Department of Computer Engineering
Faculty of Engineering, University of Peradeniya

CO321 Embedded Systems - 2023

Lab 1 - Programming AVR Microcontrollers

Programming ATmega328p in *C* language

Arduino IDE provides an abstract way to program the microcontroller easily and efficiently. In this lab we manually do what is happening inside the Arduino IDE, which is hidden to the user.

In our particular case, we use the **avr-gcc** compiler and the **avrdude** uploading tool. We can use these tools to write a program in *C* language, *instead of the Arduino language*, build/compile, and upload that program to the AVR. We build our *C* program by calling the **avr-gcc** command and upload it using **avrdude** command.

avr-gcc toolchain doesn't know the Arduino Uno board layout, it only knows the microcontroller that is mounted on it, which is the **ATmega328p**. So, we need to use the *hardware I/O registers* and *pins* of that particular chip when writing our program. However, we have to use the pins on the Arduino board for connecting to external components (because the chip is already soldered to the board). Figure 1 shows the mapping between the pins of the microcontroller and the pins of the Arduino board.

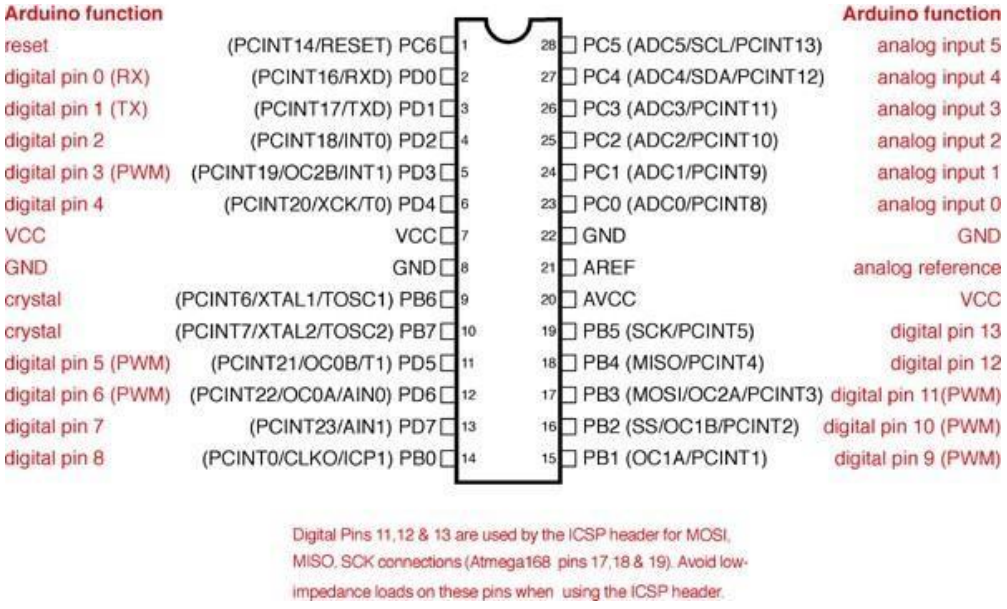

Figure 1 : Pin mapping of ATmega328P to Arduino Uno

VCC is the pin which provides power to the microcontroller and *GND* pins are the grounds. There are three ports namely *Port B (PB), Port C (PC)* and *Port D (PD)*. *PB* and *PD* are 8-bit ports and *PC* is a 7-bit port. The ports are bi-directional, and they can be used for both input

and output. Some pins have multiple functionalities as well where the alternate functionalities are shown in brackets. For example, *PC6* which is the 7th bit of *Port C* can be used as RESET input, as shown in figure 1. These different pins in the microcontroller are connected appropriately inside the Arduino and the ports are exposed to the outside by different names. For example, the *PB5* pin of the ATmega328p is exposed by the name "digital pin 13" on the Arduino.

Blinking LED Example:

Let's start with an example: An LED blinking on Arduino digital pin 13.

Digital pin 13 is mapped to the *PB5* pin of the ATmega328p chip and hence the I/O register for that pin (*PORTB*) must be programmed.

- Create a folder for your project and write the following example program in a file called "**led.c**".

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

#define BLINK_DELAY_MS

1000int main (void){

DDRB = DDRB|(1<<5);       /* configure pin 5 of PORTB for output*/

while(1){

PORTB = PORTB | (1<<5);       /* set pin 5 high to turn led on */

_delay_ms(BLINK_DELAY_MS);

PORTB = PORTB &~(1<<5);           /* set pin 5 low to turn led off */
_delay_ms(BLINK_DELAY_MS);

          }
}
```

DDRB is the **Data Direction Register** for *Port B*. Since we are using this as an output, bit 5 of this register must be set to `1`. First we read the current value in *DDRB* and then apply bitwise or with `00100000` and write back to *DDRB*. This preserves the state of other bits and change only the 5th bit.

Then inside the loop we turn on the LED and turn off while keeping enough delays in-between. The *_delay_ms*function delays for the number of milliseconds provided as an argument. `PORTB = PORTB | (1<<5)` sets the 5th bit of *Port B* register to `1`, and `PORTB = PORTB &~(1<<5)` sets the same bit to `0` while preserving the values which were there for other bits. If you don't want to preserve the other bits, you can just say `PORTB=0xFF` to turn "on" all bits in *Port B*, or say `PORTB=0x00` to turn "off" all bits
.

Now it's time to run the commands that build and upload the LED blink program.

The commands that we used to compile and upload the file "led.c" are:

```
$ avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -o led led.c
$ avr-objcopy -O ihex -R .eeprom led led.hex
$ avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U
flash:w:led.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be
performedTo disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "led.hex"
avrdude: input file led.hex auto detected as Intel Hex
avrdude: writing flash (88 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 88 bytes of flash written
avrdude: safemode: Fuses OK
avrdude done.  Thank you.
```

The **avr-gcc** compiler creates an executable binary program that contains machine code. We need to create a HEX file and then write it to the Program Memory inside the microcontroller. Binary is converted to HEX using **avr-objcopy**. The Arduino board provides an interface to the microcontroller as a virtual Serial port and we are uploading the hex file to the microcontroller via this. Uploading is done by the command **avrdude.**

The explanation of the commands are as follows.

- The first command takes the C source file and compiles it into a binary file. The options tell the compiler to optimize for code size, what the clock frequency is (16 MHz for Arduino Uno) and which particular microcontroller we're using.
- The second command converts the binary program into a HEX file format.
- The fourth command uploads the HEX data into the ATMEGA chip embedded flash, and the options tells avrdude tool to communicate using the Arduino serial protocol, through a particular serial port and to use 115200bps as the data rate.

After this is done, you should see the LED blinking.

Lab Exercises:

- Connect four (4) LEDs to *Port B* and blink all of them simultaneously.
- Using four (4) LEDs, implement a "Knight Rider" style display.
  If the four LEDs are *A, B, C* and *D* then the turn order of lights should be: ABCDCBABC……

Submit a .zip file named "groupXX.zip" containing all your code files and a screenshot/photo of the circuit.

Post-lab notes:

Some might want to try out this method at home on your own Arduino device. If you are using Linux Ubuntu, simply install the following three packages via `apt -get install`.

- `gcc-avr`
- `avr-libc`
- `avrdude`