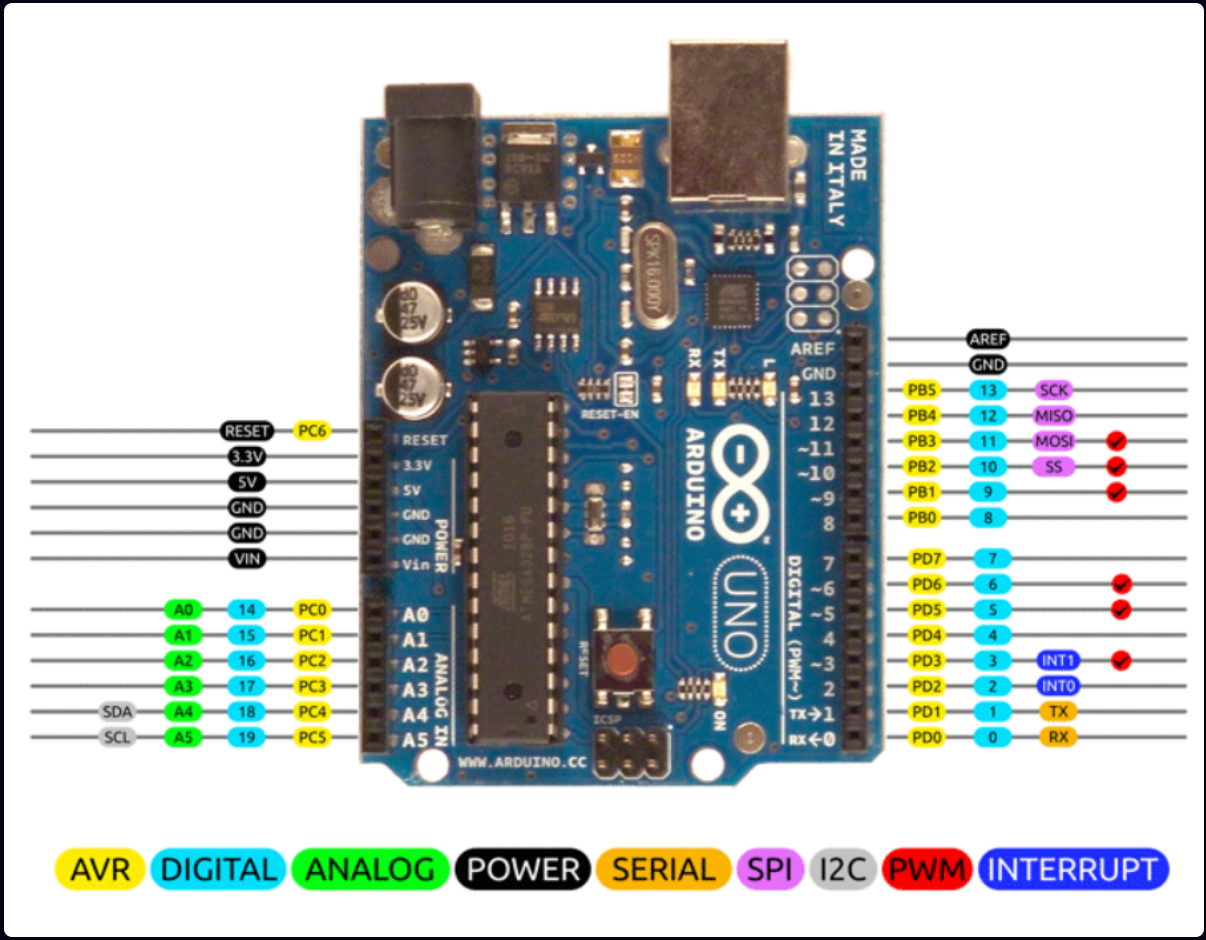
ES Lab exam

Yarab el Khlas

26 April 2025

LAB 1 - IO

- Each PORT has 3 registers. i.e. port B:
  - DDRB ⇒ Set Direction (INPUT 0, OUTPUT 1)
  - PORTB ⇒ OUTPUT ⇒ Assign Value to the port
  - PINB ⇒ INPUT ⇒ Read Value form the port



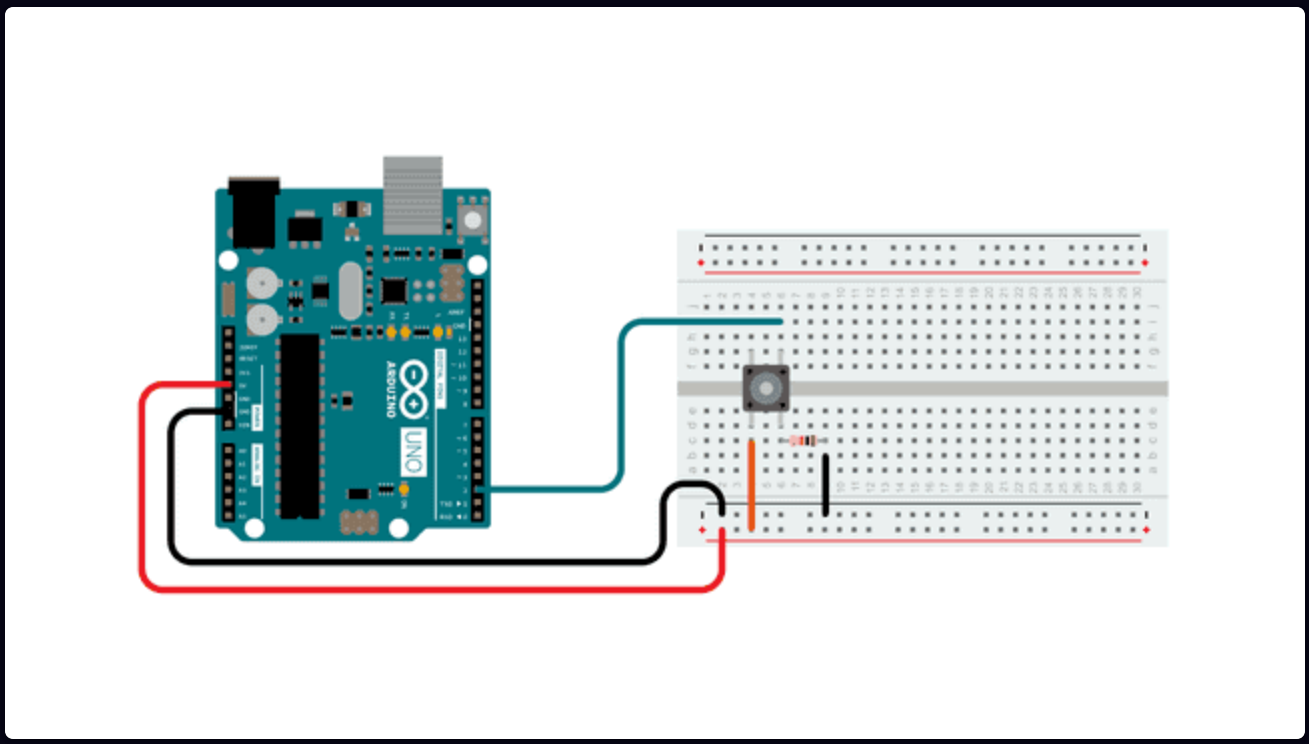
ARDUINO SHOWING PORTS

> **Notice**
> Written In Yellow are PIN Names defined in arduino
> So you can reference them in your code.
> On the other hand, SPI for example you CAN't write the port in purple. use the yellow names.

How to set, unset, toggle

```
PORTB ⊨  (1 << PB3); // set    pin PB3 to 1
PORTB &= ~(1 << PB3); // unset  pin PB3
PORTB ^=  (1 << PB3); // toggle pin PB3
```

Connecting a switch



LAB 2 - Interrupts

External Interrupts

- ★ INT0 Assigned to PIN (PD2)
- ★ INT1 Assigned to PIN (PD3)

To enable or disable interrupts

- set or unset PIN (I - 7) in SREG (Status Reg)
- Notice than in INT Init functions we disable interrupts at the beginning and re-enable them at the end.

❓ How to enable INT0 and INT1

- ▢ IN EIMSK (External Interrupt Mask)
  - ▢ set INT0 and/or INT1 (note INT0, INT1 are defined)

❓ How to set INT Configurations?

- ▢ IN EICRA (External Interrupt Control)
- ▢ based on the following table

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **EICRA – External Interrupt Control Register**
  - INT0 control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

  - INT1 control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

🔥 when an interrupt is called it goes to the following

```
ISR(INT0_vect)
{
```

```
    // HERE
}
```

🔥 and here is an example for setting-up INT0

```
void INIT_INT0(void)
{
  // disable int
  PIN_UNSET(SREG, 7);
  // sets the INT pin as input remember that PD2 maps to INT 0
  PIN_UNSET(DDRD, PD2);
  // enable INT 0
  PIN_SET(EIMSK, INT0);
  // configure INT 0
  PIN_SET(EICRA, ISC00);
  PIN_SET(EICRA, ISC01);
  // enable int
  PIN_SET(SREG, 7);
}
```

Lab 3 - Timers/PWM

- Timers are basically just timers
    - they work in parallel with the cpu and doesn't stop it

Timer math

- Required Time $Tr = 100ms$
    - ℹ️ Calculate Frequency $F = Fcpu/prescalar$
        - 💡 regularly $Fcpu = 16MHZ$
    - ℹ️ Clock Time Period $T = 1/F$
    - 🔥 Then the number of timer ticks required is
        - 💡 $\#ticks = Tr/T$
        - 💡 $TimerCount = \#ticks - 1$

Timer code types

1. Over flow mode

- Handled either manually by checking its values
- or via Interrupts

💡 Arduino has 3 Timers:

  - TIMER 0 ⇒ 8bit (0→255)
  - TIMER 1 ⇒ 16bit (0→65535)
  - TIMER 2 ⇒ 8 bit

🔥 Each timer has main 2 regs, timer 0 for example:
- [ ] `TCNT0`: Timer/Counter 0
- [ ] Stores the timer values
- [ ] Can be read and written
- [ ] `TCCR0B` Timer/Counter Control 0 B (There exist an A that we will use below)
- [ ] In which you select the prescalar

Right now, we will concentrate on the highlighted bits. The other bits will be discussed as and when necessary. By selecting these three **Clock Select Bits**, **CS02:00**, we set the timer up by choosing proper prescaler. The possible combinations are shown below.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Manual Mode Summary

We basically have to check the timer value in every iteration, and handle the logic ourselves.

2. overflow mode with interrupts

🔥 Each timer has 2 additional regs, timer 0 for example:
- [ ] `TIMSK0`: Timer Mask Register (enables INT on overflow for timer 0) when you set the pin `TOIE0`

TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2
TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1
TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0

AtMega DataSheet

- and
  - [ ] `TIFR0`: Timer/Counter 0 Interrupt Flag Register `TOV0` is set when there is an overflow (notice you now don't have to check it yourself) and it starts to execute the following ISR.

```
ISR(TIMER0_OVF_vect)
{
    // here
}
```

3. Compare Match Mode (CTC -> Clear Timer On Compare)

💡 In CTC mode we give a value to the timer and let it compare to it and fire interrupt for us when it finds it.

🔥 to enable CTC:

- [ ] In `TCCR1B` set set `WGM12`
- [ ] Then there exist another register for each timer:
  - [ ] `OCR1A` & `OCR1B` (Output Compare Register)
    - [ ] In it will give the value
- [ ] Then you have to enable INT in `TIMSK1` by setting `OCIE1A`

5. Timer Input Capture Mode (ICU)

- Captures timer value when external event (edge/input) occurs.

☑ No new registers (finally)

Configure as following

TCCR1B: Timer Counter Control Register B

7	6	5	4	3	2	1	0
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10

Bit 7 - ICNC1: Input Capture Noise canceller
Setting this bit activates noise canceller. It causes a delay of 4 clock cycles as it considers a change only if it persists for at least 4 successive system clocks.

Bit 6 - ICES1: Input Capture Edge select
Select edge detection for input capture function.
0 = Capture on falling edge
1 = Capture on rising edge

Bit 4: 3 - WGM13 : WGM12: Timer1 Mode select
These bits are used for mode selection like Normal mode, PWM mode, CTC mode etc. here we will select normal mode, so set these bits to zero.

🔥 this is how you check/wait for capture in your code

```
while ((TIFR & (1<<ICF1)) == 0);
// check if the input capture flag bit is one int Timer Flag Reg
```

Lab 4 - ADC

$$Resolution = \frac{Range}{Precision(i.e.\ 2^n)}$$

Registers Used (Yarab elkhlasssssss)

1. `ADMUX` ADC Multiplexer Selection register
   - we always set it to Vcc by doing `ADMUX = (1 << REFS0);
- `ADCSRA` ADC Control and Status regs

- **ADCSRA – ADC Control and Status Register A**

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **Bit 7 – ADEN – ADC Enable**
    This is enabled, ADC operations. Otherwise the pins behave as GPIO ports.
  - **Bit 6 – ADSC – ADC Start Conversion**
    1 is written as long as the conversion is in progress, When the conversion is complete, it returns to zero.
  - **Bit 5 – ADATE – ADC Auto Trigger Enable**
    1 enables auto trigger where the ADC will start a conversion on a positive edge of the selected trigger signal.
  - **Bit 4 – ADIF – ADC Interrupt Flag**
    This bit is set when an ADC conversion completes and the Data Registers are updated.

  - **Bit 3 – ADIE – ADC Interrupt Enable**
  - **Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits**

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

- **ADCL and ADCH – ADC Data Registers**
  The result of the ADC conversion is stored here.

🔥 slides code snippet (explained on it will)

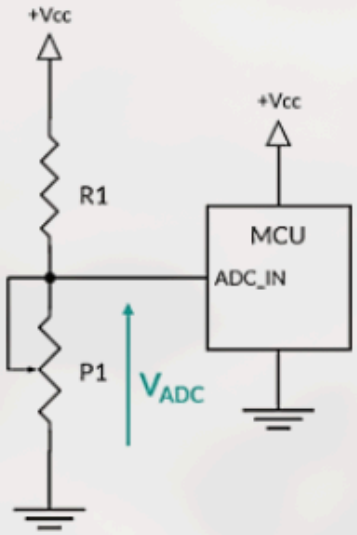
```
void adc_init()
{
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    // (1<<ADIE)=1 → set ADC interrupt enable
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
             |(1<<ADIE);
    // Set global interrupt
    sei();
}
```

Reading ADC Value

```
uint8_t ADC_endconversion_Flag=0;
int16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~5
  // ANDing with '7' will always keep the value
  // of 'ch' between 0 and 5
  ch &= 0b00000111;  // AND operation with 7
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing
  // start single conversion
  // write '1' to ADSC
  ADCSRA |= (1<<ADSC);

  // wait for conversion to complete
  // ADSC becomes '0' again
  // till then, run loop continuously
  if(ADC_endconversion_Flag==1){
    ADC_endconversion_Flag=0;
        return (ADC);
  }
  else
    return(-1);
}
```

```
ISR(ADC_vect)
{
    ADC_endconversion_Flag=1;
}
```

```
void main()
{
    uint16_t adc_result0;
    DDRB = 0x20;          // to connect led to PB5

    // initialize adc
    adc_init();
    while(1)
    {
        adc_result0 = adc_read(0);  // read adc value at PA0

        // condition for led to turn on or off
        if(adc_result0!=-1){
            if (adc_result0 > 500)
                PORTB = 0x20;
            else if ()
                PORTB = 0x00;
        }
    }
}
```



LAB 5 - SPI

NOTES:

- Make sure you connect EVERY PIN with the same as its name
- connect both arduino grounds together
- Don't forget to define the PINS first
- Not expected to come as it will require 2 arduinos for each member but it may as well come.

Code

```
// master.ino
#include <avr/interrupt.h>
#include <Arduino.h>
#include <util/delay.h>

// SPI PINS
#define MISO PB4
#define MOSI PB3
#define SCK  PB5
#define SS   PB2
```

```
void init_spi_master(void) {
  DDRB  |= (1 << MOSI) | (1 << SCK) | (1 << SS);
  DDRB  &= (1 << MISO);
  PORTB |= (1 << SS);
  SPCR   = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
}


void spi_write(uint8_t data) {
  SPDR = data;
  while (!(SPSR & (1 << SPIF)));
}


uint8_t spi_read(void) {
  SPDR = 0xFF;
  while (!(SPSR & (1 << SPIF)));
  return SPDR;
}


int main(void) {
  Serial.begin(9600);

  init_spi_master();

  PORTB &= ~(1 << SS);

  _delay_ms(500);

  uint8_t data = 1;
  while (1) {
    spi_write(data);
    data++;
    _delay_ms(10);

    uint8_t received_data = spi_read();
    Serial.print("Received: ");
    Serial.println(received_data, DEC);
  }
}
```

```
// slave.ino
#include <avr/interrupt.h>
```

```c
#include <Arduino.h>
#include <util/delay.h>

// SPI PINS
#define MISO PB4
#define MOSI PB3
#define SCK  PB5
#define SS   PB2

void init_spi_master(void) {
  DDRB  |= (1 << MOSI) | (1 << SCK) | (1 << SS);
  DDRB  &= (1 << MISO);
  PORTB |= (1 << SS);
  SPCR   = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
}

void init_spi_slave(void) {
  DDRB  |= (1 << MISO);
  DDRB  &= ~(1 << MOSI) & ~(1 << SCK) & ~(1 << SS);
  SPCR   = (1 << SPE) | (1 << SPR0);
}


void spi_write(uint8_t data) {
  SPDR = data;
  while (!(SPSR & (1 << SPIF)));
}


uint8_t spi_read(void) {
  SPDR = 0xFF;
  while (!(SPSR & (1 << SPIF)));
  return SPDR;
}


int main(void) {
  Serial.begin(9600);

  init_spi_salve();

  PORTB &= ~(1 << SS);
```

```
  uint8_t datareceived = 0;

  while (1) {
    datareceived = spi_read();
    Serial.print("Received: ");
    Serial.println(datareceived, DEC);

    datareceived += 100;
    spi_write(datareceived);
  }
}
```