TMS320 DSP
DESIGNER'S NOTEBOOK

Developing a Full-Duplex UART on the TMS320C3x

APPLICATION BRIEF: SPRA254

Ted Fried, Advanced Computer Communications
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
February 1995

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Examples

Developing a Full-Duplex UART on the TMS320C3x

Abstract

By using the general-purpose I/O pins in conjunction with two timers and an external interrupt, one can develop a very flexible full-duplex UART in software. This document discusses the implementation of an interrupt-driven, full-duplex, asynchronous serial interface, 9600-baud UART with 8 data bits, 1 stop bit, and no parity using the on-chip resources of the TMS320C3x.

Design Problem

How do I develop a full-duplex, asynchronous serial interface using the on-chip resources of the TMS320C3x?

Solution

By using the general-purpose I/O pins in conjunction with two timers and an external interrupt, one can develop a very flexible full-duplex UART in software. This solution discusses the implementation of an interrupt-driven, 9,600-baud UART with 8 data bits, 1 stop bit, and no parity.

Hardware:

The hardware interface is relatively straightforward. The receive line is connected to both the INT0 and IOF1 pins. This will trigger an interrupt on the falling edge of the start bit. The transmit line is connected to the IOF0 pin and a pull-up resistor.

Software:

The receive sequence begins when the start bit triggers the external interrupt. At the interrupt service routine, `.timer0` is then loaded with a value that will result in a delay of one-half of the bit time. The routine then loads the timer's interrupt vector, enables it, and then exits to the main program. When the timer triggers its interrupt, the main body of the receive code is then run. At this time, the line should be in the middle of the start bit. The CPU then samples IOF1 and verifies that the start bit has been read in. If not, the routine reenables the external interrupt and exits to the main program. If the start bit is verified, the timer is then loaded with the full-bit time and started. The procedure then exits to the main program.

On successive `timer0` interrupts, the received bits are shifted into a storage area in memory until a byte is read in. On the 9th interrupt, if the stop bit is verified, the routine will execute a software trap to inform the main program of the byte reception. If the stop bit is not verified, the `BAD_STOP_BIT` subroutine is called where the appropriate action is taken. After the received byte is processed, the external interrupt is then reenabled and the system waits for the next start bit.

The transmit routine begins when the main program loads a byte into the holding register and then calls TX_MAIN. This procedure loads timer1 with the full-bit time value, resets the transmit counter, sets the start bit, and enables the timer's interrupt. The routine will then exit back to the main program. It is required that the main program not call for another byte transmit until it finds the transmit counter equal to 0. On each subsequent `timer1` interrupt, the routine will shift out the transmit byte, including the stop bit, until the transmit counter is zero.

Example 1. Code Example

```
half_bit_time    .set 01ADh             ; assume 33-MHz TMS320C3x
whole_bit_time   .set 035Bh
timer_go         .set 03C1h
timer_setup      .set 0301h
int_setup        .set 0301h
iof_setup        .set 06h

timer0_vector    .word RX_TMR_INT       ; interrupt vector addresses
timer1_vector    .word TX_INT
rx_int_vector    .word RX_INT0
timer0_period    .word 0808028h         ; on-chip RAM locations
timer1_period    .word 0808038h
timer0_control   .word 0808020h
timer1_control   .word 0808030h
timer0_int_vect  .word 0809FC9h
timer1_int_vect  .word 0809FCAh
int0_vector      .word 0809FC1h
rx_byte          .word 0809FF8h
tx_byte          .word 0809FF9h
rx_counter       .word 0809FFAh
tx_counter       .word 0809FFBh

; Main setup for asynchronous serial interface to be run at power-up.

SETUP_ASYNCH:    PUSH AR7
     OR          iof_setup, IOF         ; iof setup and iof0=1
     LDI         timer_setup, AR7       ; setup timer0 and timer1
     STI         AR7, @timer0_control
     STI         AR7, @timer1_control
     LDI         rx_int_vector, AR7     ; load int0 interrupt vector
     STI         AR7, @int0_vector
     OR          int_setup, IE          ; enable interrupts
     POP         AR7
     RETS

; Start bit received, external interrupt service routine.

RX_INT0:         PUSH AR7
     XOR         01h, Ie                ; disable int0
     LDI         half_bit_time, AR7
     STI         AR7, @timer0_period    ; rx_timer period
```

```
     LDI         timer0_vector, AR7
     STI         AR7, @timer0_int_vect     ; rx-timer int vector
     LDI         timer_go, AR7
     STI         AR7, @timer0_control      ; start rx_timer
     LDI         0Ah, AR7
     STI         AR7, @rx_counter          ; reset rx_counter
     POP         AR7
     RETI

; Timer0 interrupt service routine for byte reception.

RX_TMR_INT:      PUSH AR7
     LDI         @rx_counter, AR7
     CMPI        09h, AR7                  ; are we at start bit?
     BNE         STOP                      ; no, check for stop bit
     CMPI        080h, IOF                 ; check rx_bit (IOF1)
     BLT         OK                        ; if less than 80h (IOF1=0)?
     OR          01h, IE                   ; bad start bit, reenable
INT0
     BR          CLEANUP2                  ; go back to main
OK: SUBI         01h, AR7                  ; decrement rx_counter
     STI         AR7, @rx_counter          ; update counter in memory
     LDI         whole_bit_time, AR7
     STI         AR7, @timer0_period       ; load bit time into rx_timer
     LDI         timer_go, AR7
     STI         AR7, @timer0_control      ; start rx_timer
     POP         AR7
     RETI
STOP: PUSH AR6
     LDI         @rx_byte, AR6
     DBNZ        AR7, NEXT                 ;if rx_count !=0, get next bit
     CMPI        080h, IOF                 ; check rx_bit (IOF1)
     BLT         BAD_STOP_BIT              ;GO TO INVALID STOP BIT MODULE
     LSH         -24, AR6                  ; shift rx_byte 24 bits right
     STI         AR6, @rx_byte             ; update rx_byte in memory
     TRAPU       BYTE_RECEIVED             ; TRAP RECEIVED BYTE!!
     OR          01h, IE                   ; reenable INT0\
     BR          CLEANUP
NEXT: CMPI 080h, IOF                       ; check rx_bit (IOF1)
     OR          01h, ST                   ; force carry flag to 1
     BGE         ONE                       ; if rx_bit = 1
     XOR         01h, ST                   ; set carry flag to 0
ONE: RORC AR6                              ; shift in carry bit
     STI         AR6, @rx_byte             ; update rx_byte in memory
     STI         AR7, @rx_counter          ; update counter in memory
     LDI         timer_go, AR6
     STI         AR6, @timer0_control      ; start rx_timer
CLEANUP:         POP AR6
CLEANUP2:        POP AR7 RETI

; Transmit byte main subroutine.

TX_MAIN:         PUSH AR7
```

```
     LDI         whole_bit_time, AR7
     STI         AR7, @timer1_period       ; load timer period
     LDI         timer1_vector, AR7
     STI         AR7, @timer1_int_vect     ; tx_timer int vector
     LDI         @tx_byte, AR7
     OR          0FF00h, AR7               ; mask stop bit to tx_byte
     STI         AR7, @tx_byte             ; update tx_byte
     AND         0FBh, IOF                 ; send out '0' to IOF0
     LDI         0Ah, AR7
     STI         AR7, @tx_counter          ; load counter in memory
     LDI         timer_go, AR7
     STI         AR7, @timer1_control      ; start tx_timer
     POP         AR7
     RETS

; Timer1 interrupt service routine for byte transmission.

TX_INT:          PUSH AR7
     LDI         @tx_counter, AR7       ; load in tx_counter from mem
     DBNZ        AR7, NEXT_OUT          ; if tx_counter not zero
     POP         AR7
     RETI
NEXT_OUT:        PUSH AR6
     LDI         timer_go, AR7
     STI         AR7, @timer1_control   ; start tx_timer
     LDI         @tx_byte, AR6          ; load in tx_byte from mem
     RORC        AR6                    ; next bit out is in carry
     BNC         OUT_ZERO               ; carry=0, then send out '0'
     OR          04h, IOF               ; send out '1' to IOF0
     BR          CLEANUP3
OUT_ZERO:        AND 0FBh, IOF          ; send out '0' to IOF0
CLEANUP3:        STI AR6, @tx_byte      ; update byte in memory
     STI         AR7, @tx_counter       ; update counter in memory
     POP         AR6
     POP         AR7
     RETI
```