MCU Lab 4 - Week 10
MCU: Serial Communication - UART
[2.5 Marks Total]

This computer lab will introduce the serial communication mechanisms in Atmega328P microcontroller. UART (Universal (synchronous) asynchronous receiver and transmitter) is one of the most famous serial communication peripherals due to its simplicity and fewer connections (minimal three wires). UART provides the capability of displaying the outputs to the host computer (using a serial cable) and debugging (more powerful than blinking LEDs). You can integrate UART into your project to report the system status or getting user commands.

1. Objectives

- To configure UART to transmit data using a polling method.
- To configure UART to receive a byte using a polling method.
- To use the interrupt service routine (ISR) and a transmit buffer to send bytes in an interrupt mode.

2. Marking and Due Date

- Students can form a group (maximum of **two** students) to complete the lab tasks but the demonstration will be assessed **individually**. There is a total of **2.5 marks** allocated to this lab. You need to complete the tasks during the lab session or at latest before the following week lab (in which case, students need to upload a short video to Canvas capturing the operational tasks).

3. Activity #1: Preparation

- Connect a potentiometer to the PC0 pin (or A0 in UNO).
- We will use the USB connection between the UNO board and your computer. The VSCode program supports a serial terminal (a “plug”-like icon in the blue bottom panel) which will be used to test the communication with Atmega328P. See Appendix for more details.

4. Activity #2: Polling-based Transmission

- Download “***mcu4.cpp***” from Canvas.
- Configure the UART in a simple protocol: **9600-8-N-1** (Baud rate, number of data bits, Number of parity bits, number of stop bit, respectively).
- Using the polling method, program the code, so that the Serial Monitor (in VSCode) displays the following message at 2 Hz:
    - **S2021 EMS SID: 12345678, ADC Reading: XXXX**

- where 12345678 should be replaced to your student number, and XXXX is the ADC output (integer value) of the potentiometer. *sprintf*() function in <stdio.h> can be utilised for this task.

- Show the results to get marked:

Demonstration	**Marking**
Polling-based UART Transmission with ADC reading	/1 mark

5. Activity #3: Polling-based UART Receive and Control

- Using the polling-method, program the code so that the UART takes a byte of keyboard input (the first letter of your family name, in lower case) from the Serial Monitor and ***toggles*** the transmission.
  - Keyboard input: 'a' (replace to the first letter of your family name) – toggles the sending the SID and ADC data. That is, in the next input, it will resume (toggle) sending.
  - Any other keys do not affect the transmission.

- (Extension) Modify the codes used in Activity 2, so that interrupt service routine (ISR) can transmit the data using a transmission buffer. The display format is the same as the polling-based Task 1, but the ISR codes should be checked by tutors. You can use ISR (USART_UDRE_vect) or ISR (USART_TXC_vect) for transmission (they are similar but the operation is slightly different).
- Show the results to get marked:

Demonstration	**Marking**
Polling-based UART Receive and Control the transmission	/1 mark
(Extension) Interrupt-based UART Transmission with ADC reading	/0.5 marks

Appendix A: Serial Monitor in VSCode

VSCode has a serial terminal emulator, called Miniterm, mostly for debugging purpose. You can launch the program from the PlatformIO control panel (blue) as shown in the figure below. It will show the current serial port setup, that is (9600,8,N,1) as shown in the second figure. Two things need your attention:

- As UART is asynchronous, this protocol should be met in both sides of transmission. Thus you need to program your code using the same protocol.
- Atemga328P has only UART which is shared for program uploading with USB-bridge. Whenever, you have finished using the serial monitor, it should be closed

by pressing “CTL+C” (with mouse focused on the terminal). If you failed to do that, the program uploading will fail as the port is being busy.

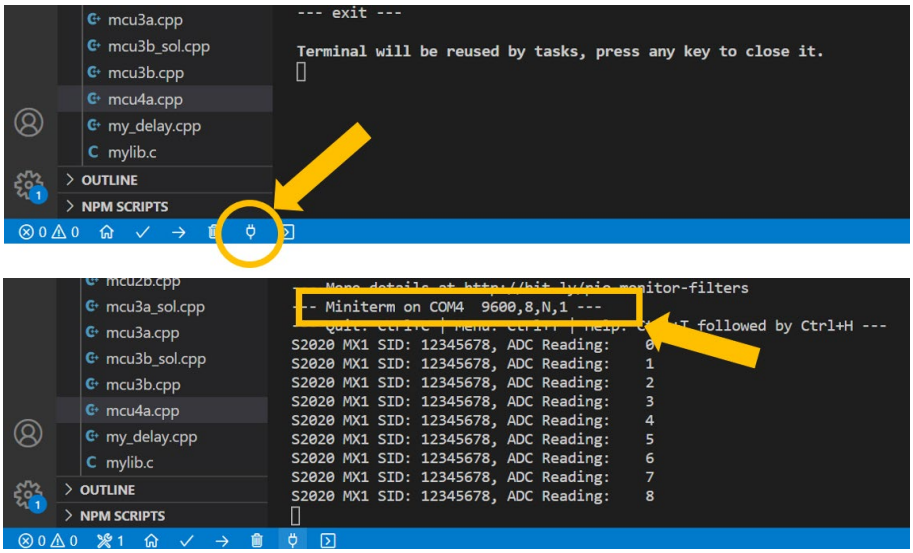

Figure A.1. Using Serial Monitor in VSCode

Appendix B – Basics of UART Operations

Microcontrollers must often exchange data with other microcontrollers or peripheral devices. Data may be exchanged by using *parallel* or *serial* techniques. With parallel techniques, an entire byte of data is sent simultaneously from the transmitting device to the receiver device. While this is efficient from a time point of view, it requires eight separate lines for the data transfer together with address lines (complicate!). In serial transmission, a byte of data is sent through a single bit at a time. Once eight bits have been received at the receiver, the data byte is reconstructed. While this is inefficient from a time point of view, it only requires a line (or two) to transmit the data.

Atmega328P microcontrollers are equipped with 3 different serial communication subsystems – a USART, a Serial peripheral interface (SPI), and a Two-wire serial interface (TWI). In serial communication, the transmitting and receiving device must be synchronised to one another and use a common data rate and protocol. In asynchronous serial communication, as in the USART, a start and stop bits are used to notify the receiver to reset the sampling timer. In synchronous communication, such as SPI or TWI, a common clock between the two devices “syncs” between the transmitter and receiver. Data transmission rates are typically specified as a *Baud* or *bits per second rate*. For example, 9600 Baud indicates the data is being transferred at 9600 bits per second. The USART and SPI are a full-duplex system meaning they have two separate hardware for the transmission and reception. TWI, however, is a half-duplex system sharing a single data line, making the connection much simpler but with the extra cost of programming.

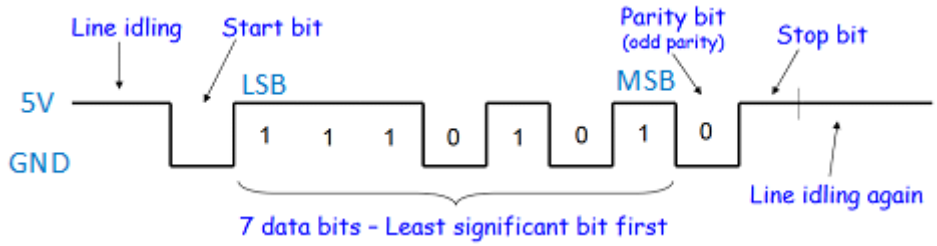

Figure B.1 An example of a UART data packet (Data = 0x57, 1-stop bit and no parity)

Figure B.1 illustrates an example of UART output.

- Each frame starts with a starting bit of '0' signalling the receiver to reset its sampling timer.
- The data is shifted out the MSB (the most significant bit) first. This example shows 7-bit data of 0x101 0111, or 0x57, an ASCII code 'W'.
- A parity bit for an error checking. If the odd-parity is enabled, the total number of 1-bits should be odd, including the parity bit.
- Each frame finishes with 1 or 2 stop bits.
- For a long distance communication, the signal level is converted to ±12V inverted signal in the RS232 system or 5V differential signal in the RS422/485 system.
- Sometimes handshaking signals (called flow control) are used to avoid data loss.

Appendix C – UART Registers

The following registers should be configured and used (see more in datasheet),

- USART Data Register (UDR0) – Read and write a byte (two separate buffers)
- USART Baud Rate Register (UBRR0H/L) – Select the bit rate (Baud rate).
- USART Control and Status Register 0 A/B/C (UCSR0A/B/C) – Configure and Status

Name: UDR0

Bit	7	6	5	4	3	2	1	0
	TXB / RXB[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Name: UCSR0A

Bit	7	6	5	4	3	2	1	0
	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

UART Receive Complete.

UART (TX) Data Register Empty.

UART Double the speed.

Name: UCSR0B

Bit	7	6	5	4	3	2	1	0
	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

UART RX Complete Interrupt Enable.

UART TX Data Register Empty Interrupt Enable.

Enable RX/TX

Name: UCSR0C

Bit	7	6	5	4	3	2	1	0
	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 / UDORD0	UCSZ00 / UCPHA0	UCPOL0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

UART Stop Bit Select. (‘0’: 1-bit)

UART Char Size. (‘11’: 8-bit)