Computer Labs: The PC's Serial Port

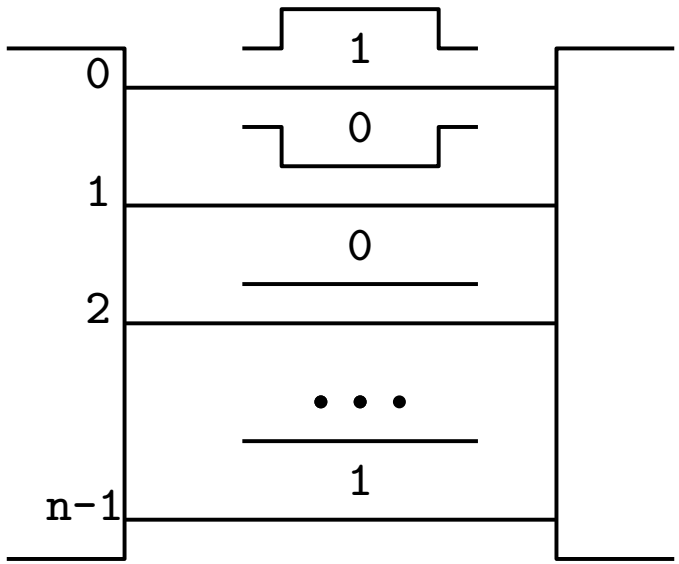
2º MIEIC

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

October 31, 2010

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)

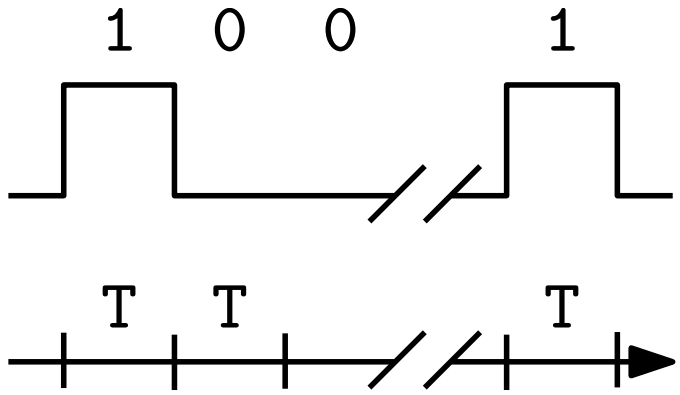
Parallel Communication **Signals** are sent simultaneously in **parallel** over several **channels**

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, 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 syncrhonization 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 special circuitry (Digital Phase-lock Loop) that uses the 0 to 1 and 1 to 0 transitions in the bit stream

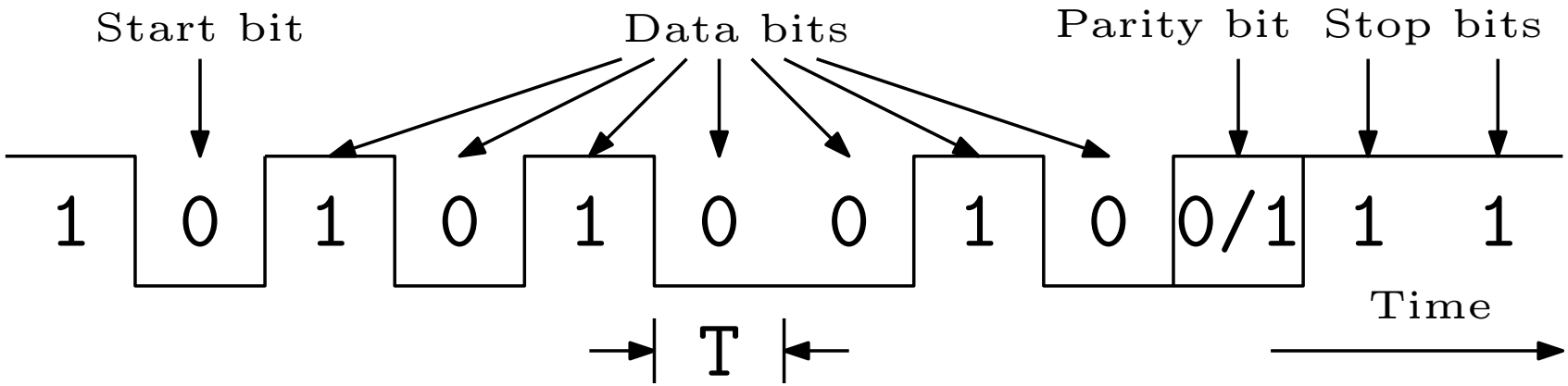
Asynchronous Comunication In which the clocks run independently

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.



Serial Communication on the PC

- Typically a PC has at least one serial port
  - This satisfies the RS-232 standard
  - Uses 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

UART Accessible (8-bit) Registers

Address	Read/Write	Mnem.	Description
0	R	DATA	Received character
	W	DATA	Character to transmit
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	DLLB	Divisor Latch Low Byte
1	R/W	DLHB	Divisor Latch High Byte

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

`DLLB` and `DLHB` 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
	1	1	1	Parity bit is 0
6				Break control: sets serial output to 0 (low)
7 (DLAB)			1	Divisor Latch Access
			0	Data access

Line Status Register (LSR)

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 leas 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)

st = inportb(ser_port + SER_LSR);
if ( st & (SER_OVERRUN_ERR | SER_PARITY_ERR | SER_FRAME_ER
     ... // 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`** (we'll call it `Transmitter Ready` for short)

Polled Operation: Transmission

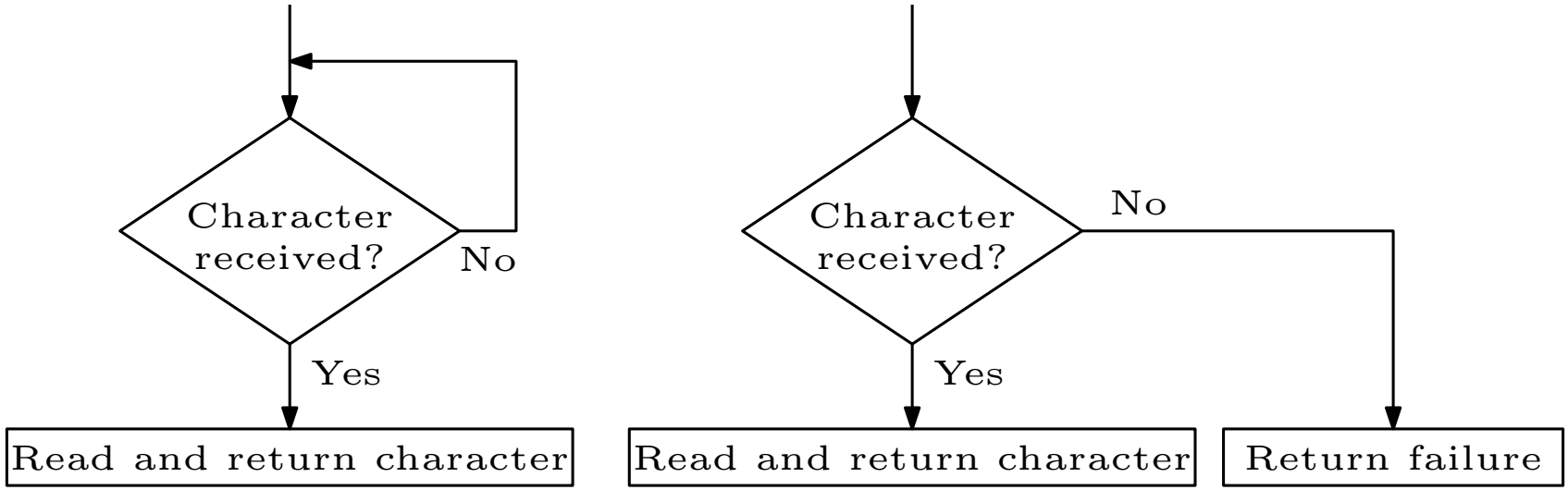
```
#define SER_LSR     5
#define SER_DATA    0
#define SER_TX_RDY (1<<5)

/* busy wait for transmitter ready */
while((inportb(ser_port + SER_LSR) & SER_TX_RDY) == 0)
    ;

/* ready for sending the next character */
outport(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

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 Receive 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

Enabling Serial Port Interrupts

IMPORTANT In some motherboards, the Aux Output 2 of the UART is connected to a register that further controls interrupts from the UART

- In addition to enable interrupts on the PIC and on the UART;
- Need to activate output _OUT2, via bit 3 of the Modem Control Register (MCR)

```
/* Enable interrupts by UART */
outportb(ser_port + SER_IER,
         SER_RX_INT_EN | SER_TX_INT_EN);
/* Enable interrupts in the motherboard */
outportb(ser_port + SER_MCR, SER_OUT2);
/* Unmask IRQ ? in the PIC */
ouportb(PIC1_MASK, inportb(PIC1_MASK) & ~(1<<?));
```

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

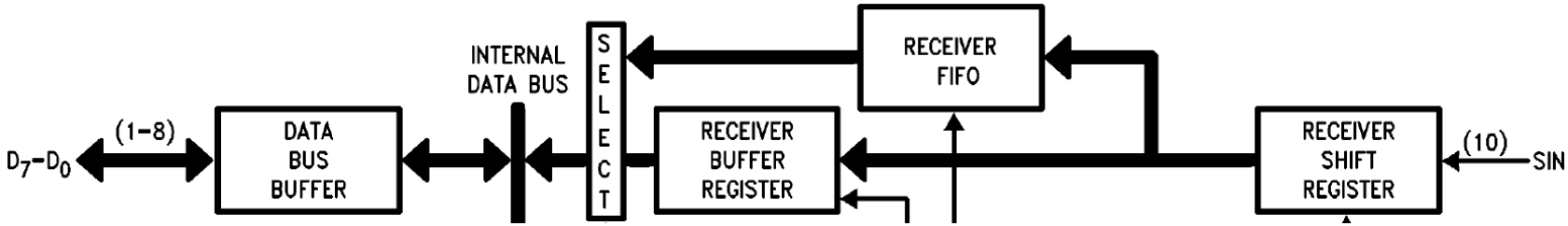
```
void ser_isr() {
    st = inportb(ser_port + SER_IIR);
    if( st & SER_INT_PEND ) {
        switch( st & INT_ID ) {
        case SER_RX_INT:
            ... /* read received character */
        case SER_TX_INT:
            ... /* put character to sent */
        case SER_RX_ERR:
            ... /* notify upper level */
        case SER_XXXX:
            ... /* depends on XXX */
        }
    }
    outportb(...); /* send EOI to PIC */
}
```

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 ...

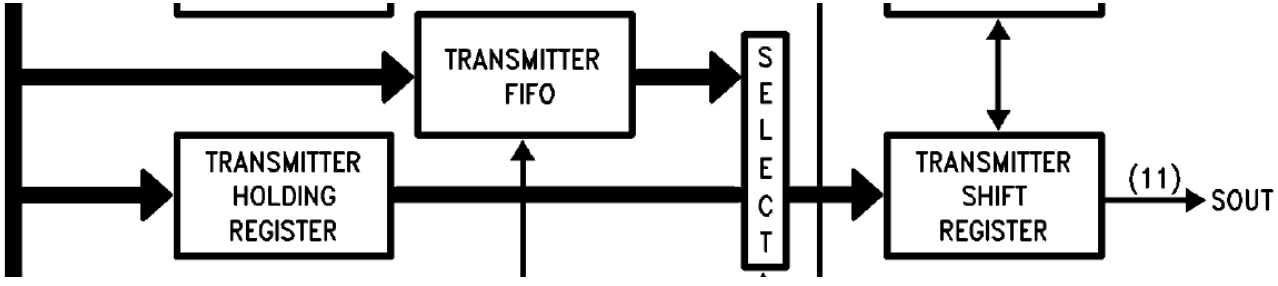
Solution Use buffering on device



- 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 ISR 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)
	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

```
#define SER_RX_RDY (1 << 0)
#define SER_DATA 0
#define SER_FCR  2 // Write only
#define SER_IIR  2 // Read only
#define SER_LSR  5

// Enable FIFOs
outportb(ser_port + SER_FCR, 0x??);
// Check FIFO state
st = inportb(ser_port + SER_IIR);

void ser_isr() { // serial port ISR
    ...
    do { // Read all characters in the FIFO
        ch = inportb(ser_port + SER_DATA);
        ...
   } while( inportb(ser_port + SER_LSR) & SER_RX_RDY);
```

Problem Reading all characters to the same variable does not make much sense. Why?

Solution ???

Solution: Queue

- The ISR puts the characters in the queue
- The program reads the characters off the queue
- Access to the queue is a critical section

```
void ser_isr() { // serial port ISR
    ...
    while( !queueFull()
           && inportb(ser_port + SER_LSR) & SER_RX_RDY) {
        queuePut(inportb(ser_port + SER_DATA));
   }
```

Question Should we also use a queue for transmission?

Modem Control Register (MCR)

Controls the interface with a MODEM

0	If set, activates the Data Terminal Ready output, signaling that the computer is ready for communicating
1	If set, activates the Request to Send output, indicating that there are data for transmission
2	If set, activates the Output 1 line
3	If set, activates the Output 2 line
4	If set, activates local loopback, which may be used for diagnostic
5,6,7	Reserved

There is a potential problem with the loop back mode:

> *and the four MODEM Control outputs (_DTR, _RTS, _OUT1 and _OUT2) are internally connected to the four MODEM Control inputs*

if output _OUT2 is indeed used for controlling UART interrupts on the motherboard.

- Also its is not clear here which outputs are connected to which inputs

Modem Status Register (MSR)

Provides state information as well as state change information regarding input lines from a MODEM

4	Set when the Clear To Send (_CTS) input is active. In loopback mode, this bit is equal to the RTS bit of the MCR.
5	Set when the Data Set Ready (_DSR) input is active. In loop-back mode, this bit is equal to the DTR bit of the MCR
6	Set when the Ring Indicator _RI input is active. In loopback mode this bit is equal to the OUT1 bit of the MCR
7	Set when the Data Carrier Detect _DCD is active. In loopback mode, this bit is equal to the OUT2 bit of the MCR

- Bits 0 to 3 provide state change information regarding these 4 inputs.
  - I.e., these bits are set to 1 whenever the corresponding control input from the MODEM changes state. They are cleared whenever the MCR is read

Reading the Keyboard without Blocking

- In its simplest version, Lab 7's program must attend to two events:

  Keyboard input: In this case, it will have to read the character and output it to the serial port

  Serial port input: In this case, it will have to read the character from the serial port and print it to the screen

Problem The standard input functions of the C library block the process until there is some input (key pressed)

- As a result, characters received by the UART may be overwritten

Solution Use the `kbhit()` which returns whether there is an input character

```
#include <pc.h>

    char c = 0;
    if( kbhit() )
        c = getch();
```

Further Reading

- National Semiconductor's PC16550D Data Sheet
- Beyond Logic's, Interfacing the Serial/RS232 Port Tutorial