Programación de Sistemas Embebidos 2020
El primer programa embebido

Índice general

Chapter 4

El primer programa embebido

Rafael Ignacio Zurita
Universidad Nacional del Comahue

En este capítulo realizaremos programación embebida directa a través de un ejemplo. Nuestro ejemplo es similar, en espíritu, al programa 'Hello, World!' que puede encontrar al comienzo de la mayoría de los libros de programación También, explicaremos las razones por la cual seleccionamos el programa particular presentado en este capítulo, y especificamos las secciones de código que son dependientes del hardware. El capítulo únicamente contiene el código fuente del programa. Se explica como compilar y ejecutar el programa en el capítulo siguiente.

Keywords: sistema embebido, primer programa embebido, ciclo de compilación, registro de E/S, compiladores cruzados (cross-compilers)

> I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free.
> —Richard Stallman 27-sep-1983, Founder of the GNU Project.

1 Hello, World!

Pareciera que todo libro sobre programación comienza con el mismo ejemplo - un programa que imprime 'Hello, World!' en la pantalla del usuario. Un ejemplo como este tan utilizado podría parecer aburrido. Entre otras cosas, el ejemplo ayuda al lector a verificar la facilidad o dificultad con la que se pueden escribir programas sencillos en el ambiente de programación utilizado. En ese sentido, el programa 'Hello, World!' es muy útil como benchmark para los usuarios de lenguajes de programación y plataformas de computación.

Los sistemas embebidos son las plataformas de computación mas desafiantes con las cuales trabajar si se iniciara el aprendizaje con el benchmark 'Hello, World!". Incluso, para algunos sistemas embebidos podría ser imposible implementar este programa. Y en los sistemas en que fuese posible, el hecho de impri-

mir mensajes de texto usualmente es más un punto de llegada que el comienzo del aprendizaje.

Una presunción principal del ejemplo 'Hello, World!' es que existe algún tipo de dispositivo de salida en el cual las cadenas de caracteres puedan ser mostradas. Una ventana de texto en el monitor del usuario usualmente sirve para ese propósito. Pero, la mayoría de los sistemas embebidos no contienen un monitor o dispositivo de salida similar. Y para los que tengan, típicamente, requieren una pieza especial de software embebido llamado controlador de pantalla (display driver), que se debe implementar en primer lugar. Si este llegara a ser el caso para un primer programa para estas plataformas entonces sería una manera muy desafiante de comenzar una carrera en programación de sistemas embebidos.

Una forma mucho mas conveniente es comenzar con una programa embebido portable, fácilmente implementable y pequeño. De esta manera hay pocas posibilidades de cometer errores. Por lo tanto, el ejemplo básico presentado en este libro elimina una de las variables si el programa no funciona correctamente la primera vez: el programa no contiene un error en el código, sino que el problema es con las herramientas de desarrollo o el proceso utilizado para crear el ejecutable.

Los programadores de sistemas embebidos deberían desconfiar de todo. Siempre se debe comenzar cada nuevo proyecto con la suposición de que nada funciona, y que lo único en que se puede confiar es en la sintaxis básica de su lenguaje de programación. Incluso las rutinas de las bibliotecas estándar podrían no estar disponibles. Generalmente, estas funciones auxiliares se toman como parte de la sintaxis básica del lenguaje C estándar, y la mayoría de los demás programadores dan por sentado que cuentan con estas rutinas. De cualquier manera, estas rutinas de bibliotecas son difíciles de portar a todas las posibles plataformas de computación, y ocasionalmente son ignoradas por los desarrolladores de compiladores para sistemas embebidos.

Por lo que no utilizaremos un programa 'Hello, World!'. En cambio, escribiremos un programa en lenguaje C lo mas simple posible que podamos, sin suponer que tenemos hardware especializado (el cual requeriría de un controlador de dispositivo) o bibliotecas con funciones como printf. A medida que se avance con los diferentes temas agregaremos gradualmente rutinas de bibliotecas estándar, y el equivalente a un dispositivo de salida de caracteres a nuestro repertorio. Para entonces, ya debería ser un experto en el campo de la programación de sistemas embebidos.

2 El programa que parpadea un LED (Blinking LED)

Casi todo sistema embebido que hemos encontrado en nuestras respectivas carreras tienen al menos un LED que puede ser controlado por software. Si los diseñadores de hardware planean no colocar un LED al circuito intente realizar lobby fuertemente, para obtener uno conectado a un pin de propósito general (general-purpose I/O (GPIO)). Como se verá luego, este LED podría ser la herramienta de depuración (debugging) más valiosa con la que se cuenta.

Un substituto popular al programa 'Hello, World!' es un programa que hace parpadear un LED en una tasa de una vez por segundo (prendido y apagado). Típicamente, el código requerido para encender y apagar un LED está acotado a unas pocas líneas de código. Por lo tanto, hay muy pocas posibilidades de cometer errores de programación. Además, como casi todo sistema embebido contiene un LED, el concepto de este programa básico es extremadamente portable.

Precisión del tiempo de parpadeo

El programar exactamente el parpadeo para que el ciclo de prendido/apagado dure un segundo es difícil. Para conocer la precisión puede utilizar un cronómetro. Simplemente inicie un cronómetro, cuente la cantidad de veces que se apaga el LED, pare el cronómetro, y vea si el número de segundos que transcurrieron coinciden con el número de parpadeos contados. Si se necesita mayor precisión simplemente cuente por mas tiempo la cantidad de 'apagados'.

El primer paso es aprender a controlar el LED que deseamos parpadear. En la placa arduino pro mini, existe un led color rojo localizado entre los pines digitales etiquetados '10' y '11' (observe la Figura 1. Se debe utilizar los esquemáticos para conocer el recorrido de la conexión, desde el LED al microcontrolador. Este es el método adecuado que necesita realizar cada vez que deba conocer cómo está conectado un LED.

En los esquemáticos de la placa se puede observar que el LED está controlado por la señal de salida PORTB5 del microcontrolador atmega328p. El manual del microcontrolador informa que esa señal es controlada por un pin GPIO del microcontrolador, bit 5 del PORTB. Por lo tanto, se necesita poder alternar ese pin del microcontrolador HIGH y LOW para obtener un programa que realice el parpadeo adecuadamente.

La estructura general del programa led_blink.c se muestra a continuación. Esta parte del programa es independiente del hardware. De cualquier manera, este asume que existen funciones llamadas led_init(), led_toggle(), y delay_ms() para inicializar el pin GPIO que controla el LED, cambiar el estado del LED, y manejar

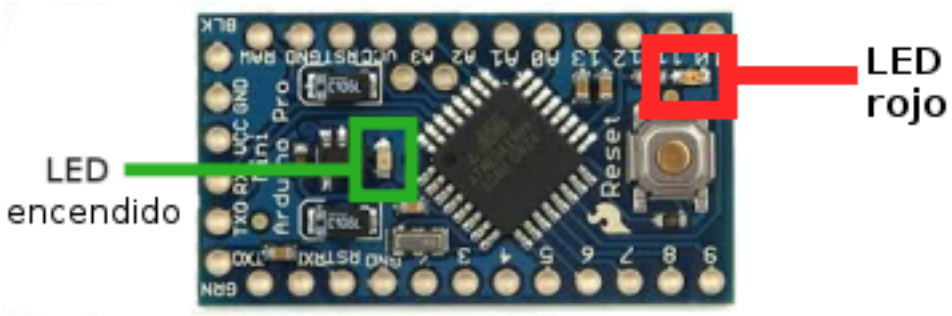

Figura 1: Ubicación del LED rojo en la placa Arduino Pro mini.

el tiempo respectivamente. Esas funciones son descriptas en las siguientes secciones, en las que realmente le encontraremos un sentido a lo que se siente al programar sistemas embebidos.

```
#include "led.h"

/************************************************************************
 *
 * Function: main
 *
 * Description: Blink the red LED once a second.
 * Notes: This routine contains an infinite loop.
 *
 ************************************************************************/
int main(void)
{
    /* configure the red LED control pin. */
    led_init( );

    while (1)
    {
        /* change the state of the red LED. */
        led_toggle( );

        /* pause for 500 milliseconds. */
        delay_ms(500);
    }
```

```
    return 0;
}
```

3 La función led_init()

Antes de comenzar a utilizar un periférico particular se debe entender el hardware utilizado para poder controlarlo.

Documentación
Toda la documentación del microcontrolador ATMEL AVR atmega328p, y de la placa arduino pro mini se encuentra disponible. Incluye las hojas de datos, los esquemáticos y los manuales de programación.

Como el LED que se necesita parpadear está conectado a uno de los pines GPIO bidireccionales del microcontrolador, nos enfocamos en estos. Frecuentemente, como es el caso con el microcontrolador atmega328p, los pines de E/S de un procesador embebido tienen varias funciones. Esto permite que el mismo pin sea utilizado como E/S controlable por el usuario o para soportar funcionalidad particular dentro del procesador. Los registros de configuración se utilizan para establecer como la aplicación utilizará cada pin especifico de un puerto.

En el atmega328p algunos pines de ciertos puertos pueden tener mas de una funcionalidad. En estos casos, un pin puede ser utilizado por el usuario (en cuyo caso se llama 'pin de propósito general') o por un periférico interno (llamado 'función alternativa del pin'). Por cada pin GPIO existen varios registros de 8 bits. Estos registros permiten configurar y controlar cada pin GPIO. La descripción del puerto que contiene los registros para configurar y controlar el pin del LED se muestra en la Figura 2.

El manual del atmega328 establece también que la configuración del pin GPIO para el LED es controlado por el bit 5 del registro DDRB (Data Direction Register). Los registros DDRX determinan si los pines se utilizarán como ENTRADA o como SALIDA. La Figura 3 muestra la descripción de los bits para los registros del puerto B, y en particular, se puede observar la ubicación del bit 5 en el registro DDRB. Este bit configura la dirección del pin GPIO que controla el LED rojo.

Estos registros de configuración y control están ubicados en el espacio de memoria, y sus direcciones se pueden observar en la Figura 3 (primer columna de la

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7...6 is used as TOSC2...1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in "Alternate Functions of Port B" on page 82 and "System Clock and Clock Options" on page 27.

Figura 2: Descripcion puerto B.

14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.4 PINB – The Port B Input Pins Address[(1)]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Figura 3: Descripcion de los registros del puerto B.

izquierda). Debido a que los registros están mapeados en memoria se los puede acceder fácilmente en C, de la misma manera que una ubicación de memoria es leída o escrita (utilizando la dirección entre paréntesis).

Bits reservados en un registro

Note, al leer el manual del atmega328, que ciertos registros contienen bits que están etiquetados como reservados. Esto es algo común en muchos registros dentro de un procesador. El manual debe indicar cómo se deben leer o escribir tales bits, por lo que es importante que no utilice esos bits para otros propósito, o modifique los mismos sin leer el manual.

Acceso a registros en espacio de E/S

Si los registros para los pines GPIO se encuentran en el espacio de E/S se los puede acceder únicamente a través de código en lenguaje ensamblador. El lenguaje ensamblador para el 80x86, por ejemplo, tiene instrucciones específicas para acceder al espacio de E/S, llamadas in y out. EL lenguaje C no tiene soporte nativo para estas operaciones. Funciones específicas en C llamadas inport y outport han sido escritas, pero son parte únicamente de los paquetes de bibliotecas estándar específicos para procesadores 80x86.

La mayoría de los registros dentro de un microcontrollador tienen una configuración predeterminada luego de un reset. Esto significa que antes de que se pueda controlar la salida de cualquier pin de E/S se debe asegurar que el pin está configurado correctamente.

Reset del software

Si se diera el caso que luego de un reset los pines están configurados como se necesitan también se debe verificar que no exista otro software que no cambió la funcionalidad de estos pines GPIO (por ejemplo un bootloader). Por lo que es una buena práctica inicializar el hardware siempre, aun cuando se verifique que el comportamiento predeterminado es el deseado.

En nuestro caso, se necesita configurar el pin GPIO 5 del puerto B como salida, a través del bit 5 del puerto DDRB (Data direction register). La máscara de bits para el pin GPIO que controla el LED rojo en la placa arduino pro mini está definido en el programa como sigue :

```
#define LED_ROJO (0x20) /* 0b00100000 */
```

Una técnica fundamental utilizada en la rutina led_init() es la lectura-modificación-escritura de un registro de hardware. Primero, se lee el contenido del registro, entonces se modifica el bit que controla el LED, y finalmente, se escribe el nuevo valor de vuelta en la ubicación del registro. El código en led_init() realiza esta operación de lectura-modificación-escritura, con el registro DDRB. Estas operaciones se realizan en lenguaje C utilizando las operaciones &= y |=. El efecto de x & = y es el mismo que el de x = x & y.

La función led_init() configura el microcontrolador atmega328 en la placa arduino pro mini para controlar el led rojo. En el código debajo note que se pone en cero el pin GPIO en el registro PORTB, para asegurarnos que el voltage de salida en el pin GPIO es establecido a cero.

```
/**************************************************************************
 *
 * Function: led_init
 *
 * Description: Initialize the GPIO pin that controls the LED.
 * Notes: This function is specific to the atmega328p.
 *
 *************************************************************************/

void led_init(void)
{
    volatile unsigned char * DDR_B = 0x24; /* direccion de DDR B */
    volatile unsigned char * PUERTO_B = 0x25; /* direccion de PUERTO B */

    /* turn the GPIO pin voltage off
     * (this should be done before the pins are configured)
     */

    *(PUERTO_B) = *(PUERTO_B) & (~ LED_ROJO);

    /* make sure the LED control pin is set to operate as OUTPUT */

    *(DDR_B) = *(DDR_B) | (LED_ROJO);
}
```

4 La función led_toggle()

Esta rutina es llamada desde un bucle infinito, y es responsable de cambiar el estado del LED. El estado del LED es controlado a través del registro PORTB. Activando (colocando un uno) el bit 5 de este puerto cambia el voltage en el pin externo a 5v (HIGH), por lo que el LED rojo se enciende. Desactivando el bit 5 de PORTB realiza la función inversa, y el voltage cambia a 0V (LOW) y el LED se apaga. Si se desea conocer por software si el LED está encendido o apagado simplemente se puede leer el bit 5 de PORTB.

Como este registro (PORTB) es de lectura escritura el estado de este bit 5 podría ser cambiado con una única instrucción en C.

```
/***************************************************************************
 *
 * Function: led_toggle
 *
 * Description: Toggle the state of one LED.
 * Notes: This function is specific to the atmega238p.
 *
 ***************************************************************************/
void led_toggle(void)
{
    volatile unsigned char * PUERTO_B = 0x25; /* direccion de PUERTO B */
    unsigned char valor_b;

    valor_b = *(PUERTO_B);

    valor_b ^= LED_ROJO;

    *(PUERTO_B) = valor_b;
}
```

5 La función delay_ms()

También necesitamos implementar una espera de 500ms entre el encendido y apagado del LED. Esta acción se realiza por medio de una espera dentro de la rutina delay_ms(). Esta rutina acepta como entrada la longitud de la espera requerida, en milisegundos, como parámetro. Dentro se multiplica ese valor por la constante CYCLES_PER_MS para obtener el número total de iteraciones que el bucle while debe realizar, para lograr la espera en tiempo requerida.

```
/* Number of decrement-and-test cycles. */
#define CYCLES_PER_MS (450)

/***************************************************************************
 *
 * Function: delay_ms
 *
 * Description: Busy-wait for the requested number of milliseconds.
```

```
 * Notes: The number of decrement-and-test cycles per millisecond
 * was determined through trial and error. This value is
 * dependent upon the processor type, speed, compiler, and
 * the level of optimization.
 *
 ***************************************************************************/
void delay_ms(int milliseconds)
{
    volatile unsigned long cycles = (milliseconds * CYCLES_PER_MS);

    while (cycles != 0)
        cycles--;

}
```

La constante especifica del hardware CYCLES_PER_MS representa el numero de veces que el procesador debe iterar a través del bucle while en un milisegundo. Para determinar este valor puede utilizar prueba y error. En secciones futuras se explica como utilizar un contador de hardware para alcanzar una mejor precisión de tiempo.

Las cuatro funciones main(), led_init(), led_toggle() y delay_ms() realizan la tarea completa del programa que parpadea el LED. Todavía se necesita entender como construir y ejecutar este programa. En los próximos dos capítulos se explican estos temas. Pero primero, realizamos algunos comentarios con respecto a los bucles infinitos y su rol en sistemas embebidos.

6 EL rol del bucle infinito

Una de las diferencias fundamentales entre programas desarrollados para sistemas embebidos y los desarrollados para otras plataformas de computación es que los programas embebidos casi siempre tienen un bucle infinito. Típicamente, este bucle rodea una parte significativa de la funcionalidad del programa, como sucede con el programa Blinking LED. El bucle infinito es necesario porque el software embebido nunca finaliza. La intención es que el programa se ejecute hasta que el mundo termine o la plataforma es reiniciada, cualquiera que suceda primero.

Además, la mayoría de los sistemas embebidos ejecutan únicamente una sola pieza de software. Aunque el hardware sea importante, el sistema no es un reloj

digital o un celular o un horno microondas sin el software. Y sin el software para la ejecución, el hardware es, simplemente, inútil. Por lo tanto, las partes funcionales de un programa embebido están siempre encerradas por un bucle infinito que se asegura de que el programa se ejecutará por siempre.

Si no se coloca el bucle infinito en el programa Blinking LED el LED hubiese parpadeado una única vez.

7 Referencias

Michael Barr. Programming Embedded Systems in C and C++ 1st Edition. ISBN-13: 978-1565923546 ISBN-10: 1565923545. O'Reilly Media; 1 edition (February 9, 1999)

8 Licencia y notas de la traducción

Rafael Ignacio Zurita (`rafa@fi.uncoma.edu.ar`) 2016-2020

Este apunte es una traducción del libro de referencia, para ser utilizado como apunte en la materia de grado 'Programación de Sistemas Embebidos' de la Facultad de Informática, Universidad Nacional del Comahue. Se han realizado modificaciones al contenido para aclarar ciertos detalles o agregar secciones nuevas. Tambien se han modificado todos los archivos fuentes de código, ya que fueron portados a la plataforma Atmel AVR atmega328.

Esta es una obra derivada del libro de referencia que ha obtenido permiso de O'Reilly para su publicación en PEDCO.

8.0.1 Permiso de publicación

```
From: Teri Finn <teri@oreilly.com>
Date: Mon, 11 Apr 2016 15:48:58 -0700
Message-ID: <CAOgscZ92-G9iT+=+nnuft4LCoHoe5o8SYPfVEKnS6AKXh8TLyQ@mail.gmail.com>
Subject: Re: Question about permission to translate some parts
To: Rafael Ignacio Zurita <rafa@fi.uncoma.edu.ar>
Content-Type: multipart/alternative; boundary=94eb2c094eeedeba0005303d5a31

--94eb2c094eeedeba0005303d5a31
Content-Type: text/plain; charset=UTF-8

Hi Rafael,

Thank you for providing this further information.  O'Reilly Media is happy
```

to grant you permission to translation the content you have proposed for posting to the Moddle site. We’re happy you find this information helpful to the students.

Best regards,

Teri Finn
O’Reilly Media, Inc.

On Thu, Apr 7, 2016 at 10:56 AM, Rafael Ignacio Zurita <
rafa@fi.uncoma.edu.ar> wrote:

[snip]