## Summary

This program demonstrates how to create a buffered interrupt-based implementation of the primary serial1 port which is supported by the 68HC11's on-chip UART.

## Description

An interrupt service routine maintains two buffers (queues) for transmitted and received characters. The standard serial routines KEY, ?KEY and EMIT are revectored to get characters from and put characters into the buffers. This allows multitasking systems to deal with fast serial I/O without losing characters.

TO USE:

simply compile this code and then execute

**`USE.Q.SERIAL1`**

to install the queued serial1 handlers. All high level code routines such as . F. DUMP etc. will work as before, but now using the buffered interrupt-based routines.

To revert to the original serial1 routines, execute:

**`STANDARD.SERIAL1`**



```
4 USE.PAGE          \ you can comment this out if your memory map is already set up.
HEX                 \ set the numeric base during compilation
A WIDTH !           \ avoid nonunique names

ANEW INTERRUPT.BASED.SERIAL1

\ NOTE: we require that the variable area is in common RAM.
\       This condition is met for all memory maps set by USE.PAGE
\       on the QED-3 Board (kernel versions 3.0 and later).
\       If you are using a V2.xx PROM, make sure to move the
\       variable area to common memory: for example, execute HEX  8E00 0 VP X!

802D CONSTANT SCCR2                 \ serial control register 2
802E CONSTANT SCSR                  \ serial status register
802F CONSTANT SCDR                  \ serial data register

02 CONSTANT TXD.MASK                \ location of TxD pin in portd & portd.direction

80 CONSTANT TDRE.MASK               \ transmit data register empty, in SCSR
20 CONSTANT RDRF.MASK               \ receive data register full, in SCSR

80 CONSTANT TIE.MASK                \ transmit interrupt enable, in SCCR2
20 CONSTANT RIE.MASK                \ receive interrupt enable, in SCCR2
08 CONSTANT TE.MASK                 \ transmitter enable, in SCCR2
```

```
BASE @ DECIMAL                          \ define buffer sizes in decimal base
50 CONSTANT TRANSMIT.BUFFER.SIZE        \ you can change this if you want.
50 CONSTANT RECEIVE.BUFFER.SIZE         \ you can change this if you want.

BASE !          \ restore original base

VARIABLE RECEIVE.BUFFER
RECEIVE.BUFFER.SIZE 2- VALLOT           \ 2- accounts for VARIABLE's 2 allocated bytes

VHERE XCONSTANT RECEIVE.BUFFER.END      \ points to end+1 xaddr of receive.buffer

VARIABLE TRANSMIT.BUFFER
TRANSMIT.BUFFER.SIZE 2- VALLOT          \ 2- accounts for VARIABLE's 2 allocated bytes

VHERE XCONSTANT TRANSMIT.BUFFER.END     \ points to end+1 xaddr of transmit.buffer


VARIABLE TRANSMIT.HEAD                  \ pointer to most recent char added to buffer
VARIABLE TRANSMIT.TAIL                  \ pointer to oldest char added to buffer
VARIABLE RECEIVE.HEAD                   \ pointer to most recent char added to buffer
VARIABLE RECEIVE.TAIL                   \ pointer to oldest char added to buffer

VARIABLE TRANSMIT.IRQ.COMING
     \ flag; if true, background interrupt will handle next char;
     \ if false, emit routine must initiate the transmit process

CODE SERVICE.TRANSMITTER     ( -- )
     \ called when a char has just finished transmitting and TDRE bit is set
     \ in SCDR; writes next char to serial port. We disable the
     \ transmitter interrupt when there are no chars left to be sent.
     TRANSMIT.TAIL DROP EXT LDX             \ X points to oldest char to transmit
     TRANSMIT.HEAD DROP EXT CPX
     EQ IF,                                 \ if head = tail: no chars left in buffer
          SCCR2 IMM LDX
          TIE.MASK  TE.MASK OR 0 IND,X BCLR     \ disable the xmit interrupt
          0 IMM LDD
          TRANSMIT.IRQ.COMING DROP EXT STD      \ clear the flag
          0D IMM LDAB
          SCDR EXT STAB          \ write char -> serial output port to clr irq flag
          RTS                               \ we're done.
     ENDIF,
     0 IND,X LDAB                           \ B <- char
     SCDR EXT STAB                          \ write char -> serial output port
     INX                                    \ increment transmit buffer pointer
     TRANSMIT.BUFFER.END DROP IMM CPX
     HS IF,                                 \ handle rollover
          TRANSMIT.BUFFER DROP IMM LDX
     ENDIF,
     TRANSMIT.TAIL DROP EXT STX             \ save updated tail pointer
     FFFF IMM LDD
     TRANSMIT.IRQ.COMING DROP EXT STD       \ set the flag: an irq will occur
     RTS
END.CODE
```

```
CODE SERVICE.RECEIVER     ( -- )
     \ puts received char into buffer and updates pointers
     RECEIVE.HEAD DROP EXT LDX            \ X points to spot for newest rcv'd char
     SCDR EXT LDAB                        \ B <- input char
     0 IND,X STAB                         \ store input char in buffer
     INX                                  \ increment receive buffer pointer
     RECEIVE.BUFFER.END DROP IMM CPX
     HS IF,                               \ handle rollover if we're at end
          RECEIVE.BUFFER DROP IMM LDX
     ENDIF,
     RECEIVE.HEAD DROP EXT STX            \ save updated head pointer
     RECEIVE.TAIL DROP EXT CPX
     EQ IF,                               \ if head = tail, we have buffer overrun
          INX
          RECEIVE.BUFFER.END DROP IMM CPX
          HS IF,                          \ handle rollover
               RECEIVE.BUFFER DROP IMM LDX
          ENDIF,
          RECEIVE.TAIL DROP EXT STX       \ bump tail, losing oldest char
     ENDIF,
     RTS
END.CODE


CODE SERIAL1.SERVICE          ( -- )
     \ handles serial interrupt for both transmitter and receiver
     SCSR EXT LDAA
     TDRE.MASK IMM ANDA        \ if interrupt was because we just xmitted a char...
     NE IF,
          CALL SERVICE.TRANSMITTER
     ENDIF,
     SCSR EXT LDAA
     RDRF.MASK IMM ANDA        \ if interrupt was because we just received a char...
     NE IF,
          CALL SERVICE.RECEIVER
     ENDIF,
     RTS
END.CODE
```

```
: Q.EMIT1     ( char -- )
     \ A queued version of emit1.  Writes a character into the transmit buffer;
     \ if the transmit interrupt has been disabled, enables it
     \ and explicitly calls SERVICE.TRANSMITTER
     \ If transmit interrupt is already enabled, just lets interrupt routine
     \ take care of the transmission of the character.
     SERIAL1.RESOURCE GET
     TRANSMIT.HEAD @ (C!)              ( -- ) \ put char in buffer
     TRANSMIT.HEAD @ 1+ DUP TRANSMIT.BUFFER.END DROP =     \ handle rollover
     IF
          DROP                                  \ drop head
          TRANSMIT.BUFFER DROP                  \ replace with start of buf; drop page
     ENDIF                                      ( new.head.pointer -- )
     BEGIN                                      ( new.head.pointer -- )
          DUP TRANSMIT.TAIL @ =
     WHILE                                      \ if new head would = tail, we must wait
          PAUSE                                 \ wait for irq to move tail
     REPEAT
     TRANSMIT.HEAD !                            \ update head pointer
     >ASSM
          TPA                                   \ save state of global I bit
          PSHA
          SEI                                   \ disable irqs
          TRANSMIT.IRQ.COMING DROP EXT LDD      \ check transmitter status
          EQ IF,                                \ if no xmitter irq is coming...
               SCCR2 IMM LDX
               TIE.MASK  TE.MASK OR 0 IND,X BSET     \ enable the xmit interrupt
               CALL SERVICE.TRANSMITTER         \ and start the transmission process
          ENDIF,
          PULA
          TAP                                   \ restore prior state of interrupts
     >FORTH
     SERIAL1.RESOURCE RELEASE
     ;

: Q.?KEY1     ( -- flag )
     \ flag is true if at least 1 char is in the queued serial1 input buffer;
     \ false if no chars are in the buffer
     SERIAL1.RESOURCE GET
     RECEIVE.HEAD @ RECEIVE.TAIL @ <>          \ if not=, chars are present
     SERIAL1.RESOURCE RELEASE
     ;
```

```
: Q.KEY1     ( -- char )
     \ waits (if necessary) for receipt of char from queued serial1
     \ and places char on data stack.
     SERIAL1.RESOURCE GET
     BEGIN
          RECEIVE.HEAD @ RECEIVE.TAIL @ =     \ if =, no chars are present
     WHILE
          PAUSE                               \ wait and pause until chars come in
     REPEAT
     RECEIVE.TAIL @ DUP (C@) SWAP             ( -- char\tail.ptr )
     1+                                       \ inc buffer pointer
     DUP RECEIVE.BUFFER.END DROP =            \ handle rollover
     IF     DROP                              \ drop prior pointer
          RECEIVE.BUFFER DROP                 \ replace with buf start; drop page
     ENDIF
     RECEIVE.TAIL !                           \ save updated pointer
     SERIAL1.RESOURCE RELEASE
     ;


\ ********************* SET IT ALL UP *********************************

: INIT.SERIAL1.BUFFERS     ( -- )
     TRANSMIT.BUFFER DROP TRANSMIT.HEAD !     \ pointer to most recent char added
     TRANSMIT.BUFFER DROP TRANSMIT.TAIL !     \ pointer to oldest char added
     RECEIVE.BUFFER DROP RECEIVE.HEAD !       \ pointer to most recent char added
     RECEIVE.BUFFER DROP RECEIVE.TAIL !       \ pointer to oldest char added
;

: USE.Q.SERIAL1     ( -- )
     \ revectors KEY, EMIT, and ?KEY in the currently active task
     \ to use the queued serial1 routines.
     \ Initializes the queued serial1 buffers and pointers.
     \ enables the serial receiver interrupt; the transmitter interrupt
     \ will be enabled when the first character is sent.
     \ This routine GLOABALLY ENABLES INTERRUPTS!
     \ Use the standard BAUD1.AT.STARTUP routine to set the baud rate;
     \ the default is 9600 baud.
     INIT.SERIAL1.BUFFERS
     0\0 SERIAL1.RESOURCE X!                  \ init resource variable
     TRANSMIT.IRQ.COMING OFF                  \ transmitter is initially off
     TXD.MASK PORTD SET.BITS
     TXD.MASK PORTD.DIRECTION SET.BITS        \ default for txd = output high
     CFA.FOR Q.EMIT1 UEMIT X!                 \ now revector serial routines
     CFA.FOR Q.KEY1 UKEY X!
     CFA.FOR Q.?KEY1 U?KEY X!
     CFA.FOR SERIAL1.SERVICE SCI.ID ATTACH    \ set up interrupt service routine
     TIE.MASK SCCR2 (CLEAR.BITS)              \ disable transmit interrupt
     RIE.MASK SCCR2 (SET.BITS)                \ enable receive interrupt
     ENABLE.INTERRUPTS                        \ globally enable irqs
     ;
```

```
: STANDARD.SERIAL1     ( -- )
     \ reverts to standard non-buffered serial1
     RIE.MASK TIE.MASK OR SCCR2 (CLEAR.BITS)    \ disable receive & xmit interrupts
     TE.MASK SCCR2 (SET.BITS)                   \ enable transmitter
     0\0 SERIAL1.RESOURCE X!                    \ init resource variable
     CFA.FOR EMIT1 UEMIT X!                     \ now revector serial routines
     CFA.FOR KEY1 UKEY X!
     CFA.FOR ?KEY1 U?KEY X!
     ;

: .STATUS     ( -- ) \ debug only
     TRANSMIT.HEAD 8 DUMP       \ xmit.head xmit.tail rcv.head rcv.tail
     RECEIVE.BUFFER 40 DUMP
     TRANSMIT.BUFFER 40 DUMP
     ;
```



Mosaic Industries

5437 Central Ave Suite 1, Newark, CA 94560 Telephone: (510) 790-8222 Fax: (510) 790-0925