Aplicaciones de uso de software – ¿ El uC elegido cuenta con recursos suficientes ... ?

Información relevante

4to Congreso Virtual de Microcontroladores y sus Aplicaciones

Disponemos con una buena variedad de Áreas Temáticas y esperamos contar con vuestra asistencia y una muestra de sus trabajos a efectos de compartir con toda nuestra comunidad. Además de los trabajos contaremos con Seminarios Web en el transcurso del Congreso que versarán sobre los temas que nos interesan

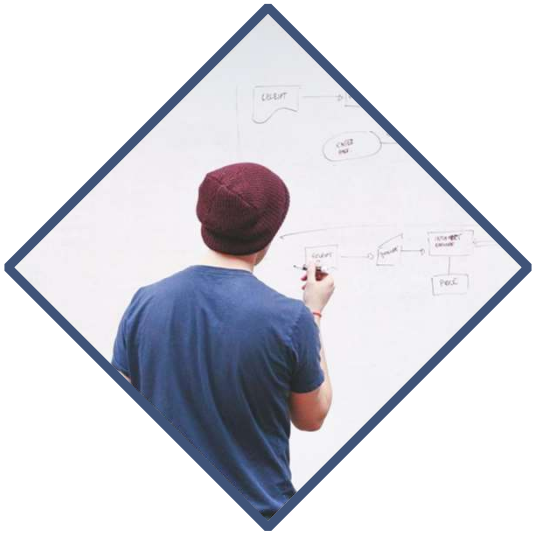
Taller de Sistemas Embebidos

Asignatura correspondiente a la **actualización 2023** del Plan de Estudios 2020 y resoluciones modificatorias, de Ingeniería Electrónica de FIUBA

Más información . . .

. . . sobre el **4to Congreso** . . . https://www.cvm.utn.edu.ar/

. . . sobre el **Taller de Sistemas Embebidos** . . . https://campusgrado.fi.uba.ar/course/view.php?id=1217



¡Hola!

Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

1

Contexto

Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Taller de Sistemas Embebidos

Segundo Ciclo

Asignatura obligatoria

3er año (6° cuatrimestre)

6 horas semanales (2 encuentros)

Objetivos & Contenidos

- (TA134) Taller de Sistemas Embebidos
  https://campusgrado.fi.uba.ar/course/view.php?id=1217

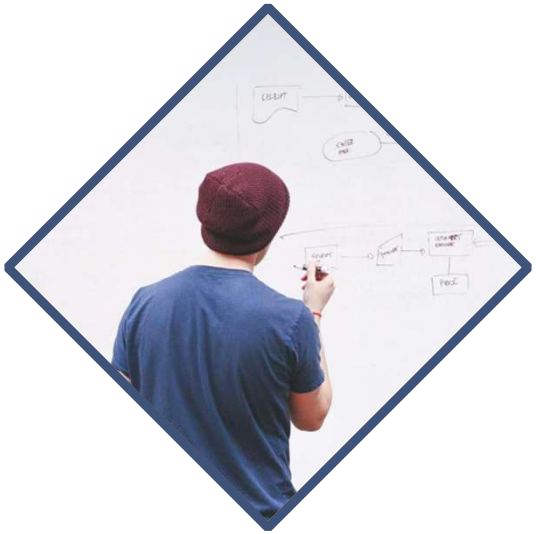
  - Objetivos:
    - Elaboración de un **Proyecto** de Sistemas Embebidos (**Intermedio**), junto con el diseño y cálculo, …
  - Contenidos:
    - Microcontroladores utilizados en Sistemas Embebidos. Introducción a la arquitectura de Microcontroladores de 32 bit o superior y sus componentes básicos, …

Objetivos & Contenidos

- Modelo del programador, modos de operación, mapa de memoria, registros, Stack, FPU, core peripherals, …
- Estrategias de control de periféricos, …
- Conversión A/D y D/A, …
- Metodologías de diseño, buenas prácticas, uso de repositorios, …
- Programación en lenguaje C, estilo de codificación, …
- Programación Modular, Bare Metal (sin Sistema Operativo), Gobernada por Eventos, Máquinas/Diagramas de Estado, …
- Técnicas de verificación y validación, On chip debugging, …

Proyecto Intermedio

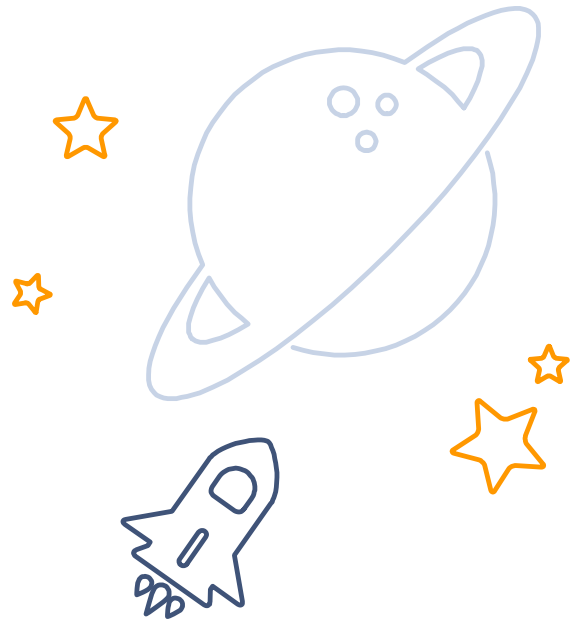
Un enfoque centrado en la práctica propia de la carrera más que en el desarrollo teórico disciplinar, con eje en la participación de las y los estudiantes



¡Hola!

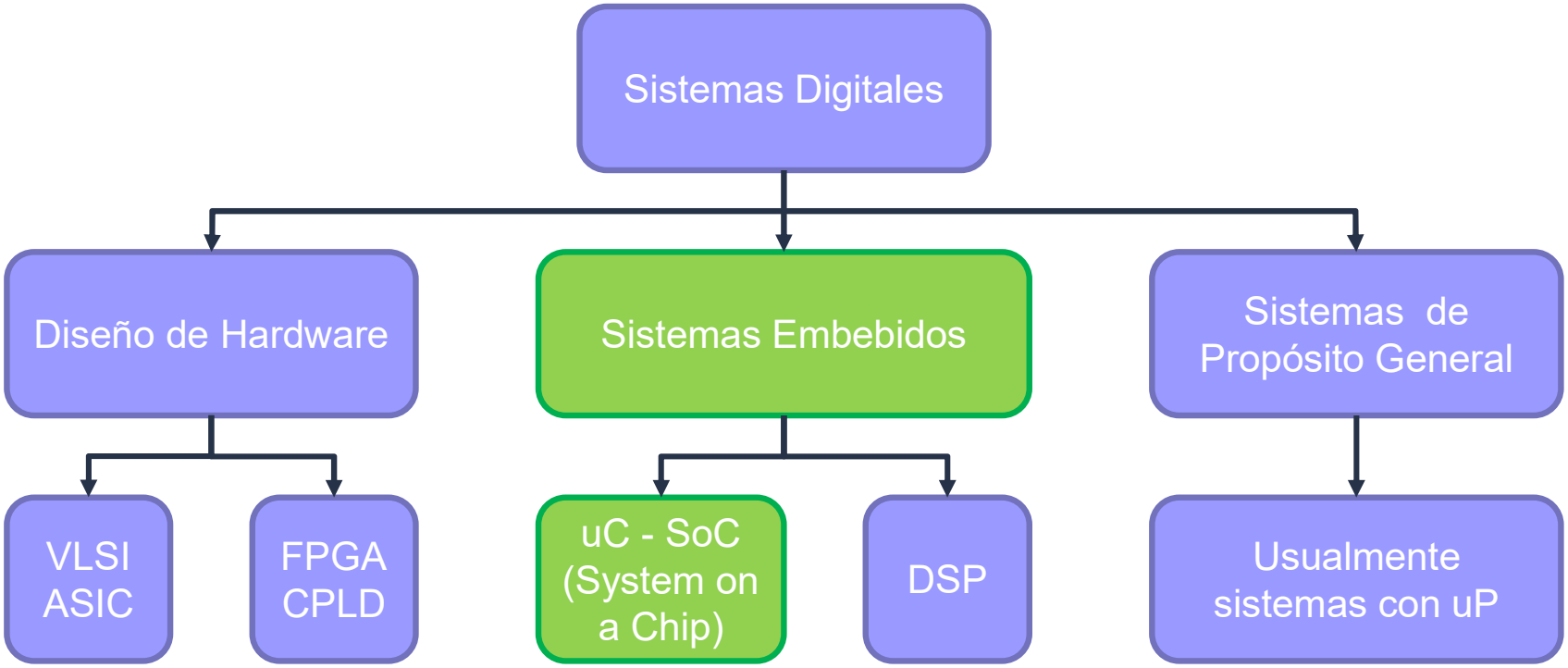
Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

Formar Ingenieros



El ingeniero aplica su intelecto y las habilidades técnicas y profesionales adquiridas, para planificar, diseñar y crear cualquier cosa que solucione problemas de la vida real

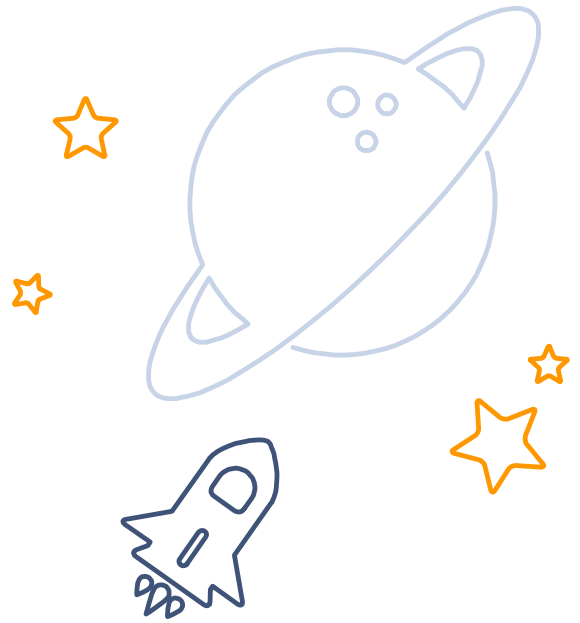
¿En qué nos concentraremos?



Estado del Arte

- Contamos con plataformas (uC - SoC/DSP/FPGA/ASIC/ etc.) de rendimiento y recursos en crecimiento que permitan atender el incremento del procesamiento, necesario para soportar periféricos avanzados con capacidad de atender nuevas conectividades e interfaces de usuario requeridas por el mercado (usuarios)
- Variada oferta de plataformas competitivas en costo, disponibilidad, soporte, herramientas (de HW & SW); en especial en el campo de los microcontroladores de nueva generación (ARM: 32 bits))
- Esto permite recurrir a las mejores prácticas de Ingeniería de Software, al uso de modelos, lenguajes de alto nivel, con y sin un sistema operativo de tiempo real (RTOS o Bare Metal), empleando técnicas de programación específicas para lograr eficiencia, confiabilidad y re-usabilidad

Solución Adecuada



. . . lo más simple posible, previa determinación del objetivo de excelencia a cumplir, obviamente contando con la documentación debida y recurriendo a la metodología de trabajo adecuada

2

Solución Adecuada

1er Cuatrimestre de 2024, dictado por primera vez . . .

Diseño/Desarrollo/Depuración de Hardware y Software: Circuito eléctrico (electrónica analógica/digital/ alimentación) – Circuito Impreso – Producto – Manufactura – Programas (Firmware/Middleware/Software)

En la toma de desiciones influyen:

- El tipo y campo de aplicación, requerimiento del cliente y del medio (regulaciones)
- La idiosincrasia, conocimiento, experiencia y habilidad del profesional
- El contexto:
  - La provisión de herramientas de diseño/desarrollo/depuración
  - La provisión de insumos (conectores/componentes electrónicos/módulos electrónicos/cables de interconexión/gabinete/embalaje/etc.)
  - La provisión de manufactura de circuito impreso/módulo electrónico/cable de interconexión/etc. (montaje y soldadura de componentes/control de calidad/prueba y puesta en marcha/montaje de módulos electrónicos y cables de interconexión en gabinete/embalaje/marcado/etc.)

Diseño/Desarrollo/Depuración de Hardware

- Circuito Eléctrico:
  - Diagrama de Bloques de funciones principales, muestra:
    - Cómo se divide el sistema en funciones principales
    - Cómo se interconectan las funciones principales entre sí
  - Diagrama de c/función principal, muestra:
    - Cómo se interconectan los componentes/módulos electrónicos entre sí
      - Adecuar cada componente/módulo electrónico al que usará
      - Cumplir 100% la hoja de datos del componente/módulo electrónico
  - Cumplir Reglas de Interconexión (Familias Lógicas/ …)

Diseño/Desarrollo/Depuración de Software

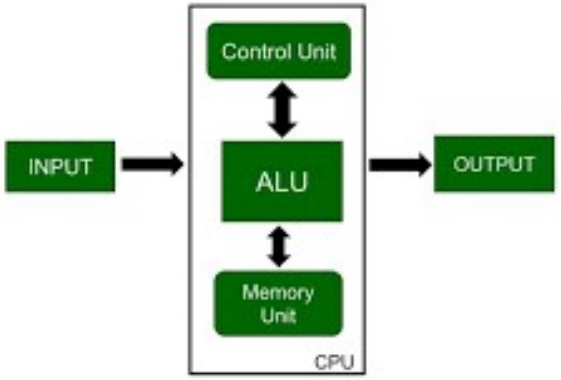
- Nos preocupa tanto el determinismo como el tiempo de respuesta del Sistema
- Sistemas Duros (Hard Real Time): restricciones de tiempo rigurosas
  - Sistemas de control en los cuales se debe escrutar, procesar y actuar en un plazo perentorio (Estrictos), …
  - …, si esto no se cumple el Sistema queda a lazo abierto, pudiendo tener consecuencias catastróficas
- Sistemas Blandos (Soft Real Time): restricciones de tiempo menos rigurosas
  - Sistemas de adquisición de datos o multimedia (Flexibles o Firmes), en …
  - …, si esporádicamente se degrada o se pierde la respuesta, puede ser inconveniente/incómodo pero sin consecuencias catastróficas

Diseño/Desarrollo/Depuración de Software

- Un lenguaje de programación es un lenguaje formal que especifica un conjunto de instrucciones para que una computadora realice tareas específicas
- Se utiliza para escribir programas y aplicaciones de software y para controlar y manipular sistemas informáticos
- Existen muchos lenguajes de programación diferentes, cada uno con su propia sintaxis, estructura y conjunto de comandos. Algunos de los lenguajes de programación más utilizados incluyen Java, Python, C++, JavaScript y C#
- La elección del lenguaje de programación depende de los requisitos específicos de un proyecto, incluida la plataforma que se utiliza, la audiencia prevista y el resultado deseado
- Los lenguajes de programación continúan evolucionando y cambiando con el tiempo, se desarrollan nuevos lenguajes y se actualizan los más antiguos para satisfacer las necesidades cambiantes

Diseño/Desarrollo/Depuración de Software

- Una computadora es un dispositivo que puede aceptar instrucciones humanas, las procesa y responde a ellas; o una computadora es un dispositivo informático que se utiliza para procesar datos bajo el control de un programa de computadora
- Un programa es una secuencia de instrucciones junto con datos
- Los componentes básicos de una computadora son:
  - Unidad de Entrada
  - Unidad Central de Procesamiento (CPU), que se divide en:
    - Unidad de Memoria
    - Unidad de Control
    - Unidad Aritmética Lógica
  - Unidad de Salida



Diseño/Desarrollo/Depuración de Software

- A la CPU se le llama el cerebro de nuestra computadora porque acepta datos, le proporciona espacio de memoria temporal hasta que se almacena (guarda), realiza operaciones lógicas en él y, por lo tanto, procesa (aquí también significa convierte) datos (en información)
- Una computadora se compone de hardware y software
- El software es un conjunto de programas que realizan múltiples tareas juntas
- Un sistema operativo también es software (software del sistema) que ayuda a los humanos a interactuar con el sistema informático
- Estos programas informáticos están escritos en un lenguaje de programación de alto nivel
- Los lenguajes de alto nivel son lenguajes casi humanos, más complejos que el lenguaje comprensible por computadora, que se llama lenguaje de máquina o lenguaje de bajo nivel (solo entiende lenguaje binario (lenguaje de 0s y 1s))

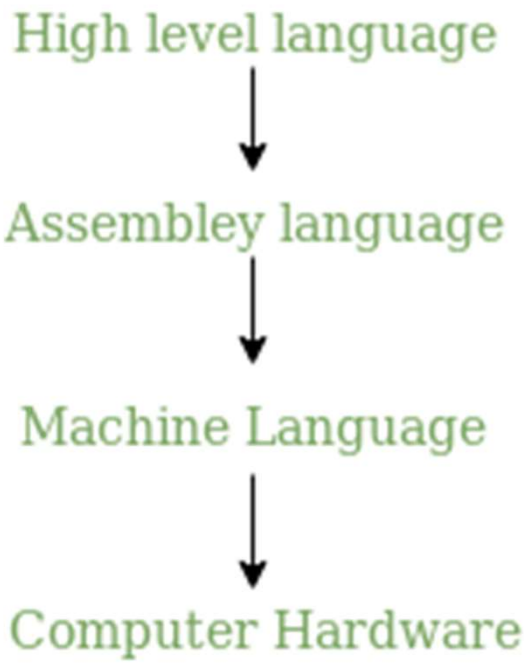
Diseño/Desarrollo/Depuración de Software

Jerarquía de los Lenguajes de Programación

Entre el lenguaje de alto nivel y el de máquina, existen lenguajes ensambladores (assembly, código de máquina simbólico).

Los lenguajes ensambladores (assembly) son específicos de la arquitectura de la computadora. Se requiere de un programa de aplicación (Assembler) para convertir código ensamblador (assembly) en código de máquina ejecutable

El lenguaje de alto nivel es portable pero requiere interpretación o compilación para convertirlo en lenguaje de máquina comprensible por la computadora

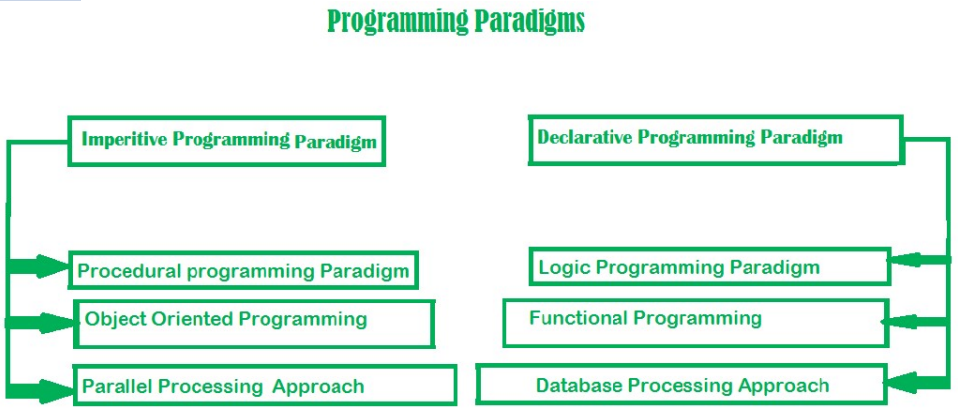


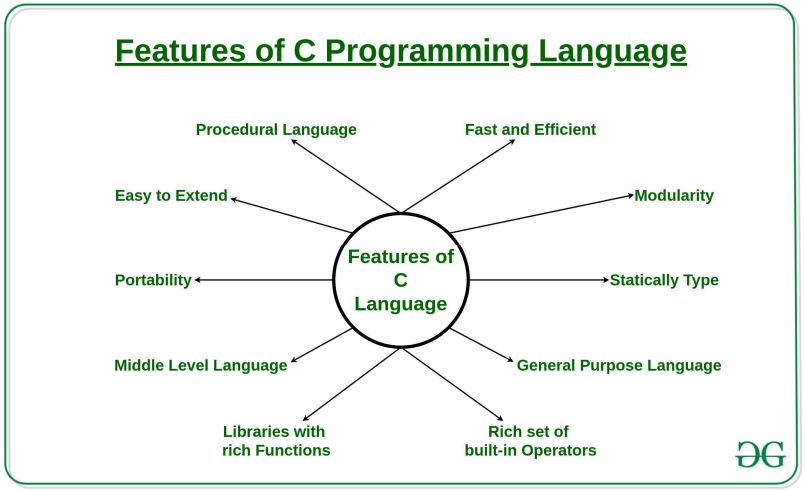
Diseño/Desarrollo/Depuración de Software

C es un lenguaje de programación procedimental. Fue desarrollado inicialmente por Dennis Ritchie en el año 1972

Se desarrolló principalmente como un lenguaje de programación de sistemas para escribir un sistema operativo

Las características principales del lenguaje C incluyen acceso de bajo nivel a la memoria, un conjunto simple de palabras clave y un estilo limpio; que lo hacen adecuado para la programación de sistemas como un sistema operativo o el desarrollo de compiladores





Diseño/Desarrollo/Depuración de Software

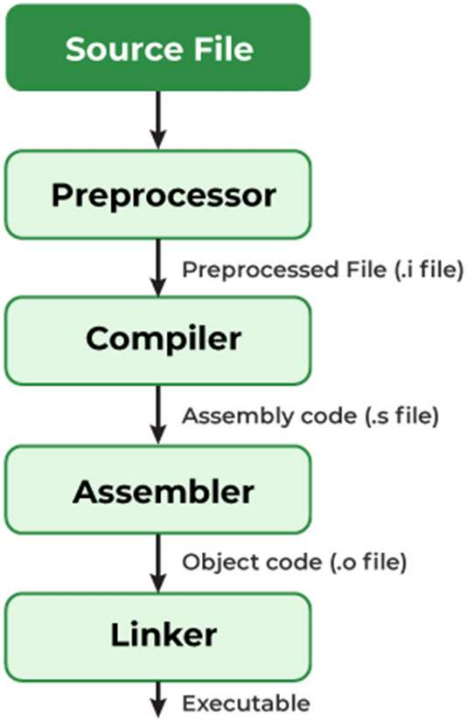
El código fuente debe pasar por varios pasos antes de convertirse en un programa ejecutable y poder correr en la CPU

Preprocessor: busca errores de sintaxis en el código fuente

Compiler/Assembler: traduce el lenguaje C/Assembly a código objeto (un código de máquina que no está listo para ejecutarse)

Linker: vincula el código objeto con funciones de biblioteca precompiladas, creando un programa ejecutable

Se requiere un Loader para carga el programa ejecutable en memoria, para su posterior ejecución



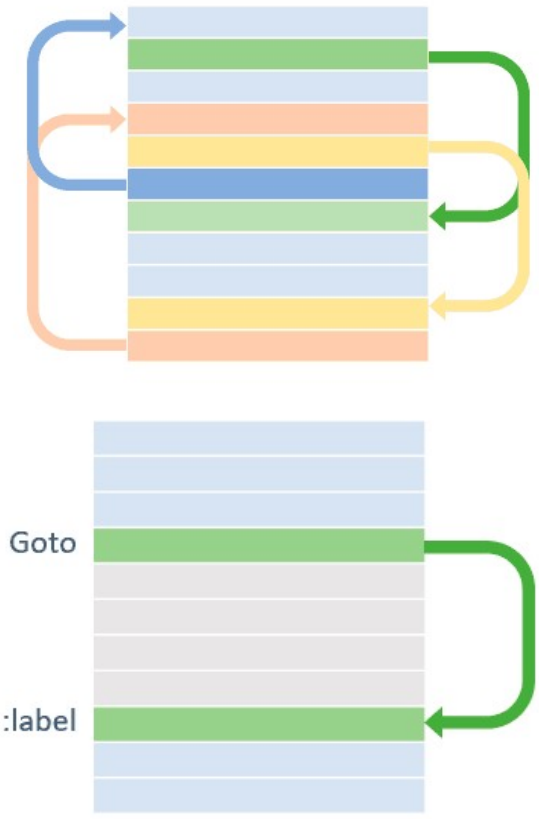
Es usual programar Sistemas Embebidos en C y utilizar bibliotecas de programas/macros tanto en Assembly, como en C++ (por sus ventajas)

"Nothing better than C", Linus Torvalds
https://www.youtube.com/watch?v=CYvJPra7Ebk

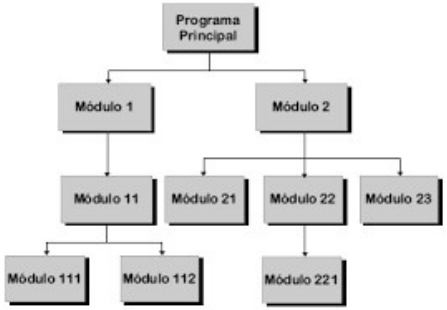
> *Un estándar de codificación en C, es un conjunto de reglas para el código fuente que adopta un equipo de programadores que trabajan juntos en un proyecto, diseño de un Sistemas Embebidos*
>
> *Embedded C Coding Standard by Michael Barr*
>
> *https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard*

Diseño/Desarrollo/Depuración de Software

- Código Spaghetti, término de uso generalizado para código no estructurado y difícil de leer
  - Tal código en cualquier código-base grande puede crear sus problemas, si no se resuelven a tiempo. Puede provocar gran desperdicio de recursos importantes, como tiempo y energía, encontrar errores/corregirlos ya que el código no tiene estructura
- Programación Estructurada, enfoque de programación en el que el programa se crea como una estructura única
  - Tal código ejecutará instrucciones de forma serial y estructurada. No admite saltar de una instrucción a otra (sin GOTO´s)
  - Ventajas: Más fácil de leer/entender/usar/desarrollar/depurar/mantener. Requiere menos esfuerzo y tiempo. Mayormente, independiente de la máquina

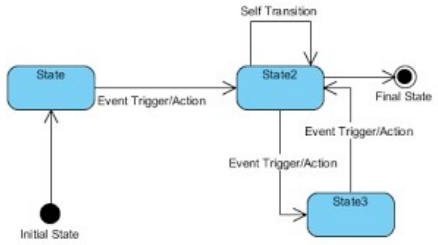


Diseño/Desarrollo/Depuración de Software



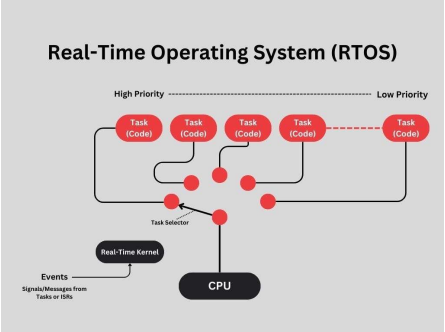
- La programación Modular es el proceso de subdividir un programa en subprogramas separados. Un módulo es un componente de software independiente. A menudo se puede utilizar en una variedad de aplicaciones y funciones con otros componentes del sistema
  - Se deben decidir las limitaciones de todos y cada uno de los módulos. De qué manera se debe dividir un programa en diferentes módulos. Comunicación entre diferentes módulos del código para la correcta ejecución de todo el programa
  - Facilidad de uso: este enfoque permite la simplicidad, podemos acceder a él en forma de módulos. Esto facilita la depuración del código y es propenso a menos errores
  - Reusabilidad: permite al usuario reutilizar la funcionalidad con una interfaz diferente sin tener que volver a escribir todo el programa
  - Facilidad de mantenimiento: ayuda a reducir las colisiones al momento de trabajar en módulos (equipo que trabajar en una aplicación grande)

Diseño/Desarrollo/Depuración de Software



- El Unified Modeling Language (UML) es un lenguaje de modelado de propósito general
  - Su objetivo es definir una forma estándar de visualizar la forma en que se ha diseñado un sistema (similar a los planos utilizados en otros campos de la ingeniería)
  - No es un lenguaje de programación, es más bien un lenguaje visual
- State Machine Diagrams (Máquinas de Estado) | Unified Modeling Language (UML), se usan para representar la condición del sistema (o parte del mismo) en instantes de tiempo finito
  - Diagrama que representa el comportamiento mediante transiciones de estados finitos
  - Se conoce como State-Chart Diagrams (Diagrama de Estados)
  - De manera simple, se utiliza un diagrama de estado para modelar el comportamiento dinámico de una clase en respuesta al tiempo y a estímulos externos cambiantes. Podemos decir que todas y cada una de las clases tienen un estado, pero no modelamos todas las clases utilizando diagramas de estado

Diseño/Desarrollo/Depuración de Software

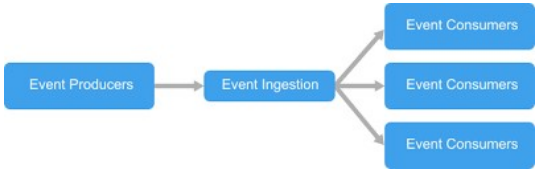


- Al desarrollar Sistemas Embebidos capaces de funcionar en Tiempo Real, una de las primeras y más importantes preguntas es si las aplicaciones deben ejecutarse bajo un Sistema Operativo de Tiempo Real (RTOS) o si se debe desarrollar una solución Bare Metal
- En la programación Bare Metal, la aplicación se escribe accediendo directamente al hardware sin utilizar una interfaz de programación externa (sin Sistema Operativo)
  - La aplicación accede directamente a los registros de hardware del microcontrolador
  - En un bucle/lazo sin fin, que ejecuta tareas con un tiempo de ejecución fijo
  - Esta ejecución secuencial sólo se desvía cuando ocurre un evento de interrupción
  - Este enfoque de desarrollo Bare Metal para Sistemas Embebidos se conoce como Super-Loop (polling & Interrupts), con modelado de tareas (diagramas de estado)

Diseño/Desarrollo/Depuración de Software

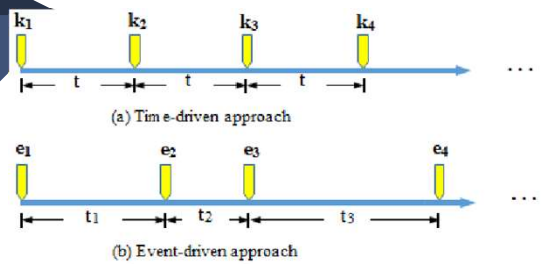
- En la programación con RTOS se utiliza un Núcleo de Sistema Operativo (Kernel) con un Programador/Planificador (Scheduler) y Controladores de Dispositivo (Device Drivers) entre el hardware y el código de la aplicación
  - Posibilita, tanto multitarea (multitasking), como multihilos (multithreading) y multiprocesamiento (multiprocessing)
  - En lugar de pocas tareas claramente definidas y definibles en términos de recursos, se pueden ejecutar muchas tareas (completa o descomponerla en hilos - threads) en uno o varios núcleos (core) de CPU (donde las tareas individuales pueden priorizarse cómodamente y agruparse según el rendimiento de los núcleos de CPU disponibles)
  - Los Sistemas Operativos de Tiempo Real con sus planificadores permiten ejecutar procesos en Concurrencia (Concurrent)/Paralelo (Parallel), de forma flexible y priorizada, asumiendo la responsabilidad de la funcionalidad del sistema. Tanto de forma Cooperativa (Cooperative) como Apropiativa (Preemptive)

Diseño/Desarrollo/Depuración de Software



- Como solución a Sistemas Embebidos orientados a Control, tenemos los Sistemas Disparados/Activados por Tiempo o por Evento (Time- and Event-Triggered Systems)
  - Un disparador/activador (trigger) es un suceso (evento/sincronismo) que provoca el inicio de alguna acción en el Sistema de Control
  - La acción puede ser la ejecución de una tarea leyendo una variable y calculando un nuevo valor de una variable de corrección, o el envío de un mensaje informando valores actuales de variables como presión o temperatura
- En el Control activado/disparado por Evento, una acción se inicia sólo si ocurre un evento significativo (condición de programa/encuesta/interrupción)
  - Por ejemplo, un sensor enviaría un mensaje solo si la temperatura ha cambiado más de 3 °C desde que se envió el último mensaje

Diseño/Desarrollo/Depuración de Software



- En el Control disparado/activado por Tiempo, todas las acciones se inician periódicamente mediante un sincronismo aportado por una Base de Tiempo (Reloj de Tiempo Real)
  - El sensor de nuestro ejemplo enviaría un mensaje en cada ciclo de reloj incluso si la temperatura permanece constante
- El control T-TS genera más comunicación y procesamiento que el control E-TS
- En un sistema distribuido con control E-TS, un componente que no recibe un mensaje de algún otro componente no sabe si no ha habido ningún evento significativo durante mucho tiempo o si el otro componente ha fallado o se ha desconectado
- En un sistema distribuido con control T-TS, los mensajes adicionales enviados actúan como latidos (heartbeats), que le dicen al receptor que el componente emisor y la conexión a ese componente todavía están activos

Diseño/Desarrollo/Depuración de Hardware

- En cuanto a herramientas recurriremos a:
  - Herramientas Informáticas CAE (Computer-Aided Engineering)
    - Captura esquemática, simulación, depuración, validación y verificación de circuitos electrónicos (analógicos/digitales)
    - Usadas en asignaturas previas (Introducción a la Ingeniería Electrónica, Sistemas Digitales, Análisis de Circuitos, Señales y Sistemas, etc.)
  - En cuanto a Módulos Electrónicos:
    - Placa de desarrollo: STM32 32-bits ARM Cortex MCUs, NUCLEO board
    - Dip-switchs, Buttons, Keyboards, Leds, LCD Displays, etc.
    - Instrumentos de Laboratorio

Diseño/Desarrollo/Depuración de Software

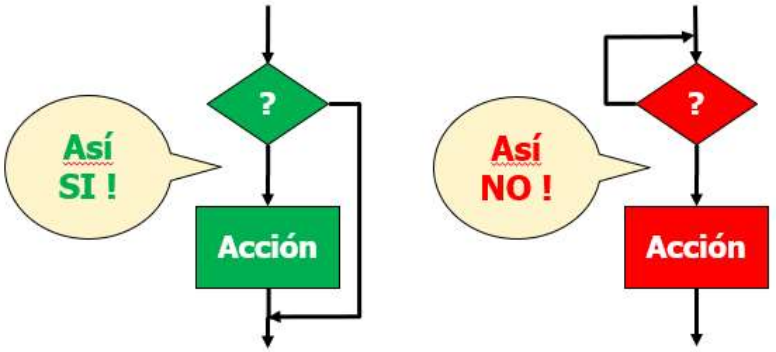
- En cuanto a herramientas recurriremos a:
  - Integrated Development Environment (IDE): STM32CubeIDE - Integrated Development Environment for STM32
  - Version Control System (VCS): Git Bash/GUI
  - Repository: GitHub
  - Software Modeling Tool: Itemis CREATE

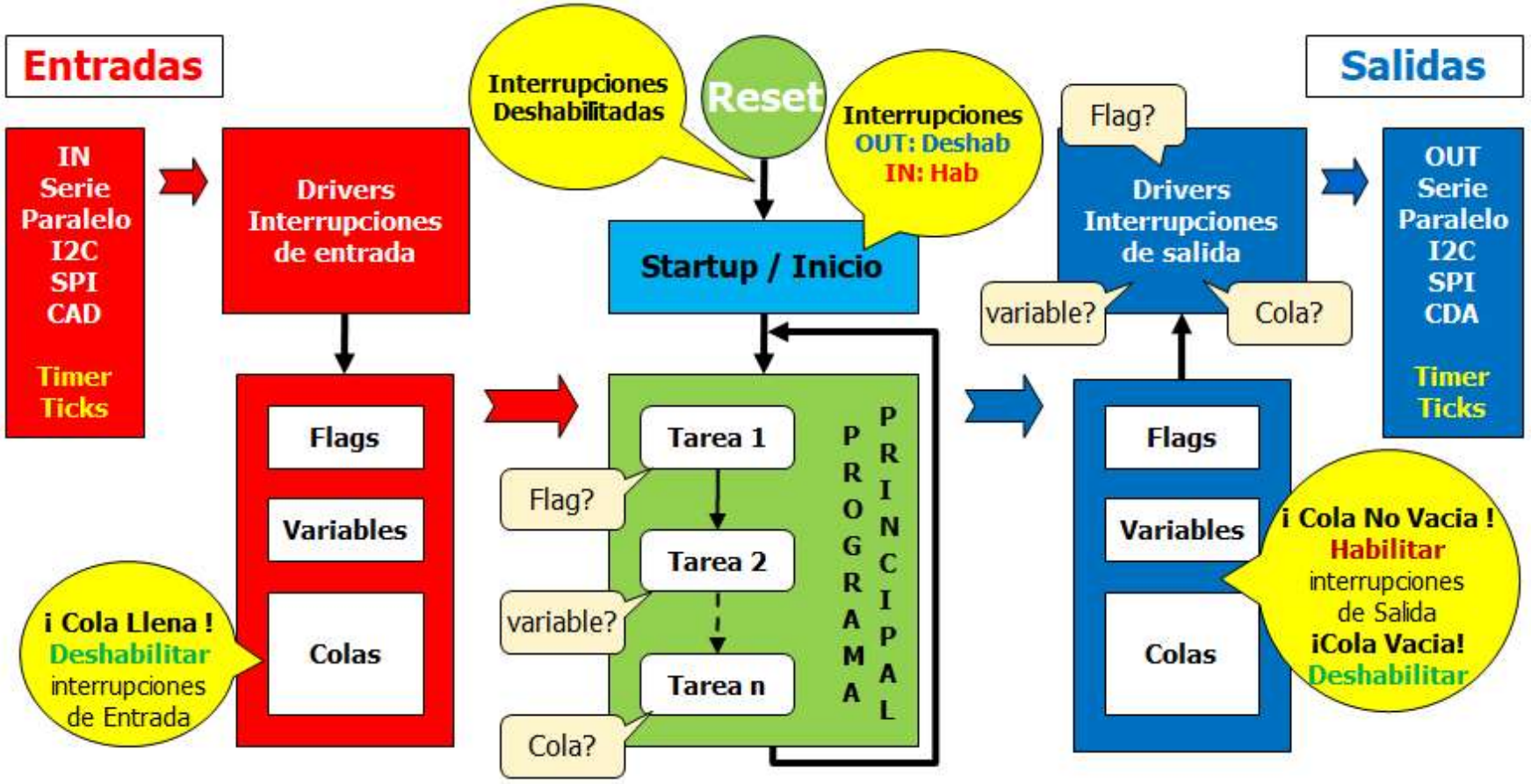
Diseño/Desarrollo/Depuración de Software

- Codificaremos en C
  - "Nothing better than C", Linus Torvalds
- Aplicando el estándar de codificación
  - Embebidos Embedded C Coding Standard by Michael Barr
- Codificaremos soluciones del tipo:
  - Estructurada, Modular
    - Escrutar, Procesar y Actuar
  - Bare Metal (sin Sistema Operativo)
    - Super-Loop (Polling & Interrupts), con modelado de tareas (Diagramas de Estado)
    - Event-Triggered Systems

Diseño/Desarrollo/Depuración de Software

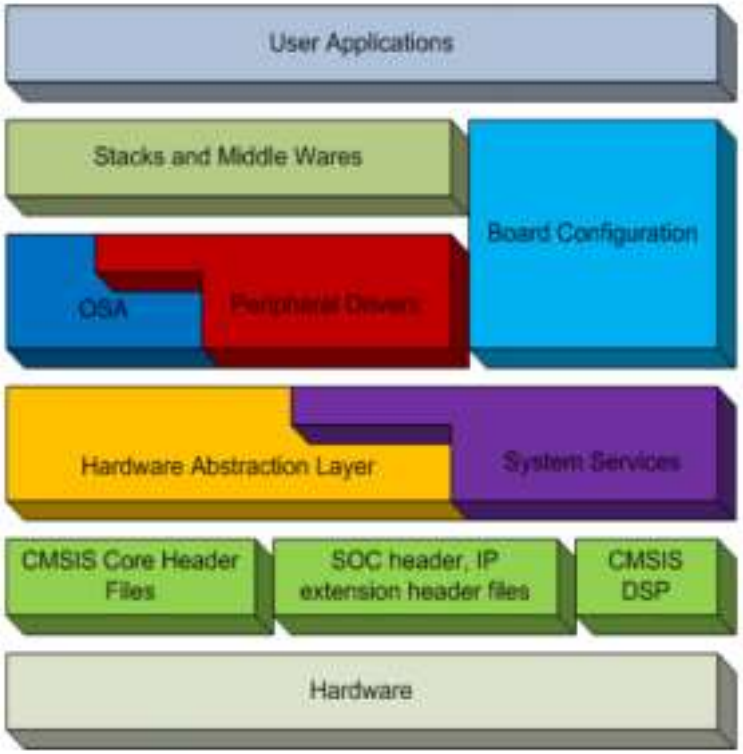
- Estructura de la Aplicación (escrutar, procesar, actuar):
  - Startup (Inicio) => Inicializaciones básicas del MCU
  - Main (Programa Principal) => Iteración perpetua de Tareas (algoritmos)
  - Drivers (Manejadores) de Entrada / Salida => Interacción con el mundo exterior (atención de eventos y sincronismos)
  - Los módulos se comunican / sincronizan mediante:
    - Flags o Variables o Colas
  - Garantizar comportamiento comunitario (que ningún módulo se apropie de la CPU), el código bloqueante es inaceptable




Entradas
IN
Serie
Paralelo
I2C
SPI
CAD
Timer
Ticks
Drivers
Interrupciones
de entrada
Flags
Variables
Colas
¡ Cola Llena !
Deshabilitar
interrupciones
de Entrada
Interrupciones
Deshabilitadas
Reset
Interrupciones
OUT: Deshab
IN: Hab
Startup / Inicio
Tarea 1
Tarea 2
Tarea n
PROGRAMA PRINCIPAL
Flag?
variable?
Cola?
Flag?
Drivers
Interrupciones
de salida
variable?
Cola?
Flags
Variables
Colas
¡ Cola No Vacia !
Habilitar
interrupciones
de Salida
¡Cola Vacia!
Deshabilitar
Salidas
OUT
Serie
Paralelo
I2C
SPI
CDA
Timer
Ticks

Diseño/Desarrollo/Depuración de Software

- Cortex Microcontroller Software Interface Standard (CMSIS)
- System on a Chip (SoC)
- Hardware Abstraction Layer (HAL)
- Operating System Abstraction (OSA) capa p/sistema operativo (MQX, FreeRTOS, uCos, etc.)
- Stacks and Middleware incluye USB stack, TCP/IP stack, Audio, Graphics, Boot Loader
- Board Support Packages (BSP) => Board Configuration



Diseño/Desarrollo/Depuración de Software

- Regla de pulgar del principiante puesto a programar en assembly, asegurar que:
  - $T_{cpu\text{-}pp} \leq 2\ T_{fmax}$ Entrada detectar/Salida generar (1)
    $\leq 1000\ T_{instrucción}$ (2)
    $\leq Tick_{min}$ (2)
    $\leq 60\sim70\% \sum (T_{cpu\text{-}driver/tarea})$ (3)
  - $T_{cpu\text{-}driver/tarea} \leq T_{cpu\text{-}pp} / 10$ (2)

(1) Nyquist-Shannon (Teorema Muestreo)

(2) JMC (Experiencia de desarrollador)

(3) Mamá de JMC (Experiencia de modista)

Diseño/Desarrollo/Depuración de Software

- MCS-51 ejecuta ~ 1MIPS c/F_{clock} = 1MHz o más (CPU de la familia Intel original o derivados) => $T_{instrucción} \approx 1\mu S$
  - $T_{cpu\text{-}pp} \leq 1mS$ / $Tick_{min} \geq 1mS$ / $T_{cpu\text{-}driver/tarea} \leq 100\mu S$
  - Bus de Datos , 1 Acumulador y Operaciones Básicas de 8 bits
  - 1 Puntero a Memoria (favor de comparar con Atmega128)
- ARM CORTEX M3 ejecuta ~ 120MIPS c/F_{clock} = 100MHz o más (CPU de la familia NXP LPC17xx) => $T_{instrucción} \approx 10nS$
  - Bus de Datos, 16 Acumuladores/Punteros y Operaciones Complejas de 32 bits
  - Manteniendo constantes los T enumerado puedo ejecutar 100 veces mayor cantidad de más poderosas instrucciones nos permite resolver problemas más complejos con soluciones de software más abstractas, portables y elaboradas

3

¿ Vamos bien ?

1er Cuatrimestre de 2024, dictado por primera vez . . .

“ Hablando de aplicaciones de uso de software, la pregunta del millón es:

¿ El microcontrolador elegido cuenta con recursos suficientes para una aplicación determinada ?

Necesitamos una lista de verificaciones

- Respecto a los requerimientos de la aplicación:
  - El uC elegido, ¿ cuenta con …
    - la capacidad de procesamiento requerido ?
    - la memoria de "datos/programa" requerida ?
    - los periféricos requeridos ?
  - Las memorias/periféricos (internos/externos al uC) elegidos, ¿ cuentan con …
    - las características/prestaciones requeridas ?
  - Las interfaces de entrada/salida elegida, ¿ cuenta con …
    - las características/prestaciones requeridas ?

Necesitamos una lista de verificaciones

- Respecto a los requerimientos de la aplicación:
  - La fuente de alimentación elegida, ¿ cuenta con …
    - la capacidad de carga requerida ?
  - Los chips elegidos, ¿ cuentan con …
    - el perfil de compatibilidad electromagnética requerido ?
    - el perfil de consumo requerido ?
    - el perfil térmico requerido ?
    - el encapsulado/tipo de montaje/embalaje requerido ?
    - el precio/lote de compra/disponibilidad requeridos ?

Necesitamos una lista de verificaciones

- Las verificaciones consisten en contrastar los requerimientos de la aplicación con:
  - especificaciones contenidas en "hojas de datos, manuales de referencia/usuario" de los componentes elegidos
  - restricciones/recomendaciones contenidas en "normas/notas" de aplicación …
  - determinaciones realizadas en:
    - prototipos virtuales construidos en base a los componentes elegidos
    - prototipos reales construidos en base a los componentes elegidos
    - tiempo de compilación del código desarrollado, uso de memoria
    - tiempo de ejecución del código desarrollado, lo funcional y lo temporal
    - consulta a los proveedores elegidos

Necesitamos una lista de verificaciones

- Usualmente un desarrollador principiante suele desatender las determinaciones realizadas en tiempo de ejecución, del código desarrollado (lo funcional y lo temporal)
- Hasta que las cosas fallan (usualmente en el campo), y como resultado de una auditoría, determina que la razón de la falla tiene que ver con lo temporal
  - En algún caso se puede superar la falla, incrementando la capacidad de procesamiento
  - Ayuda haber elegido una familia de uC que cuenta con una variedad de uC o de familias de uC con el mismo encapsulado/pinout elegido y con mayor/menor capacidad de procesamiento requerido
  - De superarlo por este medio, será necesario revisar todos los ítems de la lista de verificaciones, especialmente los afectados por el incremento de la capacidad de procesamiento

Necesitamos una lista de verificaciones

- El desarrollador principiante debe sumar un par de ítems a su lista de verificaciones:
  - la familia de uC elegida, ¿ cuenta con uC con el mismo encapsulado/pinout y con mayor/menor capacidad de procesamiento requerido ?
  - el fabricante de la familia de uC elegida, ¿ cuenta con otras familias de uC con el mismo encapsulado/pinout y con mayor/menor capacidad de procesamiento requerido ?
- Si incrementar la capacidad de procesamiento no es viable o no resuelve el problema , ¿ qué alternativa tenemos ?
  - Lo primero que surge es, optimizar los tiempos de ejecución del Código desarrollado
    - Para eso necesitamos medir dichos tiempos
      - ¿ Sabemos cómo hacerlo por HW y por SW ?

Necesitamos una lista de verificaciones

- Para medir dichos tiempos por HW … Seteamos => Reseteamos un pin de salida (GPIO), midiendo tiempos con un Osciloscopio/Analizador Lógico
  - ¿ Tenemos un pin de salida (GPIO) libre y tierra, accesibles ?
- Para medir dichos tiempos por SW … Configuramos => Inicializamos => Arrancamos => Detenemos => Leemos un contador de ciclos de reloj, convirtiendo a unidades de tiempo
  - ¿ Tenemos …
    - forma de contar ciclos de reloj, sabemos cómo se debe gestionar ?
    - memoria “programa/datos” suficiente para medir/almacenar tiempos ?
    - salida por consola para imprimir tiempos medidos ?

Necesitamos una lista de verificaciones

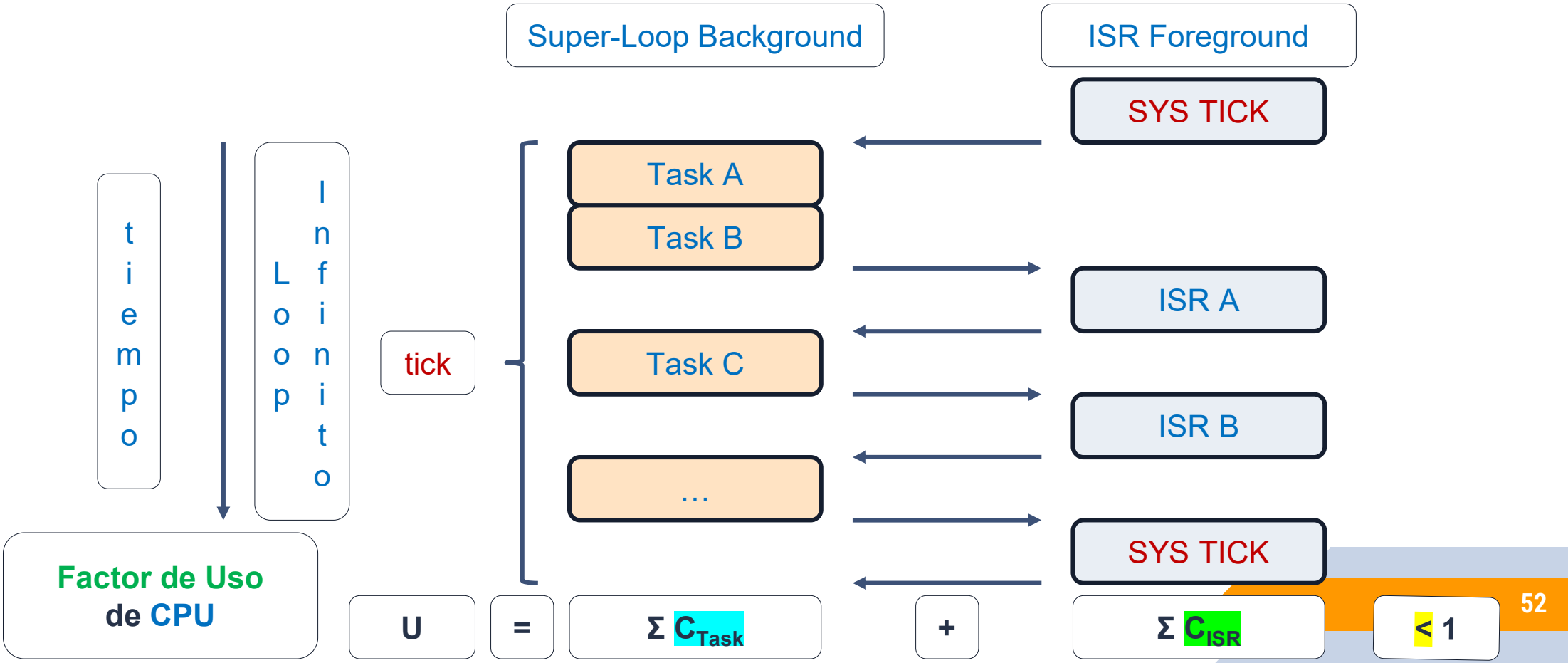
- Si optimizar los tiempos de ejecución del Código desarrollado no es viable o no resuelve el problema, ¿ qué alternativa tenemos ?
  - Lo primero que surge es, analizar el Factor de Utilización de la CPU que hacen las tareas de la aplicación durante la ejecución del Código desarrollado
  - En nuestro caso, la ejecución Código desarrollado, es …
    - Estructurada, Modular
      - Escrutar, Procesar y Actuar (tareas)
    - Bare Metal (sin Sistema Operativo)
      - Super-Loop (Polling & Interrupts), con modelado de tareas (Diagramas de Estado)
      - Event-Triggered Systems

Necesitamos una lista de verificaciones

- Considerando que, la peor situación se da cuando en una vuelta del Super-Loop …
  - se ejecuta hasta completar, al menos una vez cada Tarea (Task) de la aplicación
  - se ejecuta hasta completar, no más de una vez cada Rutina de Atención de Interrupción (ISR) de la aplicación
- En ese caso podemos tomar como tiempo de ejecución disponible en una vuelta del Super-Loop al período de ejecución de la interrupción de menor período de repetición, usualmente el tick del sistema (1 mS)
- De modo tal que, en un tick, se ejecuten una vez c/u de las Tasks e ISRs
- El Factor de Uso de CPU es: $U = \Sigma C_{Task} + \Sigma C_{ISR} < 1$ ($U \geq 1$ sobrecarga)
  - Donde C_{Task} y C_{ISR} son tiempos de ejecución o de cómputo

Foreground/Background System

¿ Cómo ganamos tiempo ?



Necesitamos una lista de verificaciones

- Estamos ante un ejecutor cíclico de tareas que corren hasta completar, por vuelta, una vez, donde el ciclo se repite cada tick (Cyclic Executive [T_{tick}] - Run to Completion)
- Podemos codificar esto de diferentes maneras, por ejemplo, para CorteX M3 de la familia STM32F1 de STMicroelectronics, desarrollando con: STM32CubeIDE + placa NUCLEO-F103RB + HAL library
  - Ejecutamos STM32CubeIDE, abrimos un workspace
  - Generamos un nuevo Proyecto STM32 para la placa NUCLEO-F103RB -> SystemCoreClock = 64Mhz ($T_{SystemCoreClock}$ =15,625nS)
  - El proyecto generado, configura la interrupción de Systick a 1mS y la usa como Base de Tiempo de la HAL Library (como lo requiere nuestro ejecutor cíclico)
  - Así que tenemos que buscar un Callback de Systick de la HAL para agregar el código que necesita el ejecutor cíclico

Necesitamos una lista de verificaciones

- Busco el Handler correspondiente al Systick de CMSIS (vector del método: SysTick_Handler) en Core/Startup/startup_stm32f103rbtx.s ->

```
.word SysTick_Handler
```

- Encontrado SysTick_Handler -> lo resalto -> Botón derecho -> Open Declaration (F3) -> nos conduce a Core/Src/stm32f1xx_it.c, en el método busco una invocación al IRQHandler correspondiente al Systick de la HAL (método: HAL_SYSTICK_IRQHandler) ->

```
void SysTick_Handler(void)
{
    /* USER CODE BEGIN SysTick_IRQn 0 */

    /* USER CODE END SysTick_IRQn 0 */
    HAL_IncTick();
    /* USER CODE BEGIN SysTick_IRQn 1 */

    HAL_SYSTICK_IRQHandler();

    /* USER CODE END SysTick_IRQn 1 */
}
```

Necesitamos una lista de verificaciones

- Si no encuentro HAL_SYSTICK_IRQHandler, agrego al método SysTick_Handler una invocación al método: HAL_SYSTICK_IRQHandler, línea de código ->

```
HAL_SYSTICK_IRQHandler();
```

- Encontrado/Agregado la invocación al método HAL_SYSTICK_IRQHandler -> lo resalto -> Botón derecho del mouse -> Open Declaration (F3) -> nos conduce a Drivers/ STM32F1xx_HAL_Driver /Src/stm32f1xx_hal_cortex.c, donde busco una invocación al Callback correspondiente al Systick de la HAL (método: HAL_SYSTICK_Callback) ->

```
void HAL_SYSTICK_IRQHandler(void)
{
    HAL_SYSTICK_Callback();
}
```

Necesitamos una lista de verificaciones

- Encontrado la invocación al método HAL_SYSTICK_Callback -> lo resalto -> Botón derecho del mouse -> Open Declaration (F3) -> nos conduce a Drivers/ STM32F1xx_HAL_Driver /Src/stm32f1xx_hal_cortex.c, donde busco el método del Callback correspondiente al Systick de la HAL (método: HAL_SYSTICK_Callback) ->

```
__weak void HAL_SYSTICK_Callback(void)
{
    /* NOTE : This function Should not be modified, when the callback is needed,
              the HAL_SYSTICK_Callback could be implemented in the user file
    */
}
```

- Dicho método está definido en forma débil (__weak), por lo tanto, mi código podría redefinirlo como (en app/src/app.c) ->

```
void HAL_SYSTICK_Callback(void)
{
    g_app_tick_cnt++; // Application Tick Counter (Cyclic Executive)
}
```

Necesitamos una lista de verificaciones

- Al código generado por STM32CubeIDE, en el método: main de Core/Src/main.c, invocaremos dos métodos del ejecutor cíclico, uno para inicializarlo (método: app_init) y otro para ejecutarlo (método: app_update) ->

```
int main(void)
{
    …
    /* USER CODE BEGIN 2 */
    app_init();
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    app_update();
    }
    /* USER CODE END 3 */
}
```

Necesitamos una lista de verificaciones

- Mi código define ambos métodos del ejecutor cíclico (en app/src/app.c) ->

```
volatile uint32_t g_app_tick_cnt;

void app_init(void)
 {
    task_a_init();                      /* Run task_x_init */
    task_b_init();
    task_c_init();

    g_app_tick_cnt = G_APP_TICK_CNT_INI;
}

void app_update(void)
{
    if (G_APP_TICK_CNT_INI < g_app_tick_cnt)/* Check if it's time to run tasks */
    {
        g_app_tick_cnt--;

        task_a_update();                /* Run task_x_update */
        task_b_update();
        task_c_update();
    }
}
```

Necesitamos una lista de verificaciones

- Al ejecutar una vuelta, usando métodos de HAL library, se podría dormir al uC
- Mi código redefine además los Callbaks de las interrupciones que requiere la aplicación (en app/src/app.c), definidos en forma débil (__weak) al momento de haber generado el proyecto STM32, o al momento de haber configurado algún periférico mediante la edición del archivo .ioc y la posterior generación de código
- En UM1850 - User manual - Description of STM32F1 HAL and low-layer drivers (https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf), para en Conversor Analógico Digital tenemos los siguientes Callbaks ->

```
void HAL_ADC_ErrorCallback(void)
void HAL_ADC_LevelOutOfWindowCallback(void) (callback of analog watchdog)
void HAL_ADC_ConvCpltCallback(void)
void HAL_ADC_ConvHalfCpltCallback(void)
void HAL_ADCEx_InjectedConvCpltCallback(void)
```

Necesitamos una lista de verificaciones

- Habiendo sobrecarga de la CPU, el peor caso se da, si en una vuelta del Super-Loop …
  - se ejecuta al menos una vez cada Tarea (Task) de la aplicación
    - ganamos tiempo ejecutando cada Tarea con el periodo & offset necesarios
  - se ejecuta no más de una vez cada Rutina de Atención de Interrupción (ISR) de la aplicación
    - ganamos tiempo, de ser posible, atendiendo por DMA a los periféricos que generan Pedidos de Atención de Interrupción (IRQs)
  - ganamos tiempo, de ser posible, aumentando el tick del sistema (más de 1 mS)
- El Factor de Uso de CPU es: $U = \Sigma C_{Task}/T_{Task} + \Sigma C_{ISR} + \Sigma C_{DMA} < 1$ …
  - y el del tick es: $U_{tick} = \Sigma C_{Task} + \Sigma C_{ISR} < 1$ ($U_{tick} \geq 1$ sobrecarga)

Foreground/Background System

SYS TICK

ISRs DMAs

Task A Task B Task C

tiempo …

tick = 0

ISRs DMAs

Task A

tick = 1

… tiempo …

tick = 1

ISRs DMAs

Task A Task B

tick = 2

… tiempo …

tick = 2

$T_{Task\ A}$= 1 tick
$\Phi_{Task\ A}$= 0 tick

$T_{Task\ B}$= 2 tick
$\Phi_{Task\ B}$= 0 tick

$T_{Task\ C}$= 3 tick
$\Phi_{Task\ C}$= 0 tick

SYS TICK

tick = 3

Foreground/Background System

SYS TICK

ISRs

DMAs

Task A

Task C

… tiempo …

SYS TICK

tick = 3

ISRs

DMAs

Task A

Task B

tick = 4

… tiempo …

tick = 4

ISRs

DMAs

Task A

tick = 5

… tiempo …

tick = 5

$T_{Task\ A}$ = 1 tick

$\Phi_{Task\ A}$ = 0 tick

$T_{Task\ B}$ = 2 tick

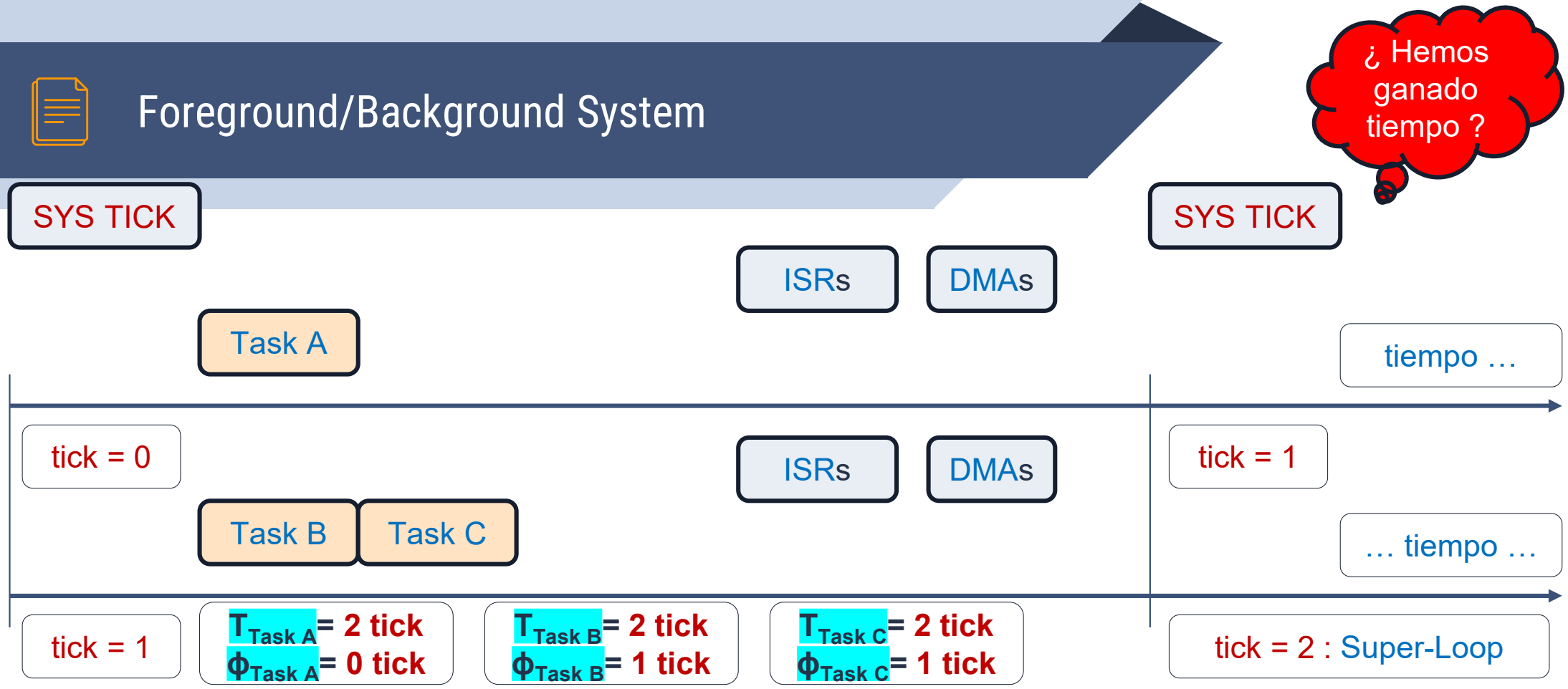
$\Phi_{Task\ B}$ = 0 tick

$T_{Task\ C}$ = 3 tick

$\Phi_{Task\ C}$ = 0 tick

tick = 6 : Super-Loop

Foreground/Background System



Foreground/Background System

¿ Hemos ganado tiempo ?

SYS TICK

ISRs

DMAs

SYS TICK

Task A

tiempo …

tick = 0

ISRs

DMAs

tick = 1

Task B

… tiempo …

tick = 1

ISRs

DMAs

tick = 2

Task C

… tiempo …

tick = 2

$T_{Task\ A}$= 3 tick
$\Phi_{Task\ A}$= 0 tick

$T_{Task\ B}$= 3 tick
$\Phi_{Task\ B}$= 1 tick

$T_{Task\ C}$= 3 tick
$\Phi_{Task\ A}$= 2 tick

tick = 3 : Super-Loop

Necesitamos una lista de verificaciones

- Estamos ante un ejecutor cíclico de tareas que corren hasta completar, en función de su periodicidad, una vez, donde el ciclo se repite cada tick (Cyclic Executive [T_{tick}] - Run to Completion)
- Partimos de la solución, modificando los métodos del ejecutor cíclico, uno para inicializarlo (método: app_init) y otro para ejecutarlo (método: app_update)
- Pero antes, defino un par de estructuras de datos, contenedoras de las particularidades de una tarea:
  - Una cuyo contenido es fijo, asignando en tiempo de compilación (cfg)
  - Otra cuyo contenido es variable, asignado en tiempo de ejecución (dta)

Necesitamos una lista de verificaciones

- La estructura de datos asignados en tiempo de compilación (cfg) ->

```
typedef struct {
            void (*task_init)(void *);              // Pointer to the task (must be
            void (*task_update)(void *);            // a 'void (void *)' function)

            void *parameters;                       // Pointer to parameters

            uint32_t period;                        // Interval (ticks) between
                                                    // subsequent runs
            uint32_t offset;                        // Offset (ticks) until
            …                               // the first run of the task
} task_cfg_t;
```

- La estructura de datos asignados en tiempo de ejecución (dta) ->

```
typedef struct {
            uint32_t ticks;             // Ticks until the task will (next) be run

            uint32_t WCET;              // Worst-case execution time (microseconds)
            …
} task_dta_t;
```

Necesitamos una lista de verificaciones

- Declaro un par de **estructuras** de datos, contenedoras de las **particularidades** de una tarea (**cfg** & **dta**) defino la cantidad de elementos contenidos en la lista ->

```
task_cfg_t task_cfg_list[] = {

            {task_a_init, task_a_update, NULL, TASK_A_PERIOD, TASK_A_OFFSET},
            {task_b_init, task_b_update, NULL, TASK_B_PERIOD, TASK_B_OFFSET},
            {task_c_init, task_c_update, NULL, TASK_C_PERIOD, TASK_C_OFFSET}

            };

#define TASK_QTY(sizeof(task_cfg_list)/sizeof(task_cfg_t))

…

task_dta_t task_dta_list[TASK_QTY];
```

Necesitamos una lista de verificaciones

- Modifico el método de inicialización ->

```
volatile uint32_t g_app_tick_cnt;
…
void app_init(void *parameters)
 {
    uint32_t index;

/* Go through the task arrays (configuration) */
for (index = 0; TASK_QTY > index; index++)
    {
        /* Run task_x_init */
        (*task_cfg_list[index].task_init)(parameters);

        /* Init variables */
        task_dta_list[index].ticks = task_cfg_list[index].offset;
        task_dta_list[index].WCET = TASK_X_WCET_INI;
    }
    g_app_tick_cnt = G_APP_TICK_CNT_INI;
}
```

- Modifico el método de ejecución ->

Necesitamos una lista de verificaciones

```
void app_update(void *parameters)
{
    uint32_t index;

    /* Check if it's time to run tasks */
    if (G_APP_TICK_CNT_INI < g_app_tick_cnt)
    {
        g_app_tick_cnt--;

        /* Go through the task arrays */
        for (index = 0; TASK_QTY > index; index++)
        {
            /* Check if it's time to run a task */
            if (0 == task_dta_list[index].ticks)
            {
                /* Run task_x_update */
                (*task_cfg_list[index].task_update)(parameters);

                /* Update variables */
                task_dta_list[index].ticks = task_cfg_list[index].period;
            }
            task_dta_list[index].ticks--;
        }
    }
}
```

Necesitamos una lista de verificaciones

- Para medir dichos tiempos por SW … Configuramos => Inicializamos => Arrancamos => Detenemos => Leemos un contador de ciclos de reloj, convirtiendo a unidades de tiempo
  - ¿ Tenemos …
    - forma de contar ciclos de reloj, sabemos cómo se debe gestionar ?
    - memoria "programa/datos" suficiente para medir/almacenar tiempos ?
    - salida por consola para imprimir tiempos medidos ?
- Si, en app/inc/dwt.h tenemos un conjunto de macros que nos permiten hacerlo ->

```
#define cycle_counter_init() ({\
            CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;/* enable DWT hardware */\
            DWT->CYCCNT = 0;/* reset cycle counter */\
            DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;/* start counting */\
            })
```

Necesitamos una lista de verificaciones

- Si, en app/inc/dwt.h tenemos un conjunto de **macros** que nos permiten hacerlo ->

```
/* reset cycle counter */
/*!< DWT Cycle Counter register */
#define cycle_counter_reset() (DWT->CYCCNT = 0)

/* start counting */
/*!< CYCCNTENA bit in DWT_CONTROL register */
#define cycle_counter_enable() (DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk)

/* disable counting if not used any more */
/*!< CYCCNTENA bit in DWT_CONTROL register */
#define cycle_counter_disable() (~DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk)

/* read cycle counter */
/*!< DWT Cycle Counter register */
#define cycle_counter_get() (DWT->CYCCNT)

#define cycles_per_us(SystemCoreClock / 1000000)
#define cycle_counter_time_us() (DWT->CYCCNT / cycles_per_us)
```

Necesitamos una lista de verificaciones

- Si, en app/inc/dwt.h tenemos un conjunto de **macros** que nos permiten hacerlo ->

```
/*  uint32_t cycle_counter = 0;
 *  uint32_t cycle_counter_time_us = 0;
 *                                                          // PC8 (GPIO)
 *  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);   // => ______
 *  cycle_counter_init();
 *  ...
 *  ...                                                     // => ______
 *                                                          //       ___
 *  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);     // => __/
 *  // or => HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);
 *  cycle_counter_reset();
 *                                                          //    ______
 *  ...                                                     // =>
 *  cycle_counter = cycle_counter_get();
 *  cycle_counter_time_us = cycle_counter_time_us();        //    __
 *  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);   // =>   \___
 *  // or => HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);
 *                                                          // => ______
 *
 *  LOGGER_LOG("Cycles: %lu - Time %lu uS\r\n", cycle_counter, cycle_counter_time_us);
 *
 */
```

Necesitamos una lista de verificaciones

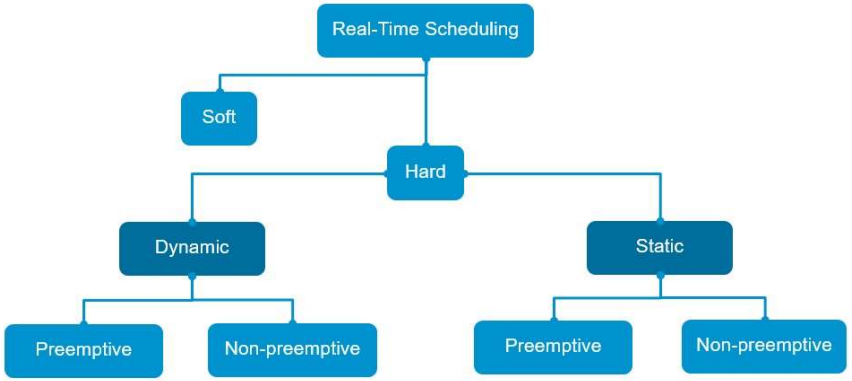
```
void HAL_SYSTICK_Callback(void)
{
    …
    //HAL_GPIO_TogglePin(LED_A_PORT, LED_A_PIN);
}

void app_init(void *parameters)
{
    …
    cycle_counter_init();
}
```

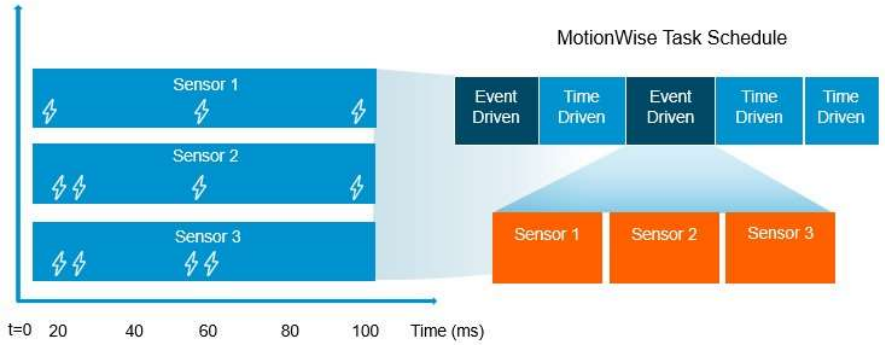
Necesitamos una lista de verificaciones

```
void app_update(void *parameters)
{
    uint32_t cycle_counter, cycle_counter_time_us;
    …
    if (0 == task_dta_list[index].ticks)
    {
        //HAL_GPIO_TogglePin(LED_A_PORT, LED_A_PIN);
        cycle_counter_reset();

        /* Run task_x_update */
        (*task_cfg_list[index].task_update)(parameters);

        cycle_counter = cycle_counter_get();
        cycle_counter_time_us = cycle_counter_time_us(cycle_counter);

        /* Update variables */
        …
        if (task_dta_list[index].WCET < cycle_counter_time_us)
        {
            task_dta_list[index].WCET = cycle_counter_time_us;
        }
    }
    …
```

Necesitamos una lista de verificaciones

- Si las optimizaciones explicadas anteriormente no son viables o no resuelven el problema, ¿ qué alternativa tenemos ?
  - Migrar la aplicación de Bare Metal (sin Sistema Operativo) a Multitarea (sin Sistema Operativo) con planificación Cooperativa (Cooperative) o Apropiativo (Preemptive)
  - Lo que permitirá que la gestión de tiempo de CPU la haga el Planificador (Scheduler) del Sistema Operativo
- Pero esta es otra historia, lo dejamos para la próxima …





Campus Grado FIUBA: **Bienvenida**

- Bienvenida: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=0
- Docentes: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=2
- Evaluación Integradora: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=3
- Correlatividades: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=4
- Contenidos: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=5
- Cronograma: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=6
- Bibliografía: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=7
- SW: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=10
- HW & FW/MW: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=11
- Files & Foldes: https://campusgrado.fi.uba.ar/course/view.php?id=1217&section=12

Referencias

- C Programming Language Tutorial - https://www.geeksforgeeks.org/
- "Nothing better than C", Linus Torvalds - https://www.youtube.com/watch?v=CYvJPra7Ebk
- Embedded C Coding Standard by Michael Barr - www.barrgroup.com/embedded-systems/books/embedded-c-coding-standard
- Development on Bare Metal vs. RTOS - https://www.sysgo.com/professional-articles/bare-metal-vs-rtos
- Difference between Multiprogramming, multitasking, multithreading and multiprocessing - https://www.geeksforgeeks.org/
- Difference between Preemptive and Cooperative Multitasking - https://www.geeksforgeeks.org/
- Event-Triggered Systems (ETS) and Time-Triggered (TTS) - https://ebrary.net/51334/computer_science/time_event_triggered_systems

Referencias

- Compiling a C Program: Behind the Scenes - https://www.geeksforgeeks.org/
- Loader in C/C++ - https://www.geeksforgeeks.org/
- Spaghetti Code - https://www.geeksforgeeks.org/
- Structured Programming Approach with Advantages and Disadvantages - https://www.geeksforgeeks.org/
- Introduction of Programming Paradigms - https://www.geeksforgeeks.org/
- Differences between Procedural and Object Oriented Programming - https://www.geeksforgeeks.org/
- Modular Approach in Programming - https://www.geeksforgeeks.org/
- Unified Modeling Language (UML) Diagrams | An Introduction - https://www.geeksforgeeks.org/
- State Machine Diagrams | Unified Modeling Language (UML) -https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/

Referencias

- CAD vs CAE vs CAM (Electronic manufacturing industry) - https://www.pcbaaa.com/cad-vs-cae-vs-cam/
- Computer Aided Software Engineering (CASE) - https://www.geeksforgeeks.org/
- What is an Arm processor? - https://www.techtarget.com/whatis/definition/ARM-processor
- STM32 32-bit Arm Cortex MCUs - https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html
- STM32 Nucleo Boards - https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html
- STM32 Nucleo expansion boards - https://www.st.com/en/evaluation-tools/stm32-nucleo-expansion-boards.html
- STM32CubeIDE - Integrated Development Environment for STM32 - https://www.st.com/en/development-tools/stm32cubeide.html

Referencias

- Git - https://git-scm.com/
- Git Tutorial - https://www.geeksforgeeks.org/
- GitHub - https://github.com/
- Git Clone vs Fork in GITHUB - https://chedyhammami.medium.com/git-clone-vs-fork-in-github-610f158d61e3
- What is an IDE? - https://www.geeksforgeeks.org/
- Itemis CREATE - https://www.itemis.com/en/products/create/
- "Nothing better than C", Linus Torvalds - https://www.youtube.com/watch?v=CYvJPra7Ebk
- Embedded C Coding Standard by Michael Barr - www.barrgroup.com/embedded-systems/books/embedded-c-coding-standard

Referencias

- ‘The Engineering of Reliable Embedded Systems’ (Ed. 2) by Michael J. Pont - https://www.safetty.net/publications/the-engineering-of-reliable-embedded-systems-second-edition
- ‘Patterns for Time-Triggered Embedded Systems’ by Michael J. Pont - https://www.safetty.net/publications/pttes
- ‘Embedded C Git Tutorial’ by Michael J. Pont - https://www.safetty.net/publications/embedded-c
- ‘Real-Time Scheduling for Safe Autonomous Driving’ by Marija Sokcevic - https://www.tttech-auto.com/knowledge-platform/real-time-scheduling-safe-autonomous-driving

Manos a la obra con el . . .

. . . Proyecto Intermedio

. . . un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**


Las y los estudiantes preguntarán:
¿en qué lío nos metimos?

¡Muchas gracias!

¿Preguntas?

. . .

Consultas a: jcruz@fi.uba.ar