

Serial UART information

- [Serial UART, an introduction](#)
- [Serial UART types](#)
- [Registers](#)

Serial UART, an introduction

A UART, universal asynchronous receiver / transmitter is responsible for performing the main task in serial communications with computers. The device changes incoming parallel information to serial data which can be sent on a communication line. A second UART can be used to receive the information. The UART performs all the tasks, timing, parity checking, etc. needed for the communication. The only extra devices attached are line driver chips capable of transforming the TTL level signals to line voltages and vice versa.

To use the UART in different environments, registers are accessible to set or review the communication parameters. Setable parameters are for example the communication speed, the type of parity check, and the way incoming information is signaled to the running software.

SPONSORED SEARCHES

64 bit computer

Q

uart serial

C

serial number checker

Q

i2c uart

C

Serial UART types

Serial communication on PC compatibles started with the 8250 UART in the IBM XT. In the years after, new family members were introduced like the 8250A and 8250B revisions and the 16450. The last one was first implemented in the AT. The higher bus speed in this computer could not be reached by the 8250 series. The differences between these first UART series were rather minor. The most important property changed with each new release was the maximum allowed speed at the processor bus side.

The 16450 was capable of handling a communication speed of 38.4 kbs without problems. The demand for higher speeds led to the development of newer series which would be able to release the main processor from some of its tasks. The main problem with the original series was the need to perform a software action for each single byte to transmit or receive. To overcome this problem, the 16550 was released which contained two on-board FIFO buffers, each capable of storing 16 bytes. One buffer for incoming, and one buffer for outgoing bytes.

A marvelous idea, but it didn't work out that way. The 16550 chip contained a firmware bug which made it impossible to use the buffers. The 16550A which appeared soon after was the first UART which was able to use its FIFO buffers. This made it possible to increase maximum reliable communication speeds to 115.2 kbs. This speed was necessary to use effectively modems with on-board compression. A further enhancement introduced with the 16550 was the ability to use DMA, direct memory access for the data transfer. Two pins were redefined for this purpose. DMA transfer is not used with most applications. Only special serial I/O boards with a high number of ports contain sometimes the necessary extra circuitry to make this feature work.

The 16550A is the most common UART at this moment. Newer versions are under development, including the 16650 which contains two 32 byte FIFO's and on board support for software flow control. Texas Instruments is developing the 16750 which contains 64 byte FIFO's.

Registers

Eight I/O bytes are used for each UART to access its registers. The following table shows, where each register can be found. The base address used in the table is the lowest I/O port number assigned. The switch bit DLAB can be found in the line control register LCR as bit 7 at I/O address base + 3.

UART register to port conversion table

I/O port	DLAB = 0		DLAB = 1	
	Read	Write	Read	Write
base	RBR receiver buffer	THR transmitter holding	DLL divisor latch LSB	
base + 1	IER interrupt enable	IER interrupt enable	DLM divisor latch MSB	
base + 2	IIR	FCR	IIR	FCR

	interrupt identification	FIFO control	interrupt identification	FIFO control
base + 3		LCR line control		
base + 4		MCR modem control		
base + 5	LSR line status	– factory test	LSR line status	– factory test
base + 6	MSR modem status	– not used	MSR modem status	– not used
base + 7		SCR scratch		

Available registers

[RBR, receiver buffer register](#)

[THR, transmitter holding register](#)

[IER, interrupt enable register](#)

[IIR, interrupt identification register](#)

[FCR, FIFO control register](#)

[LCR, line control register](#)

[MCR, modem control register](#)

[LSR, line status register](#)

[MSR, modem status register](#)

[SCR, scratch register](#)

[DLL, divisor latch LSB](#)

[DLM, divisor latch MSB](#)

The communication between the processor and the UART is completely controlled by twelve registers. These registers can be read or written to check and change the behavior of the communication device. Each register is eight bits wide. On a PC compatible, the registers are accessible in the [I/O address area](#). The function of each register will be discussed here in detail.

RBR : Receiver buffer register (RO)

The RBR, receiver buffer register 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 LSR line status register 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 (WO)

The THR, transmitter holding register is used to buffer outgoing characters. If no FIFO buffering is used, only one character can be stored. Otherwise the amount of characters depends on the type of UART. Bit 5 in the LSR, line status register can be used to check if new information must be written to THR. The value 1 indicates that the register is empty. If FIFO buffering is used, more than one character can be written to the transmitter holding register when the bit signals an empty state. There is no indication of the amount of bytes currently present in the transmitter FIFO.

The transmitter holding register is not used to transfer the data directly. The byte is first transferred to a shift register where the information is broken in single bits which are sent one by one.

IER : Interrupt enable register (R/W)

The smartest way to perform serial communications on a PC is using interrupt driven routines. In that configuration, it is not necessary to poll the registers of the UART periodically for state changes. The UART will signal each change by generating a processor interrupt. A software routine must be present to handle the interrupt and to check what state change was responsible for it.

Interrupts are not generated, unless the UART is told to do so. This is done by setting bits in the IER, interrupt enable register. A bit value 1 indicates, that an interrupt may take place.

IER : Interrupt enable register

Bit	Description
-----	-------------

0	Received data available
1	Transmitter holding register empty
2	Receiver line status register change
3	Modem status register change
4	Sleep mode (16750 only)
5	Low power mode (16750 only)
6	reserved
7	reserved

IIR : Interrupt identification register (RO)

A UART is capable of generating a processor interrupt when a state change on the communication device occurs. One interrupt signal is used to call attention. This means, that additional information is needed for the software before the necessary actions can be performed. The IIR, interrupt identification register is helpful in this situation. Its bits show the current state of the UART and which state change caused the interrupt to occur.

IIR : Interrupt identification register

Bit	Value			Description	Reset by
0	0			Interrupt pending	–
	1			No interrupt pending	–
1,2,3	Bit 3	Bit 2	Bit 1		
	0	0	0	Modem status change	MSR read
	0	0	1	Transmitter holding register empty	IIR read or THR write
	0	1	0	Received data available	RBR read
	0	1	1	Line status change	LSR read
4	1	1	0	Character timeout (16550)	RBR read
	0			Reserved	–
5	0			Reserved (8250, 16450, 16550)	–
	1			64 byte FIFO enabled (16750)	–
6,7	Bit 7	Bit 6			
	0	0		No FIFO	–
	1	0		Unusable FIFO (16550 only)	–
	1	1		FIFO enabled	–

FCR : FIFO control register (WO)

The FCR, FIFO control register is present starting with the 16550 series. This register controls the behavior of the FIFO's in the UART. 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.

FCR : FIFO control register

Bit	Value		Description
0	0		Disable FIFO's
	1		Enable FIFO's
1	0		–
	1		Clear receive FIFO
2	0		–
	1		Clear transmit FIFO
3	0		Select DMA mode 0
	1		Select DMA mode 1
4	0		Reserved
5	0		Reserved (8250, 16450, 16550)
	1		Enable 64 byte FIFO (16750)
6,7	Bit 7	Bit 6	Receive FIFO interrupt trigger level
	0	0	1 byte
	0	1	4 bytes
	1	0	8 bytes
	1	1	14 bytes

LCR : Line control register (R/W)

The LCR, line control register is used at initialization to set the communication parameters. Parity and number of data bits can be changed for example. The register also controls the accessibility of the DLL and DLM registers. These registers are mapped to the same I/O port as the RBR, THR and IER registers. Because they are only accessed at initialization when no communication occurs this register swapping has no influence on performance.

LCR : line control register

Bit	Value			Description
0,1	Bit 1	Bit 0		Data word length
	0	0		5 bits
	0	1		6 bits
	1	0		7 bits
2	1	1		8 bits
		0		1 stop bit
		1		1.5 stop bits (5 bits word)
				2 stop bits (6, 7 or 8 bits word)
3,4,5	Bit 5	Bit 4	Bit 3	
	x	x	0	No parity
	0	0	1	Odd parity
	0	1	1	Even parity
	1	0	1	High parity (stick)
6	1	1	1	Low parity (stick)
		0		Break signal disabled
		1		Break signal enabled
7		0		DLAB : RBR, THR and IER accessible
		1		DLAB : DLL and DLM accessible

Some remarks about parity:

The UART is capable of generating a trailing bit at the end of each data-word which can be used to check some data distortion. Because only one bit is used, the parity system is capable of detecting only an odd number of false bits. If an even number of bits has been flipped, the error will not be seen.

When even parity is selected, the UART assures that the number of high bit values in the sent or received data is always even. Odd parity setting does the opposite. Using stick parity has very little use. It sets the parity bit to always 1, or always 0.

Common settings are:

- 8 data bits, one stop bit, no parity
- 7 data bits, one stop bit, even parity

MCR : Modem control register (R/W)

The MCR, modem control register is used to perform handshaking actions with the attached device. In the original UART series including the 16550, setting and resetting of the control signals must be done by software. The new 16750 is capable of handling flow control automatically, thereby reducing the load on the processor.

MCR : Modem control register

Bit	Description
0	Data terminal ready
1	Request to send
2	Auxiliary output 1
3	Auxiliary output 2
4	Loopback mode
5	Autoflow control (16750 only)
6	Reserved
7	Reserved

The two auxiliary outputs are user definable. Output 2 is sometimes used in circuitry which controls the interrupt process on a PC. Output 1 is normally not used, however on some I/O cards, it controls the selection of a second oscillator working at 4 MHz. This is mainly for MIDI purposes.

LSR : Line status register (RO)

The LSR, line status register shows the current state of communication. Errors are reflected in this register. The state of the receive and transmit buffers is also available.

LSR : Line status register

Bit	Description
0	Data available
1	Overrun error
2	Parity error
3	Framing error
4	Break signal received
5	THR is empty
6	THR is empty, and line is idle
7	Errornous data in FIFO

Bit 5 and 6 both show the state of the transmitting cycle. The difference is, that bit 5 turns high as soon as the transmitter holding register is empty whereas bit 6 indicates that also the shift register which outputs the bits on the line is empty.

MSR : Modem status register (RO)

The MSR, modem status register contains information about the four incoming modem control lines on the device. The information is split in two nibbles. The four most significant bits contain information about the current state of the inputs where the least significant bits are used to indicate state changes. The four LSBs are reset, each time the register is read.

MSR : Modem status register

Bit	Description
0	change in Clear to send
1	change in Data set ready
2	trailing edge Ring indicator
3	change in Carrier detect
4	Clear to send
5	Data set ready
6	Ring indicator
7	Carrier detect

SCR : Scratch register (R/W)

The SCR, scratch register was not present on the 8250 and 8250B UART. It can be used to store one byte of information. In practice, it has only limited use. The only real use I know of is checking if the UART is a 8250/8250B, or a 8250A/16450 series. Because the 8250 series are only found in XTs even this use of the register is not commonly seen anymore.

DLL and DLM : Divisor latch registers (R/W)

For generating its 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. Because of this division, the maximum allowed communication speed is 115200 bps. Modern UARTS 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. On PC's higher frequencies than the 1.8432 MHz are rarely seen because this would be software incompatible with the original XT configuration.

This 115200 bps communication speed is not suitable for all applications. To change the communication speed, the frequency can be further decreased by dividing it by a programmable value. For very slow communications, this value can go beyond 255. Therefore, the divisor is stored in two separate bytes, the divisor latch registers DLL and DLM which contain the least, and most significant byte.

For error free communication, it is necessary that both the transmitting and receiving UART use the same time base. Default values have been defined which are commonly used. The table shows the most common values with the appropriate settings of the divisor latch bytes. Note that these values only hold for a PC compatible system where a clock frequency of 1.8432 MHz is used.

DLL and DLM : Divisor latch registers

Speed (bps)	Divisor	DLL	DLM
50	2,304	0x00	0x09
300	384	0x80	0x01
1,200	96	0x60	0x00
2,400	48	0x30	0x00
4,800	24	0x18	0x00
9,600	12	0x0C	0x00
19,200	6	0x06	0x00
38,400	3	0x03	0x00
57,600	2	0x02	0x00
115,200	1	0x01	0x00

