TMS320F281x Boot ROM Serial Flash Programming

摘要

本应用报告描述了 TI Flash 应用编程接口(API)的应用，此 API 是 TI Flash 算法的软件接口。在使用本报告之前，注意需懂得 Flash API 文档的基础。本文档不能取代 Flash API 文档，而是指导你关注包含 Flash API 文档的一系列 TI 资料中最重要的地方。详细参考文献列表参见"参考文献"一节。

目前，为 TMS320F281x 芯片进行在线(in-circuit)串行 Flash 烧写的可选方案是 Spectrum Digital 公司的 SDFlash。SDFlash 是一款卓越的基于 Windows 的对 TMS320F281x 芯片进行在线(in-circuit)烧写的软件，遵循 IEEE 标准 1149.1-1190，IEEE 标准测试访问接口和边界扫描架构或系统通讯接口(SCI)。推荐使用 GUI 方式。如果你手头没有安装了 Windows 的 PC，本文档将帮助你配置一个定制测试设备。

在本文档的适当章节可找到样例软件。本文档使用的硬件包括：

- Spectrum Digital 公司的 F2812 eZdsp(TM)和 IEEE 标准 1149.1-1190(JTAG)仿真器(XDS510(TM) USB)
- Link-research 公司的 RS-232 接口板

本应用报告讨论的工程附属代码和源代码可从 http://www-s.ti.com/sc/techlit/spraaq2.zip 下载。

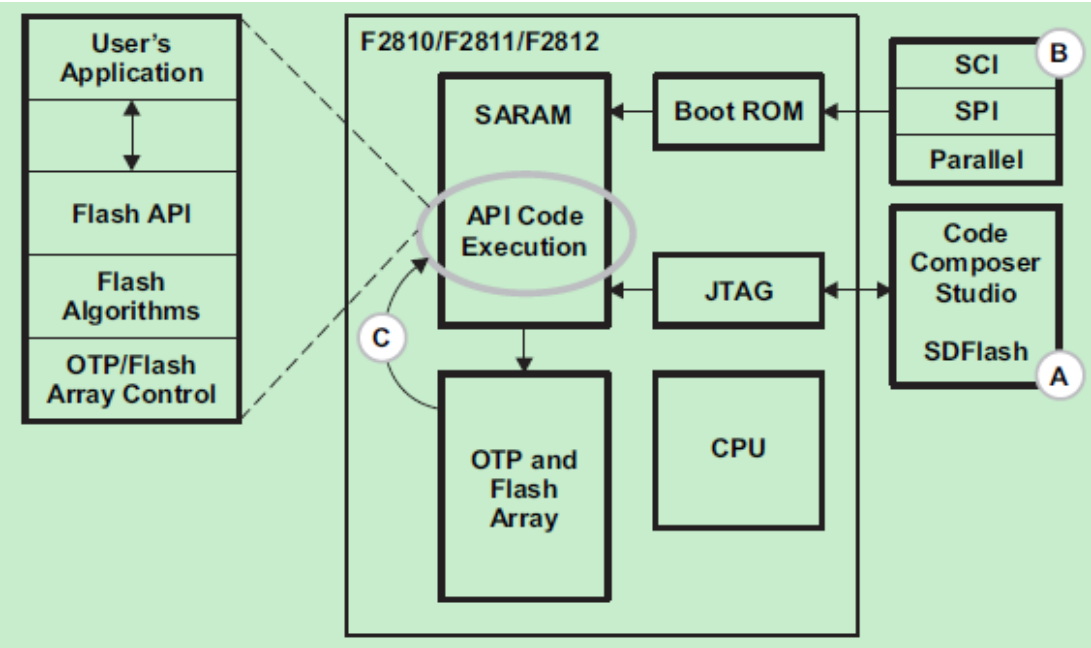
1 介绍

基于串口(RS-232)对 TMS320F2812，TMS320F2811，或 TMS320F2810(F281x)进行 Flash 烧写是目前流行的在线烧写方法。它的一个主要优点是减少了线下 Flash 烧写带来的损坏风险以及损坏处理。SCI 通讯可以从 PC、产线在线测试仪或其他处理器中产生。

本文讨论了从 F281x 引导 ROM SCI-A 对 Flash 烧写的每一个步骤：

- 将内核软件和 Flash 算法从引导 ROM 传送到目标 RAM
- 向目标 Flash 传送应用代码并烧写
- 将 Flash 烧写时间最小化

目标板支持的任何通讯方式都可以与 TI 提供的 Flash 烧写 API 一起使用。下载 TMS320F2810, TMS320F2811 和 TMS320F2812 Flash API (http://www-s.ti.com/sc/techlit/sprc125.zip)。F281x 引导 ROM 提供的将 Flash API 传送到 RAM 的方法有：SCI、SPI 和并行 GPIO。对其他通讯接口，你可以将通讯方式固化在 OTP 中或 Flash 的保护扇区中。Figure 1 展示了这些引导加载方式。



【Figure 1 F281x Flash 引导加载选项】

本应用文档使用引导 ROM 中的 SCI-A Flash 引导加载方式。

目前，为 TMS320F281x 芯片进行在线(in-circuit)串行 Flash 烧写的可选方案是 Spectrum Digital 公司的 SDFlash。根据 Spectrum Digital 公司的描述，SDFlash 是一款 Windows 软件，能通过 Spectrum Digital 公司的 JTAG 仿真器对目标 DSP 进行 Flash 烧写。使用的 Flash 烧写算法要和 DSP 的型号、容量相对应。

关于 C2000(TM) Flash 选项的详细列表，请访问 www.ti.com

2 方法论

这里描述的过程，关键是：

- F281x 引导 ROM SCI-A 方式
- 通讯内核与 Flash API(CKFA)
- 你的应用代码(AppCode)

基本过程是：

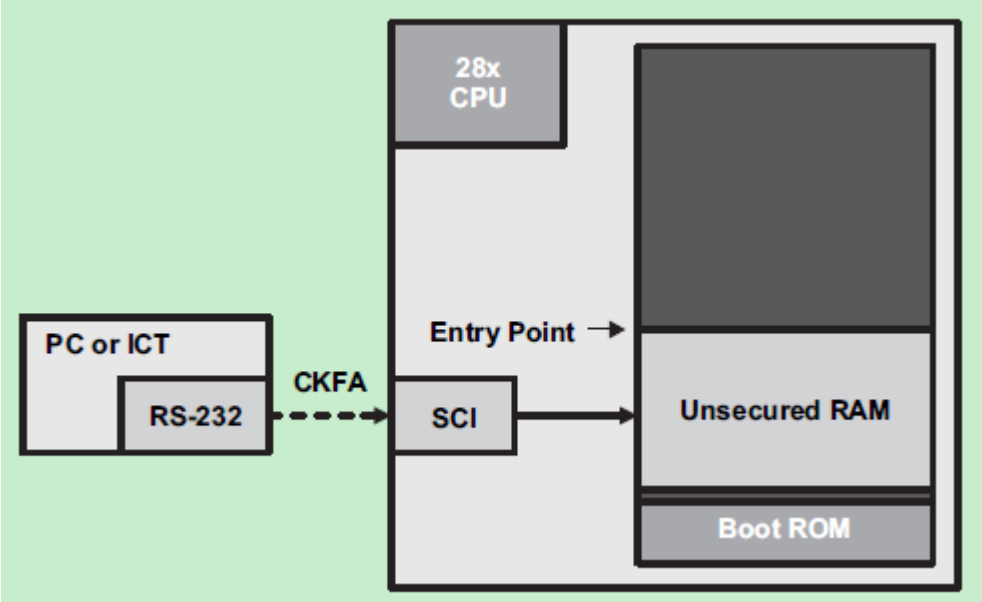
- 在引导 ROM 控制下将 CKFA 由 SCI 传送到 F281x 内部 RAM
- 在 CKFA 控制下将 AppCode 传送到 F281x RAM 并烧写到 Flash

注意：

本应用报告中使用的 CKFA 代码源自《TMS320F2810,TMS320F2811 and TMS320F2812 Flash API》（以下简称《Flash API》）Flash API 样例代码，因此，《Flash API》对 API、随 API 提供的样例代码，以及本文档来说都是极佳的参考书。在修改 CKFA 源代码之前，务必详尽阅读《Flash API》。

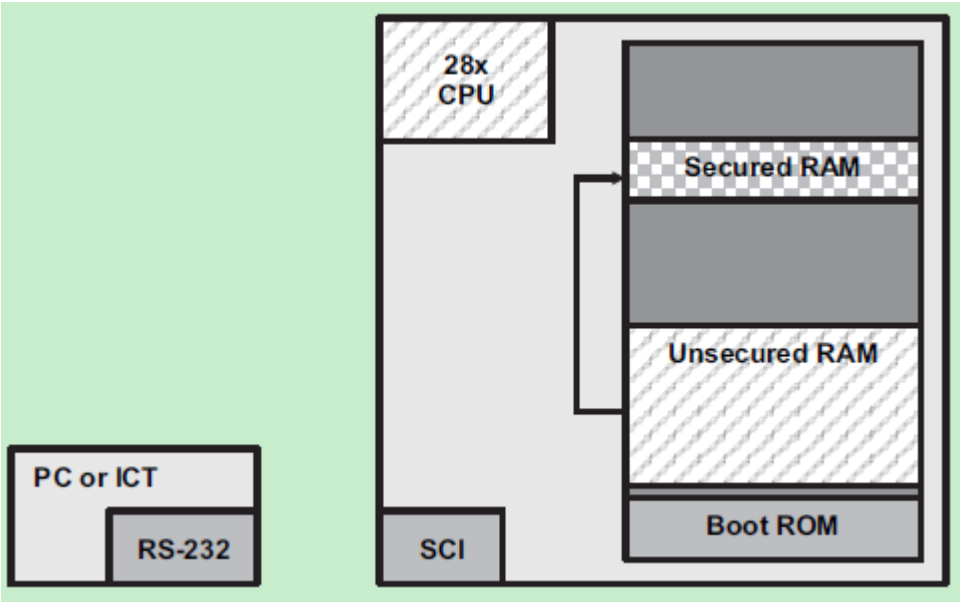
2.1 将 CKFA 传送到 F281x RAM

首先，通过引导 ROM SCI-A 方式将 CKFA 二进制文件传送到非安全 RAM 中的加载地址(Figure 2)。由于 F281x 可能处于加锁状态，因此 CKFA 代码必须首先从非安全 RAM 开始运行。更多信息，参考《TMS320x281x DSP System Control and Interrupts Reference Guide》(SPRU078)的代码安全模块(CSM)章节。



【Figure 2 将 CKFA 传送到 RAM 加载地址】

引导 ROM 代码一旦结束 CKFA 传输，就将控制权由 F281x CPU 转移给 CKFA(Figure 3)。在 CSM 被解锁后，CKFA 代码可以访问所有内部 RAM 和 Flash。



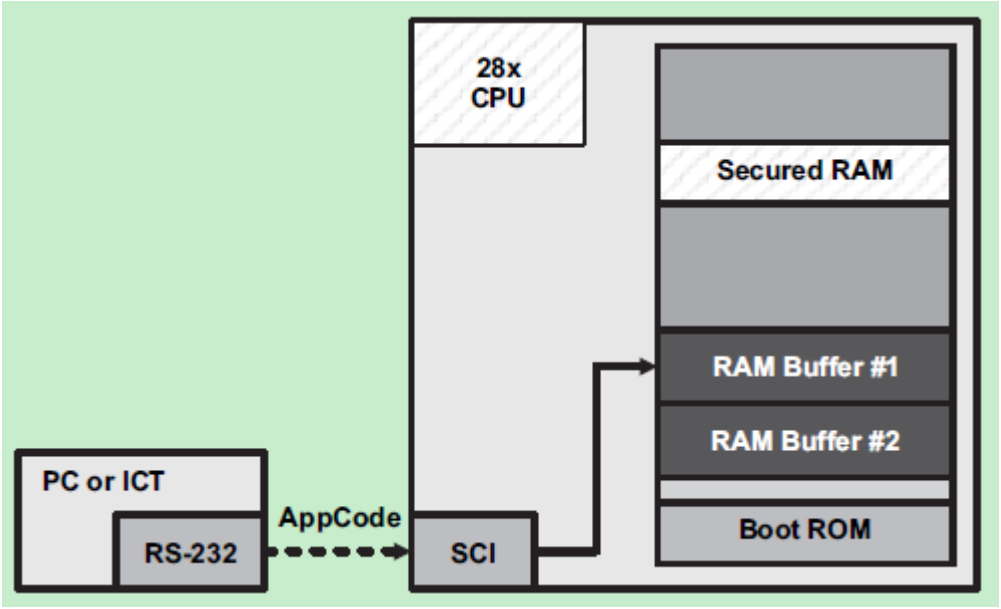
【Figure 3 CKFA 将自己传送到 RAM 运行地址】

CKFA 代码使用的 cmd 文件定义了加载地址。如果解锁成功，CKFA 代码将自己的主要部分从非安全 RAM 中的加载地址拷贝到安全 RAM 中的运行地址。如果解锁不成功，CKFA 会将状态通过 SCI 发出来。

CKFA 的目标不是使用安全 RAM，而是使 H0 大 RAM 可用于上传 AppCode。如果 CKFA 软件足够小，则它将被传送到 M0/M1 非安全 RAM，并从那里执行。

2.2 传送应用代码并烧写

在 CKFA 代码开始执行并控制了目标 DSP 的 SCI-A 外设的情况下，应用代码的二进制文件被传送到 4K word RAM buffer 1 (Figure 4)。这个 buffer 满了以后，开始 Flash 烧写，并且 CKFA 代码将应用代码的下一个 4K word 传送到 RAM buffer 2 (Figure 5)。



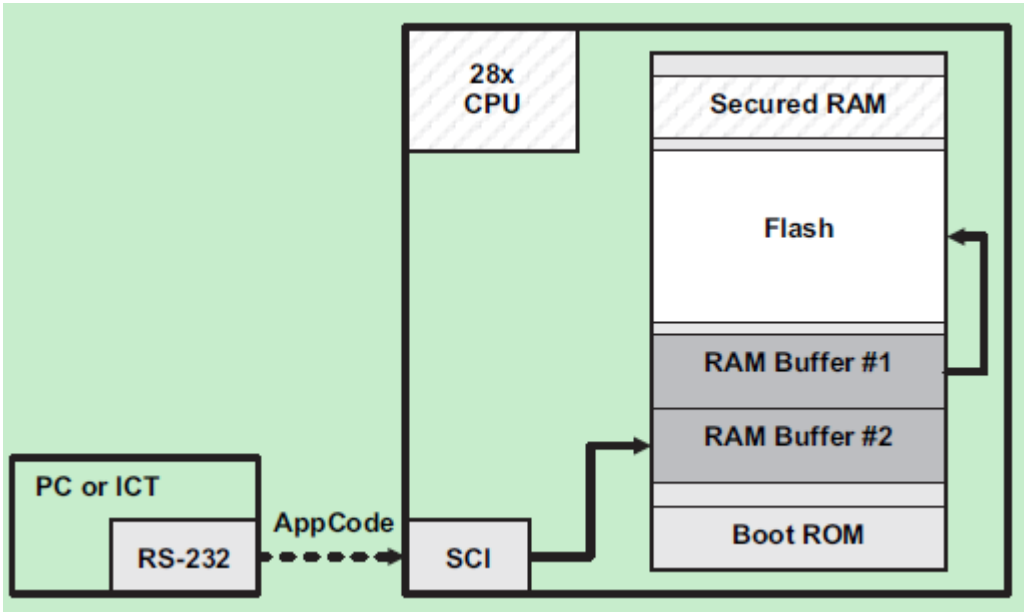
【Figure 4 CKFA 将 AppCode 传送到 RAM buffer #1】

Flash API 函数提供了一个 call-back 函数，当烧写 Flash 时，它允许 SCI 继续将数据传送到 RAM。CKFA 代码使用 call-back 函数功能烧写 Flash：

- 当向 RAM buffer 2 传送下一个 AppCode 块时，从 RAM buffer 1 烧写 Flash；
- 当向 RAM buffer 1 传送下一个 AppCode 块时，从 RAM buffer 2 烧写 Flash；

当 RAM buffer 1 填满时，Flash 烧写就开始了。当烧写 Flash 时，SCI 的 16 级 FIFO 仍在持续的接收 SCI 字符，从而减少了总的烧写时间。

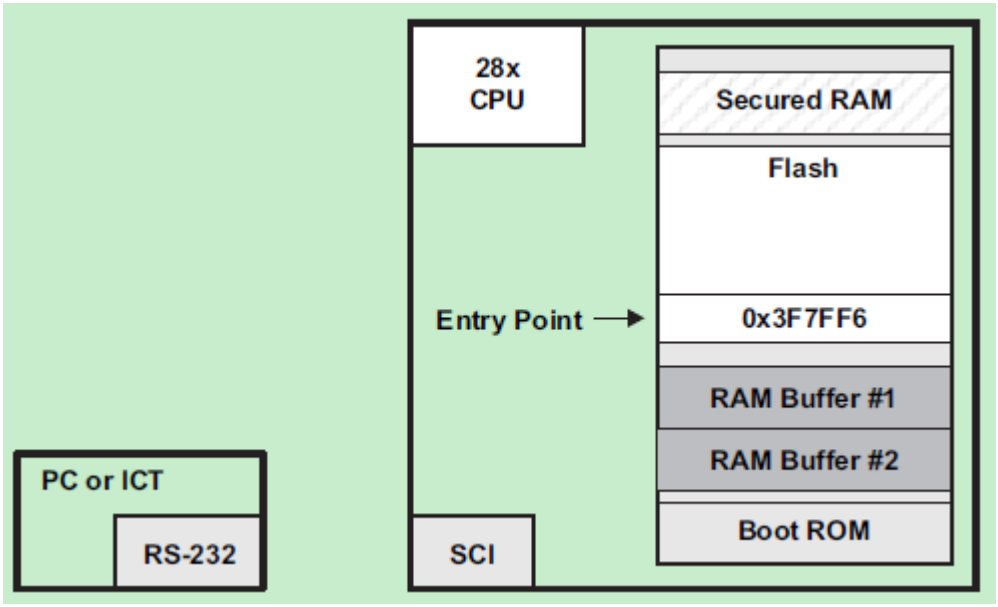
本应用文档中使用的方法要求对 F2810 的整个 64K word 空间都可以访问。这也适用于 F2811 和 F2812 的 128K word Flash。Flash 的所有地址都应可配置。因此，本文中的代码的 Flash 烧写开始地址，对于 F2810 都是固定的 0x3E8000，对于 F2811 都是固定的 0x3D8000。芯片的 memroy map 和 Flash 扇区参见附录 A 和附录 B。



【Figure 5 CKFA 开始 Flash 烧写】

2.3 烧写结束，定义应用代码的 Entry Point

当烧写结束时，F281x 程序计数器指向 Flash 的 entry point(0x3F7FF6)（Figure 5 与附录 B）。DSP 现在已经准备好运行烧写在 Flash 中的应用代码了。如果引导 GPIO 引脚被配置为 Flash 执行模式，则在重启后将运行这些代码(Figure 6)。



【Figure 6 Flash 烧写结束】

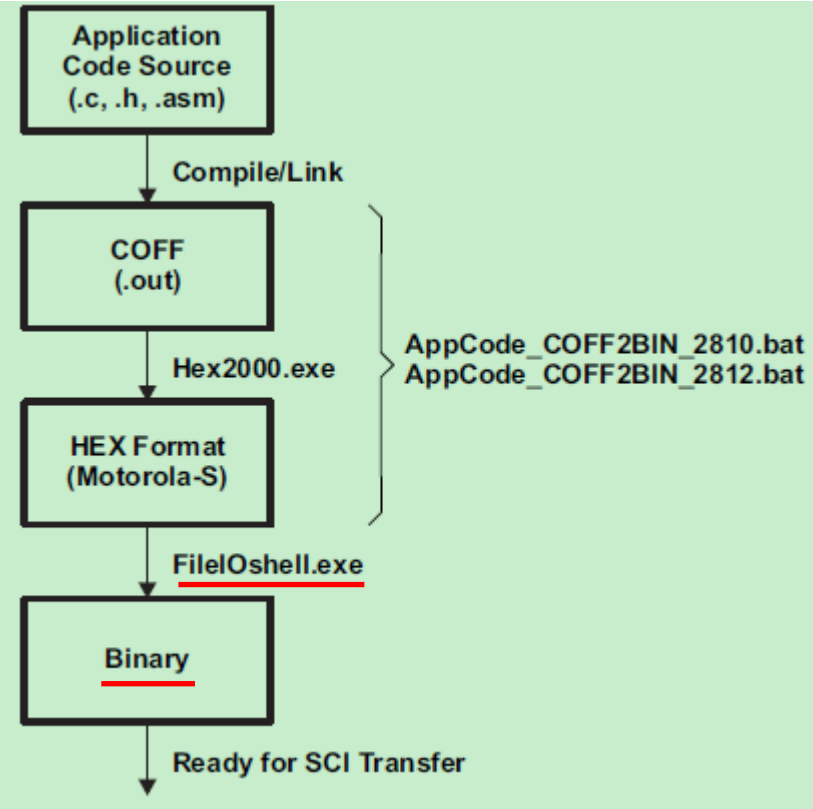
3 过程

本应用报告描述的 Flash 烧写步骤包括：准备应用代码、准备 CKFA 软件、建立串行通讯、烧写 Flash。下面的章节将详细说明这些步骤。

3.1 准备应用代码

要将应用代码烧写到 Flash，CKFA 代码需先通过 SCI 接收应用代码的二进制文件，然后对全部 Flash 地址区域进行烧写。CKFA 代码要求应用代码做如下事情：

- 填充未用的 Flash 地址区域
- 为 SCI 单独创建一个二进制文件



【Figure 7 AppCode 文件处理过程概览】

3.1.1 填充未用的 Flash 地址区域

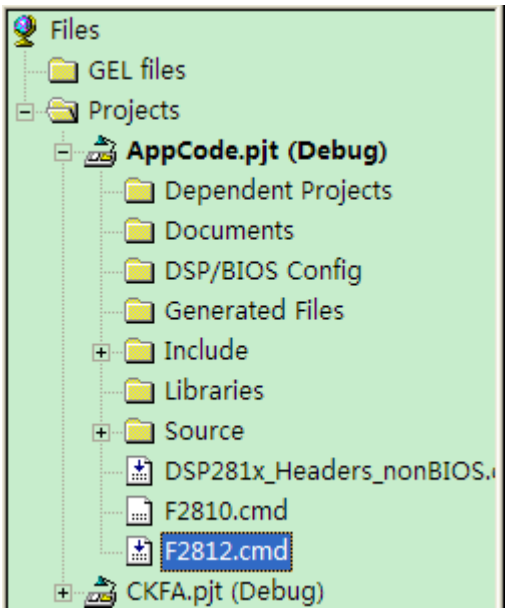
在 AppCode 通过 SCI 传送到 RAM 并随后烧写到 Flash 过程中，CKFA 软件控制 DSP。AppCode 必须被配置为填满 Flash 所有地址区域，包括未用到的地址。这样 CKFA 处理的就是连续的块数据，可以减少 Flash 烧写时间。

另外，在未用到的地址区域填充特定的固定数值会增强系统的健壮性。如果用一个已知的非法操作码填充未用地址区域，则当应用软件从程序地址之外的地址取指令时（软件 bug），就会进入非法指令陷阱。0xFFFF 就是这样一个非法操作码，详见《TMS320C28x DSP CPU and Instruction Set Reference Guide》(SPRU430)。

使用 0xFFFF 填充 Flash 未用地址区域还会减少 Flash 烧写时间，因为 Flash 在烧写时仅烧写值为 0 的位，不烧写值为 1 的位。F281x 出货时，TI 是全部擦除过的，也就是说每个 Flash 地址在出货时都是 0xFFFF。因此，在一个新的芯片上，未用地址区域都是推荐值 0xFFFF。

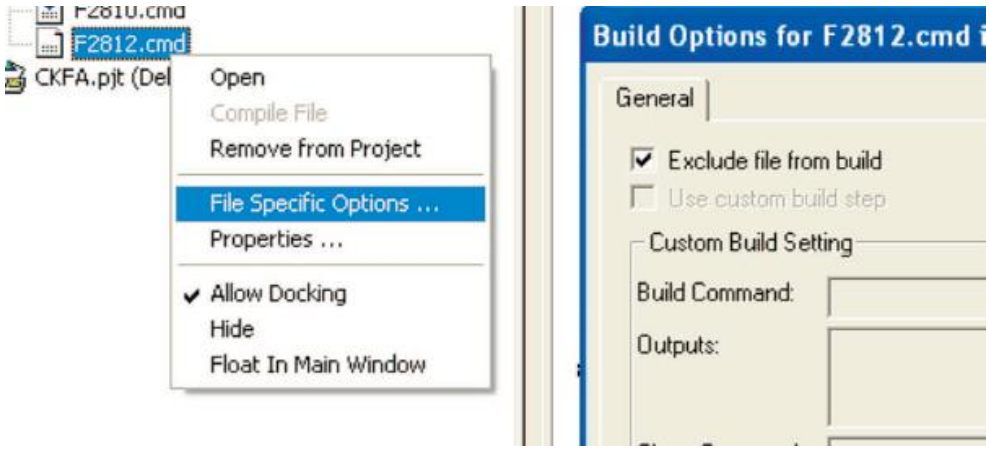
3.1.1.1 根据存储区是 64K word 还是 128K word 选择 cmd 文件

随本应用报告提供了适用于 64K word 的 F2810 和 128K word 的 F2811/F2812 的 cmd 文件。如 Figure 8 所示，F2810.cmd 包含在 AppCode 工程中，却排除在 build 过程之外，因为 F2810.cmd 文件的图标没有向下的小箭头。



【Figure 8 适用于 128Kw 的 AppCode 工程】

你可以为工程中每个文件指定 build 属性，例如排除在工程 build 之外等等。要设定文件的 build 属性，右键单击该文件，选择"File Specific Options"。取消"Exclude file from build"前面的勾，则 F2812.cmd 被包含在下一次工程 build 中(Example 12)，勾选此项，则被排除。只能从 F2812.cmd 和 F2810.cmd 中选一个，取决于你的应用软件存储器要求。



【Figure 9 将一个 cmd 文件排除在 build 之外】

在改变某个文件 build 属性之后，需要重新编译链接工程。

3.1.1.2 使用链接器填充未用的 Flash 地址区域

Figure 7 展示了 AppCode 从源代码到二进制文件的 build 过程。在链接阶段可以填充未用的 Flash 地址区域。链接器使用 cmd 文件定义目标处理器的存储区。随后它将 AppCode 的程序段和数据段放在这些存储区中。如果存储区没用完，可以使用一个填充值，保证所有的地址区域都被加载。Example 1 展示了填充值 0xFFFF 是如何被用来填充存储区 FLASHE，FLASHD 和 FLASHC 的未用区域的。注意 FLASHB 没有填充值。下一节将讨论 hex 转换器用 0xFFFF 填充 FLASHB 区。

Example 1. 用 0xFFFF 填充未用地址区域的 cmd 文件

```
MEMORY
{
PAGE 0:    /* Program Memory */
   RAML0        : origin = 0x008000, length = 0x001000     /* on-chip
RAM block L0 */
   FLASHE       : origin = 0x3E8000, length = 0x004000,  fill
=0xFFFF     /* on-chip FLASH */
   FLASHD       : origin = 0x3EC000, length = 0x004000,  fill
=0xFFFF     /* on-chip FLASH */
   FLASHC       : origin = 0x3F0000, length = 0x004000,  fill
=0xFFFF     /* on-chip FLASH */
   FLASHB       : origin = 0x3F4000, length = 0x002000     /* on-chip
FLASH */
```

Example 2 展示了链接 AppCode 产出的 MAP 文件。注意填充值 0xFFFF 在最右边列出，并且填充值与 cmd 文件中定义的存储区相关联。

FLASHB 区没有关联的填充值，MAP 文件显示出它有 0x2000 的地址区域未使用。它将在下一节中使用 hex 转换器填充。

Example 2. 用 0xFFFF 填充未用地址区域的 MAP 文件

```
******************************************************************************
             TMS320C2000 COFF Linker PC v4.1.0
******************************************************************************
OUTPUT FILE NAME:   <./Debug/AppCode.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 003ec000


MEMORY CONFIGURATION

                  name            origin    length      used    attr    fill
         ----------------------  --------  ---------  --------  ----  --------
PAGE  0:
RAML0                   00008000   00001000  00000086  RWIX
         FLASHE                   003e8000   00004000  00004000  RWIX  ffff
         FLASHD                   003ec000   00004000  00004000  RWIX  ffff
         FLASHC                   003f0000   00004000  00004000  RWIX  ffff
         FLASHB                   003f4000   00002000  00000000  RWIX
         FLASHA                   003f6000   00001f80  00001f80  RWIX  ffff
         CSM_RSVD                 003f7f80   00000076  00000076  RWIX
         BEGIN                    003f7ff6   00000002  00000002  RWIX
         CSM_PWL                  003f7ff8   00000008  00000008  RWIX
         ROM                      003ff000   00000fc0  00000000  RWIX
         RESET                    003fffc0   00000002  00000000  RWIX
         VECTORS                  003fffc2   0000003e  00000000  RWIX
```

3.1.1.3 使用 hex 转换器填充未使用 Flash 地址区域

hex 转换器(HEX2000)将链接器产出的 COFF 格式输出转换为 ASCII hex 文件。与链接器类似,此 ASCII hex 文件的格式可以由一个 cmd 文件控制。Example 3 展示了 AppCode 使用的 F2810 HEX2000 cmd 文件(AppCode_hex_2810.cmd)。HEX2000 的文档参见《TMS320C28x Assembly Language Tools User's Guide》(SPRU513)第 11 章。

Example 3. AppCode HEX2000 cmd 文件

```
AppCode.out

-map AppCode_hex.map
-o AppCode.hex
-m
-memwidth 16
-image

ROMS
{
 FLASH2810: origin = 0x3e8000, len = 0x10000, romwidth = 16, fill =
0xFFFF
}
```

在 Example 4 中，FLASHB 的填充值是 0xFFFF，其他 Flash 区由链接器填充，它们的填充值由$fillxxx 开头的标识符代表。

Example 4. 使用 0xFFFF 填充值的 hex 转换器 MAP 文件

```
*******************************************************************
***************
TMS320C2000 COFF/Hex
Converter                                                  v4.3.0
*******************************************************************
***************


INPUT FILE NAME: <AppCode.out>
OUTPUT FORMAT:   Motorola-S


PHYSICAL MEMORY PARAMETERS
   Default data width    :  16
   Default memory width :  16
   Default output width :   8
```

```
OUTPUT TRANSLATION MAP
-------------------------------------------------------------------
---------------
003e8000..003f7fff  Page=0  Memory Width=16  ROM
Width=16  "FLASH2810"
-------------------------------------------------------------------
---------------
   OUTPUT FILES: AppCode.hex [b0..b15]


   CONTENTS: 003e8000..003e80ff   .econst Data Width=2
             003e8100..003ebfff   $fill000 Data Width=2
             003ec000..003ec3b9   .text Data Width=2
             003ec3ba..003ec43f   ramfuncs Data Width=2
             003ec440..003ec458   .cinit Data Width=2
             003ec459..003effff   $fill001 Data Width=2
             003f0000..003f3fff   $fill002 Data Width=2
             003f4000..003f5fff   FILL = 0000ffff
             003f6000..003f7f7f   $fill003 Data Width=2
             003f7f80..003f7ff5   csm_rsvd Data Width=2
             003f7ff6..003f7ff7   codestart Data Width=2
             003f7ff8..003f7fff   csmpasswds Data Width=2
```

3.1.2 创建 AppCode 二进制文件

至此，所有未使用的 Flash 地址区域已经被填充了。本节将阐述如何创建适于 CKFA SCI 传输与 Flash 烧写的 AppCode 二进制文件，见 Figure 7。

本应用报告已经使用了 HEX2000 产出的 1 个 ASCII 格式文件，但有 2 个缺点：

- Flash 烧写需要的每 8bit 二进制数据，在 ASCII 文件中要占用 16bit
- 这个转换（16bit 转换为 8bit）将使 Flash 烧写时间延长

FileIOShell.exe 用于将 HEX2000 产出的 ASCII 格式转换为二进制格式。此软件被设计为将 Motorola-S 记录格式（16bit，big endian）转换为二进制文件格式。关于 Motorola-S 记录格式参见附录 D。

Example 3 展示了 HEX2000 根据 AppCode_hex_2810.cmd 生成 Motorola-S 格式的 ASCII 文件。AppCode_hex_2810.cmd 的语法如下：

- AppCode.out　　=COFF 可执行输入文件
- -map AppCode_hex.map　　=HEX2000 MAP 产出文件（其内容见 Example 2）
- -o AppCode.hex　　=产出的 hex 文件
- -m　　=指定 Motorola-S 记录格式
- -memwidth 16　　=16bit 存储位宽，big endian
- -image　　=允许使用 fill 参数

产出文件是AppCode.hex,它将被FileIOShell.exe转换为二进制格式,从而适于CKFASCI传输和Flash烧写。

Example 5. AppCode.hex(ASCII文件阅读器),Mot-S格式,FileIOShell.exe的输入文件

```
S00600004844521B
S2223E8000C1F5003EC1F5003EC1F5003EC1F5003EC1F5003EC1F5003EC1F5003
EC1F5BD
S2223E800F003EC1F5003EC1F5003EC1F5003EC1F5003EC1F5003EC0B0003EC0B
5003EAD
S2223E801EC0BA003EC0BF003EC0C4003EC0C9003EC0CE003EC0D3003EC0D8003
EC0DDF3
```

Example 6. AppCode.bin(二进制文件阅读器),二进制格式,FileIOShell.exe的输出文件

```
C1 F5 00 3E C1 F5 00 3E C1 F5 00 3E C1 F5 00 3E
C1 F5 00 3E C1 F5 00 3E C1 F5 00 3E C1 F5 00 3E
C1 F5 00 3E C1 F5 00 3E C1 F5 00 3E C1 F5 00 3E
C1 F5 00 3E C0 B0 00 3E C0 B5 00 3E C0 BA 00 3E
C0 BF 00 3E C0 C4 00 3E C0 C9 00 3E C0 CE 00 3E
C0 D3 00 3E C0 D8 00 3E C0 DD
```

3.1.2.1 从CCS生成AppCode.bin

在AppCode工程中,build配置为每次build时调用AppCode_COFF2BIN_281x.bat。这种批处理文件有2个,一个适用于64K word Flash的F2810(AppCode_COFF2BIN_2810.bat),一个适用于128K word Flash的F2811/F2812(AppCode_COFF2BIN_2812.bat)。这2个文件基本相同,不同的地方在于为HEX2000选择不同的cmd文件,以定义2种Flash范围。

Example 7. AppCode_COFF2BIN_2810.bat

```
cd debug

C:\CCStudio_v3.1\C2000\cgtools\bin\hex2000.exe
AppCode_hex_2810.cmd

FileIOShell.exe -i AppCode.hex -o AppCode.bin
```

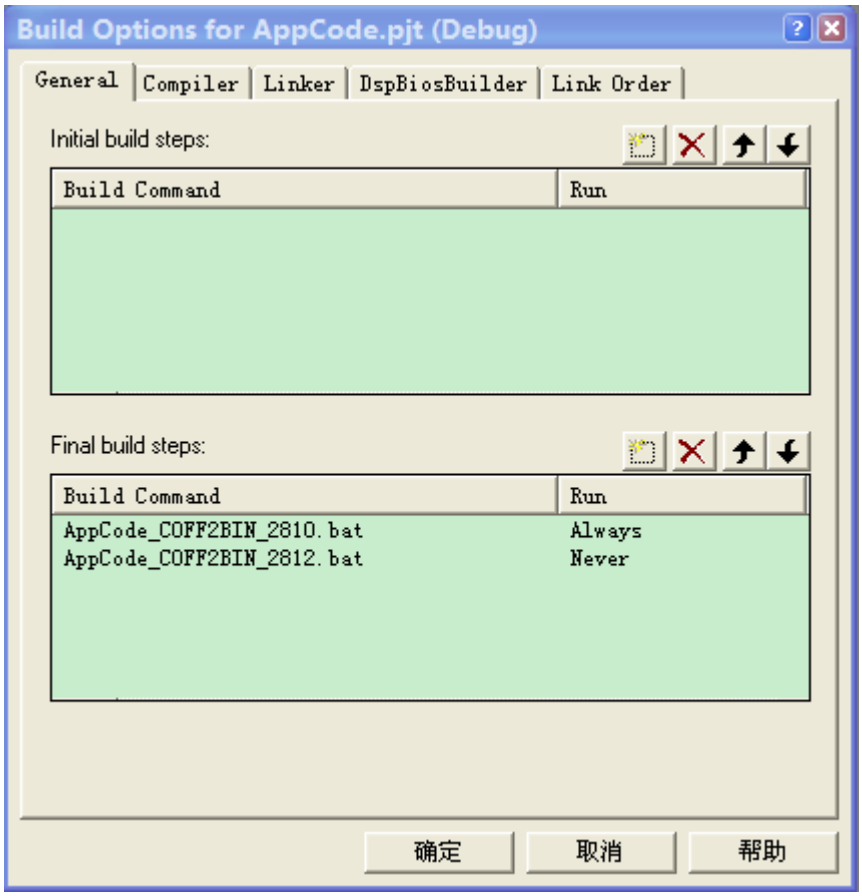
Example 8. AppCode_COFF2BIN_2812.bat

```
cd debug

C:\CCStudio_v3.1\C2000\cgtools\bin\hex2000.exe
AppCode_hex_2812.cmd
```

```
FileIOShell.exe -i AppCode.hex -o AppCode.bin
```

配置 CCS 工程 build 选项，以执行与工程存储区相关的批处理文件。



【Figure 10 AppCode CCS 工程-配置 COFF2BIN 批处理文件】

3.1.2.2 不借助 CCS 生成 AppCode.bin

如果不使用 CCS，则需将 COFF 或 Motorola-S 记录文件转换为适于 CKFA SCI 传输和 Flash 烧写的二进制文件。如果是 COFF 文件，则使用 AppCode_COFF2BIN_2810.bat 或 AppCode_COFF2BIN_2812.bat。如果是 Motorola-S 记录文件，则使用 FileIOShell Only.bat 文件。

Example 9. FileIOShell Only.bat 文件

```
FileIOShell.exe -i AppCode.hex -o AppCode.bin
```

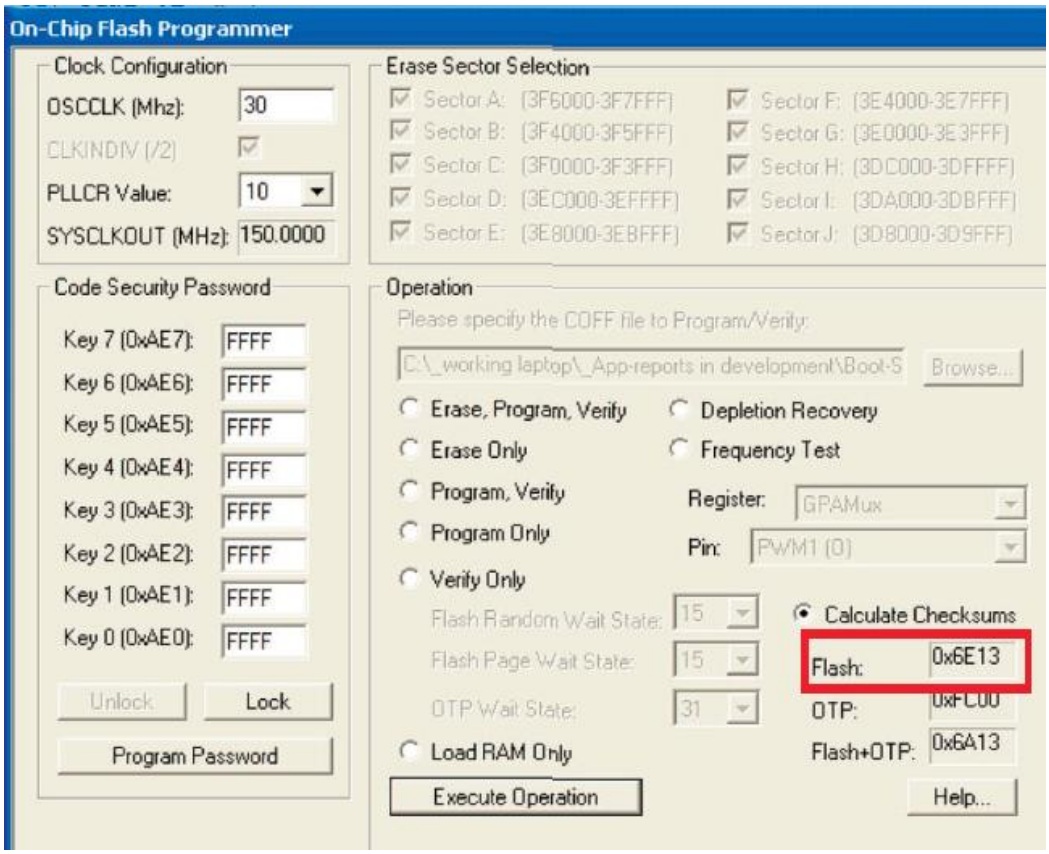
3.1.3 计算应用软件的期望校验和

应用代码的校验和可以用 CCS 的 Flash 烧写器插件计算出来。CKFA 软件使用校验和检查应用代码是否正确的烧写到 Flash 中。

3.1.3.1 使用 CCS 计算校验和

烧写 Flash 至少一次之后，就可以使用 CCS 片内 Flash 烧写器计算校验和。Flash 被主机 PC 通过 JTAG 扫描，将存储区内每个 16bit 数据相加，得到总和。在 Figure 11 中，片内 Flash 烧写器计算出应用软件的校验和为 0x6E13。

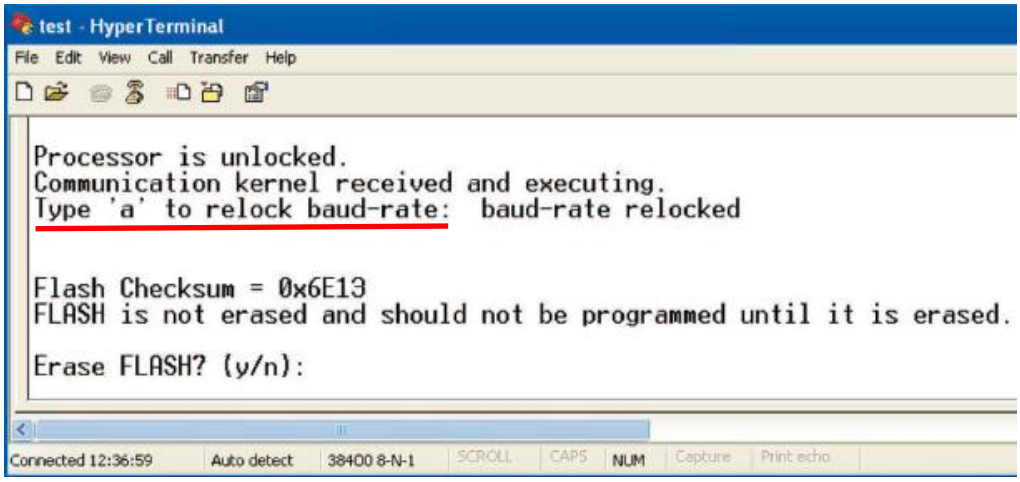
计算出校验和后，要把它加入 CKFA 软件中，以便在烧写完应用代码后进行核对。



【Figure 11 CCS 的 On-Chip Flash Programmer 计算校验和】

3.1.3.2 不借助 CCS 计算校验和

CKFA 在执行开始时计算 Flash 校验和，以确定 Flash 是否已擦除过。这个过程不借助 CCS 就计算出 AppCode 校验和。AppCode 必须通过 CKFA 烧写到 Flash。烧写完成后，如果计算的校验和与期望校验和不同，CKFA 会报告一个校验和错误，并且将计算校验和发出来。这个值是正确的 AppCode 校验和，可以把它包含在 CKFA 软件中，用于下一次 Flash 烧写后的核对。



【Figure 12 CKFA 在启动时计算 AppCode 的校验和】

3.2 准备 CKFA

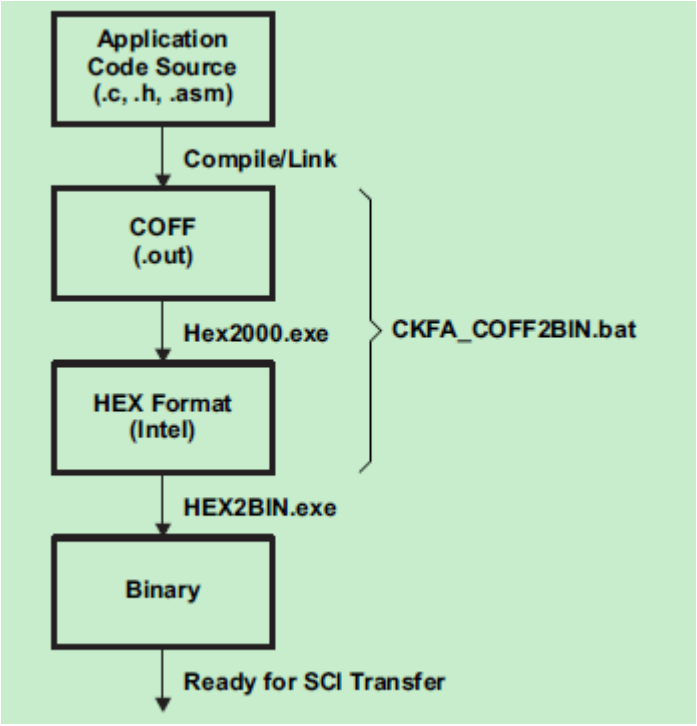
CKFA 代码包含通讯内核与 Flash API。先前已提及，CKFA 是通过 F281x SCI-A 引导代码传送到 F281x RAM 中的。传送完成后，SCI-A 引导代码将 CPU 控制权交给 CKFA 代码，CKFA 随后将应用代码传送到 RAM 中，然后将其烧写到 Flash。要完成这些，步骤如下：

1．配置 F281x PLL，调整 CPU 时钟频率
F2812 eZdsp 使用 30MHz 晶振，因此，PLL 配置为 150MHz（最大值）
注意：如在《TMS320x281x Boot ROM Reference Guide》（SPRU095）中所述，引导 ROM 不改变 PLL，因此，根据 F281x 是被 CCS 软件重启还是重新上电，PLL 的设置是不一样的。

2．用 AppCode 密码更新 CKFA，以解锁 F281x 的 CSM 模块
CKFA 软件要求 CSM 模块是解锁的。TI 的芯片出厂时，片内 Flash 是完全擦除过的，CSM 也是解锁的。因此第一次烧写不要求解锁过程。关于 CSM 的更多信息，参见《TMS320x DSP System Control and Interrupts Reference Guide》(SPRU078)。

3．配置 CKFA 适用于 64KW 或 128KW 的 Flash

4．为 SCI 传送创建一个二进制文件



【Figure 13 CKFA 文件处理概览】

3.2.1 将 CKFA 软件中的 AppCode 校验和更新为期望值

期望的 AppCode 校验和保存在 Example_Flash281x_API.c 中。计算好一个新的校验和后，用该校验和更新 CHECKSUM_EXPECTED 常量定义。CKFA 软件用这个值和它自己计算的校验和相比较。更新校验和后，需要重新编译 CKFA 软件，生成新的 CKFA 二进制文件。

Example 10. Example_Flash281x_API.c 中 CKFA 使用的期望校验和

```
#define CHECKSUM_EXPECTED 0x6E13
```

3.2.2 为 F281x 目标板晶振频率配置 CKFA 软件

Example_Flash281x_API.h 中，常量 PLLCR_VALUE 定义 PLL 倍频设置。Flash281x_API_Config.h 中，常量 CPU_RATE 定义系统的 CPU 频率设置。Flash 的时间参数基于这些设置，如果设置的不正确，Flash 可能损坏。

检查上述文件的内容，确认 PLLCR_VALUE 和 CPU_RATE 符合你的 F281x 目标板，如果不符合，修正设置值，将 CKFA 软件重新编译、链接和生成二进制文件。

Example 11. Example_Flash281x_API.h 中定义的 PLL 设置

```
/*-----------------------------------------------------------------
--------------
     Specify the PLLCR (PLL Control Register) value.
```

```
      Uncomment the appropriate line by removing the leading double
slash: //
      Only one statement should be uncommented.


      The user's application must set the PLLCR Register before calling
any
      of the Flash API functions.

      Example:  CLKIN is a 30MHz crystal.
                The user wants to have a 150Mhz CPU clock (SYSCLKOUT
= 150MHz).
                In this case, PLLCR must be set to 10 (0x000A)
                Uncomment the line: #define PLLCR_VALUE 10
                Comment out the remaining lines with a double slash:
//
------------------------------------------------------------------
------------*/


#define PLLCR_VALUE   0x000A     // SYSCLKOUT = (OSCLK*10)/2
// #define PLLCR_VALUE   0x0009     // SYSCLKOUT = (OSCLK*9)/2
// #define PLLCR_VALUE   0x0008     // SYSCLKOUT = (OSCLK*8)/2
```

Example 12. Flash281x_API_Config.h 中定义的 CPU_RATE 设置

```
/*------------------------------------------------------------------
--------------
   2. Specify the clock rate of the CPU (SYSCLKOUT) in nS.


      Take into account the input clock frequency and the PLL
multiplier
      your system will use.

      Use one of the values provided, or define your own.
      The trailing L is required tells the compiler to treat
      the number as a 64-bit value.


      Only one statement should be uncommented.
```

```
      Example:  CLKIN is a 30MHz crystal. The PLL is enabled.

                If your application will set PLLCR = 0xA then the CPU
clock
                will be 150Mhz CPU (SYSCLKOUT = 150MHz).


                In this case, the CPU_RATE will be 6.667L
                Uncomment the line:  #define CPU_RATE  6.667L
-----------------------------------------------------------------
------------*/


#define CPU_RATE    6.667L   // for a 150MHz CPU clock speed
(SYSCLKOUT)
//#define CPU_RATE    7.143L   // for a 140MHz CPU clock speed
(SYSCLKOUT)
//#define CPU_RATE    8.333L   // for a 120MHz CPU clock speed
(SYSCLKOUT)
```

3.2.3 正确配置 CKFA 中的 Flash 范围（64KW 或 128KW）

在 Flash281x_API_Config.h 中指定一个设备。通过把设备宏定义为 1 来选择 F2810、F2811 还是 F2812。

Example 13. 为 CKFA 指定 Flash 范围是 64KW 还是 128KW

```
Setting for using the F2810 (Flash281x_API_Config.h)

#define FLASH_F2810   1
#define FLASH_F2811   0
#define FLASH_F2812   0

Setting for using the F2811 (Flash281x_API_Config.h)

#define FLASH_F2810   0
#define FLASH_F2811   1
#define FLASH_F2812   0
```

3.2.4 创建 CKFA 二进制文件

配置好 CKFA 的频率、密码和 Flash 范围后，必须为引导 ROM SCI-A 传输创建 CKFA 的二进制文件。

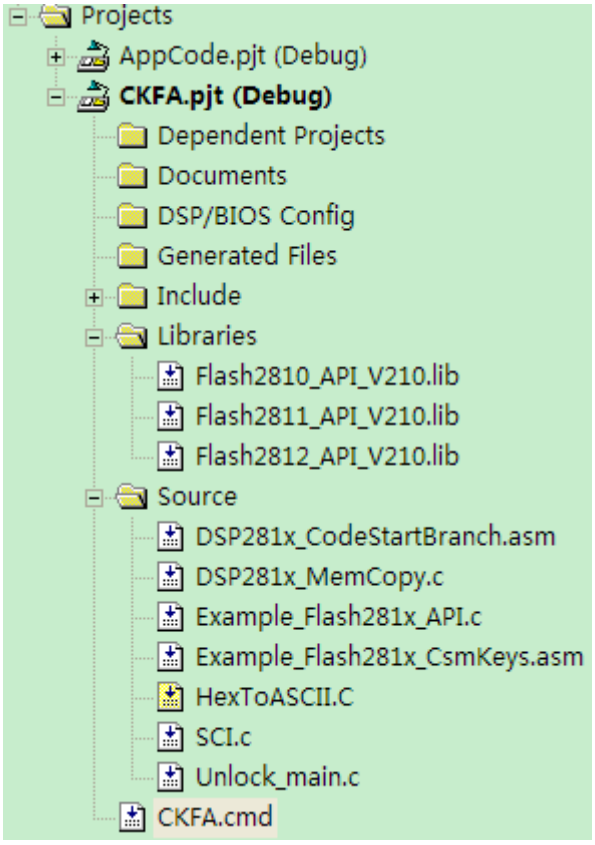
3.2.4.1 使用 CCS 重新 build CKFA 软件

在对 CKFA 软件做修改后，使用本应用报告包括的 CCS 工程重新 build 之。CCS 工程包括额外的 build 步骤，来将链接产出的文件转换为适于 SCI-A bootloader 的二进制文件。注意，在每次重新 build CKFA 软件或你自己的应用软件时，必须使用本文包含的 CCS 工程。

从 CCS 中打开 CKFA 工程。分别修改 Example_Flash281x_API.h 和 Flash281x_API_Config.h 中的 PLLCR_VALUE 和 CPU_RATE，从 Project 菜单选择 Rebuild All。

3.2.4.2 CKFA CCS 工程细节

CKFA 软件的 CCS 工程包含如下文件（Figure 14）

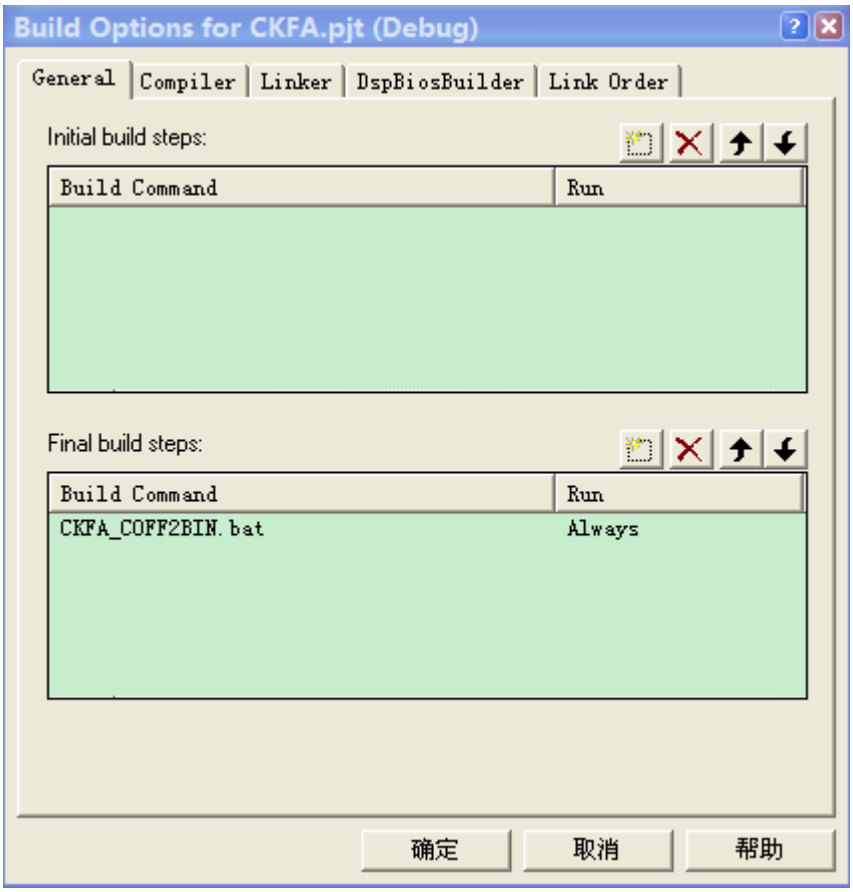


【Figure 14】

注意 CKFA 软件是基于 Flash API 样例代码的。它们的区别是，Flash API 样例代码将代码传送到 RAM 不是通过 SCI-A 引导操作，而是直接拷贝到 RAM 中的，因为 Flash API 已经固化在 F281x Flash 中了。这是现场编程的一种典型方案。本文描述的 SCI-A 引导方式是出厂前产品编程的典型方案。

3.2.4.3 从 CCS 生成 CKFA.bin

CKFA 工程 build 选项里的 Final build steps 是调用一个批处理文件将 build 产出的 COFF 可执行文件转换为适于 SCI 通讯的二进制格式。（Figure 15）



【Figure 15】

F281x SCI-A 引导方式要求传输的文件是二进制格式的，不能传输 ASCII-Hex 格式。要把 COFF 可执行格式转换为二进制格式，首先要把 COFF 格式通过 TI hex 转换工具转换为 ASCII-Hex 格式。本应用报告使用 Intel 风格 ASCII-Hex 格式。

批处理文件 CKFA_COFF2BIN.bat 内容如下。它调用 C2000 HEX 转换工具，将链接器产出的 COFF 可执行文件转换为 ASCII-Hex（Intel）格式。关于 HEX 转换工具的更多详细信息，参见《TMS320C28x Assembly Language Tools User's Guide》(SPRU513)。

Example 14. CKFA_COFF2BIN.bat

```
cd debug

C:\CCStudio_v3.1\C2000\cgtools\bin\hex2000.exe CKFA_hex.cmd

HEX2BIN CKFA.hex
```

CKFA_hex.cmd 中含有 HEX2000 转换工具的命令行指令。这个过程的关键产出是 CKFA.hex 文件，它作为 HEX2BIN 转换器的输入，生成二进制文件。

- CKFA.out = COFF 可执行格式输入文件
- -map CKFA_hex.map = HEX2000 MAP 产出文件
- -o CKFA.hex = 产出的 Hex 文件
- -I = 指定 Intel 记录格式

Example 15. CKFA_hex.cmd

```
CKFA.out


-boot
-sci8
-map CKFA_hex.map
-o CKFA.hex
-I
```

文件变成 ASCII-Hex 格式后，就可使用任一容易获取的二进制转换工具进行转换。在本应用报告中将 Intel hex 格式转换为二进制格式的工具是 HEX2BIN（http://gnuwin32.sourceforge.net/）。

CKFA.bin 遵循 SCI-A 引导加载选项要求。8bit SCI 数据流格式参见 F.3 章节。

Example 16. CKFA.bin（二进制阅读器），HEX2BIN.exe 的二进制产出

```
AA 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 F2 02 6B
01 00 00 00 01 04
FE 1F 76 C1 01 1A 1A 00 40 1A 18 FF DF 1A 1A 00 20 00 8F 6C 02 40 76
37 01 00 8F 6C 02

KeyValue = 0x08AA
Reserved = 0x0000 ...
Entry Point = 0x000002F2
Block #1 Size = 0x016B
Block #1 Destination = 0x00000100
```

3.3 用引导 ROM 的 SCI-A 代码建立通讯

下面的步骤是必需的：用 F281x 的引导 ROM 代码使能 SCI 通讯。

3.3.1 配置 F281x 目标板的 SCI-A 引导方式

重启后，F281x 扫描 4 个 GPIO 口，以确定引导方式。默认模式是 Jump to FLASH，也就是这 4 个 GPIO 口都悬空。4 个 GPIO 口电平对应的引导方式参见 Table 1。GPIOF4 有一个内部上拉电阻，通过它可以将 Jump to FLASH 设为默认。

Table 1. 引导模式 GPIO 引脚

引导模式 GPIOF4(SCITXDA)	GPIOF12(MDXA)	GPIOF3(SPISTEA)	GPIOF2(SPICLK)
GPIO 有无内部上拉 有	无	无	无
跳转到 Flash/ROM 地址 0x3F7FF6 1	X	X	X
（在重启之前这里需烧写一个跳转指令，以执行期望的重定向代码） 调用 SPI_Boot 从外部串行 SPI EEPROM 加载(1)			
0	1	X	X
调用 SCI_Boot 加载 SCI-A 0	0	1	1
跳转到 H0 SARAM 地址 0x3F8000(2) 0	0	1	0
跳转到 OTP 地址 0x3D7800 0	0	0	1
调用 Parallel_Boot 从 GPIO B 口加载 0	0	0	0

(1) 必须注意 SPICLK 翻转效应引起的外部逻辑，可能使选择引导模式出错
(2) 如果选择的引导模式是 Flash，H0，或 OTP，则 bootloader 不加载任何外部代码。

Table 2 给出本应用报告中使用的 F2812 eZdsp 开发板选择 SCI_Boot 方式的跳线设置（略）

3.3.2 配置通讯串口硬件

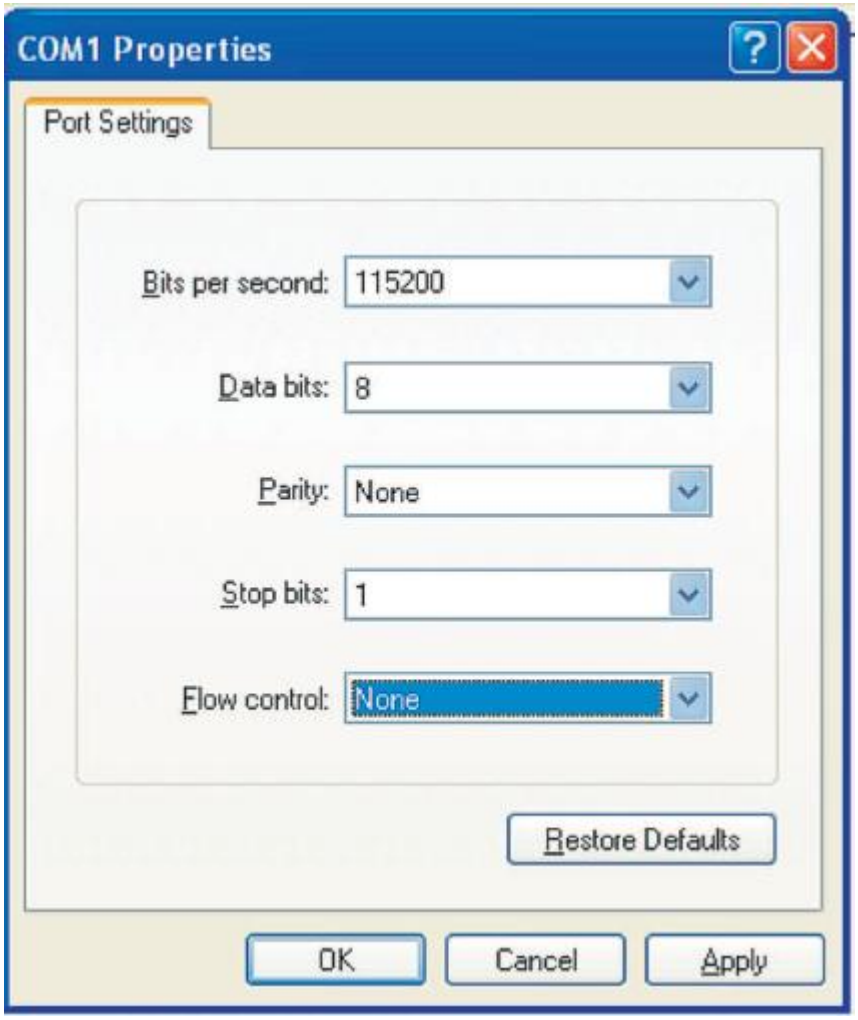
F2812 eZdsp 开发板本身没有 RS-232 收发器，本应用报告使用 Link Research 公司的 LR-2812COM 接口板直接与 F2812 eZdsp 开发板相连，同时也提供了与 PC 的 RS-232 接口。Link Research 公司的 LR-2812COM 接口板详细产品信息可从 http://www.link-research.com/下载。

3.3.3 配置 PC 串口软件

本报告中使用的 PC 端串口通讯软件是超级终端。首先配置通讯格式与引导 ROM SCI-A 引导方式相同，开始时波特率要设的低一些，例如 9600 bps，然后再增加到 38400 或 57600 bps。这可以避免一开始就遇到通讯问题。

Figure 16 给出了 COM1 串行口的配置：
- 115200 bps（推荐从 9600 bps 开始）

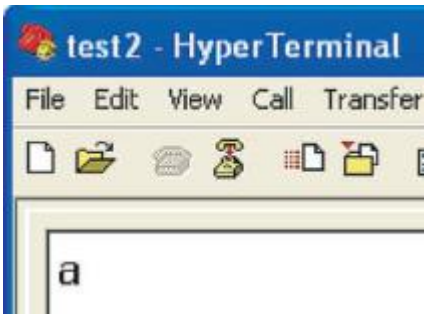
- 8-bit 数据位
- 无校验
- 1 位停止位
- 无流控制



【Figure 16. 超级终端通讯配置】

配置好超级终端以后，给 F281x 开发板下电后再上电，这样就制造了一个重启，F281x 引导 ROM 会扫描 4 个引导 GPIO 口。如果已经配置为 SCI-A 引导模式，F281x 将开始执行引导 ROM 中的 SCI_Boot 代码。

SCI_Boot 代码首先使用 SCI 端口的自动波特率特性配置 SCI-A 口的波特率。要将 F281x SCI-A 的波特率与超级终端的锁定，敲一个'a'或'A'，这是 SCI 的自动波特率特性所要求的。见《TMS320x281x Boot ROM Reference Guide》(SPRU095)中引导 ROM 代码流程图。引导 ROM 将字符回送到超级终端，则波特率就配置好了（见 Figure 17）。自动波特率功能文档见《TMS320x28xx, 28xxx DSP Serial Communication Interface (SCI) Reference Guide》(SPRU051)。



【Figure 17. 从 F2812 SCI 自动波特率逻辑发回的字符】

如果键入的字符没有发回，检查：

- 硬件是否连接到位
- F2812 目标板有没有下电再上电
- F2812 引导 GPIO 引脚（GPIOF4,F12,F3,F2）是否配置为 SCI 引导模式
- 超级终端设置是否正确

如果上述各项都检查过了，则降低超级终端的波特率设置，重新开始锁定过程。

3.4 Flash 烧写过程

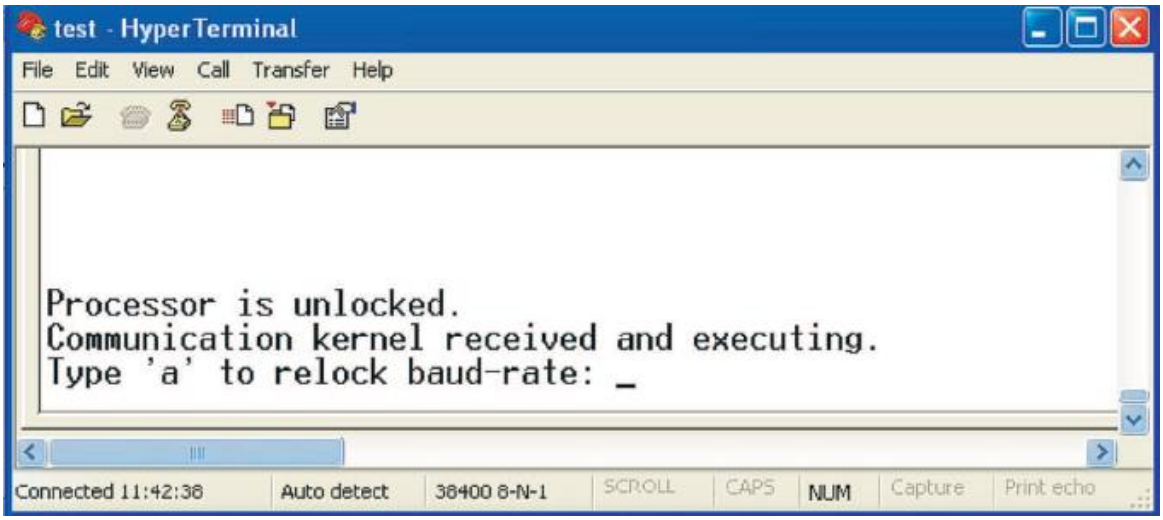
应用代码和 CKFA 软件准备好，并且建立好利用引导 ROM SCI 代码进行的串口通讯之后，就可以开始 Flash 烧写过程了。Flash 烧写包括 CKFA 软件传送、解锁 SCM、应用代码传送、应用代码校验。

3.4.1 传送 CKFA 软件

设定目标板的波特率，准备传送 CKFA。在超级终端上，点击“传送”—“发送文本文件”，从 CKFA 工程 Debug 文件夹中选择 CKFA.bin。

SCI 引导 ROM 代码会将发送的字符回送给超级终端，这可以用来检验是否发送成功。本应用报告采用的方法不是检验每个发送的字符，而是检验烧写后的 Flash 计算的校验和。

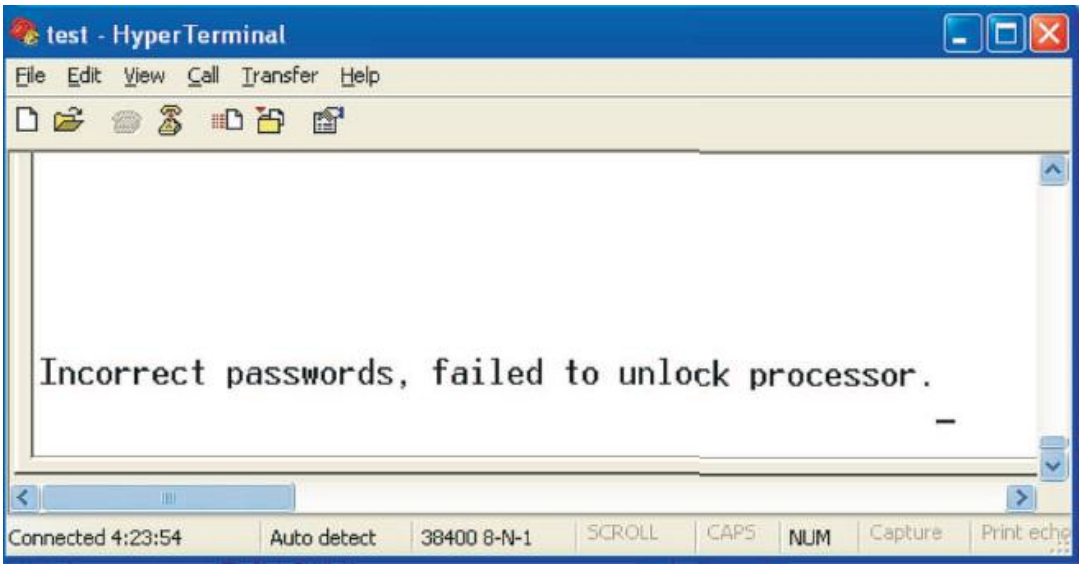
在 Figure 18 中，超级终端显示出 CKFA 代码已传送到 RAM 中并执行，CSM 也已解锁。



【Figure 18. 超级终端-CKFA 软件准备更新 F281x 波特率】

3.4.2 解锁 CSM

如果 CSM 被锁住并且 CKFA 软件中的解锁密码不对，超级终端会如 Figure 19 所示。更正 Example_Flash281x_CsmKeys.asm 中的解锁密码，重新编译 CKFA 软件，复位 DSP，重新传送 CKFA。



【Figure 19. 超级终端-F281x 上锁引起的 CKFA 传送失败】

3.4.3 使用 CKFA 接口准备目标

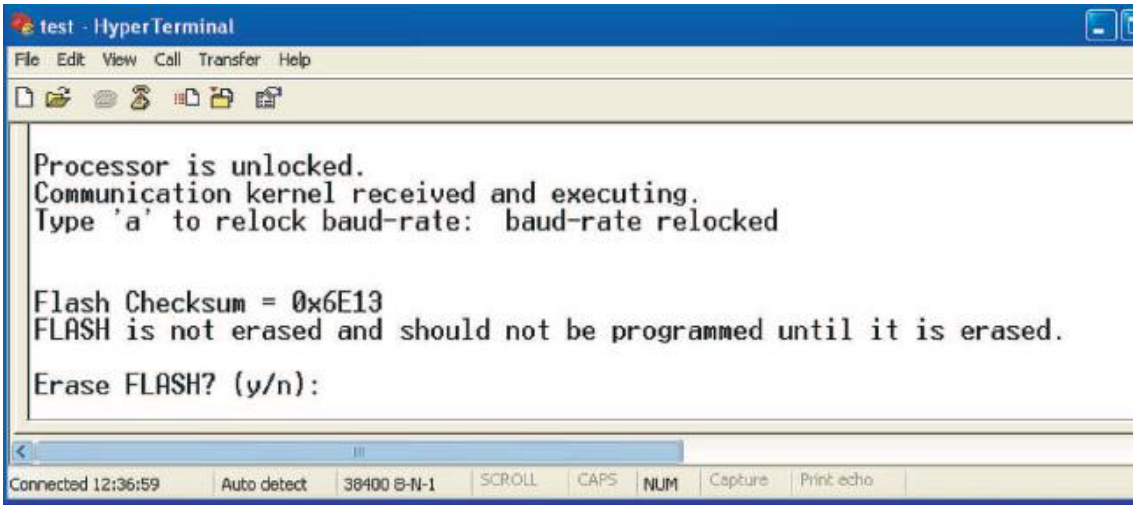
CSM 解锁后 CKFA 软件开始执行。CKFA 软件使能并配置 PLL。在本应用报告中，DSP 系统频率为 150MHz。如果频率有变，CKFA 软件需更新 SCI-A 波特率。键入'a'或'A'重新使

目标板与超级终端波特率锁定，如 Figure 20 所示。如果无法锁定波特率，降低超级终端的波特率然后重试。

波特率更新后，CKFA 软件计算 Flash 校验和并通过 SCI-A 发送。你可以利用该校验和确定：

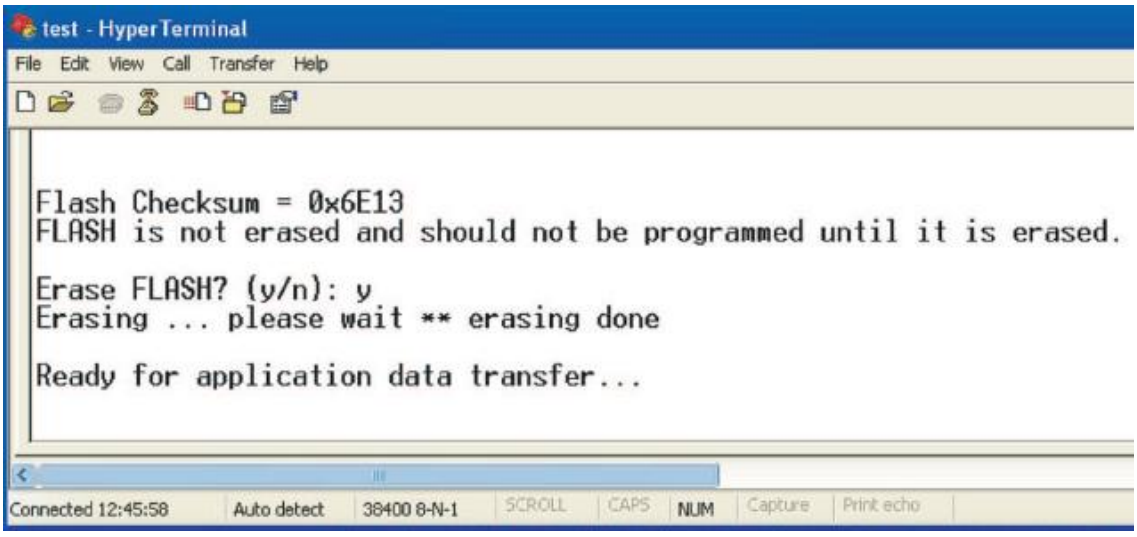
- 如果 Flash 已擦除，校验和应该是 0x0000。擦除步骤可忽略。
- 如果校验和与应用软件的期望校验和相同，则烧写成功。
- 如果校验和与期望值不同，则 Flash 需擦除，你必须对提示 Erase FLASH?回答 yes('y')（Figure 20）。

F281x 出厂时 TI 已经擦除了 Flash，因此你可以直接烧写 Flash。



【Figure 20. 超级终端-CKFA 校验和检测出 Flash 尚未擦除】

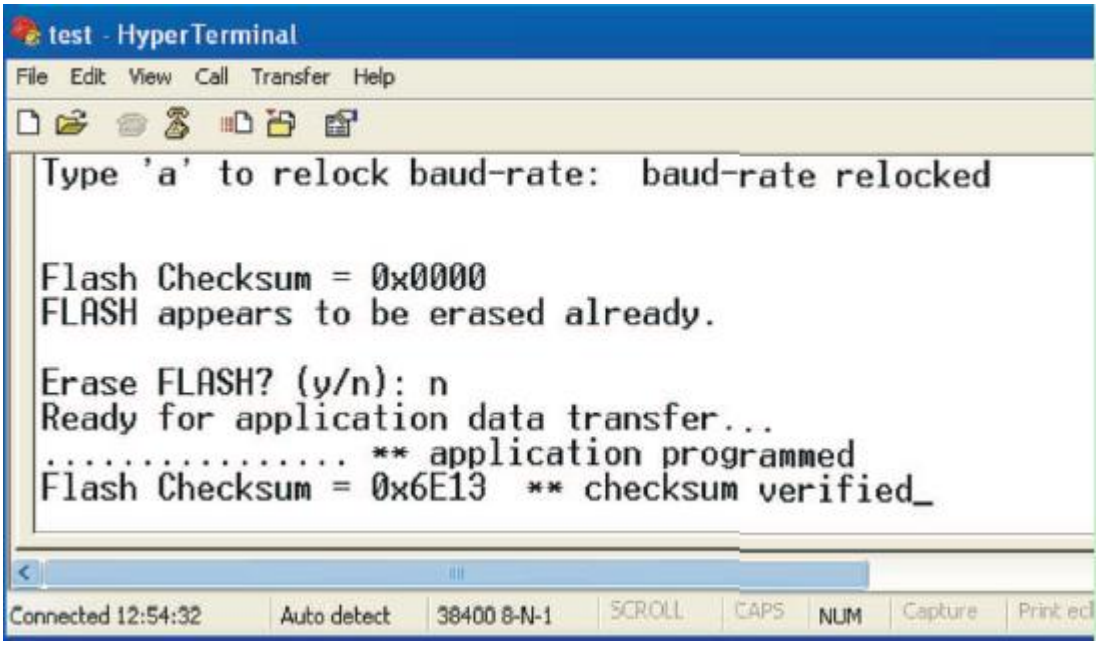
如果正在擦除，注意：擦除过程不能被打断。等待擦除过程结束。Flash 擦除后你会收到 erasing done。此时，CKFA 软件已经准备好接受应用代码（见 Figure 21）。



【Figure 21. 超级终端-CKFA 软件准备好接受和烧写应用代码】

3.4.4 传送应用代码

要利用超级终端传送应用软件，点击“传送”—“发送文本文件”，从 AppCode 工程 Debug 文件夹中选择 AppCode.bin。应用代码传送并烧写完成后，CKFA 软件会计算 Flash 校验和。校验和通过 SCI-A 发送并与 Example_Flash281x_API.c 中保存的值相比较。如果相同，则结果会通过 SCI-A 发送回来，告知用户校验和验证通过（Figure 22）。



【Figure 22. 超级终端-CKFA 软件已经传送并烧写了应用代码】

3.4.5 确认应用代码执行

F281x 现在准备好从 Flash 执行应用代码。

1. 给 F281x 目标板下电。
2. 将引导 GPIO 配置为 Jump to Flash 模式。
3. 给 F281x 目标板上电，则应用代码会从 Flash 执行。

4. Flash 烧写加速

本应用报告也致力于减少烧写时间，基于此，本报告的方法论考虑的是：

- 轮流利用 2 个 4KW 缓冲区，使应用代码的传送得以持续不断
- 消除 AppCode.bin 中的无关数据带来的开销
- 在烧写前检查 Flash，如果没必要擦除，就跳过擦除步骤
- 所有未用存储区填入 0xFFFF
- 通过 CKFA 代码设置 PLL，最大化 SCI-A 波特率

在《TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, TMS320C2812 Data Manual》(SPRS174)中，列有典型烧写时间：对 16KW 扇区是 500ms，

对 8KW 扇区是 250ms。F2810 有 3 个 16KW 和 2 个 8KW 扇区，因此，整个 64KW Flash 的典型烧写时间是 2s。

增大波特率可以显著减少烧写时间。Table 3 给出了 Flash 烧写时间，参考《TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, TMS320C2812 Data Manual》(SPRS174)。
【Table 3】

4.1 PC 到 F281x 目标板

根据超级终端的计时器，RS-232 波特率设置为 38400bps 时，烧写 64KW 的应用代码耗时 37 秒。波特率提高到 57600bps 时，烧写时间减少到 24 秒。

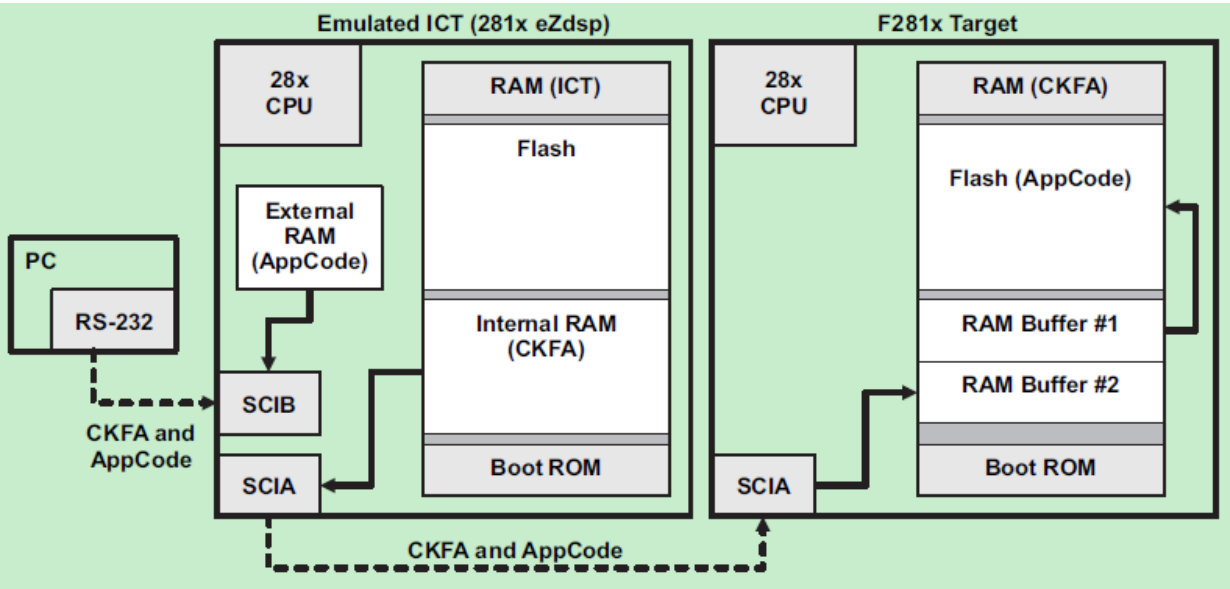
4.2 ICT 到 F281x 目标板

如果直接连接到 F281x，波特率可以增加很多；SCI-A 既能收也能发。RS-232 的传送带宽是被限制的，显著延长了烧写时间。在本应用报告中，使用仿真 ICT(EICT)硬件，烧写 64KW 的 AppCode 只用了 1.4 秒。

4.2.1 方法论

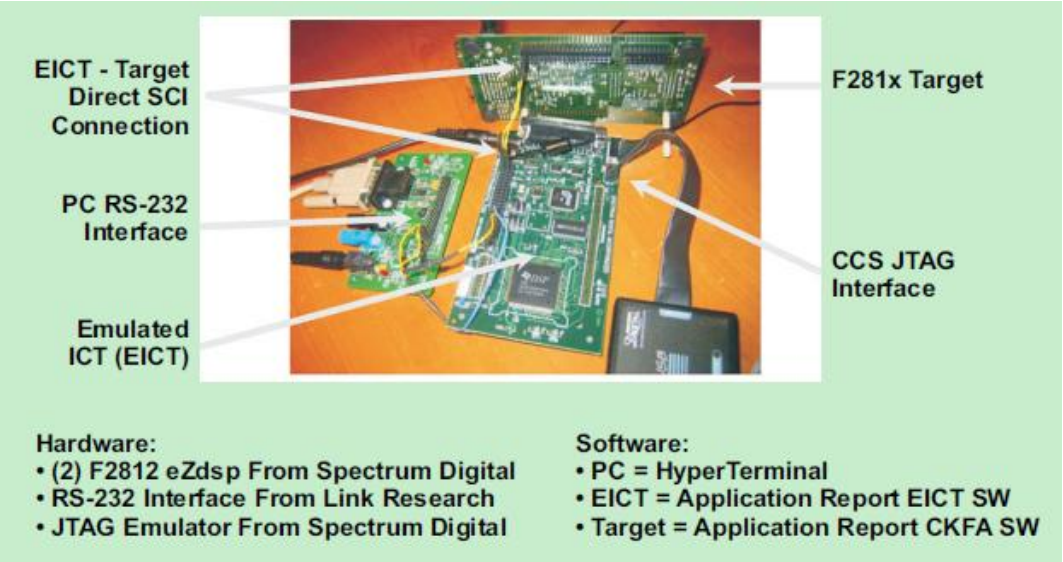
用 PC 将 CKFA 和 AppCode 二进制文件传送到 EICT 的 RAM 中。F2812 eZdsp 代表 EICT。CKFA 二进制文件存储在 eZdsp 的内部 RAM 中。AppCode 二进制文件存储在 F2812 eZdsp 中。F2812 eZdsp 有 64KW 的外部 RAM。

从 PC 到 EICT 的传输是通过 RS-232 和超级终端进行的，速率相对来说较慢。



【Figure 23】

从 EICT 到 F281x 目标板的传输较快，因为不使用 RS-232 收发器。2 块板子上 DSP 的 SCI 引脚直接相连，就像在产线上一样。CKFA 控制 F281x 目标板及其 PLL 设置。以上这些使得 EICT 与 F281x 目标板的传输波特率可达 1.875Mbps。



【Figure 24】

4.2.2 Flash 烧写加速

理解 Flash 烧写过程后，让我们看看利用本应用报告中提到的技术烧写 Flash 需要花多长时间。串行传送 64KW 或 128KW 应用代码时，烧写时间与波特率直接相关。

4.2.2.1 从 EICT 向 F281x 目标板传送 CKFA 的波特率设置

目标板引导 ROM 代码决定了 CKFA 从 EICT 传送到 F281x 的波特率，因为引导 ROM 在重启时不使能 PLL。这样，目标板的 CPUCLK 取决于输入晶振频率，对 F2812 eZdsp 来说是 30MHz。

重启后，低速外设时钟 (LSPCLK) 默认为 CPUCLK/4，引导 ROM 代码不会改变 LSPCLK。因此 LSPCLK 为 7.5MHz。

SCI 波特率寄存器最小值是 1。因此重启时 F281x 目标板最大波特率是 468Kbps。

以 468Kpbs 波特率传送 CKFA 要比通过 RS-232 以典型 PC（超级终端）波特率—38Kbps 或 56Kbps—传送要快的多。另外，相对于 AppCode 动辄 128KB 或 256KB 的巨大体积，CKFA 二进制文件要小的多（6.4KB）。因此关键是增加传送 AppCode 的波特率。

EICT:
CPUCLK = 150MHz
LSPCLK = 150MHz/2 = 75MHz

BRR=19
Baud-Rate = LSPCLK/((BRR+1)*8) = 468750 bps

Target:
CPUCLK = 30MHz 【重启时 PLL 被旁路，引导 ROM 不使能 PLL】
LSPCLK = 30MHz/4 = 7.5MHz 【重启后默认除以 4】
BRR=1 【由 SCI 自动波特率特性设置】
Baud-Rate = LSPCLK/((BRR+1)*8) = 468750 bps

4.2.2.2 从 EICT 向 F281x 目标板传送 AppCode 的波特率设置

CKFA 代码控制 F281x 目标板，因此 SCI 波特率可以配置为外部链接的硬件所能支持的最大值。F281x SCI 最大波特率为 20Mbps，受限于 F281x IO 缓冲器速度。

CKFA 软件配置 PLL 为×5，因此 CPUCLK 为 150MHz。

CKFA 软件配置 LSPCLK=CPUCLK/2，因此 LSPCLK 为 75MHz.

本应用报告中使用的硬件的 SCI 波特率最小值为 4，对应于 F2812 eZdsp 目标板最大波特率 1.875Mbps。曾测试过 BRR 值为 3（2.34Mpbs），但引起了串行通讯错误。

EICT:
CPUCLK = 150MHz
LSPCLK = 150MHz/2 = 75MHz
BRR=4
Baud-Rate = LSPCLK/((BRR+1)*8) = 1.875 Mbps

Target:
CPUCLK = 150MHz 【PLL 被 CKFA 使能】
LSPCLK = 150MHz/2 = 75MHz 【CKFA 设置为除以 2】
BRR=4 【由 SCI 自动波特率特性设置】
Baud-Rate = LSPCLK/((BRR+1)*8) = 1.875 Mbps

4.2.3 ICT Flash 烧写过程

使用 ICT，波特率就可以快的多。在前面的章节中，使用 PC 进行 Flash 烧写时，RS-232 收发器会进行速度限制。现在，将处理器的串行引脚直接相连，就没有限制了。基于 ICT 的系统，波特率可以大于 2Mbps。在本应用报告中，使用 F2812 eZdsp 仿真 ICT 时，波特率达到了 1.875Mbps。

4.2.3.1 连接 PC 与 EICT

你必须通过一个 RS-232 线缆将 PC 和用作 ICT 的 F2812 eZdsp 连接起来。由于 F2812 eZdsp 没有 RS-232 收发器,因此使用了 Link-Research 公司的 RS-232 接口产品。Link Research 公司的 LR-2812COM-2 接口板详细产品信息可从 http://www.link-research.com/下载。

LR-2812COM-2 接口板与 Spectrum Digital eZdsp 的接线（略，Table 4 略）

4.2.3.2 连接 EICT 与 F281x 目标板

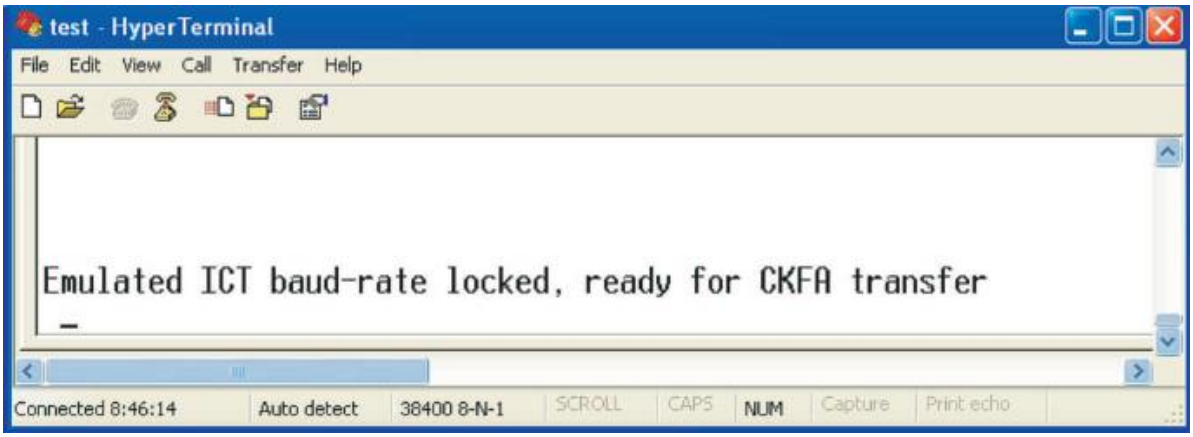
将 EICT 的 SCI-A 与目标板的 SCI-A 接口连接起来，注意 EICT 的 SCITXDA 应与目标板的 SCIRXDA 连接。（Table 5 略）

4.2.3.3 准备 EICT 软件

根据下面的步骤，连接 IEEE Std. 1149.1-1990(JTAG)仿真器与 EICT。
1. 连接 IEEE Std. 1149.1-1990(JTAG)仿真器与 EICT。
2. 打开 CCS 工作空间 SCI_FLASH_AppReport.wks。
3. 重启 CPU。
4. 加载代码。
5. 运行 real-time 模式。
6. 将 Watch 和 Memory 窗口设置为在 real-time 模式下持续更新。

4.2.3.4 锁定 PC 与 EICT 的波特率

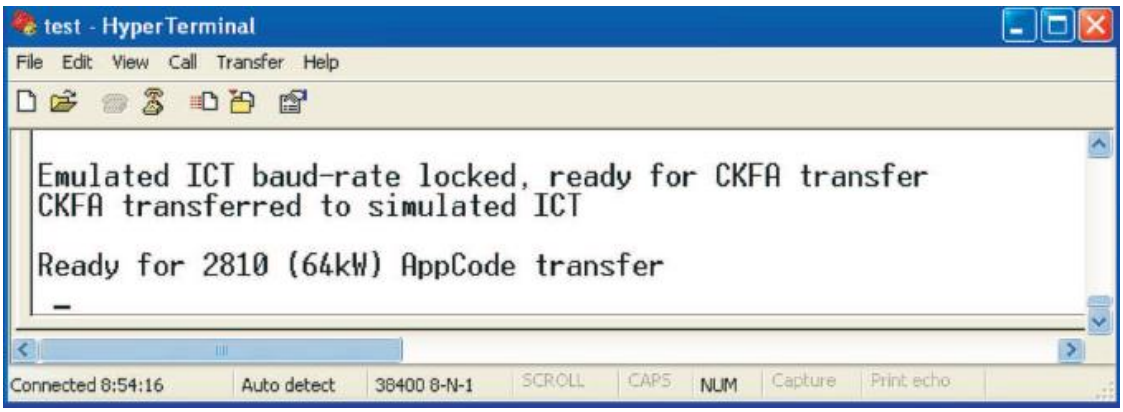
EICT 软件首先将 EICT 硬件的波特率与 PC(超级终端)的波特率匹配。在 CCS 中运行 EICT 软件，在超级终端中键入'a'或'A'（包括引号）。作为回应，EICT 软件会确认 EICT 波特率已锁定，做好了 CKFA 二进制文件的传输准备（见 Figure 25）.



【Figure 25. EICT 准备好从 PC 接收 CKFA】

4.2.3.5 从 PC 将 CKFA 和 AppCode 传送到 EICT RAM 中

按照 3.4.1 节的步骤，用超级终端传送 CKFA 软件。在超级终端上，点击“传送”—“发送文本文件”，从 CKFA 工程 Debug 文件夹中选择 CKFA.bin。



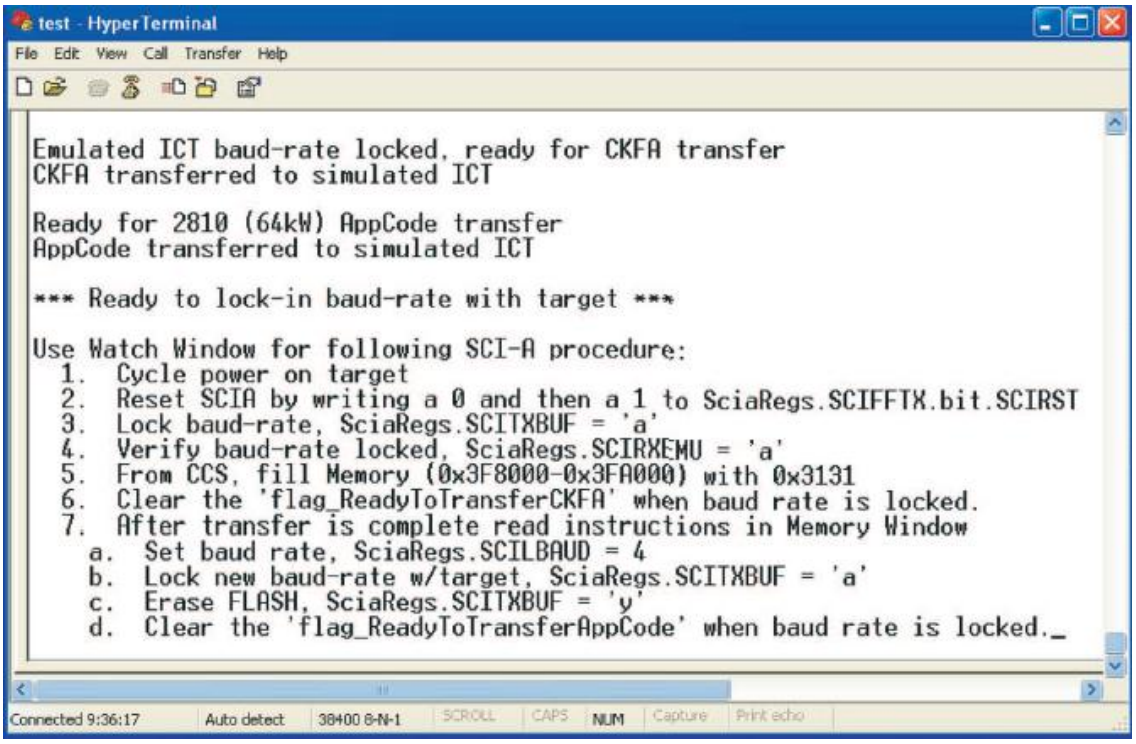
【Figure 26. EICT 准备好从 PC 接收 AppCode】

按照 3.4.4 节的步骤，用超级终端传送 AppCode 软件。在超级终端上，点击“传送”—“发送文本文件”，从 AppCode 工程 Debug 文件夹中选择 AppCode.bin。

4.2.3.6 锁定 EICT 与 F281x 目标板引导 ROM 代码的波特率

现在 CKFA 和 AppCode 二进制文件都保存在 EICT 的 RAM 中。EICT 现在可以执行前面所述的标准步骤，但不是使用慢速的超级终端 RS-232 传输，而是使用 EICT 快速直连。

EICT 将要执行的步骤传送到超级终端上(Figure 27)。PC 连接到 EICT 的 SCI-B 上，这不会对 EICT 和目标板通过 SCI-A 进行的通讯产生干扰。



【Figure 27. EICT 准备好开始 F281x 目标步骤】

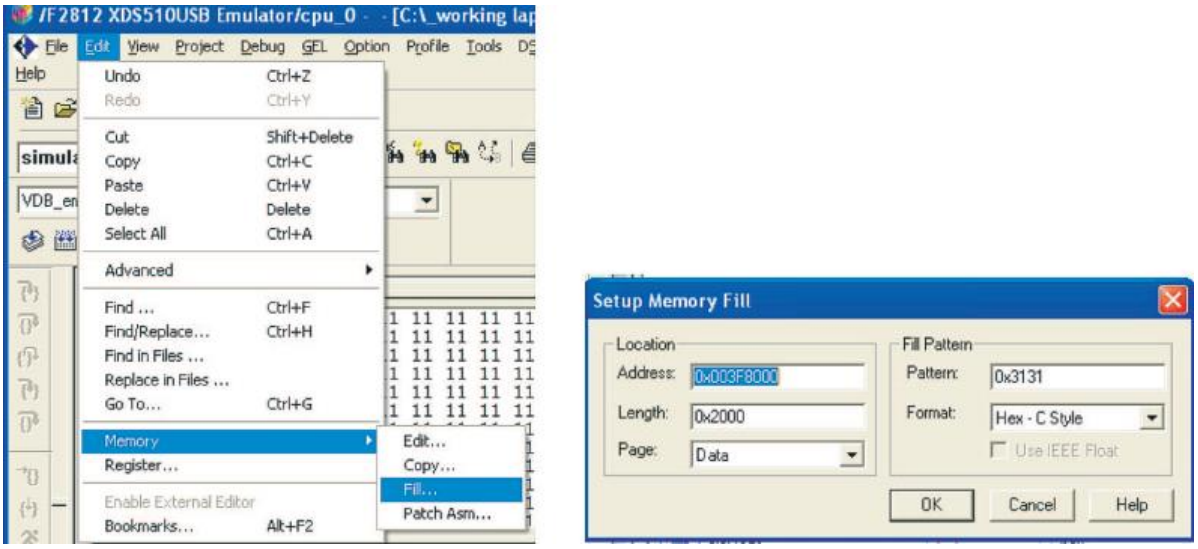
4.2.3.7 从 EICT 将 CKFA 传送到 F281x 目标板

让 CCS 运行在 real-time 模式，将 Watch 和 Memory 窗口配置为持续刷新，给 F281x 目标板先下电再上电。保持第二个 Wathc 窗口页打开。对 F2812 eZdsp，你需先将 5V 电源关掉，再重新打开。

注意 EICT 的 SCIA 有一个接收错误，典型情况下 SciaRegs.SCIRXST.all 值为 178。按照超级终端上的指令，向 SciaRegs.SCIFFTX.bit.SCIRST 先写 0 再写 1，以复位 EICT 的 SCIA。注意现在 SciaRegs.SCIRXST.all 值为 0.

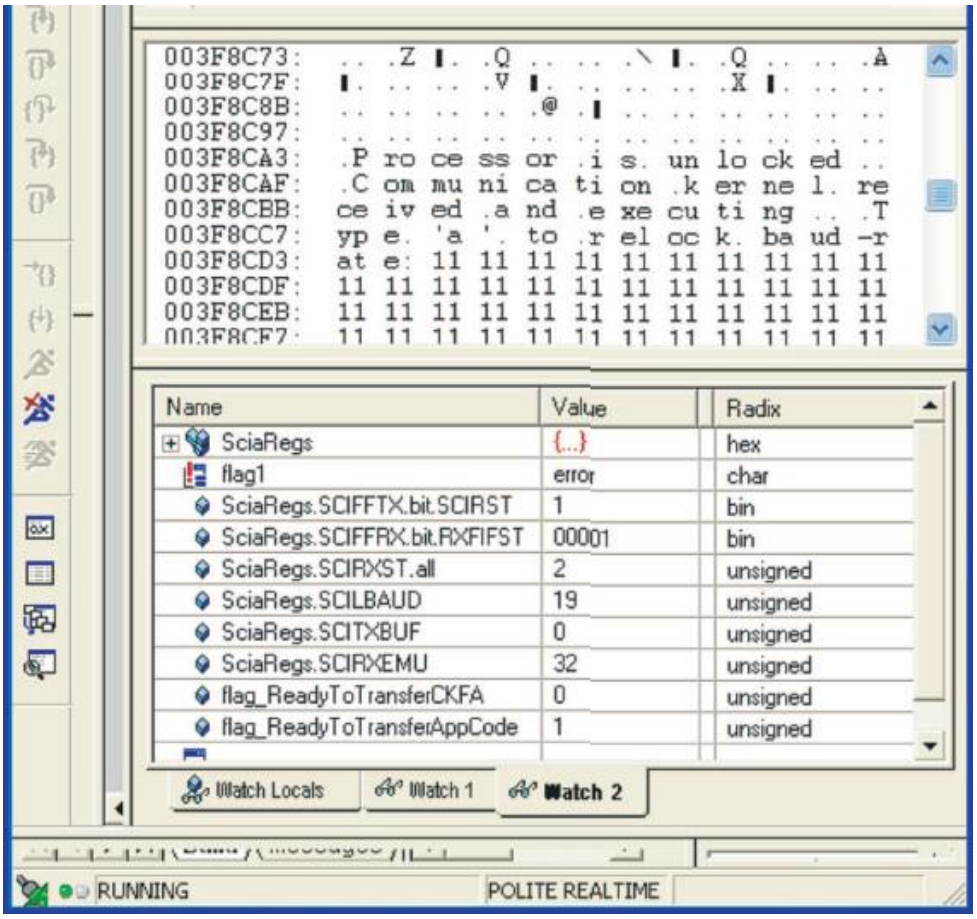
设定目标板的波特率。在观察窗口中键入自动波特率字符'a'或'A'，SciaRegs.SCITXBUF='a'。注意包括引号。作为回应，目标板会将自动波特率字符发回，你会在 Watch 窗口的 SciaRegs.SCIRXEMU 中看到它(97='a').

现在目标板准备好接收 CKFA 了。在开始传输前，准备 CCS 的 Memory 窗口来接收目标板传来的信息。在 Edit 菜单中，选择 Memory—Fill。将从 0x3F8000 开始，长 0x2000 的区域用 0x3131（"11"）填充（Figure 28）。



【Figure 28. 准备 Memory 窗口来接收目标板传来的信息】

在 Watch 窗口中，将 flag_ReadyToTransferCKFA 设为 0，以启动 CKFA 传输。当 Memory 窗口显示“Processor is unlocked. Communication kernel received and executing. Type 'a' to relock baud-rate:(见 Figure 29)”时，传输完成。下一步是增加 EICT 波特率。

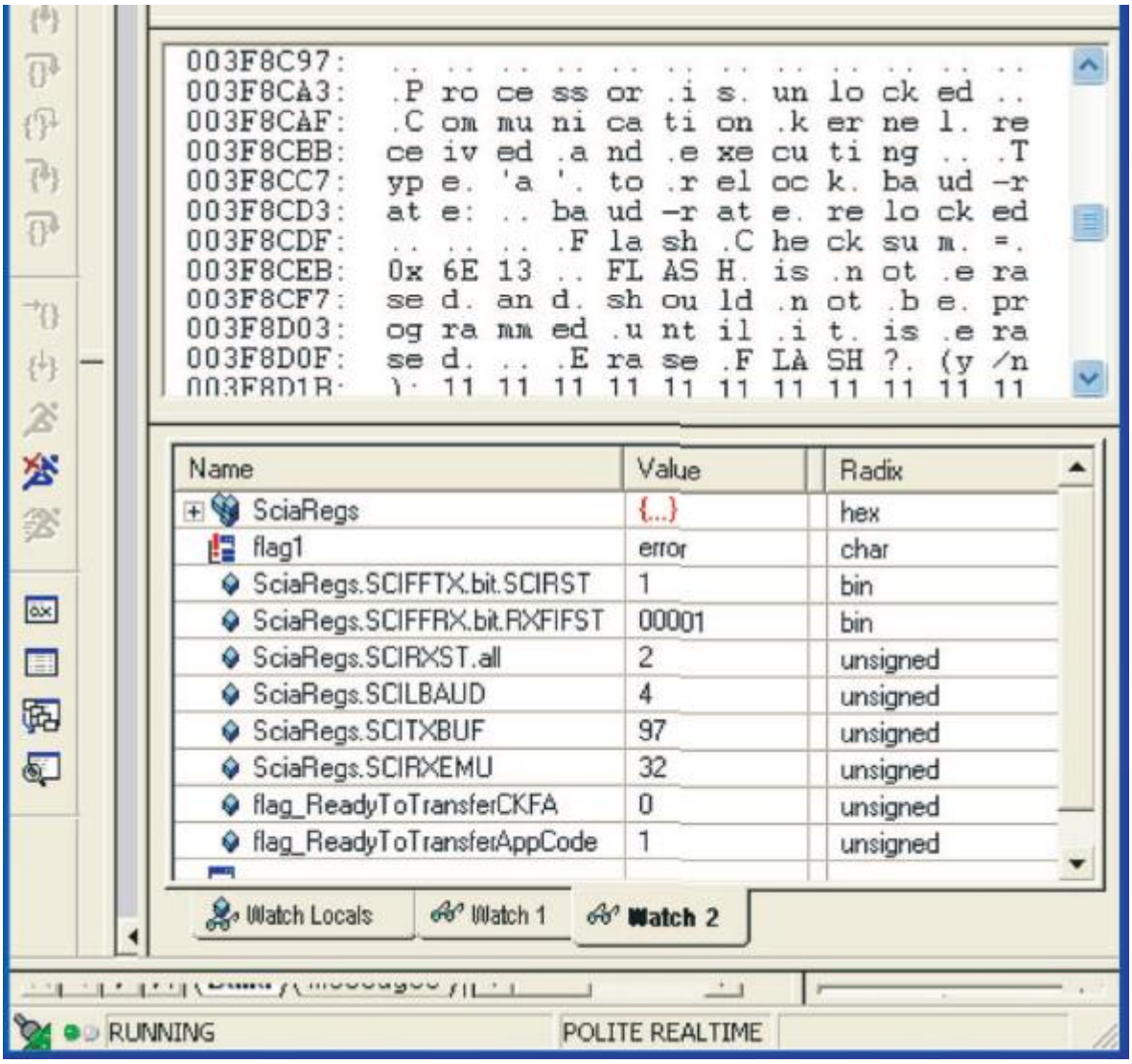


【Figure 29. 从 EICT 向目标板传送 CKFA 成功】

4.2.3.8 锁定新的最大波特率

在 Watch 窗口中，将 EICT 波特率寄存器 SciaRegs.SCILBAUD 设为 4。运行在目标板上的 CKFA 软件已经使能了 SCI-A 自动波特率逻辑，以将自身波特率与新的 EICT 波特率锁定。

在 Watch 窗口中，将 SciaRegs.SCITXBUF 设为自动波特率字符（'a'或'A'）（注意包括引号）。作为回应，目标板会发回自动波特率字符，你会在 Watch 窗口的 SciaRegs.SCIRXEMU 中看到它(97='a')（Figure 30）.



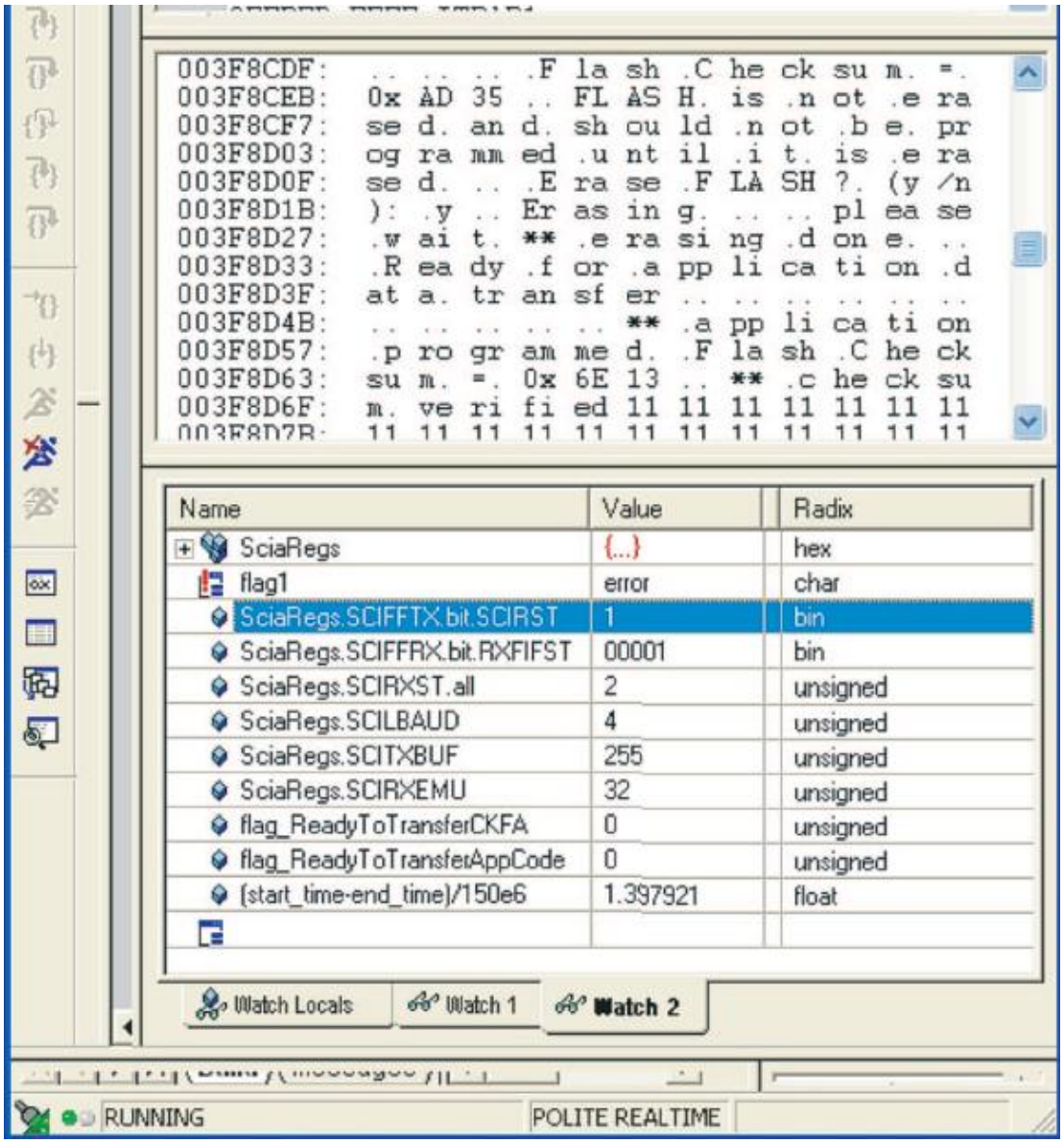
【Figure 30. CKFA 波特率重新与 EICT 锁定在 1.875Mbps】

目标板发回消息“Erase Flash?”，作为回应，在 Watch 窗口中，将 SciaRegs.SCITXBUF 设为'y'（注意包括引号）。CKFA 软件将擦除目标板的 Flash。

在 Memory 窗口中可以看到，目标板发回消息“Erasing...please wait”.

4.2.3.9 从 EICT 将 AppCode 传送到 F281x 目标板

在 Watch 窗口中，将 fllag_ReadyToTransferAppCode 设为 0，以开始 AppCode 传输。当 Memory 窗口显示“*** erasing done. Ready for application data transfer”（见 Figure 31）时，传输完成。



【Figure 31. 将 AppCode 烧写到目标板 Flash 成功】

当 Memory 窗口显示“** application programmed. Flash Checksum=0x6E13. **checksum verified”时，AppCode 传输与烧写完成。这与用超级终端烧写结束后收到的信息相同。

烧写时间列在 Watch 窗口中：

(start_time - end_time) / 150e6 = 1.398 seconds

这是使用 F28x CPU 递减计数器形成的基准时间。

这个要顶。
我做 CKFA 做完了，分享一些经验，但估计很多理解不全面，欢迎指正。

1.在 CKFA.cmd 里定义了 CKFA 写入 DSP UnsecuredRAM 的位置

```
SECTIONS
{
.text_unsecured          : > RAMM0M1          PAGE = 0
{
.\debug\obj\unlock_main.obj (.text)
.\debug\obj\DSP281x_CodeStartBranch.obj (.text)
.\debug\obj\Example_Flash281x_CsmKeys.obj (.text)
.\debug\obj\DSP281x_MemCopy.obj (.text)
rts2800_ml.lib (.text)
}
  .econst_unsecured          : > RAMM0M1 PAGE = 0
{
.\debug\obj\unlock_main.obj (.econst)
}
.............................
```

用串口工具将 CKFA 传送到上述地址暂时存放；

2. CKFA 开始运行，开始运行的地址也有定义：

```
CKFA.cmd
codestart          : > RAMM0M1          PAGE = 0
```

检查方法：CKFA_hex.map

```
Entry Point:              0x000002f2
```

最先运行的是解锁步骤，解锁成功后

3. CKFA 的 Unlocak_main.c 里

```
MemCopy(&textLoadStart, &textLoadEnd, &textRunStart);
MemCopy(&econstLoadStart,&econstLoadEnd,&econstRunStart);
main2();
```

这两句表明要把 CKFA 自身从 UnsecuredRAM 复制到 securedRAM，复制的目的是运行 CKFA 的 main2()程序

复制的地址在 CKFA.cmd 里定义

```
SECTIONS
.text                      :LOAD = RAMH0_1,
                           RUN = RAML0L1,
                           LOAD_START(_textLoadStart),
                           LOAD_END(_textLoadEnd),
                           RUN_START(_textRunStart),
                           PAGE = 1
```

```
.econst         :LOAD = RAMH0_1,
                 RUN = RAML0L1,
                 LOAD_START(_econstLoadStart),
                 LOAD_END(_econstLoadEnd),
                 RUN_START(_econstRunStart),
                 PAGE = 1
```

4.然后 CKFA 运行 AppCode 的传送。

采用双 RAM-4KW 接受，第一个接受满了，开始烧写 Flash；同时用另一个 RAM 接受下一个 4KW。

这里还比较糊涂，就先不班门弄斧了。