Computer Labs: The PC's Serial Port
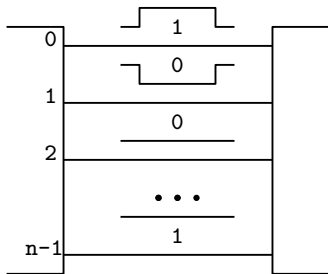## 2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

October 28, 2013

Parallel Communication (1/2)

- ▶ This is the type of communication used in the memory bus for communication between the CPU and memory
    - ▶ The memory bus has several lines, usually as many as the CPU word size (32-bit for IA-32 architecture)
    - ▶ Each bit of a memory word is placed on the corresponding line, at the same time
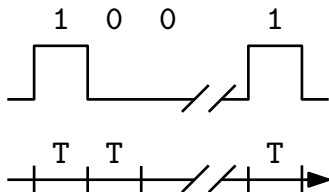
Parallel Communication (2/2)

Parallel Communication **Signals** are sent simultaneously in **parallel** over several **channels**

Signal A physical quantity that represents a sequence of bits (more generally information)

Channel A transmission medium such as a pair of wires, a frequency band of the radio spectrum, a light wavelength in an optical fiber

- ► The set of channels used for parallel comunication is often named as a **bus**. Some examples include:
  - ► Many buses used to interface with the controllers of I/0 devices, starting with ISA, ATA, SCSI, PCI and AGP

Serial Communication

Serial Communication **Signals** are sent sequentially over one
channel, also called **serial bus**. Some examples include:

- The serial port of the PC (RS-232)
- Many network technologies, such as Ethernet, WiFi,
  GSM
- Many buses used to interface with (external) I/O devices,
  such as USB, FireWire (IEEE 1394), (e)SATA

Synchronous vs. Asynchronous Serial Communication

Problem How does the receiver synchronize? I.e., how does a receiver know when a bit ends and a new one starts?

Solutions There are essentially two approaches, that differ on the synchronization between the clocks of the sender and the receiver:

Synchronous Communication In which the clocks are synchronized:

- ▶ Either by embedding the clock into the bit stream, e.g. using Manchester encoding, where a 1 is encoded as a low to high transition and a 0 as a high to low transition
- ▶ Or by using just one clock which is sent in a different "channel" (e.g. in the communication between the keyboard and the KBC)

Asynchronous Comunication In which the clocks run independently

Asynchronous Serial Communication

- ▶ So that the receiver synchronizes:
    - ▶ The data is grouped in **characters** of typically 7 or 8 bits
    - ▶ Each character is:
        - ▶ preceded by a **start bit**, usually a 0
        - ▶ followed by at least one **stop bit**, usually a 1
- ▶ Other communication parameters include:

    Parity bit This is used for simple error detection
    Bit-rate The maximum number of bits that are transmitted per time unit

    - ▶ Not to confuse with **baud-rate**, the number of **symbols** transmitted per second.
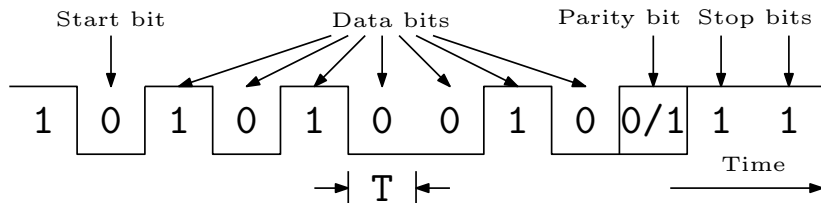
Simplex, Duplex and Half-Duplex Communication

Simplex Communication in only one direction

- ▶ One end is always the sender and the other always the receiver

(Full-)Duplex Communication in both directions, simultaneously

- ▶ Both ends play the roles of sender and receiver possibly simultaneously

Half-Duplex Communication in both directions, but **not** simultaneously

- ▶ Both ends play the roles of sender and receiver, but at any time instant an end cannot be both a sender and a receiver

Serial Communication on the PC

- ▶ PCs used to have one or two serial port
  - ▶ Conforming to the RS-232 standard
  - ▶ Using a D-9 connector, first introduced by IBM
- ▶ Each serial port is controlled by a Universal Asynchronous Receiver/ Transmitter (UART), an asynchronous communication controller
- ▶ Each UART takes:
  - ▶ Eight (8) consecutive port-numbers in the PC's I/O address
  - ▶ One IRQ line

Port	Base Address	IRQ	Vector
COM1	`0x3F8 (-0x3FF)`	4	`0x0C`
COM2	`0x2F8 (-0x2FF)`	3	`0x0B`

Lab 7: The PC's Serial Port

▶ Write functions:

```
int test_conf();
int test_set();
int test_poll()/test_int();
int test_fifo();
```

that require interfacing with the UART

▶ These functions are not the kind of functions that you can reuse later in your project
  ▶ The idea is that you design the lower level functions
  ▶ Pay attention to the way you partition your code by files
▶ What's new?
  ▶ Use the UART
    ▶ Asynchronous serial communication
  ▶ Some details will be revealed only in class
  ▶ Test your program using two VMs, connected via sockets.
  ▶ Lab will take 2 lab-classes, but in the second class you will have also to discuss your project proposal

# UART's Block Diagram

- As the bits are received they are put in the Receiver Shift Register
    - Once all the bits in a character have been received they are moved to the Receiver Buffer Register
- Characters to transmit are put in the Transmitter Holding Register
    - They are moved to the Transmitter Shift Register when it is empty
    - The Transmitter Shift Register sends the bits one after the other
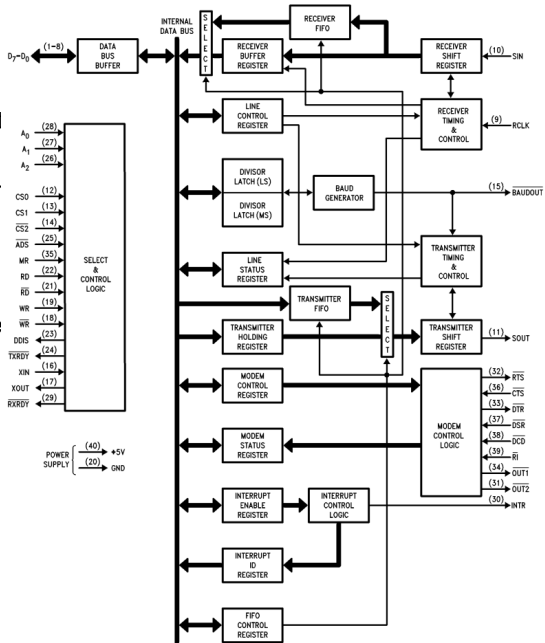
# UART Accessible (8-bit) Registers

| Address | Read/Write | Mnem. | Description |
|---------|-----------|-------|-------------|
| 0 | R | RBR | Receiver Buffer Register |
|   | W | THR | Transmitter Holding Register |
| 1 | R/W | IER | Interrupt Enable Register |
| 2 | R | IIR | Interrupt Identification Reg. |
|   | W | FCR | FIFO Control Register |
| 3 | R/W | LCR | Line Control Register |
| 4 | R/W | MCR | Modem Control Register |
| 5 | R | LSR | Line Status Register |
| 6 | R | MSR | Modem Status Register |
| 7 | R/W | SR | Scratchpad Register |

IMPORTANT Addresses 0 and 1 are overloaded, accessing different registers if bit DLAB of the LCR register is set to 1:

| Address | Read/Write | Mnem. | Description |
|---------|-----------|-------|-------------|
| 0 | R/W | DLL | Divisor Latch LSB |
| 1 | R/W | DLM | Divisor Latch MSB |

# Purpose of the Control/Status Registers

Line Control Register (LCR) Allows the setting of the main asynchronous communication parameters: number of bits per character, number of stop bits and parity

DLL and DLM Allows the setting of the the bit rate (by means of a frequency divider), via the Divisor Latches of the programmable bit-rate generator.

Line Status Register (LSR) Provides status information concerning the data transfer: whether a character was transmitted or received, and in the latter case whether an error was detected

Interrupt Enable Register (IER) Allows the selection of the events that may generate interrupts

Interrupt Identification Register (IIR) Provides information regarding the event that caused an interrupt

FIFO Control Register (FCR) Allows the control of FIFO buffering, both for reception and for transmission

IMPORTANT These registers may also include state/control bits that do not match exactly the purpose described above.

# Line Control Register (LCR)

| Bit | | | | Meaning |
|---|---|---|---|---|
| 1,0 | | | | Number of bits per char |
| | | 0 | 0 | 5 bits per char |
| | | 0 | 1 | 6 bits per char |
| | | 1 | 0 | 7 bits per char |
| | | 1 | 1 | 8 bits per char |
| 2 | | | 0 | 1 stop bit |
| | | | 1 | 2 stop bits (1 and 1/2 when 5 bits char) |
| 5,4,3 | | | | Parity control |
| | X | X | 0 | No parity |
| | 0 | 0 | 1 | Odd parity |
| | 0 | 1 | 1 | Even parity |
| | 1 | 0 | 1 | Parity bit is 1 (always) |
| | 1 | 1 | 1 | Parity bit is 0 (always) |
| 6 | | | | Break control: sets serial output to 0 (low) |
| 7 (DLAB) | | | 1 | Divisor Latch Access |
| | | | 0 | RBR (read) or THR (write) |

# Line Status Register (LSR) (Read only)

| Bit | Name | Meaning |
|-----|------|---------|
| 0 | Receiver Data | Set to 1 when there is data for receiving |
| 1 | Overrun Error | Set to 1 when a characters received is overwritten by another one |
| 2 | Parity Error | Set to 1 when a character with a parity error is received |
| 3 | Framing Error | Set to 1 when a received character does not have a valid Stop bit |
| 4 | Break Interrupt | Set to 1 when the serial data input line is held in the low level for longer than a full "word" transmission |
| 5 | Transmitter Holding Register Empty | When set, means that the UART is ready to accept a new character for transmitting |
| 6 | Transmitter Empty Register | When set, means that both the THR and the Transmitter Shift Register are both empty |
| 7 | FIFO Error | Set to 1 when there is at least one parity error or framing error or break indication in the FIFO. Reset to 0 when the LSR is read, if there are no subsequent errors in the FIFO |

Note  Bits 0 to 4 are reset when LSR is read.

Note  When using FIFO buffering the operation of these bits may be slightly different. (Check the 16550 datasheet)

# LSR and Error Detection

- ► Use the LSR to find out whether there was a
  communications error:

  Bit 1 Received char overwritten
  Bit 2 Parity error
  Bit 3 Stop bit missing

```
#define SER_OVERRUN_ERR (1<<1)
#define SER_PARITY_ERR  (1<<2)
#define SER_FRAME_ERR   (1<<3)

if( sys_inb(ser_port + SER_LSR, &st) != OK ) {
    ...
} else if {
   if ( st & (SER_OVERRUN_ERR | SER_PARITY_ERR
            | SER_FRAME_ERR) ) {
       ... // Handle error
   }
```

LSR and Polled Operation

- ▶ Use the LSR to transfer data in polled mode:

  On reception Check **bit 0, `Receiver Ready`**

  - ▶ If bit set, then read character from the Receiver Buffer Register, i.e. the register at relative address 0 (ensure `DLAB` is reset)

  On transmission Check **bit 5, `Transmitter Holding Register Empty`** (why not `Transmitter Ready`?)

Polled Operation: Transmission

```
#define SER_LSR      5
#define SER_DATA     0
#define SER_TX_RDY (1<<5)

/* no error checking */
...

/* busy wait for transmitter ready */

while( !(lsr & SER_TX_RDY) ) {
     ticksdelay();
     sys_inb(ser_port + SER_LSR, &lsr);
}

/* send character */
sys_outb(ser_port+SER_DATA, c);
```
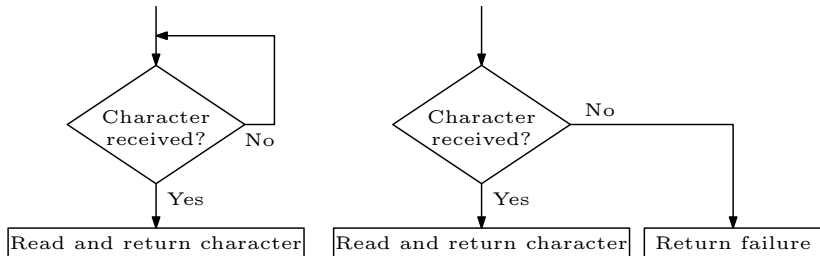
- ▶ Busy waiting on transmission is simple
  - ▶ But what if there is some problem on the UART?

Polled Operation: Reception

- ► Use busy waiting only if you are expecting to receive a character
    - ► But what if there is some problem on the other end of the line?
- ► The alternative to polled operation is to use interrupts

Lab 7: `test_conf`

What? Read and display the configuration of the UART

- ▶ Both the serial communication parameters (LCR)
- ▶ The interrupt events that are enabled, if any (IER)

For class preparation This time you need to implement it fully

- ▶ Including display the configuration in a human readable way
- ▶ You can test it by, running `stty -a < /dev/tty00`
  - ▶ Make sure you do not change the UART configuration before

Lab 7: `test_set`

What? Set the serial communication parameters of one serial port by writing to the LCR

How to test? Use `test_conf()`

- ▶ Cannot use `stty -a < /dev/tty00`
  - ▶ `stty` does not read the LCR, rather uses the configuration kept by the Minix device driver
  - ▶ But, the change in the configuration was not by the Minix device driver, hence ...

Lab 7: `test_poll`/`test_int`

What? Send/receive strings using the serial port in

- polled mode
- interrupt mode (in the second class only?)

Challenge Need to have both a sender and a receiver for testing ...

- Use executable I provide with my solution

How to test?

1. Use two VMs on the same PC, with the serial port of one connected to the serial port of another
   - Except for the number of bits per char, all the other communication parameters do not matter

The Interrupt Enable Register (IER)

- ▶ Controls whether or not the UART generates interrupts on some events
- ▶ An event will generate an interrupt if the corresponding bit of the IER is set:

Bit	Meaning
0	Enables the Received Data Available Interrupt
1	Enables the Transmitter Holding Register Empty Interrupt
2	Enables the Receiver Line Status Interrupt This event is generated when there is a change in the state of bits 1 to 4, i.e. the error bits, of the LSR
3	Enables the MODEM Status Interrupt

Note You can safely ignore the MODEM Status Interrupts in Lab Assignment 7

The Interrupt Identification Register (IIR)

- ▶ Records the source of interrupts
- ▶ The UART prioritizes interrupts in 4 levels:
    1. Receiver Line Status, i.e. receiver error interrupts
    2. Received Data Ready, i.e. received char interrupt
    3. Transmitter Holding Register Empty, i.e transmitter ready interrupt
    4. Modem status
- ▶ When the IIR is read, the UART freezes all interrupts and indicates the highest priority pending interrupt
- ▶ The meaning of the bits in the IIR is as follows:

Bit				Meaning
0				If set, no interrupt is pending
3,2,1				Interrupt pending, prioritized as follows
	0	1	1	Receiver Line Status
	0	1	0	Reived Data Available
	1	1	0	Character Timeout (FIFO), discussed below
	0	0	1	Transmitter Holding Register Empty
	0	0	0	Modem Status

Interrupt Handling

- ► The original PIC architecture supports only 15 IRQ lines (APIC has no such limitation).
- ► To overcome this, there is a need for sharing IRQs:
  - ► Within a (device) controller
  - ► Among (device) controllers

```
void ser_ih() {
    sys_inb(ser_port + SER_IIR, &iir);
    if( iir & SER_INT_PEND ) {
        switch( iir & INT_ID ) {
        case SER_RX_INT:
            ... /* read received character */
        case SER_TX_INT:
            ... /* put character to sent */
        case SER_RX_ERR:
            ... /* notify upper level */
        case SER_XXXX:
            ... /* depends on XXX */
        }
    }
}
```

Interrupt-driven Transmission

Issue When starting transmission after an idle period, (or at the beginning), the THR Empty interrupt will have already occurred

Solution The first character to be sent **may** have to be handled differently from other characters

- ▶ This also applies to the first character your program has to send – if you are asked to use interrupts on transmissions
- ▶ In our setting this is fairly easy to implement
    - ▶ Your application is a single process, thus there are no concurrency issues
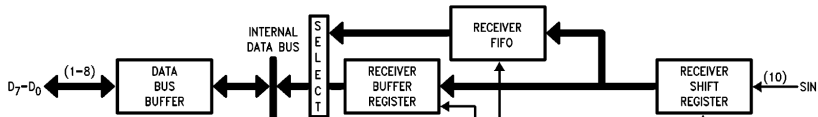
Buffering

Problem What if characters arrive faster than the program is able to read them?

- ▶ This problem occurs more frequently in polled operation
- ▶ In interrupt-driven operation this is not so common, but ...

Solution Use buffering on device

- ▶ For compatibility reasons (legacy programs), the FIFO is disabled by default

UART FIFO

- ▶ This is a HW buffer in the UART
  - ▶ Need to enable them on the FIFO Control Register (FCR)
- ▶ The IH should read all the characters available in the FIFO
  - ▶ Reduces the number of interrupts
  - ▶ Reduces further the likelihood that a character will be lost
- ▶ The UART also has a FIFO for transmission
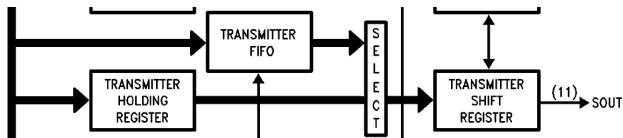
- ▶ Reduces the number of interrupts
  - ▶ Allows for faster communication, reduces idle time between chars to the minimum
- ▶ The size of both FIFOs is equal, but depends on the particular chip (16 bytes and 64 bytes are common)

UART FIFO Control Register (FCR)

Bit			Meaning
0			Set to 1 to enable both FIFOs
1			Set to 1 to clear all bytes in RCVR FIFO. Self-clearing.
2			Set to 1 to clear all bytes in the XMIT FIFO. Self-clearing
3			Not relevant for Lab 7
4			Reserved for future use
5			Enable 64 byte FIFO (for 16750 only)
7, 6			RCVR FIFO Trigger Level (Bytes) (for 16 byte FIFOs)
	0	0	1
	0	1	4
	1	0	8
	1	1	14

► Bits 7 and 6 allow to reduce the number of receiver ready interrupts

► The IIR contains also state information related to the FIFOs

Bit	Meaning
3	Character timeout: 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
5	Set to 1, if 64-byte FIFO enabled (for 16750 only);
7, 6	Set to 1, if bit 0 of FCR is set

Using the FIFOs

```
sys_outb(ser_port + SER_FCR, 0x??); // Enable FIFOs

sys_inb(ser_port + SER_IIR, &iir); // Check FIFO state

void ser_ih() { // serial port IH
    ...
    while( lsr & SER_RX_RDY ) { // Read all characters in FIFO
        ... // check errors
        sys_inb(ser_port + SER_DATA, &c);
        ... // "process" character read
        sys_inb(ser_port + SER_LSR, &lsr);
    }
```

Problem This solution is not application-independent. Why?
  ▶ Using a char variable will not work. Why?

Solution ???

Solution: Queue

- ▶ The IH puts the characters in the queue
- ▶ The program reads the characters off the queue
- ▶ In Lab 7, there is no possibility of interference

```
void ser_ih() { // serial port IH
    ...
    while( !queue_is_full(qptr) && (lsr & SER_RX_RDY)) {
        ...
    }
```

Question Should we also use a queue for transmission?

Lab 7: `test_int`/`test_poll`

What? Send/receive strings using the serial port in

- polled mode
- interrupt mode (in the second class only???)

Challenge Need to have both a sender and a receiver for testing ...

- Use executable I provide with my solution

How to test?

1. Use two VMs on the same PC, with the serial port of one connected to the serial port of another
   - Except for the number of bits per char, all the other communication parameters do not matter

Minix Notes

- Minix has a default IH for the serial port
- To prevent interference, in

  Polled operation Need to disable the interrupts in the UART

  Interrupt operation Need to subscribe the UART interrupts in exclusive mode

Lab 7: `test_fifo()`

- To be implemented only in the second class
- Program the UART to use the FIFOs either on transmission/reception
- Use FIFOs with either polling or interrupts
- You will be given the details in class
- Be prepared to develop an application independent IH

Further Reading

- National Semiconductor's PC16550D Data Sheet
- 8250 UART Programming chapter of the Serial Programming book from Wikibooks
- Beyond Logic's, Interfacing the Serial/RS232 Port Tutorial