UM0088
USER MANUAL

IP Driver Manual for the uPSD32xx Family Microcontrollers

INTRODUCTION

All the drivers provided for the uPSD have a common format that consists of a header file (.h) and a C file (.c) that make up the specific IP driver. For example, the ADC IP has two files (uPSD_ADC.c) and (uPSD_ADC.h). The header file contains the various defines and function prototype statements for the ADC code located in the C file.

All the IP modules make use of two other header files that MUST be included in order for the code to compile correctly. These files are the (upsd3200.h) file and the (upsd_hardware.h) file. The upsd3200.h file is the main define file that is used to define the common labels used for the 8032 SFRs and the various PSD configuration registers. The contents of this file should NOT be changed. Please review this file to become familiar with the common labels used.

The upsd_hardware.h file defines the labels and values for items that are unique to each hardware design. To match the specifics of their unique hardware design, the user MUST modify these items (including items such as the oscillator frequency and the CSIOP base address for the PSD registers).

One other recommended file that is the STARTU32.a51 file. This startup file is provided as part of the device drivers. The start up file will automatically disable the watch dog timer (default is enabled) and will initialize memory and the prescalers in the uPSD and set up the stack pointer. After the initialization, the start up file will jump to the beginning of your C code. The file is based off the Keil STARTU32.a51.

To use one or more of the IP drivers in your code, you would need to include the upsd3200.h file, the upsd_hardware.h file and the specific header file of the IP driver that you would like to use in your file that calls the specific IP driver functions. The IP driver C file would be part of the normal build (or makefile) process. The code generated in the IP driver C file would be called by the users code and the linker would link the two modules together.

Note: For more information, please see the application note AN1970, "I^2C Device Driver for DK3200."

DISCLAIMER OF WARRANTY

THE SOFTWARE IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ST AND ITS LICENSORS AND SUPPLIERS FURTHER DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION REMAINS WITH LICENSEE. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ST OR ITS SUPPLIERS OR LICENSORS BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF DATA OR OTHER PECUNIARY LOSS) ARISING OUT OF THIS AGREEMENT OR THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF ST OR ITS SUPPLIERS OR LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES/JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO LICENSEE. NOTWITHSTANDING THE FOREGOING, TO THE EXTENT THAT ST MAY BE HELD LEGALLY LIABLE TO LICENSEE BY A COURT OF COMPETENT JURISDICTION UNDER CONTRACT, TORT, OR ANY OTHER LEGAL THEORY, THE MAXIMUM LIABILITY OF ST SHALL NOT EXCEED ONE (1) EURO.

TABLE OF CONTENTS

IP DRIVERS AND FUNCTIONS

Analog to Digital Converter (ADC)

This driver is used to set up and read the 8-bit ADC unit in the uPSD. The driver will automatically set up the prescaler to generate an internal value close to 6MHz based upon the defined value of the oscillator frequency found in uPSD_hardwre.h. The driver will also put the selected Port 1 pins to the alternate function mode. Calls to the ADC_Read function will return the result after the ADC conversion.

Initialization Function. This function must be called first to set up the I/O port (P1.4 - P1.7) to the Alternate Function for the selected ADC channel, as well as to set up the ADC prescaler value. This function must be called for each channel that is to be used.

```
void uPSD_ADC_Init(channel)
```

Where **channel** is the unsigned character value in the range from 0 to 3 that is used to setup ADC channel 0 to 3.

Note: P1.4 is mapped to ADC channel 0, and so forth.

Read Function. Reads an analog signal from the channel specified (0 to 3) and returns the converted value as an unsigned character. The routine waits for the conversion to complete (approximately 10µs). The uPSD_ADC_Init function must be called first to set up the appropriate channels.

```
unsigned char uPSD_ADC_Read(channel)
```

Where **channel** is the unsigned character value in the range from 0 to 3 that is used to read ADC channel 0 to 3.

Note: P1.4 is mapped to ADC channel 0, and so forth.

Pulse Width Modulation (PWM)

This driver will setup the prescaler values that are used to divide the main system clock to form the input clock, which will in turn be used to define the desired repetition rate for the select PWM channel. It also sets up the PWM control register and puts the selected P4 port pin to Alternate Function mode.

Initialization Function. This function must be called first to set up the I/O port (P4.3 – P4.6) to the Alternate Function for the select PWM channel as well as to set up the PWM prescaler values for both the fixed and the variable PWM channels.

```
void uPSD_PWM_Init_8bit(unsigned char PWM_channel_no, unsigned int
                        PWM_freq8, unsigned char PWMCON_value)
```

Where:

PWM_channel_no is the selected PWM channel number (0 to 3);

PWM_freq8 is the desired input prescaler frequency to PWM unit; and the

PWMCON_value is the desired PWMCON register setup.

PMWCON[0-4]: PWM0 - PWM4 output; 0 = Open Drain; 1 = Push-Pull

PWMCON[5]: PWM enable/disable; 0 = disabled; 1 = enabled

PWMCON[6]: PWM4 polarity control

PWMCON[7]: PWM0 – PWM3 polarity control

Duty Cycle Setup. This function will set up the duty cycle for the selected PWM channel. The PWM is enabled here as well.

```
void uPSD_PWM_Channel_8bit(unsigned char PWM_channel_no,
                           unsigned char Duty_Cyl)
```

Where:

PWM_channel_no specifies the PWM channel number (0 - 3), and

Duty_Cyl specifies the duty cycle (0-0xFF), a 0x7F being 50% duty cycle for the selected PWM channel.

Period and Pulse Width Setup. This function is used to set up the PWM period and pulse width for the variable PWM channel (PWM4). It also sets up the I/O port (P4.7) to the Alternate Function.

```
void uPSD_PWM_Variable_8bit(unsigned char PWM_Period, unsigned char
                            PWM_PulseWidth)
```

Where:

PWM_Period is the desired PWM Period, and

PWM_PulseWidth specifies the desired PWM pulse width.

Note: The PWM_Period cannot have a value of '0' and its content should always be greater than the PWM_PulseWidth.

Disable. This function disables the PWM operation.

```
uPSD_PWM_Disable(void)
```

I^2C Bus Interface

Please see the application note AN1970, "I^2C Device Driver for DK3200."

Standard Serial Interface (UART)

This driver will set up the timers used for baud rate generators for the selected UART channel. It also sets up the Serial Port Control Register for the four modes for serial port communication.

Initialization Function. This function must be called first to initialize the selected UART channel, as well as to set up the timer for the baud rate generator.

```
void upsd_init_serial(unsigned char serial_info,unsigned int timer_value)
```

Where:

serial_info is the 8-bit value which contains the user's configuration information for the UART channels.

- serial_info [0] : 0 = UART0; 1= UART1
- serial_info[2:1]: 00 = fixed baudrate; 01 = use TIMER1; 10 = use TIMER2
- serial_info [3]: 0 = 9th data bit disabled; 1 = 9th data bit enabled
- serial_info [4]: 0 = disable reception (transmit enable); 1 = enable reception
- serial_info [5]: 0 = multiprocess disable, 1 = multiprocess enable
- serial_info[7:6]: serial communication mode: 00 = mode0; 01 = mode1; 10 = mode2; 11 = mode3

timer_value is the value used to set up the baud rate generator.

UART0 Interrupt. This service routine is the Interrupt Service Routine for UART0. Both the transmitted and received serial data are serviced by this routine.

```
static void uPSD_uart0_isr(void)
```

UART1 Interrupt. This service routine is the Interrupt Service Routine for UART1. Both the transmitted and received serial data are serviced by this routine.

```
static void uPSD_uart1_isr(void)
```

Baud Rate Generator. This function sets up the baud rate generator for the selected timer and uart channel.

```
unsigned int uPSD_UART_Timer(unsigned char uart_no, unsigned int baudrate,
                             unsigned char timer_no)
```

Where:

baudrate is the desired baud rate.

uart_no: 0 = UART0, 1 = UART1

Timer_no: 0 = use Timer 1, 1 = use Timer 2

ASCII Transmit. This function will transmit an ASCII character through the selected UART channel.

```
char uPSD_UART_Putchar(char c,bit uart_ch)
```

Where:

c is the ASCII character to transmit.

uart_ch is the UART channel.

uart_ch: 0 = UART0; 1 = UART1

ASCII Receive. This function will receive an ASCII character through the selected UART channel.

```
char uPSD_UART_Getchar(bit uart_ch)
```

Where:

uart_ch is the UART channel.

uart_ch: 0 = UART0; 1 = UART1

Accessing the Flash Memories

This driver provides the functions to access to the flash memories. The instruction sequence of a specific operation is listed in the Instructions Table of the uPSD32xx datasheet.

Main Flash Memory. This driver programs a single byte to the Main Flash memory, then checks the status for completion using the polling method.

```
unsigned char flash_write_with_poll(volatile unsigned char xdata* addr,
                                    unsigned char dat)
```

Where:

addr is address to write to in the Main Flash.

dat is the data to write.

Secondary Flash Memory. This driver programs a single byte to the Secondary Flash memory, then checks the status for completion using polling method.

```
unsigned char flash_boot_write_with_poll(volatile unsigned char xdata* addr,
                                         unsigned char dat)
```

Where:

addr is the address to write to in the Secondary Flash.

dat is the data to write.

Secondary Flash, Toggle Status. This driver programs a single byte to the Secondary Flash, then checks status for completion using the toggle method.

```
unsigned char flash_write_with_toggle(addr, dat)
```

Where:

addr is the address to write to in the Secondary Flash.

dat is the data to write.

Main Flash Bulk Erase. This function erases the entire Main Flash memory (all sectors).

```
unsigned char flash_erase_bulk(volatile unsigned char xdata*
                               flash_bulk_erase_address)
```

Where:

flash_bulk_erase_address can be an address that resides in any Flash segment that has a chip select.

Secondary Flash Bulk Erase. This function erases the entire Secondary Flash memory.

```
unsigned char flash_boot_erase_bulk(volatile unsigned char xdata*
                                    flash_bulk_erase_address)
```

Where:

flash_bulk_erase_address can be an address that resides in any Flash segment that has a chip select.

Main Flash Reset. This function resets the Main Flash memory to Read Array mode. After this reset, the Flash memory may be read like a ROM device.

```
void flash_reset()
```

Main Flash ID. This function reads the Flash Identifier byte from the Main Flash memory. A byte of ID value is returned.

```
unsigned char flash_read_id(flash_id_address)
```

Where:

flash_id_address is the address where the ID is stored in the Flash memory.

Main Flash Sector Erase. This function erases the specified sector in Main Flash memory.

```
unsigned char flash_erase_sector(volatile unsigned char xdata*
                                 sector_erase_address)
```

Where:

sector_erase_address is the address of the sector to be erased.

REVISION HISTORY

Table 1. Document Revision History

Date	Version	Description
June 3, 2004	1.0	First Edition

If you have any questions or suggestions concerning the matters raised in this document, please send them to the following electronic mail addresses:

ask.memory@st.com (*for general enquiries*)

Please remember to include your name, company, location, telephone number and fax number.