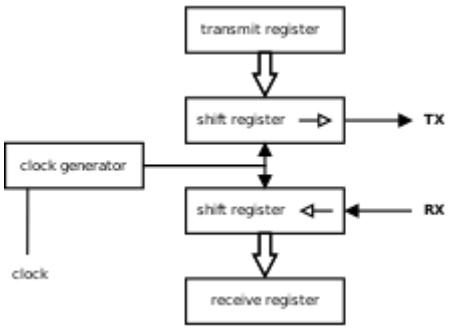
Universal asynchronous receiver-transmitter

A **universal asynchronous receiver-transmitter** (**UART** /ˈjuːɑːrt/) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel. The electric signaling levels are handled by a driver circuit external to the UART. Two common signal levels are RS-232, a 12-volt system, and RS-485, a 5-volt system. Early teletypewriters used current loops.



Block diagram for a UART

It was one of the earliest computer communication devices, used to attach teletypewriters for an operator console. It was also an early hardware system for the Internet.

A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. One or more UART peripherals are commonly integrated in microcontroller chips. Specialised UARTs are used for automobiles, smart cards and SIMs.

A related device, the universal synchronous and asynchronous receiver-transmitter (USART) also supports synchronous operation.

Contents

Transmitting and receiving serial data

The universal asynchronous receiver-transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion.[1] At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires.

The UART usually does not directly generate or receive the external signals used between different items of equipment. Separate interface devices are used to convert the logic level signals of the UART to and from the external signalling levels, which may be standardized voltage levels, current levels, or other signals.

Communication may be 3 modes:

- *simplex* (in one direction only, with no provision for the receiving device to send information back to the transmitting device)
- *full duplex* (both devices send and receive at the same time)
- *half duplex* (devices take turns transmitting and receiving)

Data framing

For UART to work the following settings need to be the same on both the transmitting and receiving side:

- Baud Rate
- Parity bit
- Data bits size
- Stop bits size
- Flow Control

UART frame, field length in Bits

1	5-9	0-1	1-2
Start Bit	Data Frame	Parity Bits	Stop Bits

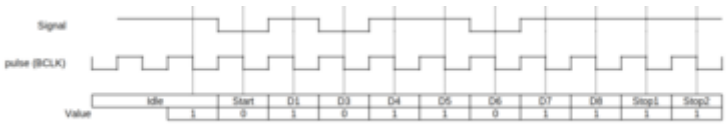
In the most common settings of 8 data bits, no parity and 1 stop bit (aka 8N1), the protocol efficiency is 80%. Ethernet by comparison is up to 97%.

A UART frame consists of 5 elements:

- Idle (logic high (1))
- Start bit (logic low (0))
- Data bits
- Parity bit
- Stop (logic high (1))

The idle, no data state is high-voltage, or powered. This is a historic legacy from telegraphy, in which the line is held high to show that the line and transmitter are not damaged.

Each character is framed as a logic low start bit, data bits, possibly a parity bit and one or more stop bits. In most applications the least significant data bit (the one on the left in this diagram) is transmitted first, but there are exceptions (such as the IBM 2741 printing terminal).



Example of a UART frame. In this diagram, one byte is sent, consisting of a start bit, followed by eight data bits (D0-7), and two stop bit, for a 11-bit UART frame. The number of data and formatting bits, the presence or absence of a parity bit, the form of parity (even or odd) and the transmission speed must be pre-agreed by the communicating parties. The "stop bit" is actually a "stop period"; the stop period of the transmitter may be arbitrarily long. It cannot be shorter than a specified amount, usually 1 to 2 bit times. The receiver requires a shorter stop period than the transmitter. At the end of each data frame, the receiver stops briefly to wait for the next start bit. It is this difference which keeps the transmitter and receiver synchronized. BCLK = Base Clock

Start bit

The start bit signals the receiver that a new character is coming.

Data bit

The next five to nine bits, depending on the code set employed, represent the character.

Parity bit

If a parity bit is used, it would be placed after all of the data bits.

It describes the odd or eveness of the number.

Stop bit

The next one or two bits are always in the **mark** (logic high, i.e., '1') condition and called the stop bit(s). They signal to the receiver that the character is complete. Since the start bit is logic low (0) and the stop bit is logic high (1) there are always at least two guaranteed signal changes between characters.

If the line is held in the logic low condition for longer than a character time, this is a break condition that can be detected by the UART.

Receiver

All operations of the UART hardware are controlled by an internal clock signal which runs at a multiple of the data rate, typically 8 or 16 times the bit rate. The receiver tests the state of the incoming signal on each clock pulse, looking for the beginning of the start bit. If the apparent start bit lasts at least one-half of the bit time, it is valid and signals the start of a new character. If not, it is considered a spurious pulse and is ignored. After waiting a further bit time, the state of the line is again sampled and the resulting level clocked into a shift register. After the required number of bit periods for the character length (5 to 8 bits, typically) have elapsed, the contents of the shift register are made available (in parallel fashion) to the receiving system. The UART will set a flag indicating new data is available, and may also generate a processor interrupt to request that the host processor transfers the received data.

Communicating UARTs have no shared timing system apart from the communication signal. Typically, UARTs resynchronize their internal clocks on each change of the data line that is not considered a spurious pulse. Obtaining timing information in this manner, they reliably receive when the transmitter is sending at a slightly different speed than it should. Simplistic UARTs do not do this; instead they resynchronize on the falling edge of the start bit only, and then read the center of each expected data bit, and this system works if the broadcast data rate is accurate enough to allow the stop bits to be sampled reliably.[2][3]

It is a standard feature for a UART to store the most recent character while receiving the next. This "double buffering" gives a receiving computer an entire character transmission time to fetch a received character. Many UARTs have a small first-in, first-out (FIFO) buffer memory between the receiver shift register and the host system interface. This allows the host processor even more time to handle an interrupt from the UART and prevents loss of received data at high rates.

Transmitter

Transmission operation is simpler as the timing does not have to be determined from the line state, nor is it bound to any fixed timing intervals. As soon as the sending system deposits a character in the shift register (after completion of the previous character), the UART generates a start bit, shifts the required number of data bits out to the line, generates and sends the parity bit (if used), and sends the stop bits. Since full-duplex operation requires characters to be sent and received at the same time, UARTs use two different shift registers for transmitted and received characters. High performance UARTs could contain a transmit FIFO (first in first out) buffer to allow a CPU or DMA controller to deposit multiple characters in a burst into the FIFO rather than have to deposit one character at a time into the shift register. Since transmission of a single or multiple characters may take a long time relative to CPU speeds, a UART maintains a flag showing busy status so that the host system knows if there is at least one character in the transmit buffer or shift register; "ready for next character(s)" may also be signaled with an interrupt.

Application

Transmitting and receiving UARTs must be set for the same bit speed, character length, parity, and stop bits for proper operation. The receiving UART may detect some mismatched settings and set a "framing error" flag bit for the host system; in exceptional cases, the receiving UART will produce an erratic stream of mutilated characters and transfer them to the host system.

Typical serial ports used with personal computers connected to modems use eight data bits, no parity, and one stop bit; for this configuration, the number of ASCII characters per second equals the bit rate divided by 10.

Some very low-cost home computers or embedded systems dispense with a UART and use the CPU to sample the state of an input port or directly manipulate an output port for data transmission. While very CPU-intensive (since the CPU timing is critical), the UART chip can thus be omitted, saving money and space. The technique is known as bit-banging.

History

Some early telegraph schemes used variable-length pulses (as in Morse code) and rotating clockwork mechanisms to transmit alphabetic characters. The first serial communication devices (with fixed-length pulses) were rotating mechanical switches (*commutators*). Various character codes using 5, 6, 7, or 8 data bits became common in teleprinters and later as computer peripherals. The teletypewriter made an excellent general-purpose I/O device for a small computer.

Gordon Bell of DEC designed the first UART, occupying an entire circuit board called a *line unit*, for the PDP series of computers beginning with the PDP-1.[4][5] According to Bell, the main innovation of the UART was its use of sampling to convert the signal into the digital domain, allowing more reliable timing than previous circuits that used analog timing devices with manually adjusted potentiometers.[6] To reduce the cost of wiring, backplane and other components, these computers also pioneered flow control using XON and XOFF characters rather than hardware wires.

DEC condensed the line unit design into an early single-chip UART for their own use.[4] Western Digital developed this into the first widely available single-chip UART, the WD1402A, around 1971. This was an early example of a medium-scale integrated circuit. Another popular chip was the SCN2651 from the Signetics 2650 family.

An example of an early 1980s UART was the National Semiconductor 8250 used in the original IBM PC's Asynchronous Communications Adapter card.[7] In the 1990s, newer UARTs were developed with on-chip buffers. This allowed higher transmission speed without data loss and without requiring such frequent attention from the computer. For example, the popular National Semiconductor 16550 has a 16-byte FIFO, and spawned many variants, including the *16C550, 16C650, 16C750, and 16C850*.

Depending on the manufacturer, different terms are used to identify devices that perform the UART functions. Intel called their 8251 device a "Programmable Communication Interface". MOS Technology 6551 was known under the name "Asynchronous Communications Interface Adapter" (ACIA). The term "Serial Communications Interface" (SCI) was first used at Motorola around 1975 to refer to their start-stop asynchronous serial interface device, which others were calling a UART. Zilog manufactured a number of Serial Communication Controllers or SCCs.

Starting in the 2000s, most IBM PC compatible computers removed their external RS-232 COM ports and used USB ports that can send data faster. For users who still need RS-232 serial ports, external USB-to-UART bridges are now commonly used. They combine the hardware cables and a chip to do the USB and UART conversion. Cypress Semiconductor and FTDI are two of the significant commercial suppliers of these chips.[8] Although RS-232 ports are no longer available to users on the outside of most computers, many internal processors and microprocessors have UARTs built into their chips to give hardware designers the ability to interface with other chips or devices that use RS-232 or RS-485 for communication.

Structure

A UART usually contains the following components:

- a clock generator, usually a multiple of the bit rate to allow sampling in the middle of a bit period
- input and output shift registers
- transmit/receive control
- read/write control logic
- autobaud measurement (optional)
- transmit/receive buffers (optional)
- system data bus buffer (optional)
- first-in, first-out (FIFO) buffer memory (optional)
- signals needed by a third party DMA controller (optional)
- integrated bus mastering DMA controller (optional)

Special transceiver conditions

Overrun error

An *overrun error* occurs when the receiver cannot process the character that just came in before the next one arrives. Various devices have different amounts of buffer space to hold received characters. The CPU or DMA controller must service the UART in order to remove characters from the input buffer. If the CPU or DMA controller does not service the UART quickly enough and the buffer becomes full, an overrun error will occur, and incoming characters will be lost.

Underrun error

An *underrun error* occurs when the UART transmitter has completed sending a character and the transmit buffer is empty. In asynchronous modes this is treated as an indication that no data remains to be transmitted, rather than an error, since additional stop bits can be appended. This error indication is commonly found in USARTs, since an underrun is more serious in synchronous systems.

Framing error

A UART will detect a *framing error* when it does not see a "stop" bit at the expected "stop" bit time. As the "start" bit is used to identify the beginning of an incoming character, its timing is a reference for the remaining bits. If the data line is not in the expected state (high) when the "stop" bit is expected (according to the number of data and parity bits for which the UART is set), the UART will signal a framing error. A "break" condition on the line is also signaled as a framing error.

Parity error

A *parity error* occurs when the parity of the number of one-bits disagrees with that specified by the parity bit. Parity checking is often used for the detection of transmission errors. Use of a parity bit is optional, so this error will only occur if parity-checking has been enabled.

Break condition

A *break condition* occurs when the receiver input is at the "space" (logic low, i.e., '0') level for longer than some duration of time, typically, for more than a character time. This is not necessarily an error, but appears to the receiver as a character of all zero-bits with a framing error.

The term "break" derives from current loop signaling, which was the traditional signaling used for teletypewriters. The "spacing" condition of a current loop line is indicated by no current flowing, and a very long period of no current flowing is often caused by a break or other fault in the line. Some equipment will deliberately transmit the "space" level for longer than a character as an attention signal. When signaling rates are mismatched, no meaningful characters can be sent, but a long "break" signal can be a useful way to get the attention of a mismatched receiver to do something (such as resetting itself). Computer systems can use the long "break" level as a request to change the signaling rate, to support dial-in access at multiple signaling rates. The DMX512 protocol uses the break condition to signal the start of a new packet.

UART models

A dual UART, or *DUART*, combines two UARTs into a single chip. Similarly, a quadruple UART or *QUART*, combines four UARTs into one package, such as the NXP 28L194. An octal UART or *OCTART* combines eight UARTs into one package, such as the Exar XR16L788 or the NXP SCC2698.

Model	Description
WD1402A	The first single-chip UART on general sale. Introduced about 1971. Compatible chips included the Fairchild TR1402A and the General Instruments AY-5-1013.[9]
Exar XR21V1410	
Intersil 6402	
CDP 1854 (RCA, now Intersil)	
Zilog Z8440	Universal synchronous and asynchronous receiver-transmitter. 2000 kbit/s. Async, Bisync, SDLC, HDLC, X.25. CRC. 4-byte RX buffer. 2-byte TX buffer. Provides signals needed by a third party DMA controller to perform DMA transfers.[10]
Z8530/Z85C30	This universal synchronous and asynchronous receiver-transmitter has a 3-byte receive buffer and a 1-byte transmit buffer. It has hardware to accelerate the processing of HDLC and SDLC. The CMOS version (Z85C30) provides signals to allow a third party DMA controller to perform DMA transfers. It can do asynchronous, byte level synchronous, and bit level synchronous communications.[11]
8250	Obsolete with 1-byte buffers. These UARTs' maximum standard serial port speed is 9600 bits per second if the operating system has a 1 millisecond interrupt latency. 8250 UARTs were used in the IBM PC 5150 and IBM PC/XT, while the 16450 UART were used in IBM PC/AT-series computers.
8251	
Motorola 6850	
6551	
Rockwell 65C52	
16450	
82510	This UART allows asynchronous operation up to 288 kbit/s, with two independent four-byte FIFOs. It was produced by Intel at least from 1993 to 1996, and Innovastic Semiconductor has a 2011 Data Sheet for IA82510.
16550	This UART's FIFO is broken, so it cannot safely run any faster than the 16450 UART. The 16550A and later versions fix this bug.
16550A	This UART has 16-byte FIFO buffers. Its receive interrupt trigger levels can be set to 1, 4, 8, or 14 characters. Its maximum standard serial port speed if the operating system has a 1 millisecond interrupt latency is 128 kbit/s. Systems with lower interrupt latencies or with DMA controllers could handle higher baud rates. This chip can provide signals that are needed to allow a DMA controller to perform DMA transfers to and from the UART if the DMA mode this UART introduces is enabled.[12] It was introduced by National Semiconductor, which has been sold to Texas Instruments. National Semiconductor claimed that this UART could run at up to 1.5 Mbit/s.
16C552	
16650	This UART was introduced by Startech Semiconductor which is now owned by Exar Corporation and is not related to Startech.com. Early versions have a broken FIFO buffer and therefore cannot safely run any faster than the 16450 UART.[13] Versions of this UART that were not broken have 32-character FIFO buffers and could function at standard serial port speeds up to 230.4 kbit/s if the operating system has a 1 millisecond interrupt latency. Current versions of this UART by Exar claim to be able to handle up to 1.5 Mbit/s. This UART introduces the Auto-RTS and Auto-CTS features in which the RTS# signal is controlled by the UART to signal the external device to stop transmitting when the UART's buffer is full to or beyond a user-set trigger point and to stop transmitting to the device when the device drives the CTS# signal high (logic 0).
16750	64-byte buffers. This UART can handle a maximum standard serial port speed of 460.8 kbit/s if the maximum interrupt latency is 1 millisecond. This UART was introduced by Texas Instruments. TI claims that early models can run up to 1 Mbit/s, and later models in this series can run up to 3 Mbit/s.
16850	128-byte buffers. This UART can handle a maximum standard serial port speed of 921.6 kbit/s if the maximum interrupt latency is 1 millisecond. This UART was introduced by Exar Corporation. Exar claims that early versions can run up to 2 Mbit/s, and later versions can run up to 2.25 Mbit/s depending on the date of manufacture.
16C850	

16950	128-byte buffers. This UART can handle a maximum standard serial port speed of 921.6 kbit/s if the maximum interrupt latency is 1 millisecond. This UART supports 9-bit characters in addition to the 5- to 8-bit characters that other UARTs support. This was introduced by Oxford Semiconductor, which is now owned by PLX Technology. Oxford/PLX claims that this UART can run up to 15 Mbit/s. PCI Express variants by Oxford/PLX are integrated with a first party bus mastering PCIe DMA controller. This DMA controller uses the UART's DMA mode signals that were defined for the 16550. The DMA controller requires the CPU to set up each transaction and poll a status register after the transaction is started to determine if the transaction is done. Each DMA transaction can transfer between 1 and 128 bytes between a memory buffer and the UART. PCI Express variants can also allow the CPU to transfer data between itself and the UART with 8-, 16-, or 32-bit transfers when using programmed I/O.
16C950	
16954	Quad-port version of the 16950/16C950. 128-byte buffers. This UART can handle a maximum standard serial port speed of 921.6 kbit/s if the maximum interrupt latency is 1 millisecond. This UART supports 9-bit characters in addition to the 5–8 bit characters that other UARTs support. This was introduced by Oxford Semiconductor, which is now owned by PLX Technology. Oxford/PLX claims that this UART can run up to 15 Mbit/s. PCI Express variants by Oxford/PLX are integrated with a first party bus mastering PCIe DMA controller. This DMA controller is controlled by the UART's DMA mode signals that were defined for the 16550. The DMA controller requires the CPU to set up each transaction and poll a status register after the transaction is started to determine if the transaction is done. Each DMA transaction can transfer between 1 and 128 bytes between a memory buffer and the UART. PCI Express variants can also allow the CPU to transfer data between itself and the UART with 8-, 16-, or 32-bit transfers when using programmed I/O.
16C954	
16C1550/16C1551	UART with 16-byte FIFO buffers. Up to 1.5 Mbit/s. The ST16C155X is not compatible with the industry standard 16550 and will not work with the standard serial port driver in Microsoft Windows.
16C2450	Dual UART with 1-byte FIFO buffers.
16C2550	Dual UART with 16-byte FIFO buffers. Pin-to-pin and functional compatible to 16C2450. Software compatible with INS8250 and NS16C550.
SCC2691	Currently produced by NXP, the 2691[3] is a single channel UART that also includes a programmable counter/timer. The 2691 has a single-byte transmitter holding register and a 4-byte receive FIFO. Maximum standard speed of the 2692 is 115.2 kbit/s. The 28L91 is an upwardly compatible version of the 2691, featuring selectable 8- or 16-byte transmitter and receiver FIFOs, improved support for extended data rates, and faster bus timing characteristics, making the device more suitable for use with high performance microprocessors. Both the 2691 and 28L91 may also be operated in TIA-422 and TIA-485 modes, and may also be programmed to support non-standard data rates. The devices are produced in PDIP-40, PLCC-44 and 44 pin QFP packages, and are readily adaptable to both Motorola and Intel buses. They have also been successfully adapted to the 65C02 and 65C816 buses. The 28L91 will operate on 3.3 or 5 volts.
SCC28L91	
SCC2692	Currently produced by NXP, these devices are dual UARTs (DUART), consisting of two communications channels, associated control registers and one counter/timer. Each communication channel is independently programmable and supports independent transmit and receive data rates. The 2692 has a single-byte transmitter holding register and a 4-byte receiver FIFO for each channel. Maximum standard speed of both of the 2692's channels is 115.2 kbit/s.
SC26C92	

SC26C92	The 26C92 is an upwardly compatible version of the 2692, with 8-byte transmitter and receiver FIFOs for improved performance during continuous bi-directional asynchronous transmission (CBAT) on both channels at the maximum standard speed of 230.4 kbit/s. The letter **C** in the 26C92 part number has nothing to do with the fabrication process; all NXP UARTs are CMOS devices. The 28L92 is an upwardly compatible version of the 26C92, featuring selectable 8- or 16-byte transmitter and receiver FIFOs, improved support for extended data rates, and faster bus timing characteristics, making the device more suitable for use with high performance microprocessors. The 2692, 26C92 and 28L92 may be operated in TIA-422 and TIA-485 modes, and may also be programmed to support non-standard data rates. The devices are produced in PDIP-40, PLCC-44 and 44 pin QFP packages, and are readily adaptable to both Motorola and Intel buses. They have also been successfully adapted to the 65C02 and 65C816 buses. The 28L92 will operate on 3.3 or 5 volts.
SC28L92	
SCC28C94	Currently produced by NXP, the 28C94 quadruple UART (QUART) is functionally similar to a pair of SCC26C92 DUARTs mounted in a common package, with the addition of an arbitrated interrupt system for efficient processing during periods of intense channel activity. Some additional signals are present to support the interrupt management features and the auxiliary input/output pins are arranged differently than those of the 26C92. Otherwise, the programming model for the 28C94 is similar to that of the 26C92, requiring only minor code changes to fully utilize all features. The 28C94 supports a maximum standard speed of 230.4 kbit/s, is available in a PLCC-52 package, and is readily adaptable to both Motorola and Intel buses. It has also been successfully adapted to the 65C816 bus.
SCC2698B	Currently produced by NXP, the 2698 octal UART (OCTART) is essentially four SCC2692 DUARTs in a single package. Specifications are the same as the SCC2692 (not the SCC26C92). Due to the lack of transmitter FIFOs and the small size of the receiver FIFOs, the 2698 can cause an interrupt "storm" if all channels are simultaneously engaged in continuous bi-directional communication. The device is produced in PDIP-64 and PLCC-84 packages, and is readily adaptable to both Motorola and Intel buses. The 2698 has also been successfully adapted to the 65C02 and 65C816 buses.
SCC28L198	Currently produced by NXP, the 28L198 OCTART is essentially an upscaled enhancement of the SCC28C94 QUART described above, with eight independent communications channels, as well as an arbitrated interrupt system for efficient processing during periods of intense channel activity. The 28L198 supports a maximum standard speed of 460.8 kbit/s, is available in PLCC-84 and LQFP-100 packages, and is readily adaptable to both Motorola and Intel buses. The 28L198 will operate on 3.3 or 5 volts.
Z85230	Synchronous/Asynchronous modes, 2 ports. Provides signals needed by a third party DMA controller needed to perform DMA transfers. 4-byte buffer to send, 8-byte buffer to receive per channel. SDLC/HDLC modes. 5 Mbit/s in synchronous mode.
Hayes ESP	1 KB buffers, 921.6 kbit/s, 8-ports.[14]
Exar XR17V352, XR17V354 and XR17V358	Dual, Quad and Octal PCI Express UARTs with 16550 compatible register Set, 256-byte TX and RX FIFOs, Programmable TX and RX Trigger Levels, TX/RX FIFO Level Counters, Fractional baud rate generator, Automatic RTS/CTS or DTR/DSR hardware flow control with programmable hysteresis, Automatic Xon/Xoff software flow control, RS-485 half duplex direction control output with programmable turn-around delay, Multi-drop with Auto Address Detection, Infrared (IrDA 1.1) data encoder/decoder. They are specified up to 25 Mbit/s. DataSheets are dated from 2012.
Exar XR17D152, XR17D154 and XR17D158	Dual, Quad and Octal PCI bus UARTs with 16C550 Compatible 5G Register Set, 64-byte Transmit and Receive FIFOs, Transmit and Receive FIFO Level Counters, Programmable TX and RX FIFO Trigger Level, Automatic RTS/CTS or DTR/DSR Flow Control, Automatic Xon/Xoff Software Flow Control, RS485 HDX Control Output with Selectable Turn-around

	Delay, Infrared (IrDA 1.0) Data Encoder/Decoder, Programmable Data Rate with Prescaler, Up to 6.25 Mbit/s Serial Data Rate. DataSheets are dated from 2004 and 2005.
Exar XR17C152, XR17C154 and XR17C158	Dual, Quad and Octal 5 V PCI bus UARTs with 16C550 Compatible Registers, 64-byte Transmit and Receive FIFOs, Transmit and Receive FIFO Level Counters, Automatic RTS/CTS or DTR/DSR Flow Control, Automatic Xon/Xoff Software Flow Control, RS485 Half-duplex Control with Selectable Delay, Infrared (IrDA 1.0) Data Encoder/Decoder, Programmable Data Rate with Prescaler, Up to 6.25 Mbit/s Serial Data Rate. DataSheets are dated from 2004 and 2005.
Exar XR17V252, XR17V254 and XR17V258	Dual, Quad and Octal 66 MHz PCI bus UARTs with Power Management Support, 16C550 compatible register set, 64-byte TX and RX FIFOs with level counters and programmable trigger levels, Fractional baud rate generator, Automatic RTS/CTS or DTR/DSR hardware flow control with programmable hysteresis, Automatic Xon/Xoff software flow control, RS-485 half duplex direction control output with selectable turn-around delay, Infrared (IrDA 1.0) data encoder/decoder, Programmable data rate with prescaler. DataSheets are dated from 2008 and 2010.

UART in modems

Modems for personal computers that plug into a motherboard slot must also include the UART function on the card. The original 8250 UART chip shipped with the IBM personal computer had a one character buffer for the receiver and the transmitter each, which meant that communications software performed poorly at speeds above 9600 bit/s, especially if operating under a multitasking system or if handling interrupts from disk controllers. High-speed modems used UARTs that were compatible with the original chip but which included additional FIFO buffers, giving software additional time to respond to incoming data.

A look at the performance requirements at high bit rates shows why the 16-, 32-, 64- or 128-byte FIFO is a necessity. The Microsoft specification for a DOS system requires that interrupts not be disabled for more than 1 millisecond at a time. Some hard disk drives and video controllers violate this specification. 9600 bit/s will deliver a character approximately every millisecond, so a 1-byte FIFO should be sufficient at this rate on a DOS system which meets the maximum interrupt disable timing. Rates above this may receive a new character before the old one has been fetched, and thus the old character will be lost. This is referred to as an overrun error and results in one or more lost characters.

A 16-byte FIFO allows up to 16 characters to be received before the computer has to service the interrupt. This increases the maximum bit rate the computer can process reliably from 9600 to 153,000 bit/s if it has a 1 millisecond interrupt dead time. A 32-byte FIFO increases the maximum rate to over 300,000 bit/s. A second benefit to having a FIFO is that the computer only has to service about 8 to 12% as many interrupts, allowing more CPU time for updating the screen, or doing other chores. Thus the computer's responses will improve as well.

See also

- Autobaud
- Baud
- Bit rate
- Comparison of synchronous and asynchronous signalling
- Crystal oscillator frequencies
- MIDI
- Synchronous serial communication

References

1. Adam Osborne, *An Introduction to Microcomputers Volume 1: Basic Concepts*, Osborne-McGraw Hill Berkeley California USA, 1980 ISBN 0-931988-34-9 pp. 116–126
2. "Determining Clock Accuracy Requirements for UART Communications" (https://pdfserv.maximintegrated.com/en/an/AN2141.pdf) (pdf). *an2141*. Maxim Integrated. 2003-08-07. Retrieved 1 November 2021.
3. "Universal asynchronous receiver/transmitter (UART)" (https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf#page=14) (PDF). *SCC2691*. Philips NXP. 2006-08-04. p. 14. Retrieved 1 November 2021.
4. C. Gordon Bell, J. Craig Mudge, John E. McNamara, *Computer Engineering: A DEC View of Hardware Systems Design* (http://bitsavers.org/pdf/dec/_Books/Bell-ComputerEngineering.pdf), Digital Press, 12 May 2014, ISBN 1483221105, p. 73
5. Allison, David. "Curator, Division of Information Technology and Society, National Museum of American History, Smithsonian Institution" (http://americanhistory.si.edu/comphist/bell.htm#Inventing%20the%20UART). *Smithsonian Institution Oral and Video Histories*. Retrieved 14 June 2015.
6. *Oral History of Gordon Bell (http://archive.computerhistory.org/resources/text/Oral_History/Bell_Gordon_1/102702036.05.01.pdf)*, 2005, accessed 2015-08-19
7. *Technical Reference 6025008* (http://www.minuszerodegrees.net/manuals/IBM_5150_Technical_Reference_6025005_AUG81.pdf) (PDF). Personal Computer Hardware Reference Library. IBM. August 1981. pp. 2–123.
8. "FTDI Products" (http://www.ftdichip.com/FTProducts.htm). *www.ftdichip.com*. Retrieved 22 March 2018.
9. *Interfacing with a PDP-11/05: the UART (http://retrocmp.com/how-tos/interfacing-to-a-pdp-1105/143-interfacing-with-a-pdp-1105-the-uart)*, blinkenbone.com, accessed 2015-08-19
10. "Zilog Product specification Z8440/1/2/4, Z84C40/1/2/3/4. Serial input/output controller" (http://www.zilog.com/docs/z80/ps0183.pdf) (PDF). 090529 zilog.com
11. "Zilog Document Download" (http://www.zilog.com/docs/serial/PS0117.pdf) (PDF). *www.zilog.com*. Retrieved 22 March 2018.
12. "FAQ: The 16550A UART & TurboCom drivers 1994" (ftp://ftp.cs.utk.edu/pub/shuford/terminal/mswin_serial_faq.txt). Retrieved January 16, 2016.
13. T'so, Theodore Y. (January 23, 1999). "Re: Serial communication with the 16650" (http://www.mail-archive.com/linux-serial@vger.rutgers.edu/msg00065.html). *The Mail Archive*. Retrieved June 2, 2013.
14. bill.herrin.us - Hayes ESP 8-port Enhanced Serial Port Manual (http://bill.herrin.us/freebies/hayes-esp8/), 2004-03-02

Further reading

- *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*; 2nd Edition; Jan Axelson; Lakeview Research; 380 pages; 2007; ISBN 978-1-931-44806-2.
- *Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks*; 1st Edition; Jan Axelson; Lakeview Research; 306 pages; 1998; ISBN 978-0-965-08192-4.
- *Serial port and Microcontrollers: Principles, Circuits, and Source Codes*; 1st Edition; Grzegorz Niemirowski; CreateSpace; 414 pages; 2013; ISBN 978-1-481-90897-9.
- Serial Programming (Wikibook) (https://en.wikibooks.org/wiki/Serial_Programming).

External links

- FreeBSD Serial and UART Tutorial (http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/), includes standard signal definitions, history of UART ICs, and pinout for commonly used DB25 connector.
- UART Tutorial for Robotics (http://www.societyofrobots.com/microcontroller_uart.shtml), contains many practical examples.