An Easy Way of Creating a C-callable Assembly Function for the TMS320C28x DSP

Todd Anderson *Digital Signal Processing Solutions*

ABSTRACT

C-callable assembly routines require an understanding of parameter-passing conventions and environments expected by the C compiler. This application report provides instructions and suggestions to configure the C compiler to assist with these issues.

Contents

List of Figures

1 Introduction

When creating a C-callable assembly function, you must follow the parameter-passing conventions of the C compiler. Determining how to pass the parameters is often the most confusing and difficult part of writing such a function. By first writing a shell of the function in C, you can easily determine how the parameters are passed and also obtain a template of the code that properly passes all parameters and manages the stack.

To create applications in a C environment, follow these steps:

1. **Compile the C-code**. If the initial C-code is fast enough or within the code size to create your application, you do not need to proceed further.
2. **Optimize the C-code**. There are various C optimizations that you can use to create your application. If these are suitable, then again, you do not need to proceed further.
3. **Use pragmas and intrinsics**. The compiler has a built-in support that allows you to stay closer to the assembly code environment. If you can use these and they allow you to stay within your size/speed constraints, then you do not need to proceed further. You can consider your application complete at this point.
4. **Use C-callable assembly language functions**. There are times when optimization and use of pragmas and intrinsics do not allow you to meet size/speed constraints. At this point, you may need to create a C-callable assembly language function (described in Section 2).

Figure 1 demonstrates a flow chart of this process.

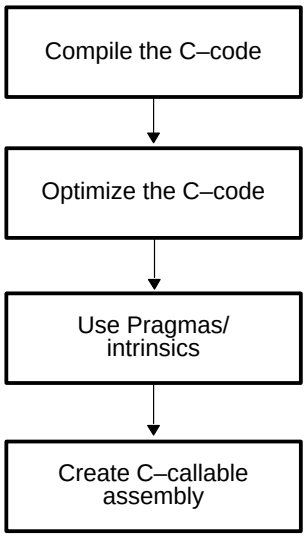

Figure 1. Recommended Flow for Creating Applications in a C Environment

2 Creating a C-Callable Assembly Function

To easily create a C-callable assembly function, follow these steps:

1. Set up a function prototype in C.
2. Set up a call to the future C-callable assembly function.
3. Create the function in C.
4. Compile the C file that has your function.
5. Look at the resulting interlisted file for the “Shell.”
6. Create a .asm file with code from the interlisted function.
7. Define the function prototype as external (if it is not already defined).
8. Add the assembly file to the Code Composer Studio project.

The following sections explain each of these steps in more detail.

2.1 Setting Up a Function Prototype in C

This is the first step in creating a C-callable assembly function.

The function prototype is set up at the beginning of the function that calls the C-callable assembly function, also referred to as “CcA.”

```
int slope(int,int,int);
```

2.2 Setting Up a Call

This is the second step in creating a C-callable assembly function.

The following code is an example of the code that is used to call the “slope” function:

```
int y = 0;
int x=1,b=2,m=3;
y = slope(b,m,x);
```

2.3 Creating the Function in C

This is the third step in creating a C-callable assembly function.

Set up a C function called “slope.” This is illustrated in Figure 2.

```
slope (b,m,x){
}
```

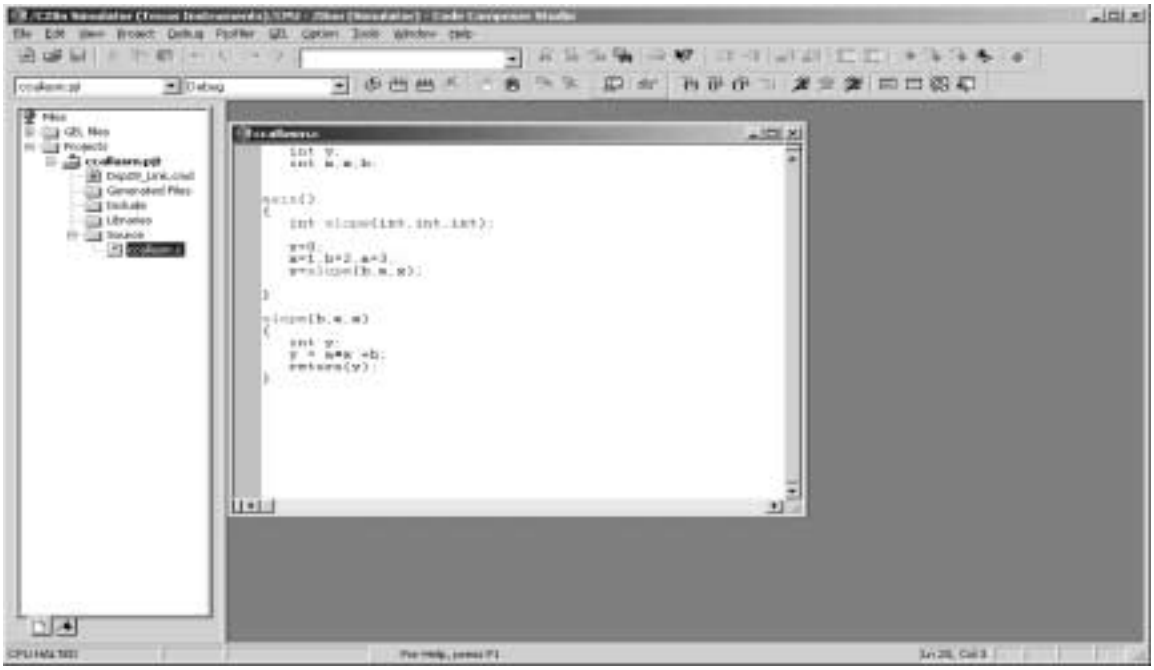

Figure 2. Setting Up a C Function Called “Slope”

In the example shown in Figure 2, note that to create the “slope” function you must:

1. Create a project called ***ccallasm.pjt***.
2. Create the source file ***ccallasm.c***.
3. Create and open a generic linker command file called ***DSP28_link.cmd***.

The C code in Figure 2 shows the combination of steps in section 1 outlined above. Later, you can rearrange the slope function to optimize your C code.

2.4 Compiling the C File That Contains Your Function

This is the fourth step in creating a C-callable assembly function.

To see the interlisted C/Assembly file, you must first configure some options in Code Composer Studio (CCS). To configure these options, in CCS, select Project → Build Options to open the Build Options dialog box.

2.4.1 Step-by-Step Compiler Options

The compiler tab in the Build Options dialog box allows you to select the necessary compiler options to compile your C file. To open the Compiler options window, in CCS, select Project → Build Options. Click on the Compiler tab in the Build Options dialog box.

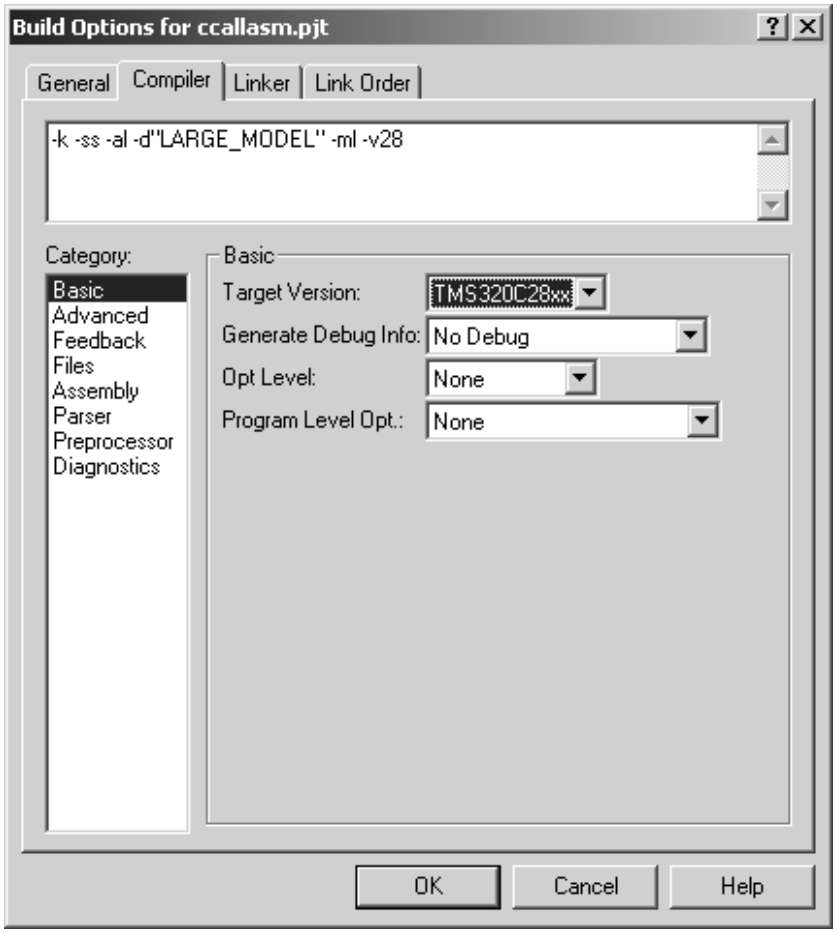

Figure 3. Basic Compiler Options

The compiler options used in the example shown in Figure 3 are:

-k: Keeps the assembly file

-ss: Performs interlisting of C and assembly

-al: Creates an assembly listing

-v28: Uses the C28x target version

- **Basic Compiler Options**. To configure the Basic compiler options, click the “Basic” option, located in the “Category:” pane of the Compiler options window.

  By selecting C28xx as the target device, the –d”LARGE_MODEL” and –ml options become part of the command line by default. The large memory model is a standard default for the C28x, making it easier to reach the full memory space (see Figure 3).

- **Advanced Compiler Options**. To configure the Advanced compiler options, click the “Advanced” option, located in the “Category:” pane of the Compiler options window.

  No specific options are selected from this window.

- **Feedback Compiler Options**. To configure the Feedback compiler options, click the "Feedback" option, located in the "Category:" pane of the Compiler options window.

  In this window, the Interlisting option is important – it allows C and Assembly to be shown in the assembly and listing files (see Figure 4).

Figure 4. Feedback Compiler Options

- **Files Compiler Options**. To configure the Files compiler options, click the “Files” option, located in the “Category:” pane of the Compiler options window.

  There are no specific options selected here. Sometimes, the Obj Directory has a “debug” directory shown. In this example, the debug directory was deselected (see Figure 5).

Figure 5. Files Compiler Options

- **Assembly Compiler Options**. To configure the Assembly compiler options, click the “Assembly” option, located in the “Category:” pane of the Compiler options window.

  You must keep the generated assembly files (–k) and optionally generate an assembly listing file (–al), as shown in Figure 6.

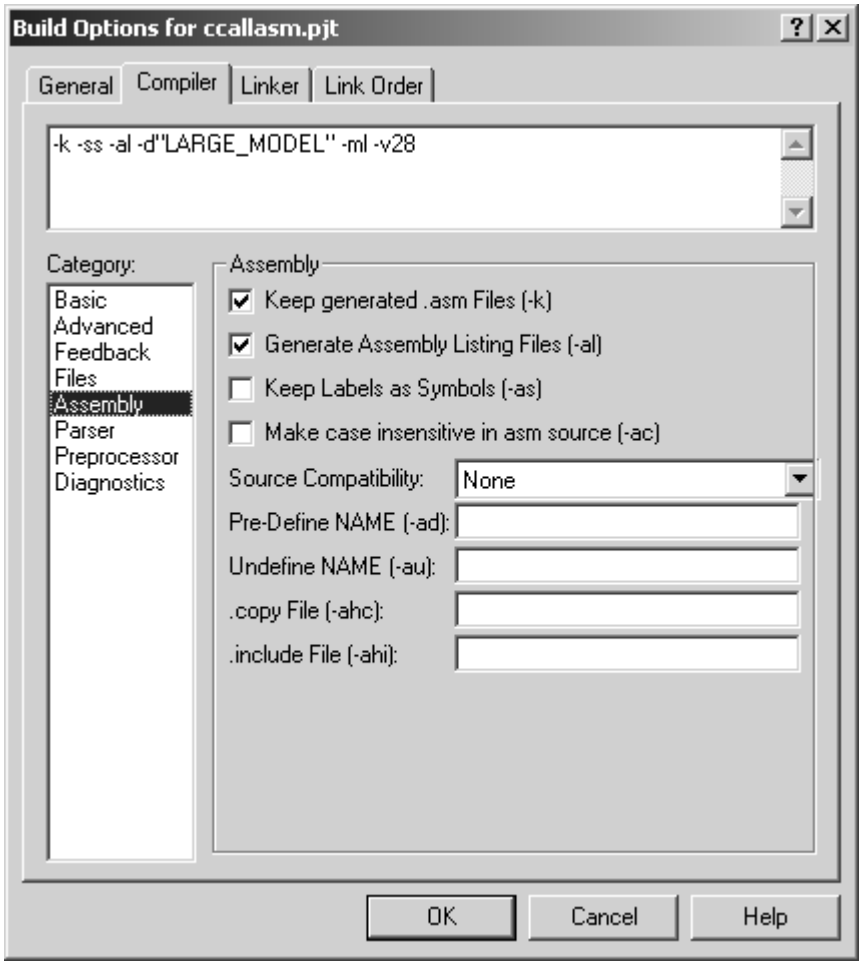

Figure 6. Assembly Compiler Options

- **Parser Compiler Options**. No specific options are selected from this window.
- **Preprocessor Compiler Options**. No specific options are selected from this window.
- **Diagnostics Compiler Options**. No specific options are selected from this window.

2.4.2 Step-by-Step Linker Options

The Linker tab in the Build Options dialog box allows you to select the linker options necessary to compile your C file. To open the Linker options window, in Code Composer Studio, select Project → Build Options. Click on the Linker tab in the Build Options dialog box.

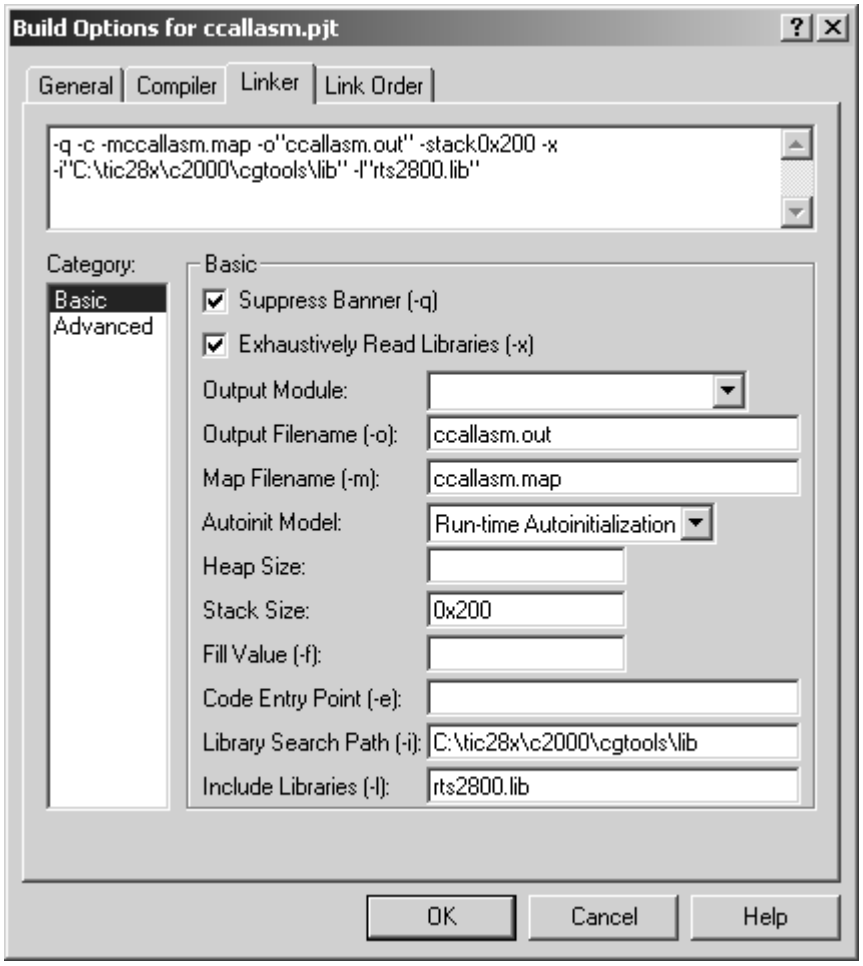

Figure 7. Basic Linker Options

- **Basic Linker Options**. To configure the Basic linker options, click the “Basic” option, located in the “Category:” pane of the Linker options window.

  The linker options used in the example shown in Figure 7 are:

  –o and –m: These options generate the named output and map files.

  –c : Use run-time autoinitialization.

  –stack: The stack size is set at a level that does not cause problems.

  –i & –l :Specify the library search path and the specific rts library.

NOTE: Autoinitialization is a feature that initializes C variables (if necessary) at system boot. For example, consider the C code “int n = 5; .“ The compiler creates a variable for the “n “; autoinitialization puts the value of “5” into “n.”

- **Advanced Linker Options**. To configure the Advanced linker options, click the “Advanced” option, located in the “Category:” pane of the Linker options window. No specific options are selected from this window.

2.5 Viewing the Resulting Interlisted File for the “Shell”

This is the fifth step in creating a C-Callable assembly function.

Figure 8 lists the segment of code that is generated from interlisting. The file, ccallasm.asm, results from selecting the C and O Assembly interlisting option during compiler configuration. The compiler adds many items to the assembly language file, most of which are NOT needed in the C-callable assembly file. Some of the unneeded items are .func/.endfunc, .sym, and .line.

The code listed in Figure 9, a crucial part of the entire code file, is the actual function that is generated by the compiler.

```
;***************************************************************
;* TMS320C2000 ANSI C Codegen                      Version 3.00 *
;* Date/Time created: Fri Aug 31 16:09:21 2001                  *
;***************************************************************
FP    .set  XAR2
      .file ”ccallasm.c”
      .global     _x
_x:   .usect      .ebss,1,1,0
      .sym  _x,_x, 4, 2, 16
      .global     _b
_b:   .usect      .ebss,1,1,0
      .sym  _b,_b, 4, 2, 16
      .global     _m
_m:   .usect      .ebss,1,1,0
      .sym  _m,_m, 4, 2, 16
      .global     _y
_y:   .usect      .ebss,1,1,0
      .sym  _y,_y, 4, 2, 16
;     c:\tic28x\c2000\cgtools\bin\ac2000.exe –qq –D_DEBUG –DLARGE_MODEL
--ml --version=28 –Ic:/tic28x/c2000/cgtools/include –D_DEBUG –DLARGE_MODEL
--ml --version=28 --keep_unneeded_types –m --i_output_file C:\DO-
CUME~1\a0192828\LOCALS~1\Temp\TI1404_2 --template_info_file C:\DO-
CUME~1\a0192828\LOCALS~1\Temp\TI1404_3 --object_file ccallasm.obj
--opt_shell 20 ccallasm.c –g –k –q –ss –al –d_DEBUG –dLARGE_MODEL –ml –v28
–g –k –q –ss –al –ic:/tic28x/c2000/cgtools/include –d_DEBUG –dLARGE_MODEL
–ml –v28 ccallasm.c
      .sect ”.text”
      .global     _main
      .sym  _main,_main, 36, 2, 0
      .func 4
```

Figure 8. Code Generated From Interlisting

```
;------------------------------------------------------------------------
;   4 | main()
;------------------------------------------------------------------------
;***************************************************************
;* FNAME: _main                          FR SIZE:   0           *
;*                                                              *
;* FUNCTION ENVIRONMENT                                         *
;*                                                              *
;* FUNCTION PROPERTIES                                          *
;*                             0 Parameter,  0 Auto,  0 SOE     *
;***************************************************************
_main:
      .line 2
;------------------------------------------------------------------------
;   6 | int slope(int,int,int);

;------------------------------------------------------------------------
      .line 5
;------------------------------------------------------------------------
;   8 | y=0;

;------------------------------------------------------------------------
        MOVW      DP,#_y
        MOV       @_y,#0                ; |8|
      .line 6
;------------------------------------------------------------------------
;   9 | x=1;b=2;m=3;

;------------------------------------------------------------------------
        MOVB      AL,#3                 ; |9|
        MOV       @_b,#2
        MOV       @_m,AL                ; |9|
        MOV       @_x,#1
      .line 7
;------------------------------------------------------------------------
;  10 | y=slope(m,x,b);

;------------------------------------------------------------------------
        MOVZ      AR4,@_b               ; |10|
        MOV       AH,@_x                ; |10|
        LCR       #_slope               ; |10|
        ; call occurs [#_slope] ; |10|
        MOVW      DP,#_y
        MOV       @_y,AL                ; |10|
        MOVB      AL,#0
      .line 9
        LRETR
        ; return occurs
      .endfunc    12,000000000h,0
```

Figure 8. Code Generated From Interlisting (Continued)

The code listed in Figure 9 is the_slope function that is generated by the compiler. This code is the basis for the C-callable assembly routine that is developed in this application note and is crucial to the creation of C-callable assembly functions.

```
      .sect ”.text”
      .global     _slope
      .sym  _slope,_slope, 36, 2, 0
      .func 14
;-----------------------------------------------------------------------
;  14 | slope(m,x,b)
;-----------------------------------------------------------------------
;***************************************************************
;* FNAME: _slope                        FR SIZE:   4           *
;*                                                             *
;* FUNCTION ENVIRONMENT                                        *
;*                                                             *
;* FUNCTION PROPERTIES                                         *
;*                            0 Parameter,  4 Auto,  0 SOE     *
;***************************************************************
_slope:
      .line 2
;* AL    assigned to _m
      .sym  _m,0, 4, 17, 16
;* AH    assigned to _x
      .sym  _x,1, 4, 17, 16
;* AR4   assigned to _b
      .sym  _b,10, 4, 17, 16
      .sym  _m,-1, 4, 1, 16
      .sym  _x,-2, 4, 1, 16
      .sym  _b,-3, 4, 1, 16
      .sym  _y,-4, 4, 1, 16
        ADDB      SP,#4
;-----------------------------------------------------------------------
;  16 | int y;
;-----------------------------------------------------------------------
        MOV       *-SP[1],AL            ; |15|
        MOV       *-SP[2],AH            ; |15|
        MOV       *-SP[3],AR4           ; |15|
      .line 4
;-----------------------------------------------------------------------
;  17 | y = m*x + b;
;-----------------------------------------------------------------------
        MOV       T,*-SP[2]             ; |17|
        MPY       ACC,T,*-SP[1]         ; |17|
        ADD       AL,*-SP[3]            ; |17|
        MOV       *-SP[4],AL            ; |17|
      .line 5
```

Figure 9. _slope Function Generated by the Compiler

```
;----------------------------------------------------------------------
;  18 | return(y);
;----------------------------------------------------------------------
      .line 6
        SUBB      SP,#4                 ; |18|
        LRETR
        ; return occurs
      .endfunc    19,000000000h,4
;***************************************************************
;* TYPE INFORMATION                                            *
;***************************************************************
```

Figure 9. _slope Function Generated by the Compiler (Continued)

2.6 Creating a .asm File With Code From the Interlisted Function

This is the sixth step in creating a C-callable assembly function.

The following is an assembly language shell based on the information that was gathered from the interlisted “slope” function. In this case, the processor uses Accumulator high and Accumulator low, along with AR4 to pass parameters.

```
;***************************************************************
;* FNAME: _slope                        FR SIZE:   4           *
;*                                                             *
;* FUNCTION ENVIRONMENT                                        *
;*                                                             *
;* FUNCTION PROPERTIES                                         *
;*                            0 Parameter,  4 Auto,  0 SOE     *
;***************************************************************
_slope:
;* AL    assigned to _m
;* AH    assigned to _x
;* AR4   assigned to _b
       ADDB      SP,#4
;----------------------------------------------------------------------
;  16 | int y;
;----------------------------------------------------------------------
       MOV       *-SP[1],AL            ; |15|
       MOV       *-SP[2],AH            ; |15|
       MOV       *-SP[3],AR4           ; |15|

;----------------------------------------------------------------------
;  17 | y = m*x + b;
;----------------------------------------------------------------------
       MOV       T,*-SP[2]             ; |17|
       MPY       ACC,T,*-SP[1]         ; |17|
       ADD       AL,*-SP[3]            ; |17|
       MOV       *-SP[4],AL            ; |17|
```

```
;----------------------------------------------------------------------
;  18 | return(y);
;----------------------------------------------------------------------

        SUBB      SP,#4                 ; |18|
        LRETR
```

This assembly language file was created from the “generated assembly” code of the compiler and it can be optimized. Since the information is already passed via AL, AH, and AR4, there is no need to deal with the stack. The routine is modified below so that the stack is never used.

```
;*************************************************************
;* FNAME: _slope                                              *
;*************************************************************
      .def           _slope
_slope:
;* AL    assigned to _m
;* AH    assigned to _x
;* AR4   assigned to _b
;----------------------------------------------------------------------
;  y = m*x + b;
;----------------------------------------------------------------------
        MOV       T,AL                  ; Load T with _m.
        MPY       ACC,T,AH              ; Multiply by _x
        ADD       ACC,AR4               ; Add _b
;----------------------------------------------------------------------
;  return(y);
;----------------------------------------------------------------------
       LRETR                           ; Return with value in ACC
```

2.7 Defining the Function Prototype as External

This is the seventh step in creating a C-callable assembly function.

```
extern int slope(int,int,int);
```

The function prototype in the C code is now updated with an “extern” statement to make it external. At this point, remove the C–function that was created to generate the shell.

The final C code looks like this:

```
   int y;
   int m,x,b;

main()
{
   extern int slope(int,int,int);

   y=0;
   x=1,b=2,m=3;
   y=slope(b,m,x);

}
```

2.8 Adding the Assembly File to the Code Composer Studio Project

This is the final step in creating a C-callable assembly function.

At this point, a C-callable assembly routine file has been created. The file must be added to your project in order to be referenced by the C module. The assembly file is referenced in the C module as an EXTERN; therefore, an error will occur during the project build if it is not added.

3 Conclusion

A quick and efficient method for creating a C-callable assembly function has been presented for the TMS320C28xx DSP family. In the particular example shown here, the original C-function created 10 assembly language instructions. Re-writing this function as a C callable assembly function created only 4 instructions.

NOTE: There are NO guarantees that the C-callable Assembly always results in code savings this large.

If there is a function where multiple parameters are being passed, and some of the parameters by necessity end up on the stack, using this same procedure allows you to build a C-callable assembly function using stack addressing modes of the C28x processor.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265