



## Desarrollo de aplicaciones en LPCXpresso basadas en RTOS

Alan Kharsansky    Mario Azcueta

4 - Marzo - 2011

## Section 1

### Introducción LPCXpresso

# Introducción

El LPCXpresso es un toolchain completo para evaluación y desarrollo con microcontroladores de NXP.

Esta compuesto por:

- LPCXpresso IDE y "development tools"
  - IDE basado en Eclipse
  - Compiler y linker GNU
  - GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional (opcional)



# Introducción

El LPCXpresso es un toolchain completo para evaluación y desarrollo con microcontroladores de NXP.

Esta compuesto por:

- LPCXpresso IDE y "development tools"
  - IDE basado en Eclipse
  - Compiler y linker GNU
  - GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional (opcional)



# Introducción

El LPCXpresso es un toolchain completo para evaluación y desarrollo con microcontroladores de NXP.

Esta compuesto por:

- LPCXpresso IDE y "development tools"
  - IDE basado en Eclipse
  - Compiler y linker GNU
  - GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional (opcional)



# Introducción

El LPCXpresso es un toolchain completo para evaluación y desarrollo con microcontroladores de NXP.

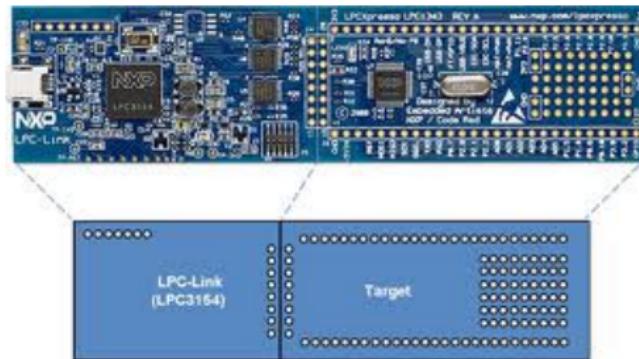
Esta compuesto por:

- LPCXpresso IDE y "development tools"
  - IDE basado en Eclipse
  - Compiler y linker GNU
  - GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional (opcional)

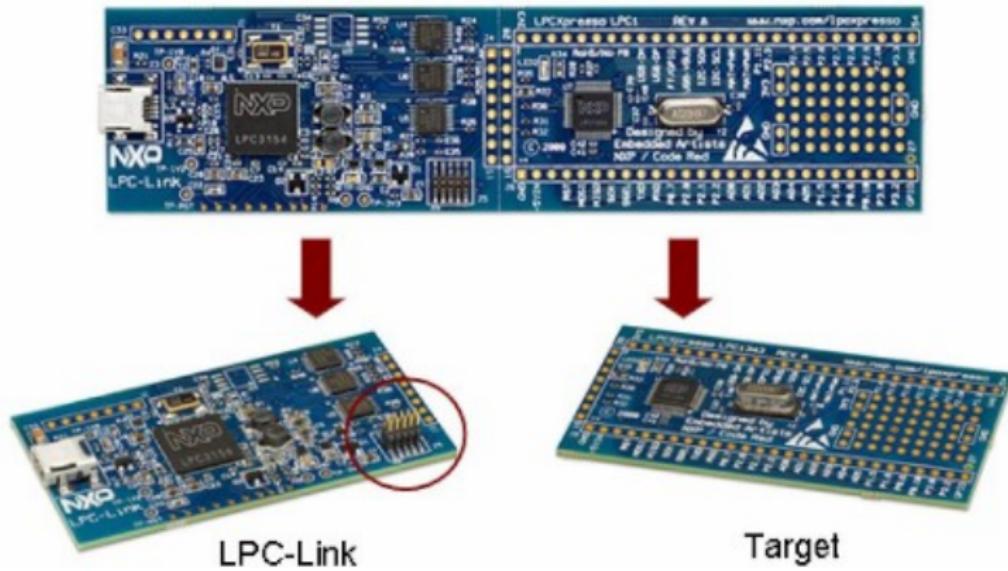


# LPCXpresso target board

El target board es un microcontrolador con todo lo necesario para encender y también una herramienta que incluye un programador y debugger.



# LPCXpresso target board



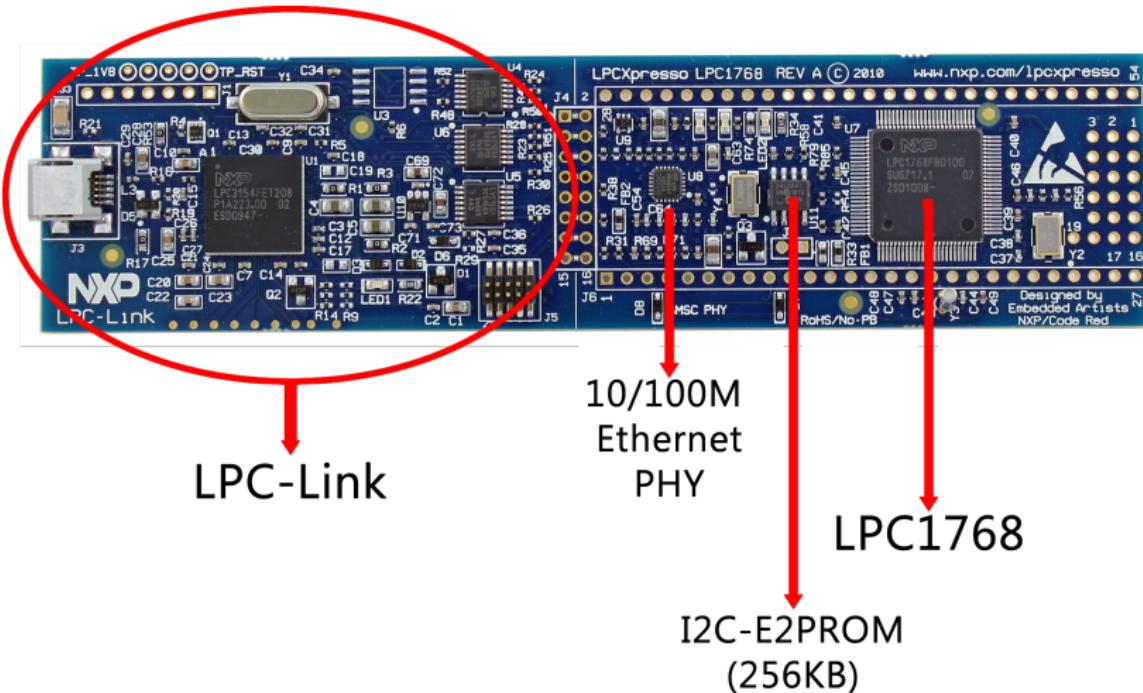
# LPCXpresso target board

Del lado del target este incluye algunos periféricos básicos y se comercializan con diferentes microcontroladores. Por el momento se encuentran disponibles:

- **LPC1114:** ARM Cortex-M0, 32KB flash, 4/8KB SRAM
- **LPC1343:** ARM Cortex-M3, 32KB flash, 8K SRAM, USB
- **LPC1768:** ARM Cortex-M3, 512KB flash, 64KB SRAM, Ethernet, USB On the go.

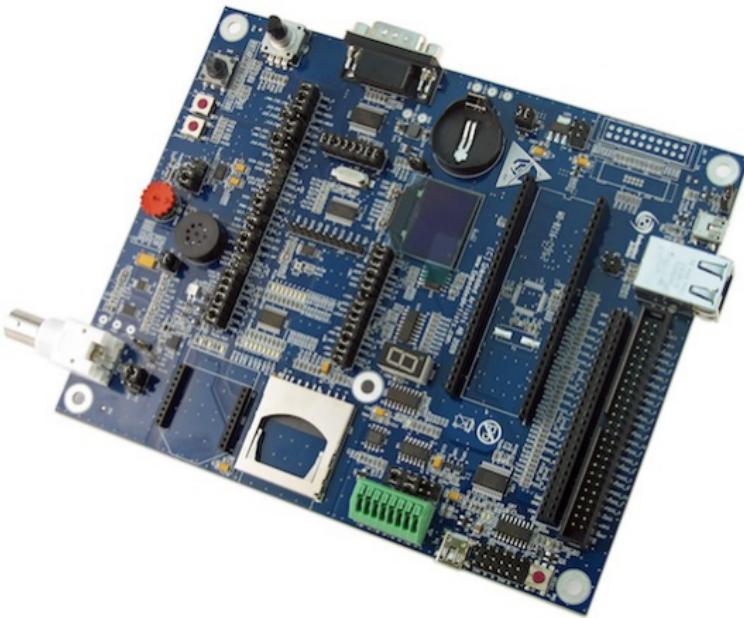
# LPCXpresso target board

En este tutorial vamos a utilizar el target que viene con el LPC1768.

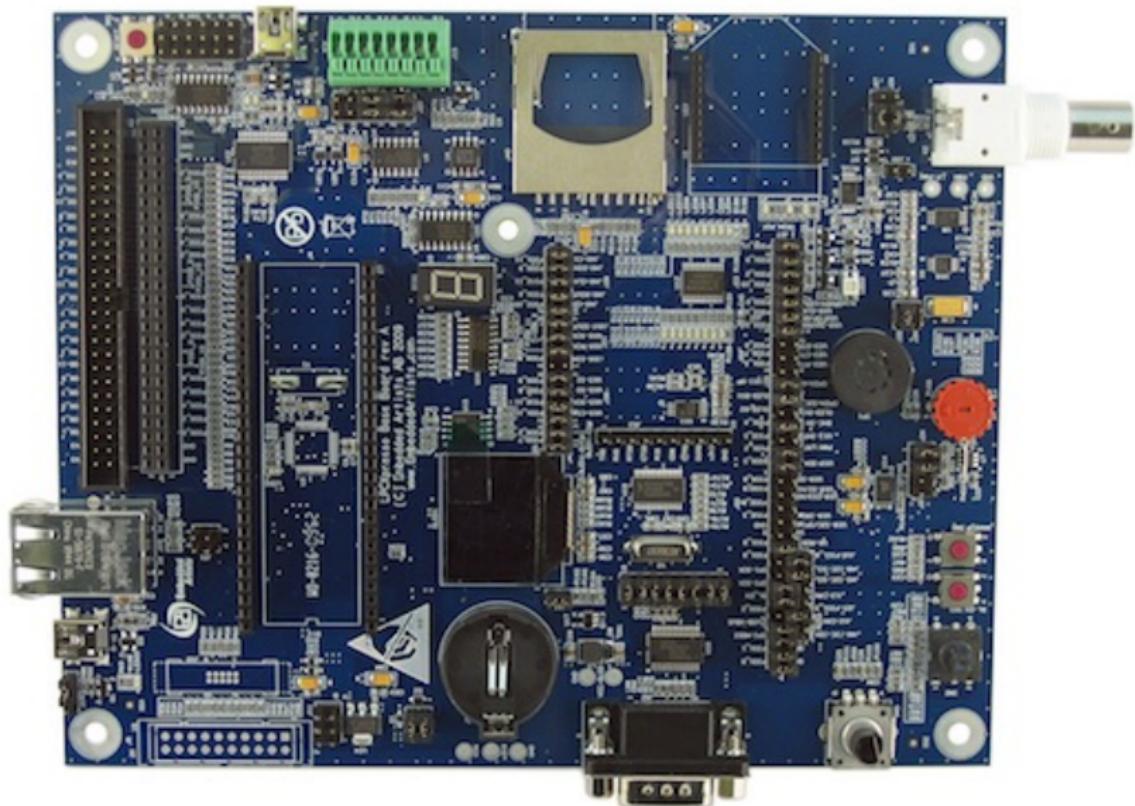


# LPCXpresso BaseBoard

El BaseBoard fue diseñado por Embedded Artists Junto con NXP. Permite conectarle un MBed o un LPCXpresso Target.



# LPCXpresso BaseBoard



# LPCXpresso BaseBoard

La placa contiene periféricos para desarrollo y experimentación:

## Generales:

- Socket for LPCXpresso and mbed module
- 50 pin expansion dual row pin/header list connector
- Battery powering (small coin battery)
- USB interface
- Reset pushbutton

## Digitales:

- RGB-LED (can be PWM controlled)
- 5-key joystick switch
- 2 pushbuttons, one for activating bootloader
- Rotary switch with quadrature encoding (timer capture)
- Temperature sensor with PWM output (timer capture)

## Analógicos:

- Trimming potentiometer input (analog input)
- PWM to analog LP-filtering (PWM output and analog input)
- Speaker output (PWM output)
- Oscilloscope probe inout stage

## Serial - UART:

- USB-to-serial bridge, with automatic ISP activation
- RS422/485 interface
- Interface socket for XBee RF-module

# LPCXpresso BaseBoard

## Continuación:

### Serial - SPI:

- Shift register driving 7-segment LED
- SD/MMC memory card interface
- Dataflash SPI-NOR flash

### Serial - I2C:

- PCA9532 port expander connected to 16 LEDs
- 8kbit E2PROM
- MMA7455L accelerometer with I2C interface
- Light sensor

### Serial - I2C/SPI

- SC16IS752 - I2C/SPI to 2xUART bridge; connected to RS232 full-modem interface and one expansion UART
- 96x64 pixel white OLED (alternative I2C/SPI interface)

### Extras

- CAN bus interface (can be simulated with LPCXpresso LPC1114/LPC1343)
- Ethernet RJ45 connector with integrated magnetic

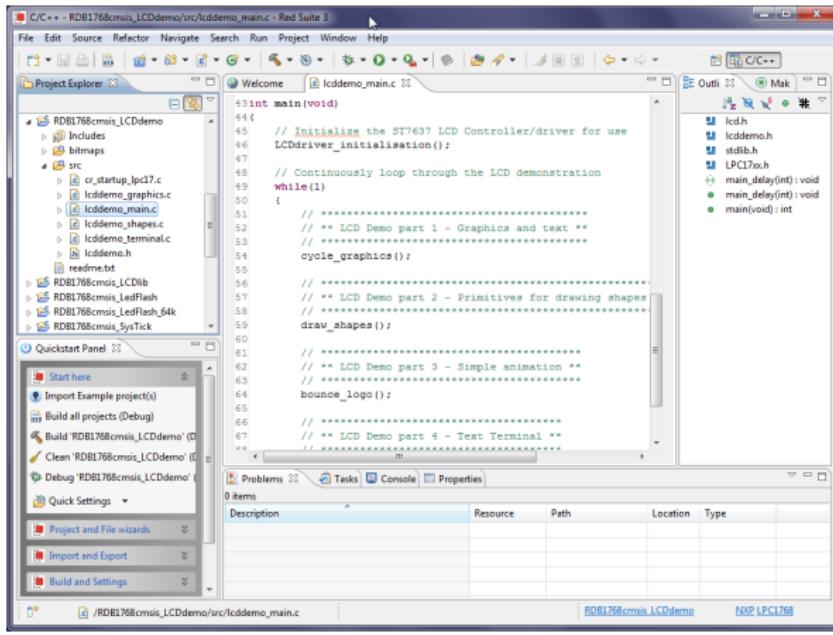
# Información adicional

Se recomienda revisar los siguientes documentos:

- LPC1768 User Manual (datasheet) [▶ Ver](#)
- LPCXpresso 1768 Target board: Esquemáticos [▶ Ver](#)
- LPCXpresso BaseBoard: Guía de usuario [▶ Ver](#)
- LPCXpresso BaseBoard: Esquemáticos [▶ Ver](#)

# LPCXpresso IDE (Eclipse)

El LPCXpresso IDE fue desarrollado por CodeRed junto a NXP. El mismo incluye un entorno de Eclipse específicamente adaptado para interactuar con el target board.



# Eclipse - Conceptos básicos

Eclipse utiliza algunos conceptos que no siempre son comunes a otros entornos de desarrollo por lo que vamos a ver algunos de ellos.

**workspace** Es el contenedor de nuestros proyectos. Estos proyectos pueden ser **aplicaciones** y/o **bibliotecas**. También almacena todas las configuraciones del entorno por lo que se puede mover muy fácilmente de computadora en computadora.

**proyecto** Este puede ser de dos tipos. Biblioteca estática o una aplicación ejecutable. Contiene archivos de código fuente (.c), encabezados (.h) y cualquier otro archivo que se deseé.

En general utilizaremos el workspace para intercambiar proyectos (en el sentido convencional de la palabra) ya que el mismo incluiría todas las bibliotecas necesarias.

# Tipos de proyectos

Los proyectos pueden ser de dos tipos:

- **Aplicaciones:** Se compilan y se pueden descargar directamente al target.
- **Bibliotecas estáticas:** Se pueden compilar, pero para usarlas, un proyecto de tipo aplicación debe hacer llamadas a las funciones que este contiene. Es decir, no puede tener un **main()**. Este tipo de proyectos no se puede descargar por si solo al microcontrolador.

# Tipos de proyectos - Ejemplo

Para exemplificar pensemos en un ejemplo de un sistema embebido: un reproductor de MP3. Este podría estar compuesto por una memoria SD, una pantalla táctil y un decodificador de MP3. Todos estos periféricos están controlados por un microcontrolador, por ejemplo un LPC1768.

# Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los prototipos de las funciones.

# Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los prototipos de las funciones.

# Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los **prototipos** de las funciones.

# Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los **prototipos** de las funciones.

# Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los **prototipos** de las funciones.

# Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los **prototipos** de las funciones.

## Section 2

### Conceptos de RTOS

# RTOS - ¿Qué es?

Un RTOS (Real Time Operating System) es un *programa* que se encarga de:

- Ordenar con precisión el tiempo de ejecución de las tareas
- Administrar los recursos del sistema como tiempo de uso de procesador, memoria, etc.
- Proveer una base consistente para el desarrollo del código

# RTOS - ¿Qué es?

Un RTOS (Real Time Operating System) es un *programa* que se encarga de:

- Ordenar con precisión el tiempo de ejecución de las tareas
- Administrar los recursos del sistema como tiempo de uso de procesador, memoria, etc.
- Proveer una base consistente para el desarrollo del código

# RTOS - ¿Qué es?

Un RTOS (Real Time Operating System) es un *programa* que se encarga de:

- Ordenar con precisión el tiempo de ejecución de las tareas
- Administrar los recursos del sistema como tiempo de uso de procesador, memoria, etc.
- Proveer una base consistente para el desarrollo del código

# RTOS - ¿Qué es?

Un RTOS (Real Time Operating System) es un *programa* que se encarga de:

- Ordenar con precisión el tiempo de ejecución de las tareas
- Administrar los recursos del sistema como tiempo de uso de procesador, memoria, etc.
- Proveer una base consistente para el desarrollo del código

# RTOS - ¿Qué es?

Las aplicaciones que utilizan un RTOS son diversas:

# RTOS - ¿Qué es?

Las aplicaciones que utilizan un RTOS son diversas:



# RTOS - ¿Qué es?

Las aplicaciones que utilizan un RTOS son diversas:



# RTOS - ¿Qué es?

Las aplicaciones que utilizan un RTOS son diversas:



# RTOS - ¿Qué es?

Las aplicaciones que utilizan un RTOS son diversas:



# RTOS - ¿Qué es?

Todas tienen un denominador común:

## TAREAS CONCURRENTES

El RTOS crea la ilusión de múltiples tareas ejecutándose en simultáneo.

# RTOS - ¿Qué es?

Todas tienen un denominador común:

## TAREAS CONCURRENTES

El RTOS crea la ilusión de múltiples tareas ejecutándose en simultáneo.

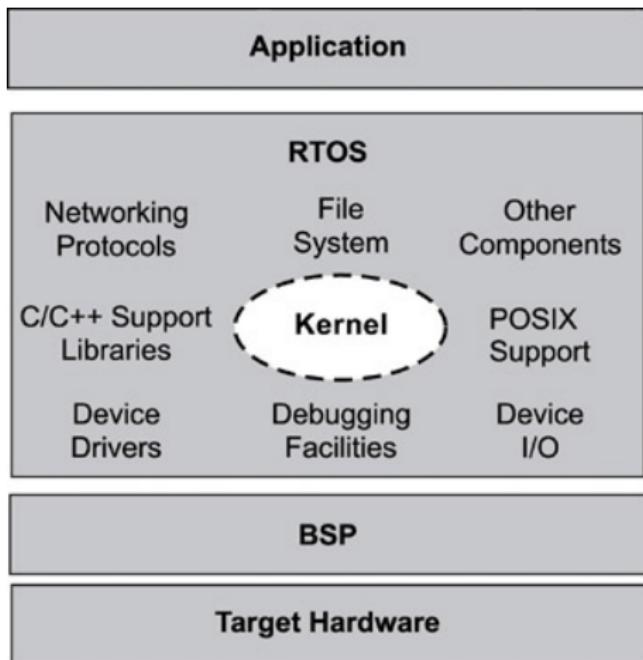
# RTOS - ¿Qué es?

Todas tienen un denominador común:

## TAREAS CONCURRENTES

El RTOS crea la ilusión de múltiples tareas ejecutándose en simultáneo.

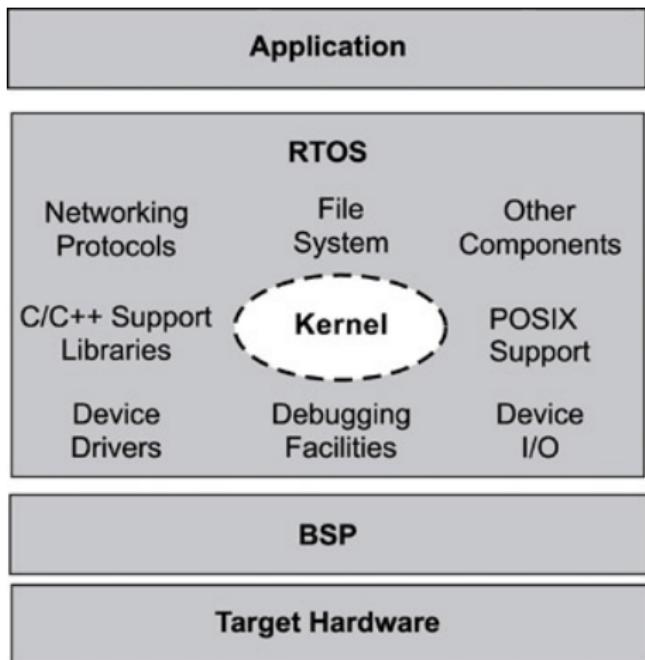
# RTOS - ¿Qué es?



El RTOS se sitúa entre la capa BSP (Board Support Package, o "port") y la capa de aplicación.

Puede incluir varios módulos (protocolos de red, sistema de archivos, etc.)

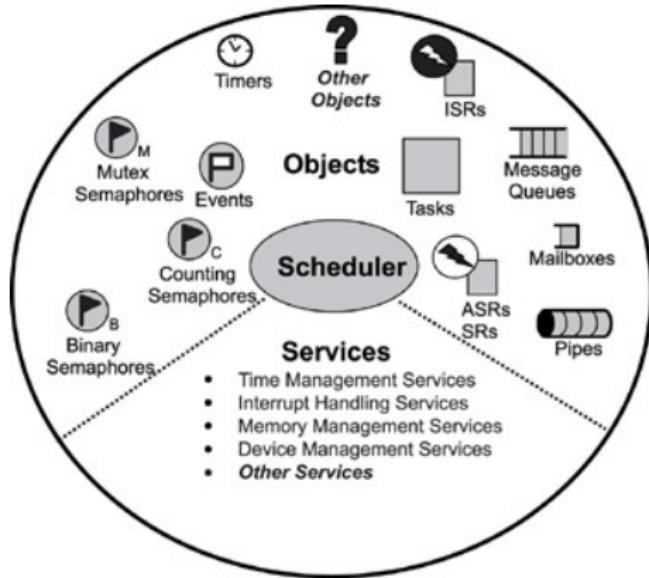
# RTOS - ¿Qué es?



El RTOS se sitúa entre la capa BSP (Board Support Package, o "port") y la capa de aplicación.

Puede incluir varios módulos (protocolos de red, sistema de archivos, etc.)

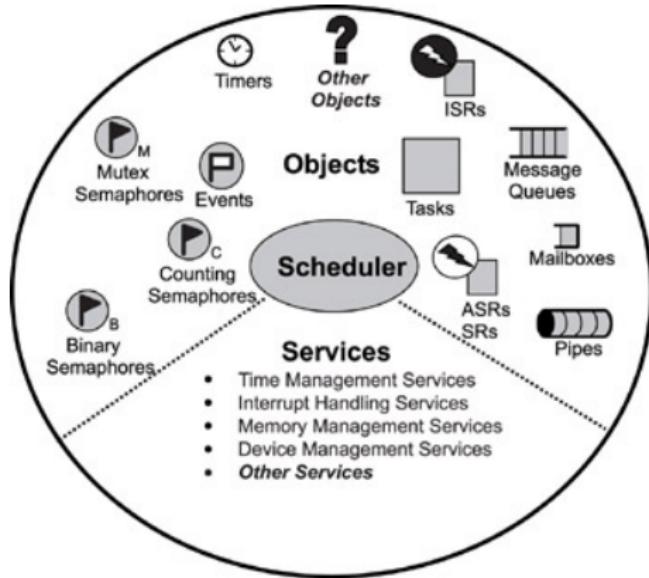
# RTOS - Componentes



Los componentes de un RTOS pueden clasificarse ampliamente en 3 grupos:

- Scheduler: maneja los hilos de ejecución de las tareas.
- Objetos: tareas, colas, semáforos, etc.
- Servicios: operaciones realizadas sobre los objetos (manejo de interrupciones, de memoria, etc.)

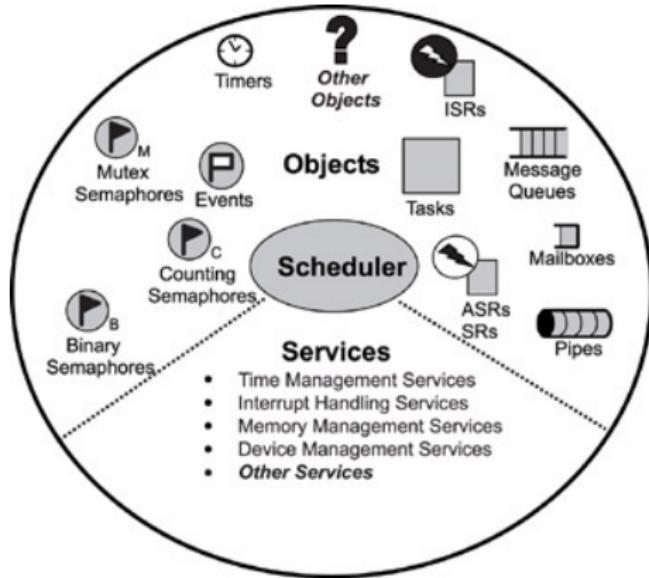
# RTOS - Componentes



Los componentes de un RTOS pueden clasificarse ampliamente en 3 grupos:

- Scheduler: maneja los hilos de ejecución de las tareas.
- Objetos: tareas, colas, semáforos, etc.
- Servicios: operaciones realizadas sobre los objetos (manejo de interrupciones, de memoria, etc.)

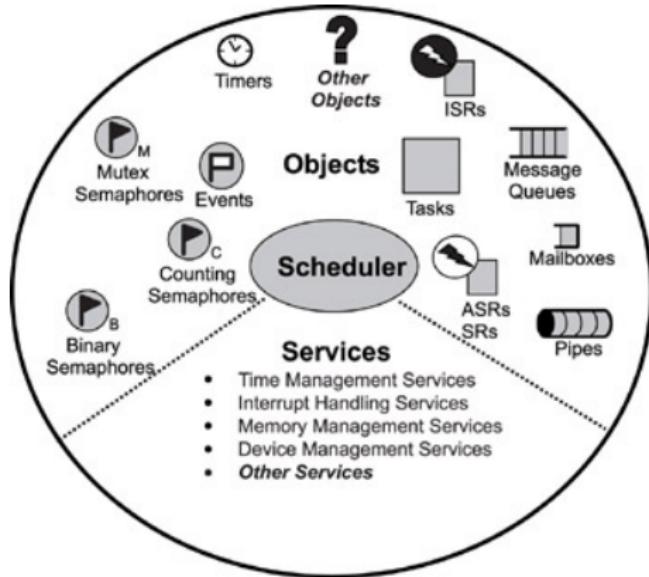
# RTOS - Componentes



Los componentes de un RTOS pueden clasificarse ampliamente en 3 grupos:

- Scheduler: maneja los hilos de ejecución de las tareas.
- Objetos: tareas, colas, semáforos, etc.
- Servicios: operaciones realizadas sobre los objetos (manejo de interrupciones, de memoria, etc.)

# RTOS - Componentes



Los componentes de un RTOS pueden clasificarse ampliamente en 3 grupos:

- Scheduler: maneja los hilos de ejecución de las tareas.
- Objetos: tareas, colas, semáforos, etc.
- Servicios: operaciones realizadas sobre los objetos (manejo de interrupciones, de memoria, etc.)

# RTOS - Scheduler

El *Scheduler* determina cuándo se ejecutará cada tarea. Existen diferentes esquemas de scheduling:

- **Cooperativo:** La tarea en ejecución cede el uso de CPU a otra voluntariamente.
- **Preemptive:** La tarea en ejecución cede el uso de CPU a otra por orden del scheduler.
  - **Priority-Based:** Se asignan *prioridades* a las tareas para acceder al uso de CPU.
  - **Round-Robin:** Se asigna un tiempo fijo de uso de CPU a cada tarea en orden circular. Puede combinarse con el uso de prioridades.

# RTOS - Scheduler

El *Scheduler* determina cuándo se ejecutará cada tarea. Existen diferentes esquemas de scheduling:

- **Cooperativo:** La tarea en ejecución cede el uso de CPU a otra voluntariamente.
- **Preemptive:** La tarea en ejecución cede el uso de CPU a otra por orden del scheduler.
  - **Priority-Based:** Se asignan *prioridades* a las tareas para acceder al uso de CPU.
  - **Round-Robin:** Se asigna un tiempo fijo de uso de CPU a cada tarea en orden circular. Puede combinarse con el uso de prioridades.

# RTOS - Scheduler

El *Scheduler* determina cuándo se ejecutará cada tarea. Existen diferentes esquemas de scheduling:

- **Cooperativo:** La tarea en ejecución cede el uso de CPU a otra voluntariamente.
- **Preemptive:** La tarea en ejecución cede el uso de CPU a otra por orden del scheduler.
  - **Priority-Based:** Se asignan *prioridades* a las tareas para acceder al uso de CPU.
  - **Round-Robin:** Se asigna un tiempo fijo de uso de CPU a cada tarea en orden circular. Puede combinarse con el uso de prioridades.

# RTOS - Scheduler

El *Scheduler* determina cuándo se ejecutará cada tarea. Existen diferentes esquemas de scheduling:

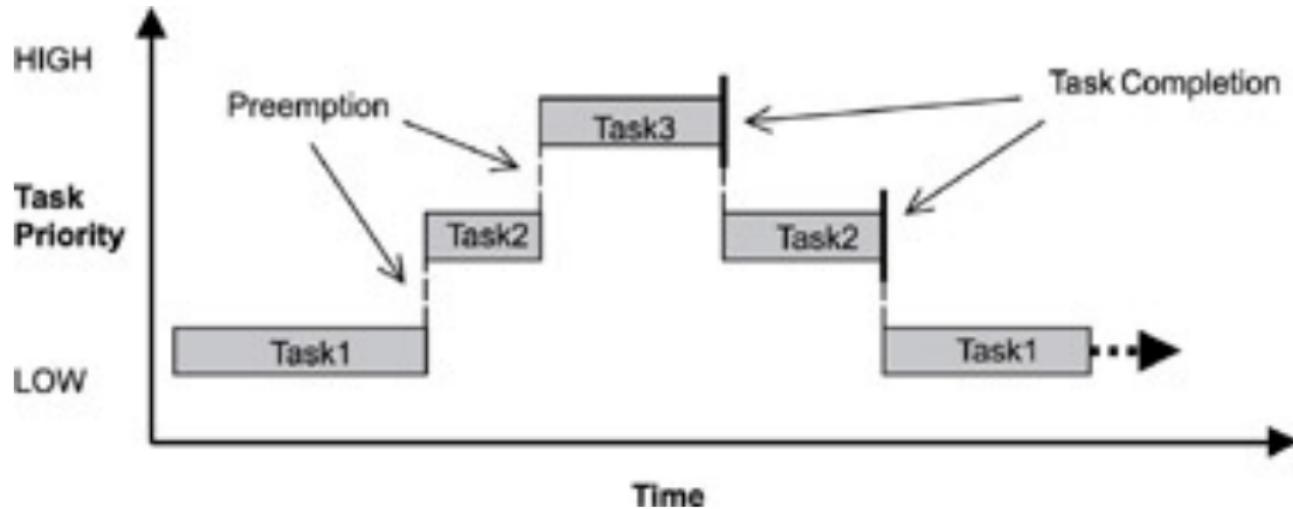
- **Cooperativo:** La tarea en ejecución cede el uso de CPU a otra voluntariamente.
- **Preemptive:** La tarea en ejecución cede el uso de CPU a otra por orden del scheduler.
  - **Priority-Based:** Se asignan *prioridades* a las tareas para acceder al uso de CPU.
  - **Round-Robin:** Se asigna un tiempo fijo de uso de CPU a cada tarea en orden circular. Puede combinarse con el uso de prioridades.

# RTOS - Scheduler

El *Scheduler* determina cuándo se ejecutará cada tarea. Existen diferentes esquemas de scheduling:

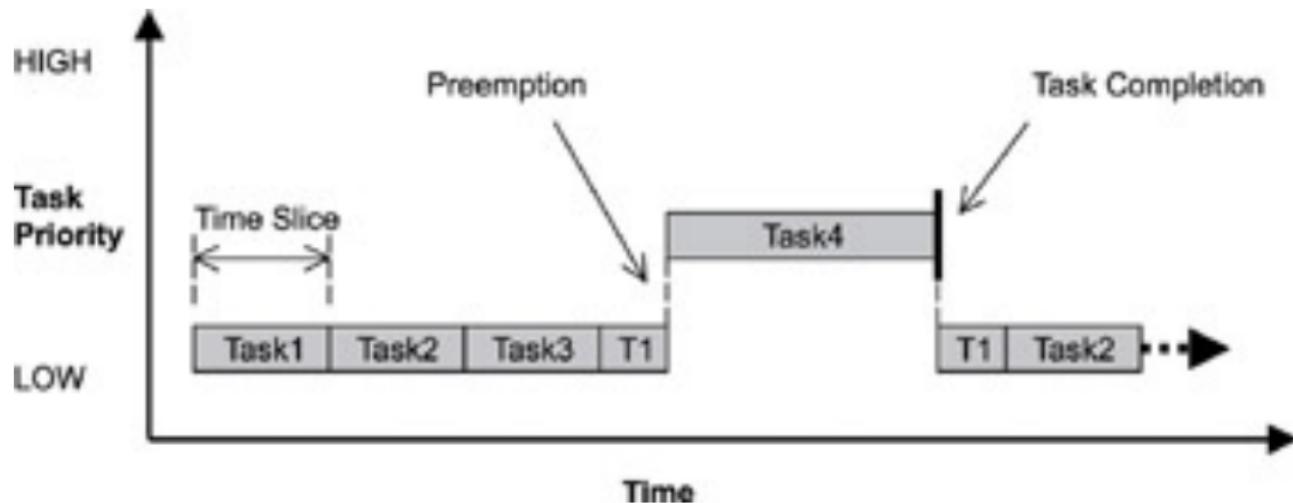
- **Cooperativo:** La tarea en ejecución cede el uso de CPU a otra voluntariamente.
- **Preemptive:** La tarea en ejecución cede el uso de CPU a otra por orden del scheduler.
  - **Priority-Based:** Se asignan *prioridades* a las tareas para acceder al uso de CPU.
  - **Round-Robin:** Se asigna un tiempo fijo de uso de CPU a cada tarea en orden circular. Puede combinarse con el uso de prioridades.

# RTOS - Scheduler



Ejemplo de esquema Priority-Based

# RTOS - Scheduler



Ejemplo de esquema Round-Robin con prioridades.

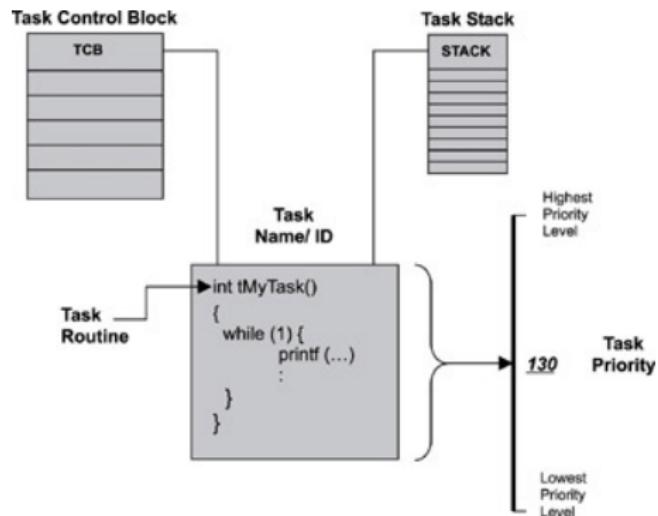
# RTOS - Tareas

- Una *tarea* es un hilo de ejecución independiente que puede competir con otras tareas por tiempo de ejecución.
- Pueden ser creadas y eliminadas en tiempo de *ejecución*.

# RTOS - Tareas

- Una *tarea* es un hilo de ejecución independiente que puede competir con otras tareas por tiempo de ejecución.
- Pueden ser creadas y eliminadas en tiempo de ejecución.

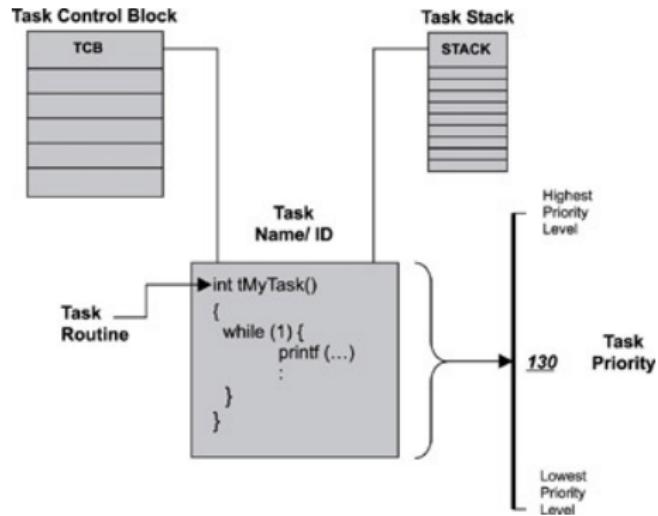
# RTOS - Tareas



Se compone de:

- Nombre/ID
- Prioridad (esquema preemptivo)
- Stack
- Rutina (código)
- Bloque de control

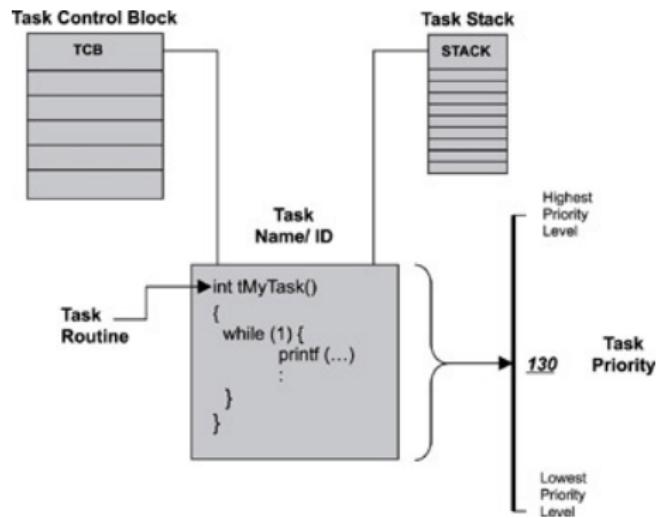
# RTOS - Tareas



Se compone de:

- Nombre/ID
- Prioridad (esquema preemptive)
- Stack
- Rutina (código)
- Bloque de control

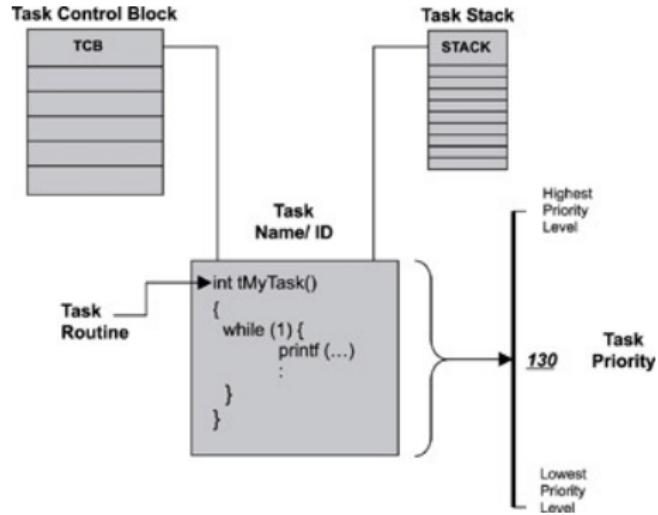
# RTOS - Tareas



Se compone de:

- Nombre/ID
- Prioridad (esquema preemptivo)
- Stack
- Rutina (código)
- Bloque de control

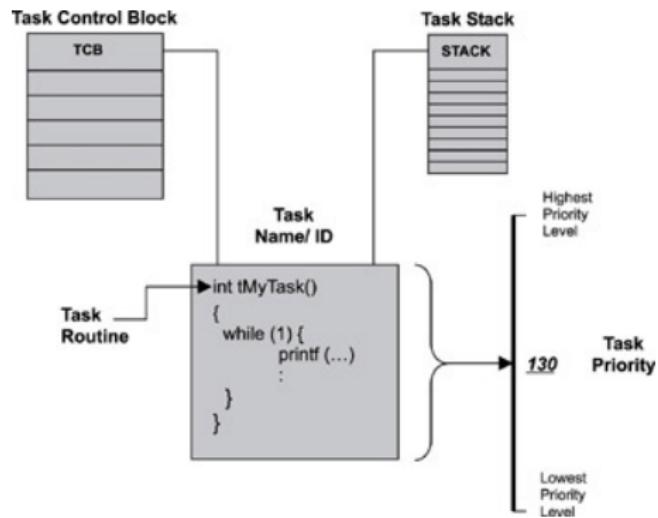
# RTOS - Tareas



Se compone de:

- Nombre/ID
- Prioridad (esquema preemptive)
- Stack
- Rutina (código)
- Bloque de control

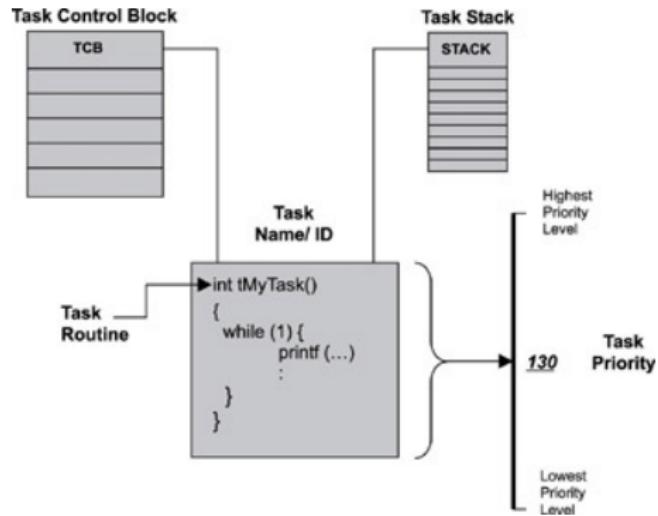
# RTOS - Tareas



Se compone de:

- Nombre/ID
- Prioridad (esquema preemptive)
- Stack
- Rutina (código)
- Bloque de control

# RTOS - Tareas



Se compone de:

- Nombre/ID
- Prioridad (esquema preemptive)
- Stack
- Rutina (código)
- Bloque de control

# RTOS - Tareas

Algunos ejemplos de tareas:

- Inicialización del sistema (ejecuta solo una vez)
- Filtrar una señal (posiblemente periódica)
- Mostrar en pantalla (puede activarse ante un evento)
- Tarea *idle*

Las tareas pueden tener diferentes requerimientos de temporización.

# RTOS - Tareas

Algunos ejemplos de tareas:

- Inicialización del sistema (ejecuta solo una vez)
- Filtrar una señal (posiblemente periódica)
- Mostrar en pantalla (puede activarse ante un evento)
- Tarea *idle*

Las tareas pueden tener diferentes requerimientos de temporización.

# RTOS - Tareas

Algunos ejemplos de tareas:

- Inicialización del sistema (ejecuta solo una vez)
- Filtrar una señal (posiblemente periódica)
- Mostrar en pantalla (puede activarse ante un evento)
- Tarea *idle*

Las tareas pueden tener diferentes requerimientos de temporización.

# RTOS - Tareas

Algunos ejemplos de tareas:

- Inicialización del sistema (ejecuta solo una vez)
- Filtrar una señal (posiblemente periódica)
- Mostrar en pantalla (puede activarse ante un evento)
- Tarea *idle*

Las tareas pueden tener diferentes requerimientos de temporización.

# RTOS - Tareas

Algunos ejemplos de tareas:

- Inicialización del sistema (ejecuta solo una vez)
- Filtrar una señal (posiblemente periódica)
- Mostrar en pantalla (puede activarse ante un evento)
- Tarea *idle*

Las tareas pueden tener diferentes requerimientos de temporización.

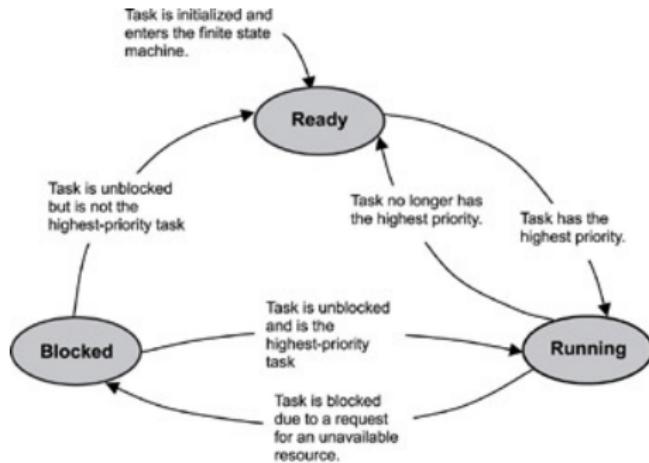
# RTOS - Tareas

Algunos ejemplos de tareas:

- Inicialización del sistema (ejecuta solo una vez)
- Filtrar una señal (posiblemente periódica)
- Mostrar en pantalla (puede activarse ante un evento)
- Tarea *idle*

Las tareas pueden tener diferentes requerimientos de temporización.

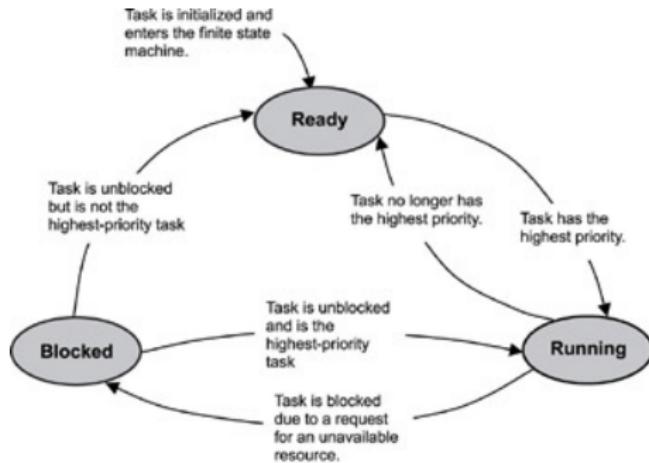
# RTOS - Tareas



Los estados posibles de una tarea son:

- **Ready:** compite por tiempo de ejecución
- **Running:** tarea activa
- **Blocked:** esperando pasar a Ready (podrá activarse ante un evento o cuando pase cierto tiempo)

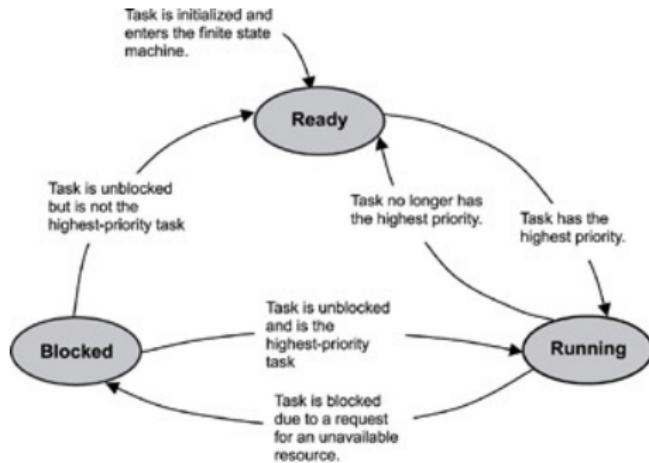
# RTOS - Tareas



Los estados posibles de una tarea son:

- **Ready:** compite por tiempo de ejecución
- **Running:** tarea activa
- **Blocked:** esperando pasar a Ready (podrá activarse ante un evento o cuando pase cierto tiempo)

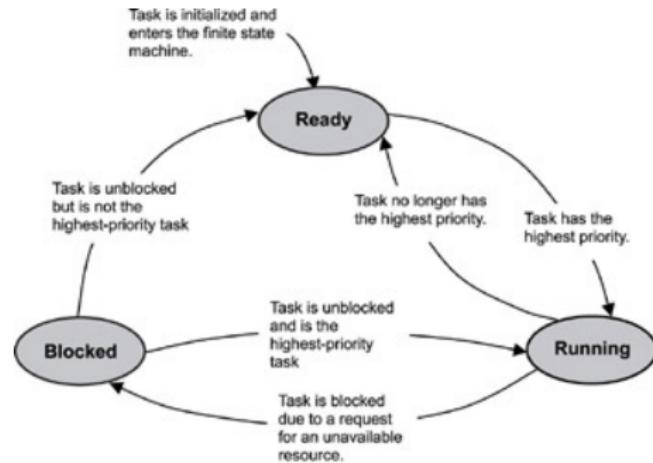
# RTOS - Tareas



Los estados posibles de una tarea son:

- **Ready**: compite por tiempo de ejecución
- **Running**: tarea activa
- **Blocked**: esperando pasar a Ready (podrá activarse ante un evento o cuando pase cierto tiempo)

# RTOS - Tareas



Los estados posibles de una tarea son:

- **Ready**: compite por tiempo de ejecución
- **Running**: tarea activa
- **Blocked**: esperando pasar a Ready (podrá activarse ante un evento o cuando pase cierto tiempo)

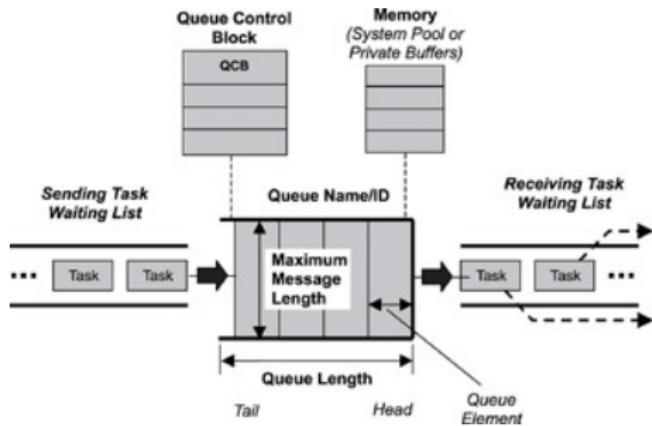
# RTOS - Tareas

Estructura típica de una tarea:

```
void tarea_tipica()
{
    /* Código de inicialización */
    for ( ;; )
    {
        /* Cuerpo principal */
        /* Llama a bloquear */
    }
    /* La tarea NUNCA debe llegar aquí */
}
```

# RTOS - Colas

Su función principal es proveer un mecanismo de intercambio de datos entre tareas. Son FIFO.

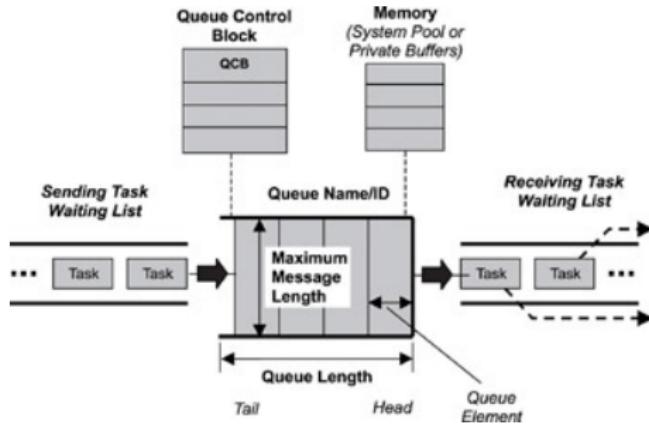


Se compone de:

- Nombre/ID
- Tamaño y tipo de datos a almacenar
- Bloque de control

# RTOS - Colas

Su función principal es proveer un mecanismo de intercambio de datos entre tareas. Son FIFO.

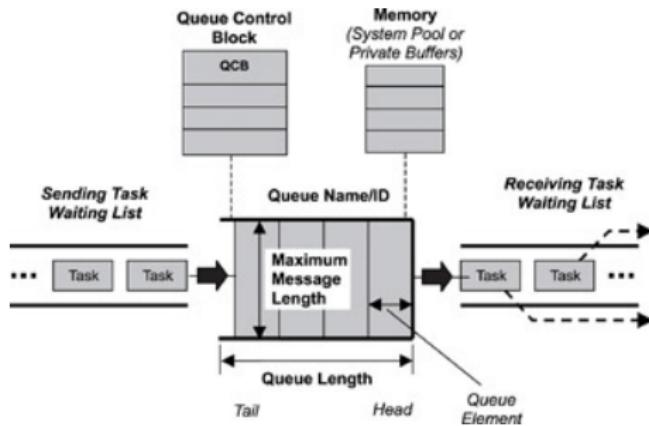


Se compone de:

- Nombre/ID
- Tamaño y tipo de datos a almacenar
- Bloque de control

# RTOS - Colas

Su función principal es proveer un mecanismo de intercambio de datos entre tareas. Son FIFO.

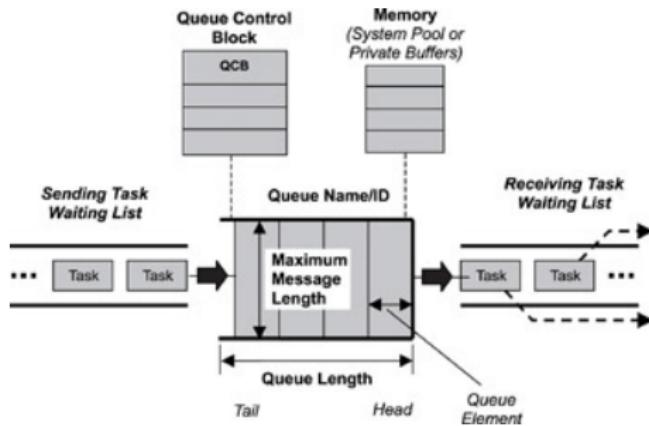


Se compone de:

- Nombre/ID
- Tamaño y tipo de datos a almacenar
- Bloque de control

# RTOS - Colas

Su función principal es proveer un mecanismo de intercambio de datos entre tareas. Son FIFO.



Se compone de:

- Nombre/ID
- Tamaño y tipo de datos a almacenar
- Bloque de control

# RTOS - Colas

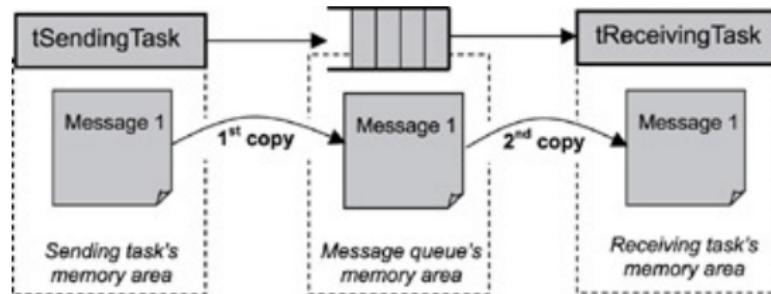
- Varias tareas pueden acceder a una misma cola.

# RTOS - Colas

- Varias tareas pueden acceder a una misma cola.
- Una tarea puede elegir bloquearse si su cola está vacía. Al llegar un elemento, automáticamente pasa a Ready.

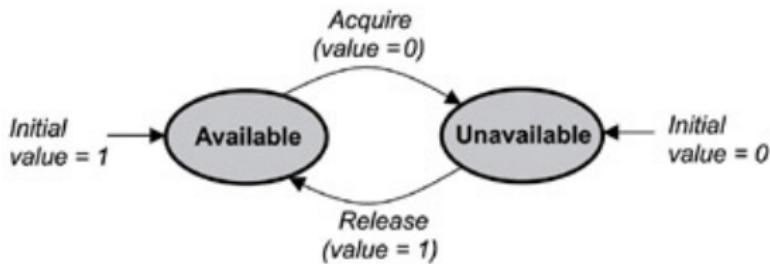
# RTOS - Colas

- Varias tareas pueden acceder a una misma cola.
- Una tarea puede elegir bloquearse si su cola está vacía. Al llegar un elemento, automáticamente pasa a Ready.
- Cuidado! Cargar datos en una cola causa una copia de los datos en memoria. Puede ser conveniente pasar un puntero.



# RTOS - Semáforos

Son objetos como las colas. Pueden ser creados y destruidos. Su función principal es proveer un mecanismo de sincronización entre tareas.

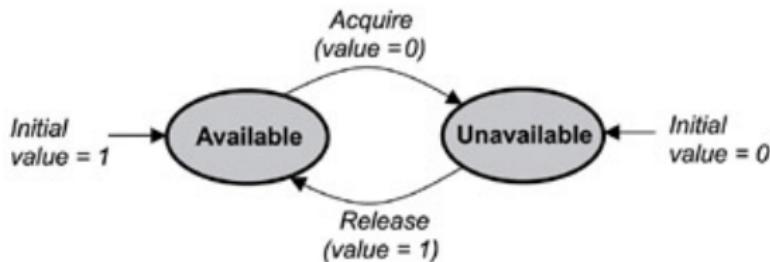


Ejemplo: acceder a un recurso (como escribir en una pantalla):

- Para acceder al recurso la tarea *pide* el semáforo
- Si está disponible, toma el semáforo y realiza la operación.
- Al terminar, lo libera para que otros puedan acceder.

# RTOS - Semáforos

Son objetos como las colas. Pueden ser creados y destruidos. Su función principal es proveer un mecanismo de sincronización entre tareas.

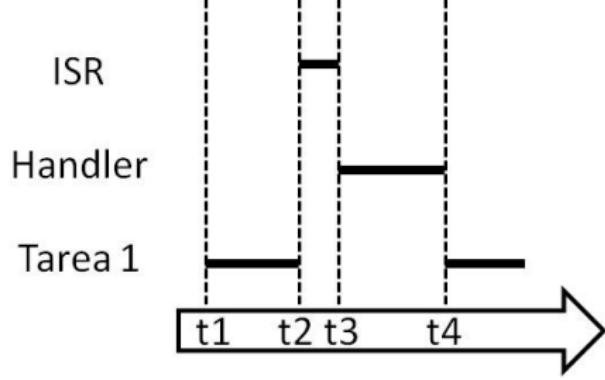


Ejemplo: acceder a un recurso (como escribir en una pantalla):

- Para acceder al recurso la tarea *pide* el semáforo
- Si está disponible, toma el semáforo y realiza la operación.
- Al terminar, lo libera para que otros puedan acceder.

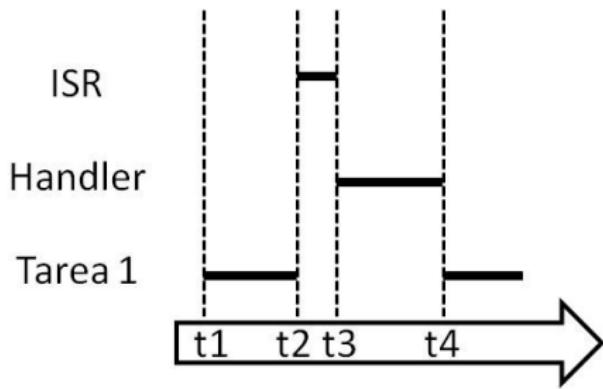
# RTOS - Semáforos

También pueden utilizarse para disparar un handler ante una *interrupción*



- La Tarea 1 está activa hasta que en t2 ocurre una interrupción.
- El ISR se ejecuta y *cede* un semáforo asociado al Handler en t3.
- El Handler se desbloquea y toma el semáforo, de la misma manera que si le llegara un dato por una cola.

# RTOS - Semáforos



- El Handler ejecuta su rutina dentro de un `for(;;)` y cuando termina quiere volver a tomar el semáforo.
- Como no hubo otra interrupción que cediera un semáforo, el Handler se bloquea y vuelve el control a la Tarea 1 en t4.

# RTOS - Semáforos

Los recién descriptos se conocen como semáforos *binarios* (tienen 2 estados). Existen otros métodos de sincronización:

- **Semáforos de conteo:** pueden tomarse o liberarse más de una vez.
- **Mutex:** similares a los semáforos binarios pero con funciones adicionales (como herencia de prioridades).

# RTOS - Semáforos

Los recién descriptos se conocen como semáforos *binarios* (tienen 2 estados). Existen otros métodos de sincronización:

- **Semáforos de conteo:** pueden tomarse o liberarse más de una vez.
- **Mutex:** similares a los semáforos binarios pero con funciones adicionales (como herencia de prioridades).

# RTOS - Semáforos

Los recién descriptos se conocen como semáforos *binarios* (tienen 2 estados). Existen otros métodos de sincronización:

- **Semáforos de conteo:** pueden tomarse o liberarse más de una vez.
- **Mutex:** similares a los semáforos binarios pero con funciones adicionales (como herencia de prioridades).

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Abstracción de la implementación del scheduling
- Mayor modularidad
- Facilita el desarrollo en grupos
- Mayor facilidad al debuggear
- Mayor reusabilidad del código
- Posibilidad de realizar *profiling* (tarea *idle*)
- Reducen el tiempo de desarrollo

y la lista sigue ...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Mejoran la escalabilidad
- Facilitan la portabilidad
- Mejoran la mantenibilidad
- Reducen la incertidumbre

etc...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Mejoran la escalabilidad
- Facilitan la portabilidad
- Mejoran la mantenibilidad
- Reducen la incertidumbre

etc...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Mejoran la escalabilidad
- Facilitan la portabilidad
- Mejoran la mantenibilidad
- Reducen la incertidumbre

etc...

# RTOS - ¿Por qué usarlo?

Existen varias razones por las cuales usar un RTOS:

- Mejoran la escalabilidad
- Facilitan la portabilidad
- Mejoran la mantenibilidad
- Reducen la incertidumbre

etc...

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Por qué decidimos usar **FreeRTOS**:

- Es de código abierto
  - Código ampliamente comentado
  - Sencillo de portar (existen más de 23 ports)
- Ocupa poco espacio en flash (~5KB) necesita poca RAM (~5KB + Heap) y el overhead que introduce es mínimo (entre 1% y 4% del tiempo de CPU) a cambio de una gran utilidad.
- Ampliamente documentado
- Existe una comunidad de usuarios importante
- Libre de regalías. Puede ser usado en aplicaciones comerciales bajo licencia GNU versión 2.

# FreeRTOS

Existen también otras licencias para este RTOS:

- Existe una licencia *comercial* con soporte oficial y garantías legales (OpenRTOS).
- Existe una licencia certificada SIL3 para *aplicaciones críticas* (SafeRTOS).
- La versión FreeRTOS 6.0.0 portada a Cortex-M3 soporta MPU (Memory Protection Unit).

# FreeRTOS

Existen también otras licencias para este RTOS:

- Existe una licencia *comercial* con soporte oficial y garantías legales (OpenRTOS).
- Existe una licencia certificada SIL3 para *aplicaciones críticas* (SafeRTOS).
- La versión FreeRTOS 6.0.0 portada a Cortex-M3 soporta MPU (Memory Protection Unit).

# FreeRTOS

Existen también otras licencias para este RTOS:

- Existe una licencia *comercial* con soporte oficial y garantías legales (OpenRTOS).
- Existe una licencia certificada SIL3 para *aplicaciones críticas* (SafeRTOS).
- La versión FreeRTOS 6.0.0 portada a Cortex-M3 soporta MPU (Memory Protection Unit).

# FreeRTOS

Existen también otras licencias para este RTOS:

- Existe una licencia *comercial* con soporte oficial y garantías legales (OpenRTOS).
- Existe una licencia certificada SIL3 para *aplicaciones críticas* (SafeRTOS).
- La versión FreeRTOS 6.0.0 portada a Cortex-M3 soporta MPU (Memory Protection Unit).

## Section 3

LPCXpresso + FreeRTOS

Es recomendable tener un **workspace** para cada proyecto que realicemos. En él configuraremos el RTOS. La estructura del workspace podrá ser:

- FreeRTOS\_Library
- Biblioteca\_de\_funciones\_1
- Biblioteca\_de\_funciones\_2
- Driver\_DAC\_ADC
- MP3\_Player

Es recomendable tener un **workspace** para cada proyecto que realicemos. En él configuraremos el RTOS. La estructura del workspace podrá ser:

- FreeRTOS\_Library
- Biblioteca\_de\_funciones\_1
- Biblioteca\_de\_funciones\_2
- Driver\_DAC\_ADC
- MP3\_Player

Es recomendable tener un **workspace** para cada proyecto que realicemos.  
En él configuraremos el RTOS. La estructura del workspace podrá ser:

- FreeRTOS\_Library
- Biblioteca\_de\_funciones\_1
- Biblioteca\_de\_funciones\_2
- Driver\_DAC\_ADC
- MP3\_Player

Es recomendable tener un **workspace** para cada proyecto que realicemos. En él configuraremos el RTOS. La estructura del workspace podrá ser:

- FreeRTOS\_Library
- Biblioteca\_de\_funciones\_1
- Biblioteca\_de\_funciones\_2
- Driver\_DAC\_ADC
- MP3\_Player

Es recomendable tener un **workspace** para cada proyecto que realicemos. En él configuraremos el RTOS. La estructura del workspace podrá ser:

- FreeRTOS\_Library
- Biblioteca\_de\_funciones\_1
- Biblioteca\_de\_funciones\_2
- Driver\_DAC\_ADC
- **MP3\_Player**

El FreeRTOS esta compuesto por:

- list.c
- queue.c
- tasks.c
- **Portable**
  - heap\_X.c
  - port.c
  - portmacro.h
- **Include**
  - FreeRTOSConfig.h
  - ...

El FreeRTOS esta compuesto por:

- list.c
- queue.c
- tasks.c
- **Portable**
  - heap\_X.c
  - port.c
  - portmacro.h
- **Include**
  - FreeRTOSConfig.h
  - ...

El FreeRTOS esta compuesto por:

- list.c
- queue.c
- tasks.c
- **Portable**
  - heap\_X.c
  - port.c
  - portmacro.h
- **Include**
  - FreeRTOSConfig.h
  - ...

Antes de "instalarlo" en nuestro proyecto, debemos configurar el comportamiento del RTOS. Para ello freeRTOS provee un archivo **FreeRTOSConfig.h** en el que mediante macros podremos configurarlo.

Este archivo tiene la siguiente forma:

```
#define configUSE_TICK_HOOK 1
#define configCPU_CLOCK_HZ ( 100000000UL )
#define configTICK_RATE_HZ ( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE ( ( unsigned short ) 100 )
#define configTOTAL_HEAP_SIZE ( ( size_t ) ( 24 * 1024 ) )
#define configMAX_TASK_NAME_LEN ( 12 )
#define configUSE_TRACE_FACILITY 0

#define INCLUDE_vTaskPrioritySet 1
#define INCLUDE_uxTaskPriorityGet 1
#define INCLUDE_vTaskDelete 1
#define INCLUDE_vTaskCleanUpResources 0
```

FreeRTOSconfig.h

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphores)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphores)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphoros)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphores)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphoros)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphoros)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphoros)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

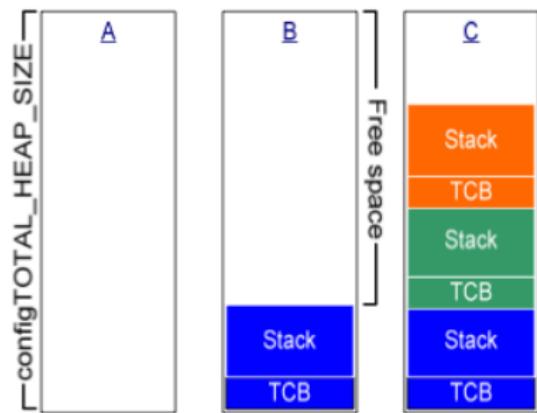
- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphoros)
- **INCLUDE\_XXXXXX** (funciones especiales)

Los aspectos más importantes que debemos configurar son:

- **configUSE\_PREEMPTION**
- **configUSE\_XXXX\_HOOK** (diferentes hooks)
- **configMAX\_PRIORITIES**
- **configCPU\_CLOCK\_HZ**
- **configTICK\_RATE\_HZ**
- **configMINIMAL\_STACK\_SIZE**
- **configTOTAL\_HEAP\_SIZE**
- **configUSE\_XXXX** (recursos: mutex, semaphoros)
- **INCLUDE\_XXXXXX** (funciones especiales)

# Total Heap Size

El freeRTOS utiliza este HEAP para almacenar el TCB y los diferentes STACKs de las tareas. También lo usará para guardar en memoria semaforos, mutex, etc.



## Subsection 3

### Instalación



- Collecting information
- Dynamic Update
- Preparing installation
- Installing Windows**
- Finalizing installation

Setup will complete in  
approximately:  
**19 minutes**

Installing Start menu items



## Surfing the Internet: safe, fast, and flexible

Windows® XP includes the most secure version of Internet Explorer to date. Many new features help enhance the security and privacy of information that you send and receive over the Internet, while simplifying the daily tasks that you perform. New innovative browser capabilities--such as a pop-up blocker, the ability to control plug-ins, and more--let you experience the Web exactly the way that you want.



No! es más facil. Simplemente tenemos que compilar la biblioteca (CONFIGURADA) de FreeRTOS y linkearla con nuestro proyecto. Luego hay que hacer algunas modificaciones al código generado por LPCXpresso.

# Tabla de Handlers (IRQ)

```

__attribute__((section(".isr-vector")))
void (* const g_pfnVectors[])(void) =
{
// Core Level - CM3
(void *)&_vStackTop,
Reset_Handler,
NMI_Handler,
HardFault_Handler,
MemManage_Handler,
BusFault_Handler,
UsageFault_Handler,
0,
0,
0,
0,
0,
SVCall_Handler,
DebugMon_Handler,
0
PendSV_Handler,
SysTick_Handler,

// Chip Level - LPC17
WDT_IRQHandler,
TIMER0_IRQHandler,
TIMER1_IRQHandler,
.....

```

```

__attribute__((section(".isr-vector")))
void (* const g_pfnVectors[])(void) =
{
// Core Level - CM3
(void *)&_vStackTop,
Reset_Handler,
NMI_Handler,
HardFault_Handler,
MemManage_Handler,
BusFault_Handler,
UsageFault_Handler,
0,
0,
0,
0,
0,
vPortSVCHandler,
DebugMon_Handler,
0
xPortPendSVHandler,
xPortSysTickHandler,

// Chip Level - LPC17
WDT_IRQHandler,
TIMER0_IRQHandler,
TIMER1_IRQHandler,
.....

```

## Section 4

Estructura de una aplicación

Luego que ya tenemos instalado el freeRTOS, tenemos que crear tareas e inicializar el **Scheduler**. Entonces una aplicación que utilice este RTOS podría ser de la siguiente manera:

```
// FreeRTOS includes
#include "FreeRTOS.h"
#include "task.h"

int main( void )
{
    // Creo las tareas
    xTaskCreate(UART_Driver,"UART",200,NULL,3,NULL);
    xTaskCreate(test1,"TEST",200,NULL,1,NULL);
    xTaskCreate(LCD,"LCD",200,NULL,2,NULL);

    // Inicio el scheduler
    vTaskStartScheduler();

    // Nunca llego aca
    for(;;);

    return 0;
}
```

## ¿Cómo es una tarea?

```
void miTarea(void * pvParameters){  
    int i;  
    // Cosas que se tengan que hacer solo una vez  
  
    // Loop principal  
    for (++;)  
    {  
        for (i =0;i <1000000;i++);  
        vTaskDelay(500/portTICK_RATE_MS );  
    }  
}
```

Nuestra tarea como diseñadores ahora es determinar que cosas de nuestras aplicaciones serán tareas del sistema, que recursos de comunicación y sincronización entre tareas tendremos que crear y cuales seran las prioridades de las mismas.

Algunas características que deben tener las tareas:

- Son funciones de C estandar
- Tiene una prioridad determinada
- Se pueden crear multiples instancias
- NO RETORNA NUNCA

Algunas características que deben tener las tareas:

- Son funciones de C estandar
- Tiene una prioridad determinada
- Se pueden crear multiples instancias
- NO RETORNA NUNCA

Algunas características que deben tener las tareas:

- Son funciones de C estandar
- Tiene una prioridad determinada
- Se pueden crear multiples instancias
- NO RETORNA NUNCA

Algunas características que deben tener las tareas:

- Son funciones de C estandar
- Tiene una prioridad determinada
- Se pueden crear multiples instancias
- NO RETORNA NUNCA

## Section 5

### Drivers

# ¿Qué es un driver?

- **Segun Wikipedia:** es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz -posiblemente estandarizada- para usarlo.

# ¿Qué es un driver?

- **Según Linux:** algo muy difícil de instalar si no nos sirve el que viene.

# ¿Qué es un driver?

- Segun Windows: KARMA

# ¿Qué es un driver?

- **Segun freeRTOS:** una tarea como cualquier otra pero con un propósito específico. (El de wikipedia)

# ¿ Por qué deberíamos usar un driver?

- Provee abstracción
- Brinda control de acceso al hardware (MUTEX?)
- Está optimizado para ocupar poco tiempo de CPU

Si queremos utilizar la UART para mandar información desde nuestras tareas podemos hacerlo de dos maneras:

- Sin drivers
- Con drivers

Una implementación SIN drivers podría ser la siguiente:

```
// FreeRTOS includes
#include "FreeRTOS.h"
#include "lpc17xx_uart.h"
#include "task.h"

int main( void )
{
    //_____
    //Configuro los pines y la UART
    UARTConfigStruct.Baud_rate = 115200
    UARTConfigStruct.Databits = UART_DATABIT_8;
    UART_Init(LPC_UART3, &UARTConfigStruct);
    UART_TxCmd(LPC_UART3, ENABLE);
    //_____
    // Creo las tareas
    xTaskCreate(printerTask1,"Task_1",200,NULL,1,NULL);
    xTaskCreate(printerTask2,"Task_2",200,NULL,1,NULL);

    // Inicio el scheduler
    vTaskStartScheduler();

    // Nunca llego aca
    for( ; );
}

return 0;
}
```

En donde cada task sería implementado de la siguiente manera:

```
// FreeRTOS includes
#include "FreeRTOS.h"
#include "lpc17xx_uart.h"
#include "task.h"

void printerTask1(void * pvParameters)

char myMsg[]="Hola _ esto _ es _ una _ prueba";

for(;;){
    UART_Send(LPC_UART3,myMsg,strlen(myMsg),BLOCKING);
    vTaskDelay(500/portTICK_RATE_MS);
}
}
```

En donde el delay podría ser diferente para el task1 y para el task2

¿Cuál sería el resultado?

Hola esto es una prueba  
Hola esto es una prueba

Una implementación CON drivers podría ser la siguiente:

```
// FreeRTOS includes
#include "FreeRTOS.h"
#include "uart_driver.h"
#include "task.h"

int main( void )
{
    // Creo las tareas
    xTaskCreate(UART_Driver,"UART",200, NULL,3, NULL);
    xTaskCreate(printerTask1,"Task_1",200, NULL,1, NULL);
    xTaskCreate(printerTask2,"Task_2",200,NULL,1,NULL);

    // Inicio el scheduler
    vTaskStartScheduler();

    // Nunca llego aca
    for( ; );
}

return 0;
}
```

En donde el driver podría tener una implementación de la siguiente forma:

```
xQueueHandle stdOutQueue;

void UART_Driver( void *pvParameters ){
    // Configuramos la UART
    UARTConfigStruct.Baud_rate = 115200;
    UARTConfigStruct.Databits = UART_DATABIT_8;
    UARTConfigStruct.Parity = UART_PARITY_NONE;
    UARTConfigStruct.Stopbits = UART_STOPBIT_1;

    // Inicializamos la UART
    UART_Init(LPC_UART3, &UARTConfigStruct);
    UART_TxCmd(LPC_UART3, ENABLE);

    stdOutQueue = xQueueCreate(10, sizeof(char *));

    for(;;){
        char * ptr;
        xQueueReceive(stdOutQueue,&ptr,portMAX_DELAY);
        UART_Send(LPC_UART3,ptr,strlen(ptr),BLOCKING);
    }
}
```

Y ahora, cada task sería implementado de la siguiente manera:

```
void printerTask1(void * pvParameters)

char myMsg[]="Hola _ esto _ es _ una _ prueba" ;

for(;;){
    char * ptr = &myMsg;
    xQueueSendToBack(stdoutQueue,&ptr,0);
    vTaskDelay(500/portTICK_RATE_MS);
}
```

Y el resultado sería:

```
Hola esto es una pruebaHola esto es una prueba
```

Ya que no hay posibilidad de que uno corte al otro por que la cola es secuencial, y hasta no terminar un mensaje, el driver no vuelve a pedir otro elemento.

## Section 6

### Profiling

El freeRTOS provee algunas herramientas que permiten realizar mediciones del desempeño del sistema en tiempo real. Esto generalmente es llamado *Profiling* y las herramientas son:

- Tracing
- RuntimeStats

El freeRTOS nos permite rastrear que es lo que hace el Sistema Operativo.  
Por ejemplo:

- Cuando entra a una tarea
- Cuando sale de una tarea
- Cuando se produce un cambio de contexto

Para utilizar las funciones de tracing, simplemente hay que habilitar el macro en el archivo **FreeRTOSConfig.h** y luego definir las macros como por ejemplo:

```
#define traceTASK_SWITCHED_OUT() log_event( pxCurrentTCB );
```

Podemos loggear estas acciones o sacarla por pines y verlas con un analizador lógico por ejemplo.

La herramienta de Run Time stats nos permite ver cuanto tiempo de CPU está utilizando cada tarea. Se la configura en el archivo **FreeRTOSConfig.h** y al llamarla devuelve en un buffer un reporte tipo texto.

Notar que es necesario proveer al RTOS de un timer para llevar el paso del tiempo con mayor resolucion que el **System Tick**.

La salida generada es:

IDLE	10200	80%
Touchscreen	50	<1%
LCD	450	3%
...		

Donde los numeros representan la cantidad de tiempo utilizado en la resolución del timer que se le haya configurado. Por ejemplo si se produce un incremento cada 1uS entonces la salida es en microsegundos.

## Section 7

Ejemplo de aplicación

# Ejemplo de aplicación - Holter cardíaco

- Un *Holter cardíaco* es un dispositivo médico para grabar la señal de ECG durante 24 o 48 horas.
- Debe portarlo un paciente sin que afecte su vida diaria
  - Dimensiones reducidas
  - Liviano
  - Bajo consumo

# Ejemplo de aplicación - Holter cardíaco

- Un *Holter cardíaco* es un dispositivo médico para grabar la señal de ECG durante 24 o 48 horas.
- Debe portarlo un paciente sin que afecte su vida diaria
  - Dimensiones reducidas
  - Liviano
  - Bajo consumo

# Ejemplo de aplicación - Holter cardíaco

- Un *Holter cardíaco* es un dispositivo médico para grabar la señal de ECG durante 24 o 48 horas.
- Debe portarlo un paciente sin que afecte su vida diaria
  - Dimensiones reducidas
  - Liviano
  - Bajo consumo

# Ejemplo de aplicación - Holter cardíaco

- Un *Holter cardíaco* es un dispositivo médico para grabar la señal de ECG durante 24 o 48 horas.
- Debe portarlo un paciente sin que afecte su vida diaria
  - Dimensiones reducidas
  - Liviano
  - Bajo consumo

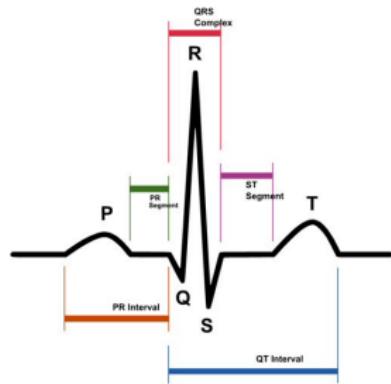
# Ejemplo de aplicación - Holter cardíaco

- Un *Holter cardíaco* es un dispositivo médico para grabar la señal de ECG durante 24 o 48 horas.
- Debe portarlo un paciente sin que afecte su vida diaria
  - Dimensiones reducidas
  - Liviano
  - Bajo consumo

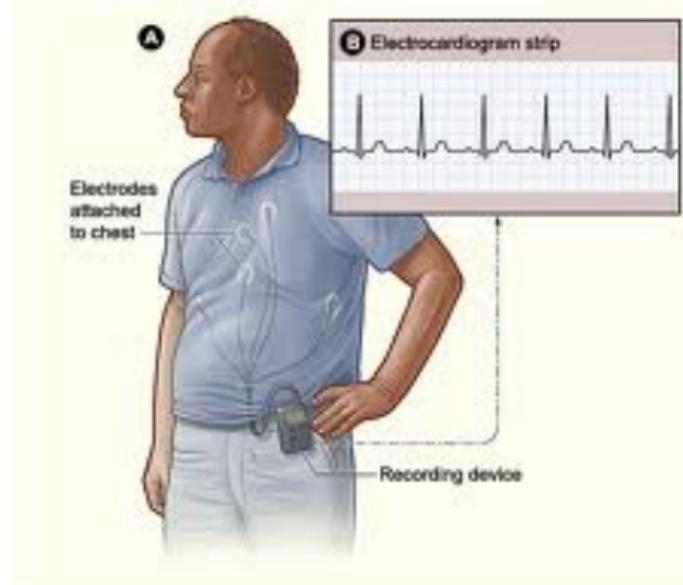
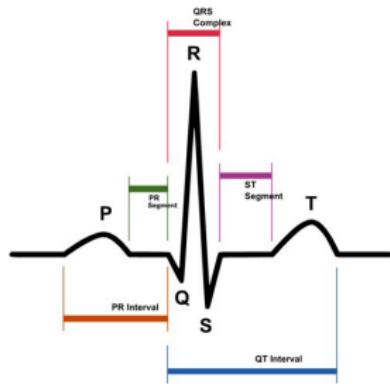
# Ejemplo de aplicación - Holter cardíaco



# Ejemplo de aplicación - Holter cardíaco



# Ejemplo de aplicación - Holter cardíaco



# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los ítems anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los ítems anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los ítems anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los ítems anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los items anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los items anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

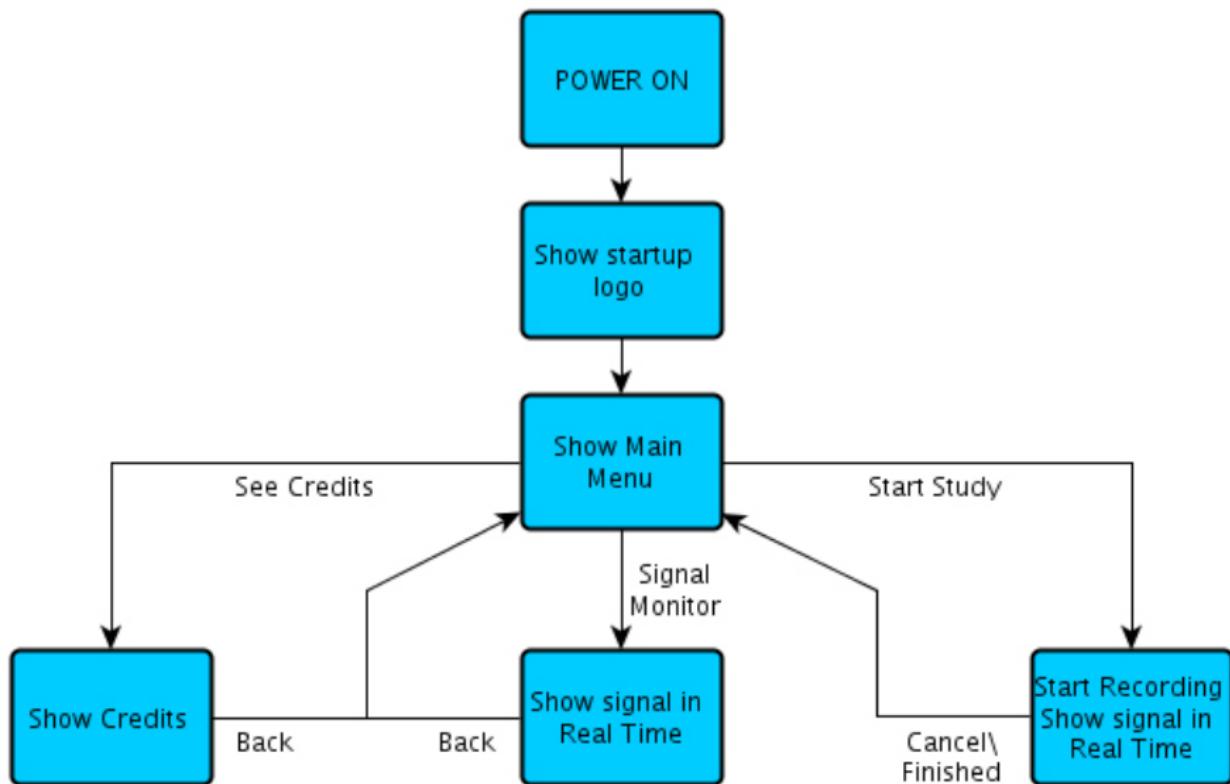
- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los items anteriores

# Holter - especificaciones técnicas

Básicamente el Holter consta de:

- Display GLCD
- Touchscreen
- Interfaz analógica (ampli de instrumentación y filtro LP)
- Tarjeta de memoria SD
- Microcontrolador LPC1768
  - Conversor A/D
  - UART
  - Funcionalidades para manejar los items anteriores

# Holter - Diagrama de estados



# Holter - Diagrama de estados

A continuación veremos como implementar estas funcionalidades con FreeRTOS ...

# Holter - GLCD

Primero veamos como controlar al GLCD...



# Holter - GLCD

El refresco del GLCD lo realiza una tarea *periódica* cada 10 Hz.  
Se basa en actualizar la RAM del controlador con el contenido de un *framebuffer*.

- Un framebuffer es un bloque de memoria reservado en la RAM del micro.
- Posee tantos bits como pixels el tenga el display
- Es un espejo de la imagen que se está mostrando...mejor dicho, lo que se está mostrando es un espejo del framebuffer.

Cómo funciona?

# Holter - GLCD

El refresco del GLCD lo realiza una tarea *periódica* cada 10 Hz.  
Se basa en actualizar la RAM del controlador con el contenido de un *framebuffer*.

- Un framebuffer es un bloque de memoria reservado en la RAM del micro.
- Posee tantos bits como pixels el tenga el display
- Es un espejo de la imagen que se está mostrando...mejor dicho, lo que se está mostrando es un espejo del framebuffer.

Cómo funciona?

# Holter - GLCD

El refresco del GLCD lo realiza una tarea *periódica* cada 10 Hz.  
Se basa en actualizar la RAM del controlador con el contenido de un *framebuffer*.

- Un framebuffer es un bloque de memoria reservado en la RAM del micro.
- Posee tantos bits como pixels el tenga el display
- Es un espejo de la imagen que se está mostrando...mejor dicho, lo que se está mostrando es un espejo del framebuffer.

Cómo funciona?

# Holter - GLCD

El refresco del GLCD lo realiza una tarea *periódica* cada 10 Hz.  
Se basa en actualizar la RAM del controlador con el contenido de un *framebuffer*.

- Un framebuffer es un bloque de memoria reservado en la RAM del micro.
- Posee tantos bits como pixels el tenga el display
- Es un espejo de la imagen que se está mostrando...mejor dicho, lo que se está mostrando es un espejo del framebuffer.

Cómo funciona?

# Holter - GLCD

El refresco del GLCD lo realiza una tarea *periódica* cada 10 Hz.  
Se basa en actualizar la RAM del controlador con el contenido de un *framebuffer*.

- Un framebuffer es un bloque de memoria reservado en la RAM del micro.
- Posee tantos bits como pixels el tenga el display
- Es un espejo de la imagen que se está mostrando...mejor dicho, lo que se está mostrando es un espejo del framebuffer.

Cómo funciona?

# Holter - GLCD

- El micro escribe directamente en el framebuffer. Es una operación muy *rápida*.
- Periódicamente, la tarea refresca el GLCD con el contenido del framebuffer. Pasa a estado activo, realiza su acción y luego se bloquea por 100ms.

Esto nos ahorra varios problemas:

- Asignando una prioridad *baja* a la tarea de refresco, aseguramos que se ejecuten otras tareas críticas.
- No dependemos de la latencia que tenga el display para escribir.
- Si a esto le sumamos usar un driver para el display, nos abstraemos 100% del hardware.

# Holter - GLCD

- El micro escribe directamente en el framebuffer. Es una operación muy *rápida*.
- Periódicamente, la tarea refresca el GLCD con el contenido del framebuffer. Pasa a estado activo, realiza su acción y luego se bloquea por 100ms.

Esto nos ahorra varios problemas:

- Asignando una prioridad *baja* a la tarea de refresco, aseguramos que se ejecuten otras tareas críticas.
- No dependemos de la latencia que tenga el display para escribir
- Si a esto le sumamos usar un driver para el display, nos abstraemos 100% del hardware.

# Holter - GLCD

- El micro escribe directamente en el framebuffer. Es una operación muy *rápida*.
- Periódicamente, la tarea refresca el GLCD con el contenido del framebuffer. Pasa a estado activo, realiza su acción y luego se bloquea por 100ms.

Esto nos ahorra varios problemas:

- Asignando una prioridad *baja* a la tarea de refresco, aseguramos que se ejecuten otras tareas críticas.
- No dependemos de la latencia que tenga el display para escribir
- Si a esto le sumamos usar un driver para el display, nos abstraemos 100% del hardware.

# Holter - GLCD

- El micro escribe directamente en el framebuffer. Es una operación muy *rápida*.
- Periódicamente, la tarea refresca el GLCD con el contenido del framebuffer. Pasa a estado activo, realiza su acción y luego se bloquea por 100ms.

Esto nos ahorra varios problemas:

- Asignando una prioridad *baja* a la tarea de refresco, aseguramos que se ejecuten otras tareas críticas.
- No dependemos de la latencia que tenga el display para escribir
- Si a esto le sumamos usar un driver para el display, nos abstraemos 100% del hardware.

# Holter - GLCD

- El micro escribe directamente en el framebuffer. Es una operación muy *rápida*.
- Periódicamente, la tarea refresca el GLCD con el contenido del framebuffer. Pasa a estado activo, realiza su acción y luego se bloquea por 100ms.

Esto nos ahorra varios problemas:

- Asignando una prioridad *baja* a la tarea de refresco, aseguramos que se ejecuten otras tareas críticas.
- No dependemos de la latencia que tenga el display para escribir
- Si a esto le sumamos usar un driver para el display, nos abstraemos 100% del hardware.

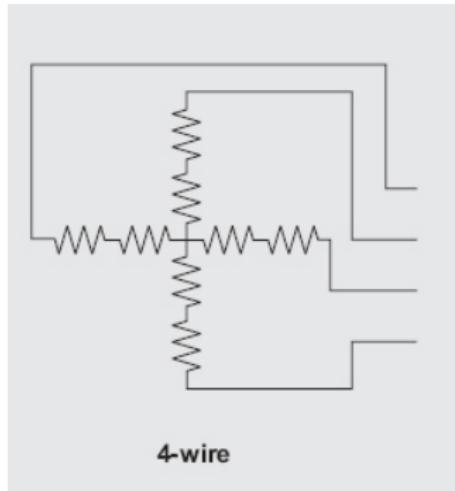
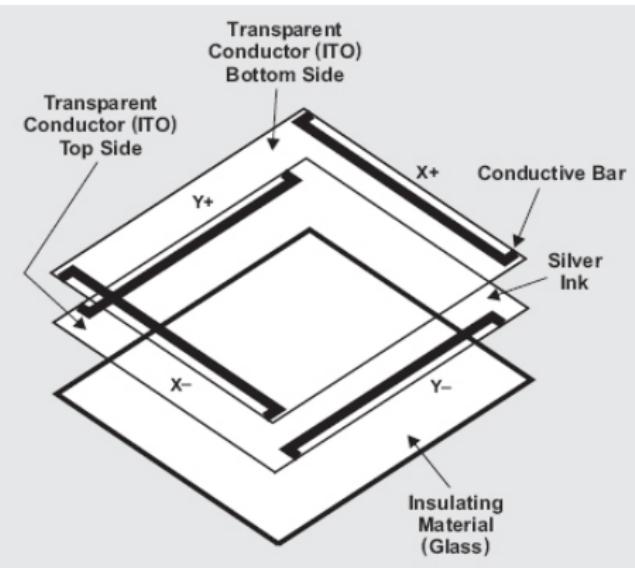
# Holter - GLCD

- El micro escribe directamente en el framebuffer. Es una operación muy *rápida*.
- Periódicamente, la tarea refresca el GLCD con el contenido del framebuffer. Pasa a estado activo, realiza su acción y luego se bloquea por 100ms.

Esto nos ahorra varios problemas:

- Asignando una prioridad *baja* a la tarea de refresco, aseguramos que se ejecuten otras tareas críticas.
- No dependemos de la latencia que tenga el display para escribir
- Si a esto le sumamos usar un driver para el display, nos abstraemos 100% del hardware.

# Holter - Touchscreen



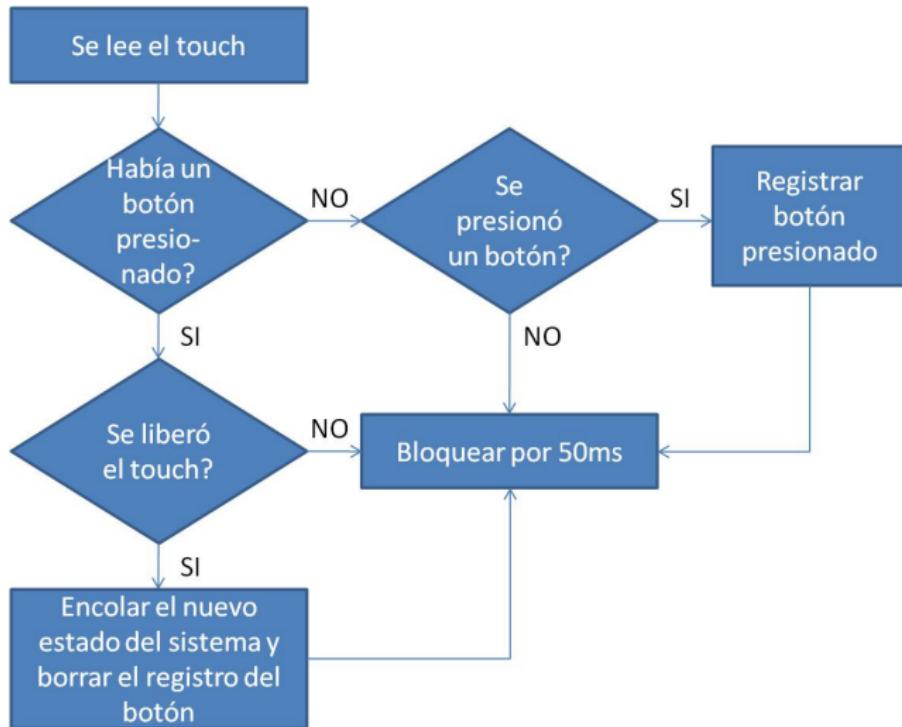
- Un touchscreen *resistivo* está compuesto de 2 capas conductoras que se tocan al aplicar presión en un punto específico.
- Se lee su estado utilizando un A/D.

# Holter - Touchscreen

- Para leer la coordenada X, se aplica tensión sobre la X y se lee la Y.
- En uno de los contactos del Y está la entrada del A/D y el otro en Hi-Z.
- Lo mismo para leer la Y. Las funciones de los pins cambian iterativamente.

# Holter - Touchscreen

Esta funcionalidad la implementamos con una tarea periódica. Se lee el estado del touch cada 50ms.



# Holter - Touchscreen

Existe otra tarea que recibe el nuevo estado en su cola, se desbloquea, y realiza el cambio de estado. Se explica más adelante.

# Tareas - Adquisición de datos

Esta tarea es claramente la tarea más importante de todo el sistema. Tiene un requerimiento de tiempo duro, y no pueden presentar corrimiento temporal. Características de la misma:

- Tiene que ejecutarse a 200 Hz, SIEMPRE ⇒ Tarea Periódica
- Tiene que ser corta (ya que se ejecuta muy seguido)
- Tiene la más alta prioridad del sistema

# Tareas - Adquisición de datos

Esta tarea es claramente la tarea más importante de todo el sistema. Tiene un requerimiento de tiempo duro, y no pueden presentar corrimiento temporal. Características de la misma:

- Tiene que ejecutarse a 200 Hz, SIEMPRE ⇒ Tarea Periódica
- Tiene que ser corta (ya que se ejecuta muy seguido)
- Tiene la **más alta** prioridad del sistema

# Tareas - Adquisición de datos

Esta tarea es claramente la tarea más importante de todo el sistema. Tiene un requerimiento de tiempo duro, y no pueden presentar corrimiento temporal. Características de la misma:

- Tiene que ejecutarse a 200 Hz, SIEMPRE ⇒ Tarea Periódica
- Tiene que ser corta (ya que se ejecuta muy seguido)
- Tiene la **más alta** prioridad del sistema

# Tareas - Adquisición de datos

Esta tarea es claramente la tarea más importante de todo el sistema. Tiene un requerimiento de tiempo duro, y no pueden presentar corrimiento temporal. Características de la misma:

- Tiene que ejecutarse a 200 Hz, SIEMPRE ⇒ Tarea Periódica
- Tiene que ser corta (ya que se ejecuta muy seguido)
- Tiene la **más alta** prioridad del sistema

# Tareas - Adquisición de datos

Esta tarea es claramente la tarea más importante de todo el sistema. Tiene un requerimiento de tiempo duro, y no pueden presentar corrimiento temporal. Características de la misma:

- Tiene que ejecutarse a 200 Hz, SIEMPRE ⇒ Tarea Periódica
- Tiene que ser corta (ya que se ejecuta muy seguido)
- Tiene la **más alta** prioridad del sistema

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

¿Que tiene que hacer esta tarea?

- Tomar un dato del ADC
- Filtrar el dato de los 50Hz de linea (operación sobre el array muy simple)
- Distribuir el dato a las tareas que lo necesiten
  - Memoria SD
  - Dibujar en pantalla
  - Detector de QRS
  - Comunicación con PC - UART

# Tareas - Adquisición de datos

## ¿Cómo lo distribuyo?

**Exacto!** usando COLAS.

Estas colas, desbloquearan tareas por lo que las tareas que veremos a continuación serán tareas que estarán bloqueadas hasta recibir datos en sus colas ¿Recuerdan el driver de UART?

## Tareas - Adquisición de datos

# ¿Cómo lo distribuyo?

**Exacto!** usando COLAS.

Estas colas, desbloquearan tareas por lo que las tareas que veremos a continuación serán tareas que estarán bloqueadas hasta recibir datos en sus colas ¿Recuerdan el driver de UART?

# Tareas - Grabación en memoria SD

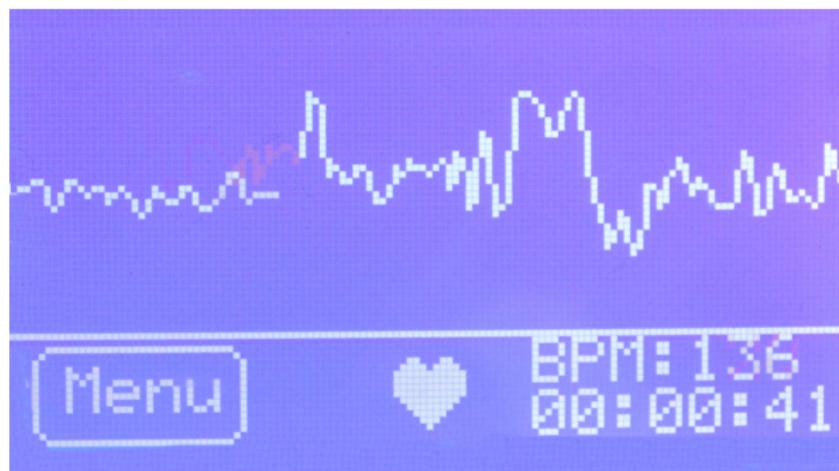
Esta tarea deberá tener un buffer que se irá completando a medida que lleguen datos.

Cuando estos lleguen, los grabaremos en la memoria. Esto se debe a que hay un mínimo de bloque que se puede grabar y si no lo utilizamos completo desperdiciaríamos demasiado tiempo. (El grabado es un proceso lento)



# Tareas - Dibujar en pantalla

Esta tarea se encarga de dibujar la señal adquirida interpolando con puntos. También deberá mostrar el tiempo del estudio y los latidos por minutos calculados.



# Tareas - Dibujar en pantalla

Pero, ¿dónde dibuja?

En la memoria de video, Exacto!

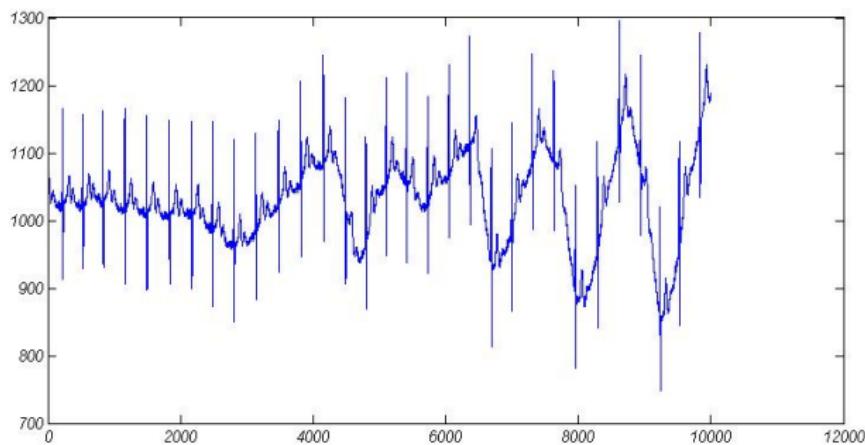
# Tareas - Dibujar en pantalla

Pero, ¿dónde dibuja?

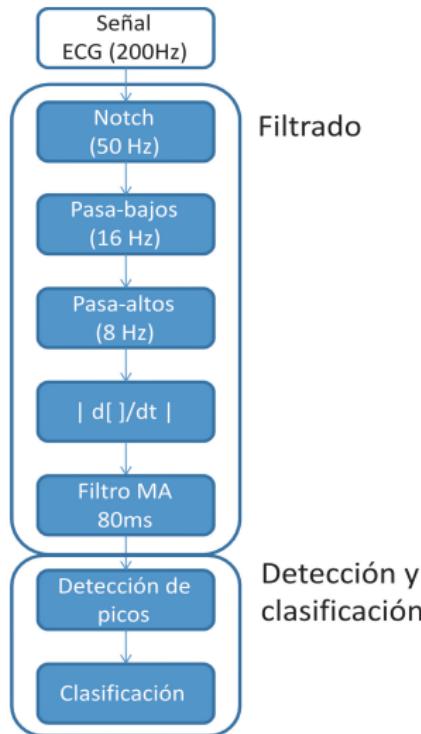
**En la memoria de video, Exacto!**

# Tareas - Detección del QRS

La señal de ECG puede ser muy variable y detectar los complejos no es un proceso tan sencillo...



# Tareas - Detección del QRS



- Para detectarlos se utiliza una implementación del *algoritmo de Hamilton*.
- Está pensado para implementarlo en micros. Usa aritmética entera.
- Va almacenando la señal en un buffer. Al llegar una nueva muestra, revisa las anteriores y detecta si hubo o no un QRS.
- Si lo detecta, devuelve cuántas muestras atrás se detectó. Tiene una latencia de aprox. 200ms.

# Tareas - Comunicación con PC

Si recordamos el driver de la UART que usaba colas, esta tarea lo único que debe hacer es adaptar el dato que recibe por la cola para ser enviado por la UART y enviarlo a la cola del driver.



# Sincronización y control de estados

Una buena manera de organizar todo esto es con una tarea que funcione como una máquina de estados. La misma funcionara con colas que le diran a que estado cambiar. Cada vez que se deba cambiar de estados, algunas tareas se destruirán y otras se crearán.

Estos cambios de estado los pueden producir:

- Eventos del touchscreen (botones)
- Alarmas de fin de estudio
- Errores

# Sincronización y control de estados

Una buena manera de organizar todo esto es con una tarea que funcione como una máquina de estados. La misma funcionara con colas que le diran a que estado cambiar. Cada vez que se deba cambiar de estados, algunas tareas se destruirán y otras se crearán.

Estos cambios de estado los pueden producir:

- Eventos del touchscreen (botones)
- Alarmas de fin de estudio
- Errores

# Sincronización y control de estados

Una buena manera de organizar todo esto es con una tarea que funcione como una máquina de estados. La misma funcionara con colas que le diran a que estado cambiar. Cada vez que se deba cambiar de estados, algunas tareas se destruirán y otras se crearán.

Estos cambios de estado los pueden producir:

- Eventos del touchscreen (botones)
- Alarmas de fin de estudio
- Errores

# Sincronización y control de estados

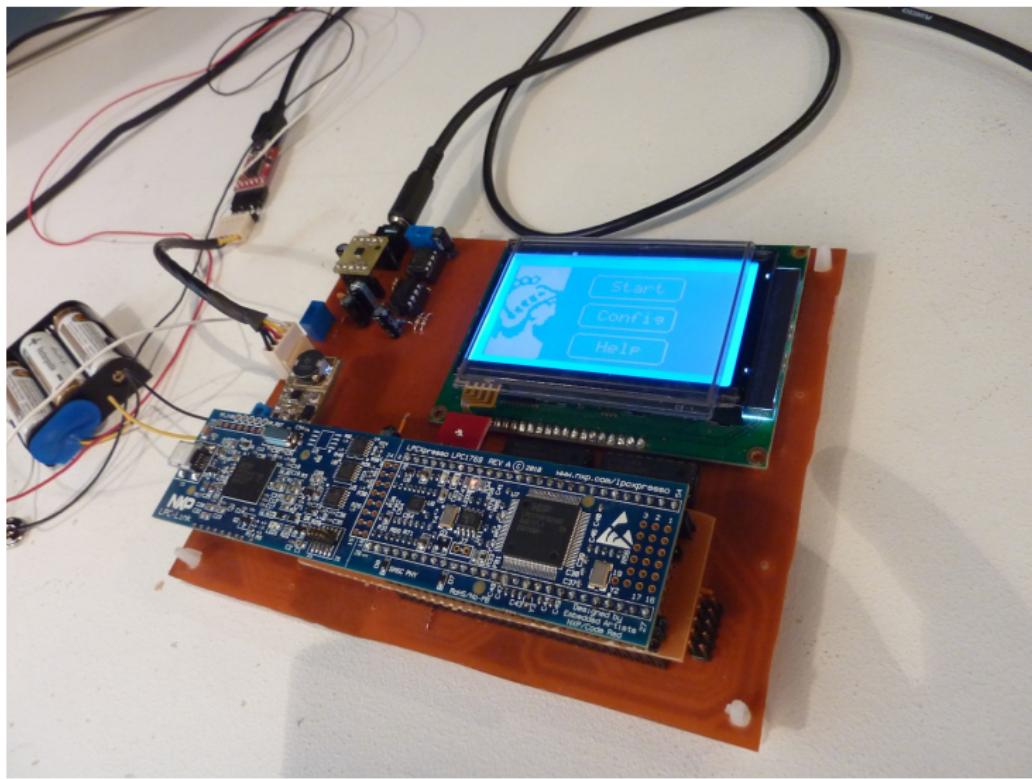
Una buena manera de organizar todo esto es con una tarea que funcione como una máquina de estados. La misma funcionara con colas que le diran a que estado cambiar. Cada vez que se deba cambiar de estados, algunas tareas se destruirán y otras se crearán.

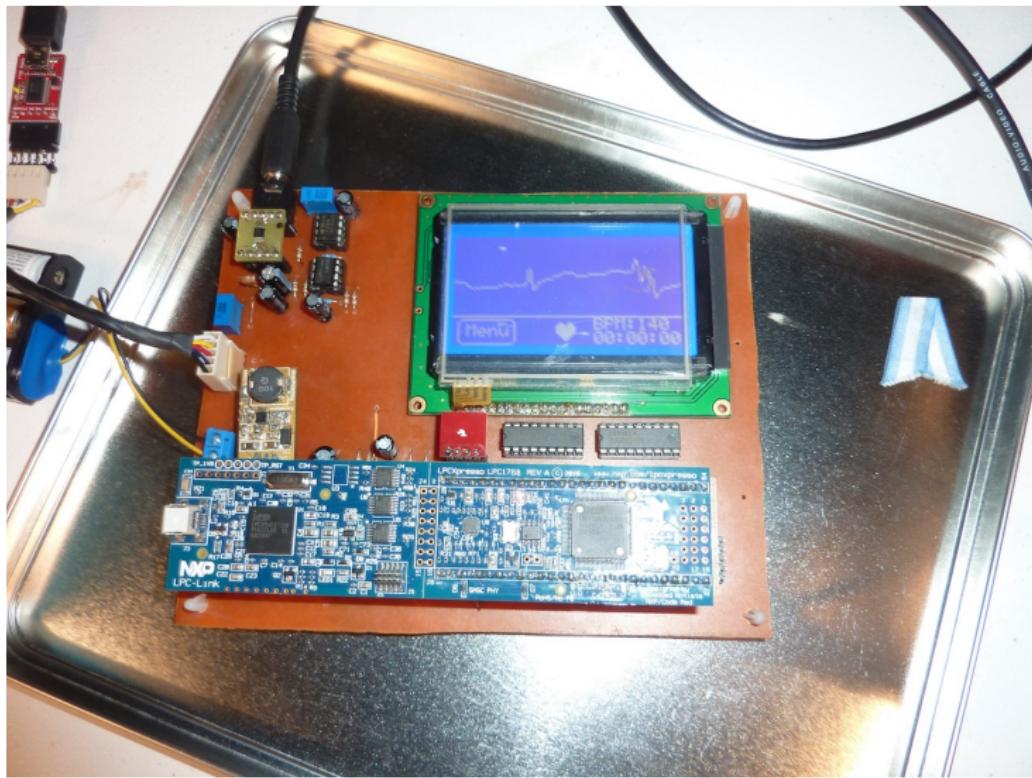
Estos cambios de estado los pueden producir:

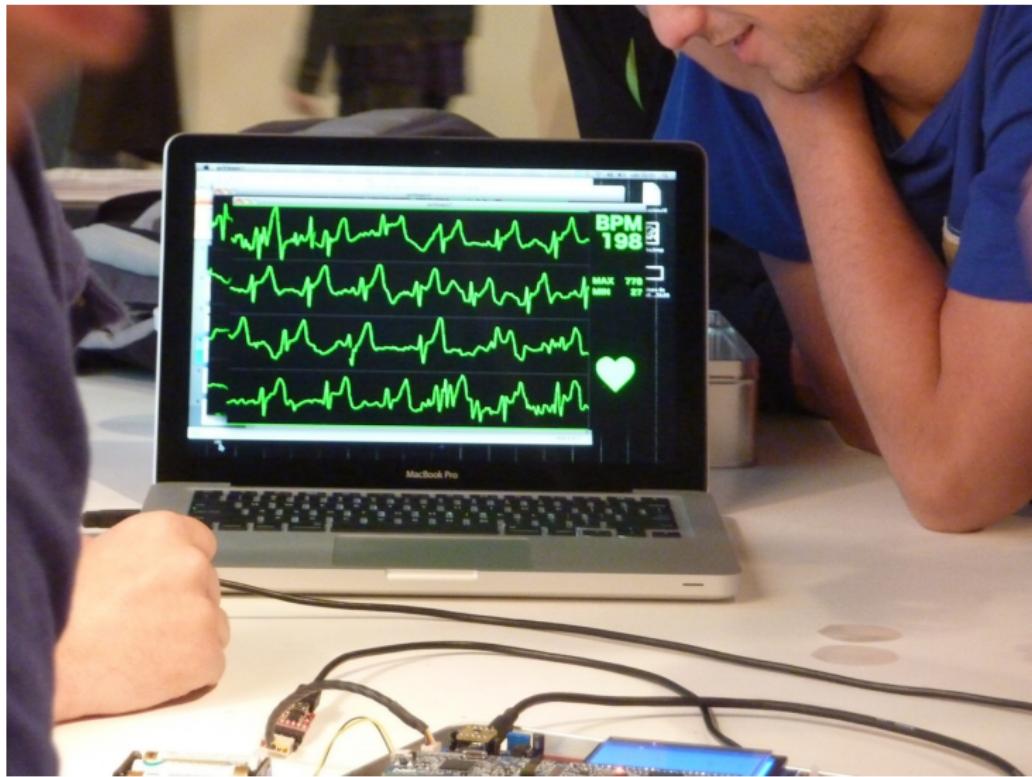
- Eventos del touchscreen (botones)
- Alarmas de fin de estudio
- Errores

# Conclusión

¿Y todo esto funciona?







# Conclusión

¿Cómo implementaría el sistema para que cada vez que se detecte un segmento QRS se muestre un corazón por 2 segundos y luego desaparezca?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?

# Conclusión

Si tenemos un RTOS entonces creamos una tarea que haga lo siguiente:

- Dibuje un corazón en pantalla (o en memoria?)
- Espere 2 segundos (bloqueado, por supuesto)
- Borre el corazón que dibujó
- Y después?



¿Y si no teníamos un RTOS?



© www.123RF.com

# Malabarismo con TIMERS!

¿Dudas?

¿Dudas? ¿Consultas?



Gracias

Muchas gracias!

# Referencias

- Barry R. Using the FreeRTOS Real Time Kernel. NXP LPC17xx Edition.
- Qing L, Yao C. Real-Time Concepts for Embedded Systems. CMP Books 2003.
- Free RTOS official web. <http://www.freertos.org/>
- E.P. Limiteds Open Source ECG Analysis Software.  
<http://www.eplimited.com/>
- Hamilton PS, Tompkins WJ. Quantitative investigation of QRS detection rules using the MIT/BIH arrhythmia database. IEEE Trans. Biomed. Eng., BME-33(12):1157-1165, 1986.
- MIT/BIH arrhythmia database.  
<http://www.physionet.org/physiobank/database/mitdb/>