Project 1: Talking to a Microprocessor on a Linux System

Jacob Hollenbeck, Boise State University 02/20/2016

Contents

1 Overview

This paper discusses initial setup and communication to a microprocessor from a debian based operating system. This document covers the required libraries, necessary tools, and hardware configurations needed. Loading a file to the board is demonstrated through the application of blinking and LED from the microprocessor. Issues that were encountered during setup are also discussed.

2 Supplies

- Atmega324p Microprocessor PDIP
- LED
- 220 ohm Resistor
- Atmel-ICE programmer with ISP cable
- Breadboard

3 Software Configuration

3.1 AVR-Toolchain

Embedded programming on Linux is a command line based operation. To use command line, the user must set up a series of programs and libraries. This is referred to as a toolchain. This particular toolchain is composed of a programmer, compiler, library, and debugger (Note that this document does not go over the debugger). The tools used for this are AVR tools. To set up AVR on linux the following packages are used:

- avrdude - program that talks to chip via programmer
- gcc-avr - compiler
- avarice - debugger server
- avr-libc - avr library

To install the necessary libraries, use the following:

```
$ sudo apt-get install avrdude gcc-avr avarice avr-libc
```

3.2 AVRdude

AVRdude is the program used to write files to the MCU via the programmer (atmel-ICE). A full command written to the MCU will look simililar to this:

```
$ avrdude -c <programmer> -p <MCU> -P <port> -U flash:w:blink.elf
```

whereas <programmer> indicates the programmer name, <MCU> the MCU type, and <port> the input port on the computer. The -U parameter will be covered later. First a programmer must be indicated. To view a list of supported programmers, use the following:

```
$ avrdude -c prgm
```

For the atmel-ICE there are several options:

```
  atmelice          = Atmel-ICE (ARM/AVR) in JTAG mode
  atmelice_dw       = Atmel-ICE (ARM/AVR) in debugWIRE mode
  atmelice_isp      = Atmel-ICE (ARM/AVR) in ISP mode
  atmelice_pdi      = Atmel-ICE (ARM/AVR) in PDI mode
```

Because ISP is the communication protocol selected for this document, the option atmelice_isp is selected. To view the list of supported MCUs, use the following command:

```
$ avrdude -c atmelice_isp -p mcu
```

The following is a snippet of the output that will occur:

```
  m32      = ATmega32
  m324p    = ATmega324P
  m324pa   = ATmega324PA
  m325     = ATmega325
```

The Atmega324p is supported so the option m324p is used. The next step is configuring the usb port. Linux operating systems use different names for their serial and usb ports. AVRdude expects the usb port to be named using the term "usb". If the user operating system does this by default then the -P option is not necessary. To test this, issue the following command:

```
$ avrdude -c atmelice_isp -p m324p
```

If the output is:

```
avrdude: jtag3_open_common(): Did not find any device matching VID 0x03eb and PID
    list: 0x2141

avrdude done.  Thank you.
```

then the port is configured by default and the hardware configuration can begin. If the output is:

```
avrdude: jtag3_open_common(): JTAGICE3/EDBG port names must start with "usb"

avrdude done.  Thank you.
```

then the port must be configured by the user. To see how this is done see Section 6.2.

4 Hardware Configuration

4.1 ISP Configuration

Using the JTAG/ISP cable that came with the atmel-ICE, connect one JTAG end to the port that is labeled AVR on the atmel-ICE. Using six jumper wires, connect the respective six ISP wires to their counterparts on the MCU (each pin of the header has a partner of the same name on the MCU). The ISP header layout can be seen in Figure 1.

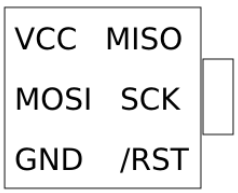

Figure 1: ISP Header

If the user wishes to use JTAG protocol, refer to the atmega324p datasheet for instruction. It is similar to ISP but uses more wires. Due to the small size of the JTAG female connector, a JTAG male connector is often necessary. However, the same procedure used with ISP is used for connecting the programmer and the MCU. The datasheet can be seen at [1]

4.2 MCU Power

The atmega324p requires a power source. The programmer does not supply power. The atmega324p can run on either 3.3 V or 5 V. Using a power supply, set the limits to 3.3 V and 100 mA. This will reduce the chances of burning up the chip. Next apply power to the V_{cc} pin of the MCU and connect the two ground pins to ground. If the programmer is connected to the MCU correctly, a green LED will turn on.

4.3 Establishing a connection

Issue the following command to check the connection from the MCU to the programmer

```
$ avrdude -c atmelice_isp -p m324p
```

If the connection is not configured properly the following readout will occur

```
avrdude: usbdev_open(): WARNING: failed to set configuration 1: could not set config
     1: Device or resource busy

avrdude: stk500v2_command(): command failed
avrdude: initialization failed, rc=-1
         Double check connections and try again, or use -F to override
         this check.


avrdude done.  Thank you.
```

If the device has been configured properly, a readout similar to the following will be produced:

```

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.07s

avrdude: Device signature = 0x1e9508

avrdude: safemode: Fuses OK (E:FF, H:99, L:E2)

avrdude done.  Thank you.
```

Check that the Device signature matches the one provided by the manufacturer. The programmer also has the ability to read the voltage. To do this, append a -v to the above command.

5 Blink an LED

Now that a connection can be established and verified, a file may be written to the MCU flash. The flash memory is reprogrammable and resides in both the Boot section and the Application section of the Atmega324p memory. The flash on this chip has a max size of 32 Kbytes [1].

5.1 Linking Process

Both assembly and C code can be compiled and loaded on to the MCU. This is possible because of the compiler. The compiler converts the given file to an object file, which is composed of machine code. The generated object file is formatted as an Executable and Linkable Format, of a .elf file[4]. This process is repeatable for multiple source files. In that instance, the source files need to compiled and linked into a single .elf file. The commands needed to compile and link C and assembly files are slightly different. Each will be covered in the following sections (Note that linking multiple source files is not covered in this document).

5.2 Blink.s

The file blink.s was written in assembly and provided from the embedded processor course at Boise State University[3]. To blink an LED, the I/O (inputs and outputs) need to be configured. These are configured by programming the DDRx. When a '1' is written to an indiviual bit in the DDRx register, that port pin is enabled as an output. If a '0' is written to the individual bit in the DDRx register, the port pin is enabled as an input. This is used in blink.s with the following commands:

```
1  ldi   r19, (1<<0) ; Set r19 = 0b00000001
2  mov   r18,r19     ; Copy r19 to r18
3  out   4,r19       ; Write r19 to I/O at 0x4
```

DDRB is accessed at its memory location of 0x4. A '1' is written to the first bit, enabling PB0 as an output. Now that the DDRx has been configured, values can be written to the port. This is done by writing a value to the port. This is seen in the following code:

```
1  out   5,r19 ; Write r19 to output at 0x5 (PB0)
```

whereasa r19 is a '1'. 0x5 is the memory location of PORTB. This 1 is xor after a certain delay, creating a blinking effect. To compile blink.s:

```
1 $ avr-as -mmcu=atmega324p --gstabs -o blink.o blink.s
2 $ avr-ld -o blink.elf blink.o
```

The full source code for blink.s can be seen in the code listing section located in the appendix.

5.3 Blink.c

The same concepts from blink.s are used in blink.c. Pointers are created to point to the spot in memory where the DDRx and PORTx are located. The DDRB and PORTB are used. The code is as follows:

```
//Create pointers to the device addresses spaced 8 bits aparts
REGISTER * DDRB = (REGISTER *) DDRB_BASE;      // DDRB_BASE     = 0x24
REGISTER * PORTB = (REGISTER *) PORTB_BASE;    // PORTB_BASE    = 0x25
```

These are located at 0x24 and 0x25 respectively. Note that these are different values than used in blink.s. If disassembled, blink.s will actually be pointed to 0x24 and 0x25. When I/O's are being used as inputs and outputs, 0x20 is added to the actual address. When they are being used to write to memory, their default values are maintained. To compile blink.c:

```
$ avr-gcc -c -mmcu=atmega324p -o blink.o blink.c
$ avr-ld -o blink.elf blink.o
```

The full source code for blink.c can be seen in the code listing section located in the appendix.

5.4 Loading to MCU

To load blink.elf to the MCU, use the following command:

```
$ avrdude -c atmelice_isp -p m324p -U flash:w:blink.elf
```

5.5 Blink Hardware Configuration

Now that the software side has been taken care of, the hardware can be discussed. Because the output was written to Port B0 the voltage will be toggled on pin PB0. This is where the anode of the LED is to be connected. The cathode is to be connected in series with a current limiting resistor to ground. The following figure demonstrates the circuit:

Figure 2: Blink circuit

6 Appendix

6.1 Code listing

Blink.s

```
; $ avr-as -mmcu=atmega324p --gstabs -o blink.o blink.s
; $ avr-ld -o blink.elf blink.o
;
; The following command was used to load the .elf file onto the
; MCU:
;
; $ avrdude -c atmelice -p m324p -U flash:w:blink.elf:e
;
; Commented by Jake Hollenbeck
```

```
;

  .text
  .org 0
  jmp main
  .org 100

main:
  ; First group of instructions sets DDRB at PB0 as an output
  ; by writing a 1 to the first pin

  ldi   r19, (1<<0) ; Set r19 = 0b00000001
  mov   r18,r19   ; Copy r19 to r18
  out   4,r19   ; Write r19 to I/O at 0x4
  rjmp  .     ; Unecessary


  bloop:
    out   5,r19   ; Write r19 to output at 0x5 (PB0)
    rcall delay   ; Call subroutine 'delay'
    eor   r19,r18   ; Exclusive or r19 and r18
    rjmp  bloop   ; Jump to 'bloop'
  ret           ; Return from bloop

delay:
  ldi   r17,15      ; Load 0x15 into r17
delay_1:
  ldi   r31,hi8(40000)  ; Set high byte to 40000
  ldi   r30,lo8(40000)  ; Set low byte to 40000
delay_2:
  sbiw  r30,1     ; Decrement r30 by 1
  brne  delay_2     ; Branch to delay_1 until r30 == 0

  dec   r17       ; Decrement 1 from r17
  brne  delay_1     ; Branch to delay 1 until r17 == 0

  ret           ;ret Return to main

  .end
```

Blink.s

```
/* C version of blink.s. Toggles an LED assigned to PORTB0 of an atmega234p. */

#include <stdio.h>
#include <stdint.h>


//Define the device addresses
#define DDRB_BASE   0x24
#define PORTB_BASE  0x25

```

```
11 //Create a new type 'REGISTER'
12 typedef unsigned int REGISTER;
13
14 int main()
15 {
16   //Create variables for loop
17   unsigned int i,j;
18
19
20   //Create and initialize registers
21   REGISTER r19 = (1 << 0);  //shift a one into r19
22   REGISTER r18 = r19;      //set r18 = r19 = 1
23
24
25   //Create pointers to the device addresses spaced 8 bits aparts
26   REGISTER * DDRB = (REGISTER *) DDRB_BASE;
27   REGISTER * PORTB = (REGISTER *) PORTB_BASE;
28
29   *DDRB = r19;  //Set the value of data at &0x24 = 0b00000001
30
31   //delays
32   while(1){
33     for(i = 15; i>0; i--){
34       for(j = 40000; j>0; j--){
35
36       }
37     }
38   r19 ^= r18;   //toggle r19 with xor
39   *PORTB = r19; //set the value of data at &0x25 = toggledd r19
40   }
41
42   return 0;
43 }
```

6.2 AVR Port Configuration

To associate the programmer with the port name "usb" a config file is created. To do this, the device ID needs to be checked. Issue the following command:

```
1 $ lsusb
```

This will display all active hardware using the usb ports. Look for the atmega name. Remember the device ID. For the atmelice, it should be 03eb:2141. Now the config file will be created. Use the following command:

```
1 $ sudo nano /etc/udev/atmelice.rules
```

Type the following in the file:

```
1   SUBSYSTEM!="usb", ACTION!="add", GOTO="atmelice_end"
2
3   # Atmel Corp. ICE
```

```
ATTR{idVendor}=="03eb", ATTR{idProduct}=="2141", MODE="660", GROUP="dialout"
#
LABEL="atmelice_end"
```

and type CTRL+X and press ENTER to save.

Next we need to link the rules and give group permissions. Use the following commands:

```
$ cd /etc/udev/rules.d
$ sudo ln ../atmelice.rules 60-atmelice.rules
```

check to make sure you have 'dialout' as a group

```
$ groups
```

and finally

```
$ sudo service udev restart
```

This section of the document was based off of Homebuilt Hardware's AVRdragon Linux tutorial [2].

References

[1] *8-bit Atmel Microcontroller with 16K/32K/64K Bytes In-System Programmable Flash*. URL: http://www.atmel.com/images/atmel-8011-8-bit-avr-microcontroller-atmega164p-324p-644p_datasheet.pdf.

[2] *AVRDragon Linux Tutorial*. URL: http://www.homebuilthardware.com/index.php/avr/linux-avrdragon-tutorial-1/.

[3] *Embedded an Portable Computing Systems*. URL: http://ece.boisestate.edu/~arlen/ECE433/Documents/t1.S.

[4] Clarence Planting. *ECE330 MICROPROCESSORS Course Reference*. 2014.