MC56F83000 development guide

The MC56F83000 is the latest DSC family, which has new features the previous family does not have, I list them here so that customer can know the features before they develop application project.

1) it no longer supports USBTAP device, so we suggest customer use third party device for example Multilink Universal from PEMicro.
   http://www.pemicro.com/

2) The MC56F83xxx has internal ROM bootloader, while the MC56F84xxx and MC56F82xxx

do not support ROM bootloader. For the MC56F83xxx, because it has on-chip ROM bootloader, it supports blhost tools, user can download code to flash with blhost tools in bootloader mode

3) Because there is ROM bootloader for MC56F83xxx, so a question arise, how does the MC56F83xxx enter bootloader mode or execute code in flash directly.

There are two structures in "Flash_config.c" to configure flash and ROM.

(a). You can change below values to configure Flash, such as secure or not, backdoor key, boot from ROM or not, etc.

```
static const uint8_t _flash_config_field[] = {

        /* NV_BACKKEY3: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY2: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY1: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY0: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY7: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY6: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY5: KEY=0xFF */ \
        0xFFU, \
        /* NV_BACKKEY4: KEY=0xFF */ \
        0xFFU, \
        /* NV_FPROT3: PROT=0xFF */ \
        0xFFU, \
        /* NV_FPROT2: PROT=0xFF */ \
        0xFFU, \
        /* NV_FPROT1: PROT=0xFF */ \
        0xFFU, \
        /* NV_FPROT0: PROT=0xFF */ \
        0xFFU, \
        /* NV_FSEC: KEYEN=1,MEEN=3,FSLACC=3,SEC=2 (Note: set SEC to
00b for secure) */ \
        0x7EU, \
        /* NV_FOPT: FOPT[7:6] = 11b means boot from ROM, other value
means boot from Flash */ \
        0x3FU, \
        /* Reserved */ \
        0xFFU, \
        /* Reserved */ \
        0xFFU
```

```
};
```

For the above red 0x3F data, it's bit7 and bit6 are zero, it means the MC56F83xxx starts from Flash. If the data is changed to 0xFF, it means the MC56F83xxx starts from bootloader after Reset.

(b)There is on-chip bootloader, another question arises, after Reset, the MC56F83xxx enter bootloader, which port does it read application code with?

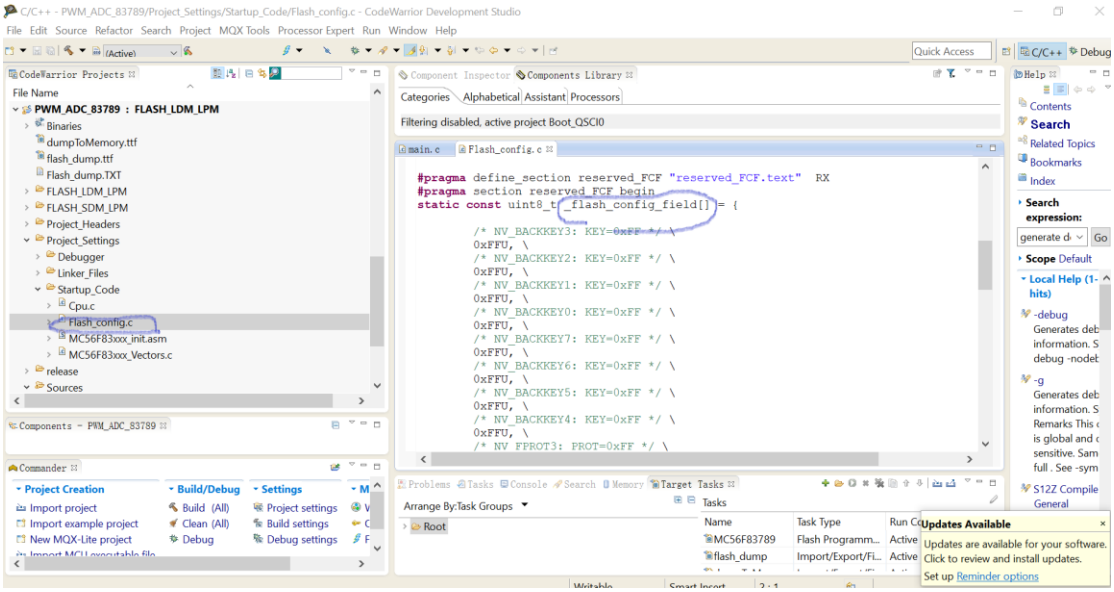
You can configure BCA of ROM with below structure variable:

```
static const Bootloader_Cofiguration_Area_TYPE BCA_config = {

        /* tag */ \
        {'k', 'c', 'f', 'g'}, \
        /* reserved */ \
    {0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU,0xffU}, \
        /* Enabled Peripherals */ \
        0xffU, \
        /* i2c slave address */ \
        0xffU, \
        /* peripheral detection timeout, in milliseconds */ \
        5000, \
        /* reserved */ \
        {0xffU,0xffU}, \
        {0xffU,0xffU}, \
        {0xffU,0xffU,0xffU,0xffU}, \
        0xffU, \
        0xffU, \
        0xffU, \
        0xffU, \
        {0xffU,0xffU,0xffU,0xffU}, \
        {0xffU,0xffU,0xffU,0xffU}, \
        /* reserved */
        0xffU, \
        /* can config1 */ \
        0xffU, \
        /* can config2 */ \
        0xffffU, \
        /* can TX Id */ \
        0xffffU, \
        /* can RX Id */
        0xffffU, \
        /* reserved */ \
        {0xffU,0xffU,0xffU,0xffU}
};
```

After Reset, if the MC56F83xxx enters bootloader, if it does not get input characters from peripherals until the time the red data specified has passed, the MC56F83xxx will jump to application code from bootloader and execute application code.
The port and corresponding pins the bootloader support are listed as following.
If the bootloader boots application code from CAN, the CAN ID and CAN baud rate must be set up in the *BCA_config array.*

The *BCA_config array and flash_config_field array are generated automatically by CodeWarrior for MCU when a new project is generated as following Fig.*



This device has various peripherals (UART, I2C, etc.) supported by the ROM Bootloader. To use an interface for bootloader communications, the peripheral must be enabled in the BCA, see the "Bootloader Configuration Area (BCA)" section. If the BCA is invalid (such as all 0xFF bytes), then all peripherals will be enabled by default. The next table shows the pads used by the ROM Bootloader.

Table 6-1. Bootloader Peripheral Pinmux

Peripheral	Instance	Alt Mode	Pins
UART/SCI	0	0	GPIOC2, QSCI0_TXD
		2	GPIOC3, QSCI0_RXD
I2C	0	0	GPIOC14, I2C0_SDA
			GPIOC15, I2C0_SCL
CAN	0	0	GPIO11, CAN Tx
			GPIO12, CAN Rx

4) For interrupt service routine, the “rti” must be the last line for the ISR. For the general subroutine, the “rts” must be the last line. In order to tell compiler whether the subroutine is ISR or general subroutine, the #pragma must be inserted as prefix.

```
#pragma interrupt on
```

```
    void pit0ISR(void){
    //Clear the PRF
    PIT0->CTRL &= ~PIT_CTRL_PRF_MASK;

    if(++updateFrqCount >= 5000){
    updateFrqCount = 0;
        updateFrqFlag = 1;

     }
    //  LED_RED_TOGGLE();
    }
    #pragma interrupt off
```

This is a general subroutine: The #pragma is not used.

```
//GPIOF8 as LED toggle
void GPIOF8Init(void)
{
    SIM->PCE0|=0x7E; //enable all GPIO port clock
    //GPIOC8 pin  is LED
    GPIOF->PER&=~(0x100);
    GPIOF->DDR|=0x100;
    GPIOF->DR|=0x100;
    GPIOF->DR^=0x100; //toggle LED
    }
```