

Bootloader Introduction



Gao Lei
Senior Automotive Applications Engineer



BootLoader的基本概念

一般来说，Boot Loader 可称作引导加载程序。

通常，Boot Loader 是严重地依赖于硬件而实现的，特别是在嵌入式系统中。在嵌入式系统中难以建立一个通用的 Boot Loader。

引导程序：就是在系统上电或复位后运行的一段小程序。这段程序将系统的软硬件环境带到一个合适的状态，为最终调用应用程序准备好正确的环境。

- 初始化硬件设备
- 建立正确的内存空间映射
- 初始化栈
- 检测并初始化内存
- 初始化全局变量

加载程序：将非易失存储器中的特定软件组件拷贝到RAM中，并运行之。

在汽车ECU中BootLoader通常指代码更新程序：

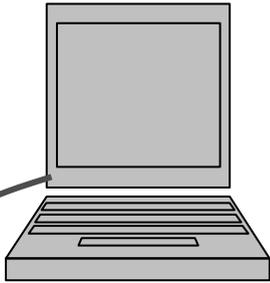
ECU在一种特殊的工作模式下，通过某种通信接口与主机（**Host**）相连。主机将新的目标代码下载到**ECU**中，**ECU**的代码得以更新。下次上电或复位后即运行新的程序。

汽车ECU采用BootLoader的好处：

- 可以将软件开发的时间延长；
- 如果用户有若干产品使用同一款**MCU**，可以降低库存的种类；
- 即使在产品的生产阶段发现了**Bug**，也可能避免灾难性的返工；
- 最终用户也可受益于产品的功能及性能上的升级。

主	目标文件解释程序
控	命令生成程序
程	通信程序
序	其它程序

主机端



	引导程序
主	通信程序
控	目标文件解释程序
程	命令解释程序
序	NVM 编程擦除程序

ECT端



SCI / CAN

如何进入Bootloader ?

- 复位时特定通信口的关键字检测（超时机制）；
- 复位时特定硬件IO的检测；
- 运行时特定通信口的关键字检测（对最终产品性能有影响）；

增量下载概念：

- 只更新部分功能代码（团队开发时使用）；
- 只更新数据（如字库，标定数据）；
- 只更新协处理器的代码或数据（XGATE）。

两次下载的概念：

- 接受Host传来的程序到RAM中并运行（其实是接受Bootloader）；
- RAM中刚收到的Bootloader负责来更新代码或数据；
- 最小的Boot程序和最大的灵活性。

作用：将**Link**后产生的可烧写的目标代码文件转换为需要编程的地址和编程数据

目标代码解释程序即可以在主机端，也可以在**ECT**一端

常用的可烧写目标代码格式

- **Intel**十六进制目标文件格式
- **Motorola S-Record**文件格式

目标代码解释程序：Motorola S-Record文件格式

S-record目标文件格式是将目标代码及目标数据以**16进制ASCII**的方式表示；好处是使得目标文件在计算机系统和开发工具之间易于转换。

记录（**Record**）的格式

S0: 为文件的开始，通常包含文件名

S1: 16位地址

S2: 24位地址

S3: 32位地址 **S7/S8/S9**

: 为文件的结束

S0140000443A5C74656636C6F6E6550726F2E61627304

S224FF8000CF24004A8044FE4A8000FE0002C011D2A70FE40000100F810002B2000000000002

...

S2 24 FE8000 C6205B0B4A8B78FEF62402874A8B8EFE4A8C34FECCC0AD3B4A80B5FECCC0FC6C D5

地址格式

Record长度

起始地址

Record数据

校验和 (Checksum)

...

S224FE94204A8F10FE068F711BF0260AEC87EE854A8CE3FE3DEE87E6865B1018E6003DEC872D

S9030000FC

校验和：将长度 / 起始地址 / 数据相加，取和的低 8 位，按位取反；

S12/S12X: *.s19文件，逻辑地址（页面）格式。 *.glo文件，全局地址格式

ECU的BootLoader常用通信接口

- 串口（SCI），可以使用一个LIN的接口
- CAN，可以采用标定用的接口

通信协议

- 波特率
- 帧结构
- 差错控制
- 握手及流量控制

以简单的串口为例

```
#define XON    0x11
#define XOFF   0x13
```

MCU中Flash的编程和擦除

- S12/S12X中的Block对编程和擦除的影响

- 通常在RAM中运行

 - 如何搬移代码到RAM中

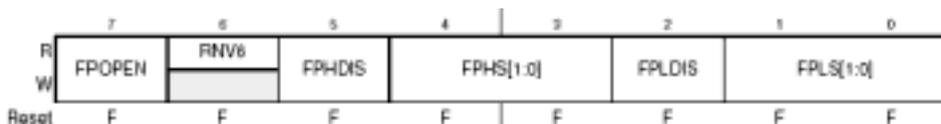
 - Link规格文件的编写

- Flash保护功能的应用

 - 将BootLoader在Flash中的映像保护起来，阻止对这部分空间的擦除和编程操作。好处是：一旦Bootloader运行出现异常（如异常断电），将不会产生难以恢复的后果。

Flash的保护方案（已S12XE 为例）

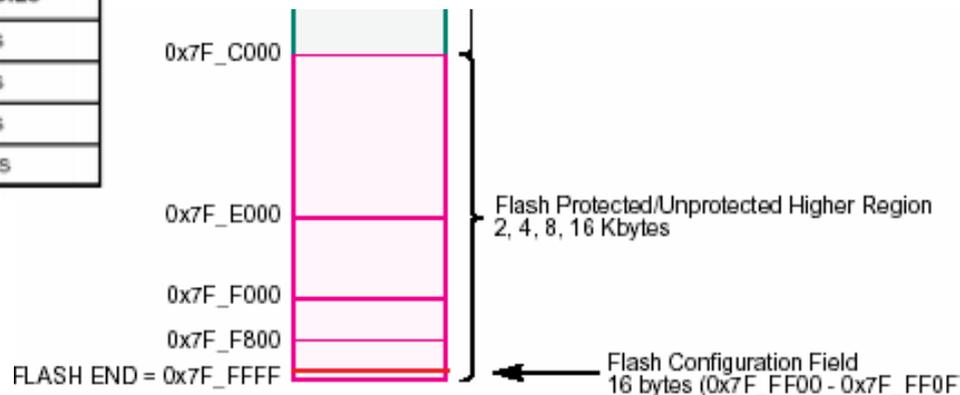
- 阻止对特定地址范围的Flash的擦除和编程操作；若出现了异常的擦写和编程操作，将产生Flash 出错中断。
- 通常有一个专用的寄存器**FPROT**来定义Flash的保护；在复位阶段从特定的Flash位置（如0x7F_FF0D, S12XE）load到FPROT中。



FPROT

Table 29-21. P-Flash Protection Higher Address Range

FPHS[1:0]	Global Address Range	Protected Size
00	0x7F_F800–0x7F_FFFF	2 Kbytes
01	0x7F_F000–0x7F_FFFF	4 Kbytes
10	0x7F_E000–0x7F_FFFF	8 Kbytes
11	0x7F_C000–0x7F_FFFF	16 Kbytes



目的：一个**BootLoader**的示例程序，可以此为基础开发自己的**BootLoader**。

基本功能：主机（**Host**）可以通过**CAN**的适配板（**Adaptor**）对目标板（**Target**）进行**Flash**的编程

基于**Freescale Demo9S12XDP512** 演示板

IDE环境是**CodeWarrior4.6**

Flash的基本擦写及编程代码

串口通信代码

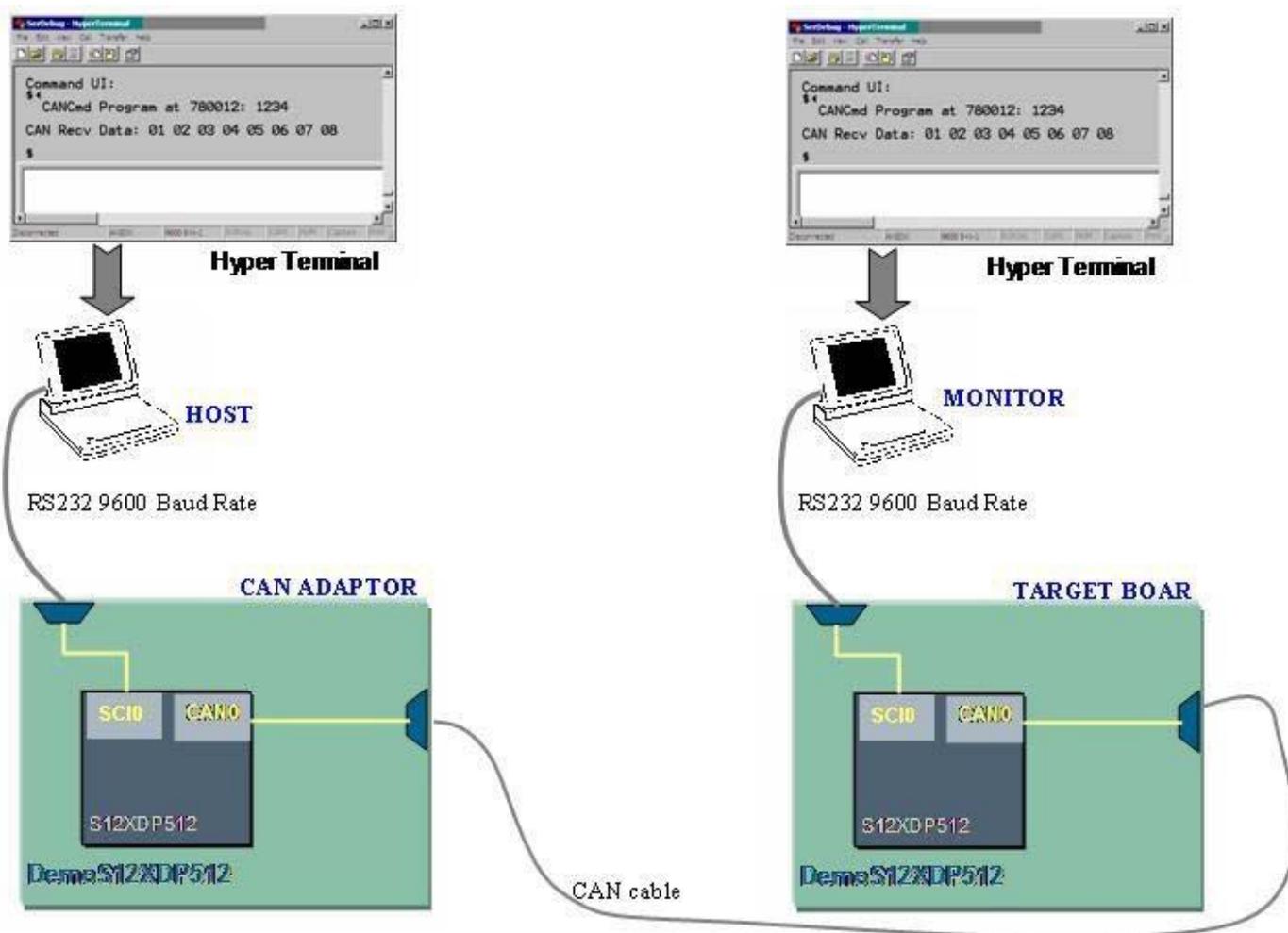
CAN通信代码

Demo板作为目标板

Demo板作为主机的**CAN**适配板

可以方便地移植到**Freescale**的**S12 / S12X**系列**MCU**上

Boot Loader 运行时框图



Boot Loader 运行时框图：主机端

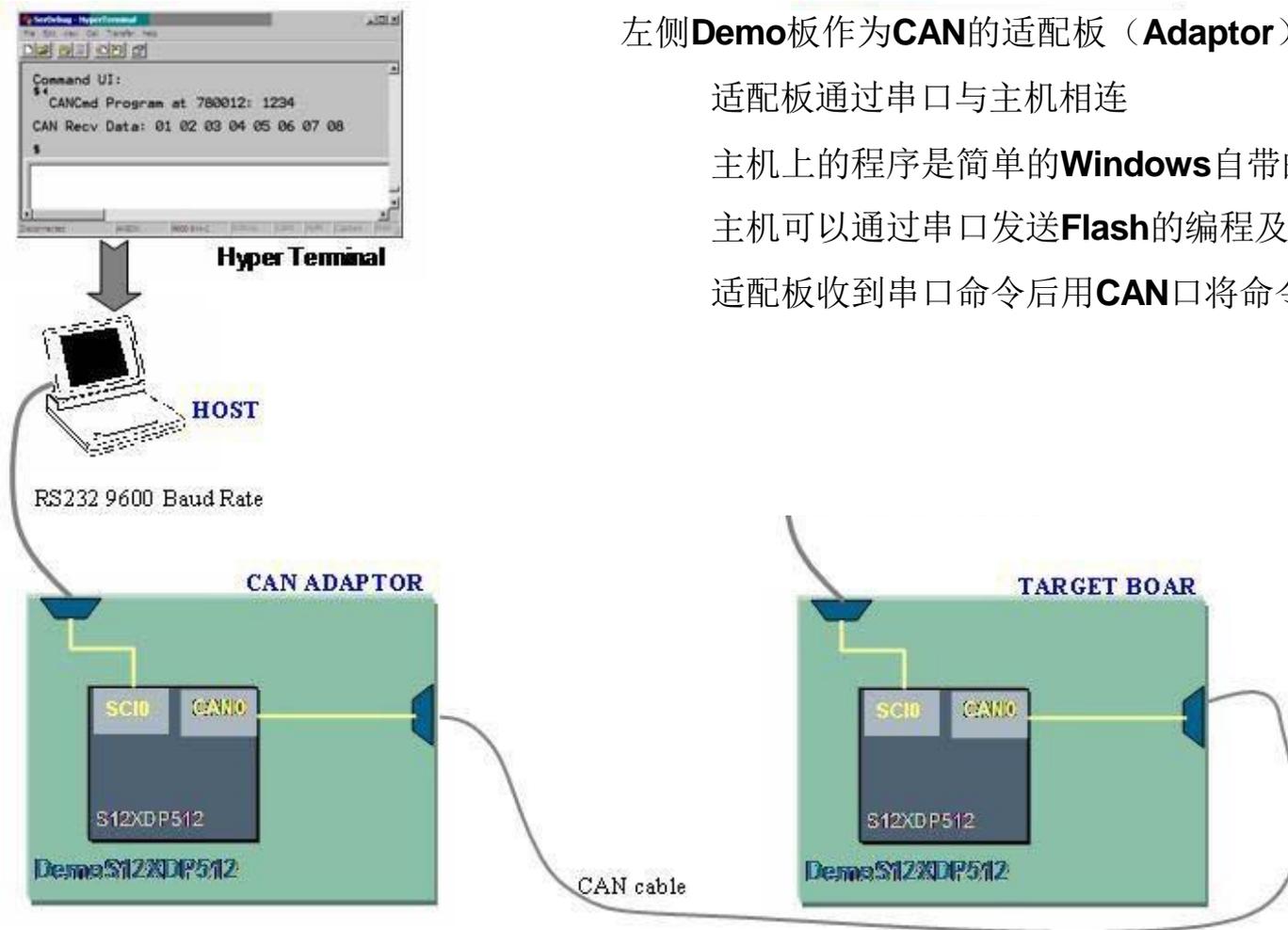
左侧**Demo**板作为**CAN**的适配板（**Adaptor**）

适配板通过串口与主机相连

主机上的程序是简单的**Windows**自带的超级终端程序

主机可以通过串口发送**Flash**的编程及擦写命令

适配板收到串口命令后用**CAN**口将命令发送到目标板



Boot Loader 运行时框图：目标机端

右侧**Demo**板作为**ECU**目标板

目标板通过串口与主机相连

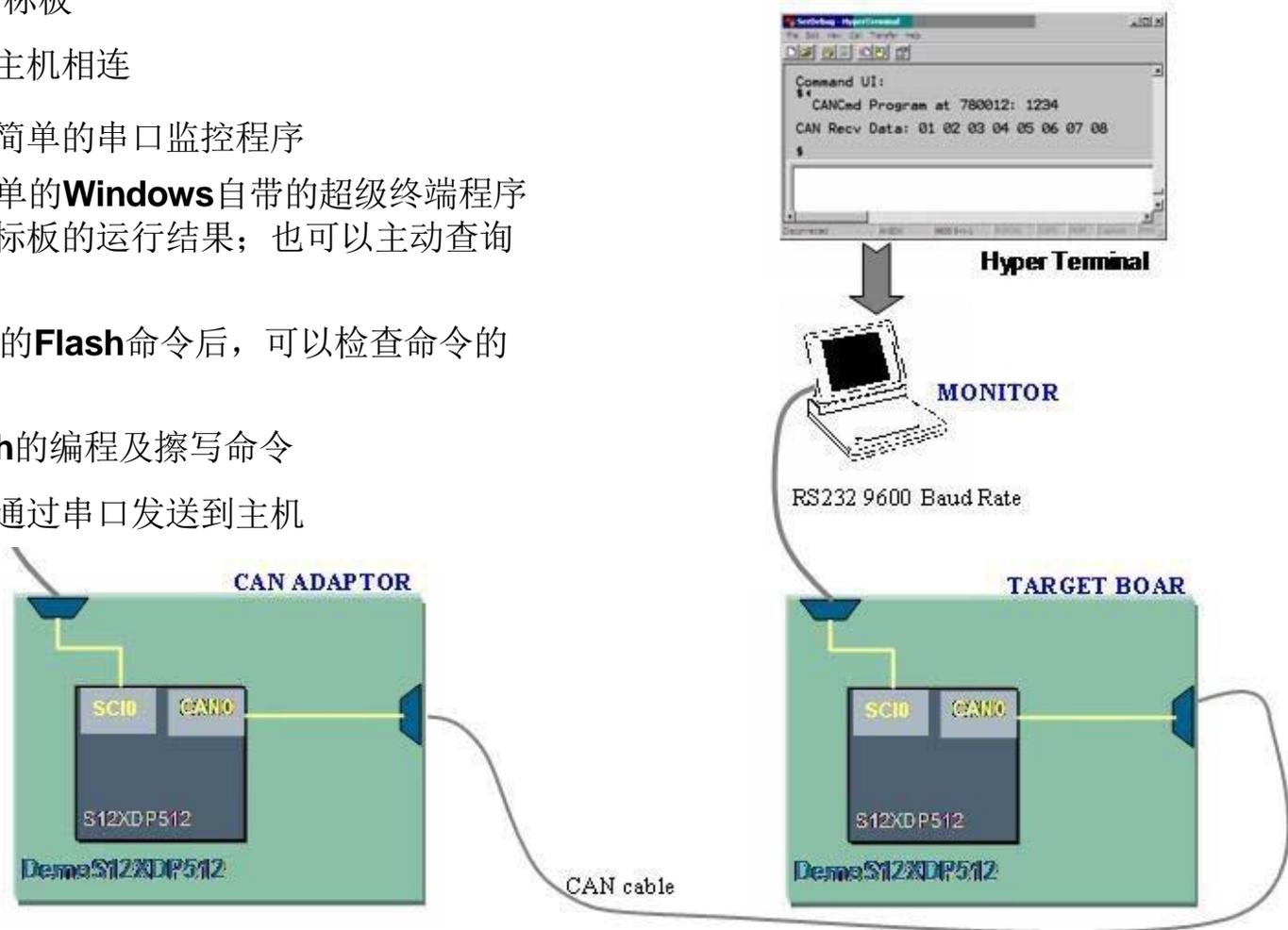
目标板上运行一个简单的串口监控程序

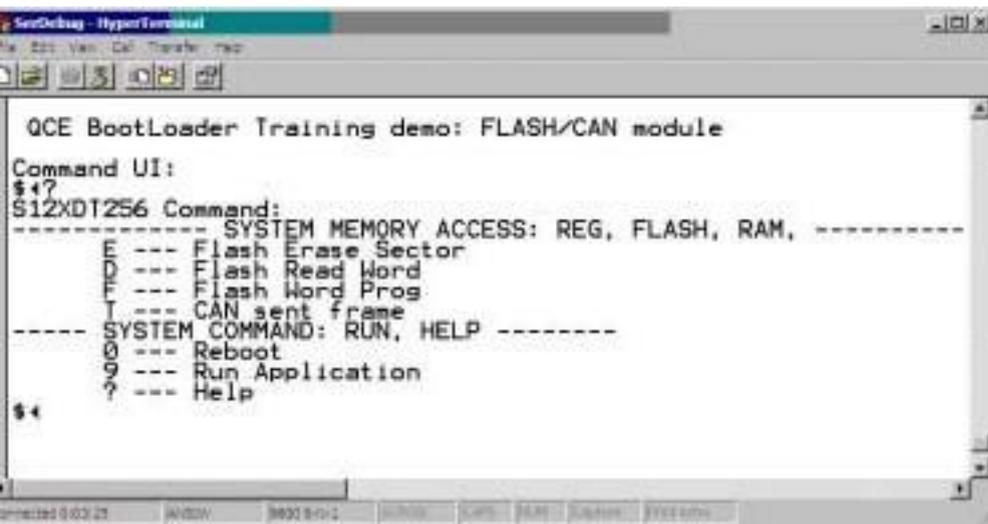
主机上的程序是简单的**Windows**自带的超级终端程序，可以实时显示目标板的运行结果；也可以主动查询目标板的状态

目标板收到**CAN**口的**Flash**命令后，可以检查命令的有效性

若合法则执行**Flash**的编程及擦写命令

将命令的执行结果通过串口发送到主机





```
QCE BootLoader Training demo: FLASH/CAN module
Command UI:
$*?
S12XDT256 Command:
----- SYSTEM MEMORY ACCESS: REG, FLASH, RAM, -----
E --- Flash Erase Sector
D --- Flash Read Word
F --- Flash Word Prog.
I --- CAN sent frame
----- SYSTEM COMMAND: RUN, HELP -----
0 --- Reboot
9 --- Run Application
? --- Help
$*
```

主机命令:

E: Flash擦除命令

D: 读字命令

F: Flash字编程命令

T: 发送CAN命令帧

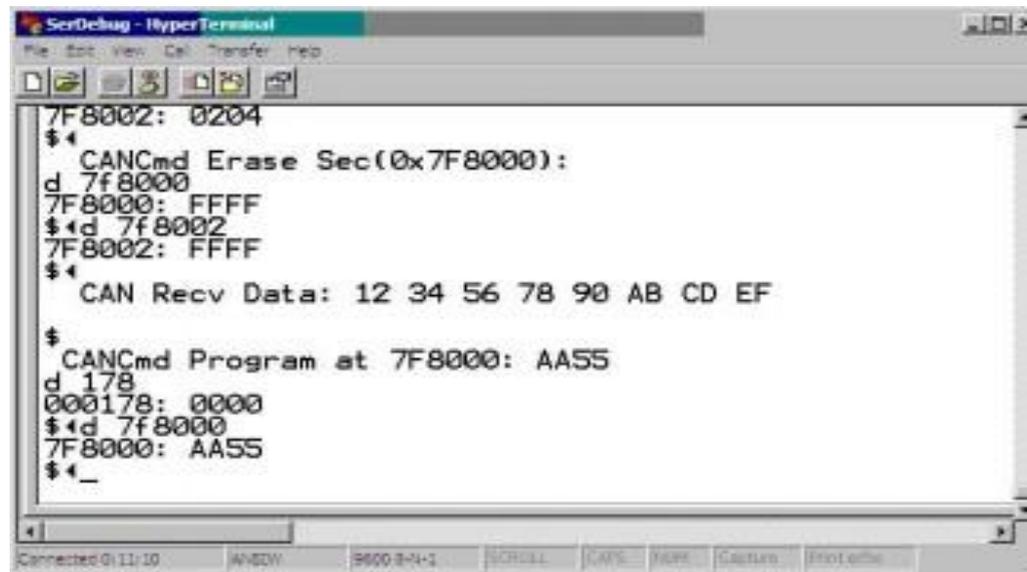
主机监控（信息显示）:

成功擦除一个**Sector**

成功编程一个字（**WORD**）

收到的**CAN**帧内容

收到命令的错误信息



```
7F8002: 0204
$*
  CANCmd Erase Sec(0x7F8000):
d 7f8000
7F8000: FFFF
$*d 7f8002
7F8002: FFFF
$*
  CAN Recv Data: 12 34 56 78 90 AB CD EF

$
  CANCmd Program at 7F8000: AA55
d 178
000178: 0000
$*d 7f8000
7F8000: AA55
$*_
```

演 示 (一)

Flash command via CAN

Flash Erase Command:

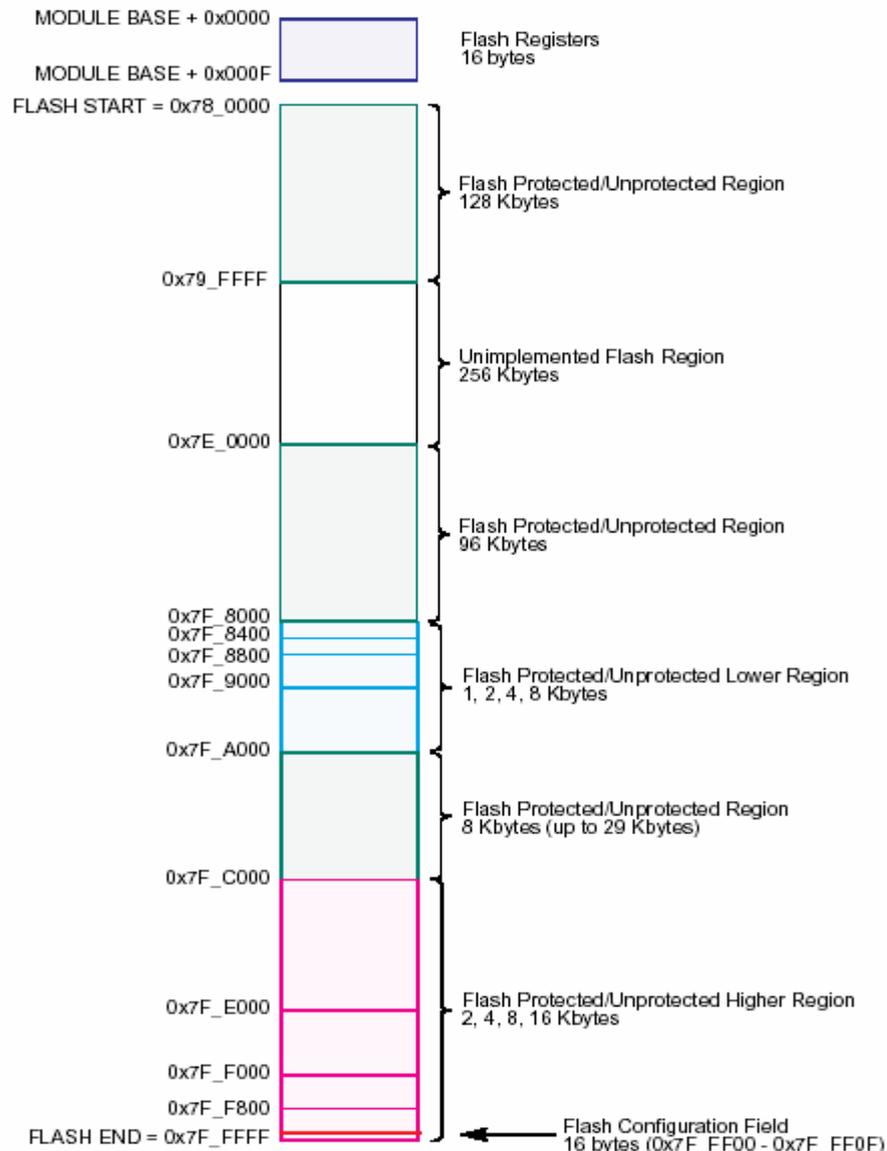
CAN0 CAN1 CAN2 CAN3
EE ADDW2 ADDW1 ADDW0

Exp:
 T EE 7F C0 00
 T EE 7F C4 00

Flash Program Command:

CAN0 CAN1 CAN2 CAN3 CAN4 CAN5
FF ADDW2 ADDW1 ADDW0 DH DL

Exp:
 T FF 78 00 00 12 34
 T FF 78 0C 00 AB CD



CAN通信程序:

mscan.c / mscan.h

Flash的编程及擦除:

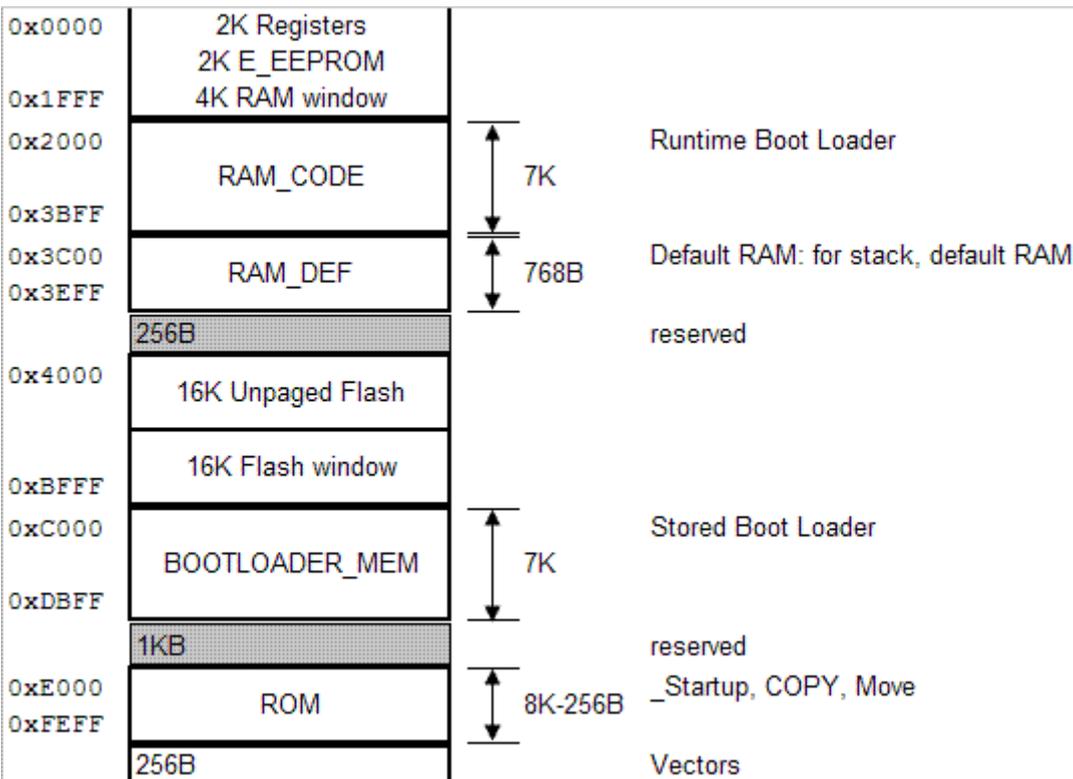
flash.c / flash.h

BootLoader的代码搬移:

P&E_Multilink_CyclonePro_linker.prm / startup.c

主控及串口通信程序:

main.c



局部**64K**空间

没有采用**Flash**的保护

Boot Loader 软件介绍: BootLoader的代码搬移

```
SEGMENTS
    RAM_DEF      = READ_WRITE    0x3C00 TO    0x3EFF;
    RAM_CODE     = READ_ONLY     0x2000 TO    0x3BFF;
    ROM          = READ_ONLY     0xE000 TO    0xFEFF;
    ...
/* relocated memory */
    BOOTLOADER_MEM = READ_ONLY 0xC000 TO 0xDBFF
                        RELOCATE_TO 0x2000;
END

PLACEMENT
    NON_BANKED,
    ...
    COPY
                        INTO ROM;
    BOOT_SEG          INTO BOOTLOADER_MEM;
    ...
END
```

P&E_Multilink_CyclonePro_linker.prm

RELOCATE_TO 关键字

```
#pragma CODE_SEG BOOT_SEG
```

代码

```
#pragma CODE_SEG DEFAULT_ROM
```

代码

#pragma 关键字

```
__EXTERN_C void _Startup(void) {  
    ...  
    MoveBootLoader();  
  
    main();  
}  
  
void MoveBootLoader(void)  
{  
    unsigned int *pRom, *pRam;  
  
    int cnt=3584; //3.5K word, 7K byte  
  
    pRom=(unsigned int*)0xC000;  
    pRam=(unsigned int*)0x2000;  
  
    while(cnt--)  
        *pRam++ = *pRom++;  
}
```

Startup.c

```
#pragma CODE_SEG BOOT_SEG
...
void MSCAN0Init(void) {
    ...
}

void MSCAN0Trans(byte * msgout)
{
    ...
}

void interrupt MSCAN0Rec(void) {
    ...
    ProcCanCmd();
}
```

CAN0初始化程序

CAN发送一个数据帧

CAN接受一个数据帧

这是一个中断服务程序

调用了执行**CAN**命令的函数

Mscan.c

```
void ProcCanCmd(void)
{
    ...
    switch (msgin[0]) {
        case ERASE_CMD:
            ...
            err_code=Flash_Erase_Sector((unsigned int *far)addr32);
            ...
            break;
        case PROGRAM_CMD:
            ...
            err_code=Flash_Write_Word((unsigned int *far)addr32,value16);
            ...
            break;
        default:
            ...
            break;
    }
}
```

```
signed char Flash_Write_Word(unsigned int *far far_address, unsigned int data)
{
    . . .
    // step1: test CBEIF flag to ensure that address/data/command buffers are empty.
    while ((FSTAT&0x80)!=0x80);

    // step2: verify all ACCERR and PVIOL flag in the FSTAT are cleared.
    FSTAT = FSTAT_PVIOL_MASK|FSTAT_ACCERR_MASK;

    // step3: write the PPAGE to select one of the page to be programmed
    PPAGE = page;

    // step4: Dummy store to page to be erased
    *address = data;

    // step5: store programming command in FCMD: FCMD_CMDB5_MASK=0x20
    FCMD = FCMD_CMDB5_MASK;

    // step6: Clear CEBIF by writing "1" to it to launch the command.
    FSTAT = 0x80;

    // step7: test CCIF
    while ((FSTAT&0x40)!=0x40);
    . . .
}
```

```
signed char Flash_Erase_Sector(unsigned int *far far_address)
{
    . . .
    // step1: test CBEIF flag to ensure that address/data/command buffers are empty.
    while ((FSTAT&0x80) !=0x80);

    // step2: verify all ACCERR and PVIOL flag in the FSTAT are cleared.
    FSTAT = FSTAT_PVIOL_MASK|FSTAT_ACCERR_MASK;

    // step3: write the PPAGE to select one of the page to be programmed
    PPAGE = page;

    // step4: Dummy store to page to be erased
    *address = 0xFFFF;

    // step5: store programming command in FCMD: FCMD_CMDB5_MASK=0x40
    FCMD = FCMD_CMDB6_MASK;

    // step6: Clear CEBIF by writing "1" to it to launch the command.
    FSTAT = 0x80;

    // step7: test CCIF
    while ((FSTAT&0x40) !=0x40);
    . . .
}
```

```
void Flash_Init(unsigned long oscclk)
{
    unsigned char fclk_val;

    // Next, initialize FCLKDIV register to ensure we can program/erase
    if (oscclk >= 12800) {
        fclk_val = oscclk/8/175 - 1; // FDIV8 set since above 12MHz clock
        FCLKDIV = FCLKDIV | fclk_val | FCLKDIV_PRDIV8_MASK;
    }
    else {
        fclk_val = oscclk/175 - 1;
        FCLKDIV = FCLKDIV | fclk_val;
    }

    FPROT = 0xFF; // Disable all protection (only in special modes)
    FSTAT = FSTAT | (FSTAT_PVIOL_MASK|FSTAT_ACCERR_MASK); // Clear any errors
}
```

主要是正确设置**Flash**的时钟

在 **main.c** 中调用了 `Flash_Init(4000);`

```
void main(void) {  
    ...  
    SetSci0Port();  
    Flash_Init(4000);  
    MSCAN0Init();  
    EnableInterrupts;  
  
    printf("\n\r QCE BootLoader Training demo: FLASH/CAN module\n\r");  
    for(;;) {  
        DbgMem();  
    }  
}
```

```
void DbgMem(void)  
{  
    while(1) {  
        switch(command) {  
            case 'D':    ...  
            case 'E':    ...  
            case 'F':    ...  
            case 'T':    ...  
            case '?':  
                printf("\nS12XDT256 Command:\n\r");  
                printf("----- SYSTEM MEMORY ACCESS: REG, FLASH, RAM, ----- \n\r");  
                printf("      E.--- Flash Erase Sector\n\r");  
                printf("      D --- Flash Read Word\n\r");  
                printf("      F.--- Flash Word Prog\n\r");  
                printf("      T --- CAN sent frame\n\r");  
                ...  
            default:    ...  
        }  
    }  
}
```

演 示 (二)

利用**CAN**的**LOOP**模式，一个**Demo**板也可以工作

Flash command via CAN

Flash Erase Command:

CAN0 CAN1 CAN2 CAN3
EE ADDW2 ADDW1 ADDW0

Exp:

T EE 7F C0 00

T EE 7F C4 00

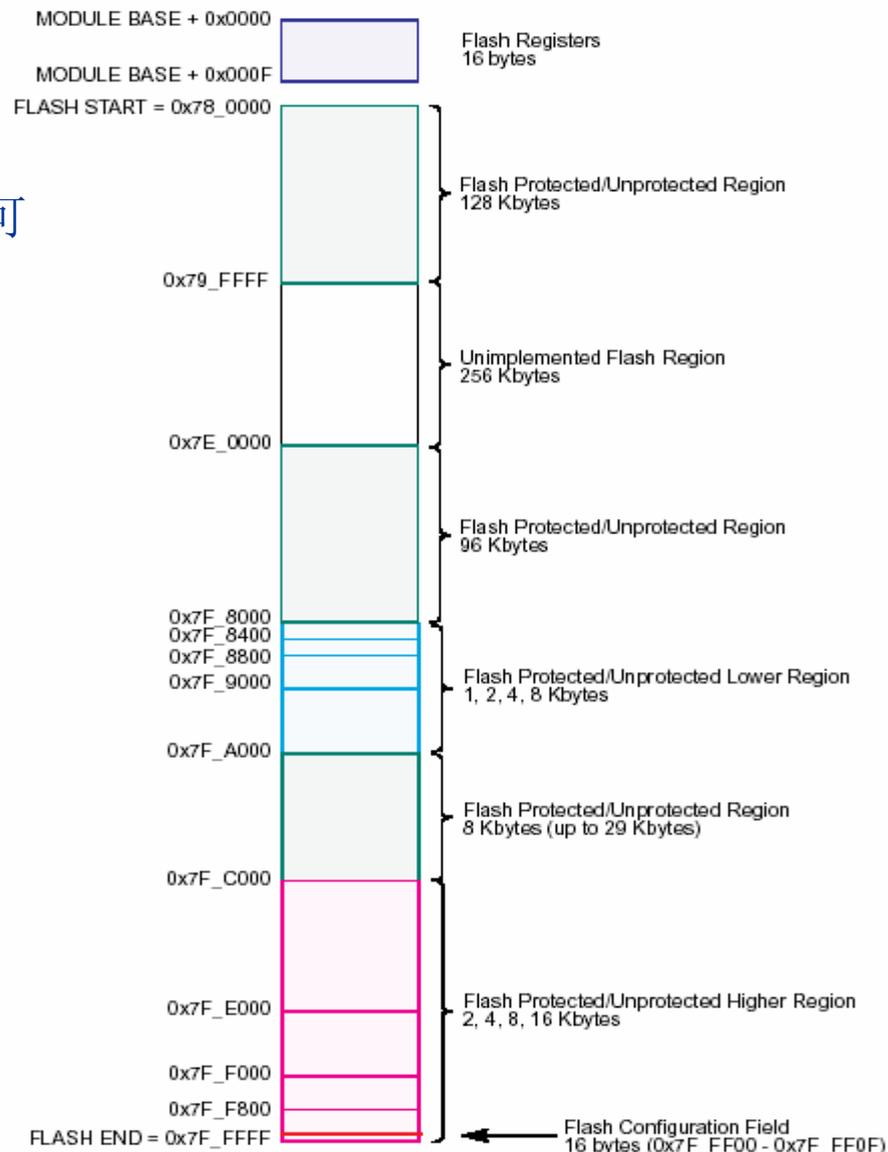
Flash Program Command:

CAN0 CAN1 CAN2 CAN3 CAN4 CAN5
FF ADDW2 ADDW1 ADDW0 DH DL

Exp:

T FF 78 00 00 12 34

T FF 78 0C 00 AB CD



Thank you!

