UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

TESIS

"IMPLEMENTACIÓN DE TAREAS EN UN SISTEMA EMBEBIDO DE TIEMPO REAL".

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

PRESENTA:

FERNANDO BUSTILLOS BOJÓRQUEZ

DIRECTOR DE TESIS:

M.C. JORGE EDSON LOYA HERNÁNDEZ

TIJUANA, B.C., MÉXICO FEBRERO DE 2012

Universidad Autónoma de Baja California

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

COORDINACIÓN DE POSGRADO E INVESTIGACIÓN

FOLIO No. 069

Tijuana, B. C., a 17 de enero de 2012

C. Fernando Bustillos Bojórquez
Pasante de: Maestro en Ciencias
Presente

El tema de trabajo y/o tesis para su examen profesional, en la Opción TESIS

Es propuesto, por el C. M.C. Jorge Edson Loya Hernández

quienes serán los responsables de la calidad de trabajo que usted presente, referido al tema "Implementación de tareas de un sistema embebido de tiempo real". el cual deberá usted desarrollar, de acuerdo con el siguiente orden:

I.- INTRODUCCIÓN
II.- MARCO TEORICO
III.- MARCO EXPERIMENTAL
IV.- RESULTADOS
V.- CONCLUSIONES Y TRABAJO FUTURO

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS QUIMICAS E INGENIERIA

Q. Noemí Hernández Hernández
Sub-Director Secretario

M.C. Jorge Edson Loya Hernández
Director de Tesis

Dr. Luis Enrique Palafox Maestre
Director

Resumen

En este trabajo se muestra un estudio introductorio de los sistemas empotrados de tiempo real. Está dirigido a presentar las bases sobre el uso de un sistema operativo de tiempo real comercial, con la finalidad de que la información que aporta este documento sea de utilidad tanto para alumnos como profesores dentro del área de sistemas digitales que deseen construir un sistema de tiempo real bajo el esquema que se menciona en este trabajo.

Se muestran los fundamentos para la creación, implementación y administración de tareas en un sistema operativo de tiempo real para el desarrollo del proyecto con ayuda del sistema operativo μC/OS II de Micrium y un microprocesador de alto rendimiento con arquitectura *ARM*. Se utiliza la medición de la *potencia óptica* a manera de experimento con el fin de estudiar el sistema operativo de tiempo real. Para incorporar el uso del sistema operativo en aplicaciones futuras, se muestra la programación y la metodología de creación de las tareas.

Agradecimientos

Un agradecimiento especial a mi tutor y director de tesis M.C. Jorge E. Loya por su paciencia, conocimientos y experiencia compartida a lo largo de este tiempo.

A mis sinodales Dr. Eduardo Álvarez, M.C. Diego A. Trujillo y M.C. Norma O. Bravo por haber dedicado su tiempo, conocimiento y sugerencias durante la revisión de este trabajo de tesis.

Al Dr. Cesar Diaz Trujillo y la Dra. Adriana Nava por su gran apoyo y confianza en los momentos difíciles durante los primeros semestres de la maestría.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo de beca, con el cual fue posible cubrir el costo de manutención y colegiatura para poder llevar acabo la realización de este humilde trabajo y la obtención del grado de maestro en ciencias.

A la Universidad Autónoma de Baja California (UABC) por haberme aceptado como estudiante de posgrado y haberme facilitado la utilización de su equipo e infraestructura durante este tiempo.

A mis padres por todo el amor desinteresado, paciencia, fueron y seguirán siendo un pilar fundamental en mi formación a lo largo de la vida, gracias a ellos hoy puedo alcanzar una de mis metas. A mi hermana, por todo su cariño brindado. A todos mis tíos que siempre me han ayudado, en especial a mi tía Adriana por su inmensurable apoyo moral y compañía.

Índice

Índice iv

Índice de figuras vii

1. Introducción 1

1.1. Objetivo 4

1.2. Metas 5

1.3. Metodología 5

2. Marco Teórico 7

2.1. Conceptos Generales 7

2.2. Planificadores 13

2.3. Técnicas de Planificación 15

2.3.1. Planificación con prioridades fijas (FPS) 17

2.3.2. Otros algoritmo de planificación 32

3. Marco Experimental 34

3.1. Descripción del Experimento 34

3.2. Descripción de la aplicación utilizada 35

3.3. Implementación de la aplicación 36

3.4. Descripción de las funciones y tareas del proyecto 40

3.4.1. Función `main()` 40
3.4.2. Tarea inicial 43
3.4.3. Tarea del sensor 46
3.4.4. Tarea del LCD 48
3.4.5. Función de la rutina de interrupción 49
3.5. Descripción breve del sensor TSL230R-LF 50
3.6. Descripción breve del RTOS μC/OS-II 51
3.7. Descripción breve del sistema de desarrollo utilizado 52

4. Resultados **53**
4.1. Análisis de la ejecución de las tareas 53
4.2. Medición de la Potencia Óptica 58

5. Conclusiones y trabajo futuro **60**

Bibliografía **63**

Anexos **67**

A. Gráficas de tareas **68**

B. Herramientas Utilizadas **71**
B.1. Ambiente de desarrollo IAR Embedded Workbench 71
B.2. Tarjeta de Desarrollo IAR STR750-SK 72
B.3. Sistema Operativo de Tiempo Real μC-OS II 75
B.4. Sensor TSL230R-LF 77

C. Código de Tareas **78**

C.1. Descripción de las funciones y tareas de la aplicación 78

C.1.1. Función main() 78

C.1.2. Función AppTareaInicial() 80

C.1.3. Función AppTareaSensor() 83

C.1.4. Función AppTareaLCD() 85

C.1.5. Función EXTINT_ISRHandler() 87

Glosario **89**

Índice de figuras

1.1. Aplicaciones de sistemas empotrados. 3

2.1. Sistema empotrado de tiempo real. 9
2.2. Diagrama de estado con los límites entre un programa, una tarea y un proceso. 11
2.3. Límite de utilización para cumplir la condición para que un conjunto de tareas sea planificable basada en el factor de utilización. 20
2.4. Línea de tiempo del conjunto de procesos A. 22
2.5. Línea de tiempo del conjunto de procesos B. 24
2.6. Línea de tiempo del conjunto de procesos C. 26

3.1. Diagrama general a bloques de la aplicación. 35
3.2. Diagrama general de las tareas. 37
3.3. Diagrama de flujo de la función `main()`. 40
3.4. Diagrama de flujo de la tarea inicial. 43
3.5. Diagrama de retrasos. 45
3.6. Diagrama de flujo de la tarea del sensor. 46
3.7. Diagrama de flujo de la tarea del LCD. 48
3.8. Diagrama de flujo de la rutina de interrupción. 49

4.1. Diagrama general de tiempos. 53

4.2. Tiempo de ejecución de las tareas. 54
4.3. Diferencial de tiempo de 9,4μs. 54
4.4. Comportamiento del conjunto de tareas. 56

A.1. Diagrama general de tiempos. D_0= Rutina del *tick*; D_1= Ejecución de la tarea inicial; D_2= Ejecución de la tarea del sensor; D_4= Tarea del μC/OS-II cuando no se está ejecutando ninguna tarea de aplicación. . . 68
A.2. Tiempo de ejecución de AppTareaSensor. 69
A.3. D_0,Rutina del Tick; D_2,Ejecución de la tarea AppTareaSensor. 69
A.4. D_2= Ejecución de la tarea AppTareaSensor; D_1= Ejecución de la tarea AppTareaInicial. 70
A.5. D_0= Ticks; D_1= Tarea c; D_2= Tarea a; D_5= Tarea b. 70

B.1. Diagrama a bloques del sensor TSL230R-LF. 77

Capítulo 1

Introducción

El rápido crecimiento de los sistemas electrónicos con los que interactúa un ser humano en su vida diaria ha propiciado un aumento en la diversidad y naturaleza de las actividades que desarrolla. Actualmente, algunas de las funciones desarrolladas por esos sistemas electrónicos son: intercambiar información entre sí, realizar actividades de cálculo intenso, poseer capacidad de interconexión con otros sistemas, monitorear y adquirir cúmulos de datos. En la actualidad se utilizan computadoras con aplicación específica llamados sistemas empotrados para realizar dichas funciones.

Un sistema empotrado (*embedded system*) es un sistema electrónico que se distingue de una computadora personal por tener recursos limitados y operar bajo restricciones propias de la tarea que está diseñado a realizar [1]. Por lo general los sistemas empotrados de baja complejidad tienen limitaciones en cuanto a tamaño de memoria, velocidad de operación, requerimientos de energía, operación en ambientes rudos, cantidad y características de los dispositivos de entrada y salida.

El desarrollo de funciones en sistemas empotrados dentro de parámetros esperados de operación, requiere la utilización de un conjunto de elementos de hardware y software

que contribuyan a la operación del sistema electrónico en forma segura y confiable. Además del uso de los componentes adecuados, la construcción física estandarizada y la correcta integración de tecnologías de vanguardia, un elemento que permite añadir flexibilidad y robustez a un sistema electrónico es el uso de un sistema operativo de tiempo real [2].

Es por esta razón que se contempla fortalecer el área de investigación en el diseño de sistemas empotrados dentro del programa de maestría y doctorado en ciencias de la ingeniería (MYDCI) de la facultad de ciencias químicas e ingeniería de la UABC. En forma adicional, se persigue formar un grupo de trabajo que pueda contribuir a las futuras actividades de diseño de sistemas electrónicos de alto desempeño que se han programado para la región. Además, se contribuirá a la formación de egresados con alto grado de especialización que logren impactar en el desarrollo de la industria electrónica regional, de acuerdo con las iniciativas mostradas en [3] y [4].

Con este antecedente, surge la idea de explorar un sistema operativo de tiempo real (Real Time Operating System; RTOS, por sus siglas en inglés) comercial como una de las actividades primarias de la línea de generación y aplicación del conocimiento en sistemas empotrados, para posteriormente utilizarla como plataforma al momento de implementar etapas o módulos de sistemas en tiempo real.

Como se muestra en la figura 1.1 los sistemas empotrados son aplicados en numerosas áreas como puede ser en: medicina, para el monitoreo de funciones vitales, sistemas de reparto de medicamentos, marcapasos, prótesis, sistemas de diálisis; en la industria automotriz se utilizan en el control de ignición, sistemas de aire acondicionado, sistema

de frenado ABS, control de bolsas de aire; mientras que en la industria aeroespacial los sistemas empotrados se emplean en sistemas de navegación, instrumentación, prevención de colisiones; en el área de manufactura se realizan tareas de automatización, control de procesos, manipuladores de robot, disparo de sistemas de potencia; para aparatos electrodomésticos y de oficina como por ejemplo estufas programables, cafeteras, hornos de microondas, máquinas de fax, copiadoras; también son utilizados en sistemas de comunicación y cómputo, en enrutadores, conmutadores, impresoras, escáneres; y además en sistemas de entretenimiento como pueden ser reproductores de música, videojuegos portátiles y juguetes simples [1, 5].

Figura 1.1: Aplicaciones de sistemas empotrados.

1.1. Objetivo

Realizar la implementación de una aplicación en tiempo real en un sistema empotrado. La implementación consiste en varios pasos. Primero es necesario dividir el funcionamiento de la aplicación a realizar en tiempo real en módulos, de manera de que cada uno realice una función específica e independiente. Una vez que se ha identificado los diferentes módulos, se procede a realizar los algoritmos y su respectivo programa en lenguaje C para cada uno de éstos, donde los programas serán las tareas a administrar por el RTOS. Posteriormente se realiza una asignación de prioridades a las tareas en base al algoritmo de calendarización a utilizar. Y por último se utilizan funciones propias del RTOS para crear y administrar las tareas en tiempo real.

Se utilizará la aplicación de un medidor de potencia óptica donde el funcionamiento es divido en tres módulos. Un módulo es encargado de atender a un sensor, el segundo módulo se encarga de determinar los botones que el usuario eventualmente presione y el tercero llevará a cabo el despliegue de información en la pantalla. Además se agrega un módulo adicional para que posteriormente sea posible agregar alguna otra función a la aplicación. La aplicación del medidor de potencia óptica permitirá mostrar el procedimiento para implementar un sistema empotrado en tiempo real mediante la ejecución de tareas independientes en un procesador de alto rendimiento con la ayuda de un RTOS, con el fin de determinar los requerimientos de utilización del CPU así como su comportamiento. Con la realización de este trabajo se podrá identificar las restricciones para crear y administrar tareas asociadas a una aplicación particular en tiempo real.

1.2. Metas

1. Estudiar la forma de trabajar del planificador en un sistema operativo de tiempo real.

2. Establecer los fundamentos para la creación, implementación y administración de tareas para un sistema con restricciones de procesamiento en tiempo real.

3. Sentar las bases para el desarrollo futuro de módulos que formen parte de un sistema empotrado de tiempo real.

1.3. Metodología

Para poder llevar a cabo la realización de este trabajo es necesario iniciar con el estudio de la arquitectura de un procesador de alto rendimiento *ARM* y una introducción a su programación, así como un estudio de las técnicas más comunes de planificación en tiempo real.

Posteriormente, con el fin de explorar la tarjeta de desarrollo y conocer la herramienta de trabajo, hay que realizar aplicaciones básicas y sencillas con los módulos, como son los temporizadores, puertos de entrada y salida, de esta manera es posible entender el procedimiento de programación del microprocesador y depuración de tareas utilizando el ambiente de desarrollo. Así mismo es necesario explorar el funcionamiento del sistema operativo de tiempo real para conocer la manera de crear y ejecutar múltiples tareas. Así como implementar una aplicación básica que sea posible ejecutarse en la tarjeta de desarrollo disponible, la cual consiste en un medidor de *potencia óptica* con fines académicos.

También es necesario estudiar las funciones básicas del sistema operativo de tiempo real para llevar a cabo la implementación del medidor de *potencia óptica*. Para implementar la aplicación se hace un análisis con el fin de determinar si una tarea es planificable, se programan las tareas de forma independiente de manera que cada tarea cumpla con su objetivo en particular sin la necesidad de depender de otra tarea, se establecen valores correspondientes a la medición del sensor de luz, se realiza la programación y depuración en la memoria de la tarjeta de desarrollo y la ejecución de dicho programa para verificar su operación, posteriormente con ayuda de un analizador lógico se miden los tiempos para observar el comportamiento de las tareas ejecutadas en el procesador, y así analizar la manera en el que el sistema operativo de tiempo real interrumpe las tareas de menor prioridad para ejecutar tareas de mayor prioridad.

Capítulo 2

Marco Teórico

En este capítulo se definen los conceptos generales relacionados a sistemas empotrados de tiempo real lo cual está directamente relacionado con el trabajo que se plantea en esta tesis, así como una descripción de los planificadores y las técnicas básicas de planificación.

2.1. Conceptos Generales

Se definen conceptos que se utilizarán frecuentemente a lo largo de la tesis, en especial con lo relevante a sistemas operativos de tiempo real en sistemas empotrados.

Un **sistema empotrado** es un sistema computacional con aplicación específica con recursos limitados y que opera bajo restricciones de la aplicación que esta diseñada a realizar. Generalmente se considera a un sistema empotrado como un sistema computarizado formado por lo menos por los siguientes elementos: unidad central de procesamiento (CPU), memoria, periféricos y elementos de entrada / salida [6] y [7].

Un **sistema de tiempo real** es un sistema que tiene limitaciones de tiempo, esto significa que el tiempo en el que el sistema produce una salida es significativo,

usualmente porque la entrada del sistema corresponde a algún evento o variable del medio ambiente. Esto propicia que la efectividad del sistema de tiempo real no solo dependa de que el sistema entregue un resultado lógico correcto, sino que también el sistema debe entregar el resultado dentro del límite de tiempo permisible propio de la aplición en el que esté siendo utilizado [8, 9, 10].

Al decir que se va a trabajar en tiempo real, se contempla que un sistema debe arrojar los resultados correctos y además que éstos deben darse dentro del tiempo esperado. Por funcionalidad correcta se entienden los resultados computacionales o lógicos, mientras que temporización correcta significa que estos cálculos computacionales deben concluir dentro de un período de tiempo definido.

De lo anterior se puede concluir que un sistema en tiempo real no es aquel que necesariamente es instantáneo o muy rápido si no el que entrega resultados precisos dentro del tiempo aceptado. Si una tarea no se puede terminar dentro del plazo específico en un sistema en tiempo real por lo que puede causar resultados catastróficos y la caída del sistema, en este caso estaremos hablando de sistemas de tiempo real críticos (*hard real-time*). Por otra parte, si una tarea no llegara a cumplirse dentro del plazo específico en el 100 % de los casos y como resultado se obtiene una baja en el rendimiento, pero el porcentaje de cumplimiento es todavía aceptable para la aplicación, entonces estaremos hablando de sistemas en tiempo real no críticos (*soft real-time*) [11] y [12].

Un **Sistema empotrado de tiempo real** es algún sistema empotrado utilizado en una aplicación de tiempo real, como se puede observar en la figura 2.1. Por lo regular, la forma de operar a un sistema empotrado consiste en construir un programa

en algún lenguaje (alto nivel, bajo nivel o combinación de ambos) para desarrollar cierta actividad utilizando los elementos del sistema mencionados anteriormente.

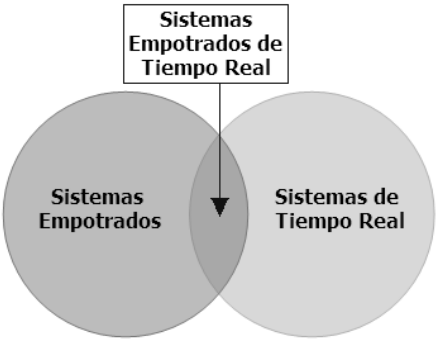

Figura 2.1: Sistema empotrado de tiempo real.

Una vez realizado el programa, se envía a la memoria del sistema para que sea ejecutado por el *CPU*. Cuando las funciones que debe realizar el sistema se vuelven más complejas, se puede emplear una interfaz entre los recursos del sistema (procesador, memoria, registros, dispositivos de entrada y salida.) y los programas que ejecuta el *CPU* para asegurar un mejor uso de los recursos [13] y [14]. Es entonces cuando se requiere el uso de un sistema operativo dentro del **sistema empotrado**, el cual se encarga de administrar los recursos del *CPU* efectuando un reparto ordenado y controlado del procesador, las memorias y los dispositivos de entrada y salida (E/S) entre los diversos programas o tareas que compiten por obtenerlos. Existen varios tipos de sistemas operativos según su aplicación, algunos ejemplos son de *mainframe*, de servidor, multiprocesador, de computadora personal, de tiempo real, integrados.

Para el trabajo que aquí se propone son de especial interés los sistemas operativos de tiempo real porque su parámetro clave es el tiempo, como en las aplicaciones de sistemas de tiempo real. Una definición concreta de un sistema operativo de tiempo

real puede encontrarse en [8] como "un programa que planifica la ejecución en tiempo, administra los recursos del sistema y provee los fundamentos para desarrollar código de aplicación". Cuando se dice que un sistema operativo es **multitarea** (multitasking) se refiere al proceso de planificar y conmutar al *CPU* entre diferentes tareas. Una de las funciones más importantes es que permite a un usuario programador administrar la complejidad inherente de un sistema de tiempo real [15].

Es posible distinguir elementos fundamentales de éstos sistemas operativos, como pueden ser los recursos, inicialización, manejo de memoria, programas, tareas, procesos, entre otros [15] y [16]. Los **recursos** son cualquier entidad de hardware o software utilizada por una o más tareas, por ejemplo, un dispositivo de entrada/salida o una variable [15]. La **inicialización** es la primera porción de código que ejecuta el sistema operativo e incluye la inicialización de variables, estructuras de datos internas y hardware [16]. Al manejar la memoria se contempla la configuración de la *pila* del sistema y de las tareas[16]. En la aplicación del medidor de *potencia óptica* de este trabajo las tareas solo comparten el procesador ya que la memoria es asignada específicamente a la tarea correspondiente.

Hay una relación en los términos de programa, tarea y proceso. Un **programa** es una secuencia de instrucciones a realizar por una computadora [17]; **una tarea** (τ) es un conjunto de instrucciones de programa que son ejecutables en una computadora, de manera informal una tarea es un programa seleccionado para su ejecución. Un programa se vuelve una tarea a partir del momento en que éste se selecciona para su ejecución hasta que finaliza la misma y se convierte de nuevo en un programa [18]; mientras que un **proceso**, en otras palabras es un programa en ejecución, es un programa que ha

iniciado pero no ha terminado, o bien es una tarea que reside en memoria. Un proceso puede ejecutarse en el momento o puede estar en espera de ser asignado al *CPU* por el planificador. Como se muestra en el diagrama de estado de la figura 2.2 un proceso es una tarea, pero una tarea no es un proceso.

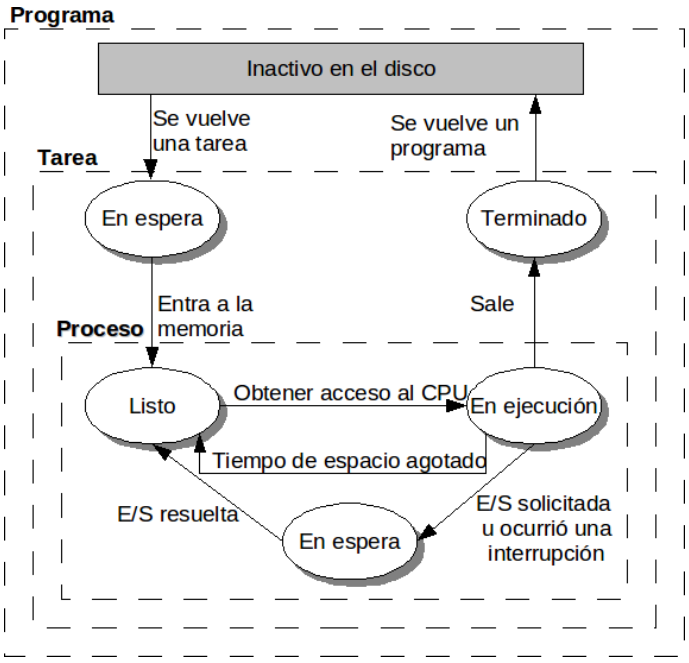

Figura 2.2: Diagrama de estado con los límites entre un programa, una tarea y un proceso.

Una tarea en tiempo real tiene la peculiaridad de ser una una tarea que tiene interacción con el mundo físico y tiene restricciones de tiempo. Estas restricciones pueden ser el **tiempo de liberación** (r), que es el tiempo en el cual una tarea τ_i está lista para ser ejecutada; el **tiempo de cálculo o ejecución** (C), que se refiere a el tiempo necesario para realizar una tarea τ_i sin interrupciones; el **plazo** (d), que es un instante de tiempo en el cual una tarea debe completarse; ó bien un conjunto de ellas.

Existen varios tipos de tareas en tiempo real, según su clasificación de limitaciones de tiempo, las mas comunes son las tareas periódicas, tareas aperiódicas y tareas esporádicas.

Las **tareas periódicas en tiempo real** son tareas que son activadas de forma regular, con un periodo (T_i) constante y tiempo de ejecución (C_i) constante y conocido; las tareas periódicas en tiempo real tienen la limitación de plazo (d_i) que por lo general es igual al periodo T_i, sin embargo es posible que el plazo d_i sea menor, igual o mayor al periodo T_i. Hay tareas periódicas síncronas y asíncronas. Las **tareas periódicas síncronas** son el conjunto de tareas periódicas que son liberadas por primera vez al mismo tiempo, mientras que las **tareas periódicas asíncronas** son el conjunto de tareas periódicas que son liberadas por primera vez en diferente tiempo.

Las **tareas aperiódicas** son tareas en tiempo real que son activadas de forma irregular en tiempo, con una tasa de activación desconocida y probablemente ilimitada.

Las **tareas esporádicas** son tareas en tiempo real que son activadas en forma irregular pero tienen un límite en la tasa de activación y un tiempo de ejecución C_i definido, el límite de la tasa de activación es el intervalo mínimo posible entre dos activaciones sucesivas (periodo mínimo). Por lo general su limitación esta definida por un plazo d_i.

2.2. Planificadores

Con el objeto de que todas las tareas en un sistema sean atendidas para tomar los recursos en diferentes momentos, es necesario contar con un planificador que lleve a cabo esta tarea, garantizando mientras sea posible que todas sean atendidas cumpliendo con sus restricciones de tiempo.

Como el tiempo es una limitación muy importante en un sistema de tiempo real, los planificadores juegan un rol muy importante dentro del sistema operativo de tiempo real, ya que esta unidad se encarga de ordenar la ejecución de un conjunto de tareas de acuerdo con un criterio predefinido [19].

Dado un conjunto de tareas concurrentes, un planificador *es un criterio predefinido que determina cuál tarea es la próxima a ejecutar en forma ordenada asignando procesos a los recursos* [20]. Un planificador es capaz de cambiar el estado de un proceso de listo al estado de ejecutándose y viceversa, dándole un lugar específico de orden entre las demás tareas.

Un esquema de planificación consta de dos características[9]:

- Un algoritmo para el ordenamiento del uso de los recursos del sistema.
- Una forma de predecir el comportamiento en el peor de los casos del sistema cuando dicho algoritmo es aplicado.

Existen diferentes tipos de planificadores:

Los planificadores estáticos, son planificadores en el cual el orden de ejecución de las tareas es asignado de acuerdo a parámetros que permanecen fijos durante la ejecución,

en este caso al momento del diseño del planificador. Mientras que en los de tipo dinámico el orden de ejecución de las tareas son asignadas de acuerdo a parámetros que pueden cambiar durante la ejecución.

También hay planificadores reentrantes o no reentrantes. En uno reentrante (*preemptive*), la ejecución de una tarea puede ser suspendida para realizar una segunda tarea con mayor prioridad y posteriormente regresar a atender la tarea previamente suspendida, en este tipo se garantiza que la tarea con mayor prioridad siempre se estará ejecutando. En un planificador no reentrante (*non preemptive*), una tarea de mayor prioridad no puede suspender a otra en ejecución y debe esperar que la tarea en ejecución se termine de atender.

Los planificadores además se pueden clasificar como planificadores en línea y fuera de línea. Los planificadores en línea toman las decisiones al momento de la ejecución de las tareas, a lo contrario de uno fuera de línea, ya que este algoritmo se ejecuta antes de la ejecución de las tareas y la planificación obtenida se guarda hasta ser necesitada.

2.3. Técnicas de Planificación

Dentro del desarrollo de los planificadores, se han generado técnicas de planificación acordes a diferentes restricciones de un sistema. Como por ejemplo el de la tasa monotónica (Rate Monotonic; RM, por sus siglas en inglés), el de del primer plazo más cercano (Early Deadline First, EDF), el de menor laxitud primero (Least Laxity First, LLF), el de de despacho paramétrico (Parametric Dispatching Algorithm), entre otros.

Liu y Layland [21] definen a los algoritmos de planificación como una serie de reglas que determinan la tarea a ser ejecutada en un momento particular.

Suposiciones sobre el ambiente

En primera instancia se utilizarán las mismas suposiciones que [21], donde se contempla que la aplicación será de tiempo real crítico, por lo tanto las siguientes suposiciones, corresponden a un conjunto de tareas de este tipo para garantizar su funcionamiento. Las cuales son las siguientes:

S1 Se asume que en la aplicación todas las tareas son periódicas y con periodo T conocido.

S2 Los plazos de las tareas son igual al periodo ($d = T$).

S3 Todos los procesos son completamente independientes uno del otro.

S4 Se ignoran las *sobrecargas del sistema* al realizar un cambio de ejecución de proceso.

S5 En cualquier momento un proceso de mayor prioridad puede suspender la ejecución de otro proceso de menor prioridad.

S6 Los procesos no se suspenden a sí mismos.

De lo anterior cabe mencionar que la primera suposición (S1), no contempla el uso de tareas esporádicas, en el caso de que se requiera utilizar tareas de este tipo es posible asignar una tarea periódica la cual tenga como objetivo verificar si alguna tarea esporádica necesita ser atendida. Para fines prácticos, se contempla el periodo mayor posible para realizar los cálculos de la tarea asignada para saber si es posible calendarizar el conjunto de tareas. En el caso de que haya procesos dependientes uno del otro (S3) sería recomendable agrupar las dos tareas considerándolas como una sola, contemplando el periodo máximo posible.

Prácticamente todos los sistemas consumen tiempo en realizar un cambio de proceso (S4). Con la documentación correspondiente es posible medir el tiempo o determinar la manera de considerar estas *sobrecargas del sistema* dependiendo de cada sistema operativo.

Un sistema operativo con preferencias es capaz de suspender un proceso de menor prioridad para atender uno de mayor prioridad (S5), sin embargo prácticamente es imposible hacer dicho cambio instantáneamente. En el caso particular del sistema operativo con el cual se realiza la aplicación, el cambio se realiza en el siguiente *tick* del sistema.

Dado que el algoritmo de la tasa monotónica contempla restricciones, este ha sido extendido por algunos autores para poder lidiar con dichas limitaciones, como por

ejemplo [22], [23], [24] y [25]. En particular, se proponen alternativas para solucionar los problemas que se presentan al manejar tareas esporádicas o no periódicas en [26] y [27].

2.3.1. Planificación con prioridades fijas (FPS)

Este tipo de planificadores son los más comunes, el planificador de prioridades fijas es un planificador fuera de línea, donde cada proceso tiene una prioridad estática y fija, la cual es calculada antes de la ejecución de las tareas, es por esto se dice que es un planificador estático.

El algoritmo de la tasa monotónica (Rate Monotonic RM) [21] es un planificador con prioridades fijas simple y óptimo en el sentido de que ningún otro algoritmo de prioridades fijas, puede planificar un conjunto de tareas que no sean planificables con el algoritmo de tasa monotónica, por lo tanto si un conjunto de tareas es planificable se garantiza que el algoritmo de tasa monotónica lo podrá planificar.

El algoritmo se fundamenta en que dado un conjunto de procesos, a cada proceso se le asigna una única prioridad P basada solamente en su periodo T, con la regla de que menor periodo, equivale a una mayor prioridad (para dos procesos i y j, si $T_i < T_j \Rightarrow P_i > P_j$). En la tabla 2.3.1 se presenta un ejemplo de un conjunto de seis procesos (a,b,c,d, e y f) y cómo son asignadas las prioridades de acuerdo a su periodo. Cabe mencionar que se utilizan números enteros para asignar las prioridades donde el número 1 indica la prioridad máxima.

Tabla 2.1: Ejemplo de asignación de prioridades de un esquema de tasa monotónica.

Proceso	**Periodo, T**	**Prioridad, P**
a	23	2
b	15	1
c	84	5
d	125	6
e	74	4
f	34	3

Factor de utilización del procesador

La fracción del tiempo que el procesador tarda en ejecutar una tarea dada T_i está dado por $\frac{C_i}{T_i}$, entonces para una cantidad n de tareas el factor de utilización U esta dado por:

$$U = \sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) \tag{2.3.1}$$

Dado esto se puede concluir lo siguiente:

El factor de utilización del procesador es la fracción de tiempo que le toma al procesador en la ejecución de un conjunto de tareas. Es igual a uno menos el tiempo que está inactivo del procesador [21].

Condición para que un conjunto de tareas sea planificable basada en el factor de utilización

Un problema con el algoritmo de tasa monotónica es que se tiene un factor de utilización de recursos menor al 100 %. En [21] Liu y Layland demostraron una manera muy simple de obtener una prueba de planificación. Cuando se utiliza un algoritmo de tasa monotónica, utilizando solo el factor de utilización de un conjunto de tareas se puede saber si es planificable. Si la condición mostrada en la ecuación 2.3.2 se cumple, se

garantiza que es factible calendarizar las tareas con el planificador de tasa monotónica (todas las tareas cumplirán sus plazos).

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) \leq N \left(2^{1/n} - 1 \right) \tag{2.3.2}$$

Donde:

N, es el número de tareas.

C_i, es tiempo de ejecución de la tarea.

T_i, es plazo de la tarea.

Tabla 2.2: Límites de utilización.

N	**Límite de utilización**
1	100.0 %
2	82.8 %
3	78.0 %
4	75.7 %
10	71.8 %
15	70.9 %

En la tabla 2.3.1 se muestran los límites permisibles con diferente número de procesos para cumplir con la condición, se puede ver que conforme el número va incrementando, el valor tiende a dejar de decrementarse asintóticamente en 69.3 % tal y como se muestra en la gráfica de la figura 2.3, donde se evalúa la relación de el límite de utilización para cumplir la condición de prueba a medida que aumenta el número de procesos.

Es sumamente importante recalcar que si un conjunto de procesos cumple la condición se garantiza que va a cumplir con todos los plazos, sin embargo en caso de no cumplirse la condición; la planificación puede o no ser factible, por lo tanto se dice que la prueba es suficiente pero no necesaria.

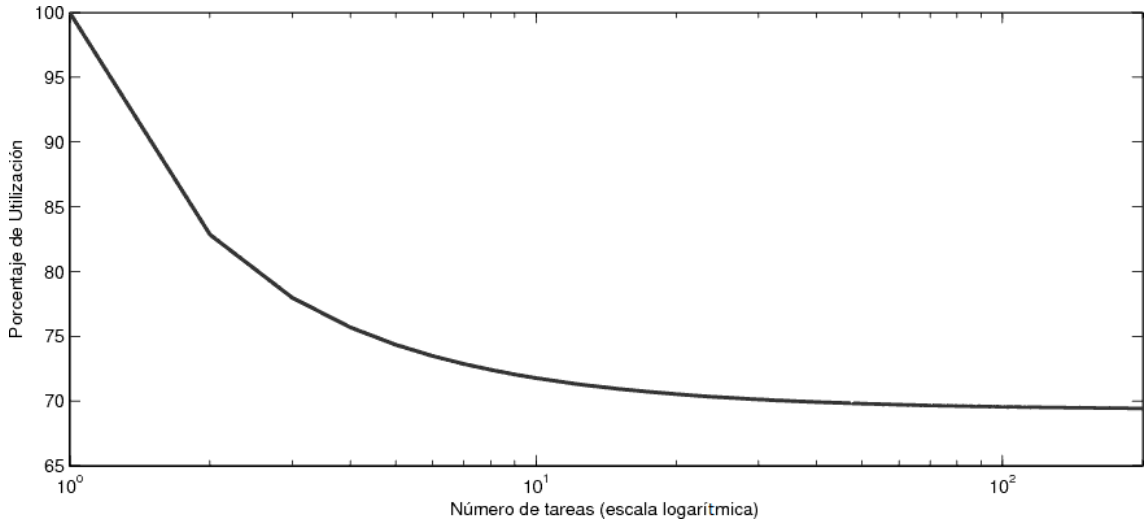

Figura 2.3: Límite de utilización para cumplir la condición para que un conjunto de tareas sea planificable basada en el factor de utilización.

El factor de utilización de una tarea(U_i) se obtiene dividiendo el tiempo de ejecución (C_i) entre el periodo (T_i) de la tarea correspondiente. A continuación se muestran ejemplos con tiempo arbitrarios, de varios conjuntos de procesos para determinar cómo es posible saber si un conjunto es planificable.

Ejemplo 1

Realizar la prueba de planificabilidad basado en el factor de utilización del conjunto de procesos A, mostrado en la tabla 2.3.1

Tabla 2.3: Conjunto de procesos A.

Proceso	**Periodo (T)**	**Tiempo de ejecución (C)**
a	60	12
b	50	15
c	40	15

Se determina el factor de utilización (U_i) de cada proceso:

$$\begin{aligned} U_a &= \frac{C_a}{T_a} = \frac{12}{60} = 0{,}2 \\ U_b &= \frac{C_b}{T_b} = \frac{15}{50} = 0{,}3 \\ U_c &= \frac{C_c}{T_c} = \frac{15}{40} = 0{,}375 \end{aligned}$$

Se asignan las prioridades de acuerdo al periodo de cada proceso, podemos ver cómo se presenta esta información en la tabla 2.3.1.

Tabla 2.4: Conjunto de procesos A.

Proceso	**Periodo (T)**	**Tiempo de ejecución (C)**	**Prioridad (P)**	**Factor de Utilización (U)**
a	60	12	3	.2
b	50	15	2	.3
c	40	15	1	.375

Utilizamos la desigualdad 2.3.2 para realizar la prueba:

$$\begin{aligned} \sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) &\leq N\left(2^{1/N}-1\right) \\ \sum_{i=1}^{3} \left(\frac{C_i}{T_i}\right) &\leq N\left(2^{1/3}-1\right) \\ 0{,}2+0{,}3+0{,}375 &\leq 0{,}78 \\ 0{,}875 &\leq 0{,}78 \end{aligned}$$

Se observa que no cumple la condición, como la utilización del conjunto de procesos es de 0,875 lo cual es mayor del límite de 0,78, por lo cual se concluye que ha fallado la prueba, y no se garantiza su planificabilidad. En la figura 2.4 se muestra el comportamiento con detalle de este conjunto de procesos A, donde al instante de tiempo 60, el proceso ***'a'*** falla su plazo ya que se está ejecutando el proceso ***'b'***, el cual es de mayor prioridad que el proceso ***'a'***.

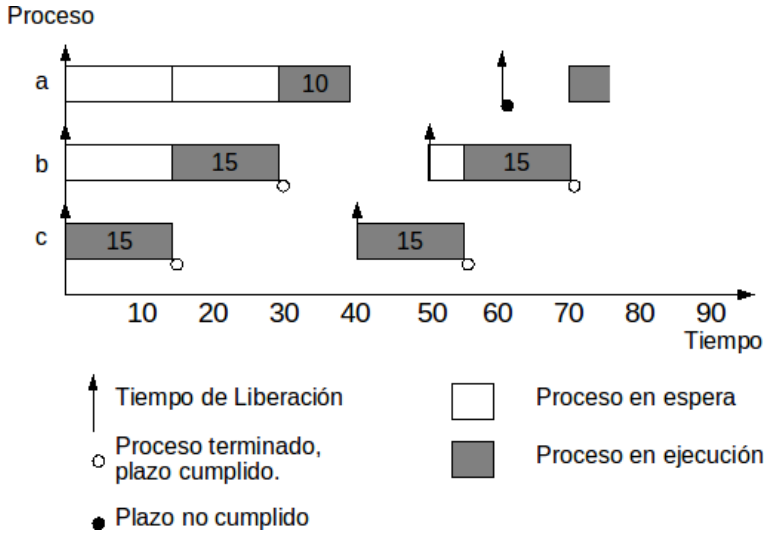

Figura 2.4: Línea de tiempo del conjunto de procesos A.

Ejemplo 2

Realizar la prueba de planificabilidad basado en el factor de utilización del conjunto de procesos B, mostrado en la tabla 2.3.1.

Tabla 2.5: Conjunto de procesos B.

Proceso	**Periodo (T)**	**Tiempo de ejecución (C)**
a	50	15
b	25	5
c	20	5

Se determina el factor de utilización (U_i) de cada proceso:

$$\begin{aligned} U_a &= \frac{C_a}{T_a} = \frac{15}{50} = 0{,}3 \\ U_b &= \frac{C_b}{T_b} = \frac{5}{25} = 0{,}2 \\ U_c &= \frac{C_c}{T_c} = \frac{5}{20} = 0{,}25 \end{aligned}$$

Se asignan las prioridades de acuerdo al periodo de cada proceso, podemos ver cómo se muestra esta información en la tabla 4.1.

Tabla 2.6: Conjunto de procesos B.

Proceso	**Periodo (T)**	**Tiempo de ejecución (C)**	**Prioridad (P)**	**Factor de Utilización (U)**
a	50	15	3	.3
b	25	5	2	.2
c	20	5	1	.25

Utilizamos la desigualdad 2.3.2 para realizar la prueba:

$$\begin{aligned} \sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) &\leq N\left(2^{1/N}-1\right) \\ \sum_{i=1}^{3} \left(\frac{C_i}{T_i}\right) &\leq N\left(2^{1/3}-1\right) \\ 0{,}3+0{,}2+0{,}25 &\leq 0{,}78 \\ 0{,}75 &\leq 0{,}78 \end{aligned}$$

La utilización es de 0,75 lo cual es menor del límite de 0,78, como si cumple la condición es posible garantizar de manera rápida y práctica que el conjunto de procesos B cumplirá todos sus plazos tal como se observa en la figura 2.5 donde se muestra el comportamiento a detalle en tiempo de este conjunto de procesos.

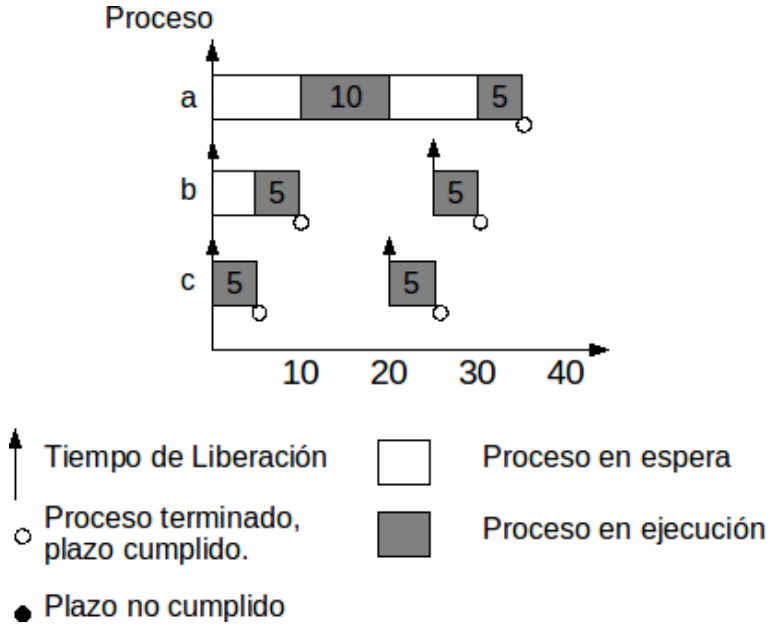

Figura 2.5: Línea de tiempo del conjunto de procesos B.

Ejemplo 3

Realizar la prueba de planificabilidad basado en el factor de utilización del conjunto de procesos C, mostrado en la tabla 2.3.1

Tabla 2.7: Conjunto de procesos C.

Proceso	**Periodo (T)**	**Tiempo de ejecución (C)**
a	95	50
b	50	10
c	25	5

Se determina el factor de utilización (U_i) de cada proceso:

$$\begin{aligned} U_a &= \frac{C_a}{T_a} = \frac{50}{95} = 0{,}53 \\ U_b &= \frac{C_b}{T_b} = \frac{10}{50} = 0{,}2 \\ U_c &= \frac{C_c}{T_c} = \frac{5}{25} = 0{,}2 \end{aligned}$$

Se asignan las prioridades de acuerdo al periodo de cada proceso, se puede ver esta información en la tabla 2.3.1.

Tabla 2.8: Conjunto de procesos C.

Proceso	**Periodo (T)**	**Tiempo de ejecución (C)**	**Prioridad (P)**	**Factor de Utilización (U)**
a	95	50	3	.53
b	50	10	2	.2
c	25	5	1	.2

Utilizamos la desigualdad 2.3.2 para realizar la prueba:

$$\begin{aligned} \sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) &\leq N\left(2^{1/N}-1\right) \\ \sum_{i=1}^{3} \left(\frac{C_i}{T_i}\right) &\leq N\left(2^{1/3}-1\right) \\ 0{,}53+0{,}2+0{,}2 &\leq 0{,}78 \\ 0{,}93 &\leq 0{,}78 \end{aligned}$$

La utilización es de 0,93 lo cual es mayor del límite de 0,78, por lo cual se concluye que ha fallado la prueba y no se garantiza su planificabilidad. Sin embargo como se muestra en la línea de tiempo para el conjunto de procesos C en la figura 2.6 este es planificable aún teniendo un porcentaje de utilización mayor al 78 %.

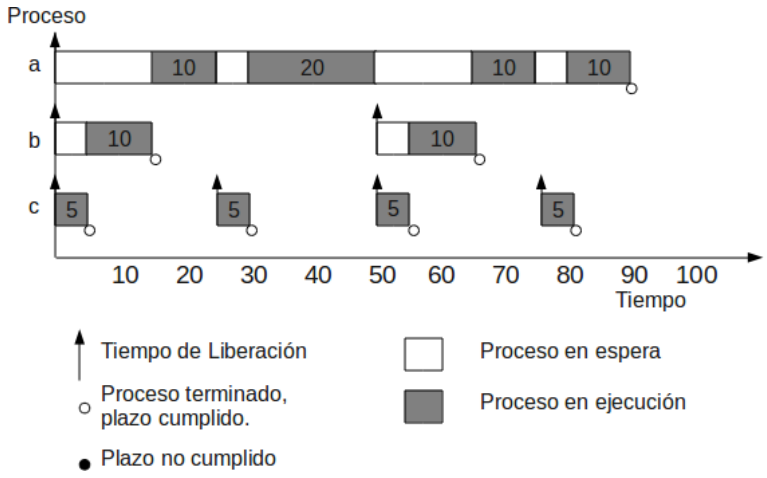

Figura 2.6: Línea de tiempo del conjunto de procesos C.

Al realizar este ejemplo se puede apreciar como anteriormente se había mencionado, que la prueba es suficiente mas no necesaria, es decir que aún después de haber hecho la prueba al conjunto de procesos C, y ésta haya fallado, es posible que la planificación del conjunto de procesos sea factible utilizando el algoritmo de la tasa monotónica.

Se puede concluir que la prueba basada en el factor de utilización tiene la ventaja de ser una prueba bastante sencilla, ya que se puede asegurar que un conjunto de procesos es planificable, solo calculando los factores de utilización y comparandolos con una desigualdad; sin embargo, tiene la desventaja de ser una prueba suficiente pero no necesaria. A continuación se presenta la prueba de análisis de respuesta en tiempo, esta prueba tiene la desventaja de ser más elaborada que la anterior porque es necesario realizar más pasos y operaciones, pero con la ventaja que es una prueba suficiente y necesaria. Por lo que es recomendable realizar la prueba de análisis de tiempo de respuesta una vez de que haya fallado la prueba del factor de utilización.

Análisis de tiempo de respuesta

El análisis de tiempo de respuesta básicamente consiste en determinar la respuesta en el peor de los casos para cada uno de los procesos y después se comparan con sus plazos.

En [28] se obtiene la ecuación para realizar el análisis de tiempo de respuesta. Analizando cada uno de los procesos por separado, para el proceso de mas alta prioridad, el peor caso (R) es igual a su tiempo de ejecución ($R = C$), debido a que éste no puede ser interrumpido por algún otro proceso. Por otro lado para todos los demás procesos, pueden ser interrumpidos si un proceso de mayor prioridad se libera al instante que se está ejecutando el proceso. Debido a esto, el peor caso de respuesta en tiempo es igual al tiempo de ejecución mas la máxima interrupción posible, es decir, para el caso en particular que todos los procesos de mayor prioridad sean liberados en el mismo instante de tiempo (instante crítico), a lo cual tenemos la siguiente ecuación.

$$R_i = C_i + I_i \tag{2.3.3}$$

Donde I es la máxima interrupción que el proceso i puede sufrir.

Considerando un proceso j con mayor prioridad que i la máxima interrupción está dada por:

$$I_i = \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.3.4}$$

Donde $\left\lceil \frac{R_i}{T_j} \right\rceil$ es la cantidad de liberaciones dentro del intervalo $[0, R_i)$, la función $\lceil \rceil$ nos indica el menor número entero mayor que el número fraccionario dentro de la función, por ejemplo: $\lceil \frac{3}{4} \rceil = 1$, $\lceil \frac{6}{4} \rceil = 2$.

Como cada proceso de mayor prioridad interfiere con el proceso i:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.3.5}$$

Donde $hp(i)$ es el conjunto de procesos con mayor prioridad que i. Sustituyendo (2.3.5) en (2.3.3) se obtiene la siguiente ecuación.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.3.6}$$

Ejemplo 4

Realizar el análisis de tiempo de respuesta para el conjunto de procesos del ejemplo 3, y determinar si es factible su calendarización.

Para la realización de este ejemplo es necesario determinar la respuesta en el peor de los casos de todos los procesos. Se puede garantizar su calendarización si el tiempo del peor caso de cada proceso (R_i) es menor o igual a su plazo correspondiente (d_i), recordando que el plazo es igual al periodo.

- Empezando con el proceso de mayor prioridad, proceso ***'c'***:

  $$R_i = C_i + I_i$$

  Como el proceso ***'c'*** es el de mayor prioridad, entonces I_c es igual a cero ya que no puede ser interrumpido por otro proceso.

  $$\begin{aligned} R_c &= C_c + I_c \\ R_c &= C_c = 5 \end{aligned}$$

  Comparando el plazo del proceso ***'c'*** ($d_c = 25$) con el peor caso calculado, se

puede determinar que cumple con la condición.

$$\begin{aligned} R_c &\leq d_c \\ 5 &\leq 25 \end{aligned}$$

- Posteriormente se realiza el análisis para el proceso ***'b'***.

$$R_b = C_b + I_b$$

Utilizando la ecuación (2.3.5), se determina la máxima interrupción (I_b) que el proceso ***'b'*** puede sufrir:

$$\begin{aligned} I_i &= \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \\ I_b &= \left\lceil \frac{R_b}{T_c} \right\rceil C_c \\ I_b &= \left\lceil \frac{15}{25} \right\rceil 5 = 5 \end{aligned}$$

Sustituyendo I_b:

$$R_b = C_b + I_b = 10 + 5 = 15$$

Comparando el plazo del proceso ***'b'*** ($d_c = 50$) con el peor caso obtenido, se puede determinar que cumple con la condición:

$$\begin{aligned} R_b &\leq d_b \\ 15 &\leq 50 \end{aligned}$$

- Por último, se realiza el mismo análisis para el proceso ***'c'***, que es el de menor prioridad.

$$R_c = C_c + I_c$$

Se determina la máxima interrupción (I_a) que el proceso ***'b'*** puede sufrir:

$$
\begin{aligned}
I_i &= \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \\
I_a &= \left\lceil \frac{R_a}{T_b} \right\rceil C_b + \left\lceil \frac{R_a}{T_c} \right\rceil C_c \\
I_a &= \left\lceil \frac{90}{50} \right\rceil 10 + \left\lceil \frac{90}{25} \right\rceil 5 = 20 + 20 = 40
\end{aligned}
$$

Sustituyendo I_a:

$$R_a = C_a + I_a = 53 + 40 = 93$$

Comparando el plazo del proceso ***'b'*** ($d_c = 95$) con el peor caso calculado, se puede determinar que cumple con la condición.

$$
\begin{aligned}
R_a &\leq d_a \\
93 &\leq 95
\end{aligned}
$$

Con este análisis se determina que el conjunto de procesos es planificable. En este ejemplo, se puede apreciar cómo a un conjunto de procesos que no pasó la prueba del factor de utilización en el ejemplo 3, se le puede realizar el análisis de tiempo de respuesta y así determinar si es factible planificarlo.

2.3.2. Otros algoritmo de planificación

El algoritmo EDF es un algoritmo de planificación dinámico porque las prioridades se asignan en el momento de ejecución de las tareas. En este algoritmo el conjunto de procesos tiene que ser ejecutado para determinar el siguiente proceso que será atendido dependiendo del plazo de las tareas, dado que el proceso que tenga el valor absoluto menor de plazo será el que tenga mayor prioridad. Los procesos con plazos cortos o cercanos tendrán prioridades altas y procesos con plazos grandes tendrán prioridades bajas.

Otro algoritmo de calendarización dinámico aparte del EDF es el LLF, en el cual la mayor prioridad se asigna a la tarea que tenga la menor laxitud; la laxitud es la diferencia de tiempo que hay de plazo para una tarea y el tiempo que se requiere para realizar dicha tarea [29, 10]. Además existen otros algoritmos de calendarización con diferentes característcas, como el algoritmo de actividad dependiente (Dependent Activity Scheduling Algorithm; DASA, por sus siglas en inglés)[30] y el algoritmo de mejor esfuerzo de Locke (Locke's Best Effort Scheduling Algorithm, LBESA) [31] buscan proveer el mejor aprovechamiento de las aplicaciones en condiciones de sobrecarga del sistema. Existen algoritmos, los cuales están hechos para trabajar con múltiples procesadores; como por ejemplo, el algoritmo de la calendarización de Pfari (Pfari Scheduling, en inglés) [32] y el algoritmo de direccionamiento enfocado y subasta (Focused addressing and bidding algorithm, en inglés) [10].

El calendarizador del RTOS comercial μC/OS-II que se utiliza en la implementación de la aplicación es un calendarizador de prioridades fijas. El calendarizador está definido por defecto por parte del diseñador del RTOS, por lo que no permite la manipulación por

parte del usuario. Con lo visto a lo largo del capítulo será posible tener el conocimiento del funcionamiento de los calendarizadores, lo cual será de utilidad al momento de decidir la manera de asignar las prioridades a las tareas, así como realizar las pruebas de planificabilidad.

Capítulo 3

Marco Experimental

3.1. Descripción del Experimento

Para el desarrollo del experimento se realiza la implementación de una aplicación en tiempo real, en general la implementación consiste en dividir la aplicación en módulos independientes, de manera que cada uno de ellos se encargue de realizar una función específica; posteriormente, se realizan los algoritmos y el código de programación en lenguaje C, para cada una de las tareas, las cuales serán administradas por el RTOS; después, se procede a trasladar los programas al RTOS creando las tareas, mediante la utilización las funciones propias del sistema operativo y asignando la prioridad correspondiente a cada tarea. Cabe mencionar que para asignar las prioridades y realizar la prueba de planificabilidad al conjunto de tareas, es necesario saber el tiempo de ejecución de ellas, esto conlleva a la necesidad de realizar la programación de las tareas para conocer el tiempo de utilización que requiere cada tarea.

Para llevar a cabo la implementación se utilizó un sistema operativo de tiempo real en un sistema empotrado que es capaz de realizar cualquier aplicación en general. Dicha aplicación está conformada por realización de varias tareas independientes, de manera que el sistema operativo es el que se encarga de administrar los recursos del sistema

empotrado de manera eficiente, de acuerdo con parámetros que el programador le asigna a las tareas, como son prioridades, plazos y cantidad de memoria de la *pila*.

3.2. Descripción de la aplicación utilizada

Con la intención de mostrar una aplicación de manera tangible, se eligió un sistema de medición de *potencia óptica* con fines académicos, dado que el tener acceso a un medidor de *potencia óptica* es de gran ayuda a los estudiantes de electrónica en el área de comunicaciones ópticas para el desarrollo de prácticas en el estudio de la óptica y la electrónica[33].

El funcionamiento general a bloques de la aplicación, como se muestra en la figura (3.1), consiste en la utilización de un convertidor de luz a frecuencia, de un microprocesador ARM y una pantalla.

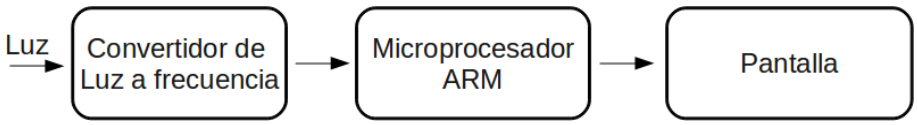

Figura 3.1: Diagrama general a bloques de la aplicación.

En la primera etapa la intensidad de luz es registrada por el sensor TSL230R-LF y proporciona una señal cuadrada con frecuencia proporcional a la intensidad de luz recibida, dicha señal es la entrada a la segunda etapa, donde una tarea del microprocesador *ARM*, calcula la frecuencia entregada por el sensor, mientras que otra tarea determina la *potencia óptica* correspondiente a la caracterización [33]. El microprocesador *ARM* se conecta a una pantalla *LCD* para desplegar los resultados obtenidos mediante una tarea independiente.

3.3. Implementación de la aplicación

Esta aplicación se puede considerar adecuada para el experimento al realizar la implementación, ya que su funcionamiento se puede programar en un sistema operativo de tiempo real utilizando tareas independientes, que como se indicó en el capítulo 2 es una cualidad necesaria. El experimento realizado en este trabajo está acotado a la utilización del RTOS, por lo que cualquier análisis sobre el estudio y caracterización del sistema de medición queda fuera del alcance de este trabajo.

Una vez identificadas las funciones generales de la aplicación, se procede a separar el funcionamiento por medio de módulos independientes para que puedan ser implementados en tiempo real. En la figura (3.2) se muestra la manera en el que los módulos quedan separados por funciones independientes, las cuales serán administradas por el RTOS.

En la figura 3.2 se muestra un diagrama de la forma en que el RTOS interactúa con las tareas de la aplicación. La aplicación esta separada por cuatro módulos, un módulo encargado de atender al sensor, uno que se encarga de el control de los botones por los cuales el usuario interactúa con el sistema, un tercer módulo se encarga del funcionamiento de la pantalla y la última tarea corresponde a una tarea adicional (la cual se asigna para que represente alguna otra función que se le quiera agregar a la aplicación, en el caso en que posteriormente pueda ser necesario que sistema crezca). Las tareas se relacionan entre sí por medio del RTOS, que es el encargado de administrar el conjunto de tareas.

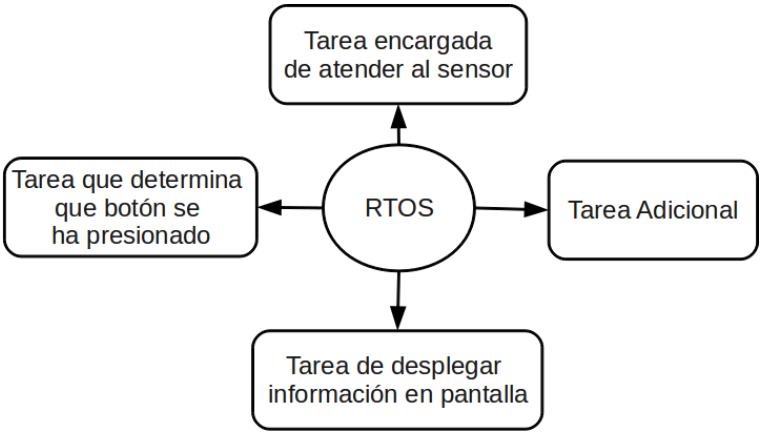

Figura 3.2: Diagrama general de las tareas.

En la tarea encargada de atender al sensor, se lee el dato del registro del módulo del temporizador, el cual tiene el valor del periodo de la señal cuadrada a la salida del sensor. Esto permite calcular la frecuencia de la señal, para posteriormente poder obtener el valor de la potencia óptica que se está midiendo, los datos leídos de frecuencia y potencia son guardados en variables globales, éstas variables son leídas posteriormente en la tarea del *LCD*, cuando la información es enviada a la pantalla.

La tarea asignada para determinar el botón presionado funciona de una manera simple, la tarea es atendida cuando se genera una interrupción externa al presionar cualquier botón funcional, y posteriormente se manda por mensaje una variable con un valor correspondiente al botón que se haya presionado. Los mensajes pueden ser enviados a una tarea a través de los servicios del núcleo del sistema operativo; un buzón de mensajes, es una variable de tipo apuntador, a través de un servicio proporcionado por el núcleo, una tarea o una rutina de interrupción puede depositar un mensaje (el puntero) en este buzón. Del mismo modo, una o más tareas puede recibir mensajes a través de un servicio proporcionado por el núcleo.

Para llevar a cabo el funcionamiento de la tarea de despliegue de información en la pantalla, primero se revisa el dato obtenido de la rutina de interrupción para la atención de los botones, con el fin de determinar que tipo de mensaje para la pantalla es el que se desplegará, posteriormente se limpia la pantalla y se despliega el mensaje para la pantalla correspondiente.

En la tarea adicional contemplada para cuando se desee agregar alguna otra función a la aplicación, el tiempo contemplado para atender a esta tarea es representado por el encendido y apagado intermitente de un *LED*.

Para fines prácticos, en el experimento se utiliza un generador de funciones, para tener una señal cuadrada con un ciclo de trabajo del 50 %. Ésta representa la salida *TTL* del sensor de luz, lo cual nos permite variar la frecuencia con mucha mayor facilidad para representar las variaciones de intensidad de luz. Para que el sensor se comporte de la manera esperada se realizó su caracterización en [33]. En el kit de desarrollo se utilizan dos botones para cambiar la pantalla del *LCD*; el botón 1, para avanzar de pantalla y el botón 2 para retroceder. Además se utiliza el *LED* 1 en la misma tarjeta en la tarea inicial para indicar la ejecución de esta tarea. Adicionalmente se emplea un analizador lógico, para observar el comportamiento de las tareas en el sistema, al iniciar cada tarea en la primera instrucción activa un *pin* del microprocesador y al finalizar cada tarea el mismo *pin* se desactiva.

Para continuar con la implementación de una aplicación en tiempo real, después de haber separado el funcionamiento de la aplicación en módulos, se procede a programar las funciones y tareas correspondientes a cada módulo en lenguaje C, para explicar cómo

se realizó esto, a continuación se muestra la descripción de las tareas y funciones del proyecto. Una vez que se haya hecho la programación en C de las tareas, será posible determinar el tiempo que ocupa el procesador en atender a cada una de ellas, para así determinar el factor de utilización del conjunto de tareas y realizar las pruebas pertinentes.

3.4. Descripción de las funciones y tareas del proyecto

A continuación se muestran y se describen las funciones y tareas utilizadas para aplicar el RTOS en las tareas de medición de *potencia óptica*.

3.4.1. Función `main()`

Como la gran mayoría de los programas en "C" el código de aplicación del RTOS inicia en la función `main()`, cuyo diagrama de flujo se muestra en la figura (3.3). A continuación se describe a detalle el funcionamiento y el código contenido en esta función:

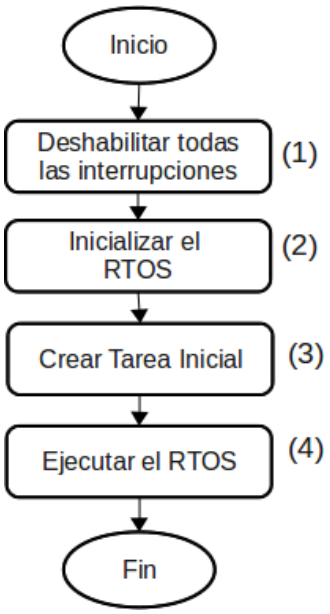

Figura 3.3: Diagrama de flujo de la función `main()`.

(1) Es necesario deshabilitar todas las interrupciones mientras se realiza el proceso de la inicialización de la aplicación, para asegurar que no ocurra alguna interrupción en el sistema antes de que esté completamente inicializado. Durante el proceso de

inicialización se configura el RTOS, los puertos del kit de desarrollo a utilizar y el controlador del *LCD*, éstos módulos son necesarios para realizar la aplicación de modo que si ocurriera una interrupción cuando se esta inicializando el sistema, es muy posible que algún puerto no quede correctamente inicializado y no pueda ser utilizado lo cual ocasionaría que el sistema no funcione o funcione incorrectamente.

(2) Es necesario inicializar el RTOS llamando a la función `OS_Init()` antes de utilizar cualquier otra función o servicio del RTOS. Al realizar esto dos cosas suceden:

- Se inicializan todas la variables y estructuras de datos propias del RTOS.
- Se crea una tarea propia del RTOS llamada `OS_TaskIdle()`, la cual tiene asignada por defecto la menor prioridad posible. La tarea `OS_TaskIdle()` es una tarea que no realiza algún propósito para la aplicación; sin embargo, es necesaria para el funcionamiento del RTOS.

Ya que el calendarizador del RTOS es el encargado de asignar los recursos del sistema a las tareas cuando sea conveniente. Para llevar a cabo esto, es necesario que las tareas estén programadas en un ciclo infinito a modo que el calendarizador determine en que momento poner a dormir o ejecutar una tarea cuando este lista para ser ejecutada. Esto conlleva a que en ocasiones se presente el caso particular en que todas las tareas de aplicación no estén listas para ser ejecutadas. Cuando se presenta este caso el RTOS ejecuta la tarea `OS_TaskIdle()` en la cual el RTOS puede poner el procesador en un estado de bajo consumo de potencia.

(3) Se crea la tarea inicial (posteriormente dentro de esta tarea se crean las otras tareas) con el objetivo que el sistema operativo administre las tareas posteriormente.

Para crear la tarea es necesario pasarle cuatro argumentos a esta función: el primer argumento es un apuntador al código de la tarea, el segundo argumento es un apuntador al argumento que es pasado a la tarea cuando esta inicia su ejecución, la cual en este caso no será utilizado, el tercero es un apuntador al tope de la *pila* que es asignada a la tarea y el último, es la prioridad que se le desea asignar a la tarea.

(4) Se da el control del programa al RTOS para que comience a administrar las tareas.

3.4.2. Tarea inicial

Esta es la tarea inicial, la cual está hecha para iniciar los módulos necesarios para la aplicación, crear las otras tareas y por último quedarse en un ciclo infinito realizando su actividad correspondiente.

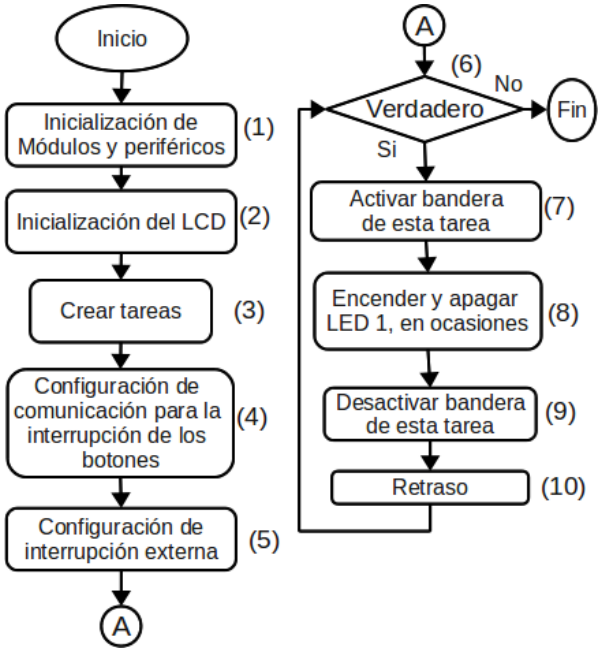

Figura 3.4: Diagrama de flujo de la tarea inicial.

Al iniciar el programa se procede con la inicialización (1) de los módulos y periféricos de la tarjeta de desarrollo que serán utilizados, Así como el código de inicialización del *LCD* (2) que especifica el fabricante.

En la etapa de crear tareas (3), se crean las otras exactamente de la misma forma que se creó la tarea inicial, las tareas creadas son la tarea del Sensor y la del *LCD*.

(4) Se utiliza una función propia del RTOS para configurar un canal de comunicación entre el manejador de la interrupción externa y la tarea del *LCD*.

(5) Se configura la interrupción externa (mencionada anteriormente) encargada de atenderse cuando haya una solicitud por parte de usuario (cuando se haya presionado algún botón).

Aquí se inicia el ciclo infinito (6) ya que cualquier tarea controlada por el RTOS debe estar en un ciclo infinito (por la razón que se explicó en el punto 2 del diagrama anterior).

(7) Un *pin* de la tarjeta de desarrollo se pone en alto para indicar en el analizador lógico cuando el μC/OS-II inicia a atender esta tarea, y posteriormente (9) se pone en nivel bajo para indicar que se ha terminado la tarea.

dentro del ciclo infinito se enciende y apaga el *LED* 1 de la tarjeta de desarrollo (8), esta tarea no es propia de la función del objetivo del medidor de *potencia óptica*, sin embargo aquí posteriormente se puede sustituir código aplicable.

(10) Se le indica a la tarea el retraso en *ticks* que se desea para la tarea. En el primer paso para realizar mediciones, a todas las tareas se le asignan deliberadamente un retraso de un *tick*, para observar el comportamiento en el analizador lógico cuando todas las tareas son liberadas en el mismo instante de tiempo. Posteriormente se le asignará un retraso mayor, para un mejor funcionamiento de la aplicación.

Al asignarle a la tarea un *tick* de retraso, significa que la tarea estará suspendida hasta el siguiente *tick*. Indicarle un 0 de argumento indica que no hay ningún retraso y

la tarea siempre estará lista para ser ejecutada, por lo que si la aplicación requiere que por lo menos la tarea se retrase un *tick*, es necesario darle 2 *ticks* de retraso. Como se muestra en la figura (3.5).

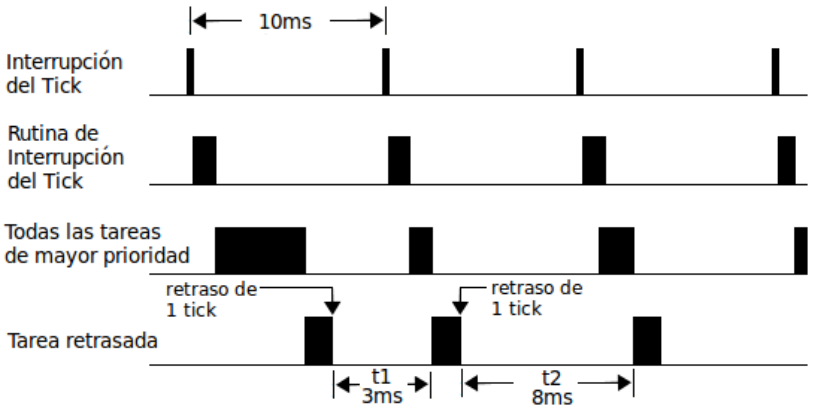

Figura 3.5: Diagrama de retrasos.

3.4.3. Tarea del sensor

En esta tarea como su nombre lo indica, se realiza la lectura del dato mas reciente leído por un módulo del temporizador de la tarjeta de desarrollo, el cual mide la frecuencia de entrada de la señal cuadrada. Para realizar la conversión de frecuencia a potencia en decibeles se utiliza la ecuación (3.5.1):

$$P(dBm) = \frac{f - 453396{,}4286}{21035{,}7143} \tag{3.4.1}$$

La ecuación (3.5.1) es utilizada para obtener el valor de la *potencia óptica* en decibeles y es obtenida mediante una interpolación lineal de la gráfica obtenida por [33], en la cual se realizó una caracterización con el sensor de luz. En la figura (3.6) se muestra el diagrama de flujo de la tarea del sensor.

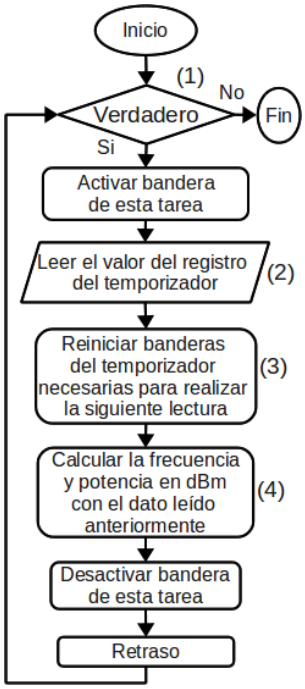

Figura 3.6: Diagrama de flujo de la tarea del sensor.

(1) Igual que las tareas anteriores, el código de la tarea se encuentra en un ciclo infinito.

(2) Se realiza la lectura del registro correspondiente al temporizador asociado a la medición del sensor.

(3) Después de hacer una lectura es necesario reiniciar a los bits correspondientes del temporizador, para que proceda a realizar la siguiente lectura.

(4) Con el valor del registro, se determina la frecuencia de la señal de entrada. y utilizando la ecuación (C.1.1) se determina la potencia en dBm.

3.4.4. Tarea del LCD

En la tarea del *LCD*, se determina que tipo de mensaje es el que se va mostrar en la pantalla, según el botón que haya sido presionado por el usuario, y posteriormente se limpia la pantalla para mandar el mensaje correspondiente, en la figura (3.7) se muestra el diagrama de flujo de la tara del *LCD*.

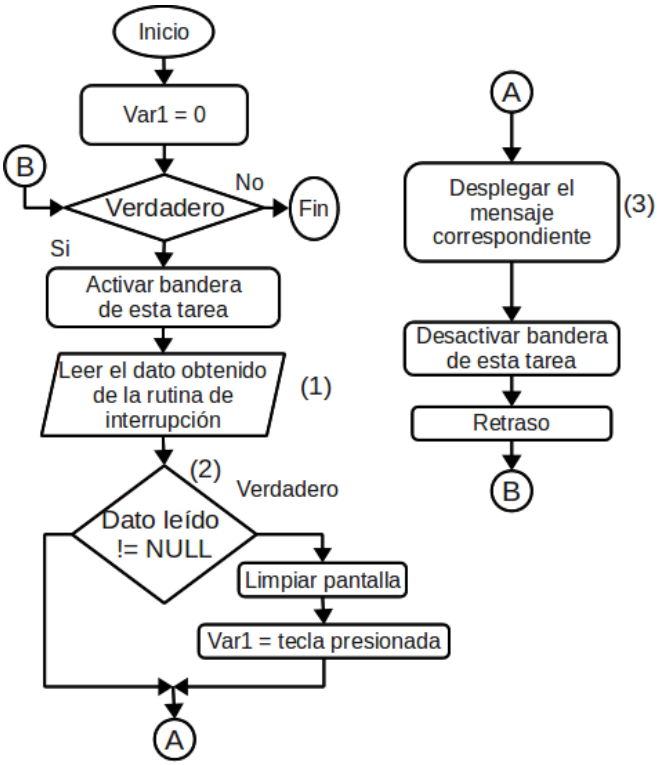

Figura 3.7: Diagrama de flujo de la tarea del LCD.

(1) Se obtiene el dato que se haya leído en la rutina de interrupción, cuando el usuario presione un botón.

(2) Si el dato leído no es un dato nulo (cuando el operador no haya presionado algún botón) no cambia el valor de la variable "var", por lo tanto se mantiene la opción de la pantalla.

(3) Dependiendo del botón que se haya presionado la variable "var" indicará que pantalla es necesario desplegar.

3.4.5. Función de la rutina de interrupción

Es la función que genera una interrupción externa en el momento que el usuario presiona un botón (figura 3.8). Se le asigna un valor a una variable con nombre *'dato'*, de *'1'* ó *'0'* , según el botón que haya sido presionando, para posteriormente mandar un mensaje con la variable *'dato'*. Si se han presionado los dos botones, se manda el valor correspondiente del botón 1.

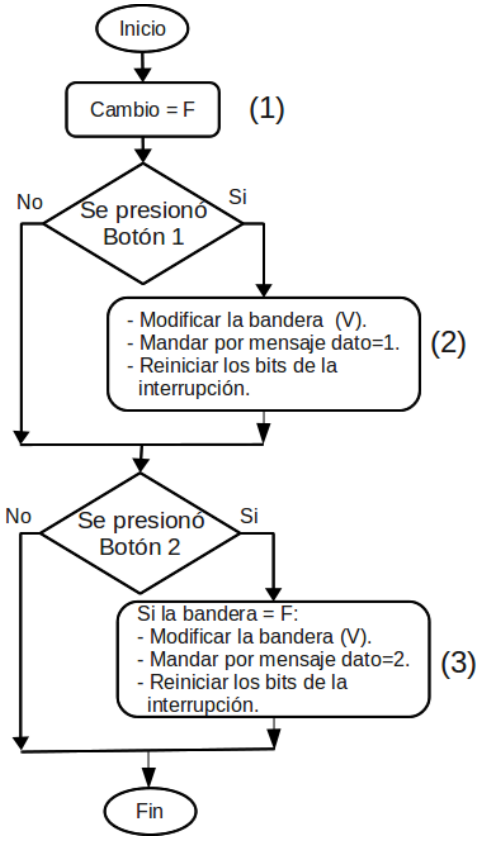

Figura 3.8: Diagrama de flujo de la rutina de interrupción.

(1) Se utiliza una bandera con el nombre *"cambio"* para validar el sistema en el caso que los dos botones sean presionados al mismo tiempo.

(2) Si se ha presionado el botón 1, se lleva a cabo el proceso de activar la bandera a *'V'* (verdadero). Posteriormente se manda por mensaje la variable, que sirve de control para seleccionar cual mensaje será mostrada en el *LCD*. Y al terminar se reinicia la bandera de la interrupción.

(3) Esta sección utiliza la misma lógica del inciso anterior, solo que en el caso de cuando se selecciona el botón 2.

3.5. Descripción breve del sensor TSL230R-LF

El convertidor combina un foto-diodo de silicio configurable y un convertidor de corriente a frecuencia en un circuito integrado (para mayor información del sensor ver al anexo B.4, en la página 77) . La salida de este dispositivo puede ser tanto un tren de pulsos o una señal cuadrada con un ciclo de trabajo del 50 %, donde la frecuencia de dicha señal es directamente proporcional a la intensidad de la luz recibida por el convertidor. Una gran ventaja de este dispositivo es que tanto la salida como la entrada son señales compatible con *TTL*, esto permite realizar una conexión directa con el controlador. El sensor es sensible a la luz en el rango de 320 nm a 1050nm.

3.6. Descripción breve del RTOS μC/OS-II

En esta aplicación se utiliza el μC/OS-II, que significa "Sistema Operativo de Microcontroladores Versión 2". El μC/OS-II es un *kernel* portable, que puede ser implementado en la memoria *ROM* de un microcontrolador, escalable, reentrante y multitarea de tiempo real para microcontroladores, microprocesadores y procesadores digitales de señales.

El código del μC/OS-II ha sido escrito principalmente en lenguaje ANSI-C (Estándar publicado por el Instituto Nacional Estadounidense de Estándares, para el lenguaje de programación C), con alguna pequeña parte de código en ensamblador. Con el fin de adaptarlo a las diferentes arquitecturas de los procesadores, el μC/OS-II se puede implementar en *CPUs* desde 8 hasta 64 bits.

Este RTOS es ampliamente utilizado en aplicaciones médicas, industriales, aeronáuticas, electrodomésticos, entre otras. Para mayor detalle sobre el μC/OS-II, consultar el anexo B, sección 3.

3.7. Descripción breve del sistema de desarrollo utilizado

El experimento se realizó en una tarjeta de desarrollo IAR STR750-SK, programada con emulador de JTAG llamado J-LINK. Éste se conecta por puerto USB a una computadora, con sistema operativo Windows XP. Se utiliza el ambiente de desarrollo IAR Embedded Workbench IDE.

La tarjeta IAR STR750-SK emplea un microcontrolador ARM7TDMI. Este dispositivo proporciona interfaces para comunicación como son: puerto UART, puerto SSP, puerto USB, puerto CAN, y puerto I^2C. Dentro de los periféricos que contiene la tarjeta, están 6 temporizadores, un convertidor A/D de 16 canales, el procesador tiene dos bancos de memoria *FLASH*: uno de 256kB de memoria de programa y uno de 16kB de memoria de datos. Para mayor detalle sobre él, consultar el anexo B, sección 2.

Capítulo 4

Resultados

4.1. Análisis de la ejecución de las tareas

En la figura (4.1) se presenta el diagrama de tiempos de las tareas cuando son liberadas al mismo tiempo.

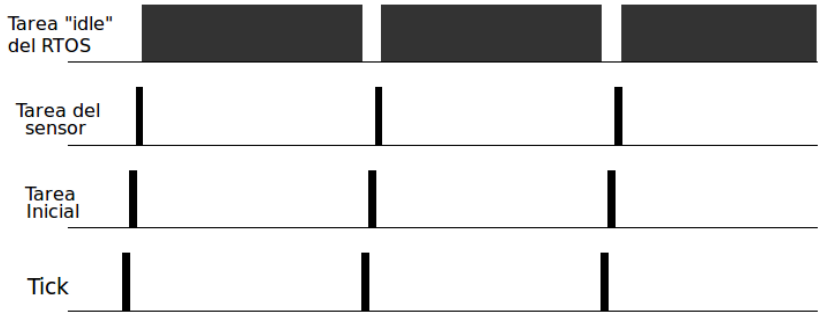

Figura 4.1: Diagrama general de tiempos.

Un parámetro importante a evaluar es determinar cuánto tiempo el procesador le dedica sus recursos para realizar una tarea, utilizando el analizador lógico es posible determinar este tiempo. Como se muestra en la figura (4.2), se representan los tiempos de las tareas; la tarea del sensor se ejecuta en 37.07μs, la llamada tarea inicial en 18.73μs y la rutina del *tick* en 16.38μs.

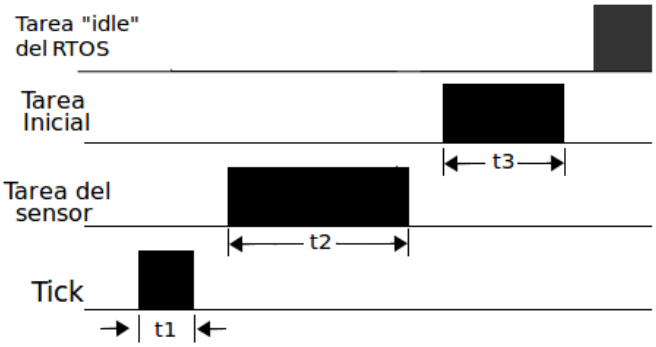

Figura 4.2: Tiempo de ejecución de las tareas.

Tiempo de ejecución de la tarea del Sensor

Cuando se está ejecutando una tarea de baja prioridad, y una de mayor prioridad requiere ser atendida, el sistema operativo le da atención a la tarea de mayor prioridad interrumpiendo la que se encuentre ejecutándose, idealmente ese cambio sería instantáneo, sin embargo realizar esto tiene un costo de tiempo (del orden de aproximadamente 10 μs para este *CPU*) y de memoria del procesador, ya que el procesador necesita guardar el contexto actual (variables y registros) de la tarea de baja prioridad, determinar cuál es la siguiente tarea a ser atendida y restaurar el estado de la tarea de mayor prioridad si ésta había sido interrumpida anteriormente. En la figura (4.3) se muestra el tiempo que toma pasar de una tarea a otra.

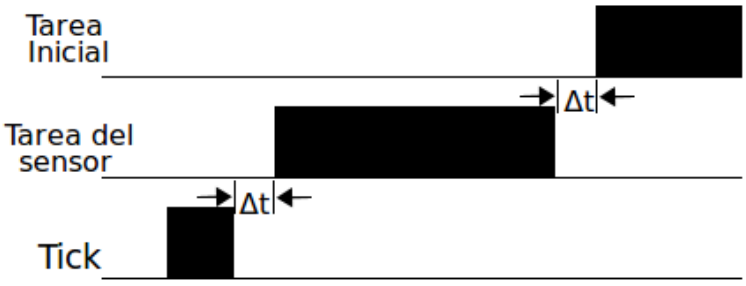

Figura 4.3: Diferencial de tiempo de 9,4μs.

En la figura 4.3 se observa un tiempo (Δt) de 9,4μs que se tarda para iniciar la tarea del sensor después de que finaliza la ejecución de la tarea del *tick*.

Con el objetivo de mostrar el funcionamiento de RTOS μC/OS-II, cuando se tienen tareas que demandan mayor tiempo de utilización del procesador, se realiza el siguiente análisis donde se agrega otra tarea, y se mantiene ocupado el procesador por mucho mas tiempo.

Para lograr lo anterior se toma como base el problema propuesto en el ejemplo 2 del capítulo anterior, donde se tienen las siguientes tareas:

Tabla 4.1: Conjunto de Tareas.

Tarea	**Periodo (T)**	**Tiempo de ejecución (C)**	**Prioridad(P)**	**Factor de Utilización (U)**
a	50	15	3	.3
b	25	5	2	.2
c	20	5	1	.25

Tal y como establece la técnica de calendarización de la tasa monotónica se asignan prioridades a las tareas de acuerdo a su periodo, con la regla de menor periodo es igual a mayor prioridad.

Anteriormente se había determinado para este conjunto de tareas que:

$$\sum_{i=1}^{N}\left(\frac{C_i}{T_i}\right) \leq N\left(2^{1/N}-1\right) \tag{4.1.1}$$

$$\sum_{i=1}^{3}\left(\frac{C_i}{T_i}\right) \leq N\left(2^{1/3}-1\right) \tag{4.1.2}$$

$$0{,}3+0{,}2+0{,}25 \leq 0{,}78 \tag{4.1.3}$$

$$0{,}75 \leq 0{,}78 \tag{4.1.4}$$

La utilización es de 0,75 lo cual es menor del límite de 0,78, como si cumple la condición es posible garantizar de manera rápida y práctica que el conjunto de procesos B cumplirá todos sus plazos.

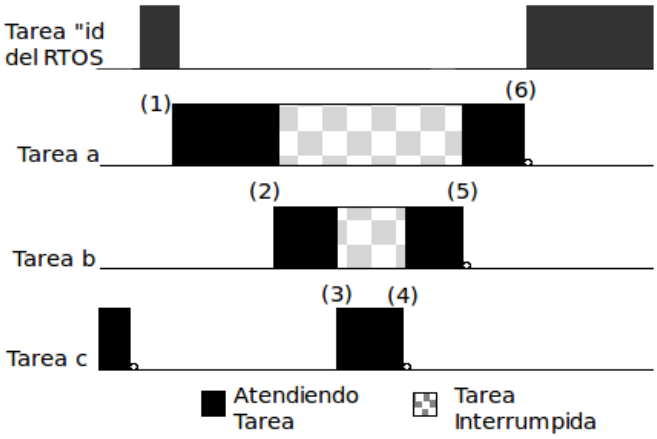

Figura 4.4: Comportamiento del conjunto de tareas.

En la figura (4.4) se muestra el comportamiento obtenido en el analizador lógico del conjunto de tareas del ejemplo, se puede analizar los siguiente:

Tiempo (1) - La tarea “a” empieza a ser ejecutada siendo la única tarea lista en ese momento.

Tiempo (2) - La tarea "b" deja de estar dormida y como tiene mayor prioridad que la tarea "a", el μC/OS-II la atiende y deja en espera a la tarea "a". El *pin* de la tarea "a" usado como bandera aún esta en alto sin embargo no está siendo atendida porque no ha llegado a la instrucción que pone en nivel bajo el *pin*).

Tiempo (3) - En este momento la tarea "c" con mayor prioridad esta lista para ser ejecutada, se interrumpe la tarea que se estaba ejecutando para atenderla.

Tiempo (4) - Se puede ver que la bandera de la tarea "c" baja, esto indica que se terminó de ejecutar la tarea con mayor prioridad; en este punto el calendarizador del μC/OS-II evalua cual de las dos tareas que están en espera tiene la mayor prioridad y es por eso que regresa a atender a la tarea "b".

Tiempo (5) - Se termina de atender la tarea "b", y el calendarizador regresa a atender la tarea que estaba pendiente.

Tiempo (6) - Se finaliza la tarea "a", y en este momento el procesador se desocupa, y regresa atender la tarea que μC/OS-II tiene por defecto.

4.2. Medición de la Potencia Óptica

En la tabla (4.2) se muestra la comparación entre la frecuencia de entrada y la frecuencia mostrada en la pantalla del sistema. Debido a las propiedades del temporizador, se puede observar claramente que a medida de que se aumenta la frecuencia que se desea medir, la precisión disminuye.

Tabla 4.2: Tabla comparativa entre la señal de entrada y la señal medida por el sistema.

Frecuencia de entrada	**Frecuencia medida**	**Error relativo Porcentual**
0.99 kHz	0.99 kHz	0 %
10.06 kHz	10.06 kHz	0 %
100.10 kHz	100.3 kHz	0.2 %
199.90 kHz	201.30 kHz	0.7 %
500.50 kHz	507.90 kHz	7.4 %

El error relativo porcentual mostrado en la tabla 4.2 se calcula utilizando la siguiente ecuación.

$$\frac{f_{real} - f_{medida}}{f_{real}} \times 100\,\% \tag{4.2.1}$$

Cuando se pretende medir frecuencias arriba de 100kHz, se obtienen diferencias en las mediciones de las frecuencias ya que el sistema no tiene auto escalamiento. Esto sucede por la forma que se mide la señal usando el temporizador, ya que cuando se detecta el flanco de subida de la señal a medir, el temporizador inicia el conteo aumentando el contador cada unidad de tiempo en incrementos definidos hasta que se detecta el siguiente flanco de subida, lo cual genera la posibilidad de medir un rango de frecuencias que correspondan al mismo valor del contador. Es por eso que en frecuencias bajas el error es despreciable ya que el contador alcanza a aumentar considerablemente a comparación del periodo de la señal a medir y el rango de frecuencias que se miden

por cada valor del contador es mínimo; en cambio entre mas alta sea la frecuencia, el contador se alcanza a incrementar menos entre dos flancos de subida, provocando que se mida un rango de frecuencias relativamente grande respecto a la señal a medir en un solo incremento del contador.

En la tabla (4.2) se muestra su equivalencia en potencia óptica, de la frecuencia que entrega el sensor.

Tabla 4.3: Equivalencia entre la frecuencia de entrada y la *potencia óptica*.

Frecuencia de entrada	**Potencia óptica calculada**	**Potencia óptica medida**
0.99 kHz	-21.506 dBm	-21.506 dBm
10.06 kHz	-21.075 dBm	-21.075 dBm
100.1 kHz	-16.795 dBm	-16.77 dBm
199.90 kHz	-12.051 dBm	-11.98 dBm
500.5 kHz	2.239 dBm	2.593 dBm

Capítulo 5

Conclusiones y trabajo futuro

Con la realización de este trabajo fue posible implementar una aplicación en tiempo real, en la cual fue necesario utilizar diferentes tareas independientes que al ejecutarse todas ellas fuera posible realizar la aplicación, un esquema totalmente diferente al basado en interrupciones.

Los sistema basado en interrupciones, son utilizados comúnmente en aplicaciones simples, las aplicaciones de estos sistemas consisten en un superciclo infinito, el cual se ejecuta en primer plano, sobre el cual se van llamando las subrutinas o interrupciones según la aplicación vaya necesitando los módulos de código o funciones correspondientes, cuenta con rutinas de interrupción para manejar eventos asíncronos, así mismo todas las rutinas críticas son atendidas por interrupciones en un segundo plano. Esto provoca que cada vez que se ejecuta el ciclo, la respuesta en tiempo puede cambiar y el tiempo de respuesta general del sistema no pueda ser determinado, debido a que el tiempo depende en cuanto tiempo se ejecute el ciclo, y este varia dependiendo las interrupciones que se generen, además de que cada modificación en el código genera que todo el sistema se vea afectado ya que cambia el tiempo de ejecución del mismo.

En comparación de utilizar un esquema tradicional basado en interrupciones, una de las grandes ventajas de realizar una aplicación en un sistema operativo de tiempo real con tareas independientes, donde cada tarea tiene asignada una prioridad única, es que el sistema puede ser incrementado al asignarle más tareas, sin que las tareas con mayor prioridad que se haya tenido anteriormente sufran algún problema, ya que la nuevas tareas serían atendidas en el tiempo que no se encuentre ocupado el procesador, esto abre la posibilidad de poder agregar más funciones a cualquier aplicación en tiempo real sin que se vea afectado el sistema en general.

Al realizar la aplicación de un medidor de *potencia óptica* utilizando un sistema operativo de tiempo real comercial, en donde se han codificado las tareas correspondientes, se dio la necesidad analizar las características y determinar los requerimientos de las tareas para poder llevar a cabo la implementación, lo cual es un paso importante para determinar si es factible calendarizarlas en tiempo real. Al adaptar el funcionamiento del medidor de *potencia óptica* en tiempo real fue posible analizar, observar y mostrar a gran esquema como trabaja el calendarizador del RTOS para asignar los recursos del procesador, en base a jerarquías establecidas.

Para implementar un conjunto de tareas en tiempo real, particularmente utilizando el RTOS μC/OS-II primeramente es necesario estructurar las tareas en forma separada, que cada tarea cumpla con una función en específica independiente de otras tareas cumpliendo con las suposiciones que se explicaron en el capítulo 2, elaborar el código correspondiente para cada funciones, para posteriormente adaptarlas al sistema operativo asignándoles prioridades y un espacio de memoria para cada tarea, además de hacer un análisis de tiempos como se realizó con las tareas del medidor de *potencia*

óptica para determinar si es factible poder calendarizarlas.

Con esto quedan sentada las bases para el desarrollo de nuevos módulos, en otras aplicaciones que puedan ser parte de un sistema empotrado de tiempo real, así como los fundamentos para la creación, implementación y administración de tareas dentro del RTOS.

A modo de incrementar la funcionalidad del medidor de *potencia óptica* se puede trabajar en un futuro en agregarle más tareas como por ejemplo: una tarea encargada para realizar un auto escalamiento cuando el sistema se acerque a medir frecuencias altas donde se pierde resolución del sistema, esto puede ser posible de la misma manera que se crearon las tareas principales. Se puede agregar una tarea que funcione como un registrador de datos, el cual pueda ser programado para que cada cierto tiempo guarde en una *memoria SD* los datos obtenidos correspondientes a la medición. Se puede agregar otra tarea con el objetivo de controlar y medir posición de un motor cuya función sea posicionar el sensor en un ángulo o posición deseada.

Como se puede deducir de lo anterior el sistema está abierto a muchas mejoras, sin embargo de la perspectiva del sistema operativo de tiempo real, cada tarea será una más a calendarizar sin importar la aplicación que se le de a ésta.

Bibliografía

[1] A. S. Berger, *Embedded Systems Design.* Embedded Systems Design, 2002. ISBN 1-57820-073-3.

[2] D. Andrews, I. Bate, T. Nolte, C. M. O. Pérez, y S. M. Petters, "Impact of embedded systems evolution on rtos use and design," *Proceedings of the 1st Workshop on Operating System Platforms for Embedded Real-Time Applications*, 2005.

[3] "Baja california: Estrategia de evolución hacia una economía basada en la tecnología y el conocimiento documento de la secretaría de desarrollo económico del gobierno del estado de baja california," Febrero 2007. Documento consultado en marzo de 2007.

[4] MEDIDA, Inc., "Laboratorio de sistemas empotrados, (apoyado por el programa para el desarrollo de la industria de software -prosoft-)," 2007.

[5] T. Noergard, *Embedded Systems Architecture.* Elsevier, 2005. ISBN 0-7506-7792-9.

[6] M. Barr, *Programming Embedded Systems in C and C++.* O'Reilly, 1999. ISBN 1-56592-354-5.

[7] K. Arnold, *Embedded Controller Hardware Design.* Newnes, 2004. ISBN 1-87807-52-3.

[8] Q. Li y C. Yao, *Real-Time Concepts for Embedded Systems*. CMP Books, 2003. ISBN 1-57820-124-1.

[9] A. Burns y Andy Wellings, *Real-Time Systems and Programming Languages*. Pearson, Addison Wesley, 2001. ISBN 0-201-72988-1.

[10] C. M. Krishna y K. Shin, "Real-time systems," *MIT Press and McGraw-Hill Company*, 1997.

[11] F. Panzieri y R. Davoli, "Real time systems: a tutorial - technical report," *Laboratory for Computer Science*, 1993.

[12] S. Manolache, P. Eles, y Z. Peng., *Real-Time Applications with Stochastic Task Execution Times*. Springer, 2007. ISBN 1-4020-5505-6.

[13] J. Catsoulis, *Designing Embedded Hardware*. O'Reilly, 2005. ISBN 0-596-00755-8.

[14] C. Walls, *Embedded Software: The Works*. Newnes, 2006. ISBN 0-7506-7954-9.

[15] J. Labrosse, *Embedded Systems Building Blocks*. CMP Books, segunda edición ed., 2002. ISBN 0-87930-604-1.

[16] A. Sloss, D. Symes, y C. Wright, *ARM System Developer's Guide, Designing and Optimizing System Software*. Elsevier, 2004. ISBN 1-55860-874-5.

[17] M. Mano, *Computer System Architecture*. Prentice Hall, 1993. ISBN 0-13-175563-3.

[18] B. A. Forouzan, *Introducción a la ciencia de la computación*. I.T.P. Latin América, 2003. ISBN 9706862854.

[19] N. Audsley, A. Burns, R. Davis, K. Tindell, y A. Wellings, "Real-time system scheduling," *Predictably Dependable Computing Systems. ESPRIT Basic Research Series, Springer*, 1995.

[20] G. Butazzo, "Hard real-time systems and programming languages," *University of York, Addison Wesley*, 1996.

[21] C. Liu y J.W. Layland., "Scheduling algorithms for multiprogramming in a hard real-time enviroment," *Journal of the Association for Computing Machinery*, 1973.

[22] L. Sha, Rajkumar, y S. Sathaye, "Generalized rate-monotonic scheduling theory a framework for developing real-time systems," *Proceedings of the IEEE*, pp. 82(1):68–82, Enero 1992.

[23] J. Lehoczky, L. Sha, y Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," *Proceedings of the IEEE Real-Time System Symposium*, pp. 166–171, Diciembre 1989.

[24] N. Audsley, A. Burns, M. Richardson, y A. Weellings, "Hard real-time scheduling: the deadline monotonic approach," *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

[25] N. Audsley, K. Tindell, y A. Burns, "The end of the line for static cyclic scheduling?," *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, pp. 36–41, 1993.

[26] J. Lehoczky y S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Proceedings of the IEEE Real-Time Systems Symposium*, 1996.

[27] S. Ramos-Thuel y J. Lehoczky, "On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems," *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 160–171, 1993.

[28] M. Joseph y P. Pandya, "Finding response times in a real-time system," *BCS Computer Journal 29(5)*, pp. 390–395, 1986.

[29] kfrazer, "Real-time operating system schedulling algorithms," 1997.

[30] R. K. Clark, "Scheduling dependent real-time activities," *PhD dissertation, Garnegie Mellon University*, 1990.

[31] B.-E. D. M. for Real-Time Schedulling, "Phd dissertation," *PhD dissertation, Garnegie Mellon University*, 1986.

[32] S. Baruah, N. Cohen, G. Plaxton, y D. Varvel, "Proportionate progress: A nation of fairness in resource allocation," *Algorithmica, Volume 15, Number 6*, June 1996.

[33] P. Ayala, J. Munguia, E. Álvarez, J. Loya, A. Vega, y J. González, "Diseño y constricción de un medidor de potencia óptica didáctico, basado en un convertidor de luz a frecuencia, y un microcontrolador ATTINY88," *SOMI XXIV, Congreso de Instrumentación, Mérida, Yucatán.*, 2009.

Anexos

Anexo A

Gráficas de tareas

A continuación se presentan imágenes obtenidas del analizador lógico que se utilizaron para describir el comportamiento de las tareas en RTOS. Para la medición se utilizó el equipo "Aligent 55622D - Mixed Signal Oscilloscope 100Mhz".

En la figura (A.1) se muestra la imagen de las tareas cuando son liberadas al mismo tiempo, correspondiente al diagrama de Gantt de la figura (4.1) del capítulo 4.

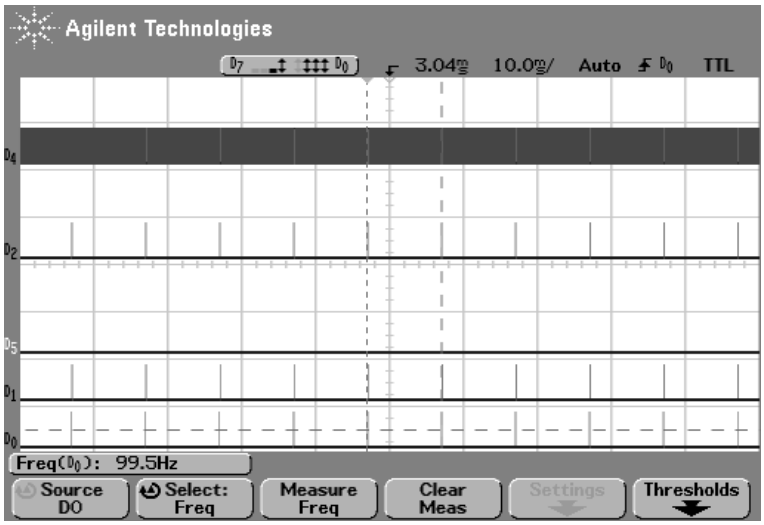

Figura A.1: Diagrama general de tiempos. D_0= Rutina del *tick*; D_1= Ejecución de la tarea inicial; D_2= Ejecución de la tarea del sensor; D_4= Tarea del μC/OS-II cuando no se está ejecutando ninguna tarea de aplicación.

La figura A.2 muestra la medición del tiempo de ejecución de la tarea del sensor:

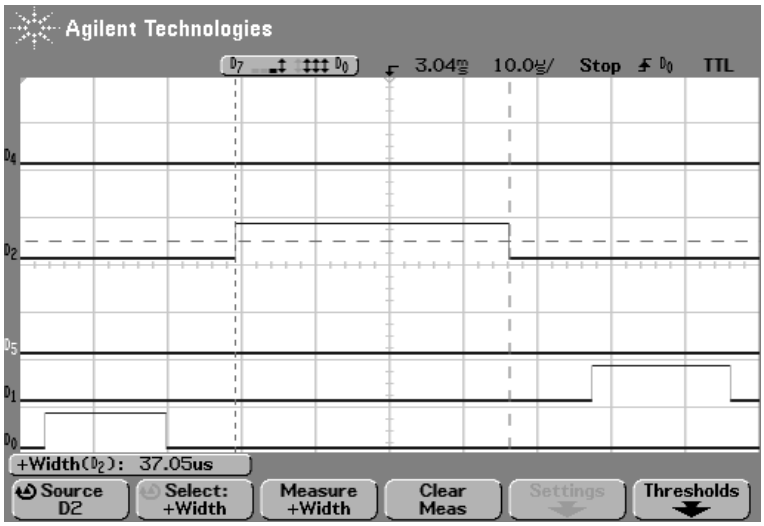

Figura A.2: Tiempo de ejecución de AppTareaSensor.

En la figura A.3 se muestra la imagen que se obtuvo en el analizador lógico donde se puede observar un tiempo de 9,4μs que se tarda para iniciar la tarea *AppTareaSensor* después de que se finaliza la ejecución de la tarea del *tick*.

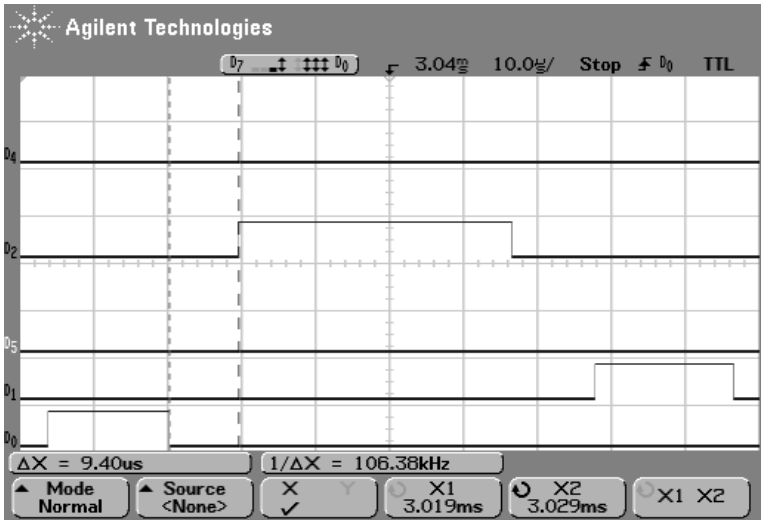

Figura A.3: D_0,Rutina del Tick; D_2,Ejecución de la tarea AppTareaSensor.

En la figura A.4 se muestra el tiempo de 11,2μs que tarda en iniciar la tarea *AppTareaInicial* después de atender la tarea *AppTareaSensor*.

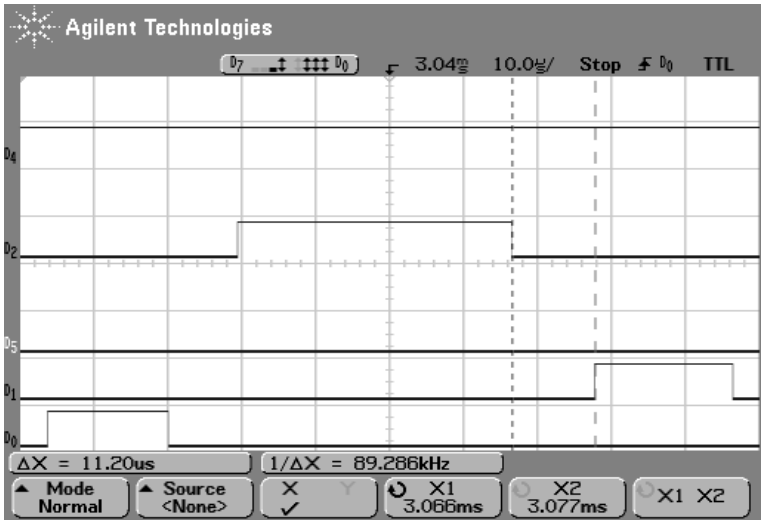

Figura A.4: D_2= Ejecución de la tarea AppTareaSensor; D_1= Ejecución de la tarea AppTareaInicial.

En la figura (A.5) se muestra el comportamiento obtenido en el analizador lógico del conjunto de tareas del diagrama de Gantt de la figura(4.4) del capítulo 4.

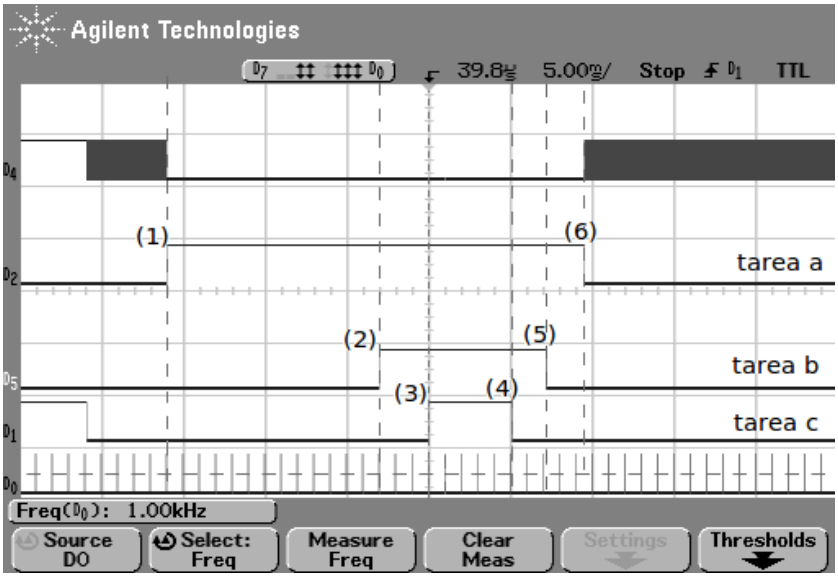

Figura A.5: D_0= Ticks; D_1= Tarea c; D_2= Tarea a; D_5= Tarea b.

Anexo B

Herramientas Utilizadas

A continuación se presenta una descripción de las herramientas utilizadas en el experimento.

B.1. Ambiente de desarrollo IAR Embedded Workbench

El Ambiente de desarrollo integrado (IDE, por sus siglas en inglés) IAR Embedded Workbench para *ARM* es un conjunto de herramientas de desarrollo para aplicaciones empotradas. Está constituido por el compilador IAR C/C++ ™, un ensamblador, un enlazador, el administrador de proyectos, y el depurador C-SPY ®. El ambiente de desarrollo IAR Embedded Workbench es capaz de generar código eficiente y confiable para dispositivos *ARM*. Particularmente se trabaja con la versión libre con la limitante de un límite de código de 32Kb.

Este ambiente soporta un amplio apoyo para múltiples dispositivos. Es compatible con núcleos de procesadores ARM7, ARM7E, ARM9, ARM9E, ARM10E, ARM11, SecurCore, Intel XScale, Cortex-M0, Cortex-M1, Cortex-M3, Cortex-M4, Cortex-R4(F) y Cortex-A5. Contiene archivos ya elaborados de definiciones de registros de los

periféricos y cargadores de la memoria *FLASH* para la mayoría de los dispositivos y tarjetas de evaluación, así como mas de 2400 ejemplos de proyectos de ejemplos para las tarjetas de evaluación de IAR Systems, Analog Devices, Atmel, Energy Micro, Freescale, OKI, NXP, ST, Texas Instruments, Toshiba, etc.

Con el depurador entre otras cosas se tiene la capacidad de generar puntos de interrupción por hardware y software, se puede visualizar las llamadas de la *pila*, ver el comportamiento de las variables, ver los datos de entrada y salida, simular interrupciones, ver el código de operación generado, se puede obtener información sobre el consumo de potencia de la aplicación correlacionada al código fuente, etc.

B.2. Tarjeta de Desarrollo IAR STR750-SK

La tarjeta de desarrollo IAR STR750-SK utiliza un microcontrolador de *ST Microelectronics* STR750FV2. Es un microcontrolador de 32 bits. lo cual ofrece alto rendimiento, bajo consumo de potencia y código denso, cuenta con un extenso conjunto de periféricos, además que incorpora una memoria *ROM FLASH*. Es compatible tanto con 3.3V y 5V, es operable en un rango de temperaturas desde -40 a 105 ℃. Esto lo hace como un microcontrolador de propósito general, adecuado para una amplia gama de aplicaciones como por ejemplo electrodomésticos, controladores de motores a pasos, periféricos USB, sistemas de alarmas, controladores lógicos programables, equipo portátil, equipo medico, etc.

Cuenta con memoria embebida *FLASH* de 256 Kbytes de capacidad disponible en el banco de memoria 1 que se utiliza para almacenar el programa y los datos,

adicionalmente se cuenta un una memoria RWW (Read While Write) de 16Kb con la particular característica que puede ser grabada o borrada mientras se ejecuta la aplicación. También cuenta con memoria embebida SRAM de 16Kbytes de alta velocidad la cual es leída o escrita a la velocidad de ejecución del procesador sin utilizar tiempos de espera.

Cuenta con una SMI (Interfaz de memoria serial) capaz de acceder directamente hasta cuatro dispositivos *FLASH*. Puede ser usado para acceder a los datos, así como ejecutar el código directamente o iniciar una aplicación en memoria externa. La memoria esta direccionada en cuatro bancos de memoria de hasta 14Mb cada uno.

El *CPU* puede trabajar a una velocidad de hasta 64MHz. La tarjeta utiliza un cristal de 4MHz, sin embargo que se puede configurar a diferente frecuencia gracias al PLL (Lazo de fase cerrado) y a varios preescaladores. De este modo se puede configurar los relojes independientes de los periféricos a la frecuencia deseada.

El microcontrolador tiene 3 temporizadores basados en un contador de 16 bits con auto recarga, cuentan con dos entradas de captura y dos salidas con comparación. Cuenta con un temporizador PWM (Modulador de ancho de pulso) que puede ser usado para control de motores.

Tiene múltiples puertos de comunicación: I^2C (inter-integrated circuit), cuya interfaz puede operar en modo multi-maestro o en modo esclavo; Puede trabajar en modo estándar y modo rápido (hasta 400KHz). Cuenta con puertos UART; los tres puertos son capaces de comunicarse a velocidades de hasta 2 Mbit/s.

Cuenta con dos puertos SSP (Synchronous Serial Peripheral); que pueden comunicarse con velocidades de hasta 8 Mbit/s en el puerto 1 y en el puerto 2 hasta 16 Mbit/s en modo de interfaz de 4 *pines* estándar full-duplex como dispositivo maestro y hasta 2.66 Mbit/s como esclavo. Soporta los protocolos de Motorola SPI o TI SSI.

El Puerto CAN (Controller Area Network), es compatible con la especificación 2.0 parte B, el puerto puede manejar velocidades de hasta 1 Mbit/s. Puede transmitir y recibir tramas estándar con identificadores de 11-bits así como tramas extendidas con identificadores de 29 bits, hasta 32 objetos de mensaje que son manejados a través de un buffer RAMinterno.

La interfaz USB es compatible con la velocidad de 12Mbs, cabe mencionar que contiene una fuente de reloj dedicado para este dispositivo de 48MHz.

El ADC es de 10 bits, convierte hasta 16 canales externos en modo de escaneo o de disparo único. Cuando trabaja en modo de escaneo, el convertidor realiza una conversión continua en un grupo de entradas análogas. El tiempo mínimo de conversión es de 3.75 μs. Los eventos generados por los temporizadores, pueden ser conectados internamente al disparo de inicio del ADC, esto permite sincronizar los temporizadores con la conversión.

Entradas y salidas de propósito general (GPIO). Cuenta con 72 *pines* de entrada y salida, los cuales pueden ser configurados por software como salidas, como entradas o como función alterna, según al puerto con el que esté compartido.

B.3. Sistema Operativo de Tiempo Real μC-OS II

El μC/OS-II es un *kernel* portable, que puede ser implementado en la memoria *ROM* de un microcontrolador, escalable, reentrante y multitarea de tiempo real para microcontroladores, microprocesadores y DSPs.

μC-OS II tiene como base el μC-OS, que fue publicado por primera vez en 1992 por Jean J. Labrosse. Miles de personas en el mundo están usando μC-OS II y μC-OS en muchos tipos de aplicaciones en sistemas empotrados, así como muchas universidades para sus cursos de sistemas de tiempo real. μC-OS II es compatible con μC-OS v1.11; la última versión publicada. A continuación se muestran la características mas importantes de este RTOS.

La mayor parte del μC-OS II está programado en código portable ANSI-C, mientras que el código específico del procesador está escrito en ensamblador. La programación en ensamblador se ha mantenido al mínimo posible para que el RTOS pueda portarse fácilmente a otro procesador. Éste puede ser portado a muchos microprocesadores siempre y cuando su arquitectura contenga un apuntador de *pila* y registros del procesador que puedan ser metidos y sacados de la *pila*. Además el compilador de C debe proveer tanto lenguaje en ensamblador o extensiones del lenguaje que permita habilitar y deshabilitar las interrupciones del microprocesador desde C. μC-OS II puede operar en la mayoría de los microcontroladores o microprocesadores y los procesadores digitales de señales (DSP) de 8, 16, 32 y aún de 64 bits.

Ya que el μC-OS II esta diseñado para sistemas empotrados, el RTOS puede ser grabado en la memoria *ROM* del microcontrolador dejándolo empotrado en el sistema.

Además es reentrante, lo que significa que el μC-OS II siempre ejecuta la tarea de mayor prioridad que esté lista, en la actualidad la mayoría de los RTOS son reentrantes. Puede manejar hasta 64 tareas y cada tarea tiene una prioridad única asignada. Cada tarea requiere su propio espacio en la *pila*, μC-OS II permite asignar diferente cantidad de espacio para cada tarea dependiendo de las necesidades de la aplicación.

Cuenta con un administrador de interrupciones. Las interrupciones pueden suspender la ejecución de una tarea, si una tarea de alta prioridad es despertada por una interrupción, la tarea de alta prioridad se ejecuta tan pronto como todas las interrupciones anidadas sean completadas. Se pueden anidar hasta 256 interrupciones.

Es robusto y confiable, como μC-OS II está basado de μC-OS, el cual ha sido utilizado cientos de veces en aplicaciones comerciales desde 1992. μC-OS II usa el mismo núcleo y la mayoría de las funciones que μC-OS. En julio del 2000, μC-OS II fue certificado en productos de aviones por la Administración Federal de Aviación de Estados Unidos para el uso en equipo aviones comerciales. Cada característica, función y línea de código del μC-OS II ha sido examinada y probada para demostrar de que es lo suficientemente segura y robusta para ser utilizado en sistemas críticos donde dependen vidas humanas.

B.4. Sensor TSL230R-LF

El convertidor programable de luz a frecuencia combina un foto-diodo configurable de silicio y un convertidor de corriente a frecuencia en un circuito integrado con tecnología CMOS. La salida que entrega el circuito integrado puede ser tanto un tren de pulsos o una señal cuadrada con frecuencia directamente proporcional a la intensidad de la luz. La sensibilidad del dispositivo puede ser seleccionada en tres rangos, con esto se tiene dos décadas de ajuste. Toda la escala de la frecuencia de salida puede ser escalada por una de los cuatro valores preestablecidos. Las entradas y las salidas son compatible con TTL, permitiendo una interfaz directa con el microcontrolador. En la figura (B.1) se muestra un diagrama a bloques del funcionamiento interno del sensor.

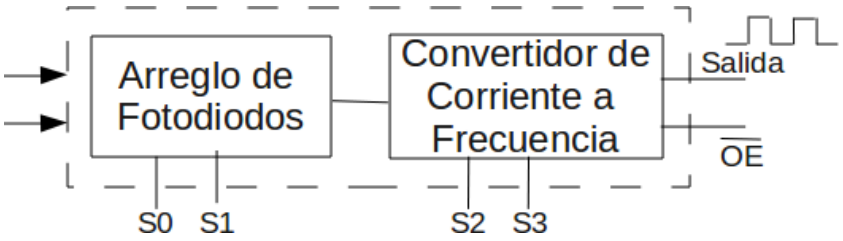

Figura B.1: Diagrama a bloques del sensor TSL230R-LF.

La sensibilidad es controlada por dos entradas lógicas (S0 y S1) es ajustada controlando la apertura del sensor, de modo que cambia el área efectiva del foto-diodo por el mismo factor de ajuste, que puede ser 1x, 10x y 100x.

La escala de la frecuencia de salida es controlada por las entradas lógicas (S2 y S3), esto se lleva acabo internamente en el circuito integrado mediante divisores de frecuencia. Se puede dividir por 2, 10, 100 y por 1 (sin división). Esto permite poder obtener mayor resolución en las mediciones cuando la frecuencia de salida está escalada.

Anexo C

Código de Tareas

C.1. Descripción de las funciones y tareas de la aplicación

Se muestra y se describe el código utilizado para realizar las tareas en tiempo real usando el sistema operativo de tiempo real μC/OS-II.

C.1.1. Función main()

```
   void  main (void){
(1)  BSP_IntDisAll();
(2)  OSInit();
(3)  OSTaskCreate(AppTareaInicial,
            (void *)0,
            (OS_STK *)&AppTareaInicialStk[APP_TAREA_INICIAL_STK_SIZE - 1],
            APP_TAREA_INICIAL_PRIO);
(4)  OSStart();
   }
```

Como la gran mayoría de los programas en "C" el código de aplicación inicia en la función `main()`, a continuación se describe el código contenido en la función main.

(1) Es necesario deshabilitar todas las interrupciones mientras se realiza el proceso de la inicialización de la aplicación, para asegurar que no ocurra alguna interrupción en el sistema antes de que esté completamente inicializado.

(2) Se debe de llamar a esta función antes de llamar cualquier otra función o servicio del μC/OS-II. Con esta función (`OSInit()`) inicializamos todas las variables y estructuras del μC/OS-II y además se crea una tarea llamada `OS_TaskIdle()`, esta tarea propia del sistema operativo μC/OS-II, siempre está lista para ser ejecutada y tiene configurada por defecto la prioridad más baja definida por la constante OS_LOWEST_PRIO.

(3) Con la función `OSTaskCreate()` se crea la tarea inicial (dentro de esta tarea después se crean las otras tareas) con el objetivo que posteriormente el sistema operativo administre las tareas. Para crear la tarea es necesario pasarle cuatro argumentos a esta función: el primer argumento es un apuntador al código de la tarea. El segundo argumento es un apuntador al dato que se desea pasar a la tarea cuando ésta inicia su ejecución, el cual en este caso no será utilizado, ya que no hay ningún dato que pasar, y es por eso que se utiliza un apuntador nulo. El tercero es un apuntador al tope de la *pila* que es asignada a la tarea. Y por último el cuarto argumento es la prioridad que se le desea asignar a la tarea.

(4) Al llamar a esta tarea se le da control al μC/OS-II para que comience a administrar las tareas.

C.1.2. Función AppTareaInicial()

Esta es la tarea inicial, la cual está hecha para iniciar los módulos, necesarios para la aplicación, crear las otras tareas y por último quedarse en un ciclo infinito realizando su actividad correspondiente.

```
static  void  AppTareaInicial (void *p_arg){ejecuta
     CPU_INT08U   k;
(1)  BSP_Init();
(2)  DispInit(2, 16);
(3)  AppCrearTareas();
(4)  EXTINT_Mbox = OSMboxCreate((void *)0);
(5)  EXTINT_Init(EXTINT_ISRHandler);

(6)  while (DEF_TRUE) {
(7)    GPIO_WriteBit(GPIO0, GPIO_Pin_1,   Bit_SET);
       for(k=0;k<5;k++){
 LED_On(1);
 LED_Off(1);
       }
(8)    GPIO_WriteBit(GPIO0, GPIO_Pin_1,   Bit_RESET);
(9)    OSTimeDly(1);
  }
}
```

(1) Al llamar BSP_Init() se ejecuta una rutina en la cual están todas las inicializaciones de los módulos y periféricos necesarios para llevar a cabo la aplicación,

como son los temporizadores, los *pines* I/O, los botones de usuario, los Relojes, etc.

(2) Inicialización del LCD utilizando el módulo μC/LCD.

(3) Dentro de esta función se crean las otras dos tareas exactamente de la misma forma previamente explicada, las tareas creadas son `AppTareaSensor()` y `AppTareaLCD()`.

(4) Con esta función propia del μC/OS-II, se configura un canal de comunicación entre el manejador de la interrupción externa, encargada de atenderse cuando haya una solicitud por parte de usuario (cuando se haya presionado un botón).

(5) Se configura la interrupción externa.

(6) Cualquier tarea controlada por el μC/OS-II debe estar en un ciclo infinito, dentro del ciclo infinito se enciende y apaga el LED 1 de la tarjeta de desarrollo, esta tarea no es propia de la función del objetivo del medidor de *potencia óptica*, sin embargo aquí posteriormente se puede sustituir código aplicable.

(7) El *pin* 1 de la tarjeta de desarrollo se pone en alto para indicar en el analizador lógico cuando se el μC/OS-II inicia a atender esta tarea y posteriormente (8) se pone en nivel bajo para indicar que se ha terminado la tarea.

(9) La función `OSTimeDly()` es una función del μC/OS-II, cuando dicha función es llamada se indica el retraso en *tick*s que se desea para la tarea. A todas las tareas se le

asignan deliberadamente un retraso de un *tick*, para observar el comportamiento en el analizador lógico cuando todas las tareas son liberadas en el mismo instante de tiempo, posteriormente se le asignará un retraso mayor, para un mejor funcionamiento de la aplicación.

En la función `OSTimeDly()` al asignarle un *tick*, significa que la tarea estará suspendida hasta el siguiente *tick*, indicarle un 0 de argumento indica que no hay ningún retraso y la tarea siempre estará lista para ser ejecutada, por lo que si la aplicación requiere que por lo menos la tarea se retrase un *tick*, es necesario darle 2 *ticks* de retraso.

C.1.3. Función AppTareaSensor()

En esta tarea como su nombre lo indica, se realiza la lectura del dato más reciente leído por un módulo del temporizador de la tarjeta desarrollo, el cual mide la frecuencia que de entrada de la señal cuadrada, para realizar la conversión de frecuencia a potencia en decibeles se utiliza la fórmula (C.1.1):

$$P(dBm) = \frac{f - 453396{,}4286}{21035{,}71429} \tag{C.1.1}$$

La fórmula (C.1.1) para utilizada para tener el valor de la *potencia óptica* en decibeles. Es obtenida mediante una interpolación lineal de la gráfica obtenida por [33], en la cual se realizó una caracterización con el sensor de luz.

```
static  void  AppTareaSensor (void *p_arg){
   while (DEF_TRUE) {
     GPIO_WriteBit(GPIO2, GPIO_Pin_4,   Bit_SET);
(1)  TIM1ICAP1 = TIM_GetICAP1(TIM1);
(2)  TIM_ClearITPendingBit(TIM1, TIM_IT_IC1|TIM_IT_IC2);
(3)  Pot_Frec = (32000 / (float)TIM1ICAP1);
(4)  Pot_dB= ( (32000 / (float)TIM1ICAP1)*1000 - 453396.4286 ) / 21035.71429;
     GPIO_WriteBit(GPIO2, GPIO_Pin_4,   Bit_RESET);
     OSTimeDly(1);
    }
}
```

(1) Con la función `TIM_GetICAP1()` se realiza la lectura del registro correspondiente al temporizador asociado al sensor.

(2) Después de hacer una lectura es necesario darle reset a los *pines*es "TIM_IT_IC1" y "TIM_IT_IC2" del temporizador, para que proceda a realizar la siguiente lectura.

(3) Con el valor del registro, se determina la frecuencia de la señal de entrada.

(4) Utilizando la fórmula (C.1.1) se determina la potencia en dBm.

C.1.4. Función AppTareaLCD()

A continuación se muestra el código de la función que se encarga de mandar los mensajes a la pantalla LCD.

```
static  void  AppTareaLCD (void *p_arg){
    char            str[20];
    CPU_INT08U     *msg;
    CPU_INT32U      var;
    (void)p_arg;
    DispInit(2, 16);
    while (DEF_TRUE) {

      GPIO_WriteBit(GPIO2, GPIO_Pin_3,   Bit_SET);
(1)   msg      = (CPU_INT08U *)OSMboxAccept(EXTINT_Mbox);
(2)   if (msg != (CPU_INT08U *)0) {
DispClrScr();
        var = *msg;
      }
(3)   if (var == 1) {
DispStr(0, 0, "  Frecuencia  ");
        sprintf(str,"P:%.3f KHz",Pot_Frec);
        DispStrChar(1, 0, str);
      }
      else if (var == 2) {
DispStr(0, 0, "  Potenica dBm  ");
        sprintf(str,"P:%.3f dBm",Pot_dB);
```

```
        DispStrChar(1, 0, str);
      }
      else{
DispStr(0, 0, "Medidor de Pot");
        DispStr(1, 0, " Optica  Ver1.0 ");
      }
      GPIO_WriteBit(GPIO2, GPIO_Pin_3,   Bit_RESET);
      OSTimeDly(1);
    }
}
```

(1) Se obtiene el dato que se haya leído en la rutina de interrupción, cuando el usuario presione un botón.

(2) Si el dato leído no es un dato nulo (cuando el operador no haya presionado algún botón) el valor de la variable "var" no cambia, por lo tanto se mantiene la opción de la pantalla.

(3) Dependiendo del botón que se haya presionado, la variable "var" indicará la pantalla a desplegar.

C.1.5. Función EXTINT_ISRHandler()

Es la función correspondiente a la interrupción externa, la cual se genera cuando el usuario ha presionado un botón. El valor asignado a la variable que es mandada por mensaje depende del botón presionado.

```
static  void  EXTINT_ISRHandler (void)
{
    CPU_BOOLEAN  change;
    change = DEF_FALSE;
    if (EXTIT_GetITStatus(EXTIT_ITLine7) == SET) {
        change      = DEF_TRUE;
        if (EXTIT_GetITStatus(EXTIT_ITLine8) == RESET) {
            EXTINT_Char = 1;
(1)         OSMboxPost(EXTINT_Mbox, &EXTINT_Char);
        }
        EXTIT_ClearITPendingBit(EXTIT_ITLine7);
    }
    if (EXTIT_GetITStatus(EXTIT_ITLine8) == SET) {
        if (change == DEF_FALSE) {
            EXTINT_Char = 2;
(2)         OSMboxPost(EXTINT_Mbox, &EXTINT_Char);
        }
        EXTIT_ClearITPendingBit(EXTIT_ITLine8);
    }
(3) EIC->IPR = (CPU_INT32U)(1 << EXTIT_IRQChannel);
}
```

(1) Cuando dentro de la rutina de interrupción se ha determinado que el botón presionado fue el botón 1, se manda la variable `EXTINT_Mbox` por mensaje.

(2) De la misma manera que en el punto anterior, se realiza lo correspondiente cuando el botón 2 es el que se ha presionado.

(3) Se limpia el la bandera que corresponde al bit de la interrupción externa.

Glosario

A

ARM (Advanced RISC Machine) Es un arquitectura RISC de microprocesadores de 32 bits desarrollados por ARM Holdings.

C

CPU (Central Processing Unit) Contiene una unidad aritmética y lógica para la manipulación de datos, varios registros para almacenar los datos y circuitos de control para leer de la memoria y ejecutar instrucciones.

F

FLASH Se conocen como circuitos de memoria estática, debido a que el contenido de cada posición de la memoria se mantiene en tanto se mantenga la alimentación eléctrica del circuito integrado.

K

kernel El kernel o núcleo es la parte de un sistema operativo responsable de la administración de tareas y la comunicación entre éstas.

L

LCD (Liquid Crystal Display). Tecnología utilizada en monitores y pantallas, funciona a través de ventanas de cristal líquido que permiten el paso de

la luz cuando son expuestos a un voltaje controlado y es utilizada para mostrar imagen o texto.

LED (Light Emitting Diode). El diodo emisor de luz es un dispositivo semiconductor, que al ser polarizado de forma directa libera energía en forma de fotones, los cuales son emitidos como luz visible.

M

mainframe En un sistema operativo mainframe múltiples usuarios acceden simultáneamente a través de otras terminales. Estos usuarios comparten recursos y pueden intercambiar información. En tales casos, el sistema operativo se diseña para maximizar la utilización de recursos, asegurar que todo el tiempo de CPU, memoria y E/S disponibles se usen de forma eficiente y que todo usuario disponga sólo de la parte equitativa que le corresponde.

memoria SD Es una memoria tipo flash portatil de acceso aleatorio con bajo consumo de potencia, ampliamente utilizadas en dispositivos electrónicos comerciales como cámaras digitales, agendas electrónicas, reproductores de música,etc.

N

non preemptive (No reentrante) Es un sistema que no permite ser suspendido, si una tarea de mayor prioridad está esperando ser atendida, ésta será atendida cuando el procesador sea desocupado por la tarea actual.

P

pila (Stack) Una pila es una estructura de datos en la cual, el primer dato que se obtiene de ella es el último que se colocó.

pin (Terminal) Terminal o patilla a cada uno de los contactos metálicos de un conector o de un componente electrónico, fabricado de un material conductor de electricidad.

potencia óptica Es el flujo de energía luminosa que atraviesa determinado punto en un tiempo especificado.

preemptive (Reentrante) Un sistema es reentrante (preemptive) cuando el sistema permite suspender la ejecución de un proceso, para ejecutar algún otro proceso de mayor prioridad y la información de estado del proceso original pueda ser guardado en memoria para posteriormente continuar con con el proceso suspendido .

R

RAM (Random Access Memory) Memoria por la cual el CPU puede acceder a cualquier parte de la memoria en forma aleatoria independientemente de su posición en la escritura.

ROM (Read Only Memory) Memoria de solo lectura, es una memoria que sólo ejecuta operación de lectura; no tiene la posibilidad de escritura.

S

sobrecargas del sistema Utilización adicional de los recursos que realiza el sistema operativo cuando éste suspende la ejecución de una tarea, para poder atender otra tarea de mayor prioridad.

T

tick Un contador que cuenta las interrupciones en intervalos del temporizador y es utilizado por un sistema operativo de tiempo real para los servicios básicos del temporizador, tales como proporcionar la resolución mínima para los tiempos de esperas en las llamadas.

TTL (Transistor-Transistor Logic). La lógica transistor a transistor es una tecnología de fabricación de circuitos digitales que utiliza el transistor bipolar como el elemento principal en el circuito, normalmente trabaja con alimentación de 5v.