UNIVERSAL ASYNCHRONOUS RECEIVER/TRANSMITTER - UART-RS232

The Universal ***Asynchronous Receiver/Transmitter (UART)*** controller is the key component of the serial communications subsystem of a microprocessors, microcontrollers and computers. The UART can take bytes of data in parallel fashion and transmits the individual bits in a sequential fashion. At the same time, a second UART can receive serial bits sent by UART and convert the bits into complete bytes. There are two types of serial transmission: Synchronous and Asynchronous. The asynchronous communication is known as UART and asynchronous is known as ***Universal Synchronous-Asynchronous Receiver/Transmitter (USART)***. Latest standard UART employ ***FIFO*** buffer for improved functional capability. UARTs are generally used to modem control functions. Hence they have several functional control registers using which efficient interface can be established with host processor. Embedded processors employ built in hard core UART block while soft IP UART provide programmable and reconfigurable flexibilities which are very much advantages in FPGA applications. Hardware Descriptive Languages (HDL) like verilog can be used for the behavioral description of the UART.

The UART device changes incoming parallel data to serial data which can be sent on a communication line. A second **UART** is used to receive the information which converts serial data to parallel. The **UART** performs all the tasks, timing, parity checking, etc. needed for the communication. The UART requires external line drivers (***EIA RS 232 C*** interface) to interface to external world. In computer systems, the UART is connected to circuitry that generates signals that comply with the EIA RS232-C specification. There is also a ***CCITT*** standard named ***V.24*** that resembles the specifications included in RS232-C. Registers are accessible to set or review the communication parameters to use the **UART** in different environments. Using these registers the communication speed (***baud rate***), the type of parity check, and the way incoming information is signaled to the running software are set according to the requirement of host processor.

Serial UART types

PC compatible serial communication started with the **8250 UART** in the **IBM XT machines**. Then UART is upgraded to **8250A, 8250B** and then **16450 (**manufactured by National Semiconductor) which is implemented in the **AT** machines. The higher bus speed could not be reached by the **8250** series but newer **16450** were capable of handling a communication speed of 38.4 kbs. **16450** had 1 byte FIFO. The later improved version**16550A** contained two on-board ***FIFO buffers***, each capable of storing 16 bytes. One buffer for transmitter and one buffer for receiver. This made it possible to increase maximum reliable communication speeds to 115.2 kbs and use effectively in modems with on-board compression. DMA access ability is provided in **16550**. Two pins were redefined for this purpose. **DMA** transfer is not used with most applications. The most common **UART** used is**16550A**. Newer versions such as **16650** contain two 32 byte FIFO's and on board support for software flow control are latest advancements in industry. Texas Instruments is developing the **16750** which contains 64 byte FIFO's.

A UART usually contains the following components:

- **Baudrate clock generator**: Multiple of the bit rate to improve sampling in the middle of a bit period. For generating this timing information, each **UART** uses an oscillator generating a frequency of about 1.8432 MHz. This frequency is divided by 16 to generate the time base for communication. Hence the maximum allowed communication speed is 115200 bps. **UART**s like the **16550** are capable of handling higher input frequencies up to 24 MHz which makes it possible to communicate with a maximum speed of 1.5 Mbps.
- **Input and output shift registers**: Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms. These registers shifts the data that has to be serially transmitted or serially received.
- **Transmit and receive control**: This control logic checks for the control signals from host processor to start or stop the transmission and reception of the data bits. In case of any error it also generates error signals.
- **Optional transmit and receive buffers:** Buffers can be used to hold the data temporarily.
- **Optional parallel data bus buffer:** This buffer improves the speed.
- **Optional FIFO:** The UART works by writing data from the host processor to its FIFO buffers, and feeding the data from the buffer to the serial device in the format dictated by the user (typically 8-N-1).

Serial Data Format and Asynchronous Serial Transmission

As the name indicates, asynchronous transmission need not send clock signal to send the data to the receiver. The sender and receiver must agree on timing parameters in advance and special bits such as start and stop bits are added to each word which is used to synchronize the sending and receiving units. The UART serial data format is shown in Figure (1).

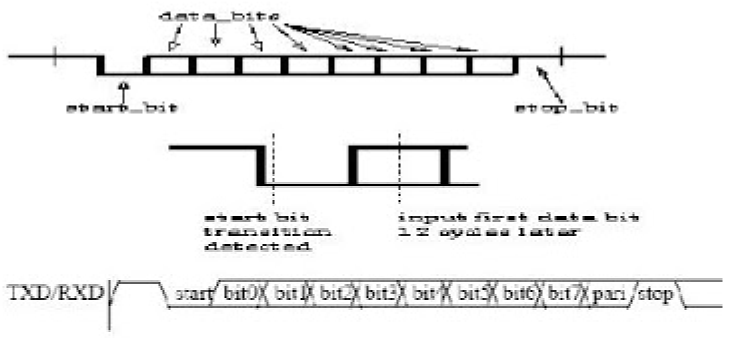

Figure (1) Serial Data Format

A bit called the "**Start Bit**" is added to the beginning of each word that is to be transmitted. The **Start Bit** indicates the start of the data transmission and it alerts the receiver that a word of data is about to be sent. Upon reception of start bit the clock in the receiver goes into synchronization with the clock in the transmitter. The accuracy of these two clocks should not deviate more than 10% during the transmission of the remaining bits in the word.

The individual bits of the word of data are sent after the start bit. Least Significant Bit (LSB) is sent first. The transmitter does not know when the receiver has read at the value of the bit. The transmitter begins transmitting the next bit of the word on next clock edge.

Parity bit is be added when the entire data word has been sent. This bit can be used to detect errors at the receiver side. Then one **Stop Bit** is sent by the transmitter to indicate the end of the valid data bits.

On the receiver side once it receives all of the bits in the data word, it can check for the***Parity Bits***. To accomplish this task both transmitter and receiver must agree on whether a Parity Bit is to be used. Then ***Stop Bit*** is encountered by receiver. A missing ***stop bit*** may result entire data to be garbage. This will cause a ***Framing Error*** and will be reported to the host processor when the data word is read. ***Framing Error*** can be caused due to mismatch of transmitter and receiver clocks.

The UART automatically discards the Start, Parity and Stop bits irrespective of whether data is received correctly or not. If the sender and receiver are configured identically, these bits are not passed to the host. To transmit new word, the Start Bit for the new word is sent as soon as the Stop Bit for the previous word has been sent.

The transmission speed in asynchronous communication is measured by ***Baud Rate***. A Baud Rate represents the number of bits that are actually being sent over the media. The Baud rate includes the Start, Stop and Parity bits. The ***Bit rate*** (***Bits per Second-bps***) represents the amount of data that is actually sent from the transmitting device to the other device. Speeds for UARTs are in bits per second (bit/s or bps), although often incorrectly called the baud rate. Standard baud rates are: 110, 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400, 460800, 921600, 1382400, 1843200 and 2764800 bit/s.

UART Registers

Twelve registers control the communication between the processor and the **UART.**Behavior of the communication can be changed by reading or writing registers. Each register is eight bits wide. On PC compatible devices, the registers are accessible in the I/O address area. The function of each register is discussed here in brief. The registers are shown in Figure

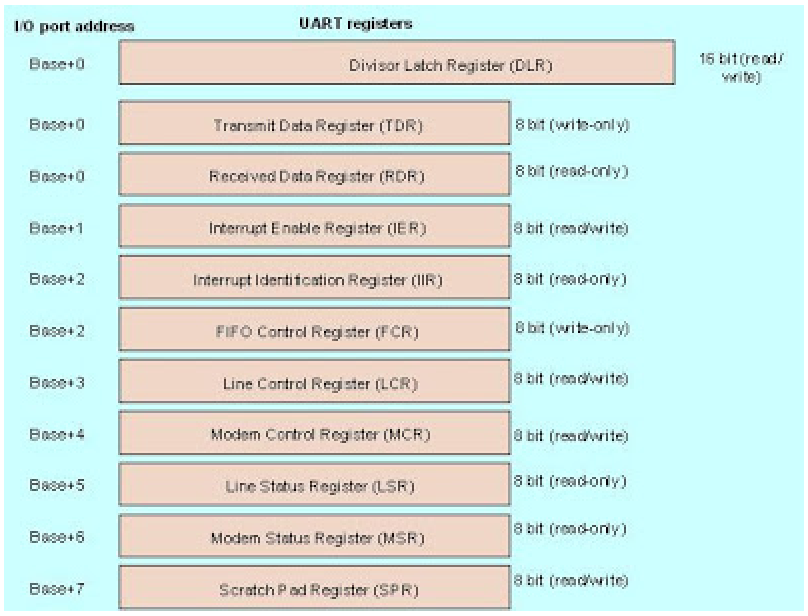
I/O port address
UART registers
Base+0
Divisor Latch Register (DLR)
16 bit (read/write)
Base+0
Transmit Data Register (TDR)
8 bit (write-only)
Base+0
Received Data Register (RDR)
8 bit (read-only)
Base+1
Interrupt Enable Register (IER)
8 bit (read/write)
Base+2
Interrupt Identification Register (IIR)
8 bit (read-only)
Base+2
FIFO Control Register (FCR)
8 bit (write-only)
Base+3
Line Control Register (LCR)
8 bit (read/write)
Base+4
Modem Control Register (MCR)
8 bit (read/write)
Base+5
Line Status Register (LSR)
8 bit (read-only)
Base+6
Modem Status Register (MSR)
8 bit (read-only)
Base+7
Scratch Pad Register (SPR)
8 bit (read/write)

Figure (2) The 16550 UART registers

RBR: Receiver buffer register

The ***Receiver Buffer Register (RBR)*** contains the byte received if no FIFO is used, or the oldest unread byte with FIFO's. If FIFO buffering is used, each new read action of the register will return the next byte, until no more bytes are present. Bit 0 in the ***Line Status Register (LSR)*** can be used to check if all received bytes have been read. This bit will change to zero if no more bytes are present.

THR: Transmitter holding register

Transmitter Holding Register (THR) is used to buffer outgoing characters. Without FIFO buffering, only one character can be stored. Otherwise the amount of characters depends on the type of **UART**. To check if new information must be written to **THR** Bit 5 in the***Line Status Register (LSR)*** can be used. Empty register is indicated by the value 1. If FIFO buffering is used, more than one character can be written to the transmitter holding register when the FIFO is empty.

IER: Interrupt enable register

In interrupt driven configuration, the **UART** will signal each change by generating a processor interrupt. A software routine must be read interrupt signal to handle the interrupt and to check what state change was responsible for it. ***Interrupt enable register (IER)*** is used to enable the interrupt.

IIR: Interrupt identification register

The ***Interrupt Identification Register (IIR)*** bits show the current state of the **UART** and which state change caused the interrupt to occur. Based on bit values of the IIR interrupt can be serviced.

FCR: FIFO control register

The ***FIFO control register (FCR)*** is present starting with the **16550** series. The behavior of the FIFOs in the **UART** is controlled by this register. If a logical value 1 is written to bits 1 or 2, the function attached is triggered. The other bits are used to select a specific FIFO mode.

LCR: Line control register

The ***Line Control Register (LCR)*** is used at initialization to set the communication parameters such as parity, number of data bits etc. The register also controls the accessibility of the **DLL** and **DLM** registers.

MCR: Modem control register

Handshaking actions with the attached device are accomplished by the ***Modem Control Register (MCR).*** In the UART series **16550**, setting and resetting of the control signals must be done by software. But in the new **16750**, flow control automatically handled.

LSR: Line status register

The ***Line Status Register (LSR)*** shows the current state of communication. Errors, the state of the receiver and transmit buffers are available.

MSR: Modem status register

The ***Modem Status Register (MSR)*** contains information about the four incoming modem control lines on the device. The four most significant bits contain information about the current state of the inputs. The least four significant bits are used to indicate state changes. Each time the register is read the four LSB's are reset.

DLL and DLM: Divisor latch registers

The communication speed of the UART is changed by using a programmable value stored in **Divisor Latch Registers *DLL*** and ***DLM*** which contains the least and most significant registers.

UART Transmitter

The block representation of serial data transmission is depicted in Figure (3).The serial transmit block has FIFO buffer into which data is written by the host processor. After the data is written into the buffers it is transmitted serially onto ***tx***. As long as the FIFO is not full the serial transmit block sets the signal ***tx_en*** high.

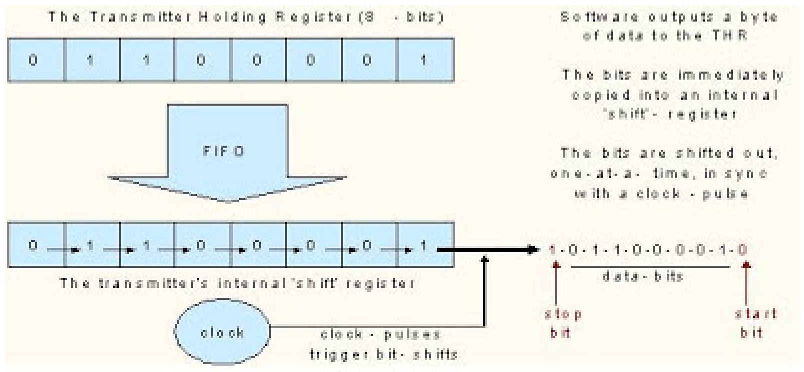
The Transmitter Holding Register (8 - bits)
0 1 1 0 0 0 0 1
FIFO
Software outputs a byte of data to the THR
The bits are immediately copied into an internal 'shift' - register
The bits are shifted out, one-at-a- time, in sync with a clock - pulse
0 1 1 0 0 0 0 1
The transmitter's internal 'shift' register
1-0-1-1-0-0-0-0-1-0
data-bits
stop bit
start bit
clock
clock - pulses trigger bit- shifts

Figure (3) Serial data transmission

Transmit FIFO

The FIFO is 8-bit by 32-word. It receives control signals from the serial transmit block. The data on signal data_bus is written into its buffer. At the same time the write pointer is incremented. The data is read onto FIFO and 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_en*** is set low when the FIFO is full.

Serial Transmit Block

This component is responsible for serial transmission of data onto ***tx***. It generates the requisite control signals for reading and writing the transmit FIFO. 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***. 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 transmit control block controls the whole process of transmission. It is modeled in the form of a state machine.

UART-Receiver

The block representation of serial data reception is depicted in Figure (4).The serial receive block can also has a FIFO buffers. The block checks for the parity and the validity of the data frame on the ***rx*** input and then writes correct data into its buffers. It also sets the signal ***byte_ready*** low if its FIFO is empty.

Receive FIFO

The FIFO is 8-bit wide and 32 byte deep. It receives control signals from the serial receive block. The data received from the receive block written into its buffer. The write pointer is cleared when the write pointer reaches its maximum limit before further increment.

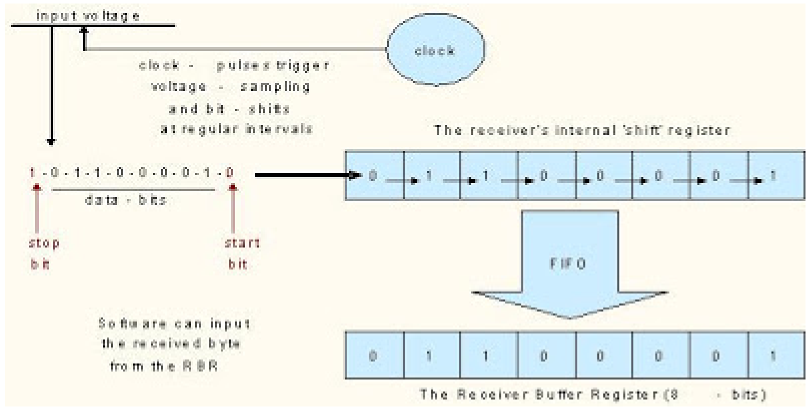
input voltage
clock
clock - pulses trigger voltage - sampling and bit - shifts at regular intervals
The receiver's internal 'shift' register
1 - 0 - 1 - 1 - 0 - 0 - 0 - 0 - 1 - 0
data - bits
stop bit
start bit
0 1 1 0 0 0 0 1
FIFO
Software can input the received byte from the RBR
0 1 1 0 0 0 0 1
The Receiver Buffer Register (8 - bits)

Figure (4) Serial data reception

Serial Receive Block

Serial data is received by this component. It generates the requisite control signals for reading and writing the receive FIFO. It generates required sample clock to sample the incoming data and determine the baud rate of the incoming data.

UART Errors

Overrun Error

An "**overrun error**" occurs when the UART cannot process the byte that just came in before the next one arrives. The host processor must service the UART in order to remove characters from the buffer. If the host processor does not service the UART and the buffer becomes full, then ***Overrun Error*** will occur.

Framing Error

A "**Framing Error**" occurs when the designated "start" and "stop" bits are not valid. Start bit acts as a reference for the remaining bits. When the "stop" bit is expected if the data line is not in the expected idle state a ***Framing Error*** will occur.

Parity Error

A "**Parity Error**" occurs when the number of "active" bits does not agree with the specified parity configuration of the UART.

External Interface

The external signalling levels that are used between different equipment are not generated by UART. An interface is used to convert the logic level signals of the UART to the external signalling levels. Examples of standards for voltage signalling are **RS-232, RS-422**and **RS-485** from the **EIA**. For embedded system applications UARTs are commonly used with RS-232. It is useful to communicate between microcontrollers and also with PCs. **MAX 232** is one of the example ICs which provide RS232 level signals.

Source : http://asic-soc.blogspot.in/2007/12/universal-asynchronous.html