Universidad de Chile
Facultad de Cs. Físicas y Matemáticas
Departameto de Ingeniería Eléctrica
EL6908 - Introducción al Trabajo de Título

Contextualización

EL6908 - Introducción al Trabajo de Título

Diseño e Implementación del Software de Control para el Computador a Bordo de un Pico-Satélite

Autor : Carlos González C.
Profesor Guía : Marcos Díaz Q.
Profesor EL6908 : Jorge Lopez H.
23 de noviembre de 2012
Santiago, Chile.

Índice

Índice de figuras

Índice de cuadros

1. Contextualización

En este capítulo se discuten los aspectos teóricos relacionados con el desarrollo de un proyecto de software para sistemas embebidos pensados en ser utilizados en misiones aeroespaciales, en específico el control del computador a bordo de un pico-satélite.

1.1. Sistemas embebidos

Los sistemas embebidos a diferencia de un computador personal que es usado con fines generales para una amplia variedad de tareas, son sistemas computacionales normalmente utilizados para atender una cantidad limitada de procesos, realizar tareas específicas o dotar de determinada inteligencia a un sistema más complejo. Un sistema embebido está compuesto por uno o más microcontroladores pequeños que cuentan con periféricos para manejar diferentes protocolos de comunicación; conversores ADC; timers; puertos de entrada y salida digitales, todo en integrado en un mismo chip para guardar espacio y ahorrar energía. Parte fundamental de un sistema embebido es el software que provee la funcionalidad final, usualmente se usa el término firmware para referirse a este código con que se programa el microcontrolador el cual por lo general es específico para la plataforma de hardware y se relaciona a muy bajo nivel. A diferencia de un computador de propósito general donde el usuario puede cargar una serie de programas en él para un amplio rango de usos, el usuario de un sistema embebido no tiene la capacidad de reprogramarlo fuera de las posibilidades que el desarrollador ha brindado al sistema[8].

Para el diseño de sistemas embebidos se debe considerar ciertos aspectos que los diferencian de otros tipos de sistemas de computacionales, tales como[7]:

- Un sistema embebido se mantiene siempre funcionando y debe proveer respuesta en tiempo real. Se debe diseñar considerando una operación continua y una posible reconfiguración del sistema estando ya en marcha.
- Las interacciones con el sistema pueden ser impredecibles y no se tiene control sobre ellas. Existen sistemas que son controlados por el usuario mediante una interfaz preparada para ellos, mientras que otros sistemas deben atender eventos imprevistos sin dejar de realizar tareas rutinarias.
- Existen limitaciones físicas. Normalmente estos sistemas poseen limitadas características de: poder de cómputo, memoria de datos y de programa; espacio físico; y disponibilidad de energía.
- El diseño de software para sistemas embebidos requiere una interacción de bajo nivel. Existe una amplia gama de plataformas de hardware para desarrollar sistemas embebidos y se requiere interactuar también con una amplia gama de dispositivos externos. Por esto se requiere desarrollar capas de drivers de periféricos que oculten las diferencias de hardware a la aplicación final del sistema.
- Es importante considerar aspectos de seguridad y confiabilidad del sistema durante todo su desarrollo debido a que la mayoría de los sistemas embebidos son usados para controlar otros sistemas críticos en diversos procesos.

1.1.1. Microcontroladores PIC

Todo sistema embebido está formado fundamentalmente por un microcontrolador que brinda la capacidad de cómputo y el control de diferentes periféricos que normalmente están integrados en el mismo chip. Entre

los principales fabricantes de microcontroladores se encuentran: Microchip, Texas Instrument, ARM, Motorola, NVidia. Este trabajo se concentra en los microcontroladores PIC desarrollados por la compañía Microchip. La familia de microcontroladores PIC es bastante amplia adaptándose a un amplio rango de necesidades, la tabla 1 resume las principales características de los diferentes modelos y puede ser utilizada como una guía para determinar el dispositivo adecuado según la aplicación:

Cuadro 1: Guía de microcontroladores PIC

Familia	Instrucciones	Datos	Memoria Programa	Memoria RAM	Velocidad	Periféricos	Usos
PIC10	12 bit	8 bit	512 Words	64 Bytes	16MHz	IO, ADC	Espacio reducido, bajo costo. Lógica digital, control IO
PIC12	12 bit	8 bit	4 Kwords	256 Bytes	32MHz	IO, ADC, TIMER, USART	Bajo costo. Logica digital, control IO, sensores
PIC16	14 bit	8 bit	16 Kwords	2 Kbytes	48MHz	IO, ADC, TIMER, USART, PWM	Control, sensores, recolección de datos, display, interfaz serial.
PIC18	16 bit	8 bit	64 Kwords	4 Kbytes	64MHz	IO, ADC, TIMER, USART, PWM, USB, CAN, ETHERNET, LCD	Control, sensores, datos, interfaz serial, ethernet, display.
PIC24	24 bit	16 bit	512 Kbytes	96 Kbytes	70 MIPS	IO, ADC, TIMER, USART, PWM, USB, CAN, ETHERNET, RTCC, DMA, CRC, PPS	Uso general
dsPIC	24 bit	16 bit	512 Kbytes	54 Kbytes	70 MIPS	IO, ADC, DAC, TIMER, USART, PWM, USB, CAN, ETHERNET, RTCC, DMA, CRC, PPS, CODEC, QEI	Uso general. Procesamiento señales. Control de motores.
PIC32	32 bit	32 bit	512 Kbytes	128 Kbytes	80 MHz	IO, ADC, TIMER, USART, PWM, USB, CAN, ETHERNET, RTCC, DMA, CRC, PPS, CODEC, CTMU	Uso general
Los valores representan el tope de linea para cada familia							

A continuación se describen las características específicas de los microcontroladores PIC24 que corresponde al dispositivo utilizado en este trabajo.

Arquitectura Poseen un juego de instrucciones *Reduced Instruction Set Computing* (RISC) (80 instrucciones) de ancho fijo en 24 bits que en su mayoría se ejecutan en un solo ciclo excepto: divisiones; cambios de contexto; y acceso por tabla a memoria de programa[3]. Se basa en una arquitectura Harvard modificada de 16 bits de datos[2] lo que significa que el dispositivo posee una memoria de datos tipo *Random Access Memory* (RAM) separada de la memoria de datos (FLASH) pudiendo acceder de manera independiente e incluso simultáneamente a las instrucciones del programa y a los datos de este alojados en RAM. La arquitectura de la CPU la completan una *Arithmetic Logic Unit* (ALU) con *hardware* dedicado para realizar multiplicaciones y divisiones. El detalle de la arquitectura del microcontrolador PIC24 se detalla en la figura 1. También posee un vector de hasta 128 interrupciones con capacidad para atender hasta 8 de ellas lo que permite liberar al procesador de la espera de sucesos asíncronos ya que son notificados y atendidos de manera específica en una rutina de atención de la interrupción.

Periféricos La familia de microcontroladores PIC24F integra en el mismo chip una serie de periféricos que permiten realizar funciones específicas a través de hardware especialmente diseñado. Esto hace que el PIC24F se convierta en un sistema embebido capaz de ser utilizados en aplicaciones que requieran: conversores análogos a digitales; temporizadores; comunicación síncronas y asíncronas como RS232, SPI o I2C; USB o Ethernet; manteniendo acotados los costos del sistema. Una lista de los periféricos disponibles para estos microcontroladores se detalla en la figura 2. El acceso para configurar, guardar y obtener datos de estos periféricos se

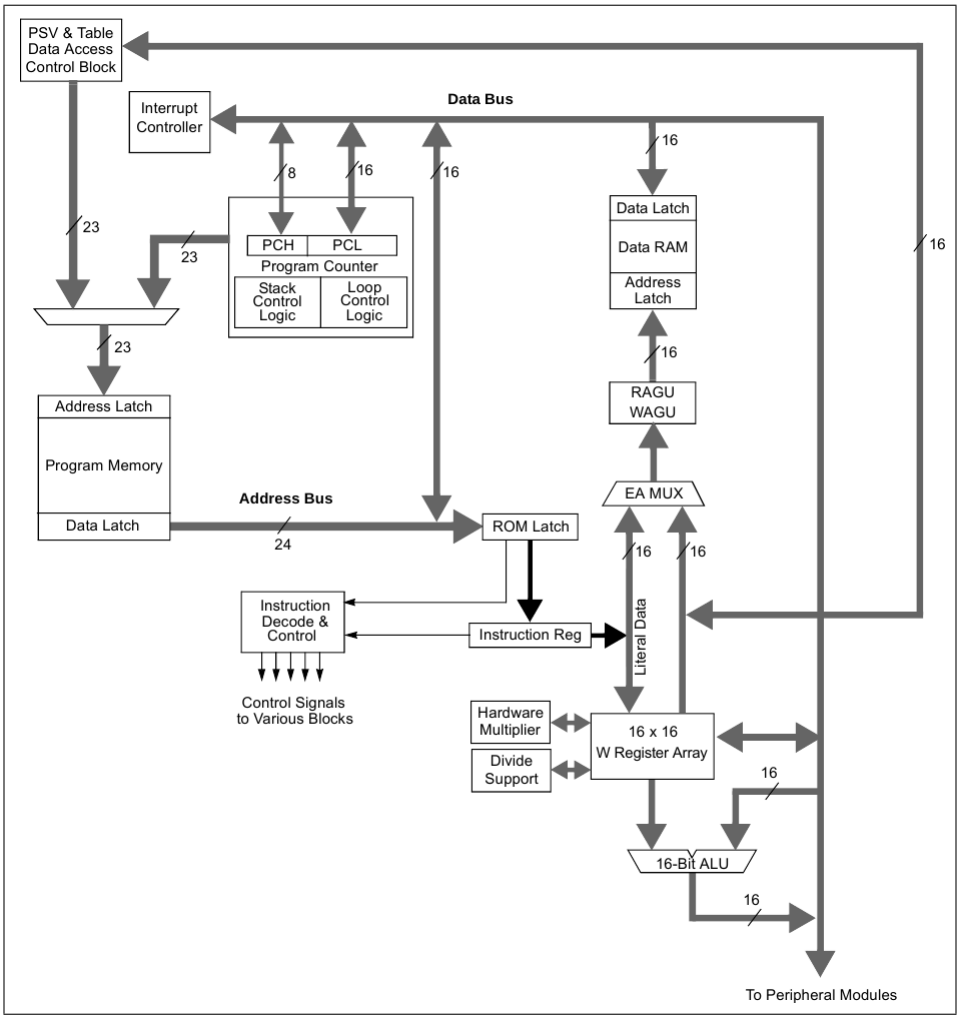

Figura 1: Arquitectura de la CPU del PIC24F

realiza a través de registros que están mapeados en la memoria de datos del microcontrolador por lo tanto comparten el bus de datos y no son necesarias instrucciones extras para su integración. Junto con una completa documentación de la arquitectura y funcionamiento de cada periférico, los compiladores de lenguaje C de Microchip proveen librerías para acceder a estas funcionalidades a través de una API de más alto nivel.

1.2. Sistemas Operativos

Un sistema operativo es la aplicación base de un sistema computacional pues brinda servicios básicos al resto de las aplicaciones de uso general que se ejecutan en el computador. El sistema operativo es la capa entre el hardware y las aplicaciones. El hardware puede variar considerablemente entre un sistema y otro, por eso se necesita una capa de abstracción que haga a la aplicación independiente de la plataforma en que se ejecuta. Para esto el sistema operativo provee servicios que usan interfaces de bajo nivel con el hardware las cuales no están disponibles para la aplicación. La figura 3 es un esquema que ilustra un sistema operativo como una capa de abstracción del hardware. Ejemplos de sistemas operativos los son UNIX, GNU/Linux, FreeRTOS, entre otros.

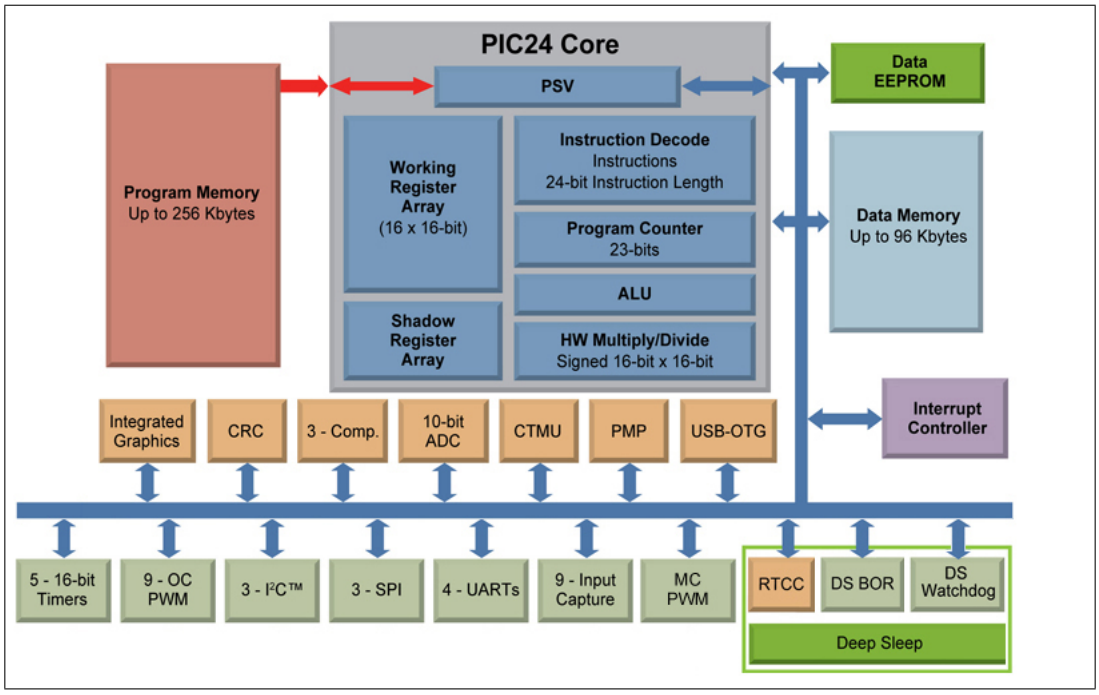

Figura 2: Arquitectura del PIC24F

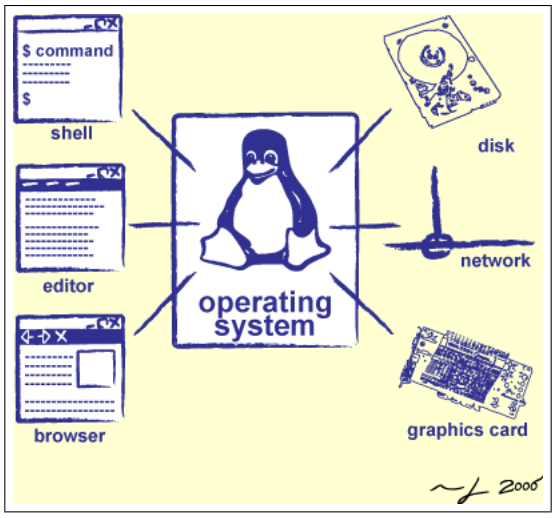

Figura 3: Sistema Operativo como capa de abstracción

1.2.1. Sistemas Operativos de Tiempo Real

Parte fundamental de un sistema operativo es su *scheduler*, módulo encargado de intercambiar entre las múltiples tareas que se deben ejecutar cediendo un espacio de tiempo para utilizar le procesador. La forma en que trabaja el *scheduler* define el tipo de sistema operativo que se posee. Un sistema operativo de tiempo real posee un *scheduler* diseñado para proveer un flujo de ejecución determinista, pues solo sabiendo con exactitud la tarea que el sistema ejecutara en un determinado momento se pueden cumplir los requerimientos estrictos de *timing*[12]. Esto es un aspecto de especial interés en sistemas embebidos que normalmente requieren respuesta

en tiempo real ante eventos no predecibles.

La figura 4 demuestra la forma de conseguir un sistema de tiempo real mediante el uso de prioridades para las diferentes tareas. En este ejemplo la mayor parte del tiempo el sistema está en estado *idle*, sin código que ejecutar. Sin embargo ante la precencia de ciertos evento el sistema debe responder de manera instantánea cambiado de contexto a la tarea correspondiente. Ciertas tareas pueden requerir un estricto *timing* ejecutándose de manera periódica, por ejemplo. En este caso se asigna una alta prioridad para asegurar que el sistema operativo siempre ejecutara esta tarea cuando corresponda.

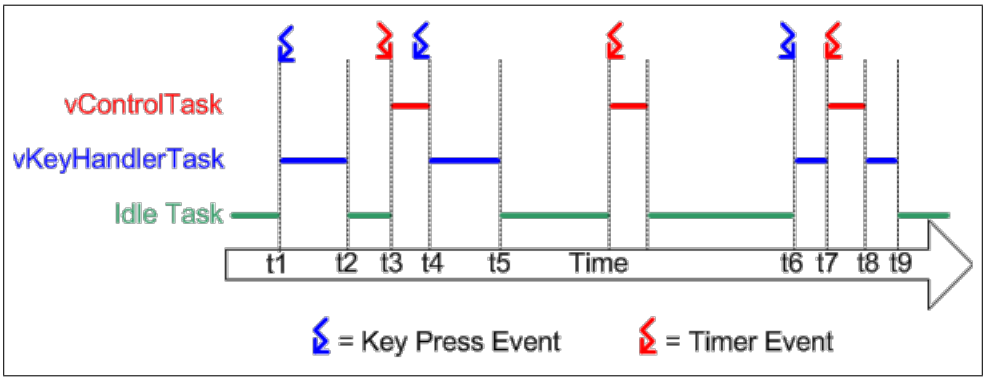

Figura 4: *Real time scheduling*

1.2.2. FreeRTOS

FreeRTOS es un tipo de *Real Time Operating System* (RTOS) que está diseñado para ser lo suficientemente pequeño como para ser utilizado en un microcontrolador como sistemas embebidos[12]. Como estos sistemas son realmente limitados, normalmente no existe la posibilidad de ejecutar un sistema operativo completo como GNU-Linux, pero FreeRTOS provee un kernel capaz de manejar múltiples tareas con prioridades; comunicación entre tareas; *timing* y primitivas de sincronización. Por su reducido tamaño no provee funcionalidades de alto nivel como una consola de comandos; así como tampoco funcionalidades de bajo nivel como controladores para el hardware o periférico. Entre sus principales características se encuentran:

- *Scheduler pre-emptive* o cooperativo.
- Sincronización y comunicación entre tareas a través de colas, semáforos, semáforos binarios y mutexes.
- Mutexes con herencia de prioridades.
- Software *timers*.
- Bajo consumo de memoria (Entre 6K y 10K en ROM).
- Altamente configurable.
- Detección de *stack overflow*
- Soporte oficial a 33 arquitecturas de sistemas embebidos.
- Estructura de código portable, escrito en C.

- Licenciado bajo GPL modificada que permite su uso comercial sin publicar código fuente.
- Gratuito
- Amplia documentación, foros y asistencia técnica.

Funcionamiento Los conceptos fundamentales detrás del funcionamiento de FreeRTOS son las tareas y el *scheduler*. Una tarea es un hilo de procesamiento, normalmente una función que se ejecuta de manera continua. Una tarea se puede encontrar dos estados fundamentales: ejecutándose y no ejecutándose. Cuando una tarea se está ejecutando tiene el control del procesador y el código dentro de la función que representa a la tarea es procesado. El estado "no ejecutándose" en realidad consta de tres sub-estados como se observa en la figura5: se inicia en un estado "listo" lo que indica que la tarea está en condiciones de ser selecciona por el *scheduler* y pasar a estado "ejecutándose"; el estado "bloqueado" significa que la tarea no está disponible para ser ejecutada pues está en espera de algún evento, por ejemplo, la liberación de un mutex; cuando la tarea está en estado "suspendida" tampoco se puede ejecutar, la tarea debe explícitamente reanudarse para quedar en condiciones de ser ejecutada.

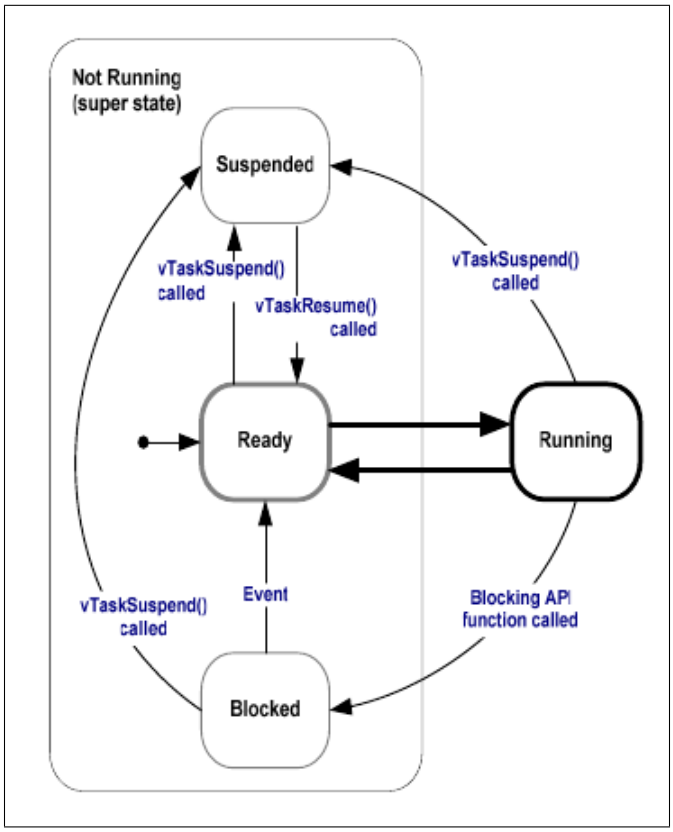

Figura 5: Tareas de FreeRTOS, diagrama de estados

La creación de taras y el control de sus estados se realiza a través de la API de FreeRTOS que documenta claramente todas las posibles operaciones que se pueden realizar sobre una tarea.

El *scheduler* es la parte fundamental del kernel que controla la ejecución de las diferentes tareas disponibles. Su objetivo es generar la sensación de estar en un ambiente multitarea, cuando en realidad sólo una tarea puede

ejecutarse a la vez ya que se posee un solo procesador. Como se detalla en la figura 6 la función del *scheduler* es entregar una porción de tiempo de ejecución fijo a una tarea y una vez que este tiempo se cumple se debe guardar el estado de ejecución de esta tarea y se procede a ejecutar otra. Así cada una de las tareas se ejecuta durante una pequeña porción de tiempo, hasta que completa su trabajo; si el tiempo de proceso asignado a cada tarea es lo suficientemente pequeño pareciera que muchas cosas ocurrieron simultáneamente. Claramente una sola tarea tomaría, en términos absolutos, menos tiempo en terminar si no fuera interrumpida, pero se gana un sistema más fluido cuando se deben ejecutar en conjunto tareas que toman mucho tiempo de proceso y otras relativamente cortas.

Figura 6: *Scheduling* de tareas

El algoritmo de *scheduling* se basa en un sistema de prioridades, donde la tarea en condiciones de ejecutarse que tenga la mayor prioridad siempre debe ser ejecutada. Si varias tareas en estado "listo" comparten la misma prioridad se aplica un algoritmo de *round-robing*. Las tareas que están en estado "suspendido" y "bloqueado" nunca son seleccionadas por el *scheduler* y por lo tanto no consumen recursos. Haciendo un correcto uso de las prioridades y los diferentes estados se consigue un sistema que se ejecuta de manera fluida y haciendo un uso óptimo del procesador.

1.3. Ingeniería de Software

1.3.1. Licencias de Software

Se entiende por licencia de software a un contrato entre el desarrollador del software y el usuario final para su utilización según una serie de términos o condiciones. La licencia puede ceder ciertos derechos al usuario final; controlar la cantidad de copias que puede utilizar; el ámbito geográfico y temporal para la utilización; o bien proteger al desarrollador frente a la utilización del programa informático que se licencia. Existen al menos tres tipos de licencias de software:

- **Privativas:** El software es distribuido al usuario bajo un *End-User License Agreement* (EULA) en que el propietario fija las condiciones de uso y se reserva la propiedad del programa informático. Generalmente

impide el acceso al código fuente; la realización de ingeniería inversa; el uso del software por más de un usuario; se entrega el derecho de uso por un tiempo definido y por lo general provee cierta asesoría técnica. Es común que se debe pagar por el uso de un programa bajo este tipo de licencia.

- **Libres:** Este tipo de licencias otorgan al receptor la libertad de usar, estudiar, compartir y modificar el software. Existen varias licencias que cumplen con esta definición, por ejeplo: MIT, BSD, **LGPL!** (**LGPL!**) y GPL. Se dividen en dos tipos básicos licencias con *copyleft* y sin *copyleft*. Se habla de una licencia con *copyleft* cuando esta indica que el software derivado debe mantener la misma licencia original impidiendo la generación de software privativo a partir de desarrollos libres, por ejemplo. Por lo general cumplir esta licencia requiere que se garantice el acceso al código fuente, de aquí el termino conocido como software de código abierto. La mayoría del software bajo estas licencias se distribuye de forma gratuita aunque su uso comercial no está necesariamente prohibido. No todo software gratuito es necesariamente software libre.
- **Dominio Público:** Si un programa informático se distribuye sin ningún tipo de licencia, se dice que es de dominio público. No posee ningún tipo de restricción sobre su uso, así como ninguna responsabilidad sobre sus creadores.

La aplicación de licencias se rige según las normativas legales locales. En el caso de las licencias libres por lo general basta con distribuir el texto de la licencia junto con la aplicación y agregar un encabezado indicando el tipo de licencia que se utiliza. Para atribuir autoría se suele utilizar las definiciones de la Convención de Berna, que indica que todo lo que se escribe queda automáticamente sujeto a copyright desde el momento en que la obra es fijada en un soporte material[11]. En Chile la legislación vigente al respecto es la Ley 17.336 de propiedad Intelectual.

1.3.2. GPL

La GPL es una licencia de software creada en 1989 por la *Free Software Foundation* que busca declarar que el software así licenciado es software libre y por lo tanto el usuario posee los siguientes derechos[10]:

- Libertad de usar el software para cualquier propósito.
- Libertad de modificar el software para adaptarlo a las propias necesidades.
- Libertad para compartir el software.
- Libertad para publicar los cambios que se han realizado.

Actualmente se encuentra en su versión 3.0 que a diferencia de versiones previas es una licencia con *copyleft* y agrega cláusulas para proteger la libertad del usuario frente a nuevas prácticas en contra del software libre, tales como[10]:

- Tivoización: El término se refiere a la práctica de limitar los derechos de los usuarios que compran sistemas que funcionan con software libre mediante mecanismos de hardware, por ejemplo evitando la ejecución de versiones modificadas del software embebido.
- Leyes que prohíben software libre: Se asegura de que ciertas leyes puedan limitar los derechos del usuario de un software con licencia GPL.
- Uso malicioso de patentes: Evita el uso indiscriminado de patentes, como por ejemplo, intentar obtener beneficios patentado desarrollos de software libre lo cual es una amenaza a la liberta de los usuarios.

Aplicación de la licencia Para licenciar un proyecto de software libre bajo la GPL V3.0 se deben seguir los siguientes pasos[11]:

1. Agregar un aviso informativo del *copyright* al inicio de cada archivo con código fuente de la aplicación. Un ejemplo es la siguiente línea: «Copyright 2012 Universidad de Chile»
2. Bajo el aviso de *copyright* se agrega una autorización de copia indicando que el software se distribuye bajo los términos de la licencia GPL. Un ejemplo de este aviso se cita en un ejemplo posterior.
3. Se debe incluir junto al código fuente el texto completo de la licencia en un archivo llamado `LICENSE`. El texto de la licencia se puede obtener desde el siguiente enlace: `http://www.gnu.org/licenses/gpl.txt`.

A continuación se detalla el encabezado que debería incluir cada fichero del proyecto de software hipotético llamado `Foobar` que es licenciado bajo GPL.

```
Foobar - Description - «Copyright 2012 Universidad de Chile»

This file is part of Foobar.

Foobar is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

Foobar is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with Foobar.  If not, see <http://www.gnu.org/licenses/>.
```

Referencias

[1] Microchip, *MPLABX IDE User's Guide*, USA: Microchip Technology Incorporated, 2011.

[2] Microchip, *PIC24F Family Reference Manual*, USA: Microchip Technology Incorporated, 2006.

[3] Microchip, *16 bit MCU and DSC Programmer's Reference Manual*, USA: Microchip Technology Incorporated, 2011.

[4] Microchip, *PIC24FJ256GA110 Datasheet*, USA: Microchip Technology Incorporated, 2006.

[5] R. Barry, *Using the FreeRTOS Real Time Kernel. A Practical Guide, PIC32 Edition.*, 1st ed., USA: Real Time Engineers, 2009.

[6] R. Barry, *The FreeRTOS Reference Manual*, 1st ed., USA: Real Time Engineers, 2011.

[7] I. Sommerville, *Software Engineering*, 9nd ed., USA: Addison-Wesley, 2011.

[8] S. Heat, *Embedded Systems Design*, 2nd ed., England: Newnes, 2003.

[9] L. Di Jasio *Programming 16-Bit PIC Microcontrollers in C* , 1st ed. USA: Elsevier, 2007.

[10] Free Software Foundation. *GNU General Public License* [En Línea]. USA, 2011. `http://www.gnu.org/licenses/gpl.html` [Consulta: 22 noviembre 2012]

[11] Free Software Foundation. *Preguntas frecuentes acerca de las licencias de GNU* [En Línea]. USA, 2011. `http://www.gnu.org/licenses/gpl-faq.html` [Consulta: 22 noviembre 2012]

[12] Real Time Engineers. *About FreeRTOS* [En Línea]. USA, 2012. `http://www.freertos.org/about-RTOS.html` [Consulta: 22 noviembre 2012]