GETTING STARTED WITH THE TMS320F24x PROCESSOR

Tutorial 1: A First Program

New Instructions Introduced

```
B
CLRC
```

New Flag Introduced

```
XF
```

Introduction

This series of tutorials has been designed to take the first time user of the TMS320F24x family of DSP devices through some of the commonly used instructions and peripherals.

Simple programs are given, which can be used on TMS320F243 DSP Starter Kit (DSK). Many of the principles and sample code can also be applied to other devices in the ‘2000 family such as the TMS320LF2407.

When the tutorials have been completed, the reader will be able to use the TMS320F24x to perform real-world tasks, such as to take analog and digital inputs, filter them and use them to control outputs such as motor speed, brightness of lights etc.

Assumptions

These tutorials assume no previous experience of assembly language programming or digital signal processing (DSP). However, some basic knowledge of high-level programming is assumed because several of the examples use C code. Some basic electronics knowledge would also be useful for connecting external components to the DSK.

As far as using the TMS320F243 DSK is concerned, it is assumed that the hardware and software have been successfully installed and that the user understands the basic operation e.g. how to load the sample program, single-step it and run it, how to read the CPU registers etc.

For those readers without the TMS320F243 or other DSK, then this tutorial can be used for general information and to supplement the Texas Instruments Databooks, references to which are given at the end of the tutorial.

Text Editor

The TMS320F243 DSK does not provide a text editor. However, a suitable text editor is supplied as standard with Microsoft Windows, namely Notepad. Alternately, one of the many programmer's editors (Brief, CodeWright) etc. can be used, as can the integrated development environment (IDE) from a C/C++ compiler. Note that a word-processor should not be used because it puts control characters into the text; these cannot be processed.

Writing a First Program

We shall start by writing a very simple first program. Using a text editor, carefully type in the two lines as shown in Example 1-1. The word `start` must go in the first column.

Example 1-1.

```
        .setsect ".text",   8800h

start:  B start             ; Branch to the label start.
```

Save the program as `myprog1.asm` in the same directory as the program `simple.asm`, as supplied with the TMS320F243 DSK e.g.:

```
C:\specdig\dsk24x\sd24xasm\myprog1.asm
```

Should the TMS320F243DSK have been installed in a different directory then the path will need to be changed accordingly.

Upper Case and Lower Case

Because the assembler supplied with the TMS320F243 DSK is not generally case sensitive, we could equally well have written the program in Example 1-1 as:

Example 1-2.

```
        .SETSECT ".TEXT",   8800H

START:  B START             ; BRANCH TO THE LABEL START.
```

In these tutorials we will remain with the convention used in Example 1-1. That is, we will use upper case for instructions and lower case for other information. This maintains compatibility with the Texas Instruments databooks.

Assembling the Program

We have entered and saved our first program `myprog1.asm` where the extension `.asm` denotes that the program is an assembly language file. This needs to be converted into a `.dsk` file the TMS320F243 DSK can use. To do this we need to run the executable file `sd24xasm.exe` supplied with the DSK. This is the *assembler*.

To assemble our saved program, we go into MS-DOS and at the C:\ prompt type:

```
C:\specdig\dsk24x\sd24xasm\sdx24asm.exe -l myprog1.asm
```

If all is well and there are no typographical errors, then a message such as the following will be seen:

```
C:\specdig\dsk24x\sd24xasm>sd24xasm -l myprog1.asm
SDI24xasm Assembler             Version 1.0.1   1999.
Copyright (c) 1999              Spectrum Digital Incorporated.
  PASS 1
  PASS 2

No Errors,  No warnings

C:\specdig\dsk24x\sd24xasm>
```

Care is required when entering the text. In the event of incorrect typing, an error or warning will be generated. For example, if we were to enter Example 1-1 and inadvertently miss out the `t` at the end of the word `start`, a message such as the following would be generated:

```
C:\specdig\dsk24x\sd24xasm>sd24xasm -l myprog1.asm
SDI24xasm Assembler             Version 1.0.1   1999.
Copyright (c) 1999              Spectrum Digital Incorporated.
  PASS 1
  PASS 2
start:  B star ; Branch forever to label start
"myprog1.asm", line 3:  Invalid addressing mode or undefined label

 1 Error,  No warnings
 Errors in source. Object file not generated.

C:\specdig\dsk24x\sd24xasm>
```

When an error occurs, the assembler tells us where the problem is. We must then go back to the text editor, make the necessary corrections and assemble again.

Note that the assembler has used the word `source`, which refers to the program file being assembled, here `myprog1.asm`. The word `object` refers to the output file produced by the assembler.

Automating the Assembly Process

In the course of a project, our program will need to be assembled many times.
To save time when assembling programs, it is worth setting up a batch file such as the following:

```
sd24xasm.exe -l %1.asm
```

Enter above line and save it as `C:\specdig\dsk24x\sd24xasm\a.bat`

In order to assemble `myprog1.asm`, then in the directory

`C:\specdig\dsk24x\sd24xasm>` we type:

```
a myprog1
```

When we do this, the above line expands to:

```
sd24xasm.exe -l myprog1.asm
```

This will cause the assembler to run in a similar fashion to the version written out in full. Note that there is no need to type in `.asm` at the end of `myprog1`.

Some programmer's editors provide facilities to assemble the program from the editor itself. In this case, the batch file is not required.

Analysis of the Program

So far we have typed in a minimal assembly language program, but what does it actually do?

The first line `.setsect ".text", 8800h` tells the assembler that we want the instructions to be put into the segment named `".text"`. The word `".text"` is slightly misleading in that it is where we put the instructions, rather than just text. The value `8800h` is the address in program memory (code) where we want the program to start and the letter `h` means hexadecimal.

If this line is missed out, the assembler will create its own segment named `".text"` but beginning at 0 in code - not where we run the program. This is outside the program memory space of the TMS320F243 DSK, and hence the program will not run.

The second line

`start: B start` is the actual code to be executed and consists of three elements.

The term `start:` is a *label* and this must be at the beginning of a line. Any word in the first column is taken to be a *label*. A *label* is a name of a particular place in the program, rather like the name of a town on a map. The colon `:` after the label is optional.

The letter `B` is an abbreviation of the word *branch*. This is the *instruction* to be executed and tells the DSP chip what to do. Here `B` (branch) carries out the same operation as the GOTO in BASIC/C.

The word `start` following the instruction `B` is the *operand*. An operand is a piece of additional information required by the instruction. In this case, the operand taken by the instruction `B` (branch) is *where* the program is to go to next.

Although the assembler is not case sensitive, we cannot use a lower-case *label* with an upper-case *operand*:

```
start: B START ; Incorrect. Label and operand different.
```

In this example, the assembler will look for the uppercase label `START,` which it will not be able to find, and hence generate an error.

Finally, the words preceded by a semi-colon `;` are what are known as *comments*. Comments are for the reader's benefit only and are ignored by the assembler. They provide information as to what is going on in the program. It is good practice to comment the code. Consider the case where we write a program then come back to it six months later, having completely forgotten what we have written.

The List File

When we typed in

```
sd24xasm.exe -l myprog1.asm
```

we typed in `-l` . Here the `-l` is used to specify the option of the assembler, which causes a *list* file to be produced.

When the program `myprog1.asm` is assembled, the assembler produces the additional list file

```
myprog1.lst
```

The extension `.lst` means *list*. The *list* file tells us how the assembler interpreted our source file `myprog1.asm.` The details of the list file `myprog1.lst` are as follows:

```
SDI24xASM Assembler        Version 1.0.1   Sun Mar 12 14:23:50 2000
Copyright (c) 1999         Spectrum Digital Incorporated

myprog1.asm
PAGE     1

0001                      .setsect ".text", 8800h
0002
0003   8800 7980 start:  B start ; Branch to label start
0003   8801 8800'
0004

 No Errors, No Warnings
```

The first column gives line numbers 0001 to 0004.

The second column gives the addresses in code where the instructions are placed. For example, the instruction `B` is at address 8800h.

Finally, the third column gives the number into which an instruction is converted. For example, the instruction `B` (branch) is converted to the number `7980h` and the operand `start` is converted to the number `8800h`. Although the numbers may not mean a lot to us, they are understood by the TMS320F243 DSK.

The .DSK File

When the program `myprog1.asm` is assembled, the assembler also produces the file

`myprog1.dsk`

This file `myprog1.dsk` is loaded into the GO-DSP debugger. It is the code in format that the debugger can understand. It contains the code plus some extra information.

```
K_D203_1.01_myprog1.asm
1880078800F
98800B7980B880070180F
:
```

Creating a Template

The code given in Example 1-1 is the minimum code that can be used for a program.

We can put some additional information around it and save it as a template for future use. The additional information could include name of program, date, what the program does, revisions, uses of physical pins etc.

Some examples of templates can be found in the Texas Instruments databooks, details of which are given in the References section of this tutorial.

Many organizations have what is known as a *coding standard*. This is a document, which gives a list of what are considered good and bad programming practices as applied to that organization.

Other Assemblers

The assembler supplied with the TMS320F243 DSK is a simple two-pass assembler. This is quick and easy to use. The limitation of this is that only a single module of code can be used. The assembler only accepts the mnemonics found in the TMS320F/C24x DSP Controllers, CPU and Instruction Set and does not support instructions from earlier devices.

For industrial projects, Texas Instruments provides what is known as a COFF assembler (common output file format). This supports large multi-module programs and allows the usage of instructions from other devices such as the TMS320C2x family.

LEDS

On the TMS320F243 DSK there are two light emitting diodes (LEDs). One is for the power-supply and is always on. The second is available to the user and can be controlled by software. This is the XF LED.

The XF LED on the DSK is driven from a physical pin on the TMS320F243 device. This is the XF pin (external flag).

Turning off the XF LED

At power-up, the XF LED is on. In order to turn the XF LED off, we need to execute the following instruction:

Example 1-3.

	`CLRC XF`	`; Turn off the XF LED.`

In Example 1-3 we have used the new instruction `CLRC XF`. The letters `CLRC` are an abbreviation of *clear control flag*. Here the word *clear* means make 0.

The instruction `CLRC` takes a single operand, which is the name of a specific control flag. Here we have used the operand `XF` (external flag). After this instruction is executed, the state of the XF flag will be 0. In later tutorials we shall use the instruction `CLRC` to control other flags.

The method of writing an instruction such as `CLRC` is referred to as a *mnemonic*. It is simply a short way of writing the instruction, which aids memory.

In order to write an entire program that will turn off the XF LED we write:

Example 1-4.

	.setsect ".text",	8880h
start:	CLRC XF	; Turn off the XF LED.
	B *start*	; Branch to the label start.

TMS320F243 DSK EXPERIMENTS

Equipment Required

TMS320F243 DSK

Experiment 1-1.

Objective: To run the first program

Follow the instructions earlier in this tutorial to enter Example 1-1 into a text editor, save it and assemble it. Load the file `myprog1.dsk` into the TMS320F243 DSK and run the program.

Both the Power LED and the XF LED should be on and remain on.

Experiment 1-2.

Objective: To turn off the XF LED

Enter the code shown in Example 1-3 into a text editor. Save the program as `myprog2.asm` and assemble it using `sd24xasm.exe`. Load `myprog2.dsk` into the TMS320F243DSK but *do not* run it.

When the program is loaded in the DSK, the XF LED will be on. If the StepInto, StepOver or Run Button is now clicked upon, then the XF LED will go off and stay off.

When the instruction `CLRC XF` is executed, the physical output pin XF on the TMS320F243 is pulled to low, which turns off the XF LED.

Design Task 1-1.

Using some examples from the databooks given in the References section of this tutorial, develop a standard template that can be used as a starting point for a program. As a minimum the template should contain the name of the program, date, author and purpose of code.

Self-Test Questions..... *Click Here To View* Answers!

1.	When editing a program, why do we not use a word processor?
2.	The extension .asm at the end of a file name means which of the following? a) All smashed and mangled b) Always save in memory c) Any Sunday or Monday d) Assembly language
3.	What is an *assembler*?
4.	The instruction `B` means which of the following?
	a) Block b) Borrow c) Branch d) Break
5.	What is an *operand*?
6.	Why does the following line of code not assemble? `start: B START ; Branch to label start.`
7.	What is a *list* file?
8.	The instruction `CLRC` means which of the following?
	a) Clear Carry b) Clear Control Bit c) Clear Reverse Carry d) Create LR checksum
9.	What is wrong with the following line of code? `CLC XF`
10.	What is the *external flag* (XF)?
11.	What is a *mnemonic*?

12.	Correct the three errors in the following code:
	`.setsect "text", 8800h`
	`start: CLR XF`
	`B start:`

References

TMS320F/C24x DSP Controllers. CPU and Instruction Set. Reference number SPRU160.

TMS320F/C240 DSP Controllers Peripheral Library and Specific Devices. Reference Number SPRU161.

TMS320F243, TMS320F241 DSP Controllers. Reference Number SPRS064.

CLICK TO VIEW
Tutorials
1 2 3 4 5 6 7 8 9 10

Click here to view..... Route Map