



Vietnamese-German University

**Electrical Engineering & Information Technology**

# Lab Experiment

# Microcontroller

---

**Tri. B. Minh**

8th August, 2019

**Prof. Peter Nauth**

## Contents

Syllabus .....	2
Getting started Atmel Studio V7.....	3
<b>Lab Experiment 1. Input output ports.....</b>	<b>11</b>
Overview.....	11
Tasks:.....	11
Hardware setup and prepare.....	11
Example Code: .....	12
<b>Lab Experiment 2: Timer .....</b>	<b>13</b>
Task:.....	13
Hardware setup and prepare.....	13
Some Hint:.....	16
Example Code: .....	17
Question: .....	18
<b>Lab Experiment 3: Interrupt .....</b>	<b>19</b>
Hardware setup and prepare.....	19
Task:.....	20
Some Hint:.....	20
Register Description.....	21
<b>Hand on Project: Fun with LCD.....</b>	<b>23</b>
<b>Lab Experiment 4: Analog to Digital Converter (ADC).....</b>	<b>25</b>
Task:.....	25
Hardware setup and prepare.....	25
Some Hint:.....	26
Reference.....	28

# Syllabus

## ▪ Laboratories content

### ▪ **Lab Experiment 1: Input Output Port**

- Getting Started with IDE Atmel Studio and STK600 development kit
- Simulation program on IDE
- STK600 board with ATMEGA 2560 integrated connect to LED and switch port
- Blink LED Port with delay 200 ms
- Write a program which assigns each of the buttons SW0 to SW7 display via LED0 to LED7
- Write a program which will increase and decrease a variable when you press button SW0 and SW7, respectively.
- Laboratory Report & Submit the commented program code

### ▪ **Lab Experiment 2: Timer and Pulse Width Modulation (PWM)**

- Connect Port D (PD) with 8 switches and Port B (PB) with 8 LEDs. Write and execute programs PB2 blink at a frequency of 5Hz. (*Note: Do not use delay function*)
- Design a program that turns the eight LEDs like a running light from left to right and switch back. Every 200 ms, the running light should move one step further.
- Generate a PWM signal with frequency is 50 Hz. If PB0 is pressed, the duty cycle is 5%, if PB1 is pressed, the duty cycle is 7.5%. Using the CTC Mode and Fast PWM mode. Observation the signal on Oscilloscope.
- Laboratory Report & Submit the commented program code

### ▪ **Lab Experiment 3: Interrupts**

- Connect PD with 8 switches and PB with 8 LEDs.
- Write a program that flashes an LED with a frequency of 1 Hz. External Interrupt for stop and start flashing LED.
- Write and execute programs using the external interrupts, when pressing the switch connected to PD0 a variable "count" is incremented and pressing the switch PD1 "count" is decremented. Display "count" variable on Port B. Set IRQ-sense on positive edge for incrementation and on negative edge for decrementation.
- Laboratory Report & Submit the commented program code

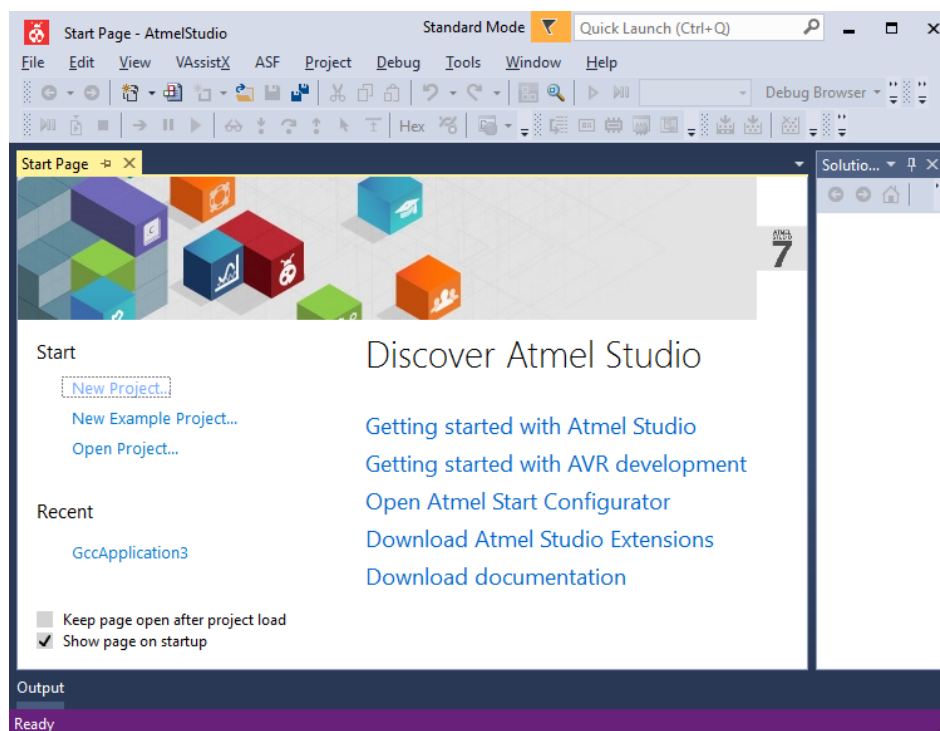
### ▪ **Lab Experiment 4: Analog to Digital Converter (ADC)**

- Write and execute programs providing functions as follows: Digitize an analog voltage applied to channel ADC 3, read the result from ADC result register than display 8 bit MSB by 8 LED, connecting to PORTB. Use ADC without interrupt.
- Same tasks 4.1 but use ADC with interrupts. When press the button the ADC will start operate.
- Same task 4.1 show ADC data on the LCD.
- Laboratory Report & Submit the commented program code

# Getting started Atmel Studio V7

## 1. Starting Atmel Studio V7

Atmel® Studio is an Integrated Development Environment (IDE) for writing and debugging AVR® /ARM® applications in Windows® XP/Windows Vista® / Windows 7/8 environments. Atmel Studio provides a project management tool, source file editor, simulator, assembler, and front-end for C/C++, programming, and on-chip debugging. Atmel Studio supports the complete range of Atmel AVR tools. Each new release contains the latest updates for the tools as well as support for new AVR/ARM devices. Atmel Studio has a modular architecture, which allows interaction with 3rd party software vendors. GUI plugins and other modules can be written and hooked to the system



**Figure 1:** The Project Related Section of the Start Page

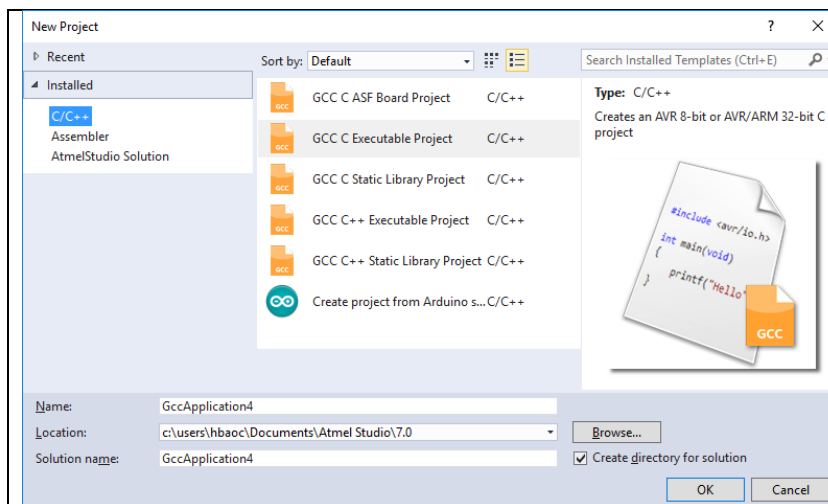
The left section of the start page contains project-related items:

- **New project** - Use this to create a new project. If you are new to the concept of software development with Atmel Studio.
- **New example project** - To take a step-by-step tour of the available Atmel platforms' functionalities using the Atmel Software Framework, click this button.
- **Open project** - Load an existing project, not mentioned on the **Recent** projects pane.

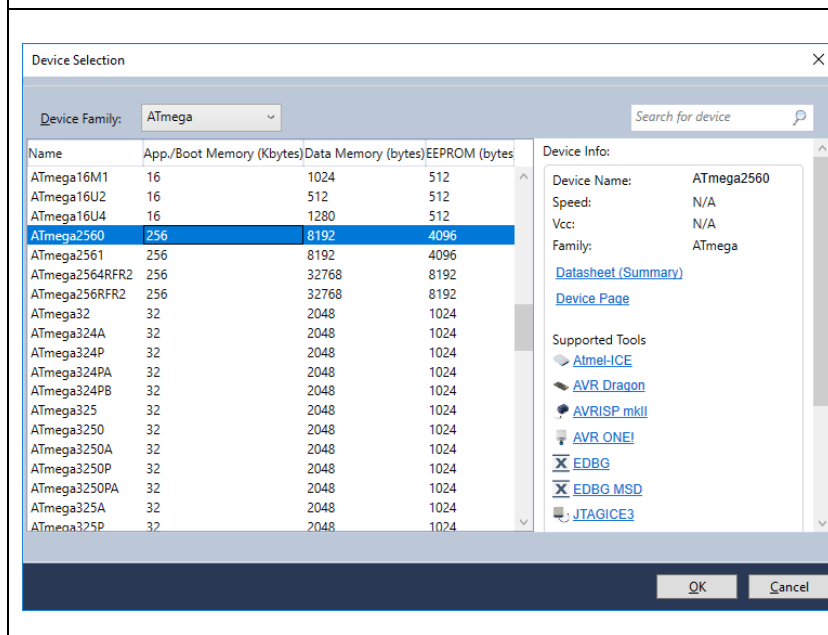
- The **Recent** projects lists the most recently opened projects. Clicking on any of the links will open the project, restoring it and the GUI to its last saved settings. You can select the number of projects you would like to be shown in the Menus and Settings.

## 2. Creating a new Atmega2560 project

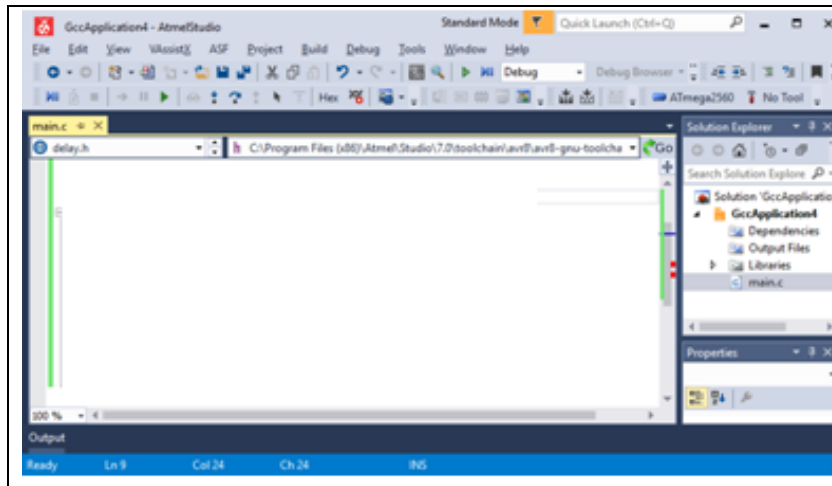
Atmel Studio is based on Visual Studio, and hence the application development process is organized into projects. The following sections demonstrates how to create a new GCC C executable project and write a simple application



- On start page (figure1). Click **New project** to create new atmel studio project. The project wizard appears.
- Select C/C++ GCC C Executable Project. this template to create an AVR 8-bit or AVR/ARM 32-bit GCC project.
- In the **Name** box, type a name for the new project. Change the location if need
- Click **OK**.

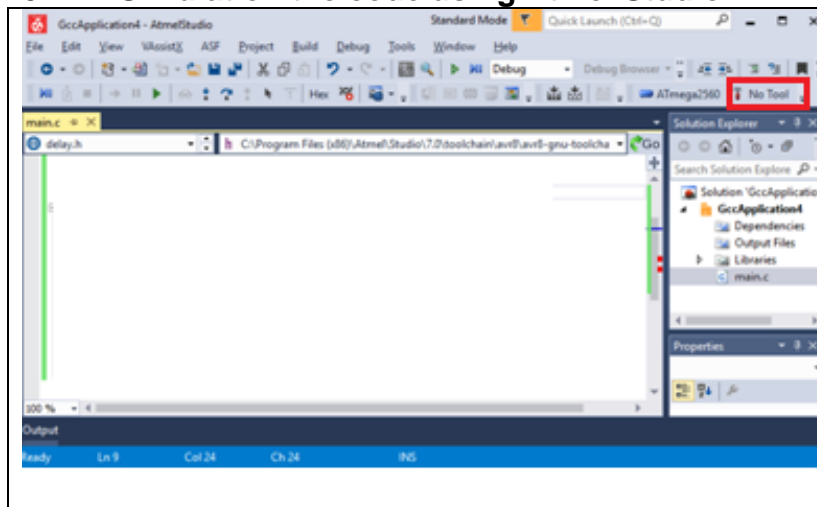


- Select **Device Family** : Atmega
- **Device name** : Atmega2560
- Click **OK**.

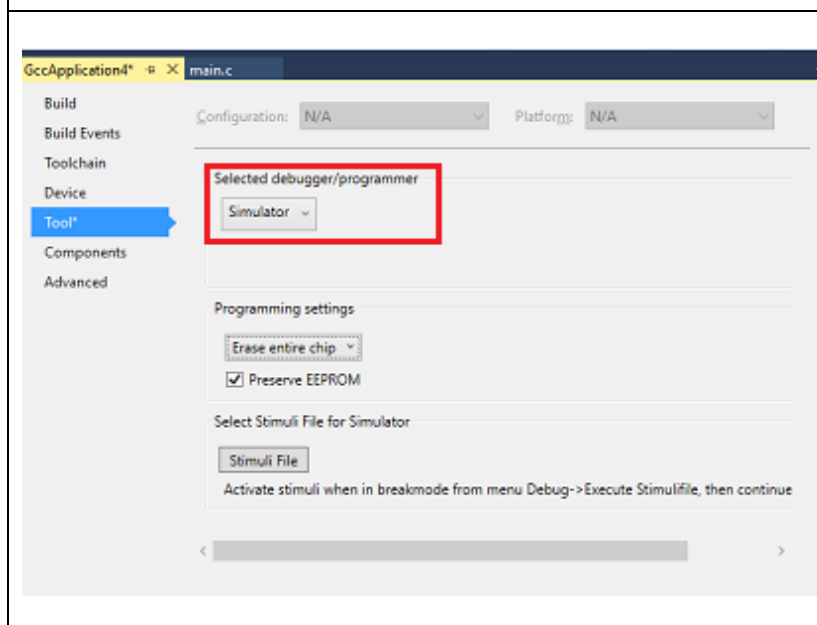



- At this point you can write your code in the editor.
- Tip: press Ctrl + K + D to format the code and make it beautiful

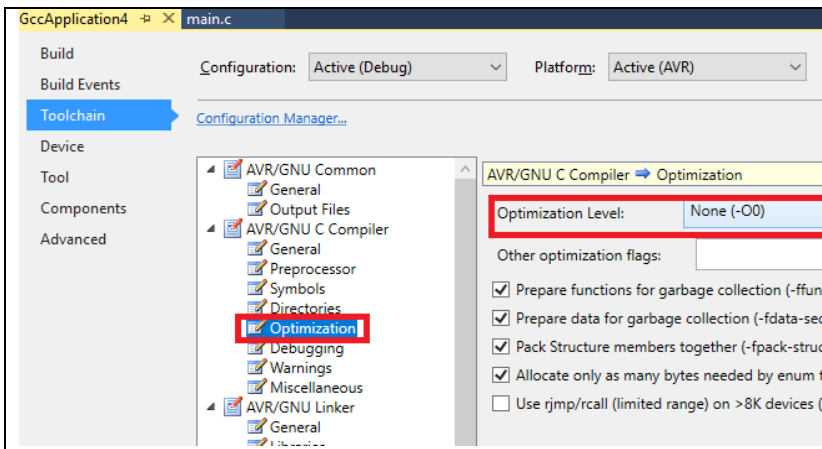
### 3. Simulation the code using Atmel Studio



- To configure simulator tool, click on red box

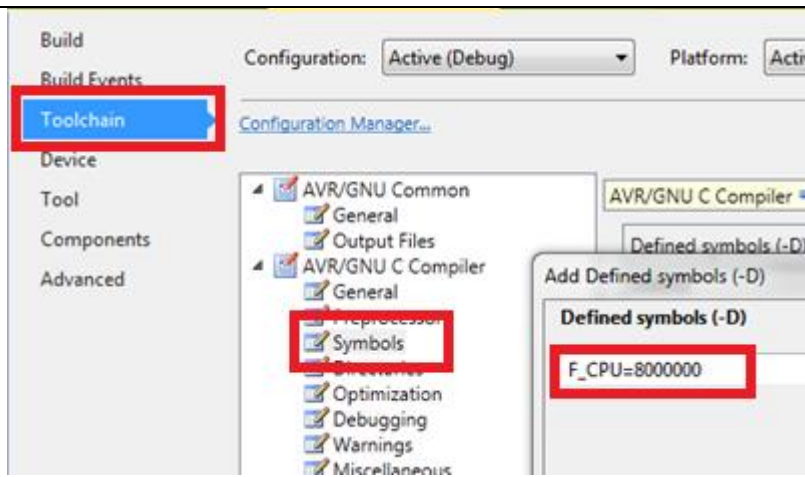


- Select simulator in box **Selected debugger/programmer**
- Click save  or **Ctrl + S** to save the configuration
- Close this window
- The other way to access this window is menu Project/Properties ( ALT +F7)



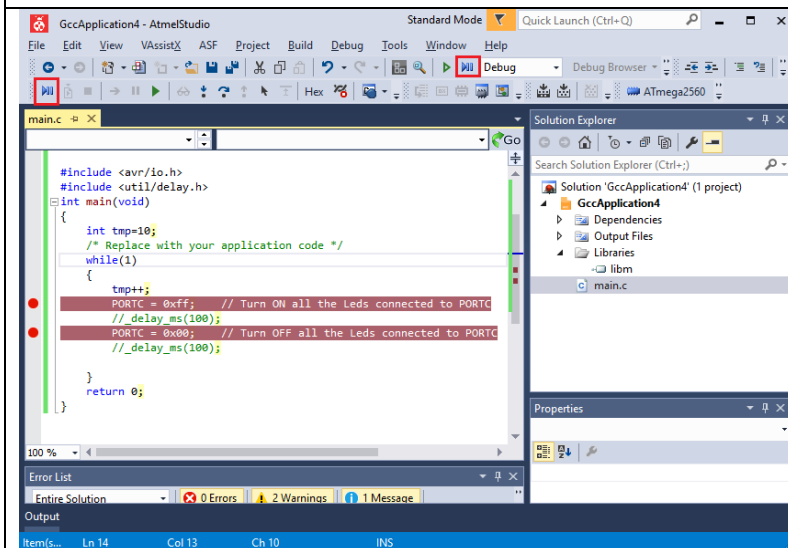
*This is an optional step. if the compiler optimize your code and you cannot debug it, chose this option to disable optimization .Warning, if do that, some delay function may not work*


- Press ALT + F7 or select menu Project/ Properties to open properties menu
- In Optimization / Optimization level choose None(-O0) or -O1

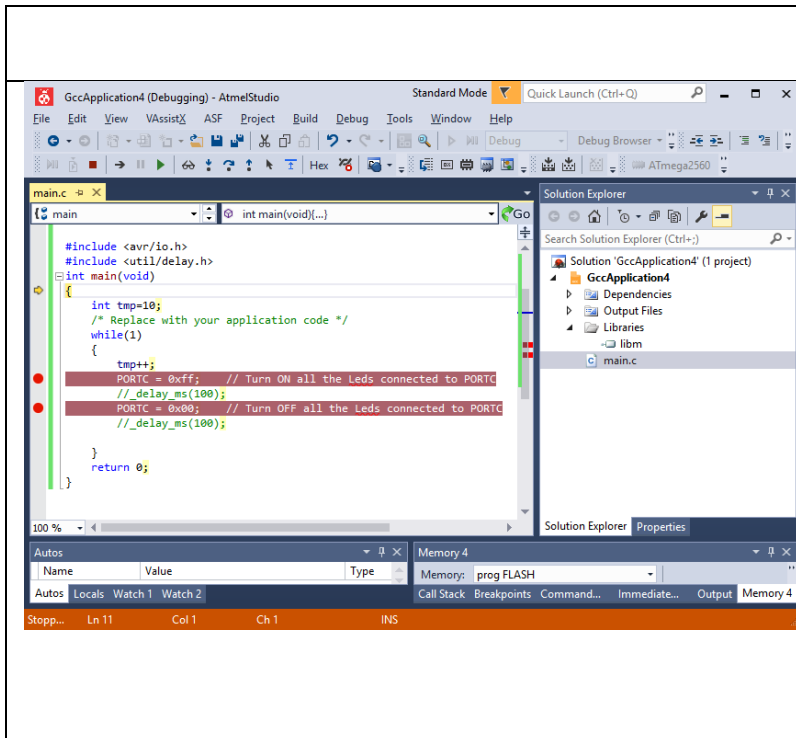


• *Declare clock using by AVR. Since Atmel AVR microcontrollers can be clocked from a variety of clock sources and can be made to run at any speed below 8MHz (newer AVRs can be clocked upto 20MHz!). So we must tell the compiler at what speed our chip is running to that it (the compiler) can calculate delays properly*

- Click OK to set F\_CPU
- Click save  or **Ctrl + S** to save the configuration

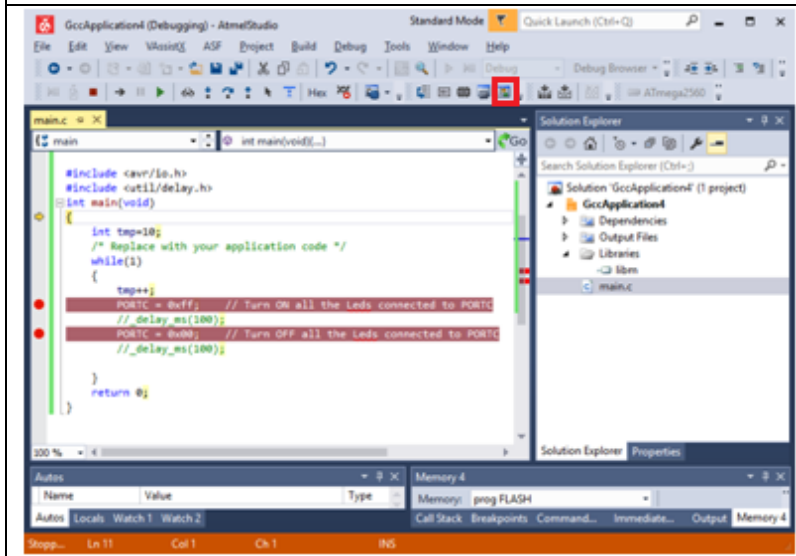


- Click start Debug  to start simulation
- Double Click on the side of editor to set break point ( red dot)
- **Don't use delay in simulation mode. In the current implementation stimulation are only evaluated between CPU single steps, meaning that the delay may be longer than specified if it would end in the middle of a**

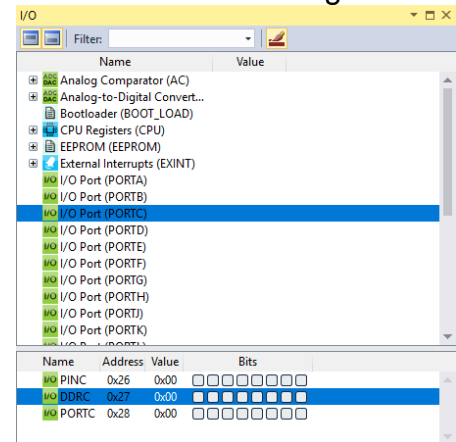


**multi-cycle instruction → cause program hang**

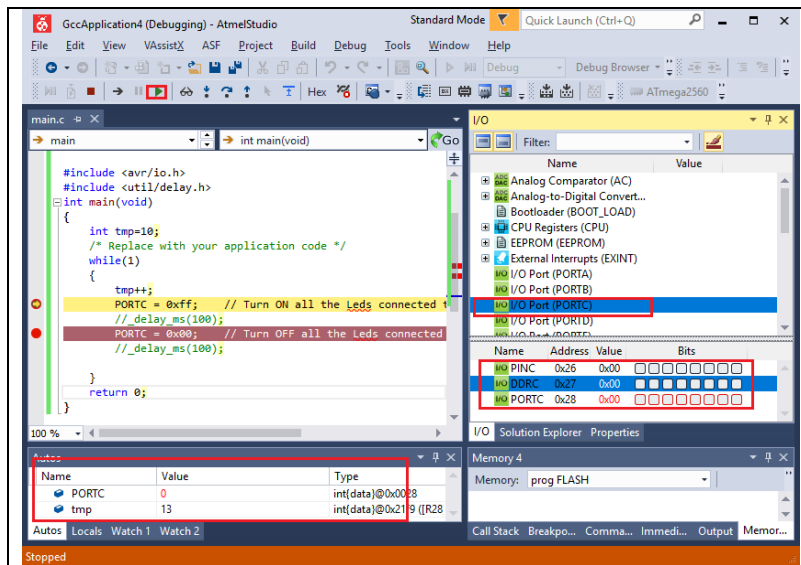
- The Editor when start simulator. The list below is some buttons (in red box) useful in debug mode like other IDE ( eg. Visual studio,..):
  - continue ( F5)
  - QuickWatch( Shift + F9)
  - Step Into(F11)
  - Step Over(F10)
  - Step Out(Shift +F1)
  - Run to Cursor(Shift + F10)
  - Reset( Shift +F5)
  - I/O to view IOstatus




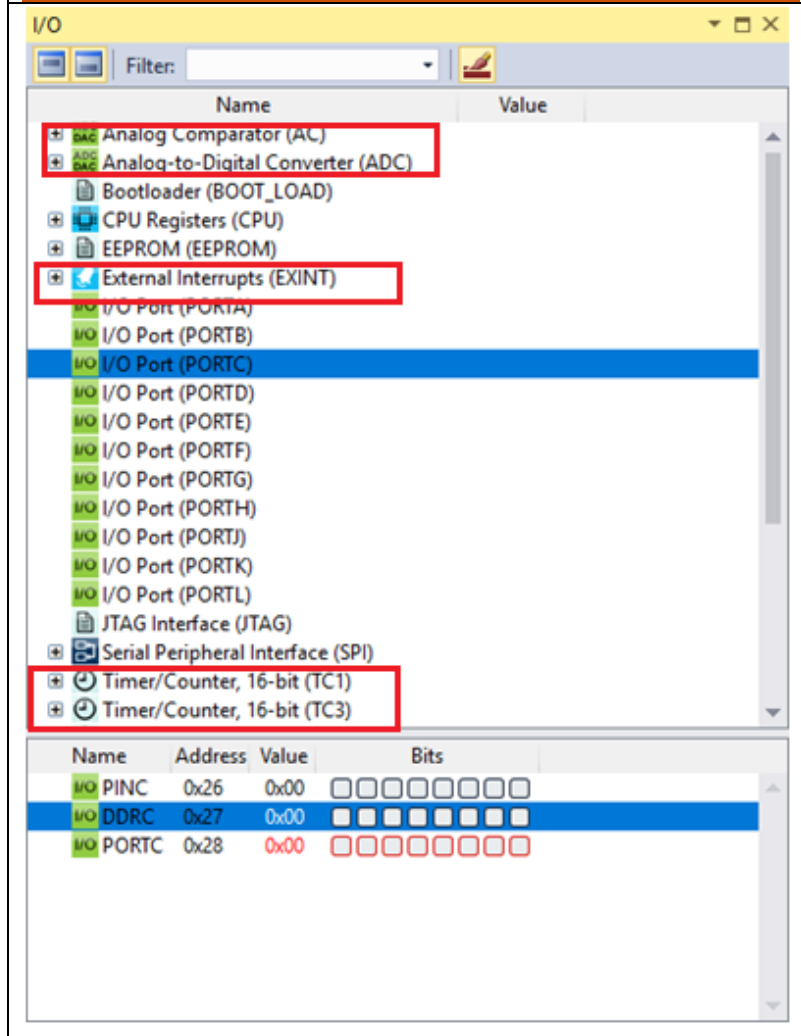
- Click on I/O view to view the status of I/O register





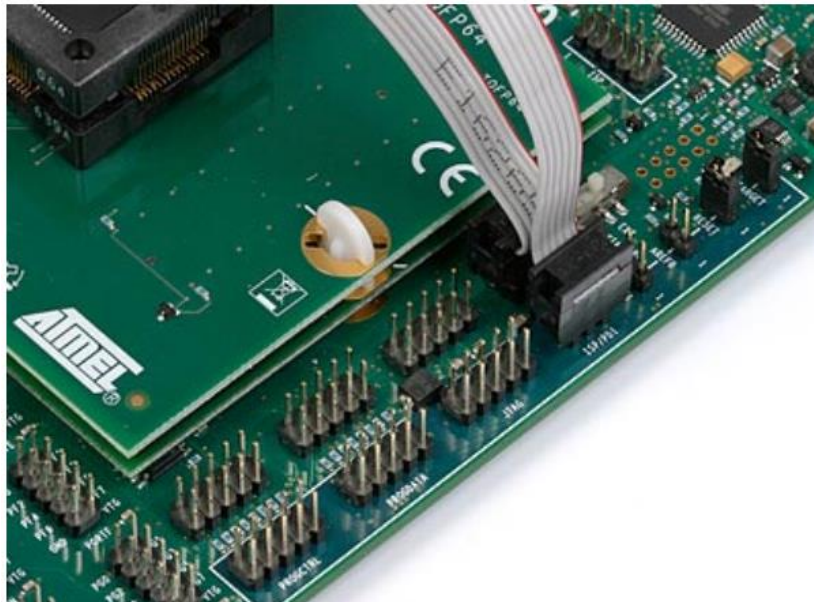


- Step by step click  to run step by step and view status of I/O and variable

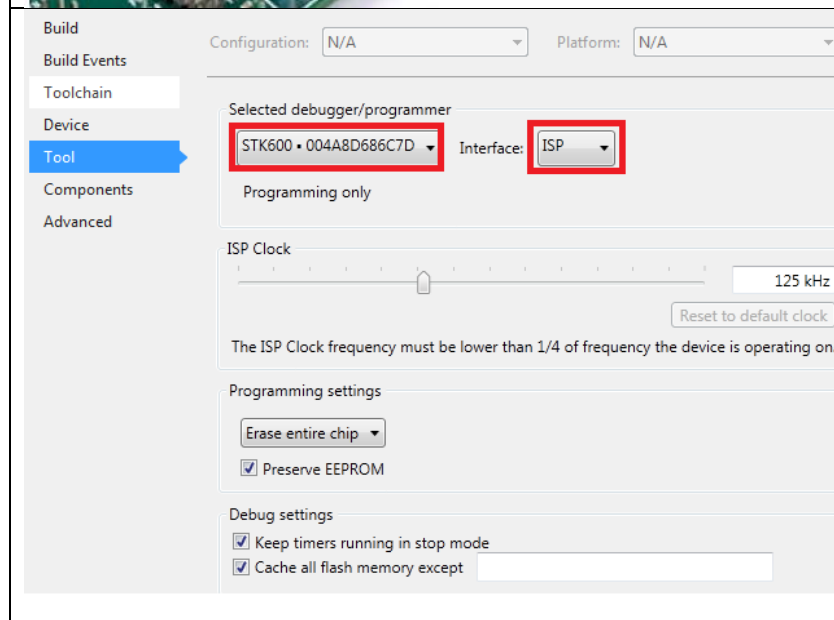



- In I/O status view , also support to simulate ADC/ External Interrupt and also Timer counter

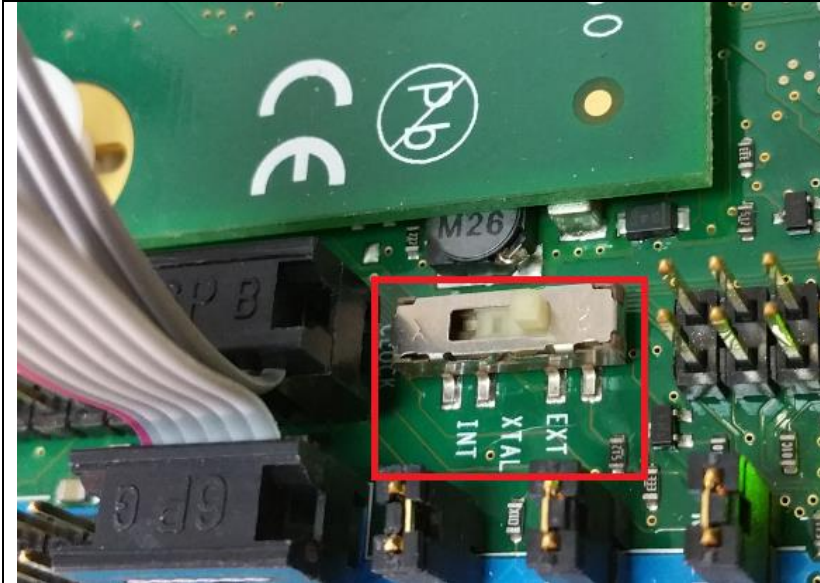
#### 4. Create project, build, and load to STK600 board



- Hardware setup like this:
- ISP programming using internal SPI (Serial Peripheral Interface) to download code into the Flash and EEPROM memory. ISP programming requires only VCC, GND, RESET, and three signal lines for programming. Connect a 6-wire cable between the two 6-pin ISP headers on the STK600. See picture beside
- Plug the USB cable to PC and install the STK600 driver.



- Create new project like project with simulator.
- *By pass step set optimization option.*
- Press ALT + F7 or select menu Project/ Properties to open properties menu
- Select STK 600 JTAG
- Click save  or **Ctrl + S** to save JTAG configuration



- Change **Clock SW** to **EXT**, that mean the CPU using the Programmable Clock Generator

Device Programming

Tool: STK600 | Device: ATmega2560 | Interface: ISP | **Apply** | Device signature: not read | Read


↓

Tool: STK600 | Device: ATmega2560 | Interface: ISP | **Apply** | Device signature: --- | Read | Target Voltage: 5.0 V | Read

Interface settings  
 Tool information  
**Board settings**  
 Card stack  
 Device information  
 Oscillator calibration  
 Memories  
 Fuses  
 Lock bits  
 Production file

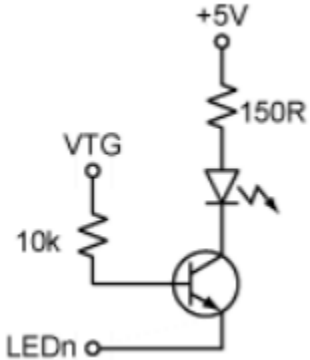
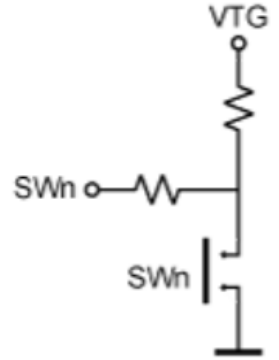
	VTarget	ARef0	ARef1	Clock generator
Generated	5.0 V	0.00 V	0.00 V	8.003 MHz
Measured	5.0 V	0.00 V	0.00 V	

Read **Write**

- Click to Device programming  (Ctrl+Shift+P)
- Click **Apply**
- Go to **Board setting** and set **VTarget =5V**, and clock generator **8Mhz**

# Lab Experiment 1. Input output ports

## Overview

LED	SWITCH
 <p>The LEDs are labeled LED0 to LED 7. The corresponding pins on the LED header have the same labels. The LED hardware is shown in the figure below. The transistor circuit ensures the LED brightness is independent of the target voltage.</p> <p>To light one of the LEDs, the corresponding pin found on the LEDs header must be pulled to GND.</p>	 <p>When pressing one of the switches, the corresponding SW pin on the SWITCHES header will be pulled low. When the switch is released, the switch's 10 kΩ pull-up will pull the line to VTG. The 150Ω resistor prevents a large current to flow to ground in case of wrong wiring.</p>

## Tasks:

- 1.1 Blink single LED PB4 in Port B with delay in 1000 ms.
- 1.2 Modify this code to blink the PB4, PB5 and PB6 with delay in 500 ms. Another LEDs in Port B will shut down. (read more page 71- AVR datasheet)
- 1.3 Blink all Port B with delay 200 ms.
- 1.4 Write a program which assigns each of the buttons SW0 to SW7 display via LED0 to LED7
- 1.5 Write a program which will increase and decrease a variable when you press button SW0 counting down when pressing SW7 counting up.
- 1.6 Laboratory Report & Submit the commented program code

## Hardware setup and prepare

No.	Equipment	Quantity
1	STK600-ATMEGA2560	1
2	2x5 Cable	2

3	2x3 Cable for ISP Programming	1
4	USB Cable	1

Connect **Port B** to **LED Port** by 2x5 cable and **Port D** to **SWITCH** by 2x5 cable.

Connect **ISP Programming 2x3 Cable**



**Example Code:**

**Task 1.1 Solution** Blink single LED PB4 in Port B with delay in 1000 ms.

```

/*
 * Blink single LED.c
 * Created: 8/6/2018 1:44:29 PM
 * Author : KNIGHT
 */
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRB = (1<<PB4);
    while(1){
        PORTB = (1<<PB4);
        _delay_ms(1000);
        PORTB = (0 << PB4);
        _delay_ms(1000);
    }
    return 1;
}

```

# Lab Experiment 2: Timer

## Task:

### Timer

2.1 Connect Port D (PD) with 8 LEDs. Write and execute programs PD2 blink at a frequency of 5 Hz. (Note: Do not use delay function)

2.2 Design a program that turns the eight LEDs like a running light from left to right and switch back. Every 200 ms, the running light should move one step further.

### Pulse Width Modulation (PWM)

2.3 Connect Oscilloscope to Pin D5 and Switch to port B. Generate a PWM signal with frequency is 50 Hz. If PB0 is pressed, the duty cycle is 5%, if PB1 is pressed, the duty cycle is 7.5%. Using the CTC Mode and Fast PWM mode. Observation the signal on Oscilloscope.

2.4 Using Fast PWM mode with Timer 1, configuration clock with 16MHz. Generate a PWM signal with frequency is 1000Hz. If PD0 is pressed, the duty cycle is 5%, If PD1 is pressed, the duty cycle is 50%. Observation the signal via pin OC1A.

2.5 Laboratory Report & Answers question & submit the commented program code.

## Hardware setup and prepare

No.	Equipment	Quantity
1	STK600-ATMEGA2560	1
2	2x5 Cable	2
3	2x3 Cable for ISP Programming	1
4	USB Cable	1

### For Timer Task 2.1 and 2.2

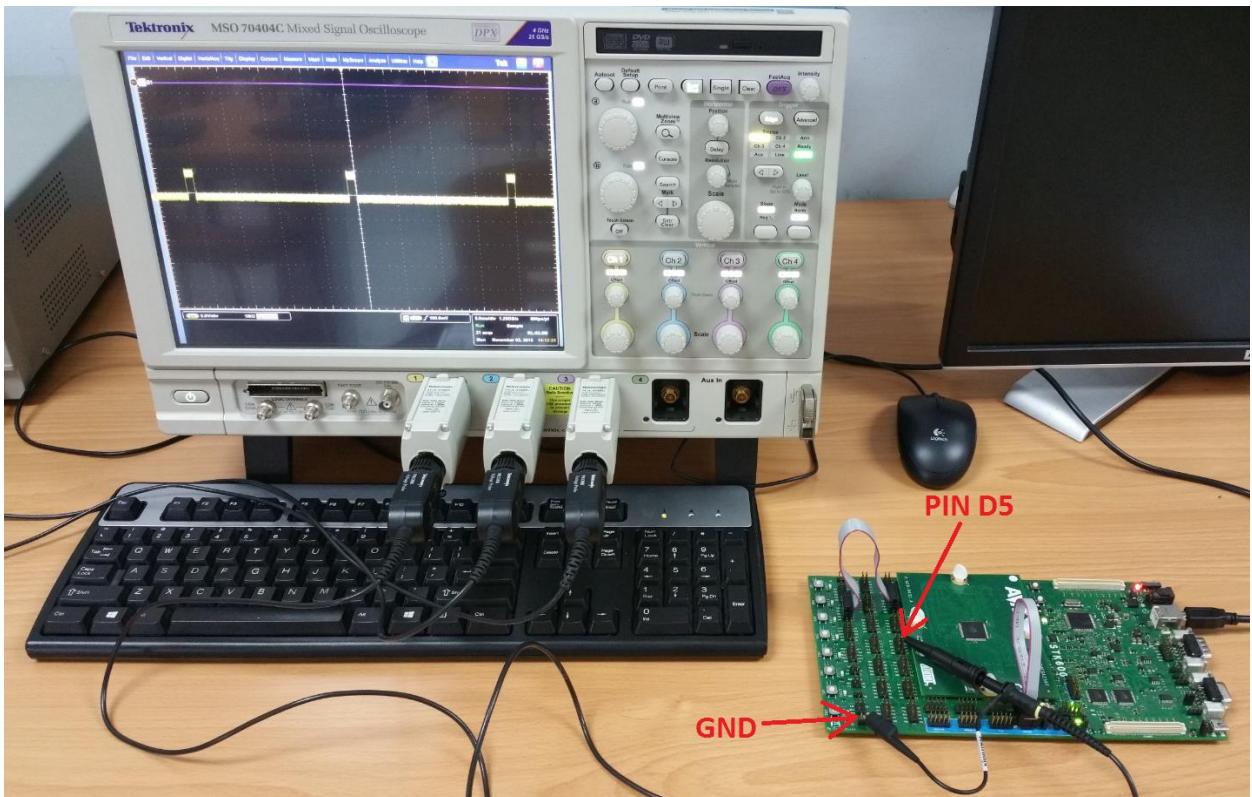
Connect **Port D** to **LED Port** by 2x5 cable and **Port B** to **SWITCH** by 2x5 cable.

Connect **ISP Programming** 2x3 Cable

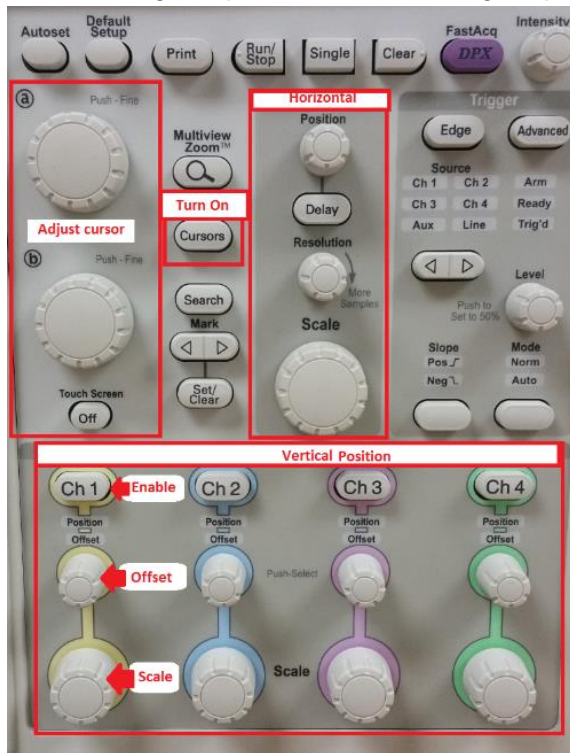


### For Pulse Width Modulation (PWM) Task 2.3

Connect one of Probe Oscilloscope to Pin D5 and reference wire to GND.



Then setting the parameters following steps:



**1. Vertical position:**

- Push **Ch1** to Enable Channel
- Adjust the **Offset** to offset the signal
- Adjust the **Scale** to get value: **5.0 V/div**

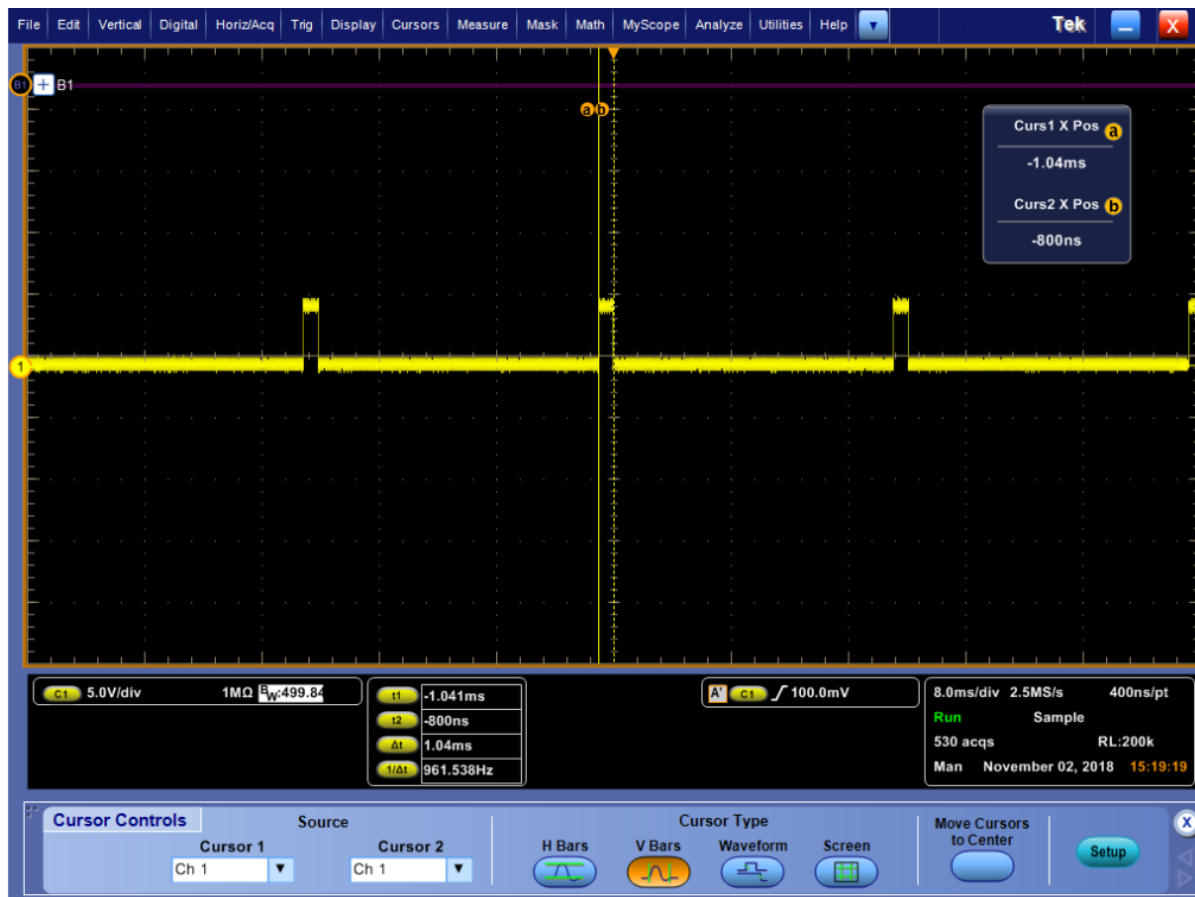
**2. Horizontal Position:**

- Adjust **Resolution** knob to get Sample Rate = **2.5 MS/s**
- Adjust the **Scale** knob to get horizontal time scale = **8 ms/div**

**3. Measurement:**

- Press to turn on the **Cursors**
- Adjust the cursor **a** and **b** to measure the duty cycle and period of the signal. Push **a** and **b** to get fine movement.

*Let see the reference set up below.*





## Some Hint:

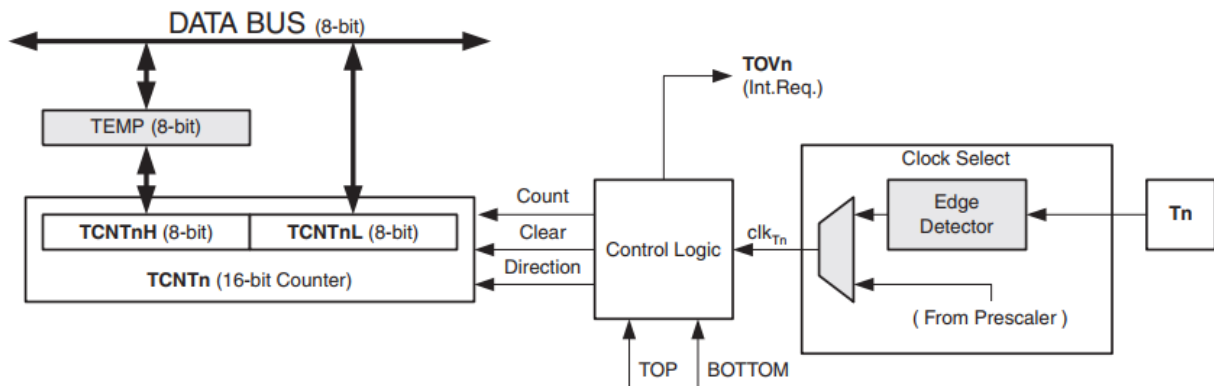
**Hint Task 2.1.** We can use the 16 bit Timer 1 with prescale =256 and F\_CPU=8 MHz.

Example Calculation with PB2 blink 200ms:

$F\_CPU / \text{prescale} / 5 (\text{state change per second})$  with  $F\_CPU = 8 \text{ MHz}$

A divisor of 256 count to **6250 = 0x186A**

$200 \text{ ms} = (1/F\_CPU) * 256 * 6250 = 125 \text{ ns} * 256 * 6250$



The 16 bit counter is mapped into two 8-bit I/O memory: Counter High (TCNTnH) and Counter Low (TCNTnL). This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus.

Prescaler base on the Register **TCCR1B**

### TCCR1B – Timer/Counter 1 Control Register B

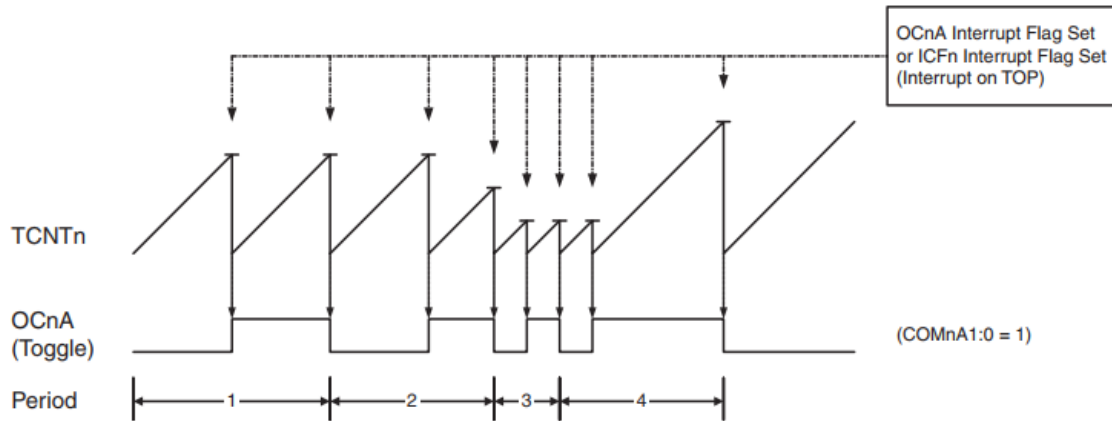
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Table 17-6.** Clock Select Bit Description

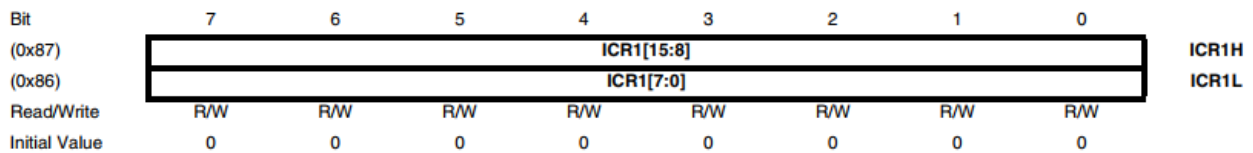
CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

**Hint Task 2.3.** The CTC Mode (Clear Timer Compare) Match Mode (Datasheet page145), the OCRnA or ICRn Register are used to manipulate the counter resolution. In the CTC Mode the counter is cleared to the zero when the counter value (TCNTn) matches either the OCRnA or ICRn.

**Figure 17-6.** CTC Mode, Timing Diagram



### ICR1H and ICR1L – Input Capture Register 1



### Example Code:

**Task 2.1 Solution** Connect Port B (PB) with 8 LEDs. Write and execute programs PB2 blink at a frequency of 5 Hz.

```

#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>

void program(void); //Create a sub-program1

int main (void)
{
    program();
}

void program(void)
{
    /**
     * Hardware configuraion:
     * PORTD: LED
     */
}

```

```

/*****
Calculation:
F_CPU/prescale/5(state change per second) with F_CPU=8 MHz
A divisor of 256 count to 6250= 0x186A
200 ms = (1/F_CPU)*256*6250 = 125 ns *256*6250
*****/
uint16_t mark=6250;

DDRD =0xFF;
PORTD =0xFF;
TCCR1B |=(1<<CS12); //Set up timer 1: CS12 (clock/256)

while(1)
{
    while(TCNT1 <= mark); //While loop compare with value
    TCNT1 = 0;
    PORTD ^= (1 << PIND2); // toggle PB2
}
}

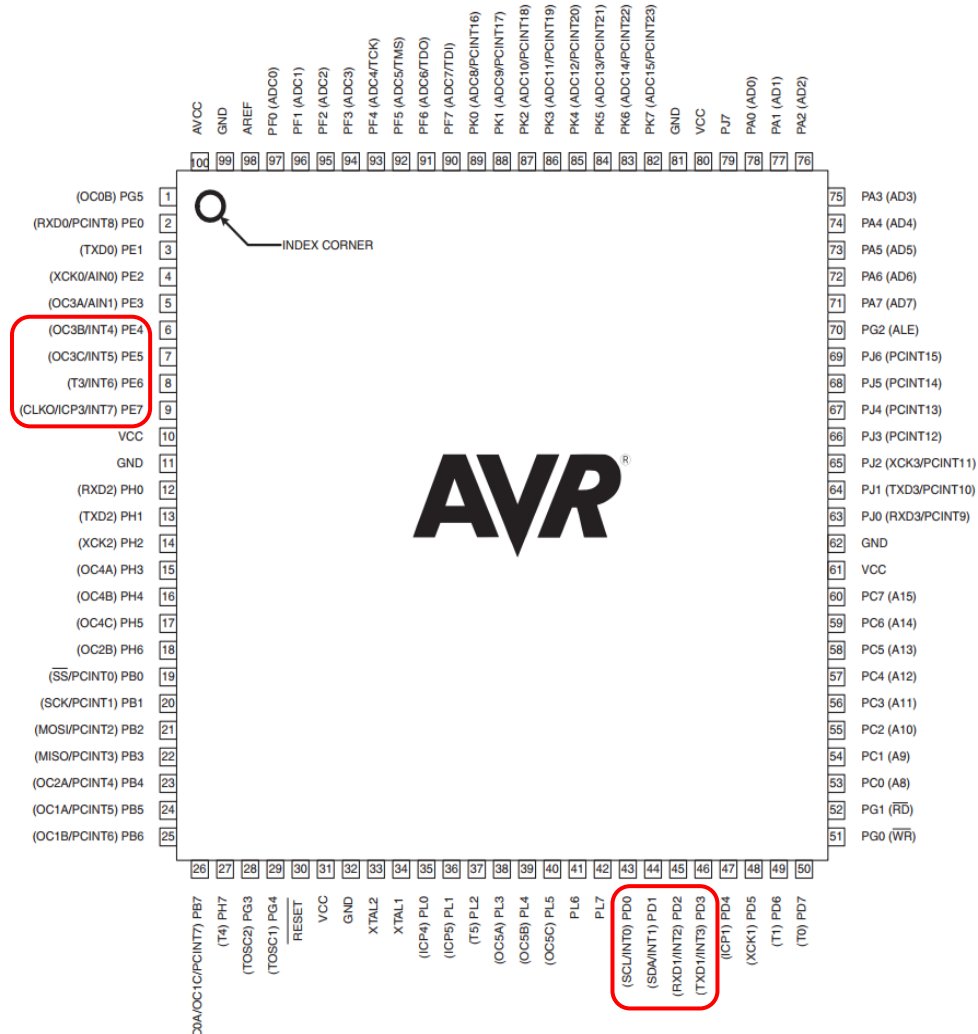
```

### Question:

1. If we change the  $F\_CPU$  from 8 MHz to 16 MHz. How to calculate the Time1 to get the frequency operating with 5Hz ?
2. What is difference between CTC Mode and Fast PWM mode?

# Lab Experiment 3: Interrupt

## Hardware setup and prepare



No.	Equipment	Quantity
1	STK600-ATMEGA2560	1
2	2x5 Cable	2
3	2x3 Cable for ISP Programming	1
4	USB Cable	1

Connect **Port B** to **LED Port** by 2x5 cable and **Port D** to **SWITCH** by 2x5 cable.

Connect **ISP Programming** 2x3 Cable



## Task:

3.1. Write a program that flashes an LED with a frequency of 1 Hz (with delay function and without delay function). Using the external interrupt for stop and start flashing LED.

3.2 Write and execute programs using the external interrupts, when pressing the switch connected to PD0 a variable "count" is incremented and pressing the switch PD1 "count" is decremented. Display "count" variable on Port B. Positive edge for incrementation and on negative edge for decrementation.

3.3 Laboratory Report should include a flowchart for each subtask & submit the commented program code

## Some Hint:

Interrupts are signal provides to the the CPU of the MCU (Microcontroller Unit), ether from the internal peripheral modules or external pins of MCU. Enabling the CPU make a jump to execute instruction routines in some predefined location based on the interrupt that occurred. When the CPU complete the routine it get back to the location from where it had made a jump.

Vector no.	Program address	Source	Port pins in ATmega2560	Interrupt definitions
1	\$0000	RESET	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	PD0	External Interrupt Request 0
3	\$0004	INT1	PD1	External Interrupt Request 1
4	\$0006	INT2	PD2	External Interrupt Request 2
5	\$0008	INT3	PD3	External Interrupt Request 3
6	\$000A	INT4	PE4	External Interrupt Request 4
7	\$000C	INT5	PE5	External Interrupt Request 5
8	\$000E	INT6	PE6	External Interrupt Request 6
9	\$0010	INT7	PE7	External Interrupt Request 7
10	\$0012	PCINT0	PCINT0:7 – PB0:7	Pin Change Interrupt Request 0
11	\$0014	PCINT1	PCINT8 – PE0 PCINT9:15 – PJ0:PJ6	Pin Change Interrupt Request 1
12	\$0016	PCINT2	PCINT16:23 – PK0:7	Pin Change Interrupt Request 2

## Register Description

### EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

- **Bits 7:0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits**

**Table 15-1.** Interrupt Sense Control<sup>(1)</sup>

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

## EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0									
0x1D (0x3D)	<table border="1"><tr><td>INT7</td><td>INT6</td><td>INT5</td><td>INT4</td><td>INT3</td><td>INT2</td><td>INT1</td><td>INT0</td></tr></table>								INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0										
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									

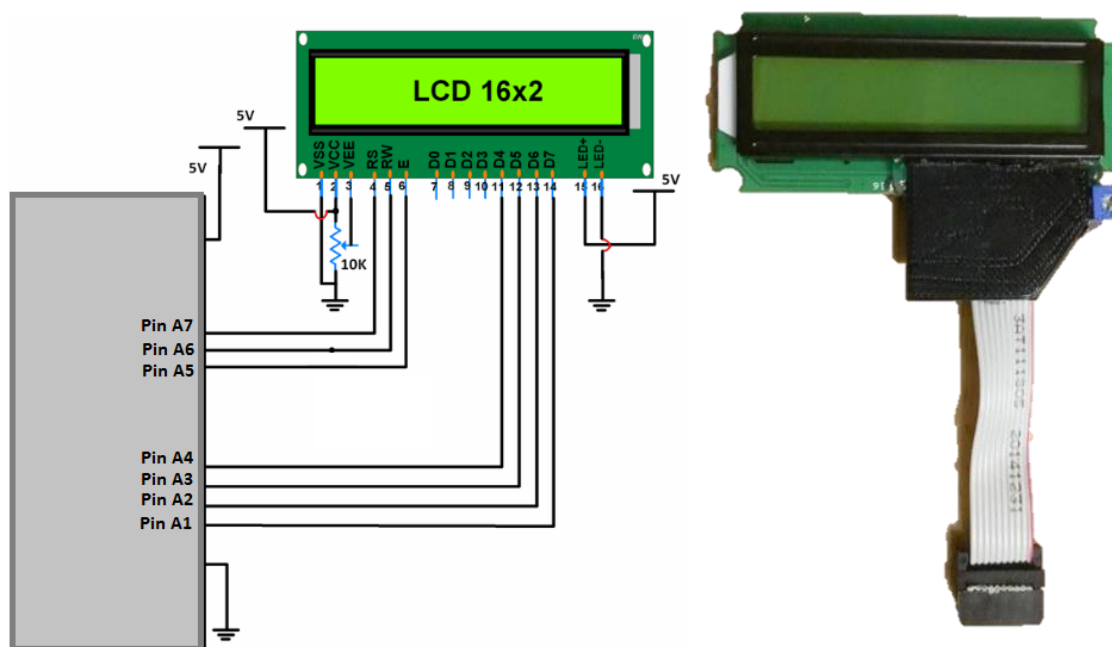
- **Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable**

# Hand on Project: Fun with LCD

We will make a hand on project with LCD. Using the schematic bellow to create your project.

Material needed

No.	Equipment	Quantity
1	LCD 16x2	1
2	2x5 Cable	1
3	Potential meter	1
4	Soldering wire, electrical wire, hot glue	



When we have finished the LCD hand-on project. Next, we are using the lcd library (**lcd.h** and **lcd.c**) to make a program show the text on the LCD. (This library should deliver on the lab)

```
/*
 * Task4.2_LCD_Task.cpp
 * Author : KNIGHT
 */
#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "lcd/lcd.h"
```



```

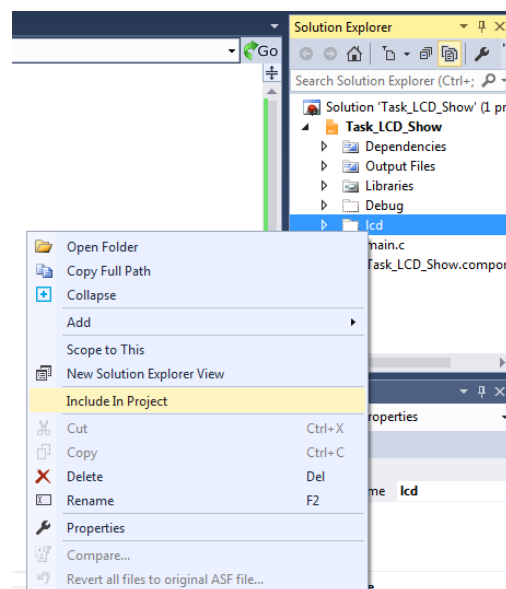
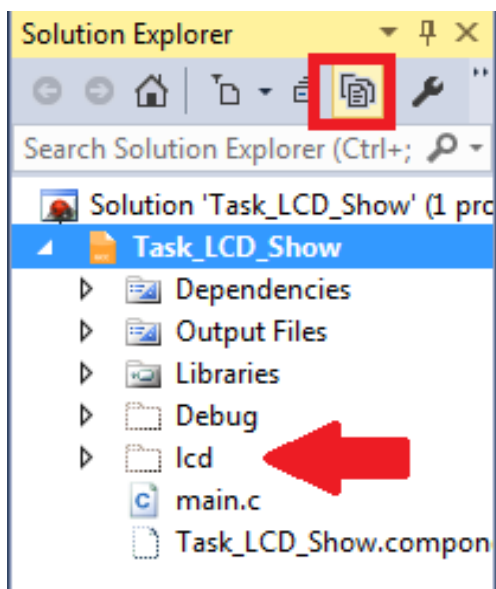
/*Hardware Configuration:
   PORT A = LCD (4bit mode)
   -----
   | PIN A1 = DB7 (LCD_DATA_PIN_3) |
   | PIN A2 = DB6 (LCD_DATA_PIN_2) |
   | PIN A3 = DB5 (LCD_DATA_PIN_1) |
   | PIN A4 = DB4 (LCD_DATA_PIN_0) |
   | PIN A5 = Enable                |
   | PIN A6 = R/W                   |
   | PIN A7 = RS                    |
   -----
*/
void program1 (void);

int main(void)
{
    // Enable your program here
    program1();
}

void program1(void)
{
    lcd_init(LCD_DISP_ON);
    lcd_clrscr();
    lcd_gotoxy(0,0);
    lcd_puts("Text on line 1");
    lcd_gotoxy(0,1);
    lcd_puts("Text on line 2");
}

```

Add **lcd.h** library to the project:



**Step1.** Show all files → **Step2.** Right click on the lcd folder → **Step3.** Choose Include In Project  
**Step4.** Add the `#include "lcd/lcd.h"` in the **main.c** program

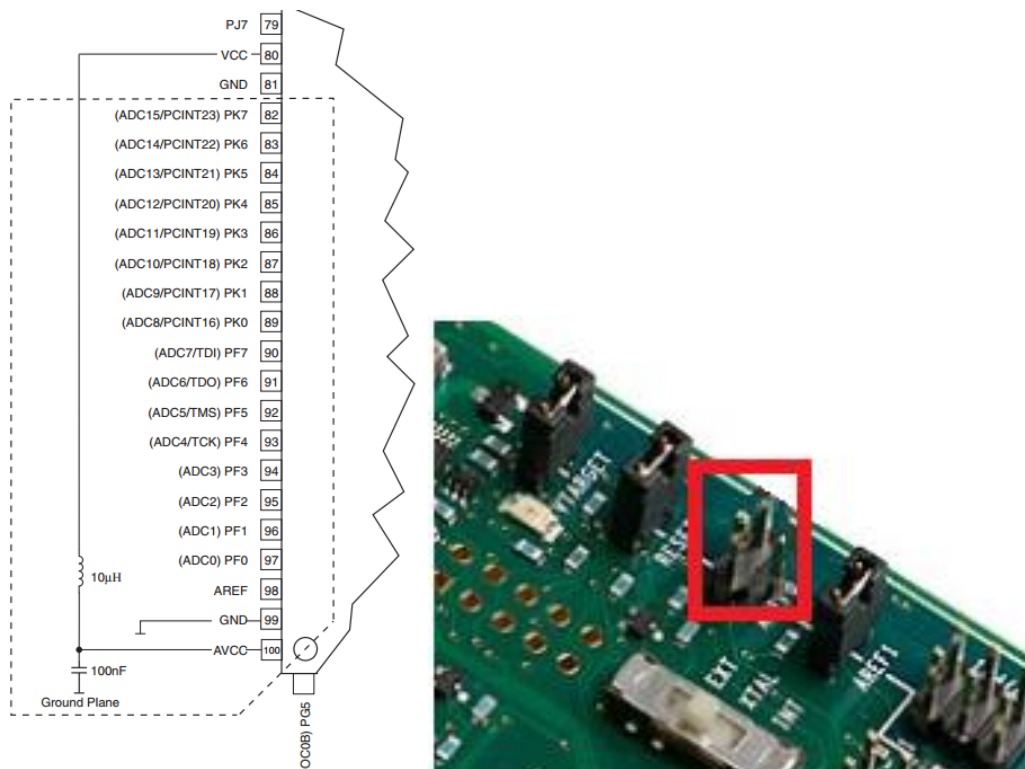
# Lab Experiment 4: Analog to Digital Converter (ADC)

## Task:

- 4.1 Write and execute programs providing functions as follows: Digitize an analog voltage applied to channel ADC 3, read the result from ADC result register than display 8 bit MSB by 8 LED, connecting to PORTB. Use ADC without interrupt.
- 4.2 Same tasks 4.1 but use ADC with interrupts. When press the button the ADC will start operate.
- 4.3 Same the task 4.1 but show value ADC on LCD.
- 4.4 Laboratory Report should include a flowchart for each subtask & submit the commented program code.

## Hardware setup and prepare

- Connect potential meter to pin ADC3. ADC3 pin is PF3 of ATMEGA2560



- Remove the jumper of the **AREF0** on STK Board.

## Some Hint:

The ATMEGA2560 has ADC with 10-bit resolution. It has 16 multiplexed single ended input channels and 14 differential input channels. 4 differential input channels are with optional gain of 10x and 200x.

### ADC Initialization:

**ADMUX:** The register contains reference selection bits , ADLAR bit which decides whether result of conversion is left adjusted or right adjusted and MUX4:0 bits select the input channel of ADC and gain selection (Table 26-4 in Atmel-2560-8-bit-AVR-Microcontroller- datasheet).

**Table 26-4.** Input Channel Selections

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
000000	ADC0	N/A		
000001	ADC1			
000010	ADC2			
000011	ADC3			
000100	ADC4			
000101	ADC5			
000110	ADC6			
000111	ADC7			

### ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 26-3. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

**Table 26-3.** Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection <sup>(1)</sup>
0	0	AREF, Internal $V_{REF}$ turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see “ADCL and ADCH – The ADC Data Register” on page 286.

- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. See Table 26-4 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

## ADCSRA: ADC Control and Status Register A

Bit (0x7A)	7	6	5	4	3	2	1	0	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**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

**Table 26-5.** ADC Prescaler Selections

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

*Enjoy your code (^-^)!*

## Reference

[1] [https://ccrma.stanford.edu/wiki/Microcontroller\\_Architecture](https://ccrma.stanford.edu/wiki/Microcontroller_Architecture)

[2] Setup and use the AVR Timer Application

[http://ww1.microchip.com/downloads/en/AppNotes/Atmel-2505-Setup-and-Use-of-AVR-Timers\\_ApplicationNote\\_AVR130.pdf](http://ww1.microchip.com/downloads/en/AppNotes/Atmel-2505-Setup-and-Use-of-AVR-Timers_ApplicationNote_AVR130.pdf)

[3] Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561\_datasheet

