TEXAS INSTRUMENTS

TUSB2136/TUSB3210/ TUSB3410/TUSB5052

Firmware Debugging Guide

User's Guide

May 2004

MSDS Bus Solutions

SLLU027A

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.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

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

Preface

Read This First

About This Manual

The TI USB device controllers TUSB2136, TUSB3210, TUSB3410, and TUSB5052 do not support in-circuit emulation (ICE) for debugging. However, there are alternative methods that can be used by the firmware developer to aid the coding process. This document describes these alternative methods.

How to Use This Manual

This document contains the following chapters:

Chapter 1, *Introduction*, provides a general overview of the debugging process and environment.

Chapter 2, *Debug Strings*, has basic instructions and examples for outputting from the microcontroller strings containing information that is useful for debugging purposes.

Chapter 3, *In-System Debugging Using Keil's ISD51 Feature*, gives general information and specific pointers for the use of Keil debugging software with the TUSBxxxx.

Chapter 4, *File List*, contains a source code listing for each C routine and header file used for debugging.

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's. Examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

  Here is a sample program listing:

```
0011  0005  0001          .field    1, 2
0012  0005  0003          .field    3, 4
0013  0005  0006          .field    6, 3
0014  0006               .even
```

Here is an example of a system prompt and a command that you might enter:

```
C:  csr -a /user/ti/simuboard/utilities
```

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered. Here is an example of a directive syntax:

  .asect "*section name***",** *address*

  .asect is the directive. This directive has two parameters, indicated by *section name* and *address*. When you use .asect, the first parameter must be an actual section name, enclosed in double quotes; the second parameter must be an address.

- Square brackets (**[** and **]**) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you don't enter the brackets themselves. Here is an example of an instruction that has an optional parameter:

  LALK *16−bit constant [, shift]*

  The LALK instruction has two parameters. The first parameter, *16-bit constant*, is required. The second parameter, *shift*, is optional. As this syntax shows, if you use the optional second parameter, you must precede it with a comma.

  Square brackets are also used as part of the pathname specification for VMS pathnames; in this case, the brackets are actually part of the pathname (they are not optional).

- Braces ({ and }) indicate a list. The symbol **|** (read as *or*) separates items within the list. Here is an example of a list:

  ```
  { * | *+ | *− }
  ```

  This provides three choices: `*`, `*+`, or `*−`.

  Unless the list is enclosed in square brackets, you must choose one item from the list.

- Some directives can have a varying number of parameters. For example, the .byte directive can have up to 100 parameters. The syntax for this directive is:

  .byte $value_1$ *[, ... ,* $value_n$*]*

  This syntax shows that .byte must have at least one value parameter, but you have the option of supplying additional value parameters, separated by commas.

Related Documentation From Texas Instruments

TUSB2136 Universal Serial Bus Keyboard Hub Controller Data Manual, TI literature number SLLS442

TUSB3210 Universal Serial Bus General-Purpose Device Controller Data Manual, TI literature number SLLS466

Trademarks

Contents

Chapter 1

Introduction

TI's TUSB2136, TUSB3210, TUSB3410, and TUSB5052 are based on 8052 microcontroller cores, allowing the system designer much flexibility. There are many advantages to this. The 8052 is a common architecture, and therefore many programmers are already familiar with it. Also, many existing resources are available for the 8051/8052 architecture, including sample code. Any existing 8051/8052 development environment can be used to write and compile firmware for these devices, such as Keil's PK51 environment and IAR's Embedded Workbench.

Despite the simplicity of the 8052, it is still necessary to debug the code and to have the necessary tools and techniques for this task. Because of their customized pinout and lack of JTAG or hardwired debug capability, it is not possible to use existing in-circuit emulation (ICE) tools with TUSBxxxx. However, alternative methods can be used.

The first level of debug is the use of debug output strings on the UART interface. The second and more sophisticated approach is the use of the ISD51 feature included with Keil's 8051/8052 development tools. This document describes techniques for doing so.

Chapter 2

Debug Strings

A simple technique is to use the serial interface to output values and strings to the PC, which can help determine the status of the microcontroller unit (MCU) and internal memory.

To use this technique, the unit under test must have a serial port interface that is attached to a terminal (such as a PC running the Windows™ Hyperterminal™ application). Simple function routines that output values to the UART are added to the code and are called periodically within the code.

For example, to output the port-1 status of the 8052:

```
rs232PutHex(P1);
```

To print a string:

```
rs232PutString(”Hello World!”);
```

Source code for these routines is shown in Chapter 4. Note that the necessary header files must be included in the project file.

Chapter 3

In-System Debugging Using Keil's ISD51 Feature

The ISD51 in-system debugger feature in Keil's 8051 development tools provides a complete suite of debug functions for TUSBxxxx. This includes the following features:

- ❏ Single-step through the code
- ❏ Set breakpoints
- ❏ View/change CPU registers and memory
- ❏ Access the special function registers (SFRs)

ISD51 is available in Keil's C51 environment, version 6.23 and later. It is available only in the PK51 Professional Developer's Kit.

Keil provides full documentation for the feature, and this documentation should serve as the ultimate reference. For more details on ISD51 operation and additional features, see http://www.keil.com.

Topic **Page**

3.1 Operation

Use of ISD51 is straightforward.

1) Copy the ISD51.A51 and ISD51.H files, provided by Keil, to your project folder.
2) Add ISD51.A51 to the µVision2 project.
3) Add ISD51.H to the C module that contains the main C function.
4) Check the configuration settings in ISD51.H and modify them if necessary.
5) Add serial port and baud rate initialization code for the on-chip UART to your C *main* function.
6) Add an appropriate ISD51 startup function to your C code.

With ISD51, a serial interrupt function for the TUSBxxxx UART is added to your user program. When code is executed with the µVision2 debugger, the 8052 enters this interrupt function. While program execution is stopped, the 8052 program runs only the ISD51 interrupt, allowing communication with the debugger. When the debugger commands the interrupt function to resume, the 8052 exits the ISD51 interrupt function and executes the user program.

Code execution can be controlled using software breakpoints. If breakpoints are set, the 8052 enters the interrupt routine after each instruction, which checks whether a breakpoint address has been reached. If so, it connects to the µVision2 debugger; otherwise, it returns execution to the program.

3.2 Resource Requirements

ISD51 requires some of the system resources to execute:

- ❑ One of the on-chip UARTs
- ❑ About one-half Kbyte program code
- ❑ Six bytes stack space
- ❑ One byte of IDATA RAM
- ❑ Two bytes of IDATA RAM for each software breakpoint, defined top down from 0xFF

An effect of running ISD51 is a slowdown in execution speed. Software breakpoints, when used, slow execution on the order of 100×. For portions of code that are negatively affected by this slowdown or are speed-critical, ISD51 can be enabled/disabled programmatically, thereby allowing selective use of the feature.

3.3 Limitations

The ISD51 feature has a few limitations:

- Code banking is not supported.
- PDATA variables cannot be reviewed.
- Breakpoints and single-stepping do not work in interrupt service routines.
- Only part of the SFR can be changed in the ISD debugger; see ISD51.A51 for the details.

Chapter 4

File List

This chapter contains a source code listing for each C routine and header file used in TUSB2136/TUSB3210/TUSB3410/TUSB5052 firmware debugging.

Topic **Page**

4.1 C Source Code

4.1.1 VOID rs232Initialization(VOID);

Initialize the 8052 serial interface, including timer mode, baud rate, serial mode, and so on.

```
VOID rs232Initialization(VOID)
{
// take care of TMOD, SCON setting, because Timer 0&1 use
// them together
   TMOD &= 0x0F;   // Mask Timer 1 high nibble
   TMOD |= 0x20;   // Set Timer 1 to mode 2 (AUTO RELOAD)
   SCON = 0x40;    // Set serial port for mode 1
// 11.0592MHz uses & SMOD = 0, 24MHz or above uses & SMOD
// = 1
   PCON = 0x80;           // Set SMOD = 1
   TH1 = RS232_BAUDRATE; // reference header
   TR1 = 1;               // enable Timer 1
   TI = 1;                // Set Transmit Interrupt flag 1
                          // to transmit ready
   RI = 0;                // Set Receive Interrupt flag 0
                          // to receive ready
}
```

4.1.2 VOID rs232PutChar(BYTE bData);

Send an unsigned character to the 8052 serial interface. Based on this routine, you can create a print string routine and call the C standard library sprintf() to do any translation.

```
VOID rs232PutChar(BYTE bData)
{
   while(TI!=1);   // wait until last byte transfer
                   // complete
   TI=0;           // clear Transmit Interrupt flag
   SBUF=bData;     // transmit c to 8052's Serial Buffer
}
```

4.1.3 VOID rs232PutHex(BYTE bData);

Send two hexadecimal digits to the 8052 serial interface.

```
VOID rs232PutHex(BYTE bData)
{
   BYTE hexValue;
   hexValue = (bData & 0xF0)>> 4;
   if(hexValue < 10 )
      rs232PutChar(hexValue + '0');
   else
      rs232PutChar(hexValue + 55);
   hexValue = (bData & 0x0F);
   if(hexValue < 10 )
      rs232PutChar(hexValue + '0');
   else
      rs232PutChar(hexValue + 55);
}
```

4.1.4 VOID rs232PutString(char *str);

Send a string which is terminated by \0 to the 8052 serial interface.

```
void rs232PutString(char *str)
{

   while(*str != '\0') rs232PutChar(*str++);

}
```

4.1.5 char rs232GetChar(void);

Get one byte from the 8052 serial interface.

```
char rs232GetChar(void)
{
   while(!RI);
   RI = 0;
   RETURN (SBUF);
}
```

4.2 Header Files

4.2.1 RS232DBG.h

```
#ifndef _RS232DBG_H_
#define _RS232DBG_H_
/*-------------------------------------------------------+
| Include files                                          |
+-------------------------------------------------------*/

/*-------------------------------------------------------+
| Function Prototype                                     |
+-------------------------------------------------------*/
VOID rs232Initialization(VOID);
VOID rs232PutChar(BYTE bData);

/*-------------------------------------------------------+
| Type Definition & Macro                                |
+-------------------------------------------------------*/
// replace follow as desire baud rate
#define RS232_BAUD RATE BAUD4800_12000

/*-------------------------------------------------------+
| Constant Definition                                    |
+-------------------------------------------------------*/
// 11.0592MHz & SMOD = 0
#define BAUD4800_11059 256-(28800/4800)  // 250
#define BAUD9600_11059 256-(28800/9600)  // 253

// 24, 48, 96MHz & SMOD = 1
#define BAUD4800_12000 256-(62500/4800)  // 242.9791666
#define BAUD4800_24000 256-(125000/4800) // 229.9583333
#define BAUD9600_12000 256-(62500/9600)  // 249.4895833
// don't use
#define BAUD9600_24000 256-(125000/9600) // 242.9791666
#define BAUD9600_48000 256-(250000/9600) // 229.9583333
#define BAUD9600_96000 256-(500000/9600) // 203.9166666
```

4.2.2 Types.h

```
#ifndef _TYPES_H_
#define _TYPES_H_

#ifdef __cplusplus
extern "C"
{
#endif
/*-------------------------------------------------------+
| Include files                                          |
+-------------------------------------------------------*/
/*-------------------------------------------------------+
| Function Prototype                                     |
+-------------------------------------------------------*/
/*-------------------------------------------------------+
| Type Definition & Macro                                |
+-------------------------------------------------------*/
typedef char             CHAR;
typedef unsigned char    UCHAR;
typedef int              INT;
typedef unsigned int     UINT;
typedef short            SHORT;
typedef unsigned short   USHORT;
typedef long             LONG;
typedef unsigned long    ULONG;
typedef void             VOID;
typedef unsigned long    HANDLE;
typedef char *           PSTR;
typedef int              BOOL;
typedef double           DOUBLE;
typedef unsigned char    BYTE;
typedef unsigned char *  PBYTE;
typedef unsigned int     WORD;
typedef unsigned long    DWORD;
/*-------------------------------------------------------+
| Constant Definition                                    |
+-------------------------------------------------------*/
#define YES 1
#define NO  0
#define TRUE  1
#define FALSE 0
#define NOERR 0
#define ERR   1
#define NO_ERROR 0
#define ERROR    1
#define DISABLE 0
#define ENABLE  1

/*-------------------------------------------------------+
| End of header file                                     |
+-------------------------------------------------------*/
#ifdef __cplusplus
}
#endif
#endif /* _TYPES_H_ */
```