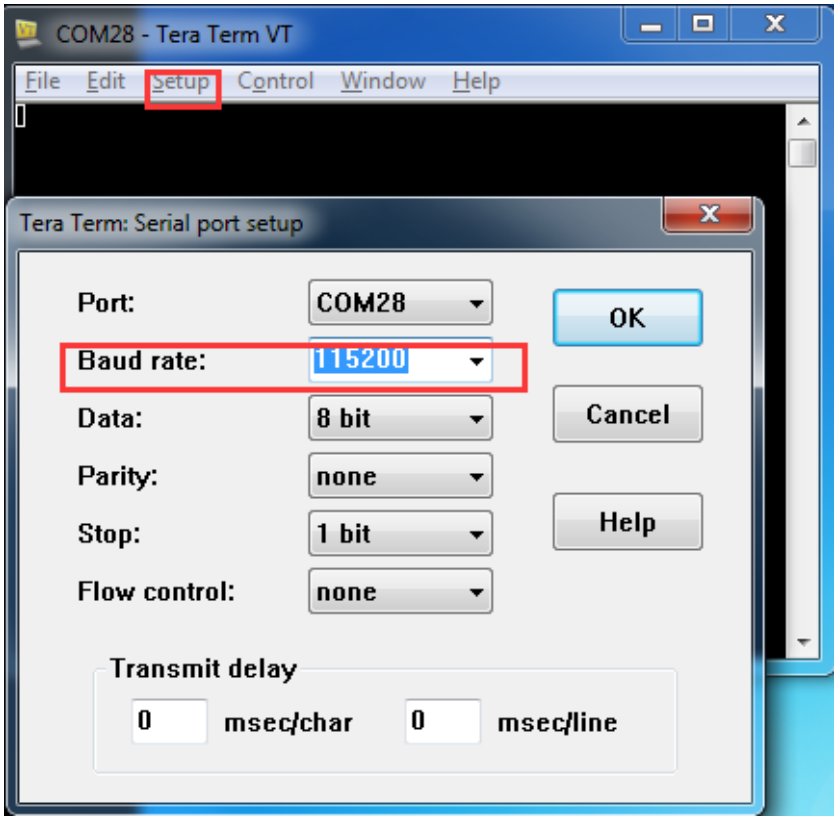

Figure 6. Configure Tera Term-2

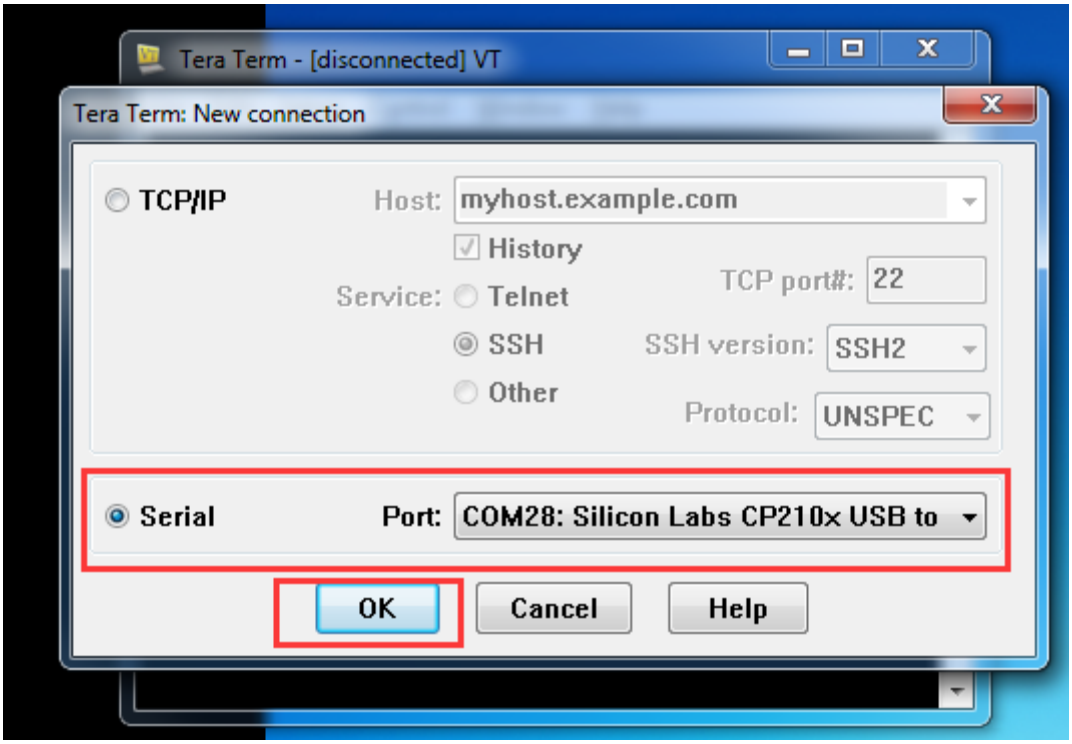

Figure 7. Configure Tera Term-3

4). Press the reset button of TRK-KEA128 board .

5). Select File->Transfer->XMODEM->send, select “LED_D3_demo.bin” to download the application image.

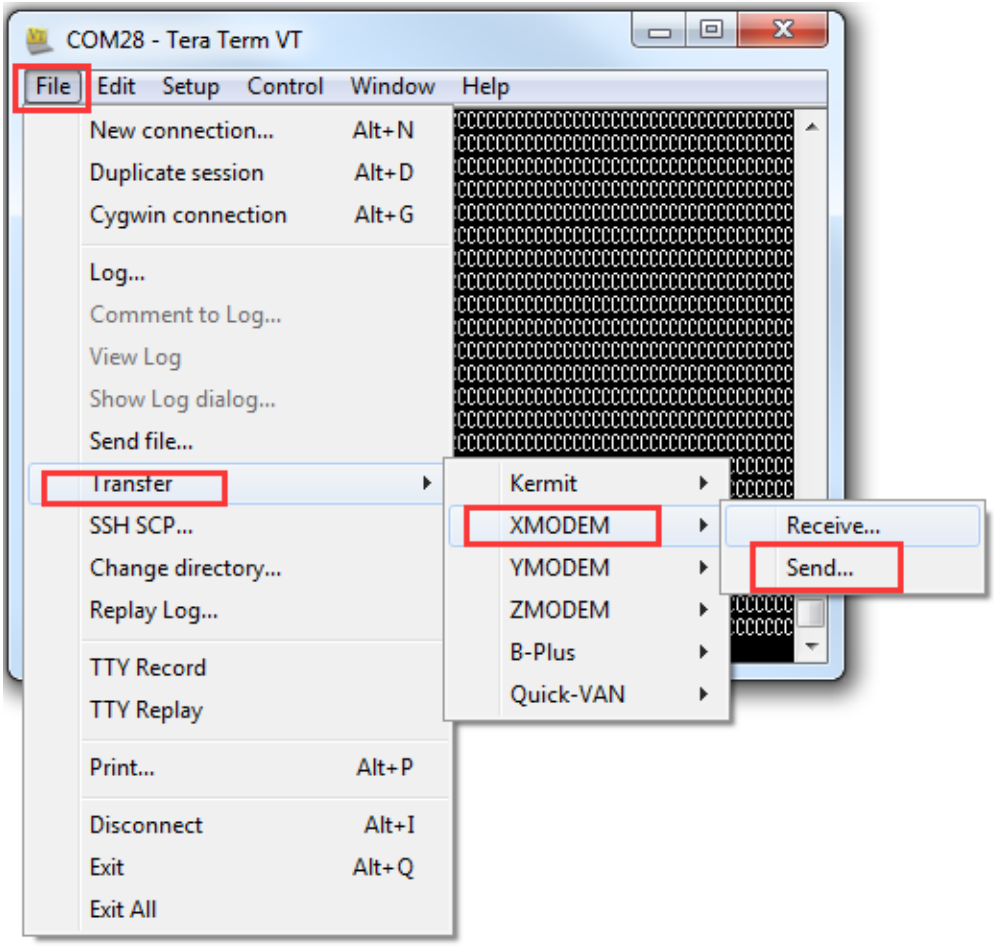

Figure 8. Download Application Image-1

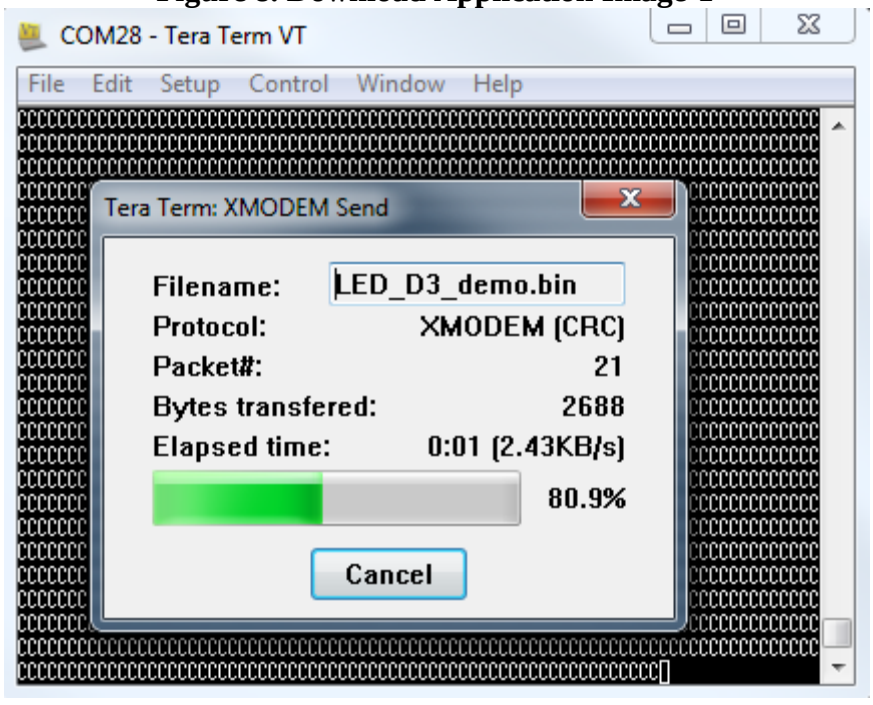

Figure 9. Download Application Image-2

6). After image downloading is completed, the bootlaoder will jump to the application code, the D3 LED will be blinking. Then you can also download the “LED-D8_demo.bin” image, if the D8 LED be blinking , it meaning the application code update successfully.

6. Reference

1) KE06 CAN Bootloader Design (document AN5219)
2) KEA128 Sub-Family Reference Manual
3) Kinetis Bootloader to Update Multiple Devices in a Field Bus Network(decument AN5204)