
	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-015
		Publicado: 00/00/0000
		Página 1 de 15

Revisión	Fecha	Comentario	Autor
0	21/12/2009		Ulises Bigliati

ÍNDICE

INTRODUCCIÓN	2
Objetivos	2
Fuera de alcance	2
EL HARDWARE	3
El módulo de procesamiento central	3
Circuito de soporte del bus 1-Wire	4
EL SOFTWARE	4
Arquitectura de red 1-Wire diseñada para el proyecto	4
Desarrollo de la interfaz web de la aplicación	6
Creación de enlaces de HTML a las variables de nuestra aplicación embebida.....	6
Generación del código fuente C mediante el compilador PBuilder.....	6
Utilización de las rutinas de stub.....	7
Compilación y linkeado.....	7
Aspectos prácticos del desarrollo de la interfaz web	7
Estructura de archivos web del proyecto	7
Contenido de PBuilder.pbb:.....	9
Ejecución de PBuilder.exe.....	9
Resultados de la compilación HTML -> ANSI C.....	10
La versión ANSI C de values.xml (parte estática).....	10
El módulo “_v.c” con las funciones stub (parte dinámica).....	11
El programa principal	13
El resultado final, la interfaz para el usuario.....	14
CONSIDERACIONES FINALES	15
Hardware	15
Connect ME 9210.....	15
iButtons.....	15
Software:	15
NET+OS.....	15
API's 1-Wire.....	15

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 2 de 15

Introducción

En esta ocasión nos encontramos en situación de probar el nuevo módulo para desarrollo de Digi, el Connect ME 9210. Se trata de la evolución del ya conocido Connect ME popularmente utilizado en su versión Plug&Play que en este caso viene equipado con un microprocesador ARM9 @75Mhz, es pin a pin compatible con su predecesor e incluye interfaz JTAG para sus versiones de desarrollo mientras que en sus versiones de producción no se incluye esta interfaz.

Como una buena excusa para el proyecto de demostración elegimos realizar la implementación completa del protocolo 1-Wire. Particularmente para la lectura de iButtons sensores de temperatura.

Dicho proyecto se estructura mediante dos ejes:

- La construcción de un bus de iButtons que admita cambios dinámicos, es decir, un número variable de iButtons que pueden conectarse y desconectarse del bus en cualquier momento.
- El diseño de una interfaz web capaz de monitorear en tiempo real el estado del bus y de graficar, también en tiempo real los valores de temperatura obtenidos de los dispositivos seleccionados.

Objetivos

- Exponer los aspectos mas relevantes del desarrollo de interfaces web dinámicas utilizando el entorno de desarrollo el NET+OS de Digi.
- Testear el nuevo hardware de Digi Connect ME 9210¹.
- Implementar una bus 1-Wire que admita la conexión y desconexión de n dispositivos en forma dinámica.

Fuera de alcance


- No es objeto de esta nota explicar fundamentos teóricos ni especificaciones del estándar 1-Wire² para la utilización de los iButtons de Maxim.
- Tampoco lo es el abordaje de la descripción del lenguaje de scripting³ del lado del servidor empleado por el entorno de desarrollo NET+OS.
- Queda fuera del alcance de esta nota detalles de la operatoria del S.O. ThreadX utilizado como núcleo del entorno NET+OS⁴.
- Los detalles de la programación web utilizada (Javascript, AJAX, CSS, Flash, etc) fueron convenientemente abordados en notas de aplicación previas (CoAN-013 y otras) por lo tanto dichos conceptos no serán repetidos en el presente trabajo.

¹ Para obtener detalles sobre el hardware consultar el manual de referencia del fabricante en el siguiente enlace: http://ftp1.digi.com/support/documentation/90000897_G.pdf

² A estos efectos remitirse a la documentación del fabricante, fundamentalmente "App Note 937 Book of iButton® Standards". Y en general en el link <http://www.maxim-ic.com/products/ibutton/> secciones **Software Resources** y **Technical Support**.

³ A estos efectos se debe recurrir a la documentación provista por Digi en su manual de referencia de programación "Advanced Web Server Toolkit" que puede encontrarse dentro de la instalación del NET+OS o en el siguiente enlace: http://ftp1.digi.com/support/documentation/90000684_H.pdf

⁴ Para mayores detalles al respecto consultar nuestra nota "CoAN-002" y la documentación del fabricante "Programmer's Guide" y "Kernel Guide" (90000785_E y 90000788_A)

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-015
		Publicado: 00/00/0000
		Página 3 de 15

El hardware

El módulo de procesamiento central

El Connect ME 9210 está basado en ARM9 @75Mhz con 2 o 4 MB de NOR flash y 8 MB SDRAM; es la evolución del tradicional Connect ME y ambos modelos son pin a pin compatibles. El C-ME, basado en ARM7 @55Mhz, es típicamente utilizado en su versión Plug&Play como un elemento de co-procesamiento. Pero el C-ME 9210 ha pasado a ser un dispositivo de procesamiento central en el cual podremos depositar la responsabilidad del control completo de una aplicación para la cual lo consideremos adecuado.

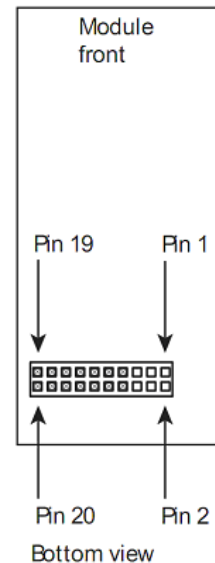
El C-ME 9210 cuenta con interfaz Ethernet 10/100Base-T y 10 GPIO entre los cuales se comparten:

- 1 UART con todas sus líneas de hand-shake (TXD, RXD, RTS, CTS, DTR, DSR y DCD)
- Hasta 3 entradas de interrupciones externas
- I2C, SPI y las nuevas FIM (Flexible Interface Modules) que a su vez pueden implementar alguna de las siguientes interfaces:
 - UART
 - 1-Wire
 - USB device (próximamente)
 - CAN (próximamente)

Estas FIM son periféricos de apoyo al procesador central que pueden configurarse para actuar según lo desee el usuario conforme a alguna de las interfaces listadas arriba y están basadas en microcontroladores DRPIC165X @300Mhz, sus pines de I/O se multiplexan con las líneas de GPIO.

A continuación puede apreciarse el pin-out del módulo:


Pin	UART [AB]	GPIO [ME/ W-ME]	GPIO [ME 9210]	Ext IRQ [ME/ W-ME]	Ext IRQ [ME 9210]	I2C [ME 9210]	SPI [ME 9210]	FIM [ME 9210]	Timer [ME 9210]	Other [AB]
1										VETH+
2										VETH-
3-6	Positions Removed									
7	RXD	A3	GPIO[3]				DATA IN	PIC [3]		
8	TXD	A7	GPIO[7]				DATA OUT		Timer Out 7 Timer In 8	
9	RTS	A5	GPIO[5]		3		CLK		Timer Out 6	
10	DTR	A6	GPIO[6]						Timer In 7	
11	CTS	A1	GPIO[1]		0			PIC[1]		
12	DSR	A2	GPIO[2]		1			PIC[2]		
13	DCD	A0	GPIO[0]				EN	PIC[0]		
14										/RST
15										3.3V
16										GND
17		C4	GPIO[12]			SDA	CLK			RESET_ DONE
18		C1	GPIO[9]	1	0	SCL				
19	Reserved									
20		C5	GPIO [13]				CLK		Timer Out 9	/INIT



La versión para desarrollo incluye además una interfaz JTAG que no es necesaria en sus versiones de producción, ya que la imagen del firmware puede transferirse vía Ethernet.

Otra característica interesante que incluyen estos módulos es la capacidad de funcionamiento en modos de bajo consumo:

- On-the-fly clock scaling
- Modo sleep
- Eventos de scaling/wake-up configurables (IRQ externa, UART, Ethernet, etc.)

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 4 de 15

Por ejemplo, mientras el consumo típico es de 346mA, en modo de baja velocidad (/16 clock scaling) puede bajar a 186 mA. Y en modo sleep, con Wake-up por IRQ externa y Ethernet desactivada puede rondar por los 34 mA.

Circuito de soporte del bus 1-Wire

Volviéndonos hacia los requerimientos de nuestra aplicación, vamos a analizar cuáles serían los componentes electrónicos indispensables para implementar nuestra red basada en el protocolo 1-Wire que, como su nombre lo indica, funciona sobre un solo hilo conductor (mas el contacto de tierra).

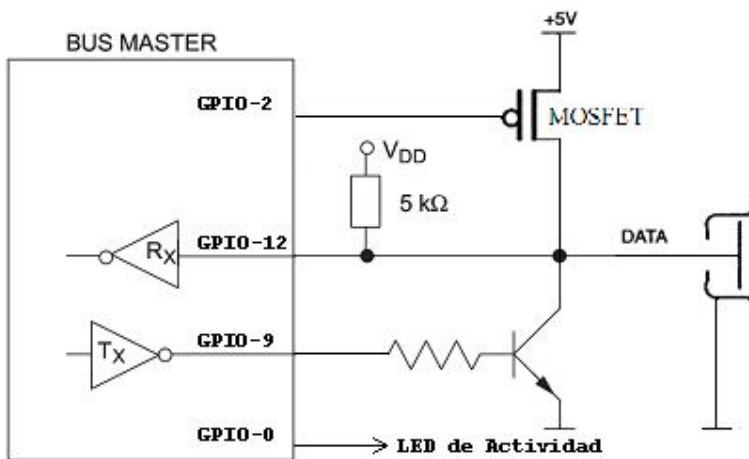
En el caso en que dispusiéramos de puertos de entrada/salida con la capacidad de funcionar en forma bidireccional sin configuración previa (tal como en el caso de la arquitectura 8051) sólo haría falta una resistencia de pull-up conectada a una única línea de I/O que oficiaría como bus 1-Wire.

Pero como en nuestro caso no tenemos esa posibilidad debemos utilizar dos pines de I/O uno como entrada (GPIO12) conectado a un pull-up y directamente al bus 1-Wire y otro como salida (GPIO9) conectado a la base de un transistor de uso general cuyo colector está conectado directamente a la línea de datos.

Por otra parte tenemos la necesidad de agregar un transistor MOSFET que funcionará a modo de strong pull-up activado por la línea GPIO2. Esto es necesario para admitir la operación de iButtons del tipo DS1920 que son sensores de temperatura que requieren de un flujo de corriente mayor durante el tiempo que necesitan para realizar las mediciones.

Adicionalmente hemos previsto una salida (GPIO0) para manejar un LED de monitoreo de actividad.

A continuación puede verse el esquema de conexión necesario para el proyecto:




El software

Arquitectura de red 1-Wire diseñada para el proyecto

El protocolo 1-Wire de Dallas-Maxim se estructura según una arquitectura de capas conforme al modelo OSI. Para facilitarnos su implementación el fabricante nos provee en forma gratuita una serie de archivos fuente que constituyen una interfaz de programación que nos permite incorporar el protocolo completo en nuestro diseño, lo cual podremos hacer seguramente tras no pocas modificaciones adaptativas (al menos así fue en nuestro caso), pero en líneas generales esta biblioteca de funciones nos ahorrará mucho tiempo y horas de lectura.

Volviendo a nuestro proyecto, a continuación puede apreciarse la arquitectura que resultó de la utilización de las API's de Dallas⁵. Si bien se ha desarrollado con NET+OS 7.4 de Digi, un entorno de desarrollo basado en un RTOS, el código fuente esquematizado abajo podrá ser incorporado a cualquier proyecto ANSI C, ya


⁵ <http://www.maxim-ic.com/products/ibutton/software/1wire/wirekit.cfm>

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 5 de 15

que la programación está realizada íntegramente en dicho lenguaje y solo será necesario hacer los ajustes pertinentes de las rutinas de más bajo nivel, particularmente en relación con las temporizaciones, la inhibición de interrupciones y el acceso a los puertos de I/O.



* *iButton.h* es invocado por todos los demás fuentes ya que contiene definiciones y macros compartidas por todo el código relacionado con iButtons. Solo por claridad se omitió la representación gráfica.

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-015
		Publicado: 00/00/0000
		Página 6 de 15

Desarrollo de la interfaz web de la aplicación

Nuestro proyecto se compone de dos bloques fundamentales, por un lado obtenemos el manejo de una red 1-Wire, y por el otro la presentación de datos en tiempo real obtenidos de aquella red en forma dinámica vía interfaz web; para lograr la implementación de esto último *NET+OS* nos proporciona un servidor web basado en un motor llamado *RomPager*⁶ y presenta un Toolkit para asistirnos en el desarrollo de aplicaciones web avanzadas. Este toolkit se compone fundamentalmente de un compilador llamado *PBuilder*⁷, y de la documentación que define el lenguaje de scripting⁸ del lado servidor utilizado por *RomPager*.

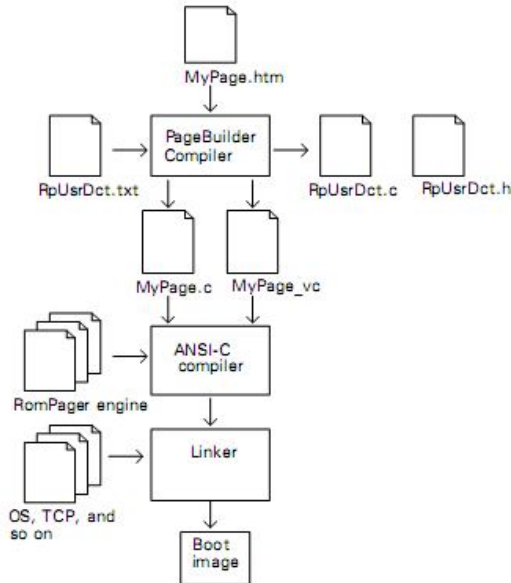
Creación de enlaces de HTML a las variables de nuestra aplicación embebida

Para vincular nuestra interfaz web a las variables generadas por nuestra aplicación debemos utilizar el lenguaje de scripting del servidor. Entonces, para manipular variables desde (y hacia) nuestra interfaz web, debemos insertar convenientemente ciertos tags especiales de comentario HTML en nuestros archivos web, según se definen en la referencia de programación web (nota al pie n^o7).

Una vez creadas las páginas y demás archivos con contenido dinámico, estas se procesan mediante el compilador *PBuilder* para generar la versión ANSI C de aquellas páginas, y crear las estructuras de código necesarias para que el servidor web *RomPager* sea capaz de acceder a las variables de nuestra aplicación cuando las páginas web son solicitadas por el browser.

Generación del código fuente C mediante el compilador PBuilder

El próximo paso será pasar las páginas HTML a través del compilador *PBuilder* para producir los archivos fuente en lenguaje C. *PBuilder.exe* normalmente se encuentra en el directorio /bin dentro de la instalación del *NET+OS*. Es conveniente copiar este ejecutable al directorio web del proyecto. A continuación vemos en la figura⁹ el flujo de archivos que se genera a través del proceso de compilado y linkeado:



PBuilder usa los archivos web como entrada para generar dos archivos de salida:

- El fuente en C que representa el formato de almacenamiento interno de las páginas HTML (y otros elementos web) que además contiene las estructuras para las variables dinámicas y estáticas (*MyPage.c*).
- Un set de rutinas de stub que permiten el enlace de la interfaz web con nuestra aplicación (*MyPage_vc*).

Procesamiento batch con archivo .pbb:

Para facilitar la compilación de mas de un archivo HTML, (o elemento web en general) en vez de generalos uno a uno se utiliza el archivo batch (.pbb) . *PBuilder* accederá a este archivo y al recorrerlo procesará secuencialmente a todos los elementos web de la aplicación.

Cabe destacar que el compilador utiliza un diccionario interno para comprimir el HTML de uso


común y uno opcional provisto por el usuario (*RpUsrDct.txt*) .

⁶ Producto de un tercero llamado *Allegro Software Development Corporation*

⁷ Se encuentra dentro del directorio /bin, dentro de la instalación del *NET+OS*.

⁸ Referencia de programación *Advanced Webserver Toolkit* que se encuentra entre la documentación del *NET+OS*.

⁹ Figura obtenida de la documentación anteriormente citada.

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 7 de 15

Utilización de las rutinas de stub

Una vez procesados todos los elementos web mediante el compilador *PageBuilder (PBuilder.exe)* podremos disponer de las rutinas de stub que nos facilitarán el acceso a las variables que cada página necesite. Dichas funciones se encuentran dentro de los archivos "mypage_v.c" creados respectivamente por cada archivo "mypage.*" (no necesariamente tienen que ser un página HTML). La tarea restante es escribir el cuerpo de esas funciones para acceder a las variables que nos interesan de nuestra aplicación principal y así exponerlas ante la interfaz web. Por supuesto, en los casos en que las citadas rutinas no se necesitan pueden desecharse.

Compilación y linkeado

El paso final será la compilación y linkeado habituales, dado que el compilador PBuilder genera ANSI C a partir de nuestras páginas y otros elementos web, podremos realizar estas tareas con cualquier compilador estándar generando una imagen de nuestra aplicación embebida que incluirá la porción web del proyecto.

Aspectos prácticos del desarrollo de la interfaz web

A fines de evitar excesivas demoras producto de la recompilación y linkeado de todo el sitio web junto con la aplicación principal mas el S.O. cada vez que el sitio cambia, recomendamos que el desarrollo del sitio web que se desee embeber en un proyecto sea realizado en su mayor parte en forma externa al NET+OS con cualquier editor de contenido web, es decir, sería recomendable replicar el directorio web del proyecto para trabajar sobre este en forma totalmente independiente del NET+OS al menos hasta que el diseño y desarrollo del sitio estén lo suficientemente avanzados.

En forma posterior a la creación del sitio, una vez identificado el contenido dinámico, se debería comenzar a insertar las porciones de código de script que correrán del lado del servidor (*tags* de comentarios de *RomPager*)¹⁰.

A continuación, lo más razonable sería crear el archivo que auxiliará a PBuilder para compilar secuencialmente todos los elementos de nuestro sitio, es decir el archivo con extensión .pbb que, como se verá a continuación, contiene una lista de todos los elementos a compilar.

Por supuesto, debemos copiar el compilador PBuilder.exe al directorio de trabajo y cada vez que sea necesario realizaremos la compilación del sitio web en este directorio auxiliar.

Una vez compilado el proyecto web podremos transferir su contenido resultante dentro del proyecto NET+OS (esto quedará mucho más claro al examinar la estructura de archivos).

En este punto se debe comenzar a definir las funciones de stub que generarán el contenido dinámico del sitio. Finalmente se compila, linkea y prueba todo el proyecto en conjunto.

Importante:

Es una buena práctica organizar el contenido del sitio según los tipos de archivo; así, dentro del directorio "/web" tendremos el directorio "/css", "/js", "/html", etc.

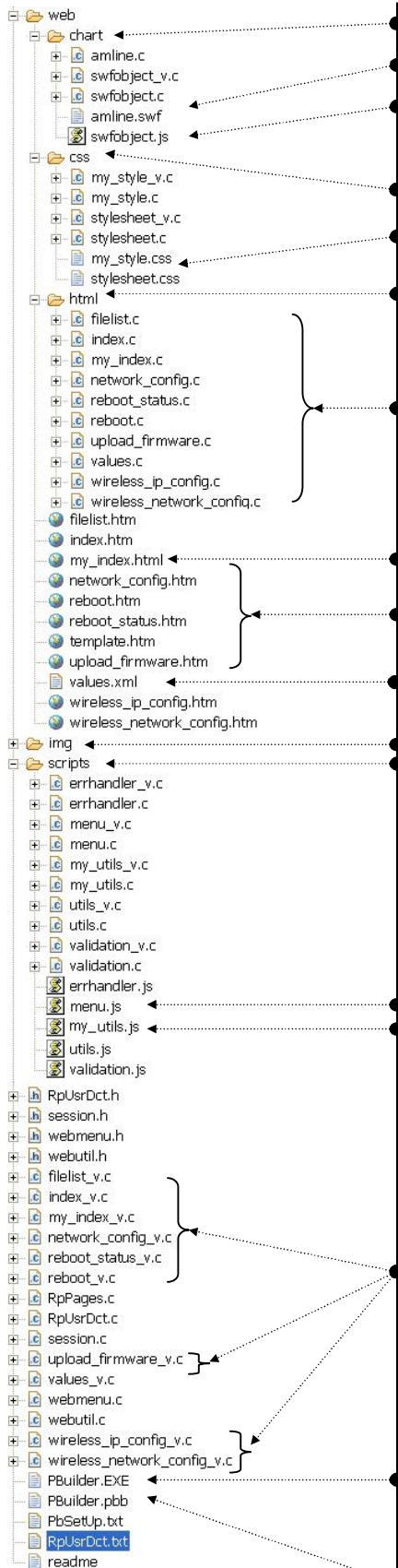
Como resultado de esta organización, cuando PBuilder cree las versiones ".c" y "_v.c" de cada elemento web indicado en el archivo batch ".pbb" depositará esos archivos junto a cada elemento original.


Si aquellos archivos (.c" y "_v.c") ya existían, PBuilder los sobrescribirá con la nueva versión, lo cual no es un problema para los archivos de contenido estático, pero si lo es para los de contenido dinámico. Por esta razón, es muy recomendable mover a los archivos de stub (mypage_v.c) al directorio web raíz ("/web") de lo contrario, con cada compilación de PBuilder se sobrescribirían las rutinas de stub que ya hemos modificado con la versión *vacía* de estas, lo cual no sería nada agradable. Finalmente, se debería eliminar todo archivo "_v.c" duplicado que no nos interese, ya que de lo contrario obtendremos varios errores durante la compilación.

Estructura de archivos web del proyecto

A continuación veremos como se estructura lógica y físicamente en términos de archivos la interfaz web de nuestra aplicación, con lo cual podremos darnos una mejor idea de cómo se trabaja en este aspecto sobre el entorno de desarrollo NET+OS que como puede verse, para el desarrollo de aplicaciones web nos proporciona un set de herramientas de terceros.

¹⁰ Ver documentación (Advanced Web Server Toolkit)



	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 9 de 15

Contenido de PBuilder.pbb:

A continuación podemos apreciar el contenido del archivo de texto que presenta el listado de elementos web (páginas HTML, archivos Java Script, archivos XML, etc, etc.) que necesitamos procesar con el compilador PBuilder.exe.

Al iniciar un proyecto típico de NET+OS con interfaz web obtendremos una estructura web predeterminada (que de ser necesario podremos modificar) cuyos elementos estarán listados en el archivo “.pbb” según puede verse a continuación señalados por la primer llave.

Seguidamente debemos continuar la lista con las referencias a los archivos propios de la aplicación que estamos construyendo, lo cual está señalado por la última llave.

```
html/index.htm
html/network_config.htm
html/wireless_network_config.htm
html/wireless_ip_config.htm
html/upload_firmware.htm
html/reboot.htm
html/reboot_status.htm
html/filelist.htm
css/stylesheet.css
img/logo.gif
scripts/errhandler.js
scripts/utils.js
scripts/validation.js
```

Archivos a compilar predefinidos por el entorno NET+OS para la interfaz web de un proyecto típico.

```
css/my_style.css
scripts/my_utils.js
scripts/menu.js
html/my_index.html
html/values.xml
chart/amline.swf
chart/swfobject.js
```

Referencias a los archivos específicos de nuestra aplicación que debemos agregar para que el compilador PBuilder genere a partir de nuestros elementos web referenciados el código C necesario para el compilador ANSI C estándar.

Ejecución de PBuilder.exe

Una vez que hayamos completado el archivo PBuilder.pbb y que estemos en condiciones de comenzar a probar nuestras páginas web nos resta ejecutar el compilador PBuilder.exe. Haremos esto desde línea de comandos, ubicándonos dentro del directorio que aloja a los elementos web del proyecto y al ejecutable PBuilder propiamente dicho, que tal como se indicó más arriba, fue copiado en el directorio web para facilitar el proceso de compilación durante el desarrollo y normalmente podremos encontrarlo en el directorio “bin” dentro de la instalación del NET+OS.


Como ejemplo de lo mencionado, desde un sistema operativo Windows podríamos hacer lo siguiente:

- 1) Ir a “Inicio->Ejecutar”
- 2) Típear “cmd” y presionar “Aceptar”.
- 3) En la ventana de terminal escribiríamos algo parecido a: “cd PATH” siendo PATH la ruta del directorio web de nuestro proyecto en la cual se encuentra el compilador PBuilder y presionamos [ENTER]. Por ejemplo: cd C:\DigiProjects\workspace\OneWireTest\web\ [ENTER]
- 4) Finalmente, tipeamos “PBuilder” en la línea de comandos y presionamos [ENTER].

Así el compilador revisará el listado definido en el archivo PBuilder.pbb e irá compilando uno a uno cada archivo, en caso de encontrar algún error lo reportará indicando el número de línea donde fue generado, de lo contrario, al finalizar el proceso obtendremos la estructura de archivos de código fuente en C que permitirá vincular la interfaz web con la aplicación principal.

Cabe mencionar que el descripto es un proceso iterativo, que repetiremos seguramente muchas veces durante el desarrollo, por lo tanto para evitar repetir cada vez los pasos descriptos podría ser conveniente generar un archivo “.bat” tal como se ve en el siguiente ejemplo:

```
@echo off
cd "C:\DigiProjects\workspace\OneWireTest\web\"
PBuilder
pause
exit
```

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 10 de 15

Resultados de la compilación HTML -> ANSI C

Anteriormente fue descrita en forma genérica la estructura de archivos resultante del proceso de compilación de los elementos HTML.

Ahora veamos un ejemplo concreto que nos dará una idea más acabada de la forma de trabajar con NET+OS y con proyectos que involucren interfaces web. Por su simplicidad tomamos como ejemplo el archivo XML que utilizamos en nuestra aplicación como transporte de datos entre el núcleo del programa y la interfaz web a desplegarse en el browser, esto es, el archivo "values.xml" que físicamente luce como puede verse en la *figura 1* en tiempo de desarrollo, es decir que así lo vemos en el editor del NET+OS:

Archivo "values.xml" Figura 1

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?access-control allow="*"?>
<values>
  <!-- RpNamedDisplayText Name=temp
RpTextType=ASCII
RpGetType=Function
RpGetPtr=GetTemp -->
  <!-- RpEnd -->
<dt>
  <!-- RpNamedDisplayText Name=time
RpTextType=ASCII
RpGetType=Function
RpGetPtr=GetTime -->
  <!-- RpEnd -->
</dt>
</values>
```

Remarcado en azul se encuentra la porción de código correspondiente al lenguaje de script que el compilador *PageBuilder (PBuilder.exe)* sabrá interpretar para generar a partir de este el código fuente en lenguaje C a partir del cual el servidor web *RomPager* sabrá generar en forma dinámica los datos que necesitamos para alimentar a nuestra aplicación web.

A continuación, en la *figura 2*, vemos en su versión de ejecución, al mismo archivo en la forma en que se presenta ante el explorador web para ser procesado para extraer los datos que el usuario final requiere.

Nótese que todo el código de script fue reemplazado por los datos que en verdad son la razón de ser de todo el proceso, es decir los números de serie de los dispositivos presentes en el bus junto con la temperatura que reportan obtenida en tiempo real, todo esto generado dinámicamente, pues los dispositivos pueden ser agregados o quitados del bus en cualquier momento. El proceso de obtención y suministro de los datos para conformar este archivo se realiza gracias al código intermedio que analizaremos a continuación, las versiones "values.c" y "values_v.c" del archivo original de la *figura 1*.


Archivo "values.xml" Figura 2

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?access-control allow="*"?>
<values>
  <ib>
    <n>012B45E6A89CF</n>
    <t>24.65</t>
  </ib>
  <ib>
    <n>012B45E6A89CA</n>
    <t>24.65</t>
  </ib>
  <dt> 14/12/09 10:12:58 </dt>
</values>
```

La versión ANSI C de values.xml (parte estática)

En la figura 3 podemos observar la estructura de la versión "values.c" del archivo XML generada por *PBuilder*. En dicho archivo se encuentra el código que debe ser transmitido textualmente por el servidor web *RomPager* y también se encuentran las referencias internas necesarias para que el servidor web sepa cómo generar los datos que son variables, en este caso estamos hablando de los datos definidos por el lenguaje de script que está en azul en la *figura 3*. Nótese que en líneas generales lo que se está haciendo es generar una tabla conteniendo por un lado los datos estáticos y por otro las referencias a las funciones que devolverán los datos que son dinámicos. *PBuilder* realiza esta tarea por cada elemento web que esté definido en el sistema.

Es destacable que si bien es importante conocer la existencia de este archivo, este no es relevante desde el punto de vista del programador, que realmente puede prescindir (y es recomendable que así lo haga) de su manipulación.

 CONTINEA <small>Microprocesamiento modular + Conectividad</small>	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 11 de 15

Archivo "values.c"

Figura 3

```

/* Created with PageBuilder version 4.04 on Wed Dec 16 10:28:09 2009 */
#include "AsExtern.h"
#if RomPagerServer
/* ***** */
/* *      Built from "html\values.xml"      * */
/* ***** */
extern rpObjectDescription Pgvalues;
static char Pgvalues_Item_1[] =
"<?xml version='1.0' encoding='ISO-8859-1'?>\n"
"<?access-control allow='*'?>\n"
"<values>\n";
extern char *GetTemp(void);
static rpNamedTextDisplayItem Pgvalues_Item_2 = {
"temp",
(void *) GetTemp,
eRpVarType_Function,
eRpTextType_ASCII,
20};
static char Pgvalues_Item_3[] =
"<dt>\n";
extern char *GetTime(void);
static rpNamedTextDisplayItem Pgvalues_Item_4 = {
"time",
(void *) GetTime,
eRpVarType_Function,
eRpTextType_ASCII,
20};
static char Pgvalues_Item_5[] =
"</dt>\n"
"</values>\n";
static rpItem Pgvalues_Items[] = {
{ eRpItemType_DataZeroTerminated, (void *) &Pgvalues_Item_1 },
{ eRpItemType_NamedDisplayText, (void *) &Pgvalues_Item_2 },
{ eRpItemType_DataZeroTerminated, (void *) &Pgvalues_Item_3 },
{ eRpItemType_NamedDisplayText, (void *) &Pgvalues_Item_4 },
{ eRpItemType_DataZeroTerminated, (void *) &Pgvalues_Item_5 },
{ eRpItemType_LastItemInList }
}

```

El módulo "_v.c" con las funciones stub (parte dinámica)

Los archivos "_v.c" son la pieza central de la generación dinámica de contenido web del lado de servidor dentro del entorno de desarrollo del NET+OS. Estos archivos, también son generados tras el procesamiento de cada elemento web por parte del compilador *PBuilder* y contienen la estructura necesaria para que el servidor web RomPager pueda vincularse con el núcleo de nuestra aplicación. Esta vinculación se realiza mediante la definición de funciones de stub. En la figura 4 puede verse la estructura que corresponde al archivo *values_v.c* tal como la genera *PBuilder* tras procesar al archivo *values.xml*.

Archivo "values_v.c"


Figura 4

```

/* Created with PageBuilder version 4.04 on Mon Dec 7 15:33:58 2009 */
#include "AsExtern.h"
#if RomPagerServer
/* ***** */
/* *      Built from "html\values.xml"      * */
/* ***** */
extern char *GetTemp(void);
char *GetTemp(void) {
static char result = 0;
return &result;
}

extern char *GetTime(void);
char *GetTime(void) {
static char result = 0;
return &result;
}
#endif /* RomPagerServer */;

```

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 12 de 15

Nótese la existencia de dos funciones (`GetTemp` y `GetTime`) que respectivamente devuelven un puntero a caracter que será utilizado para *llenar* el espacio *indicado* por los scripts remarcados en azul en la *figura 1*. Finalmente, la tarea del programador consiste en escribir el código correspondiente al cuerpo de esas dos funciones y en agregar los elementos necesarios a fines de devolver convenientemente los datos que dinámicamente son adquiridos y generados por el núcleo de la aplicación. A continuación se muestra el archivo anterior pero modificado de acuerdo a las necesidades particulares del proyecto que estamos desarrollando, es decir, el monitoreo en tiempo real de un bus 1-Wire :

```

/* Created with PageBuilder version 4.04 on Mon Dec 7 15:33:58 2009 */
#include "AsExtern.h"

#include "nsuptime.h"
#include "ibutton.h"
#include "ibThermo.h"
#if RomPagerServer
/* ***** */
/* * Built from "html\values.xml" * */
/* ***** */
static char buffer[MAXDEVICES * 25]={0};

extern char *GetTemp(void);
char *GetTemp(void)
{
    int i, dev=getCurNumDevices(),len=0;
    char aux[17];

    buffer[0] = 0;
    for(i=0; i<dev ;i++ )
    {
        getNs(i, aux);
        len=sprintf(&buffer[len], "<ib>\n\t<n>%s
[%s]</n>\n\t<t>%5.2f</t>\n</ib>\n",aux, getDevType(i),getSample(i));
        len= strlen(buffer);
    }
    return buffer;
}


static char mytime[20];
extern char *GetTime(void);
char *GetTime(void)
{
    unsigned long days=0, hours=0, minutes=0, seconds = ns_get_uptime_sec();
    if((minutes = (seconds / 60)) != 0){
        if((hours = (minutes / 60)) != 0){
            if((days = (hours / 24)) != 0){
                hours = (hours % 24);
            }
            minutes = (minutes % 60);
        }
        seconds = (seconds % 60);
    }
    sprintf(mytime,"%02lu d %02lu h %02lu m %02lu s\n",days, hours, minutes, seconds);

    return mytime;
}

#endif /* RomPagerServer */

```

Nótese principalmente la inclusión de los headers necesarios (`ibutton.h`, `ibThermo.h`, etc.) para enlazarse con la aplicación principal, la declaración de buffers de intercambio (`mytime[]`, `buffer[]`) y finalmente el cuerpo de las funciones que tras obtener los datos deseados y escribir con ellos los buffers, devuelven una referencia a estos últimos para que el servidor web genere la estructura final del archivo, tal como se presentó al principio mediante la *figura 2*.

 CONTINEA <small>Microprocesamiento modular + Conectividad</small>	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 13 de 15

El programa principal

Nos resta hablar sobre el módulo principal que reúne todas las partes, pero, no obstante su noble tarea no hay mucho más que decir con respecto a aquel, pues lo único que se realiza desde el punto de inicio de nuestra aplicación es la inicialización del driver 1-Wire que hemos construido, y tras ello, este driver se encarga del resto, es decir, monitorear permanentemente el bus para detectar los anfitriones que se hayan conectados a aquel y de ser posible, obtener muestras de temperatura en caso de que la naturaleza del dispositivo así lo permita.

Reiteramos que, al estar todo aquello resuelto dentro del driver no queda mucho trabajo que hacer desde el programa principal (módulo "root.c") y lo único que hacemos aquí además de inicializar el driver es crear un thread para el parpadeo del LED. De todas maneras a continuación analizamos el módulo "root.c":

```

#include <stdio.h>
#include <stdlib.h>
#include <tx_api.h>
#include <appservices.h>
#include "bsp_api.h"
#include "gpiomux_def.h"
#include "ibutton.h"

static TX_THREAD thrLed;
static void led_entry (ULONG initial_input);

#define LED_PIN 0
#define LED_SET_STATE(bit) NAssetGPIOPin(LED_PIN, !bit)

#define LEAST_PRIORITY 31
#define HIGHEST_PRIORITY 0

void applicationStart (void)
{
    initAppServices();
    NAconfigureGPIOPin(LED_PIN, NA_GPIO_OUTPUT_STATE,
        BSP_GPIO_INITIAL_STATE_OUTPUT_DRIVER_LOGIC0);
    tx_thread_create(&thrLed, "Led", led_entry, 0x1234, malloc (8192), 8192,
        LEAST_PRIORITY, LEAST_PRIORITY, 1, TX_AUTO_START);


    ibDriverInit();

    return;
}

static void led_entry (ULONG initial_input)
{
    static int st=FALSE;
    while(1){
        LED_SET_STATE( (st = !st) );
        tx_thread_sleep(50);
    }
}


```

- API del S.O. ThreadX
- Header de acceso a configuración de servicios del sistema, como por ejemplo, el servidor web, TCP/IP, etc.
- API de abstracción de hardware
- Definiciones para acceder a los GPIO
- Nuestro driver 1-Wire
- Declaración de hilo para el LED.
- Declaración de función que se asociará al hilo anterior.
- Macro para encender/apagar el LED
- Inicializa servicios del sistema
- Configura GPIO para manejar el LED
- Crea e inicializa el hilo que gobernará el LED
- Inicialización del driver 1-Wire
- Definición de tarea de manejo del LED.

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 14 de 15

El resultado final, la interfaz para el usuario

Habiendo concluido con la descripción de los aspectos más relevantes de este tipo de proyectos, solo nos resta mostrar cual es el resultado obtenido para el usuario final del *producto*. La que sigue es la imagen de



la interfaz web estándar que nos presentan los módulos Digi, a la cual le agregamos un link adicional al menú (“Ibuttons Demo”) para que este nos dirija hacia la aplicación web que construimos y que puede apreciarse a continuación. En ella podemos ver los diferentes elementos que fuimos definiendo en tiempo de diseño durante el proyecto:

Hacia la izquierda se encuentra el menú desplegable Javascript que pusimos como ejemplo para una posible ampliación de la aplicación.

Le sigue el cuerpo principal de la página, en el cual, bajo el subtítulo “Listado de nodos” se muestra en forma dinámica cualquier dispositivo 1-Wire que

aparezca en el bus, con su dirección y el tipo de dispositivo, si algún dispositivo se retira del bus, esto es reflejado inmediatamente en el listado sin necesidad de ninguna intervención del usuario ni de refrescos de pantalla, esto es, recordemos, gracias a la técnica de programación web basada en AJAX que se alimenta en este caso del archivo “values.xml” el cual fue descrito en detalle más arriba.

Finalmente, ocupando el mayor espacio de la página vemos la graficación en tiempo real según el intervalo especificado y el o los dispositivos seleccionados, ya que pueden graficarse varios simultáneamente.

El componente gráfico está basado en el objeto Flash que incluimos en el proyecto, dentro del subdirectorio “/web/chart” y para la generación de los gráficos, se vale nuevamente de AJAX y del archivo “values.xml”.



The screenshot shows the 'OneWireTest' application interface. On the left is a navigation menu with options like 'Menú de ejemplo', 'Página de inicio', 'Página B', and 'Página C'. The main area is divided into two sections: 'Listado de nodos' (Node List) and a real-time chart.

Listado de nodos:

- 1044ED4D010800C4 [DS1920]
- 2135FB17000000BC [DS1921G]

Generador de gráficos en tiempo real:

Intervalo de muestreo: 2000 [Iniciar monitoreo]


Activity from selected devices over 1-Wire bus

The chart displays a line graph of activity over time. The y-axis ranges from 27.5 to 29.5. The x-axis shows time intervals: 00 d 00 h 01 m 25 s, 00 d 00 h 01 m 38 s, 00 d 00 h 01 m 51 s, and 00 d 00 h 02 m 03 s. A red line shows the activity level, which starts around 27.5, rises to a peak of approximately 29.0, and then gradually declines.

At the bottom of the chart, a checkbox is checked for the selected device: 1044ED4D010800C4 [DS1920].

At the bottom of the interface, there is a status bar with fields for 'Fecha y horas' (00 d 00 h 02 m 06 s), 'Usuario', and 'Perfil del usuario'.

Esperando 10.0.0.56...

	Evaluación de Connect ME 9210 con NET+OS	Nota de Aplicación
	Red dinámica de iButtons monitoreable vía Web.	CoAN-014
		Publicado: 00/00/0000
		Página 15 de 15

Consideraciones finales

Tras haber transitado los diferentes caminos que nos presentó este proyecto podemos extraer consideraciones de diversa índole ya que tuvimos la oportunidad de evaluar aspectos bastante heterogéneos dentro de esta aplicación.

Hardware

Connect ME 9210

- Es un potente y compacto módulo que está solo limitado por la cantidad de líneas de I/O pero puede convertirse en el procesador central de múltiples aplicaciones basadas en web. Por otra parte, la escasez de I/Os en algunos casos podría solventarse utilizando SPI o I2C, o incluso 1-Wire.
- En cuanto al costo, siendo este similar al viejo Connect ME Plug&Play, puede considerarse como atractivo desde el punto de vista de la versatilidad que ofrece el 9210 en comparación con el ME, y siendo además pin a pin compatibles se convierte en la alternativa directa para un upgrade.
- Como desventajas notamos la falta de hardware de RTC y circuitería para batería de respaldo, y hasta el momento no le encontramos demasiada utilidad al FIM, que a pesar de poseer la utilidad 1-Wire, solo ofrecía las primitivas de más bajo nivel, con lo cual tras algunos intentos infructuosos desistimos de utilizarla para recurrir a las API 1-Wire del fabricante, las cuales nos brindaron un excelente resultado.

iButtons

La implementación del bus 1-Wire ofrece muy buenas opciones de diseño ya que se trata de una interfaz cableada económica y digna de ser tenida en cuenta entre nuestras alternativas de diseño.

Software:

NET+OS

- En ocasiones se torna lento por tratarse de la convergencia de varias tecnologías y de varios fabricantes conformando un conjunto de herramientas de desarrollo: Cygwin, compilador C/C++ GNU, Eclipse, ThreadX, Allegro, Segger, el BSP de Digi, etc. Siendo esta la contrapartida del acceso a tecnologías standards y altamente probados en la industria.
- En cuanto al desarrollo web desde NET+OS notamos algo incomodo el hecho de utilizar una herramienta externa al IDE para generar la versión compilada del sitio web embebido.
- En relación con los programas de demostración y ejemplos ofrecidos por el fabricante para facilitarnos las tareas de desarrollo debemos admitir que la oferta puede ser mejorada.
- Si hablamos del diseño de una aplicación contando con el respaldo de un S.O. podemos dar el visto bueno, ya que además se presenta una API de acceso a sus servicios, poderosa y bien documentada.

API's 1-Wire

Finalmente, nos queda decir que las API de 1-Wire que utilizamos fueron muy útiles ya que nos permitieron implementar las operaciones más complejas del protocolo con poco esfuerzo, y además se encuentran bien estructuradas y documentadas. Solo hay que tomarse el trabajo de extraer lo que nos hace falta y adaptarlo a las particularidades de la plataforma y del uso que queremos darles.