Using C to Access Data Stored in Program Memory on the TMS320C54x DSP

David M. Alter *DSP Applications - Semiconductor Group*

ABSTRACT

Efficient utilization of available memory in a TMS320C54x DSP sometimes requires the placement of data in program space memory (as opposed to data space memory). However, accessing this data using the C programming language is problematic, since the C-compiler provides no mechanism for accessing program space. This application note presents a solution for accessing data stored in program space memory using C on the TMS320C54x DSP. A C-callable library of six assembly code functions for accessing data in program memory (including extended program memory) is also provided.

This application report contains project code that can be downloaded from this link. http://www-s.ti.com/sc/psheets/spra177a/spra177a.zip

Contents

Tables

Introduction

On TMS320C54x devices, it is sometimes desirable to place data in program space memory rather than in data space memory. This is especially true on TMS320C54x devices that support extended program space addressing. On such devices, programmers may exhaust the 64Kw of data space memory, but still have an excess of program memory available. When working in the C programming language however, it is not sufficient to simply link the data into the program space, as the C-compiler expects all constants (and variables) to be in data memory. Further complicating the problem is the fact that the C-compiler allocates only 16-bits for data pointers. Therefore, if the data variables are linked to extended program memory (i.e. above 64Kw), the compiler will not create a pointer containing the full 23-bit address.

This application reports presents a solution for accessing data in program memory using C. First, a fully relocatable approach for getting the C-compiler to generate a 23-bit pointer to program space is presented. Relocatable means that no hard-coding of extended program memory addresses or pages is used. Next, a small library of six C-callable assembly functions is given, along with example code showing usage of these functions.

Generating 23-bit Program Memory Pointers in C

The TMS320C54x Optimizing C-Compiler utilizes only 16-bits for data pointers (see Reference 1, revision F, page 5-6). This includes all labels and symbols that represent an address for any data variable, structure, etc. The 16-bit pointers are sufficient for accessing data located in the first 64Kw of program memory. However, in order to access data variables located in extended program memory (i.e. above 64Kw), a method is needed to get the compiler to generate at least a 23-bit pointer, which is the combined width of the 16-bit PC (program counter) plus the 7-bit XPC (extended program counter) in the C54x CPU. The 23-bit pointer can then passed to a C-callable assembly function, which can access the data at the specified program memory address and perform the desired task.

The trick to getting the C-compiler to generate a full 23-bit address is to declare the data symbol as a void function in the calling function. With the far memory model (compiler option -mf), the C-compiler maintains 23 bits of address for the function. The actual function pointer occupies 2 words (32-bits) in the DSP memory, although only 23 bits are significant (the upper 9 bits are zero).

The Code Download

Download Package Contents

A code download accompanies this application report. To obtain the download, go to the Texas Instruments website, http://www.ti.com, and type in the literature number of this application report in the search box. A description of each file in the download is given in Table 1.

Table 1. Description of Code Download Files

File Name	Description
.pfunc\makenear.bat	Windows batch file for building a PFUNC library with near memory model
.pfunc\makefar.bat	Windows batch file for building a PFUNC library with far memory model
.\pfunc\include\pfunc.h	C include file that supports the PFUNC libraries
.\pfunc\lib\pfunc.lib	near memory model function library
.\pfunc\lib\pfunc_ext.lib	far memory model function library
.\pfunc\src\blkread.asm	source file for PFUNC_blkRead() function
.\pfunc\src\blkwrite.asm	source file for PFUNC_blkWrite() function
.\pfunc\src\strread.asm	source file for PFUNC_strRead() function
.\pfunc\src\strwrite.asm	source file for PFUNC_strWrite() function
.\pfunc\src\wordread.asm	source file for PFUNC_wordRead() function
.\pfunc\src\wordwrite.asm	source file for PFUNC_wordWrite() function
.\example\example.pjt	C5000 Code Composer Studio v2.1 project file for the example
.\example\main.c	main() function for the example
.\example\table.asm	assembly code data table file for the example
.\example\vc5416.cmd	VC5416 DSP linker command file for the example (far memory model)
.\example\Debug\example.map	.map file from the pre-built example (far memory model)
.\example\Debug\example.out	pre-built example executable (far memory model)

The PFUNC Code Library

Six C-callable functions for manipulating data stored in program memory have been hand-coded in TMS320C54x mnemonic assembly language for efficiency. These functions are:

Table 2. PFUNC Library Functions

Function Name	Description
PFUNC_blkRead()	copies a block from program memory to data memory
PFUNC_blkWrite()	copies a block from data memory to program memory
PFUNC_strRead()	copies a string from program memory to data memory
PFUNC_strWrite()	copies a string from data memory to program memory
PFUNC_wordRead()	copies a word from program memory to data memory
PFUNC_wordWrite()	copies a word from data memory to program memory

The heart of each function uses the READA or WRITA mnemonic assembly function to access the data in program memory. Each function is described in greater detail in Appendix A of this application report.

To facilitate incorporation into the readers application, the functions have been packaged into two different libraries. The library *pfunc.lib* has been built using the near memory model. The library *pfunc_ext.lib* has been built using the far memory model. The reader should include the correct library into his Code Composer project. A header file *pfunc.h* has also been provided that contains a function prototype for each function. This file should be included in the C-source file of any function that will be calling a library function. The libraries have been built using the TMS320C54x Code Generation Tools v3.70 (included in C5000 Code Composer Studio v2.1).

Source code has been provided for each function in the event that the user needs to modify a function or would like a basis for creating new functions. The source files make use of the built-in assembler constant __far_mode to differentiate between far memory model and near memory model. The __far_mode constant is set by the ASM500 assembler tool and also the CL500 compiler shell. It is set to 0 when the near memory model is used (no -mf option) and set to 1 when the far memory model is used (-mf assembler or compiler option). The provided batch files *makenear.bat* and *makefar.bat* can be used to rebuild the near and far model PFUNC function libraries from the source files at a command prompt. Usage is:

```
makenear yourlib
```

or

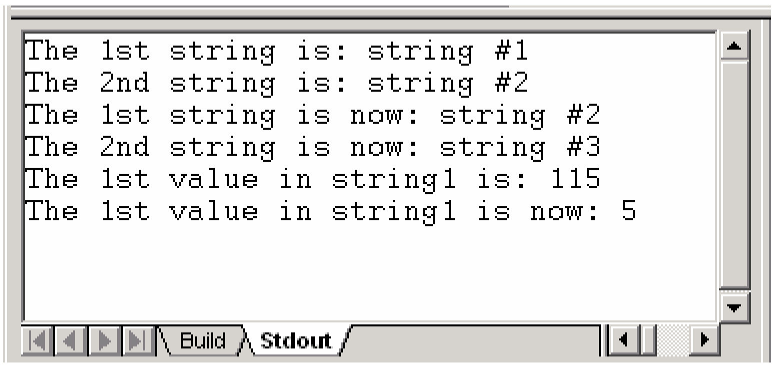
```
makefar yourlib
```

where "yourlib" is the name of the library you wish to create. The author chose the names *pfunc.lib* for the near model library, and *pfunc_ext.lib* for the far model library, but you may use any names you want. Alternately, you can simply include the modified source file directly into your Code Composer Studio project, and not build a library at all.

Example of PFUNC Library Use

The .\example directory contains an example of PFUNC function use. This example uses the far memory model, and hence uses the *pfunc_ext.lib* library. To use the example program, copy the file *.\pfunc\include\pfunc.h* into the *.\example* directory. Then, start Code Composer Studio and load the file *example.pjt* using the Project->Open menu command. You can then build and run the example. Note that the linker command file *vc5416.cmd* is designed for a VC5416 DSP target. If running on other than a VC5416 DSP (or the Code Composer Studio code simulator configured for VC5416 DSP), the user will need to modify the linker command file accordingly.

When the program is run, the Stdout window in Code Composer Studio should show the following:



```
The 1st string is: string #1
The 2nd string is: string #2
The 1st string is now: string #2
The 2nd string is now: string #3
The 1st value in string1 is: 115
The 1st value in string1 is now: 5
```

If you encounter difficulties either loading the project or building the code, it is most likely a path problem with your Code Composer Studio setup. A pre-built executable file *.\example\Debug\example.out* has been provided which should produce the above Stdout window when run on a VC5416 DSP. Use the File->Load Program menu command in Code Composer Studio to directly load this executable file.

Here is what the function main() does. First, it reads string1 and string2 using the PFUNC_strRead() function, and prints them to the Stdout window. Next, it overwrites string1 with the string2 using PFUNC_strWrite(), and then reads string1 back and prints it to Stdout. It next reads string3 using the PFUNC_blkRead() function, and overwrites string2 with string3 using PFUNC_blkWrite(). Note that the terminating null character of a string is included in the block length. Finally, main() reads the first 16-bit word in string1 using PFUNC_wordRead(), and then overwrites this word in string1 with the value "5." The first word in string1 originally had the value "115," which is the numerical value of the ASCII character 's', the first letter in the word 'string.'

Note that the string tmp_string must be declared of sufficient length to hold the copied program memory strings. The PFUNC functions do not cross-check source and destination lengths, and will overwrite other data if the destination length is smaller than the source string length or block size.

Constructing the Program Memory Data Values

The program memory data values are best constructed using assembly language. The file *table.asm* shows an example of how to do this. This particular example shows three strings which are to be stored in program memory. If non-string data is desired (i.e. 16-bit words), simply use the .int directive in place of the .char directive when declaring the data. The .sect directive places this data in the initialized section called "table". The "table" section is linked to program memory in the linker command file *vc5416.cmd*. The .def directive allows the named labels to be accessed by code in other source files.

Note that the terminating zero has been manually added to the strings, since termination of strings by a trailing zero (null character) is a C-language convention. The assembler does not automatically add the zero. If non-string data is being stored in program memory, the terminating zero is not needed.

References

1. *TMS320C54x Optimizing C Compiler User's Guide* (SPRU103)
2. *TMS320C54x DSP Mnemonic Instruction Set* (SPRU172)

Appendix A. PFUNC Function Library Technical Reference

Note: The PFUNC data type is defined in the file *.\include\pfunc.h*. It is used as a matter of convenience for function prototyping, since passed parameters of type PFUNC must be declared as void function pointers, and not as PFUNC data types in the calling function.

PFUNC_blkRead — Copies a block from (extended) program memory to data memory

Function

```
void PFUNC_blkRead(
     PFUNC addrProg,
     int *ptrData,
     unsigned int length
);
```

Arguments

addrProg	address of (extended) program memory source block
PtrData	pointer to data memory destination block

Return Value None

Description

Copies a block of 16-bit words from (extended) program memory to data memory. The source code for this function uses the built in assembler constant __far_mode such that it can be assembled for either the near or far memory model. If using the PFUNC libraries, *pfunc.lib* is for near memory model, and *pfunc_ext.lib* is for far memory model.

This function is similar to PFUNC_strRead except that PFUNC_strRead uses the terminating null character in a string to mark the end of the block, whereas this function passes the length of the block as a parameter.

Example

```
#define  N  20
extern void addrProg(void);
int ptrData[N];
int length = N;
PFUNC_blkRead(addrProg, ptrData, length);
```

PFUNC_blkWrite — Copies a block from data memory to (extended) program memory

Function

```
void PFUNC_blkWrite(
     PFUNC addrProg,
     int *ptrData,
     unsigned int length
);
```

Arguments

addrProg	address of (extended) program memory destination block
PtrData	pointer to data memory source block

Return Value

None

Description

Copies a block of 16-bit words from data memory to (extended) program memory. The source code for this function uses the built in assembler constant __far_mode such that it can be assembled for either the near or far memory model. If using the PFUNC libraries, *pfunc.lib* is for near memory model, and *pfunc_ext.lib* is for far memory model.

This function is similar to PFUNC_strWrite except that PFUNC_strWrite uses the terminating null character in a string to mark the end of the block, whereas this function passes the length of the block as a parameter.

Example

```
#define  N  20
extern void addrProg(void);
int ptrData[N];
int length = N;
PFUNC_blkWrite(addrProg, ptrData, length);
```

PFUNC_strRead — Copies a string from (extended) program memory to data memory

Function

```
void PFUNC_strRead(
     PFUNC addrProg,
     int *strData,
);
```

Arguments

addrProg	address of (extended) program memory source string
PtrData	pointer to data memory destination string

Return Value

None

Description

Copies a string from (extended) program memory to data memory. The source code for this function uses the built in assembler constant __far_mode such that it can be assembled for either the near or far memory model. If using the PFUNC libraries, *pfunc.lib* is for near memory model, and *pfunc_ext.lib* is for far memory model.

This function is similar to PFUNC_blkRead except that PFUNC_blkRead passes the length of the block as a parameter, whereas this function uses the terminating null character in a string to mark the end of the block.

Example

```
#define  N  20
extern void addrProg(void);
char strData[N];
PFUNC_strRead(addrProg, strData);
```

PFUNC_strWrite — Copies a string from data memory to (extended) program memory

Function

```
void PFUNC_strWrite(
    PFUNC addrProg,
    int *strData,
);
```

Arguments

addrProg	address of (extended) program memory destination string
PtrData	pointer to data memory source string

Return Value

None

Description

Copies a string from data memory to (extended) program memory. The source code for this function uses the built in assembler constant __far_mode such that it can be assembled for either the near or far memory model. If using the PFUNC libraries, *pfunc.lib* is for near memory model, and *pfunc_ext.lib* is for far memory model.

This function is similar to PFUNC_blkWrite except that PFUNC_blkWrite passes the length of the block as a parameter, whereas this function uses the terminating null character in a string to mark the end of the block.

Example

```
#define N 20
extern void addrProg(void);
char strData[N];
PFUNC_strWrite(addrProg, ptrData);
```

PFUNC_wordRead Copies a word from (extended) program memory to data memory

Function	int PFUNC_wordRead(
	    PFUNC addrProg
	);

Arguments	addrProg address of (extended) program memory source word
Return Value	wordData destination word in data memory
Description	Copies a single 16-bit word from (extended) program memory to data memory. The source code for this function uses the built in assembler constant __far_mode such that it can be assembled for either the near or far memory model. If using the PFUNC libraries, *pfunc.lib* is for near memory model, and *pfunc_ext.lib* is for far memory model.

Example	extern void addrProg(void);
	int wordData;
	wordData = PFUNC_strRead(addrProg);

PFUNC_wordWrite — Copies a word from data memory to (extended) program memory

Function

```
void PFUNC_wordWrite(
     PFUNC addrProg
     int wordData
);
```

Arguments

addrProg	address of (extended) program memory destination word
wordData	source word in data memory

Return Value

none

Description

Copies a single 16-bit word from (extended) program memory to data memory. The source code for this function uses the built in assembler constant __far_mode such that it can be assembled for either the near or far memory model. If using the PFUNC libraries, *pfunc.lib* is for near memory model, and *pfunc_ext.lib* is for far memory model.

Example

```
extern void addrProg(void);
int wordData;
PFUNC_strRead(addrProg, wordData);
```

Revision History

SPRA177A – July, 2005. Code change only. Fixed bug in PFUNC_blkWrite(). Changed section name "table" to "mytable" in table.asm since "table" is a reserved name in Code Composer Studio v3.1.

SPRA177 – March, 2002. Original

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265