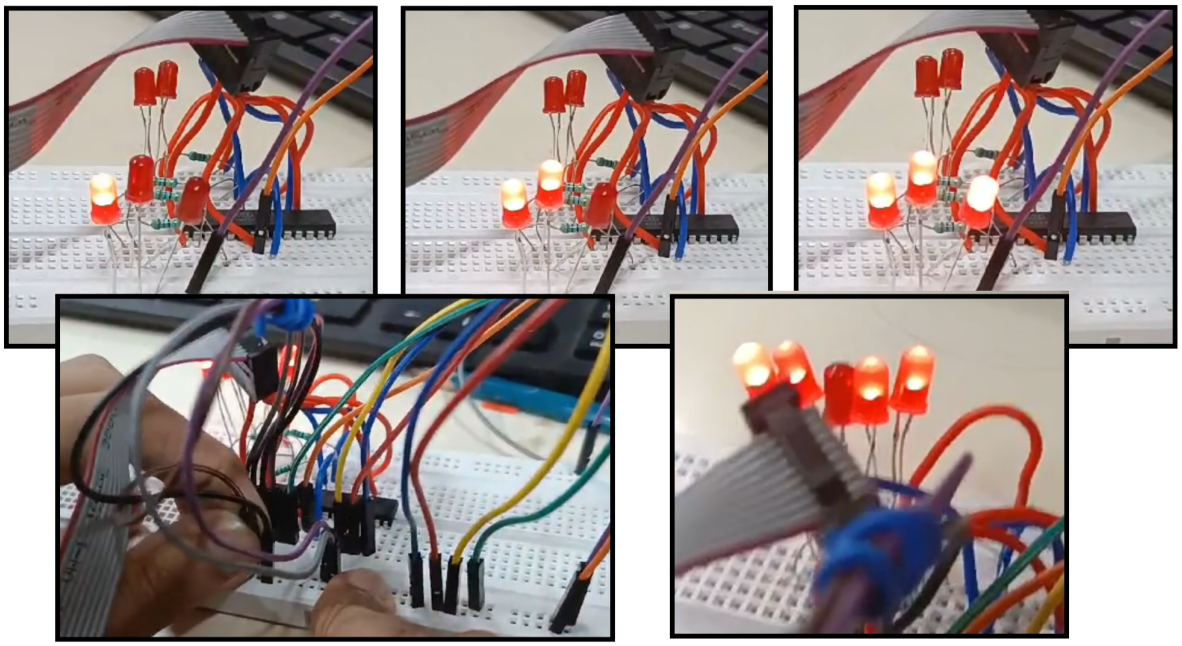

EE2016: Assignment-5

By Siva(EE23B151),
Bhavesh(EE23B017)
and Naveen(EE23B113)

Group 28

Date: 23/9/2024

Indian Institute of Technology Madras

Contents

1 Introduction

This experiment involves programming an AVR micro-controller using C program to do tasks whose outputs are shown via LEDs. The AVR chip is placed on a bread-board and connected to a USBASP board which helps in giving power to the Atmega chip as well as uploading programs from computer to the chip.

1.1 USBASP board

The USBasp is a popular USB-based programmer for AVR microcontrollers. It allows users to program these chips directly via USB, eliminating the need for traditional serial or parallel port programmers. Designed for ease of use, it's widely adopted by hobbyists and professionals alike for embedded system development.

Figure 1: USBASP board

1.2 Key Features

The key features of the USBasp are as follows:

- **In-System Programming (ISP):** USBasp uses the ISP protocol, allowing the microcontroller to be programmed while embedded in a circuit.
- **USB Connectivity:** The programmer connects to the computer via USB and supports full-speed USB communication. It is compatible with Windows, Linux, and macOS through the `libusb` driver.
- **Supported Microcontrollers:** USBasp supports a wide range of AVR microcontrollers, including ATmega8, ATmega16, ATmega32, ATmega328, and many more.
- **Open Source Design:** Both the hardware and firmware are open source, allowing users to modify and customize the design.
- **Power Supply Options:** USBasp can power the target microcontroller with 5V or 3.3V, or the microcontroller can be powered externally.
- **Simple Driver Installation:** After installing the `libusb` driver, the USBasp integrates smoothly with programming tools like `AVRDUDE`.
- **Programming Software Compatibility:** USBasp works with `AVRDUDE`, as well as GUI environments such as `Arduino IDE` and `Atmel Studio`.

2 Tasks

2.1 Task 1: Connecting micro-controller and USBASP board

Connection Table

Below is a table showing how to connect the USBasp ISP header to the ATmega8 micro-controller:

ISP Pin	ATmega8 Pin Number	USBASP Pin Number
MOSI	Pin 17	Pin 1
MISO	Pin 18	Pin 9
SCK	Pin 19	Pin 7
RESET	Pin 1	Pin 5
VCC	Pin 7	Pin 2
GND	Pin 8	Pin 10

Table 1: Connections between USBasp programmer and ATmega8 microcontroller.

It is recommended to connect both GND pins on an ATmega8 (Pins 8 and 22) to reduce power supply variation and avoid power supply noise.

2.2 Task 2: Setting up USBASP, AVR Burn-o-Mat, Java Runtime, and WinAVR for AVR programming

The above software were already set up in the lab computer (downloaded from required links given in the task file). Throughout the experiment, we were required only to add the **hex file** (which we compiled using Microchip studio installed in our laptop) to AVR Burn-o-mat and write to it.

2.3 Task 3: Turn ON LED permanently using C program

Code description

- `DDRD |= (1 « DDD0)`: The operation (1 « DDD0) shifts the binary 1 to the left by DDD0 positions (which is 0 for PD0). The |= operator sets the corresponding bit in DDRD to 1, making PD0 an output pin. This configures the pin to send signals (control an LED, in this case).
- `PORTD |= (1 « PD0)`: Sets PD0 high, turning the LED on.

Hardware Connections

- Connect the **anode** of the LED to **PD0** (Pin 2 in Atmega8).
- Connect the **cathode** of the LED to one end of a 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

C Codes, Hex files and Outputs

C code: The C file for the task.
Hex file: The hex file after building the solution for the above C file using Microchip Studio.

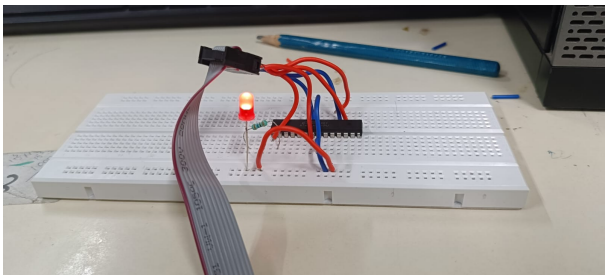

Figure 2: LED turned ON permanently

2.4 Task 4: Making the LED blink with 1 sec gap

Code description

Similarly as before, we set Port D(pin 0) PD0 to high that initially lights up the LED.

- After that, we run an infinite loop by using `while(1)` to blink the LED.
- Set PD0 to high and then wait for 1000 ms(1 sec) by using `_delay_ms(1000)`
- After 1 sec, to turn off the LED , set PD0 to low using bitwise and with negation of `(1 « PD0)`.
- Then wait for another 1 sec using `_delay_ms(1000)` and this way the LED blinks with 1 second gap.

Hardware Connections

The hardware connections are same as previous task (LED anode to PD0, cathode to one end of resistor and the other end to GND).

C Codes and Hex files

C code: The C file for the task.
Hex file: The hex file after building the solution for the above C file using Microchip studio.

2.5 Task 5: Make two LEDs blink alternatively

Code description

- Firstly set PD0 and PD1 as output pins using `DDRD |= (1 « DDD0)`and `DDRD |= (1 « DDD1)`.
- Next, run an infinite loop. Set PD0 high using `PORTD |= (1 « PD0)` and simultaneously set PD1 low using `PORTD &= ~(1 « PD1)`. Then make LED1 turned on for 1 second using `_delay_ms(1000)`.

- Now, after 1 sec set PD1 high and simultaneously set PD0 low. Then make LED2 turned on for 1 second and the process continues. In this way, the two LEDs blink alternatively.

Hardware Connections

The connections are as follows:
For LED1 :

- Connect the **anode** of the LED1 to **PD0** (Pin 2 in Atmega8).
- Connect the **cathode** of the LED1 to one end of a 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

For LED2 :

- Connect the **anode** of the LED2 to **PD1** (Pin 3 in Atmega8).
- Connect the **cathode** of the LED2 to one end of another 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

C Codes and Hex files

C code: The C file for the task.
Hex file: The hex file after building the solution for the above C file using Microchip studio.

2.6 Task 6: Making the LED blink with a given pattern

Pattern : We were to implement the pattern 00, 01, 10, 11 using two LEDs.

Code description

- Firstly set PD0 and PD1 as output pins using `DDRD |= (1 « DDD0)`and `DDRD |= (1 « DDD1)`.
- Next, run an infinite loop. Set both PD0 and PD1 low using `PORTD &= ~(1 « PD0)` and `PORTD &= ~(1 « PD1)`. Then wait for 1 second using `_delay_ms(1000)`. Set PD0 high using `PORTD |= (1 « PD0)` and simultaneously set PD1 low using `PORTD &= ~(1 « PD0)`. Then make LED1 turned on for 1 second using `_delay_ms(1000)`.
- Now, after 1 sec set PD1 high and simultaneously set PD0 low. Then make LED2 turned on for 1 second.
- Finally set both PD0 and PD1 high and wait for 1 second. Then the loop continues again, making PD0 and PD1 low and so on. This way the pattern is executed.

Hardware connections

The connections are exactly the same as the previous task.

2.6.1 C Code, Hex file and Output

C code: The C file for the task.
Hex file: The hex file after building the solution for the above C file using Microchip studio.
Video link: output for this task is shown in this video.

2.7 Task 7: Showing output for addition of two 2-bit numbers using three LEDs

Code Description

- Firstly, set PD0, PD1 and PD2 as our output pins using `DDRD |= (1 « DDD0)`,`DDRD |= (1 « DDD1)` and `DDRD |= (1 « DDD2)`
- Then define the 2 bit numbers that are added up. In our case, the two numbers were N1 = 10 and N2 = 11 (in binary).
- The sum of N1 and N2 were stored in the variable Sum.
- Store the LSB of sum in variable S1, MSB in S2, and carry bit in C using bitwise AND of sum & 0b00000001, sum & 0b00000010, and sum & 0b00000100.
- Right shift S2 by 1 position using S2 = S2 » 1 and right shift C by 2 position using C = C » 1, this makes both S2 and C a one bit number
- Now, we run a loop for each S1, S2 and C. When S1 is high, the LED connected to PD0 is turned on and turned off when zero, when S2 is high, the LED connected to PD1 is turned on and turned off when zero, when C is high the LED connected to PD2 is turned on and turned off when zero.

Hardware connections

For LED1 :

- Connect the **anode** of the LED1 to **PD0** (Pin 2 in Atmega8).
- Connect the **cathode** of the LED1 to one end of a 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

For LED2 :

- Connect the **anode** of the LED2 to **PD1** (Pin 3 in Atmega8).
- Connect the **cathode** of the LED2 to one end of another 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

For LED3 :

- Connect the **anode** of the LED3 to **PD2** (Pin 4 in Atmega8).
- Connect the **cathode** of the LED3 to one end of another 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

C Code

C code: The C file for the task.

2.8 Task 8: 3-bit Johnson counter using three LEDS

Pattern for Johnson Counter : 000, 100, 110, 111, 011, 001, 000, ...

Code Description

- Firstly, set PD0, PD1 and PD2 as our output pins using `DDRD |= (1 « DDD0)`,`DDRD |= (1 « DDD1)` and `DDRD |= (1 « DDD2)`
- Now, run an infinite loop. Firstly, set all output pins to low. Then delay for 1 second.
- Set PD2 to high and rest as it is and delay for 1 second.
- Set PD1 to high and delay for 1 second.
- Set PD0 to high and delay for 1 second.
- Set PD2 to low and rest as it is and delay for 1 second.
- Set PD1 to low and delay for 1 second.
- Set PD0 to low and delay for 1 second, and the loop continues again thus generating the pattern.

Hardware Connections

The connections are exactly the same as previous task.

C Code and Output

C code: The C file for the task.
Video link: output for this task is shown in this video.

2.9 Task 9: Using PORTC instead of PORTD for tasks 7 and 8

Code Description

In the codes of the above tasks, the following changes were made :

- DDRD → DDRC
- DDD0 → DDC0, DDD1 → DDC1, DDD2 → DDC2
- PORTD → PORTC
- PD0 → PC0, PD1 → PC1, PD2 → PC2.

Hardware Connections

The connections are exactly the same, except that the output pins are now changed.

- **For LED1 : PD0** changed to PC0 (Pin 23 of Atmega8)
- **For LED2 : PD1** changed to PC1 (Pin 24 of Atmega8)
- **For LED3 : PD2** changed to PC2 (Pin 25 of Atmega8)

C Code

C code: The C file for the task.

2.10 Task 10: Showing output for addition of two 4-bit numbers using three LEDs

Code Description

- Set PORTD pins as output pins (PD0 to PD3) using `DDRD |= (1 « DDD0)`, `DDRD |= (1 « DDD1)` and so on.
- Then define the 4 bit numbers that are to be added up. In our case, the two numbers were N1 = 1010 and N2 = 1101.
- Store their sum in the Sum variable.
- Extract each bit of the Sum variable. Store LSB in S1 obtained by bitwise AND of Sum & 0b00000001, store the next bit in S2 by bitwise AND of Sum & 0b00000010 and right shift it by 1, the next bit in S3 (Sum & 0b00000100) and right shift by 2, the next bit in S4 (Sum & 0b00001000) and right shift by 3, the carry bit in C (Sum & 0b00010000) and right shift by 4.
- Then run a loop for LEDs to glow. Each bit of the sum (S1, S2, S3, S4, and C) is displayed on an LED connected to PORTD:
  - If the bit is 1, the corresponding LED is turned on by setting the pin to HIGH.
  - If the bit is 0, the corresponding LED is turned off by setting the pin to LOW.

Hardware connections

For LED1 :

- Connect the **anode** of the LED1 to **PD0** (Pin 2 in Atmega8).
- Connect the **cathode** of the LED1 to one end of a 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

For LED2 :

- Connect the **anode** of the LED2 to **PD1** (Pin 3 in Atmega8).
- Connect the **cathode** of the LED2 to one end of another 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

For LED3 :

- Connect the **anode** of the LED3 to **PD2** (Pin 4 in Atmega8).
- Connect the **cathode** of the LED3 to one end of another 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

For LED4 :

- Connect the **anode** of the LED4 to **PD3** (Pin 5 in Atmega8).
- Connect the **cathode** of the LED4 to one end of another 300 ohm **resistor**
- Connect the other end of the resistor to **GND**.

C Code

C code: The C file for the task.

2.11 BONUS part of Task 10

In addition to actual task, instead of hard coding the 4-bit input numbers in the program and run it only for one set of values, we gave our inputs directly to the ports of Atmega8 in the breadboard and could change one set of values to another.

Code description :

The code is slightly modified from the previous. The changes are given below:

- Set PORTD pins (PD0 to PD7) as input pins and PORTC pins (PC0 to PC3) as output pins.
- The input is declared to PIND, the program reads the 8-bit input from PORTD. PIND holds the current state of the pins on PORTD.
- The first four bits from right is stored in N1, the remaining bits are shifted to the right by four units and stored in N2.

  The other parts of code remains the same.

Hardware Connections

- Give your inputs to **PD0** till **PD7** (**PD0** to **PD3** is the first 4 bit number, **PD4** to **PD7** is the next 4 bit number
- **LED 1** : anode to **PC0** and cathode to **GND**
- **LED 2** : anode to **PC1** and cathode to **GND**
- **LED 3** : anode to **PC2** and cathode to **GND**
- **LED 4** : anode to **PC3** and cathode to **GND**

C Code, Video link and Outputs

C code: The C file for the task.
Video link: output for this task is shown in this video.

Figure 3: $(1101)_2 + (1110)_2 = (11011)_2$

Figure 4: $(0100)_2 + (0000)_2 = (00100)_2$