Revisión	Fecha	Comentario	Autor
0	16/08/2011	Módulos: Digi CME9210 S.O.: DEL5.2 / Linux kernel 2.6 Accesorios: ADC MCP3208	Ulises Bigliati

Índice

Sobre esta nota

Para la realización de este trabajo asumimos que el lector dispone de un kit de desarrollo de Digi para linux embebido (DEL) correctamente instalado. La versión utilizada para este trabajo fue la 5.2 corriendo sobre la distribución de linux contenida en el DVD de instalación. El sistema completo fue instalado sobre una máquina virtual VirtualBox 4.0.12.
El hardware utilizado para probar la aplicación consistió en el ConnectME 9210 y fue utilizado dentro de la placa de prototipos que viene con el kit. Por otra parte se utilizó un conversor analógico/digital MCP3208 (las conexiones entre este dispositivo y la placa de prototipos se indicarán más abajo).
En este trabajo ofrecemos los archivos necesarios para que el lector pueda reproducir la demo, para esto, existen tres opciones:

1. Ejecutar inmediatamente el sistema: En el directorio `linux_images/` se pueden encontrar las imagenes del kernel y del rootfs para grabarlas en el módulo y ejecutar directamente el sistema.
2. Incorporación de los archivos necesarios a un proyecto DEL vacío: En el directorio `demo_files/` se encuentran todos los archivos necesarios para reproducir este trabajo dentro de cualquier proyecto DEL5.2. que incluya al kernel[1] y al roofs. Nótese que cada archivo se encuentra en su ruta correspondiente de destino donde debe ser ubicado, esto es dentro de `[SU_PROYECTO_DEL_5.2]/build/rootfs/`. Así, por ejemplo, el archivo `spi_demo.html` debe copiarse en `build/rootfs/usr/share/www`. El archivo `add_files.sh`, debe copiarse en `[SU_PROYECTO_DEL5.2]/configs/` y así con el resto de los archivos que están en `demo_files/`
   En el caso de `spidev_test.c` se trata del archivo de código fuente principal para ser copiado en el directorio raíz de un proyecto de aplicación común de DEL5.2. Una vez copiados todos los recursos se está en condiciones de modificar, compilar, debugear, etc.
3. Importación de los proyectos completos. Dentro del directorio `exported_projects/` están los proyectos completos, el de aplicación y el del kernel+rootfs listos para ser importados desde el entorno de desarrollo obteniéndose los mismos resultados que en el

[1] En este caso, para que el programa `spidev_test.c` funcione es necesario configurar el kernel para incluir el driver SPI a la vez que se anula el driver.

punto 2, con la ventaja adicional de que el kernel ya quedará configurado para utilizar debidamente el driver SPI. Recomendamos esta alternativa.

Cualquier fuera la opción elegida podrá verse la demostración en funcionamiento al acceder al webserver del módulo que esté ejecutando el sistema:

```
http://[IP_CME9210]/spi_demo.html
```

Alcance

Se asume que el bootloader del hardware a utilizar fue debidamente configurado para utilizarse desde el entorno de desarrollo, esto es básicamente, que las variables de entorno relacionadas con la conectividad entre el módulo y el host de desarrollo estén bien definidas para permitir la ejecución, debug, acceso TFTP para actualización de firmware, etc.
Dejamos entonces, fuera del alcance de esta nota, todo aspecto de configuración básica referido al target y al host de desarrollo, siendo que asumimos que ambos están listos para trabajar en conjunto. Remitimos al lector que requiera efectuar estas configuraciones a la documentación provista por el fabricante para tal fin:

- Building_Your_First_Application.pdf
- U-Boot_Reference_Manual.pdf
- Manual de ayuda en línea del IDE (*Help → Help Content*)

Descripción general del proyecto

Se trata de una aplicación que accede al puerto SPI para leer un conversor AD (MCP3208) y luego exponer vía Web dinámicamente las mediciones obtenidas.

Para ello, se realiza un programa ejecutable que accede al puerto SPI mediante un driver que debe ser cargado previamente en el kernel. La lectura del ADC se realiza en forma iterativa e ininterrumpida actualizando en cada acceso un archivo XML con los valores que son leídos desde el dispositivo serial.

Aquel archivo es accedido por el usuario mediante un browser. Esto se logra gracias a la intervención de un objeto CGI invocado mediante código Javascript utilizando AJAX para solicitar periódica y asíncronamente el archivo XML, cuyo nombre es pasado como parámetro al objeto CGI.
Con los valores transportados en el XML se actualiza la página cada 1 segundo mostrandolos al usuario sin requerir su intervención y sin necesidad de refrescar la página.

Requerimientos del proyecto

La realización del proyecto requiere:

- Creación, configuración y compilación del kernel y del rootfs.
- Creación de la aplicación de lectura del bus SPI.
- Creación de la página web html (incluyendo javascript, AJAX y CGI).
- Modificación del roofs para contener los archivos involucrados y los cambios en los scripts.

El proyecto del kernel

Se debe crear un nuevo proyecto que incluya el kernel y el rootfs. En el kernel, lo importante es agregar el soporte para SPI como un driver preinstalado (no como módulo) y deshabilitar el puerto serie.
Ver en la ayuda en línea (*Help → Help Content*) *Digi ESP for Embedded Linux → 9.8. Serial Peripheral Interface (SPI)*.
Tener en cuenta que como consecuencia de la habilitación del puerto SPI se anula el puerto serial en modo UART.

Hardware

El puerto RS232 no se podrá utilizar luego del booteo de Linux porque estará funcionando en modo SPI. En cambio, sí podrá utilizarse mientras el bootloader mantenga el control del sistema. Si se utiliza la placa incluida en el kit de desarrollo se debe mover el jumper P5 hacia la posición 2-3 para usar el hardware en modo SPI y en posición 1-2 para utilizar el puerto RS232, esto será útil cuando sea necesario interactuar con uBoot.

Se utilizan los pines definidos para SPI según el manual "Hardware Reference" del módulo utilizado, en nuestro caso, ConnectME 9210. La configuración es la siguiente:

Función default	Pin módulo	Función	Pin en P7 (GPIO port, kit desarrollo)
GPIO-0	13	CS	1
GPIO-5	9	CLK	7
TX	8	MOSI	6
RX	7	MISO	8

Por el lado del ADC3208, solo se deben conectar los pines asignados con las señales CS, MOSI, MISO y CLK a los pines correspondientes según puede verse en la hoja de datos del componente. La alimentación de 3.3V puede obtenerse fácilmente de la placa.

Proyecto del programa ejecutable

Se desarrolla modificando el ejemplo spidev_test_c presente en la instalación del entorno de desarrollo. De esta forma se accede fácilmente al puerto SPI para leer el ADC MCP3208 y se genera un archivo xml con los valores obtenidos. Dicho archivo es llamado values.xml y se genera en el directorio /tmp. El archivo ejecutable generado se llama "spidev_test_c". Este ejecutable será incluido luego en el file system para formar parte del sistema linux que estamos desarrollando. A continuación resaltamos las líneas más importantes del código fuente de nuestro programa ejecutable:

incluimos el header que nos da acceso al driver SPI:

```
#include <linux/spi/spidev.h>
```

Más adelante encontramos la identificación del driver SPI:

```
static const char *device = "/dev/spidev1.0";
```

La función **transfer** es la que efectivamente realiza la comunicación con el dispositivo ADC mediante SPI y devuelve el valor leído, construye los comandos e interpreta las respuestas para devolver los valores entregados por el ADC cuantificados según su resolución:

```
static int transfer(int fd, uint8_t ch)
{
	...
	//el siguiente llamado efectúa el intercambio de datos sobre SPI.
	// Los comandos se envían en el buffer "tr" y también allí se
	// recojen las respuestas.
	ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
	...
}
```

Dentro del main() encontramos las llamadas de apertura del puerto:
`fd = open(device, O_RDWR);` y las llamadas a `ioctl()` para la configuración del modo de uso del puerto.

Más adelante, encontramos un loop en el cual se llama a la función **`transfer`**`()` de a 100 veces por cada canal a fin de promediar las lecturas para estabilizarlas:

```
aux=100;
[...]
while(aux--){
      for (ret=0;ret<8;ret++)
            v[ret]+=transfer(fd,ret);
}
```

Una vez hecho esto tenemos las lecturas acumuladas de los 8 canales, así que resta dividir cada acumulador por 100 para obtener los promedios y escribir con ellos el archivo de datos XML:

```
flog = fopen("/tmp/values.xml", "w");
fputs("<?xml version='1.0' encoding='ISO-8859-
1'?>\r\n<values>\r\n\t<analog>\r\n", flog);
for (ret=0;ret<8;ret++){
      val = v[ret]*3.3/409600;
      fprintf(flog,"\t\t<ch>%.04f</ch>\r\n", val);
}
fputs("\t</analog>\r\n</values>\r\n",flog);
fclose(flog);
```

Hecho lo precedente el archivo XML queda escrito y listo para utilizarse en la ruta /tmp/values.xml. El procedimiento se repite indefinidamente dentro del programa.

La interfaz web

Se realiza en html y javascript utilizando metodología AJAX y CGI para acceder y mostrar el archivo XML generado por el ejecutable. El servidor web utilizado es el servidor normal que viene instalado por defecto y no se requiere ningún servicio avanzado.

El objeto CGI

El objeto CGI fue reutilizado y no fue necesario escribirlo especialmente porque ya existía en el sistema linux de Digi y realizaba exactamente la función adecuada.
Ese CGI está disponible por defecto como demostración en el rootfs con el nombre “display” en la ruta: /usr/share/www/cgi-bin/.
La función de este CGI es tan genérica que puede utilizarse en muchas aplicaciones web ya que simplemente toma el nombre de un archivo como parámetro y entrega al browser su contenido.

El archivo XML

Está situado en "/tmp/values.xml" y será accedido y entregado al browser cliente por el CGI. El archivo values.xml tiene esta forma:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
      <values>
            <analog>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
                  <ch>x.xxxx</ch>
            </analog>
      </values>
```

La página HTML

La parte estática de la página web se basa prácticamente en una tabla que es construida “al vuelo” con JavaScript:

```
<script language="javascript" type="text/javascript">
      for(var z=0; z<8; z++)
            document.write(   "<tr>"+
                              "<td class=\"data\"><a>An"+z+"</a></td>"+
                              "<td class=\"data\"><a
name=\"analog\"></a></td>"+
                              "<td class=\"data\"><a>V</a></td>"+
                              "</tr>");
</script>
```

Así se definen 8 elementos <a> con la propiedad “name” establecida en “analog”.
Esta estructura será referenciada como se verá más adelante.

AJAX

Mediante AJAX se solicita el CGI pasándole como parámetro la ruta del archivo /tmp/values.xml. Esta solicitud se realiza cada 1 segundo de acuerdo a un timer definido en nuestro código JavaScript. La recepción del archivo XML por parte del objeto CGI se maneja en forma asíncrona. Y luego se muestran los valores actualizandolos sin requerirse la intervención del operador y sin refrescos de página.
Cuando la página HTML es cargada por primera y única vez, se ejecuta un primer llamado a la función de JavaScript que realiza todo el trabajo con implementación de AJAX incluido. Esto es, tras el evento **onload** del **body** ejecutar la función **ajaxFunction():**

```
<BODY bgColor="#ffffff"  onload="ajaxFunction();">
```

Y la función **ajaxFunction()** está definida como sigue (por claridad omitimos código que suele utilizarse habitualmente para control de errores, puede verse el código completo en los archivos del proyecto):

```
function ajaxFunction()
{           ...
            ...
      //instanciación del objeto que implementa el comportamiento más
      //importante utilizado con AJAX (XMLHttpRequest)
            xmlHttp = new XMLHttpRequest();
            ...
            ...
      // ".onreadystatechange" es el evento del objeto instanciado
      // anteriormente que informa sobre cambios en el estado del proceso
      // de comunicaciones y a este evento se le asigna como puede verse,
      // la funcionalidad deseada.
      xmlHttp.onreadystatechange    =
      function()
      {
      // Si hubo cambios y la propiedad ".readyState" es igual a 4,
      // entonces hay datos listos para procesar:
            if(xmlHttp.readyState==4)
            {//recepción del archivo
                  var cgiStr = xmlHttp.responseText;
                  var ini = cgiStr.indexOf("<?");          //Extracción
                  var end = cgiStr.indexOf("</pre>") ;     //de código
                  cgiStr = cgiStr.substring(ini, end);     //XML

                  var parser = new DOMParser();       //Parsing del XML
```

```
                //Extracción de datos
                var doc = parser.parseFromString( cgiStr ,'text/xml');
                var xmlCh = doc.getElementsByTagName("ch");
                var htmlCh = document.getElementsByName("analog");

                //Asignación de datos
                for(var i=0; i<=7; i++)
                      htmlCh[i].innerHTML =
                      xmlCh[i].childNodes[0].nodeValue;
                // Fecha y hora
                document.getElementsByName("datetime")[0].innerHTML  =
                getTime();
        }
}
//Requerimiento GET solicitando el objeto CGI con el nombre del
//archivo XML como parámetro.
xmlHttp.open("GET","/cgi-bin/display?file=%2Ftmp%2Fvalues.xml",
             true);
xmlHttp.send(null);

//Recursividad para generar el refresco de los datos 1 segundo más
//tarde
timer=setTimeout('ajaxFunction()',1000);

}
```

Otros objetos de la interfaz web

Para completar la interfaz web contamos además de los objetos descriptos de la hoja de estilos en cascada (.css) y de un archivo de imágen para el logo.

Proyecto del RootFS

La etapa final de este desarrollo es incluir los elemento generados en la estructura de archivos del sistema linux embebido para que todo trabaje en conjunto. Los siguientes son los archivos junto con su ruta de instalación correspondiente, así que solo hay que copiar esos archivos en el directorio adecuado dentro de la carpeta de nuestro proyecto rootfs:

- El ejecutable para lectura del ADC por SPI:
  /usr/share/www/cgi-bin/spidev_test_c
- La página web junto con la hoja de estilos y logo:
  /usr/share/www/spi_demo.html
  /usr/share/www/style.css
  /usr/share/www/conti.gif
- Un archivo xml inicial en (solo con fines documentales, ya que es generado por la app):
  /tmp/values.xml
- El script de autoejecución para iniciar la aplicación luego del booteo:
  /etc/init.d/S100spidemo.sh

Este script, además de la línea que indica cual es el intérprete de comandos:

```
#!/bin/sh
```

incluye solo una línea adicional para ejecutar automáticamente la aplicación de lectura del ADC:

```
./usr/share/www/cgi-bin/spidev_test_c &
```

Finalmente se modifica el archivo **/configs/add_files.sh** esto es para agregar comandos que se ejecutan antes de la compilación permitiendo copiar los archivos mencionados a la ruta de exportación del directorio NFS (esto es más que nada con fines de depuración). Ej:

```
cp ${ROOTFS_DIR}/usr/share/www/spi_demo.html   ${DEL_NFS_DIR}/usr/share/www/spi_demo.html
cp ${ROOTFS_DIR}/usr/share/www/conti.gif       ${DEL_NFS_DIR}/usr/share/www/conti.gif
cp ${ROOTFS_DIR}/usr/share/www/style.css       ${DEL_NFS_DIR}/usr/share/www/style.css
cp ${ROOTFS_DIR}/usr/share/www/cgi-bin/spidev_test_c
                                  ${DEL_NFS_DIR}/usr/share/www/cgi-bin/spidev_test_c
cp ${ROOTFS_DIR}/tmp/values.xml               ${DEL_NFS_DIR}/tmp/values.xml
```

Nota:
Todos los arhivos que se agregan al rootfs así como los que ya existen pero deben modificarse están resguardados en el directorio /iweb_demo. Este es un directorio creado como back-up para evitar posibles pérdidad de archivos ante comandos “rebuild”.

Prueba de la aplicación

La puesta en marcha de la aplicación puede efectuarse en modo de desarrollo (1), o en modo de producción(2).
1) El modo de desarrollo sería haciendo bootear al módulo vía TFTP con el comando del bootloader `dboot linux tftp` lo cual extrae la imagen del kernel desde el host, y utiliza el file system via NFS.
2) El modo de producción, sería grabando las imágenes generadas del kernel y el file system en el módulo, esto también es via TFTP, pero usando los comandos del bootloader:

```
update linux tftp
update rootfs tftp
```

En ambos casos se asume la correcta compilación e instalación de las imagenes del kernel y el rootfs en el IDE que se ejecuta en el host, esto es, haber obtenido resultados libres de errores luego de ejecutar los comandos “buil” e ”install” sobre sendos proyectos.

Una vez que el sistema esté ejecutandose, podemos ver la aplicación funcionando desde cualquier explorador apuntando a la dirección: `http://[IP_CME9210]/spi_demo.html`
También podemos abrir una terminal Telnet contra el módulo y comprobar repetidamente como cambia el archivo XML (`/tmp/values.xml`) según las lecturas que se van tomando desde el ADC. Desde una consola, esto sería:

```
telnet [IP_CME9210]
```

y después, al ejecutar repetidas veces la siguiente línea veríamos cambiar el archivo de datos:

```
cat /tmp/values.xml
```