单 CPU 双项目开发实现更好的维护性和可行性

JOHNSON CHEN/ RICKY ZHANG

摘要

为了便于移植,提高灵活性并进行更好的维护,越来越多的应用代码被划分为不同的模块。但是如果团队各不相同，甚至不同的开发方，例如一方开发应用软件，另一方开发核心算法，双项目将始终是首选方法。

本应用手册将讨论如何创建双项目，如何生成一个项目的符号库，以及在双项目之间传递数据的多种方法。此外，手册中还介绍了如何在一个项目中调用另一个项目的函数，如何在 CCS 里面如何进行双项目调试，同时还将提供基于 F280025 例程供参考。

Contents

Figures

1 介绍

为了便于移植，提高灵活性并进行更好的维护，越来越多的应用代码被划分为不同的模块。但是，如果团队各不相同，甚至不同的开发方，例如一方开发应用软件，另一方开发核心算法，双项目将始终是首选方法。

下面将以 F280025 为例，介绍如何一步步创建双项目，在双项目之间传递数据的多种方法， 如何创建一个项目的符号库，如何在一个项目中调用另一个项目的函数，如何在 CCS 里面调试带有符号库的双项目，同时还将提供基于 F280025 参考代码。

与单个项目相比，创建双项目需要重新配置内存映射，并设置不同的代码起始入口以及各自的 Flash 和 RAM 空间。

本文例程项目一和项目二都是从下面项目导入

C:\ti\c2000\C2000Ware_4_00_00_00\driverlib\f28002x\examples\empty_projects

本文双项目使用场景假设如下:

a. 项目二为主项目，项目一为从项目。
b. 项目二将调用项目一里的函数 test()。
c. 项目一使用 RAMLS7 作为 RAM 空间以及 FLASH_BANK0_SEC14 和 FLASH_BANK0_SEC15 作为 Flash 空间，项目二使用除了 RAMLS7 以外的 RAM 空间以及除 FLASH_BANK0_SEC14 和 FLASH_BANK0_SEC15 以外的 Flash 空间。
d. 两个项目共用堆栈，堆栈使用 RAMM1。

2 创建和配置双项目

下面将详细介绍双项目具体创建和配置步骤。

2.1 创建和配置项目一

导入 C:\ti\c2000\C2000Ware_4_00_00_00\driverlib\f28002x\examples\empty_projects 项目后，在项目名上点击右键选择“Rename”(如图 1)将项目重命名为 F28002x_Project1.

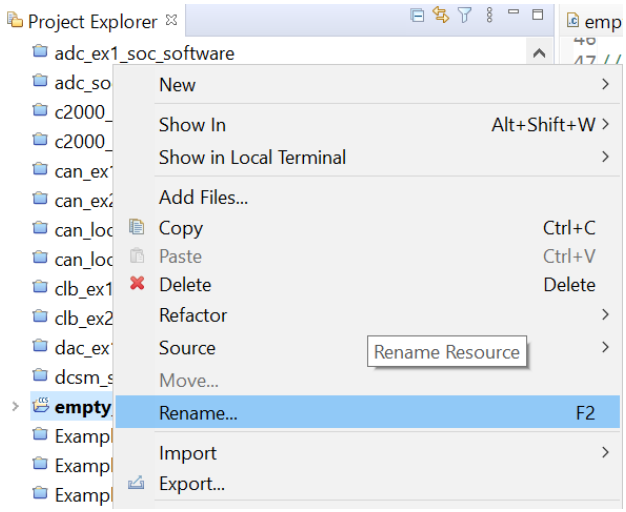

图 1. 项目重命名

项目一中，将需要给其它项目调用的函数放到一个或者多个文件中，例如，将 test()函数放到 test.c 中。这样方便后面创建符号库时，只把需要调用的函数符号信息给到项目二。

第一步：新建 test.c 文件， test.c 内容如下：

```
// test.c
int test(int a, int b)
{
    int c;
    c = a + b;
    return ( c );
}
```

第二步：新建 test.h 文件，以方便两个项目之间调用 test()函数， test.h 内容如下：

```
// test.h
extern int test(int a, int b);
```

第三步：创建项目一 main.c 内容如下：

```
// main.c
```

```
#include "driverlib.h"
#include "device.h"
#include "test.h"
int Var1=0;
void main(void)
{
    int  temp1,temp2;
    temp1 = 55;
    temp2 = 100;
    for(;;)
    {
       Var1 = test(temp1 , temp2 );
     }
}
```

第四步：项目一 CMD 分配如下：

a. 使用 RAMLS7 作为 RAM 空间。
b. 使用 FLASH_BANK0_SEC14 和 FLASH_BANK0_SEC15 扇区作为 Flash 空间，为方便使用将 FLASH_BANK0_SEC14 和 FLASH_BANK0_SEC15 合并为了 FLASH_BANK0_SEC1415。
c. 堆栈使用 RAMM1。

```
MEMORY
{
   BEGIN              : origin = 0x08E000, length = 0x000002
   BOOT_RSVD          : origin = 0x00000002, length = 0x00000126
   RAMM1            : origin = 0x00000400, length = 0x000003F8
   RAMLS7           : origin = 0x0000B800, length = 0x00000800
   BOOTROM          : origin = 0x003F0000, length = 0x00008000
   BOOTROM_EXT      : origin = 0x003F8000, length = 0x00007FC0
   RESET            : origin = 0x003FFFC0, length = 0x00000002
   /* Flash sectors */
   FLASH_BANK0_SEC1415 : origin = 0x08E002, length = 0x001FEE

}
SECTIONS
{
   codestart        : > BEGIN, ALIGN(8)
   .text            : > FLASH_BANK0_SEC1415,  ALIGN(8)
```

```
    .cinit          : > FLASH_BANK0_SEC1415,  ALIGN(8)
    .switch         : > FLASH_BANK0_SEC1415,  ALIGN(8)
    .reset          : > RESET,                  TYPE = DSECT /* not used, */
    .stack          : > RAMM1
    .init_array     : > FLASH_BANK0_SEC1415,  ALIGN(8)
    .bss            : > RAMLS7
    .bss:output     : > RAMLS7
    .bss:cio        : > RAMLS7
    .const          : > FLASH_BANK0_SEC1415,  ALIGN(8)
    .data           : > RAMLS7
    .sysmem         : > RAMLS7
   .TI.ramfunc    : LOAD = FLASH_BANK0_SEC1415,
                    RUN = RAMLS7,
                    LOAD_START(RamfuncsLoadStart),
                    LOAD_SIZE(RamfuncsLoadSize),
                    LOAD_END(RamfuncsLoadEnd),
                    RUN_START(RamfuncsRunStart),
                    RUN_SIZE(RamfuncsRunSize),
                    RUN_END(RamfuncsRunEnd),
                    ALIGN(8)
  Project1_Function : > FLASH_BANK0_SEC1415,  ALIGN(8)
       {
              -l test.obj(.text)
       }
}
```

上面 CMD 里面 Project1_Function 段分配的目的是把 test.c 或者 test.asm 文件里的代码分配到 Project1_Function 这个段里面， 这样可以只将其它项目要使用到的函数符号生成到符号库里，减少两个项目之间的冲突，以便后面双项目调试。Project1_Function 段名后面会用到。

如果有多个函数在不同源文件中需要给其它项目调用，可以在｛｝中多添加几行 “-l xxxx.obj(.text)” 内容，xxxx 为源文件名。

2.2 创建和配置项目二

导入 C:\ti\c2000\C2000Ware_4_00_00_00\driverlib\f28002x\examples\empty_projects 项目后，在项目名上点击右键选择“Rename”将项目重命名为 F28002x_Project2.

第一步：将项目一里的 test.h 添加到项目二中。

第二步: 创建项目二 main.c 内容如下：

```
#include "driverlib.h"
#include "device.h"
#include "test.h"
int R_Test1, R_Test2, R_Test3;
void main(void)
{
    R_Test2 = 55;
    R_Test3 = 100;
    for(;;)
    {
        R_Test1 = test(R_Test2 , R_Test3 );
    }
}
```

第三步:项目二 CMD 分配如下：

a. 使用 RAMLS4，RAMLS5，RAMLS6 作为 RAM 空间，为方便使用将 RAMLS4，RAMLS5，RAMLS6 合并为 RAMLS456。
b. 使用 FLASH_BANK0_SEC0 到 FLASH_BANK0_SEC13 扇区作为 Flash 空间，为方便使用将 FLASH_BANK0_SEC0 到 FLASH_BANK0_SEC13 扇区合并为了 FLASH_BANK0_13。
c. 堆栈使用 RAMM1。

```
MEMORY
{
   BEGIN            : origin = 0x080000, length = 0x000002
   BOOT_RSVD        : origin = 0x00000002, length = 0x00000126
   RAMM0            : origin = 0x00000128, length = 0x000002D8
   RAMM1            : origin = 0x00000400, length = 0x000003F8

   /* Combining all the LS RAMs */
   RAMLS456         : origin = 0x0000A000, length = 0x00001800
  // RAMLS7         : origin = 0x0000B800, length = 0x00000800  /* Used by project1 */
   RAMGS0           : origin = 0x0000C000, length = 0x000007F8
   BOOTROM          : origin = 0x003F0000, length = 0x00008000
   BOOTROM_EXT      : origin = 0x003F8000, length = 0x00007FC0
   RESET            : origin = 0x003FFFC0, length = 0x00000002
```

```
    /* Flash sectors */
    /* Combining sector0-13 */
    FLASH_BANK0_SEC0_13 : origin = 0x080002, length = 0x00DFFE
   // FLASH_BANK0_SEC14 : origin = 0x08E000, length = 0x001000 /* Used by project1 */
   // FLASH_BANK0_SEC15 : origin = 0x08F000, length = 0x000FF0 /* Used by project1 */

}
SECTIONS
{
   codestart        : > BEGIN, ALIGN(8)
   .text            : > FLASH_BANK0_SEC0_13,  ALIGN(8)
   .cinit           : > FLASH_BANK0_SEC0_13,  ALIGN(8)
   .switch          : > FLASH_BANK0_SEC0_13,  ALIGN(8)
   .reset           : > RESET,                  TYPE = DSECT /* not used, */
   .stack           : > RAMM1
   .init_array      : > FLASH_BANK0_SEC0_13,  ALIGN(8)
   .bss             : > RAMLS456
   .bss:output      : > RAMLS456
   .bss:cio         : > RAMLS456
   .const           : > FLASH_BANK0_SEC0_13,  ALIGN(8)
   .data            : > RAMLS456
   .sysmem          : > RAMLS456
    ramgs0 : > RAMGS0
  .TI.ramfunc      : LOAD = FLASH_BANK0_SEC0_13,
                     RUN = RAMGS0,
                     LOAD_START(RamfuncsLoadStart),
                     LOAD_SIZE(RamfuncsLoadSize),
                     LOAD_END(RamfuncsLoadEnd),
                     RUN_START(RamfuncsRunStart),
                     RUN_SIZE(RamfuncsRunSize),
                     RUN_END(RamfuncsRunEnd),
                     ALIGN(8)
}
```

3 在两个项目之间传递数据

为了更好的信息安全保护，通常一个项目会是保密的，这个保密的项目将提供库文件给另一个项目调用。这个库文件可以是带有具体二进制代码的库或者代码已经烧录到芯片里面的只包含符号信息的库。无论是哪种库，两个项目之间总是需要交换一些数据。

通常有下面几种方式来交换数据：

a. 变量绝对地址数据读写；

b. 函数调用来传递单个或多个数据；

c. 函数调用来传递结构体数据，可以包括多个变量或者指针；

和变量类似， 函数调用有下面一些方法：

a. 绝对地址函数调用；

```
#define InitAnalogSystemClock (unsigned short (*) (unsigned short ClockDivider))0x0024021E
// call InitAnalogSystemClock() fn at address 0x00240214
unsigned short asysclk = (*InitAnalogSystemClock)(ACLKDIV4);
```

b. 符号库调用；

c. 结构体变量函数指针；

```
struct DEMO
{
	int x,y;
	int (*func)(int,int); //function pointer
}; demo
int add2(int x,int y)
{
	return x+y;
}
int c;
demo.func = &add2; // structure function pointer initialization
c = demo.func(3,4);
```

绝对地址函数调用的方法，需要从编译完成的项目的 map 文件中，手动查找相应函数的具体地址。如果函数较多，这将会是不小的工作量。另外，如果源文件有改动，编译后函数地址会改变，因此需要用户重新去 map 文件里面手动查找并更新函数的地址，这样不利于项目维护。

使用符号库调用的方法可以自动生成函数的地址信息，因此具有更好的可维护性，对于函数数量较多的场景来说，还可以大幅减少工作量。因此如何创建一个符号库变得尤为重要。

在本文中，项目二为主项目，项目一为从项目，所以只有项目二会以符号库的方式调用项目一里的函数，如果项目一需要调用项目二的函数可以使用绝对地址来调用。接下来将介绍如何创建一个项目的符号库。

4 如何创建一个项目的符号库

两个项目会烧录到芯片不同的 Flash 扇区里面，项目二将会调用项目一里面的函数，因为项目一已经烧录到了 Flash 里面， 因此没必要将项目一的代码加到项目二里面， 项目二只需要知道项目一里面的函数的地址就可以。

为了实现这个需求，我们可以让项目一生成符号库，符号库只会包含函数的地址信息而不会包含具体代码。符号库生成后，只需要把项目一的符号库添加到项目二进行编译就可以了。

下面为生成符号库的具体方法：

a. 如图 2 所示，在项目一“Properties”里面，点击“Build ->C2000 Linker -> Linker Output。”在“Detailed link information data-base into <file>”栏输入文件名 Project1.xml （文件名可以根据需要自己设置）。

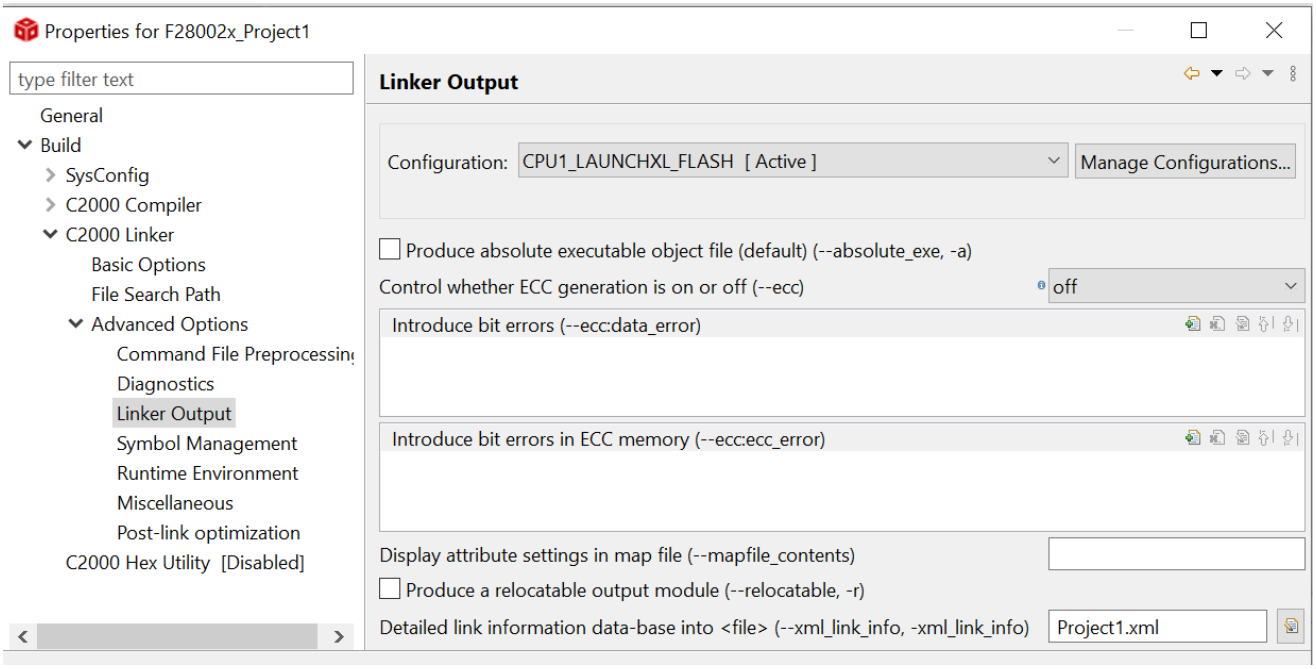

图 2. 项目一 Linker Output 设置

b. 编译项目一，编译通过后会生成 Project1.xml，Project1.xml 在后面的步骤里会用到。

c. 安装 Strawberry PERL 软件。 (free -- can download from Strawberry Perl website: https://strawberryperl.com/)。

d. 下载并安装最新的 Code Gen Tools XML Perl scripts utility。(https://software-dl.ti.com/ccs/non-esd/releases/other/applications_packages/cg_xml/index.htm)。

e.创建一个文件夹，例如 test，将项目一编译生成的 Project1.xml 拷贝到这个路径下。

f.在上面新建的文件夹下，新建一个 Project1_symbollibrary.txt 文件，将文件名改成 Project1_symbollibrary.bat。然后将下面命令行拷贝到 Project1_symbollibrary.bat 文件里。

```
PATH = "C:\ti\ti-cgt-c2000_20.2.1.LTS\bin";C:\Strawberry\perl\bin;C:\Strawberry\c\bin;C:\Strawberry\perl\site\bin;

del *tmp*.asm

del *tmp*.obj

del *.lib

perl C:\TEST\FlashAPI\cgxml-2.61.00\map\get_rom_symbols.pl Project1.xml -s=Project1_Function

cl2000 --float_support=fpu32 --tmu_support=tmu0 --abi=eabi -g -pdsw225 -d"_DEBUG" -d"LARGE_MODEL" -ml -v28 *.asm

ar2000 r  Project1_SymbolLibrary.lib *.obj

del *tmp*.obj *.asm
```

注释：

1.用户实际使用的话，可以根据实际需要更改 C2000 编译器路径和文件名 (Project1.xml 以及 Project1_SymbolLibrary.lib)，Project1_Function 段是在前面项目一 CMD 里定义的。

2.如果需要把项目一整个代码中的函数生成符号库信息，可以将 Project1_Function 改成 .text。

3.PATH 这行的后面几个路径中间不能有空格。

g.双击 Project1_symbollibrary.bat 文件，这样就生成了项目一的符号库 Project1_SymbolLibrary.lib。这时我们可以把 Project1_SymbolLibrary.lib 添加到项目二里面， 这样项目二就可以用同一个函数名来调用 Flash 里的项目一函数。

5 进行双项目调试

前面已经生成了项目一的符号库，我们可以把项目一符号库添加到项目二里面，然后进行项目二编译，编译完成后会生成项目二的.out 文件。

尽管只是符号库， 但只要有项目的.out 文件， 我们也可以两个项目一起调试。

第一步：我们把项目一通过“load program”烧录到 Flash 里面。如图 3 所示，假设项目一“On-Chip Flash”里面是默认设置(如图 3)，这样将会擦除整个 Flash，然后烧录代码。

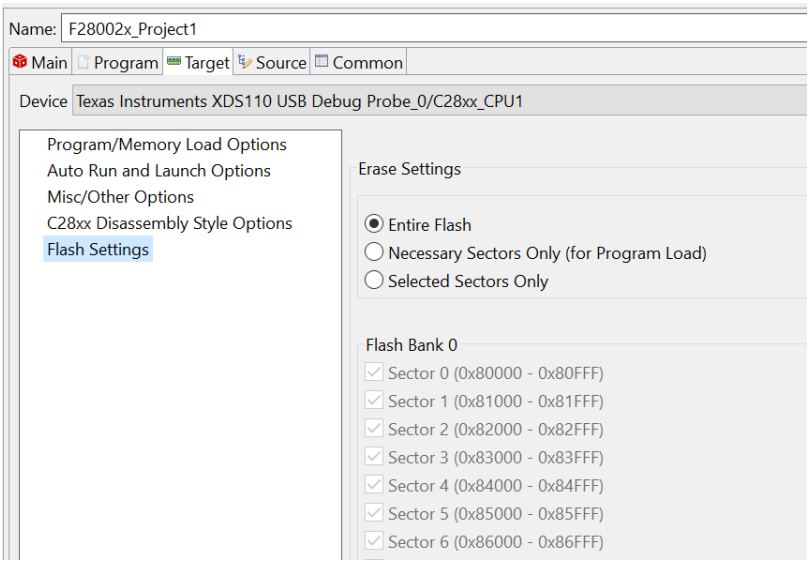

图 3. 项目一 On-Chip Flash 设置

第二步：如图 4 所示，在项目二的“On-Chip Flash”里面更改“Flash Program Setting”为“Necessary Sectors Only(for Program Load).”

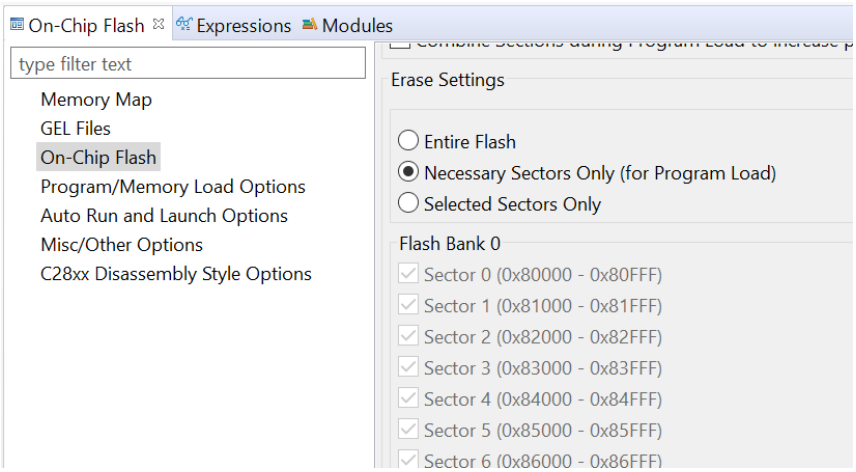

图 4. 项目二 On-Chip Flash 设置

这样可以防止在加载项目二的时候所有的 Flash 扇区被擦除掉，这样设置将只会擦除项目二用到了的 Flash 扇区。

项目二加载完后， 两个项目都已经烧录到了芯片 Flash 里面。因为我们最后加载的是项目二的代码，因此项目二的符号信息已经存在。这时我们可以正常调试项目二， 但是不会有项目一的符号信息，因为在加载项目二的时候， 把项目一的符号信息覆盖掉了。

为了方便调试项目一，接下来我们可以通过下面步骤把两个项目的符号信息添加进来。

a. 如图 5 所示，可以通过“Run-> Load-> Add Symbols”。这时要特别注意，我们要选择 Add Symbols 而不是 Load Symbols。

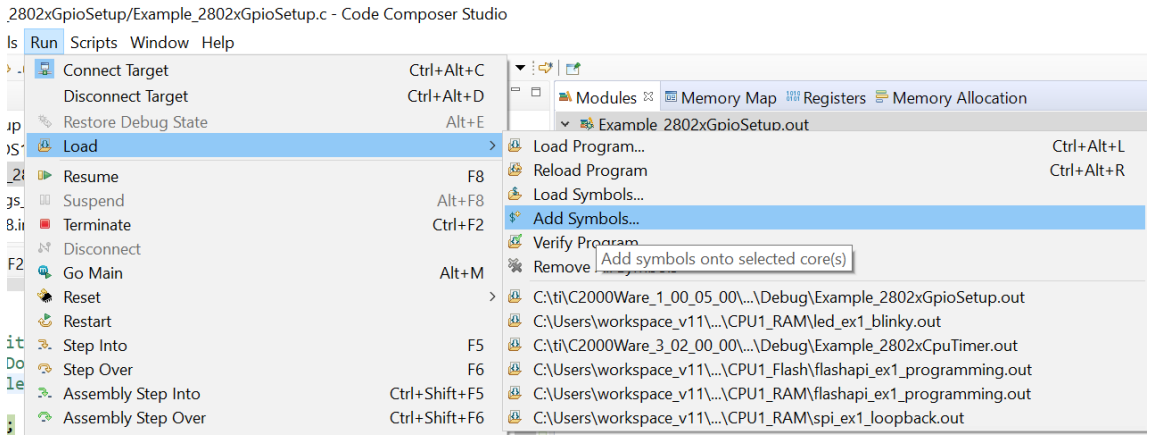

图 5. 添加项目符号

b. 在弹出来的窗口中点击“Browse”。

c. 然后选择项目一的 .out 。

d. 重复 a 和 b 步骤把项目二的.out 加载进来。

这样两个项目的符号信息都有了。可以通过“View -> Modules”来确认一下,如图 6 所示。

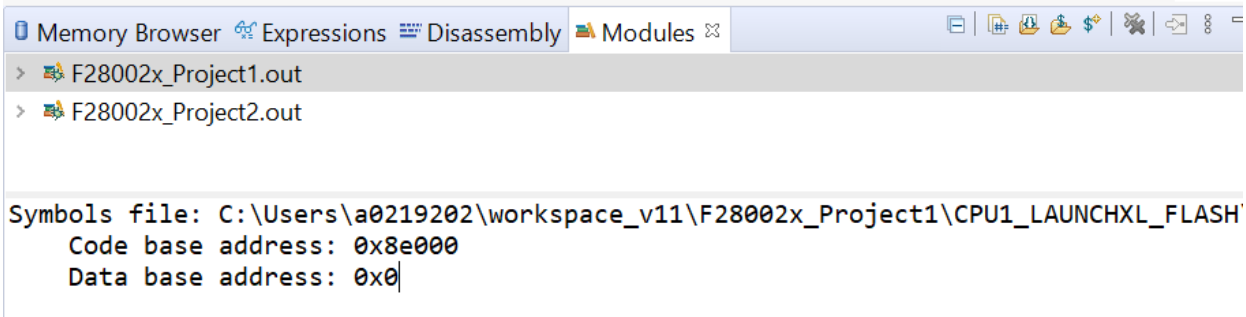

图 6. Modules 窗口

接下来可以进行双项目调试。

如图 7 所示，可以在“Expressions”窗口中查看两个项目的全局变量。其中 Var1 是项目一中的全局变量，R_Test1，R_Test2 R_Test3 是项目二中的全局变量。

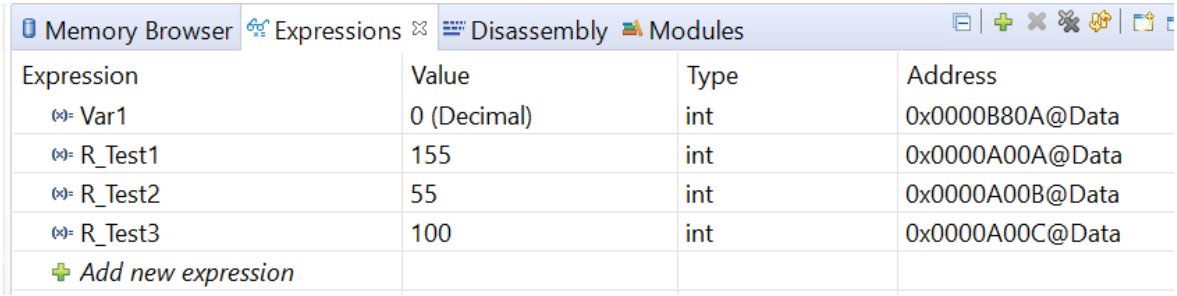

图 7. 实时调试双项目

6 结论

本文介绍了如何在双项目之间调用函数， 如何生成一个项目的符号库， 双项目之间几种不同的传输数据方式，以及如何在 CCS 里面调试带符号库的双项目。

基于 F280025 的例程，也验证本文方法的可行性。

其它 C2000 MCU 系列如：F28003x,F28004x,F2837x,F2838x 等均可参照本文方法实现双项目开发。

7 参考文献

1. TMS320F280025 Datasheet (SPRSP45)
2. TMS320F280025 Technical Reference Manual (SPRUIN7)

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265