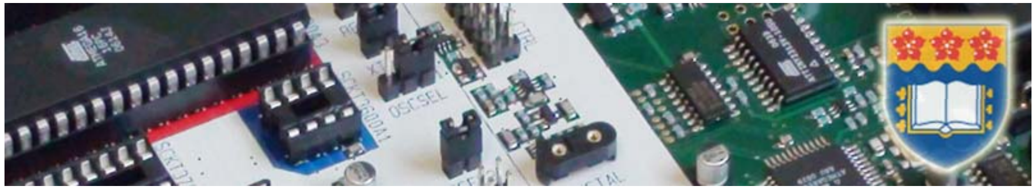


ECTE333
Lecture 8 - Serial Communication

School of Electrical, Computer and Telecommunications Engineering
University of Wollongong
Australia

ECTE333's schedule

Week	Lecture (2h)	Tutorial (1h)	Lab (2h)
1	L7: C programming for the ATMEL AVR		
2		Tutorial 7	Lab 7
3	L8: Serial communication		
4		Tutorial 8	Lab 8
5	L9: Timers		
6		Tutorial 9	Lab 9
7	L10: Pulse width modulator		
8		Tutorial 10	Lab 10
9	L11: Analogue-to-digital converter		
10		Tutorial 11	Lab 11
11	L12: Revision lecture		
12			Lab 12
13	L13: Self-study guide (no lecture)		
Final exam (25%), Practical exam (20%), Labs (5%)			

Lecture 8's sequence

8.1 Serial communication — The basics

8.2 Serial communication in ATmega16

8.3 Example application & Debugging tool

An application of serial communication

www.elec.uow.edu.au/avr/ecte333/pan_tilt_camera.mp4

An STK500 board is programmed to control a pan-tilt video camera, via a serial connection. In this lecture, you'll learn to create such a program.

8.1 Serial communication — The basics

- Computers transfer data in two ways: parallel and serial.
  - **Parallel**: Several data bits are transferred simultaneously, e.g. to printers and hard disks.
  - **Serial**: A single data bit is transferred at one time.

- Advantages of serial communication: longer distances, easier to synchronise, fewer IO pins, and lower cost.

- Serial communication often requires
  - **Shift registers**: convert a byte to serial bits and vice versa.
  - **Modems**: modulate/demodulate serial bits to/from audio tones.

Synchronous versus asynchronous

- **Synchronous serial communication**
  - The clocks of the sender and receiver are synchronised.
  - A block of characters, enclosed by synchronising bytes, is sent at a time.
  - Faster transfer and less overhead.
  - **Examples**: serial peripheral interface (SPI) by Motorola,
    binary synchronous communication (BISYNC) by IBM.

- **Asynchronous serial communication**
  - The clocks of the sender and receiver are not synchronised.
  - One character (8 or 7 bits) is sent at a time, enclosed between a start bit and one or two stop bits. A parity bit may be included.
  - **Examples**: RS-232 by Electronic Industries Alliance,
    USART of ATmega16

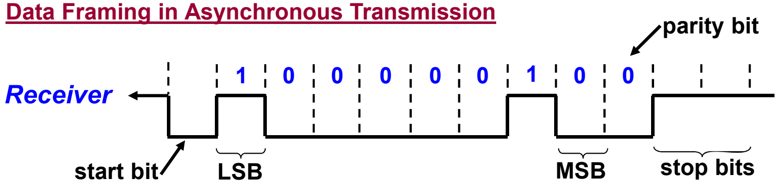
Data framing examples

Data Framing in Synchronous BISYNC

SYN	SYN	STX	DATA FIELD	ETX	BCC	PAD

BISYNC Control Characters
SYN (16h): synchronisation
STX (02h): start of text
ETX (03h): end of text
BCC: block checksum char
PAD (FFh): end of frame block

Data Framing in Asynchronous Transmission



Sending character "A" (41h = 0100 0001)
8-bit data, 1 start bit, 2 stop bits, even-parity

Serial communication terminology

- **Baud rate**: The number of bits sent per second (bps).
  Strictly speaking, it is the number of signal changes per second.

- **Parity bit**: A single bit for error checking, sent with data bits to make the total number of 1's
  - even (for even parity), or
  - odd (for odd parity).

- **Start bit**: to indicate the start of a character. Its typical value is 0.

- **Stop bit**: to indicate the end of a character. Its typical value is 1.

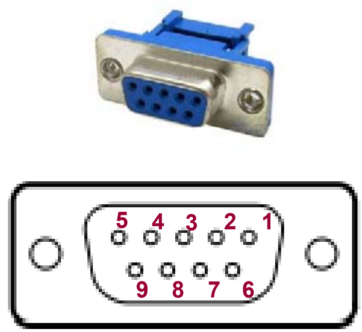
The RS-232 standard

- The RS-232 (latest revision RS-232E) is a widely used standard for serial interfacing.
- It covers four main aspects.
  - **Electrical**: voltage level, rise and fall time, data rate, distance.
  - **Functional**: function of each signal
  - **Mechanical**: number of pins, shape & dimension of connectors.
  - **Procedural**: sequence of events for transmitting data.

The RS-232 standard

- It defines 25-pin D connectors. In many cases, 9-pin connectors are used.
- RS-232 specifies the baud rate up to 20Kbps, and the cable length up to 15m. In practice, it supports up to 56Kbps & 30m of shielded cables.

Male DB9 connector



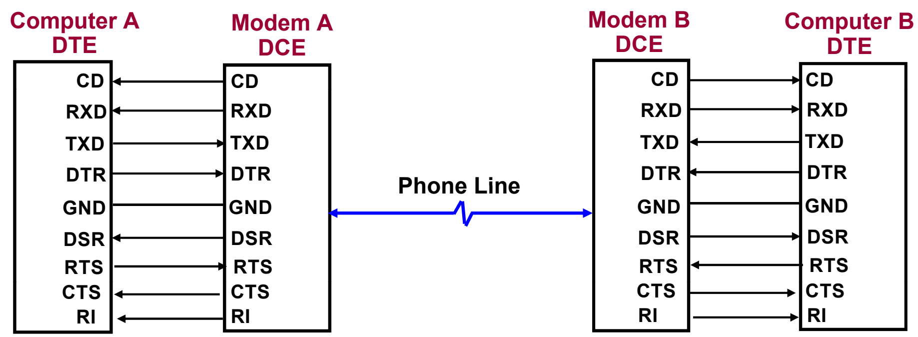
Female DB9 connector

RS-232 9-pin connector

Pin	Name	Description	
1	$\overline{CD}$	Carrier Detect:	DCE has detected a carrier tone
2	RXD	Received Data:	incoming data from DCE
3	TXD	Transmit Data:	outgoing data to DCE
4	DTR	Data Terminal Ready:	DTE is connected and turned on
5	GND	Ground	
6	$\overline{DSR}$	Data Set Ready:	DCE is connected and turned on
7	$\overline{RTS}$	Request To Send:	DTE has data to send
8	$\overline{CTS}$	Clear To Send:	DCE can receive data
9	RI	Ring Indicator:	synchronised with the phone's ringing tone

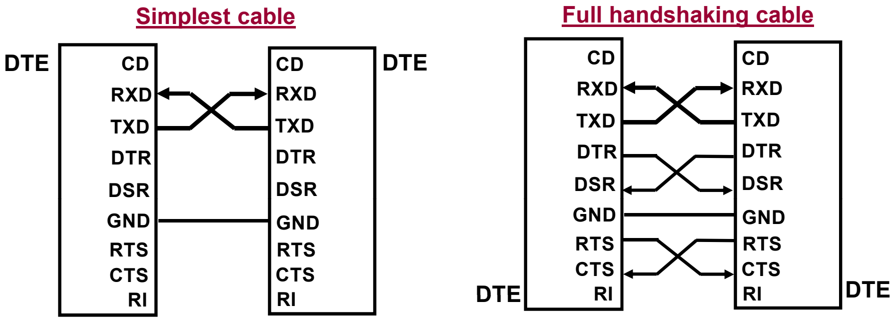
- **Data Terminal Equipment** (DTE) essentially refers to the computer.
- **Data Communication Equipment** (DCE) essentially refers to a remote device or modem.
- These terms are needed to explain the pin functions.

Modem connection



- RS-2322 was originally used with modems to connect two PCs over the public phone lines.
- When computer A has data to send, it assert its RTS pin.
- Modem A will assert its CTS when it is ready to receive.
- Computer A transmits data through its TXD.

Null-modem connection



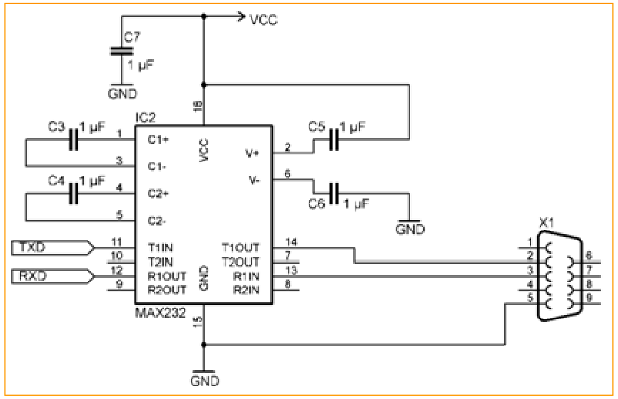
- RS-232 is now mainly used to connect a microcontroller with PC or peripheral devices (e.g. GPS receiver, infrared range finder, camera).
- This configuration is known as null-modem.
- Key idea:
  - Connect *pin TXD of a DTE* with pin *RXD of the other DTE*.
  - Wire other pins to support flow control.

RS-232 interface and MAX232 chip

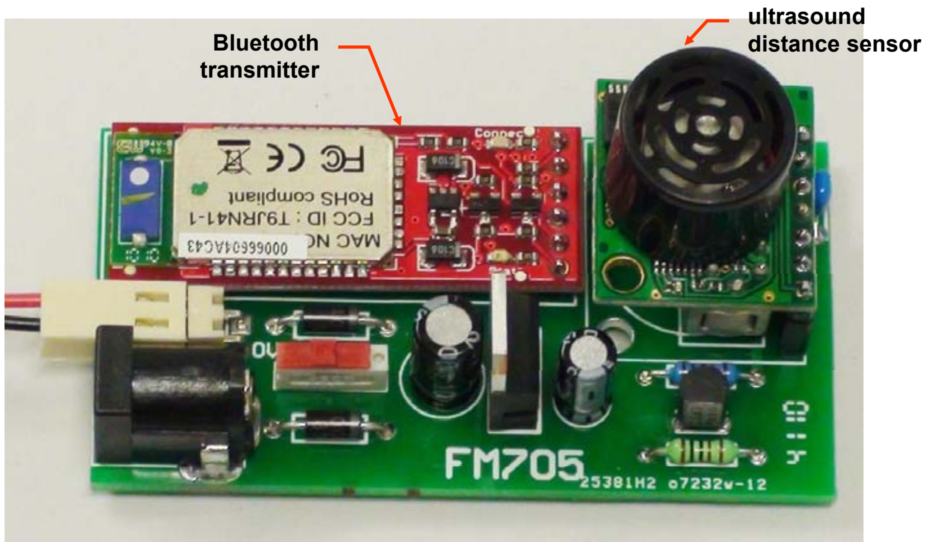
- Compared to TTL in computer electronics, RS-232 interface uses different voltage levels.

Logic	RS-232 levels	TTL levels
1	[-15V, -3V]	[+2V, +5V]
0	[+3V, +15V]	[0V, +0.8V]

- A level converter is required between RS-232 interface and TXD/RXD pins of microcontroller.
- MAX232 chip is often used for this purpose.

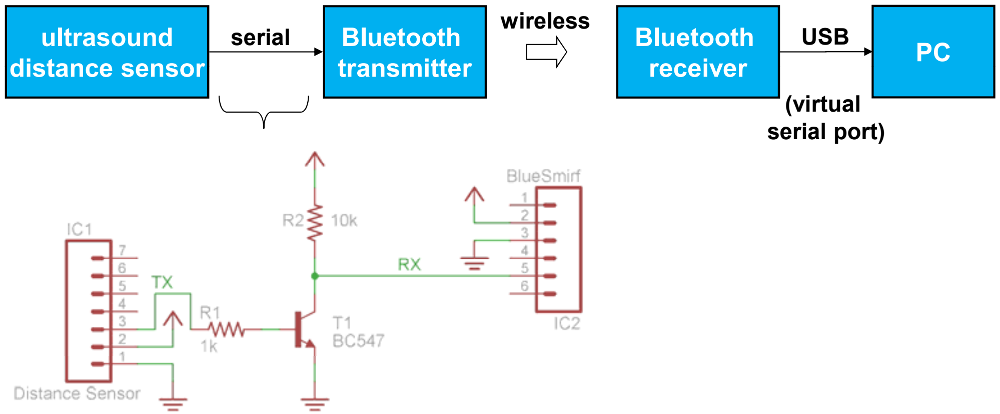


Serial communication — An example



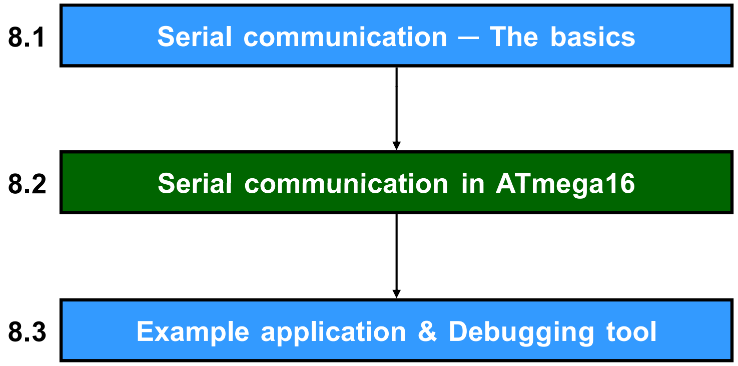
This device measures distance to the nearest obstacle, and transmits it via Bluetooth to PC.

Serial communication — An example



- The sensor sends data via a serial interface to Bluetooth transmitter.
- A Bluetooth receiver, connected to a PC, is configured as a serial port.
- A demo, created by Adrian Herrera, is shown in the lecture.

Lecture 8's sequence



8.2 Serial communication in ATmega16

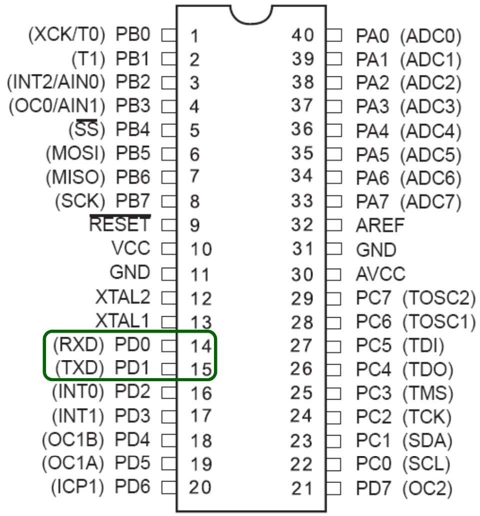
- ATmega16 has 3 subsystems for serial communication.
  - Universal Synchronous & Asynchronous Receiver & Transmitter (**USART**)
  - Serial Peripheral Interface (**SPI**)
  - Two-wire Serial Interface (**TWI**)
- **USART**:
  - We focus on this subsystem in this lecture.
  - Supports full-duplex mode between two devices.
  - Typically used in asynchronous communication.
  - Start bit and stop bit are used for each byte of data.

8.2 Serial communication in ATmega16

- **Serial Peripheral Interface (SPI)**
  - The receiver and transmitter share a common clock line.
  - Supports higher data rates.
  - The transmitter is designated as the master, the receiver as the slave.
  - Examples of devices using SPI: liquid crystal display, high-speed analogue-to-digital converter.
- **Two-wire Serial Interface (TWI)**
  - Connect several devices such as microcontrollers and display boards, using a two-wire bus.
  - Up to 128 devices are supported.
  - Each device has a unique address and can exchange data with other devices in a small network.

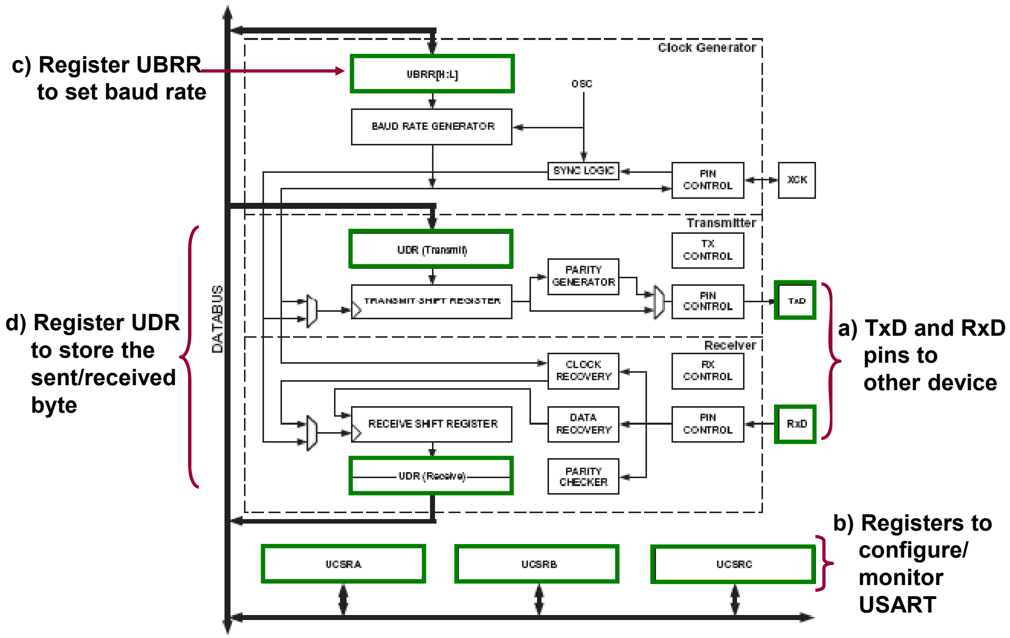
Serial USART — An overview

- USART of the ATmega16 supports
  - baud rates from 960bps to 57.6kbps,
  - character size: 5 to 9 bits,
  - 1 start bit,
  - 1 or 2 stop bits,
  - optional parity bit (even or odd parity).
- Common baud rates are 19200, 9600, 4800, 2400, and 1200 bps.



ATmega16 chip

Serial USART — Block diagram



Serial USART — Hardware elements

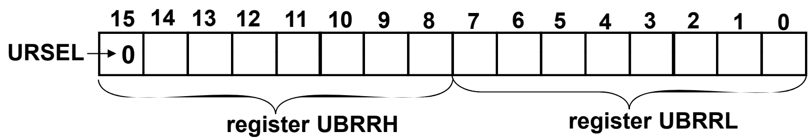
- USART Clock Generator:
  - to provide clock source.
  - to set baud rate using UBRR register.
- USART Transmitter:
  - to send a character through TxD pin.
  - to handle start/stop bit framing, parity bit, shift register.
- USART Receiver:
  - to receive a character through RxD pin.
  - to perform the reverse operation of the transmitter.
- USART Registers:
  - to configure, control, and monitor the serial USART.

Serial USART — Three groups of registers

- USART Baud Rate Registers
  - UBRRH and UBRRL
- USART Control and Status Registers
  - UCSRA
  - UCSRB
  - UCSRC
- USART Data Registers
  - UDR
- Understanding these registers is essential in using the serial port. Therefore, we'll study these registers in depth.

USART Baud Rate Registers

- Two 8-bit registers together define the baud rate.



$$\text{baud rate} = \frac{\text{system clock frequency (Hz)}}{16 \times (\text{UBRR} + 1)}$$

$$\text{UBRR} = \frac{\text{system clock frequency (Hz)}}{16 \times \text{baud rate}} - 1$$

- Example: Find UBRR registers for baud rate of 1200bps, assuming system clock is 1MHz.
  - UBRR = 1000000/(16 × 1200) — 1 = 51_d = 0033_H.
  - Therefore, UBRRH = 00_H and UBRRL = 33_H.
  - C code: `UBRRH = 0x00; UBRRL = 0x33;`

USART Control and Status Register A: UCSRA

7	6	5	4	3	2	1	0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- MPCM: 1 to enable multi-processor com mode
- U2X: 1 to double the transmission speed
- PE: 1 when there is parity error
- DOR: 1 when there is data overun
- FE: 1 when there is frame error
- UDRE: 1 when USART data register is empty
- TXC: 1 when no new data in transmit buffer (tx complete)
- RXC: 1 when receive buffer has unread data (rx complete)

USART Control and Status Register B: UCSRB

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

- TXB8: Tx extra data bit for 9-bit character size
- RXB8: Rx extra data bit for 9-bit character size
- UCSZ2: bit UCSZ2 to decide character size
- TXEN: 1 to enable USART transmitter: Pin D.1 = TXD pin
- RXEN: 1 to enable USART receiver: Pin D.0 = RXD pin
- UDRIE: 1 to enable USART Data Register Empty Interrupt
- TXCIE: 1 to enable TX Complete Interrupt, valid only if Global Interrupt Flag = 1 and TXC = 1
- RXCIE: 1 to enable RX Complete Interrupt, valid only if Global Interrupt Flag = 1 and RXC = 1

USART Control and Status Register C: UCSRC

7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

- UCPOL: Clock polarity, used with synchronous
- UCSZ1, UCSZ0: Used with UCSZ2 to select character size
- USBS: To select stop bit modes: 0 → 1 stop bit, 1 → 2 stop bits
- UPM1, UPM0: To select parity mode: 00 no parity, 10 even party, 11 odd parity
- UMSEL: To select USART modes: 0 asynchronous, 1 synchronous
- URSEL: Must be set to 1 to write to UCSRC. Note: UCSRC and UBRRH share same location.

Setting character size

- Character size (5, 6, 7, 8, 9) is determined by three bits
  - bit UCSZ2 (in register UCSRB),
  - bit UCSZ1 and bit UCSZ0 (in register UCSRC).
- Example: For a character size of 8 bits, we set

  UCSZ2 = 0, UCSZ1 = 1, and UCSZ0 = 1.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

USART Data Register

- Register UDR is the buffer for characters sent or received through the serial port.

- To start sending a character, we write it to UDR:

```
unsigned char data;
data = 'a';
UDR = data;        // start sending character
```

- To process a received character, we read it from UDR:

```
unsigned char data;
data = UDR;       // this will clear UDR
```


Serial USART — Main tasks

- There are 4 main tasks in using the serial port.

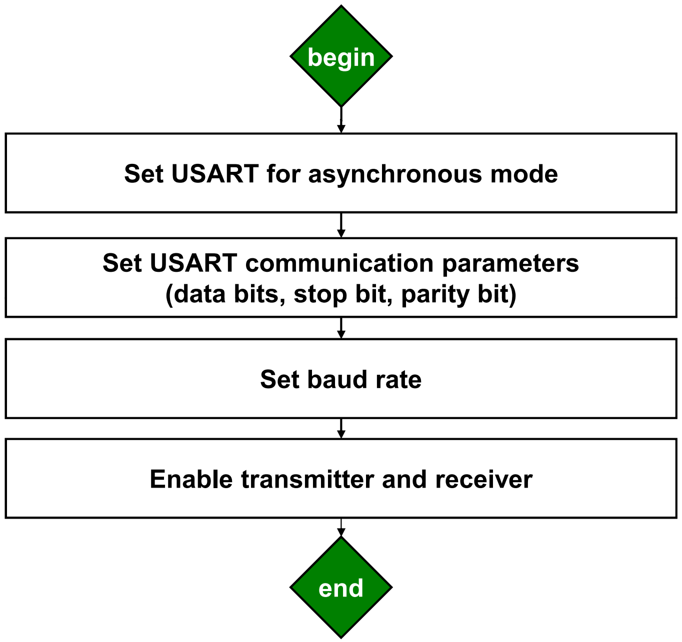
8.2.1 Initialising the serial port.

8.2.2 Sending a character.

8.2.3 Receiving a character.

8.2.4 Sending/receiving a formatted string.

8.2.1 Initialising serial port

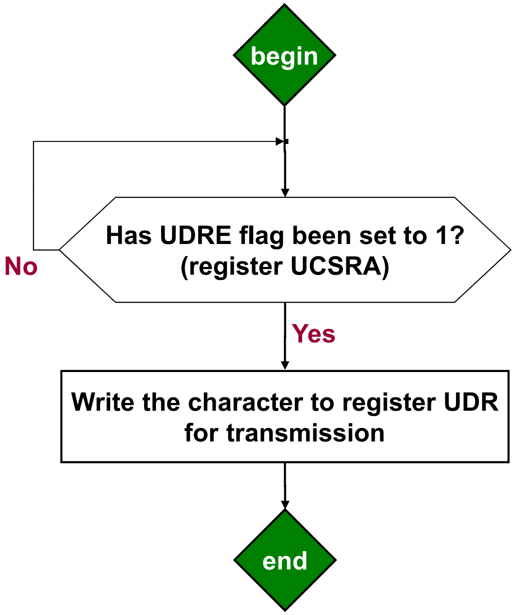


Initialising serial port — Example

Initialise serial port of ATmega16 to baud rate 1200 bps, no parity, 1 stop bit, 8 data bits. Assume a clock speed of 1MHz.

```
void USART_init(void){
   // Asynchronous mode, no parity, 1 stop bit, 8 data bits
   UCSRC = 0b10000110;

   // Normal speed, disable multi-proc
   UCSRA = 0b00000000;

   // Baud rate 1200bps, assuming 1MHz clock
   UBRRL = 0x33;
   UBRRH = 0x00;

   // Enable Tx and Rx, disable interrupts
   UCSRB = 0b00011000;
}
```

8.2.2 Sending a character



Sending a character — Example

Write a C function to send a character through ATmega16 serial port.

```
void USART_send(unsigned char data){
   // Wait until UDRE flag = 1
   while ((UCSRA & (1<<UDRE)) == 0x00){;}
   // Write char to UDR for transmission
   UDR = data;
}
```

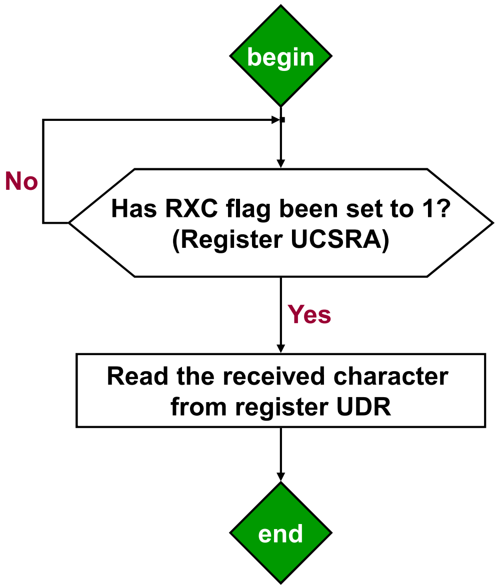
UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
1<<UDRE	0	0	1	0	0	0	0	0
bit-wise AND	0	0	UDRE	0	0	0	0	0

Constant UDRE is defined in avr/io.h

`#define UDRE 5`

Bit-wise AND returns zero if bit UDRE = 0

8.2.3 Receiving a character



Receiving a character — Example

Write a C function to receive a character via ATmega16 serial port.

```
unsigned char USART_receive(void){
   // Wait until RXC flag = 1
   while ((UCSRA & (1<<RXC)) == 0x00){;}
   // Read the received char from UDR
   return (UDR);
}
```

UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
1<<RXC	1	0	0	0	0	0	0	0
bit-wise AND	RXC	0	0	0	0	0	0	0

Constant RXC is defined in avr/io.h

`#define RXC 7`

Bit-wise AND returns zero if bit RXC = 0

8.2.4 Sending/receiving a formatted string

- In ANSI C, the header file <stdio.h> has two functions for formatted strings: printf and scanf.

- Function printf sends a formatted string to the standard output device, which is usually the video display.

```
unsigned char a, b;
a = 2; b = 3;
printf("first = %d, second = %d, sum = %d", a, b, a + b);
```

- Function scanf reads a formatted string from the standard input device, which is usually the keyboard.

```
unsigned char a, b;
scanf("%d %d", &a, &b); // get integers a, b from input string
```


Sending/receiving formatted strings

- Being able to send/receive formatted strings through a serial port is useful in microcontroller applications.

- To this end, we configure the serial port as the standard input and output devices.

- General steps:

  1) Write two functions to send and receive a character via serial port.

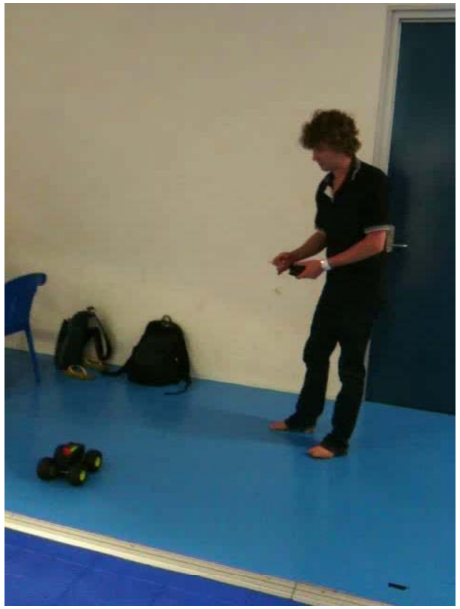
  2) In main(), call fdevopen() to set the two functions as the handlers for standard output and input devices.

  3) Use printf/scanf as usual. Formatted strings will be sent/received via serial port.

Sending/receiving formatted strings — Example

```
#include <avr/io.h>
#include <stdio.h>

int USART_send(char c, FILE *stream){
   // wait until UDRE flag is set to logic 1
   while ((UCSRA & (1<<UDRE)) == 0x00){;}
   UDR = c; // Write character to UDR for transmission
}

int USART_receive(FILE *stream){
   // wait until RXC flag is set to logic 1
   while ((UCSRA & (1<<RXC)) == 0x00){;}
   return (UDR); // Read the received character from UDR
}

int main(void){
   unsigned char a;
    // … Code to initialise baudrate, TXD, RXD, and so on is not shown here
   // Initialise the standard IO handlers
   stdout = fdevopen(USART_send, NULL);
   stdin = fdevopen(NULL, USART_receive);

   // Start using printf, scanf as usual
   while (1){
       printf("\n\rEnter a = ");
       scanf("%d", &a); printf("%d", a);
   }
}
```


AVR Demo: Remote controller for car



ECTE350 Third-prize Trade Fair 2010.

ZigBee, accelerometer, ATmega16

(Jarod Chadwick et al.)

Lecture 8's sequence

8.1 Serial communication — The basics

8.2 Serial communication in ATmega16

8.3 Example application & Debugging tool

8.3 Example application

video

- The MCAM100 is a programmable pan-tilt video camera.
- It is controlled via a serial connection: 8 data bits, 1 stop bit, no parity bit, baud rate 9600bps.
- Sending character '4' or '6' turns the camera left or right, respectively.
- We'll write an ATmega16 program to rotate the camera repeatedly.

camera.c

```c
#include <avr/io.h>
void delay(void){
    for (int i = 0; i < 1000; i++)
         for (int j = 0; j < 100; j++)
                asm volatile("nop");
}

void USART_init(void){
    UCSRA = 0b00000010; // double speed, disable multi-proc
    UCSRB = 0b00011000; // Enable Tx and Rx, disable interrupts
    UCSRC = 0b10000110; // Asyn mode, no parity, 1 stop bit, 8 data bits
    // in double-speed mode, UBRR = Fclock/(8xbaud rate) - 1
    UBRRH = 0; UBRRL = 12; // Baud rate 9600bps, assuming 1MHz clock
}

void USART_send(unsigned char data){
    while ((UCSRA & (1<<UDRE)) == 0x00){;} // wait until UDRE flag = 1
    UDR = data; // Write character to UDR for transmission
}

int main(void) {
    unsigned char i;
    USART_init(); // initialise USART
    while (1) {
         for (i = 0; i < 10; i++){   // rotate left 10 times
            USART_send('4');
             delay();
         }
         for (i = 0; i < 10; i++){   // rotate right 10 times
             USART_send('6');
             delay();
         }
    }
}
```


Debugging tool: Hyper Terminal

- Sending/receiving data through serial port is useful for debugging a microcontroller program.
- A program for monitoring serial data is Hyper Terminal.

  It is built-in in Windows XP. For Windows 7, download it at:

  www.uow.edu.au/~phung/teach/ecte333/HyperTerminal.zip
- Hyper Terminal is used to
  - create a serial connection between the PC and the microcontroller.
  - send a text string to the microcontroller.
  - receive a text string sent from the microcontroller.
- For example, let's use Hyper Terminal to debug the program camera.c.

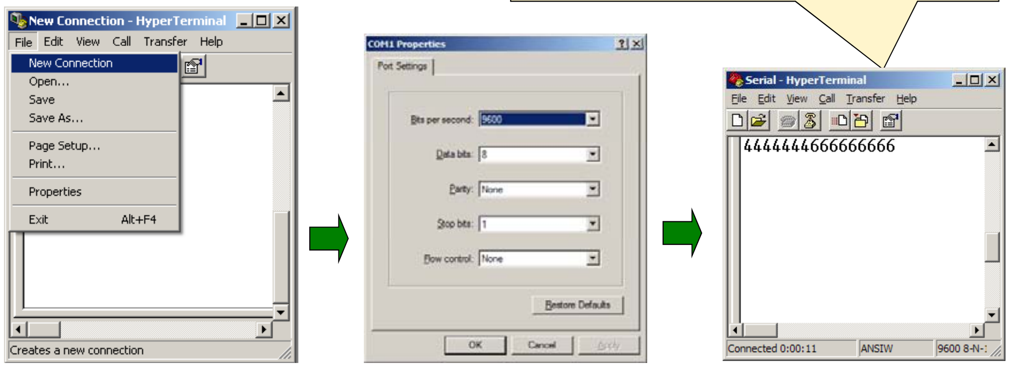
Debugging tool: Hyper Terminal

- **Step 1**:
  - Download and run camera.hex on the STK500 board.
  - Remove the serial cable from RS232 Control connector.
- **Step 2**:
  - Attach the serial cable to RS232 Spare connector.
  - Connect pin RXD RS232 Spare to pin D.0.
  - Connect pin TXD RS232 Spare to pin D.1.
- **Step 3**:
  - Start Hyper Terminal program.
  - Configure baud rate, parity, data bit, stop bit, flow control.

Debugging tool: Hyper Terminal

camera.c running on ATmega16:

```
USART_send('4');
…
USART_send('6');
```



1. Create a new connection that uses a COM port of PC
2. Enter COM port parameters that match the C program, & Connect
3. HyperTerminal displays all messages sent by the C program.

Lecture 8's summary

- What we learnt in this lecture:
  - Basics of serial communication.
  - Serial communication subsystem in ATmega16.
  - Using serial port to send/receive characters and formatted strings.
- What are the next activities?
  - Tutorial 8: 'Serial Communication' .
  - Lab 8: 'Serial Communication'
    - Complete the online Pre-lab Quiz for Lab 8.
    - Write programs for Tasks 1, 2, 3 of Lab 8.
    - See video demos of Lab 8: [avr]/ecte333/lab08_task1.mp4 [avr]/labs/lab08_task2.mp4, [avr]/ecte333/lab08_task3.mp4

Lecture 8's references

- Atmel Corp., 8-bit AVR microcontroller with 16K Bytes In-System Programmable Flash ATmega16/ATmega16L, 2007, [USART]. **Manual**
- S. F. Barrett and D. J. Pack, Atmel AVR Microcontroller Primer: Programming and Interfacing, 2008, Morgan & Claypool Publishers, [Chapter 8: Serial Communication Subsystem].
- D. V. Gadre, Programming and Customizing the AVR Microcontroller, McGraw-Hill, 2001, [Chapter 7: Communication Links for the AVR Processor].

Lecture 8's references

- M. Mazidi, J. Mazidi, R. McKinlay, "The 8051 microcontroller and embedded systems using assembly and C," 2nd ed., Pearson Prentice Hall, 2006, [Chapters 10].

- M. Mazidi and J. Mazidi, "The 8086 IBM PC and compatible computers," 4th ed., Pearson Prentice Hall, 2003, [Chapters 17].

- P. Spasov, "Microcontroller technology the 68HC11," 3rd ed., Prentice Hall, 1999, [Chapters 10].

- H. Huang, "MC68HC12 an introduction: software and hardware interfacing," Thomson Delmar Learning, 2003, [Chapter 9].

