INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL

"ESTUDIO DEL SISTEMA OPERATIVO UCOS Y SU APLICACIÓN EN LA DETECCIÓN DE SECUENCIAS DE EVENTOS"

TESINA

QUE PARA OBTENER LA

ESPECIALIDAD EN SISTEMAS INMERSOS

PRESENTA:

SAMUEL HERRERA TORRES

BAJO LA DIRECCIÓN DE:

M. EN C. DAVID JAIME SAUCEDO MARTÍNEZ

DICIEMBRE 2011 **TIJUANA, B. C., MÉXICO**

SIP-14-E

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESINA

En la Ciudad de Tijuana, B.C. siendo las 15:15 horas del día 8 del mes de diciembre del 2011 se reunieron los miembros de la Comisión Revisora de la Tesina, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de CITEDI para examinar la tesina titulada:

ESTUDIO DEL SISTEMA OPERATIVO uCOS Y SU APLICACIÓN EN LA DETECCIÓN DE SECUENCIAS DE EVENTOS.

Presentada por el alumno:

HERRERA	TORRES	SAMUEL
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	0	0	3	0	4

aspirante de:

ESPECIALIDAD EN SISTEMAS INMERSOS

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESINA***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director(a) de tesina

M.C. DAVID JAIME SAUCEDO MARTÍNEZ

DR. ROBERTO HERRERA CHARLES

ESP. EDUARDO BARBA CERVANTES

PRESIDENTE DEL COLEGIO DE PROFESORES

DR. LUIS ARTURO GONZÁLEZ HERNÁNDEZ

S. E. P.
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION Y DESARROLLO
DE TECNOLOGIA DIGITAL
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de Tijuana, Baja California, el día 14 del mes Diciembre del año 2011, el (la) que suscribe Samuel Herrera Torres alumno (a) del Programa de ESPECIALIDAD EN SISTEMAS INMERSOS con número de registro A100304, adscrito al CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de M. en C. David Jaime Saucedo Martínez y cede los derechos del trabajo intitulado Estudio del sistema operativo VCOS y su aplicación en la detección de secuencias de eventos, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección: Av. del Parque No. 131O, Mesa de Otay, Tijuana, Baja California, México C.P. 22510. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Herrera Torres S.
Samuel Herrera Torres.

Nombre y firma

Estudio del sistema operativo ucos y sus aplicación en la detección de secuencias de eventos

Resumen

En este trabajo se presentan las características y clasificaciones básicas de los sistemas de tiempo real y los sistemas operativos de tiempo real (RTOS). Se muestran además las características principales del RTOS µC/OS-III y de la arquitectura ARM con un ejemplo de aplicación en un sistema embebido basado en un microcontrolador con núcleo ARM (Cortex-M3). Haciendo uso de algunos de los servicios que presta un sistema operativo de tiempo real en un sistema embebido, se utilizan APIs para el desarrollo de un detector de eventos y un generador de eventos para pruebas. El generador sigue el modelo cliente-servidor entre el sistema embebido y una PC.

Palabras clave: RTOS, ARM, UCOS, Cortex-M3

Study of the UCOS operating system and it's application in the detection of a sequence of events.

Abstract

This work presents the basic characteristics and classifications of real time systems and the real time operating systems (RTOS). It also presents the main characteristics of the µC/OS-III RTOS and the ARM architecture with an example of its application in an embedded system based in the ARM (Cortex-M3) architecture. Some of the services given by the RTOS in an embedded system and APIs are used to develop an event detector and generator to test the system. The event generator follows a client-server model between the embedded system and a PC.

Key words: RTOS, ARM, UCOS, Cortex-M3

Agradecimientos

A mi director de tesina por su apoyo y observaciones para el desarrollo de esta tesina.

A todos mis maestros que me compartieron sus conocimientos y al CITEDI/IPN por permitirlo.

Tijuana, Baja California, México Samuel Herrera Torres

Diciembre de 2011

Índice de contenido

Índice de tablas

Índice de Figuras

Índice de Acrónimos en inglés utilizados en esta tesina

RTOS	*Real Time Operating System* (Sistema operativo de tiempo real)
ARM	*Advanced RISC Machines* (Máquinas RISC avanzadas)
IP Core	I*ntellectual Property core* (Núcleo de propiedad intelectual)
µC/OS	Micro C Operative System (Sistema Operativo µC)
ANSI	*American National Standards Institute* (Instituto nacional de estándares americanos)
ROM	*Read Only Memory* (Memoria de solo lectura)
RISC	*Reduced Instruction Set Computer* (Computadora de conjunto de instrucciones reducido)
API	*Application programming interface* (Interfaz de programación de aplicaciones)
SRAM	*Static Random-Access Memory* (Memoria estática de acceso aleatorio)
USB OTG	*Universal Serial Bus On the Go* (Bus universal en serie en movimiento)
CAN	*Controller area network* (Red de área de controladores)
SD/MMC	*Secure Digital/MultiMediaCard*
IDE	*Integrated Development Environment* (Entorno de desarrollo integrado)
MISRA	*the Motor Industry Software Reliability Association* (Asociación de fiabilidad del software en la industria automotriz)
AMD	*Advanced Micro Devices* (Micro dispositivos avanzados)
GPIO	*General Purpose Input/Output* (Entrada/salida de propósito general)
ASCII	*American Standard Code for Information Interchange* (Código “standard” americano para el intercambio de información)
BSP	*Board Support Package* (Paquete de soporte de tarjeta)
CPU	*Central Processing Unit* (Unidad central de procesamiento)
PWM	*Pulse-Width Modulation* (Modulación de ancho de pulso)
DMA	*Direct Memory Access* (Acceso directo a memoria)
USART	*Universal Asynchronous Receiver/Transmitter* (Transmisor-receptor asíncrono universal)
LIN	*Local Interconnect Network* (Red de interconexión local)
IrDA SIR	*the Infrared Data Association, Serial InfraRed* (Asociación de datos infrarrojos, Infrarrojo en serie)

1. Introducción

Los sistemas embebidos se encuentran en gran variedad de dispositivos de uso común en la vida diaria y cada día se introducen en nuevos dispositivos para mejorar el desempeño y aumentar las funciones que ofrecen al usuario. Al aumentar la complejidad y variedad de las tareas a realizar por dichos dispositivos, se hace necesario organizar dichas tareas de tal manera que disminuya el tiempo de desarrollo, disminuya la complejidad, se facilite la programación y se permita la actualización del software de forma mas efectiva; esto ha hecho necesario el uso de sistemas que gestionen los recursos y faciliten la interacción entre el programador y el hardware, esto es posible con la introducción de sistemas operativos. En el caso de aplicaciones de tiempo real, es necesario utilizar un sistema operativo de tiempo real (RTOS).

Igualmente, se han desarrollado dispositivos para hacer frente a demandas de desempeño. La arquitectura ARM se ha desarrollado durante las últimas décadas, basada en el modelo de diseñar y licenciar la propiedad intelectual (IP Core) a diferencia del modelo de diseñar y producir circuitos integrados. Esto da una gran flexibilidad y permite generar una arquitectura común y flexible entre varios fabricantes de dispositivos.

1.1. Objetivo

1.1.1. Objetivo general

- Obtener un dominio sobre el sistema operativo de tiempo real µCOS que nos permita poder aplicarlo en diferentes áreas e implementar un detector de secuencias de eventos.

1.1.2. Objetivos Particulares

- Aprender los principios generales de operación de un sistema operativo de tiempo real.
- Obtener conocimientos sobre la arquitectura ARM y su modelo de programación que permitan la implementación de un sistema operativo de tiempo real en un dispositivo basado en dicha arquitectura.
- Desarrollar una aplicación basada en un sistema operativo de tiempo real y aplicarla en un sistema embebido con arquitectura ARM.

1.2. Aplicaciones

1.2.1. Sistemas de Tiempo Real

Los sistemas de tiempo real son aquellos en los que se requiere una respuesta en un tiempo limite o se corre el peligro de una falla en el sistema, estos se dividen de acuerdo a si esta falla se considera catastrófica o no; como se vera posteriormente.
Un sistema controlador de temperatura en un reactor nuclear que requiere una acción de control de temperatura dentro de un tiempo limitado es un sistema de tiempo real.
El rango de las aplicaciones que requieren un RTOS es muy grande y va desde los aparatos electrodomésticos hasta la industria aeroespacial.

1.2.2. Arquitectura ARM

Los procesadores ARM se encuentran en gran variedad de dispositivos como dispositivos de red, almacenamiento en disco duro y principalmente en dispositivos móviles en los que se requiere bajo consumo de potencia por ejemplo los teléfonos celulares. ARM se ha enfocado en dispositivos de alto volumen de producción y bajo costo, por lo que sus dispositivos difícilmente se encuentran en aplicaciones de muy alto desempeño y alto costo como computadoras personales móviles [1].

1.3. Organización de la tesina

El presente capítulo da una introducción a las necesidades que motivaron el tema de la tesina, muestra los objetivos generales y particulares, además de la organización de los capítulos. El capítulo 2 introduce conceptos de sistemas de tiempo real, sistemas operativos de tiempo real, además de una breve historia de la arquitectura ARM y los productos actuales basados en esta arquitectura. El capítulo 3 muestra las características del hardware y software usado, así como la descripción del sistema propuesto para resolver el problema planteado. El capítulo 4 muestra una descripción de los pasos seguidos para la configuración y programación del micro-controlador. El capítulo 5 muestra los resultados obtenidos así como las conclusiones. Ademán se presentan los materiales de referencia. El apéndice A muestra secciones del código generado.

2. Antecedentes Teóricos

2.1. Sistemas de Tiempo Real

2.1.1. Definición

Se dice que un sistema es de tiempo real si su correcta operación se determina no solamente de un funcionamiento correcto en su respuesta (respuesta correcta), sino también del tiempo de la misma (respuesta a tiempo). Esto es, los sistemas de tiempo real deben responder no solo correctamente pero también en un tiempo definido. “El tiempo entre la presentación de un conjunto de entradas a un sistema (estímulo) y la realización del comportamiento requerido (respuesta), [...] es llamado tiempo de respuesta de un sistema” [2]. Son en general sistemas críticos, por lo que el tiempo de respuesta debe estar garantizado.

2.1.2. Clasificación

Estos sistemas se pueden dividir de acuerdo a la tolerancia en el tiempo de su respuesta en:

Tiempo real estricto “Hard real time”: Todas las acciones deben de ocurrir dentro del plazo especificado. La acción después del tiempo definido se considera una falla, debido a que puede producir una falla crítica en el sistema completo.
Algunos ejemplos son los sistemas médicos como son marca-pasos, en los que se pone en peligro la vida del usuario en caso de una respuesta fuera de tiempo; Sistemas automotrices como son los frenos ABS en los que es necesaria la respuesta a tiempo para evitar colisiones; entre otros.

Tiempo real flexible “Soft real time”: En estos sistemas se tolera cierta latencia en la respuesta y se considera no crítica.
Un ejemplo es un sistema de vídeo en el que se puede permitir la omisión del despliegue de un cuadro en el vídeo sin influir en forma catastrófica en el sistema.

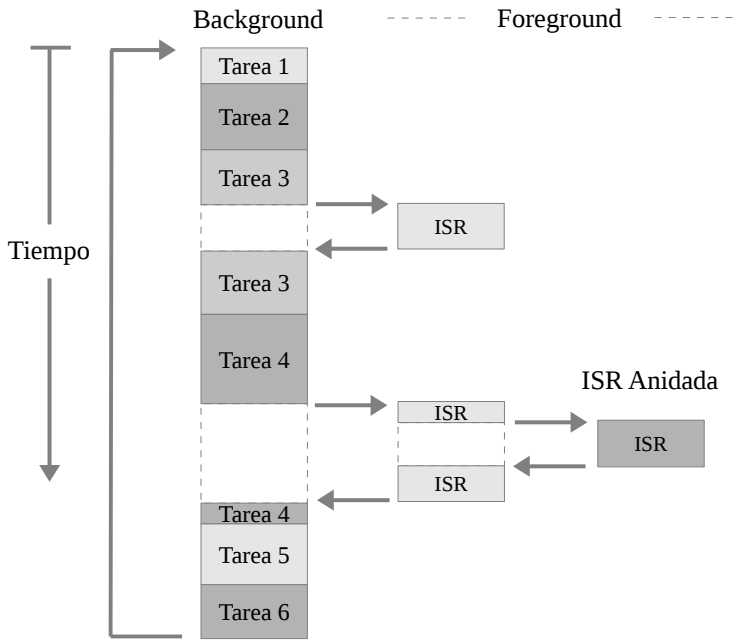

Figura 1. Sistemas Background/Foreground

Los sistemas de baja complejidad están diseñados tradicionalmente como sistemas "Background/Foreground", en los que la aplicación se diseña como un ciclo infinito, con rutinas que realizan la operación ("Background") y rutinas de interrupción que atienden eventos asíncronos ("Foreground"). En sistemas multitareas se hace uso de un sistema operativo o kernel, que es software que maneja el tiempo y recursos de un microprocesador, microcontrolador o procesador digital de señales [3].

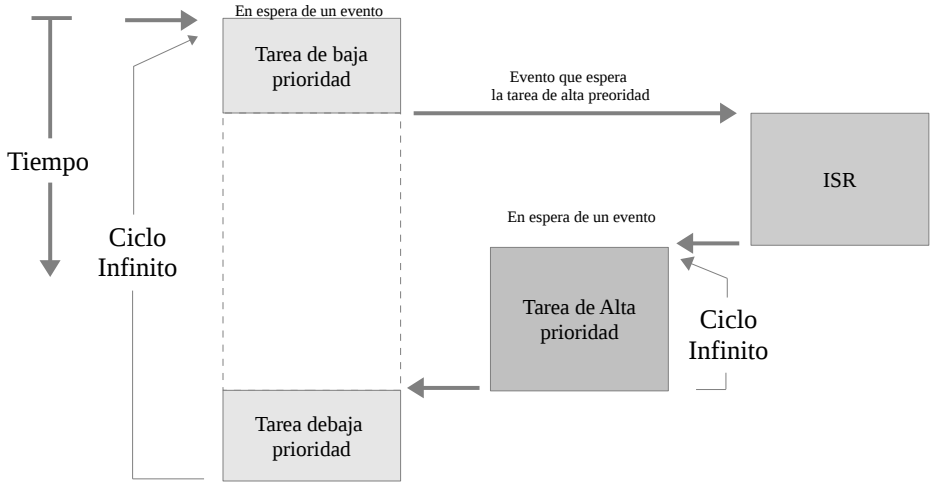

Figura 2. Sistemas de desalojo (preemtive)

El proceso de diseño de un sistema de tiempo real implica la separación del trabajo en tareas, cada una responsable de una porción del trabajo.

2.2. Sistemas Operativos de Tiempo Real

Los Sistemas Operativos de Tiempo Real RTOS (Real Time Operating System) deben proporcionar tres funciones específicas relacionadas con las tareas: planificación (scheduling); despacho (dispatching); intercomunicación y sincronización [2]. El planificador determina la siguiente tarea a ejecutar, mientras que el despachador realiza las operaciones necesarias para iniciar las tareas. La intercomunicación y sincronización entre tareas asegura que las tareas cooperen entre ellas. El objetivo principal del RTOS es satisfacer los requerimientos de tiempo real y ofrecer un ambiente de multitareas flexible y robusto.

2.3. µC/OS-III

Desarrollado por Jean J. Labrosse y publicado originalmente en un artículo de dos partes por la revista Embedded System Programming en 1992. Fue posteriormente publicado en forma de libro y comercializado. La tercera versión es desarrollada por Micrium Inc. y fue publicada en forma de libro en 2009 [3].

Entre las características más importantes del sistema operativo se encuentran:

- Tranportable a diferentes arquitecturas como ARM7/9, Cortex-MX, PowerPC, Microblaze, Coldfire, i.MX entre otros.
- Sistema operativo multitareas (preemptive).
- La respuesta a las interrupciones son predecibles (determinista)
- Número ilimitado de tareas. Solo limitado por los recursos de hardware.
- Número ilimitados de prioridades.
- Número ilimitado de objetos del kernel (semáforos, colas de mensajes, etc.)
- Diferentes tareas al mismo nivel de prioridad.
- Servicios avanzados: manejo de tareas, manejo de tiempos, semáforos, banderas de eventos, colas de mensajes, “mutexes”, timers de software entre otros.
- La mayor parte del código fuente está disponible en lenguaje ANSI C.
- Máximo tamaño en ROM (ROM footprint, unscaled) de 24 Kbytes y mínimo (scaled) de 6 Kbytes.
- Se acompaña de un libro de texto y manual de referencia.

2.4. Arquitectura ARM

2.4.1. Historia

ARM Holdings es una compañía inglesa basada en Cambridge, fue fundada en 1990 como un proyecto conjunto entre Acorn Computers, Apple Computer (Apple Inc.) y VLSI Technology [4]. A diferencia de otras compañía ARM Holdings solo licencia su tecnología (el núcleo de la CPU) como propiedad intelectual; esto es, no produce circuitos sino solo los diseña [5], dejando la producción a sus

licenciatarios como son Intel, Texas Instruments, Freescale o ST Microelectronics entre otros, quienes utilizan esta tecnología junto con tecnología propia (como son periféricos, tecnologías de fabricación, etc.) para producir procesadores o microcontroladores. De esta forma se produjeron en conjunto cerca de 2,900 millones de procesadores en 2007. [6]

La arquitectura ARM fue desarrollada por ARM Holdings y es una arquitectura RISC de 32 bits. Desde sus inicios, ARM ha introducido mejoras y nuevas características a su arquitectura en nuevas versiones de la misma.

La filosofía RISC es implementada con 4 reglas de diseño principales:

1. *Instrucciones*: Posee un número de instrucciones reducido que pueden ser ejecutadas en un ciclo. Cada instrucción es de un tamaño fijo para permitir traer ("fetch") futuras instrucciones antes de decodificar la instrucción actual.
2. *Pipelines*: El procesamiento de instrucciones es dividido en unidades menores para ser ejecutado en paralelo por pipelines.
3. *Registros*: Tiene un número grande de registros de propósito general. Cada registro puede contener datos o una dirección.
4. *Arquitectura load-store*: El procesador opera con datos en los registros. Operaciones separadas de load-store (carga-descarga) transfieren datos entre los registros y la memoria externa.

2.4.2. Familias ARM

La tabla 1 muestra las versiones de la arquitectura y las familias de dispositivos desarrolladas bajo esa versión de la arquitectura.

Arquitectura	Familia
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale
ARMv6	ARM11
ARMv7	Cortex

Tabla 1. Revisiones y familias de la arquitectura ARM

El nombre Cortex es usado a partir de la arquitectura versión 7 (ARMv7) y está dividido en 3 perfiles:

Perfil A (ARMv7-A): Diseñado para dispositivos de alto desempeño como teléfonos móviles, con aplicaciones complejas y sistemas operativos de alto nivel como son Linux y Symbian.

Perfil R (ARMv7-R): Diseñado para sistemas embebidos de alto desempeño con necesidades de tiempo real, como son Sistemas de frenado automotriz (ABS) y controladores de disco duro.

Perfil M (ARMv7-M): Diseñado para aplicaciones de bajo costo y bajo consumo de potencia como son sistemas de control industrial, incluyendo control en tiempo real [7].

Todos los dispositivos fabricados bajo la licencia de ARM (a partir del ARM7TDMI) soportan 2 conjuntos de instrucciones:

- Conjunto de instrucciones ARM. Instrucciones de longitud fija de 32 bits.
- Conjunto de instrucciones Thumb/Thumb-2. Instrucciones de 16 bits (Thumb) y 16 bits/32 bits (Thumb-2) que producen un código más compacto (Thumb es un subconjunto de Thumb-2) [7].

2.4.3. Cortex M3

El nucleo Cortex-M3 proporciona no solo el CPU, sino además el sistema de interrupciones, estructura de bus, sistema de depurado y mapa de memoria [8].

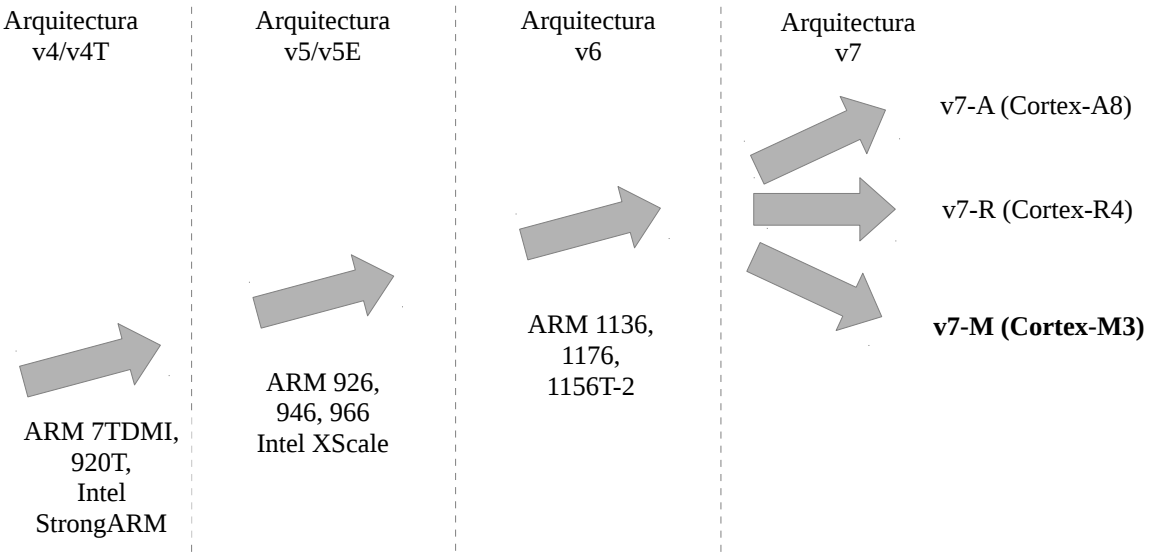

Figura 3. Revisiones recientes de la arquitectura ARM

Los microcontroladores basados en el núcleo Cortex M3 solo soportan el conjunto de instrucciones Thumb-2 que es una ampliación del Thumb para incluir instrucciones de 32 bit pero manteniendo una densidad de código similar [9].

Características:

- Pipeline de 3 etapas y “branch speculation” (predictor de saltos)
- 8 a 256 niveles de prioridad en las interrupciones [10].
- Arquitectura Harvard.
- Registros de 32 bits:
  - r0 a r12, 13 registros (32 bits) de propósito general
  - r13, Stack pointer (SP)
  - r14, Link Register (LR)
  - r15, Program Counter (PC)
  - Además de algunos registros de propósito específico (xPSR)

3. Descripción del Sistema

3.1. Tarjeta de desarrollo

El desarrollador del RTOS ha publicado libros de texto especializados [3] en los que se incluyen conceptos de RTOS en general así como una descripción de µC/OS-III y su API; además de la implementación del sistema operativo con un conjunto de aplicaciones de ejemplo para las tarjetas de desarrollo que se muestran en la tabla 2. La existencia de este conjunto de tarjetas de desarrollo para las cuales se cuenta con textos especializados, redujo las opciones disponibles.

Aun cuando se hubiera podido adaptar el RTOS a una plataforma diferente, esto se encuentra fuera de los objetivos de esta tesina por lo que se seleccionó la tarjeta de desarrollo dentro de este conjunto.

Tarjeta	Chip	Fabricante del chip	IDE
Keil MCB1700	LPC1768	NXP	Keil MDK Evaluation Version
YRDKRX62N	RX62N	Renesas	Renesas High-Performance Embedded Workshop (HEW)
YRDKSH7216	SH7216	Renesas	Renesas High-Performance Embedded Workshop (HEW)
µC/Eval-STM32F107	STM32F107	STMicroelecronics	IAR Systems Embedded Workbench
EVM-EVALBOT	LM3S9B92	Stellaris	IAR Systems Embedded Workbench

Tabla 2. Tarjetas de desarrollo disponibles al momento de la selección.

3.1.1. Criterios de selección

Al momento de la elección de la placa de desarrollo se tomó en cuanta el conjunto de tarjetas de desarrollo de la tabla 2 para la selección. Se evaluaron estas opciones en los siguientes criterios:

- Arquitectura del procesador.
- Periféricos disponibles.
- Precio.
- Disponibilidad.

La arquitectura es la misma en todas las opciones disponibles ARMv7 (microcontroladores basada en el núcleo Cortex-M3 pero de fabricantes distintos). El conjunto de periféricos no presenta diferencias considerables como para ser determinantes en la elección. Por lo que se optó por la tarjeta de desarrollo µC/Eval-STM32F107, principalmente por razón de disponibilidad y precio.

3.1.2. Características

La tarjeta cuenta con un microcontrolador de la familia STM32 Connectivity Line (STM32F107) fabricado por STMicroelectronics. La configuración genérica de un microcontrolador de la familia STM32 se presenta en la figura 4.

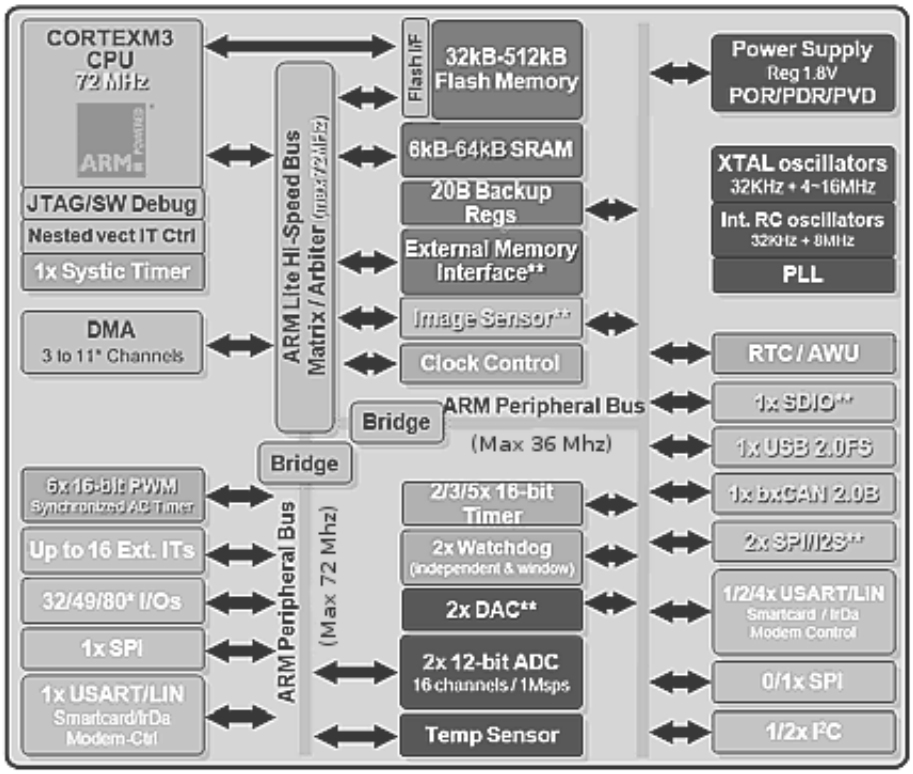

Figura 4. Microcontrolador de la familia STM32 Connectivity Line

Esta familia está enfocado a aplicaciones de tiempo real.

La tarjeta de desarrollo se muestra en la figura 5 y cuenta con las siguientes características:

- STM32F107 Cortex™-M3 a 72 MHz con:
  - 256 Kbytes de Flash
  - 64 Kbytes de SRAM
- 10/100 Mbps Conector de Ethernet
- USB-OTG
- RS-232C
- CAN
- SD/MMC
- STLM75 Sensor de temperatura
- 3 LEDs de usuario (rojo, amarillo y verde)
- Botón de reinicio
- Conector de E/S (pin headers)
- Área para prototipos
- J-Link SWD para depuración de código
- Energizado por el conector USB o fuente externa de 5V
- RoHS
- Cristal de 25 MHz para el microcontrolador.
  - Cristal de 32.768 MHz para el reloj de tiempo real (RTC).

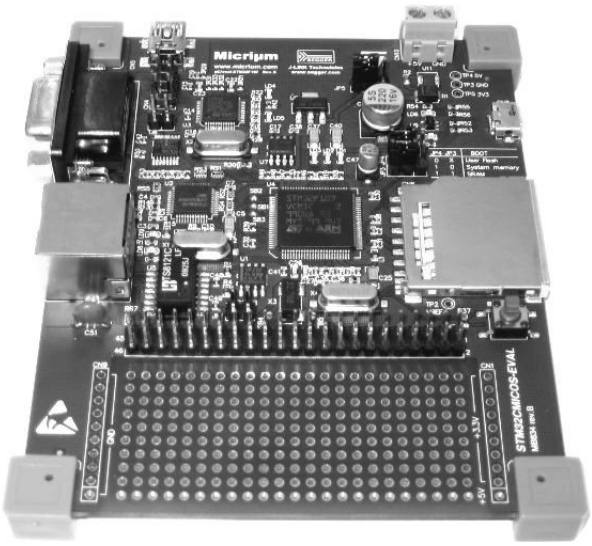

Figura 5. Tarjeta de desarrollo µC/Eval-STM32F107

3.2. Entorno de desarrollo Integrado (IDE)

Como IDE se utilizo IAR Embedded Workbench para ARM v5.4, desarrollado por IAR. La pagina de inicio del IDE se muestra en la figura 6.

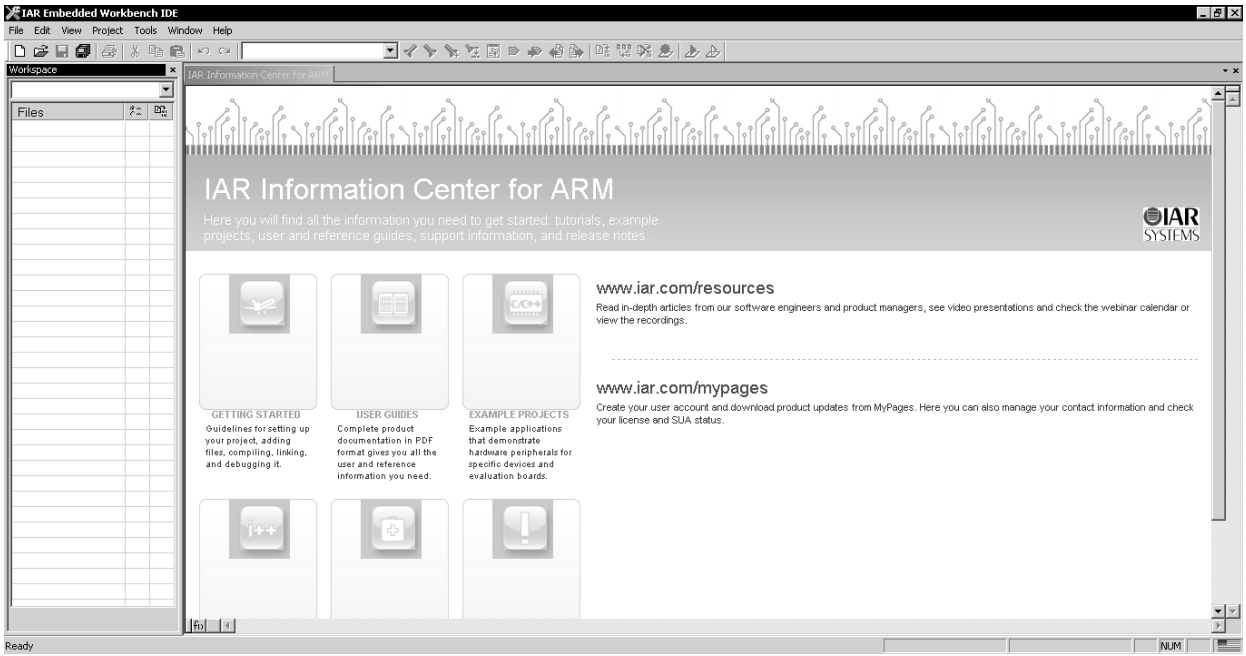

Figura 6. Pagina de inicio del entorno de desarrollo IAR Embedded Workbench para ARM v5.4

Los ejemplo que se incluyen con el RTOS fueron desarrollados en este IDE y se encuentran disponible para descargar en Internet. El IDE puede ser descargado en su versión Kickstart (32 kbyte de limite en el tamaño del código, sin soporte para MISRA C entre otras limitaciones) desde el sitio de IAR [11] o en su versión de 30 días.

3.2.1. Requerimientos

Como sistema huésped se requiere [12]:

- Windows 2000 (SP4) / Windows XP (SP2) / Windows Vista.
- Procesador Pentium o mejor.
- 1 GB de RAM mínimo.
- Hasta 1.5 GB de espacio en disco duro.
- Además de un puerto USB.

El equipo usado consta de una PC con procesador AMD Turion, 2GB de memoria, sistema operativo Windows XP (SP3) y disco duro de 160GB.

3.2.2. Características

Disponible para varias familias de microcontroladores como son PIC, AVR32, Cold Fire, 8051 entre otras.
La versión ARM brinda soporte para los núcleos:

- Cortex-A9
- Cortex-A8
- Cortex-R4(F)
- Cortex-M4
- Cortex-M3
- ARM11
- ARM10E
- ARM9E (ARM926EJ-S, ARM946E-S and ARM966E-S, ARM968E-S)
- ARM7E (ARM7EJ-S)
- Entre otros.

Entre sus características mas importantes se encuentran las siguientes:

- Ensamblador y compilador C/C++.
- Optimizaciones de código C/C++ para ARM.
- C-SPY Simulador y depurado en hardware.
- Múltiples proyectos en el mismo espacio de trabajo.
- Todas las bibliotecas requeridas para ISO/ANSI C y C++.
- Soporte para el “standard” ISO/ANSI C94.

3.3. Aplicación

La aplicación presenta una solución al problema de la captura de señales que arriban muy cercanas en el tiempo o simultaneas; donde el sistema tiene que atenderlas en tiempo real para evitar

lecturas erróneas. El diagrama a bloques se muestra en la figura 7.

3.3.1. Descripción

El sistema consta de un unidad de detección de secuencias de eventos basada en el microcontrolador STM32F107 de ST Electronics, usando una computadora como medio de despliegue de información. Además de un ejercitador de eventos basado en el mismo microcontrolador, que simulará la generación de los eventos, usando una computadora como interfaz de usuario donde se realizará la captura de mandos.
Para fines prácticos de nuestra aplicación se hacen las siguientes definiciones:

- Un evento es definido como un flanco de subida en un voltaje digital de 0 a 5V (0 a 1 digital).
- El ejercitador cuenta con 9 salidas digitales programables inicializadas a cero.
- La salida en la que se produzca un evento se mantendrá en 1 digital por la duración de la prueba o hasta que reinicie el sistema.

Los mandos ingresados por el usuario – los cuales contienen el evento y el tiempo de la ocurrencia - se integran en una plantilla de acuerdo a la cual se generarán los eventos correspondientes.

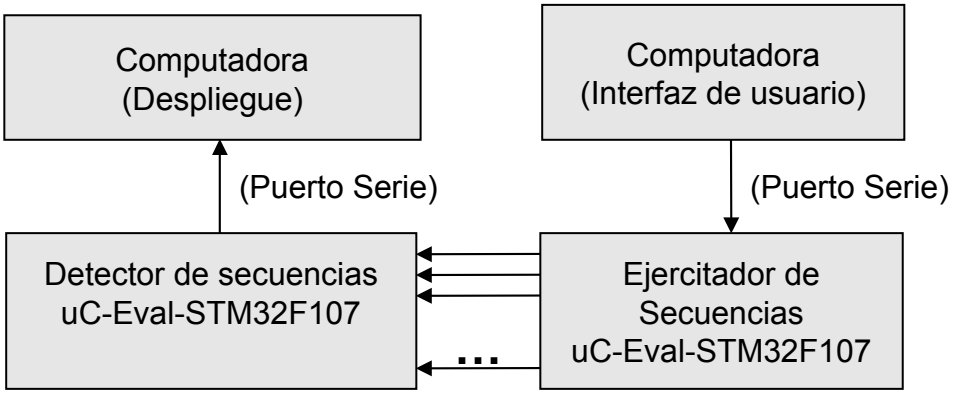

Figura 7. Diagrama a bloques de la aplicación

3.3.2. Detector

El detector consta de un sistema de captura que depende del sistema operativo y su latencia en la respuesta a las interrupciones. Pero además hace uso de timers para capturar el evento. Su diagrama a bloques se presenta en la figura 8.

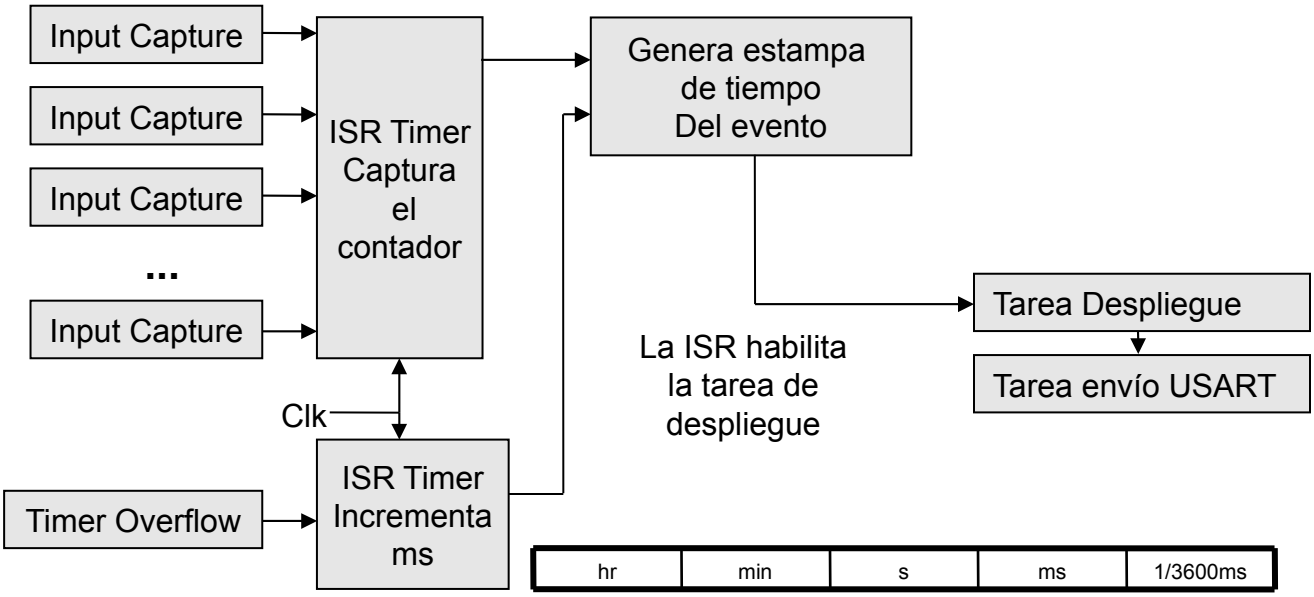

Figura 8. Diagrama a bloques del detector de secuencias

3.3.3. Ejercitador

El ejercitador fue desarrollado en el modelo de servidor-cliente en el que un cliente (PC) envía mandos a un servidor que los interpreta y ejecuta.

3.3.3.1. Servidor

El servidor en la tarjeta de desarrollo interpreta las órdenes enviadas por el cliente en la PC y a su vez ejecuta dichas órdenes colocando puertos de entrada/salida (GPIO) del microcontrolador en su estado requerido.
Almacena los mandos en un arreglo; El cual se puede interpretar como una mascara del estado del puerto de salida y el retardo entre el evento actual y el próximo evento (si este sucede).
Este arreglo es leído al producir los eventos.

3.3.3.2. Cliente

Se utilizó Visual Studio 2008 como entorno de desarrollo y se generó un esquema de comunicación a través del puerto serial. Sus características se presentan a continuación.

Los mandos posibles son de la siguiente forma y se ingresan en modo consola:

- **sec A B**
  Guarda una secuencia de señales definidas por A con un retardo B ms entre las señales definidas y además define el retardo de la primera señal de la secuencia A con respecto a la última definida previamente a la secuencia A. A puede ser un conjunto de señales del 1 al 9 y B es un número mayor que 0 que define el retardo en ms.

  Por ejemplo: sec 2468 100
  Guarda una secuencia de flancos de subida en las señales 2,4,6 y 8 con un retardo de 100 ms

relativo a la señal anterior.

- **sim A B**
  Guarda un conjunto de señales simultaneas en el tiempo, definidas por A con un retardo de B ms con respecto a la señal anterior. A puede ser un conjunto de señales del 0 al 8 y B es un número que define el retardo en ms.

  Por ejemplo: sim 1357 500
  Produce un flanco de subida en forma simultanea en las señales 1,3,5 y 7 con un retardo de 500ms con respecto a la señal anterior.
  Un caso especial se produce para la primera señal, en la que se produce una espera desde el punto en que se inició la cuenta de tiempo sin haber una señal definida en tal momento.

- **Enable**
  Genera las señales definidas previamente usando los comandos sec y sim. La recepción de esta orden en el servidor es considerada el punto de tiempo 0ms para la generación de las señales.

- **reset**
  Borra todas las señales guardadas en memoria.

- **open**
  Abre el puerto serie para comunicación. Por omisión (default) se intenta la comunicación usando el puerto COM1, en caso de que el puerto a usar tenga otro nombre se utiliza este comando. No genera transmisión serial, sólo abre el puerto.

  Por ejemplo: open COM2 Abre el puerto COM2, un mensaje de error se muestra en caso de error al realizar el mando.

Cada mando se codifica en una serie de caracteres ASCII de longitud variable para ser transmitidos al servidor. Con las siguientes características:

3.3.3.3. Esquema de codificación de mensajes.

El mando es codificado antes de ser enviado al servidor. El mensaje codificado se genera de la siguiente forma:

- Un carácter inicial: Codifica el mando a ejecutar por el servidor y puede ser uno de los siguientes:
  - **S** Guarda señales secuenciales (sec).
  - **P** Guarda señales paralelas (sim).
  - **E** Genera las señales definidas previamente (enable).
  - **R** Borra todas las señales guardadas (reset).

- Una serie de caracteres numéricos ASCII de longitud variable que simbolizan las señales a generar (1,2,3,4,5,6,7,8 y 9).

- El carácter “**:**” (dos puntos), usado para delimitar las serie anterior y la cadena siguiente.

- Una cadena de caracteres ASCII de longitud variable con el valor del retardo entre las señales en ms como se vió en la descripción del ejercitador de señales.

- Un carácter “**;**” como marcador de final de la transmisión del mando a ejecutar.

Por ejemplo, la captura del siguiente mando genera el mensaje correspondiente.

Mando	Cadena transmitida
sec 1234 400	S1234:400;
sim 56 1000	P56:1000;
enable	E:;
reset	R:;

Estos mensajes son interpretados por el servidor y ejecutados respectivamente. Otro ejemplo y su diagrama de tiempos se presentan en la sección 4.4.2.

4. Configuración y Programación

4.1. Sistema Operativo

El sistema operativo cuenta con una estructura de archivos definida en el capitulo 1 del texto[3], en donde se hace diferencia entre las siguientes categorías de archivos, su función y localización:

- Código de aplicación.
  (\Micrium\Software\EvalBoards\Micrium\uC-Eval-STM32F107\IAR\<proyecto>)
  Es el código de la aplicación generado por el usuario. El programa principal es generalmente llamado app.c. Puede además incluir archivos de cabecera escritos por el usuario..

- **CPU**
  (\Micrium\Software\CPU\<manufacturer>\<architecture>\)
  No presente en nuestra configuración.

- **Board Support Package (BSP)**
  (\Micrium\Software\EvalBoards\Micrium\uC-Eval-STM32F107\IAR\BSP\)
  Bibliotecas para manejo del harware de la tarjeta, comúnmente es inherente al microcontrolador o tarjeta de desarrollo. Por ejemplo funciones para prender LEDs presentes en la tarjeta o configuración de periféricos.

- **uC/OS-III, Código fuente independiente del CPU**
  (\Micrium\Software\uCOS-III\Cfg\Templates, \Micrium\Software\uCOS-III\Source)
  Los archivos en estos directorios están disponibles a los usuarios con licencia.

- **uC/OS-III, Código fuente específico del CPU**
  (\Micrium\Software\uCOS-III\Ports\<architecture >\<compiler>)
  Localidad de los archivos de la transportación a una CPU específica.

- **uC/CPU, Código fuente específico del CPU**
  (\Micrium\Software\uC-CPU, \Micrium\Software\uC-CPU\<architecture >\<compiler>)
  uC/CPU consiste de archivos que encapsulan funciones especificas del CPU y tipos de datos específicos del compilador.

- **uC/LIB, Biblioteca de funciones transportables**
  (\Micrium\Software\uC-LIB, \Micrium\Software\uC-LIB\Ports\<architecture >\<compiler>)
  uC/CPU consiste de una biblioteca de funciones altamente transportables.

La biblioteca de funciones de configuración y manejo de los periféricos del microcontrolador en lenguaje C se encuentran incluidas en el BSP pero se pueden obtener o actualizar con código proporcionado por el fabricante del microcontrolador (SMT). Junto con ayuda y ejemplos de uso (STM32F10x_StdPeriph_Lib_V3.3.0 [10]).

El código del sistema operativo está disponible para descarga del sitio de Micrium, el cual incluye el

código como se describió en el apartado anterior (Micrium-Book-uCOS-III-STM32F107.exe).
Los proyectos de ejemplos se desarrollaron en IAR Embedded Workbench para ARM. La configuración del OS se realiza llamando funciones dentro de la API del OS.

4.2. Descripción de los Periféricos Usados

La descripción de los periféricos usados se presenta a continuación. El uso y configuración es diferente en el detector y el ejercitador y se presenta en la sección 4.3.

4.2.1. Puertos de Entrada/Salida

Puertos de entrada/salida de 16 bits. La familia SMT32 cuenta con 5 puertos de entrada salida de propósito general (A hasta E), pero no todos están disponibles en forma completa en la tarjeta de desarrollo o son utilizados para otros propósitos (conexión externa para periféricos).

4.2.1.1. Características

Cada pin del puerto puede ser configurado individualmente en diferentes modos:
Como Entrada:

- Modo analógico
- Entrada flotante
- Entrada con pull-up / pull-down
- Reservada

Como Salida (2MHz, 10MHz y 50MHz velocidad máxima):

- Salida de propósito general push-pull
- Salida de propósito general Open-drain
- Salida de propósito alternativo Push-pull
- Salida de propósito alternativo Open-drain

4.2.1.2. Descripción funcional

La Descripción funcional de los puertos de entrada/salida se presenta en la figura 9.

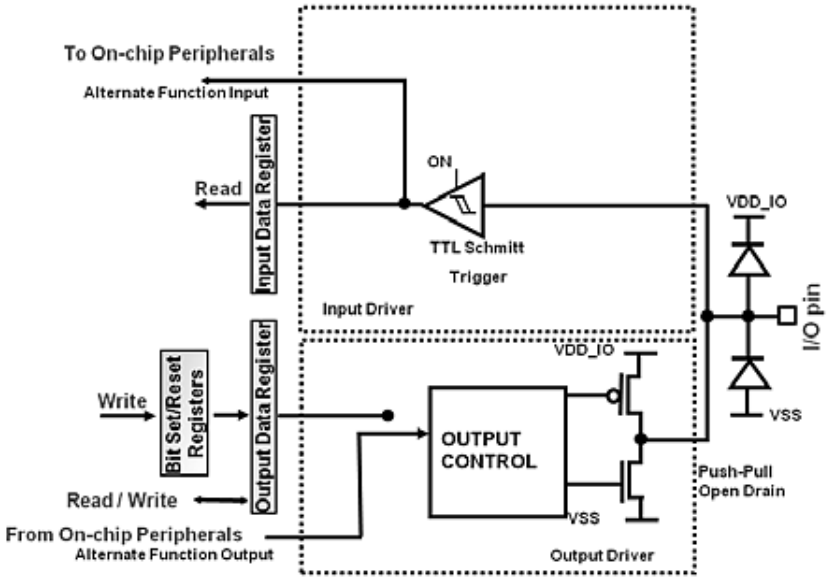

Figura 9. Entrada/salida de propósito general

4.2.1.3. Registros

Cada puerto de entrada/salida de uso general (GPIOx donde x es desde A hasta E) cuenta con:

- Dos registros de configuración de 32 bits (GPIOx_CRL, GPIOx_CRH).
  Definen el modo de uso del pin. (entrada o salida, modo análogo, etc.)

- Dos registros de datos de 32 bits (GPIOx_IDR, GPIOx_ODR).
  - PIOx_IDR, contienen el estado lógico de los pins usados como entrada y es solo lectura.
  - GPIOx_ODR, contiene el estado de los pins usados como salidas y se puede leer y escribir.

- Un registro set/reset de 32 bits (GPIOx_BSRR)
  Usado para hacer set/reset a los pins en forma atómica.

- Un registro para reset de 16 bits (GPIOx_BRR)
  Igual que el anterior.

- Un registro lock de 32 bits (GPIOx_LCKR)
  Usado para evitar la modificación subsecuente de los bits del puerto hasta el próximo reset.

Todos los puertos tienen capacidad de interrupción externa; para ser usado como interrupción externa el puerto debe de ser configurado como entrada y además configurar el modo de interrupción externa EXTI.

Para optimizar el número de periféricos disponibles, es posible hace un “remapeo” de algunas funciones o otro pin. Esto se hace por software al configuran los registros correspondientes (AFIO).

4.2.2. Timers

La Familia de microcontroladores STM32 (Connectivity line) cuenta con 2 tipos de timers:

- General-purpose timers (TIM2 a TIM5). Timers de propósito general.
- Advanced-control timers (TIM1 y TIM8) Cuentan con las mismas características generales de los timers de propósito general pero además cuentan con características especiales.

4.2.2.1. Características generales.

Entre las características principales de los timers (TIM2 a TIM5 y TIM1 y TIM8), se encuentran las siguientes:

- Contador de 16 bits. Ascendente, descendente, ascendente-descendente
- Preescalador de 16 bits. Permite dividir la frecuencia del contador en cualquier factor de 1 a 65535.
- Hasta 4 canales independientes para:
  - Input-capture
  - Output-compare
  - PWM (Edge and Center-aligned Mode)
  - One-pulse mode output
- Salidas complementarias con tiempo muerto programable.
- Circuito de sincronización para controlar el timer con señales externas e interconectar varios timers.
- Contador de repetición para actualizar los registros del timer solo después de un número de ciclos del contador.
- Generación de interrupción o DMA para los siguientes eventos:
  - Sobreflujo
  - Evento de disparo
  - Input Capture
  - Output Compare

4.2.2.2. Características especificas.

Los timers TIM2 al TIM5, se encuentran conectados al bus APB1 (vea el apartado 4.2.4.1. para mas información sobre los buses) con velocidad máxima de 36Mhz.
El timer 1 (TIM1) y el timer 8 (TIM8), se encuentran conectados al bus APB2 con velocidad máxima de 72Mhz.
Los timers 1 y 8 son timers de control avanzado y pueden ser usados para medir el ancho de pulso de una señal de entrada, generar ondas de salida (output compare, PWM, complementary PWM with dead-time insertion) y son independientes de los timers de propósito general.
Los timer 1 y 8 comparten las características antes mencionadas de los timers de propósito general pero además cuenta con características especiales, entre las cuales se encuentran las siguientes:

- Salidas complementarias con tiempo muerto programable.
- Circuito de sincronización para controlar el timer con señales externas e interconectar varios timers.
- Contador de repetición para actualizar los registros del timer solo después de un número de ciclos del contador.

4.2.2.3. Descripción funcional

La descripción funcional de los puertos de entrada/salida se presenta en la figura 10.

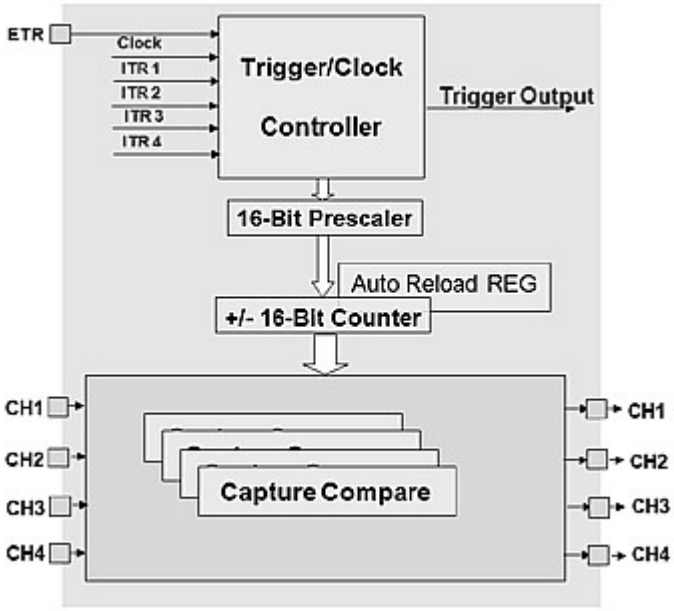

Figura 10. Timer genérico de la familia STM32

4.2.2.4. Registros

Los timers de propósito general cuentan con los siguientes registros.

- TIMx_CR1, Registro de control 1. Permite la habilitación del timer y configuración del modo de conteo (ascendente/descendente, etc.)
- TIMx_CR2, Registro de control 2. Configuración del origen del disparos para sincronización de timers esclavos, entre otras.
- TIMx_SMCR, Registro de control de modo esclavo. Configuración de la fuente del disparo externo, polaridad y selección de modo de esclavo.
- TIMx_DIER, Registro de habilitación de DMA/Interrupción. Habilitación de las fuentes de DMA.
- TIMx_SR, Registro de estatus. Banderas de sobre captura, interrupción por captura, disparo y sobre flujo.
- TIMx_EGR, Registro de generación de eventos. Configura la generación de interrupciones o DMA en los eventos de disparo, captura o sobre flujo.
- TIMx_CCMR1, Registro de modo “capture/compare” 1. Varias configuraciones de los canales 1 y 2 como “input capture”/”output compare”.
- TIMx_CCMR2, Registro de modo “capture/compare” 2. Varias configuraciones de los canales 3 y 4 como “input capture”/”output compare”.
- TIMx_CCER, Registro de habilitación de “capture/compare”. Habilitación de los 4 canales del timer y configuración de la polaridad en la captura.
- TIMx_CNT, Registro de contador. Almacena el valor del contador.
- TIMx_PSC, Registro del “prescaler”. Contiene el valor que será cargado en el prescaler en cada

sobre flujo.

- TIMx_ARR, Registro de “auto-reload” Contiene el valor que será cargado en el registro de auto-reload real.
- TIMx_RCR, Registro de contador de repetición.
- TIMx_CCR1, Registro de “capture/compare” 1.
- TIMx_CCR2, Registro de “capture/compare” 2.
- TIMx_CCR3, Registro de “capture/compare” 3.
- TIMx_CCR4, Registro de “capture/compare” 4.
- TIMx_BDTR, Registro de paro y tiempo muerto.
- TIMx_DCR, Registro de control DMA.
- TIMx_DMAR, Dirección DMA para transferencia completa.

4.2.3. Puerto serie

El Transmisor-Receptor asíncrono universal (synchronous asynchronous receiver transmitter, USART) ofrece una forma sencilla de intercambio de información full-duplex con equipo externo. Ofrece una gran variedad de velocidades en baudios usando un generador fraccional de frecuencia de baudios (fractional baud rate generator).

4.2.3.1. Características

- Full Duplex, comunicación asíncrona.
- Formato “standard” NRZ (No return to Zero)
- Generador fraccional de frecuencias de baudios (hasta 4.5 Mbits/s).
- Ancho de palabra programable (8 ó 9 bits).
- Bits de alto configurables (1 ó 2 bits de alto)
- Soporta el “standard” LIN.
- Salida de reloj del transmisor para comunicación síncrona.
- Codificador-decodificador IrDA SIR.
- Comunicación Half-duplex de un sólo cable.
- Comunicación multibuffer configurable usando DMA.
- Bits de habilitación separados para transmisión y recepción.
- Banderas de detección de envío:
  - Buffer de recepción lleno.
  - Buffer de transmisión lleno.
  - Fin de transmisión.
- Control de paridad.
- Cuatro banderas de detección de error:
  - Error de sobreflujo
  - Error de ruido.
  - Error de trama.
  - Error de paridad.
- Diez fuentes de interrupción con banderas.

4.2.3.2. Descripción funcional.

El periférico esta conectado externamente con otro dispositivo por 3 pins. Cualquier comunicación USART bidireccional requiere un mínimo de dos pins: Recepción de datos (RX) y transmisión de datos (TX). Otros pins son necesarios para otros modos como son:

- Modo síncrono:
  - Salida de reloj del transmisor (SCLK),
- Modo IrDA::
  - Entrada de datos IrDA (IrDA_RDI)
  - Salida de transmisión de datos (IrDA_TDO)
- Modo de control de flujo por hardware:
  - Clear to send, en alto impide la transmisión de datos después de la trasmisión actual (nCTS)
  - Request to send, en bajo indica que está listo para recibir datos (nRTS).

La trama se compone de lo siguiente:

- Bit de inicio.
- Una palabra de datos (8 ó 9 bits) bit menos significativo primero.
- 0.5, 1, 1.5, 2 bits de paro indicando que la trama está completa.
- Esta interfaz usa un generador fraccionaria de velocidades de baudios. Con 12 bits de mantisa y 4 bits de fracción.

4.2.3.3. Registros.

- Un Registro de estado (USART_SR)
- Registro de Datos (USART_DR)
- Registro de velocidad de baudios (USART_BRR), 12 bits de mantisa y 4 bits de fracción.
- 3 registros de control (USART_RCx).
- Un registro de GuardTime en caso usar modo Smartcard.

4.2.4. Otros

4.2.4.1. APB1, APB2

Los periféricos en la familia STM32 se encuentran conectados en 2 “buses avanzados de periféricos ARM” (ARM Advanced Peripheral Busses). Denominados APB1 y APB2. Cada bus APB se conecta en un puente a la matriz del bus principal AHB (ARM Advanced High Speed Bus).

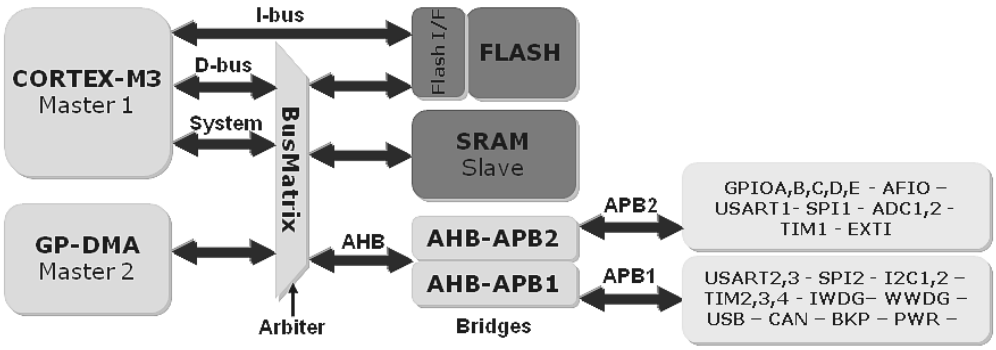

Figura 11. Estructura interna de buses del microcontrolador STM32

El bus APB2 puede operar a 72MHz pero el APB1 está limitado a 36MHz. Es importante notar esto debido a que dependiendo del bus será la velocidad de operación del periférico. Esta estructura se muestra en la figura 11.

4.2.4.2. Controlador de Interrupción (NVIC).

El controlador de interrupciones anidadas es un periférico del núcleo ARM. Para el procesador que estamos utilizando:

- Soporta hasta 82 canales de interrupciónes enmascarables (depende del dispositivo)
- 16 niveles de prioridad. Nivel 0 es la mayor prioridad en la interrupción.
- Detección de nivel y pulso en las señales de interrupción.
- Cambio de prioridad dinámico.
- Encadenado de interrupciones (tail-chaining).

4.3. Configuración de periféricos

La configuración de los periféricos se realizó en código C, usando funciones y macros definidas en el BSP. Una lista completa así como todas las macros definidas se pueden encontrar en el archivo de ayuda de la biblioteca “standard” de la familia STM32F10x [12].

4.3.1. Detector

El código del programa en el detector consiste de un programa principal (app.c), así como un archivo de funciones (app_user.h) donde se encuentran las funciones usadas en la aplicación.

4.3.1.1. Sistema Operativo

La configuración e inicialización del sistema operativo se realizó haciendo llamadas a funciones propias del sistema operativo dentro del programa principal (app.c).

- `void OSInit (OS_ERR *p_err)`: Inicializa el sistema operativo y debe ser llamada antes de llamar cualquier otra función.
- `void OSStart (OS_ERR *p_err)`: Inicia el modo multitareas. Entrega el control al sistema operativo para ejecutar la tarea con mas prioridad definida antes del llamado de la función, no regresa a la función que la llama. Debe ser llamada después de `OSInit ()`.

El fabricante del sistema operativo recomienda crear una sola tarea antes de iniciar el modo multitareas. Lo que permite determinar qué tan rápido es el procesador y calcular el porcentaje del CPU usado en la ejecución [3]. Otras tareas pueden ser creadas dentro de esta tarea.

Además se llama a las siguientes funciones :

- `BSP_Init ()`: Realiza una inicialización de la tarjeta de desarrollo y fija los relojes a partir del reloj externo.

  HSE = 25MHz (Reloj externo)
  PLL3=(HSE/5)*10
  PLL2= (HSE/5)*8
  PLL1 = (PLL2 / 5) * 9 = 72Mhz
  HCLK = SYSCLK = PLL1 = 72MHz (Reloj del sistema)
  AHBCLK= HCLK = 72MHz (Reloj del Bus principal)
  APB2CLK = AHBCLK = 72 Mhz. (Reloj para GPIOx, TIM1)
  APB1CLK = AHBCLK/2 = 36 Mhz. (Reloj para USARTx, TIMx)

- `CPU_Init ()`: Realiza una inicialización del procesador.

4.3.1.2. Puertos de Entrada/Salida

Antes de utilizar los periféricos es necesario habilitar el reloj para cada periférico, esto se realiza en la función de usuario `Clock_Config ()`.
El reloj se habilita para los siguientes periféricos

- APB1: TIM2, TIM3, TIM4
- APB2: GPIOB, GPIOC, GPIOD, GPIOE, TIM1, AFIO, USART2

Los puertos usados se configuraron de acuerdo a la Tabla 3, usando la funciona `GPIO_Init ()`, dentro de la configuración de los timers.
La configuración se realizó llenando los valares deseados en una estructura del tipo `GPIO_InitTypeDef` y pasando esta estructura a la función `GPIO_Init ()`.
Los campos de la estructura `GPIO_InitTypeDef` son los siguientes:

- `GPIO_Pin`: Pin del puerto a configurar.
  - `GPIO_Pin_x`: 1 al 15
  - `GPIO_Pin_ALL`: Todos los pins.
- `GPIO_Mode`: Modo de uso.
  - En modo entrada:
    - `GPIO_Mode_AIN`: Modo analógico
    - `GPIO_Mode_IN_FLOATING`: Entrada flotante
    - `GPIO_Mode_IPD`: Entrada con pull-down
    - `GPIO_Mode_IPU`: Entrada con pull-up
    - Reservada
  - En modo salida:
    - `GPIO_Mode_Out_PP`: Salida de propósito general push-pull
    - `GPIO_Mode_Out_OD`: Salida de propósito general Open-drain
    - `GPIO_Mode_AF_PP`: Salida de propósito alternativo Push-pull
    - `GPIO_Mode_AF_OD`: Salida de propósito alternativo Open-drain

- `GPIO_Speed`: Velocidad de salida máxima
  - `GPIO_Speed_10MHz`: 10MHz
  - `GPIO_Speed_2MHz`: 2MHz
  - `GPIO_Speed_50MHz`: 50MHz

Pin	Función	I/O	Velocidad
GPIOE9	TIM1 canal 1	Entrada Pull-Down	50 Mhz
GPIOE11	TIM1 canal 2	Entrada Pull-Down	50 Mhz
GPIOE13	TIM1 canal 3	Entrada Pull-Down	50 Mhz
GPIOE14	TIM1 canal 4	Entrada Pull-Down	50 Mhz
GPIOC6	TIM3 canal 1	Entrada Pull-Down	50 Mhz
GPIOC7	TIM3 canal 2	Entrada Pull-Down	50 Mhz
GPIOC8	TIM3 canal 3	Entrada Pull-Down	50 Mhz
GPIOC9	TIM3 canal 4	Entrada Pull-Down	50 Mhz
GPIOB9	TIM4 canal 4	Entrada Pull-Down	50 Mhz
GPIOD5	USART2 TX	Salida Push-Pull	-
GPIOD6	USART2 RX	Entrada Input Floating	2 Mhz

Tabla 3. Configuración de puertos de entrada salida en el detector.

4.3.1.3. Timers

Se utilizaron los timers TIM1, TIM3 en sus cuatros canales y el canal 4 del TIM4.
La configuración general de timer se realiza usando una estructura `TIM_TimeBaseInitTypeDef`.
Los campos de la estructura `TIM_TimeBaseInitTypeDef` son los siguientes:

- `TIM_ClockDivision`: Define el divisor entre la frecuencia del reloj del timer (CK_INT) y el reloj de muestreo usado por los filtros digitales (ETR, TIx).
  - `TIM_CKD_DIVx`: 1,2, o 4.
- `TIM_CounterMode`: Define el modo de conteo. Alineado al centro (El timer cuenta desde 0 hasta el valor de auto recarga y de regreso a 0), hacia abajo o hacia arriba.
  - `TIM_CounterMode_CenterAligned1`: El contador inicia del valor de auto recarga hacia abajo.
  - `TIM_CounterMode_CenterAligned2`: El contador inicia de 0 al valor de auto recarga.
  - `TIM_CounterMode_CenterAligned3`: El contador cuenta hacia arriba y hacia abajo.
  - `TIM_CounterMode_Down`: El contador cuenta hacia abajo.
  - `TIM_CounterMode_Up`: El contador cuenta hacia arriba.
- `TIM_Period`: Define el valor para ser cargado en el registro de auto recarga. Entero entre 0x0000 y 0xFFFF.
- `TIM_Prescaler`: Especifica el valor al que se dividirá el valor del reloj. Entero entre 0x0000 y 0xFFFF.
- `TIM_RepetitionCounter`: Especifica el valor del contador de repeticiones. Cuando este valor se decrementa hasta cero se produce un evento y se reinicia el contador. Sólo válido para el timer 1. valor entero entre 0x00 y 0xFF.

Se utilizó un divisor de frecuencia de 1, conteo ascendente, periodo de 36000 y preescalador de 1 para los timers 2, 3 y 4. Un divisor de frecuencia de 2, conteo ascendente, periodo de 36000 y preescalador

de 0 para el timer 1.

Los parámetros anteriores generan los siguientes tiempos:

APB2CLK = 72 Mhz. (Reloj para TIM1)
APB1CLK = 36 Mhz. (Reloj para TIM2, TIM3, TIM4)

Se hizo un remapeo completo de los pins (TIM1, TIM2 y TIM3) para tener acceso a ellos desde el conector CN3 en la tarjeta de desarrollo (`GPIO_PinRemapConfig ()`).

Se usó el modo de maestro esclavo (`TIM_SelectSlaveMode ()`) para sincronizar el contador de los timers como se muestra en la figura 12.

Se configuró TIM2 para generar un disparo en cada sobreflujo (TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update)). A su vez se configuraron los timers TIM1, TIM3 y TIM4 para recibir entrada de disparo de TIM_TS_ITR1, que corresponde al disparo desde TIM2 (TIM_SelectInputTrigger(TIMx, TIM_TS_ITR1)).

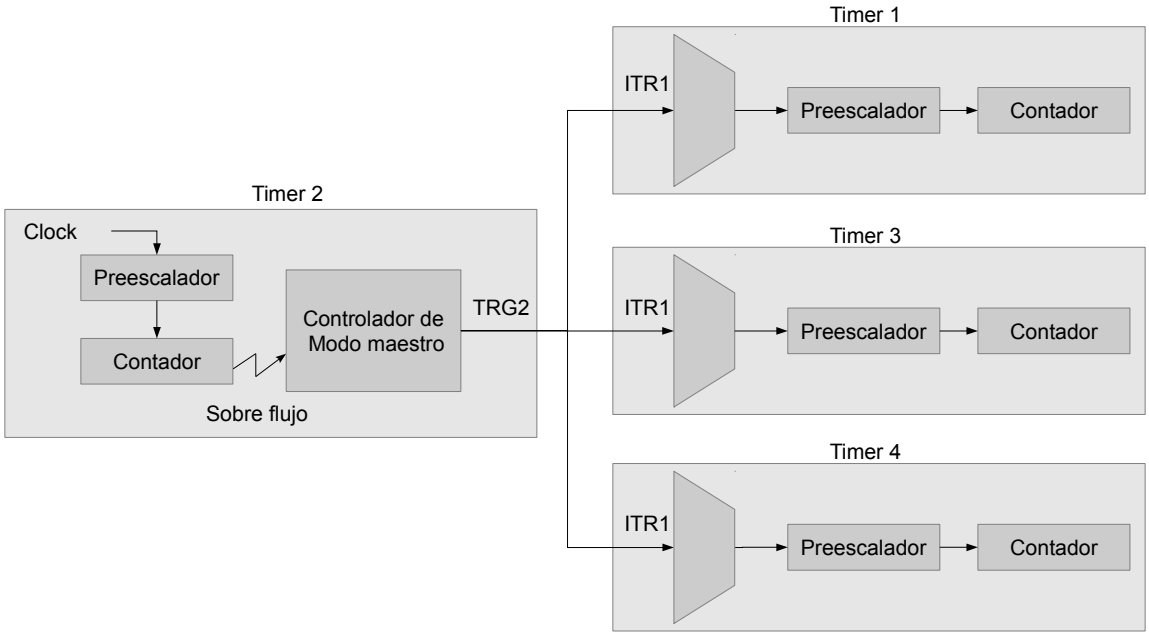

Figura 12. Configuración maestro-esclavo de los timers

La configuración de cada canal individual como Input Capture se realizó llenando los valores deseados en una estructura `TIM_ICInitTypeDef` y pasando esta estructura a la función `TIM_ICInit ()`. Los campos relevantes de la estructura son los siguientes:

- `TIM_Channel`: Canal del timer a configurar.
  - `TIM_Channel_x`: 1 al 4.
- `TIM_ICFilter`: Número de ciclos necesario de la señal de entrada para ser considerada como una transición válida. (entero entre 0x0 y 0xF)

- `TIM_ICPolarity`: Polaridad de la transición a capturar.
  - `TIM_ICPolarity_Rising`: Polaridad de subida.
  - `TIM_ICPolarity_Falling`: Polaridad de bajada.
- `TIM_ICPrescaler`: Número de transiciones válidas para generar una captura (generar un evento). Con esto se puede generar un captura cada transición, cada 2, 4 u 8 transiciones válidas
- `TIM_ICPSC_DIVx`: 1,2,4 u 8.
- `TIM_ICSelection`: Captura directa (canal 1 con IC1, etc), indirecta (canal 1 con IC2, canal 2 con IC1, etc) o conectada a TRC (de un timer esclavo).
  - `TIM_ICSelection_DirectTI`: Captura directa (canal 1 con IC1, etc).
  - `TIM_ICSelection_IndirectTI`: indirecta (canal 1 con IC2, canal 2 con IC1, etc).
  - `TIM_ICSelection_TRC`: Conectada a TRC (de un timer esclavo).

Las entradas se configuraron en modo input-capture con flanco de subida, un filtro de 5 ciclos, una captura por cada flanco y captura directa.

Se habilitó además la interrupción por sobre flujo (`TIM_IT_CCx`) en los canales usados usando la función `TIM_ITConfig ()`.

4.3.1.4. Puerto serie

El puerto serie es usado para el envío de mensajes hacia una PC en la que se visualizan los tiempos de las eventos registrados. Los pins se configuran de acuerdo a la tabla 3.
Se realiza la configuración del reloj con valores por “default” usando una estructura `USART_ClockInitTypeDef`.

La configuración se realiza en una estructura `USART_InitTypeDef`. Los campos modificados son los siguientes:

- `USART_BaudRate`: Define la velocidad de transmisión en baudios.
- `USART_WordLength`: Especifica el número de bits de datos transmitidos o recibidos en una paquete. Este valor puede ser 8 ó 9 bits.
  - `USART_WordLength_8b`: 8 bits en paquete.
  - `USART_WordLength_9b`: 9 bits en paquete.
- `USART_StopBits`: Especifica el número de bits de paro transmitidos. Este valor puede se 1 ó 2.
  - `USART_StopBits_1`: 1 bit de paro.
  - `USART_StopBits_2`: 2 bits de paro.
- `USART_Parity`: Especifica el modo de paridad. Cuando la paridad se encuentra habilitada, esta es insertada como el bit mas significativo (MSB).
  - `USART_Parity_Even`: El bit de paridad es calculado para obtener un número par de unos.
  - `USART_Parity_No`: No se genera bit de paridad.
  - `USART_Parity_Odd`: EL bit de paridad es calculado para obtener un número impar de unos.
- `USART_HardwareFlowControl`: Establece si el control de flujo por hardware se encuentra habilitado.
  - `USART_HardwareFlowControl_CTS`: CTS es usado.
  - `USART_HardwareFlowControl_None`: No se usa control de flujo.
  - `USART_HardwareFlowControl_RTS`: RTS es usado

    - `USART_HardwareFlowControl_RTS_CTS`: RTS/CTS son usados.
- `USART_Mode`: Especifica si la trasmisión o recepción se encuentran habilitadas.
    - `USART_Mode_Rx`: La transmisión está habilitada.
    - `USART_Mode_Tx`: La recepción esta habilitada.

Se realiza un remapeo para tener disponible los pins del USART2 en el conector de la tarjeta de desarrollo (`GPIO_Remap_USART2`).

4.3.1.5. Controlador de Interrupción (NVIC)

Los periféricos configurados para producir interrupción son TIM1, TIM3 y TIM4.
Para eso se uso una estructura del tipo `NVIC_InitTypeDef`. Que cuenta con los siguientes campos:

- `NVIC_IRQChannel`: Especifica el canal de IRQ a habilitar o deshabilitar.
- `NVIC_IRQChannelCmd`: Especifica si el canal IRQ esta habilita o deshabilitado.
    - `ENABLE`: Habilitado.
    - `DISABLE`: Deshabilitado.
- `NVIC_IRQChannelPreemptionPriority`: Especifica la prioridad de desalojo del canal y es un valor entre 0 y 15; donde 0 es la prioridad mas alta.
- `NVIC_IRQChannelSubPriority`: Especifica la sub-prioridad del canal y es un valor entre 0 y 15.

Se habilitó la interrupción global para TIM3 y TIM4 (`BSP_INT_ID_TIM3` y `BSP_INT_ID_TIM4`) y la de captura para TIM1 (`BSP_INT_ID_TIM1_CC`). En el caso de TIM3 y TIM4 solo se cuenta con una interrupción general, por lo que se hizo uso de la bandera de captura para verificar que la interrupción se generó por una captura ya dentro de la rutina de interrupción. Se les asigno la mayor prioridad de desalojo (0) y una sub-prioridad de 1 a los 3 times por igual.

Dentro del programa principal se le asignó una rutina de interrupción para cada canal IRQ habilitado.
Por ejemplo, el siguiente llamado:

```
BSP_IntVectSet (BSP_INT_ID_TIM3, TIM3_isr);
```

Asigna la función `TIM3_isr ()` al canal `BSP_INT_ID_TIM3` (Interrupción general del canal 3).

4.3.2. Ejercitador

El código del programa en el ejercitador consiste de un programa principal (app.c), así como un archivo de funciones (app_user.h) donde se encuentran las funciones usadas en la aplicación. La configuración se realizó en gran medida en la misma forma que en el detector, se presentan las puntos en los que se difiere.

4.3.2.1. Sistema Operativo

La configuración del sistema operativo en el ejercitador se realizó de la misma forma que en el detector.

4.3.2.2. Puertos de Entrada/Salida

El reloj se habilita para los siguientes periféricos
APB1:

GPIOE, TIM1, AFIO, USART2

Los puertos usados y su configuración se muestran en la tabla 4.

Pin	Función	I/O	Velocidad
GPIOE15	Salida 9	Salida Pull-Down	50 Mhz
GPIOE14	Salida 8	Salida Pull-Down	50 Mhz
GPIOE13	Salida 7	Salida Pull-Down	50 Mhz
GPIOE12	Salida 6	Salida Pull-Down	50 Mhz
GPIOE11	Salida 5	Salida Pull-Down	50 Mhz
GPIOE10	Salida 4	Salida Pull-Down	50 Mhz
GPIOE9	Salida 3	Salida Pull-Down	50 Mhz
GPIOE8	Salida 2	Salida Pull-Down	50 Mhz
GPIOE7	Salida 1	Salida Pull-Down	50 Mhz
GPIOD5	USART2 TX	Salida Push-Pull	-
GPIOD6	USART2 RX	Entrada Input Floating	2 Mhz

Tabla 4. Configuración de puertos de entrada salida en el ejercitador

4.3.2.3. Timers

Sólo se utilizó TIM1 como contador de tiempo, generando una interrupción en sobre flujo (`BSP_INT_ID_TIM1_UP`). Se utilizó un divisor de frecuencia de 1, conteo ascendente, periodo de 9000 y preescalador de 1. No se realizo “remapeo” de pins al no tener salidas, ni se utilizó modo maestro esclavo.

4.3.2.4. Puerto serie

El puerto serie se configuró en la misma forma que en detector excepto que se utilizó principalmente como entrada y genera una interrupción al recibir un dato (`USART2_IRQChannel`).

4.3.2.5. Controlador de Interrupción (NVIC)

Los periféricos configurados para producir interrupción son TIM1 y USART2.
TIM1: Interrupción en sobre flujo. Prioridad 0 (la mas alta).
USART2: Interrupción global, se verificó la bandera de Rx para realizar la rutina sólo en recepción de dato. Prioridad 1.

4.4. Programación de la Aplicación

4.4.1. Detector

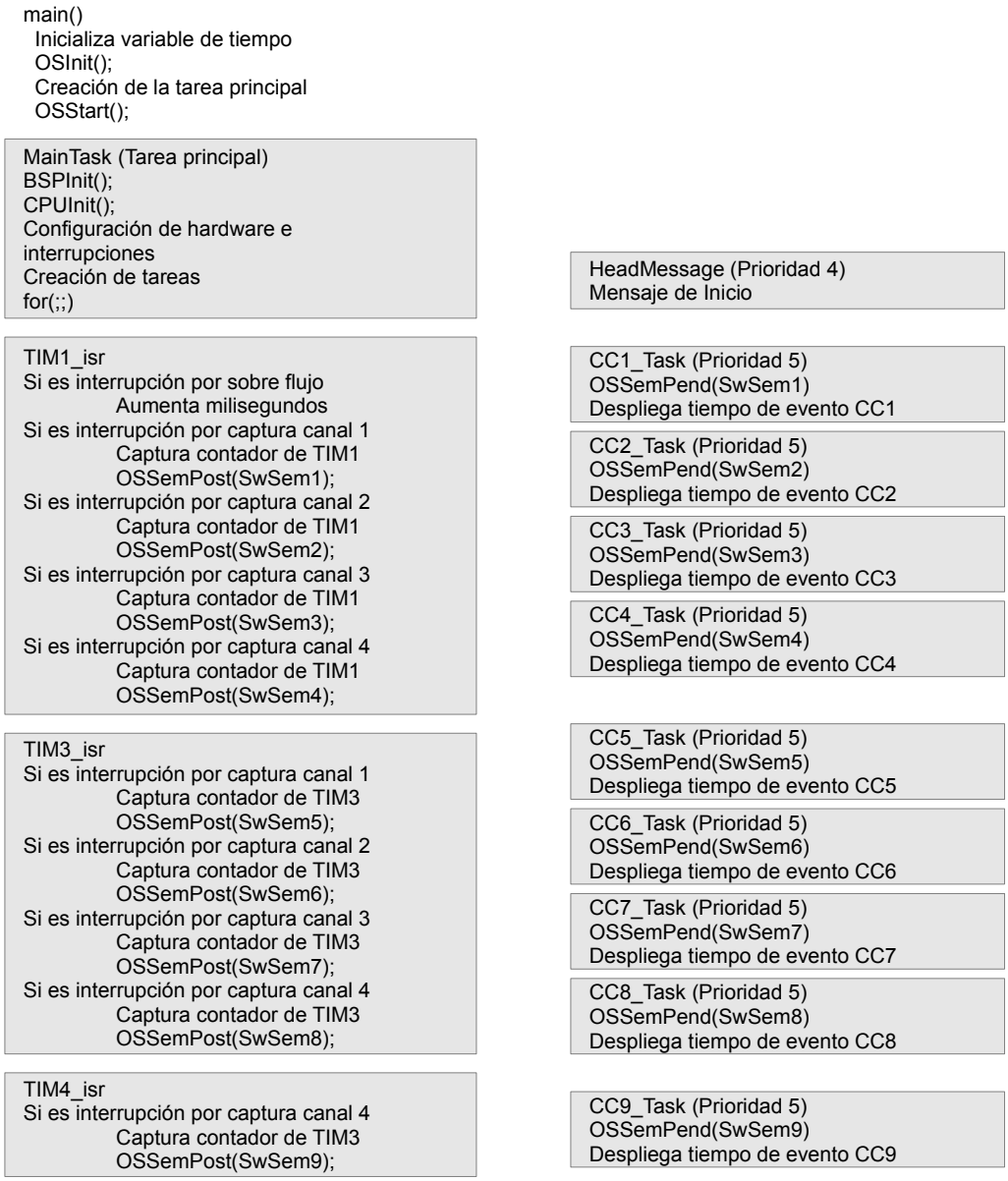

Figura 13. Diagrama de tareas en el detector

El programa principal (main()) se encarga de inicializar el OS y generar una tarea (MainTask). Una vez iniciado el OS se ejecuta esta única tarea. En esta tarea se generan las demás tareas necesarias.

Se utilizan estructuras del tipo USR_Time_TypeDef para almacenar la estampa de tiempo del último evento generado en cada canal.

Los campos de esta estructura son los siguientes:

- CPU_INT16U _useg: Fracción de milisegundo, es capturada del contador del timer. Es un numero de 0 a 36000 por lo que indica 1/36000 fracciones de ms.
- CPU_INT64U _mseg: milisegundos
- CPU_INT16U _seg: Segundos
- CPU_INT16U _min: Minutos
- CPU_INT16U _hrs: Horas
- CPU_INT16U _dys: Días

Los valores de los días, horas, minutos, segundos y milisegundos son calculados a partir de contador general de milisegundos (usando la función Conv_mseg ( )), que indica los milisegundos desde que se inicio el sistema. Este valor se reinicia con cada reset.

Las tareas generadas son las siguientes:

- *MainTask*: Tarea principal, se inicializa la tarjeta de desarrollo y el CPU. Además se generan las demás tareas. Se suspende al terminar la inicialización.
- *HeadMessage*: Tarea de mayor prioridad (4), manda un mensaje de inicio y se suspende.
- *CC1_Task*: Tarea de prioridad menor (5). Envía mensajes sobre las capturas en el canal 1. Espera al semáforo SwSem1, que indica que se realizó una captura en el canal; lee la estampa de tiempo de la estructura Captura1 y envía el mensaje a despliegue.
- *CC2_Task*: Realiza la misma función en el canal 2 que CC1_Task realiza en el canal 1, tiene la misma prioridad. Usando el semáforo SwSem2 y la estructura Captura2.
- *CC3_Task*: Tarea de envío de mensajes sobre las capturas en el canal 3.
- *CC4_Task*: Tarea de envío de mensajes sobre las capturas en el canal 4.
- *CC5_Task*: Tarea de envío de mensajes sobre las capturas en el canal 5.
- *CC6_Task*: Tarea de envío de mensajes sobre las capturas en el canal 6.
- *CC7_Task*: Tarea de envío de mensajes sobre las capturas en el canal 7.
- *CC8_Task*: Tarea de envío de mensajes sobre las capturas en el canal 8.
- *CC9_Task*: Tarea de envío de mensajes sobre las capturas en el canal 9.

Todas las tareas de despliegue de mensajes tienen la misma prioridad.

Las rutinas de interrupción se encargan de incrementar el contador de milisegundos (TIM3_isr es usada para este propósito) y capturar el valor del registro TIMx_CNT (fracción de milisegundo) al realizarse un evento para calcular la estampa de tiempo del mismo evento (TIMx_isr, donde x es 1,3 o 4). El diagrama de las tareas y las rutinas de interrupción se presenta en la figura 13.

4.4.2. Ejercitador

4.4.2.1. Servidor

El programa principal cumple la misma función que en el detector. Al igual que la tarea generada (MainTask), sin embargo no es necesario generar tareas adicionales. Las rutinas de interrupción se encargan de realizar el trabajo necesario en el ejercitador.
Se utiliza un arreglo de estructuras del tipo USR_Signal_TypeDef, sus campos son los siguientes:

- CPU_INT16U _Mask: Mascara para ser escrita al puerto de salida, el bit 7 en 1 representa un evento en la señal 1, el bit 8 en 1 representa un evento en la señal 2 y así sucesivamente hasta el bit 15 en 1 representando un evento en la señal 9.

- CPU_INT64U _Delay: Representa el retardo entre el tiempo del evento o los eventos definidos y el próximo evento. La rutina de interrupción del USART2 se encarga de almacenar los datos recibidos, decodificar y ejecutar los mandos al recibir un mensaje completo. El esquema de codificación se muestra en el apartado 3.3.3.3. Un ejemplo de los mandos capturados, las cadenas transmitidas y la forma en la que se llena la estructura se presenta en la figura 14, donde *k* es el número de eventos validos en el arreglo.

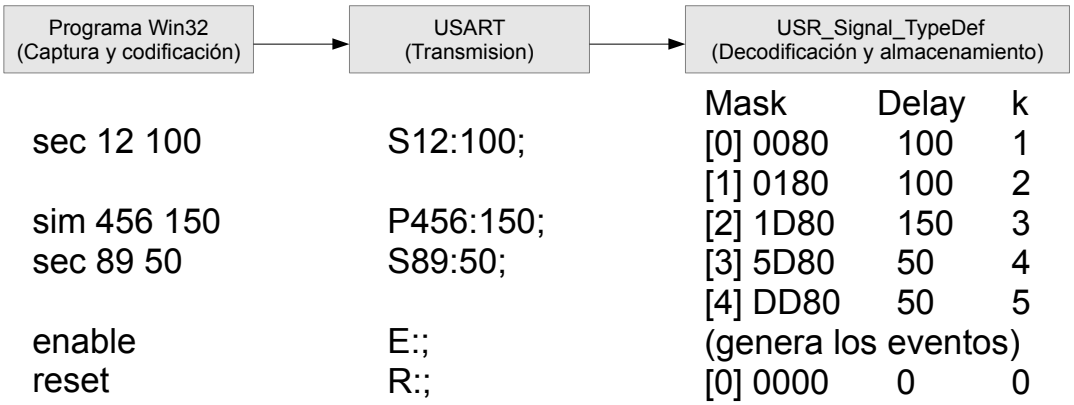

Figura 14. Esquema de captura, trasmisión y almacenamiento de mandos

La rutina de interrupción del TIM1 recorre el arreglo de señales, generando los eventos y sus respectivos retardos en el mando enable. El diagrama de tiempo de eventos generados se presenta en la figura 15.

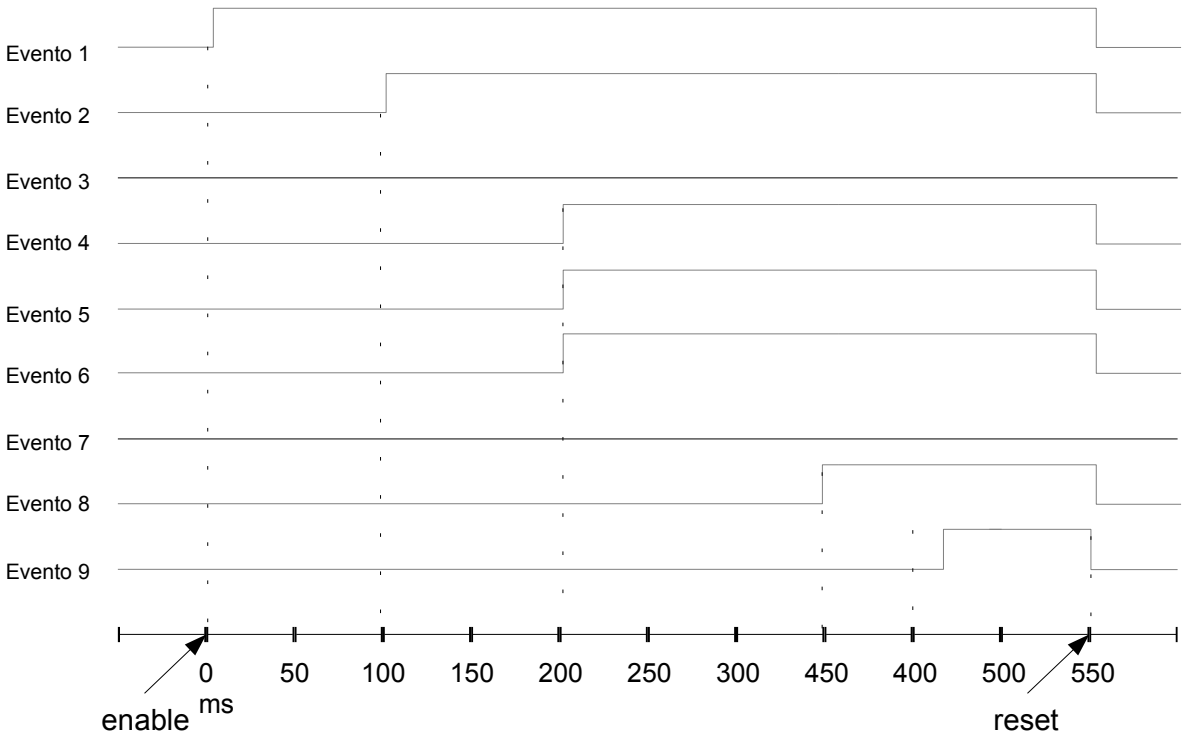

Figura 15. Diagrama de tiempo de eventos generados

4.4.2.2. Cliente

El programa cliente es un programa para windows que se encarga de la captura, codificación de los mandos y envío de las cadenas de caracteres codificadas.
El programa consiste de un ciclo infinito de captura de mandos. Los mandos permitidos y el esquema de codificación se muestran en la sección 3.3.3.2 y 3.3.3.3 respectivamente. El programa termina al recibir el mando “exit”.

Resultados y conclusiones

Se cumplió el objetivo planteado al inicio del desarrollo de tesina. Esto es, se obtuvo un conocimiento de la arquitectura ARM y de sistemas operativos de tiempo real. Además se dio uso a varios periféricos del microcontrolador para capturar eventos simultáneos, lo cual no sería posible hacer usando sólo el OS e interrupciones.

La implementación del RTOS se realizó sin complicaciones mayores al contar con ejemplos que se usaron como plantilla. Además de contar con la documentación proporcionada por los fabricantes del microcontrolador (STMicroelectronics) y del RTOS (Micrium), así como el libro de texto que incluye el manual de la API y las ayuda y ejemplos de uso del firmware.

El sistema operativo fue utilizado para la aplicación. Se generaron tareas y se usaron facilidades del OS como son semáforos. La programación se realizó en lenguaje de alto nivel, pero se mantuvo una visión del hardware y sus configuración de bajo nivel (en sus registros). El paquete de funciones (firmware) desarrollado por el fabricante del microcontrolador proporciona una forma fácil y rápida para acceder al hardware en lenguaje de alto nivel.

Además de la detección de eventos se implementó un sistema bidireccional de comunicación entre una PC y un sistema embebido usando una consola de mandos para generar los eventos. La aplicación se limitó a generar eventos y se reciben los mensajes sobre la detección del evento en una terminal. Esto se puede implementar en sistemas de monitoreo en los que el sistema requiere de acciones del usuario. Con lo que se convierte en una forma práctica de comunicación en sistemas en los que por razones de espacio o costo, no es posible contar con unidades de despliegue o interacción con el usuario, pero se cuenta con una PC que se puede conectar y actuar como unidad de despliegue y envío de mandos para diagnóstico o configuración.

Referencias

[1] Sloss, Andrew N, et al., ARM System Development’s Guide, 2004
[2] Laplante, Phillip A, Real-time systems design and analysis : an engineer’s handbook, 2004
[3] Labrosse, Jean J., uc/OS-III THe Real-Time Kernel, 2010
[4] ARM Limited. Sep, 2010, http://www.arm.com/about/company-profile/milestones.php
[5] ARM Limited. Sep, 2010, http://www.arm.com/about/company-profile/index.php
[6] ARM Limited. Sep, 2010, http://www.arm.com/about/newsroom/19813.php
[7] Yiu, Joseph, The Definitive Guide to the ARM Cortex-M3, 2007
[8] Hitex (UK) Ltd, The Insider's Guide to the STM32 ARM based Microcontroller, 2009
[9] ARM Limited, Cortex M3 Revision: r1p1 Technical Reference Manual, 2007
[10] ARM Limited. Sep, 2010, http://www.arm.com/products/processors/cortex-m/cortex-m3.php
[11] IAR Systems. Ago, 2010, http://supp.iar.com/Download/SW/?item=EWARM-KS32
[12] STMicroelectronics, STM32F10x Standard Peripherals Firmware Library v3.5.0 , 2011

Apéndice A

La captura de los eventos se realiza en la rutina de interrupción de TIM1, TIM3 y TIM4. La estampa de tiempo se genera usando el valor de los milisegundos de la variable `Time._mseg` (actualizada en el sobreflujo de TIM3) y el valor del registro de captura. El código para la captura en los canales de TIM3 se muestra a continuación.

```
static void TIM3_isr (void)
{
  /* Si la interrupción se genero por Sobreflujo */
  if (TIM_GetFlagStatus (TIM3 , TIM_FLAG_Update))
  {
    /* Actualiza el contador de ms (solo TIM3)*/
    Time._mseg++;
    TIM_ClearFlag (TIM3 , TIM_FLAG_Update);
  }
  /* Si la interrupción se genero por Captura en CH1 */
  if (TIM_GetFlagStatus (TIM3 , TIM_FLAG_CC1))
  {
    Captura5._mseg = Time._mseg;
    Captura5._useg = TIM_GetCapture1 (TIM3);
    TIM_ClearFlag (TIM3 , TIM_FLAG_CC1);
    OSSemPost (&SwSem5, OS_OPT_POST_ALL, &err);
  }
  /* Si la interrupción se genero por Captura en CH2 */
  if (TIM_GetFlagStatus (TIM3 , TIM_FLAG_CC2))
  {
    Captura6._mseg = Time._mseg;
    Captura6._useg = TIM_GetCapture2 (TIM3);
    TIM_ClearFlag (TIM3 , TIM_FLAG_CC2);
    OSSemPost (&SwSem6, OS_OPT_POST_ALL, &err);
  }
  /* Si la interrupción se genero por Captura en CH3 */
  if (TIM_GetFlagStatus (TIM3 , TIM_FLAG_CC3))
  {
    Captura7._mseg = Time._mseg;
    Captura7._useg = TIM_GetCapture3 (TIM3);
    TIM_ClearFlag (TIM3 , TIM_FLAG_CC3);
    OSSemPost (&SwSem7, OS_OPT_POST_ALL, &err);
  }
  /* Si la interrupción se genero por Captura en CH4 */
  if (TIM_GetFlagStatus (TIM3 , TIM_FLAG_CC4))
  {
    Captura8._mseg = Time._mseg;
    Captura8._useg = TIM_GetCapture4 (TIM3);
    TIM_ClearFlag (TIM3 , TIM_FLAG_CC4);
    OSSemPost (&SwSem8, OS_OPT_POST_ALL, &err);
  }
}
```

TIM2 es usado como maestro de TIM1, TIM3 y TIM4. Su configuración se muestra a continuación.

```
CPU_VOID TIM2_Config (void)
{
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

  TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseStructure.TIM_Period = 1;
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseInit (TIM2, &TIM_TimeBaseStructure);

  TIM_SelectMasterSlaveMode (TIM2, TIM_MasterSlaveMode_Enable);

  TIM_SelectOutputTrigger (TIM2, TIM_TRGOSource_Update);
}
```

A continuación se presenta el ejemplo de configuración de un timer (TIM3), los demás timers (TIM1 y TIM4) se configuran en manera similar.

```
CPU_VOID TIM3_Config (void)
{
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

  TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseStructure.TIM_Period = 36000;
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseInit (TIM3, &TIM_TimeBaseStructure);

  GPIO_PinRemapConfig (GPIO_FullRemap_TIM3, ENABLE);

  TIM_ITConfig (TIM3 , TIM_IT_Update , ENABLE);

  TIM_SelectSlaveMode (TIM3, TIM_SlaveMode_Trigger);
  TIM_SelectInputTrigger (TIM3, TIM_TS_ITR1);
}
```

La rutina de configuración de un canal de captura se muestra a continuación, se utiliza como ejemplo TIM3 en su canal 1, la configuración es similar en los demás canales de captura (Canal 1, 2, 3 y 4 del TIM1; canal 2, 3 y 4 del TIM3; y canal 4 del TIM4).

```
CPU_VOID TIM3_CH1_Config (void)
{
  TIM_ICInitTypeDef TIM_ICInitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;

  TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
  TIM_ICInitStructure.TIM_ICFilter = 1;
  TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
  TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
  TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
  TIM_ICInit (TIM3 , &TIM_ICInitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init (GPIOC, &GPIO_InitStructure);

  TIM_ITConfig (TIM3 , TIM_IT_CC1 , ENABLE);
}
```