Appendix 3. How to Convert Projects from Keil to CCS

Most of the examples in this book follow the Keil™ uVision® syntax. An equally powerful code development tool is the Texas Instruments Code Composer Studio™. The purpose of this Appendix is to illustrate how to convert files from Keil to CCS. Program A3.1 shows the equivalent code and order of use in an assembly file. The subroutine will input from Port A bit 5 and store the value into global variable **M** (0 or 0x20).

```
;Keil
       THUMB
       AREA    DATA, ALIGN=2

       EXPORT  M
M      SPACE   4

   AREA |.text|,CODE,READONLY,ALIGN=2

PORTA  EQU 0x400043FC
BIT5   EQU 0x20
       EXPORT  InputPA5
InputPA5

       LDR R0,=PORTA   ;R0 = &PORTA
       LDR R1,[R0]     ;R1 = PORTA
       AND R1,R1,#BIT5 ;Mask
       LDR R2,=M       ;R2 = &M
       STR R1,[R2]     ;M = PA5
       BX  LR

       END
```

```
;CCS
      .thumb                ;1)
      .data                 ;2)
      .align  4             ;3)
      .global M             ;4)
M     .field  32            ;5)
      .align  2             ;6)
      .text                 ;7)
PtM   .field M,32           ;8)
PORTA .field 0x400043FC,32  ;8)
BIT5  .equ   0x20           ;9)
      .global InputPA5      ;10)
      .thumbfunc InputPA5   ;11)
InputPA5: .asmfunc          ;12)
       LDR R0,PORTA         ;13)
       LDR R1,[R0]
       AND R1,R1,#BIT5
       LDR R2,PtM           ;13)
       STR R1,[R2]
       BX  LR
      .endasmfunc           ;12)
      .end                  ;14)
```

Program A3.1. This illustrates the order and syntax of pseudo-ops in assembly files.

1) Use Thumb assembly language
2) This is a data section (variables typically go in RAM)
3) Align on 32-bit boundary
4) Declare the variable **M** globally visible to other files including to C programs
5) Define an uninitialized 32-bit object and call it M
6) Align on 16-bit boundary
7) This is a text section, which is executable code and callable from C (in ROM)
8) **.field** defines 32-bit objects and initialize them as pointers to M and to Port A
9) **.equ** defines a numerical constant
10) Declare it globally visible to other files including to C programs
11) There is a thumb function with this name
12) **.asmfunc** and **.endasmfunc** help with debugging, marking beginning and end
13) A pointer-constant is stored in ROM, and PC relative addressing is used
14) Marks the end of the file

One of the difficulties in translating Keil to CCS is that the Keil syntax of **LDR R#,=Label** is not supported in CCS. So, to access variables and I/O ports we need to define a 32-bit pointer-constant using the .field pseudo-op. The actual machine code created by these two assemblers is virtually identical. The only difference is where in ROM the pointer-constant resides. In CCS you explicitly position the pointer-constants, and in Keil, the assembly automatically positions them.

In CCS there MUST be a 'main' function, if you have to you can alias it using substitution of symbols

```
.asg   "main", XXXXXXX
```

where **XXXXXXX** is the function name you want to substitute for main

In Keil you could write these four invalid instructions

```
AND  R0,R1,#0x00FFFFFFF
MOV  R1,#-1
ORR  R2,#0x0FFFFFF
CMP  R3,#-100
```

and it would be automatically converted to equivalent valid instructions

```
BIC  R0,R1,#0xFF000000
MVN  R1,#0
ORN  R2,#0xF0000000
CMN  R3,#100
```

In CCS you have to do this manually.

Each compiler has its own syntax for handling inline assembly. The syntax for inline assembly in C is illustrated in Program A3.2. Both compilers follow the AAPCS convention for passing parameters and saving registers.

```
// Keil
__asm void Delay(unsigned long ulCount){
    subs r0, #1
    bne  Delay
    bx   lr
}
```

```
// CCS
void Delay(unsigned long ulCount){
__asm ( "    subs    r0, #1\n"
        "    bne     Delay\n"
        "    bx      lr\n");
}
```

Program A3.2. This illustrates inline assembly in C programs.

The CCS code requires the quotation marks with a new line character at the end of each assembly line. This is a clever hack around to enable multiple lines to be written as one line. In essence Keil allows straight inline assembly, whereas in CCS you have to specify it as a string that will then be inserted. If you have to use assembly it is better to place it in a separate file, because inline assembly can be difficult to debug and makes the code less portable.

The example files of this book are posted on the book's web site and have versions for both compilers. For help with CCS equivalents please reference the document spnu118j.pdf (which can be found on www.TI.com).