Bulk Performance Analysis of the 8x930Ax USB Controller

August 20, 1997

1.0 About This Document

The purpose of this document is to demonstrate the bulk data transfer capability of the 8x930Ax USB controller. The performance data provided in this document is based on actual lab tests.

1.1 Additional Information Sources

Intel documentation may be obtained by calling your local Intel Sales Representative or Intel Literature Sales at 1-800-548-4725, or through Intel's World Wide Web location at: www.intel.com/design/usb/ .

2.0 Summary

The tests were conducted using a COMPAQ Presario PC, two Intel PDK PC's, an 8x930Ax Nohau In-Circuit Emulator, and an Intel 8x930Ax evaluation board. More complete details on the test setup are provided in subsequent sections.

USB defines four transfer types: Bulk, Control, Isochronous and Interrupt. This document provides performance data solely for bulk transfer. Since the program execution speed of the 8x930Ax controller varies depending on the location of code/data (internal or external), the 8x930Ax was tested in three different ways:

1. Using Internal ROM only;
2. Using a combination of both internal ROM and external ROM; and
3. Using external ROM only

Table 1 shows the performance data for the three code locations. When program execution is from internal ROM the data transfer rate is very close to the theoretical maximum for bulk transfers. When program execution is from external ROM the data transfer rate is lowest, due to the code fetch required to external memory.

Table 1. 8x930Ax Controller Bulk Transfer Rates

Code Location	Bulk IN (Mbps)	Bulk OUT (Mbps)
Internal ROM	9.216	9.216
Internal/External ROM	7.168	7.168
External ROM	5.632	5.632
*Bulk theoretical	9.728	

3.0 USB Bulk Packet

USB defines the allowable maximum bulk data payload sizes as 8, 16, 32, or 64 bytes. The best transfer rate is achieved using a data payload size of 64 bytes. This is because there are less protocol overhead as larger data payloads are used. Bulk also has a better data transfer rate than isochronous, because of the ability to do multiple data transfers per frame. But the trade-off is that bulk does not have guaranteed frame time, as isochronous does. Theoretically, bulk data can be transferred up to 9.728 Mbps, while isochronous is 8.18Mbps. Refer to Section 5.8, Bulk Transfers, of the Universal Serial Bus Specification V1.0 for more details on bulk performance.

4.0 Test Setup

This section describes the controller setup, PC setup and measurement techniques.

4.1 8x930Ax Controller Setup

The 8x930Ax controller was tested using endpoint one (EP1) and running at an internal clock frequency (Fclk) of 12 MHz. The 8x930Ax controller was also setup to transfer USB data at full speed (12Mbps). Endpoint one was chosen because it has the largest USB FIFO/buffer sizes. The dual-buffering feature of the 8x930Ax controller USB FIFO was also used. Dual-buffering allows faster data transmission because it allows the CPU to access one of the two data sets to load or unload data while the USB hardware can access the second data set.

4.2 Host PC Setup

A 166 MHz COMPAQ Presario PC was used as the host PC to send and receive USB data. A DOS program was used to generate USB transactions. The test was carried out using only one USB device on the bus, for example, only the 8x930Ax controller.

4.3 Monitoring and Measuring USB Transactions

An Intel PDK PC, equipped with a CATC USB Detective Probe, was used to monitor and measure USB traffic. The transfer rates were measured by using the number of USB data packets transferred per USB frames.

5.0 Test Descriptions

As explained above, the 8x930Ax was tested using three different methods; internal ROM only, combination of internal/external ROM, and external ROM only. This section explains how each method was used to test the 8x930Ax controller. In all three methods, the 8x930Ax controller was configured to run in paged mode and source.

5.1 Internal ROM

To emulate internal ROM, a Nohau in-circuit emulator was used. The in-circuit emulator employs high-speed SRAMs to emulate internal ROM. The emulator was installed in an Intel PDK PC. The sample code examples in Appendix A were used to run on the emulator. Eighteen (18) 64-byte packets were measured per USB frame for both transmit and receive directions. This results in a transfer rate of (18 packets * 64 bytes/packet * 8 bits/byte * 1000) bps or 9.216 Mbits/sec.

5.2 Internal/External ROM Combination

In this test, an Intel 8x930Ax evaluation board was used to run the sample code examples in Appendix A. In these tests, all the code, except for the "block move" routines, were placed in external memory. An 8x930Ax with a programmed internal ROM was used. The "block move" routines similar to the ones shown in the sample code examples in Appendix A were called internally. Thus, each time the "block move" routines were called, the 8x930Ax fetched code from the internal ROM. This setup gave the second best results. Fourteen (14) 64-byte packets were measured per USB frame in both transmit and receive directions. This results in a transfer rate of (14 packets * 64 bytes/packet * 8 bits/byte * 1000) bps or 7.168 Mbps.

5.3 External ROM

In this test, an Intel 8x930Ax evaluation board was used to run the sample code examples in Appendix A. In these tests, all the code were placed in external memory, even the "block move" routines. Using this setup, eleven (11) 64-byte packets were measured per USB frame in both transmit and receive directions. This results in a transfer rate of (11 packets * 64 bytes/packet * 8 bits/byte * 1000) bps or 5.632 Mbps.

6.0 Conclusion

Based on the test results the 8x930Ax controller performed the best when using only internal ROM. The results when using only internal ROM are very close to the theoretical maximum rate (9.728 Mbps). It is also important to note that the performance to transfer data over USB may vary based on the nature of the application. For example, it may depend on how much of the 8x930Ax controller is utilized in the application environment to handle other chores besides transferring USB data.

Appendix A

This appendix contains the sample assembly code examples used to perform receive and transmit tests. Both examples are interrupt-driven. After endpoint one is setup, the program runs in an infinite loop waiting for a USB interrupt - endpoint one interrupt in this case. In the interrupt service routine (ISR), the "block move" routine is called to either empty or fill the USB FIFO. After the interrupt is serviced, the program returns to the main loop.

Two techniques are used to optimize the code to get better throughput:

1. The "block move" routine are enrolled instead of using a loop reiterating sixty-three times. To implement the "block move" routine in a loop form would require a "compare and jump" instruction, which would take a hit.
2. In the receiving example, before the ISR returns to the main loop after servicing the request, it checks to see if another data set were received while the previous one was being serviced. If so, the ISR empties the second data set and then returns to the main loop, otherwise it simply returns to the main loop and waits for an interrupt.

In the transmitting example, before the ISR returns to the main loop after servicing the request, it checks to see if the other data set has already been transmitted. If so, the ISR loads the data set and then returns to the main loop, otherwise it just returns to the main loop and waits for an interrupt.

Sample Code for Bulk IN Test

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Bulk IN performance test. Uses internal and external move64 inline
; code to verify/compare performance. This test is done using HCD_LITE,
; which is DOS-based application to generate USB transactions.
; This test disregards USB bus errors and uses interrupt-driven mechanism.
; When this code is used with the emulator, the external move64 routine is
; used.
;
; Last Updated: 4/30/97
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
INCLUDE 8X930AX.inc
DEFINE VECTOR_SEG, SPACE=CODE, ORG=FF4000h
SEGMENT VECTOR_SEG
RESET_VECTOR:   LJMP START_EXEC
INT0_VECTOR:    LJMP $
                ds 5
TIM0_VECTOR:    LJMP $
                ds 5
INT1_VECTOR:    LJMP $
                ds 5
TIM1_VECTOR:    LJMP $
                ds 5
SER_VECTOR:     LJMP $
                ds 5
TIM2_VECTOR:    LJMP $
                ds 5
PCA_VECTOR:     LJMP $
                ds 13
SOF_VECTOR:     LJMP SOF_ISR
                ds 5
USB_FUNC_VECTOR LJMP FUNC_ISR
                ds 5
SUS_RSM_VECTOR  LJMP $
                ds 25h
TRAP_VECTOR:    LJMP $
END_VECTORS:
DEFINE CODE_SEGMENT, SPACE=CODE, ORG=FF4080H
SEGMENT CODE_SEGMENT
START_EXEC:
        CLR LC          ; enable high clock mode
        mov SP,  #LOW(STACK_DATA)
        mov SPH, #HIGH(STACK_DATA)
        MOV EPINDEX, #01h
        MOV EPCON, #03h
        MOV TXCON, #C4h
        MOV TXCNTL, #00h
        MOV TXCON, #C4h
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Enable function interrupt
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        MOV IEN0, #80h  ; enable global interrupt
        MOV IEN1, #03h  ; enable function interrupt & SOF interrupt
```

```
        MOV FIE, #04h   ; enable transmit interrupt
        MOV P1, #00h
        MOV WR0, #00FFh
        MOV WR2, #4000h
        MOV WR8, #00FFh
        MOV WR10, #6000h
        MOV WR12, #00FFh
        MOV WR14, #7000h
        MOV TMOD, #01h
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Get 2 datasets ready
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        MOV R4, TXFLG   ; check for room
        ANL R4, #C0h
        CJNE R4, #C0h, _EMPTY0  ; go get more data
        SETB P1.0
        LJMP $           ; should not get here
_EMPTY0:
        MOV TH0, #00h
        MOV TL0, #00h
        SETB TCON.4     ; start timer
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The LCALL on the following line calls either internal ROM or
; the BLOCK_MOVE routine located in this assembly file. One
; the LCALL line must be commented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        LCALL 2B83h     ; call internal ROM move64 routine
        LCALL _BLOCK_MOVE ; external move64 routine
        CLR TCON.4      ; stop timer
        MOV R4, TH0
        MOV @DR12, R4
        INC DR12, #1
        MOV R4, TL0
        MOV @DR12, R4
        INC DR12, #1
        MOV TXCNTL, #64 ; load FIFO
        MOV R4, TXFLG   ; check for room
        ANL R4, #C0h
        CJNE R4, #C0h, _EMPTY1  ; go get more data
        SETB P1.1
        LJMP $  ; should not get here
_EMPTY1:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The LCALL on the following line calls either internal ROM or
; the BLOCK_MOVE routine located in this assembly file. One
; the LCALL line must be commented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        LCALL 2B83h     ; call internal ROM move64 routine
        LCALL _BLOCK_MOVE ; external move64 routine
        MOV TXCNTL, #64 ; load FIFO
        MOV R5, #00h
        SETB P1.2
        LJMP $  ; wait here for interrupt
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This is the external block move routine which is called
; to move data from external into the FIFO. A similar routine
; exists internal to the 8x930Ax controller.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_BLOCK_MOVE:
_0:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_1:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_2:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_3:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_4:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_5:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_6:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_7:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_8:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_9:     MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_10:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_11:    MOV WR6, @DR0
        MOV TXDAT, R6
```

```
        MOV TXDAT, R7
        INC DR0, #2
_12:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_13:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_14:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_15:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_16:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_17:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_18:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_19:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_20:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_21:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_22:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_23:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_24:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
```

```
        INC DR0, #2
_25:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_26:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_27:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_28:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_29:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_30:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
_31:    MOV WR6, @DR0
        MOV TXDAT, R6
        MOV TXDAT, R7
        INC DR0, #2
        RET
;
; This routine serves as the ISR for the USB function interrupt.
;
FUNC_ISR:
        CLR FTXD1       ; clear pending interrupt
        PUSH R4
JB TXSTAT.1, _ERROR0
        MOV R4, TXFLG   ; check for room
        ANL R4, #C0h
        CJNE R4, #C0h, _EMPTY2   ; go get more data
        POP R4
        RETI
_EMPTY2:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;
; The LCALL on the following line calls either internal ROM or
; the BLOCK_MOVE routine located in this assembly file. One
; the LCALL line must be commented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;
;         LCALL 2B83h     ; call internal ROM move64 routine
         LCALL _BLOCK_MOVE ; call external move64 routine
         MOV TXCNTL, #64 ; load FIFO
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Fill the next dataset to save some context-switch time.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        JB TXSTAT.1, _ERROR0
        MOV R4, TXFLG   ; check for room
        ANL R4, #C0h
        CJNE R4, #C0h, _EMPTY3   ; go get more data
        POP R4
        RETI
_EMPTY3:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The LCALL on the following line calls either internal ROM or
; the BLOCK_MOVE routine located in this assembly file. One
; the LCALL line must be commented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        LCALL 2B83h     ; call internal ROM move64 routine
        LCALL _BLOCK_MOVE ; external move64 routine
        MOV TXCNTL, #64 ; load FIFO
        POP R4
        RETI
SOF_ISR:
        RETI
_ERROR0:
        CLR IEN1.1
        MOV P1, #0Eh
        LJMP $
_ERROR1:
        CLR IEN1.1
        MOV P1, #0Fh
        LJMP $
DEFINE PDATA_SEG, SPACE=PDATA
SEGMENT PDATA_SEG
STACK_DATA:     ds 100h
```

Sample Code for Bulk OUT Test

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Bulk IN performance test. Uses internal and external move64 inline
; code to verify/compare performance. This test is done using HCD_LITE,
; which is DOS-based application to generate USB transactions.
; This test disregards USB bus errors and uses interrupt-driven mechanism.
; When this code is used with the emulator, the external move64 routine is
; used.
;
; Last Updated: 4/30/97
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
INCLUDE 8X930AX.inc
DEFINE VECTOR_SEG, SPACE=CODE, ORG=FF4000h
SEGMENT VECTOR_SEG
RESET_VECTOR:   LJMP START_EXEC
INT0_VECTOR:    LJMP $
                ds 5
TIM0_VECTOR:    LJMP $
                ds 5
INT1_VECTOR:    LJMP $
                ds 5
TIM1_VECTOR:    LJMP $
                ds 5
SER_VECTOR:     LJMP $
                ds 5
TIM2_VECTOR:    LJMP $
                ds 5
PCA_VECTOR:     LJMP $
                ds 13
SOF_VECTOR:     LJMP SOF_ISR
                ds 5
USB_FUNC_VECTOR LJMP FUNC_ISR
                ds 5
SUS_RSM_VECTOR  LJMP $
                ds 25h
TRAP_VECTOR:    LJMP $
END_VECTORS:
DEFINE CODE_SEGMENT, SPACE=CODE, ORG=FF4080H
SEGMENT CODE_SEGMENT
START_EXEC:
        CLR LC ; Enable high-clock mode
        mov SP,  #LOW(STACK_DATA)
        mov SPH, #HIGH(STACK_DATA)
        MOV EPINDEX, #01h
        MOV EPCON, #0Ch
        MOV TXCON, #04h ; 256/256
        MOV RXCON, #94h ; BULK mode
        MOV P1, #00h
        MOV IEN0, #80h  ; enable global interrupt
        MOV IEN1, #03h  ; enable function interrupt & SOF interrupt
        SETB FIE.3      ; enable receive interrupt
        MOV WR0, #00FFh;
        MOV WR2, #7000h
```

```
        MOV WR8, #00FFh
        MOV WR10, #6000h
        MOV WR12, #00FFh
        MOV WR14, #6100h
        MOV TMOD, #01h
        MOV R5, #00h
        SETB P1.2
        LJMP $  ; wait for function interrupt here
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This is the external block move routine which is called
; to move data from FIFO to external. A similar routine
; exists internal to the 8x930Ax controller.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_BLOCK_MOVE:
_0:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_1:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_2:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_3:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_4:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_5:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_6:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_7:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_8:     MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_9:     MOV R6, RXDAT
        MOV R7, RXDAT
```

```
        MOV @DR0, WR6
        INC DR0, #2
_10:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_11:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_12:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_13:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_14:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_15:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_16:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_17:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_18:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_19:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_20:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_21:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_22:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
```

```
        INC DR0, #2
_23:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_24:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_25:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_26:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_27:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_28:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_29:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_30:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
_31:    MOV R6, RXDAT
        MOV R7, RXDAT
        MOV @DR0, WR6
        INC DR0, #2
        RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This function serves as the function ISR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
FUNC_ISR:
CLR FRXD1        ; clear pending interrupt
        PUSH R4
JB RXSTAT.1, _ERROR0
        MOV R4, RXFLG
        ANL R4, #C0h
        CJNE R4, #00h, _DATA_RECEIVED0    ; data packet received
        POP R4
        RETI
_DATA_RECEIVED0:
_OKAY0:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; The LCALL on the following line calls either internal ROM or
; the BLOCK_MOVE routine located in this assembly file. One
; the LCALL line must be commented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        LCALL _BLOCK_MOVE      ; call external routine
        LCALL 3B83h     ; call internal routine
        SETB RXFFRC     ; release data set
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Check if a NEW packet was received before leaving the ISR.
; This may save some context-switch time.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
_MORE_DATA:
         JB RXSTAT.1, _ERROR0
        MOV R4, RXFLG
        ANL R4, #C0h
        CJNE R4, #00h, _DATA_RECEIVED1    ; data packet received
        POP R4
        RETI
_DATA_RECEIVED1:
_OKAY1:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The LCALL on the following line calls either internal ROM or
; the BLOCK_MOVE routine located in this assembly file. One
; the LCALL line must be commented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        LCALL _BLOCK_MOVE      ; call external routine
        LCALL 3B83h     ; call internal routine
        SETB RXFFRC     ; release data set

        POP R4
        RETI
SOF_ISR:
        RETI
_ERROR0:
        CLR IEN1.1       ; stop interrupt
        MOV P1, #0Ch
        LJMP $
_ERROR1:
        CLR IEN1.1       ; stop interrupt
        MOV P1, #0Dh
        LJMP $
_ERROR2:
        CLR IEN1.1       ; stop interrupt
        MOV P1, #0Eh
        LJMP $
_ERROR3:
        CLR IEN1.1       ; stop interrupt
        MOV P1, #0Fh
        LJMP $
DEFINE PDATA_SEG, SPACE=PDATA
SEGMENT PDATA_SEG
STACK_DATA:     ds 100h
```