EE2016 - Microprocessor Theory + Lab

Experiment-5: AVR I/O Port Configuration and Interfacing

1 Introduction

This experiment involves an AVR Atmega8 microcontroller on a breadboard. The microcontroller (on the breadboard) is connected to a USBASP board as shown in Figure 1 and, in turn, this is connected to the personal computer (desktop). The USBASP board itself includes a microcontroller and this provides an elegant arrangement to program the Atmega8 on the breadboard. The exact connections between the USBASP and the microcontroller (on the breadboard) are shown in Figure 2. While doing the experiment, make these connections as per the figure.

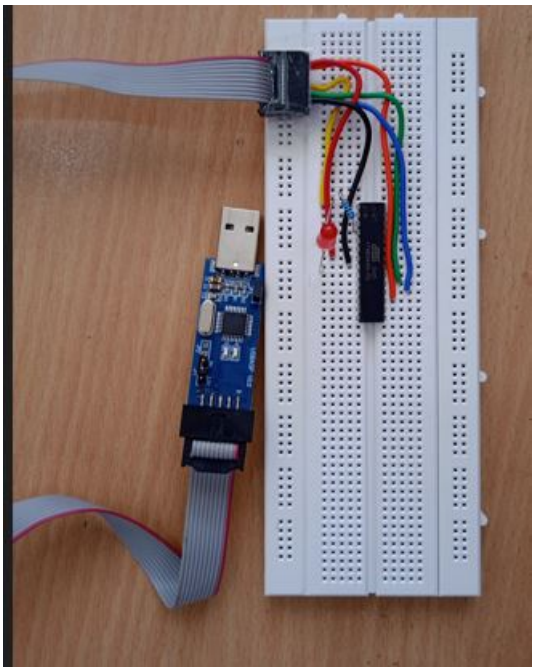

Figure 1: Snapshot of connection between USBASP board and the Atmega8 on the breadboard

Note: It may NOT be safe to try this experiment on your laptop.

2 Configuration of Ports

The architecture of AVR microcontrollers is register-based: information in the microcontroller such as the program memory, state of input pins and state of output pins is stored in registers. There are a total of 32 8-bit registers. Atmega8 has 23 I/O pins. These pins are grouped under what are known as ports. A port can be visualized as a bidirectional buffer with a specific address between the CPU and the external world. The CPU uses these ports to read input from and write output to them. The Atmega8 microcontroller has 3 ports: PortB, PortC, and PortD. Each of these ports is associated with 3 registers - DDRx, PORTx and PINx which set, respectively, the direction, output and input functionality of the ports. Each bit in these registers configures a pin of a port. Bit0 of a register is mapped to Pin0 of a particular port, Bit1 to Pin1, Bit2 to Pin2 and so on. The DDRx and PORTx registers are explained below since these are the ones we will use in this experiment (we will only deal with output of data here). PINx register would be of interest when we read in information.

2.1 Register DDRx

DDR stands for Data Direction Register and 'x' indicates a port alphabet. As the name suggests, this register is used to set the direction of port pins to either input or output. For input, we set to 0 while for output, we set to 1. For instance, let us consider PortB. To set this port as input or output, we need to initialize DDRB. Each bit

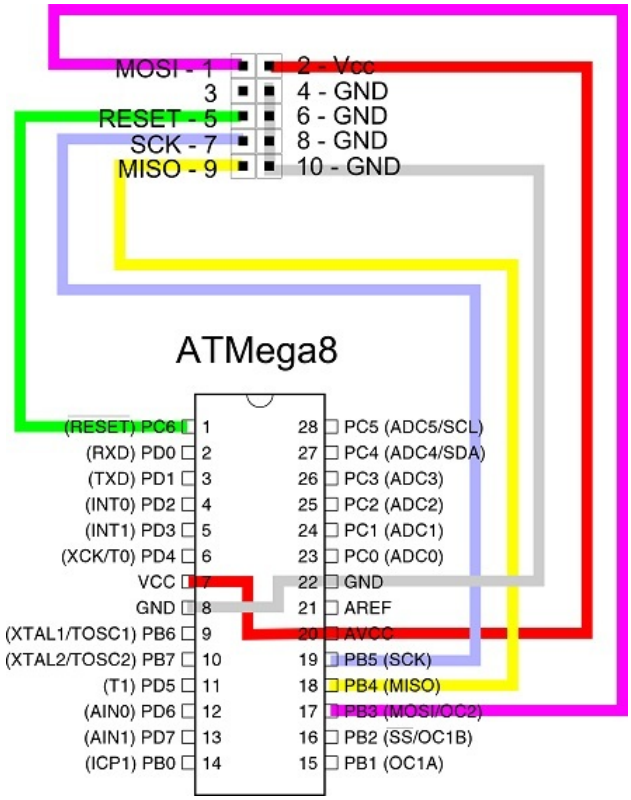

Figure 2: ATmega8 (on breadboard) to USBASP board Connections

in DDRB corresponds to the respective pin in PortB. Suppose we write DDRB = 0xFF, then all bits in PortB are configured to 'Output'. Similarly, DDRB=0x00 configures the port to be 'Input'.

2.2 Register PORTx

PORTx sets a value to the corresponding pin. DDRx can set a bit to be either input or output while PORTx changes its functionality based on it. For example, PORTC = 0x01 will enable (or light up) an LED connected to this port after the appropriate DDRC setting.

Note: In this experiment, we will primarily use LEDs for showing the output of various tasks. LEDs can, in principle, be connected to different ports. For example, if you use PORTD, then all 8 pins are available to connect an LED (via a series resistor). If you use PORTC, pins PC0 to PC5 are available.

3 Programming Environment

This experiment involves various tasks by the microcontroller. All of this will be accomplished via C programs. A template for C programs you will write will look like the one shown next.

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

int main(void)
{

    while (1)
    {



    }
}
```

The include file "io.h" is required to "recognize" DDRC, PORTC within the C program. The "delay.h" file is required when you want to include commands like *_delay_ms(xxx)* to incorporate delays in your code. The *while (1)* acts as an infinite loop and executes until a break is issued explicitly. To perform a 'build' on the C programs, you need to use the Microchip studio software. As with most other packages, you need to open a new project. The remaining steps are as follows.

- Choose *GCC C Executable project* and specify the folder to save.
- In the device list, search for *ATmega8*.
- Write the program and save as *.c*.
- Go to *Build* and then *Build Solution* to generate a .hex file.

Once Build is completed, the *hex* file needs to be downloaded to the flash memory on Atmega8. This will require use of a software tool called *Burn-o-Mat*.

4 Tasks to be performed

Task 1: Make the connections shown in Figure 2.

Task 2: Download the USBASP driver from https://www.fischl.de/usbasp or https://zadig.akeo.ie. Also download the AVR Burn-o-Mat software using http://avr8-burn-o-mat.aaabbb.de/ and the Java runtime environment from http://java.sun.com/javase/downloads/index.jsp. Then download the WinAVR software from https://sourceforge.net/projects/winavr (WinAVR includes avr-gcc compiler, avrdude programmer etc. and some of these are required to successfully write to the flash). Configure the AVR Burn-o-Mat software (using the **Settings** option inside the window that shows up) so that (i) the port is set to USB and (ii) the programmer is set to USBASP.

Task 3: Connect an LED along with a series resistor (about 330 ohms) to the Port D pin 0 (PD0) of ATmega8. The other end of this "wire" will be connected to **GND**. Write a C program to turn on this LED permanently. Once the program is written, use Microchip studio to compile and create a .hex file. You may ignore any warnings in the Microchip studio window. Then load this file by entering the full path in the box labelled *Flash* in the AVR8 Burn-o-Mat window and then click on *Write*.

Task 4: Modify the program for task 3 to cause the LED to blink with about 1 second gap.

Task 5: Now connect one more LED to port D (via a 300Ω resistor) and write a C program to make the two LEDs blink alternately.

Task 6: Next, modify your C program to make the two LEDs produce the following sequence: 00, 01, 10, 11, 00, ...

Task 7: Hard code a pair of 2-bit numbers in your program and perform addition of the numbers. Show the results of the addition on three LEDs. Note that you need to include a 300Ω resistor too while connecting the third LED to your breadboard.

Task 8: Setup the three LEDs to implement a 3-bit Johnson counter.

Task 9: In the tasks above, you are asked to use Port D for connecting the LED(s). Now we will do tasks 7, 8 and 9 using Port C pins. Go through the Atmega8 pin diagram and identify what pins on Port C can be used. Then connect one or more LEDs suitably to Port C (pins) and achieve the same objective. In this case, you will also need to suitably set the data direction register for port C and then make appropriate assignments to PORTC.

Task 10: Hard code two 4-bit numbers in your program. Perform addition of the two 4-bit numbers and show the results on five LEDs connected to Port C. Note that you need to connect 330Ω resistors in series with each LED.

Task 11: Submit a report on the experiment on Moodle (within a week of this experiment). One report per group (with the names of the group members) is sufficient. The report should contain details of the solution (including the code for each task) and your observations (in programming, debugging etc.). Please note that reports that closely match those of other groups will be penalized.