Robotic Arm

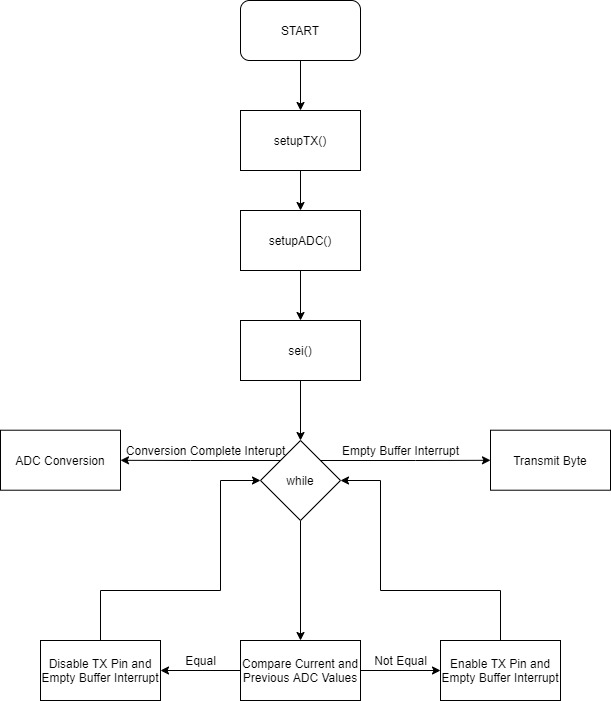
Ali Aqdas | Ali Hamza Malik

# **Abstract**

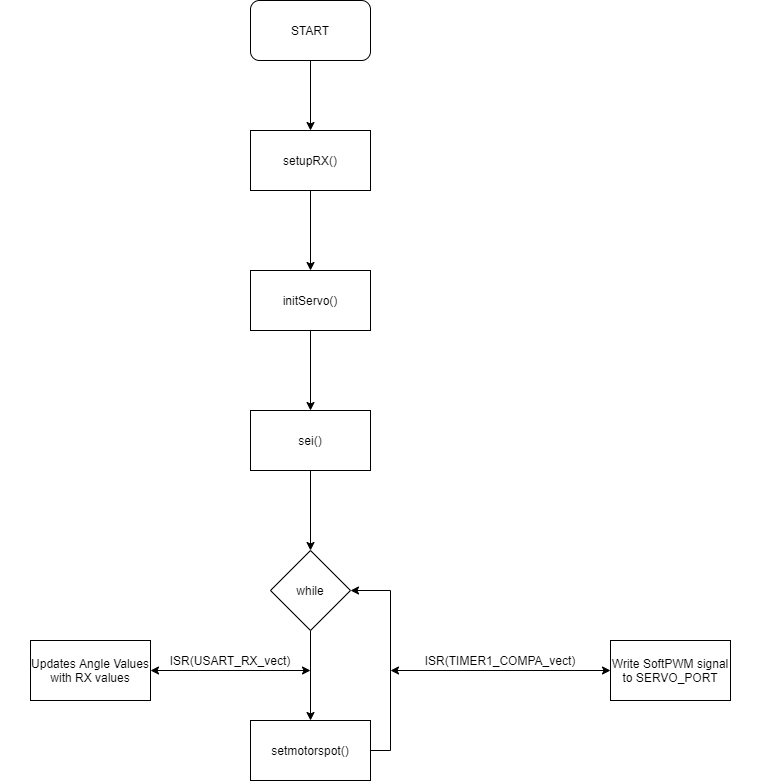
Our aim is to develop a Robotic Arm which controls servo angle values using an ATmega328p Microcontroller. We have developed a robotic arm based on two ATmega328p microcontrollers, one for controlling the Motors and the other for controllers. Eight Servo Motors are controlled by potentiometer-based controllers. The functioning number of motors can be varied from one to eight which enables the Microcontroller to be wired up to a Robotic Arm with DOF (Degrees of freedom) ranging from 1 to 8. The Motors can be controlled with a Controller based on Bluetooth, Wi-Fi, Radio Receiver, USB through RX pin in Serial Format.

# **Block Diagram**

## Transmitter



## Receiver



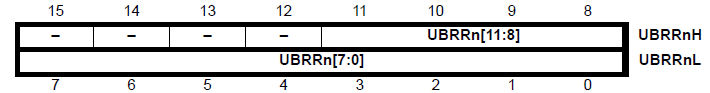
# **Code Description**

## Setup Analog Digital Converter (ADC)

We configure ADC in **setupADC** function using **ADCMUX** which lets us select Analog Input Pin and Standard Voltage Source for Comparison. **ADCSCRx** are control and status registers that let us enable and start conversion of analog signal. They also allow us to setup interrupt once the conversion is completed and adjust the 10-bit result. We have setup the interrupt and left adjusted the result. **DIDR0** allows us to disable digital input on analog pins. We have used all the analog pins available and thus disabled digital input on all pins.

## Setup Transmitter and Receiver

The USART transmitter is configured in **setupTX and setupRX** function. The baud rate is provided in the UBRR(Baud Rate Register). The 16-bit value provided to the register is given by the following formula



**UCSRnx** are Control and Status Registers that let us configure and view the status of USART Transmitter and Receiver. We have configured 8-bit Asynchronous USART with one stop bit and even parity. We have also enabled interrupt in case of empty buffer for Transmitter and interrupt on the completion of receive on the Receiver.

## Data Transmission

We have applied a check to determine if the current ADC data matches the previous ADC. If they are equal the transmit pin is disabled to save power. If they aren’t equal the transmit pin and empty buffer interrupt is enabled. Thus, entire array of ADC values is transmitted.

## Analog Digital Conversion

Analog Digital Conversion is done pin by pin. The interrupt is setup to occur whenever a conversion is complete. Each time interrupt is called the analog pin to be converted changes and resets after the number the of Potentiometers attached. All these digital values are left adjusted and stored in an array so we can easily transmit a byte without any significant data loss.

## Setup 16-Bit Timer and Soft PWM with ISR

The soft PWM signals are generated on **SERVO\_PORT** using the 16-bit timer and interrupts. The timer is setup to run in **normal mode** so that the output pins of the 16-bit timer **OC1A/OC1B** which are disconnected, which means the pins can be used as normal I/O pins. The Timer is now started with no pre-scaling which allows us to reduced the noise in the Soft PWM signal due to the Toggling or Switching of I/O pins. Now, the **SERVO\_PORT** pins are driven high or low for the ON time stored in an array called **servoTicks** which store the compare time which are then written to the **OCR1A** Compare Register which determines when the signal is to be turned off. When this value on the **OCR1A** Compare Register is reached, the TIMER1\_COMPA\_vect Interrupt is trigged which determines which pin to Toggle and Toggles it OFF at the specific value for **OCR1A** and the sets the **OCR1A** Compare Register to new value with certain space from the current value of **OCR1A.**The servo counter variable is then incremented so that the interrupt runs for the next servo pin at the specific time.

As the Timer is running without a prescale, the Tick values for **OCR1A** Compare Register are translated as:

So, we have about increments which translate into . This gives us a lot of resolution to control the PWM signal. For our purposes, the angle values which are 8-bit values from 0 to 255 are mapped into a range of 16000 to 32000. This is accomplished using the map() function.

## Mapping Values

The function map() converts values form one rage to another. It is a Linear mapping which translates the input value from an expected domain to the required range This is done mathematically by writing the OUTPUT value in the required range by using point-slope form of linear equation of line:

## Writing Values to OCR1A

The map function is used to rescale input values from a certain range which is 0 to 255 for an 8-bit value to the range of ticks for OCR1A timer which is 16000 to 32000. The input value is then changed for required number of servo in the **servoTime** array which is then read into **OCR1A** in the **ISR Interrupt.**

List of Functions is provided in the appendix under **Functions**. The details of the registers used in the code is available in ATmega328p datasheet available online through the [link](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf).

# Future Prospects

The Robotic Arm has ability to receive control commands through Serial Communication, thus we can further extend the design for the arm to be controlled by Smartphone through Bluetooth, over Internet through Wi-Fi and simply through USB connected to a computer in the future.

High Level Movements can also be expected in the future through the use of inverse kinematics and AI Assisted Controls.

# Conclusion

We have successfully designed a robotic arm based on the ATmega328p Microcontrollers with serial communication for control commands. The control is currently provided with eight potentiometers, thus a total of eight of motors, which provide a wide range of movements.

# Appendix

## Functions

### unsigned int map(int x, float xmin,float xmax,float ymin,float ymax);

Maps x from input range xmin to xmax to output range ymin to ymax.

### void setupRX();

Setup the USART RX Registers to Receive values over the RX pin.

### void setupTX();

Setup the USART TX Registers to Transmit values over the TX pin.

### void initServo();

Initializes Timer in Normal Mode without Prescale and Enables SERVO\_PORT as OUTPUT by setting SERVO\_DDR to 0XFF.

### void writeangle(uint8\_t servo, uint8\_t angle );

Writes angle value in Range 0 to 180 to Servo number= servo.

### void writeus(uint8\_t servo, uint8\_t time );

Writes time value in Range 1000us to 2000us to Servo number= servo.

### void writepotval(uint8\_t servo, uint8\_t val );

Writes val value in Range 0 to 255 to Servo number= servo.

### void setmotors();

Writes angle values stored in array motoranglesvalues in range of 0 to 180 to Servo motors

### void setmotorspot();

Writes 8-bit values for angles stored in array motoranglesvalues in range of 0 to 255 to Servo motors

### void selectADC(uint8\_t ADC\_pt);

Modifying first four bits of ADMUX to select ADC Port.

### void setupADC();

Setup ADC.

### void startADC();

Starts ADC Conversation.

### void checkeq(void);

Checks if array is same at the beginning of array transmit.