CSSE2010

CSSE2010/CSSE7201
Learning Lab 16

Serial Communications
ADC

School of Information Technology and Electrical Engineering
The University of Queensland

Today

- AVR Serial Communication
  - in C with interrupts
- Analog to Digital Conversion (ADC)
  - Using the Joystick

- Outline
  - Introduction – some reminder of lecture content
  - Datasheet searching
  - Code modification

Recall from Lecture 18:

AVR USART

- USART = Universal Synchronous and Asynchronous Receiver/Transmitter
- Key points for us
  - **Serial** communication (we communicate *frames* 1 bit at a time)
  - **Asynchronous**
    - Receiver and transmitter can have different clock rates
    - Clock not transmitted with data
- Often see term **UART** used
- Datasheet pages 175 to 201

Recall from Lecture 18:

AVR USART & Baud Rate

USART = Universal Synchronous and Asynchronous Receiver/Transmitter

- Key points for us:
  - **Serial** communication (we communicate 1 bit at a time)
  - **Asynchronous -** Clock not transmitted with data
- AVR supports frames of 5 to 9 bits, one start bit (0), one or two stop bits (1), optional parity bit

Baud rate = symbols per second

- Not the same as bit rate
  - e.g. 10 symbols per 8 bits (1 start bit + 8 data bits + 1stop bit)
  - 9600 baud = 7680 bits per second = 960 bytes per second

AVR Baud Rate

- Baud rate register is set based on
  - Device clock speed (8MHz for us)
  - Number of clock cycles between samples – 16 samples per symbol
- From datasheet:

$$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$$

- n = serial port number (0 or 1)
- UBRRn is the 12-bit value in two I/O registers
  - UBRR0 = UBRR0H (4-bits) and UBRR0L (8-bits)
  - UBRR1 = UBRR1H (4-bits) and UBRR1L (8-bits)

Exercise – Baud Rate

- What value needs to be placed in Baud Rate Register UBRR0 (registers UBRR0H and UBRR0L) so that serial port 0 operates at 19200 baud (on an 8MHz device, asynchronous normal mode)?

- All values you work out will be needed in later code task

Recall from Lecture 18:

Relevant Registers

Control and Status Registers

- See UCSRnA (page 193 of datasheet)
  and UCSRnB (page 194)
  and UCSRnC (page 195)

USART Data Register

- UDR0 (for serial port 0)
  - Actually two registers - one for reading, one for writing
  - Data written to this register is transmitted
  - Data arriving over the serial port can be read from this register

Exercise – Serial Control

- What value (binary) needs to be placed in UCSR0C so that serial port 0 operates asynchronously with 8-bit data, one stop bit, no parity?

Recall from Lecture 18:

Serial Interrupts

- 3 interrupt sources associated with each AVR serial port
- Receiving data:
  - **Receive complete**
    - Frame (usually character) received
- Transmitting data:
  - **Data Register empty**
    - Ready to accept new data for transmission
  - **Transmit complete**
    - Frame sent and no data waiting to be sent

Exercise - UCSR0B Register

- What value necessary to
  - enable transmission and reception
  - enable interrupt when data arrives

AVR Interrupts in C

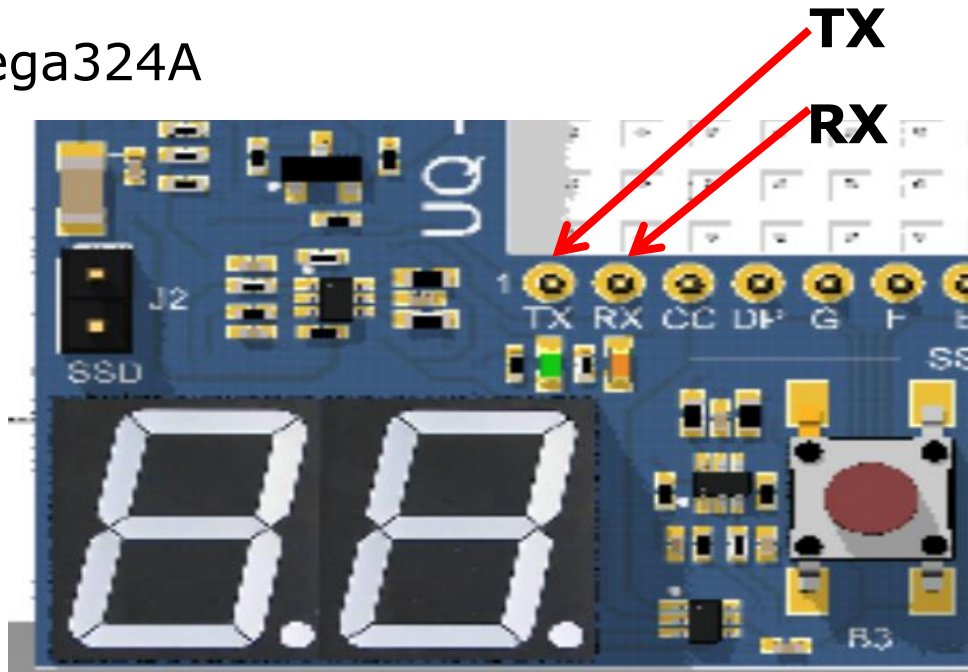
- Reminder - we just declare functions using a special macro and interrupt vector name
- Example - for timer/counter 1 output compare match A:

```
ISR(TIMER1_COMPA_vect) {
    ...
}
```

- Where do we find out the vector names to use?
  - See datasheet (add **_vect** to source name) OR
  - See AVR C library documentation (via Blackboard or Atmel Studio)
    - avr/interrupt.h documentation
- What name do we use for ATmega324A serial port 0 receive complete interrupt?

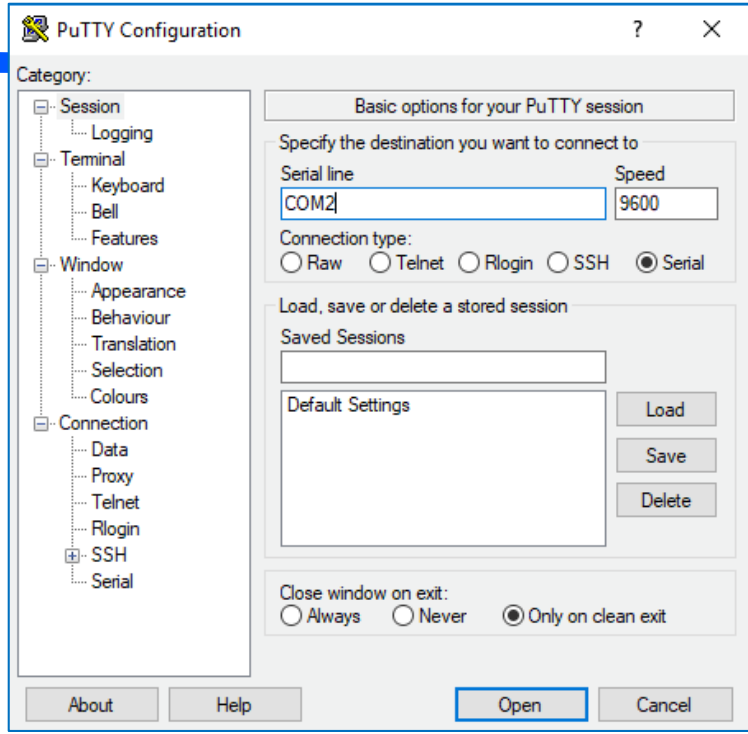
Serial Communication on the IOBoard

- TX = PC transmits
  - output from IO board
  - connect to input on ATmega324A board (AVR RX pin)
- RX = PC receives
  - input to IO board
  - connect to output on ATmega324A board (AVR TX pin)
- LEDs show communication



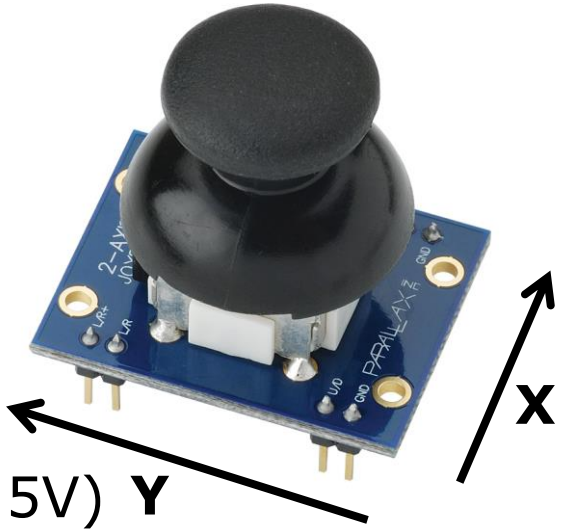
Serial communications on the PC

PuTTY



- We're using a USB-serial device
  - If not recognised, may have to install FTDI driver (see Blackboard - Software)
- On PC we use a terminal program
  - Putty – choose "Serial" communication
- Need to tell Putty:
  - COM port (which serial port you're communicating over)
    - May need to use device manager to determine this – it will change if you use a different USB port
  - Baud rate
  - Start/stop/parity bits etc – defaults are usually OK

Joystick



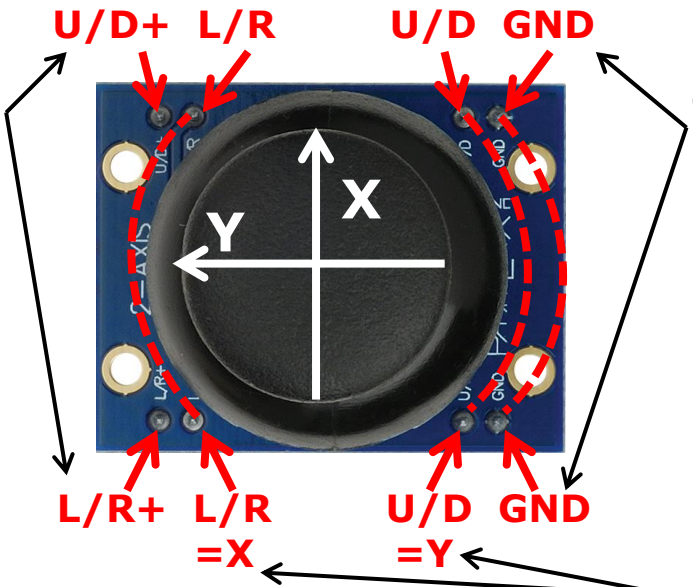
- Designed to be placed in breadboard
- Connections:
  - **L/R+** and **U/D+** (connect to VCC = 5V)
  - 2 x **GND** (connect one to GND)
  - 2 x **L/R** = X – analog voltage indicating X position
    - 0V = left hand side, 5V = right hand side
  - 2 x **U/D** = Y – analog voltage indicating Y position
    - 0V = bottom, 5V = top

Joystick (cont)

Try not to lose the black foam protecting the pins

- Place this way in breadboard
  - at right hand end – allow enough space for MCU board and programmer
- Note that X is to the top of the breadboard as you would normally look at it

Need to connect both of these to VCC

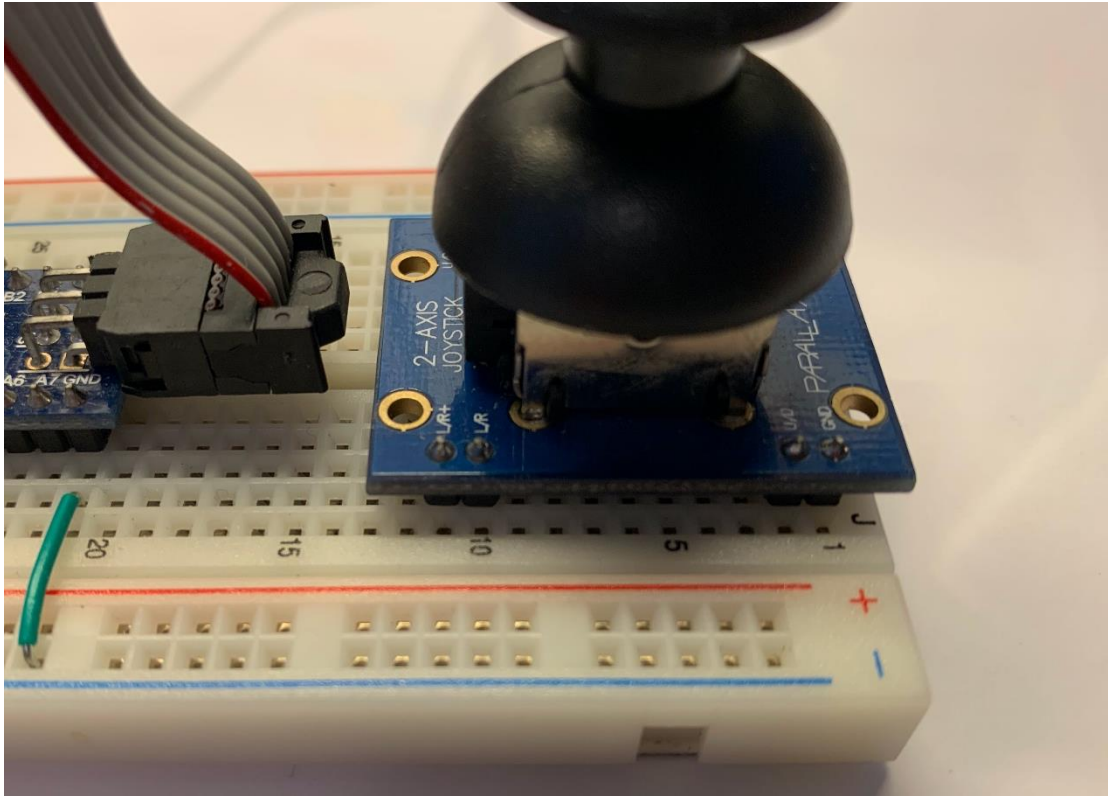


Connect one of these to GND

(Other pairs of pins are connected – only need to connect to one of each pair.)

Connect these to analog inputs on MCU

Joystick on Breadboard



Recall from lecture 18

ATmega324A ADC

- Port A pins can be used as analog inputs
- One input can be converted at a time (using an analog multiplexer)
- Can choose Vref - we'll pick Vcc = 5V
- ADC resolution = 10 bits (0 to 1023)
- Conversion takes time
  - Up to 3200 clock cycles (0.4ms at 8MHz)
  - Can trade-off accuracy for speed
  - Must poll completion bit or set up interrupt to know when finished

Exercise

- What value needs to go in ADMUX to use AVCC as the reference voltage, right adjust the result and select input ADC1? (Consult datasheet)

Tasks – AVR projects

- Complete lab16-1.c
  - Program to receive characters over serial port, convert to upper case if necessary, send them back
  - Test on board, connect
    - RX0 on AVR to TX on IOboard, TX0 on AVR to RX on IOBoard
- Complete lab16-2.c
  - Program to receive characters and convert digits to word equivalents (1->one etc).
    - Because we want to send multiple characters at once we need to buffer them – we use a circular buffer
  - Program is quite long though required changes are short – make sure you understand the program – ask if not sure. (Note baud rate is different.)
- Joystick ADC exercise – lab16-3.c
  - Create a project using multiple files (drag files into Solution Explorer)
    - lab16-3.c, serialio.c, serialio.h
  - Serial IO is connected to C standard input/output – so can use printf() etc.
  - Complete the program (configuring ADC)