Experiment-2

Aim: Hands-on experimentation of ATMega32 GPIO programming in C.

Objectives: After successfully completion of this experiment students will be able to,

- Use C language for ATMega32 microcontroller programming on AVRStudio.
- Experiment with GPIO of ATMega32 on ATMega32 AVR Development Board.

Equipment required:

- Windows7 or later based host computer
- ATMega32 Development board
- USBasp Programmer
- Jumper Wires
- LED
- PUSH BUTTON

Software required:

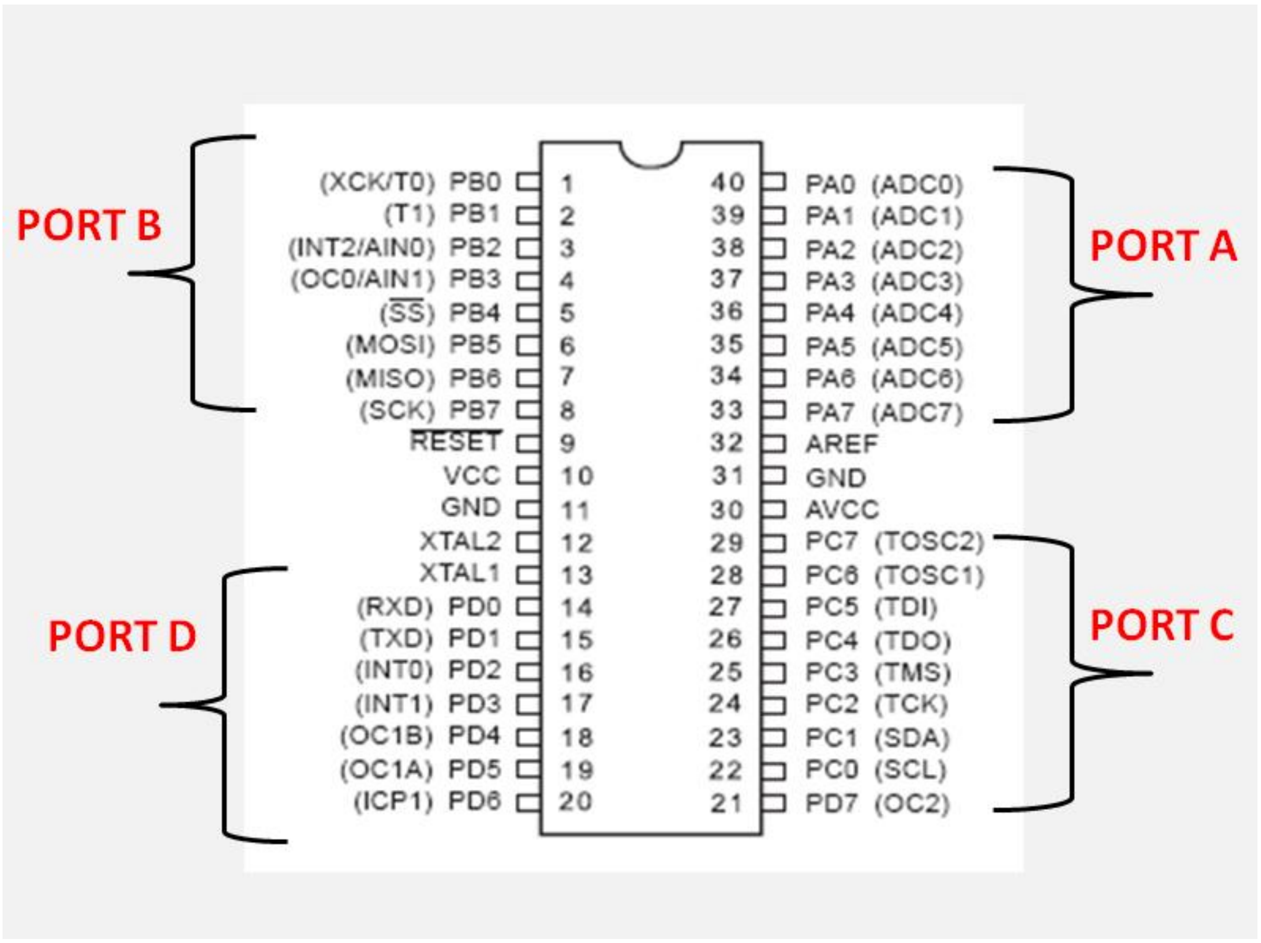
- AVR Studio7 installation setup
- USBasp driver installation setup

Theory:

Introduction of GPIO:

Interaction with peripherals is central to many microcontroller applications. Although the exact capabilities vary from one processor to the other, a set of features known as General Purpose Input/Output (GPIO) is available in most microcontrollers for the same. In short, microcontrollers can interact with the outside world with the help of GPIO's. All microcontrollers contain I/O ports to allow data to be read from switches, sensors etc. and to write to LED's, LCD displays, motors etc.

We will learn how to use AVR ports and program them to read or write from port pins. Now the port has multiple pins associated with it. For example Atmega32 has 8-bit port, i.e. it has 8 pins in a single port. Each bit represents a pin i.e. bit 0 represents pin 0 on that port and so on. As you can see in the diagram given below, Atmega32 has 4 ports named as A, B, C & D. Each of these ports has 8-pins (micro-controller pins).



Note that these port pins might have multiple functionality i.e. you can use them as GPIO or as UART or ADC etc. You can use a port pin for a single functionality at a time.

I will frequently refer to 'configuring pin' or simply 'pin'. Remember, a port has multiple pins. Thus in order to change setting for one port, you have to change setting for all port pins of that port. To change setting for one single pin of the port, you have to change a particular bit in **associated register**.

All its ports are 8 bit wide. Every port has 3 registers associated with it each one with 8 bits. Every bit in those registers configure pins of particular port. Bit0 of these registers is associated with Pin0 of the port, Bit1 of these registers is associated with Pin1 of the port, …. and like wise for other bits.

These three registers are as follows :
(x can be replaced by A,B,C,D as per the AVR you are using)
– DDRx register
– PORTx register
– PINx register

where

DDRn – Data Direction Register

PORTn – Port Output data Register

PINn – Port Input Register

7	6	5	4	3	2	1	0
PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0

PORTA

7	6	5	4	3	2	1	0
DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0

DDRA

7	6	5	4	3	2	1	0
PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0

PINA

DDRx register

DDRx (Data Direction Register) configures data direction of port pins. Means its setting determines whether port pins will be used for input or output. Writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes corresponding port pin as output.

Example:

- to make all pins of port A as input pins :
  DDRA = 0b00000000;
- to make all pins of port A as output pins :
  DDRA = 0b11111111;
- to make lower nibble of port B as output and higher nibble as input :
  DDRB = 0b00001111;

PINx register

PINx (Port IN) used to read data from port pins. In order to read the data from port pin, first you have to change port's data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins.

Now there are two input modes. Either you can use port pins as tri stated inputs or you can activate internal pull up. It will be explained shortly.

Example :

to read data from port A.
DDRA = 0x00;   //Set port a as input

- x = PINA;   //Read contents of port a

PORTx register

PORTx is used for two purposes.

1) To output data : when port is configured as output

When you set bits in DDRx to 1, corresponding pins becomes output pins. Now you can write data into respective bits in PORTx register. This will immediately change state of output pins according to data you have written.

In other words to output data on to port pins, you have to write it into PORTx register. However do not forget to set data direction as output.

Example :

to output 0xFF data on port b
```
DDRB = 0b11111111;      //set all pins of port b as outputs
PORTB = 0xFF;           //write data on port
```
●

to output data in variable x on port a
```
DDRA = 0xFF;            //make port a as output
PORTA = x;              //output variable on port
```
●

to output data on only 0th bit of port c
```
DDRC |= 0b00000001;         //set only 0th pin of port c as output
```
- PORTC |= 0b00000001;   //make it high.

Enabling Internal Pull Up Resistors:

Making the DDRx bits to 0 will configure the PORTx as Input. Now the corresponding bits in PORTx register can be used to enable/disable pull-up resistors associated with that pin.To enable pull-up resistor, set bit in PORTx to 1, and to disable set it to 0.

```
DDRB  = 0x00;  // Configure the PORTB as Input.
PORTB = 0xFF;  // Enable the internal Pull Up resistor of PORTB.
DDRD = 0xff;  // Configure PORTD as Output
PORTD = PINB; // Read the data from PORTB and send it to PORTD.
```

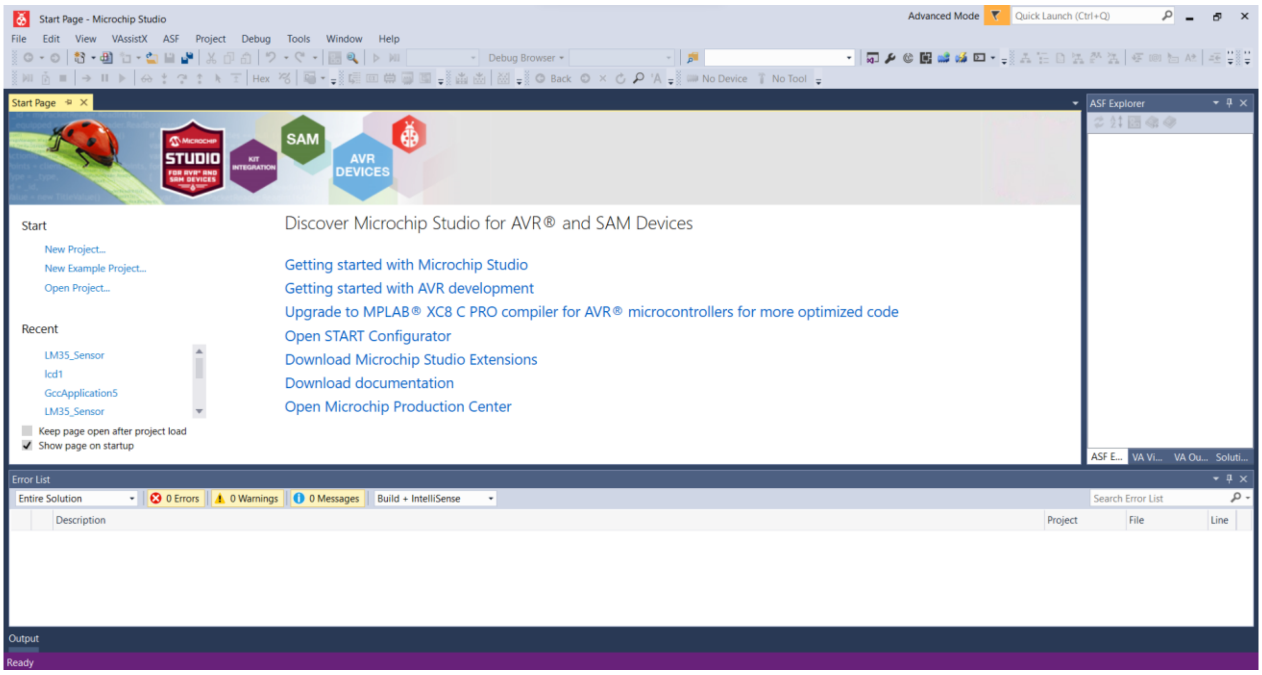
Procedure:

Similar to printing 'Hello World' in C or C++, the very first step towards programming a microcontroller is Blinking a LED with a delay. Atmega32 is a very popular high performance 8 bit AVR Microcontroller. For this example project we need to use two registers DDR and PORT. DDR stands for Data Direction Register, it determines the direction (Input/Output) of each pins on the microcontroller. HIGH at DDR register makes corresponding pin Output while LOW at DDR register makes corresponding pin Input. PORT register is the output register which determines the status of each pin of a particular port. HIGH at PORT register makes corresponding pin Logic HIGH (5V) while LOW at PORT register makes corresponding pin Logic LOW (0V).

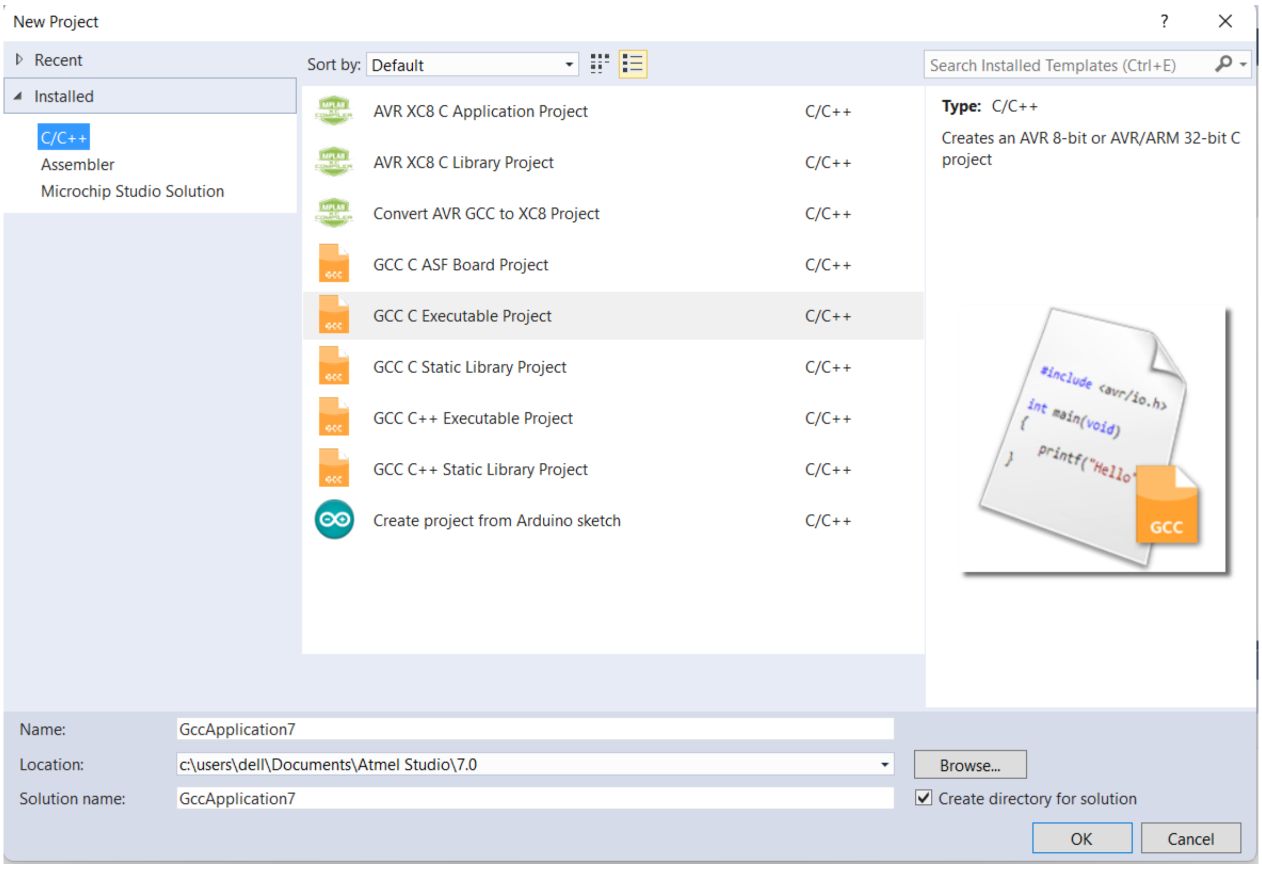
Getting started with Microchip studio 7

1. Download and Install Atmel Studio.

2. Open Atmel/Microchip Studio


Start Page - Microchip Studio
Advanced Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX ASF Project Debug Tools Window Help
Debug Browser
Hex
Back
No Device
No Tool
Start Page
ASF Explorer
STUDIO
KIT INTEGRATION
SAM
AVR DEVICES
Start
New Project...
New Example Project...
Open Project...
Recent
LM35_Sensor
lcd1
GccApplication5
LM35_Sensor
Keep page open after project load
Show page on startup
Discover Microchip Studio for AVR® and SAM Devices
Getting started with Microchip Studio
Getting started with AVR development
Upgrade to MPLAB® XC8 C PRO compiler for AVR® microcontrollers for more optimized code
Open START Configurator
Download Microchip Studio Extensions
Download documentation
Open Microchip Production Center
ASF E... VA Vi... VA Ou... Soluti...
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Description
Project
File
Line
Output
Ready

After Opening Atmel Studio
3. Select **New Project from File**


New Project

Sort by: Default

Search Installed Templates (Ctrl+E)

- Recent
- Installed
  - C/C++
  - Assembler
  - Microchip Studio Solution

AVR XC8 C Application Project	C/C++
AVR XC8 C Library Project	C/C++
Convert AVR GCC to XC8 Project	C/C++
GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

Name: GccApplication7

Location: c:\users\dell\Documents\Atmel Studio\7.0

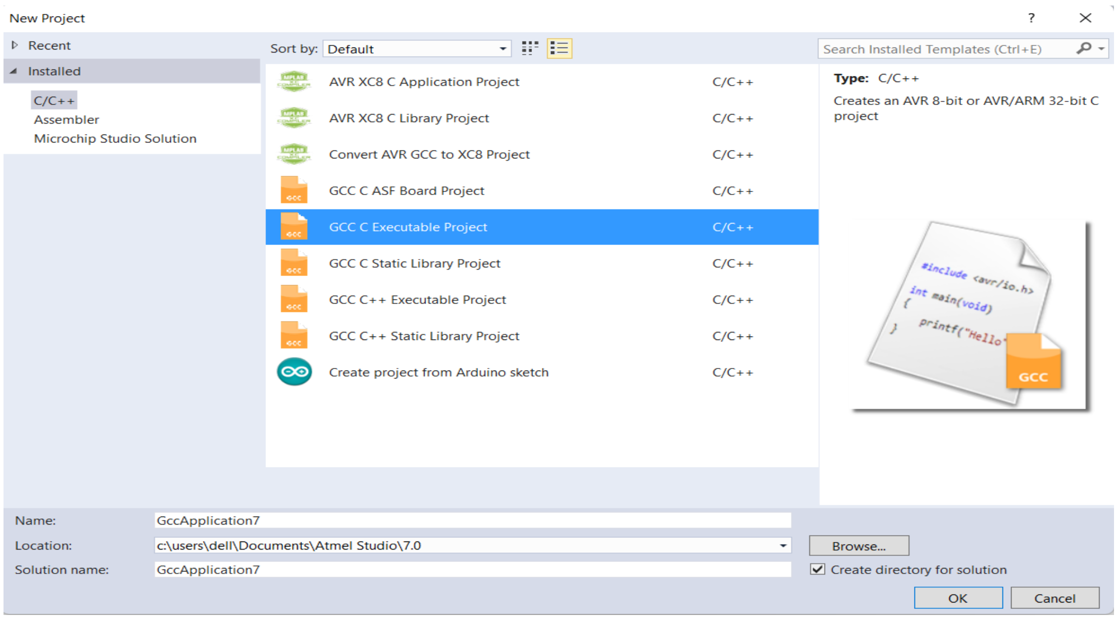
Solution name: GccApplication7

Browse...

☑ Create directory for solution

OK Cancel

4. Select **GCC C Executable Project,** give a project name, solution name, location in which project is to be saved and click OK.


New Project

Sort by: Default

Search Installed Templates (Ctrl+E)

- Recent
- Installed
  - C/C++
  - Assembler
  - Microchip Studio Solution

AVR XC8 C Application Project	C/C++
AVR XC8 C Library Project	C/C++
Convert AVR GCC to XC8 Project	C/C++
GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

Name: GccApplication7

Location: c:\users\dell\Documents\Atmel Studio\7.0

Solution name: GccApplication7

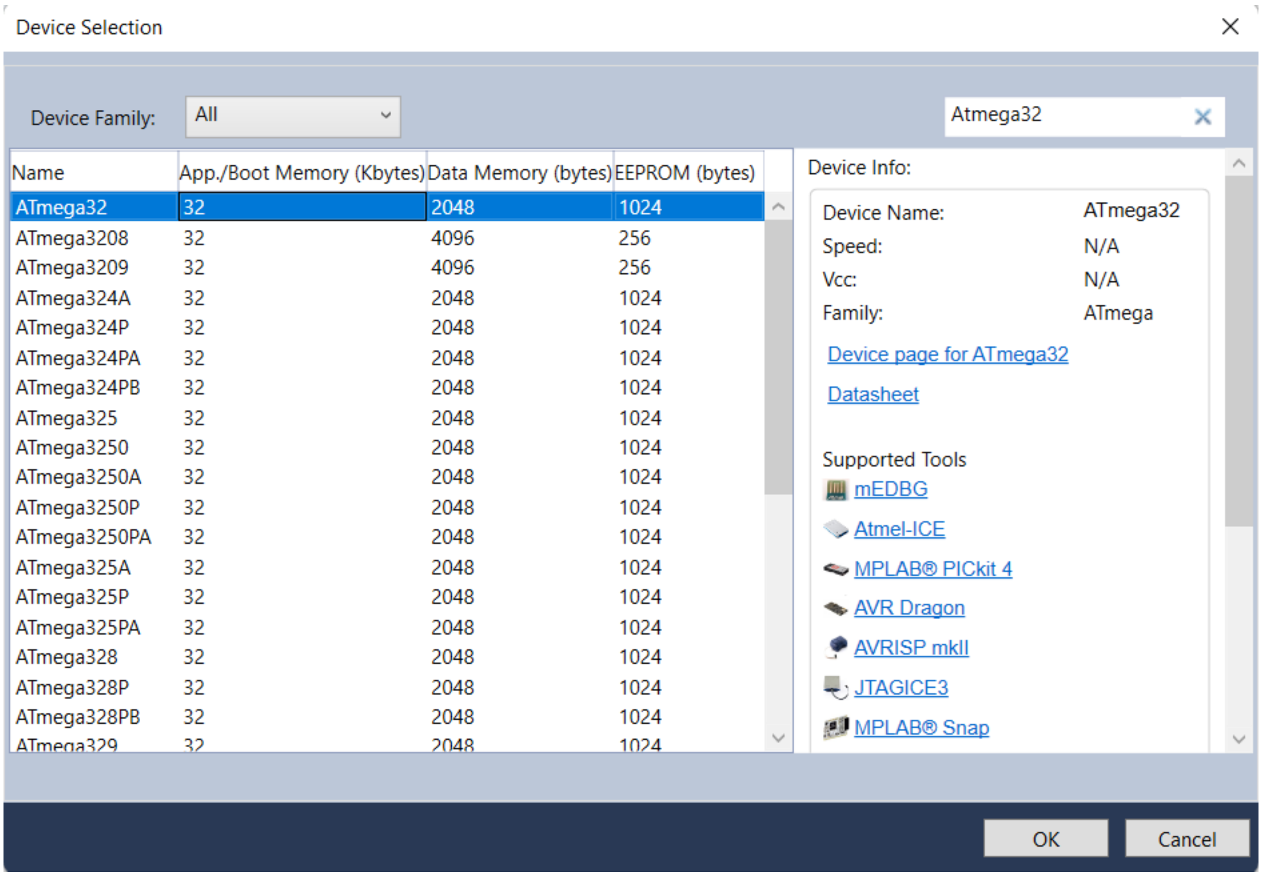
Browse...

☑ Create directory for solution

OK Cancel

5. Selecting Microcontroller

Search ATMEGA32 on the screen and select that microcontroller.


Device Selection

Device Family: All

Atmega32

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes)
ATmega32	32	2048	1024
ATmega3208	32	4096	256
ATmega3209	32	4096	256
ATmega324A	32	2048	1024
ATmega324P	32	2048	1024
ATmega324PA	32	2048	1024
ATmega324PB	32	2048	1024
ATmega325	32	2048	1024
ATmega3250	32	2048	1024
ATmega3250A	32	2048	1024
ATmega3250P	32	2048	1024
ATmega3250PA	32	2048	1024
ATmega325A	32	2048	1024
ATmega325P	32	2048	1024
ATmega325PA	32	2048	1024
ATmega328	32	2048	1024
ATmega328P	32	2048	1024
ATmega328PB	32	2048	1024
ATmega329	32	2048	1024

Device Info:

Device Name:	ATmega32
Speed:	N/A
Vcc:	N/A
Family:	ATmega

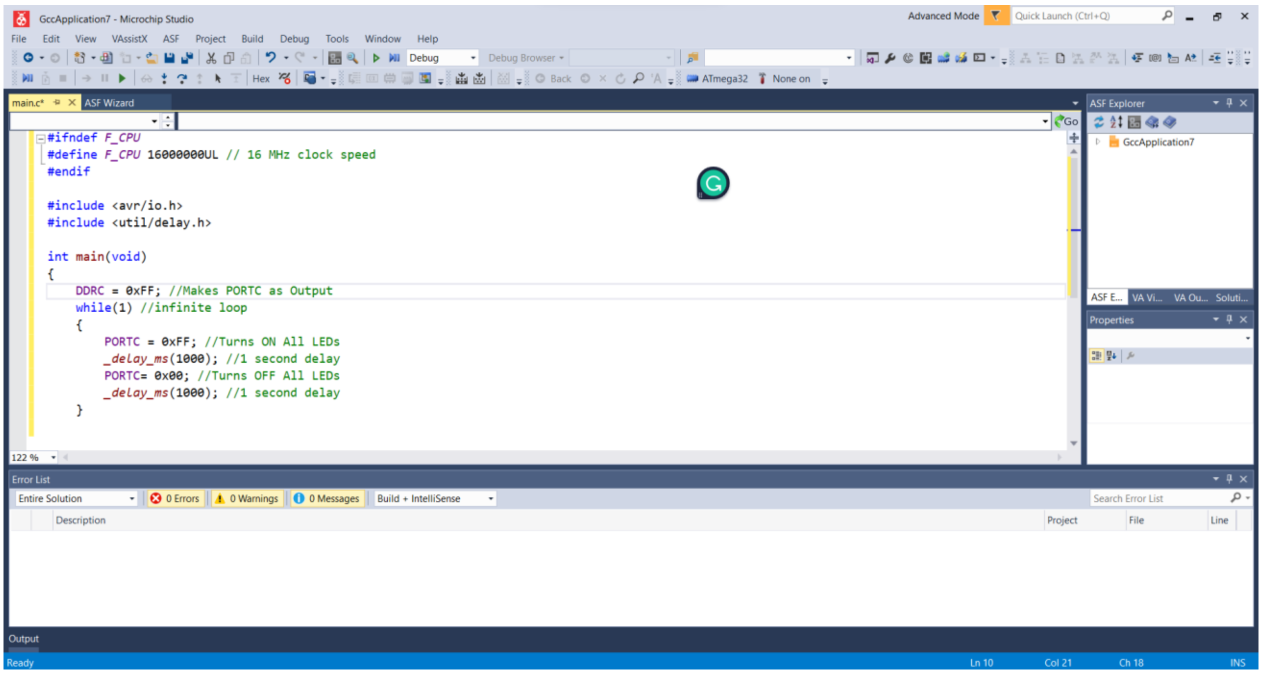
Device page for ATmega32

Datasheet

Supported Tools
- mEDBG
- Atmel-ICE
- MPLAB® PICkit 4
- AVR Dragon
- AVRISP mkII
- JTAGICE3
- MPLAB® Snap

OK Cancel

6. Enter the Program



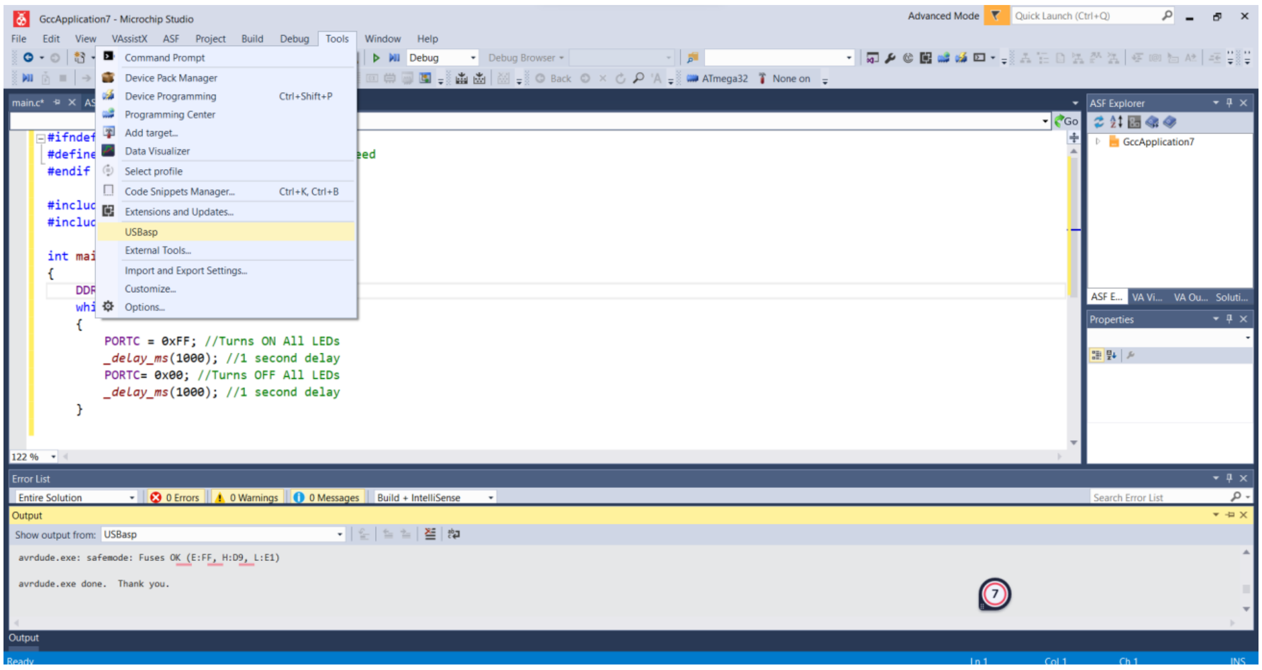
```c
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRC = 0xFF; //Makes PORTC as Output
    while(1) //infinite loop
    {
        PORTC = 0xFF; //Turns ON All LEDs
        _delay_ms(1000); //1 second delay
        PORTC= 0x00; //Turns OFF All LEDs
        _delay_ms(1000); //1 second delay
    }
```

7. Then click on **Build** >> **Build Solution** or **Press F7** to generate the hex file.

8. To load the hex file on the development board, use the external tool USBasp created in Experiment 1.



```
PORTC = 0xFF; //Turns ON All LEDs
_delay_ms(1000); //1 second delay
PORTC= 0x00; //Turns OFF All LEDs
_delay_ms(1000); //1 second delay
}
```

```
avrdude.exe: safemode: Fuses OK (E:FF, H:D9, L:E1)

avrdude.exe done.  Thank you.
```

C Code:

```
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
  DDRB = 0xFF; //Nakes PORTC as Output
  while(1) //infinite loop
  {
    PORTB = 0xFF; //Turns ON All LEDs
    _delay_ms(1000); //1 second delay
    PORTB= 0x00; //Turns OFF All LEDs
    _delay_ms(1000); //1 second delay
  }
}
```

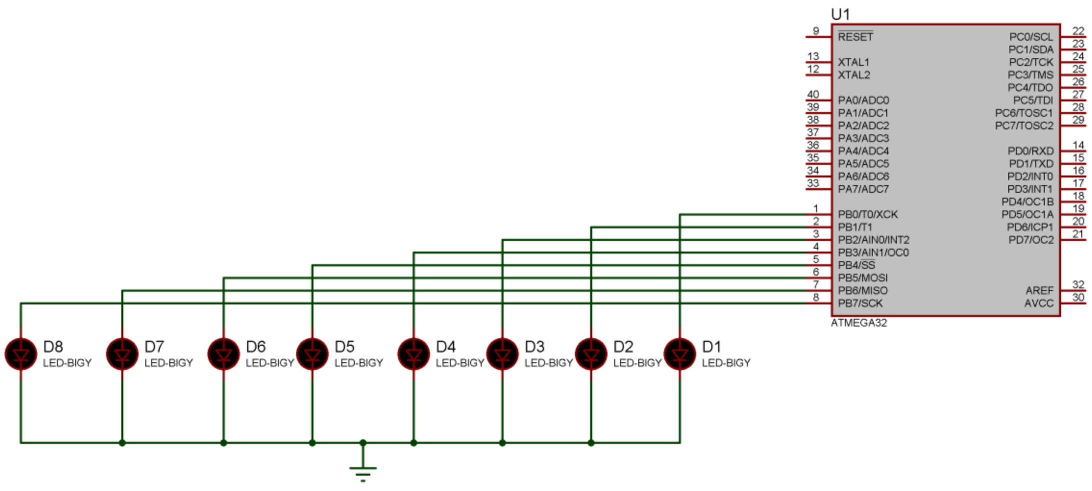
- DDRC = 0xFF makes all pins on PORTC as output pins
- PORTC = 0xFF makes all pins on PORTC Logic High (5V)
- PORTC = 0x00 makes all pins on PORTC Logic Low (0V)
- _delay_ms(1000) provides 1000 milliseconds delay.
- while(1) makes an infinite loop

You have seen that PORT registers are used to write data to ports. Similarly to read data from ports PIN registers are used. It stand for Port Input Register. eg : PIND, PINB

NOTE:

You may like to set or reset individual pins of PORT or DDR registers or to know the status of a specific bit of PIN register. There registers are not bit addressable, so we can't do it directly but we can do it through program. To make 3ed bit (PC2) of DDRC register low we can use *DDRC &= ~(1<<PC2).* (1<<PC2) generates the binary number 00000100, which is complemented 11111011 and ANDed with DDRC register, which makes the 3ed bit 0. Similarly *DDRC |= (1<<PC2)* can be used set the 3ed bit (PC2) of DDRC register and to read 3ed bit (PC2) we can use *PINC & (1<<PC2).* Similarly we can set or reset each bit of DDR or PORT registers and able to know the logic state of a particular bit of PIN register.

OUTPUT:



CONCLUSION:

By Performing this Experiment I get to know the various aspects of GPIO in AVR Boards.

Experiment-2 Post Lab Exercise

Student Name: Aryan Langhanoja
Enrollment No: 92200133030

Answer the following questions:

1) Find the contents of PORTC after execution of the following code:

```
PORTC = 0;
PORTC = PORTC | 0x99; PORTC = ~PORTC;
```

- PORTC after the Execution will be 0x66

2) Find the contents of PORTC after execution of the following code:

```
PORTC = ~ (0<<3);
```

- PORTC after the Execution will be 0xFF

3) To configure the output at pin 3 of PORTB, which register should I use?

- DDRB

4) When the PORTx register has 1 value, what does it mean?

- When the PORTx register has a value of 1, it means that the corresponding pins of the microcontroller's GPIO (General Purpose Input/Output) port are set to a logic HIGH level. In other words, the pins configured as outputs are driven HIGH.

5) What is the difference between PORTC = 0x00 and DDRC = 0x00?

- PORTC = 0x00 means that Each Pin pf PORTC has LOW and DDRC = 0x00 means that each pin of Port C will be the input pin

6) At port A, one push button has been connected and at port D, one LED has been connected, now I am writing a program in such a way that when the push button is pressed, the LED is on and when the push button is not pressed, led is off. For this, which GPIO registers am I going to use?

- PINA and PORTD as well as we have to set DDRA and DDRC Accordingly.