Avr-Microcontrollers-in-Linux-Howto

Revision History

Revision 44	2009-04-20 17:13:07	Revised by: jdd
obfuscate e-mail at the author demand		
Revision 43	2009-03-29 20:49:12	Revised by: ranjeeth
Revision 42	2009-03-29 20:45:09	Revised by: jdd
reverting after publication		
Revision 41	2009-03-29 20:41:59	Revised by: jdd
edit to export docbook		
Revision 40	2009-03-28 21:07:57	Revised by: RickMoen
Adjust style of mailto link to author's original preference.		
Revision 39	2009-03-28 21:05:45	Revised by: RickMoen
Revert e-mail obfuscation, supply some missing punctuation, [re-]fix capitalisation, fix and clarify new run-on sentence.		
Revision 38	2009-03-23 18:21:24	Revised by: ranjeeth
Revision 37	2009-03-23 18:19:51	Revised by: ranjeeth
Revision 36	2009-03-23 17:20:32	Revised by: ranjeeth
Revision 35	2009-03-23 17:19:47	Revised by: ranjeeth
Revision 34	2009-03-23 10:26:00	Revised by: jdd
publishing tests		
Revision 33	2009-03-23 10:25:06	Revised by: jdd
Revision 32	2009-03-23 10:24:24	Revised by: jdd
use of macro to obfuscate e-mail		
Revision 31	2009-03-23 10:11:47	Revised by: jdd
change was only to export docbook without admonitions		
Revision 30	2009-03-23 10:09:15	Revised by: jdd
Revision 29	2009-03-23 10:05:07	Revised by: jdd
Revision 28	2009-03-23 09:58:40	Revised by: jdd
adding the wiki as link		
Revision 27	2009-03-23 09:53:28	Revised by: RickMoen
Insert needed space characters into "pin9", "pin10", "pin25" constructs		
Revision 26	2009-03-23 09:49:05	Revised by: RickMoen
Make markup of all the software items mentioned be consistent		
Revision 25	2009-03-23 09:43:40	Revised by: RickMoen
Remove a couple of stray commas.		

Revision 24	2009-03-23 09:41:30	Revised by: RickMoen
Polishing up a few last bits of punctuation		
Revision 23	2009-03-23 04:11:31	Revised by: ranjeeth
Revision 22	2009-03-23 04:10:34	Revised by: ranjeeth
Revision 21	2009-03-23 04:09:07	Revised by: ranjeeth
Revision 20	2009-03-17 16:20:27	Revised by: ranjeeth
Revision 19	2009-03-17 16:18:33	Revised by: ranjeeth
Revision 18	2009-03-16 19:10:30	Revised by: RickMoen
Fix hyperlink, fix English diction of new sentence.		
Revision 17	2009-03-16 18:50:19	Revised by: ranjeeth
Revision 16	2009-03-16 11:36:49	Revised by: RickMoen
A couple of punctuation nits		
Revision 15	2009-03-16 11:21:14	Revised by: RickMoen
Corrected numerous run-on sentences, punctuation and grammar errors, and questionable idiom.		
Revision 14	2009-03-15 10:27:24	Revised by: BordenRhodes
Cleaned up basic grammar and spelling		
Revision 13	2009-03-15 09:52:41	Revised by: jdd
Revision 12	2009-03-15 09:51:37	Revised by: jdd
add link to the manual of avr-libc		
Revision 11	2009-03-14 21:47:18	Revised by: jdd
end of basic conversion		
Revision 10	2009-03-14 21:40:18	Revised by: jdd
Revision 9	2009-03-14 21:39:26	Revised by: jdd
Revision 8	2009-03-14 21:33:04	Revised by: jdd
add the image		
Revision 7	2009-03-14 21:27:59	Revised by: jdd
Revision 6	2009-03-14 21:26:48	Revised by: jdd
Revision 5	2009-03-14 21:25:13	Revised by: jdd
Firts step - conversion fro html by jdd and basic format edition		
Revision 4	2009-03-14 21:22:37	Revised by: jdd
Revision 3	2009-03-14 21:18:25	Revised by: jdd
Revision 2	2009-03-14 21:13:09	Revised by: jdd
Revision 1	2009-03-14 21:10:33	Revised by: jdd

Table of Contents

1. HOWTO

AVR Microcontrollers in Linux HOWTO,
(ranjeeth_gecmail[at]yahoo[dot]com)

This HOWTO is for readers wishing to program an AVR microcontroller using a GNU/Linux machine. For burning your code, we will be using a parallel port. You may need other electronics components (like a few resistors, capacitors, parallel port connector, etc.), which can be bought from any electronics shop.

---

2. AVR Microcontrollers in Linux HOWTO

2.1. Licence

2.2. What Is a Microcontroller?

A microcontroller is a single-chip computer. It has internal RAM, ROM, timers, counters, interrupt circuitry, I/O ports, analog comparators, serial USARTs, analog to digital converters, watchdog timers, and a RISC architecture. When you are using a microprocessor, you cannot program it alone. You need other components, like RAM, ROM, timers, etc. For programming, you should know its architecture thoroughly: You must read the datasheet for your microcontroller.

2.3. Software Required

binutils: Tools like the assembler, linker, etc.

gcc-avr: The GNU C compiler (cross-compiler for avr).

avr-libc: Package for the AVR C library, containing many utility functions.

uisp: A Micro In-System Programmer for Atmel's AVR MCUs (for burning code to MCUs' memory).

The following Atmel microcontrollers are supported by avr-gcc in Linux:

at90s Type Devices

at90s2313, at90s2323, at90s2333, at90s2343, at90s4414, at90s4433, at90s4434, at90s8515, at90s8515, at90s8515, at90s8535, at90s1200.

atmega Type Devices

atmega103, atmega603, atmega8, atmega48, atmega88, atmega8515, atmega8535, atmega16, atmega161, atmega162, atmega163, atmega165, atmega168, atmega169, atmega32, atmega323, atmega325, atmega3250, atmega64, atmega645, atmega6450, atmega128.

attiny Type Devices

attiny22, attiny26, attiny26, attiny13, attiny13, attiny13, attiny13, attiny2313, attiny11, attiny12, attiny15, attiny28.

Other AVR Devices

avr2, at90c8534, at86rf401, avr3, at43usb320, at43usb355, at76c711, avr4, avr5, at90can128, at94k, avr1.

binutils: Programs to manipulate binary and object files that may have been created for Atmel's AVR architecture. This package is primarily for AVR developers and cross-compilers.

gcc-avr: The GNU C compiler, a fairly portable optimising compiler that supports multiple languages. This package includes C language support.

avr-libc: Standard library used for developing C programs for Atmel AVR microcontrollers. This package contains static libraries, as well as needed header files.

uisp: Utility to program AVR chips with object code created by gcc-avr. It supports in-system programming.

You download the above packages untar, configure, and install it. If you are using Debian or Ubuntu, these packages are available in your distribution: Install them using apt or synaptic package manager.

2.4. Hello World

We are writing **hello world** for the atmega8 microcontroller, which has a 28-pin, 8-bit, RISC architecture.

Before proceeding, have a look at this manual about __avr-libc__, which will help you program better, and understand. Also, refer to the datasheets for the various AVR microcontrollers.

Here is our first program:

```
/* ledblink.c, an LED blinking program */
#include<avr/io.h>
#include<util/delay.h>
void sleep(uint8_t millisec)
{
        while(millisec)

        {
                _delay_ms(1);/* 1 ms delay */
                millisec--;
        }
}
main()
{

        DDRC |=1<<PC2;  /* PC2 will now be the output pin */
        while(1)
        {
                PORTC &= ~(1<<PC2);/* PC2 LOW */
                sleep(100);/* 100 ms delay */

                PORTC |=(1<<PC2); /* PC2 HIGH */
                sleep(100);/* 100 ms delay */
        }
}
```

2.4.1. Code Explanation

The GNU C compiler for the Atmel family identifies all functional units within the microcontroller with meaningful names. Thus, writing `PORTC=0xff' will result in the compiler generating machine code that writes 0xff to I/O port C, which will set all port C pins to logic high. Because ports are bidirectional, we must decide whether each pin should act as input or output. If the i'th bit of a register called DDRC (data direction register C) is 1, then the i'th pin of PORTC's i'th pin will be an output. Otherwise, it will act as an input pin. (Note that pin and bit numbers start at zero.) To make an LED blink, you have to make a pin high, then low. (Here, we use PORTC's 2nd port. That is, PC2 will be the 25th pin.) There should be a delay between the two. This is what the rest of the code does. For the delay, we use built-in function *_delay_ms(1)*, which causes a 1 ms delay.

2.4.2. Compilation

```
avr-gcc -mmcu=atmega8 Os ledblink.c o ledblink.o
```

which will result in object file ledblink.o. Now, we will covert it to hex file, suitable for burning to the microcontroller's memory.

```
avr-objcopy -j .text -j .data -O ihex  ledblink.o  ledblink.hex
```

We are converting it to a hex file because, for burning the code to atmega8, we will use uisp, whose input file must be a .hex file.

Notice that you can *less* the ledblink.hex file.

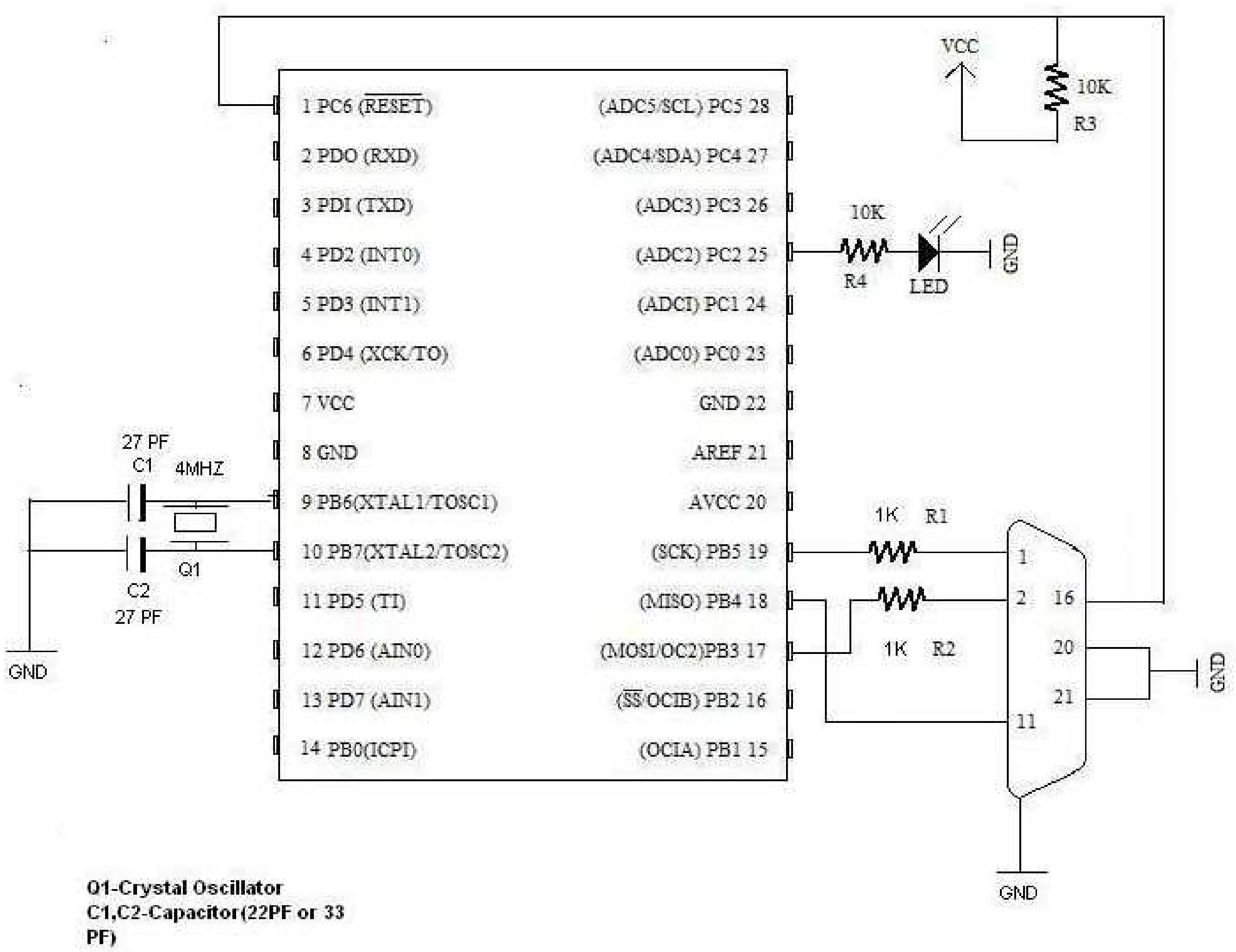
```
:1000000012C02BC02AC029C028C027C026C025C0C6
:1000100024C023C022C021C020C01FC01EC01DC0DC
:100020001CC01BC01AC011241FBECFE5D4E0DEBF28
:10003000CDBF10E0A0E6B0E0EAE8F0E002C0059035
:100040000D92A036B107D9F710E0A0E6B0E001C0EC
:100050001D92A036B107E1F70CC0D2CF282FE4ECF7
:10006000F9E004C0CF010197F1F721502223D1F725
:100070000895CFE5D4E0DEBFCDBFA29AAA9884E66A

:0A008000EDDFAA9A84E6EADFF9CF6B
:00000001FF
```

2.4.3. Burning the Code

2.4.3.1. Hardware

We will be using the parallel port for burning. First, we have to develop a burning circuit for it.



This is the circuit for the atmega8 microcontroller. Pin 9 & pin 10 are connected by a 4 MHz crystal oscillator, which is the external clock. The bottom right connector is for a parallel port.

If you are using any other microcontroller, as mentioned above, you should change accordingly. }}}

You should watch for *RESET,XTAL1,XTAL2,SCK,MISO,MOSI pins*, and connect.

2.4.3.2. Software

Now, we will burn ledblink.hex to the microcontroller.

```
uisp -dprog=dapa -dlpt=0x378
```

You should get message *Atmega8 Found*.

dprog is the programming method specifier, which in this case is *dapa*, i.e., Direct AVR Parallel Access. *dlpt* is for the parallel device setting, which is 0x378, the parallel port's device address.

```
uisp -dprog=dapa -dlpt=0x378 --erase
```

Will erase the microcontroller's code.

```
uisp -dprog=dapa -dlpt=0x378 --upload if=ledblink.hex
```

Will *upload* the *I*nput *F*ile ledblink.hex

Notice that you can see the LED at pin 25 blinking.

2.5. Author

Comments to: mailto:ranjeeth_gecmail[at]yahoo[dot]com

Ranjeeth Weblog

Govt Engg College Sreekrishnapuram,

Palakkad,Kerala India.

2.6. Last version

You may find the last up-to-date version of this HOWTO on the LDP wiki.