High-Performance Serial Transfers Using the Intel386™ EX Microprocessor

The new Intel386™ EX microprocessor contains two independent asynchronous serial I/O (SIO) channels. Each channel is compatible with the 16450 UART found in PC-AT systems. Although a serial transfer rate exceeding 19.2K or 38.4K baud is not very useful in PC/DOS compatible applications, it could be vital for a traditional embedded application. The relevant question to be raised then is, "What is the highest serial transfer rate (i.e., baud rate) achievable?" The answer to this question has two components: one, the limits imposed on the achievable transfer rate by the design of the SIO module; and two, how fast one really can transfer the serial data without impacting the CPU performance substantially. Implications of both these issues will be discussed in this article. Also, two possible workarounds will be reviewed, one of which is offered on the Intel386 EX microprocessor.

Design Limits

The 16450 is an asynchronous block (i.e., not clocked) that has a 16-bit baud counter that divides the clock signal coming into the SIO block by a programmed value. It takes a certain amount of time for a signal to propagate through the gates to be available at the end of the chain. This mandates that the circuit cannot be operated beyond a certain frequency for reliable operation. The limit for the 16450 UART used in the Intel386 EX microprocessor is 17 MHz. This incoming clock will be divided by the baud divisor register. Even when this register is programmed to zero, it divides the clock by 2. Thus, the maximum baud clock derived for the Tx and Rx engines is $17/2 = 8.5$MHz. Also, for asynchronous transfers the baud clock is 16 times the baud rate due to sampling requirements. This would mean that maximum baud rate achievable with the 16450 SIO module used in the Intel386 EX microprocessor is approximately **530 Kbps.**

CPU Overhead for Serial Transfers

Although the SIO module can transfer at 530 Kbps, this does not necessarily mean that such a high baud rate can be achieved in a system environment. The following examines the impact of the serial transfer on the CPU bandwidth.

On the Intel386 EX microprocessor, the serial channel interrupts could be programmed to be tied to the on-chip master 8259A interrupt controller. For each channel, the receive and transmit interrupts are combined. The following is a typical scenario related to serial transfers:

- Whenever a serial interrupt line is asserted, it takes a short period of time before it is recognized by the 8259A.
- It goes through priority resolver logic to make sure that other higher priority interrupts are not present, then assuming that the serial interrupt is not masked, the 8259A places an interrupt request to the CPU.
- The interrupt is recognized on the instruction boundary, then the processor pushes the instruction pointer and flags on the stack before branching to the interrupt service routine (ISR).
- A typical ISR requires that all the segment and general-purpose registers be pushed on the stack so that their contents are not destroyed.
- Also, since receive and transmit interrupts are combined, the ISR needs to read the Interrupt ID register to determine the source of the interrupt and jump to appropriate receive or transmit subroutine.
- The serial data is then transferred from (to) the SIO buffer to (from) the memory.
- All registers are popped back from the stack and CPU continues from the Instruction Pointer saved on the stack.

Thus the approximate time[1] it takes to transfer one byte of serial data is:

$$T_{xfer\text{-}byte} = T_{recognition} + T_{latency} + T_{determine\text{-}source} + T_{move} + T_{return}$$
$$= 5 + 392 + 25 + 2 + 22$$
$$= 446 \text{ clocks}$$

Assuming serial transfer with 8-bit data, 1 start bit, and 1 stop bit, at a given baud rate, the number of characters transferred per second is:

$$\#\text{of char/sec} = \frac{\text{BaudRate}}{10}$$

Thus the total number of clock cycles needed to maintain the baud rate is:

$$T_{xfer\text{-}all\text{-}char} = 446 \times \left(\frac{\text{Baud Rate}}{10}\right)$$
$$= 44.6 \times \text{Baud Rate}$$

The resulting CPU overhead to maintain a given baud rate at a given frequency F is:

$$\%\text{CPUOverhead} = \left(\frac{44.6 \times \text{Baud Rate}}{F}\right) \times 100$$

Figure 1 shows the above relationship at a 25 MHz operating frequency.

Adding a FIFO to the SIO Block[2]

The addition of a 16-byte FIFO to the serial control unit will allow characters to be stuffed in this buffer until it's full. The interrupt will be sent to the CPU only when the FIFO buffer is full or about to be full. There will be some additional time required to input the data, increment the pointers, and move the data. The time to transfer 16 bytes of serial data at a time could be approximated at:

$$T_{xfer\text{-}fifo} = T_{xfer\text{-}byte} + 320$$
$$= 446 + 320$$
$$= 766 \text{ clocks}$$

1. The values used here for the number of clocks required for various tasks are hypothetical numbers based on typical cases. Actual numbers may vary based on other system design aspects.
2. This feature is not available on the Intel386 EX microprocessor.

Thus,

$$\%\text{CPUOverhead} = \left(\frac{76.6 \times \text{Baud Rate}}{16 \times F}\right) \times 100$$

Figure 2 depicts the above relationship for a 25 MHz operating frequency. This method of reducing CPU overhead to perform high-speed serial transfers is employed in the 16550 UART found in some newer PC systems.

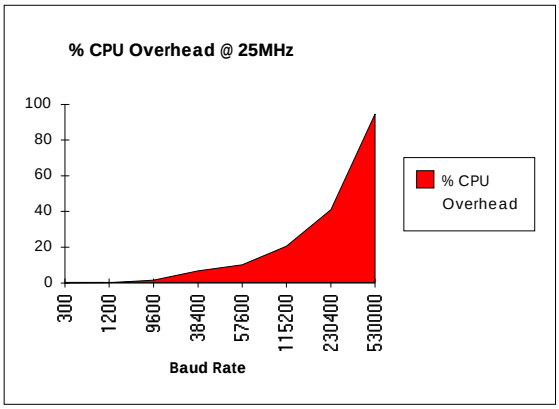

Figure 1. % CPU Overhead for Method Using UART Interrupts

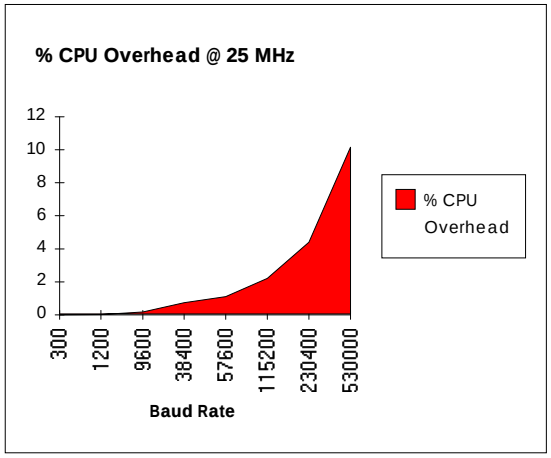

Figure 2. %CPU Overhead for Method Using FIFO in UART

Connecting Serial Interrupts to the DMA

It is clear from the above discussion that the main bottleneck to the process of serial data transfer is the interrupt latency of the processor. This bottleneck could be removed if the serial interrupts could initiate

a DMA transfer and then use the DMA to transfer the serial data to and from memory.

To calculate the transfer time with DMA, the time it takes to arbitrate the bus for DMA transfers needs to be accounted for.

$$T_{xfer\text{-}byte\text{-}DMA} = T_{wait\text{-}for\text{-}bus} + T_{move}$$
$$= 18 + 2$$
$$= 20 \text{ clocks}$$

Thus,

$$\%CPUOverhead = \left(\frac{2 \times Baud\ Rate}{F}\right) \times 100$$

Figure 3 shows the percentage of CPU overhead versus the baud rate at a 25 MHz operating frequency.

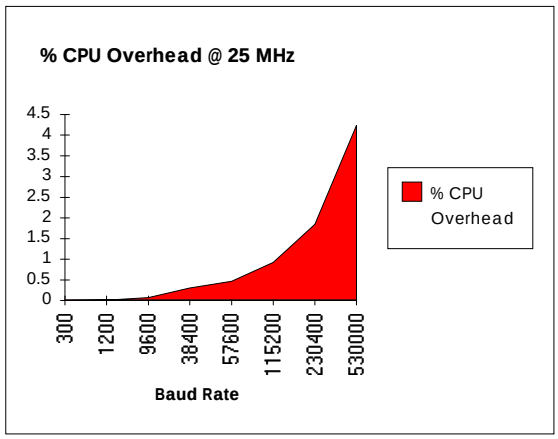

Figure 3. % CPU Overhead for Method Using DMA for Serial Transfer

As can be seen from the above discussion, the Intel386 EX microprocessor provides a method to perform high-speed serial transfers that is more efficient, in terms of reducing CPU overhead, than the 16550 implementation using FIFO buffers.

Synchronous Serial Transfer Rates

Similar arguments could be made for synchronous serial transfers when the SSIO interrupts are tied to the Intel386 EX microprocessor's DMA. Using this method, transfer rates up to 6 Mbps at 25 MHz operating frequency are possible.

Conclusion

The constraints imposed on performance by design implementation of the 16450 SIO block were reviewed. It was shown that, even in its simplest form, the SIO implementation on the Intel386 EX microprocessor could achieve the highest possible baud rate of 530 Kbps at 25 MHz operating frequency, although it will occupy 95% of CPU bandwidth. Performance improvements by using a 16-byte FIFO buffer (similar to a 16550 implementation) were discussed. It was also demonstrated that the feature to tie the serial interrupts to the DMA, provided on the Intel386 EX microprocessor, is the most effective method to free up the CPU. With this method, the maximum baud rate of 530 Kbps would occupy only 5% of CPU bandwidth. A similar scheme between SSIO and DMA would attain transfer rates up to 6 Mbps.