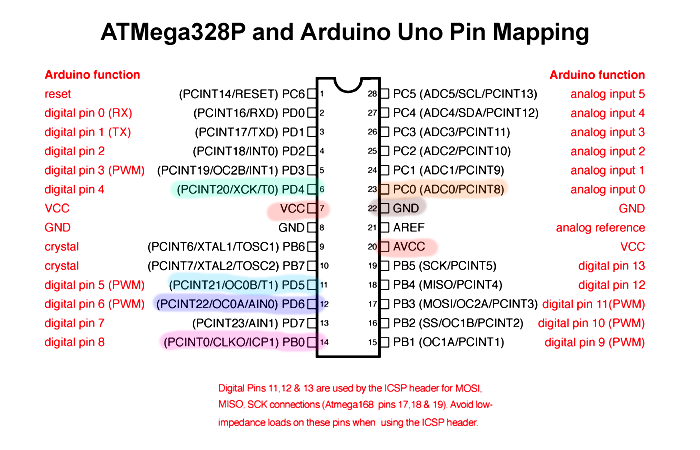
Write, implement, and demonstrate using Atmel Studio 7 a C code for the AVR  
ATMEGA328p microcontroller that performs the following functions:  
You’ll use the ADC, and PWM/CCP Module of the ATmega328/p to set and determine  
the speed of the DC Motor.

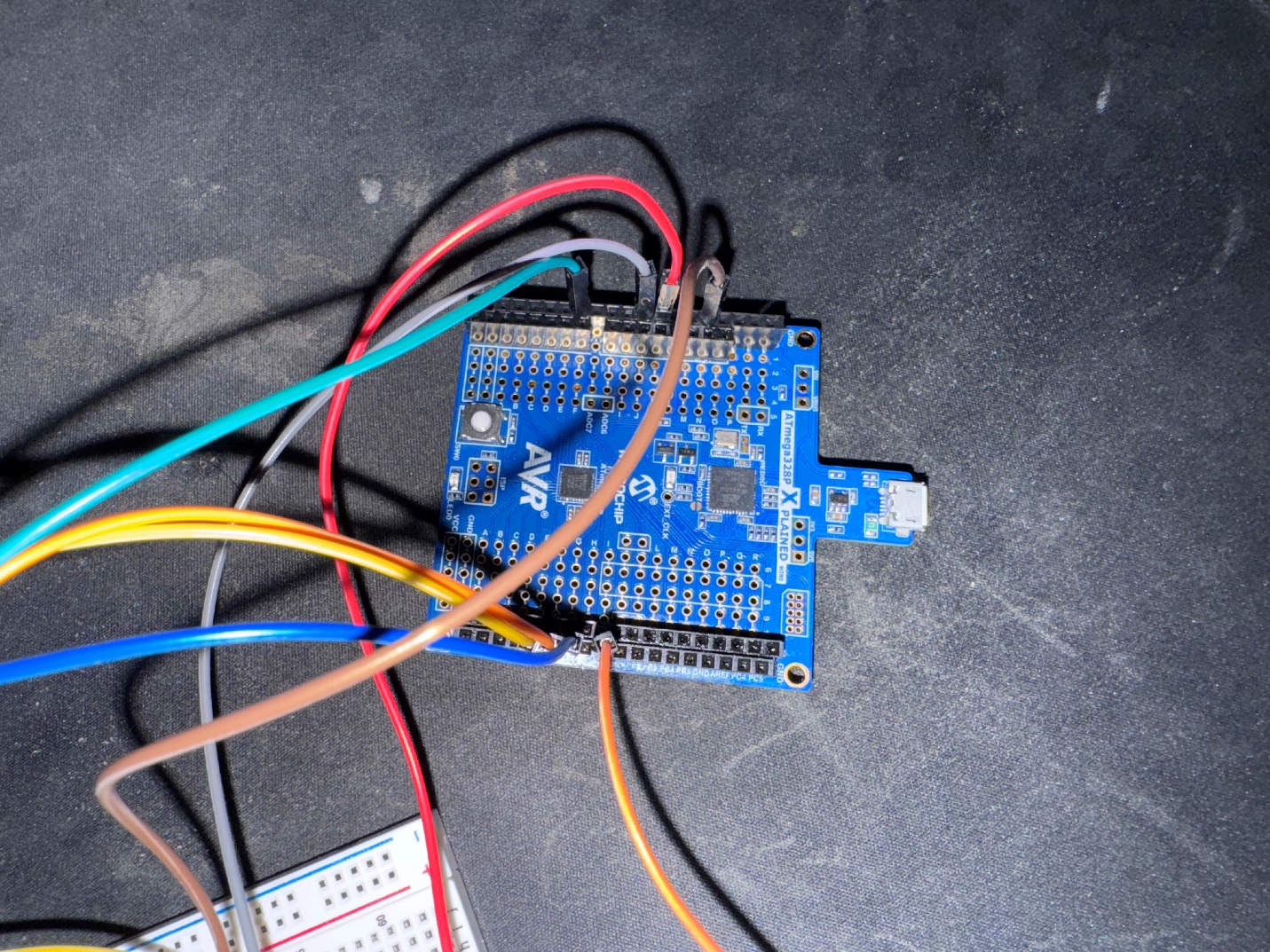
1. Using the Potentiometer connected to ADC0, translate the ADC value (0~1023) to  
PWM value/speed of the motor (0~255 if using Timer0/2). Verify the operation.

2. Using the CCP capture pin of PWM1, in mode 1x determine the speed of the DC  
Motor for a set ADC Pot value/position.

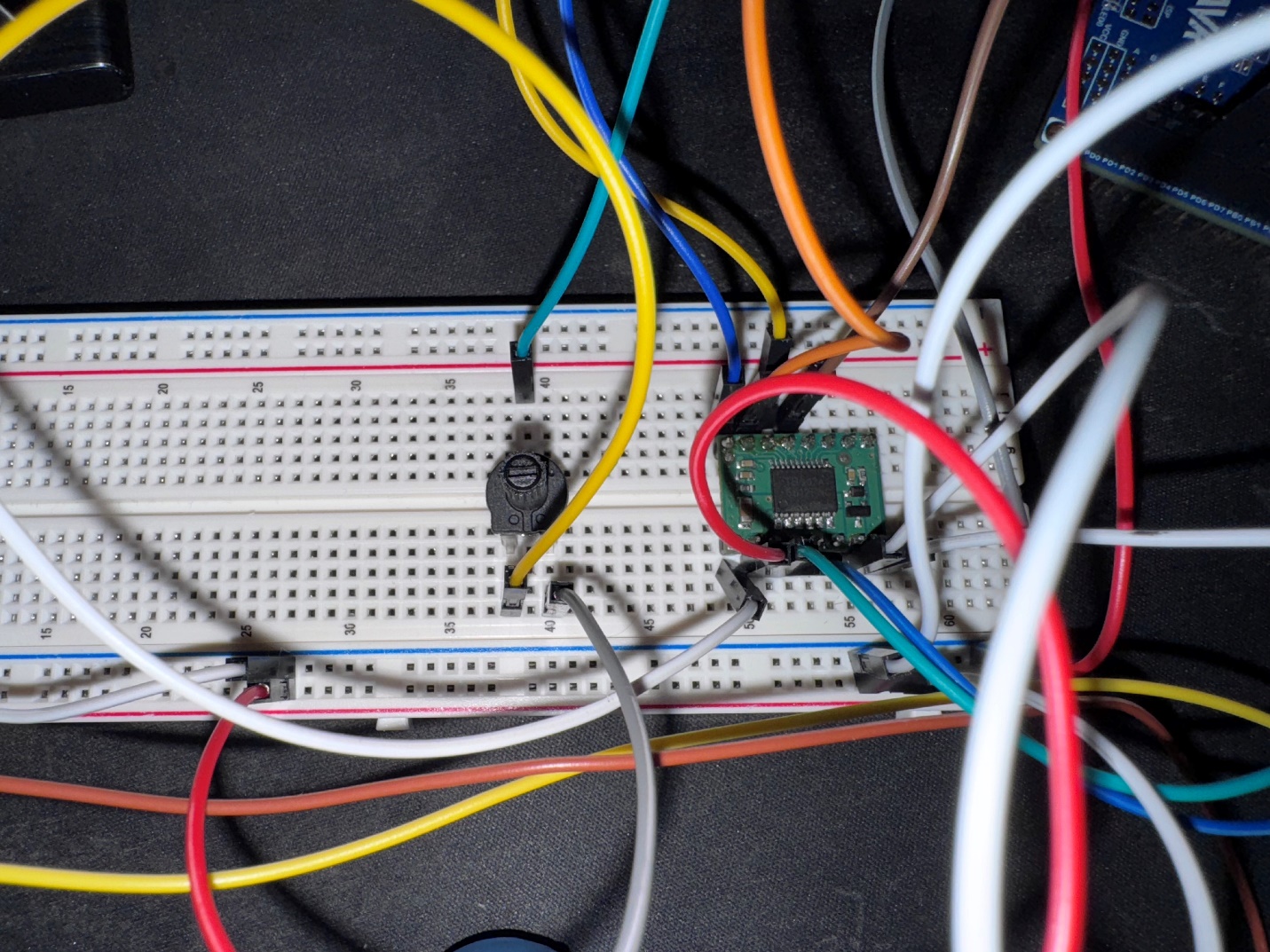
3. Develop a UART GUI interface to override the ADC speed control, the user input  
will control the speed of the motor. Plot the set speed and current speed using a  
UART GUI tool.

**Components Used/Connected**

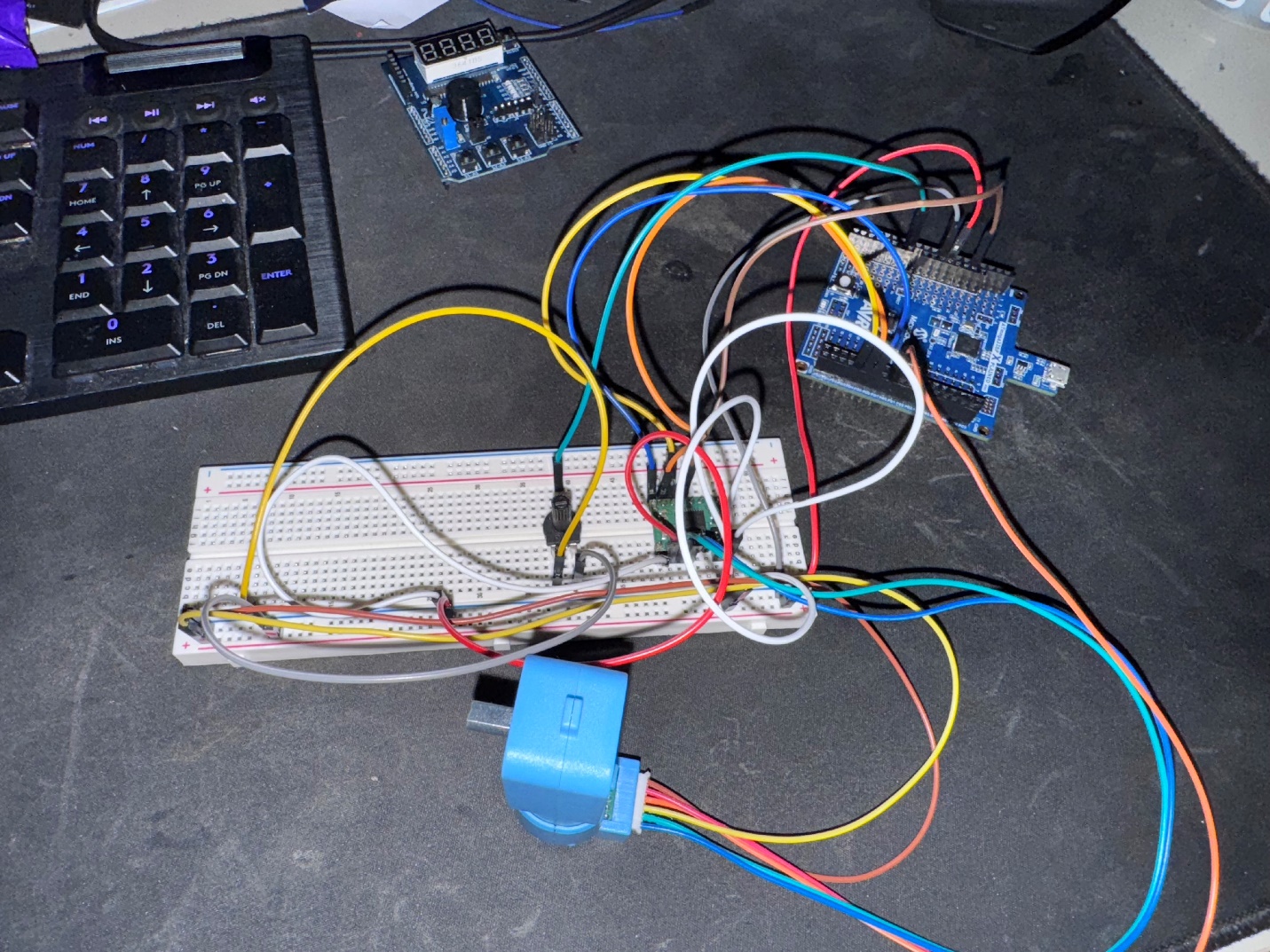




Atmega328p wiring



Breadboard layout.



Whole setup.

A computer screen shot of a circuit board

AI-generated content may be incorrect.

Schematic showing how everything was set up.

**AVR C Code**

/\*

\* DA5.c

\*

\* Created: 4/28/2025 10:44:50 PM

\* Author : enriq

\*/

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdlib.h>

#include <stdio.h>

// Motor driver pins

#define IN1\_PIN PD4

#define IN2\_PIN PD5

#define PWM\_PIN PD6

// Variables

volatile uint16\_t icr\_last = 0;

volatile uint16\_t period\_counts = 0;

volatile uint8\_t new\_period = 0;

uint8\_t set\_speed = 0; // 0–255, PWM

uint8\_t measured\_speed = 0; // 0–255, from measurement

// UART parsing

char rx\_buf[4];

uint8\_t rx\_pos = 0;

uint8\_t override\_flag = 0;

uint8\_t override\_speed = 0;

// UART at 9600 baud

static int uart\_putchar(char c, FILE \*stream) {

while (!(UCSR0A & (1<<UDRE0)));

UDR0 = (uint8\_t)c;

return 0;

}

static FILE uart\_str = FDEV\_SETUP\_STREAM(uart\_putchar, NULL, \_FDEV\_SETUP\_WRITE);

void uart\_init(void) {

UBRR0 = F\_CPU/16/9600 - 1;

UCSR0B = (1<<TXEN0)|(1<<RXEN0);

UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);

stdout = &uart\_str;

}

// ADC

void adc\_init(void) {

ADMUX = (1<<REFS0);

ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);

}

uint16\_t adc\_read(void) {

ADCSRA |= (1<<ADSC);

while (ADCSRA & (1<<ADSC));

return ADC;

}

// PWM (PD6)

void pwm0\_init(void) {

DDRD |= (1<<PWM\_PIN);

TCCR0A = (1<<COM0A1)|(1<<WGM01)|(1<<WGM00);

TCCR0B = (1<<CS01);

}

// ICP1 (PB0)

ISR(TIMER1\_CAPT\_vect) {

uint16\_t ic = ICR1;

period\_counts = ic - icr\_last;

icr\_last = ic;

new\_period = 1;

}

void icap\_init(void) {

DDRB &= ~(1<<PB0);

TCCR1B = (1<<ICNC1)|(1<<ICES1)|(1<<CS11);

TIMSK1 = (1<<ICIE1);

}

// Motor direction pins (PD4/PD5)

void motor\_dir\_init(void) {

DDRD |= (1<<IN1\_PIN)|(1<<IN2\_PIN);

PORTD |= (1<<IN1\_PIN); // IN1=1

PORTD &= ~(1<<IN2\_PIN); // IN2=0 → forward

}

int main(void) {

motor\_dir\_init();

pwm0\_init();

adc\_init();

uart\_init();

icap\_init();

sei();

while (1) {

if (UCSR0A & (1<<RXC0)) {

char c = UDR0;

if (c >= '0' && c <= '9' && rx\_pos < sizeof(rx\_buf) - 1) {

rx\_buf[rx\_pos++] = c;

}

else if ((c=='\r' || c=='\n') && rx\_pos) {

rx\_buf[rx\_pos] = '\0';

uint16\_t v = atoi(rx\_buf);

if (v > 255) v = 255;

override\_speed = (uint8\_t)v;

override\_flag = 1;

rx\_pos = 0;

}

}

// set\_speed

if (override\_flag) {

set\_speed = override\_speed;

} else {

uint16\_t raw = adc\_read();

set\_speed = raw >> 2;

}

OCR0A = set\_speed;

// Update measured\_speed

if (new\_period) {

new\_period = 0;

if (period\_counts) {

uint32\_t freq = F\_CPU/8/period\_counts;

measured\_speed = (freq > 255 ? 255 : (uint8\_t)freq);

} else {

measured\_speed = 0;

}

}

// Print on BAUD 9600

uint16\_t temp = ADC; // last ADC reading

printf("%u %u %u\n", temp, set\_speed, measured\_speed);

\_delay\_ms(200);

}

}

A screenshot of a computer program

AI-generated content may be incorrect.

Successful Compilation

A screen shot of a computer

AI-generated content may be incorrect.

Successfully reading values from motor and potentiometer.

The first value is ADC, second value is the potentiometer value, and third value is the measured speed.

A screenshot of a computer

AI-generated content may be incorrect.

SerialPlot test