Hitachi America, Ltd.

Application Note
H8/330, H8/325

Carol Jacobson

MPB Serial Format by Software

INTRODUCTION

A two or three wire serial bus affords a simple communication path between a master CPU and a single slave device. When the master needs to communicate with more than one slave on the same bus a method must be devised to ensure that each slave only receives it's own messages and that only one device transmits at any time. One way to accomplish this control is to precede data transfers with an ID byte containing an "address" or identification for the intended receiver. When a receiver recognizes it's own ID it begins to collect subsequent transfers until another ID byte occurs. This scheme has been dubbed by several manufacturers as the "**M**ulti-**P**rocessor **B**it" format or protocol (sometimes referred to as "wake-up" mode).

Hitachi's most recent H-series controllers, the H8/329, H8/338 and 300H series, support multi-processor communication (MPB) by hardware. The MPB protocol supported operates in asynchronous mode, without parity, for 7 or 8 bit data and with either 1 or 2 stop bits. The maximum speed is limited only by the serial port's maximum asynchronous data rate. Unfortunately, this feature is not available in earlier H8 products such as the H8/330 or H8/325 series. With a few resurrections, however, earlier products can support a software implementation of MPB and thus interface to a MPB serial bus.

HOW DOES MPB WORK ?

Asynchronous serial communication using multi-processor format requires two steps:

- The transmitter sends an ID frame containing MPB= 1

- The transmitter sends data frames containing MPB= 0

ID frames are distinguished from data frames by the value of the MPB. Receivers connected to the bus must in some way evaluate every frame transferred on the bus to determine the state of the MPB bit. When a receiver detects MPB= 1, the associated data character contains an ID which the receiver should then evaluate. If the received ID matches the receivers own ID, the receiver "wakes-up" and begins to capture the ensuing data frames. The receiver continues to evaluate the MPB of each frame until detecting another frame with MPB= 1. The receiver again evaluates the character of this frame for its own ID. If the new ID doesn't match it's own, the receiver stops collecting data but continues to evaluate the MPB of all subsequent frames.

HITACHI MICRO-CONTROLLERS

The H8/338 and H8/329 series micro-controllers recognize four asynchronous MPB formats:

Serial Frame Bit Number

1	2 -- 8	9	10	11	12
s	8-bit data character		**MPB**	stop	stop
s	8-bit data character		**MPB**	stop	
s	7-bit data character	**MPB**	stop	stop	
s	7-bit data character	**MPB**	stop		

Hitachi's H8 family SCI port supports four standard Asynchronous formats without parity:

Serial Frame Bit Number

1	2 -- 8	9	10	11
s	8-bit data character		stop	stop
s	8-bit data character		stop	
s	7-bit data character	stop	stop	
s	7-bit data character	stop		

The MPB bit always holds the 9th or 10th position in the frame and immediately follows the last data bit. In a standard asynchronous frame, hardware design defines the 9th and 10th positions as either the last data bit or a stop bit. Therefore to implement the MPB protocol using a standard frame, we must find a way either to insert an extra bit (MPB) or re-define the 9th or 10th

position. Since inserting a bit into a pre-defined frame requires disassembling and re-assembling the frame, the CPU overhead and external hardware required make this approach impractical.

SOFTWARE "MPB"

However, we can re-define the 9th bit of an 8-bit character frame by concatenating a 7-bit character with a software "MPB." In effect, D7 of an 8-bit character becomes the MPB. The resulting 8-bit character frame transferred by the H8/330/325 looks identical to the 7-bit character frames transferred by devices operating in MPB format. Other devices on the bus must transmit and receive 7-bit characters with MPB while the H8/330/325 receives and transmits 8-bit character frames with D7 acting as the MPB. In this way the H8/330 or H8/325 can be connected to a MPB serial bus using 7-bit character frames.

Serial Frame Bit Number

	1	2 -- 8	9	10
H8/330/325	s	D0 -- D6	***D7=MPB***	stop
MPB Bus	s	7-bit data character	MPB	stop

H8/330/325 software must set and interpret the 9th frame bit as a MPB bit. As a receiver, the H8/330/325 must read each frame transmitted on the bus to evaluate the 9th bit to determine the nature of the character received. If the bit is high, software compares the remaining 7 character bits to its own ID. If there is a match, the receiver continues to receive 8-bit characters, ignoring D7, the last bit. Data frames will always have the 9th bit (MSB or D7) low. As a transmitter, the H8/330/325 must set D7 of the ID character high and ensure D7 of a data byte is always low.

The following software routines for the H8/330 transmit and receive 7-bit data according to MPB protocol.

HITACHI
Hitachi America, Ltd. • San Francisco Center • 2000 Sierra Point Parkway • Brisbane, CA 94005-1819 • (415) 589-8300

```
;*****************************************************************************
;*****************************************************************************

;                          H8/330 MPB demo program  RECEIVE - MPBR.src
;
;This routine receives a block of 80 bytes of 7-bit data using asynchroous format
;with MPB support.  The receiver is set to receive 8-bit charaters. The 8th bit
;contains the transmitters MPB. Software checks each byte received for the MPB set =
;1.  If the bit is set, the receiver checks for it's own ID.  If the MPB is not set
;software checks a semiphore set within SCI register set.  If the flag is set, the
;receiver accepts the data.  If the flag is cleared the data is ignored.  Registers
;R5 & R6 (& R3l demos only) provide pointers and must be dedicated to the
;SCI routine.

;This routine was written and tested using the ASE / H8/330 operating in mode 2.  An
;H8/330 EVB provided the target system.  The receiver operates under interrupt
;control supported by the routine RIE_INTERRUPT.  A second interrupt routine, ERROR,
;responds to ERI (receiver error) interrupts by clearing the error flag, toggling
;LED 4 and returning to the main routine.

;Receiver and transmitter both operate from an external serial clock driven by the
;transmitters on-chip PWM

;The transmitter for the demo was an H8/338 EVB operating in asynchronous mode with
;MPB enabled.

        .include "c:\demos\h8330.inc"

RE              .equ    h'04
rdrf            .equ    h'06
_data           .equ    h'fd80
_last           .equ    h'fdd0
sys1_ID         .equ    h'0A
sci0_flags      .equ    h'ffdb          ;sci0_flags is really the Transmit Data
Register

        .org h'500

        mov.w   #h'ff80,r7
        ldc     #0,ccr                  ;enable interrupts
        mov.b   #h'ff,r3l
        mov.b   r3l,@p6_ddr
        mov.b   #0,r3l
        mov.b   r3l,@p6_dr              ;turn off LED's for demo

;Initialize SCI port.  This section should be part of the controllers initialization
;routine.  No registers are saved.

        mov.b   #0,r0l
        mov.b   r0l,@sci0_smr
        mov.b   r0l,@sci0_flags         ;clear the TDR register "flags"
        mov.b   #h'42,r0l
        mov.b   r0l,@sci0_scr           ;set up for REI interrupt & external clock

;enable receiver, R6 must be dedicated to receive function. R5 is also used
;but only to detect end of buffer for demo purpose.

ENABLE_RECEIVER:

        mov.w   #_last,r5                ;last buffer location
        bset    #RE,@sci0_scr           ;enable receiver
```


HITACHI
Hitachi America, Ltd. • San Francisco Center • 2000 Sierra Point Parkway • Brisbane, CA 94005-1819 • (415) 589-8300

```
sleep:  sleep                           ;wait for RxI interrupt,  sleep or
                                        ;return to a main routine
        bra     sleep

;The CPU will be interrupted after each byte received on the bus.  The overhead
;required to respond to invalid data with a 19.2K bit rate is approx. 10%.
;This routine assumes R5, R6 & R3l are dedicated to the SCI.
;*****************************************************************************
RIE_INTERRUPT:

        push    r0

        mov.b   @SCI0_rdr,r0l           ;get the byte
        bclr    #rdrf,@sci0_ssr         ;clear receiver flag

        btst    #7,r0l                  ;look for MPB bit set
        bne     ID_chk                  ;if MPB= 0 it's data,


        btst    #0,@sci0_flags          ;In this example the SCI port functions as a
                                        ;receiver only. Since the transmit function
                                        ;is idle, the TDR register is available as a
                                        ;R/W register.  We use this to hold a
                                        ;semiphore flag indicating the receiver is
                                        ;accepting data. If the tranmitter will be
                                        ;used for Full Duplex operation the flag
                                        ;register can be any available RAM location.

        beq     RTE                     ;if the flag is set then data is valid,
                                        ;else it's someone else's data


Receive_data:

        mov.b   r0l,@r6
        adds.w  #1,r6
        cmp.w   r5,r6
        bne     RTE
        bclr    #0,@sci0_flags          ;if last byte received or buffer full
                                        ;clear the valid data flag
        jsr     @LEDS
        mov.w   #_data,r6               ;reset data pointer

RTE:
        pop     r0
        rte



ID_chk:
        bclr    #7,r0l                  ;clear the MPB bit
        cmp     #sys1_id,r0l            ;check for receiver's ID
        bne     RTE                     ;if not receivers ID, then return to sleep
                                        ;or main

        bset    #0,@p6_dr              ;set LED 0 indicating receiver accepts the ID
        mov.w   #_data,r6               ;reset data pointer for new data block
        bset    #0,@sci0_flags  ;else set the flag to receive subsequent data
        bra     RTE                     ;return to main routine


LEDS:
        bnot    r3l,@p6_dr
```


HITACHI
Hitachi America, Ltd. • San Francisco Center • 2000 Sierra Point Parkway • Brisbane, CA 94005-1819 • (415) 589-8300

```
        inc     r3l
        cmp     #h'06,r3l
        bne     rts
        mov.b   #0,r3l
rts:    rts

ERROR:
        bclr    #5,@sci0_ssr            ;clear overflow error
        bclr    #3,@sci0_ssr            ;clear framing error
        bnot    #4,@p6_dr               ;toggle LED #4
        rte

        .org            rxi0_vec
        .data.w rie_interrupt•

        .org    eri0_vec
        .data.w error

        .end


;*******************************************************************************

;*******************************************************************************

;           H8/330 MPB DEMO PROGRAM  TRANSMIT - MPBT.SRC

;
;This routine transfers a block of 80 bytes of 7-bit data using asynchroous format
;with software MPB.  7-bit characters are sent using an 8-bit format.  The 8th bit
;contains the MPB.  The first byte contains the receivers ID followed by the data
;bytes.

;This routine was written and tested using the ASE / H8/330 operating in mode 2.  An
;H8/330 EVB provided the target system.  This example operates as a sub-routine with
;the transmitter controlled by polling.  Registers R0, R3 R5 & R6 are saved at the
;beginning and re-stored before returning to the main calling routine.

;Receiver and transmitter both operate from an external serial clock driven by the
;transmitters on-chip PWM.

;The receiver for the demo was an H8/338 EVB operating in asynchronous mode with MPB
;enabled.

        .include "c:\demos\h8330.inc"

                .org    h'500

sys1_ID                 .equ    h'0A
acknw                   .equ    h'00
tdre                    .equ    h'07
TE                      .equ    h'05
_data                   .equ    h'FD80
_last                   .equ    h'FDD0

                mov.w   #h'ff80,r7

                push    r0
                push    r3                      ;r3 is used for demo only
                push    r5
                push    r6
```

```
                ldc     #0,ccr                  ;enable interrupts
                mov.b   #h'ff,r0l
                mov.b   r0l,@p6_ddr
                mov.b   #0,r0l
                mov.b   r0l,@p6_dr              ;turn off the EVB leds for demo


;Fill 80 locations with data from 0 to 79 to transmit

                mov.b   #0,r0l
                mov.w   #_data,r1
mov:            mov.b   r0l,@r1
                inc     r0l
                inc     r1l
                cmp     #h'50,r0l
                bne     mov
                mov.b   #0,r0l
                mov.b   r0l,@r1                 ;last byte = 00

;initialize a pwm for the clock source

                mov.b   #h'00,r0l
                mov.b   r0l,@PWM0_tcr           ;set for phi/2, bit clock = phi/32
                mov.b   #h'7d,r0l
                mov.b   r0l,@pwm0_dtr           ;50% duty cycle
                mov.b   #1,r3l                  ;


;Initialize SCI port

                mov.b   #00,r0l
                mov.b   r0l,@sci0_smr           ;async, 8 bits, no parity, 1 stop,
sys clock
                mov.b   r0l,@sci0_scr           ;clock = off
                bset    #7,@pwm0_tcr            ;start the clock
                mov.b   #h'22,r0l
                mov.b   r0l,@sci0_scr           ;set TE & CKE1 (external clock)

;Transmit

_MPB:
;Add the mpb to the sys_ID

                mov.b   #sys1_ID,r0l
                bset    #7,r0l                 ;add the MPB to the MSB !

;
SEND_ID:
;send the receiver ID
;
                mov.b   r0l,@sci0_tdr
                bset    #TE,@sci0_scr           ;set the TE bit

tdrei:          btst    #tdre,@sci0_ssr         ;look for TDR to TSR
                beq     tdrei

                bclr    #tdre,@sci0_ssr

                bset    #0,@p6_dr               ;set LED 0 to signal ID byte sent
;
;
;
SEND_Data:
```

```
;get data from RAM, add the MPB bit and send

                mov.w   #_last,r5               ;initialize last data address
                mov.w   #_data,r6               ;initialize start address

tdres:
                btst    #tdre,@sci0_ssr         ;look for TDRE = 1
                beq     tdres

                mov.b   @r6,r0l
                mov.b   r0l,@sci0_tdr           ;load the data

                bclr    #tdre,@sci0_ssr
                adds.w  #1,r6

                cmp.w   r6,r5                   ;look for last address
                bne     tdres
                jsr     @LEDS                   ;toggle LEDs with each 80 byte loop

                jmp    @_MPB            ;start again for demo. Else continue to
                                        ;END_TRANS & return to calling routine


END_TRANS:
tdree:          btst    #tdre,@sci0_ssr
                beq     tdree                   ;wait for last byte into TSR
                bclr    #TE,@sci0_scr           ;turn off the transmitter
                bclr    #tdre,@sci0_ssr         ;clear tdre
                pop     r6
                pop     r5
                pop     r3
                pop     r0
                rts

LEDS:           bnot    r3l,@p6_dr              ;this routine sequentially inverts
                                                ;LEDs 0-6 after 80 bytes are
                                                ;transferred.
                inc     r3l
                cmp     #h'6,r3l
                bne     rts
                mov.b   #0,r3l
rts:            rts


                .end
```


HITACHI
Hitachi America, Ltd. • San Francisco Center • 2000 Sierra Point Parkway • Brisbane, CA 94005-1819 • (415) 589-8300