Application Report
SPRA380 – April 2002

Using C to Access Data Stored in Program Space Memory on the TMS320C24x DSP

David M. Alter — *DSP Applications - Semiconductor Group*

ABSTRACT

Efficient utilization of available memory in a TMS320C24x™ DSP sometimes requires the placement of data in program space memory (as opposed to data space memory). In particular, the on-chip flash (or ROM) in the program space provides a large nonvolatile memory for storing constant arrays, look-up tables, and string tables. 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 C-callable library of six assembly code functions for accessing data in program space memory. These functions work with all members of the C24x™ DSP generation, including all C/F24x, LC/LF240x, and LC/LF240xA devices.

Note: These functions cannot be used to program the on-chip flash memory. Programming the flash requires using one of several different flash programming utilities. These utilities are available on the Texas Instruments (TI) website, www.ti.com.

Contents

Tables

Introduction

On TMS320C24x devices, it is sometimes desirable to place data in program space memory rather than in data space memory. In particular, the on-chip flash (or ROM) in the program space provides a large nonvolatile memory for storing constant arrays, look-up tables, and string tables. 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 space memory. No mechanism exists in the C-compiler for accessing program space memory, other than at code-initialization time.

One method for overcoming this problem is to copy the data from the flash (or ROM) into data space RAM as part of the code-initialization process. This could be done using a custom assembly code routine, or the C-compiler does provide some built-in capability for initializing global and static variables and constants. The C-code could then access the copies of the data in the data space during code execution. The downside of this approach is that each word of data now consumes two words of memory: the original data in the flash, and a copy of the data in data space RAM. Large arrays of constants or string tables can quickly use up the valuable on-chip RAM available in C24x generation DSPs.

A better approach is to temporarily copy the value of interest from the flash to data RAM only when it is needed at run time. C-code can then access the temporary copy (e.g., as a local variable located on the software stack), and dispatch the value as required. This approach avoids the double memory usage problem at the expense of using some CPU clock cycles to temporarily copy the data from program memory each time the value is accessed at run time.

This application reports utilizes this temporary copy method, and presents a C-callable library of six assembly code functions for accessing data in program memory. A simple C-code example illustrating their usage is also provided. These functions work with all members of the C24x DSP generation, including all C/F24x, LC/LF240x, and LC/LF240xA devices.

The Code Download

Download Package Contents

A code download accompanies this application report. This download may be obtained from the Texas Instruments website, www.ti.com. A description of each file in the download is given in Table 1.

Table 1. Description of Code Download Files

File Name	Description
.pfunc\make.bat	Windows batch file for building the PFUNC library
.\pfunc\include\pfunc.h	C language header file for PFUNC library functions
.\pfunc\lib\pfunc.lib	the 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.mak	C24x Code Composer v4.1x 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\lf2407a.cmd	TMS320LF2407A DSP linker command file for the example
.\example\example.map	.map file from the prebuilt example
.\example\example.out	prebuilt example executable

The PFUNC Code Library

Six C-callable functions for manipulating data stored in program space memory have been hand-coded in assembly language for efficiency. These functions are listed in Table 2.

Table 2. PFUNC Library Functions

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

The heart of each function uses the TBLR or TBLW assembly instruction to access the data in program space memory. Each function is described in greater detail in Appendix A of this application report.

It is important to understand that the three write functions cannot be used to program the on-chip flash memory of the flash devices in the C24x DSP generation. Programming the flash requires using device-specific flash programming utilities. These utilities are available on the TI website, www.ti.com. The three data write functions listed in Table 2 are mostly included in the library for completeness. There is no internal program RAM on current C24x generation devices that is not dual-mapped into data space, so it is unlikely that one would have much use for the write functions. The only conceivable exception to this would be if an external RAM (or peripheral device) were decoded only in the program space.

To facilitate incorporation into the reader's application, the functions have been packaged into a library called *pfunc.lib* (PFUNC stands for **p**rogram memory **func**tion). The reader should include this library into their Code Composer project. A header file, *pfunc.h*, which contains a function prototype for each function, has also been provided. 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 TMS320C2x/C2xx/C5x Code Generation Tools v7.00 (included in C24x Code Composer v4.1x).

Source code has been provided for each function should the user want to modify a function or would like a basis for creating new functions. The provided batch file *make.bat* can be used to rebuild the PFUNC function library from the source files at a command prompt. The usage is:

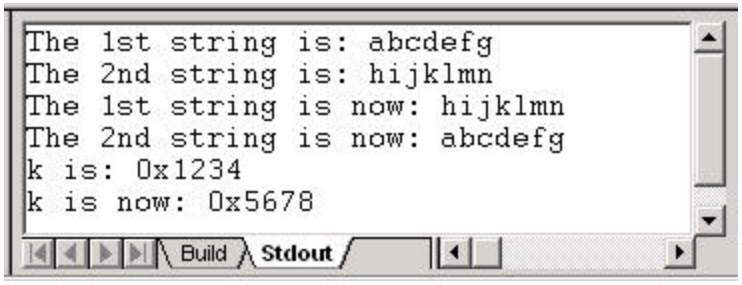
```
make yourlib
```

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

Example of PFUNC Library Use

The .\example directory contains an example of PFUNC function use. In reality, this example does not do much! It is really there just to provide an illustration of the coding needed to use the PFUNC functions. The example may be run on the LF2407 or LF2407A evaluation module (EVM) with a JTAG emulator attached, and C24x Code Composer v4.1x running on the host PC. To use the example program, copy the file *.\pfunc\include\pfunc.h* into the *.\example* directory. Then, start Code Composer and load the file *example.mak* using the Project->Open menu command. You can then build and run the example.

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



If you encounter difficulties either loading the project or building the code, it is most likely a file path problem. A prebuilt executable file *.\example\example.out* has been provided which should produce the above Stdout window when run on a LF2407 or LF2407A EVM. Use the File->Load Program menu command in Code Composer to directly load this executable file, and then run the program.

Here is what main() does. First, it reads string1 and string2 using the PFUNC_strRead() and PFUNC_blkRead() functions respectively, and prints them both to the Stdout window. Next, it overwrites string1 with the string2, and string2 with string1, using PFUNC_strWrite() and PFUNC_blkwrite(), respectively. It then reads both strings back and prints them to the Stdout window. It next reads the single word at the label "k" using PFUNC_wordRead(), and prints its value to the Stdout window. Finally, it overwrites "k" with a different value using PFUNC_wordWrite(), and then reads it back and prints it to the Stdout window.

Note that the temporary arrays (e.g., tmp1[ ] and tmp2[ ]) must be declared of sufficient length to hold the copied program memory strings or blocks. 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. Also, PFUNC_blkRead() and PFUNC_blkWrite() are designed to read blocks of data type int, but here they are being used to read strings (which are blocks of data type char). Therefore, typecasting has been used in the function calls for each.

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 two strings and one 16-bit word that are to be stored in program memory. The "table" section is linked to program space memory in the linker command file *lf2407a.cmd*. The .def directive allows the named labels to be accessed by code in other source files. Additional information on the assembly language elements used may be found in the *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide (SPRU018).*

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. When non-string data is being declared (e.g., the word at the symbol k), the terminating zero is not used.

References

1. *TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide* (SPRU024)
2. *TMS320F/C24x DSP Controllers Reference Guide: CPU and Instruction Set* (SPRU160)
3. TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide (SPRU018).

Appendix A. PFUNC Function Library Technical Reference

PFUNC_blkRead	Copies a block of words from program space to data space

Function

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

Arguments

addrProg	source block address in program space
ptrData	pointer to data space destination block

Return Value

None

Description

Copies a block of 16-bit words from program space to data space.

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 int addrProg;
int ptrData[N];
unsigned int length = N;
PFUNC_blkRead(&addrProg, ptrData, length);
```

PFUNC_blkWrite Copies a block of words from data space to program space

Function

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

Arguments

addrProg	destination block address in program space
ptrData	pointer to data space source block

Return Value

None

Description

Copies a block of 16-bit words from data space to program space RAM.

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.

This function cannot be used to write to the on-chip flash memory. Programming the flash requires using one of several different flash programming utilities. These utilities are available on the TI website, www.ti.com.

Example

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

PFUNC_strRead Copies a string from program space to data space

Function

```
void PFUNC_strRead(
    char *addrProg,
    char *strData,
);
```

Arguments

addrProg	source string address in program space
strData	pointer to data space destination string

Return Value

None

Description

Copies a string from program space to data space.

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 char addrProg;
char strData[N];
PFUNC_strRead(&addrProg, strData);
```

PFUNC_strWrite	Copies a string from data space to program space

Function

```
void PFUNC_strWrite(
    char *addrProg,
    char *strData,
);
```

Arguments

addrProg	destination string address in program space
strData	pointer to data space source string

Return Value

None

Description

Copies a string from data space to program space.

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.

This function cannot be used to write to the on-chip flash memory. Programming the flash requires using one of several different flash programming utilities. These utilities are available on the TI website, www.ti.com.

Example

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

PFUNC_wordRead Copies a word from program space to data space

Function

```
int PFUNC_wordRead(
    int *addrProg
);
```

Arguments

addrProg source word address in program space

Return Value

wordData destination word in data space

Description

Copies a single 16-bit word from program space to data space.

Example

```
extern int addrProg;
int wordData;
wordData = PFUNC_wordRead(&addrProg);
```

PFUNC_wordWrite Copies a word from data space to program space

Function

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

Arguments

addrProg	destination word address in program space
wordData	source word in data space

Return Value none

Description Copies a single 16-bit word from data space to program space.

This function cannot be used to write to the on-chip flash memory. Programming the flash requires using one of several different flash programming utilities. These utilities are available on the TI website, www.ti.com.

Example

```
extern int addrProg;
int wordData;
PFUNC_wordWrite(&addrProg, wordData);
```

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