ECTE333 Lecture 8 - Serial Communication

ECTE333’s schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Lecture (2h)</th>
<th>Tutorial (1h)</th>
<th>Lab (2h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L7: C programming for the ATMEL AVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Tutorial 7</td>
<td></td>
<td>Lab 7</td>
</tr>
<tr>
<td>3</td>
<td>L8: Serial communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Tutorial 8</td>
<td></td>
<td>Lab 8</td>
</tr>
<tr>
<td>5</td>
<td>L9: Timers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tutorial 9</td>
<td></td>
<td>Lab 9</td>
</tr>
<tr>
<td>7</td>
<td>L10: Pulse width modulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Tutorial 10</td>
<td></td>
<td>Lab 10</td>
</tr>
<tr>
<td>9</td>
<td>L11: Analogue-to-digital converter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Tutorial 11</td>
<td></td>
<td>Lab 11</td>
</tr>
<tr>
<td>11</td>
<td>L12: Revision lecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Lab 12</td>
</tr>
<tr>
<td>13</td>
<td>L13: Self-study guide (no lecture)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

null-modem connection

pan-tilt programmable video camera

TXD = Port D.1
RXD = Port D.0

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

<table>
<thead>
<tr>
<th>Data Framing in Synchronous BISYNC</th>
<th>BISYNC Control Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN SYN STX</td>
<td>SYN (16h): synchronisation</td>
</tr>
<tr>
<td></td>
<td>STX (02h): start of text</td>
</tr>
<tr>
<td></td>
<td>ETX (03h): end of text</td>
</tr>
<tr>
<td></td>
<td>BCC: block checksum char</td>
</tr>
<tr>
<td></td>
<td>PAD (FFh): end of frame block</td>
</tr>
</tbody>
</table>

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.

---

RS-232 9-pin connector

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

- 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-232 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.

- 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.
Null-modem connection

RS-232 interface and MAX232 chip

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

- 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

- 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.
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.

**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.
Serial USART — Block diagram

c) Register UBRR to set baud rate

d) Register UDR to store the sent/received byte

a) TxD and RxD pins to other device

b) Registers to configure/monitor USART

ECTE333 © Lam Phung, 2014. 21/49

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.

ECTE333 © Lam Phung, 2014. 22/49

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.

ECTE333 © Lam Phung, 2014. 23/49

USART Baud Rate Registers

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

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

**Example:** Find UBRR registers for baud rate of 1200bps, assuming system clock is 1MHz.

- UBRR = \( \frac{1000000}{(16 \times 1200) - 1} = 51_d = 0033_{H} \).
- Therefore, UBRRH = 00_H and UBRRL = 33_H.
- *C code:* UBRRH = 0x00; UBRRL = 0x33;

ECTE333 © Lam Phung, 2014. 24/49
**USART Control and Status Register A: UCSRA**

- **RXC**: Receive complete.
- **TXC**: Transmit complete.
- **UDRE**: USART Data Register Empty.
- **FE**: Frame error.
- **DOR**: Data overrun.
- **PE**: Parity error.
- **U2X**: 8 to 9-bit character size.
- **MPCM**: Multi-processor communication mode.

**Notes:**
- 1 to enable multi-processor com mode
- 0 to double the transmission speed
- 0 when there is parity error
- 0 when there is data overrun
- 1 when USART data register is empty
- 0 when no new data in transmit buffer (tx complete)
- 1 when receive buffer has unread data (rx complete)

---

**USART Control and Status Register B: UCSRB**

- **RXCIE**: USART receiver: Pin D.0 = RXD pin
- **TXCIE**: USART transmitter: Pin D.1 = TXD pin
- **UDRIE**: USART Data Register Empty Interrupt
- **RXEN**: To enable USART receiver: Pin D.0 = RXD pin
- **TXEN**: To enable USART transmitter: Pin D.1 = TXD pin
- **UCSZ2**: Rx extra data bit for 9-bit character size
- **TXB8**: Tx extra data bit for 9-bit character size
- **RXB8**: Rx extra data bit for 9-bit character size
- **UCSZ0**: to decide character size

**Notes:**
- 1 to enable USART transmitter: Tx complete interrupt, valid only if Global Interrupt Flag = 1 and TXC = 1
- 1 to enable USART receiver: Rx complete interrupt, valid only if Global Interrupt Flag = 1 and RXC = 1
- 1 to enable USART Data Register Empty interrupt
- 1 to enable TX Complete Interrupt, valid only if Global Interrupt Flag = 1 and TXC = 1

---

**USART Control and Status Register C: UCSRC**

- **URSEL**: Used with UCSZ2 to select character size
- **UMSEL**: Clock polarity, used with synchronous
- **UPM1/UPM0**: Used with UCSZ2 to select character size
- **USBS**: To select stop bit modes: 0 → 1 stop bit, 1 → 2 stop bits
- **UCSZ1/UCSZ0**: To select parity mode: 0 = no parity, 10 = even parity, 11 = odd parity
- **UCPOL**: To select UART modes: 0 = asynchronous, 1 = synchronous

**Notes:**
- 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.

<table>
<thead>
<tr>
<th>UCSZ2</th>
<th>UCSZ1</th>
<th>UCSZ0</th>
<th>Character Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5-bit</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6-bit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7-bit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8-bit</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9-bit</td>
</tr>
</tbody>
</table>
**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:
  ```c
  unsigned char data;
  data = 'a';
  UDR = data;  // start sending character
  ```

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

```
void USART_init(void){
  // Asynchronous mode, no parity, 1 stop bit, 8 data bits
  UCSRC = 0b10000110;
  UCSRA = 0b00000000;
  UBRRH = 0x00;
  UBRRL = 0x33;
  UCSR = 0b00000000;
  UCSR = 0b00000000;
  // Enable Tx and Rx, disable interrupts
  UCSR = 0b00000000;
}
```
### 8.2.2 Sending a character

**begin**

Has UDRE flag been set to 1? (register UCSRA)

**No**

**begin**

Has RXC flag been set to 1? (Register UCSRA)

**No**

Read the received character from register UDR

**end**

**Yes**

Write the character to register UDR for transmission

**end**

### Sending a character – Example

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

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

<table>
<thead>
<tr>
<th>UCSRA</th>
<th>RXC</th>
<th>TXC</th>
<th>UDRE</th>
<th>FE</th>
<th>DOR</th>
<th>PE</th>
<th>U2X</th>
<th>MPCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;&lt;UDRE</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

**begin**

Has RXC flag been set to 1? (Register UCSRA)

**No**

**begin**

Has UDRE flag been set to 1? (register UCSRA)

**No**

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

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

<table>
<thead>
<tr>
<th>UCSRA</th>
<th>RXC</th>
<th>TXC</th>
<th>UDRE</th>
<th>FE</th>
<th>DOR</th>
<th>PE</th>
<th>U2X</th>
<th>MPCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;&lt;RXC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Constant RXC is defined in avr/io.h

#define RXC 7

Bit-wise AND returns zero if bit RXC = 0

### Receiving a character – Example
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.

```c
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.

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

Sending/receiving formatted strings — Example

```c
#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 I/O handlers
    stdout = fdevopen(USART_send, NULL);
    stdin = fdevopen(NULL, USART_receive);

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

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.

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

- **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
  UCSR0B = 0b00011000; // Enable Tx and Rx, disable interrupts
  UCSR0C = 0b10000110; // Asynchronous mode, no parity, 1 stop bit, 8 data bits
  // in double-speed mode, UBRR = Fclock/(8*baud rate) - 1
  UBRRH = 0; UBRRL = 12; // Baud rate 9600bps, assuming 1MHz clock
}

void USART_send(unsigned char data){
  while ((UCSR0A & (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.

---

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]/ecte333/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].


- D. V. Gadre, Programming and Customizing the AVR Microcontroller, McGraw-Hill, 2001, [Chapter 7: Communication Links for the AVR Processor].
Lecture 8’s references


