Programar y depurar código para el MSP430 en Linux

El presente documento describe los pasos necesarios para instalar un ensamblador, monitor, compilador y depurador para la linea de microcontroladores MSP430 bajo la distribución Ubuntu. Para propósitos de este documento llamaremos a la MSP-EXP430G2 como LaunchPad.

1. Instalación del ensamblador y monitor

Existen varios ensambladores disponibles para la linea de MSP430. Una alternativa de código abierto es `naken_asm`, el cual es un ensamblador ligero y fácil de usar, que incluye soporte para varias arquitecturas de microcontroladores y microprocesadores, entre ellas la línea MSP430. Para instalarlo en tu computadora deberás seguir los pasos que a continuación se listan.

1. Descarga el código fuente de la siguiente liga
   `http://downloads.mikekohn.net/naken_asm/naken_asm-2015-04-04.tar.gz`
2. Abre una Terminal y ejecuta el siguiente comando:

```
[usr1@server ~] tar -zxvf ~/Downloads/naken_asm-2015-04-04.tar.gz
```

3. Cambia el directorio de trabajo actual al directorio que acabas de crear.

```
[usr1@server ~]$ cd naken_asm-2015-04-04
```

4. Ejecuta el siguiente comando para configurar los archivos de creación

```
[usr1@server naken_asm-2015-04-04]$ ./configure
```

5. Compila el código fuente con el comando

```
[usr1@server naken_asm-2015-04-04]$ make
```

6. Instala el archivo ejecutable resultante

```
[usr1@server ~]$ sudo make install
```

El programa monitor con el que vamos a trabajar tiene por nombre `mspdebug`, y nos permitirá cargar nuestro código objeto en la memoria FLASH del MSP430, así como realizar acciones básicas de depuración como ejecutar paso a paso nuestro programa, establecer puntos de paro condicionales e incondicionales, desplegar el contenido de regiones de memoria así como de los registros del MSP430. Para instalar y configurar `mspdebug` debemos seguir los pasos que a continuación se listan

1. Descarga e instala `mspdebug` usando el administrador de paquetes `apt-get`

   ```
   [usr1@server ~]$ sudo apt-get install mspdebug
   ```

2. Configura el puerto virtual asociado a tu LaunchPad de forma que tengas permisos de lectura y escritura.

   ```
   [usr1@server ~]$ sudo sh -c "cat > /etc/udev/rules.d/93-msp430uif.rules"
   ```

   Después de presionar ↵ deberás teclear las siguientes lineas de texto:

   ```
   ATTRS{idVendor}=="2047",ATTRS{idProduct}=="0010",MODE="0666",ENV{ID_MM_DEVICE_IGNORE}="1"
   ATTRS{idVendor}=="0451",ATTRS{idProduct}=="f432",MODE="0666",ENV{ID_MM_DEVICE_IGNORE}="1"
   ^D
   ```

   (`^D` representa la combinación de teclas ctrl + D)
   con lo cual crearás un archivo que le indicará al manejador de dispositivos que todos los usuarios tienen permiso de acceso al puerto virtual asociado a tu LaunchPad, y que esta no debe se tratada como modem. Para que los cambios surtan efecto, debes reiniciar el servicio `udev` en Ubuntu

   ```
   [usr1@server ~]$ sudo service udev restart
   ```

3. Finalmente, para evitar un retraso de 10s en el reconocimiento del dispositivo, se le indicará al kernel, mediante el siguiente comando, que no trate de accesar un segundo dispositivo asociado a la LaunchPad.

   ```
   [usr1@server ~]$ sudo sh -c "cat > /etc/modprobe.d/msp430uif.conf"
   ```

   Después de presionar ↵ deberás teclear las siguientes lineas de texto:

   ```
   options usbhid quirks=0x0451:0xf432:0x20000000,0x2047:0x0010:0x20000000
   ^D
   ```

   Para que los cambios surtan efecto, ejecuta el comando

   ```
   [usr1@server ~]$ sudo rmod usbhid && modprobe usbhid
   ```

Para demostrar el uso del ensamblador, vamos a tomar el programa `blink.asm` listado en Código 1. Una ves que lo hayas capturado en un editor de textos, ejecuta el siguiente comando

```
[usr1@server ~]$ naken_asm -o blink.hex blink.asm
```

donde `blink.hex` es el programa objeto que se cargará a la LaunchPad mediante el comando

```
[usr1@server ~]$ mspdebug rf2500 "prog blink.hex"
```

donde `rf2500` es el protocolo utilizado por el monitor para comunicarse con la LaunchPad. Si no surge ningún contratiempo, deberás ver que ambos LEDs en la LaunchPad comienzan a parpadear.

Código 1 : blink.asm

```
.include "msp430g2553.inc"

  org 0xf800
start:
  ;mov.w #0x5a80, &WDTCTL
  mov.w #WDTPW|WDTHOLD, &WDTCTL
  mov.b #0x41, &P1DIR
  mov.w #0x01, r8
repeat:
  mov.b r8, &P1OUT
  xor.b #0x41, r8
  mov.w #40000, r9
waiter:
  dec r9
  jnz waiter
  jmp repeat

  org 0xfffe
  dw start ; set reset vector to 'init' label
```

2. Instalación del compilador y depurador

El compilador que vamos a instalar tomará código fuente en lenguaje C y lo traducirá a código máquina capaz de ejecutarse en el MSP430. Este proceso se conoce como compilación cruzada dado que el proceso de compilación se ejecuta en una arquitectura diferente a la objetivo. Las herramientas que vamos a ocupar se derivan de `gcc`, que es el compilador de código abierto para lenguaje C, y pueden ser fácilmente instaladas en Ubuntu mediante el administrador de paquetes mediante el comando

```
[usr1@server ~]$ sudo apt-get install msp430-libc binutils-msp430 gcc-msp430 gdb-msp430 msp430mcu
```

Los nombres de los paquetes pueden variar ligeramente en algunas distribuciones, por ejemplo el paquete `gcc-msp430` puede llamarse también `msp430-gcc`. En algunas versiones de Ubuntu el comando anterior puede presentar problemas al instalar el paquete `gdb-msp430` y arrojar un error, si esto es así, intenta con el siguiente comando

```
[usr1@server ~]$ sudo apt-get -o Dpkg::Options::="--force-overwrite" install gdb-msp430
```

para después instalar los paquetes restantes con el comando

```
[usr1@server ~]$ sudo apt-get install msp430-libc binutils-msp430 gcc-msp430 msp430mcu
```

Para ilustrar el uso del compilador, vamos a requerir de dos archivos, los cuales puedes descargar de la siguiente liga: `http://kali.azc.uam.mx/erm/Media/1123021/cblink.zip`.

El archivo `cbink.c` contiene el programa fuente listado en Código 2, mientras que el archivo `Makefile` contiene los parámetros necesarios para compilar el código fuente.

Código 2 : cblink.c

```
#include <msp430g2231.h>
volatile unsigned int i = 0;
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Detiene el temporizador "watchdog".
  P1DIR |= 0x41; // Configura P1.0 y P1.6 como salida
  for (;;) // Este ciclo FOR se repetira indefinidamente
    {
      for(i=0; i< 20000; i++){ // Retardo entre parpadeo de los LEDs
        if (i == 0)
           P1OUT ^= 0x01; // Cambio de estado del LED rojo (P1.0)
        if (i == 6000)
           P1OUT ^= 0x40; // Cambio de estado del LED verde (P1.6)
      }
    }
}
```

Código 3 : Makefile

```
# Compilador a usarse
CC=msp430-gcc
# Banderas para compilacion
CFLAGS= -Wall -g -mmcu=msp430g2452
# Archivo objeto
OBJS=cblink.o

# Reglas para compilacion
all: $(OBJS)
    $(CC) $(CFLAGS) -o cblink.elf $(OBJS)

# Esta es una regla implicita que dice como compilar todos los
# archivos con extension *.c para obtener archivos *.o
# el valor de $< se toma del lado derecho de los dos puntos
%.o: %.c
    $(CC) $(CFLAGS) -c $<

# Borra los archivos *.o y *.elf
clean:
    rm -fr cblink.elf $(OBJS)
```

Una vez que hayas descargado el archivo `cblink.zip`, tendrás que descomprimirlo de la siguiente forma

```
[usr1@server ~]$ unzip ~/Downloads/cblink.zip
```

Para compilar el código fuente, simplemente ejecuta el siguiente comando dentro del directorio que se creo

```
[usr1@server cblink]$ make
```

Una vez compilado el código fuente, asegúrate que tu LaunchPad este conectada a tu computadora mediante el cable USB que viene con ella. Ahora, carga el código objeto resultante a la LaunchPad

```
[usr1@server cblink]$ mspdebug rf2500 "prog cblink.elf"
```

Si todo sale bien, verás que los LEDs se encenderán uno después del otro, con un pequeño retardo de diferencia, para después apagarse en sentido inverso.

A continuación utilizaremos `mspdebug` en conjunto con `gdb-msp430` para depurar el programa cargado en la LaunchPad. Primero debemos indicarle `mspdebug` que servirá de interfaz para `gdb-msp430`

```
[usr1@server cblink]$ mspdebug rf2500 "gdb"
```

Al oprimir ↵ observarás que varios mensajes son desplegados en la terminal,

```
MSPDebug version 0.23 - debugging tool for MSP430 MCUs
Copyright (C) 2009-2015 Daniel Beer <dlbeer@gmail.com>
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Chip info database from MSP430.dll v3.3.1.4 Copyright (C) 2013 TI, Inc.

Trying to open interface 1 on 001
Initializing FET...
FET protocol version is 30394216
Set Vcc: 3000 mV
Configured for Spy-Bi-Wire
fet: FET returned error code 4 (Could not find device or device not supported)
fet: command C_IDENT1 failed
Using Olimex identification procedure
Device ID: 0x2452
  Code start address: 0xe000
  Code size         : 8192 byte = 8 kb
  RAM  start address: 0x200
  RAM  end   address: 0x2ff
  RAM  size         : 256 byte = 0 kb
Device: MSP430G2xx2
Number of breakpoints: 2
fet: FET returned NAK
warning: device does not support power profiling
Chip ID data: 24 52
Bound to port 2000. Now waiting for connection...
```

estos indican que tu LaunchPad se esta comunicando con tu computadora. Presta atención al último mensaje, el cual indica que `mspdebug` esta esperando por una conexión a traves del puerto 2000. Ya que la terminal actual se encuentra ocupada por `mspdebug`, tendremos que abrir otra terminal, desde donde ejecutaremos

```
[usr1@server cblink]$ gdb-msp430 cblink.elf
```

lo cual inicia `gdb-msp430`, y le señala que quieres depurar el programa objeto `cblink.elf`. Después de una serie de mensajes, que nos indican que `gdb-msp430` ha sido iniciado y que el programa objeto `cblink.elf` ha sido cargado a memoria, veremos un *prompt* nuevo, desde donde podremos interactuar con `gdb-msp430`. Ahora, ordenaremos a `gdb-msp430` conectarse a la LaunchPad a traves de `mspdebug`

```
(gdb) target remote localhost:2000
```

Si el comando anterior se ejecuto sin errores, podremos usar comandos para examinar variables y secciones de memoria.

El compilador inserta un poco de código adicional que inicializa algunos registros, por lo que nuestra función `main()` no inicia inmediatamente. Para localizar automáticamente el inicio de `main()`, pondremos un punto de paro (*breakpoint*) de la siguiente forma

```
(gdb) break main
```

y iniciaremos la ejecución de nuestro programa

```
(gdb) c
```

con lo cual el MSP430 iniciará la ejecución del programa que tiene en memoria y se detendrá en la localidad donde inicia la función `main()`.

Existe un sin número de documentación acerca de `gdb` en Internet [1], pero lo mas efectivo es usar la ayuda que esta incluida en el mismo `gdb-msp430`, la cual se invoca de la siguiente manera

```
(gdb) help
```

A continuación se presenta una lista de los comandos más usados:

- `list` – Muestra unas cuantas lineas de código fuente a partir de la linea actual.
- `next` – Ejecuta todas las instrucciones en ensamblador correspondientes a la línea de código fuente actual.
- `nexti` – Ejecuta la instrucción en ensamblador indicada por el PC.
- `info reg` – Muestra el valor actual de los registros del CPU.
- `info break` – Muestra una lista de los puntos de paro existentes.
- `disassemble /m` – Muestra el código ensamblador correspondiente a cada una de las líneas de código fuente correspondiente al bloque en el que se encuentra el contador de programa (PC).
- `disable 1` – Deshabilita el punto de ruptura 1.
- `condition 1 i == 3` – Especifica una condición de paro en el punto de ruptura 1. La condición de paro es `i == 3`.
- `monitor reset` – Reinicia el contador de programa.
- `monitor <cmd>` – Ejecuta el comando `<cmd>` del monitor `mspdebug` desde `gdb-msp430`. La lista de comandos disponibles en `mspdebug` se puede encontrar en [2]

Finalmente, para cerrar una sesión en `gdb-msp430` tecleamos

```
(gdb) quit
A debugging session is active.

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y
```

Referencias

[1] [Online]. Available: https://www.gnu.org/software/gdb/documentation/

[2] [Online]. Available: http://pm1.bu.edu/~tt/mspdebug+launchpad.pdf