Programando el microcontrolador AVR con GCC usando libc 1.0.4

por Guido Socher (homepage)

Sobre el autor:

A Guido le gusta Linux porque es un buen sistema donde desarrollar uno mismo su propio hardware.

Traducido al español por:
Alberto Pardo
<apardoyo(at)yahoo.es>

Resumen:

El microcontrolador de 8 bits AVR de tecnologia RISC de Atmel es un microcontrolador muy popular. Este microcontrolador es un chip con EPROM, Ram, un conversor Analógico−Digital, unas cuantas entradas y salidas digitales, timers, una UART para comunición RS 232 y muchas otras cosas.

Sin embargo, lo mejor es su entorno de programación disponible en Linux: Se puede programar este microcontrolador in C usando GCC.

Ya escribí en Marzo del 2002 un artículo sobre el mismo tema. Han canbiado muchas cosas en el desarrollo de avr−libc y Atmel ha dejado de fabricar el microcontrolador AT90S4433 que use en el 2002 . Por lo que este artículo es una actualización del publicado en Marzo del 2002. Utilizaré la libc−1.0.4 y el microcontrolador ATmega8.

Este artículo sólo es una introducción. En posteriores artículos construiremos interesantes montajes basados en este microcontrolador. Pero ahora vamos a conocer al ATmega8.

Introducción

Mucha gente se interesó por la programación de microcontroladores despues del artículo que escribí en el 2002.Sin embargo tener el sistema de desarrollo listo y funcionando es el proceso más complicado. Si algo falla y no tienes pistas de por donde puede fallar, entonces es el momento de planterase una serie de posibilidades como causantes del fallo : ¿El cable esta mal? ¿Es un fallo del circuito? ¿Es correcta la instalación? ¿Esta desactivado el puerto paralelo en la Bios?.

Para facilitar la entrada al excitante mundo de los microcontroladores la empresa shop.tuxgraphics.org ofrece un CD botable con el manual y el hardware de programación. Todo lo que necesitas es arrancar con este CD. No hace falta instalar el software, por lo que no se modificara nada en tu ordenador.

Particularmente, el uso de este CD autoarrancable me presenta la ventaja de no preocuparme por todas las actualizaciones del kernel o instalaciones de software en mi PC. Si tengo que actualizar el software del microcontrolador, arranco desde el CD y todo lo tengo listo y funcionando.

Independientemente de este CD, esplicaré a continuación la instalación del entorno de desarrollo de avr en GCC. Si tienes el CD de tuxgraphics continua en el capítulo "Un pequeño proyecto de test".

Instalación del software : Todo lo que necesitas

Para usar el entorno de desarrollo GNU C necesitaras el siguiente software:

binutils−2.15.tar.bz2	Disponible en: ftp://ftp.gnu.org/gnu/binutils/ o cualquier mirror. Ejemplo: ftp://gatekeeper.dec.com/pub/GNU/binutils/
gcc−core−3.4.2.tar.bz2	Disponible en: ftp://ftp.gnu.org/gnu/gcc/ o cualquier mirror. Ejemplo: ftp://gatekeeper.dec.com/pub/GNU/gcc/
avr−libc−1.0.4.tar.bz2	La libreria AVR C disponible en: http://savannah.nongnu.org/projects/avr−libc/
uisp−20040311.tar.bz2	El software de programación AVR disponible en: http://savannah.nongnu.org/projects/uisp

Instalaremos todos los programas en /usr/local/avr. De esta manera los mantendremos separados del compilador de C de Linux. Crea este directorio con el comando:

```
    mkdir /usr/local/avr

Añades el camino en el PATH:
    mkdir /usr/local/avr/bin
    export PATH=/usr/local/avr/bin:${PATH}
```

Instalacion del Software : las bintuils de GNU

El paquete de las binutils proporciona todas la utilidades necesarias a bajo nivel para construir los fichero objeto. Estas utilidades incluyen: Un ensamblador AVR (avs−asm),un enlazador o linkador (avr−ld),la libreria

"handling tools" (avr−ranlib,avr−ar), los programas para generar los ficheros objeto para poder ser cargados a la EEPROM del microcontrolador (avr−objcopy), el desensamblador (avr−objdump) y las utilidades avr−strip y avr−size.

Ejecuta las siguientes instruciones para construir e instalar las binuitls:

```
tar jxvf binutils-2.15.tar.bz2
cd binutils-2.15/
mkdir obj-avr
cd obj-avr
../configure --target=avr --prefix=/usr/local/avr --disable-nls
make

# as root:
make install
```

Añade la linea /usr/local/avr/lib al fichero /etc/ld.so.conf y ejecuta el comando /sbin/ldconfig para reconstruir la cache del enlazador o linkador.

Instalación del Software : AVR gcc

avr−gcc será nuestro compilador de C.

Ejecuta el siguiente comando para construirlo e instalarlo:

```
tar jxvf gcc-core-3.4.2.tar.bz2
cd gcc-3.4.2

mkdir obj-avr
cd obj-avr
../configure --target=avr --prefix=/usr/local/avr --disable-nls --enable-language=c

make

# as root:
make install
```

Instalación del Software : La libreria de C de AVR

La actual libreria de C es lo suficientemente estable comparada con la que presente en Marzo del 2002. Ejecuta el siguiente comando para construirlo e instalarlo

```
tar jxvf avr-libc-1.0.4.tar.bz2
cd avr-libc-1.0.4
PREFIX=/usr/local/avr
export PREFIX
sh -x ./doconf
./domake

cd build
#as root:
make install
```

Instalación del Software : El Programador

El software de programación carga el codigo objeto en la EEPROM de nuestro microcontrolador.

El programador uisp para Linux es un buen programador. Se puede usar desde el "Makefile". Tan solo con añadir la regla "make load", podras compilar y cargar el software en un paso.

Ver como se instala uisp:

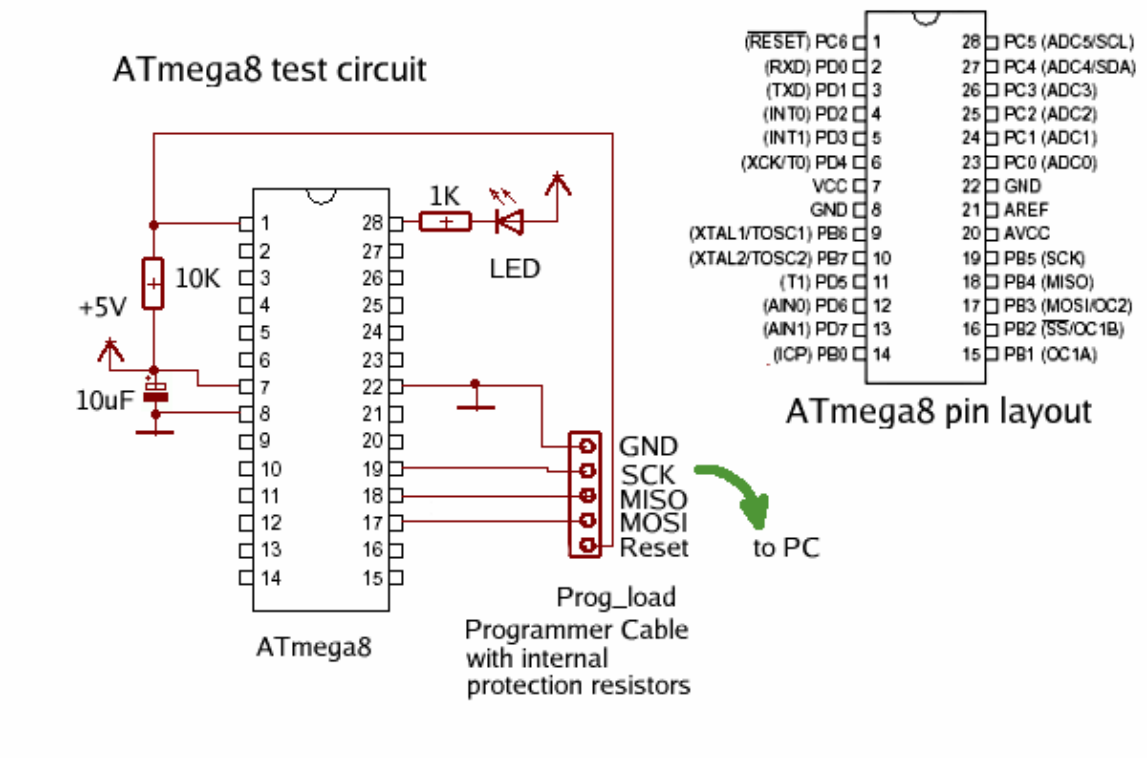
```
tar jxvf uisp-20040311.tar.bz2.tar
cd uisp-20040311
./configure --prefix=/usr/local/avr
make

# as root:
make install
```

Un pequeño proyecto de test

Empezaremos con un circuito sencillo de test, el cual ampliaremos más adelante.

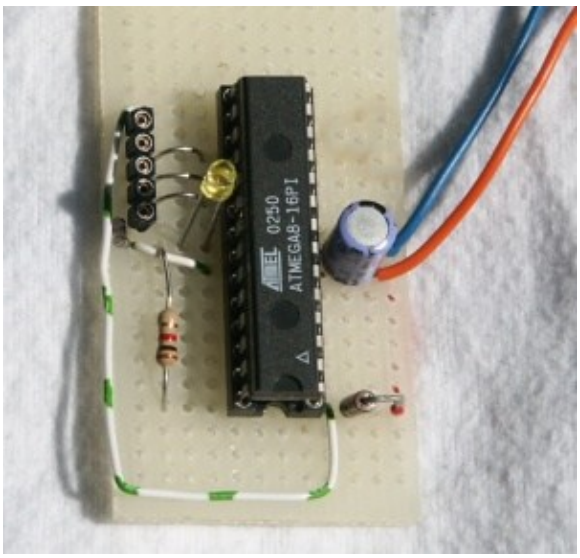
Este circuito lo podremos usar como un test sencillo de entorno para desarrollos más complejos de hardware. Podrás cargar facilmente el software en el microcontroloador y comprobar los sensores conectados o el equipo de medida.

El siguiente programa de test hará parpadear a un led.



Hardware necesario

Necesitaras los componentes que aparecen en el listado de la tabla. Aunque se trata de un microcontrolador muy común,te resultará más facil localizarlo en un gran distribuidor de componentes electrónicos : www.conrad.de (germany), www.selectronic.fr (france), digikey.com (US, CA), etc... También puedes adquirir todo el kit o sólo el microcontrolador en shop.tuxgraphics.org



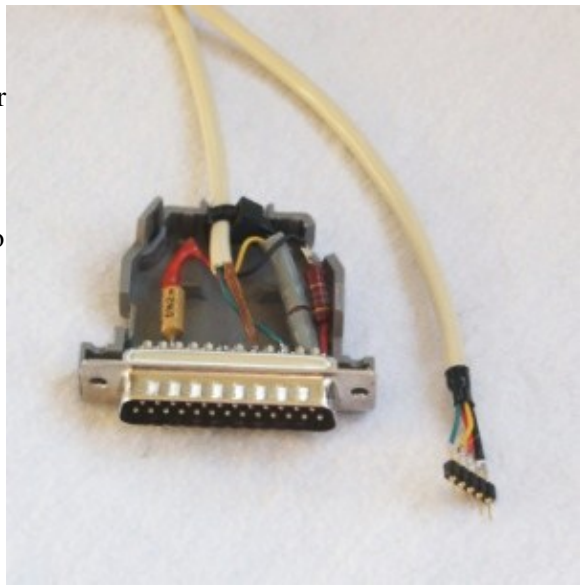
1 x ATmega8 versión DIP , procesador Atmel 8 bit Avr risc.
Zocalo 1 x 28 pins 7.5mm Los zocalos de 28 pins son dificiles de localizar.Tener en cuenta que suelen ser de 14 mm de ancho , pero para el montaje necesitamos un zocalo de 7.5mm.
1 x resistencia de 10K (código de color : marrón,negro, naranja) 1 x resistencia de 1K (código de color : marrón,negro,rojo) 1 x 10uF condensador electrolítico Algunos cables

1 x LED matrix board (protoboard o placa perforada con pads soldables)
El siguiente material se necesita para el programador (no es necesario si se tiene el "Kit AVR de programación en Linux" de tuxgraphics): 1 x conector DB25 para conectar al puerto paralelo. Cualquier tipo de conector/zocalo de 5 pins para conectar al programador. Recomiendo el uso de conectores de precisión strip (similares a los zocalos). 1 x Resistencia de 220 Ohm (código de color : rojo,rojo,marrón) 2 x Resitencia de 470 Ohm (código de color : amarillo,violeta,marrón)

Además de una fuente electrónica de 5V o en su defecto , una bateria de 4,5V.

Habrás observado que no es necesario un cristal. Actualmente el ATmega8 tiene incorporado un oscilador. Se puede usar este oscilador cuando no se necesite un alta precisión de reloj. Por contra, si necesitas construir un equipo de medida o usar el interface UART/RS232 necesitaras el cristal. El tipo de oscilardo usado se puede modificar con el programa. Por defecto (selección de fabrica) esta ativo el oscilador interno de 1MHz.

Construyendo el hardware del programador



Los microcontroladores AVR permiten la programación "in circuit". (ISP).
Esto significa que no es necesario extraer el microcontrolador de la placa de circuito impreso para reprogramarlo.
Encontraras varios tipos de programadores desde 50 hasta 150 euros. De cualquier forma, ejecutando Linux es posible tener un programador sencillo. Necesitarás un puerto paralelo libre y el siguiente cable.

Notar que se trata de un programador mejorado respecto al presentado en el artículo presentado en Marzo del 2002. Las resistencias protectoras se construyen en el interior del programador. Esto ahorra espacio y componentes en la placa del cirucuito impreso. A continuación se puede ver como se cablea:

pin sobre la pcb	pin sobre el AVR	resistencia de proteción	Pin sobre el puerto paralelo
5	Reset (1)	––	Init (16)
4	MOSI (17)	470 Ohm	D0 (2)
3	MISO (18)	220 Ohm	Busy (11)
2	SCK (19)	470 Ohm	Strobe (1)
1	GND	––	GND (18)

El cable no ha de ser más largo de 70cm.

La resistencia de protección se puede montar dentro del mismo conector. Tal como se muesta en la foto de la derecha.

Esribiendo el software

El ATmega 8 se puede programar en C standard con la ayuda de gcc. Puede ser útil conocer algo de ensamblador del AVR , pero no es necesario.

El libc de AVR libc viene con un manual avr−libc−user−manual−1.0.4.pdf (1139921 bytes) que documenta todas las librerias disponibles en C. En la web de Atmel (www.atmel.com, ir a: avr products −> 8 bit risc−> Datasheets), podrás descargar la "data sheet" completa. En ella se describen todos los registros y como usarlos.

Una cosa a tener en cuenta cuando se usa el microcontrolador es que sólo dispone de pocos bytes de RAM. Esto significa que no puedes declarar estructuras largas de datos o strings. Los programas no deben contener demasiadas llamadas anidadas o de recursión.

Pasemos a la práctica. Vamos a escribir un programa que haga parpadear nuestro led a intervalos de 0.5 segundos. No es muy útil, pero ya esta bien para empezar.

La avr−libc ha cambiado bastante. Antes se activaba un bit en el puerto con sbi y lo desactivabas cbi. Ahora estas fuciones se han eliminado. No obstante, presento un programa usando estas funciones eliminadas:

```
/* Definición de "defines" para una compatibilidad futura */
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

void main(void)
{
      /* INICIALIZACION */
      /* activa PC5 como salida */
      sbi(DDRC,PC5);


      /* Parpadeo, Parpadeo ... */
      while (1) {
            /* led on, pin=0 */
            cbi(PORTC,PC5);
            delay_ms(500);
            /* Activa la salida a 5V, LED apagado */
            sbi(PORTC,PC5);
            delay_ms(500);
      }
}
```

El siguiente ejemplo hace exactamente lo mismo, pero usando la nueva sintaxis:

```
void main(void)
{
      /* INICIALIZACION */
      /* activa PC5 como salida */
      DDRC|= _BV(PC5);


      /* Parpadeo, Parpadeo ... */

      /* PC5 esta a 5 (ver el fichero include/avr/iom8.h) y _BV(PC5) es 00100000 */
```

```
        while (1) {
              /* led on, pin=0 */
              PORTC&= ~_BV(PC5);
              delay_ms(500);
              /* Activa la salida a 5V, LED desactivado */
              PORTC|= _BV(PC5);
              delay_ms(500);
        }
}
```

El código superior muestra lo simple que es escribir un programa. Mirando sólo el programa principal, la función delay_ms esta incluida en el listado completo (avrm8ledtest.c). Para usar el pin PC5 como salida se de activar el bit PC5 en el registro de dirección de datos del port C(DDR). Despues puedes activar PC5 a 0V con la función cbi(PORT,PC5) (Desactiva el bit PC5) o a 5V con sbi(PORTC,PC5))(activa el bit PC5).El valor de "PC5" se define en iom8.h que se incluye via io.h. No hace falta que te preocupes por ello. Si has escrito programas para sistemas multiusuario / multitarea como Linux, sabrás que no se ha de programar un bucle infinito.Esto consumirá tiempo de CPU y enlentecerá el sistema mucho.Pero el caso de AVR es diferente.No tenemos varias tareas y no hay ningún otro programa ejecutandose. Aquí no hay un sistema operativo. Es normal tener un bucle infinito.

La compilación y la Carga

Antes de nada, asegurate que tienes /usr/local/avr/bin en el PATH. Si es necesario edita tu fichero .bash_profile o .tcshrc y añadelo:

export PATH=/usr/local/avr/bin:${PATH} (para bash)
setenv PATH /usr/local/atmel/bin:${PATH} (para tcsh)

Utilizamos el puerto paralelo u el uisp para programar el AVR. Uisp usa el interface ppdev del kernel. Por lo tanto necesitaras tener los siguientes modulos del kernel cargados:

```
# /sbin/lsmod
parport_pc
ppdev
parport
```

Verifica con el siguiente comando : /sbin/lsmod , que están cargados. En otro caso, cargalos (como root) con:

```
modprobe parport
modprobe parport_pc
modprobe ppdev
```

Sería buena idea que se ejecuten automáticamente durante el startup. Puedes añadirlos en un script rc (ejemplo para Redhat /etc/rc.d/rc.local).
Para poder acceder al interface ppdev como usuario, el root te ha de dar permiso de acceso mediante el comando:

chmod 666 /dev/parport0

Asegurate que el daemon de la impresora no está ejecutandose ya que utiliza el puerto paralelo. Si esta ejecutandose es mejor desactivar este daemon antes de conectar el cable paralelo. Ahora ya esta todo listo para compilar y programar nuestro microcontrolador.

El paquete para nuestro programa test (avrm8ledtest−0.1.tar.gz) incluye un fichero make. Tan sólo tienes que teclear:

make
make load

Esto compilará y cargará el software.No te explicaré los detalles de todas las instruciones. Las puedes ver en Makefile y son siempre las mismas. No puedo recordarlas todas. Si quieres escribir un programa diferente,entoces reemplaza en los lugares pertinentes de avrm8ledtest en el Makefile con el nombre de tu programa.

Algunas binutils interesantes

Hay algunas binutils que resultan más interesantes que el propio proceso de compilación.
Esta utilidades no han cambiado desde Marzo del 2002. Revisa el capítulo "Algunas binutils interesantes" en article231, March 2002.

Ideas y sugerencias

Para muchos casos el ATmega8 es compatible con el AT90S4433.Tendrás que programar el microcontrolador para usar el oscilador externo y podrás utilizar con pocos cambios todo el hardware que presenté en articulos anteriores. Desgraciadamente no he tenido tiempo de probar todos los circuitos con el ATmega8. Para evitar problemas, sería mejor usar el viejo AT90S4433. Pero si no te importa los problemas y deseas solventarlos entonces prueba el ATmega8 con los artículos y montajes previos.

A continuación tienes una lista de los artículos realizados con el hardware antiguo:

- Un panel de control LCD para tu servidor linux
- Una fuente de alimentación controlada por un microcontrolador
- Un frecuenciometro con rango de 1Hz a 100MHz con display LCD e interface RS232
- Un display LCD USB en Linux con watchdog y pulsadores
- Construcción de un robot autónomo guiado por luz

Nota: El programador presentado el este artículo, ya incorpora las resistencias protectoras que se montaban en los circuitos propuestos en pasados artículos. Para poder usar las placas propuestas en esos artículos, se han de sustituir las resistencias protectoras por un cable o puente.

Atmel suministra una nota de aplicación "AVR081: Replacing AT90S4433 by ATmega8" donde se indican todas las incompatibilidades : at90s4433_to_atmega8.pdf (101343 bytes)

Referencias

- La AVRlib Pascal Stang's : http://www.procyonengineering.com/avr/avrlib/index.html o http://hubbard.engr.scu.edu/embedded/avr/avrlib/
- el assebler tavrasm para Linux: www.tavrasm.org
- **Todo el software y la documentación mencionada en este articulo**
- El website de atmel: www.atmel.com
- La web de la tienda de tuxgraphics: shop.tuxgraphics.org
  (Donde puedes obtener el CD de programación del AVR en Linux, kits y microcontroladores)

Contactar con el equipo de LinuFocus http://www.LinuxFocus.org	Información sobre la traducción: en −−> −− : Guido Socher (homepage) en −−> es: Alberto Pardo <apardoyo(at)yahoo.es>

2005−01−12, generated by lfparser_pdf version 2.51