Engineer-to-Engineer Note

ANALOG DEVICES Technical notes on using Analog Devices DSPs, processors and development tools Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Formatted Print to a UART Terminal with Blackfin® Processors

Contributed by Andreas Pellkofer

Rev 3 – June 21, 2012

Introduction

This Engineer-to-Engineer Note discusses advanced implementations of various output functions for the Analog Devices Blackfin® processor family. The VisualDSP++® development tools provide C-standard I/O support through functions such as printf and fprintf. The achieved output performance is sufficient enough for sending sporadic (debug) information to the Output window (console) of the VisualDSP++ IDDE. However, multiple prints can cause an immense decrease of the application's speed.

All derivatives of the Blackfin processor family feature the UART module, a PC-style industrystandard compatible peripheral. Additionally, the software overhead is very low. In principle, one single output pin is sufficient for the purpose. The speed is fast enough to send lots of information to a PC terminal application with minimal impact to the performance of the actual application.

B COM11 - PuTTY
,888800.
,588888880.
Y888888888L
YessessesL
8888888888
d8888888888888888888888888888888888888
188888888888888888888888888888888888888
] 88888888PP''''
]8888888P' .
,00000. Model: ADSP-BF518F]88888P ,00000088b.
,8888888p]8888P ,8888888888888
d88P'888[000' 0000000 000]888P ``' ,888P88888880
,888 J88P J88P d8P88 d88P888 ,88P,88P `P'' ,88P',888888888L d888088P' 888' ,8P,88 ,88P 88P d88b88P
d888088F'888',8P,88,88P 88P d88b88Pd88888[088888888P8888.
,8888888. J88P 88'd8P d88 '',88888P d8' d8888888888888888 88Pb.
d88P 888P 888' d8P 88[,88P,o_ d88888. ,8P d8888P' 88888P' P']8b_
,888']888'J88P d888888 d88'J88 ⁻ ,88Pd88b JP d888P 888P',op , d88P
d8888888P d88bo.,88P'Y8P 888o88P d88']88b d' ,8P'_odb ''',d8P _o8P ,P',,d8L
PPPPPPP' `PPPPP YPP PPP `PPPP' <ppp ,0888'="" ,p="" 0888="" 88088888.="" d888888.<="" td=""></ppp>
d' d88888888888882_0888882_0888882_08888888
idl: 0xD9200210C02E783A dP d8888888888888888888888888888888888
idh: 0xD4B6752D9D4B1E00 dd8',8888888888888888888888888888888888
d88P d888888888888888888888888888888888
File:/Common Code\BfDebugger.c Line: 555 Function: WelcomeMessage
Build Date: Jan 19 2010 Build Time: 14:27:57
Core Clock: 30000000 Hz
System Clock: 75000000 Hz
This is a standard message
This is an error message

Figure 1. UART Terminal output example

Copyright 2010-2012, Analog Devices, Inc. All rights reserved. Analog Devices assumes no responsibility for customer product design or the use or application of customers' products or for any infringements of patents or rights of others which may result from Analog Devices assistance. All trademarks and logos are property of their respective holders. Information furnished by Analog Devices applications and development tools engineers is believed to be accurate and reliable, however no responsibility is assumed by Analog Devices regarding technical accuracy and topicality of the content provided in Analog Devices Engineer-to-Engineer Notes.



This EE-Note provides three different ways to implement a powerful debug system, which keeps the look and feel of C-standard I/O functions. It includes software examples for currently available Blackfin evaluation boards. The three methods are described in the sections:

- Accessing the UART Directly
- Replacing the Standard Device
- Extending I/O Support to New Devices

It is assumed that the audience of this EE-Note is familiar with the C programming language and the I/O features which are implemented in the standard. For details, refer to [1].

To understand this EE-Note, it is necessary to work with the example code provided in the associated .zip file.

Preparations

All the methods described later will require the Blackfin processor's UART interface to be set up properly. The examples provide target-specific functions to do this. They can be found in the ADSP-BF5xx-UART.c and ADSP-BF60x-UART.c files, respectively. The functions are used to initialize the Blackfin ports, the UART, and the routines to send out buffer contents over the UART.

The associated header files, named ADSP-BF5**xx**-UART.h, contain information about the UART device number to be used, the DMA channel, and indicate whether autobaud detection is used instead of a hard-coded bitrate.

Another set of required sources is BfDebugger.c and BfDebugger.h. Their functions are described in the following sections.

Accessing the UART Directly

Direct access, the first method illustrated in this EE-Note, is based on a set of preprocessor macros and uses the string library (string.h) to

generate a formatted string for UART transmission.

The functions can be found in BfDebugger.c and BfDebugger.h and use the following prototypes:

```
short udprintf(unsigned char UartNum,
unsigned char DmaChan, const char
*format, /* args */ ...);
short uprintf(unsigned char UartNum,
const char *format, /* args */ ...);
```

The look and feel is like printf, which takes a variable list of arguments. The argument format contains a set of conversion specifiers, directives, and ordinary characters that are used to control how the data is formatted.

```
udprintf(0,9,"Hello UART%d\n",0);
uprintf(0,"Hello UART%d\n",0);
```

A carriage return and C-style NULL termination is inserted automatically. The first arguments are the UART number and the DMA channel number, respectively.

The function returns the number of characters transmitted, or a negative value if unsuccessful.

Preprocessor Definitions

To assist the programmer with a flexible support of the uprintf functions versus the regular printf functions, a set of preprocessor macros helps to switch between different modes. Use only one at the same time:

- __DEBUG_UART_DMA__ Directs prints to the UART, DMA mode
- __DEBUG_UART__ Directs prints to the UART, Core mode
- __DEBUG_VDSP__ Directs prints to the VisualDSP++ Output window (console)



The last two definitions are just an additional feature for completeness and use the standard fprintf() function.

• __VERBOSITY__ [int] Level of verbosity (LoV)

This definition allows masking or unmasking certain messages using a level hierarchy. So for example, ____VERBOSITY___ is defined as 2, so any messages using the preprocessor macros in the next section and have n set to 1 or 2 (DEBUG(1, args) or DEBUG(2, args)) are printed while 3 (DEBUG(3, args)) or higher are masked by the preprocessor.

Preprocessor Macros

A set of preprocessor macros adds additional functionality such as evaluation of a desired verbosity level and automatic color output. They insert, according to ADSP-BF5xx-UART.h., the correct UART number and DMA channel, respectively.

- DEBUG(n, *format, /* args */ ...) No color
- INFO(n, *format, /* args */ ...) Yellow text
- ERROR(n, *format, /* args */ ...) Red text
- MSG(n, *format, /* args */ ...) Green text

ANSI escape sequences are used to color the output. A list of available ESC codes can be found in [4].

Finally, the macros DEBUG_OPEN() and DEBUG_CLOSE(), both of which are required only once in the application, are used to prepare and close all required resources, including the UART setup.

```
DEBUG_OPEN();
DEBUG(1,"Hello UART%d\n",0);
ERROR(1,"This is an error message\n");
DEBUG_CLOSE();
```

C/C++ Run-Time I/O Library

This section discusses C/C++ run-time (CRT) I/O library features, which can be used to send debug information from within an application. Additional examples showing how to work with the I/O library can be found in [2] and [3].

Stdio.h Basics

The stdio.h header file defines a set of functions, macros, and data types for performing input and output. The standard functions for generating formatted output are the printf family. The printf function sends its arguments to stdout, which by default, is the VisualDSP++ console output window. The programmer can change the device or file to which stdout refers. The fprintf function writes to a file-pointer. The associated file is opened / closed on the PC by the fopen / fclose function.

File I/O Support

File I/O support is described in detail in [1]. By default, the standard C functionality is achieved through a device called PrimIO. It is set up and registered on startup by the CRT and handles the three standard files (stdin, stdout, and stderr) and any other stream to be opened by the user, unless the default device is changed.

Replacing the Standard Device

This method is radical. Any printf call will be formatted as usual by the I/O subsystem but directed to the _write function (see DevEntry structure, defined in the device.h header file), which is associated with your new device. The downside is that you lose the print-to-console functionality. The advantage is that existing source files may remain virtually unchanged. Only a few additions to the project are required.

The documentation describes what changes are required to devtab.c and primiolib.c. These files are prepared and included in the example projects already.



A modification to devtab.c is required if a device should be pre-registered when the CRT is setting up.

If a new device claims the standard streams instead of the PrimIO device, primiolib.c must be modified.

The BfDebugger.c file defines the device with the required function pointers:

struct DevEntry UartIODevice

The BfDebugger.h file provides the preprocessor definitions used to activate the replacing method:

- ___UART_IO_DEVICE___
- ___PRE_REGISTERING_DEVICE___

The following definitions determine whether the UART transfer is done by Core or DMA. Use only one at a time:

- ____DEBUG_UART_DMA____
- ____DEBUG_UART____

Now any printf output is redirected.

Extending I/O Support to New Devices

The last method is a bit more elegant than the previous one. It uses the possibility to extend I/O support to new devices, too. But now standard functionality is preserved.

The same sources as the previous example are used. Pre-registering of the device is not done.

```
int PrimDevId =
get_default_io_device();
int SecDevId =
add_devtab_entry(&UartIODevice);
set_default_io_device(SecDevId);
pUartIOFile = fopen("uart","w");
if(setvbuf(pUartIOFile,buf,_IOLBF,size
of(buf)) == -1) { return -1; }
```

set_default_io_device(PrimDevId);
fprintf(pUartIOFile,"Test\n");
printf("Test\n");
fclose(pUartIOFile);

The above example saves the current device ID (PrimDevId). Then a new device (UartIODevice) is added and marked as default. A fopen operation opens a dummy file on the default device. Immediately *afterwards*, a new buffer is required to be associated with the new device. Line buffering is preferred for console output. The previous default device can be restored, and writes can be performed to the new device. A printf output will still be seen to the VisualDSP++ Output window.

SDTIO System Service

The System Services, starting with VisualDSP++ Update 8, provide a similar approach for redirecting a printf to UART. The examples can be found in the specific Evaluation Board examples folder:

```
Services, stdio, char_echo
```

UART Direct Access vs. I/O Device

The downside of using an I/O device rather than accessing the UART directly is the DMA mode. Before updating the buffer, any ongoing DMA transfer must be finished first. This is ensured by a function named

UartDmaWaitForDmaDone.

Since the I/O system that prepares the buffer is called in the very first stage, polling of the DMA_DONE status bit must be performed immediately after starting the transfer. Accessing the UART directly allows moving the wait routine right before preparing the buffer to be transmitted.



Console Application

Figure 1 shows the console output with the Windows program PuTTY. See [5] for download details. This program can be used as an alternative terminal window as HyperTerminal® has been removed from Windows Vista®. The window must be set to at least 91 columns to show the "welcome screen" displayed in the picture.

The Blackfin application features a UART talkthrough mode based on interrupts. This means you can type characters in you console application window and get them bounced back and displayed on the screen.

CrossCore Embedded Studio® Support

The document and especially the attached ZIP file have been updated to support the new members of the Blackfin family, ADSP-BF60x processors. This processor family is only supported by the new IDE, CrossCore Embedded Studio® software. A dual core project running on the ADSP-BF609 ET-KIT Lite® Evaluation Board is included.

Conclusion

This EE-Note has shown some methods to create debug information and print them without the low-speed Output window (console) output of the VisualDSP++ debugger. The provided examples can be used as is on any Blackfin EZ-KIT Lite evaluation board and easily included in existing projects with similar targets.

References

- [1] *VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processors*. Rev. 5.2, September 2009. Analog Devices, Inc.
- [2] VisualDSP++ 5.0 Blackfin examples: Services\File System\VDK\shell_browser\. Analog Devices, Inc.
- [3] *VisualDSP++ 5.0 Blackfin examples: LAN\FileServerStdio*. Analog Devices, Inc.
- [4] ASCII-Table.com: http://ascii-table.com/ansi-escape-sequences.php
- [5] *PuTTY.org:* http://www.putty.org
- [6] CrossCore® Embedded Studio 1.0.0 C/C++ Compiler and Library Manual for Blackfin Processor. Rev. 1.0, March 2012. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – January 26, 2010 by Andreas Pellkofer	Initial release.
Rev 2 – December 10, 2010 by Andreas Pellkofer	Chapter Console Application updated. Example Code enhanced and extended to new Blackfin Evaluation Boards.
Rev 3 – June 21, 2012 by Andreas Pellkofer	Example Code enhanced and extended to new Blackfin Evaluation Boards. Support for ADSP-BF60x Blackfin Processor family added.