HLAB07
Atmega328 UART Serial Communication – UART (Universal Asynchronous Receiver and Transceiver)

This lab will introduce the Atmega328 Universal Synchronous and Asynchronous Receiver and Transmitter (USART), or more commonly known as UART, which is a highly flexible communication method in microprocessor systems. A UART is typically used to send debugging messages to a host computer, a much powerful debugging tool than blinking LEDs.

1. Aims

- To be able to change the Atmega328 clock source to an external crystal oscillator.
- To understand the UART transmit and receive using a polling method.
- To implement a UART program which can receive a line of characters from the terminal using a polling method.

2. Changing the Atmega328 Clock Source

The Atmega328P processor is using an internal oscillator in default, which is 8 MHz with a clock division by 8, generating 1 MHz clock signal (see more details in the datasheet, Section 31 and Table 31-7 shown in Appendix A). Instead, we will use the external 20 MHz crystal oscillator provided in the Atmega328 kit which is much faster and reliable. To change the clock source, we need to change the so called ***fuse bytes*** which can be accessed through the USB programmer.

- Follow the instructions in Appendix A to change the fuse setting.
- Modify the LED blink code (in HLAB06) so that the LED blinks at 1 Hz properly.

3. Setting up UART Connection

In this lab, we will use the basic capability of the UART system. Figure 1 shows the related UART pins in Port D. Once the UART transmitter and receiver are enabled, the pins PD0 and PD1 act as RX and TX outputs. The LED connected in PB1 will be used for the debugging purpose.

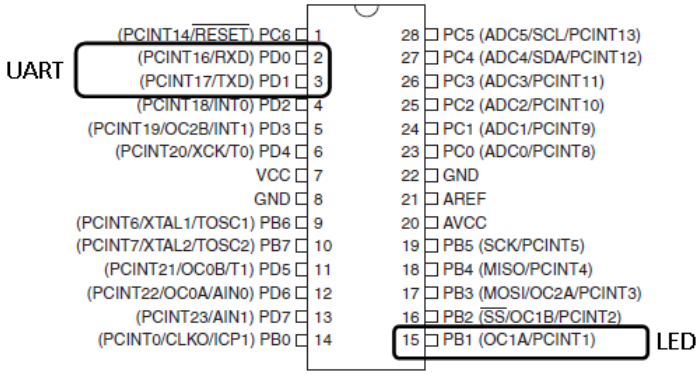

Figure 1 Atmega328 UART TX/RX pins.

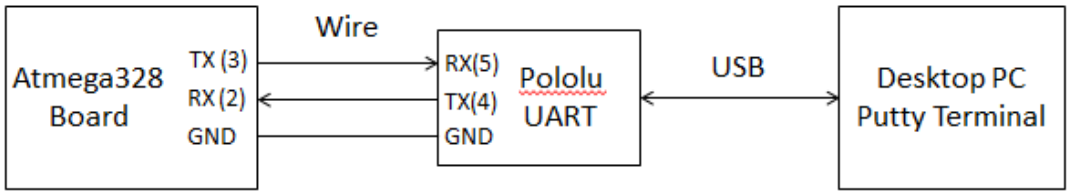

Figure 2 Crossover connections between two UARTs.

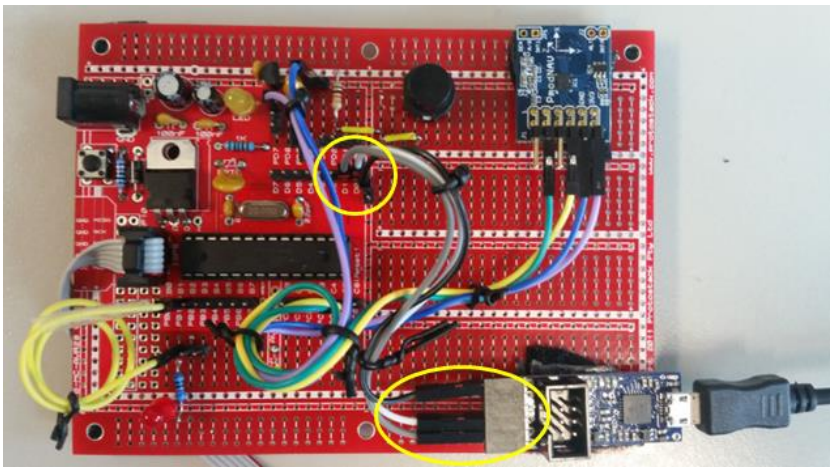

Figure 3 An example of the UART connection (in yellow circles) to the Pololu UART.

Figures 2 and 3 illustrate the serial connection between the Atmega328 board and the desktop PC using a Pololu Serial-to-USB adaptor. Note that the TX pin in the Atemga328 is connected to the RX pin in the USB adapter and the RX pin to the TX pin in a crossover way.

- Connect the Atmega328 board to the desktop PC using the Pololu USB-to-Serial adaptor as shown in Figures 2 and 3. You can use Velcro pads to attach the USB adapter on the protoboard.
- Open the Putty terminal in Windows, which will open a window similar to Figure 4. Select the "*Serial*" category for further setup.

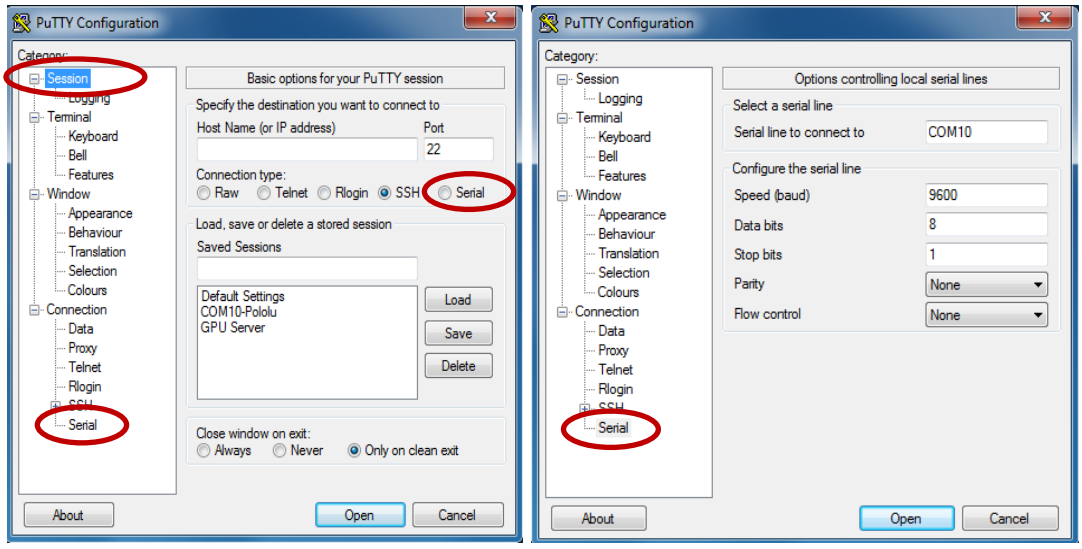

Figure 4 Putty Session window (left) and Serial setting page (right).

- Type in the Pololu serial port number ("pavr2cmd --ttl-port" in a Windows command prompt) as well as the 9600-baud rate, 8-data bits, 1-stop bit, no parity and no flow control. For future usage, you might *save* the session which can be reloaded in next time.
- Now the Putty terminal is ready to receive or transmit data. Note that it is an ASCII terminal, and thus it can only displays the ASCII-coded bytes properly. If you wish to check the proper installation of the Putty terminal and the Pololu serial adapter, you can try the loopback test as shown in Appendix D.

4. UART Programming – Polling

Now we can program the UART to send and receive a byte data. If you are not familiar with the UART operations, please read Appendix-A first. Three flow-charts of the UART operations are summarised in Figure 5 below. The first step is to initialise the UART with Baud rate (or bits per second), data size, stop bits and parity, followed by enabling the chip. To send a byte data, we simply write a byte to UDR (UART Data Register) whenever the data register is empty. In a polling method, we continuously read the status register to see whether or not the UDRE (UDR Empty)-bit is set. Receiving a byte is similar to the transmitting. We continuously read the status register to check whether or not the RXC (Receiver Complete)-bit is set. If yes, then we can read a byte from the UDR register. The following registers are related to UART operations (see more in Appendix).

- UDR0 (USART0 Data Register) – Read and write a byte (two separate buffers)
- UBRR0H/L (USART0 Baud Rate Register) – Select the Baud rate.
- UCSR0A/B/C (USART0 Control and Status Registers) – Configure and check the status

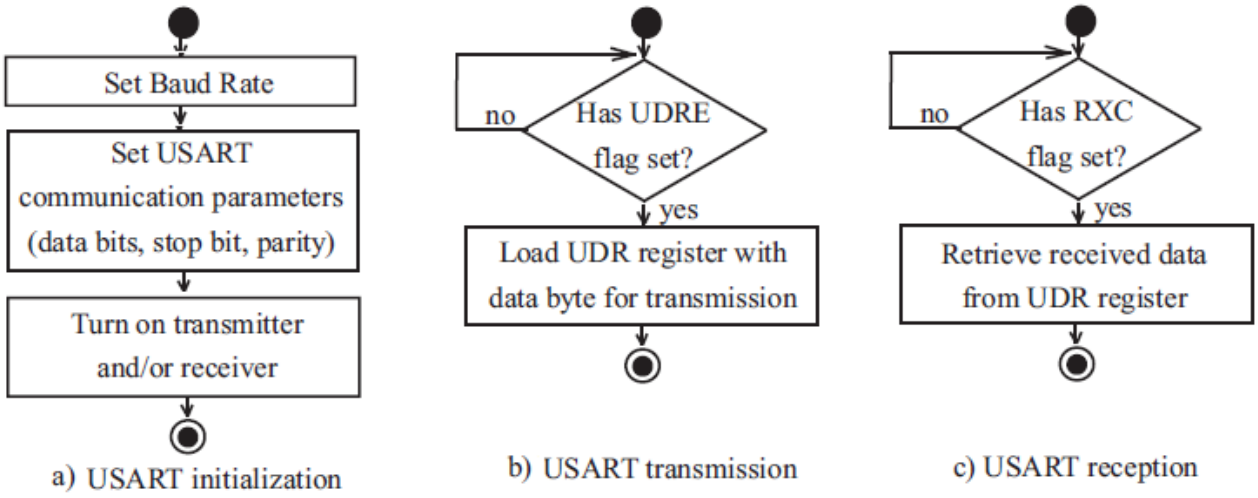

Figure 5 Flowcharts for UART operations – initialization (a), polling-based transmit (b) and receive (c)

We will use the polling-based transmit and receive in this lab.

- Open Atmel Studio and create a new project. Setup the device type and ISP as in HLAB6. Download 4 tutorial codes in Wattle
  - *2_uart_tx.c*
  - *3_uart_tx_str.c*
  - *4_uart_rx.c*
  - *5_uart_rx_str.c* (In-complete code)
- You can add each file to the project (in this case you need to remove the *main.c* from the project) or simply copy and paste the code to the *main.c* file.
- The codes are without comments. Add proper comments so that other students or tutors can understand the operation. The register definitions are summarised in Appendix C for your convenience or refer the datasheet.
- Build and run each executable on the Atmega328 board. Open the Putty terminal in the Windows to display or send the data to the board.
- **Complete the "5_uart_rx_str.c" program** which can receive and display a line of ASCII data the terminal. When an [enter] key (or a carriage return '\r') is typed at the end of the line in the Putty terminal, it will indicate the end of the line. The Atmega board should transmit back the whole line data to the terminal. **Demonstrate the result to the tutors to get marks.**

- Congratulation! You have a serial connection which can be used for debugging together with the blinking LED. These two tools are the most common debugging tools in the embedded system development.

Appendix A – How to change the Atemga328 clock source

The Atmega328P controller is using an internal RC (resistor-capacitor) oscillator which is 8 MHz in a default setup, and the clock is further divided by 8, generating 1 MHz clock signal (see more details in the datasheet and Section 31 and Table 31-7 shown below. Please note that '0" means selected/programmed which is opposite to the conventional use). The Atmega328P protoboard kit provides a 20 MHz crystal oscillator which is faster and reliable than the internal one. To change the clock source, we need to change the ***fuse byte*** which can be accessed through the USB ISP programmer. The default value of the lower fuse register is ***0x62*** as in Table 31-7. We will change this value to ***0xFF*** (an external crystal oscillator + no clock division + slowest reset start-up time).

- Connect the Pololu USB programmer to the Atmega board and the PC. Power the board.
- Open Atmel Studio, and open the previous project used in HLAB06 (so we don't need to setup the device type and ISP again).
- Select *Tools* and select *Device Programming* which opens a page like Figure A1.
- Make sure the *Fuse Resistor Values* are read as shown (default values). Make sure you have soldered the 20MHz oscillator before proceed. Otherwise, the chip cannot power up waiting for an external clock signal.
- Now, type the new 0xFF value to the "LOW" fuse register. Click the *Program* button. It will generate a warning window as in Figure A2, and press the *Continue* button. It will display an OK message in the text terminal.
- Done! The Atmega328 chip now uses the 20 MHz clock. To confirm this, download again the blinking LED code to the board by pressing F5. You will notice that the LED is blinking much faster than 1 Hz (20 times faster!). This is due to that the delay function we used still thinks the clock is 1 MHz. To fix this, we need to let the compiler know by adding a line
  ```
  #define F_CPU 20000000
  ```
- Re-download the code (press F5) and confirm that the LED blinks at 1 Hz properly.

Table 31-7. Fuse Low Byte

Low Fuse Byte	Bit No.	Description	Default Value
CKDIV8(4)	7	Divide clock by 8	0 (programmed)
CKOUT(3)	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)(1)
SUT0	4	Select start-up time	0 (programmed)(1)
CKSEL3	3	Select Clock source	0 (programmed)(2)
CKSEL2	2	Select Clock source	0 (programmed)(2)
CKSEL1	1	Select Clock source	1 (unprogrammed)(2)
CKSEL0	0	Select Clock source	0 (programmed)(2)

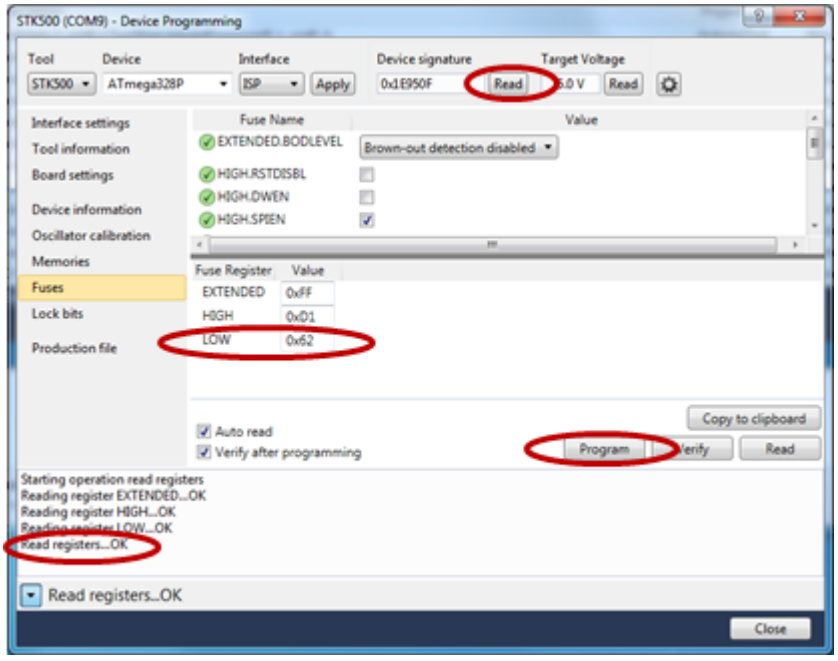

Figure A1 Device programming page

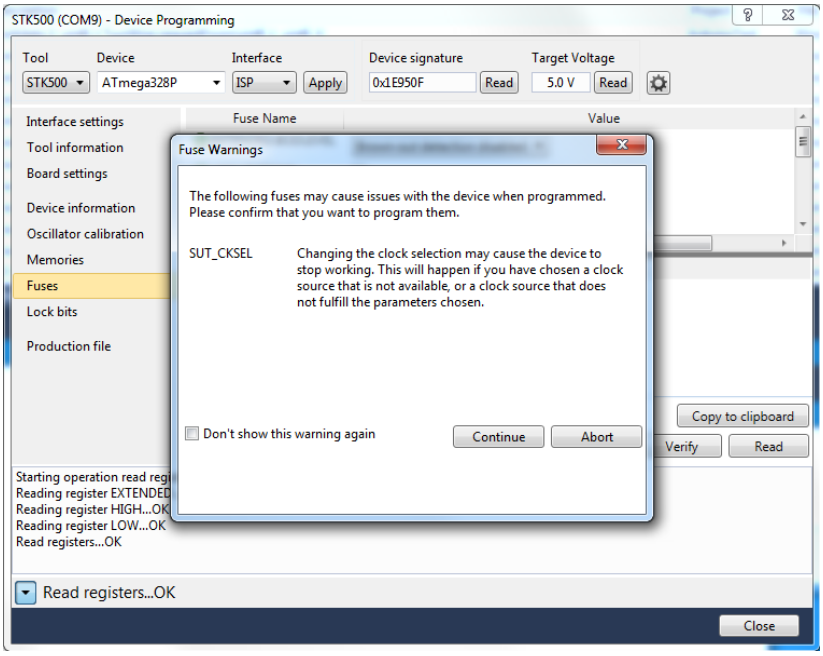

Figure A2 Changing the LOW-byte in device programming

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.

Atmega328 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.

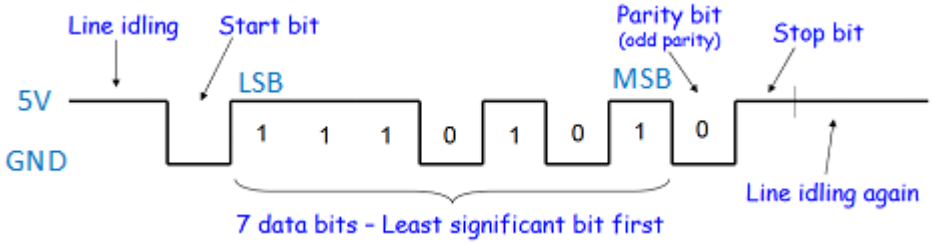

Figure B.1 An example of a UART data packet (Data = 0x57, 1-stop bit and no parity)

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)

Appendix D – Putty Loopback Test

You can test the Pololu UART and Putty terminal using a loopback test, by connecting the Pololu Tx and Rx pins as shown in below. Any char typed in the Putty terminal will be sent to the receiver and will be displayed in the terminal.

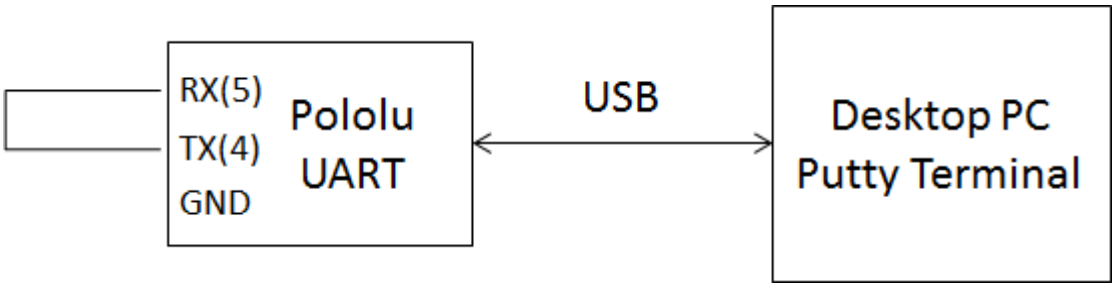

Figure D.1 A crossover connection of the TX and RX pins in Pololu UART.