Programar y depurar código para el MSP430 en Mac OS X

El presente documento describe los pasos necesarios para instalar un ensamblador, monitor, compilador y depurador para la linea de microcontroladores MSP430 bajo el sistema operativo Mac OS X Yosemite. Para propósitos de este documento llamaremos a la MSP-EXP430G2 como LaunchPad.

1. Instalación del manejador de paquetes

El objetivo del manejador de paquetes es facilitar la instalación del compilador y depurador. El manejador de paquetes que vamos a instalar se llama MacPorts. Existe una guia de instalación bastante fácil de seguir en `http://guide.macports.org`, pero básicamente tienes que realizar tres pasos:

1. **Descarga e instala X11**. Probablemente ya este instalado en tu sistema operativo, pero si no, lo único que tienes que hacer es descargar el archivo `XQuartz-2.7.7.dmg` de `http://xquartz.macosforge.org`. Una vez terminada la descarga, abre el archivo y sigue los pasos de instalación indicados.
2. **Instala/actualiza Xcode**. Si Xcode no esta incluido en tu sistema operativo, es necesario instalarlo desde App Store. Si ya ha sido instalado, asegurate que tienes la última versión, lo cual puedes comprobar si App Store no te pide que actualices Xcode. Después que se haya descargado e instalado, o actualizado correctamente, abre una terminal y ejecuta el comando xcode−select −−install el cual iniciará la instalación de las herramientas Çommand Line Tools". Si en lugar de ver una ventana de instalación, obtienes el siguiente mensaje de error

   ```
   server:~ usr1$ xcode−select −−install
   Can't install the software because it is not currently available from the Software Update server.
   ```

   Es probable que las herramientas requeridas ya hayan sido instaladas.
3. **Descarga e instala MacPorts**. Del sitio web `http://www.macports.org/install.php` descarga el paquete correspondiente a tu versión del sistema operativo. Una vez descargado, abre el paquete y sigue las instrucciones de instalación.

2. 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:

```
server:~ usr1$ tar -zxvf ~/Downloads/naken_asm-2015-04-04.tar.gz
```

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

```
server:~ usr1$ cd naken_asm-2015-04-04
```

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

```
server:naken_asm-2015-04-04 usr1$ ./configure
```

5. Compila el código fuente con el comando

```
server:naken_asm-2015-04-04 usr1$ make
```

6. Instala el archivo ejecutable resultante

```
server:naken_asm-2015-04-04 usr1$ sudo make install
```

7. Copia los archivos cabecera a una ubicación conocida

```
server:naken_asm-2015-04-04 usr1$ sudo mkdir /usr/local/share/naken_asm/
Password:
server:naken_asm-2015-04-04 usr1$ sudo cp -r include/ /usr/local/share/naken_asm/include
```

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` usaremos MacPorts mediante el siguiente comando

```
server:~ usr1$ sudo port install mspdebug
```

Ahora necesitamos instalar un controlador que permita la comunicación entre `mspdebug` y nuestra LaunchPad. Primero tenemos que bajar el paquete de instalación de

`http://energia.nu/files/MSP430LPCDC-1.0.3b-Signed.zip`,

descomprimir el archivo y seguir el proceso de instalación.

Para demostrar el uso del ensamblador, vamos a tomar el programa `blink.asm` listado en Código 2. Una ves que lo hayas capturado en un editor de textos, ejecuta el siguiente comando

```
server:blink usr1$ naken_asm -I /usr/local/share/naken_asm/include/msp430/ -o blink.hex blink.asm
```

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

```
server:blink usr1$ 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

```
1  .include "msp430g2553.inc"
2
3    org 0xf800
4  start:
5    ;mov.w #0x5a80, &WDTCTL
6    mov.w #WDTPW|WDTHOLD, &WDTCTL
7    mov.b #0x41, &P1DIR
8    mov.w #0x01, r8
9  repeat:
10   mov.b r8, &P1OUT
11   xor.b #0x41, r8
12   mov.w #40000, r9
13 waiter:
14   dec r9
15   jnz waiter
16   jmp repeat
17
18   org 0xfffe
19   dw start ; set reset vector to 'init' label
```

3. 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 Mac OS X usando MacPorts mediante el comando

```
server:blink usr1$ sudo port install msp430-libc msp430-binutils msp430-gcc msp430-gdb msp430mcu
```

Actualmente existe un problema al compilar `msp430-gdb` con la última versión de Xcode. Si `msp430-gdb` no se instala correctamente, sigue los pasos que a continuación se listan

1. Abre una terminal y ejecuta los siguientes comandos

```
server:~ usr1$ cd /opt/local/var/macports/sources/rsync.macports.org/release/tarballs/port
s/cross/msp430-gdb
server:msp430-gdb usr1$ sudo chmod go+w Portfile
```

2. Después de proporcionar tu contraseña, usa tu editor favorito para añadir las siguientes lineas al archivo `Portfile`
   - Modifica el texto que empieza en la linea 41 para que se lea

     ```
     configure.args --target=${name_target} \
     --disable-Werror \
     --enable-werror=no \
     ```

   - Modifica el texto en la linea 50 para que se lea

     ```
     compiler.blacklist *clang*
     compiler.fallback-append macports-gcc-4.9
     ```

3. Guarda las modificaciones y ejecuta el comando

   ```
   server:msp430-gdb usr1$ sudo port install msp430-gdb
   ```

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 3, 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 *.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

```
server:~ usr1$ unzip ~/Downloads/cblink.zip
```

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

```
server:cblink usr1$ 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

```
server:cblink usr1$ 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 `msp430-gdb` para depurar el programa cargado en la LaunchPad. Primero debemos indicarle `mspdebug` que servirá de interfaz para `msp430-gdb`

```
server:cblink usr1$ 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

```
server:cblink usr1$ msp430-gdb cblink.elf
```

lo cual inicia `msp430-gdb`, y le señala que quieres depurar el programa objeto `cblink.elf`. Después de una serie de mensajes, que nos indican que `msp430-gdb` ha sido iniciado y que el programa objeto `cblink.elf` ha sido cargado a memoria, veremos un *prompt* nuevo, desde donde podremos interactuar con `msp430-gdb`. Ahora, ordenaremos a `msp430-gdb` 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 [**?**], pero lo mas efectivo es usar la ayuda que esta incluida en el mismo `msp430-gdb`, 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 `msp430-gdb`. La lista de comandos disponibles en `mspdebug` se puede encontrar en [**?**]

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

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

    Inferior 1 [Remote target] will be killed.

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