LAB #1

1 LAB OBJECTIVES

- To be familiar with some the simulation software: PROTEUS VSM
- To learn how to create firmware using Atmel Studio
- How to simulate embedded system using PROTEUS and Atmel Studio
- To examine the I/O port operation using a simulator.
- Review C programming/compiling.
- Learn how to address ports/pins and set i/o configuration.
- Masking, toggling, if-then, subroutines, and looping.
- To create and debugging firmware using Atmel Studio
- To examine the I/O port operation using a simulator.
- Learn how to address ports/pins and set I/O configuration.
- Understand I/O pin drive and input levels.

2 INTRODUCTION TO ATMEL STUDIO

3 Introduction

This tutorial will teach you how to write, compile, and trace a simple program in Atmel Studio. Atmel studio is an Integrated Development Environment based on Microsoft's Visual studio IDE. It is used for developing and debugging all AVR and SAM microcontroller applications. The Atmel Studio 7 IDP gives you an environment to write, build and debug your applications written in C/C++ or assembly code. It also connects to the debuggers, programmers and development kits that support AVR and SAM devices.
The IDE was initial developed by Atmel corporation but since 2016, Microchip Technology acquired the Atmel Corporation and, for all practical purses, Atmel studio is deprecated and replaced by Microchip Studio. We will use this IDE for this lab while I am still reevaluating a better IDE to use[1].

[1]There exist many IDEs for embedded development. Prominently, PlatformIO IDE provides a new generation toolset for embedded C/C++ development and has an easy to use plugin for the incredibly popular Microsoft's Visual Studio Code. We might use it in future lab exercises.

4 Installing and using Atmel Studio

1. Download the newest version of Atmel Studio from the microchip webiste Atmel website
2. Run the downloaded program to install the Atmel Studio IDE.
3. Creating the first project
   - Go to the File menu. Choose New and then Project (Figure 1).

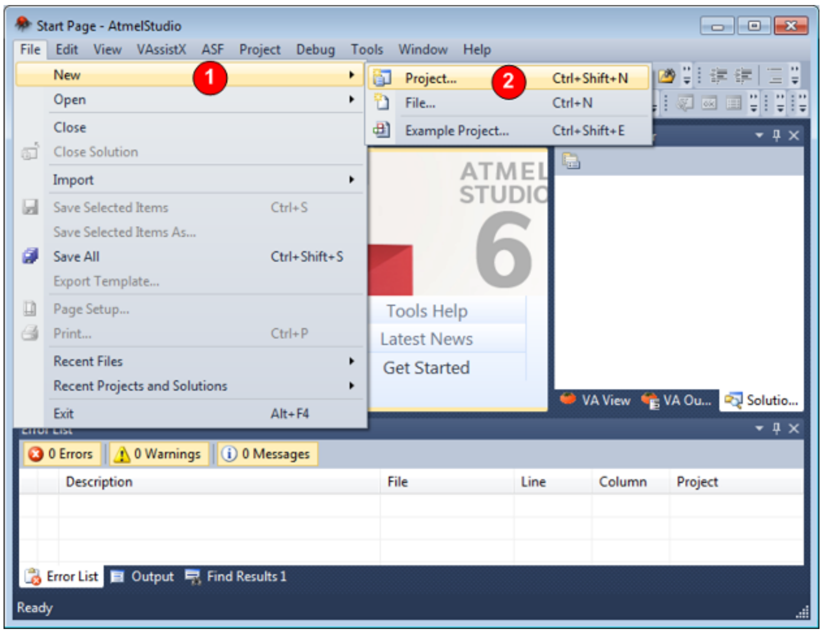
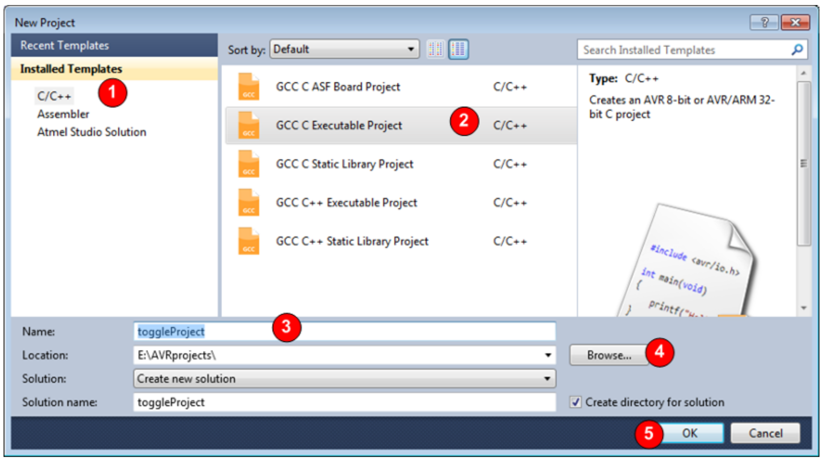


FIG 1. Creating a new Atmel project


New Project
Recent Templates
Installed Templates
C/C++ 1
Assembler
Atmel Studio Solution
Sort by: Default
Search Installed Templates
GCC C ASF Board Project C/C++
GCC C Executable Project 2 C/C++
GCC C Static Library Project C/C++
GCC C++ Executable Project C/C++
GCC C++ Static Library Project C/C++
Type: C/C++
Creates an AVR 8-bit or AVR/ARM 32-bit C project
3
Browse... 4
5 OK
Cancel

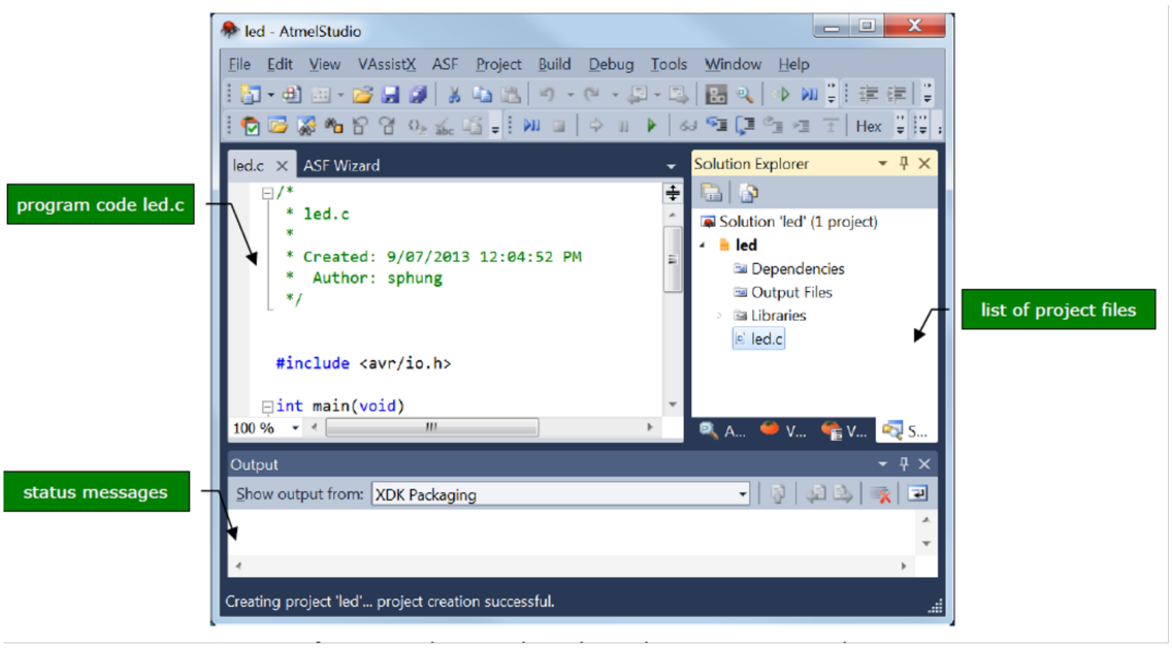
Name: toggleProject
Location: E:\AVRprojects\
Solution: Create new solution
Solution name: toggleProject
[✓] Create directory for solution

FIG 2. Select project type

   - In the opened dialog (Figure 2)
     - Choose C/C++.
     - Select GCC C Executable Project.
     - Give a name to the project (or just call it blink)

  - Choose the path where you like to save the project by clicking on the Browse button.
  - Finally press OK.
- In the Device Selection dialog
  - Select megaAVR as the Device family.
  - Select the ATMega328 chip
  - Select OK.

4. A Project will be created as shown in Figure 3



```
/*
 * led.c
 *
 * Created: 9/07/2013 12:04:52 PM
 *  Author: sphung
 */


#include <avr/io.h>

int main(void)
```

Creating project 'led'... project creation successful.

FIG 3. Atmel studio code editor

5. Enter the C code shown in Listing 1. It is not important to understand the code at this stage, but you can do that by reading the C comments[2].

```
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
  DDRB |= 1 << PB0;
  while (1)
  {
    PORTB ^= 1 << PB0;
    _delay_ms(1000);
  }
}
```

Listing 1: Source code

[2]An interested reader can follow this link to understand how the code works

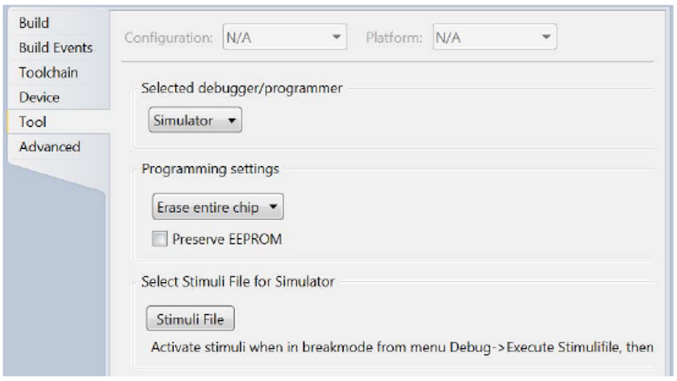
5 Compiling C code to HEX file

- Click menu Build — Build Solution to compile the C code (or just press F7).
- If there is no error message, a file called project name .hex will be produced. This file contains the machine code that is ready to be downloaded to the ATmega16 microcontroller. The file is stored in sub-folder “debug” or “release” of your project.
- If there are error messages, check your C code. Most often, error messages are caused by typographical or syntax errors. Atmel Studio will show the line numbers where errors appear in the C code.

6 Debugging C program using the simulator

Debugging is an essential aspect in any type of programming. This section will show you how to debug a C program at source-code level, using Atmel Studio. Basically, you can execute a C program one line at a time, and observe the effects on the CPU registers, IO ports, and memory. This is possible because Atmel Studio provides a software simulator for many AVR microcontrollers, including the ATmega168 chip.
To debug your project:

- Start the debugger by selecting menu Debug — Start Debugging and Break. Atmel Studio will require you to specify a debugger.
- Select “Simulator”, as shown in Figure 4



Configuration: N/A Platform: N/A

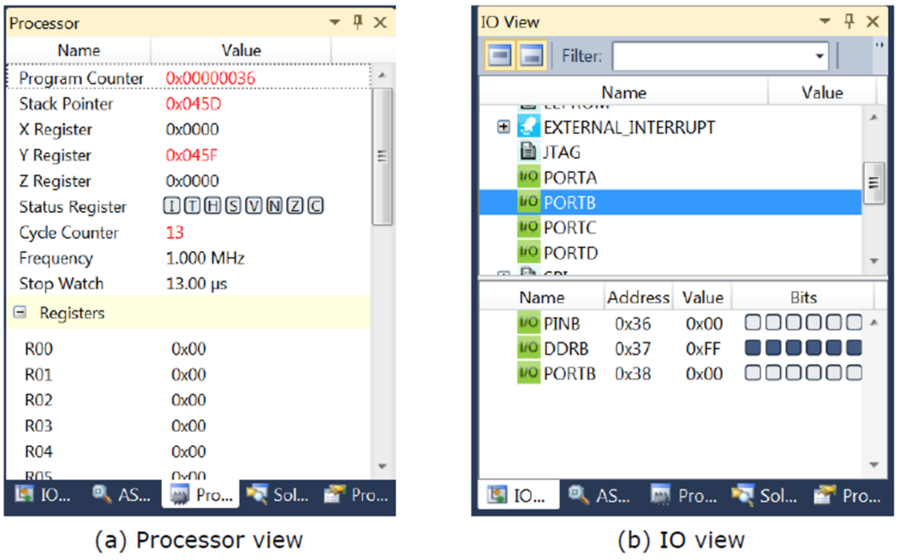
Selected debugger/programmer
Simulator

Programming settings
Erase entire chip
☐ Preserve EEPROM

Select Stimuli File for Simulator
Stimuli File
Activate stimuli when in breakmode from menu Debug->Execute Stimulifile, then

FIG 4. Configuration of the debugger

- Atmel Studio lets you examine the contents of CPU registers and IO ports. To enable these views (Figure 5), select menu Debug — Windows and then Processor View or I/O View.
- A yellow arrow (Figure 6) will appear in the code window; it indicates the C instruction to be executed next.
- Select menu Debug — Step Into (or press hot-key F11) to execute the C instruction at the yellow arrow. Note that Port B Data Direction Register (DDRB) has been changed to 0xFF.



Processor

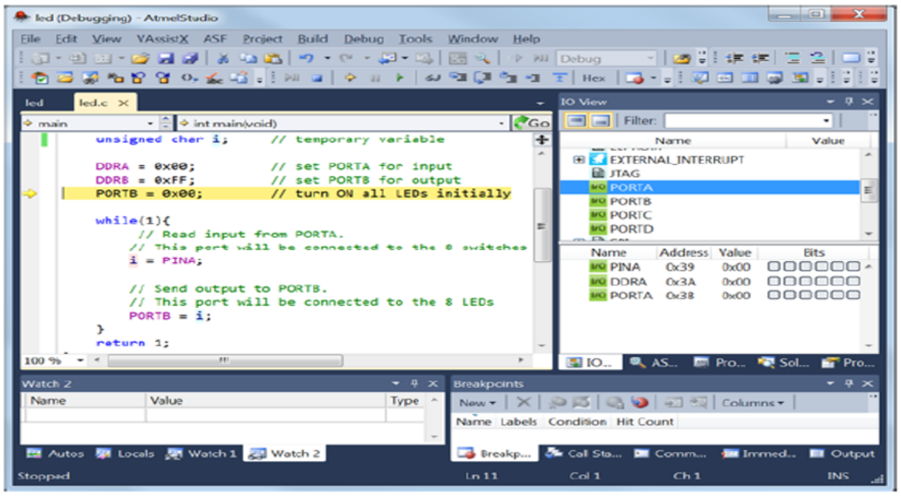
Name	Value
Program Counter	0x00000036
Stack Pointer	0x045D
X Register	0x0000
Y Register	0x045F
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	13
Frequency	1.000 MHz
Stop Watch	13.00 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00

IO View

Name	Address	Value	Bits
PINB	0x36	0x00	☐☐☐☐☐☐
DDRB	0x37	0xFF	■■■■■■
PORTB	0x38	0x00	☐☐☐☐☐☐

(a) Processor view (b) IO view

FIG 5. The contents of CPU registers and IO ports



```
unsigned char i;         // temporary variable

DDRA = 0x00;             // set PORTA for input
DDRB = 0xFF;             // set PORTB for output
PORTB = 0x00;            // turn ON all LEDs initially

while(1){
      // Read input from PORTA.
     // This port will be connected to the 8 switches
     i = PINA;

     // Send output to PORTB.
     // This port will be connected to the 8 LEDs
     PORTB = i;
}
return 1;
```

Name	Address	Value	Bits
PINA	0x39	0x00	☐☐☐☐☐☐
DDRA	0x3A	0x00	☐☐☐☐☐☐
PORTA	0x3B	0x00	☐☐☐☐☐☐

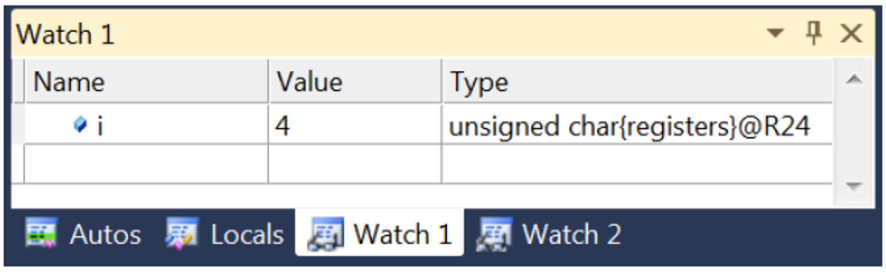
led (Debugging) - AtmelStudio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
led.c
main
int main(void)
IO View
Watch 2
Breakpoints
Autos Locals Watch 1 Watch 2
Stopped Ln 11 Col 1 Ch 1 INS

FIG 6. Debugging —The yellow arrow indicates the C instruction to be executed next

- While debugging the C program, you can change the contents of a register. For example, to change Port A Input Pins register (PINA), click on the value column of PINA and enter a new value. This change will take effect immediately. Subsequently, the contents of PORTB will be 0x04 after running the two C instructions

```
     i  = PINA;
PORTB  =  i;
```

- To monitor a C variable, select the variable name in the code window, click menu Debug — Quick Watch, and then click button Add Watch. The variable will be added to a watch window (Figure 7)
- The Debug menu provides many other debugging options, such as running up to a break point, or stepping over a function or a loop. To view the assembly code along with the C code, select menu Debug — Windows — Disassembly.



Name	Value	Type
i	4	unsigned char{registers}@R24

FIG 7. Watch window

- To stop debugging, select menu Debug — Stop Debugging.

7 Hardware simulation with PROTEUS

7.1 What is Proteus?

Proteus contains everything you need to develop, test and virtually prototype your embedded system designs based around the Atmel AVR series of microcontrollers. The unique nature of schematic based microcontroller simulation with Proteus facilitates rapid, flexible and parallel development of both the system hardware and the system firmware. This design synergy allows engineers to evolve their projects more quickly, empowering them with the flexibility to make hardware or firmware changes at will and reducing the time to market.
Proteus VSM models will fundamentally work with the exact same HEX file as you would program the physical device with, binary files (i.e. AVR Hex files) produced by any assembler or compiler.

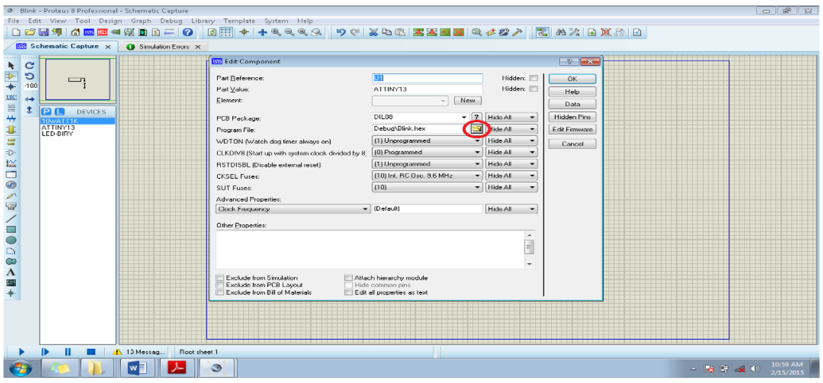
7.2 Get started to proteus

- Install proteus VSM (The detailed instructions will be given to you later)
- Select create a new Project
  - This wizard guides you through the setup of your Proteus project.
  - There is a start page in which you specify the project name and destination directory and then potentially three main screens for schematic, PCB and firmware.
  - Select create schematic by checking the box at the top of the screen and then select the default template, then press next
  - The next screen asks you to create a PCP. Just don't create on for the moment. Press next
  - Add the ATmega168 the schematic , by pressing P. You will then be presented with a window that allows you to add components. For example, to:
    - ∗ To add the ATmega168 microcontroller, just search for "ATmega168"
    - ∗ To add a red led, just search for "red led"
    - ∗ To add a 1k resistor, just search for "1k resistor"

    * To add a button switch, just search for button
    * The library contains many component. Just search for what you're looking for.
  - Build the circuit
    * Add the ATmega168 microcontroller
    * Add a red LED and rename it to LED1
    * Connect the LED to pin PB0 of the microcontroller
    * Add a current limiting resistor, and change its value to 240 Ohms
    * Add a ground

7.3 Running the simulation in Proteus

- Right click the ATtiny13 microcontroller and select edit properties (Figure 8). The browse for the hex file created with Atmel studio. Note: The hex file is usually located in the debug folder or in the release folder depending on how you compiled your program.


Blink - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Simulation Errors
DEVICES
ATTINY13
LED-BIRY
OK
Help
Data
Hidden Pins
Edit Firmware
Cancel
New
Root sheet 1

Edit Component

Part Reference: U1 Hidden:
Part Value: ATTINY13 Hidden:
Element:
PCB Package: DIL08 Hide All
Program File: Debug\Blink.hex Hide All
WDTON (Watch dog timer always on) (1) Unprogrammed Hide All
CLKDIV8 (Start up with system clock divided by 8) (0) Programmed Hide All
RSTDISBL (Disable external reset) (1) Unprogrammed Hide All
CKSEL Fuses: (10) Int. RC Osc. 9.6 MHz Hide All
SUT Fuses: (10) Hide All
Advanced Properties:
Clock Frequency (Default) Hide All
Other Properties:
Exclude from Simulation
Exclude from PCB Layout
Exclude from Bill of Materials
Attach hierarchy module
Hide common pins
Edit all properties as text

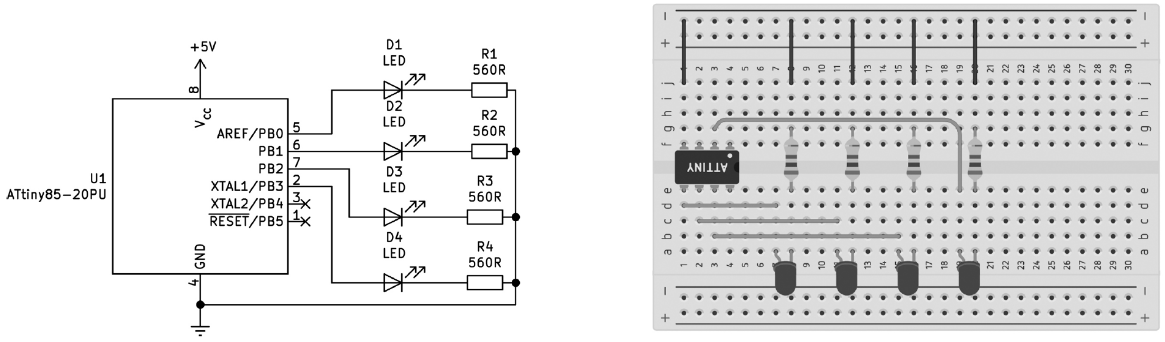
FIG 8. Specifying the hex file that will run on the microprocessor

- Select the hex file, and the press OK to dismiss the property screen
- Run the simulation
- Now, press the run button. The LED should blink
- Congratulation! Your systems works as expected!

8 Lab instruction exercise

Section 8 and Section 8 show a circuit consisting of four LEDs connected to an ATtiny13 microcontroller:

- LED 1 (D1) is green and is connected to PB0 of the MCU
- LED 2 (D2) is red and is connected to PB1 of the MCU
- LED 3 (D3) is blue and is connected to PB2 of the MCU



(a) Circuit diagram **(b)** Circuit layout on a breadboard

FIG 9. An ATtiny13 MCU powered circuit

- LED 4 (D4) is yellow and is connected to PB3 of the MCU

Write a C program that blink the 4 LED in a loop based on the value of a counter ($0 <= counter < 1000$) as follows:

- If the counter is a multiple of 3, turn on LED1 only
- If the counter is a multiple of 5, turn on LED2 only
- If the counter is a multiple of both 3 and 5, turn on LED 3 only
- If the counter is neither a multiple of 3 nor a multiple of 5, turn on LED4 only
- The counter is reset to zero If the counter is greater than 100.

HINT:

- This is a various of the classic Fizz buzz game.
- You are encouraged to take a look at a sample implementation of the implementation of fizz buzz problem

9 Lab submission

- I will check your work (Proteus simulation and the source code) at the beginning of the next lecturer.
- You can do the lab in a group of no more than 3 students
- The lab is handed out on October 3, 2022 and is due on **October 11, 2022**.
- Because the course will only lasts for 11 weeks, this is a hard deadline and **late submissions are not accepted after the deadline**.
- If you're not available in the lab, you will get a zero for the assignment (unless you can provide a documented evidence for your absence)