Cika Tutorial: CTU-012 Título: XBee Programable Autor: Sergio R. Caprile, Senior Engineer							
Revisiones	Fecha	Comentarios					
0	16/02/11						
1	17/06/11	Agrega algunos periféricos del micro de aplicación					

El presente tutorial tiene por objeto presentar la forma de utilizar el XBee PRO ZB Programable. Para información sobre la utilización de otros módulos XBee ZB, se sugieren los CTC-058 al 062.

Índice de contenido

Utilización del bootloader. 2 Configuración y actualización de firmware del EM250. 2 Carga o actualización del firmware de aplicación del MC9S08QE32 2 Desarrollo de aplicaciones en C. 3 Manejo de I/O del procesador. 3 Comunicación con un host por puerto serie. 3 Modo polled. 3 Modo polled. 4 Firmware AT 4 Firmware API. 4 Modo polled. 5 Datos de la aplicación 5 Zons de memoria y vectores de interrupciones. 5 IDE. 5 IDE. 5 IDE. 6 Processor Expert. 6 Coldég. 7 </th <th>Descripción del módulo</th> <th>1</th>	Descripción del módulo	1
Configuración y actualización del firmware de aplicación del MC9S08QE32 2 Carga o actualización del firmware de aplicación del MC9S08QE32 2 Desarrollo de aplicaciones en C 3 Manejo de I/O del procesador 3 Comunicación con un host por puerto serie 3 Modo polled. 3 Modo polled. 3 Comunicación con el EM250 4 Firmware AT 4 Firmware API 4 Modo polled. 4 Modo polled. 4 Modo polled. 4 Firmware API. 4 Modo polled. 5 Datos de la aplicación 5 Zonas de memoria y vectores de interrupciones. 5 Herramientas de desarrollo. 5 IDE 5 Include files. 6 Processor Expert. 6 Clock. 6 SCI2 7	Utilización del bootloader	2
Carga o actualización del firmware de aplicación del MC9S08QE32 2 Desarrollo de aplicaciones en C. 3 Manejo de I/O del procesador. 3 Comunicación con un host por puerto serie. 3 Modo polled. 3 Maniem interrupciones. 3 Comunicación con el EM250. 4 Firmware AT 4 Firmware API. 4 Modo polled. 5 Zonas de memoria y vectores de interrupciones. 5 IDE. 5 Include files. 5 Include files. 6 Processor Expert. 6 Clock. 6 SC12. 7 Generación del código. 7 Alternativas Open Source. 11 FreesBe. 11 Apéndice: Periféricos del MC9S08QE32. 12 ADC	Configuración y actualización de firmware del EM250	2
Desarrollo de aplicaciones en C	Carga o actualización del firmware de aplicación del MC9S08QE32	2
Manejo de I/O del procesador. 3 Comunicación con un host por puerto serie. 3 Modo polled. 3 Mediante interrupciones. 3 Comunicación con el EM250. 4 Firmware AT 4 Firmware API. 4 Modo polled. 5 Zonas de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 IDE. 6 Processor Expert. 6 Clock 6	Desarrollo de aplicaciones en C	3
Comunicación con un host por puerto serie. 3 Modo polled. 3 Mediante interrupciones. 3 Comunicación con el EM250. 4 Firmware AT. 4 Firmware API. 4 Modo polled. 44 Mediante interrupciones. 4 Comunicación con el bootloader. 5 Datos de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 IDE 5 Include files. 5 Include files. 6 Processor Expert. 66 Clock. 6 SC12 7 Generación del código. 8 Programador-Debugger 9 Multilink (P&E Microcomputer Systems). 10 Alternativas Open Source. 11 In FreesBee. 11 Apéndice: Periféricos del MC9S08QE32 12 ADC. 12 I2 13 SPI. 14 sin el bootloader. 14 con el bootloader. 14	Manejo de I/O del procesador	3
Modo polled. 3 Mediante interrupciones. 3 Comunicación con el EM250. 4 Firmware AT 4 Firmware API. 4 Modo polled. 5 Datos de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 IDE 5 Include files. 5 Include files. 6 Processor Expert. 6 Colck. 6 SC12 7 Generación del código. 8 Programador-Debugger 9 Multilink (P&E Micro	Comunicación con un host por puerto serie	3
Mediante interrupciones. 3 Comunicación con el EM250. 4 Firmware AT. 4 Firmware API. 4 Modo polled. 4 Mediante interrupciones. 4 Comunicación con el bootloader. 5 Datos de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 IDE 5 Include files. 6 Processor Expert. 66 Clock. 66 SCI2. 7 Generación del código. 8 Programador-Debugger. 9 Multilink (P&E Microcomputer Systems). 10 Alternativas Open Source. 11 FreesBee. 11 Ejemplos. 11 ADC. 12 I2C. 13 SPI. 14 RTC. 14 sin el bootloader. 14	Modo polled	3
Comunicación con el EM250. 4 Firmware AT 4 Firmware API. 4 Modo polled. 4 Mediante interrupciones. 4 Comunicación con el bootloader. 5 Datos de la aplicación 5 Zonas de memoria y vectores de interrupciones. 5 IDE 5 Include files. 6 Processor Expert. 6 Clock. 66 SCI2 7 Generación del código. 8 Programador-Debugger. 9 Multiluk (P&E Microcomputer Systems). 10 Alternativas Open Source. 11 FreesBee. 11 Ejemplos. 11 Apéndice: Periféricos del MC9S08QE32. 12 ADC. 12 I2C. 13 SPI 14 RTC. 14 sin el bootloader. 14	Mediante interrupciones	3
Firmware AT.4Firmware API.4Modo polled.4Mediante interrupciones.4Comunicación con el bootloader.5Datos de la aplicación.5Zonas de memoria y vectores de interrupciones.5IDE.5Include files.5Processor Expert.6Clock.6SCI2.7Generación del código.8Programador-Debugger9Multilink (P&E Microcomputer Systems).10Alternativas Open Source.11FreesBee.11Apéndice: Periféricos del MC9S08QE32.12ADC.12I2C.13SPI.14RTC.14sin el bootloader.14con el bootloader.14con el bootloader.14	Comunicación con el EM250	4
Firmware API 4 Modo polled 4 Mediante interrupciones 4 Comunicación con el bootloader. 5 Datos de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 Herramientas de desarrollo. 5 IDE. 55 Include files. 6 Processor Expert. 6 Clock. 6 SC12 7 Generación del código. 8 Programador-Debugger. 9 Multilink (P&E Microcomputer Systems). 10 Alternativas Open Source. 11 FreesBee 11 Apéndice: Periféricos del MC9S08QE32 12 ADC. 12 12C. 13 SPI 14 RTC. 14 sin el bootloader. 14 con el bootloader. 14	Firmware AT	4
Modo polled. 4 Mediante interrupciones. 4 Comunicación con el bootloader 5 Datos de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 Herramientas de desarrollo. 5 IDE. 5 Include files. 6 Processor Expert. 6 Clock. 6 SCI2 7 Generación del código. 8 Programador-Debugger. 9 Multilink (P&E Microcomputer Systems). 10 Alternativas Open Source. 11 FreesBee. 11 Spendice: Periféricos del MC9S08QE32. 12 ADC. 12 I2C 13 SPI. 14 sin el bootloader. 14 con el bootloader. 14	Firmware API	4
Mediante interrupciones 4 Comunicación con el bootloader 5 Datos de la aplicación 5 Zonas de memoria y vectores de interrupciones 5 Herramientas de desarrollo 5 IDE 5 Include files 6 Processor Expert 6 Clock 6 SC12 7 Generación del código 8 Programador-Debugger 9 Multilink (P&E Microcomputer Systems) 10 Alternativas Open Source 11 FreesBee 11 Ejemplos 11 ADC 12 I2C 13 SPI 14 RTC 14 sin el bootloader 14 con el bootloader 14	Modo polled	4
Comunicación con el bootloader. 5 Datos de la aplicación. 5 Zonas de memoria y vectores de interrupciones. 5 Herramientas de desarrollo. 5 IDE. 5 Include files. 6 Processor Expert. 6 Clock. 6 SC12. 7 Generación del código. 8 Programador-Debugger. 9 Multilink (P&E Microcomputer Systems). 10 Alternativas Open Source. 11 FreesBee. 11 Ejemplos. 11 ADC. 12 I2C. 13 SPI. 14 RTC. 14 sin el bootloader. 14	Mediante interrupciones	4
Datos de la aplicación5Zonas de memoria y vectores de interrupciones.5Herramientas de desarrollo.5IDE5Include files.6Processor Expert.6Clock.6SCI2.7Generación del código.8Programador-Debugger.9Multilink (P&E Microcomputer Systems).10Alternativas Open Source.11FreesBee.11Ejemplos.11ADC.12ADC.12SPI.13SPI.14RTC.14sin el bootloader.14con el bootloader.14	Comunicación con el bootloader	5
Zonas de memoria y vectores de interrupciones5Herramientas de desarrollo5IDE5Include files6Processor Expert6Clock6SCI27Generación del código8Programador-Debugger9Multilink (P&E Microcomputer Systems)10Alternativas Open Source11FreesBee11Ejemplos11Apéndice: Periféricos del MC9S08QE3212ADC12I2C13SPI14RTC14sin el bootloader14con el bootloader14	Datos de la aplicación	5
Herramientas de desarrollo	Zonas de memoria y vectores de interrupciones	5
IDE.5Include files.6Processor Expert.6Clock.6SCI2.7Generación del código.8Programador-Debugger.9Multilink (P&E Microcomputer Systems).10Alternativas Open Source.11FreesBee.11Ejemplos.11Apéndice: Periféricos del MC9S08QE32.12ADC.12I2C.13SPI.14RTC.14sin el bootloader.14Local.14Acon el bootloader.14	Herramientas de desarrollo	5
Include files	IDE	5
Processor Expert6Clock6SCI27Generación del código8Programador-Debugger9Multilink (P&E Microcomputer Systems)10Alternativas Open Source11FreesBee11Ejemplos11ADC12I2C13SPI14RTC14sin el bootloader.14Locale Adder.14Locale Adder.14	Include files	6
Clock	Processor Expert	6
SCI2.7Generación del código8Programador-Debugger9Multilink (P&E Microcomputer Systems)10Alternativas Open Source11FreesBee11Ejemplos11ADC12ADC12I2C13SPI14RTC14sin el bootloader14	Clock	6
Generación del código.8Programador-Debugger.9Multilink (P&E Microcomputer Systems).10Alternativas Open Source.11FreesBee.11Ejemplos.11Apéndice: Periféricos del MC9S08QE32.12ADC.12I2C.13SPI.14RTC.14sin el bootloader.14	SCI2	7
Programador-Debugger9Multilink (P&E Microcomputer Systems)10Alternativas Open Source11FreesBee11Ejemplos11Apéndice: Periféricos del MC9S08QE3212ADC12I2C13SPI14RTC14sin el bootloader14con el bootloader14	Generación del código	8
Multilink (P&E Microcomputer Systems).10Alternativas Open Source.11FreesBee.11Ejemplos.11Apéndice: Periféricos del MC9S08QE32.12ADC.12I2C.13SPI.14RTC.14sin el bootloader.14con el bootloader.14	Programador-Debugger	9
Alternativas Open Source.11FreesBee.11Ejemplos.11Apéndice: Periféricos del MC9S08QE32.12ADC.12I2C.13SPI.14RTC.14sin el bootloader.14con el bootloader.14	Multilink (P&E Microcomputer Systems)	10
FreesBee11Ejemplos11Apéndice: Periféricos del MC9S08QE3212ADC12I2C13SPI14RTC14sin el bootloader14con el bootloader14	Alternativas Open Source	11
Ejemplos11Apéndice: Periféricos del MC9S08QE3212ADC12I2C13SPI14RTC14sin el bootloader14con el bootloader14	FreesBee	11
Apéndice: Periféricos del MC9S08QE32	Ejemplos	11
ADC	Apéndice: Periféricos del MC9S08QE32	12
I2C	ADC	12
SPI	I2C	13
RTC	SPI	14
sin el bootloader	RTC	14
con el bootloader14	sin el bootloader	14
	con el bootloader	14
PWM (TPM)14	PWM (TPM)	14

Descripción del módulo

Los módulos XBee-PRO ZB Programables incluyen un procesador MC9S08QE32 (core HCS08) de Freescale además del EM250 de Ember. Este procesador adicional permite que el usuario pueda cargar en él el firmware de su agrado, dedicándolo a la tarea que necesite realizar. Trae cargado de fábrica un bootloader que permite actualizar el firmware del EM-250 (el XBee-PRO ZB en sí) y cargar una aplicación en el HCS08, mediante XMODEM, utilizando (por ejemplo) X-CTU.

Utilización del bootloader

El bootloader que viene pre-cargado en el módulo es quien inicia el procesador, verifica la presencia de una aplicación, y la ejecuta si corresponde. Cuando el módulo está corriendo el bootloader (no tiene ninguna aplicación cargada o ésta devolvió el control al bootloader), enviando un retorno de carro (tecla <ENTER>) por su puerto serie a 9600bps, o bien desde otro XBee ZB, obtendremos el menú del bootloader:

```
B-Bypass Mode
F-Update App
T-Timeout
V-BL Version
A-App Version
R-Reset
>
```

Bypass Mode: El bootloader realiza una conexión virtual entre el puerto serie y el EM250, comportándose como un XBee PRO ZB hasta el siguiente reset.

Update App: Comienza el handshake del protocolo XMODEM en espera del envío de la aplicación.

App Version: Devuelve el texto que la aplicación eligió para identificarse.

Reset: Reinicia al procesador, si existe una aplicación cargada la ejecuta, sino permanece en el bootloader.

En el caso que la aplicación cargada no prevea la devolución del control del procesador al bootloader, es posible evitar el inicio de la misma mediante una combinación de señales en algunos de los pines de I/O al momento de aplicar alimentación al módulo: $\overline{RTS} = 1, \overline{DTR} = 0, DIN = 0$.

En la placa XBoard, esto corresponde a insertar en el conector serie una tira de pines con el siguiente conexionado:



Dichos pines de I/O ya pueden volver a su estado normal (retirar el conector en la XBoard), pero el módulo no debe perder la alimentación pues de reiniciarse lo haría de forma normal.

Configuración y actualización de firmware del EM250

El módulo debe estar corriendo el bootloader, es decir, no debe tener una aplicación cargada, ésta debe tener una forma de escapar al bootloader, o se lo inicia mediante la combinación de pines de I/O.

En X-CTU, se selecciona la solapa *Terminal* para poder ingresar un $\langle ENTER \rangle$ y observar el menú del bootloader. En dicho menú, se selecciona la opción *B*, a partir de este instante el EM250 es accesible desde X-CTU como en cualquier otro módulo, con lo cual podemos realizar el upgrade de firmware y configurarlo como cualquier módulo.

Carga o actualización del firmware de aplicación del MC9S08QE32

El módulo debe estar corriendo el bootloader, es decir, no debe tener una aplicación cargada, ésta debe tener una forma de escapar al bootloader, o se lo inicia mediante la combinación de pines de I/O.

En X-CTU o cualquier programa de comunicaciones, seleccionamos la opción F. El bootloader inicia el handshake de XMODEM (veremos una serie de letras C) y debemos iniciar el envío del firmware mediante este protocolo, utilizando paquetes de 64 bytes. Si utilizamos X-CTU (recomendado), la opción *XModem* se encuentra en el menú cuando está seleccionada la solapa *Terminal* (que necesitamos para ingresar al menú del bootloader).

Terminada la carga, podemos consultar la versión de la aplicación mediante la opción A, y correrla mediante la opción R.

Desarrollo de aplicaciones en C

A continuación realizamos una breve descripción de los recursos de que disponemos para el desarrollo de aplicaciones en C.

Manejo de I/O del procesador

El entorno de desarrollo define los bits individuales como *bitfields* dentro de una *union*; es decir, es posible operar directamente sobre un bit en una sentencia C. La relación entre los pines de I/O del módulo XBee y los del procesador se realiza incluyendo el archivo *pin_mapping.h.*

```
#include "derivative.h"
#include "pin mapping.h"
```

Los puertos de I/O tienen (al menos) un registro para controlar el estado, otro para seleccionar el sentido de comunicación, y otro para habilitar la resistencia interna de pull-up. La nomenclatura de este archivo utiliza _D para el registro de dirección (entrada/salida) y _PE para habilitación del pull-up (**P**ull-up Enable). Por ejemplo: operamos sobre $DIO8/\overline{DTR}$:

IO_DIO8_DTR=0; // bajo

seleccionamos si $DIO7/\overline{CTS}$ es entrada o salida:

IO_DIO7_CTS_HOST_D=0; // entrada

habilitamos el resistor de pull-up:

IO_DIO7_CTS_HOST_PE=1; // habilitado

Comunicación con un host por puerto serie

Dicha comunicación utiliza la UART1 del MC9S08QE32. El módulo del procesador se denomina SCI1 (Serial Communications Interface). La inicialización la podemos realizar mediante una herramienta del entorno de desarrollo. Dado que se utiliza un clock interno del procesador, el fabricante del módulo sugiere limitar la velocidad de operación a 9600bps

Modo polled

Para una comunicación polled (sin interrupciones), podemos: revisar si se recibió un dato:

```
if ( SCI1S1 RDRF )
```

leer el dato recibido:

dato = SCI1D;

revisar si es posible enviar un dato:

```
if ( SCI1S1_TDRE )
```

enviar un dato:

SCI1D = dato;

Mediante interrupciones

Si deseamos utilizar interrupciones, deberemos escribir los handlers correspondientes. Si utilizamos la herramienta de inicialización que se provee en el entorno de desarrollo, los handlers son definidos en el archivo MCU_init.c que se genera:

```
interrupt void isrVsciltx(void)
{
   /* Write your interrupt code here ... */
}
/* end of isrVsciltx */
interrupt void isrVscilrx(void)
{
   /* Write your interrupt code here ... */
}
/* end of isrVscilrx */
```

El interrupt handler para interrupciones de recepción, debe leer el registro de estado SCIISI a modo de reconocimiento de la interrupción.

Comunicación con el EM250

Dicha comunicación utiliza la SCI2 del MC9S08QE32. Nuevamente, la inicialización la podemos realizar mediante una herramienta del entorno de desarrollo. En esta UART veremos lo que hayamos grabado y configurado en el EM250, es decir, es como si tuviéramos conectado un XBee PRO ZB. Por defecto, el firmware grabado es API y es lo que el bootloader espera. Es posible operar sobre un firmware AT si se prescinde del acceso remoto al bootloader o se lo modifica. La velocidad de operación es la que se configure en el EM250, y por defecto es de 9600bps. Se recomienda no excederla, y si se desea que el bootloader pueda comunicarse con el EM250, no cambiarla.

Firmware AT

Cargado el firmware AT, operaremos sobre la SCI2 de igual modo que como hiciéramos sobre la SCI1. Los registros son: SCI2S1 y SCI2D.

Firmware API

Este es el firmware por defecto. Si bien tanto el bootloader como la aplicación de ejemplo que distribuye el fabricante tienen implementaciones de API, sugerimos la utilización de la implementación que se describe en CAN-089. Para ello, deberemos proveer dos funciones que realicen la interfaz entre las funciones de API y la SCI. Éstas son TX() y RX(), cuyas características se describen en CAN-089.

Modo polled

El siguiente listado corresponde a una implementación mínima:

```
#include "apiframe.h"
#include "apicommon.h"
#include "apizb.h"
int TX(unsigned char data)
{
        if (SCI2S1 TDRE)
               SCI2D = data;
       else
               return(-1);
       return(1);
}
int RX()
{
        if (SCI2S1 RDRF)
               return SCI2D;
        return(-1);
}
```

Mediante interrupciones

En caso de utilizarse interrupciones, el interrupt handler de recepción deberá colocar los datos en un buffer, del que serán extraídos por RX(); mientras que TX() colocará los datos en un buffer del que serán extraídos por el interrupt handler de transmisión.

Comunicación con el bootloader

La comunicación entre la aplicación y el bootloader se realiza mediante variables compartidas situadas en un área especial de RAM:

```
#include "common.h"
#include "sharedRAM.h"
```

La aplicación tiene la opción de devolver el control al bootloader si lo desea, indicando una causa de reset en una variable compartida. Dichas causas se definen en el archivo *common.h*:

APP_CAUSE_NOTHING: reset sin causa aparente. Valores entre 0 y 255 son ignorados por el bootloader y pueden ser utilizados por la aplicación.

```
APP_CAUSE_FIRMWARE_UPDATE: solicita al bootloader inicie un proceso de firmware update
APP_CAUSE_BYPASS_MODE: instruye al bootloader a que se coloque en modo bypass
APP_CAUSE_BOOTLOADER_MENU: instruye al bootloader a que no inicie la aplicación.
```

La cesión de control al bootloader se realiza dejando expirar el watchdog timer. Por ejemplo:

```
for(;;) {
    ____RESET_WATCHDOG();
    if( quiero ) {
        DisableInterrupts;
        AppResetCause = APP_CAUSE_BOOTLOADER_MENU;
        for(;;); // espera timeout de WDT
    }
    ...
}
```

Datos de la aplicación

El texto que indica el nombre y versión de la aplicación se indica al bootloader mediante variables compartidas, de la siguiente forma:

```
#pragma CONST_SEG APPLICATION_VERSION
static const uint8 version[] = "miprograma 1.0";
#pragma CONST_SEG DEFAULT
static const unsigned long pAppVersion @0x0000F1BC = (unsigned long)version;
```

Zonas de memoria y vectores de interrupciones

Para que la aplicación utilice las zonas de memoria que le corresponden, debemos indicárselo al linker del entorno de desarrollo. El archivo que realiza esta función se encuentra en el directorio *prm* y su nombre por defecto es *Project.prm*. Es posible definir otro archivo mientras se lo indiquemos al IDE.

Al final de dicho archivo encontramos la definición de los vectores de inicialización e interrupciones:

```
//VECTOR 0 _Startup /* Reset vector: this is the default entry point for an application. */
VECTOR ADDRESS 0x0000F1FE _Startup /*VECTOR ADDRESS 0x0000F1EA vReset */
```

A continuación, se agregarán entradas para los vectores de interrupciones que se utilicen, si corresponde (por ejemplo una SCI).

Herramientas de desarrollo

En esta sección veremos las opciones de desarrollo de que disponemos.

IDE

CTU-012, XBee Programable

El entorno de desarrollo empleado es el Freescale Codewarrior for Microcontrollers 6.3 SE. Si bien existen versiones más recientes basadas en Eclipse, la información provista por Digi y todo lo realizado en este documento emplean la versión 6.3

La creación de un nuevo proyecto se realiza mediante una wizard que guía al usuario en los pasos a seguir. entre ellos están la elección de la herramienta de programación y depuración, y la inicialización del chip.

Los archivos de código fuente se ubican en el directorio Sources del proyecto.

Include files

Los archivos *.h* que compartimos entre varios proyectos, como por ejemplo los mencionados en este documento: *pin_mapping.h, common.h* y *sharedRAM.h*, pueden ubicarse a nuestro gusto. Para que el compilador los encuentre, debemos agregar el directorio a la lista de *Access paths*. La forma más simple es por lo menos agregar uno de los archivos en cuestión al proyecto, en la sección *Includes*, para que el IDE agregue automáticamente la ruta a la lista:

🖪 Standard Settings [cart	et.mcp]	? 🛛
Target Settings Panels	Access Paths	
🖃 Target 🔺	User Paths	Always Search User Paths
Target Settings	C System Paths	Source relative includes
Access Paths	U.s. Datha	
- Build Extras	User Paths	
File Mappings	(Project)	
Source Trees	Project/prm	
Assembler for HC08	 {Project}Sources 	
Burner for HC08	Compiler}lib\hc08c\device\src	
Compiler for HC08	Compiler}lib\hc08c\lib	
Importer for HC08	{Compiler}lib\hc08c\src	
Linker for HC08		
Editor		
Custom Keywords		
		T
		Þ
v	Add Default Host Flags:	Add Change Remove
	Factory Settings Revert	Import Panel Export Panel
		OK Cancel Apply

Processor Expert

Esta es la herramienta encargada de realizar la inicialización del chip. De forma gráfica se accede a los registros de control del micro y sus periféricos, y esta herramienta genera el código correspondiente. El IDE lo incluye automáticamente entre los fuentes disponibles. Si bien el micro es inicializado por el bootloader, la aplicación dispone de total control sobre éste y, excepto algunos registros que sólo pueden escribirse en el arranque, puede configurarlo a su agrado.

Clock

La captura siguiente muestra la configuración de clock para máxima velocidad:

💊 Inspector MC9S08QE32CLD 🛛 🛛 🔀								
Component Parameters				Register Details				
Clock settings				Name	Address	Init.value	Register Map	
Source CPU clock	nternal Clock 📃 👻	31.25 kHz	Ð	ICSC1	0x0038	04	H 00000000	
- 🖃 Internal clock			Ð	ICSC2	0x0039	00		•
🖌 🖌 Internal oscillator fr 3	31.25	31.25 kHz	Ð	ICSTRM	0x003A	????????		•
🖌 🗸 Internal ref. clock f 🛛	Disabled 🛛 🖸		Ð	ICSSC	0x003B	0111000?		•
📙 🗆 Initialize trim vay	ies 🖸		Ð	SRS	0x1800	82 1		•
🖌 🖌 Trim value addre F	FFAF H		Ð	SOPT1	0x1802	42	H	•
🛛 🕹 🖌 🖌 Fine trim value a F	FFAE H		Ð	SOPT2	0x1803	10		•
- 🕀 External clock	Disabled 🖸 🖸		Ð	SDIDH	0x1806	????0000		•
- 🖂 Bus freq. divider 🛛 1	T T	1	Ð	SDIDL	0x1807	1F (•
🛛 - 🖌 Internal bus clock 🛛 1	19.0	19.0 MHz; (38/1/	Ð	SPMSC1	0x1808	10		
Fixed frequency cloc	0.015625		Ð	SPMSC2	0x1809	02		•
FLL mode	Engaged 📃 👻	FEI	Ð	SPMSC3	0x180B	00		•
- 🗆 Ref. clock sour	nternal Clock		Ð	SCGC1	0x180E	FF	H	•
🛛 🕹 🖌 Ref. clock freq. 🛛	0.03125	Warning: The refi	Ð	SCGC2	0x180F	FF	H	•
E DCO mode	Fine tuned 32kHz 💿 💌	Fine tuned 32kHz	Ð	PTASE	0x1841	00		
FLL mult. factor	1216 👻	1216	Ð	PTADS	0x1842	00		
FLL output clock fi		38.0 MHz; (0.031	Ð	PTBSE	0x1845	00	H 0000000	0
Lew-power mode:			Ð	PTBDS	0x1846	00		
🖌 Initialization interrupt pr i	nterrupts enabled 📃 💌	1	Ð	PTCSE	0x1849	00		
🗄 Internal resource m			Ð	PTCDS	0x184A	00	H 0000000	0
🗄 Internal peripherals			Ð	PTDSE	0x184D	00		
E CPU interrupts			Ð	PTDDS	0x184E	00	<u>H 00000000</u>	
			Ð	PTESE	0x1851	00		0
			Ð	PTEDS	0x1852	00		0
			Ð	NVBACKKEYO	OxFFBO	FF	<u>+ </u>	•
			Ð	NVBACKKEY1	0xFFB1	FF	<u>+ </u>	•
			Ð	NVBACKKEY2	0xFFB2	FF		•
			Ð	NVBACKKEY3	0xFFB3	FF	<u>+</u> • • • • • • • • • •	•
			Ð	NVBACKKEY4	0xFFB4	FF I	<u>+ </u>	•
			Ð	NVBACKKEY5	0xFFB5	FF I		•
			Ð	NVBACKKEY6	0xFFB6	FF	H	•
			Ð	NVBACKKEY7	0xFFB7	FF	H	•
				NVPROT	0xFFBD	FF I		•
			Ð	NYOPT	UxFFBF	7E		9
Disable Peripheral In	nitialization	?	• Vi	ew Register Map 1 0 🌒 undefined, rese	rved, read-only	/ 🔽 🖸	K <u>C</u> ancel	?

SCI2

La captura siguiente muestra la configuración de la SCI2 para operación en modo polled:

baud baud de vakeup vakeup vakeup vakeu poration vated vakeu poration vakeu poration vated v	ISCI2		Name SCGC1 SCI28DH SCI28DL SCI2C1 SCI2C2 SCI2S1 SCI2S2 SCI2C3 SCI2C3 SCI2D	Add Ox Ox Ox Ox Ox Ox Ox	Iress 180E 1870 1871 1872 1873 1874 1875 1876 1877	Init.value FF H 00 H 78 H 00 H 00 H 00 H 00 H 00 H	Register Map
baud t baud de vakeup t bit DLE bit peration ted Q peration Q Q Q Q Q Q Q Q Q Q Q Q Q			SCGC1 SCI2BDH SCI2BDL SCI2C1 SCI2C2 SCI2S1 SCI2S2 SCI2S2 SCI2C3 SCI2C3 SCI2D	0x 0x 0x 0x 0x 0x 0x 0x	180E 1870 1871 1872 1873 1874 1875 1876 1877	FF H 00 H 78 H 00 H 00 H 00 H 00 H	
2 baud 2 de vakeup bit bit tbit vakeup cbit tbit vakeup vakeup tbit tbit vakeup tbit tbit vakeup va	2 2 2 2 2 2 2 2 2 2 2 2 2 2		SC128DH SC122DL SC12C1 SC12C2 SC12S1 SC12S2 SC12S2 SC12C3 SC12D	0x 0x 0x 0x 0x 0x 0x 0x	1870 1871 1872 1873 1874 1875 1876 1877	00 H 78 H 00 H 00 H 00 H 00 H 00 H	
beration view of the second se			SCI2BDL SCI2C1 SCI2C2 SCI2S1 SCI2S2 SCI2C3 SCI2C3 SCI2D	0x 0x 0x 0x 0x 0x	1871 1872 1873 1874 1875 1876 1877	78 H 00 H 00 H 00 H 00 H 00 H	
baud			SCI2C1 SCI2C2 SCI2S1 SCI2S2 SCI2C3 SCI2C3 SCI2D	0x 0x 0x 0x 0x	1872 1873 1874 1875 1876 1877	00 H 0C H 00 H 00 H	
t baud	2 1 2 2 2 2 2 2 2 2 2 2 2 2 2		SCI2C2 SCI2S1 SCI2S2 SCI2S2 SCI2C3 SCI2D	0x 0x 0x 0x	1873 1874 1875 1876 1877	00 H 00 H 00 H	
de vakeup bit IDLE bit Deration ted Deration bits			SCI2S1 SCI2S2 SCI2C3 SCI2C3 SCI2D	0x 0x 0x 0x	1874 1875 1876 1877	00 H 00 H 00 H	
Je vakeup			SCI2S2 SCI2C3 SCI2D	Ox Ox Ox	1875 1876 1877	00 H 00 H 00 H	
vakeup va			SCI2C3 SCI2D	0x 0x	1876 1877	н 00 н 00	
vakeup v bit v IDLE bit v peration v ted v ted v poset of the second seco)))))))))))))))))))		SCI2D	0x	1877	H 00	
vakeup • tik tiki • IDLE bit • D Deration • oits • ted • ted • O							
vakeup vibit vibit vibit vibit vibit vibit vibit vibit vibits vib							
: bit IDLE bit peration bits ted ted S							
IDLE bit							
veration v peration v ted v ted v ted v D							
peration ted ted D							
peration v bits v ted v ted v	 						
bits v ted v ted v	2 2 2						
ted ▼ ted ✓	2						
ted 🗾 🗸 D)) 						
0							
	Vsci2tx						
0)						
9	X						
	Vsci2rx						
Ð	X						
5	X						
5	X						
9	>						
Ð	>						
0	>						
	2 2 2 2 2 2 2 2 2 2 2	Vsci2tx 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Vsci2m Q Q Q Q Q	Vsci2rx Q </td <td>Vsci2nx Vsci2nx O O O O O O O O O O O O O</td> <td>Vsci2tx Q Q Q Q Q Q Q Q Q Q Q</td> <td></td>	Vsci2nx Vsci2nx O O O O O O O O O O O O O	Vsci2tx Q Q Q Q Q Q Q Q Q Q Q	

Generación del código

Una vez depurada la aplicación, generamos el código a ser cargado mediante XMODEM utilizando el *Burner*. Accedemos a él de una forma un poco compleja, como podemos ver en la captura siguiente:

CTU-012, XBee Programable

Freescale CodeWarrior	
ne cut wew search Protesson Expert Device Initialization window hep	
Construction Construction Construction Construction Image: Sources Construction Construction Construction Image: Source Construction Construction Construction Image: Source Construction Construction Construction Image: Source	
Factory Settings Revert Import Panel Export Panel	
OK Cancel Apply	
	1.

Una vez allí, la instruimos para que genere binario por la totalidad del espacio de memoria disponible, ya que esto es lo que espera el bootloader:

Burner			Burner		×
Input/Output Content	Command File		Input/Output Conte	nt Command File	
Format:	C Motorola S C Intel Hex • Binary	SRecord Configuration Type: © Automatic	Input Input File:	%ABS_FILE%	
Data Bus:	 1 Byte 2 Bytes 4 Bytes 	O 52 O 53	Output	Com Settings Baud Rate: 960	
Swap Bytes:		Bytes per Line: 32	C Com2	Parity: No	ne 🔽
Undef Byte (hex):	ff		C Com3	Data Bits: 8 B	its 🔽
Origin:	8400 Extrac	ts all butes from 0x8400 to 0xf1ff			
Length:	6e00	ling 0xf1ff)	File	%ABS_FILE%.bin	
Destination Offset:	, lo		1st Byte (msb)	•	Execute
	Acep	tar Cancelar Ayuda		Acept	ar Cancelar Ayuda

El archivo así generado tendrá el nombre *Project.abs.bin*, y se encontrará en el directorio *bin* del proyecto. Éste es el archivo que debemos utilizar para realizar la carga de la aplicación por XMODEM en el producto final.

Programador-Debugger

Sin duda una herramienta de este tipo acelera el proceso de desarrollo y depuración del código. No obstante, si se desea conservar el bootloader, no es posible utilizarla para la grabación final de la aplicación, a menos que se incluya todo el código (bootloader y aplicación) en un proyecto.

La conexión de esta herramienta con el módulo se realiza mediante cuatro pines:

señal	pin
BDM	8
RESET	5
VDD	1
GND	10

Estas herramientas esperan encontrarse con un conector de pines de paso .1", cuyo pinout es el siguiente:

	1		
BDM	Ð.	0	GND
	O	Ο	RST
	0	0	VDD

En la placa XBoard existe un conector de este tipo. Para adecuar una placa XBoard más antigua a un XBee Programable, podemos conectarnos en los siguientes puntos:



Es conveniente trabajar primero con la aplicación en sí, grabándola directamente con el programador/debugger. Cuando ya tengamos la aplicación depurada compilamos para poder generar el ejecutable que subiremos por XMODEM al módulo. Esto requiere de disponer de dos versiones del archivo de definiciones para el linker, una que mapee los vectores para que la aplicación tenga el control, y otra que respete los lineamientos del bootloader. El primer caso es generado automáticamente por el entorno de desarrollo, y el segundo caso lo hemos analizado unos apartados atrás.

Multilink (P&E Microcomputer Systems)

Esta es la herramienta sugerida por Digi. Es un producto de costo elevado y buenas prestaciones.

Lo único que debemos tener en cuenta al utilizar esta herramienta es el seteo correspondiente a la referencia interna del clock de la CPU, en 31250 Hz.

La captura siguiente muestra esta situación:

PEMICRO Connection Manager							
You have selected to display this dialog on startup. Specify communications parameters and click OK							
Connection port and Interface Type							
Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port							
Port: USB-ML-12 Rev C on USB1 (Name=PE5515179) (Autodetected)							
Interface Detected : Firmware Version : Socket Programming Adapter Settings							
Target CPU Information							
CPU: HCS08 Processor - Autodetect							
MCU reset line: MCU Voltage:							
Reset Options Delay after Reset and before communicating to target for milliseconds (decimal).							
Cyclone Pro Power Control (Voltage> Power-Out Jack) Provide power to target Regulator Output Voltage Power Down Delay 250 mS Regulator Output Voltage Power off target upon software exit							
Trim Control Default trim reference frequency is : 32768.00 Hz. (Valid Range: 31250.00 to 39062.50 Hz)							
Connect (Reset) Hotsync Abort ✓ Show this dialog before attempting to contact target (Otherwise only display on Error)							

Alternativas Open Source

Existen muchas alternativas basadas en código y hardware abiertos. Por ejemplo hemos ensayado el WTUSBDML de Witztronics. El circuito esquemático y el código pueden obtenerse en la Internet.

Estas herramientas, al menos en Code Warrior V6.3, presentan el inconveniente de no preservar la calibración de la referencia del oscilador interno del microcontrolador.

FreesBee

Cika ha desarrollado una herramienta basada en código y hardware abiertos, denominada FreesBee. Esta herramienta es de muy bajo costo y se la describe en su manual correspondiente. En el mismo encontraremos además los pasos y estrategias a seguir para desbloquear el bootloader y regenerar el valor de calibración de la referencia del oscilador interno del microcontrolador.

Ejemplos

Un ejemplo concreto, que además contiene los include files mencionados, se encuentra en la Nota de Aplicación CAN-092.

De él debemos extraer:

- en el directorio Sign
 - los include files para definición de los pines: *pin_mapping.h*
 - y comunicación con el bootloader: common.h y sharedRAM.h
- en el directorio Sign\CartelBeeAPI\prm:
 - los archivos de definición del linker, para trabajar con: Program sibl.prm
 - y sin el bootloader: Program nobl.prm

Apéndice: Periféricos del MC9S08QE32

Si bien la literatura de Freescale es amplia, presentamos aquí un pequeño resumen para una rápida utilización de los periféricos del micro.

ADC

El conversor A/D del micro de aplicación es bastante poderoso, pudiendo operar del reloj del sistema o de uno propio. En el primer caso, debe tenerse en cuenta el divisor a utilizar para respetar la frecuencia máxima de operación.

Los pines a utilizar como entradas analógicas se configuran en una pareja de registros de control, *APCTL1* y *APCTL2*; mientras que la señal a convertir se selecciona en el registro de control *ADCSC1*.

La resolución del conversor se selecciona mediante el registro *ADCCFG*, que además permite configurar el tiempo de sampling y el clock de conversión. Existe además una lógica de comparación que permite detectar (sin intervención de la CPU) si la señal convertida se encuentra dentro de un rango específico configurable.

El siguiente listado muestra una configuración sugerida, en la cual operamos el ADC por polling con una resolución de 12-bits:

```
ADCCFG ADLPC=0;
                               // clock speed = rápido (standard, no "low power")
//ADCCFG ADIV
                               // no usamos divisor
ADCCFG ADLSMP=0;
                               // short sample time
                               // 12-bits
ADCCFG MODE=1;
ADCCFG_ADICLK=3;
ADCSC2_ACFE=0;
                               // asynchronous ADC clock
                               // disable compare logic
                               // software trigger
ADCSC2 ADTRG=0;
APCTL1=APCTL1 ADPC0 MASK+APCTL1 ADPC3 MASK; // pines usados como entradas analógicas
APCTL2=0;
                               // demás pines: general purpose I/O
```

La conversión se inicia al seleccionar la entrada (escribiendo *ADCSC1*), y el valor convertido está disponible cuando el flag *COCO* (COnversion COmplete) en el registro *ADCSC1* es colocado en estado alto:

```
/*
ADCSC1_AIEN=0; // no interrupts
ADCSC1_ADCO=0; // single conversion
ADCSC1_ADCH=0; // channel 0
*/
ADCSC1=0;
while(!ADCSC1_COCO);
```

El tiempo de conversión, con la configuración sugerida, es de algo menos de 15us.¹ El canal seleccionado corresponde al siguiente mapa:

pin XBee	nombre	canal ADC
20	IO_DIO0_ADC0_COMMISSIONING	0
19	IO_DIO1_ADC1	3
18	IO_DIO2_ADC2	6
17	IO_DIO3_ADC3	9
11	IO_DIO4_ADC4	7

El valor de la magnitud analógica se recupera conociendo el valor de la tensión de referencia. En nuestro caso, proveemos 2,75V por el pin correspondiente:

¹ basado en un clock asincrónico de aproximadamente 3,3 MHz, con 23 ciclos de clock para conversión (7us), 5us de jitter de inicio y 5 bus clocks adicionales.

Nótese que el registro ADCR es en realidad una concatenación de los registros ADCRH y ADCRL; el compilador realiza una lectura de 16-bits que los accede en el orden correcto.

Existe además la posibilidad de disparar automáticamente el ADC mediante una señal configurable, tema que no desarrollaremos aquí.

l²C

}

El controlador I²C permite operar como slave o como master, en un ambiente multi-master. Veremos aquí cómo utilizarlo como único master para acceder a uno o más slaves en el bus, en modo polled. Inicialización del controlador:

```
IICF=0x14; // rate = bus clock / 80 (ver hoja de datos)
IICC2=0;
IICC1_IICEN=1; // enable module
IICC1_MST=0; // slave mode (?)
IICS_SRW=0; // receive mode (?)
```

Antes de iniciar cualquier nueva operación, debemos chequear si la última se ha completado:

while(IICS BUSY);

Chequeamos cada transacción mediante el flag IICS_TCF. Este flag se resetea al comenzar la transacción, por lo que debemos esperar antes de revisarlo por primera vez:

```
void wait(void) {
  uint8_t i;
    for(i=0;i<32;i++);
    while(!IICS TCF);</pre>
```

Escritura de un dato en un slave:

```
while(IICS BUSY);
IICC1_TX=1;
                               // transmit mode
IICC1 MST=1;
                               // master mode (generate START)
                               // address + WR
IICD=(0x1C<<1)+0;</pre>
wait();
err=0;
if(!IICS RXAK) {
       IICD=0x02;
                              // dato a escribir en el slave
       wait();
} else err=1;
                               // device not present
IICC1 MST=0;
                               // slave mode (generate STOP)
```

Lectura de un dato en un slave:

```
while(IICS BUSY);
IICC1_TX=1;
IICC1_MST=1;
                                // transmit mode
                                // master mode (generate START)
// address + WR
IICD=(0x1C<<1)+0;</pre>
wait();
err=1:
if(!IICS RXAK) {
                                         // dirección o registro en el slave
        IICD=0x06;
        wait();
        if(!IICS RXAK){
                IICC1 RSTA=1;
                                        // repeated START
                IICD=(0x1C<<1)+1;</pre>
                                        // address + RD
                wait();
                if(!IICS RXAK) {
                                                // send ACK
                        IICC1_TXAK=1;
                        IICC1 TX=0;
                                                // receive mode
                                                // inicia receive sequence
                        x = TTCD:
                        wait();
                        IICC1 MST=0;
                                                // slave mode (generate STOP)
                        x=IICD;
                                                // obtiene el dato que leyó
                        err=0
                }
       }
} // else device not present
if(err)
```

```
IICC1 MST=0;
```

// slave mode (generate STOP)

SPI

El controlador SPI permite operar como slave o como master. Veremos aquí cómo utilizarlo como master para acceder a uno o más slaves en el bus, en modo polled.

El pin \overline{SS} puede ser comandado directamente por el controlador. Sin embargo, por generalidad lo controlaremos manualmente:

```
IO_DIO3_ADC3_D=1; // SS as output
IO_DIO3_ADC3=1; // SS inactive
```

Inicialización del controlador:

```
SPIBR=0x02; // rate = bus clk / 8
SPIC1=0;
SPIC2=0;
SPIC1_MSTR=1; // master mode
SPIC1_CPOL=0; // mode 0
SPIC1_CPHA=0;
SPIC1_SPE=1; // enable module
```

Para escribir datos, podemos chequear el flag SPIS_SPTEF. Sin embargo, este flag indica que el registro está listo para aceptar un nuevo byte, no que terminó de enviarlo. El final de una transacción se detecta mediante el flag SPIS_SPRF, que indica que hay un byte disponible para ser leído.

Escritura y lectura de datos multi-byte en un slave:

```
IO_DIO3_ADC3=0; // SS active
SPID=0x02; // dato 1 a escribir
while(!SPIS_SPRF);
x=SPID; // lectura, clear SPRF flag
SPID=0x85; // dato 2 a escribir
while(!SPIS_SPRF);
IO_DIO3_ADC3=1; // SS inactive
x=SPID; // lectura, clear SPRF flag
```

RTC

El micro posee un módulo que le permite generar interrupciones como una base de tiempo. Entre las opciones de clocking, disponemos de un oscilador de muy bajo consumo de 1KHz (30%). Además de un prescaler, posee un contador de módulo variable.

La inicialización del módulo RTC para obtener interrupciones cada 100ms es la siguiente:

RTCMOD=0; RTCSC = 0x1D; // interrupt every 100ms, internal 1-KHz osc

La rutina de interrupciones deberá resetear el flag de interrupt:

No deberemos olvidarnos de la correspondiente entrada para el vector número 24 en Program.prm:

sin el bootloader

VECTOR 24 Vrtc_isr

con el bootloader

VECTOR ADDRESS 0x0000F1CE Vrtc isr

PWM (TPM)

CTU-012, XBee Programable

El micro cuenta con tres módulos timer con capacidad de generación de PWM, además de captura y comparación. El listado siguiente muestra la inicialización de uno de los canales para obtener una señal de ciclo de trabajo 1/16 a la frecuencia de bus dividida por 256 (resolución de 8-bits), prescalada 32 veces:

TPM1MOD = 0x00FF; // Modulo value: 256 TPM1C1SC = 0x28; // PWM mode, clears output on channel value match TPM1SC = 0x0D; // Overflow interrupt disabled, edge-aligned, busclk/32 TPM1C1V = 0x0010; // Channel value. duty = 1/16. freq = bus/(32*256)