# 16550
# OPERATION MANUAL

# NS16550A Universal Asynchronous Receiver/Transmitter with FIFOs†

## General Description

The NS16550A is an improved version of the NS16450 Universal Asynchronous Receiver/Transmitter (UART). The improved specifications ensure compatibility with the NS32532 and other state-of-the-art CPUs. Functionally identical to the NS16450 on powerup (CHARACTER mode)* the NS16550A can be put into an alternate mode (FIFO mode) to relieve the CPU of excessive software overhead.

In this mode internal FIFOs are activated allowing 16 bytes (plus 3 bits of error data per byte in the RCVR FIFO) to be stored in both receive and transmit modes. All the logic is on chip to minimize system overhead and maximize system efficiency. Two pin functions have been changed to allow signalling of DMA transfers.

The UART performs serial-to-parallel conversion on data characters received from a peripheral device or a MODEM, and parallel-to-serial conversion on data characters received from the CPU. The CPU can read the complete status of the UART at any time during the functional operation. Status information reported includes the type and condition of the transfer operations being performed by the UART, as well as any error conditions (parity, overrun, framing, or break interrupt).

The UART includes a programmable baud rate generator that is capable of dividing the timing reference clock input by divisors of 1 to $(2^{16}-1)$, and producing a 16 × clock for driving the internal transmitter logic. Provisions are also included to use this 16 × clock to drive the receiver logic. The UART has complete MODEM-control capability, and a processor-interrupt system. Interrupts can be programmed to the user's requirements, minimizing the computing required to handle the communications link.

The UART is fabricated using National Semiconductor's advanced scaled N-channel silicon-gate MOS process, XMOS.

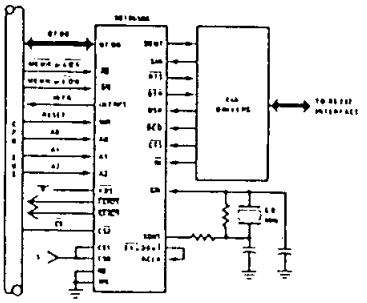
*Can also be reset to NS16450 Mode under software control.

†Note: This part has a patent pending.

## Features

- Capable of running all existing 16450 software.
- Pin for pin compatible with the existing 16450 except for CSOUT (24) and NC (29). The former CSOUT and NC pins are TXRDY and RXRDY, respectively.
- After reset, all registers are identical to the 16450 register set.
- In the FIFO mode transmitter and receiver are each buffered with 16 byte FIFO's to reduce the number of interrupts presented to the CPU.
- Adds or deletes standard asynchronous communication bits (start, stop, and parity) to or from the serial data.
- Holding and shift registers in the NS16450 Mode eliminate the need for precise synchronization between the CPU and serial data.
- Independently controlled transmit, receive, line status, and data set interrupts.
- Programmable baud generator divides any input clock by 1 to $(2^{16}-1)$ and generates the 16 × clock.
- Independent receiver clock input.
- MODEM control functions (CTS, RTS, DSR, DTR, RI, and DCD).
- Fully programmable serial-interface characteristics:
  - 5-, 6-, 7-, or 8-bit characters
  - Even, odd, or no-parity bit generation and detection
  - 1-, 1½-, or 2-stop bit generation
  - Baud generation (DC to 256k baud).
- False start bit detection.
- Complete status reporting capabilities.
- TRI-STATE® TTL drive for the data and control buses.
- Line break generation and detection.
- Internal diagnostic capabilities:
  - Loopback controls for communications link fault isolation
  - Break, parity, overrun, framing error simulation.
- Full prioritized interrupt system controls.

## Basic Configuration



TL/C/8852–1

# Table of Contents

## 1.0 Absolute Maximum Ratings

**If Military/Aerospace specified devices are required, contact the National Semiconductor Sales Office/ Distributors for availability and specifications.**

| | |
|---|---|
| Temperature Under Bias | 0°C to +70°C |
| Storage Temperature | −65°C to +150°C |
| All Input or Output Voltages with Respect to $V_{SS}$ | −0.5V to +7.0V |
| Power Dissipation | 1W |

*Note: Maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under DC electrical characteristics.*

## 2.0 DC Electrical Characteristics

$T_A$ = 0°C to +70°C, $V_{CC}$ = +5V ±5%, $V_{SS}$ = 0V, unless otherwise specified.

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| $V_{ILX}$ | Clock Input Low Voltage | | −0.5 | 0.8 | V |
| $V_{IHX}$ | Clock Input High Voltage | | 2.0 | $V_{CC}$ | V |
| $V_{IL}$ | Input Low Voltage | | −0.5 | 0.8 | V |
| $V_{IH}$ | Input High Voltage | | 2.0 | $V_{CC}$ | V |
| $V_{OL}$ | Output Low Voltage | $I_{OL}$ = 1.6 mA on all (Note 1) | | 0.4 | V |
| $V_{OH}$ | Output High Voltage | $I_{OH}$ = −1.0 mA (Note 1) | 2.4 | | V |
| $I_{CC}$(AV) | Avg. Power Supply Current ($V_{CC}$) | $V_{CC}$ = 5.25V<br>No Loads on output<br>SIN, DSR, DCD,<br>CTS, RI = 2.0V<br>All other inputs = 0.8V | | 160<br>(Note 2)<br><br>140<br>(Note 3) | mA<br><br>mA |
| $I_{IL}$ | Input Leakage | $V_{CC}$ = 5.25V, $V_{SS}$ = 0V<br>All other pins floating.<br>$V_{IN}$ = 0V, 5.25V | | ±10 | µA |
| $I_{CL}$ | Clock Leakage | | | ±10 | µA |
| $I_{OZ}$ | TRI-STATE Leakage | $V_{CC}$ = 5.25V, $V_{SS}$ = 0V<br>$V_{OUT}$ = 0V, 5.25V<br>1) Chip deselected<br>2) WRITE mode,<br>chip selected | | ±20 | µA |
| $V_{ILMR}$ | MR Schmitt $V_{IL}$ | | | 0.8 | V |
| $V_{IHMR}$ | MR Schmitt $V_{IH}$ | | 2.0 | | V |

Note 1: Does not apply to XOUT
Note 2: $T_A$ = 25°C
Note 3: $T_A$ = 70°C

## Capacitance $T_A$ = 25°C, $V_{CC}$ = $V_{SS}$ = 0V

| Symbol | Parameter | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $C_{XIN}$ | Clock Input Capacitance | $f_c$ = 1 MHz<br>Unmeasured pins returned to $V_{SS}$ | | 15 | 20 | pF |
| $C_{XOUT}$ | Clock Output Capacitance | | | 20 | 30 | pF |
| $C_{IN}$ | Input Capacitance | | | 6 | 10 | pF |
| $C_{OUT}$ | Output Capacitance | | | 10 | 20 | pF |

## 3.0 AC Electrical Characteristics $T_A$ = 0°C to +70°C, $V_{CC}$ = +5V ±5%

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| $t_{ADS}$ | Address Strobe Width | | 60 | | ns |
| $t_{AH}$ | Address Hold Time | | 0 | | ns |
| $t_{AR}$ | $\overline{RD}$, RD Delay from Address | (Note 1) | 30 | | ns |
| $t_{AS}$ | Address Setup Time | | 60 | | ns |
| $t_{AW}$ | $\overline{WR}$, WR Delay from Address | (Note 1) | 30 | | ns |
| $t_{CH}$ | Chip Select Hold Time | | 0 | | ns |
| $t_{CS}$ | Chip Select Setup Time | | 60 | | ns |
| $t_{CSR}$ | RD, RD Delay from Chip Select | (Note 1) | 30 | | ns |
| $t_{CSW}$ | $\overline{WR}$, WR Delay from Select | (Note 1) | 30 | | ns |
| $t_{DH}$ | Data Hold Time | | 30 | | ns |
| $t_{DS}$ | Data Setup Time | | 30 | | ns |
| $t_{HZ}$ | RD, RD to Floating Data Delay | @100 pF loading (Note 3) | 0 | 100 | ns |
| $t_{MR}$ | Master Reset Pulse Width | | 5 | | μs |
| $t_{RA}$ | Address Hold Time from $\overline{RD}$, RD | (Note 1) | 20 | | ns |
| $t_{RC}$ | Read Cycle Delay | | 125 | | ns |
| $t_{RCS}$ | Chip Select Hold Time from $\overline{RD}$, RD | (Note 1) | 20 | | ns |
| $t_{RD}$ | $\overline{RD}$, RD Strobe Width | | 125 | | ns |
| $t_{RDD}$ | $\overline{RD}$, RD to Driver Enable/Disable | @100 pF loading (Note 3) | | 60 | ns |
| $t_{RVD}$ | Delay from RD, RD to Data | @100 pF loading | | 125 | ns |
| $t_{WA}$ | Address Hold Time from $\overline{WR}$, WR | (Note 1) | 20 | | ns |
| $t_{WC}$ | Write Cycle Delay | | 150 | | ns |
| $t_{WCS}$ | Chip Select Hold Time from $\overline{WR}$, WR | (Note 1) | 20 | | ns |
| $t_{WR}$ | $\overline{WR}$, WR Strobe Width | | 100 | | ns |
| $t_{XH}$ | Duration of Clock High Pulse | External Clock (8.0 MHz Max.) | 55 | | ns |
| $t_{XL}$ | Duration of Clock Low Pulse | External Clock (8.0 MHz Max.) | 55 | | ns |
| RC | Read Cycle = $t_{AR}$ + $t_{RD}$ + $t_{RC}$ | (Note 4) | 280 | | ns |
| WC | Write Cycle = $t_{AW}$ + $t_{WR}$ + $t_{WC}$ | | 280 | | ns |
| **Baud Generator** | | | | | |
| N | Baud Divisor | | 1 | $2^{16}-1$ | |
| $t_{BHD}$ | Baud Output Positive Edge Delay | 100 pF Load | | 175 | ns |
| $t_{BLD}$ | Baud Output Negative Edge Delay | 100 pF Load | | 175 | ns |
| $t_{HW}$ | Baud Output Up Time | $f_X$ = 8.0 MHz, ÷2, 100 pF Load | 75 | | ns |
| $t_{LW}$ | Baud Output Down Time | $f_X$ = 8.0 MHz, ÷2, 100 pF Load | 100 | | ns |
| **Receiver** | | | | | |
| $t_{RINT}$ | Delay from $\overline{RD}$, RD (RD RBR/or RD LSR) to Reset Interrupt | 100 pF Load | | 1 | μs |
| $t_{SCD}$ | Delay from RCLK to Sample Time | | | 2 | μs |
| $t_{SINT}$ | Delay from Stop to Set Interrupt | (Note 2) | | 1 | RCLK Cycles |

Note 1: Applicable only when $\overline{ADS}$ is tied low

Note 2: In the FIFO mode (FCR0 = 1) the trigger level interrupts, the receiver data available indication, the active RXRDY indication and the overrun error indication will be delayed 3 RCLKs. Status indicators (PE, FE, BI) will be delayed 3 RCLKs after the first byte has been received. For subsequently received bytes these indicators will be updated immediately after RDRBR goes inactive. Timeout interrupt is delayed 8 RCLKs.

Note 3: Charge and discharge time is determined by $V_{OL}$, $V_{OH}$ and the external loading.

Note 4: In FIFO mode RC = 425 ns (minimum) between reads of the receiver FIFO and the status registers (interrupt identification register or line status register).

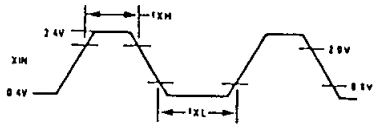
## 3.0 AC Electrical Characteristics (Continued)

| Symbol | Parameter | Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| **Transmitter** | | | | | |
| $t_{HR}$ | Delay from $\overline{WR}$, WR (WR THR) to Reset Interrupt | 100 pF Load | | 175 | ns |
| $t_{IR}$ | Delay from $\overline{RD}$, RD (RD IIR) to Reset Interrupt (THRE) | 100 pF Load | | 250 | ns |
| $t_{IRS}$ | Delay from Initial INTR Reset to Transmit Start | — | 8 | 24 | BAUDOUT Cycles |
| $t_{SI}$ | Delay from Initial Write to Interrupt | (Note 1) | 16 | 24 | BAUDOUT Cycles |
| $t_{STI}$ | Delay from Stop to Interrupt (THRE) | (Note 1) | 8 | 8 | BAUDOUT Cycles |
| $t_{SXA}$ | Delay from Start to TXRDY active | 100 pF Load | | 8 | BAUDOUT Cycles |
| $t_{WXI}$ | Delay from Write to TXRDY inactive | 100 pF Load | | 195 | ns |
| **Modem Control** | | | | | |
| $t_{MDO}$ | Delay from $\overline{WR}$, WR (WR MCR) to Output | 100 pF Load | | 200 | ns |
| $t_{RIM}$ | Delay to Reset Interrupt from $\overline{RD}$, RD (RD MSR) | 100 pF Load | | 250 | ns |
| $t_{SIM}$ | Delay to Set Interrupt from MODEM Input | 100 pF Load | | 250 | ns |

Note 1: This delay will be lengthened by 1 character time, minus the last stop bit time if the transmitter interrupt delay circuit is active. (See FIFO Interrupt Mode Operation).

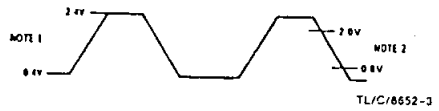
## 4.0 Timing Waveforms (All timings are referenced to valid 0 and valid 1)

**External Clock Input (8.0 MHz Max.)**




TL/C/8652-2

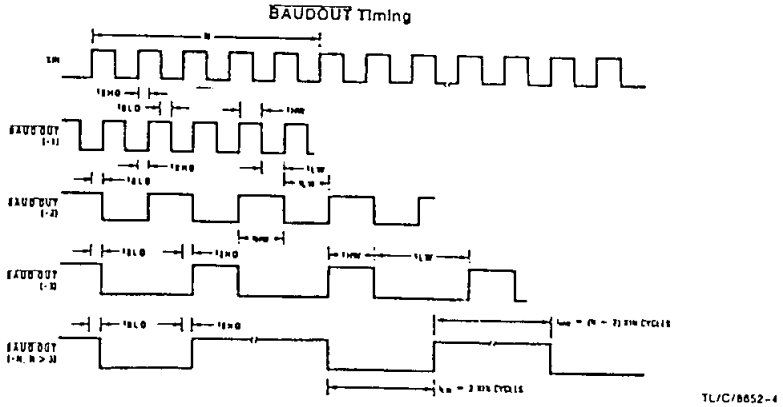
**AC Test Points**




TL/C/8652-3

Note 1: The 2.4V and 0.4V levels are the voltages that the inputs are driven to during AC testing.
Note 2: The 2.0V and 0.8V levels are the voltages at which the timing tests are made.

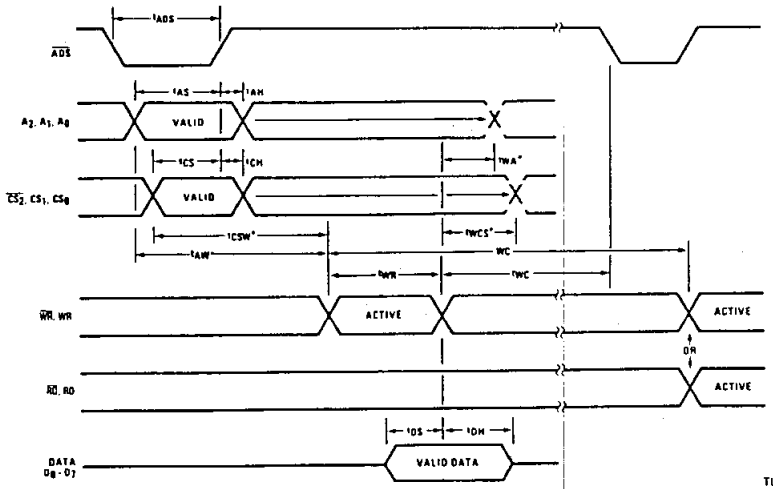
**$\overline{BAUDOUT}$ Timing**




TL/C/8652-4

## 4.0 Timing Waveforms (Continued)

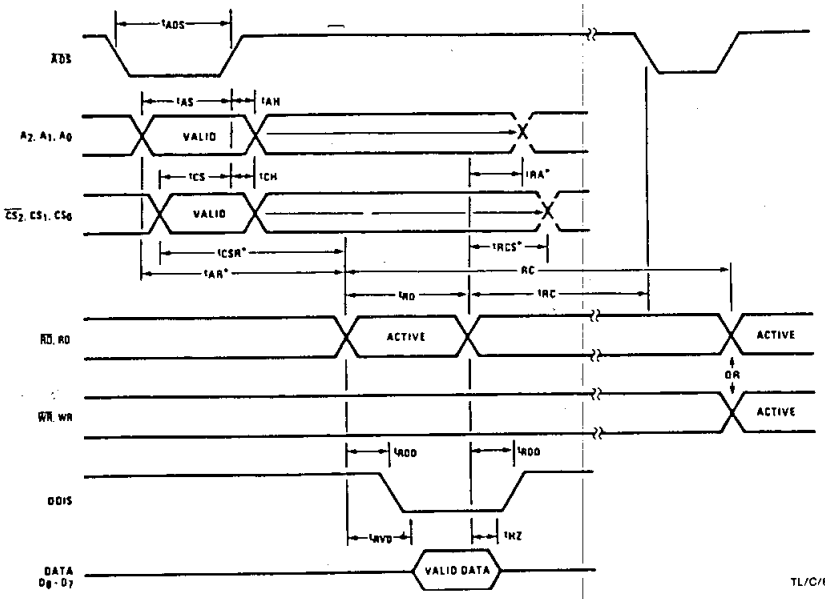
### Write Cycle




TL/C/8652-5

*Applicable Only When $\overline{ADS}$ is Tied Low.

### Read Cycle

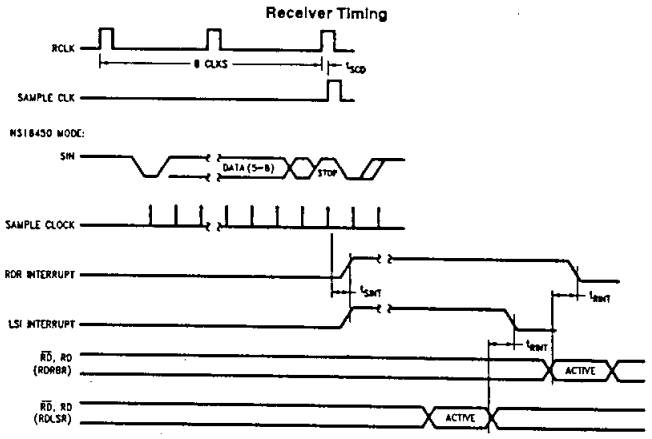



TL/C/8652-6

*Applicable Only When $\overline{ADS}$ is Tied Low.

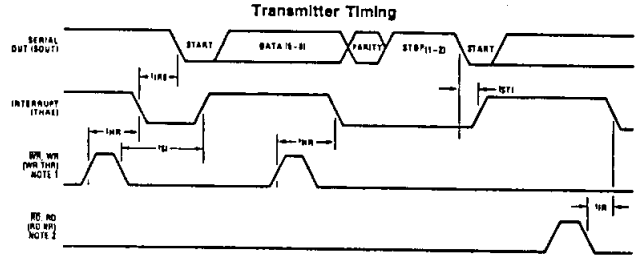
## 4.0 Timing Waveforms (Continued)

### Receiver Timing



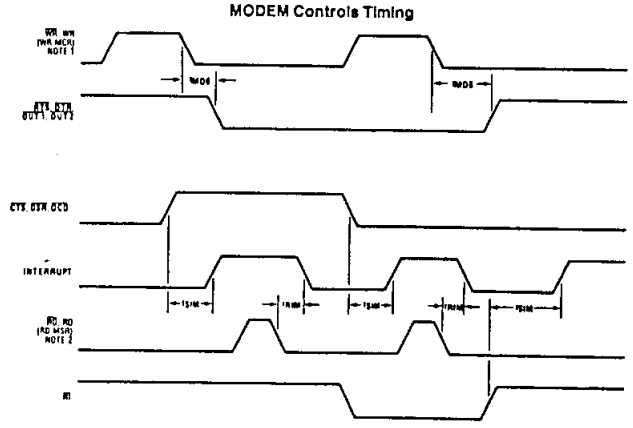

TL/C/8652-7

### Transmitter Timing




TL/C/8652-8

### MODEM Controls Timing

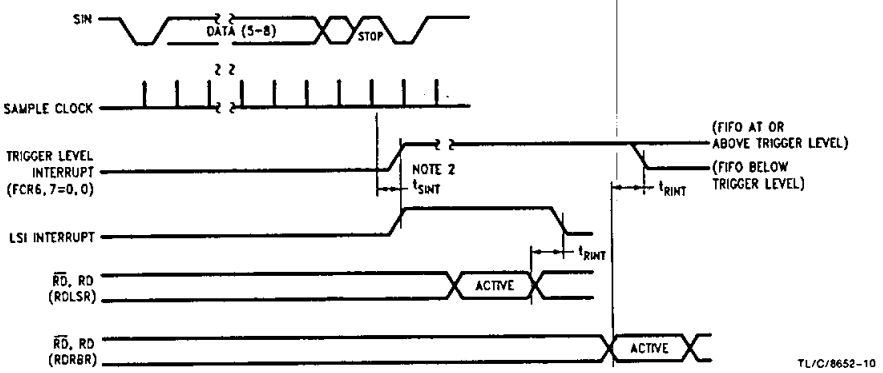



TL/C/8652-9

Note 1: See Write Cycle Timing
Note 2: See Read Cycle Timing

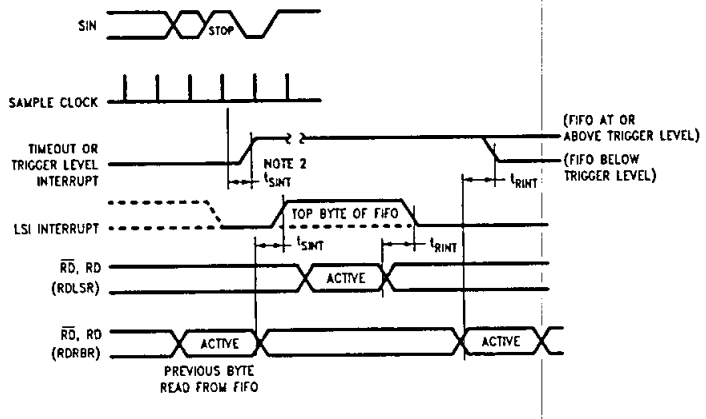
## 4.0 Timing Waveforms (Continued)

**RCVR FIFO First Byte (This Sets RDR)**



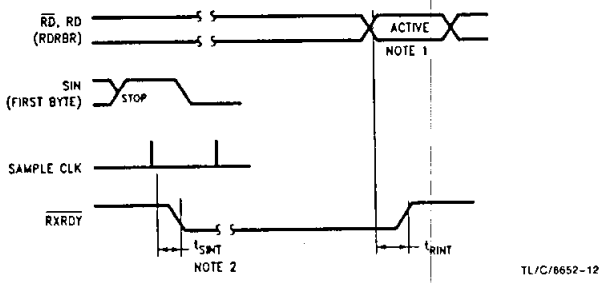

TL/C/8652-10

**RCVR FIFO Bytes Other Than the First Byte (RDR Is Already Set)**




TL/C/8652-11

**Receiver Ready (Pin 29) FCR0 = 0 or FCR0 = 1 and FCR3 = 0 (Mode 0)**



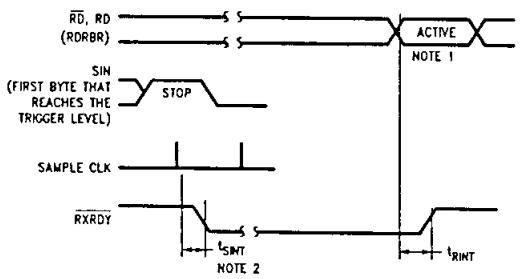

TL/C/8652-12

Note 1: This is the reading of the last byte in the FIFO.

Note 2: If FCR0 = 1, then $t_{SINT}$ = 3 RCLKs. For a timeout interrupt, $t_{SINT}$ = 8 RCLKs.

## 4.0 Timing Waveforms (Continued)

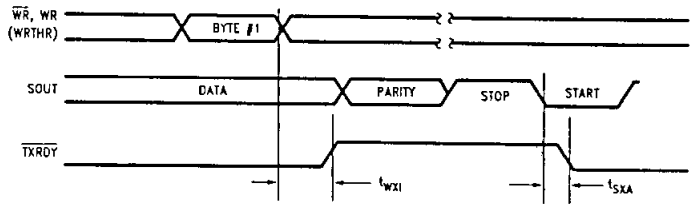
**Receiver Ready (Pin 29) FCR0 = 1 and FCR3 = 1 (Mode 1)**




TL/C/8652-13

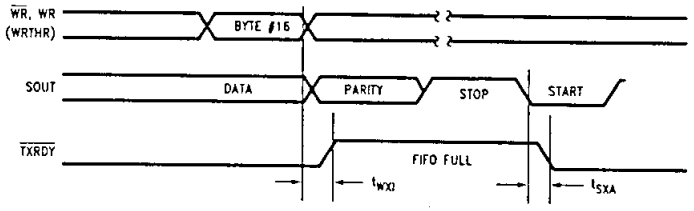
**Note 1:** This is the reading of the last byte in the FIFO.
**Note 2:** If FCR0 = 1, $t_{SINT}$ = 3 RCLKs.

**Transmitter Ready (Pin 24) FCR0 = 0 or FCR0 = 1 and FCR3 = 0 (Mode 0)**



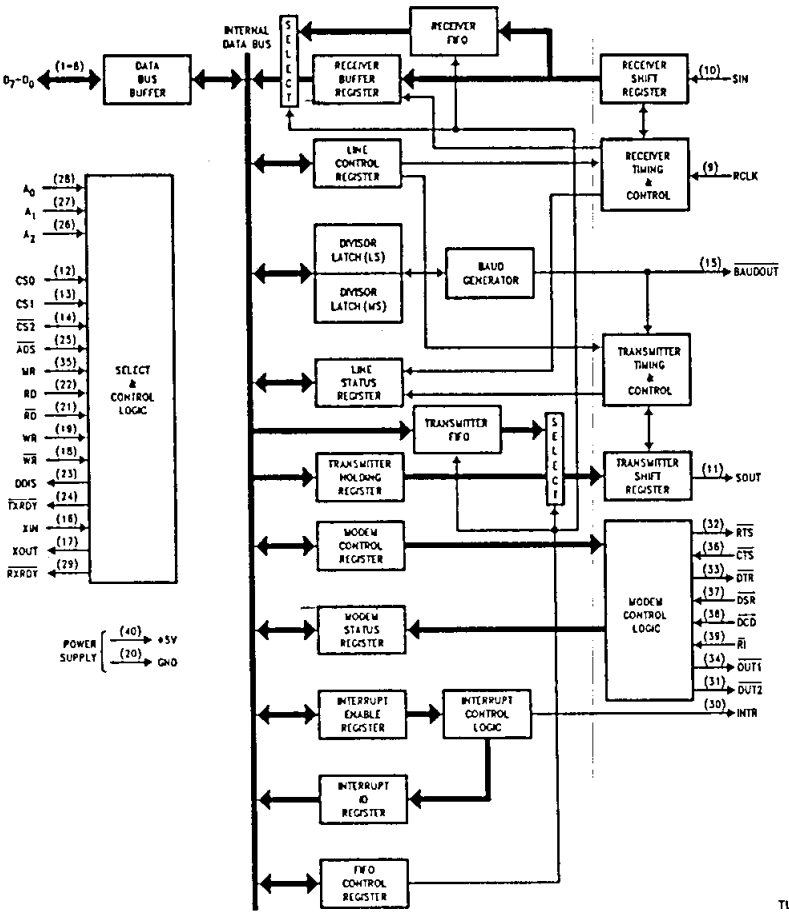

TL/C/8652-14

**Transmitter Ready (Pin 24) FCR0 = 1 and FCR3 = 1 (Mode 1)**




TL/C/8652-15

## 5.0 Block Diagram




Note: Applicable pinout numbers are included within parenthesis.

## 6.0 Pin Descriptions

The following describes the function of all UART pins. Some of these descriptions reference internal circuits.

In the following descriptions, a low represents a logic 0 (0V nominal) and a high represents a logic 1 (+2.4V nominal).

### 6.1 INPUT SIGNALS

**Chip Select (CS0, CS1, $\overline{CS2}$), Pins 12–14:** When CS0 and CS1 are high and $\overline{CS2}$ is low, the chip is selected. This enables communication between the UART and the CPU. The positive edge of an active Address Strobe signal latches the decoded chip select signals, completing chip selection. If $\overline{ADS}$ is always low, valid chip selects should stabilize according to the $t_{CSW}$ parameter.

**Read (RD, $\overline{RD}$), Pins 22 and 21:** When RD is high or $\overline{RD}$ is low while the chip is selected, the CPU can read status information or data from the selected UART register.

Note: Only an active RD or $\overline{RD}$ input is required to transfer data from the UART during a read operation. Therefore, tie either the RD input permanently low or the $\overline{RD}$ input permanently high, when it is not used.

**Write (WR, $\overline{WR}$), Pins 19 and 18:** When WR is high or $\overline{WR}$ is low while the chip is selected, the CPU can write control words or data into the selected UART register.

Note: Only an active WR or $\overline{WR}$ input is required to transfer data to the UART during a write operation. Therefore, tie either the WR input permanently low or the $\overline{WR}$ input permanently high, when it is not used.

## 6.0 Pin Descriptions (Continued)

**Address Strobe ($\overline{ADS}$), Pin 25:** The positive edge of an active Address Strobe ($\overline{ADS}$) signal latches the Register Select (A0, A1, A2) and Chip Select (CS0, CS1, CS2) signals.

Note: An active $\overline{ADS}$ input is required when the Register Select (A0, A1, A2) signals are not stable for the duration of a read or write operation. If not required, tie the $\overline{ADS}$ input permanently low.

**Register Select (A0, A1, A2), Pins 26–28:** Address signals connected to these 3 inputs select a UART register for the CPU to read from or write to during data transfer. A table of registers and their addresses is shown below. Note that the state of the Divisor Latch Access Bit (DLAB), which is the most significant bit of the Line Control Register, affects the selection of certain UART registers. The DLAB must be set high by the system software to access the Baud Generator Divisor Latches.

**Master Reset (MR), Pin 35:** When this input is high, it clears all the registers (except the Receiver Buffer, Transmitter Holding, and Divisor Latches), and the control logic of the UART. The states of various output signals (SOUT, INTR, $\overline{OUT\ 1}$, $\overline{OUT\ 2}$, $\overline{RTS}$, $\overline{DTR}$) are affected by an active MR input (Refer to Table I.) This input is buffered with a TTL-compatible Schmitt Trigger with 0.5V typical hysteresis.

REGISTER ADDRESSES

| DLAB | $A_2$ | $A_1$ | $A_0$ | Register |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Receiver Buffer (read), Transmitter Holding Register (write) |
| 0 | 0 | 0 | 1 | Interrupt Enable |
| X | 0 | 1 | 0 | Interrupt Identification (read) |
| X | 0 | 1 | 0 | FIFO Control (write) |
| X | 0 | 1 | 1 | Line Control |
| X | 1 | 0 | 0 | MODEM Control |
| X | 1 | 0 | 1 | Line Status |
| X | 1 | 1 | 0 | MODEM Status |
| X | 1 | 1 | 1 | Scratch |
| 1 | 0 | 0 | 0 | Divisor Latch (least significant byte) |
| 1 | 0 | 0 | 1 | Divisor Latch (most significant byte) |

**Receiver Clock (RCLK), Pin 9:** This input is the 16 × baud rate clock for the receiver section of the chip.

**Serial Input (SIN) Pin 10:** Serial data input from the communications link (peripheral device, MODEM, or data set).

**Clear to Send ($\overline{CTS}$), Pin 36:** When low, this indicates that the MODEM or data set is ready to exchange data. The $\overline{CTS}$ signal is a MODEM status input whose conditions can be tested by the CPU reading bit 4 (CTS) of the MODEM Status Register. Bit 4 is the complement of the $\overline{CTS}$ signal. Bit 0 (DCTS) of the MODEM Status Register indicates whether the $\overline{CTS}$ input has changed state since the previous reading of the MODEM Status Register. $\overline{CTS}$ has no effect on the Transmitter.

Note: Whenever the CTS bit of the MODEM Status Register changes state, an interrupt is generated if the MODEM Status Interrupt is enabled.

**Data Set Ready ($\overline{DSR}$), Pin 37:** When low, this indicates that the MODEM or data set is ready to establish the communications link with the UART. The $\overline{DSR}$ signal is a MODEM status input whose condition can be tested by the CPU reading bit 5 (DSR) of the MODEM Status Register. Bit 5 is the complement of the $\overline{DSR}$ signal. Bit 1 (DDSR) of the MODEM Status Register indicates whether the $\overline{DSR}$ input has changed state since the previous reading of the MODEM Status Register.

Note: Whenever the DSR bit of the MODEM Status Register changes state, an interrupt is generated if the MODEM Status Interrupt is enabled.

**Data Carrier Detect ($\overline{DCD}$), Pin 38:** When low, indicates that the data carrier has been detected by the MODEM or data set. The $\overline{DCD}$ signal is a MODEM status input whose condition can be tested by the CPU reading bit 7 (DCD) of the MODEM Status Register. Bit 7 is the complement of the $\overline{DCD}$ signal. Bit 3 (DDCD) of the MODEM Status Register indicates whether the $\overline{DCD}$ input has changed state since the previous reading of the MODEM Status Register. $\overline{DCD}$ has no effect on the receiver.

Note: Whenever the DCD bit of the MODEM Status Register changes state, an interrupt is generated if the MODEM Status Interrupt is enabled.

**Ring Indicator ($\overline{RI}$), Pin 39:** When low, this indicates that a telephone ringing signal has been received by the MODEM or data set. The $\overline{RI}$ signal is a MODEM status input whose condition can be tested by the CPU reading bit 6 (RI) of the MODEM Status Register. Bit 6 is the complement of the $\overline{RI}$ signal. Bit 2 (TERI) of the MODEM Status Register indicates whether the $\overline{RI}$ input signal has changed from a low to a high state since the previous reading of the MODEM Status Register.

Note: Whenever the RI bit of the MODEM Status Register changes from a high to a low state, an interrupt is generated if the MODEM Status Interrupt is enabled.

**$V_{CC}$, Pin 40:** +5V supply.

**$V_{SS}$, Pin 20:** Ground (0V) reference.

### 6.2 OUTPUT SIGNALS

**Data Terminal Ready ($\overline{DTR}$), Pin 33:** When low, this informs the MODEM or data set that the UART is ready to establish a communications link. The $\overline{DTR}$ output signal can be set to an active low by programming bit 0 (DTR) of the MODEM Control Register to a high level. A Master Reset operation sets this signal to its inactive (high) state. Loop mode operation holds this signal in its inactive state.

**Request to Send ($\overline{RTS}$), Pin 32:** When low, this informs the MODEM or data set that the UART is ready to exchange data. The $\overline{RTS}$ output signal can be set to an active low by programming bit 1 (RTS) of the MODEM Control Register. A Master Reset operation sets this signal to its inactive (high) state. Loop mode operation holds this signal in its inactive state.

**Output 1 ($\overline{OUT\ 1}$), Pin 34:** This user-designated output can be set to an active low by programming bit 2 (OUT 1) of the MODEM Control Register to a high level. A Master Reset operation sets this signal to its inactive (high) state. Loop mode operation holds this signal in its inactive state. In the XMOS parts this will achieve TTL levels.

**Output 2 ($\overline{OUT\ 2}$), Pin 31:** This user-designated output that can be set to an active low by programming bit 3 (OUT 2) of the MODEM Control Register to a high level. A Master Reset operation sets this signal to its inactive (high) state. Loop mode operation holds this signal in its inactive state. In the XMOS parts this will achieve TTL levels.

**TXRDY, RXRDY, Pins 24, 29:** Transmitter and Receiver DMA signalling is available through two pins (24 and 29). When operating in the FIFO mode, one of two types of DMA signalling per pin can be selected via FCR3. When operating as in the NS16450 Mode, only DMA mode 0 is allowed. Mode 0 supports single transfer DMA where a transfer is made between CPU bus cycles. Mode 1 supports multi-

## 6.0 Pin Descriptions (Continued)

transfer DMA where multiple transfers are made continuously until the RCVR FIFO has been emptied or the XMIT FIFO has been filled.

**RXRDY Mode 0:** When in the NS16450 Mode (FCR0 = 0) or in the FIFO Mode (FCR0 = 1, FCR3 = 0) and there is at least 1 character in the RCVR FIFO or RCVR holding register, the RXRDY pin (29) will be low active. Once it is activated the RXRDY pin will go inactive when there are no more characters in the FIFO or holding register.

**RXRDY Mode 1:** In the FIFO Mode (FCR0 = 1) when the FCR3 = 1 and the trigger level or the timeout has been reached, the RXRDY pin will go low active. Once it is activated it will go inactive when there are no more characters in the FIFO or holding register.

**TXRDY Mode 0:** In the NS16450 Mode (FCR0 = 0) or in the FIFO Mode (FCR0 = 1, FCR3 = 0) and there are no characters in the XMIT FIFO or XMIT holding register, the $\overline{\text{TXRDY}}$ pin (24) will be low active. Once it is activated the TXRDY pin will go inactive after the first character is loaded into the XMIT FIFO or holding register.

**TXRDY Mode 1:** In the FIFO Mode (FCR0 = 1) when FCR3 = 1 and there is at least one unfilled position in the XMIT FIFO, it will go low active. This pin will become inactive when the XMIT FIFO is completely full.

**Driver Disable (DDIS), Pin 23:** This goes low whenever the CPU is reading data from the UART. It can disable or control the direction of a data bus transceiver between the CPU and the UART.

**Baud Out ($\overline{\text{BAUDOUT}}$), Pin 15:** This is the 16 × clock signal from the transmitter section of the UART. The clock rate is equal to the main reference oscillator frequency divided by the specified divisor in the Baud Generator Divisor Latches. The $\overline{\text{BAUDOUT}}$ may also be used for the receiver section by tying this output to the RCLK input of the chip.

**Interrupt (INTR), Pin 30:** This pin goes high whenever any one of the following interrupt types has an active high condition and is enabled via the IER: Receiver Error Flag; Received Data Available; timeout (FIFO Mode only); Transmitter Holding Register Empty; and MODEM Status. The INTR signal is reset low upon the appropriate interrupt service or a Master Reset operation.

**Serial Output (SOUT), Pin 11:** Composite serial data output to the communications link (peripheral, MODEM or data set). The SOUT signal is set to the Marking (logic 1) state upon a Master Reset operation.

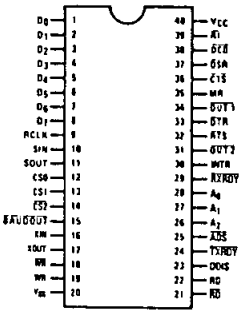
### 6.3 INPUT/OUTPUT SIGNALS

**Data ($D_7$-$D_0$) Bus, Pins 1-8:** This bus comprises eight TRI-STATE input/output lines. The bus provides bidirectional communications between the UART and the CPU. Data, control words, and status information are transferred via the $D_7$-$D_0$ Data Bus.

**External Clock Input/Output (XIN, XOUT) Pins 16 and 17:** These two pins connect the main timing reference (crystal or signal clock) to the UART.

## 7.0 Connection Diagrams

**Dual-In-Line Package**

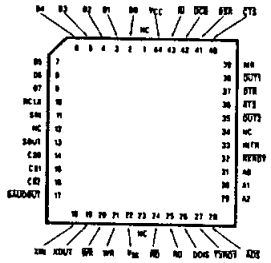


TL/C/8652–17

**Top View**

**Order Number NS16550AN**
**See NS Package Number N40A**

**Chip Carrier Package**



TL/C/8652–18

**Top View**

**Order Number NS16550AV**
**See NS Package Number V44A**

**TABLE I. UART Reset Configuration**

| Register/Signal | Reset Control | Reset State |
|---|---|---|
| Interrupt Enable Register | Master Reset | **0000** 0000 (Note 1) |
| Interrupt Identification Register | Master Reset | 00**00** 0001 |
| FIFO Control | Master Reset | 00**00** 0000 |
| Line Control Register | Master Reset | 0000 0000 |
| MODEM Control Register | Master Reset | **0000** 0000 |
| Line Status Register | Master Reset | 0110 0000 |
| MODEM Status Register | Master Reset | XXXX 0000 (Note 2) |
| SOUT | Master Reset | High |
| INTR (RCVR Errs) | Read LSR/MR | Low |
| INTR (RCVR Data Ready) | Read RBR/MR | Low |
| INTR (THRE) | Read IIR/Write THR/MR | Low |
| INTR (Modem Status Changes) | Read MSR/MR | Low |
| $\overline{OUT\ 2}$ | Master Reset | High |
| $\overline{RTS}$ | Master Reset | High |
| $\overline{DTR}$ | Master Reset | High |
| $\overline{OUT\ 1}$ | Master Reset | High |
| RCVR FIFO | MR/FCR1•FCR0/ΔFCR0 | All Bits Low |
| XMIT FIFO | MR/FCR1•FCR0/ΔFCR0 | All Bits Low |

**Note 1:** Boldface bits are permanently low.

**Note 2:** Bits 7–4 are driven by the input signals.

**TABLE II. Summary of Registers**

| Bit No. | Register Address | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 DLAB = 0 | 0 DLAB = 0 | 1 DLAB = 0 | 2 | 2 | 3 | 4 | 5 | 6 | 7 | 0 DLAB = 1 | 1 DLAB = 1 |
| | Receiver Buffer Register (Read Only) | Transmitter Holding Register (Write Only) | Interrupt Enable Register | Interrupt Ident. Register (Read Only) | FIFO Control Register (Write Only) | Line Control Register | MODEM Control Register | Line Status Register | MODEM Status Register | Scratch Register | Divisor Latch (LS) | Divisor Latch (MS) |
| | RBR | THR | IER | IIR | FCR | LCR | MCR | LSR | MSR | SCR | DLL | DLM |
| 0 | Data Bit 0 (Note 1) | Data Bit 0 | Enable Received Data Available Interrupt (ERBFI) | "0" if Interrupt Pending | FIFO Enable | Word Length Select Bit 0 (WLS0) | Data Terminal Ready (DTR) | Data Ready (DR) | Delta Clear to Send (DCTS) | Bit 0 | Bit 0 | Bit 8 |
| 1 | Data Bit 1 | Data Bit 1 | Enable Transmitter Holding Register Empty Interrupt (ETBEI) | Interrupt ID Bit (0) | RCVR FIFO Reset | Word Length Select Bit 1 (WLS1) | Request to Send (RTS) | Overrun Error (OE) | Delta Data Set Ready (DDSR) | Bit 1 | Bit 1 | Bit 9 |
| 2 | Data Bit 2 | Data Bit 2 | Enable Receiver Line Status Interrupt (ELSI) | Interrupt ID Bit (1) | XMIT FIFO Reset | Number of Stop Bits (STB) | Out 1 | Parity Error (PE) | Trailing Edge Ring Indicator (TERI) | Bit 2 | Bit 2 | Bit 10 |
| 3 | Data Bit 3 | Data Bit 3 | Enable MODEM Status Interrupt (EDSSI) | Interrupt ID Bit (2) (Note 2) | DMA Mode Select | Parity Enable (PEN) | Out 2 | Framing Error (FE) | Delta Data Carrier Detect (DDCD) | Bit 3 | Bit 3 | Bit 11 |
| 4 | Data Bit 4 | Data Bit 4 | 0 | 0 | Reserved | Even Parity Select (EPS) | Loop | Break Interrupt (BI) | Clear to Send (CTS) | Bit 4 | Bit 4 | Bit 12 |
| 5 | Data Bit 5 | Data Bit 5 | 0 | 0 | Reserved | Stick Parity | 0 | Transmitter Holding Register (THRE) | Data Set Ready (DSR) | Bit 5 | Bit 5 | Bit 13 |
| 6 | Data Bit 6 | Data Bit 6 | 0 | FIFOs Enabled (Note 2) | RCVR Trigger (LSB) | Set Break | 0 | Transmitter Empty (TEMT) | Ring Indicator (RI) | Bit 6 | Bit 6 | Bit 14 |
| 7 | Data Bit 7 | Data Bit 7 | 0 | FIFOs Enabled (Note 2) | RCVR Trigger (MSB) | Divisor Latch Access Bit (DLAB) | 0 | Error in RCVR FIFO (Note 2) | Data Carrier Detect (DCD) | Bit 7 | Bit 7 | Bit 15 |

**Note 1:** Bit 0 is the least significant bit. It is the first bit serially transmitted or received.

**Note 2:** These bits are always 0 in the NS16450 Mode.

## 8.0 Registers

The system programmer may access any of the UART registers summarized in Table II via the CPU. These registers control UART operations including transmission and reception of data. Each register bit in Table II has its name and reset state shown.

### 8.1 LINE CONTROL REGISTER

The system programmer specifies the format of the asynchronous data communications exchange and set the Divisor Latch Access bit via the Line Control Register (LCR). The programmer can also read the contents of the Line Control Register. The read capability simplifies system programming and eliminates the need for separate storage in system memory of the line characteristics. Table II shows the contents of the LCR. Details on each bit follow:

**Bits 0 and 1:** These two bits specify the number of bits in each transmitted or received serial character. The encoding of bits 0 and 1 is as follows:

| Bit 1 | Bit 0 | Character Length |
|---|---|---|
| 0 | 0 | 5 Bits |
| 0 | 1 | 6 Bits |
| 1 | 0 | 7 Bits |
| 1 | 1 | 8 Bits |

**Bit 2:** This bit specifies the number of Stop bits transmitted and received in each serial character. If bit 2 is a logic 0, one Stop bit is generated in the transmitted data. If bit 2 is a logic 1 when a 5-bit word length is selected via bits 0 and 1, one and a half Stop bits are generated. If bit 2 is a logic 1 when either a 6-, 7-, or 8-bit word length is selected, two Stop bits are generated. The Receiver checks the first Stop-bit only, regardless of the number of Stop bits selected.

**Bit 3:** This bit is the Parity Enable bit. When bit 3 is a logic 1, a Parity bit is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. (The Parity bit is used to produce an even or odd number of 1s when the data word bits and the Parity bit are summed.)

**Bit 4:** This bit is the Even Parity Select bit. When bit 3 is a logic 1 and bit 4 is a logic 0, an odd number of logic 1s is transmitted or checked in the data word bits and Parity bit. When bit 3 is a logic 1 and bit 4 is a logic 1, an even number of logic 1s is transmitted or checked.

**Bit 5:** This bit is the Stick Parity bit. When bits 3, 4 and 5 are logic 1 the Parity bit is transmitted and checked as a logic 0. If bits 3 and 5 are 1 and bit 4 is a logic 0 then the Parity bit is transmitted and checked as a logic 1. If bit 5 is a logic 0 Stick Parity is disabled.

**Bit 6:** This bit is the Break Control bit. It causes a break condition to be transmitted to the receiving UART. When it is set to a logic 1, the serial output (SOUT) is forced to the Spacing (logic 0) state. The break is disabled by setting bit 6 to a logic 0. The Break Control bit acts only on SOUT and has no effect on the transmitter logic.

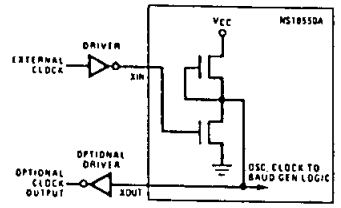
Note: This feature enables the CPU to alert a terminal in a computer communications system. If the following sequence is followed, no erroneous or extraneous characters will be transmitted because of the break.

1. Load an all 0s, pad character, in response to THRE.
2. Set break after the next THRE.
3. Wait for the transmitter to be idle, (TEMT = 1), and clear break when normal transmission has to be restored.

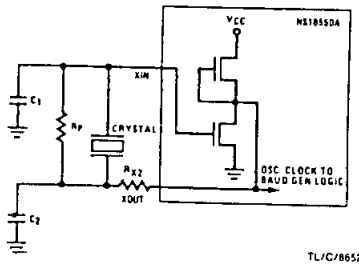
During the break, the Transmitter can be used as a character timer to accurately establish the break duration.

**Bit 7:** This bit is the Divisor Latch Access Bit (DLAB). It must be set high (logic 1) to access the Divisor Latches of the Baud Generator during a Read or Write operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmitter Holding Register, or the Interrupt Enable Register.

### 8.2 TYPICAL CLOCK CIRCUITS




TL/C/8652-19




TL/C/8652-20

Typical Crystal Oscillator Network

| CRYSTAL | $R_P$ | $R_{X2}$ | $C_1$ | $C_2$ |
|---|---|---|---|---|
| 3.1 MHz | 1 MΩ | 1.5k | 10-30 pF | 40-60 pF |
| 1.8 MHz | 1 MΩ | 1.5k | 10-30 pF | 40-60 pF |

TABLE III. Baud Rates Using 1.8432 MHz Crystal

| Desired Baud Rate | Decimal Divisor Used to Generate 16 x Clock | Percent Error Difference Between Desired and Actual |
|---|---|---|
| 50 | 2304 | — |
| 75 | 1536 | — |
| 110 | 1047 | 0.026 |
| 134.5 | 857 | 0.058 |
| 150 | 768 | — |
| 300 | 384 | — |
| 600 | 192 | — |
| 1200 | 96 | — |
| 1800 | 64 | — |
| 2000 | 58 | 0.69 |
| 2400 | 48 | — |
| 3600 | 32 | — |
| 4800 | 24 | — |
| 7200 | 16 | — |
| 9600 | 12 | — |
| 19200 | 6 | — |
| 38400 | 3 | — |
| 56000 | 2 | 2.86 |

## 8.0 Registers (Continued)

### 8.3 PROGRAMMABLE BAUD GENERATOR

The UART contains a programmable Baud Generator that is capable of taking any clock input from DC to 8.0 MHz and dividing it by any divisor from 2 to $2^{16}-1$. 4 MHz is the highest input clock frequency recommended when the divisor = 1. The output frequency of the Baud Generator is 16 × the Baud [divisor # = (frequency input) ÷ (baud rate × 16)]. Two 8-bit latches store the divisor in a 16-bit binary format. These Divisor Latches must be loaded during initialization to ensure proper operation of the Baud Generator. Upon loading either of the Divisor Latches, a 16-bit Baud counter is immediately loaded.

Tables III, IV and V provide decimal divisors to use with crystal frequencies of 1.8432 MHz, 3.072 MHz and 8 MHz, respectively. For baud rates of 38400 and below, the error obtained is minimal. The accuracy of the desired baud rate is dependent on the crystal frequency chosen. Using a divisor of zero is not recommended.

### 8.4 LINE STATUS REGISTER

This register provides status information to the CPU concerning the data transfer. Table II shows the contents of the Line Status Register. Details on each bit follow.

**Bit 0:** This bit is the receiver Data Ready (DR) indicator. Bit 0 is set to a logic 1 whenever a complete incoming character has been received and transferred into the Receiver Buffer Register or the FIFO. Bit 0 is reset to a logic 0 by reading all of the data in the Receiver Buffer Register or the FIFO.

**Bit 1:** This bit is the Overrun Error (OE) indicator. Bit 1 indicates that data in the Receiver Buffer Register was not read by the CPU before the next character was transferred into the Receiver Buffer Register, thereby destroying the previous character. The OE indicator is set to a logic 1 upon detection of an overrun condition and reset whenever the CPU reads the contents of the Line Status Register. If the FIFO mode data continues to fill the FIFO beyond the trigger level, an overrun error will occur only after the FIFO is full and the next character has been completely received in the shift register. OE is indicated to the CPU as soon as it happens. The character in the shift register is overwritten, but it is not transferred to the FIFO.

TABLE IV. Baud Rates Using 3.072 MHz Crystal

| Desired Baud Rate | Decimal Divisor Used to Generate 16 x Clock | Percent Error Difference Between Desired and Actual |
|---|---|---|
| 50 | 3840 | — |
| 75 | 2560 | — |
| 110 | 1745 | 0.026 |
| 134.5 | 1428 | 0.034 |
| 150 | 1280 | — |
| 300 | 640 | — |
| 600 | 320 | — |
| 1200 | 160 | — |
| 1800 | 107 | 0.312 |
| 2000 | 96 | — |
| 2400 | 80 | — |
| 3600 | 53 | 0.628 |
| 4800 | 40 | — |
| 7200 | 27 | 1.23 |
| 9600 | 20 | — |
| 19200 | 10 | — |
| 38400 | 5 | — |

**Bit 2:** This bit is the Parity Error (PE) indicator. Bit 2 indicates that the received data character does not have the correct even or odd parity, as selected by the even-parity-select bit. The PE bit is set to a logic 1 upon detection of a parity error and is reset to a logic 0 whenever the CPU reads the contents of the Line Status Register. In the FIFO mode this error is associated with the particular character in the FIFO it applies to. This error is revealed to the CPU when its associated character is at the top of the FIFO.

**Bit 3:** This bit is the Framing Error (FE) indicator. Bit 3 indicates that the received character did not have a valid Stop bit. Bit 3 is set to a logic 1 whenever the Stop bit following the last data bit or parity bit is detected as a logic 0 bit (Spacing level). The FE indicator is reset whenever the CPU reads the contents of the Line Status Register. In the FIFO mode this error is associated with the particular character in the FIFO it applies to. This error is revealed to the CPU when its associated character is at the top of the FIFO. The UART will try to resynchronize after a framing error. To do this it assumes that the framing error was due to the next start bit, so it samples this "start" bit twice and then takes in the "data".

**Bit 4:** This bit is the Break Interrupt ($\overline{BI}$) indicator. Bit 4 is set to a logic 1 whenever the received data input is held in the Spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + Parity + Stop bits). The $\overline{BI}$ indicator is reset whenever the CPU reads the contents of the Line Status Register. In the FIFO mode this error is associated with the particular character in the FIFO it applies to. This error is revealed to the CPU when its associated character is at the top of the FIFO. When break occurs only one zero character is loaded into the FIFO. The next character transfer is enabled after SIN goes to the marking state and receives the next valid start bit.

**Note:** Bits 1 through 4 are the error conditions that produce a Receiver Line Status interrupt whenever any of the corresponding conditions are detected and the interrupt is enabled.

TABLE V. Baud Rates Using 8 MHz Crystal

| Desired Baud Rate | Decimal Divisor Used to Generate 16 x Clock | Percent Error Difference Between Desired and Actual |
|---|---|---|
| 50 | 10000 | — |
| 75 | 6667 | 0.005 |
| 110 | 4545 | 0.010 |
| 134.5 | 3717 | 0.013 |
| 150 | 3333 | 0.010 |
| 300 | 1667 | 0.020 |
| 600 | 833 | 0.040 |
| 1200 | 417 | 0.080 |
| 1800 | 277 | 0.080 |
| 2000 | 250 | — |
| 2400 | 208 | 0.160 |
| 3600 | 139 | 0.080 |
| 4800 | 104 | 0.160 |
| 7200 | 69 | 0.644 |
| 9600 | 52 | 0.160 |
| 19200 | 26 | 0.160 |
| 38400 | 13 | 0.160 |
| 56000 | 9 | 0.790 |
| 128000 | 4 | 2.344 |
| 256000 | 2 | 2.344 |

## 8.0 Registers (Continued)

TABLE VI. Interrupt Control Functions

| FIFO Mode Only | Interrupt Identification Register | | | Interrupt Set and Reset Functions | | | |
|---|---|---|---|---|---|---|---|
| Bit 3 | Bit 2 | Bit 1 | Bit 0 | Priority Level | Interrupt Type | Interrupt Source | Interrupt Reset Control |
| 0 | 0 | 0 | 1 | — | None | None | — |
| 0 | 1 | 1 | 0 | Highest | Receiver Line Status | Overrun Error or Parity Error or Framing Error or Break Interrupt | Reading the Line Status Register |
| 0 | 1 | 0 | 0 | Second | Received Data Available | Receiver Data Available or Trigger Level Reached | Reading the Receiver Buffer Register or the FIFO Drops Below the Trigger Level |
| 1 | 1 | 0 | 0 | Second | Character Timeout Indication | No Characters Have Been Removed From or Input to the RCVR FIFO During the Last 4 Char. Times and There Is at Least 1 Char. in It During This Time | Reading the Receiver Buffer Register |
| 0 | 0 | 1 | 0 | Third | Transmitter Holding Register Empty | Transmitter Holding Register Empty | Reading the IIR Register (if source of interrupt) or Writing into the Transmitter Holding Register |
| 0 | 0 | 0 | 0 | Fourth | MODEM Status | Clear to Send or Data Set Ready or Ring Indicator or Data Carrier Detect | Reading the MODEM Status Register |

**Bit 5:** This bit is the Transmitter Holding Register Empty (THRE) indicator. Bit 5 indicates that the UART is ready to accept a new character for transmission. In addition, this bit causes the UART to issue an interrupt to the CPU when the Transmit Holding Register Empty Interrupt enable is set high. The THRE bit is set to a logic 1 when a character is transferred from the Transmitter Holding Register into the Transmitter Shift Register. The bit is reset to logic 0 concurrently with the loading of the Transmitter Holding Register by the CPU. In the FIFO mode this bit is set when the XMIT FIFO is empty; it is cleared when at least 1 byte is written to the XMIT FIFO.

**Bit 6:** This bit is the Transmitter Empty (TEMT) indicator. Bit 6 is set to a logic 1 whenever the Transmitter Holding Register (THR) and the Transmitter Shift Register (TSR) are both empty. It is reset to a logic 0 whenever either the THR or TSR contains a data character. In the FIFO mode this bit is set to one whenever the transmitter FIFO and shift register are both empty.

**Bit 7:** In the NS16450 Mode this is a 0. In the FIFO mode LSR7 is set when there is at least one parity error, framing error or break indication in the FIFO. LSR7 is cleared when the CPU reads the LSR, if there are no subsequent errors in the FIFO.

*Note:* The Line Status Register is intended for read operations only. Writing to this register is not recommended as this operation is only used for factory testing.

### 8.5 FIFO CONTROL REGISTER

This is a write only register at the same location as the IIR (the IIR is a read only register). This register is used to enable the FIFOs, clear the FIFOs, set the RCVR FIFO trigger level, and select the type of DMA signalling.

**Bit 0:** Writing a 1 to FCR0 enables both the XMIT and RCVR FIFOs. Resetting FCR0 will clear all bytes in both FIFOs. When changing from FIFO Mode to NS16450 Mode and vice versa, data is automatically cleared from the FIFOs. This bit must be a 1 when other FCR bits are written to or they will not be programmed.

**Bit 1:** Writing a 1 to FCR1 clears all bytes in the RCVR FIFO and resets its counter logic to 0. The shift register is not cleared. The 1 that is written to this bit position is self-clearing.

**Bit 2:** Writing a 1 to FCR2 clears all bytes in the XMIT FIFO and resets its counter logic to 0. The shift register is not cleared. The 1 that is written to this bit position is self-clearing.

**Bit 3:** Setting FCR3 to a 1 will cause the RXRDY and TXRDY pins to change from mode 0 to mode 1 if FCR0 = 1 (see description of RXRDY and TXRDY pins).

**Bit 4, 5:** FCR4 to FCR5 are reserved for future use.

**Bit 6, 7:** FCR6 and FCR7 are used to set the trigger level for the RCVR FIFO interrupt.

| 7 | 6 | RCVR FIFO Trigger Level (Bytes) |
|---|---|---|
| 0 | 0 | 01 |
| 0 | 1 | 04 |
| 1 | 0 | 08 |
| 1 | 1 | 14 |

### 8.6 INTERRUPT IDENTIFICATION REGISTER

In order to provide minimum software overhead during data character transfers, the UART prioritizes interrupts into four levels and records these in the interrupt Identification Register. The four levels of interrupt conditions in order of priority are Receiver Line Status; Received Data Ready; Transmitter Holding Register Empty; and MODEM Status.

## 8.0 Registers (Continued)

When the CPU accesses the IIR, the UART freezes all interrupts and indicates the highest priority pending interrupt to the CPU. While this CPU access is occurring, the UART records new interrupts, but does not change its current indication until the access is complete. Table II shows the contents of the IIR. Details on each bit follow:

**Bit 0:** This bit can be used in a prioritized interrupt environment to indicate whether an interrupt is pending. When bit 0 is a logic 0, an interrupt is pending and the IIR contents may be used as a pointer to the appropriate interrupt service routine. When bit 0 is a logic 1, no interrupt is pending.

**Bits 1 and 2:** These two bits of the IIR are used to identify the highest priority interrupt pending as indicated in Table VI.

**Bit 3:** In the NS16450 Mode this bit is 0. In the FIFO mode this bit is set along with bit 2 when a timeout interrupt is pending.

**Bits 4 and 5:** These two bits of the IIR are always logic 0.

**Bits 6 and 7:** These two bits are set when FCR0 = 1.

### 8.7 INTERRUPT ENABLE REGISTER

This register enables the five types of UART interrupts. Each interrupt can individually activate the interrupt (INTR) output signal. It is possible to totally disable the interrupt system by resetting bits 0 through 3 of the Interrupt Enable Register (IER). Similarly, setting bits of the IER register to a logic 1, enables the selected interrupt(s). Disabling an interrupt prevents it from being indicated as active in the IIR and from activating the INTR output signal. All other system functions operate in their normal manner, including the setting of the Line Status and MODEM Status Registers. Table II shows the contents of the IER. Details on each bit follow.

**Bit 0:** This bit enables the Received Data Available Interrupt (and timeout interrupts in the FIFO mode) when set to logic 1.

**Bit 1:** This bit enables the Transmitter Holding Register Empty Interrupt when set to logic 1.

**Bit 2:** This bit enables the Receiver Line Status Interrupt when set to logic 1.

**Bit 3:** This bit enables the MODEM Status Interrupt when set to logic 1.

**Bits 4 through 7:** These four bits are always logic 0.

### 8.8 MODEM CONTROL REGISTER

This register controls the interface with the MODEM or data set (or a peripheral device emulating a MODEM). The contents of the MODEM Control Register are indicated in Table II and are described below.

**Bit 0:** This bit controls the Data Terminal Ready ($\overline{DTR}$) output. When bit 0 is set to a logic 1, the $\overline{DTR}$ output is forced to a logic 0. When bit 0 is reset to a logic 0, the $\overline{DTR}$ output is forced to a logic 1.

**Note:** The $\overline{DTR}$ output of the UART may be applied to an EIA inverting line driver (such as the DS1488) to obtain the proper polarity input at the succeeding MODEM or data set.

**Bit 1:** This bit controls the Request to Send ($\overline{RTS}$) output. Bit 1 affects the $\overline{RTS}$ output in a manner identical to that described above for bit 0.

**Bit 2:** This bit controls the Output 1 ($\overline{OUT\ 1}$) signal, which is an auxiliary user-designated output. Bit 2 affects the $\overline{OUT\ 1}$ output in a manner identical to that described above for bit 0.

**Bit 3:** This bit controls the Output 2 ($\overline{OUT\ 2}$) signal, which is an auxiliary user-designated output. Bit 3 affects the $\overline{OUT\ 2}$ output in a manner identical to that described above for bit 0.

**Bit 4:** This bit provides a local loopback feature for diagnostic testing of the UART. When bit 4 is set to logic 1, the following occur: the transmitter Serial Output (SOUT) is set to the Marking (logic 1) state; the receiver Serial Input (SIN) is disconnected; the output of the Transmitter Shift Register is "looped back" into the Receiver Shift Register Input; the four MODEM Control inputs ($\overline{CTS}$, $\overline{DSR}$, $\overline{RI}$, and $\overline{DCD}$) are disconnected; and the four MODEM Control outputs ($\overline{DTR}$, $\overline{RTS}$, $\overline{OUT\ 1}$, and $\overline{OUT\ 2}$) are internally connected to the four MODEM Control inputs, and the MODEM Control output pins are forced to their inactive state (high). In the diagnostic mode, data that is transmitted is immediately received. This feature allows the processor to verify the transmit-and received-data paths of the UART.

In the diagnostic mode, the receiver and transmitter interrupts are fully operational. Their sources are external to the part. The MODEM Control Interrupts are also operational, but the interrupts' sources are now the lower four bits of the MODEM Control Register instead of the four MODEM Control inputs. The interrupts are still controlled by the Interrupt Enable Register.

**Bits 5 through 7:** These bits are permanently set to logic 0.

### 8.9 MODEM STATUS REGISTER

This register provides the current state of the control lines from the MODEM (or peripheral device) to the CPU. In addition to this current-state information, four bits of the MODEM Status Register provide change information. These bits are set to a logic 1 whenever a control input from the MODEM changes state. They are reset to logic 0 whenever the CPU reads the MODEM Status Register.

The contents of the MODEM Status Register are indicated in Table II and described below.

**Bit 0:** This bit is the Delta Clear to Send (DCTS) indicator. Bit 0 indicates that the $\overline{CTS}$ input to the chip has changed state since the last time it was read by the CPU.

**Bit 1:** This bit is the Delta Data Set Ready (DDSR) indicator. Bit 1 indicates that the $\overline{DSR}$ input to the chip has changed state since the last time it was read by the CPU.

**Bit 2:** This bit is the Trailing Edge of Ring Indicator (TERI) detector. Bit 2 indicates that the $\overline{RI}$ input to the chip has changed from a low to a high state.

**Bit 3:** This bit is the Delta Data Carrier Detect (DDCD) indicator. Bit 3 indicates that the $\overline{DCD}$ input to the chip has changed state.

**Note:** Whenever bit 0, 1, 2, or 3 is set to logic 1, a MODEM Status Interrupt is generated.

**Bit 4:** This bit is the complement of the Clear to Send ($\overline{CTS}$) input. If bit 4 (loop) of the MCR is set to a 1, this bit is equivalent to RTS in the MCR.

**Bit 5:** This bit is the complement of the Data Set Ready ($\overline{DSR}$) input. If bit 4 of the MCR is set to a 1, this bit is equivalent to DTR in the MCR.

## 8.0 Registers (Continued)

**Bit 6:** This bit is the complement of the Ring Indicator ($\overline{RI}$) input. If bit 4 of the MCR is set to a 1, this bit is equivalent to OUT 1 in the MCR.

**Bit 7:** This bit is the complement of the Data Carrier Detect ($\overline{DCD}$) input. If bit 4 of the MCR is set to a 1, this bit is equivalent to OUT 2 in the MCR.

### 8.10 SCRATCHPAD REGISTER

This 8-bit Read/Write Register does not control the UART in anyway. It is intended as a scratchpad register to be used by the programmer to hold data temporarily.

### 8.11 FIFO INTERRUPT MODE OPERATION

When the RCVR FIFO and receiver interrupts are enabled (FCR0 = 1, IER0 = 1) RCVR interrupts will occur as follows:

A. The receive data available interrupt will be issued to the CPU when the FIFO has reached its programmed trigger level; it will be cleared as soon as the FIFO drops below its programmed trigger level.

B. The IIR receive data available indication also occurs when the FIFO trigger level is reached, and like the interrupt it is cleared when the FIFO drops below the trigger level.

C. The receiver line status interrupt (IIR = 06), as before, has higher priority than the received data available (IIR = 04) interrupt.

D. The data ready bit (LSR0) is set as soon as a character is transferred from the shift register to the RCVR FIFO. It is reset when the FIFO is empty.

When RCVR FIFO and receiver interrupts are enabled, RCVR FIFO timeout interrupts will occur as follows:

A. A FIFO timeout interrupt will occur, if the following conditions exist:

- at least one character is in the FIFO
- the most recent serial character received was longer than 4 continuous character times ago (if 2 stop bits are programmed the second one is included in this time delay).
- the most recent CPU read of the FIFO was longer than 4 continuous character times ago.

This will cause a maximum character received to interrupt issued delay of 160 ms at 300 BAUD with a 12 bit character.

B. Character times are calculated by using the RCLK input for a clock signal (this makes the delay proportional to the baudrate).

C. When a timeout interrupt has occurred it is cleared and the timer reset when the CPU reads one character from the RCVR FIFO.

D. When a timeout interrupt has not occurred the timeout timer is reset after a new character is received or after the CPU reads the RCVR FIFO.

When the XMIT FIFO and transmitter interrupts are enabled (FCR0 = 1, IER1 = 1), XMIT interrupts will occur as follows:

A. The transmitter holding register interrupt (02) occurs when the XMIT FIFO is empty; it is cleared as soon as the transmitter holding register is written to (1 to 16 characters may be written to the XMIT FIFO while servicing this interrupt) or the IIR is read.

B. The transmitter FIFO empty indications will be delayed 1 character time minus the last stop bit time whenever the following occurs: THRE = 1 and there have not been at least two bytes at the same time in the transmit FIFO, since the last THRE = 1. The first transmitter interrupt after changing FCR0 will be immediate, if it is enabled.

Character timeout and RCVR FIFO trigger level interrupts have the same priority as the current received data available interrupt; XMIT FIFO empty has the same priority as the current transmitter holding register empty interrupt.

### 8.12 FIFO POLLED MODE OPERATION

With FCR0 = 1 resetting IER0, IER1, IER2, IER3 or all to zero puts the UART in the FIFO Polled Mode of operation. Since the RCVR and XMITTER are controlled separately either one or both can be in the polled mode of operation.

In this mode the user's program will check RCVR and XMITTER status via the LSR. As stated previously:

LSR0 will be set as long as there is one byte in the RCVR FIFO.

LSR1 to LSR4 will specify which error(s) has occurred. Character error status is handled the same way as when in the interrupt mode, the IIR is not affected since IER2 = 0.

LSR5 will indicate when the XMIT FIFO is empty.

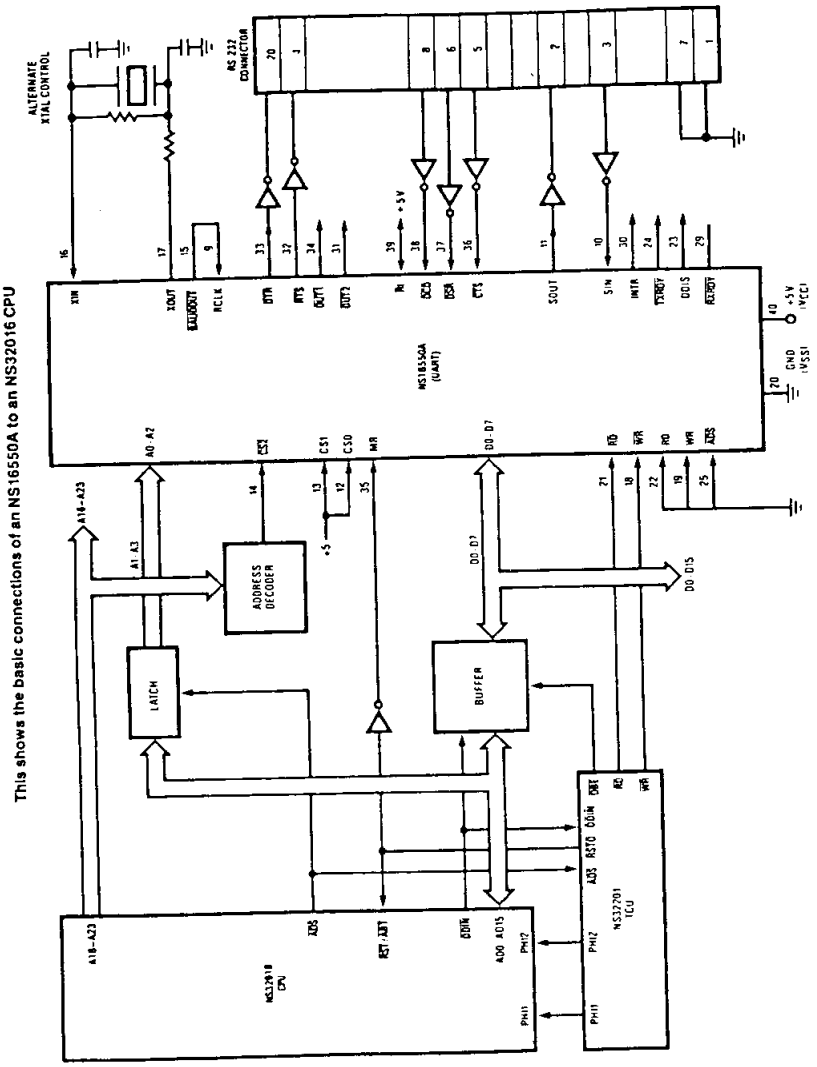
LSR6 will indicate that both the XMIT FIFO and shift register are empty.

LSR7 will indicate whether there are any errors in the RCVR FIFO.

There is no trigger level reached or timeout condition indicated in the FIFO Polled Mode, however, the RCVR and XMIT FIFOs are still fully capable of holding characters.

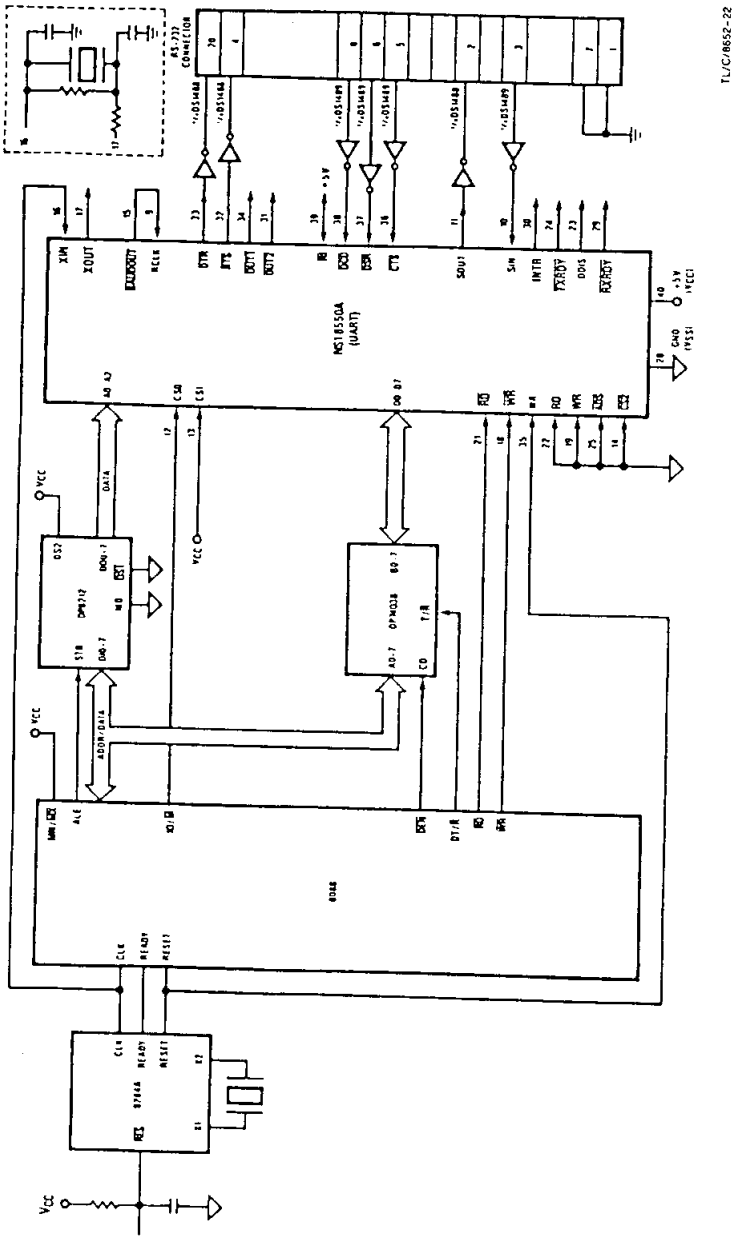
## 9.0 Typical Applications

TL/C/6652-21




This shows the basic connections of an NS16550A to an NS32016 CPU

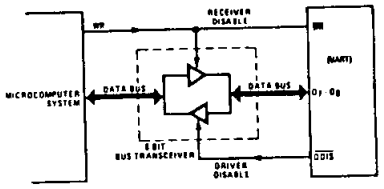
## 9.0 Typical Applications (Continued)




This shows the basic connections of an NS16550A to an 8088 CPU

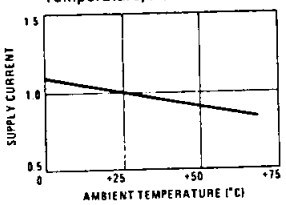
## 9.0 Typical Applications (Continued)

**Typical Interface for a High-Capacity Data Bus**



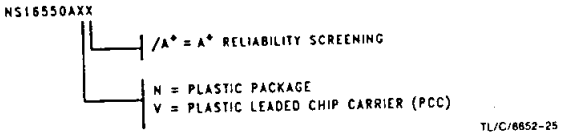

TL/C/8652-23

**Typical Supply Current vs. Temperature, Normalized**




TL/C/8652-24

## 10.0 Ordering Information




TL/C/8652-25

## 11.0 Reliability Information

Gate Count 3,400
Transistor Count 10,300

# The NS16550A: UART Design and Application Considerations

## BACKGROUND

UARTs like other system components have evolved for many years to become faster, more integrated and less expensive. The rise in popularity of the personal computer with its focus and competition primarily centered on an architecture introduced by IBM®, has driven both UART performance and software compatibility issues. As transmission rates have increased, the amount of time the CPU has for other tasks while handling an active serial channel has been sharply reduced. One byte of data received at 1200 baud (8.3 ms) is received in $\frac{1}{16}$th the time at 19.2 kbaud (520 μs). Software compatibility among the PC-based UARTs is critical due to the thousands of existing programs which use the serial channel and the new programs continually being offered.

Higher baud rates and compatibility requirements influence new UART designs. These two constraints result in UARTs that are capable of higher data rates, increasingly independent of CPU intervention and providing more autonomous features, while maintaining software compatibility. These development paths have been brought together in a new UART from National Semiconductor designated the NS16550A.

The NS16550A has all of the registers of its two predecessor parts (INS8250 and NS16450), so it can run all existing IBM PC, XT, AT, RT and compatible serial port software. In addition, it has a programmable mode which incorporates new high-performance features. Of course, all of these advanced features are useful in any asynchronous serial communications application regardless of the host architecture.

The reader is assumed to be familiar with the standard features of the NS16450, so this paper will concentrate mainly on the new features of the NS16550A. If the reader is unfamiliar with these UARTs it is advisable to start by reading their data sheets.

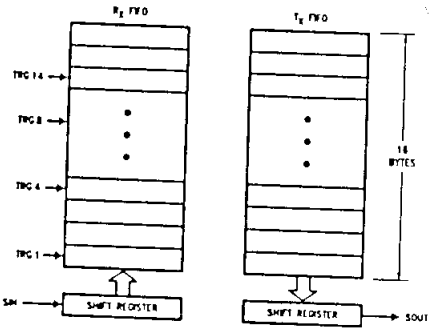
The first section reviews some of the design considerations and the operation of the NS16550A advanced features. The second section shows an NS16550A initialization routine written in 80286 assembly code with an explanation of the routine. The third section gives a detailed example of communications drivers written to interface two NS16550As on individual boards. These drivers are written for use with National Semiconductor's DB32032 evaluation boards, but can be ported to any NS32032-based system containing an NS32202 (ICU).

## 1.0 Design Considerations and Operation of the New UART Features

In order to optimize CPU/UART data transactions, the UART design takes into consideration the following constraints:

1. The CPU is usually much faster than the UART at transferring data. A high speed CPU could transfer a byte of data to/from the UART in a minimum of 280 ns. The UART would take over 1800 times longer to transmit/receive this data serially if it were operating at 19.2 kbaud.
2. There is a finite amount of wasted CPU time due to software overhead when stopping its current task to service the UART (context switching overhead).
3. The CPU may be required to complete a certain portion of its current task in a multitasking system before servicing the UART. This delay is the CPU latency time associated with servicing the interrupt. The amount of time that the receiver can accept continuous data after it requests service from the CPU constrains CPU latency time.

The design constraints listed above are met by adding two FIFOs and specialized transmitter/receiver support circuitry to the existing NS16450 design. The FIFOs are 16 bytes deep—one holds data for the transmitter, the other for the receiver (see *Figure 1*). Similarity between the FIFOs stops with their size, as each has been customized for special




TL/C/9313–1

**FIGURE 1. Rx and Tx FIFOs**

transmitter or receiver functions. Each has support circuitry to minimize software overhead when handling interrupts. The NS16550A receiver optimizes the CPU/UART data transaction via the following features:

1. The depth of the Receiver (Rx) FIFO ensures that as many as 16 characters will be ready to transfer when the CPU services the Rx interrupt. Therefore, the CPU transfer rate is effectively buffered from the serial data rate.
2. The program can select the number of bytes required in the Rx FIFO (1, 4, 8 or 14) before the UART issues an interrupt. This allows the software to modify the interrupt trigger levels depending on its current task or loading. It also ensures that the CPU doesn't continually waste time switching context for only a few characters.

3. The Rx FIFO will hold 16 bytes regardless of which trigger level the CPU selects. This makes allowances for a variety of CPU latency times, as the FIFO continues to fill after the interrupt is issued.

The NS16550A transmitter optimizes the CPU/UART data transaction via the following features:

1. The depth of the Transmitter (Tx) FIFO ensures that as many as 16 characters can be transferred when the CPU services the Tx interrupt. Once again, this effectively buffers the CPU transfer rate from the serial data rate.
2. The Transmitter (Tx) FIFO is similar in structure to FIFOs the user may have previously set up in RAM. The Tx depth allows the CPU to load 16 characters each time it switches context to the service routine. This reduces the impact of the CPU time lost in context switching.
3. Since a time lag in servicing an asynchronous transmitter usually has no penalty, CPU latency time is of no concern to transmitter operation.

### TX AND RX FIFO OPERATION

The Tx portion of the UART transmits data through SOUT as soon as the CPU loads a byte into the Tx FIFO. The UART will prevent loads to the Tx FIFO if it currently holds 16 characters. Loading to the Tx FIFO will again be enabled as soon as the next character is transferred to the Tx shift register. These capabilities account for the largely autonomous operation of the Tx.

The UART starts the above operations typically with a Tx interrupt. The NS16550A issues a Tx interrupt whenever the Tx FIFO is empty and the Tx interrupt is enabled, except in the following instance. Assume that the Tx FIFO is empty and the CPU starts to load it. When the first byte enters the FIFO, the Tx FIFO empty interrupt will transition from active to inactive. Depending on the execution speed of the service routine software, the UART may be able to transfer this byte from the FIFO to the shift register before the CPU loads another byte. If this happens, the Tx FIFO will be empty again and typically the UART's interrupt line would transition to the active state. This could cause a system with an interrupt control unit to record a Tx FIFO empty condition, even though the CPU is currently servicing that interrupt. Therefore, after the first byte has been loaded into the FIFO the UART will wait one serial character transmission time before issuing a new Tx FIFO empty interrupt.

This one character Tx interrupt delay will remain active until at least two bytes have been loaded into the FIFO, concurrently. When the Tx FIFO empties after this condition, the Tx interrupt will be activated without a one character delay.

Rx support functions and operation are quite different from those described for the transmitter. The Rx FIFO receives data until the number of bytes in the FIFO equals the selected interrupt trigger level. At that time if Rx interrupts are enabled, the UART will issue an interrupt to the CPU. The Rx FIFO will continue to store bytes until it holds 16 of them. It will not accept any more data when it is full. Any more data entering the Rx shift register will set the Overrun Error flag. Normally, the FIFO depth and the programmable trigger levels will give the CPU ample time to empty the Rx FIFO before an overrun occurs.

One side-effect of having a Rx FIFO is that the selected interrupt trigger level may be above the data level in the FIFO. This could occur when data at the end of the block contains fewer bytes than the trigger level. No interrupt would be issued to the CPU and the data would remain in the UART. To prevent the software from having to check for this situation the NS16550A incorporates a timeout interrupt.

The timeout interrupt is activated when there is at least one byte in the Rx FIFO, and neither the CPU nor the Rx shift register has accessed the Rx FIFO within 4 character times of the last byte. The timeout interrupt is cleared or reset when the CPU reads the Rx FIFO or another character enters it.

These FIFO related features allow optimization of CPU/UART transactions and are especially useful given the higher baud rate capability (256 kbaud). However, in order to eliminate most CPU interactions, the UART provides DMA request signals. Two DMA modes are supported: single-transfer and multi-transfer. These modes allow the UART to interface to higher performance DMA units, which can interleave their transfers between CPU cycles or execute multiple byte transfers.

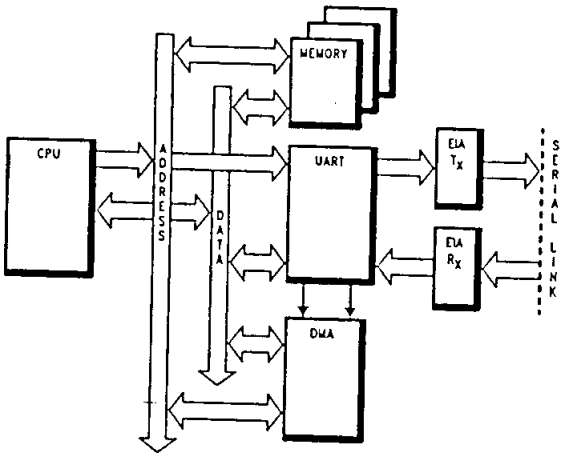
In single-transfer mode the receiver DMA request signal (Rx RDY) goes active whenever there is at least one character in the Rx FIFO. It goes inactive when the Rx FIFO is empty. The transmitter DMA request signal (Tx RDY) goes active when there are no characters in the Tx FIFO. It goes inactive when there is at least one character in the Tx FIFO. Therefore, in single-transfer mode active and inactive DMA signals are issued on a one byte basis.

In multi-transfer mode Rx RDY goes active whenever the trigger level or the timeout has been reached. It goes inactive when the Rx FIFO is empty. Tx RDY goes active when there is at least one unfilled position in the Tx FIFO. It goes inactive when the Tx FIFO is completely full. Therefore in multi-transfer mode active and inactive DMA signals are issued as the FIFO fills and empties. With 2 DMA channels (one for each Rx and Tx) assigned to it, the NS16550A could run somewhat independently of the CPU when the DMA unit transfers data composed of blocks with checksums.

### SYSTEM OPERATION: THE NS16550A VS THE NS16450

Consider the typical system interface block diagram in *Figure 2*. This is a simple diagram, but it includes all of the components that typically interact with a UART. The advantages of the NS16550A over the NS16450 can be illustrated by comparing some of the system constraints when each UART is substituted into this basic system.

Both RS-232C and RS-422A interfaces can be used with either UART, however, the NS16550A can drive these interfaces up to 256 kbaud. Regarding the RS-422A specifica-




TL/C/9313-2

FIGURE 2. Typical System Interface

tion (max. 10 Mbaud) this is significantly faster than the NS16450 (max. 56 kbaud).

The NS16450 has no DMA request signals, so the DMA unit would not interact with the NS16450. The NS16550A, however, has DMA request signals and two modes of data transfer, as previously described, to interface with a variety of DMA units.

The greatest advantages of the NS16550A over the NS16450 are seen when considering the CPU/UART interface. Some characteristics of the transactions occurring between the CPU and the UART were previously cited. However, optimizing these transactions involves two issues:

1. Decreasing the amount of time the CPU interacts with the UART.
2. Increasing the amount of data transferred between the CPU and UART during their interaction time.

These optimization criteria are directly opposed to each other, but various features on the NS16550A have improved both.

One of the more obvious ways to decrease the CPU/UART interaction time is to decrease the time it takes for the transaction to occur. The NS16550A has an access cycle time that is almost 25% shorter than the NS16450. In addition, other timing parameters were made faster to simplify high speed CPU interactions.

The actual software required to transfer the data between the CPU and the UART is a small percentage of that required to support this transfer. However, each time a transfer occurs in the NS16450, this support software (overhead) must also be executed. With the NS16550A each time the UART needs service the CPU can theoretically transfer 16 bytes while only running through its overhead once. Tests have shown that this will increase the performance by a factor of 5 at the system level over the NS16450.

Another time savings for the CPU is a new feature of the UART interrupt structure. Unlike most other UARTs with Rx FIFOs, the NS16550A will issue an interrupt when there are characters below the interrupt trigger level after a preset time delay. This saves the extra time spent by the CPU to check for bytes that are at the end of a block, but won't reach the interrupt level.

Since the NS16550A register set is identical to the NS16450 on power-up, all existing NS16450 software will run on it. The FIFOs are only enabled under program control.

All of this added performance is not without some tradeoffs. Two of the NS16450 pins, no connect (NC) and chip select out (CSOUT) have been replaced by the RxRDY and TxRDY pins. Most serial cards that currently use the NS16450 don't use these pins, so in those situations the NS16550A could be used as a plug-in upgrade. The software drivers for the NS16550A operating in FIFO mode need to be a little more sophisticated than for the NS16450. This will not cause a great penalty in CPU operating time as there is only one additional UART register to program and one to check during the initialization. One additional service routine is required to handle Rx timeout interrupts. This routine does not execute, except during intermittent transmissions or as described above.

All of these speed improvements and allowances for software constraints will make the NS16550A an optimal UART for both multi-tasking systems and multiport systems. Multi-tasking systems benefit from the increased time and flexibility offered to the CPU during context switching. Multiport systems, such as terminal concentrators, benefit from the on-board FIFOs and relatively autonomous functions of the UART.

## SYSTEM INTERRUPT GENERATION

As a prelude to the topic of the next section (80286™-based system initialization) a review of a typical PC hardware interrupt path is given. This concerns only the interrupt path between the UART and the CPU (see *Figure 3*).

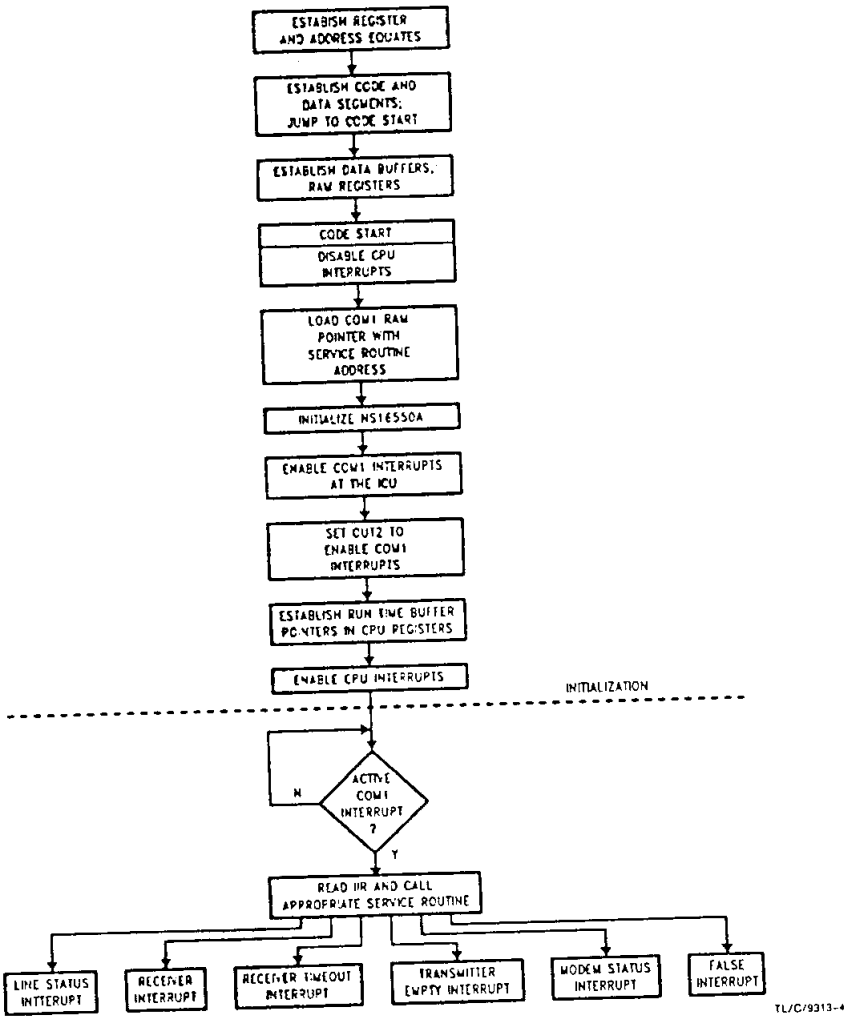
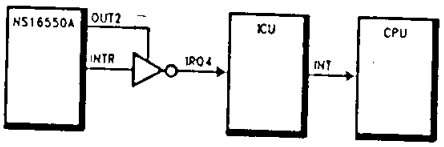



FIGURE 4. NS16550A Initialization and Driver Flowchart




TL/C/9313-3

FIGURE 3. Typical PC Interrupt System Hardware

In order to enable interrupts from the UART to the CPU each hardware device must be correctly initialized. While initializing the hardware path, CPU interrupts are turned off to avoid false interrupts from this path. This initialization should be as short as possible to avoid other devices "stacking up" interrupts during this time.

After the NS16550A is initialized the bits 0–3 in the Interrupt Enable Register (IER) are set enabling all UART interrupts. Also, bit 3 in the Modem Control Register (MCR) is set to enable the buffer between the UART and the ICU.

The ICU has bit 4 of its Interrupt Mask Register (IMR) cleared, allowing interrupts occuring on IRQ4 to be transferred to the CPU via the group interrupt (INT). Finally, CPU interrupts are enabled again via the STI instruction.

The programmer should be aware that the ICU will be initialized for edge-triggered interrupts and that the UART always produces level active interrupts. This allows the system to get into a situation where the UART has multiple interrupts pending (signaled via a constantly high INTR), but the ICU fails to respond because it expects an edge for each pending interrupt. To avoid this situation, the programmer should disable all UART interrupts via the IER when entering each UART interrupt service routine and then reenable all UART interrupts that are to be used just before exiting each interrupt service routine.

### SUMMARY

Up to this point the features of the NS16550A have been described, some of the design goals that resulted in these features have been reviewed, and a comparison has been given between it and the NS16450. Increases in bus speed and specialized functions make this part both faster from the hardware point of view and more efficient from the software point of view.

## 2.0 NS16550A Initialization

This initialization can be used on any 80286-based system; it enables both FIFOs and all interrupts on the UART. Additional procedures would have to be written to actually transfer data and service interrupts. These procedures would be similar in form to the 32000-based example in the next section, but the code would be different. The general flow of the initialization is shown in *Figure 4* and described below.

### DETAILED SOFTWARE DESCRIPTION

The first block in the initialization establishes abbreviations for the NS16550A registers and assigns addresses to them. The next three blocks establish code and data segments for the 80286. After jumping to the code start, the program disables CPU interrupts (CLI) until it has finished the initialization routine. Other interrupts may be active while CPU interrupts are masked, so the section of code following CLI should be as short as possible. The next block replaces the existing COM1 interrupt vector with the address of NS16550A interrupt handler (INTH in this case).

Initialization of the NS16550A is similar to the NS16450, except that there is one additional register to program which controls the FIFOs (Refer to the datasheet for a complete description). The sequence shown here sets bit 7 (DLAB) of the line control register (LCR), which enables access to the baud rate generator divisor. The divisor programmed is 0006 (19.2 kbaud) in this example. Programming the LCR again resets bit 7 (allowing access to the operational registers) and programs each frame for 7 data bits, one stop bit and even parity. The additional register that needs to be programmed in the NS16550A is the FIFO control register (FCR). The FCR data is 1100 0001. Bits 6 and 7 set the Rx FIFO interrupt trigger level at 14 characters. Bits 5 and 4 are reserved. Bit 3 keeps the DMA signal lines in mode 0. Setting bits 2 and 1 clear the Tx and Rx FIFOs, but this is done automatically when the FIFOs are first enabled by setting bit 0. Bit 0 of the FCR should ALWAYS BE SET whenever changes are to be made to the other bits of the FCR and the UART is to remain in FIFO Mode. When the FIFOs on the NS16550A are enabled bits 6 and 7 in the Interrupt Identification Register are set. Thus the program can distinguish between an NS16450 and an NS16550A, taking advantage of the FIFOs.

Sending a 0F to the Interrupt Enable Register enables all UART interrupts. The next two register accesses, reading the Line Status Register and the Modem Status Register, are optional. They are conservatively included in this initialization in order to defeat false interrupt indications in these registers caused by noise on the external lines.

The next block of code enables the interrupt signal to go beyond the UART through the system hardware. In many popular 80286-based personal computers, an interrupt control unit (ICU) has its mask register at I/O address 21H. To enable interrupts through this ICU for COM1 without disturbing other interrupts, the Interrupt Mask Register (IMR) is read. This data is combined with 1110 1111 via an AND instruction to unmask the COM1 interrupt and then loaded it back to the IMR. On these personal computers there is also a buffer on the interrupt line between the UART and ICU. This buffer is enabled by setting the OUT2 bit of the MODEM Control Register in the UART.

Before enabling CPU interrupts (STI) pointer registers to the data buffers of each service routine are loaded. After enabling CPU interrupts this program jumps to a holding loop to wait for an interrupt, whereas most programs would continue initializing other devices or jump to the system loop.

```
        TITLE   550APP.ASM - NS16550A INITIALIZATION
;
;ESTABLISH NS16550A REGISTER ADDRESS/DATA EQUATES
;
;************ UART REGISTERS ************************
;
rxd     EQU 3F8H            ;RECEIVE DATA REG
txd     EQU 3F8H            ;TRANSMITT DATA REG
ier     EQU 3F9H            ;INTERRUPT ENABLE REG
dll     EQU 3F8H            ;DIVISOR LATCH LOW
dlh     EQU 3F9H            ;DIVISOR LATCH HIGH
iir     EQU 3FAH            ;INTERRUPT IDENTIFICATION REG
fcr     EQU 3FAH            ;FIFO CONTROL REG
lcr     EQU 3FBH            ;LINE CONTROL REG
mcr     EQU 3FCH            ;MODEM CONTROL REG
lsr     EQU 3FDH            ;LINE STATUS REG
msr     EQU 3FEH            ;MODEM STATUS REG
scr     EQU 3FFH            ;SCRATCH PAD REG
;
;***************** DATA EQUATES *****************
;
bufsize EQU 7CFH            ;TX AND RX BUFFER SIZE
dosrout EQU 25H             ;DOS ROUTINE SPECIFICATION
intnum  EQU 0CH             ;INTERRUPT NUMBER (0CH = COM1)
icumask EQU 0EFH            ;ICU INTERRUPT ENABLE MASK
divacc  EQU 80H             ;DIVISOR LATCH ACCESS CODE
lowdiv  EQU 06H             ;LOWER DIVISOR
uppdiv  EQU 00H             ;UPPER DIVISOR
dataspc EQU 1AH             ;DLAB = 0, 7 BITS, 1 STOP, EVEN
fifospc EQU 0C1H            ;FIFOS ENABLED, TRIG = 14, DMA MODE = 0
setout2 EQU 08H             ;SETTING OUT2 ENABLES INTRs TO THE ICU
intmask EQU 0FH             ;UART INTERRUPT ENABLE MASK
;
;************ ESTABLISH CODE AND DATA SEGMENTS ******************
;
cseg    SEGMENT PARA PUBLIC "code"
        ORG     100H
        ASSUME  CS:cseg,DS:cseg
INIT:
        PUSH    CS
        POP     DS
        JMP     START
;
;********* ESTABLISH DATA BUFFERS AND RAW REGISTERS ********
;
msflag  DB      0
txflag  DB      0
sbuf    DB      bufsize DUP ("S")       ; STRING BUFFER
rbuf    DB      bufsize DUP ("R")       ; RECEIVE BUFFER
sbufe   EQU     sbuf + bufsize          ; END OF STRING BUFFER
rbufe   EQU     rbuf + bufsize          ; END OF RECEIVE BUFFER
;
START:
        CLI                         ;>>> DISABLE CPU INTERRUPTS <<<
```

```
;
;****** LOAD NEW INTERRUPT SERVICE ROUTINE POINTER FOR COM1 ***
;
        PUSH    DS              ;SAVE EXISTING DATA SEG
        MOV     AH,dosrout      ;DESIGNATE FUNCTION NUMBER
        MOV     AL,intnum       ;DESIGNATE INTERRUPT
        PUSH    CS              ;ALIGN CODE SEG
        POP     DS              ;WITH DATA SEG
        MOV     DX,OFFSET INTH  ;SPECIFY SERVICE ROUTINE OFFSET
        INT     21H             ;REPLACE EXISTING INTR VECTOR
        POP     DS              ;RESTORE CURRENT DATA SEG
;
;*************** INITIALIZE NS16550A **********************
;
;This enables both FIFOs for data transfers at 19.2 kbaud using
;7 bit data, 1 stop bit and even parity. The Rx FIFO interrupt
;trigger level is set at 14 bytes.
        MOV     AL,divacc       ;SET-UP ACCESS TO DIVISOR LATCH
        MOV     DX,lcr
        OUT     DX,AL
        MOV     AL,lowdiv       ;LOWER DIVISOR LATCH, 19.2 kbaud
        MOV     DX,dll
        OUT     DX,AL
        MOV     AL,uppdiv       ;UPPER DIVISOR LATCH
        MOV     DX,dlh
        OUT     DX,AL
        MOV     AL,dataspc      ;DLAB = 0, 7 BITS, 1 STOP, EVEN
        MOV     DX,lcr
        OUT     DX,AL
        MOV     AL,fifospc      ;FIFOS ENABLED, TRIGGER = 14,
        MOV     DX,fcr          ;DMA MODE = 0
        OUT     DX,AL
        MOV     AL,intmask      ;ENABLE ALL UART INTERRUPTS
        MOV     DX,ier
        OUT     DX,AL
        MOV     DX,lsr          ;READ THE LSR TO CLEAR ANY FALSE
        IN      AL,DX           ;STATUS INTERRUPTS
        MOV     DX,msr          ;READ THE MSR TO CLEAR ANY FALSE
        IN      AL,DX           ;MODEM INTERRUPTS
;
;*************** ENABLE COM1 INTERRUPTS *********************
;
        IN      AL,21H          ;CHECK IMR
        AND     AL,icumask      ;ENABLE ALL EXISTING AND COM1
        OUT     21H,AL
        MOV     AL,setout2      ;SET OUT2 TO ENABLE INTR
        MOV     DX,mcr
        OUT     DX,AL
;
;********* ESTABLISH RUN TIME BUFFER POINTERS IN REGISTERS ***
;
        MOV     SI,OFFSET sbuf
        MOV     DI,OFFSET rbuf
        MOV     BX,OFFSET sbuf
        MOV     BP,OFFSET rbuf
        STI     ;>>> ENABLE CPU INTERRUPTS <<<
```

## 3.0 Board to Board Communications with the NS16550A

The following section describes the hardware and software for a fully asynchronous two board application. The two boards communicate simultaneously with each other via the NS16550As. Predetermined data is exchanged between the NS16550As and checked by the software for accuracy. Any data mismatches are flagged and stop the programs. Any data errors (i.e. overrun, parity, framing or break) will also stop the program. The NS16550A interface schematic, software flow chart and software are provided.

### HARDWARE REQUIREMENTS

Running this application requires two NS32032-based boards. Each board must have one CPU, one ICU (NS32202), 256k of RAM (000000–03FFFF), the capability of running a monitor program (MON 32) and the capability of interfacing with a terminal. If MON 32 is not available, the display monitor service calls (SVC) must be altered to interface properly to the available terminal driver routines. In addition to these requirements, the NS16550A is enabled starting at address 0d00000.

The system described above was implemented on two DB32032 boards and used as an alpha site to test the NS16550A during its development. An NS16550A and appropriate decode logic were wirewrapped to each board (see *Figure 5*). As shown, an 8 MHz crystal is used to drive the baud rate generator, but for baud rates at or below 56 kbaud a 1.8432 MHz crystal can be substituted with changes to the divisor. Once this hardware is on both boards 5 connections between the NS16550As must be made—SIN to SOUT, SOUT to SIN, $\overline{CTS}$ to $\overline{RTS}$, $\overline{RTS}$ to $\overline{CTS}$, and GND to GND. Each DB32032 board has a port for attaching a terminal and a port available for downloading code. The applications software for these boards is downloaded from a VAX™ running the GNX™ debugger (V1.02). Once the downloads are complete to both boards the program D1APPS.EXE is started, then D2APPS.EXE is started.

If a VAX or the GNX debugger is not available the code can be loaded into PROMs and run directly.

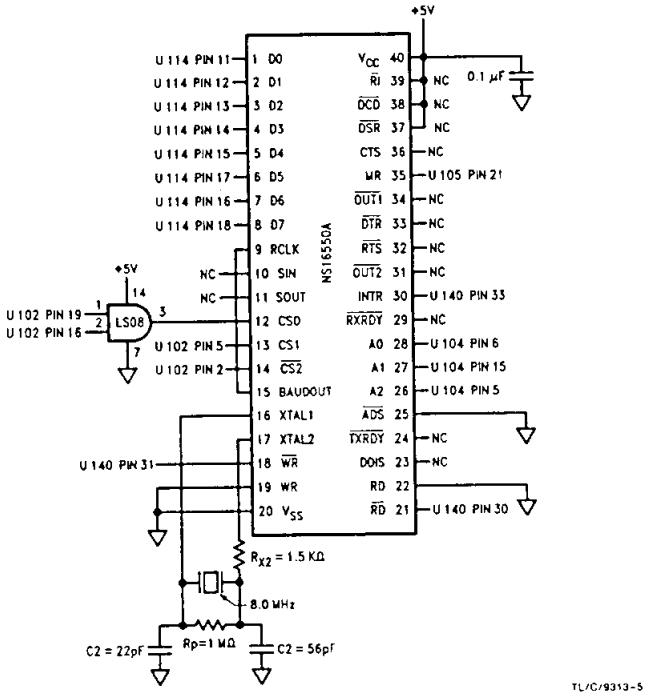



FIGURE 5. NS16550A and DB32032 Board Interconnections

## SOFTWARE OVERVIEW

The programs shown at the end of this application note are the assembly listings for D1APPS.ASM and D2APPS.ASM. These can be assembled, linked and loaded to form the executable (.EXE) files. The flowchart shown before them illustrates both programs.

Both programs are interrupt driven. D1APPS.EXE has its transmitter empty interrupt disabled until it receives its first 16 bytes from D2APPS.EXE. This allows the two programs to be started at different times. Data flow is controlled between the programs via $\overline{RTS}$ and $\overline{CTS}$ handshakes. D1APPS.EXE is started first and it loops until the first data from D2APPS.EXE arrives. As D1APPS.EXE exits its receiver interrupt routine, it enables its transmitter interrupt and begins to send bytes to D2APPS.EXE.

Transmission of a block of 16 bytes occurs when the Tx FIFO of the NS16550A is empty, the Tx interrupt is enabled and the receiver activates its clear to send ($\overline{CTS}$) signal. Each transmitter sends the next sequential block of data from a 256 byte buffer. When the bottom of the buffer is reached, the transmitter starts at the top of the buffer, again. The data transmitted from D1APPS.EXE to D2APPS.EXE is 00 to FF and from D2APPS.EXE to D1APPS.EXE is FF to 00. Since these are bench test programs for the NS16550A, the receiver subroutines compare the data they receive with the data they expect. This is done on a block-by-block basis and any mismatches result in both a message sent to the terminal and the program stopping.

## DETAILED SOFTWARE DESCRIPTION

Initialization begins by equating NS16550A and ICU (NS32202) registers to the addresses in memory. The equates finish with a list of offsets associated with the static base register. These offsets give the starting locations for the RAM areas assigned to be data buffers. These include the UART interrupt entry offset (irl_mod); the string (sbuf), receive (rbuf), compare (cbuf) buffers and the interrupt table offset (intable).

At the code start (START::) the processor is put in the supervisor mode so that the interrupt dispatch table can be transferred from ROM to RAM. This transfer is essential in order to change the starting address of the UART interrupt service routine. To do this the interrupt service routine offset from the code start is calculated (isr-start). Combining this with the module table address (set-up by the linker, i.e., 9020) results in the interrupt table descriptor entry for UART interrupt service routine (isrent).

The next two sections of code load the data to be transmitted and compared into the RAM buffers sbuf and cbuf, respectively. The two programs differ at this point—D1APPS.EXE transmits 00 to FF and compares FF to 00 sequentially. D2APPS.EXE transmits FF to 00 and compares 00 to FF sequentially.

The NS16550A initialization starts with setting the divisor latch access bit, so the divisor can be loaded. It then determines the serial data format and disables all UART interrupts. The NS16550A initialization finishes by enabling and resetting the FIFOs and programming the receiver interrupt level for 14 bytes.

Next the ICU interrupt registers are set-up and interrupts are enabled. In program D1APPS.ASM the initialization finishes by enabling the receive data and line status interrupts. Since the transmitter FIFO empty interrupt is disabled D1APPS.EXE will stay in its hold loop until it receives data from D2APPS.EXE. D2APPS.EXE has its transmitter FIFO empty interrupt enabled at the end of its initialization, so it will send one block of 16 characters to D1APPS.EXE immediately.

When there are no interrupts pending and no service routines being executed, the programs run in a holding loop until the next interrupt.

Whenever the CPU enters the service routine (isr:) it checks the interrupts identification register (IIR) for the type of interrupt pending and branches to the appropriate subroutine. If the IIR value doesn't match a known interrupt condition, an invalid interrupt message is sent to the terminal and the program stops. Out of the five possible interrupts, two (line status and receiver timeout) have simple routines that only send a message to the terminal and then branch to the receiver data available routine. Modem status interrupts send a message to the CRT and then stop the program. Two robust interrupt service routines exist—one for the receiver and one for the transmitter.

The receiver interrupt service routine (rdai:) does the following:

1. Disables the $\overline{RTS}$ signal which stops the transmitter on the other board from sending more data.
2. Transfers all data from the UART Rx FIFO to the RAM receiver buffer (rbuf).
3. Branches to the compare subroutine when all data is transferred from the Rx FIFO.
4. Enables Tx interrupts in D1APPS.EXE.
5. Enables the $\overline{RTS}$ signal which allows the transmitter on the other board to send another block of data.

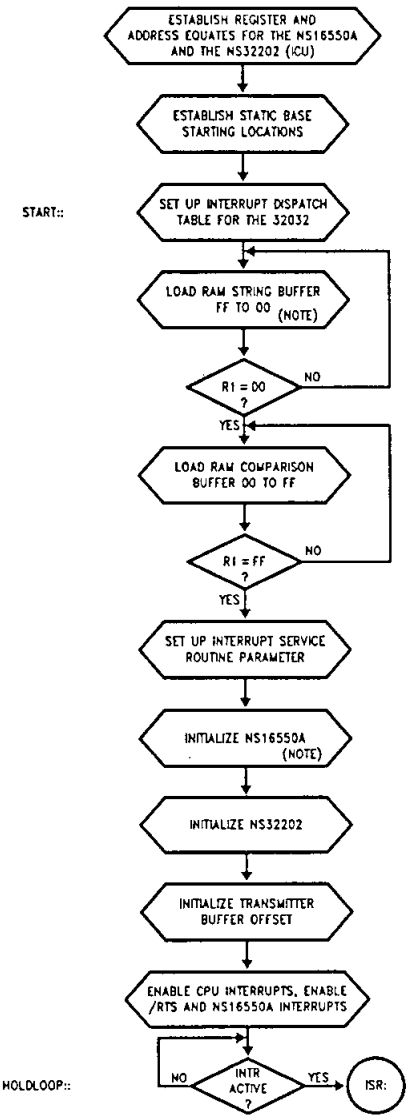
The compare interrupt service routine (compare:) does the following:

1. Aligns the receive buffer pointer to the last character taken in to the receive buffer (rbuf).
2. Compares each new byte in rbuf with the expected value (data stored in cbuf).
3. Sends a data mismatch message to the terminal and stops the program if the rbuf data fails to match the cbuf data.
4. Returns to rdai: when all of the new data in rbuf has compared successfully.

The transmitter interrupt service routine (threi:) does the following:

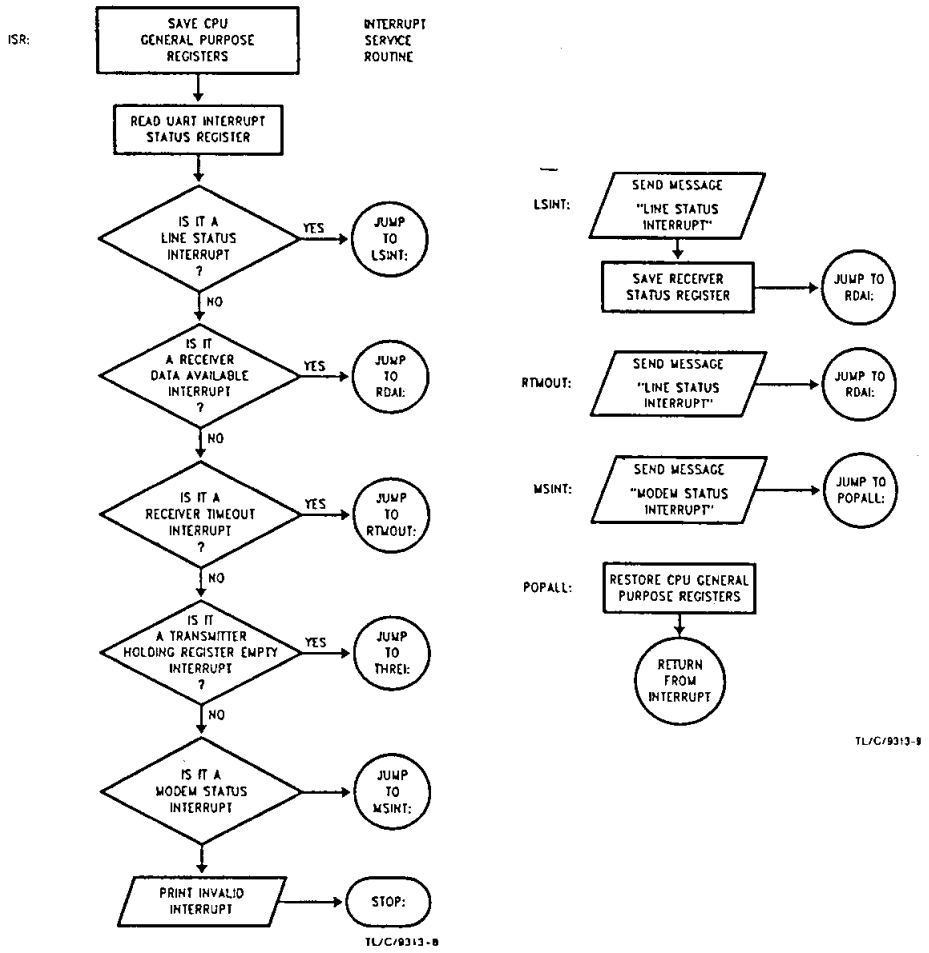
1. Decides whether to send 16 or 15 bytes in a block of data. Note: This decision is for testing purposes.
2. Sends one byte of data.
3. Checks for an active $\overline{CTS}$ condition. If it is active then it sends another byte of data. It continues to check and send a byte of data until all 15 or 16 bytes are sent.

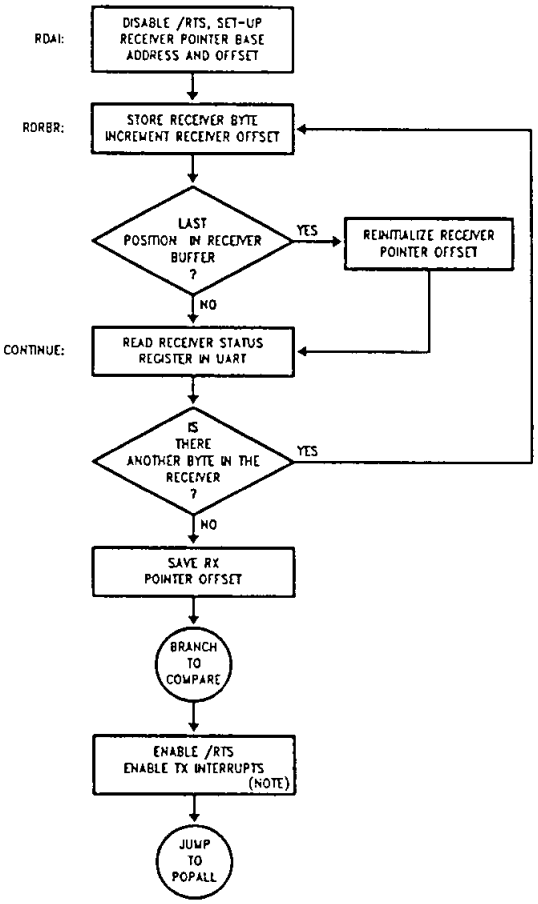
**DIAPPS.ASM Flow Chart**




TL/C/9313–7

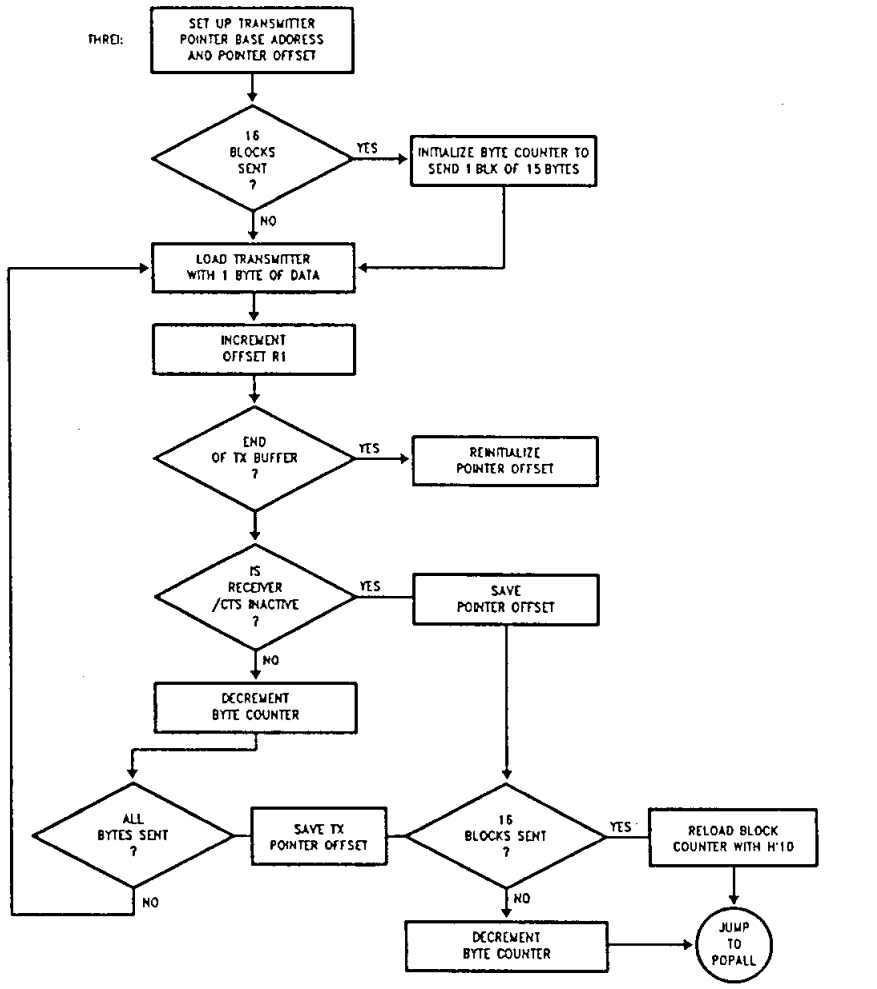
Note: This part of the software differs slightly in D2APPS ASM

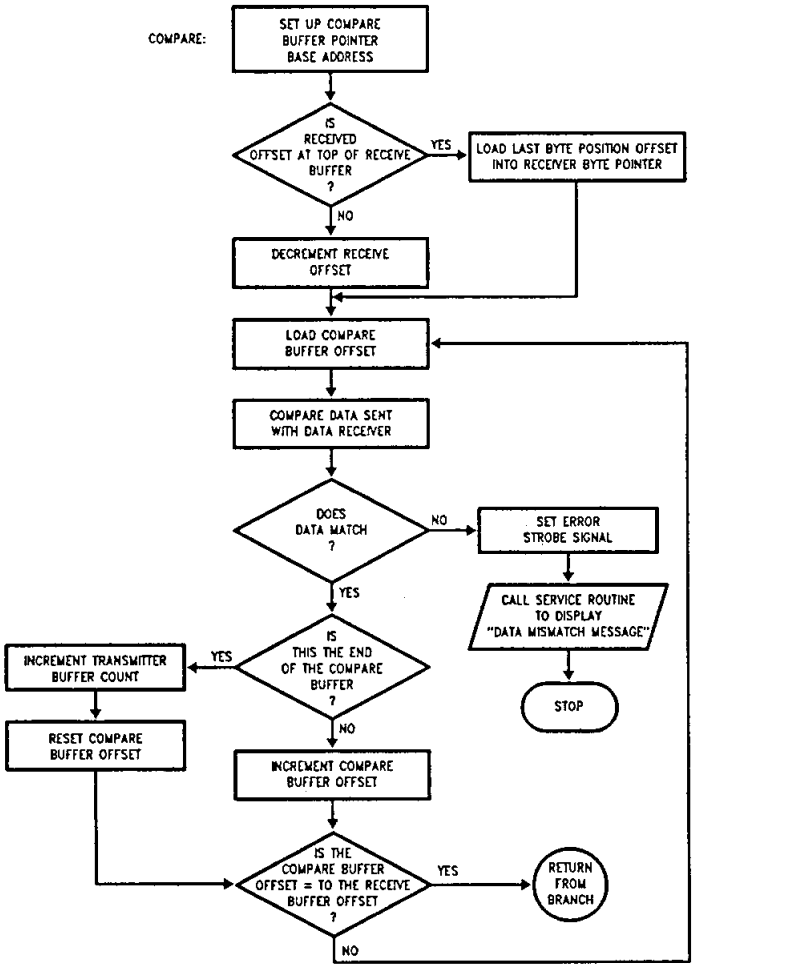

ISR:
SAVE CPU GENERAL PURPOSE REGISTERS
INTERRUPT SERVICE ROUTINE
READ UART INTERRUPT STATUS REGISTER
IS IT A LINE STATUS INTERRUPT ?
YES
JUMP TO LSINT:
NO
IS IT A RECEIVER DATA AVAILABLE INTERRUPT ?
YES
JUMP TO RDAI:
NO
IS IT A RECEIVER TIMEOUT INTERRUPT ?
YES
JUMP TO RTMOUT:
NO
IS IT A TRANSMITTER HOLDING REGISTER EMPTY INTERRUPT ?
YES
JUMP TO THREI:
NO
IS IT A MODEM STATUS INTERRUPT
JUMP TO MSINT:
PRINT INVALID INTERRUPT
STOP:
TL/C/9313-8
LSINT:
SEND MESSAGE "LINE STATUS INTERRUPT"
SAVE RECEIVER STATUS REGISTER
JUMP TO RDAI:
RTMOUT:
SEND MESSAGE "LINE STATUS INTERRUPT"
JUMP TO RDAI:
MSINT:
SEND MESSAGE "MODEM STATUS INTERRUPT"
JUMP TO POPALL:
POPALL:
RESTORE CPU GENERAL PURPOSE REGISTERS
RETURN FROM INTERRUPT
TL/C/9313-9




TL/C/9313-10

Note: This part of the software differs slightly in D2APPS ASM


THRD:
SET UP TRANSMITTER POINTER BASE ADDRESS AND POINTER OFFSET
16 BLOCKS SENT ?
YES
INITIALIZE BYTE COUNTER TO SEND 1 BLK OF 15 BYTES
NO
LOAD TRANSMITTER WITH 1 BYTE OF DATA
INCREMENT OFFSET R1
END OF TX BUFFER ?
YES
REINITIALIZE POINTER OFFSET
IS RECEIVER /CTS INACTIVE ?
YES
SAVE POINTER OFFSET
NO
DECREMENT BYTE COUNTER
ALL BYTES SENT ?
SAVE TX POINTER OFFSET
16 BLOCKS SENT ?
YES
RELOAD BLOCK COUNTER WITH H'10
NO
NO
DECREMENT BYTE COUNTER
JUMP TO POPALL
TL/C/9313-11


COMPARE:
SET UP COMPARE BUFFER POINTER BASE ADDRESS
IS RECEIVED OFFSET AT TOP OF RECEIVE BUFFER ?
YES
LOAD LAST BYTE POSITION OFFSET INTO RECEIVER BYTE POINTER
NO
DECREMENT RECEIVE OFFSET
LOAD COMPARE BUFFER OFFSET
COMPARE DATA SENT WITH DATA RECEIVER
DOES DATA MATCH ?
NO
SET ERROR STROBE SIGNAL
CALL SERVICE ROUTINE TO DISPLAY "DATA MISMATCH MESSAGE"
STOP
YES
IS THIS THE END OF THE COMPARE BUFFER ?
YES
INCREMENT TRANSMITTER BUFFER COUNT
RESET COMPARE BUFFER OFFSET
NO
INCREMENT COMPARE BUFFER OFFSET
IS THE COMPARE BUFFER OFFSET = TO THE RECEIVE BUFFER OFFSET ?
YES
RETURN FROM BRANCH
NO

TL/C/9313-12

```
;3/30/87.....D1APPS.ASM.........ADAPTED ORIGINALLY FROM D1RON56K.ASM
;
;THIS PROGRAM RUNS USING 2 DB32000 BOARDS WITH 16550As ENABLED AT ADDRESS 0d00000
;WIRE-WRAPPED ON THE BOARDS. THIS SOFTWARE TRANSMITS THE DATA 00 THROUGH FF
;REPEATEDLY TO THE REMOTE UART AND EXPECTS TO REPEATEDLY RECEIVE THE DATA FF
;THROUGH 00 FROM THE REMOTE UART. IT SHOULD BE RUN IN CONJUNCTION WITH THE
;PROGRAM D2APPSC.ASM RUNNING ON THE OTHER DB32000 BOARD. THE TX PIN OF
;THIS 16550A SHOULD CONNECT TO THE RX PIN OF THE 16550A ON THE OTHER BOARD AND
;VICE VERSA. ALSO, THE CTS PIN OF THIS 16550A SHOULD BE CONNECTED TO THE RTS PIN
;OF THE 16550A ON THE OTHER BOARD AND VICE VERSA. THIS WILL ENABLE THE
; APPROPRIATE HANDSHAKES TO OCCUR.
;
;TO RUN THIS PROGRAM YOU MUST:
;
;       1. CONNECT THE RX & TX OF THE 2 16550As ON THE 2 DB32000 BOARDS
;       2. CONNECT THE CTS & RTS OF THE 2 16550As ON THE 2 DB32000 BOARDS
;       3. DOWNLOAD D1APPS.EXE TO THIS BOARD VIA THE GNX DEBUGGER [REV 1.02]
;       4. DOWNLOAD D2APPS.EXE TO OTHER BOARD VIA THE GNX DEBUGGER [REV 1.02]
;       5. START D1APPS.EXE RUNNING ON THIS DB32000 BOARD
;       6. START D2APPS.EXE RUNNING ON THE OTHER DB32000 BOARD
;
;PROGRAM DETAILS:
;
;
; ISR contains the TX SERVICE ROUTINE
;
; TX OVERWRITES are PREVENTED by the ICU
;
; TX FIFO is CLEARED before a transmission
;
; DATA SENT  00 ------ FF
;
; DATA RECEIVED  and COMPARED FF ------ 00
;
; BAUDRATE 128k WITH A 8.0 MHZ XTAL INPUT TO THE 16550A
                                                    ;
;********************** ESTABLISH 16550A REGISTER ADDRESSES ********************
                                                    ;
        .globl          ier                         ;
            .set rxd,     0x0d00000     ;Equate registers to their addresses
            .set txd,     0x0d00000     ;
            .set ier,     0x0d00004     ;
            .set iir,     0x0d00008     ;
            .set fcr,     0x0d00008     ;
            .set lcr,     0x0d0000c     ;
            .set mcr,     0x0d00010     ;
            .set lsr,     0x0d00014     ;
            .set msreg,   0x0d00018     ;
            .set scr,     0x0d0001c     ;
                                        ;
;****************** ESTABLISH ADDRESSES FOR THE 32202 (ICU) ********************
                                        ;
            .set a0,4                   ;Establish address alignment
                                        ;between CPU and ICU
            .set icu_hvct,0             ;ICU register addresses
            .set icu_avct,1 *a0         ;
            .set icu_elgt,2 *a0         ;
            .set icu_tpl,4  *a0         ;
            .set icu_ipnd,6 *a0         ;
            .set icu_isrv,8 *a0         ;
```

TL/C/9313-13

```
            .set icu_imsk,10 *a0          #
            .set icu_csrc,12 *a0          #
            .set icu_fprt,14 *a0          #
            .set icu_mctl,16 *a0          #
            .set icu_ciptr,18 *a0         #
            .set icu_pdat,19  *a0         #
            .set icu_ips,20   *a0         #
            .set icu_pdir,21  *a0         #
            .set icu_cctl,22  *a0         #
            .set icu_cict1,23 *a0         #
                                          #
                                          #First ICU register address
                                          #
            .set icu_addr,0xfffe00        #
                                          #
#************************ STATIC BASE STARTING LOCATIONS *********************
                                          #
            .set irl_mod, 17*4            #
            .set irl_off, 17*4+2          #
            .set start2, 0x0              #The following are static base variables
            .set start1, 0x0a             #used as base pointers. Start1/2 = flags
            .set txflag, 0x14             #txflaf = flag, sbuf = area used to
            .set sbuf, 0x1e               #store data to be transmitted, rbuf =
            .set rbuf, 0x41e              #area used to store received data,
            .set cbuf, 0x61e              #cbuf = area used to store compare
            .set intable, 0x81e           #buffer, intable = base pointer to the
                                          #interrupt table
                                          #
#********************** SET UP DISPATCH TABLE FOR THE 32032 ******************
                                          #
start::     bicpsrw $(0x100)              #Clear intr's
            movd $0x0c,r0                 #Set for monitor svc to move intbase
            movd $0x055555555,r1          #from ROM to ram because you have
            addr intable(sb),r2           #to change the address for the
            movd $0x0c,r3                 #interrupt service routine.
            svc                           #Actual svc for move
            sprd intbase,r2               #Put base addr of intbase in r2
            movd isrent,irl_mod(r2)       #Put offset of isr into 1st location
                                          #of dispatch table
                                          #
#********************** LOAD TRANSMITTER BUFFER (00 to FF) *******************
                                          #
senddat:    addr  sbuf(sb),r0             #R0 contains string buffer ptr.
            movd  $0,r1                   #R1 contains offset
            movb  $0,r2                   #Init data reg.
sbufloop:   movb  r2,0(r0)[r1:b]          #Load char. to string buffer
            addqw 1,r1                    #Increment offset ptr.
            addqw 1,r2                    #Increment data
            cmpw  r1,$256                 #Check for 256 chars. loaded
            bne   sbufloop                #Jump back if not done
                                          #
#********************** LOAD COMPARISON BUFFER (FF TO 00)*********************
                                          #
compdat:    addr  cbuf(sb),r0             #R0 contains pointer
            movd  $0,r1                   #R1 contains offset
            movb  $0x0ff,r2               #Init data reg.
cbufloop:   movb  r2,0(r0)[r1:b]          #Load char. to compare buffer
            addqw 1,r1                    #Increment ptr. offset
            subb  $1,r2                   #Decrement data
            cmpw  r1,$256                 #Check for 256 chars. loaded
```

TL/C/9313-14

```
                bne    cbufloop              #Jump back if not done
                                             #
#*************** SET UP INTERRUPT SERVICE ROUTINE PARAMETERS ******************
                                             #
                movd $0x0ff,start2(ab)       #Initialize compare
                movd $0x0ff,start1(ab)       #Initialize receiver data intr
                movd $16,blk16cnt            #Initialize 16 byte block counter
                movd $0,soufcnt              #Initialize string buffer transmitted
                                             #count
                                             #
#************************** 16550A INITIALIZATION *****************************
                                             #
                movb  $0x080,lcr             #Set dlab = 1 for divisor latch access
                movb  $4,txd                 #Low divisor latch 128k w/8.0 MHz xtal
                movb  $0,ier                 #Upper divisor latch
                movb  $0x003,lcr             #Dlab = 0, 8 bits, no parity, 1 stop
                movb $0,ier                  #Disable UART interrupts
                movb  $0x0c7,fcr             #Fifo=> trigger = 14, reset & enable
                                             #
#*************************** INITIALIZE 32202 (ICU) ***************************
                                             #
          movd    $icu_addr,r0               #R0 = icu address
          movb    $0xca,icu_mctl(r0)         #Set mode : 8 bit bus mode,
                                             #           freeze counters,
                                             #           disable interrupts,
                                             #           fixed priority.
          movqb 0,icu_cctl(r0)               #Halt the counters
          movqb -1,icu_ips(r0)               #Set all pins to interrupt source
          movqb 0,icu_carc(r0)               #No cascaded interrupts (low reg)
          movqb 0,icu_carc+a0(r0)            # (high reg)
          movb $0x10,icu_svct(r0)            #Set interrupt base vector
          movqb -1,icu_elgt(r0)              #Set level triggering mode (low reg)
          movqb -1,icu_elgt+a0(r0)           #(high reg)
          movqb $2,icu_tpl(r0)               #Set level triggering mode (low reg)
          movqb 0,icu_tpl+a0(r0)             #(high reg)
          movqb 0,icu_fprt(r0)               #Set highest priority to 0 (low reg)
          movqb 0,icu_fprt+a0                #(high reg)
          movqb 0,icu_iserv(r0)              #Clear intr in-service regs (low reg)
          movqb 0,icu_iserv+a0(r0)           #(high reg)
          movqb -1,icu_imsk(r0)              #Mask all intr (low reg)
          movqb -1,icu_imsk+a0(r0)           #(high reg)H
          setcfg [1]                         #Enable vectored intrp (I=1)
          movd $icu_addr,r0                  #
          movb $0x02,icu_mctl(r0)            #Fixed mode, 8 bit bus mode
          movb $0x010,icu_cctl(r0)           #Set to internal sampling
          movb $0xfd,icu_imsk(r0)            #Enable irl
          movb $0xff,icu_imsk+a0(r0)         #Mask all other interrupts
          bisparw $(0x8000)                  #Enable cpu intr's
                                             #
          movd $0,r1                         #Initialize transmitter buffer offset
                                             #
#************************** ENABLE 16550A INTERRUPTS **************************
                                             #
               movb $2,mcr                   #Clear out1, out2 and enable rts
endinit:       movb $0x05,ier                #Enable all but modem status interrupts
                                             #and the THRE so the boards can be
                                             #started.
                                             #
#********************** ENDLESS LOOP WAITING FOR INTERRUPTS *******************
                                             #
```

TL/C/9313-15

```
holdloop:       nop                     #
                br holdloop             #
                                        #
#****************************** INTERRUPT HANDLER ******************************
                                        #
isr:            save [r0,r1,r2,r3,r4,r5,r6,r7]
                movb iir,r0             #R0- contains iir
                cmpb r0,$0x0c6          #
                beq laint               #Line status interrupt
                cmpb r0,$0x0c4          #
                beq rdai                #Receiver interrupt
                cmpb r0,$0x0cc          #
                beq rtmout              #Rec timeout interrupt
                cmpb r0,$0x0c2          #
                beq threi               #THRE interrupt
                cmpb r0,$0x0c0          #
                beq maint               #Modem status interrupt
                                        #
                                        #
#************************** INVALID INTERRUPT ROUTINE **************************
                                        #
                save [r0,r1,r2,r3]      #
                movd $4,r0              #
                addr message2,r1        #
                movd $21,r2             #
                movd $0,r3              #
                svc                     #
                restore [r0,r1,r2,r3]   #
                                        #
                                        #
                jump stop               #Restore all registers
                                        #
                                        #
#********************* RECEIVER TIMEOUT INTERRUPT ROUTINE **********************
                                        #
rtmout:         jump rdai
                                        #
#*************************** RECEIVER INTERRUPT ROUTINE ************************
                                        #
#This portion of the program is reached by a positive test for the received data
#available interrupt. Once in this routine each byte is removed from the FIFO,
#placed in a designated static base memory location and the LSR is tested to see
#if the data ready (DR) bit is still set. Data is removed from the FIFO and
#placed in memory until the DR bit is no longer set. The data sent will be
#compared to known data, located in another designated static base location, by
#calling the compare subroutine.
                                        #
rdai:           movb $0,mcr             #Disable RTS; stop transmission
                addr rbuf(sb),r4        #r4 contains rbuf base address
                movd rbufoff,r6         #Put rbuf offset runner into r6
rdrbr:          movb rxd,0(r4)[r6:b]    #Store a byte in the receiver buffer
                cmpb $0x00,0(r4)[r6:b]  #Is it the last character
                addqw 1,r6              #Increment offset ptr.
                addqw 1,rbufoff         #Track r6
                bne continue            #
                movw $0,r6              #Reset pointer offset
                movw $0,rbufoff         #Reset rbufoff
continue:       movb lsr,r3             #Read lsr
                andb $01,r3             #Mask all but bit 0
                cmpb $01,r3             #
```

TL/C/9313-16

```
                beq rdrbr               #Read rbr again if set
                movd r6,rbufoff         #Put result of r6 back into rbufoff
                bar compare             #
                movb $7,ier             #Turn on tranamitter interrupts
                movb $2,mcr             #Enable rts
                jump popall             #
                                        #
#****************************** TRANSMIT ROUTINE ******************************
                                        #
#Before the tranamitter aenda data, the data has been loaded into static base
#memory for tranamiaaion. The tranamitter routine ia called to aend data. (ie
#THREI ia aet) Data ia aent aa 16 blocka of 16 bytea and 1 block of 15 bytea
#continuously. NOTE: Before tranamiaaion occura /CTS is checked to enaure that
#the receiver ia ready.
                                        #
threi:          addr sbuf(sb),r0        #R0 containa base pointer
                movw xmitoff,r1         #aetup xmit ptr offaet
                cmpd $0,blk16cnt        #Check to see if it is the 16th block *
                beq send15              #Yea, aend only 15 bytea inatead of 16 *
                movd $0x10,r7           #No, aend 16 bytea *
                jump sendnext           #Jump around 15 byte load *
send15:         movd $0x0f,r7           #Load counter for 15 byte load *
sendnext:       movb 0(r0)[r1:b],txd    #Load a byte into the tranamitter
                addqw 1,r1              #
                cmpw r1,$256            #Are we one addreaa paat end of table
                beq reload              #Yea, reload ptr
finish:         save [r7]               #
                movb mareg,r7           #Read modem atatua reg
                andb $0x10,r7           #Mask all bita except CTS (MSR4)
                cmpb $0,r7              #Check for diaabled CTS
                reatore [r7]            #
                beq abort               #Wait for active CTS (MSR4=1)
                aubb $1,r7              #No, decrement counter and continue
                cmpb $0,r7              #Ia byte counter 0?
                bne aendnext            #No, aend next byte
abort:          movw r1,xmitoff         #aave xmit ptr offaet in ram
                cmpd $0,blk16cnt        #Check to aee if it ia 16th block *
                beq aetand16            #Yea, reload block counter *
                aubb $1,blk16cnt        #Decrement block counter *
                jump popall             #Finiahed aending 16 bytea
aetand16:       movd $16,blk16cnt       #Reload block counter *
                jump popall             #Finiahed aending 15 bytea *
reload:         movd $0,r1              #Reaet offaet
                jump finish             #Go back and finiah
                                        #
#************************ LINE STATUS INTERRUPT ROUTINE ************************
                                        #
laint:          aave [r0,r1,r2,r3]      #
                movd $4,r0              #
                addr meaaage6,r1        #
                movd $25,r2             #
                movd $0,r3              #
                avc                     #
                reatore [r0,r1,r2,r3]   #
                movb lar,r3             #Read lar
                                        #
                                        #
                jump rdai               #
                                        #
#************************ MODEM STATUS INTERRUPT ROUTINE ***********************
```

TL/C/9313-17

```
msint:          save [r0,r1,r2,r3]
                movd $4,r0
                addr message7,r1
                movd $26,r2
                movd $0,r3
                svc
                movb 0x0d00018,r0
                restore [r0,r1,r2,r3]
                jump popall
#***************************** COMPARE DATA ROUTINE ***************************

#This subroutine is called by the receiver interrupt routine which has set the
#receiver offset (rbufoff) to point at the last byte received. This subroutine
#uses the compare offset (compoff) pointer as the pointer for both receive
#buffer data and compare buffer data. Each location is compared to ensure data
#sent is identical to data received. This is done until compoff equals rbufoff
#stopping the process and returning from the interrupt. NOTE: Data being
#received is known data and an exact copy is loaded into memory prior to any
#transmission.

compare:        addr cbuf(sb),r1        #R1- base address of cbuf base
                cmpd $0,r6              #Check for potential invalid subtraction
                beq zeror6              #Jump around subtraction
                subd $1,r6              #
                jump compbyte           #Jump around subtraction fix
zeror6:         movd $0xff,r6           #
compbyte:       movd compoff,r5         #
                cmpb 0(r1)[r5:b],0(r4)[r5:b] #Compare data sent to data received
                bne wrong               #Branch and set out1 if wrong
                                        #
                cmpb $0x00,0(r4)[r5:b]  #Check for end of buffer
                bne notend              #Branch and increment pointers
                jump reload1            #Test for having compared all bytes
                                        #
notend:         addd $1,compoff         #Increment pointer
notend1:        cmpd r5,r6              #
                beq bye                 #
                jump compbyte           #
                                        #
reload1:        addd $1,sbufcnt         #Increment transmiter cnt
                movd $0,compoff         #Reload offset of pointer
                jump notend1            #
                                        #
wrong:          nop                     #
                movb $0x0c,mcr          #Set out 2, for error strobe
                                        #
#************************ DATA MISMATCH MESSAGE ******************************
                                        #
                save [r0,r1,r2,r3]      #Save register for supervisor call
                movd $4,r0              #Value required by svc call
                addr message8,r1        #Mover address of message into r1
                movd $17,r2             #Number of characters into r2
                movd $0,r3              #Value required by svc call
                svc                     #Actual call
                restore [r0,r1,r2,r3]   #Restore registers
                                        #
stop:           nop                     #
                jump stop               #Test point
                                        #
```

TL/C/9313-18

```
bye:            ret 0

;************************** RETURN FROM INTERRUPT **************************

popall:         restore [r0,r1,r2,r3,r4,r5,r6,r7]
                reti

;********************************* Messages *********************************

                message1: .byte 13,10,"Compare Complete",13,10
                message2: .byte 13,10,"Invalid Interrupt",13,10
                message3: .byte 13,10,"Receiver Timeout",13,10
                message4: .byte 13,10,"Receive data available Interrupt",13,10
                message5: .byte 13,10,"THRE Interrupt",13,10
                message6: .byte 13,10,"Line Status Interrupt",13,10
                message7: .byte 13,10,"Modem Status Interrupt",13,10
                message8: .byte 13,10,"Data Mismatch",13,10
                xmitoff: .double 0
                compoff: .double 0
                blk16cnt: .double 0
                sbufcnt: .double 0
                rbufoff: .double 0
ierent: .word   0x9020                  ;Mod table
        .word   ier-start               ;Offset of service routine for
                                        ;Dispatch table.
```

TL/C/9313-19

```
#3/30/87.....D2APPS.ASM.........ADAPTED ORIGINALLY FROM D1RON56K.ASM
#
#THIS PROGRAM RUNS USING 2 DB32000 BOARDS WITH 16550As ENABLED AT ADDRESS
#0d00000 WIRE-WRAPPED ON THE BOARDS. THIS SOFTWARE TRANSMITS THE DATA FF
#THROUGH 00 REPEATEDLY TO THE REMOTE UART AND EXPECTS TO REPEATEDLY RECEIVE
#THE DATA 00 THROUGH FF FROM THE REMOTE UART. IT SHOULD BE RUN IN CONJUNCTION
#WITH THE PROGRAM D1APPS.ASM RUNNING ON THE OTHER DB32000 BOARD. THE TX PIN OF
#THIS 16550A SHOULD CONNECT TO THE RX PIN OF THE 16550A ON THE OTHER BOARD AND
#VICE VERSA. ALSO, THE CTS PIN OF THIS 16550A SHOULD BE CONNECTED TO THE RTS PIN
#OF THE 16550A ON THE OTHER BOARD AND VICE VERSA. THIS WILL ENABLE THE
# APPROPRIATE HANDSHAKES TO OCCUR.
#
#TO RUN THIS PROGRAM YOU MUST:
#
#       1. CONNECT THE RX & TX OF THE 2 16550As ON THE 2 DB32000 BOARDS
#       2. CONNECT THE CTS & RTS OF THE 2 16550As ON THE 2 DB32000 BOARDS
#       3. DOWNLOAD D2APPS.EXE TO THIS BOARD VIA THE GNX DEBUGGER [REV 1.02]
#       4. DOWNLOAD D1APPS.EXE TO OTHER BOARD VIA THE GNX DEBUGGER [REV 1.02]
#       5. START D1APPS.EXE RUNNING ON THE OTHER DB32000 BOARD
#       6. START D2APPS.EXE RUNNING ON THIS DB32000 BOARD
#
#PROGRAM DETAILS:
#
#
# ISR contains the TX SERVICE ROUTINE
#
# TX FIFO is CLEARED before a transmission
#
# DATA SENT  FF ------ 00
#
# DATA RECEIVED  and COMPARED 00 ------ FF
#
# BAUDRATE 128k WITH A 8.0 MHZ XTAL INPUT TO THE 16550A
                                        #
#********************** ESTABLISH 16550A REGISTER ADDRESSES ********************
                                        #
        .globl          isr             #
           .set rxd,     0x0d00000       #Equate registers to their addresses
           .set txd,     0x0d00000       #
           .set ier,     0x0d00004       #
           .set iir,     0x0d00008       #
           .set fcr,     0x0d00008       #
           .set lcr,     0x0d0000c       #
           .set mcr,     0x0d00010       #
           .set lsr,     0x0d00014       #
           .set msreg,   0x0d00018       #
           .set scr,     0x0d0001c       #
                                        #
#****************** ESTABLISH ADDRESSES FOR THE 32202 (ICU) ******************
                                        #
           .set a0,4                    #Establish address alignment
                                        #between CPU and ICU
           .set icu_hvct,0              #ICU register addresses
           .set icu_svct,1 *a0          #
           .set icu_elgt,2 *a0          #
           .set icu_tpl,4  *a0          #
           .set icu_ipnd,6 *a0          #
           .set icu_iscv,8 *a0          #
           .set icu_imsk,10 *a0         #
           .set icu_csrc,12 *a0         #
```

TL/C/9313-20

```
            .set icu_fprt,14 *a0          #
            .set icu_mctl,16 *a0          #
            .set icu_ciptr,18 *a0         #
            .set icu_pdat,19  *a0         #
            .set icu_ipa,20   *a0         #
            .set icu_pdir,21  *a0         #
            .set icu_cctl,22  *a0         #
            .set icu_cictl,23 *a0         #
                                          #
                                          #First ICU register address
                                          #
            .set icu_addr,0xfffe00        #
                                          #
#************************ STATIC BASE STARTING LOCATIONS **********************
                                          #
            .set ir1_mod, 17*4            #Dispatch table offset for IR1 entry
            .set sbuf, 0x1e               #sbuf = area used to
            .set rbuf, 0x41e              #store data to be transmitted, rbuf =
            .set cbuf, 0x61e              #area used to store received data,
            .set intable, 0x81e           #cbuf = area used to store compare
                                          #buffer, intable = base pointer to the
                                          #interrupt table
                                          #
#********************** SET UP DISPATCH TABLE FOR THE 32032 ********************
                                          #
start::     bicpsrw $(0x100)              #Clear intr's
            movd $0x0c,r0                 #Set for monitor svc to move intbase
            movd $0x055555555,r1          #from ROM to ram because you have
            addr intable(sb),r2           #to change the address for the
            movd $0x0c,r3                 #interrupt service routine.
            svc                           #Actual svc for move
            sprd intbase,r2               #Put base addr of intbase in r2
            movd ierent,ir1_mod(r2)       #Put offset of ier into 1st location
                                          #of dispatch table
                                          #
#********************* LOAD TRANSMITTER BUFFER (FF to 00) **********************
                                          #
senddat:    addr  sbuf(sb),r0             #R0 contains string buffer ptr.
            movd  $0,r1                   #R1 contains offset
            movb  $0x0ff,r2               #Init data reg.
sbufloop:   movb  r2,0(r0)[r1:b]          #Load char. to string buffer
            addqw 1,r1                    #Increment offset ptr.
            subb  $1,r2                   #Increment data
            cmpw  r1,$256                 #Check for 256 chars. loaded
            bne   sbufloop                #Jump back if not done
                                          #
#********************** LOAD COMPARISON BUFFER (00 TO FF) **********************
                                          #
compdat:    addr  cbuf(sb),r0             #R0 contains pointer
            movd  $0,r1                   #R1 contains offset
            movb  $0,r2                   #Init data reg.
cbufloop:   movb  r2,0(r0)[r1:b]          #Load char. to compare buffer
            addqw 1,r1                    #Increment ptr. offset
            addqw 1,r2                    #Decrement data
            cmpw  r1,$256                 #Check for 256 chars. loaded
            bne   cbufloop                #Jump back if not done
                                          #
#*************** SET UP INTERRUPT SERVICE ROUTINE PARAMETERS ******************
                                          #
            movd $16,blk16cnt             #Initialize 16 byte block counter
```

TL/C/9313-21

```
                                        #
#************************** 16550A INITIALIZATION ****************************
                                        #
                movb  $0x080,lcr        #Set dlab = 1 for divisor latch access
                movb  $4,txd            #Low divisor latch 56k w/8.0 xtal
                movb  $0,ier            #Upper divisor latch
                movb  $0x003,lcr        #Dlab = 0, 8 bits, no parity, 1 stop
                movb  $0,ier            #Disable UART interrupts
                movb  $0x0c7,fcr        #Fifo=> trigger = 14, reset & enable
                                        #
#***************************** INITIALIZE 32202 (ICU) *************************
                                        #
                                        #
          movd   $icu_addr,r0           #R0 = icu address
          movb   $0xca,icu_mctl(r0)     #Set mode : 8 bit bus mode,
                                        #           freeze counters,
                                        #           disable interrupts,
                                        #           fixed priority.
          movqb 0,icu_cctl(r0)          #Halt the counters
          movqb -1,icu_ipa(r0)          #Set all pins to interrupt source
          movqb 0,icu_csrc(r0)          #No cascaded interrupts (low reg)
          movqb 0,icu_csrc+a0(r0)       # (high reg)
          movb $0x10,icu_svct(r0)       #Set interrupt base vector
          movqb -1,icu_elgt(r0)         #Set level triggering (low reg)
          movqb -1,icu_elgt+a0(r0)      #(high reg)
          movqb $2,icu_tpi(r0)          #Set high polarity mode (low reg)
          movqb 0,icu_tpi+a0(r0)        #(high reg)
          movqb 0,icu_fprt(r0)          #Set highest priority to 0 (low reg)
          movqb 0,icu_fprt+a0           #(high reg)
          movqb 0,icu_iserv(r0)         #Clear intr in-service regs (low reg)
          movqb 0,icu_iserv+a0(r0)      #(high reg)
          movqb -1,icu_imsk(r0)         #Mask all intr (low reg)
          movqb -1,icu_imsk+a0(r0)      #(high reg)H
          setcfg [i]                    #Enable vectored intrp (I=1)
          movd $icu_addr,r0             #
          movb $0x02,icu_mctl(r0)       #Fixed mode, 8 bit bus mode
          movb $0x010,icu_cctl(r0)      #Set to internal sampling
          movb $0xfd,icu_imsk(r0)       #Enable irl
          movb $0xff,icu_imsk+a0(r0)    #Mask all other interrupts
          bispsrw $(0x800)              #Enable cpu intr's
                                        #
#************************** ENABLE 16550A INTERRUPTS **************************
                                        #
                movb $2,mcr             #Clear outl, out2 and enable rts
endinit:        movb $0x07,ier          #Enable all but modem status interrupts
                                        #
#********************** ENDLESS LOOP WAITING FOR INTERRUPTS *******************
                                        #
holdloop:       nop                     #
                br holdloop             #
                                        #
#******************************* INTERRUPT HANDLER ****************************
                                        #
isr:            save [r0,r1,r2,r3,r4,r5,r6,r7]
                movb iir,r0             #R0- contains iir
                cmpb r0,$0x0c6          #
                beq laint               #Line status interrupt
                cmpb r0,$0x0c4          #
                beq rdai                #Receiver interrupt
                cmpb r0,$0x0cc          #
```

TL/C/9313-22

```
                beq rtmout              #Rec timeout interrupt
                cmpb r0,$0x0c2          #
                beq threi               #THRE interrupt
                cmpb r0,$0x0c0          #
                beq maint               #Modem status interrupt
                                        #
                                        #
#************************* INVALID INTERRUPT ROUTINE ************************
                                        #
                save [r0,r1,r2,r3]      #
                movd $4,r0              #
                addr message2,r1        #
                movd $21,r2             #
                movd $0,r3              #
                svc                     #
                restore [r0,r1,r2,r3]   #
                                        #
                                        #
                jump stop               #Restore all registers
                                        #
                                        #
#********************* RECEIVER TIMEOUT INTERRUPT ROUTINE *********************
                                        #
rtmout:         jump rdai               #
                                        #
#************************** RECEIVER INTERRUPT ROUTINE **********************
                                        #
#This portion of the program is reached when the received data available
#interrupt is active. Once in this routine each byte removed from the FIFO
#is placed in the designated static base memory location (labelled rbuf).
#The data ready bit (DR) in the LSR is checked before each byte is removed
#from the FIFO. Data sent will be compared to known data in another designated
#static base area (labelled cbuf) by calling the compare subroutine.
                                        #
rdai:           movb $0,mcr             #Disable RTS; stop transmission
                addr rbuf[sb],r4        #r4 contains rbuf base address
                movd rbufoff,r6         #Put rbuf offset runner into r6
rdrbr:          movb rxd,0(r4)[r6:b]    #Store a byte in the receive buffer
                cmpb $0xff,0(r4)[r6:b]  #Is it the last character
                addqw 1,r6              #Increment offset ptr.
                addqw 1,rbufoff         #Track r6
                bne continue            #
                movw $0,r6              #Reset pointer offset
                movw $0,rbufoff         #Reset rbufoff
continue:       movb lsr,r3             #Read lsr
                andb $01,r3             #Mask all but bit 0
                cmpb $01,r3             #
                beq rdrbr               #Read rbr again if set
                movd r6,rbufoff         #Put result of r6 back into rbufoff
                bsr compare             #
                movb $2,mcr             #Enable rts
                jump popall             #
                                        #
#****************************** TRANSMIT ROUTINE ****************************
                                        #
#The transmitter sends data previously loaded into the static base memory area
#labelled sbuf. Thids routine sends data as 16 blocks of 16 bytes and 1 block
#of 15 bytes, continuously. NOTE: Before each block transmission occurs /CTS
#is checked to ensure that the receiver ready.
                                        #
```

TL/C/9313-23

```
threi:          addr sbuf(sb),r0        #R0 contains base pointer
                movw xmitoff,r1         #setup xmit ptr offset
                cmpd $0,blk16cnt        #Check to see if it is the 16th block
                beq send15              #Yes, send only 15 bytes instead of 16
                movd $0x10,r7           #No, send 16 bytes
                jump sendnext           #Jump around 15 byte load
send15:         movd $0x0f,r7           #Load counter for 15 byte load
sendnext:       movb 0(r0)[r1:b],txd    #Load a byte into the transmitter
                addqw 1,r1              #
                cmpw r1,$256            #Are we one address past end of table
                beq reload              #Yes, reload ptr
finish:         save [r7]
                movb msreg,r7           #Read modem status reg
                andb $0x10,r7           #Mask all bits except CTS (MSR4)
                cmpb $0,r7              #Check for disabled CTS
                restore [r7]
                beq abort               #Leave on inactive CTS (MSR4=0)
                subb $1,r7              #No, decrement counter and continue
                cmpb $0,r7              #Is byte counter 0?
                bne sendnext            #No, send next byte
abort:          movw r1,xmitoff         #save xmit ptr offset in ram
                cmpd $0,blk16cnt        #Check to see if it is 16th block
                beq setsnd16            #Yes, reload block counter
                subb $1,blk16cnt        #Decrement block counter
                jump popall             #Finished sending 16 bytes
setsnd16:       movd $16,blk16cnt       #Reload block counter
                jump popall             #Finished sending 15 bytes
reload:         movd $0,r1              #Reset offset
                jump finish             #Go back and finish
                                        #
#************************ LINE STATUS INTERRUPT ROUTINE ************************
                                        #
lsint:          save [r0,r1,r2,r3]      #
                movd $4,r0              #
                addr message6,r1        #
                movd $25,r2             #
                movd $0,r3              #
                svc                     #
                restore [r0,r1,r2,r3]   #
                movb lsr,r3             #Read lsr
                jump rdai               #
                                        #
#************************ MODEM STATUS INTERRUPT ROUTINE ************************
                                        #
msint:          save [r0,r1,r2,r3]      #
                movd $4,r0              #
                addr message7,r1        #
                movd $26,r2             #
                movd $0,r3              #
                svc                     #
                movb 0x0d000018,r0      #
                restore [r0,r1,r2,r3]   #
                jump popall             #
#****************************** COMPARE DATA ROUTINE ******************************
                                        #
#The receiver subroutine branches to this subroutine after it has removed all of
#the data from the Rx FIFO. The receive offset (rbufoff) is changed to point to
#the last byte received in rbuf. The compare offset (compoff) points to each
#byte in the receive buffer and its associated byte in the compare register.
#Compoff is incremented after each successful comparison and the comparisons
```

TL/C/9313-24

```
#end when compoff equals rbufoff. NOTE: Data being received by this test program
#is known data and a copy of it is loaded into cbuf before transmissions begin.
                                        #
compare:        addr cbuf(sb),r1        #R1- base address of cbuf base
                cmpd $0,r6              #Check for potential invalid subtraction
                beq zeror6              #Jump around subtraction
                subd $1,r6              #
                jump compbyte           #Jump around subtraction fix
zeror6:         movd $0xff,r6           #
compbyte:       movd compoff,r5         #
                cmpb 0(r1)[r5:b],0(r4)[r5:b] #Compare data sent to data received
                bne wrong               #Branch and set outl if wrong
                                        #
                                        #
                cmpb $0xff,0(r4)[r5:b]  #Check for end of buffer
                bne notend              #Branch and increment pointers
                jump reloadi            #Test for having compared all bytes
                                        #
                                        #
notend:         addd $1,compoff         #Increment pointer
notendi:        cmpd r5,r6              #
                beq bye                 #
                jump compbyte           #
                                        #
reloadi:        addd $1,sbufcnt         #Increment transmiter cnt
                movd $0,compoff         #Reload offset of pointer
                jump notendi            #
                                        #
wrong:          movb $0x0c,mcr          #Set out 2, for error strobe
                                        #
#************************ DATA MISMATCH MESSAGE ******************************
                                        #
                save [r0,r1,r2,r3]      #Save register for supervisor call
                movd $4,r0              #Value required by svc call
                addr message8,r1        #Mover address of message into r1
                movd $17,r2             #Number of characters into r2
                movd $0,r3              #Value required by svc call
                svc                     #Actual call
                restore [r0,r1,r2,r3]   #Restore registers
                                        #
stop:           nop                     #
                jump stop               #Test point
                                        #
bye:            ret 0                   #
                                        #
#***************************** RETURN FROM INTERRUPT ***************************
                                        #
popall:         restore [r0,r1,r2,r3,r4,r5,r6,r7]
                reti                    #
                                        #
#********************************** Messages ***********************************
                                        #
                message1: .byte 13,10,"Compare Complete",13,10
                message2: .byte 13,10,"Invalid Interrupt",13,10
                message3: .byte 13,10,"Receiver Timeout",13,10
                message4: .byte 13,10,"Receive data available Interrupt",13,10
                message5: .byte 13,10,"THRE Interrupt",13,10
                message6: .byte 13,10,"Line Status Interrupt",13,10
                message7: .byte 13,10,"Modem Status Interrupt",13,10
                message8: .byte 13,10,"Data Mismatch",13,10
```

TL/C/9313-25

# A Comparison of the INS8250, NS16450 and NS16550A Series of UARTs

National currently produces seven versions of the INS8250 UART. Functionally, these parts appear to be the same, however, there are differences that the designer and purchaser need to understand. For each version, this document provides a brief overview of their distinct characteristics, a detailed function and timing section, a discussion of software compatibility issues and the AC timing parameters.

## 1.0 Part Summary

The seven versions currently produced are designated INS8250, INS8250-B, INS8250A, NS16450, INS82C50A, NS16C450, and NS16550A. These devices are grouped below by process type.

### NMOS DEVICES

1. INS8250: This is the original version produced by National. It is the same part as the INS8250-B, but with faster CPU bus timings.
2. INS8250-B: This is the slower speed (CPU bus timing) version of the INS8250. It is used by many popular 8088-based microcomputers.

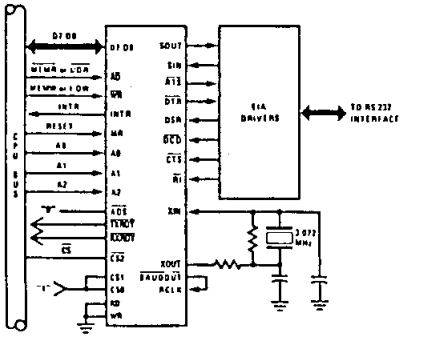
### XMOS DEVICES

1. INS8250A: This is a revision of the INS8250 using the more advanced XMOS process. The INS8250A is better than the aforementioned parts due to the redesign (compare section 2.0 to 3.0) and the following process characteristics—closer threshold voltage control, more reliably implemented process topography and finer control over the active area critical dimensions. XMOS and CMOS parts should be used for all new designs. This part is used in many popular 8086-based microcomputers.
2. NS16450: This is the faster speed (CPU bus timing) version of the INS8250A. It is used by many popular 80286-based microcomputers.
3. NS16550A: This is the newest member of the UART family. It powers-up in the NS16450 mode and is completely compatible with all software written for the NS16450. It has advanced features such as on-board FIFOs, a DMA interface, faster CPU bus timings and a much higher maximum baud rate than the NS16450. The NS16550A should be used for all new non-CMOS designs, including those that were originally done with the NS16550. It is used in recent versions of popular 80286-based, 80386-based and ROMP-based microcomputers. Software written for the NS16550 is completely compatible with the NS16550A. Section 5.0 describes how the software can distinguish between the NS16550 and the NST6550A.
4. NS16550: This part powers-up in the NS16450 mode and is completely compatible with all software written for the NS16450. It has advanced features, such as a DMA interface. The on-board FIFOs are essentially non-functional. This part was issued on a limited basis. Any user that wants this part should order the NS16550A. Section 5.0 describes the differences between the NS16550 and the NS16550A in detail.

### CMOS DEVICES

1. INS82C50A: This is a CMOS version of the INS8250A. It functions identically and for most AC parameters has the same timing specification as the INS8250A (see Section 4.0). It draws approximately 1/10 (10 mA) of the maximum operating current of the INS8250A.
2. NS16C450: This is a CMOS version of the NS16450. It functions identically and for most AC parameters has the same timing specification as the NS16450 (see Section 4.0). It draws approximately 1/12 (10 mA) of the maximum operating current of the NS16450.

**Note:** The XMOS and CMOS UARTs are not plug-in replacements for the INS8250/INS8250-B when used with ICUs that are in the popular edge-triggered configuration. However, there are easily implemented adjustments to the driving software or associated hardware that will allow these parts to be a plug-in replacement (see Section 6.0).




TL/C/9320–1

**FIGURE 1. Connection Diagram**

## 2.0 INS8250 and INS8250-B Functional Considerations

Designers using these NMOS parts should be well aware of the following considerations.

1. The Modem Status and Line Status registers are master-slave registers which transfer data from the master to the slave only when the INS8250, INS8250-B is not enabled. Thus, if the UART is never disabled:

   —The status registers are never updated.

   —The character in the transmit holding register will be transmitted repeatedly.

   —The CPU cannot read the current error status indication.

Recommendation: Disable the INS8250, INS8250-B between accesses.

2. At power-on the UART will occasionally transmit a random character. This occurs after release of the master reset and before it receives data from the CPU. The THRE and TSRE bits are unreliable at this time, due to their unpredictable state at power-on.

   Recommendation A: Use the following initialization routine:

   —Master reset.

   —Enable loopback mode (this causes any randomly sent characters to be sent to the receiver).

   —Load baud rate generator and initialize line control register.

   —Wait one character time and then clear the receiver buffer by reading it and clear any errors by reading the line status and modem status registers.

   —Disable the loopback mode.

   The INS8250, INS8250-B is now initialized for normal operation and the THRE and TSRE bits are reliable. This procedure can be used with the INS8250A, NS16450 and INS82C50A, although it is unnecessary.

   Recommendation B: Use one of the modem output lines to gate the transmitter data line.

3. When the transmitter interrupt is enabled, an interrupt occurs immediately regardless of the transmitter holding register's state. Furthermore, the first valid interrupt condition will probably be missed.

   Recommendation: Use the following procedure to solve this problem:

   —Wait for the transmitter holding register to empty.

   —Disable microprocessor interrupts.

   —Write to the interrupt enable register. The interrupt indication that would normally appear at this time will be cleared by a previously stored reset, if the IIR has been read prior to this

   **Note:** Whenever the IIR register is read and an active THRE interrupt will be cleared. If no THRE interrupt is active then the first THRE interrupt after the reading of IIR will be cleared).

   —Write to the interrupt enable register, again. Since there is no read of the IIR before this second write IER, there will be no stored reset to clear the normal THRE interrupt.

   —Enable microprocessor interrupts.

4. If data is not valid before and after WR or $\overline{WR}$ is active, then the bits of the internal register being addressed may change unpredictably. This could temporarily change any programmable UART function controlled by the addressed register. This situation exists because the INS8250, -B accepts data via fall-through latches that are enabled by the WR or $\overline{WR}$ going active rather than latched on the trailing edge of WR or $\overline{WR}$. Examples of this are glitches on the modem control lines or a temporary break on the serial output line while a command is written to the MCR or the LCR registers.

   Recommendation A: To avoid these problems the data must be valid just before, throughout and just after activation of WR or $\overline{WR}$.

   When using an 8088, 8086, 6800 or 8048 microprocessor, delay the leading edge of the write strobe until the data is stable. The above precaution is unnecessary when using the 8080, the NSC800™ or the Z80 microprocessors. Designs using a 32016 or 80286 should use the 16450, which avoids this problem by not having fall-through latches (see Section 3.0, Item 1).

   **Note:** The temporary break caused by a spurious glitch on LCR6 can also be avoided by setting the loopback mode prior to writing to the line control register.

5. The transmitter generates start bits longer than the rest of the data by approximately 1 μs. This is due to a look-ahead circuit that sends the start bit while data is being transferred from the transmitter holding register to the transmitter shift register. At 56 kB this causes a 6.25% error.

   Recommendation: Be aware that the last stop bit will be reduced by an equivalent amount of time (approximately 1 μs).

6. If the CPU is slow in servicing the UART it could read current status (LSR) and then the next data byte (RBR), instead of the current data byte. An example of this type of failure would be losing a received character without an overrun indication. This occurs when the CPU reads the receiver buffer when another character from the shift register was being transferred to it. UART registers are updated as soon as the received data is available (i.e., the receive buffer register is updated as soon as all of the data bits have been received, the parity flag is updated as soon as the parity bit is received, the overrun flag is updated as soon as the stop bit is received, etc.).

   Recommendation: The CPU must read the buffer sooner.

7. The transmitter character may be erroneous, if the INS8250, -B transmits with 5 data bits and 1 and ½ stop bits.

   Recommendation: Use only 1 stop bit.

8. Writing a "1" to bit 1 of the Interrupt Enable Register (IER1), when the Transmitter Holding Register is not empty sets the THRE interrupt, regardless of the THRE status bit condition.

   Recommendation: Only set bit 1 in the Interrupt Enable Register (IER1) if the Transmitter Holding Register is empty.

9. When multiple interrupts are pending, the interrupt line (INTR) pulses low after each interrupt instead of remaining high continuously.

   Recommendation: This will not cause problems in normal operation, however, it is a condition necessary for compatibility in some popular 8088-based microcomputers that use an edge-triggered ICU (see Section 6.0).

10. Bit No. 6 (TSRE) of the line status register is set as soon as the transmitter shift register empties whether or not the transmitter holding register contains a character. Bit No. 6 is then reset when the transmitter shift register is reloaded.

    Recommendation: This will not cause problems in normal operation. However, it is a function tested on some popular 8088-based microcomputer systems diagnostic programs.

### 2.1 ADDITIONAL FUNCTIONAL CONSIDERATIONS

When using the INS8250-B in full duplex operation with the THRE interrupt enabled and either one or both of the higher priority interrupts enabled (Receiver Data Available, Receiv-

er Line Status), the THRE interrupt indication may be lost. This is only possible if both data transmission and reception are occurring simultaneously. To avoid this problem use one of the three software aids listed below. The first two should be inserted in the Receiver Data Available and/or Receiver Line Status service routines. The last one should be inserted in the THRE service routine. Any of the following will result in successful operation given the above circumstance.

**SOFTWARE AIDS**

1. While inside the higher order interrupt service routines; test the THRE bit, if it is 1 then set IER1.
2. While inside the higher order interrupt service routines; test the THRE bit, if it is 1 then set a flag and service the transmitter as soon as you exit the routine.
3. Poll THRE (LSR5) instead of using the IIR.

## 3.0 INS8250A and NS16450 Function and Timing Considerations

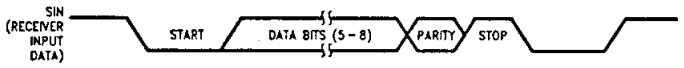
1. Chip select does not affect data transfers from the master register to the slave register. Therefore, the UART doesn't have to be deselected before it can offer valid status updates to the CPU.
2. The master reset (MR) input has a Schmitt Trigger circuit added to it.
3. A transmitter interrupt occurs only if the transmitter holding register is empty when bit 1 of the Interrupt Enable Register (IER) is set.
4. The UARTs latch data written to them on the trailing edge of the WR or $\overline{WR}$ signal, so data does not need to be valid for the total time write is active.
5. The loopback diagnostic function sets the modem control outputs $\overline{RTS}$, DTR, OUT1 and OUT2 to their inactive state (logic "1"), so they will send no spurious signals.
6. A one byte scratch pad register is included at location 111. This register is not on the INS8250 or -B.
7. When multiple interrupts are pending the interrupt line remains high rather than pulsing low after each interrupt is serviced. The INS8250A and NS16450 have level sensitive interrupts as opposed to edge-triggered interrupts. This requires a change in the UART driver software or associated hardware if the INS8250A, NS16450 is used with some popular microcomputers, and their edge-triggered ICUs (see Section 6.0).
8. Bit 6 of the line status register is set to 1 when both the transmitter holding and shift register are empty. This causes the INS8250A and NS16450 to be incompatible with some INS8250 software utilizing this bit.

**3.1 TIMING CONSIDERATIONS**

1. A start bit will be sent typically 16 clocks (1 bit time) after the WRTHR signal goes active.
2. The leading edge of WRTHR resets THRE and TEMT.
3. All of the line status errors and the received data flag (DR, data ready) are set during the time of the first stop bit.
4. TEMT is set 2 RCLK clock periods after the stop bit(s) are sent.
5. The modem control register updates the modem outputs on the trailing edge of WRMCR.

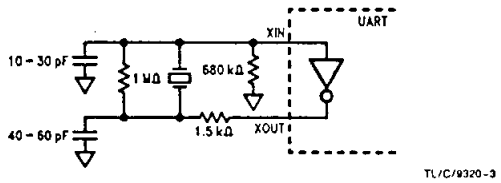
**3.2 CRYSTAL REQUIREMENTS**

There have been reports that certain types of 1.8432 MHz crystals have not been starting when used with the INS8250s (excluding the INS82C50A). The problem is with the smaller size versions of the crystal and their higher ESR values. In order to overcome this problem the following circuit should be used.




TL/C/9320–2

**FIGURE 2. Serial Data Timing**




TL/C/9320–3

**FIGURE 3. The Oscillator Circuit**

Crystal parameter for the above circuit are:

| | |
|---|---|
| type | AT cut |
| resonance | fundamental (parallel) |
| load capacitor | 20 pF – 32 pF |
| max. $R_S$ | 1k @ 1 MHz, 500 @ 5 MHz |
| cal. tolerance | +0.005% @ 25°C |
| drift tolerance | +0.005% @ 0°C – +70°C |
| overall tolerance | +0.01% |

### 3.3 ADDITIONAL FUNCTIONAL CONSIDERATIONS

When using either the INS8250A or the NS16450 in full duplex operation with the THRE interrupt enabled and either one or both of the higher priority interrupts enable (Receiver Data Available, Receiver Line Status), the THRE interrupt indication may be lost. This is only possible if both data transmission and reception are occurring simultaneously. To avoid this problem use one of the three following software aids. The first two should be inserted in the Receiver Data Available and/or Receiver Line Status service routines. The last one should be inserted in the THRE service routine. Any of the following will result in successful operation given the above circumstance.

SOFTWARE AIDS

1. Disable and then reenable transmitter interrupts via IER1 after the last time the IIR is read in higher order interrupt service routines.
2. While inside the higher order interrupt service routines; test the THRE bit, if it is 1 then set a flag and service the transmitter as soon as you exit the routine.
3. Poll THRE (LSR5) instead of using the IIR.

## 4.0 INS82C50A and NS16C450 Function and Timing Considerations

All of the information presented in Sections 3.0 through 3.2 is applicable to the CMOS parts. In addition, the following items specify differences between XMOS and CMOS parts. They are applicable to the CMOS parts only:

1. Anytime a reset pulse is issued to the INS82C50A or NS16C450 the divisor latches must be rewritten with the appropriate divisors in order to start the baud rate generator.
2. $t_{SI}$ is from 16 to 48 RCLK cycles in length.

## 5.0 NS16550A and NS16550 Function and Timing Considerations

All of the information present in Sections 3.0 and 3.1 is applicable to the NS16550A and NS16550.

The primary difference between these two parts is in the operation of the FIFOs. The NS16550 will sometimes transfer extra characters when the CPU reads the RX FIFO. Due to the asynchronous nature of this failure there is no workaround and the NS16550 should NOT be used in the FIFO mode. The NS16550A has no problems operating in the FIFO mode and should be used on all new designs.

The programmer should note the difference in the function of bit 6 in the Interrupt Identification Register (IIR6). This bit is permanently at logical 0 in the NS16550. In the NS16550A this bit will be set to a 1 when the FIFOs are enabled. In both parts bit 7 of the IIR is set to a 1 when the FIFOs are enabled. Therefore, the program can distinguish when the FIFOs are enabled and whether the part is an NS16550A or an NS16550 by checking these two bits. In order to enable the FIFO mode and set IIR6 and IIR7 bit 0 of the FIFO Control Register (FCR0) should be set. Remember unless both bits IIR6 and IIR7 are set, the program should not transfer data via the FIFOs.

The following are improvements in the AC timings for the NS16550A over the NS16450:

1. $t_{AR}$ changes from 60 ns to 30 ns.
2. $t_{CSW}$ changes from 50 ns to 30 ns.
3. $t_{CSR}$ changes from 50 ns to 30 ns.
4. RC changes from 360 ns to 280 ns.
5. $t_{RC}$ changes from 175 ns to 125 ns.
6. $t_{DS}$ changes from 40 ns to 30 ns.
7. $t_{DH}$ changes from 40 ns to 30 ns.
8. Timing parameters specified by $t_{SINT}$ will change in some cases when the FIFOs are enabled. Refer to the data sheet for specific changes.

## 6.0 Software Compatibility

The first part produced (INS8250-B) had some flaws and the first revision of that part (INS8250A) resolved those flaws. Between the time of the first part and the first revision, use of the INS8250-B in personal computers became quite common. Two of the conditions present in the INS8250-B are required in many of those personal computers (see Items 9 and 10 in Section 2.0). These two detect multiple pending interrupts from the INS8250-B and test the baud rate. These two conditions were eliminated in the revision part and all parts thereafter. Thus, the more recent UARTs require that one of the following recommendations or a similar change is made to the target system. Changing the software or hardware allows the more recent UARTs to replace the INS8250-B. If the target system services the UART via polling rather than interrupts, then all of the more recent parts will be plug-in replacements for the INS8250-B.

Note: The NS16550A has two pins with new functions (see the data sheet for specifics).

### 6.1 USING THE INS8250A, NS16450, INS82C50A, NS16C450 AND NS16550A WITH EDGED-TRIGGERED ICUs

Using these UARTs with an edge-triggered ICU as in some of the popular microcomputers requires a signal edge on the INTR pin for each pending UART interrupt. Otherwise, when multiple interrupts are pending the interrupt line will be constantly high active and the edge-triggered ICU will not request additional service for the UART.

### 6.2 CREATING AN INTERRUPT EDGE VIA SOFTWARE

This is done by disabling and then re-enabling UART interrupts via the Interrupt Enable Register (IER) before a specific UART interrupt handling routine (line status errors, received data available, transmitter holding register empty or modem status) is exited. To disable interrupts write H'00 to the IER. To re-enable interrupts write a byte containing ones to the IER bit positions whose interrupts are supposed to be enabled.

### 6.3 CREATING AN INTERRUPT EDGE IN HARDWARE

This is done externally to the UART. One approach is to connect the INTR pin of the UART to the input of an AND gate. The other input of this AND gate is connected to a signal that will always go low active when the UART is ac-

cessed (see *Figure 4*). The output of the AND gate is used as the interrupt to the ICU.

**Note:** This simple hardware recommendation will result in one invalid interrupt being generated, so the software routine should be able to handle this. The example shown below was tested using a modified asynchronous card in a few 8088-based microcomputer systems.

## 7.0 Acknowledgements

The editor expresses his gratitude to all of the applications, design and field applications engineers whose laboratory and field research have discovered most of the technical information used in this document.

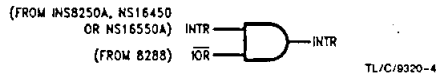



**FIGURE 4: Creating an INTR Edge In Hardware**

## AC Electrical Characteristics $T_A = 0°C$ to $+70°C$, $V_{CC} = 5V \pm 5\%$

| Symbol | Parameter | Conditions | NS16550A | | NS16450 NS16C450 | | INS8250A INS82C50A | | INS8250 | | INS8250-B | | Units |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{ADS}$ | Address Strobe Width | | 60 | | 60 | | 90 | | 90 | | 120 | | ns |
| $t_{AH}$ | Address Hold Time | | 0 | | 0 | | 0 | | 0 | | 60 | | ns |
| $t_{AR}$ | RD/$\overline{RD}$ Delay from Address | (Note 1) | 30 | | 60 | | 80 | | 110 | | 110 | | ns |
| $t_{AS}$ | Address Setup Time | | 60 | | 60 | | 90 | | 110 | | 110 | | ns |
| $t_{AW}$ | WR/$\overline{WR}$ Delay from Address | (Note 1) | 30 | | 60 | | 80 | | 160 | | 160 | | ns |
| $t_{CH}$ | Chip Select Hold Time | | 0 | | 0 | | 0 | | 0 | | 60 | | ns |
| $t_{CS}$ | Chip Select Setup Time | | 60 | | 60 | | 90 | | 110 | | 110 | | ns |
| $t_{CSC}$ | Chip Select Output Delay from Select | (Notes 1, 8) | | NA | | 100 | | 125 | | 200 | | 200 | ns |
| $t_{CSR}$ | RD/$\overline{RD}$ Delay from Chip Select | (Note 1) | 30 | | 50 | | 80 | | 110 | | 110 | | ns |
| $t_{CSS}$ | Chip Select Output Delay from Strobe | | | NA | | NA | | NA | 0 | 150 | 0 | 150 | ns |
| $t_{CSW}$ | WR/$\overline{WR}$ Delay from Select | (Note 1) | 30 | | 50 | | 80 | | 160 | | 160 | | ns |
| $t_{DH}$ | Data Hold Time | | 30 | | 40 | | 60 | | 60 | | 100 | | ns |
| $t_{DS}$ | Data Setup Time | | 30 | | 40 | | 90 | | 175 | | 350 | | ns |
| $t_{HZ}$ | RD/$\overline{RD}$ to Floating Data Delay | (Notes 3, 8) | 0 | 100 | 0 | 100 | 0 | 100 | 0 | 150 | 0 | 150 | ns |
| $t_{MR}$ | Master Reset Pulse Width | | 5 | | 5 | | 10 | | 25 | | 25 | | ns |
| $t_{RA}$ | Address Hold Time from RD/$\overline{RD}$ | (Note 1) | 20 | | 20 | | 20 | | 10 | | 10 | | ns |
| $t_{RC}$ | Read Cycle Delay | | 125 | | 175 | | 500 | | 1735 | | 1735 | | ns |
| $t_{RCS}$ | Chip Select Hold Time from RD/$\overline{RD}$ | (Note 1) | 20 | | 20 | | 20 | | 50 | | 50 | | ns |
| $t_{RD}$ | RD/$\overline{RD}$ Strobe Width | | 125 | | 125 | | 175 | | 175 | | 350 | | ns |
| $t_{RDA}$ | RD/$\overline{RD}$ Strobe Delay from $\overline{ADS}$ | | NA | | NA | | NA | | 0 | | 0 | | ns |
| $t_{RDD}$ | RD/$\overline{RD}$ Driver Enable/Disable | (Notes 3, 8) | | 60 | | 60 | | 75 | | 150 | | 250 | ns |
| $t_{RVD}$ | Delay from RD/$\overline{RD}$ to Data | (Note 8) | | 125 | | 125 | | 175 | | 250 | | 300 | ns |
| $t_{WA}$ | Address Hold Time from WR/$\overline{WR}$ | (Note 1) | 20 | | 20 | | 20 | | 50 | | 50 | | ns |
| $t_{WC}$ | Write Cycle Delay | | 150 | | 200 | | 500 | | 1785 | | 1785 | | ns |

**Note 1:** Applicable only when $\overline{ADS}$ is tied low.

**Note 3:** Charge and discharge time is determined by $V_{OL}$, $V_{OH}$ and the external timing.

**Note 8:** Loading of 100 pF.

NA = Not Applicable.

## AC Electrical Characteristics $T_A$ = 0°C to +70°C, $V_{CC}$ = 5V ±5% (Continued)

| Symbol | Parameter | Conditions | NS16550A | | NS16450 NS16C450 | | INS8250A INS82C50A | | INS8250 | | INS8250-B | | Units |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{WCS}$ | Chip Select Hold Time from WR/$\overline{WR}$ | (Note 1) | 20 | | 20 | | 20 | | 50 | | 50 | | ns |
| $t_{WDA}$ | WR/$\overline{WR}$ Delay from Address | | NA | | NA | | NA | | 50 | | 50 | | ns |
| $t_{WR}$ | WR/$\overline{WR}$ Strobe Width | | 100 | | 100 | | 175 | | 175 | | 350 | | ns |
| $t_{XH}$ | Duration of Clock High Pulse | (Note 4) | 55 | | 140 | | 140 | | 140 | | 140 | | ns |
| $t_{XL}$ | Duration of Clock Low Pulse | (Note 4) | 55 | | 140 | | 140 | | 140 | | 140 | | ns |
| RC | Read Cycle = $t_{AR}$ + $t_{DIW}$ + $t_{RC}$ | | 280 | | 360 | | 755 | | 2000 | | 2205 | | ns |
| WC | Write Cycle = $t_{DDA}$ + $t_{DDW}$ + $t_{WC}$ | | 280 | | 360 | | 755 | | 2100 | | 2305 | | ns |
| **BAUD GENERATOR** | | | | | | | | | | | | | |
| N | Baud Divisor | | 1 | $2^{16}-1$ | 1 | $2^{16}-1$ | 1 | $2^{16}-1$ | 1 | $2^{16}-1$ | 1 | $2^{16}-1$ | ns |
| $t_{BHD}$ | Baud Output Positive Edge Delay | (Note 8) | | 175 | | 175 | | 250 | | 250 | | 250 | ns |
| $t_{BLD}$ | Baud Output Negative Edge Delay | (Note 8) | | 175 | | 175 | | 250 | | 250 | | 250 | ns |
| $t_{HW}$ | Baud Output Up Time | (Note 5) | 75 | | 250 | | 250 | | 330 | | 330 | | ns |
| $t_{LW}$ | Baud Output Down Time | (Note 6) | 100 | | 425 | | 425 | | 425 | | 425 | | ns |
| **RECEIVER** (Note 2) | | | | | | | | | | | | | |
| $t_{RINT}$ | Delay from RD/$\overline{RD}$ (RD RBR/RDLSR) to Reset Interrupt | (Note 8) | | 1000 | | 1000 | | 1000 | | 1000 | | 1000 | ns |
| $t_{SCD}$ | Delay from RCLK to Sample Time | | | 2000 | | 2000 | | 2000 | | 2000 | | 2000 | ns |
| $t_{SINT}$ | Delay from Stop to Set Interrupt | | | 1 RCLK | | 1 RCLK | | 1 RCLK | | 2000 | | 2000 | ns |

**Note 1:** Applicable only when $\overline{ADS}$ is tied low.

**Note 2:** For the NS16550A in the FIFO Mode (FCR0 = 1) the trigger level and timeout interrupts, the receiver data available indication, the active RXRDY indication and the overrun error indication will be delayed 3 RCLKs. Status indicators (PE, FE, BI) will be delayed 3 RCLKs after the first byte has been received. For subsequently received bytes these indicators will be updated immediately after RDRBR goes inactive.

**Note 4:** The maximum external clock for the NS16550A is 8 MHz, NS16450 and INS8250A is 3.1 MHz and INS8250 and INS8250-B is 3.1 MHz. 100 pF load.

**Note 5:** The maximum external clock for the NS16550A is 8 MHz, NS16450 and INS8250A is 3.1 MHz and INS8250 and INS8250-B is 3.1 MHz. 100 pF load. This parameter is tested on the NS16550A and guaranteed by design on all other parts.

**Note 6:** The maximum external clock for the NS16550A is 8 MHz, NS16450 and INS8250A is 2.1 MHz and INS8250 and INS8250-B is 3.1 MHz. 100 pF load. This parameter is tested on the NS16550A and guaranteed by design on all other parts.

**Note 8:** Loading of 100 pF.

NA = Not Applicable.

## AC Electrical Characteristics $T_A$ = 0°C to +70°C, $V_{CC}$ = 5V ±5% (Continued)

| Symbol | Parameter | Conditions | NS16550A | | NS16450 NS16C450 | | INS8250A INS82C50A | | INS8250 | | INS8250-B | | Units |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| **TRANSMITTER** | | | | | | | | | | | | | |
| $t_{HR}$ | Delay from WR/$\overline{WR}$ (WR THR) to Reset Interrupt | (Note 8) | | 175 | | 175 | | 1000 | | 1000 | | 1000 | ns |
| $t_{IR}$ | Delay from RD/$\overline{RD}$ (RD IIR) to Reset Interrupt (THRE) | (Note 8) | | 250 | | 250 | | 1000 | | 1000 | | 1000 | ns |
| $t_{IRS}$ | Delay from Initial INTR Reset to Transmit Start | (Note 10) | 8 | 24 | 24 | 40 | 24 | 40 | | 16 | | 16 | Baudout Cycles |
| $t_{SI}$ | Delay from Initial Write to Interrupt | (Notes 7, 9) | 16 | 24 | 16 | 24 | 16 | 24 | | 50 | | 50 | Baudout Cycles |
| $t_{SS}$ | Delay from Stop to Next Start | | | NA | | NA | | NA | | 1000 | | 1000 | ns |
| $t_{STI}$ | Delay from Stop to Interrupt (THRE) | (Note 7) | 8 | 8 | 8 | 8 | 8 | 8 | | 8 | | 8 | Baudout Cycles |
| $t_{SXA}$ | Delay from Start to TXRDY Active | (Note 8) | | 8 | | NA | | NA | | NA | | NA | Baudout Cycles |
| $t_{WXI}$ | Delay from Write to TXRDY Inactive | (Note 8) | | 195 | | NA | | NA | | NA | | NA | ns |
| **MODEM CONTROL** | | | | | | | | | | | | | |
| $t_{MDO}$ | Delay from WR/$\overline{WR}$ (WR MCR) to Output | (Note 8) | | 200 | | 200 | | 1000 | | 1000 | | 1000 | ns |
| $t_{RIM}$ | Delay to Reset Interrupt from RD/$\overline{RD}$ (RD MSR) | (Note 8) | | 250 | | 250 | | 1000 | | 1000 | | 1000 | ns |
| $t_{SIM}$ | Delay to Set Interrupt from MODEM Input | (Note 8) | | 250 | | 250 | | 1000 | | 1000 | | 1000 | ns |

**Note 7:** This delay will be lengthened by 1 character time, minus the last stop bit time if the transmitter interrupt delay circuit is active.

**Note 8:** Loading of 100 pF.

**Note 9:** For both the NS16C450 and INS82C50A the value of $t_{SI}$ will range from 16 to 48 baudout cycles.

**Note 10:** For both the NS16C450 and the INS82C50A the value of $t_{IRS}$ will range from 24 to 40 baudout cycles.

NA = Not Applicable.

NOTE:

NOTE:

NOTE:

NOTE: