

# CSE



Department of Computer Science & Engineering

## Universal Asynchronous Receiver Transmitter (UART - 8251)

### CSCE 612 – Project #2 Specification

VLSI Design Lab  
Department of Computer Science and Engineering  
University of South Carolina

Dr. James P. Davis  
jimdavis@cse.sc.edu

Revision: v.04, 7 March 2003

## 1. UART Functionality

The UART is a universal asynchronous receiver/transmitter, which is modeled on the real-world Intel® 8251 peripheral interface adapter component. In the model we are considering, the UART consists of three main blocks.

- a serial transmit block
- a serial receive block and
- a CPU Interface (I/F) block.

The serial transmit block has two buffers (FIFO) into which data is written by the CPU I/F block. After the data is written into the buffers it is transmitted serially onto TXD. As long as the FIFO is not full the serial transmit block sets the signal TX\_RDY high.

The serial receive block has four buffers (FIFO). The block checks for the parity and the validity of the data frame on the RXD input and then writes correct data into its buffers. It also sets the signal RX\_RDY low if its FIFO is empty.

The CPU I/F block is responsible for reading the status register, data register and writing data into interrupt enable register and data register. It receives control signals from the CPU for performing certain tasks. The different functions for the set of control signals is given in a tabular form below.

D_XS	R/W	CPU I/F Function
0	R	Read status register
0	W	Write interrupt enable register
1	R	Receive data from receive block into data register.
1	W	Write data register & transmit data into transmit FIFO.

The XINT is asserted when there is an interrupt factor, i.e. atleast one of status register bits is asserted, and is also not masked by a corresponding bit in the interrupt enable register. Bits 0, 1, 2 of the interrupt enable register mask the bits 0, 1, 2 of the status register. The block diagram for the UART with its I/O ports and three main blocks is given below in Figure 1.

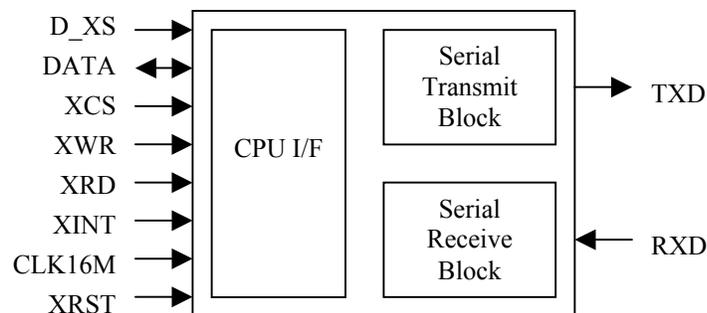
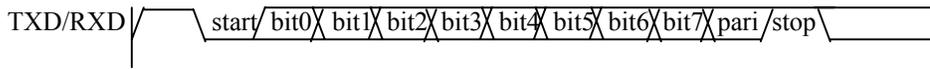


Figure 1. Basic UART block diagram.

The timing chart for the reading and writing operations, and the serial data format, are given below in Figure 2 (2.1 through 2.3).



*Pari* indicates parity bit.

Figure 2.1: Serial data format

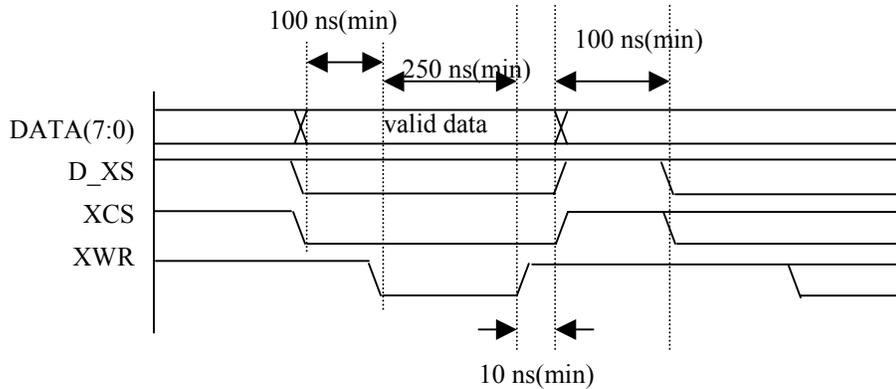


Figure 2.2: CPU Write timing

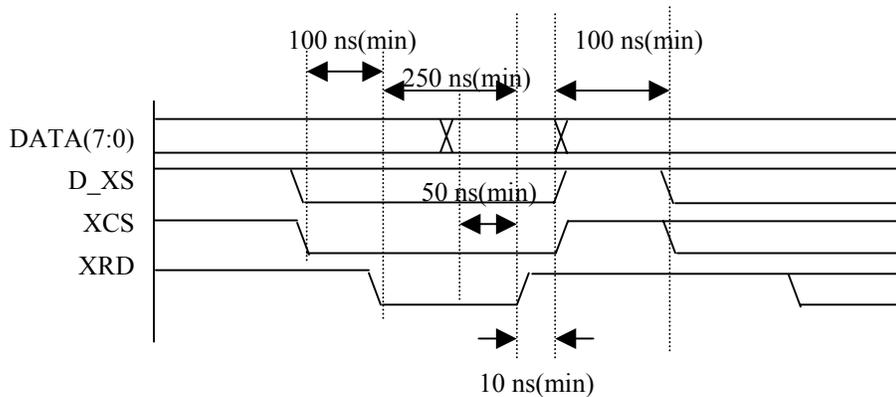


Figure 2.3: CPU Read timing

## 2. System partitioning and Component Description

The UART can be divided into several sub-components, according to different functionality. The description of each of these components is given next section. The block diagram depicting the more detailed component partitioning is shown in Figures 3 and 4.

The block diagram shows the different components. The D\_XS, XCS, DATA, XWR, XRD inputs are synchronized with the clock by their respective synchronizing blocks each of which register the signals twice.

Port	Type	Description	Width
D_XS	Input	CPU control signal	1
DATA	InOut	Data received from or sent to the CPU	8
XCS	Input	Chip Select	1
XWR	Input	CPU write control signal	1
XRD	Input	CPU read control signal	1
CLK16M	Input	16 MHz system clock	1
XRST	Input	System reset	1
XINT	Output	Interrupt Factor	1

Table 1. Pin Description of 8251 UART.

The CPU I/F registers the status register, interrupt enable register, and data register are modeled separately. Each of these components have DXS1, X\_WR/X\_RD as control signals. The transmit and receive FIFO's are separated from their corresponding control blocks the transmit and receive blocks. The RXD is passed through an IFF and the TXD goes through an OFF before being output. The data goes through an OFF before being written onto DATA output.

## 2.1. The Components

**DATASynch:** This component registers the DATA signal twice so as to synchronize it with the system clock CLK16M. The synchronized signal is data\_bus1.

**DXSSynch:** This component registers the D\_XS signal twice so as to synchronize it with the system clock CLK16M. The synchronized output is DXS1.

**XCSSynch:** This component registers the XCS signal twice so as to synchronize it with the system clock CLK16M. The synchronized output is XCS1.

**XWRSynch:** This component registers the XWR signal twice so as to synchronize it with the system clock CLK16M. The synchronized output is X\_WR.

**XRDSynch:** This component registers the XRD signal twice so as to synchronize it with the system clock CLK16M. The synchronized output is X\_RD.

**RXDIFF:** The RXD input is synchronized with the clock before being read by the receive block. The synchronized output is r\_xd.

**DATAOFF:** The data from the data register/status register is registered once before being written onto DATA output.

**Data Tristate Buffer:** This component drives the data bus output. It sets it to data\_bus2 when the XRD is asserted and to high impedance otherwise.

**XINTOFF:** The interrupt factor signal xintd is registered once before being output as XINT.

**TXDOFF:** The transmit data signal from the serial transmit block txd is synchronized with the clock before being output onto TXD output.

**Status Register:** This component represents the status of the UART. The register has TX\_RDY, RX\_RDY, PERR as its contents corresponding to bits 0, 1, 2 respectively. Its data is used to generate the interrupt factor xintd.

**Interrupt Enable Register:** The contents of this register are used to mask the interrupts the CPU does not want to process. Data on the data\_bus1 bus (bits 2 down to 0) is written into this register when both DXS1 and X\_WR are low. The XINT generator uses this register contents to mask the unwanted interrupts.

**XINT Generator:** This component generates the interrupt from the status register data and the interrupt enable register data. The equation for the interrupt signal xintd generation is as given below.

$$\begin{aligned} \text{xintd} \leq & \text{not}((\text{SR\_data}(0) \text{ and } (\text{not IER\_data}(0))) \\ & \text{or } (\text{SR\_data}(1) \text{ and } (\text{not IER\_data}(1))) \\ & \text{or } (\text{SR\_data}(2) \text{ and } (\text{not IER\_data}(2)))) \end{aligned}$$

**Transmit FIFO:** The FIFO is 8-bit by 2-word. It receives control signals from the serial transmit block. The data on signal data\_bus1 is written into its buffer when WRP is asserted. At the same time the write pointer is incremented. The data is read onto the stb\_fifo\_data signal when the stb\_fifo\_read is asserted. The stb\_fifo\_read\_inc asserted at the same time as stb\_fifo\_read, increments the read pointer by one. The read pointer is reset when the read pointer has reached its maximum. The write pointer is cleared when the write pointer has reached its maximum. The TX\_RDY is set low when the FIFO is full.

**Receive FIFO:** The FIFO is 8-bit by 4-word. It receives control signals from the serial receive block. The data received from the receive block, rec\_data is written into its buffer when srb\_fifo\_write is asserted. The srb\_fifo\_write\_inc asserted at the same time as srb\_fifo\_write, increments the write pointer by one.

The data is read onto the data\_bus2 signal when the XRD is asserted. The srb\_fifo\_read\_inc asserted at the same time as srb\_fifo\_read increments the read pointer by one. The read pointer is reset when the read pointer has reached its maximum value. The write pointer is cleared when the write pointer reaches its maximum limit before further increment. The RX\_RDY is asserted low when the FIFO is empty.

**Serial Transmit Block:** This component is responsible for serial transmission of data onto TXD. It generates the requisite control signals for reading and writing the transmit FIFO. This component can be divided into sub-components to make modeling easier. The block diagram for this is given below in Figure 3.

All the sub-components have XCS1 as chip enable and XRST as reset signals. The transmit clock counter counts the CLK16M clock cycles and sets the stb\_clk16 high after every 16 clock cycles. This signal is used as an enable by the transmit data counter, and the transmit block. The transmit data counter keeps count of the number of data bits transmitted onto tx\_d. The data count is incremented when stb\_dci is asserted and cleared when stb\_dcc is asserted. These signals are provided by the transmit control block. The parity counter counts the number of bits that were high in the eight bits of data being transmitted. The parity count is incremented on

assertion of stb\_pci and cleared on assertion of stb\_pcc. These two signals are provided by the transmit block.

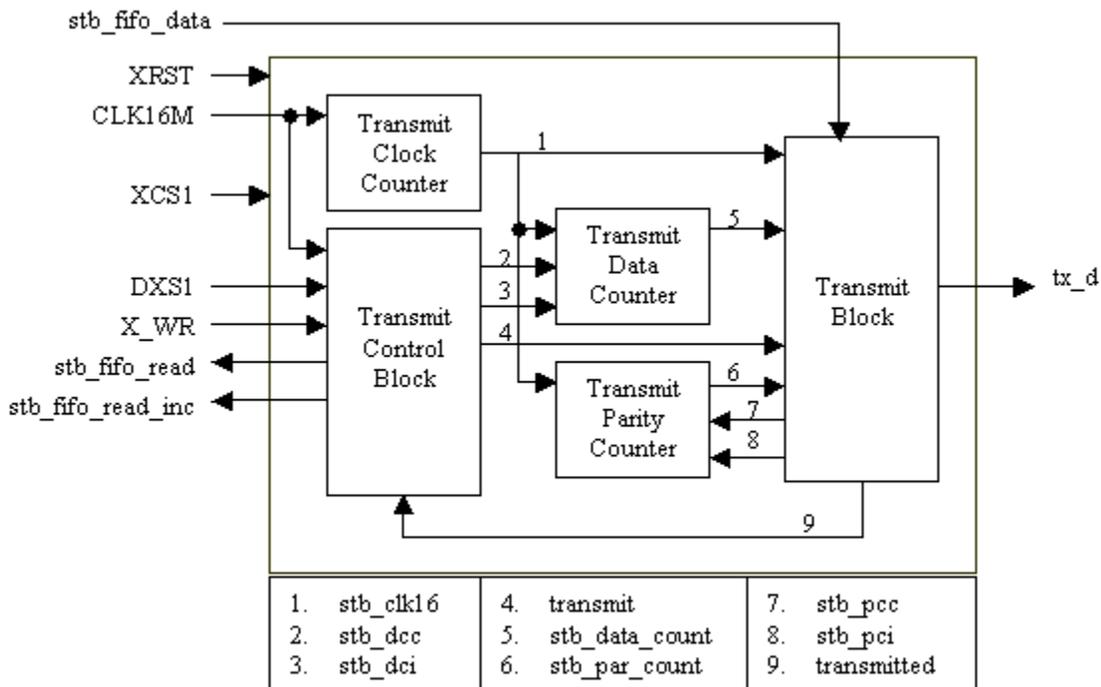


Figure 3. Block diagram of the Serial Transmit Block

The transmit control block controls the whole process of transmission. It is modeled in the form of a state machine. The state machine has three states namely: IDLE, FIFO\_READ, DATA\_TRANSMIT. Initially the machine is in the IDLE state. When DXS1 is high and XWR is low it jumps to FIFO\_READ state. In the FIFO\_READ state the data in the FIFO is read onto its output stb\_fifo\_data by setting stb\_fifo\_read and stb\_fifo\_read\_inc high. It then jumps to DATA\_TRANSMIT state. In DATA\_TRANSMIT state the transmit and the stb\_dci signals are asserted. The machine waits in this state until the signal transmitted is asserted by the transmit block and upon which it asserts stb\_dcc and goes back to IDLE state. When XRST is asserted it resets all its output signals.

The transmit block has stb\_clk\_16 as clock and XRST as asynchronous reset. It is enabled when transmit signal is asserted. It then transmits data serially onto the tx\_d depending upon the value of the stb\_data\_count. It sends the start bit when the count is less than 1. It then transmits the data bit by bit on every stb\_clk16 high until the count reaches 9. After this it sends the parity bit corresponding to the parity count stb\_par\_count. When the count becomes greater than 10 it transmits stop bit and asserts transmitted signal.

**Serial Receive Block:** This component is responsible for serial receiving of data on RXD. It generates the requisite control signals for reading and writing the receive FIFO. This component can be further divided into sub-components to make modeling easier. The block diagram for this is given below in Figure 4.



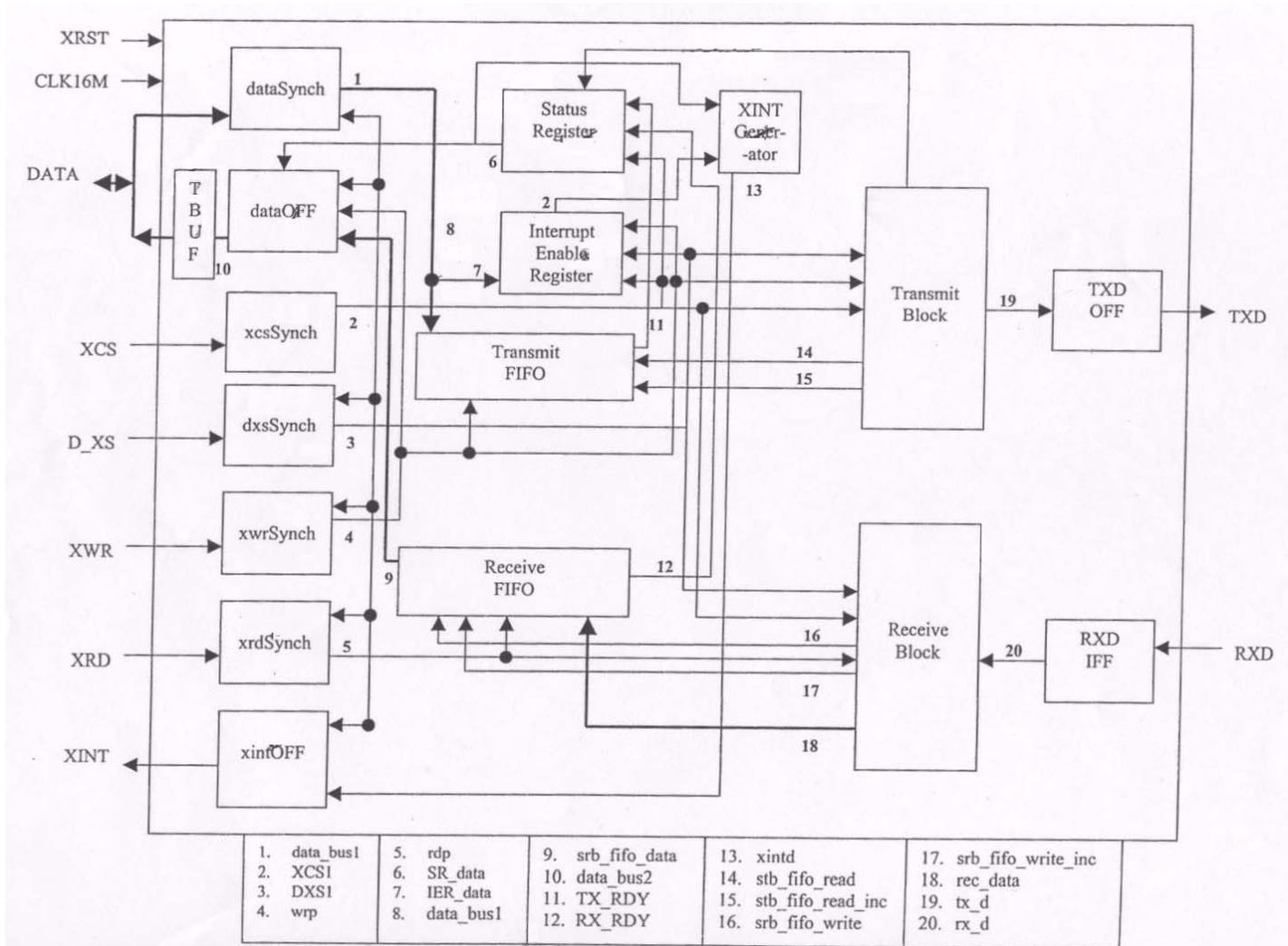


Figure 5. Detailed block diagram of the UART.

The receive control block controls the whole receiving process. It is modeled in the form of a state machine. The state machine has four states namely: IDLE, FIFO\_WRITE, FIFO\_READ, DATA\_RECEIVE. Initially the machine is in the IDLE state. In this state when the start bit is received on rx\_d it jumps to DATA\_RECEIVE state. In the DATA\_RECEIVE state, receive and the srb\_dci signals are asserted. The machine waits in this state until the signal received is asserted by the receive block. When received is asserted it checks for PERR and x\_fre before it asserts srb\_dcc and jumps to FIFO\_WRITE state. If PERR is high or x\_fre is low it jumps to IDLE state instead. In the FIFO\_WRITE state it asserts the srb\_fifo\_write and srb\_fifo\_write\_inc signals and then jumps to IDLE state so that it remains in the FIFO\_WRITE state for only one clock cycle. This ensures that the data is written in only one of the buffers, as the FIFO read and write processes are clock sensitive.

The receive block has srb\_clk\_16 as clock and XRST as asynchronous reset. It is enabled when receive signal is asserted. It then receives data serially from rx\_d depending upon the value of the srb\_data\_count. It receives data bit by bit into rec\_data on every srb\_clk16 high until the data count reaches 8. After this it receives the parity bit. When the count becomes greater than 8 it checks for the stop bit.

If the stop bit is not received the signal `x_fre` is set indicating a wrong frame of data has been received. At the same time it also checks for the parity by checking whether the parity count `srb_par_count` corresponds to the parity bit received. If there is a clash it sets `PERR` high indicating parity error. When the data count reaches 10 it asserts `received` indicating that the serial data has been received.

### 3. Project Milestones and Deliverables

The time duration of this project is 2 ½-3 weeks. The deliverables are to be provided by the student teams (consisting of 2 team members per team) in 2 separate milestones: (1) definition of simulation model for the various modules, along with simulation results, and (2) definition of the system model and the system test bench, along with simulation data.

We will take some class time to go over this design and the modeling. You might want to obtain more detailed information regarding the definition of the 8251 UART model from a data book. (You might find a suitable specification of the more detailed functionality out on the web, as this project is based on the Intel® discrete IC of the same designation).

### 4. Applications of the Intel® 8251

The 8251 was originally created as a discrete IC component back in 1975, but continues to be used in a number of applications—now as a small IP core model. It continues to be written about as a standard serial interface component in architectures based on 8-bit, 16-bit and 32-bit Intel® microprocessors. Some of the figures that follow are from Antonakos, 3<sup>rd</sup> edition, who uses the device for interfacing with the older microprocessors. These figures, taken from various locations on other course websites, are from this text (which we don't use here at USC).

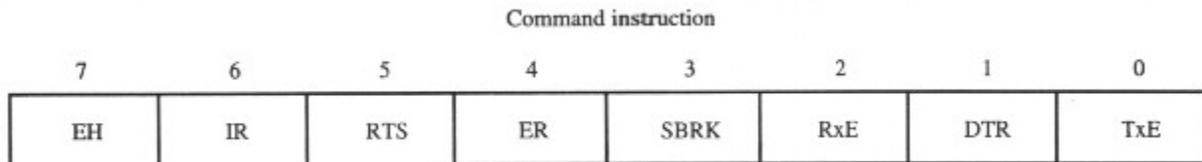


Figure 6. Programming Interface to the 8251 – Command Word. (source, Antonakos, 3<sup>rd</sup> ed., 1999)

The 8251 PIA/UART is a programmable device that can be set up under program control. The 8-bit data word port also doubles as a command and status port, depending on whether the mode of the device is set for control or data.

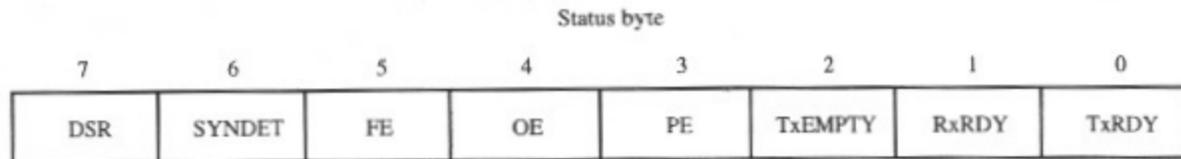


Figure 7. Programming Interface to the 8251 – Status Word. (source, Antonakos, 3<sup>rd</sup> ed., 1999)

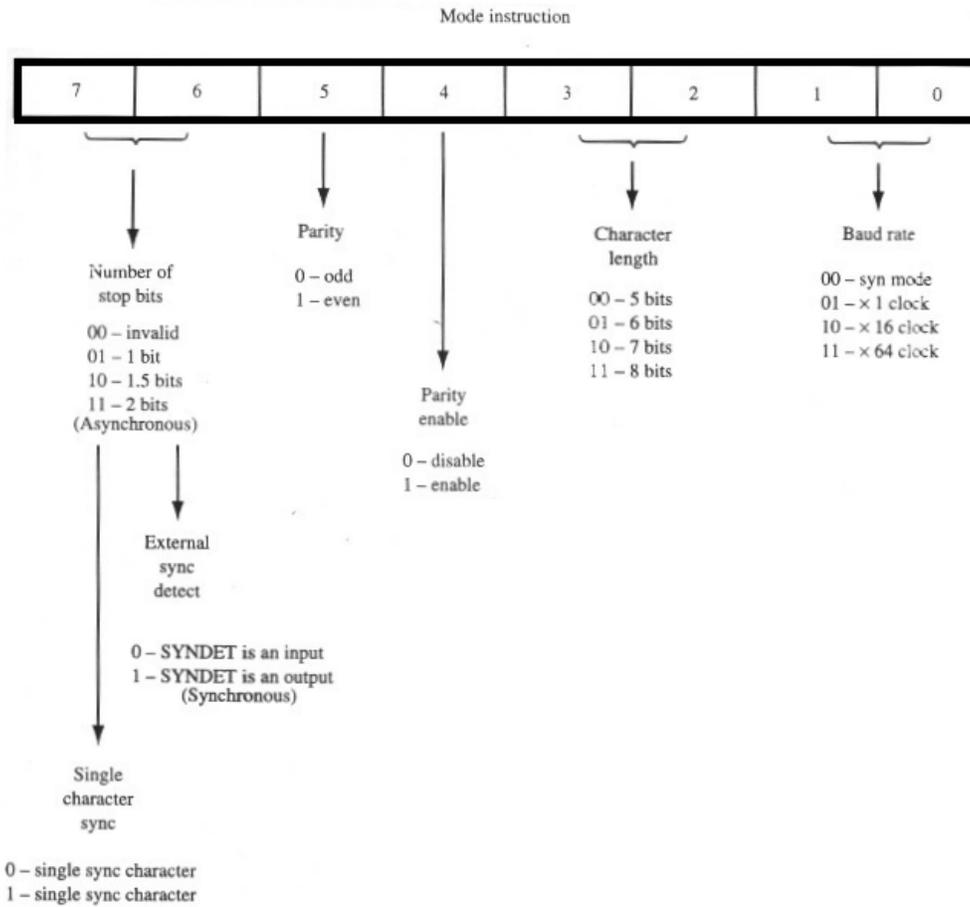


Figure 8. Programming Interface to the 8251 – Mode Word. (source, Antonakos, 3<sup>rd</sup> ed., 1999)

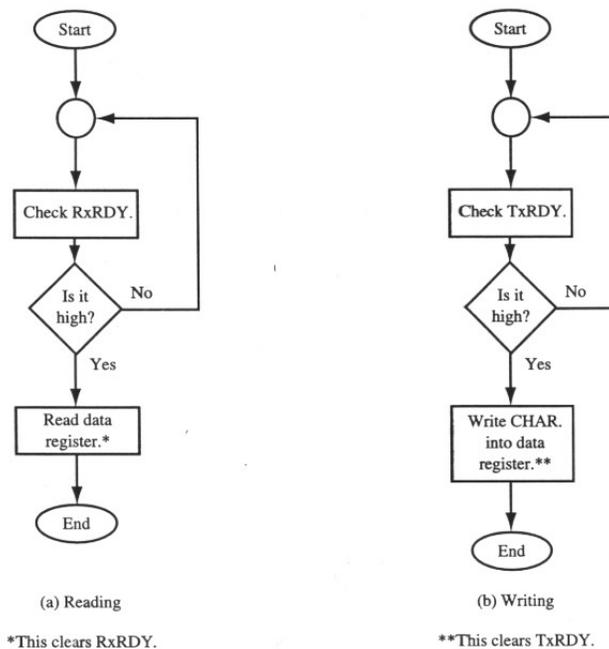


Figure 9. Programming Interface to the 8251 – Polling Operations. (source, Antonakos, 3<sup>rd</sup> ed., 1999)



## 5. References

1. H. Kobayashi, Universal Asynchronous Receiver Transmitter (UART – 8251), Project Specification for EECE 611, Fall 1997.
2. James L. Antonakos, *An Introduction to the Intel Family of Microprocessors: A Hands-on Approach Utilizing the 80x86 Microprocessor Family*, 3<sup>rd</sup> Edition, Prentice-Hall Publishers, Inc., 1999.
3. Intel Corporation, *Peripheral Devices Data Book*, 1987.