Hitachi America, Ltd.

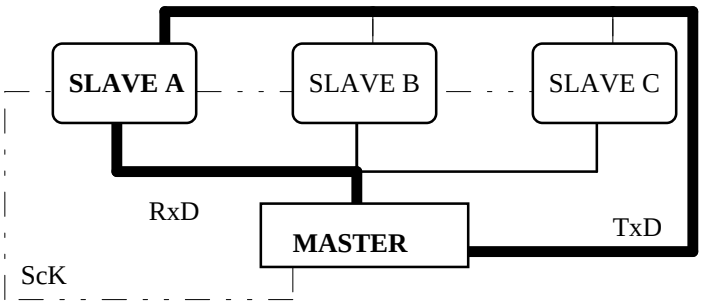
TN-0155

Application Engineering
TechNote

Carol Jacobson

Working with Multi-Processor Serial Control

Newly released products from the H8/300 family including the H8/338, H8/329 series and 300H products incorporate multi-processor bit (also called 9th bit or wake-up mode) support for asynchronous serial communications. The concept behind MPB operation is to allow a master to communicate with several receivers on a single serial chain by assigning each receiver assigned an ID code.



In this example the master communicates with Slave A while Slaves B & C are idle.

RECEIVER

All receivers must in some way monitor each frame transmitted on the bus. When the receivers detect a frame with the MPB bit set= 1, they "wake-up" and compare the frame data to their own IDs. A receiver only captures asynchronous data frames preceded by a frame containing the its own ID. The subsequent data frames have the MPB cleared to 0. The MPB bit, therefore, differentiates an ID frame from a data frame.

For Example: A serial stream to send three bytes of data to two different receivers might look as follows,

s	8-bit ID Byte	MPB	Stop
1	D0 -- D7	1	1

1st ID Frame

s	8-bit Data Byte	MPB	Stop
1	D0 -- D7	0	1

1st Data Frame

s	8-bit Data Byte	MPB	Stop
1	D0 -- D7	0	1

2nd Data Frame

s	8-bit Data Byte	MPB	Stop
1	D0 -- D7	0	1

3rd Data Frame

s	8-bit ID Byte	MPB	Stop
1	D0 -- D7	1	1

2nd ID Frame

s	8-bit Data Byte	MPB	Stop
1	D0 -- D7	0	1

1st Data Frame

s	8-bit ID Byte	MPB	Stop
1	D0 -- D7	0	1

2nd Data Frame

s	8-bit Data Byte	MPB	Stop
1	D0 -- D7	0	1

3rd Data Frame

When operating in MPB mode, SCI on-chip circuitry automatically receives and evaluates each frame without interfering with CPU performance. When the SCI detects a frame with MPB set = 1, the internal circuit clears the multi-processor interrupt enable bit (MPIE), sets the MPB bit in the Serial Status Register (SSR) and allows the receive-end (RxI) and receive-error (ERI) interrupts. In this way the SCI and CPU ignore data not preceded by the correct ID byte.

After the byte with MPB= 1 has shifted into the Receive Data Register, the MPIE bit is cleared and the SCI issues a RxI interrupt (assuming there were no errors). The interrupt handler routine should compare this first byte to the receiver's own ID. If there is a mis-match, subsequent data is not meant for this receiver. In this case, the interrupt handler should re-set the MPIE bit to disable receiver interrupts and prevent data from moving into the RDR. Hardware continues to evaluate the MPB of each frame on the bus.

If the ID matches the receivers ID, the CPU can return to the main routine until the first data byte causes another RxI interrupt. As long as MPIE remains clear, hardware does not evaluate the MPB bit of a frame. After receiving each byte, the interrupt handler must also evaluate the MPB bit of the SSR to determine whether the byte is data or another ID byte. If the MPB is high, the frame received is another ID byte that should be compared to the receivers own. In this way, the SCI and CPU continue to receive data until either detecting another receiver's ID or transmission halts.

TRANSMITTER

When using MPB format the transmitter must insert the correct MPB bit into each frame. For the H8 products this requires two steps: software must select MPB format and software must set the correct MPBT bit value in the Serial Status Register. When sending an ID frame the MPBT bit must be set high to send a frame with MPB= 1. Before sending data, software must clear MPBT to send MPB= 0 with each frame.

Software Example

The following software examples demonstrate MPB operation. In these examples the slave receiver executes a main routine (represented as a sleep / bra loop) until interrupted by the RxI. The RxI interrupt handler performs an ID check, clears the MPIE bit and re-transmits the ID byte back to the master (this step is not part of the MPB protocol and not required) before returning to 'MAIN'. The master follows the ID frame with data frames containing the lower byte of slave RAM addresses. As the slave receiver captures each frame it reads the address, fetches data stored at the address and sends the data back to the master. Slave transmissions to the master are not preceded by an ID frame as the master expects to receive the data and does not need to be interrupted.

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

```
Microtec Research ASMH83    Version 1.0A    Jul 28 17:52:58 1993     Page 1


Command line:  C:\ASMH83\ASMH83.EXE -l tn_338.src
Line        Addr
1                                       ;                       SLAVE   RECEIVER   MPB       ROUTINE
2
3                                       ;This routine executes an asyncronous slave receiver operation with multi-processor
4                                       ;bit control.  The slave CPU remains in the main routine until the SCI detects a data
5                                        ;frame with MPB= 1. As long as MPB=0, the SCI ignors the incomming frame. Although
6                                        ;the receive interrupts are enabled, the SCI prevents the interrupt from occurring until MPB= 1.
7                                       ;When MPB= 1 the SCI allows REI  & ERI interrupts and begins to receive data.  The SCI
8                                       ;interrupts the CPU to fetch the data after each byte received .
9
10                                      ;In this example a master transmitter sends the slave receiver an ID frame.  The slave receiver
11                                      ;compares the ID to it's own.  If there is a match, the slave echo's the ID back to the master.
12                                      ;
13                                      ;The master receives the echoed ID and begins to send data frames containing the lower
14                                      ;half of an addddress in the slave's RAM the master wants to read. The slave CPU pre-loads
15                                      ;R0H with the upper address byte, H'FE.  Up to 256 bytes of data can be read beginning
at
16                                      ;H'FE00.  The slave fetches the data, transmits the data back to the master then returns to the
17                                      ;main routine until the next interrupt.
18
19                                      ;The receiver must check the MPB bit of each frame to determine whether the frame is a data

20                                      ;frame (in this case containing address bytes) or another ID frame.
21
22
23                                              .include        "c:\demos\h8338.inc"
23.161
24
25          0500                                start                   .equ    h'500
26          FF80                                stack                   .equ    h'ff80
27          00FE                                data_stack              .equ    h'fe                    ;upper address of data destination
pointer
28          000A                                sys_ID                  .equ    h'0A                    ;receivers ID
29
30                                                              .org    start
31
32          0500 7907 FF80                      mov.w       #stack,r7
33          0504 0700                           ldc         #0,ccr                          ;enable interrupts
34          0506 F804                           mov.b       #h'04,r0l
35          0508 38D8                           mov.b       r0l,@sci0_smr                   ;8 bits per char, MPB mode
36          050A 38C3                           mov.b       r0l,@stcr                       ;enable MPB
37
38          050C F872                           mov.b       #h'72,r0l
39          050E 38DA                           mov.b       r0l,@sci0_scr                   ;enable external clock, set RE, TE, MPIE & REI
40
41          0510 0180                   main:   sleep                                       ;wait for REI interrupt (this could also be
                                                                                            continuation to
42          0512 40FC                           bra         main                            ;a main routine)
43
44
45          0514 6DF0                     MPB:  push        r0                              ;this is an interrupt routine so we store the
used registers
46          0516 7EDC 7310                      btst        #1,@sci0_ssr                    ;look for MPB= 1, if MPB = 1 check for
receivers ID
47          051A 4612                           bne         chk_ID
48

49          051C F0FE                           mov.b       #h'fe,r0h                       ;restore the upper address byte
50          051E 28DD                           mov.b       @sci0_rdr,r0l                   ;get the lower address byte
51
52          0520 7FDC 7260                      bclr        #6,@sci0_ssr                    ;clear RDRF
53
54          0524 6808                           mov.b       @r0,r0l                         ;get the data at the address
received
55
56          0526 5E00 054A                      jsr         @send_r0l                       ;send the data byte
57          052A 6D70                           pop         r0
58                                              rte                                         ;return to main routine
59
60                                      chk_ID:
61          052E 28DD                           mov.b       @sci0_rdr,r0l                   ;get the character
62          0530 7FDC 7260                      bclr        #6,@sci0_ssr                    ;clear RDRF
63
64          0534 A80A                           cmp         #sys_ID,r0l                     ;If the ID doesn't match the data isn't for
this receiver.
65          0536 4608                           bne         again                          ;Return to the MPB loop or
to a main routine.
66
67          0538 5E00 054A                      jsr         @send_r0l                       ;If it is this receiver's ID re-transmit the
ID back
68          053C 6D70                           pop         r0
69          053E 5670                           rte                                         ;return to main routine
70
71          0540 1B87                 again:    subs.w      #2,r7                           ;reset the stack pointer from the
sub-routine jump
72          0542 7FDA 7030                      bset        #3,@sci0_scr                    ;reset the MPIE bit to disable
receiver
73          0546 6D70                           pop         r0
74          0548 5670                           rte                                         ;return to the main routine
75
76                                  send_r0l:
77
```

```
78          054A 7EDC 7370                  btst        #7,@sci0_ssr                    ;verify TDRE= 1, TDR is empty
79          054E 47FA                       beq         send_r0l
80
81          0550 38DB                       mov.b       r0l,@sci0_tdr                   ;move the data into TDR.
82          0552 7FDC 7270                  bclr        #7,@sci0_ssr                    ;clear TDRE
83
84          0556 5470                                   rts
85
86                                                      .org    h'0038
87          0038 0514               R                   .data.w MPB
88                                                      .end

 Errors: 0, Warnings: 0



Microtec Research ASMH83    Version 1.0A    Jul 28 18:45:54 1993     Page 1


Command line:  C:\ASMH83\ASMH83.EXE -l tn_t.src
Line        Addr
1                                   ;
2                                   ;
3                                   ;                       MASTER TRANSMITTER MPB ROUTINE
4
5
6
7                                   ;This routine controls the MASTER Transmitter for an MPB asynchronous serial bus.  This example
8                                   ;was designed to operate as a sub-routine.  Two registers are stored and recalled, R0 and R6.
9
10                                  ;The masters PWM drives the bus serial clock.
11
12                                  ;The master, in this case an H8/338,  sends an ID frame to one of it's receivers (also an H8/338) to
13                                  ;request information stored in the slaves RAM, beginning at address FE01.  The slave receives and
14                                  ;recognizes it's own ID then echoes the ID back to the master.  The master begins transmitts a data
15                                  ;frame containing an address in slave RAM.  As the slave receives each address, it fetches the
16                                  ;requested data and sends it to the master.
17
18                                  ;The master continues to send data frames (MPB= 0) containing addresses until it's buffer is
19                                  ;filled (in this case when R6= buffer_end = h'fdbf)
20
21          8400                            MPB_transmit            .equ    h'8400
22          FF80                            stack                   .equ    h'ff80
23          FDD0                            data_store              .equ    h'fdd0
24          000A                            sys_ID                  .equ    h'0a
25          00E0                            buffer_end              .equ    h'e0
26
27                                          .include    "c:\demos\h8338.inc"
27.161
28
29                                          .org        MPB_transmit
30
31          8400 7907 FF80                  mov.w   #stack,r7
32          8404 5E00 8408          main:   jsr     @start
33
34                                  ;initialize a pwm for the clock source
35
36                                  start:
37          8408 6DF0                                           push    r0
38          840A 6DF6                                           push    r6
39
40                                  ;set-up the PWM for a 100K clock
41
42          840C F800                                           mov.b       h'00,r0l
43          840E 38A0                                           mov.b       0l,@pwm0_tcr            ;set for phi/2, bit clock = phi/32
44          8410 F87D                                           mov.b       h'7d,r0l
45          8412 38A1                                           mov.b       0l,@pwm0_dtr            ;50% duty cycle
46                                  ;initialize the SCI
47          8414 F804                                           mov.b       h'04,r0l
48          8416 38D8                                           mov.b       0l,@sci0_smr            ;set for 8-bits, with MPB
49          8418 38C3                                           mov.b       0l,@stcr                ;nable MPB
50          841A 7906 FDD0                                      mov.w       data_store,r6           initialize data pointer
51
52                                  ;send ID byte
53          841E 7FA0 7070                                      bset        7,@pwm0_tcr             start the clock
54          8422 F832                                           mov.b       h'32,r0l
55          8424 38DA                                           mov.b       0l,@sci0_scr            enable external clock & set TE & RE
56
57                                  ;Note: MPIE is not set here.  MPIE "wakes-up" a serial port when an ID frame is received.  In the
case
58                                  ;of the bus master, the SCI port is already active and in control of the bus.
59
60          8426 7FDC 7000                                      bset        0,@sci0_ssr             set the MPBT bit to send the ID
byte
61          842A F80A                                           mov.b       sys_ID,r0l              set-up the receivers ID
62
63          842C 5E00 8458                                      jsr         @tdre                   send the ID frame
64          8430 5E00 8466                                      jsr         @receive                look for ID returned
65
66          8434 A00A                                           cmp.b       sys_ID,r0h              ;If it's the ID the master sent
continue.
67          8436 4704                                           beq         ours                    ;If it's not the ID sent, set
flags& 68           8438 5A00 2134                              jmp         @stop                   ;This could also be an error
handling routine
69
```

```
70          843C 7FDC 7200                          ours:       bclr        #0,@sci0_ssr            ;clear the MPBT bit for data
(addr.) byte1          8440 F801                                mov.b   #h'01,r0l       ;set-up the first address location
72
73          8442 5E00 8458                          send:       jsr         @tdre                   ;send the address
74          8446 5E00 8466                                      jsr         @receive                ;receive the data
75
76          844A 6CE0                                           mov.b       r0h,@-r6                ;store the data
77          844C 0A08                                           inc         r0l                     ;set the next address
78          844E A8E0                                           cmp.b       #buffer_end,r0l         ;look for the end of the buffer
79          8450 46F0                                           bne         send
80
81          8452 6D76                                           pop         r6
82          8454 6D70                                           pop         r0
83          8456 5470                                           rts                             ;return to the calling routine
84
85          8458 7EDC 7370                          tdre:       btst        #7,@sci0_ssr
86          845C 47FA                                           beq         tdre
87          845E 38DB                                           mov.b       r0l,@sci0_tdr         ;send the byte
88          8460 7FDC 7270                                      bclr        #7,@sci0_ssr            ;clear TDRE
89          8464 5470                                           rts
90
91                                                  receive:
92          8466 7EDC 7360                                      btst        #6,@sci0_ssr            ;look for RDRF = 1
93          846A 47FA                                           beq         receive
94          846C 20DD                                           mov.b       @sci0_rdr,r0h
95          846E 7FDC 7260                                      bclr        #6,@sci0_ssr            ;clear RDRF
96          8472 5470                                           rts
97
98                                                              .org 8500
99
100         2134 7900 FFFF                          stop:       mov.w   #h'ffff,r0                ;loop in this routine if ID
received doesn't match
101         2138 40FA                                           bra     stop                    ;the ID sent
102
103                                                             .end

 Errors: 0, Warnings: 0
```


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