FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Sistemas Embebidos
El Diseño en Ingeniería Electrónica

Laboratorio de Sistemas Embebidos

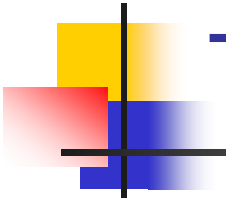
http://laboratorios.fi.uba.ar/lse/
seminario-embebidos@googlegroups.com

66.48 & 66.66 Seminario de Electrónica: Sistemas Embebidos

Curso de Posgrado: Introducción a los Sistemas Embebidos

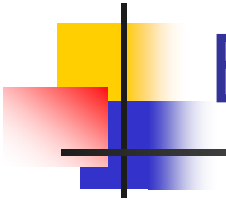
Ingeniería en Electrónica – FI – UBA

Buenos Aires, 13 de Marzo de 2012



Temario

- Introducción
- Estado del arte
- Problemática general
- Criterios de diseño
- Casos típicos de estudio
- Un ejemplo de aplicación

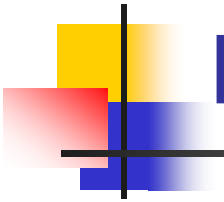


Estado del Arte, ¿cuál es?

- Lineamientos a seguir:
  - KISS “Keep It Simple, Stupid”
  - DFE “Design for Excellence”
  - Documentarse debidamente antes de comenzar el diseño

KISS, ¿qué significa?

- KISS es un acronismo en inglés que significa:
  - "Keep It Simple, Stupid" (Mantenlo simple, estúpido)
- Una acepción menos chocante es:
  - "Keep It Short and Simple" (Mantenlo corto y simple)
- Comenzó a usarse en EEUU en los años 60 (en relación con el proyecto Apollo)
- Deriva del "Principio de Economía" de William of Ockham (siglo XIV DC) y variantes formuladas por Leonardo da Vinci, Isaac Newton, Albert Einstein y otros



KISS, ¿qué se procura?

- Afirma que la simplicidad es la clave del éxito de un diseño en ingeniería
- En el desarrollo de sistemas complejos en ingeniería debemos:
  - Desarrollar empleando partes sencillas, comprensibles que redundará en errores de fácil detección y corrección.
  - Rechazar lo rebuscado e innecesario
- En otras palabras advierte al diseñador para que en su labor no "compre" problemas sino que "venda" soluciones

DFE, ¿qué significa?

- DFE "Design For Excellence":
  - Manufacture and Assembly
  - Reliability
  - Testing and Service
  - Disassembly and Reassembly
  - Use and Operability
  - Green, Environment and Recycling
  - Quality and Cost
  - Logistic
  - Inspection and International
  - Etc., etc.

DFE, ¿qué se procura?

- Diseñar para la excelencia no implica la implementación de todos y cada uno de los ítems listados, ya que cualquier actividad de diseño estará fuertemente condicionado por dos factores:
  - La idiosincrasia tanto del diseñador como del medio en que éste se desempeña
  - El contexto en que se lleve a cabo el diseño propiamente dicho

Documentarse antes de …, ¿qué significa?

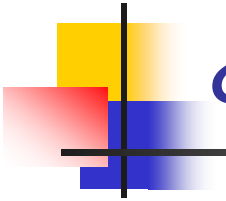
- Seguramente Ud. no es el primero que intenta resolver el problema que enfrenta, por tal motivo es recomendable que recopile toda documentación referida al diseño que está por encarar, a las técnicas y/o herramientas que pueden serle útiles para el diseño, etc., etc.; como por ejemplo:
  - Hojas de Datos (Fe de Erratas)
  - Notas de Aplicación
  - Ejercicios o Ejemplos de Diseño
  - Manuales de Usuario
  - Manuales de Referencia Técnica
  - Etc., etc.

Documentarse antes de …, ¿qué se procura?

- Aproveche las facilidades que ofrecen las vías de comunicación actuales para la búsqueda de información
- Procure encarar la búsqueda con sentido común y criterio
- Recuerde que la búsqueda en si misma es un medio y no un fin
- Lea, analice y clasifique toda la documentación recopilada
- Durante la etapa de diseño saque provecho de la información recopilada "aprendiendo del trabajo de los demás"

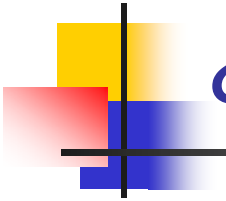
Problemática General

- Debemos detenernos a analizar lo siguiente:
  - Metodología de trabajo
  - Diseño electrónico Analógico/Digital (Hard & Soft)
  - Dibujo del Impreso
  - Componentes
  - Producto
  - Fabricación del circuito impreso
  - Fabricación del Producto
  - Etc., etc.



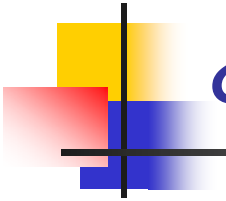
¿Porqué nos preocupa el tiempo?

- Existen gran cantidad de sistemas con restricciones temporales:
  - **Sistemas de control**: el proceso de control implica mediciones, cálculo y actuación, a resolver en un periodo de muestreo (mS)
    - Si esto no ocurre, la planta queda a lazo abierto, pudiendo tener consecuencias catastróficas
  - **Sistemas multimedia**: un reproductor de DVD debe leer, decodificar y presentar en pantalla una imagen cada 20 mS
    - Si una imagen no se llegase a visualizar esporádicamente no pasará nada, salvo una pequeña molestia por parte del usuario
  - **Simuladores interactivos**: en un videojuego es necesario leer el estado del joystick, calcular la siguiente escena y representarla lo suficientemente rápido como para que el movimiento sea fluido
    - Si no es posible refrescar la imagen en menos de 20 mS el usuario no se sentirá cómodo, obviamente el "congelamiento" esporádico de la imagen no es catastrófico



¿Porqué nos preocupa el tiempo?

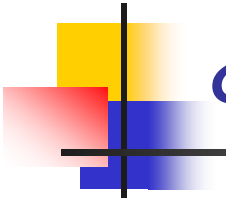
- Resumiendo habrá aplicaciones en las cuales es necesario:
  - Garantizar los tiempos de respuesta (Estricto o **Hard Real Time**)
  - Normalmente cumplir restricciones temporales (No Estricto o **Soft Real Time**)
- En un sistema de tiempo real la corrección del resultado depende tanto de su validez lógica como del instante en que se produce:
  - Asegurar **determinismo** & **tiempo de respuesta**
  - P/tiempo de ejecución: contemplar el caso más desfavorable
  - Atención: Tiempo Real no es igual a Rápido (ver caso por caso)



¿Cómo administramos el tiempo?

- En los sistemas de control existen cuatro niveles jerárquicos de software:
  - Adquisición de datos / Actuadores
  - Algoritmos de control (PID)
  - Algoritmos de supervisión (trayectorias)
  - Interfaz de usuario, registro de datos, etc.

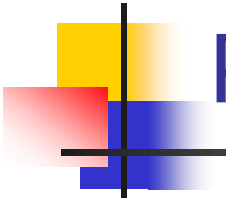
- Un programa secuencial se divide en funciones que se ejecutan según un orden preestablecido, mientras que un sistema en tiempo real se divide en tareas que se ejecutan en paralelo
  - La ejecución en paralelo se consigue como la sucesión rápida de actividades secuenciales



¿Cómo administramos el tiempo?

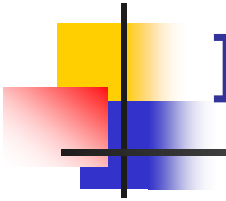
- Contamos con varias técnicas para ejecutar tareas en paralelo:
  - **Procesamiento secuencial** (Lazo de barrido o scan)
  - **Interrupciones** (Background/Foreground)
  - **Multitarea cooperativo**
  - **Multitarea expropiativo** (preemptive)

- Los más usados por los principiantes son el **Procesamiento Secuencial** (lazo perpetuo de barrido de tareas) e **Interrupciones** (una o más tareas asociadas a las interrupciones en 1° plano y otra tarea en 2° plano encargada de ejecutar un lazo perpetuo dentro del cual puede implementarse un procesamiento secuencial, la interfaz de usuario o no hacerse nada). En conjunto con las **máquinas de estado finito** constituyen una de las soluciones posibles de lo que se denomina **programación gobernada por eventos** (event driven programming)



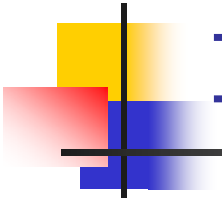
Procesamiento Secuencial

- Ventajas
  - Muy fácil de implantar
  - Las tareas pueden compartir información sin problemas
  - No hay sobrecarga por cambio de contexto
- Desventajas
  - La **latencia** puede ser grande
  - Es complejo implantar tareas con distintos periodos de muestreo
- Conclusión
  - Sólo es válido para sistemas muy sencillos



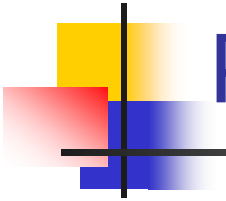
Interrupciones (Foreground/Background)

- Ventajas
  - Baja latencia
  - No necesita software adicional (núcleo S.O.)
- Desventajas
  - Necesita soporte hardware
  - Sólo válido si hay una interrupción para cada tarea
  - Complejo de programar
- A tener en cuenta:
  - Si las tareas son largas la latencia aumenta (pasar a 2° plano)
  - Compartir datos entre tareas puede ser problemático (deshab. IRQ´s)
  - Cuando la tarea de interrupción produce datos en ráfagas o el procesamiento de datos no finaliza entre datos y da tiempo a procesar todos los datos (utilizar colas FIFO p/vincular tareas de 1° y 2° plano)



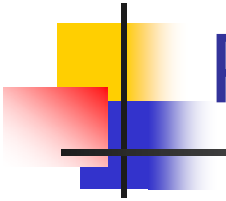
Interrupciones (Foreground/Background)

- Problemas:
  - La latencia depende del tiempo de ejecución de tareas de 2° plano
  - Si las tareas tardan en ejecutarse se pueden desbordar colas
  - Difícil de gestionar las prioridades
  - Se pierde tiempo verificando si hay datos en las colas
- Solución:
  - Añadir una tarea (**planificador** o scheduler) quien decide qué tarea de 2° plano se debe ejecutar en c/momento basándose en:
    - Cuál tiene datos para ejecutarse
    - Cuál es la más prioritaria
  - Efecto "colateral":
    - Pueden existir tareas de 2o plano no asociadas a interrupción
    - Éstas pueden ejecutarse periódicamente o cuando no haya que hacer



Planificadores o Schedulers

- Existen dos tipos de planificadores o schedulers:
  - Cooperativos o No Expropiativos (Non preemptive)
    - Cada tarea debe llamar al planificador periódicamente o cuando no tenga nada que hacer para ceder la CPU (yield)
    - El planificador decide si hay una tarea más prioritaria que ejecutar:
      - Si: ejecuta un cambio de contexto y cede la CPU a la siguiente tarea
      - No: continúa con la tarea anterior (si ésta no ha terminado)
  - Expropiativos (Preemptive)
    - Se ejecuta periódicamente (disparado por interrupción)
    - Si hay que ejecutar una tarea más prioritaria que la actual, expropia la CPU a la tarea que se esté ejecutando y se la cede (cambio de contexto mediante)



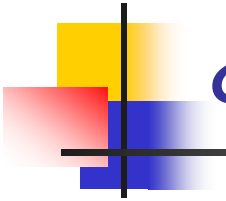
Planificadores o Schedulers

- Problemas de los planificadores cooperativos:
  - La latencia depende del tiempo empleado por las tareas entre dos cesiones consecutivas de la CPU
  - Difícil de garantizar las temporizaciones

- Metodología de programación con planificadores cooperativos:
  - Hay que llamar al planificador para ceder la CPU frecuentemente:
    - En cada iteración de un lazo largo
    - Intercaladas entre cálculos complejos
  - Si no se cede la CPU mientras se accede al recurso compartido no pasa nada, mientras que en los expropiativos:
    - Inhabilitar las interrupciones (atención con aumento de latencia)
    - Inhabilitar los cambios de tareas de 2° plano
    - Lazos de espera => Semáforos

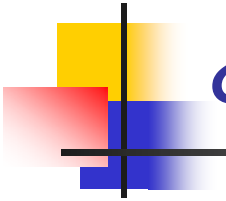
Metodología de Trabajo

- Optamos por el más usado, simple y seguro
- Recomendaciones:
  - Procure aprender del método
  - Procure adaptarlo a su gusto
  - Si no esta conforme con el método:
    - Genere su propio método, pero use uno, pues:
      - Sin método cada diseño nos obliga a comenzar de cero



¿Cuál es el método?

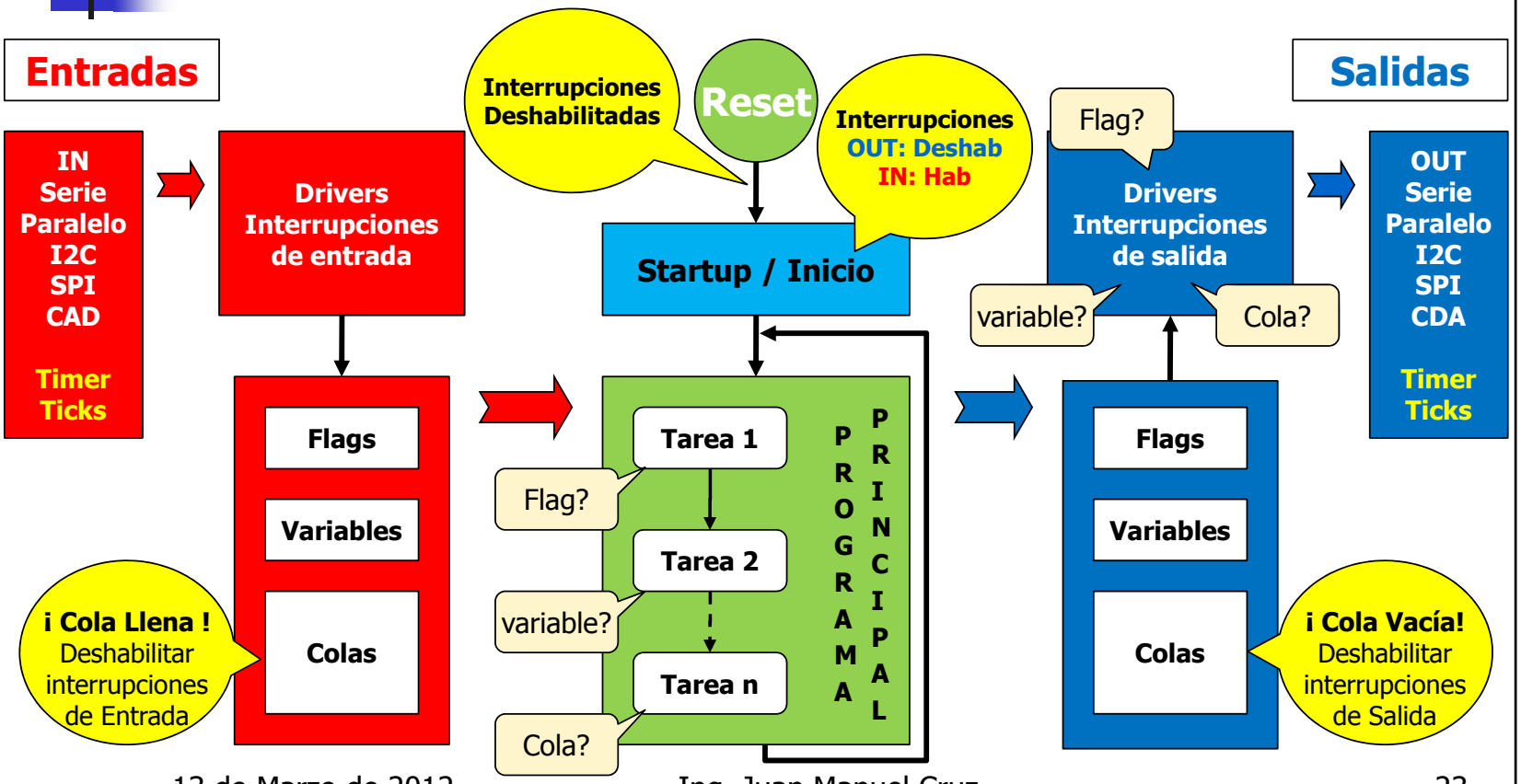
- El método más usado simple y seguro para desarrollar aplicaciones con micro consiste en fraccionar la solución en módulos simples:
  - Startup / Inicio => Inicializaciones básicas del micro
  - Programa Principal => Iteración perpetua de Tareas (algoritmos de control)
  - Manejadores / Drivers de Entrada / Salida => Interacción con el mundo exterior (atención de eventos y sincronismos)



¿En qué consiste el método?

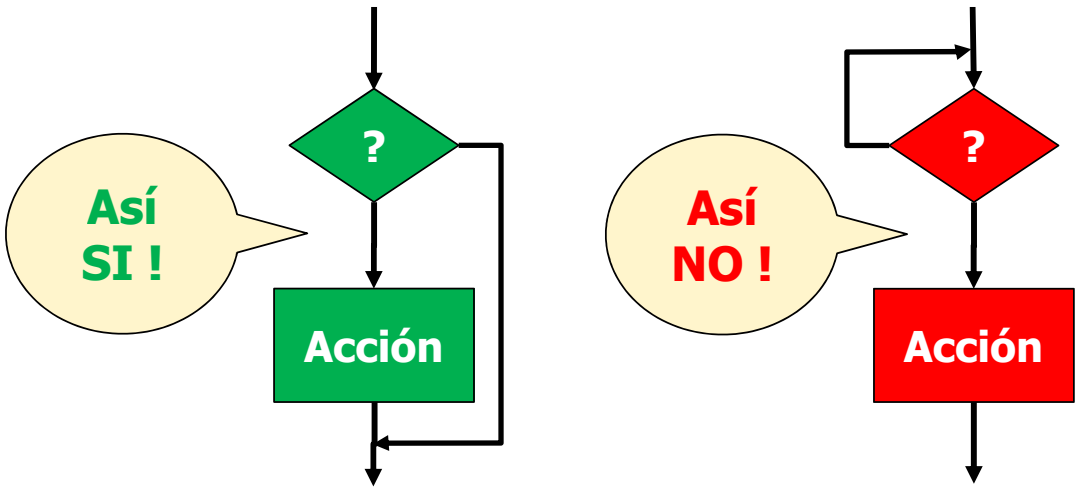
- Comunicar los módulos simples mediante:
  - Flags / Semáforos
  - Variables
  - Colas
- Garantizar que ningún módulo se apropie de la CPU (comportamiento comunitario)

Método de Diseño (1 de 2)



Método de Diseño (2 de 2)

- Garantizar que ningún módulo se apropie de la CPU (comportamiento comunitario o no bloqueante)



Criterios de diseño

- Como principiantes debemos asegurar que:
  - Tcpu-pp ≤ 1/2 Fmax Entrada a detectar/Salida a generar❶
    ≤1000 Tinstrucción ❷
    ≤ Tick-mín ❷
    ≤ 60~70% ∑ (Tcpu-driver/tarea) ❸
  - Tcpu-driver/tarea ≤ Tcpu-pp / 10 ❷

❶ Shanon (Teorema Muestreo)
❷ JMC (Experiencia de desarrollador)
❸ Mamá de JMC (Experiencia de modista)

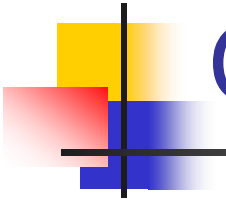
Casos típicos de estudio

- MCS-51 ejecuta ~ 1MIPS con Fcristal = 12MHz & DIVclock = 12 (CPU de la familia Intel original)

  - Tinstrucción ≈ 1µS Tcpu-pp ≤ 1mS
  - Tick-mín ≥ 1mS Tcpu-driver/tarea ≤ 100µS

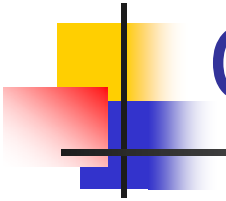
```
unsigned char data timerTickUChar;          // 0 a 255mS
unsigned int data timerTickUInt;            // 0 a 65.535mS
unsigned long int data timerTickULongInt; // 0 a 4.294.967.295mS

if (!timerTickUXxx)
        timerTickUXxx--;
```



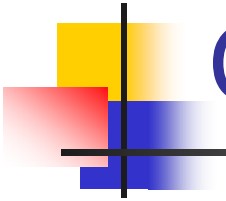
Criterios de diseño

- En las rutinas de atención de interrupción (**ISR**) sólo se hará lo estrictamente necesario para **atender al hardware**. Por lo que es necesaria una comunicación entre las ISR´s (1° plano) y las tareas en 2° plano
  - Pude haber **problemas de coherencia** de datos si la tarea de 2° plano usa **datos compartidos** de una manera no **atómica**
    - Una parte de programa es **atómica** si **no puede ser interrumpida**
    - Usar facilidades del lenguaje & compilador (p./ej: en C **volatile**)
  - Existen varias técnicas para no inhabilitar las interrupciones, entre las cuales la más segura es el uso de colas circulares
  - En caso de tratarse de flags lo ideal es que las tareas productoras los fuercen a "1" y las consumidoras los lean y los fuercen a "0"



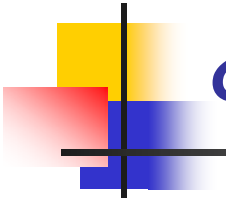
Criterios de diseño

- El uso de máquinas de estado facilitará la resolución tanto en las tareas de 1° como en las de 2° plano
- Las máquinas de estado finito permiten la fragmentación de problemas de naturaleza secuencial en una secuencia de acciones/actividades gobernadas tanto por eventos como por sincronismos
- Las máquinas de estado finito son una de las soluciones posibles de lo que se denomina programación gobernada por eventos (event driven programming)



Criterios de diseño

- Puede recurrirse a un planificador o scheduler para administrar las tareas en 2° plano del tipo:
  - Lazo de barrido
  - Cola de funciones sin prioridades
  - Cola de funciones con prioridades
- Referencia bibliográfica:
  - Sistemas Empotrados en Tiempo Real - José Daniel Muñoz Frías (descarga gratuita en: http://www.lulu.com/product/ebook/sistemas-empotrados-en-tiempo-real/17496389?productTrackingContext=product_view/more_by_author/right/1)

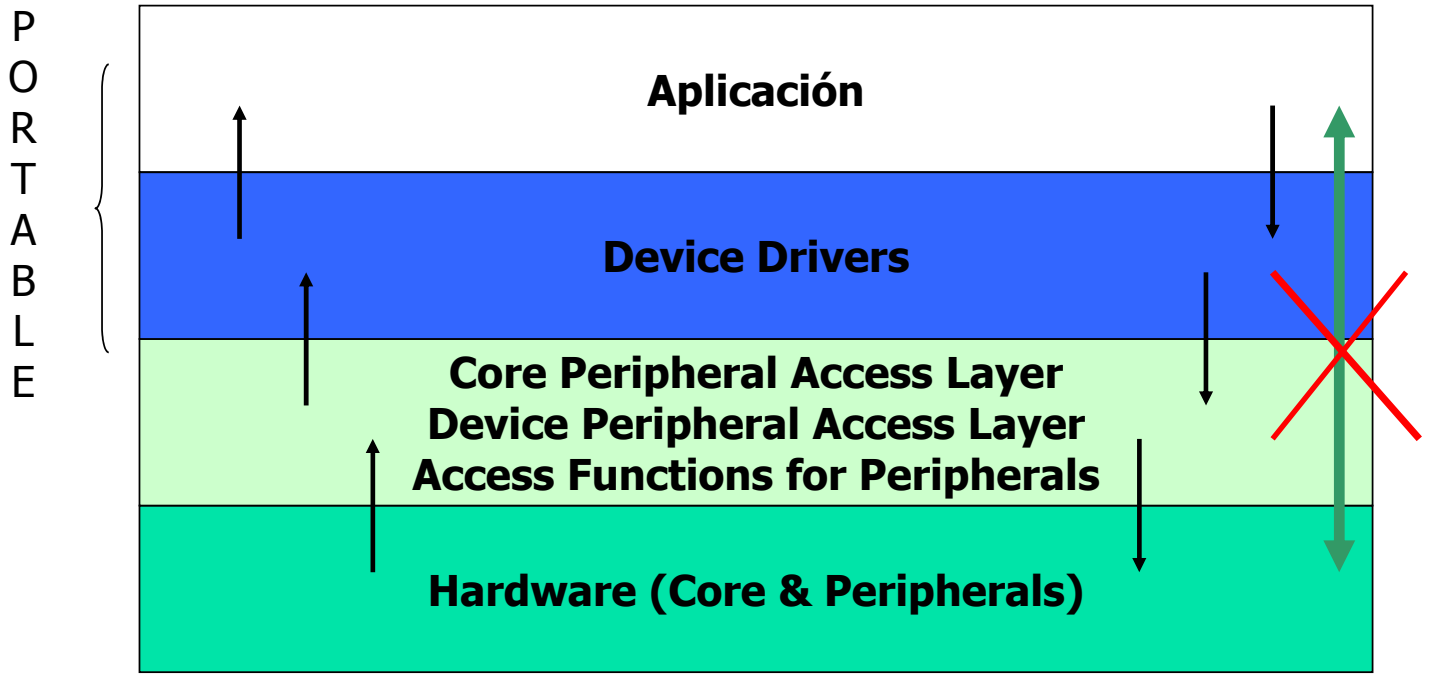


¿Cómo mejorar el método?

- Para mejorar el método es conveniente experimentar e incorporar
  - Nuevas tecnologías y arquitecturas
  - Lenguajes de programación de alto nivel (C, C++, Java, etc.)
  - Técnicas de Modelado de software (objetos, eventos, etc.)
  - Herramientas de Ingeniería del Software
  - Sistemas Operativos de Tiempo Real (RTOS)
  - Nuevas conectividades y dispositivos de conversión A/D y D/A
  - Modularizar y estructurar la solución en capas de software:
    - Capa de acceso al Hardware
    - Capa de Manejadores de Dispositivos
    - Capa de Aplicación

¿Cómo mejorar el método?

- Dividir la solución en capas aporta portabilidad



Conclusiones

- Cada diseño tiene una **"solución adecuada"**
  - Pues cualquier actividad de diseño estará fuertemente condicionado por dos factores:
    - La idiosincrasia tanto del diseñador como del medio en que éste se desempeña
    - El contexto en que se lleve a cabo el diseño propiamente dicho
  - Esta terminología es de aplicación en países periféricos como el nuestro con condiciones desfavorables para el desarrollo de la industria electrónica local (baja escala de consumo del mercado local, dificultades para exportar, inercia en incorporar innovaciones tecnológicas, etc.)
- Cada diseño tiene una **"solución adecuada"**