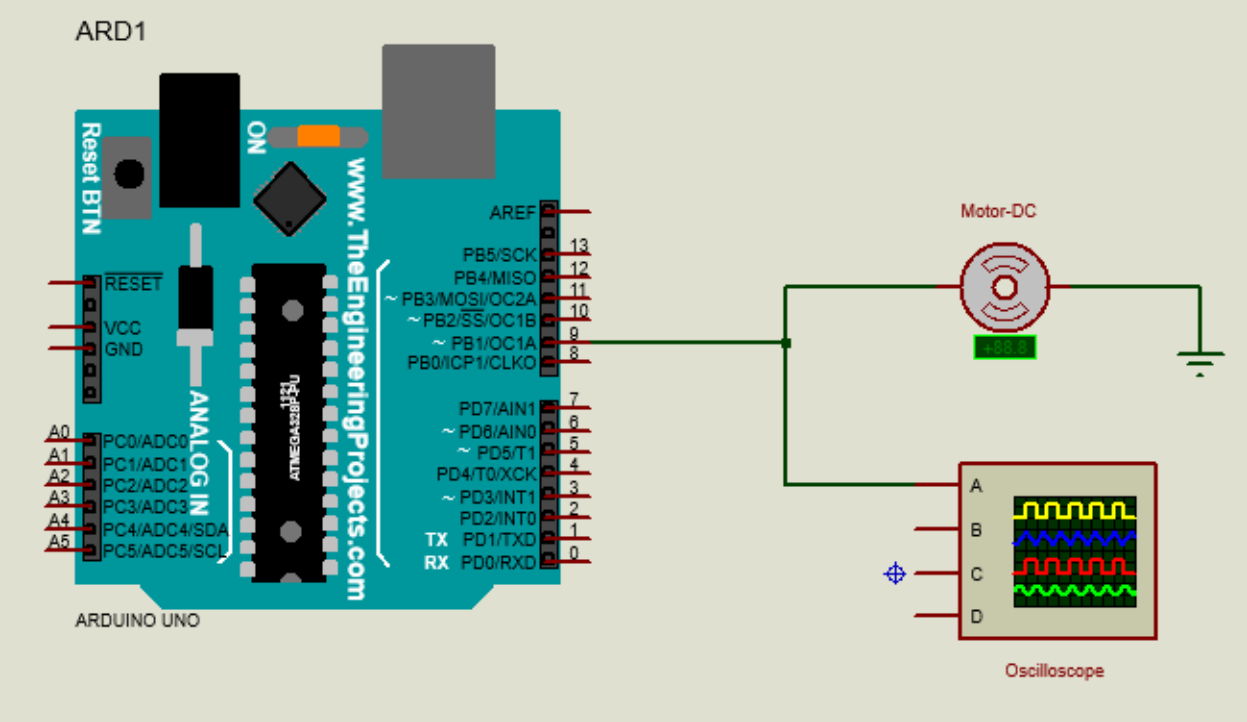
# LAB – 04

You are highly encouraged to go through the **Fast PWM Mode** topic in ATmega32P datasheet. (from page 101)

## TASK 01: GENERATING PWM USING TIMER1

Implement the given circuit in your Proteus software.




Go through the datasheet and please read about below registers configurations for **Fast PWM Mode**

### Step 01: Enable the inverting mode

**Table 15-3. Compare Output Mode, Fast PWM(1)**

| COM1A1/COM1B1 | COM1A0/COM1B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC1A/OC1B disconnected. |
| 0 | 1 | WGM13:0 = 14 or 15: Toggle OC1A on compare match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected. |
| 1 | 0 | Clear OC1A/OC1B on compare match, set OC1A/OC1B at BOTTOM (non-inverting mode) |
| 1 | 1 | Set OC1A/OC1B on compare match, clear OC1A/OC1B at BOTTOM (inverting mode) |

## Step 02: Enable the Fast PWM

**Table 15-5. Waveform Generation Mode Bit Description(1)**

| Mode | WGM13 | WGM12 (CTC1) | WGM11 (PWM11) | WGM10 (PWM10) | Timer/Counter Mode of Operation | TOP | Update of OCR1x at | TOV1 Flag Set on |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Normal | 0xFFFF | Immediate | MAX |
| 1 | 0 | 0 | 0 | 1 | PWM, phase correct, 8-bit | 0x00FF | TOP | BOTTOM |
| 2 | 0 | 0 | 1 | 0 | PWM, phase correct, 9-bit | 0x01FF | TOP | BOTTOM |
| 3 | 0 | 0 | 1 | 1 | PWM, phase correct, 10-bit | 0x03FF | TOP | BOTTOM |
| 4 | 0 | 1 | 0 | 0 | CTC | OCR1A | Immediate | MAX |
| 5 | 0 | 1 | 0 | 1 | Fast PWM, 8-bit | 0x00FF | BOTTOM | TOP |
| 6 | 0 | 1 | 1 | 0 | Fast PWM, 9-bit | 0x01FF | BOTTOM | TOP |
| 7 | 0 | 1 | 1 | 1 | Fast PWM, 10-bit | 0x03FF | BOTTOM | TOP |
| 8 | 1 | 0 | 0 | 0 | PWM, phase and frequency correct | ICR1 | BOTTOM | BOTTOM |
| 9 | 1 | 0 | 0 | 1 | PWM, phase and frequency correct | OCR1A | BOTTOM | BOTTOM |
| 10 | 1 | 0 | 1 | 0 | PWM, phase correct | ICR1 | TOP | BOTTOM |
| 11 | 1 | 0 | 1 | 1 | PWM, phase correct | OCR1A | TOP | BOTTOM |
| 12 | 1 | 1 | 0 | 0 | CTC | ICR1 | Immediate | MAX |
| 13 | 1 | 1 | 0 | 1 | (Reserved) | – | – | – |
| 14 | 1 | 1 | 1 | 0 | Fast PWM | ICR1 | BOTTOM | TOP |
| 15 | 1 | 1 | 1 | 1 | Fast PWM | OCR1A | BOTTOM | TOP |

## Step 03: Select pre-scalar value 1 (No pre-scaling)

**Table 15-6. Clock Select Bit Description**

| CS12 | CS11 | CS10 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{I/O}$/1 (no prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (from prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (from prescaler) |
| 1 | 0 | 0 | $clk_{I/O}$/256 (from prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (from prescaler) |
| 1 | 1 | 0 | External clock source on T1 pin. Clock on falling edge. |
| 1 | 1 | 1 | External clock source on T1 pin. Clock on rising edge. |

For Step 01, 02 and 03 we need to set the bits in TCCR1A and TCCR1B registers.

### 15.11.1 TCCR1A – Timer/Counter1 Control Register A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x80) | COM1A1 | COM1A0 | COM1B1 | COM1B0 | – | – | WGM11 | WGM10 | TCCR1A |
| Read/Write | R/W | R/W | R/W | R/W | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### 15.11.2 TCCR1B – Timer/Counter1 Control Register B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x81) | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 | TCCR1B |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

## Step 04: Set the ICR1 value to 49999 and OCR1A to 24999

**The Final code of your task should be like in the below**

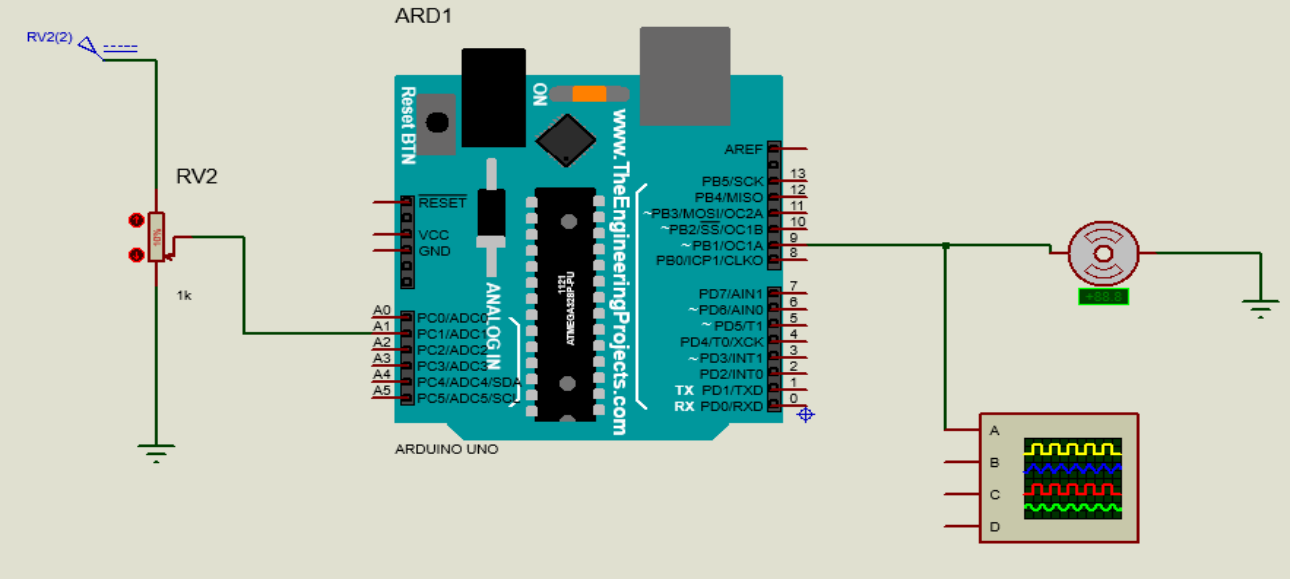
```
#define F_CPU 16000000UL
#include <avr/io.h>

void init_PWM()
{

	//*
	Please insert your codes here to (your code should do the followings)
	1.  Enable Fast PWM
	2.  Select the Inverting Mode
	3.  Set no pre-scaling
	*//

	ICR1 = 49999; // top value
}


int main(void)
{
    init_PWM();

    while (1)
    {
		OCR1A = 24999;
    }
}
```

# TASK 02: PWM CONTROL USING ADC WITH ATMEGA328P MICROCONTROLLER

**Objective:**

Learn to interface an ADC (Analog-to-Digital Converter) with PWM (Pulse Width Modulation) to control a motor speed based on an analog input.

Please implement the following circuit in Proteus




## Step 01: Configuring PWM Setup

- Set the appropriate Data Direction Register (DDR) to configure the PWM pin as an output.
- Configure Timer Control Registers (TCCR1A, TCCR1B) to set Fast PWM mode and non-inverting waveform generation mode.
- Set the Timer Control Register (TCCR1B) to adjust the prescaler for setting PWM frequency (optional).
- Set the TOP value in the ICR1 register for 16-bit PWM (e.g. ICR1 = 59999).

```
void init_PWM()
{
        DDRX =                  // Set PORTB as output
        TCCR1A |=               // Non-inverting mode, Mode 14 (Fast PWM)
        TCCR1B |=               // Fast PWM with ICR1 as top, no prescaling
        ICR1 = 59999;           // Top value
}
```

Note: You need to complete the code (i.e. set correct pins in the registers)

---

## Step 02: Setting Up ADC Configuration

1. **Initialize ADC:**
    - ❖ Enable the ADC.
    - ❖ Configure the ADC Prescaler to 128.
    - ❖ Select the ADC Reference Voltage source (select internal 1.1V reference) by setting appropriate bits in the ADC Multiplexer Selection Register (ADMUX).
2. **Selecting ADC Input Channel:**
    - ❖ Choose the ADC input channel.

```
void init_ADC()
{
	ADCSRA |=		// ADC enabled and prescaler set to 128
	ADMUX |=		// Internal 1.1V reference, left adjust result
	ADMUX |=		// select ADC input channel
}
```

Note: You need to complete the code (i.e. set correct pins in the registers)

---

## Step 03: Reading ADC Value

1. **Start Conversion:**
    - ❖ Trigger the ADC conversion by setting the ADC Start Conversion (ADSC) bit in the ADC Control and Status Register A (ADCSRA).
2. **Wait for Conversion:**
    - ❖ Implement a loop to wait for the conversion to complete by checking the ADC Conversion Complete (ADSC) bit in the ADC Control and Status Register A (ADCSRA).
3. **Read ADC Result:**

- Retrieve the ADCH result from the ADC Data Register (ADCH for 8-bit resolution) after the conversion is complete.

```
uint8_t read_ADC()
{
	// Start conversion
	// Wait for conversion to complete
	// Return high byte of ADC result (8-bit resolution)
}
```

Note: Please complete the code based on the comments in above snippet.

---

## Step 04: Implementing Main Program Logic

1. **Main Program Loop:**

   - Continuously read the ADC value.
   - Scale the ADC value to match the PWM duty cycle range.
   - Update the PWM duty cycle (e.g., OCR1A register) based on the scaled ADC value.

2. **Delay for Stability:**

   - Add a small delay to stabilize the system and ensure smooth PWM operation.

```
int main(void)
{
	init_PWM();
	init_ADC();

	uint16_t adc_value;
	uint16_t pwm_value;

	while (1)
	{
		adc_value =		// Read ADC value
		pwm_value = (uint16_t)(((uint32_t)adc_value * 60000) / 255);
// Scale to 0-59999
		OCR1A =			// Set PWM duty cycle

		// Small delay for stability
	}
}
```

Note: You need to complete the code.

---