Application Report
SLAA307A – July 2006 – Revised August 2007

Using the Timer_A UART Library

Lane Westlund ..... MSP430 Applications

ABSTRACT
This document serves as an overview describing the use of the Timer_A UART library within an existing C or assembly project. The Timer_A UART library encapsulates commonly used routines for transmitting and receiving bytes using UART. These functions are written in assembly to be optimized for the MSP430, but can be called from any C program that includes the ta_uart.h header file.

1 Introduction

It is often desirable for smaller memory MSP430 devices without a UART-capable hardware module to communicate through UART. It might also be the case that a MSP430 device with a UART capable hardware module is already using that module, and additional UART communication is needed. In these cases, it becomes necessary to allow for UART communication through use of the Timer_A module. Using the Timer_A for UART communication is preferable over 'bit-banging' a port pin, as it allows for more precise bit timing and for the device to remain in a low-power mode between bits.

The Timer_A UART library includes the files necessary to configure the Timer_A to transmit full-duplex UART communication over a variety of speeds. The library encapsulates the functions relating to servicing interrupts, and receiving or sending data. This allows users of the library to simply initialize the library, and call send or receive functions on each byte transmitted.

Also included in the library are a series of code examples, intended to illustrate various methods of initializing and using the library.

2 Usage From C

```
void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer

  TI_initTimer(callBack, _32Khz_ACLK_04800_Bitime, TASSEL_1);
  _EINT();
  LPM3;
}

int callBack( unsigned char c )
{
  TI_TA_UART_STATUS_FLAGS &= ~TI_TA_RX_RECEIVED; // allows for RX during TX
  TI_TX_Byte( c );                          // echo byte
  return TA_UART_STAY_LPM;                  // return to LPM3
}
```

The file ta_uart.h must be included in order to gain access to the variables and functions when using the library from a C program.

This program is a demonstration of a simple UART echo program. Characters that are received are then immediately transmitted back to the sender.

In order to set up the library to receive and send bytes, the initTimer function needs to be called only once at the start of the program. It is this function that initializes the timer to be able to receive and send bytes with the correct timing.

The first parameter to this function is a function pointer. This function pointer serves as a callback function. This callback function must accept a char as its only parameter and return an integer. Once the library has successfully received a UART byte, it passes this byte back to the main program by calling the callback function with the received byte as a parameter.

The next parameter is the bit time. It is the amount of Timer_A clocks to time for bit. This number is dependent on the baud rate desired and the clock source given to Timer_A, but typically can be computed as:

Bit time = Timer_A input clock/Desired baud rate

The header file for this library includes several predefined values for various baud rates based on specific clock settings.

The final parameter is the setting for the TASSEL bits for the TACTL register of Timer_A.

Note: This parameter is ORed directly into the TACTL register. Therefore, the bits are in positions 9 and 8. It is recommended to simply use the standard bit definitions as seen in the example. See the appropriate MSP430 User's Guide for more information on Timer_A input clocks and register settings.

After the initialization function has been called properly and the GIE bit is enabled, UART bytes are received by the library. Once a byte is received, the callback function is called. The parameter to the callback function is the received byte. It should be noted that this callback function is called from within the Timer_A1 interrupt service routine, so all effort should be spent to minimize its processing time.

In the previous example, the callback function simply returns a value. When the callback function returns a value other than 0, the library knows to exit low-power mode. Returning a 0 causes the device to return to the low-power mode it was in before receiving the character. For convenience, these return values have been given easy-to-read definitions in the header file. In this example, the definition is used to cause the device to stay in LPM3.

In the example, the callback function is used to simply call the TX_Byte() function. This example echoes the received character. Before calling the TX_Byte() function, the TI_TA_RX_RECEIVED bit is cleared from TI_TA_UART_STATUS_FLAGS. This bit blocks the receiving of further bytes. Clearing this bit allows bytes to be received during the transmission of a byte.

Another possibility would be to have the ISR have the device exit its low-power mode. The main loop can handle the data transmission by calling TX_Byte() and passing it the global TI_RXData, which is filled with the last byte received.

3 Usage From Assembly

```
#include   <msp430x11x1.h>
EXTERN     initTimer
EXTERN     TX_Byte
EXTERN     TI_RXData
EXTERN     TI_TA_UART_STATUS_FLAGS
#define    TI_TA_TX_READY    0x01
#define    TI_TA_RX_RECEIVED 0x02
;-------------------------------------------------------------------------------
            RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
            RSEG    CODE
;-------------------------------------------------------------------------------
RESET       mov.w   #SFE(CSTACK),   SP      ; Initialize stackpointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
            mov.w   #callBack,      R12     ; move function ptr, 1st param
            mov.w   #0x07,          R14     ; Bitime for 4800, 2nd param
            push.w  #TASSEL_1               ; Select Clock Source, 3rd param
            call    #TI_initTimer
            add.w   #0x02,          SP      ; cleanup from parameters
Mainloop    bis.w   #LPM3+GIE,      SR      ; Enter LPM3, enable interrupts

callBack
            bic.b   #TI_TA_RX_RECEIVED, &TI_TA_UART_Status_Flags
            call    #TI_TX_Byte
            mov.w   #0x00,          R12     ; return 0 to stay in LPM3
            ret
;-------------------------------------------------------------------------------
;           Interrupt Vectors
            COMMON  INTVEC
;-------------------------------------------------------------------------------
            ORG     RESET_VECTOR            ; MSP430 RESET Vector
            DW      RESET                   ;
            END
```

The assembly example has exactly the same functionality as the previous C example and is included in the zip file.

4 Compiling the Library

This library is distributed as source code and is intended to be compiled with a project. Because it will be compiled, there are some steps that could be required before using the library:

- Changing the include file in the library to match the device
- Changing the definition of TACLK_MCLK_SYNC (see Section 4.3)
- Changing the definition of USE_C_CONTEXT_SAVE (see Section 4.4)

4.1 *Using Timer_A With the Library*

The library relies on Timer_A operating in continuous mode. Furthermore, the library makes use of CCR1 and CCR2 and their associate interrupt service routine (ISR). This being the case, CCR0 and its ISR are still left available for use.

As long as the operating mode and clock source are not changed for Timer_A during its operation, UART communications are not affected. See the source code included in the example usage directory for an example of using CCR0 to time an interval during UART communication. Using the additional registers of Timer_A5 is not possible with the library, however, the source code of the library is provided and could be modified to allow for this.

4.2 Timing Considerations

As with any serial communications interface, it is important to consider both the speed at which data can be received, as well as the speed at which the data can be processed. For data to be processed, the CPU has to store the data, and possibly perform some calculations, before receiving a subsequent byte. Because the callback function used by this library is called from an ISR, it is important to consider this timing. If interrupts are enabled and the RX flags are cleared in the callback function (as is done in some included examples), the callback function must be written so that it exits before another byte is received. Failing to do this could lead to nested interrupts, stack overflows, and corrupted data.

4.3 MCLK and Timer Clock

An important step in using the timer as a UART module is reading the current count of Timer_A. If MCLK and the Timer Clock are synchronized, this can be accomplished by simply moving the contents of the TAR register to TACCR2. Following this, an offset can be added to TACCR2 to time the first bit.

If MCLK and the Timer Clock are not synchronized, a simple move could yield unpredictable results. For this reason, the timer is used in synchronized capture mode to record the most recent Timer_A count value. This method is reliable, but more time consuming.

Due to these tradeoffs, it is left to the end user to determine if these clocks are synchronized. In the case that they are not synchronized, TACLK_MCLK_SYNC should be commented out in IAR and set to 0 in CCE. Doing this causes the code to be compiled in slightly different ways.

4.4 C Context Save

If a C callback function is provided, it is necessary for the library to preserve additional CPU registers (C scratch registers) across the callback function call, as these registers could be affected by the callback function. If it can be guaranteed that the callback function does not modify the C scratch registers, they need not be preserved. Preserving the C scratch registers consumes additional code space and execution cycles. Therefore a macro within the library is provided to control whether to preserve the C scratch registers (USE_C_CONTEXT_SAVE is commented in in IAR, or set to 1 in CCE) or to leave them as is (USE_C_CONTEXT_SAVE is commented out in IAR, or set to 0 in CCE). This tradeoff is left to the end user. Refer to the compiler documentation for more details regarding the C calling conventions.

4.5 Cycle Count and Code Size

Table 1. Cycle Count (IAR)

Function	TA_UART.s43
TI_initTimer	59
TI_setCallbackFunction	13
TI_TX_Byte	~64 (depending on loops)

Table 2. Code Size (IAR)

TACLK_MCLK_SYNC	USE_C_CONTEXT_SAVE	SIZE (bytes)
0	0	296
0	1	308
1	0	280
1	1	292

5 Included Library Files

Note: In the zip file accompanying this application report, there are directories for the IAR and CCE source. The files in these directories are functionally equivalent, and only contain minor changes to allow for compiling using CCE or IAR, respectively.

ta_uart.s43

This file includes all needed functions and variables to perform UART communication using Timer_A for a variety of speeds.

ta_uart.h

This file includes the definitions for the functions and variables used in ta_uart.s43. This file must be included in any C program that uses the library. It also includes a series of precomputed bit times for use by the timer initialization function.

ta_uart_32khz_9600.s43

This file is included specifically for the case when 9600 baud communication is driven using a 32-kHz clock. The library contains the identical interface, except that the init timer function needs only a callback function, because the timer source and baud rate are implied.

ta_uart_32khz_9600.h

This file includes the nessecary definitions and functions for using the 32-kHz 9600-baud version of the library. This file must be included in any C program that uses this library.

5.1 Variable Description

Note: All TI_UART library variables begin with the ‘TI_’ prefix in order to avoid collision with any other variable names used in an end application.

TI_RXData

This is the last byte to have been received by the UART library. In the callback function, this value is identical to the callback function’s parameter. This variable is provided as a convenient method of gaining access to the last received character outside of the callback function. Outside the callback function, it cannot be ensured that this value will remain consistent, as further received characters cause the TI_UART library to change its value. Because this frequency is application dependent, it remains up to the user to understand if this value must be saved to another location before use and how quickly it must be used before being changed.

TI_TA_UART_StatusFlags

This byte contains several bits whose role is defined using the included header. These bits are:

TI_TA_RX_RECEIVED

This bit is set when a byte has been received into TI_RXData. This bit must be cleared using software, in order to allow further bytes to be received.

TI_TA_TX_READY

This bit is set when the library is ready to transmit a byte. When TX_Byte is called, it clears this bit. Subsequent calls to TX_Byte immediately return. When the byte is transmitted, this bit is reset to one.

5.2 Function Description

TI_initTimer(…..)

This function initializes the Timer_A for UART character transmission and reception. The required parameters are:

- **int (*callBack)(unsigned char)**

  This is the function which is called when a UART character is received. This function should return a zero if low-power mode should be maintained, or a non-zero value to exit the previous low-power mode.
- **unsigned int Bitime**

  This value is the number of Timer_A clocks that should be counted in order to time one bit.
- **unsigned int clock_source**

  This is the setting for the TASSEL bits for the TACTL register of Timer_A.

Note: This parameter is ORed directly into the TACTL register, therefore, the bits are in positions 9 and 8. It is recommended to simply use the TASSEL standard bit definitions from the MSP430 header files for this parameter.

TI_setCallbackFunction(int (*callBack)(unsigned char))

This function takes a function pointer identical to the initTimer() function. This function can be used to change the callback function without reinitializing the timer. Note that if a UART RX is in process while this function is called, the newly received byte is sent to the new callback function.

TI_TX_Byte(unsigned char c)

Calling this function causes the TA_UART library to transmit the character in variable ‘c’.