SASE 2018

SIMPOSIO ARGENTINO DE SISTEMAS EMBEBIDOS

15 | 16 | 17 DE AGOSTO

RTOS Workshop

Msc. Ing. Carlos Centeno

Grupo de Investigación y Transferencia en Electrónica Avanzada

G.In.T.E.A - UTN FRC

Temario

- Consideraciones del Workshop
- Topología Super Loop
- Topología RTOS
  - Generalidad. Requisitos de Hardware. Opciones Disponibles
  - Tareas
    - TCB
    - Estados Definidos
    - Prioridades

Temario

- Sincronización con Eventos
  - Semaforos
  - Mailbox
  - Queues
  - Deadlock
- Cambio de Contexto
- Resumen de Aspectos Relevantes

Consideraciones Especiales

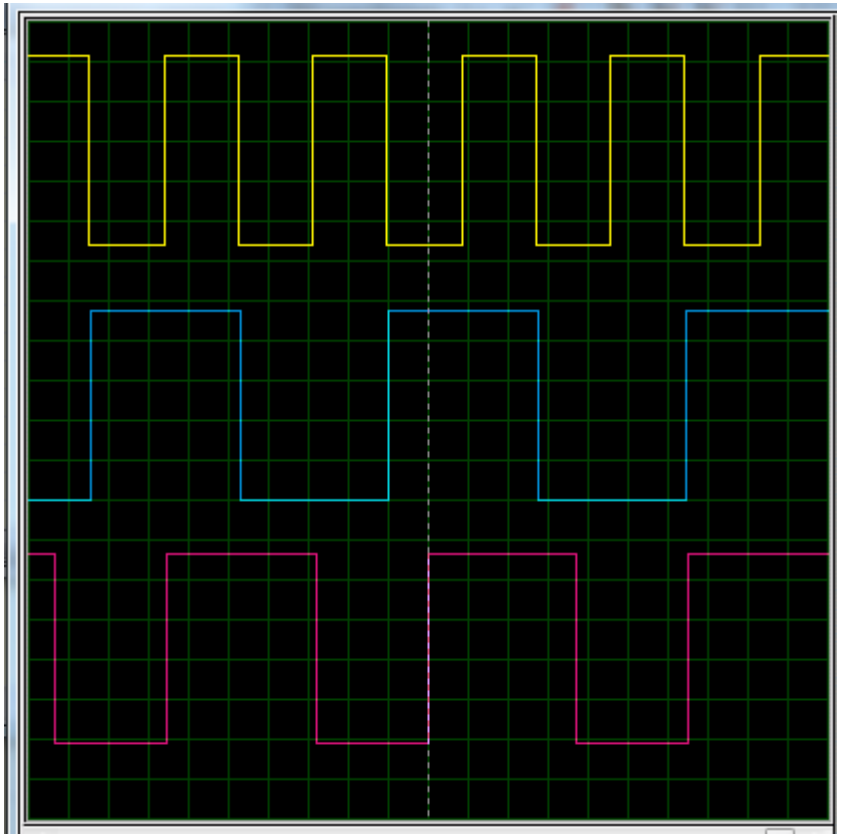
- Cuando usar RTOS
  - Es conveniente
  - Que recursos necesito
  - Cual es la definición concreta de RTOS
- Cuales son las opciones en el mercado
- En el workshop NO se explicará como usar un RTOS determinado.
- El objetivo es conocer los aspectos MAS Relevantes del uso de un RTOS.

Super Bucle

- Resolvamos un pequeño problema.
  - Ejemplo 1
  - Implementar un Sistema Embebido que controle tres secuencias temporales en salidas digitales.
  - Usar topología Super Loop.
  - El control de tiempo se realiza con espera pasiva.

Secuencia

- Secuencia 1:
  - Alto : 1mS
  - Bajo : 1mS
- Secuencia 2:
  - Alto : 2mS
  - Bajo : 2mS
- Secuencia 3:
  - Alto : 3mS
  - BAjo: 4mS



Super Bucle

```
void main(void)
{
    for(;;)
    {
        Task1();
        Task2();
        Task3();
    }
}
```

Super Bucle

- Cada ***"tarea"*** es una función en C.
- Se llaman por turno desde el bloque principal.
  - Se ejecutan rápidamente y regresan al bloque principal.
  - Pueden usar una variable de estado.
  - Se usan esperas pasivas – delay.
    - **NO** existen prioridades.
    - **NO** hay timers.
    - **NO** hay comunicación entre tareas

Control del Tiempo - Timers

- Para el control del Tiempo se puede utilizar la ISR de un timer para incrementar una variable Global.

```
unsigned int Tics;   // variable global
void timerIsr(void)
{
        Tics++;
}
```

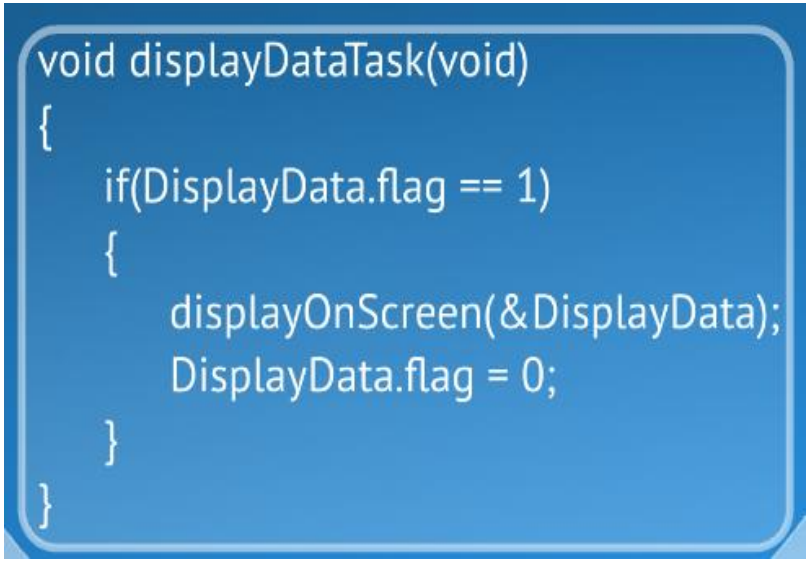
Control del Tiempo

- La bandera tiempo10 se incrementa en la ISR de un timer.

```
void main (void)
{
    int i;
    while(1){
        if(tiempo10)
        {
            for(i=0; i<3; i++)
            {
                if(switchChange(i))
                    changeMotor(i);
            }
            tiempo10 = 0;
        }
    }
}
```

Comunicación

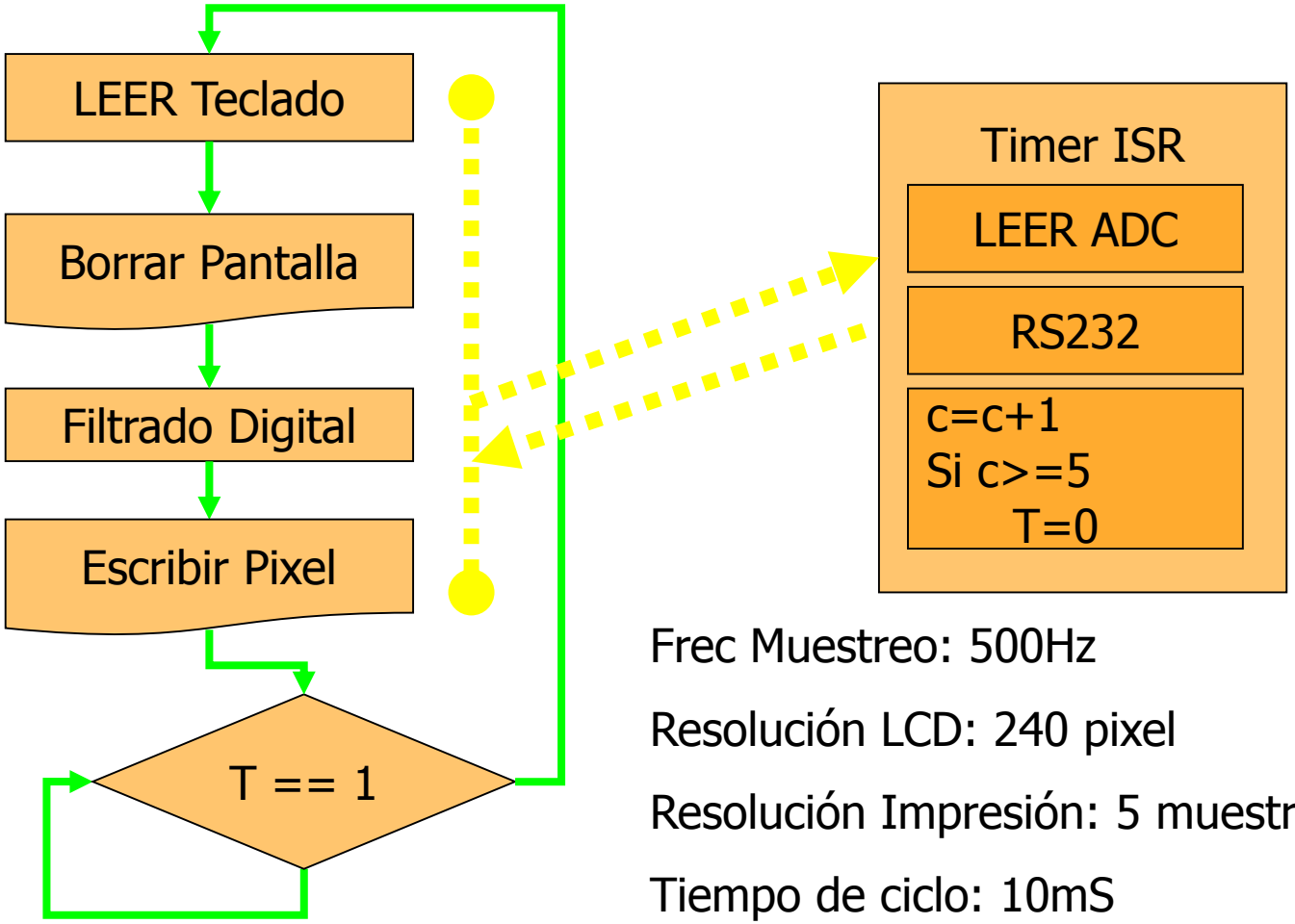
- Para la comunicación entre tareas se utiliza una variable global que contiene el flag de señalización y el dato propiamente dicho.



```
void displayDataTask(void)
{
    if(DisplayData.flag == 1)
    {
        displayOnScreen(&DisplayData);
        DisplayData.flag = 0;
    }
}
```

Prioridades

- La prioridad de ejecución depende del orden en el bloque principal.
- Una proceso crítico debe ejecutarse dentro de un servicio de Interrupciones.



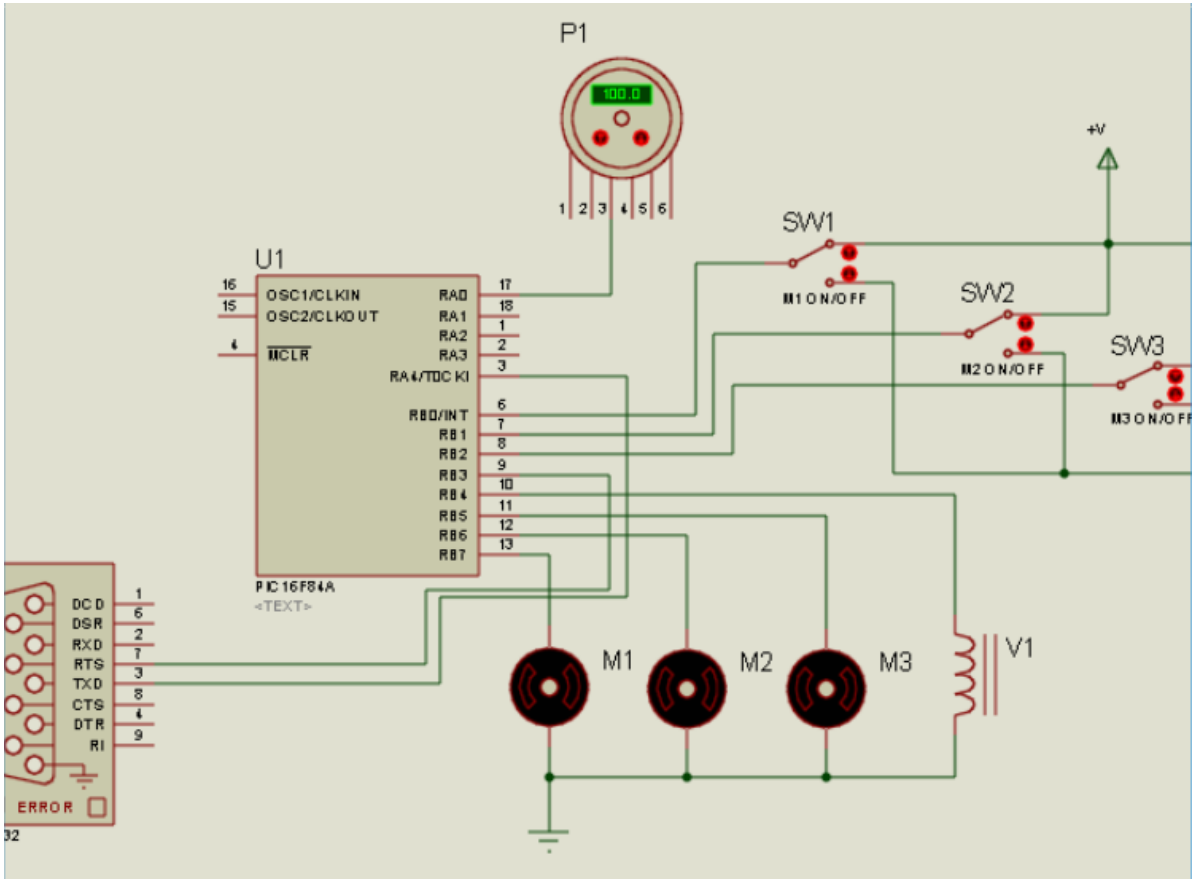
Frec Muestreo: 500Hz

Resolución LCD: 240 pixel

Resolución Impresión: 5 muestras por pixel

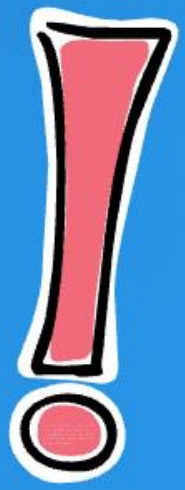
Tiempo de ciclo: 10mS

Ejemplo de Sistema



Solución Propuesta

```
void main (void)
{
    int i;
    for(;;)
    {
        checkMotorsSwitch();
        checkPresion();
        checkRS232();
    }

}
```



Solución Propuesta

```
void main (void)
{
    int i;
    for(;;)
    {
        checkMotorsSwitch();
        checkPresion();
        checkRS232();
    }

}
```

Si la velocidad de ejecución del lazo completo es menor que la velocidad de los datos que llegan por el puerto serial, es posible perder información

- Con control del Tiempo del ciclo para cada proceso

```
if(tiempo10)
{
    for(i=0; i<3; i++)
    {
        if(switchChange(i))
            changeMotor(i);
    }
    tiempo10 = 0;
}
```

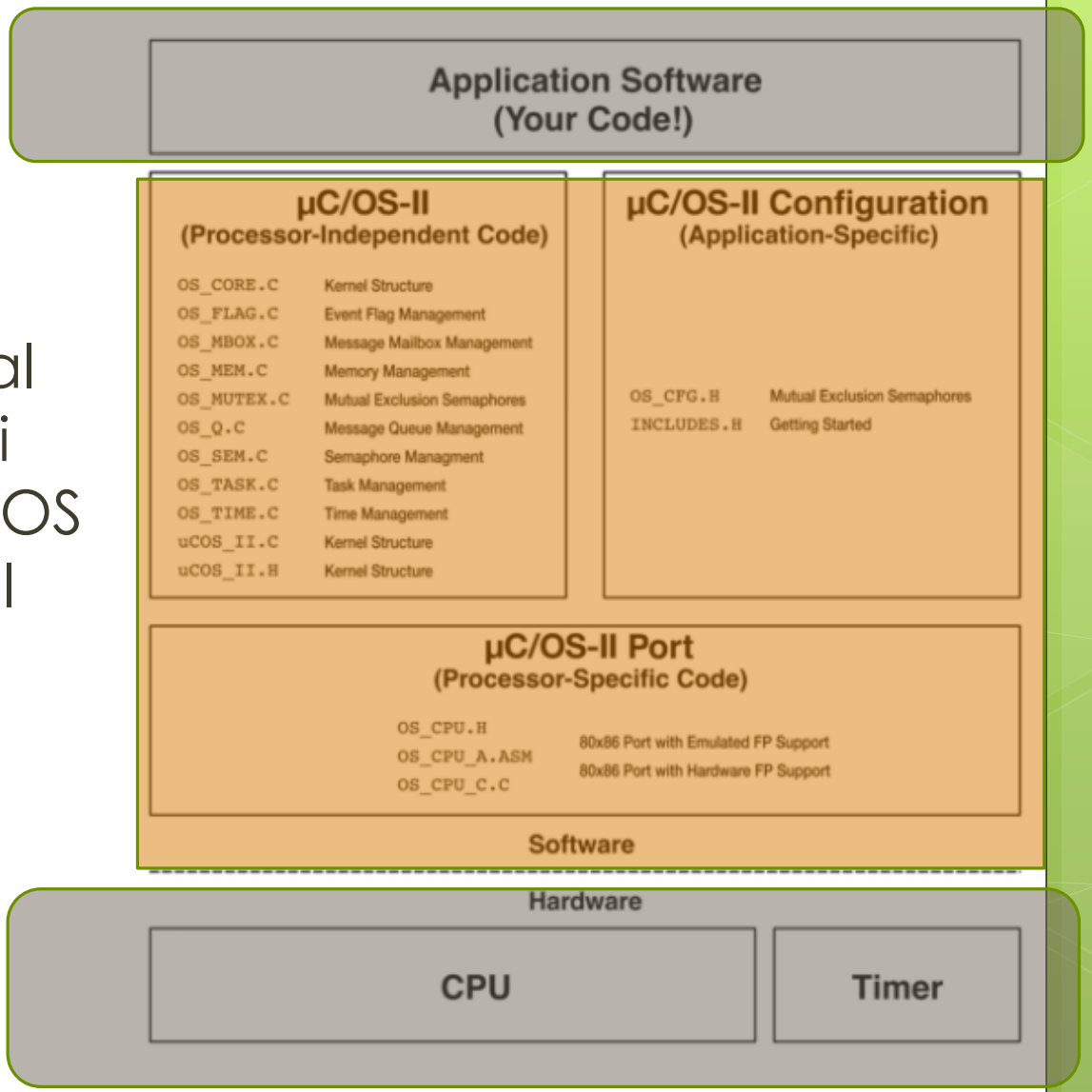
```
if(tiempo50)
{
    switch(valveState)
    {
        case OPEN:
            if(presion < 90)
            {
                closeValve();
                valveState = CLOSE;
            }
            break;
        case CLOSE:
            if(presion > 100)
            {
                openValve();
                valveState = OPEN;
            }
            break;
    }
    tiempo50 = 0;
}
```

¿Conviene usar RTOS?

- Se debe pensar en cambio de plataforma.
  - Uso de STACK
  - Micro del ejemplo no soporta RTOS
- La solución super loop es compacta y fácil de entender .
- No se pueden implementar prioridades.
- El tiempo del bucle depende de como se resolvió a nivel de código la solución del problema.

RTOS

- Estructura final de software si usamos un RTOS como uCOS-II



Porque usar un RTOS

- Se evalúa el sistema y surge:
  - Hay tareas independientes
    - Botones para Configuración
      - TASK 1
    - Conversor Analógico Digital – ADC
      - TASK 2
    - Comunicación Serie
      - TASK 3
  - Hay un tiempo para respetar
  - Hay una comunicación serie que puede requerir una cola de mensajes

RTOS - Generalidad

- Se introducen conceptos
  - ***Tareas***
    - TCB
    - Stack
    - Estados Definidos
  - Preemptive
  - Reentrancia
  - Eventos
    - ECB

- Las procesos del super loop se dividen en tareas.
  - Se debe asignar un Task Control Block (TCB).
  - Se debe asignar STACK.
  - Se debe asignar una PRIORIDAD.

RTOS - Generalidad

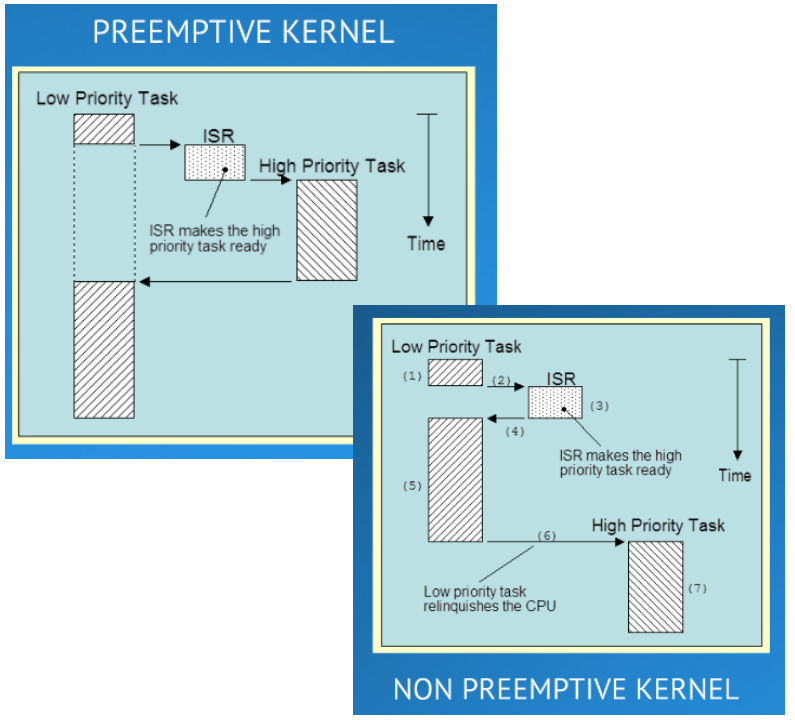
- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
  - ***Estados Definidos***
  - Preemptive
  - Reentrancia
  - Eventos
    - ECB

- Según el RTOS que se elija serán los estados posibles de cada tarea.

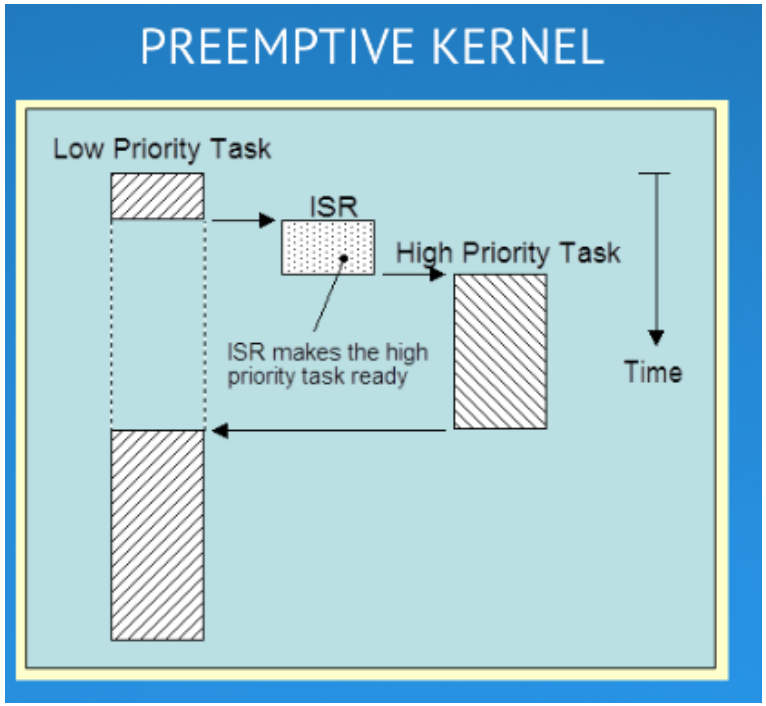
RTOS - Generalidad

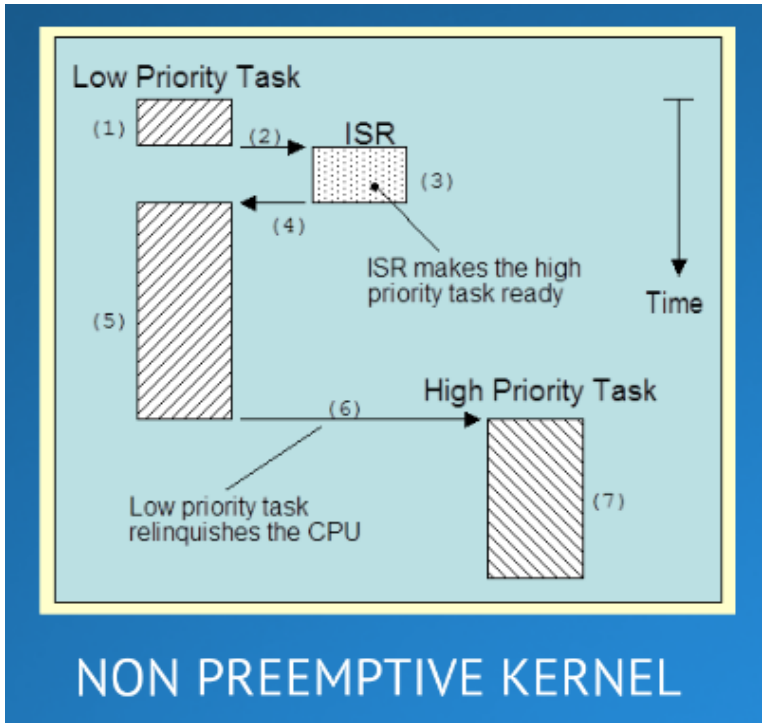
- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
    - Estados Definidos
  - ***Tipo RTOS***
  - Reentrancia
  - Eventos
    - ECB

- TIPOS



RTOS - Generalidad

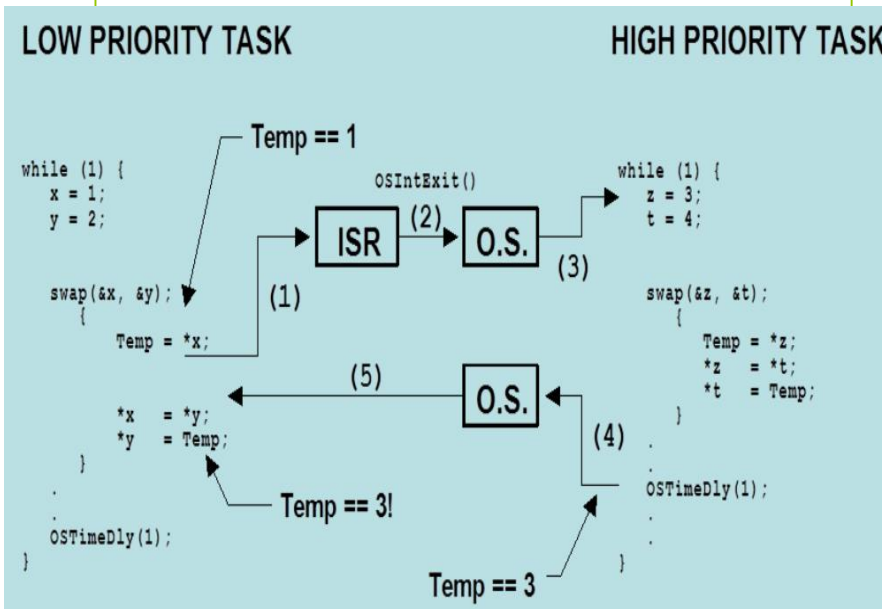




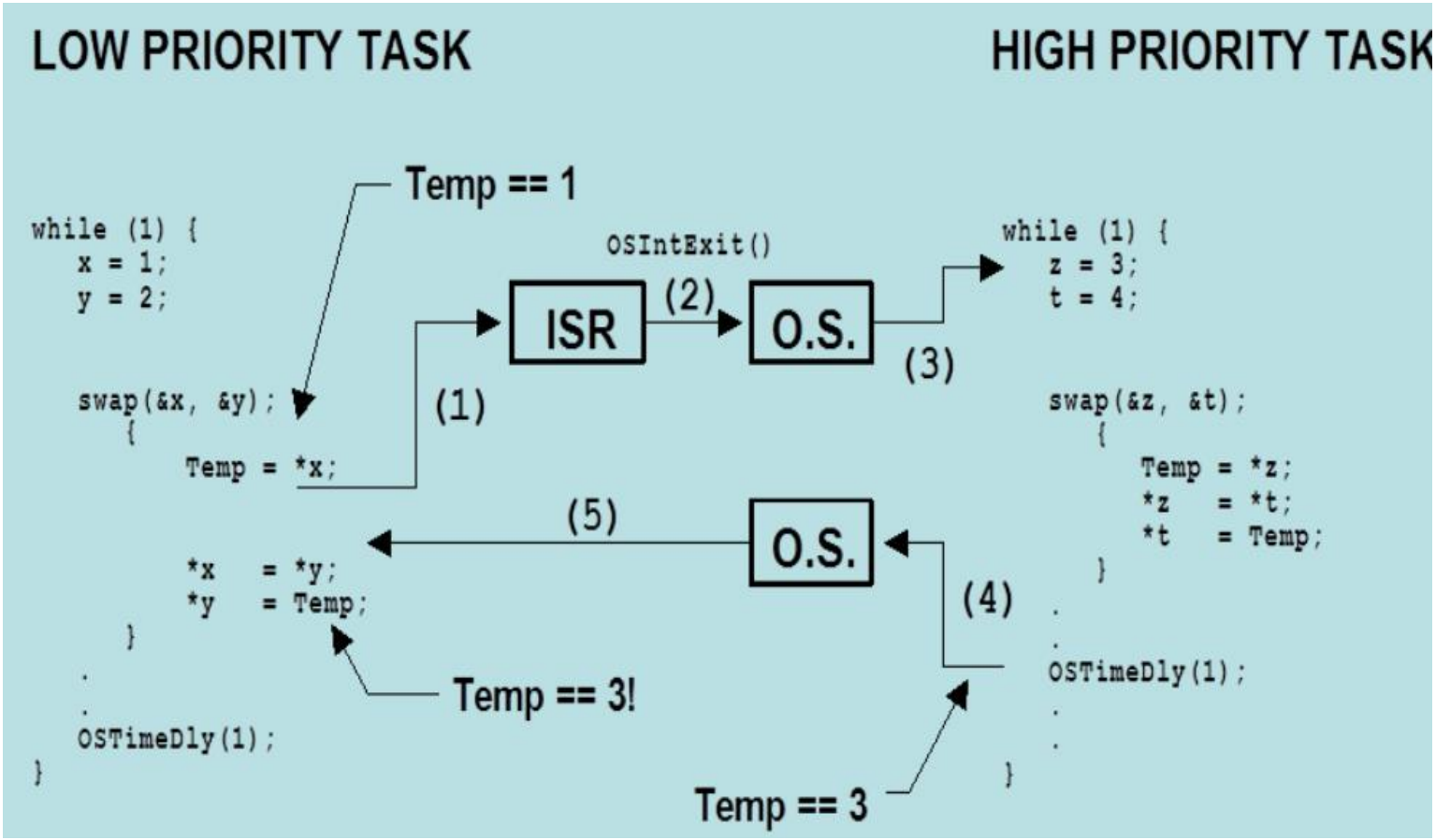
RTOS - Generalidad

- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
    - Estados Definidos
  - Preemptive
  - ***Reentrancia***
  - Eventos
    - ECB

- Reentrancia



RTOS - Generalidad



RTOS - Generalidad

- Se introducen conceptos
  - Tareas
    - TCB
    - Stack
    - Estados Definidos
  - Preemptive
  - Reentrancia
  - ***Eventos***
    - ECB

- Eventos
  - Semáforos
    - Binarios
    - MUTEX
    - Contadores
  - Mailbox
  - Queues

Requisitos de Hardware

- Memoria RAM
  - Para almacenar las estructuras de control del RTOS.
    - TCB
    - ECB
    - Stack
- Memoria ROM
  - Para almacenar la Aplicación, el PORT, y las funcionalidades el RTOS.
- Timer
  - Para generar la base de tiempo del RTOS

Opciones RTOS

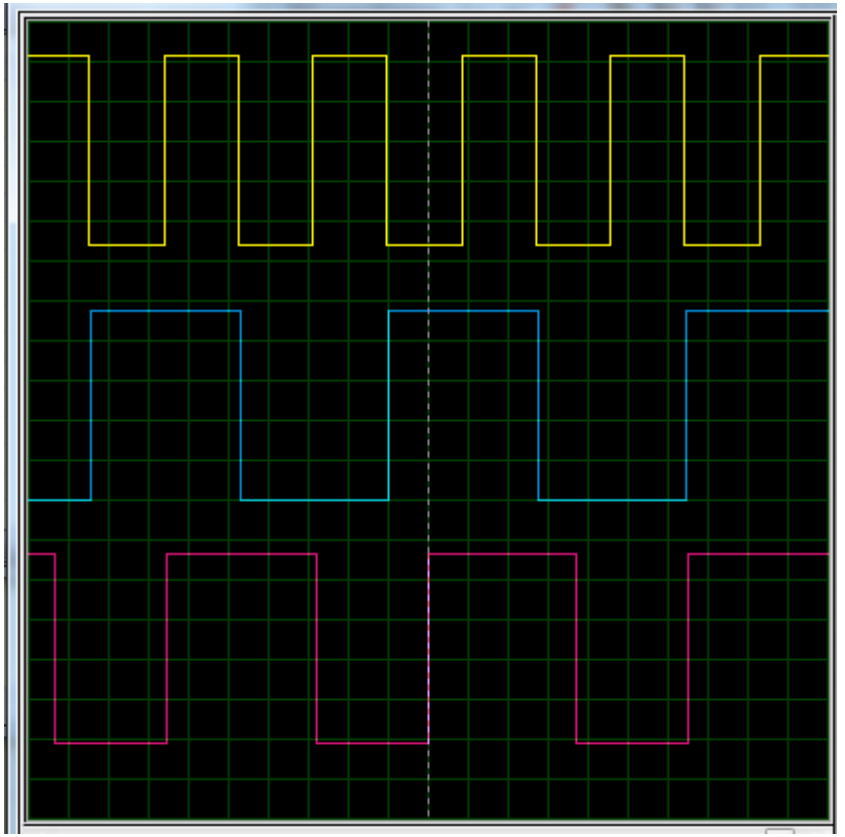
- uCOS III
  - https://www.micrium.com/rtos/kernels/
- freeRTOS
  - https://www.freertos.org/
- AVIX RT
  - http://www.avix-rt.com/
- THREAD**X RTOS**
  - https://rtos.com/

Ejemplo 1

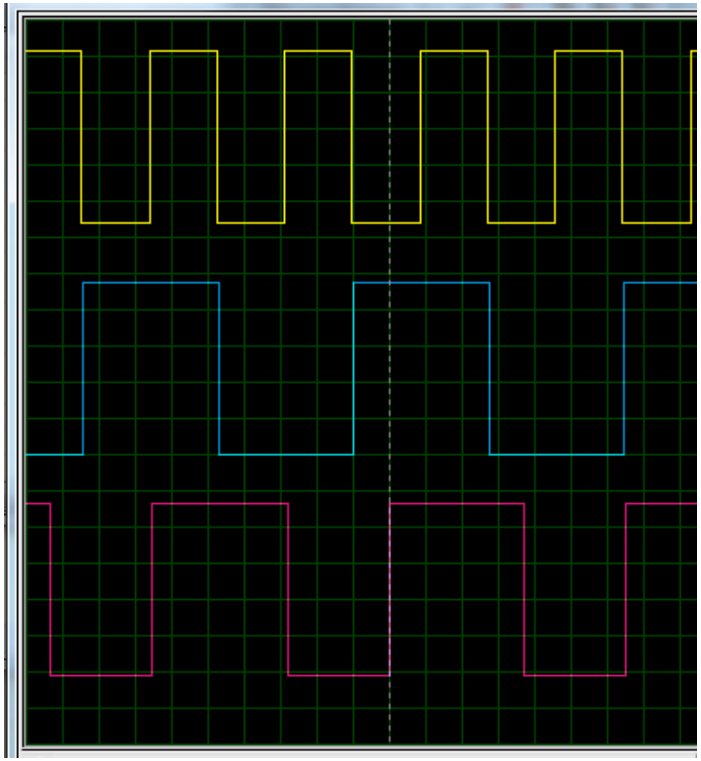
- Implementar un Sistema Embebido que controle tres secuencias temporales en salidas digitales.
- Usar topología Super Loop.
- El control de tiempo se realiza con espera pasiva.

Secuencia

- Secuencia 1:
  - Alto : 1mS
  - Bajo : 1mS
- Secuencia 2:
  - Alto : 2mS
  - Bajo : 2mS
- Secuencia 3:
  - Alto : 3mS
  - BAjo: 4mS



Secuencia Solución



```
// SUPER loop
for(;;){
    switch(cont1){
        case 2: salidaLed1 = 1;
                break;
        case 4: salidaLed1 = 0;
                cont1 = 0;
                break;
    }
    switch(cont2){
        case 4: salidaLed2 = 1;
                break;
        case 8: salidaLed2 = 0;
                cont2 = 0;
                break;
    }
    switch(cont3){
        case 3: salidaLed3 = 1;
                break;
        case 7: salidaLed3 = 0;
                cont3 = 0;
                break;
    }
    Delay10TCYx(20L);
    cont1++;
    cont2++;
    cont3++;
}
```

Espera

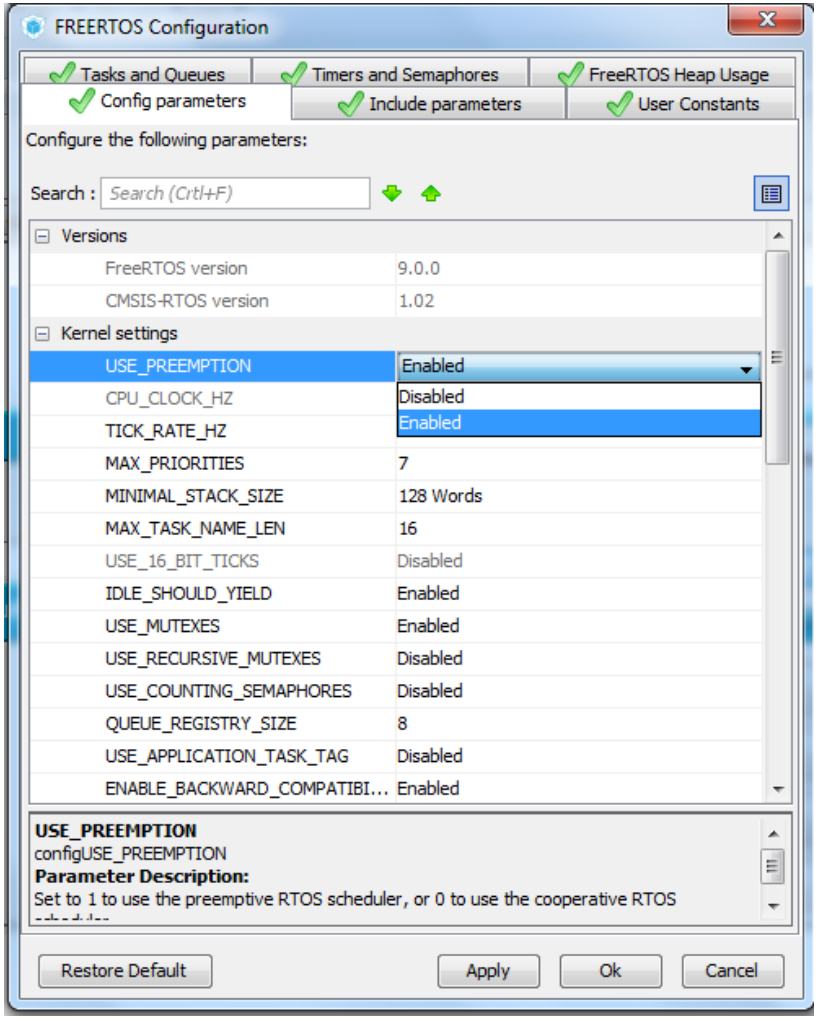
Solución - RTOS

- Configuración Inicial del RTOS
- Debo Crear las Tareas
  - Requieren Task Control Block - TCB.
  - Requieren STACK.
  - Se deben definir Prioridades.
- Se deben conocer los estados posibles.
  - Se debe tener presente en el diseño el tiempo que necesita el RTOS para funcionar.
- Se deben usar SERVICIOS del KERNEL.

Configuración Inicial

- Se debe definir la estructura inicial del sistema RTOS.
  - Activar / Desactivar funcionalidades
  - Establecer la cantidad de Tareas que se usaran
  - Establecer la cantidad de Eventos que se utilizaran
  - Definir el modo de trabajo.
    - Preemptive/Non Preemptive/Round Robin

Configuración Inicial freeRTOS

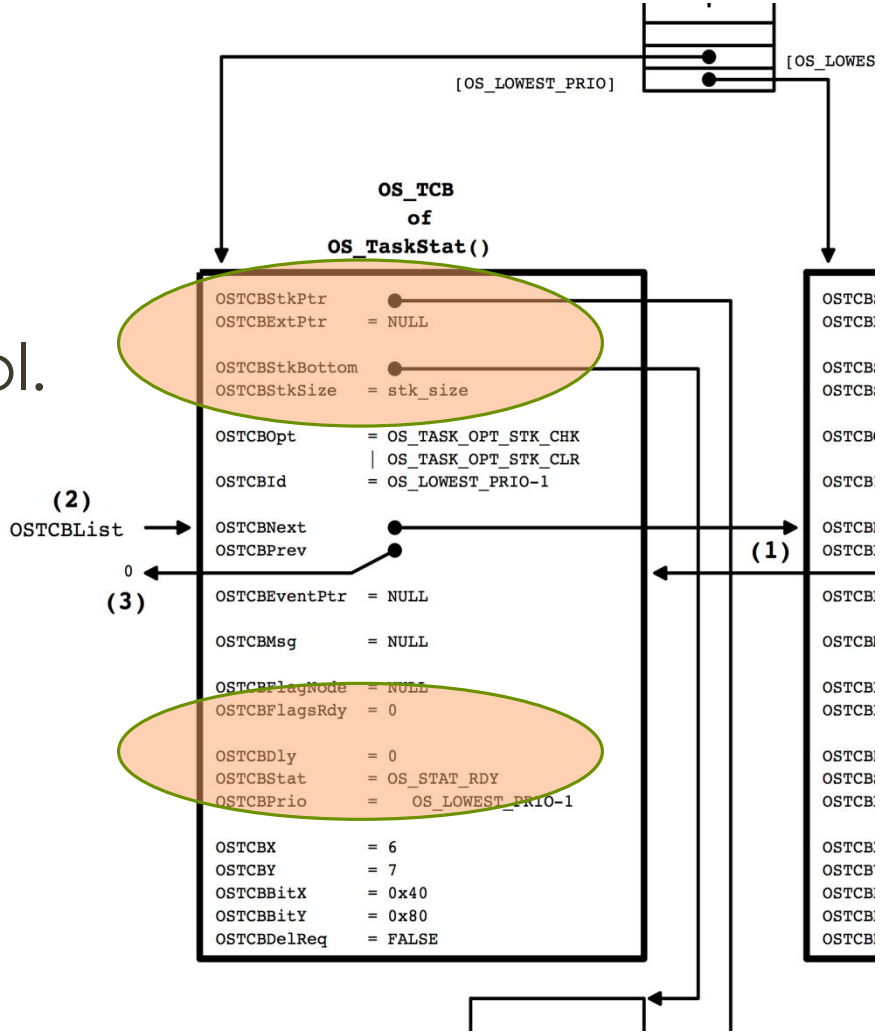


Configuración Inicial uCOS-II

```
#define OS_MAX_TASKS             8L    /* Max. number of tasks in your application ...                */
                                        /* ... MUST be >= 2                                            */

#define OS_LOWEST_PRIO          15L    /* Defines the lowest priority that can be assigned ...        */
                                        /* ... MUST NEVER be higher than 63!                           */

#define OS_TASK_IDLE_STK_SIZE  100L    /* Idle task stack size (# of OS_STK wide entries)             */

#define OS_TASK_STAT_EN          0     /* Enable (1) or Disable(0) the statistics task               */
#define OS_TASK_STAT_STK_SIZE  100L    /* Statistics task stack size (# of OS_STK wide entries)       */

#define OS_ARG_CHK_EN            1     /* Enable (1) or Disable (0) argument checking                */
#define OS_CPU_HOOKS_EN          1     /* uC/OS-II hooks are found in the processor port files       */


                                        /* ---------------------- EVENT FLAGS ---------------------- */
#define OS_FLAG_EN               0     /* Enable (1) or Disable (0) code generation for EVENT FLAGS   */
#define OS_FLAG_WAIT_CLR_EN      1     /* Include code for Wait on Clear EVENT FLAGS                  */
#define OS_FLAG_ACCEPT_EN        1     /*     Include code for OSFlagAccept()                         */
#define OS_FLAG_DEL_EN           1     /*     Include code for OSFlagDel()                            */
#define OS_FLAG_QUERY_EN         1     /*     Include code for OSFlagQuery()                          */


                                        /* -------------------- MESSAGE MAILBOXES -------------------- */
#define OS_MBOX_EN               0           /* Enable (1) or Disable (0) code generation for MAILBOXES
#define OS_MBOX_ACCEPT_EN        1           /*     Include code for OSMboxAccept()
#define OS_MBOX_DEL_EN           1           /*     Include code for OSMboxDel()
#define OS_MBOX_POST_EN          1           /*     Include code for OSMboxPost()
#define OS_MBOX_POST_OPT_EN      1           /*     Include code for OSMboxPostOpt()
#define OS_MBOX_QUERY_EN         1           /*     Include code for OSMboxQuery()
```

Task Control Block TCB

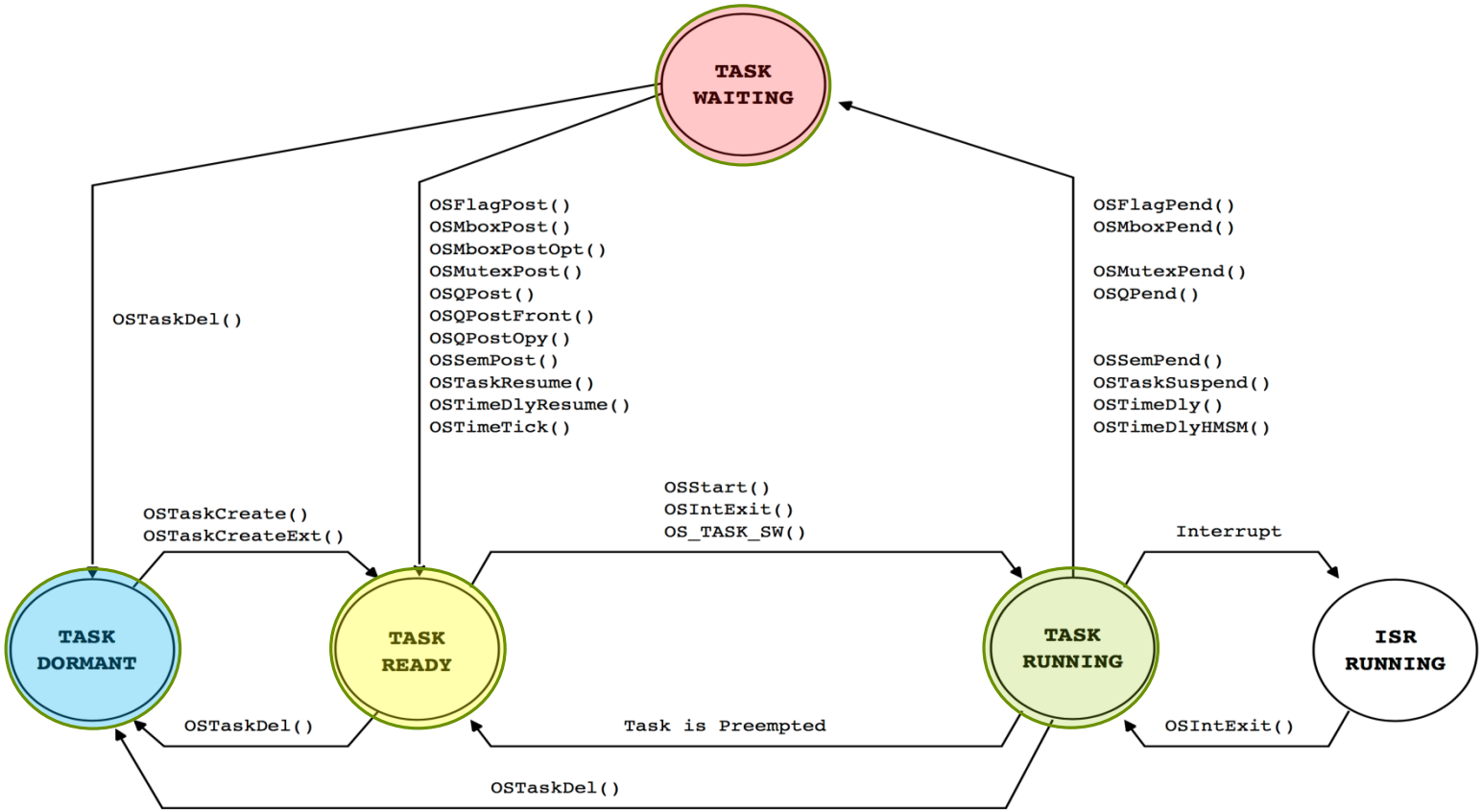
- Cada Tarea a ser usada requiere una estructura de control.
  - Estado
  - Timeout
  - Tamaño Stack
    - Punteros STACK
  - Prioridad



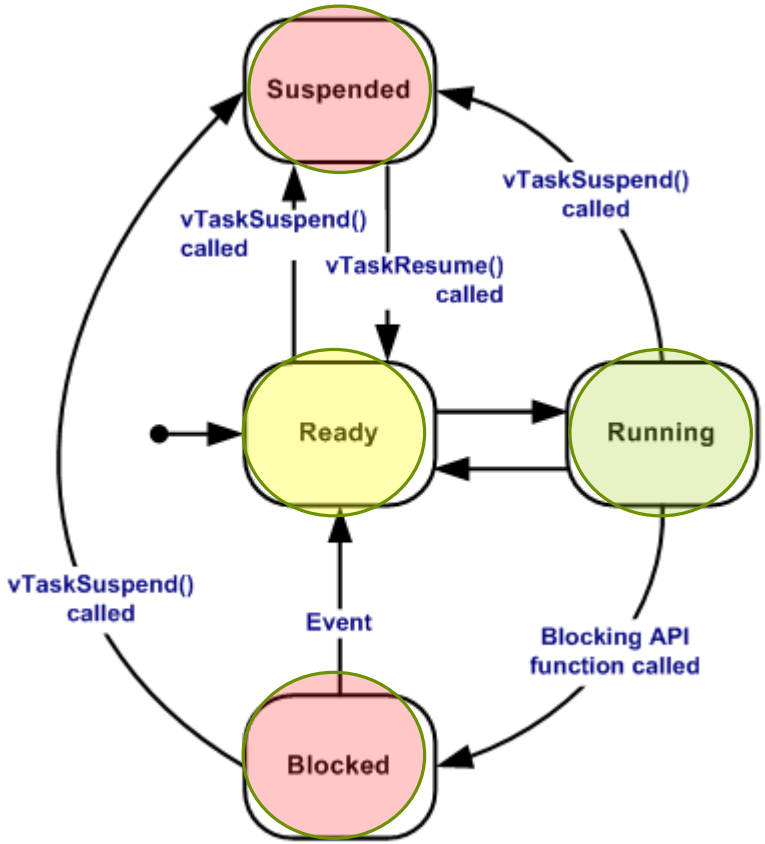
Estados Definidos RTOS

- Es deseable conocer como RTOS administra el uso del CPU por parte de los distintas Tareas.
- El uso de los Servicios del RTOS determinará en que estado estará una Tarea.
  - Ready → Espera para usar el CPU
  - Run → Usa el CPU
  - Wait – Block - Suspend → Espera un timeout o la llegada de un evento.

Estados Definidos RTOS



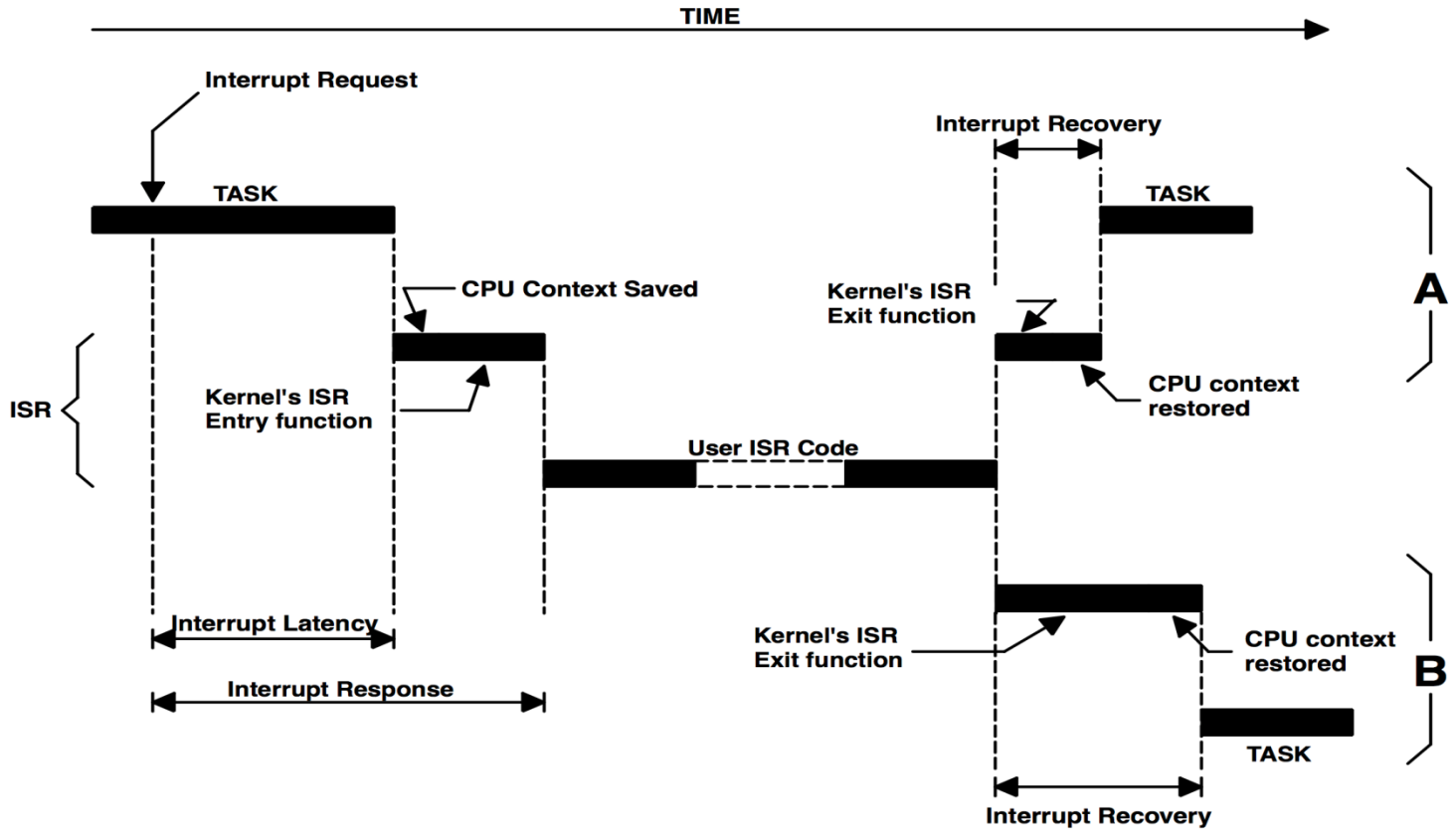
Estados Definidos RTOS



Cambio de Contexto

- Es el proceso en el cual cambia la tarea que usará el CPU.
- El encargado de dicha administración es el Scheduler.
  - Depende del modo de trabajo.
  - Las prioridades de las tareas en estado Ready.
  - Resultan del Timer Tick.
  - Resultan del uso de un Servicio.

Cambio de Contexto



Creación de Tareas

- Debe estar definida en la configuración inicial la cantidad de Tareas
  - Reserva inicial de TCB.
- Se deben definir las prioridades.
- Se debe establecer el tamaño del STACK de cada Tarea.

Creación de Tareas freeRTOS

```
  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,   (const portCHAR *)"Blink"   // A name just for humans
    ,   128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,   NULL
    ,   2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the
    ,   NULL );

  xTaskCreate(
    TaskAnalogRead
    ,   (const portCHAR *) "AnalogRead"
    ,   128  // Stack size
    ,   NULL
    ,   1  // Priority
    ,   NULL );
}
```

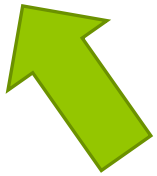
Creación de Tareas freeRTOS

```
/* definition and creation of myBinarySem02 */
osSemaphoreDef(myBinarySem02);
myBinarySem02Handle = osSemaphoreCreate(osSemaphor

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* Create the thread(s) */
/* definition and creation of defaultTask
osThreadDef(defaultTask, StartDefaultTask sPriorityNormal, 0, 128);
defaultTaskHandle = osThreadCreate(osThre defaultTask), NULL);

/* definition and creation of myTask02 */
osThreadDef(myTask02, StartTask02, osPriorityLow, 0, 128);
myTask02Handle = osThreadCreate(osThread(myTask02), NU );

/* definition and creation of myTask03 */
osThreadDef(myTask03, StartTask03, osPriorityBelowNor , 0, 128);
myTask03Handle = osThreadCreate(osThread(myTask03), N );
```

```
/* StartTask02 function */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */

  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_14);

  }
  /* USER CODE END StartTask02 */
}
```

Creación de Tareas

```
        OSTaskCreate(Task1, (void *)0, &Task1Stk[0], task1PRIO);
        OSTaskCreate(Task2, (void *)0, &Task2Stk[0], task2PRIO);
        OSTaskCreate(Task3, (void *)0, &Task3Stk[0], task3PRIO);

//-----------------------------------------------
void Task1(void *pdata)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif

    for(;;)
    {
        salidaLed1 = 0;
        OSTimeDly(2);    // un tick --> 10mSeg
        salidaLed1 = 1;
        OSTimeDly(2);
    }
}
```



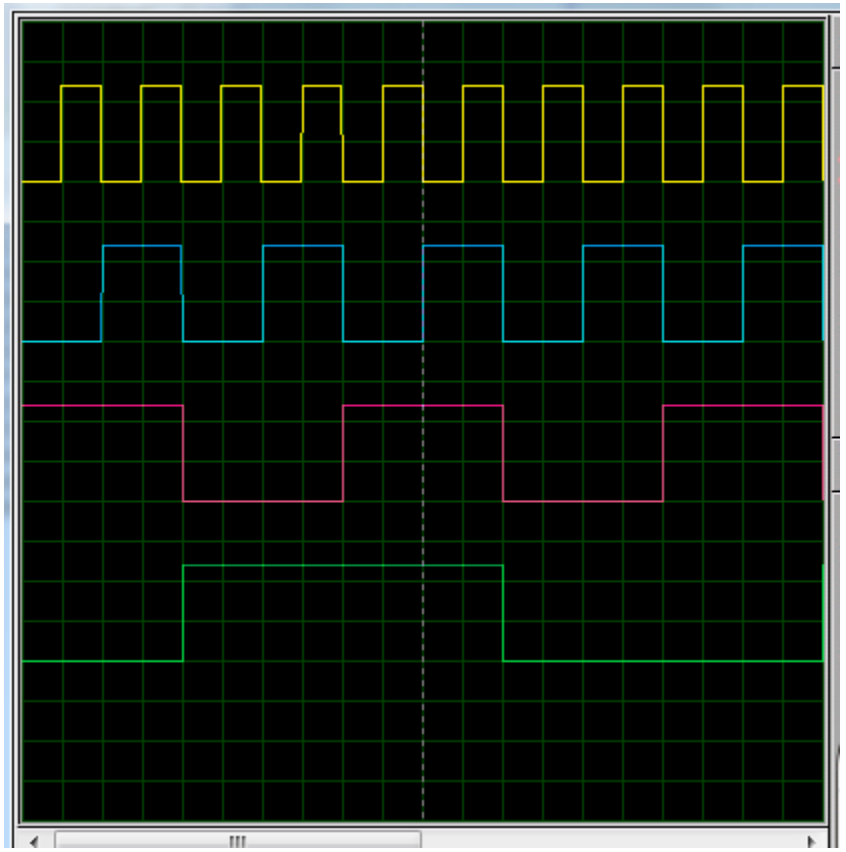
Ejemplo 1 RTOS

Ejemplo 2

- Implementar un programa que genere una secuencia en salidas digitales, garantizando la evolución temporal de cada una de ellas.
  - Las salidas deben estar ***Sincronizadas*** con la salida 1.
- Usar topología RTOS.

Secuencia Temporal

- Posible Solución
- Sincronización con Semáforos Binarios.

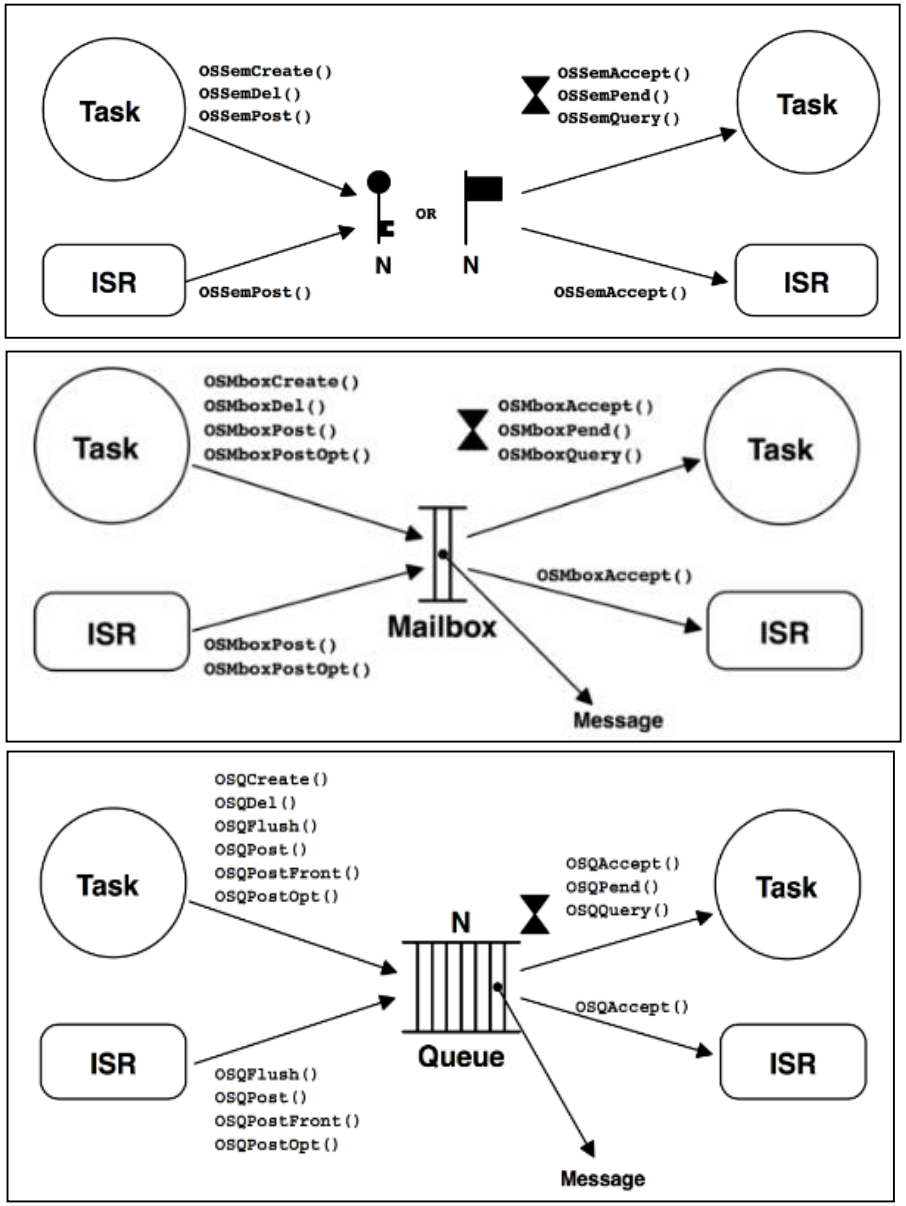


Sincronización

- Se utilizan entidades denominadas EVENTOS.
- Requieren estructuras de control denominadas ECB.
- Precauciones.
  - Se debe definir correctamente como será la relación entre Tareas.
    - Evitar el “deadlock”.
    - Usar timeout para detectar esta instancia.

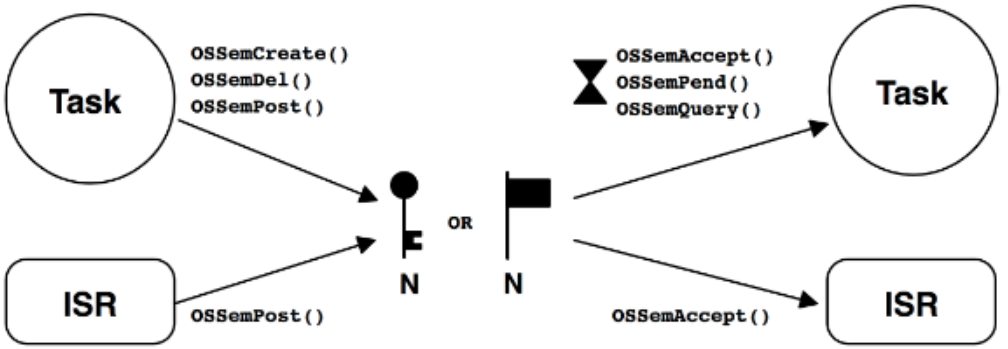
Eventos

- Semáforos
- Mailbox
- Queues



Eventos - Semáforos

- Semáforos Binarios
- Semáforos Contadores
- Semáforos Mutex



```
//-----------------------------------
void Task1(void *pdata)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif

    for(;;)
    {
        salidaLed_1 = ~salidaLed_1;
        if(salidaLed_1==0)
            OSSemPost(Semaforo1);
        OSTimeDly(10);
    }
}
```

Prioridad 1

```
for(;;)
{
    OSSemPend(Semaforo1,0,&err);
    salidaLed_2 = ~salidaLed_2;
    if(salidaLed_2==0)
        OSSemPost(Semaforo2);
    OSTimeDly(10);
}
```

Prioridad 2

Prioridad 3

```
for(;;)
{
    OSSemPend(Semaforo2,0,&err);
    salidaLed_3 = ~salidaLed_3;
    if(salidaLed_3==0)
        OSSemPost(Semaforo3);
    OSTimeDly(10);
}
```

Prioridad 4

```
INT8U *err;
INT16U timeOut=0;

for(;;)

    OSSemPend(Semaforo3,timeOut,err);
    salidaLed_4 = ~salidaLed_4;
    OSTimeDly(10);
}
```

Ejemplo 2 SEM

Ocurrencia de Deadlock

```
INT16U timeOut = 0;
for(;;)
{
    OSSemPend(Semaforo2,timeOut,error);
    switch(*error){
        case OS_NO_ERR:
            Nop();
            break;
        case OS_TIMEOUT:
            Nop();
            break;
    }

    salidaLed_1 = 1;
    OSSemPost(Semaforo1);
    OSTimeDly(20);
}
```

```
for(;;)
{
    OSSemPend(Semaforo1,0,err);
    salidaLed_2 = 1;
    OSSemPost(Semaforo2);
    OSTimeDly(20);
}
```

Ejemplo 3 DEAD

Ocurrencia de Deadlock

```
INT16U timeOut = 20;
for(;;)
{
    OSSemPend(Semaforo2,timeOut,error);
    switch(*error){
        case OS_NO_ERR:
            Nop();
            break;
        case OS_TIMEOUT:
            Nop();
            break;
    }

    salidaLed_1 = 1;
    OSSemPost(Semaforo1);
    OSTimeDly(20);
}
```

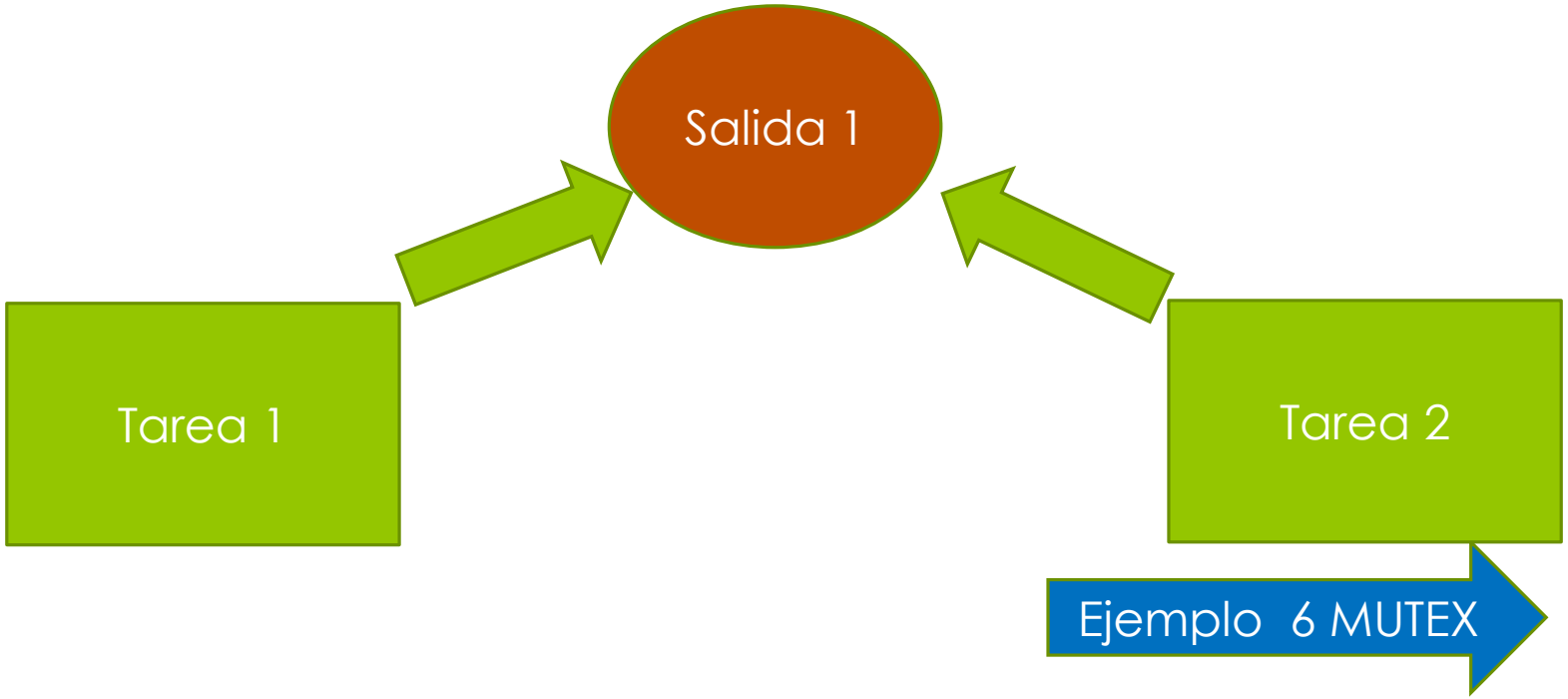
```
for(;;)
{
    OSSemPend(Semaforo1,0,err);
    salidaLed_2 = 1;
    OSSemPost(Semaforo2);
    OSTimeDly(20);
}
```

Eventos - Mutex

- Cuando se requiere usar un recurso/periférico por más de una tarea.
  - Ejemplo SPI
    - ADC
    - LCD
    - MEMORIA

Ejemplo

- Dos tareas desean acceder a un puerto I/O.



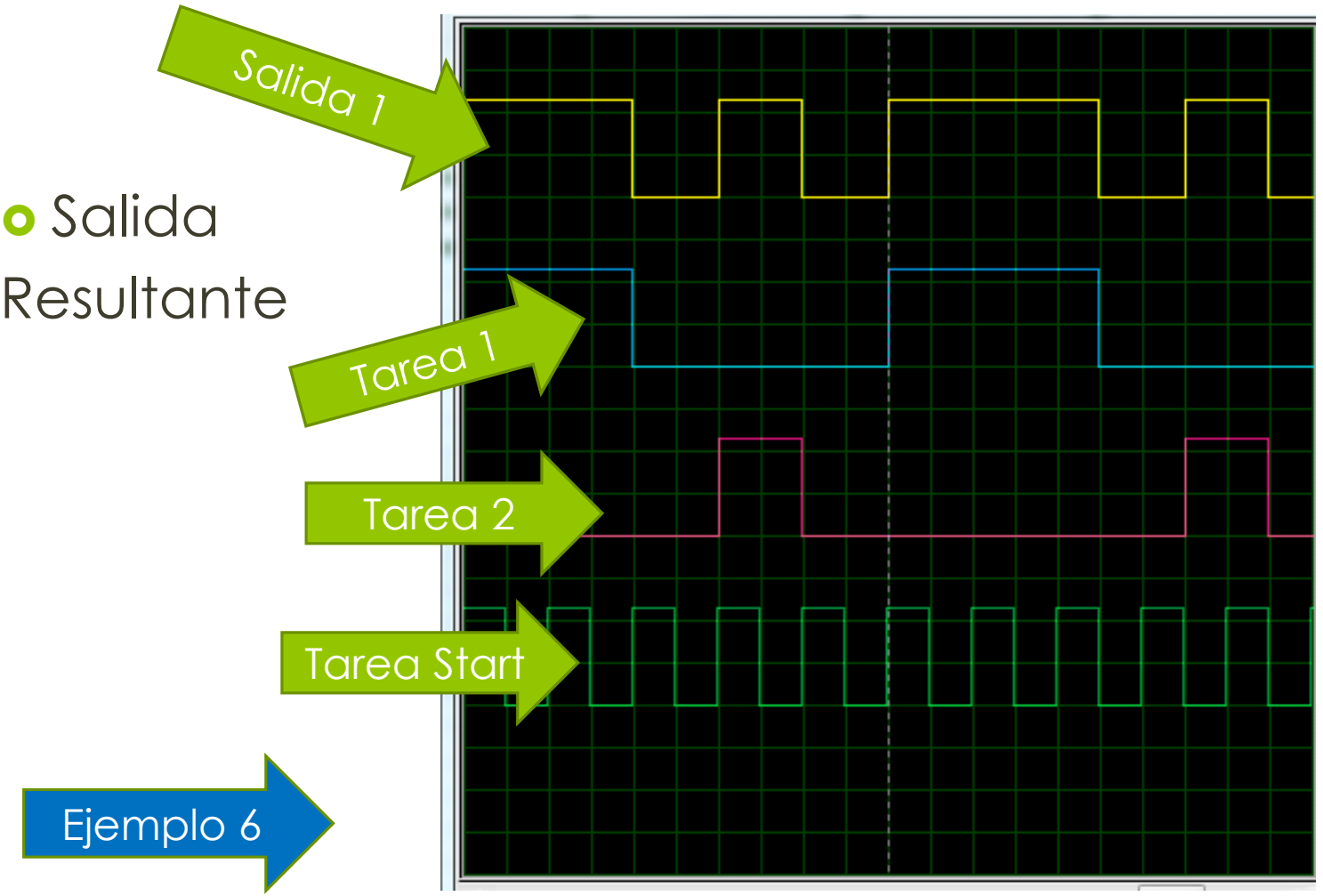
Ejemplo

- Dos tareas desean acceder a un puerto I/O.

```
for(;;) {
    OSMutexPend(ADC_Mutex, 0, &err);

    salidaLED_1 = 1;
    salidaLED_2 = 1;
    OSTimeDly(5);

    salidaLED_1 = 0;
    salidaLED_2 = 0;
    OSTimeDly(2);

    OSMutexPost(ADC_Mutex);
}
```

```
for(;;){
    OSMutexPend(ADC_Mutex, 0, &err);

    salidaLED_1 = 1;
    salidaLED_3 = 1;
    OSTimeDly(2);

    salidaLED_1 = 0;
    salidaLED_3 = 0;
    OSTimeDly(2);

    OSMutexPost(ADC_Mutex);
}
```

- Salida Resultante



Ejemplo

- Dos tareas desean acceder a un puerto I/O.
- Usar Activación/ Desactivación de Interrupciones.
- Usar Activación/Desactivación del Scheduler.

Solución

```
for(;;) {
    OS_ENTER_CRITICAL();

    salidaLED_1 = 1;
    salidaLED_2 = 1;

    OS_EXIT_CRITICAL();

    OSTimeDly(2);

    salidaLED_1 = 0;
    salidaLED_2 = 0;
    OSTimeDly(2);
}
```

```
for(;;){
    OSSchedLock();

    salidaLED_1 = 1;
    salidaLED_3 = 1;
    OSTimeDly(5);

    salidaLED_1 = 0;
    salidaLED_3 = 0;
    OSTimeDly(5);

    OSSchedUnlock();
```

Ejemplo 10 CRITICAL

Eventos - Mailbox

- Entidad que permite transferir información entre Tareas.
- Se puede usar para Sincronizar dos Tareas.
- Se asemeja a una variable de tipo global.

Mailbox - Ejemplo

- Control de una salida Digital en base a un valor que resulta de un canal ADC.

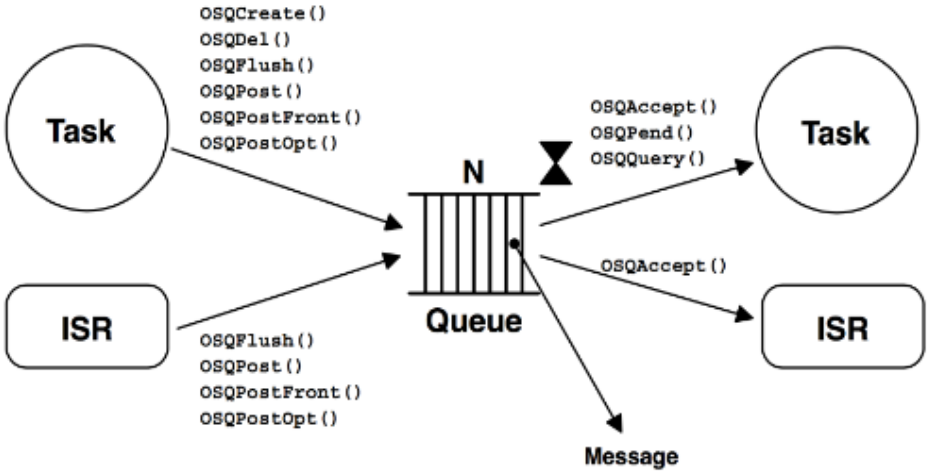
```
for(;;)
{
    OSTimeDly(5);
    uiValorADC = convertirADC();
    OSMboxPost(ADC_Mbox, (void *)&uiValorADC);
}
```

```
{
    rxmsg = OSMboxPend(ADC_Mbox, 0, err);
    inc_P = *rxmsg;

    if(*rxmsg >= 250)
        salidaRELE = 1;
    else
        salidaRELE = 0;

    OSTimeDly(1);
}
```

Ejemplo 5 MBOX

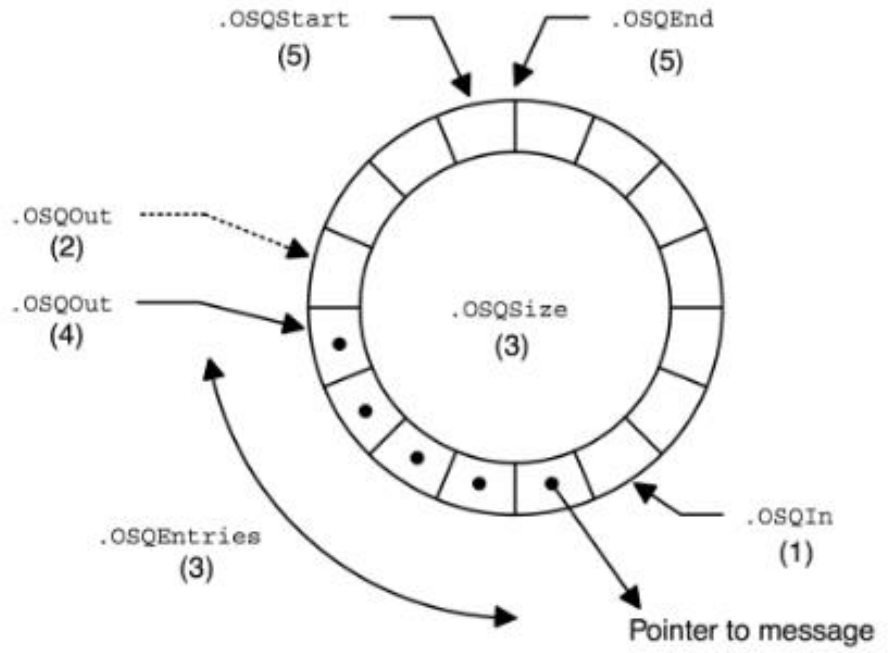
Eventos - Queues

- Entidad que permite transferir una cola de mensajes.
- Es posible almacenar mensajes siguiendo una lógica FIFO o LIFO.



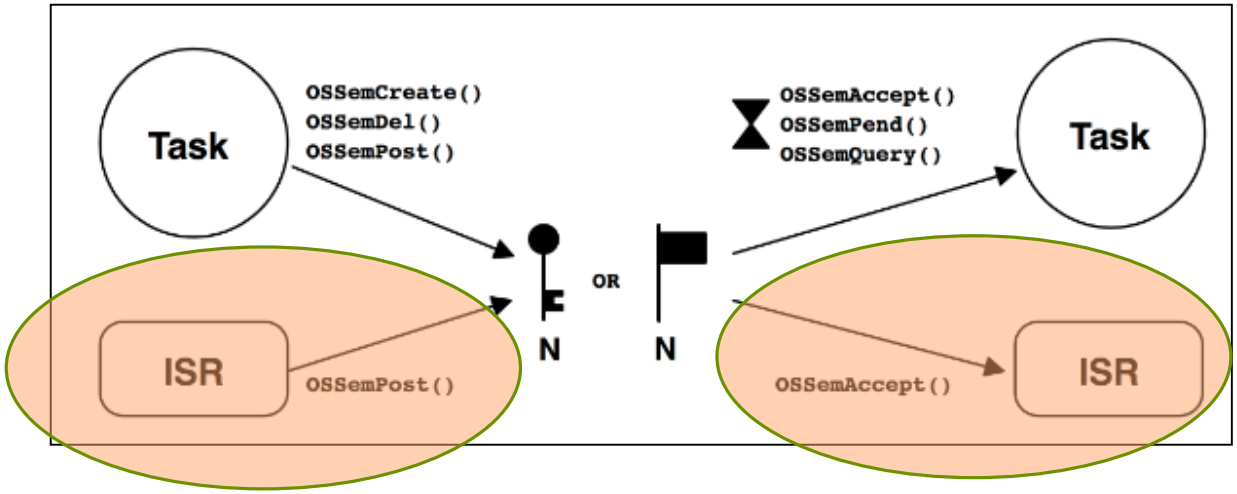
Eventos - Queues

- Se almacenan los mensajes en una estructura de buffer circular.



Sincronización desde ISR

- Es posible usando EVENTOS sincronizar una Tarea con la ocurrencia de una Interrupción.



Reentrancia

- El concepto es crucial al momento de la definiciones de la solución del problema.
- Recordar que existen cambios de contexto durante toda la vida operacional del software desarrollado.

Reentrancia

Baja Prioridad

```
for(;;)
{
    OSTimeDly(5);
    uiValorADC = convertirADC();

    uiValorADC = 222;
    demoraLarga(5);

    OSMboxPost(ADC_Mbox, (void *)&uiValorADC);
}
```

Alta Prioridad

```
for(;;)
{
    OSTimeDly(2);
    uiValorADC = convertirADC();
    uiValorADC = 333;
    OSMboxPost(ADC_Mbox, (void *)&uiValorADC);
}
```

Cambios de Contexto

- Se producen cambios de contexto cuando expira el tiempo establecido para funcionar.
- Se producen cuando una Tarea solicita un Servicio del RTOS.
- Los cambios de contexto pueden provocar instancias de Deadlock si no son tenidos en cuenta.

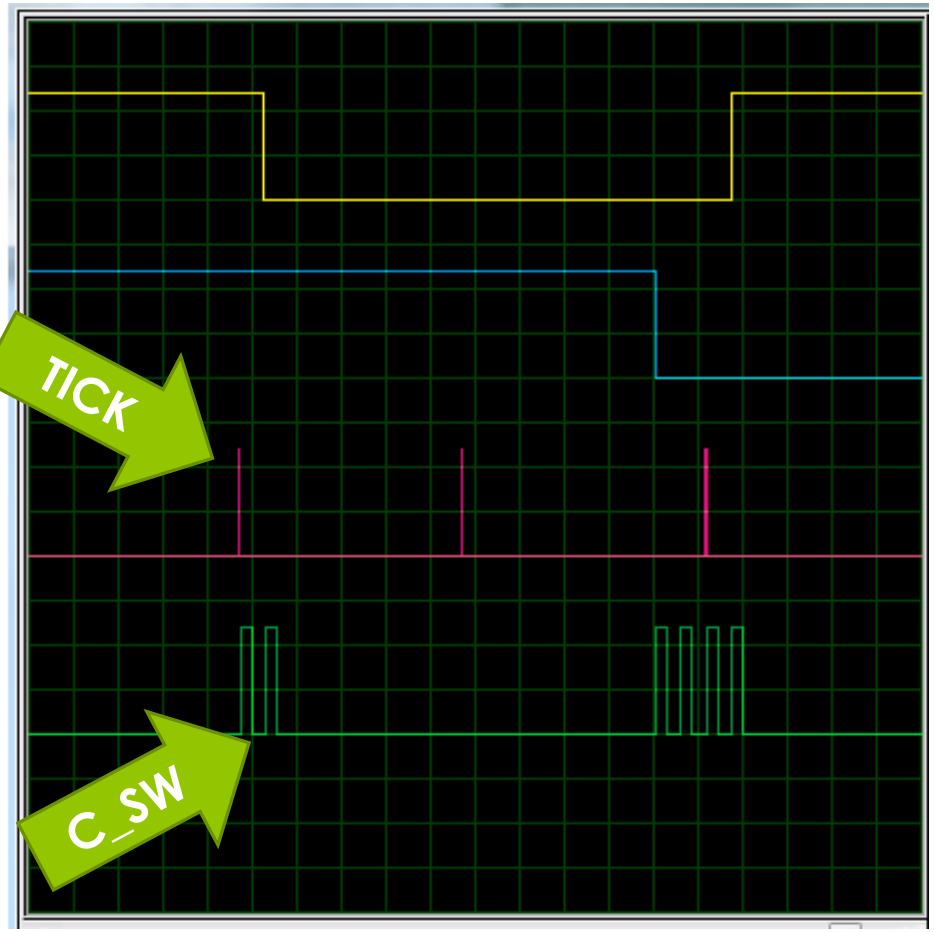
Ejemplo 9 PREEMPTIVE

Cambios de Contexto

- Se puso para el ejemplo un led que se activa y desactiva cuando se produce el Context Switch,
- Se puso un led que se activa cuando se ejecuta la ISR del timer.

```
void CPUInterruptHook(void)
{
    if(INTCONbits.TMR0IF) {

        salidaLed3 = 1;

        INTCONbits.TMR0IF = 0;

        TMR0H = 60;
        TMR0L = 251;

        OSTimeTick();
        salidaLed3 = 0;
    }
}
```

```
void OSTaskSwHook (void)
{
    salidaLed4 = 1;
    Delay10TCYx(10L);
    salidaLed4 = 0;
    Delay10TCYx(10L);
}
```

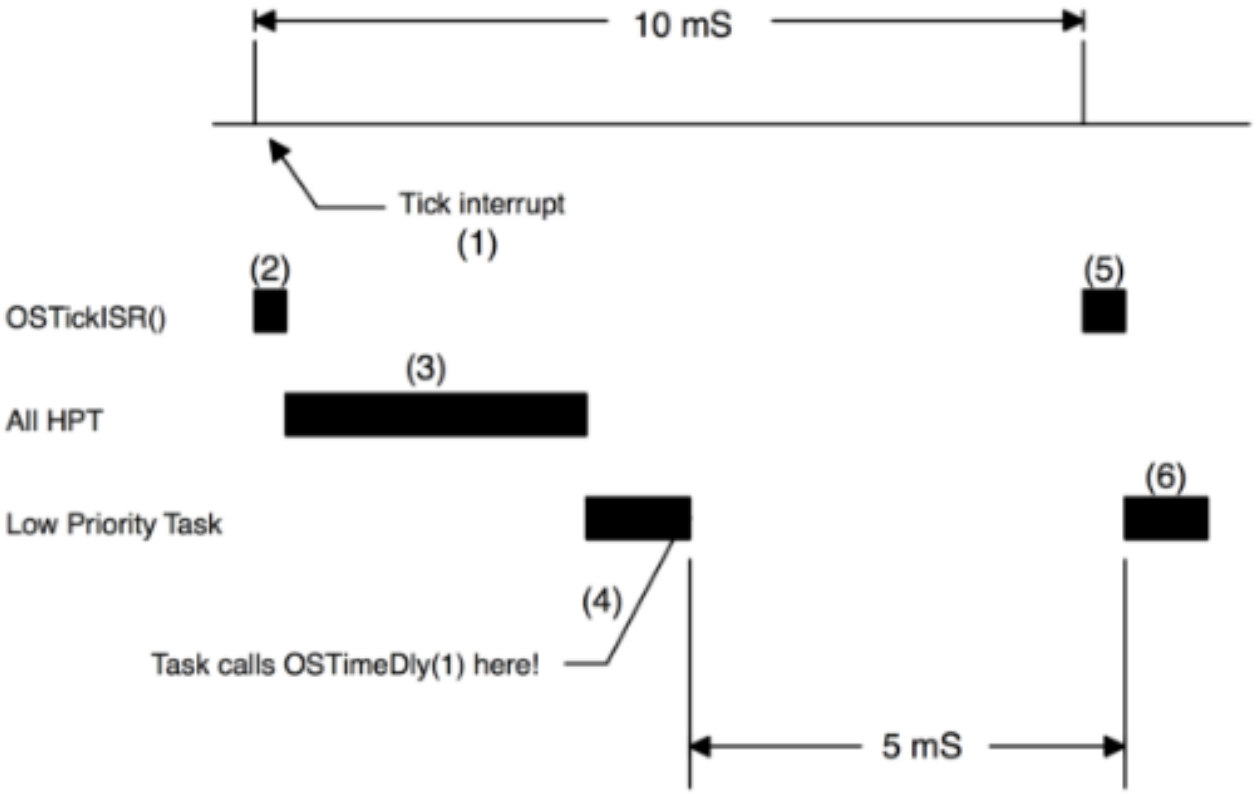


Ejemplo 9

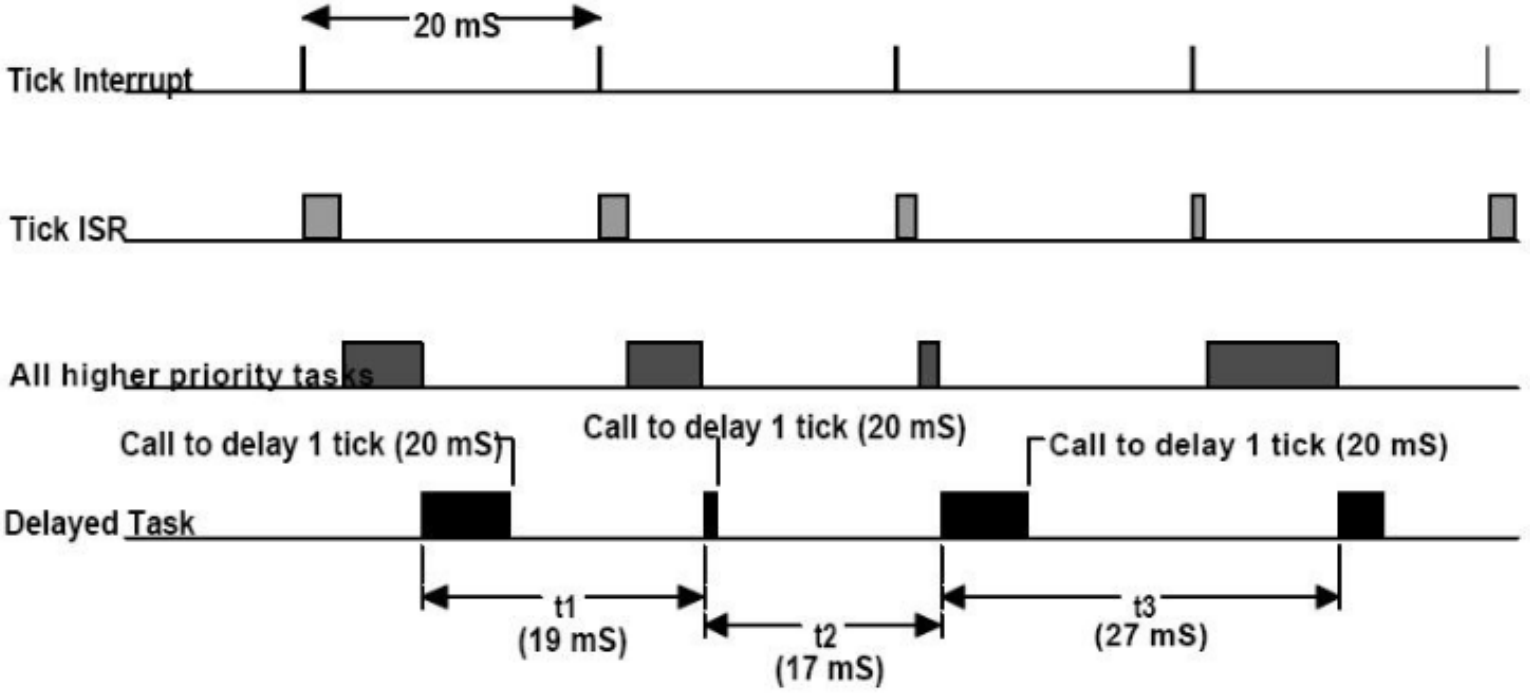
Resolución de Tiempos

- La tarea de MAS Alta prioridad tendrá la posibilidad de ejecutarse dentro del tiempo establecido.
- Las tareas de menor prioridad ocuparán al CPU pudiendo no ejecutarse dentro del tiempo del sistema.
  - Ejecución completa antes de que se produzca un tick del sistema.

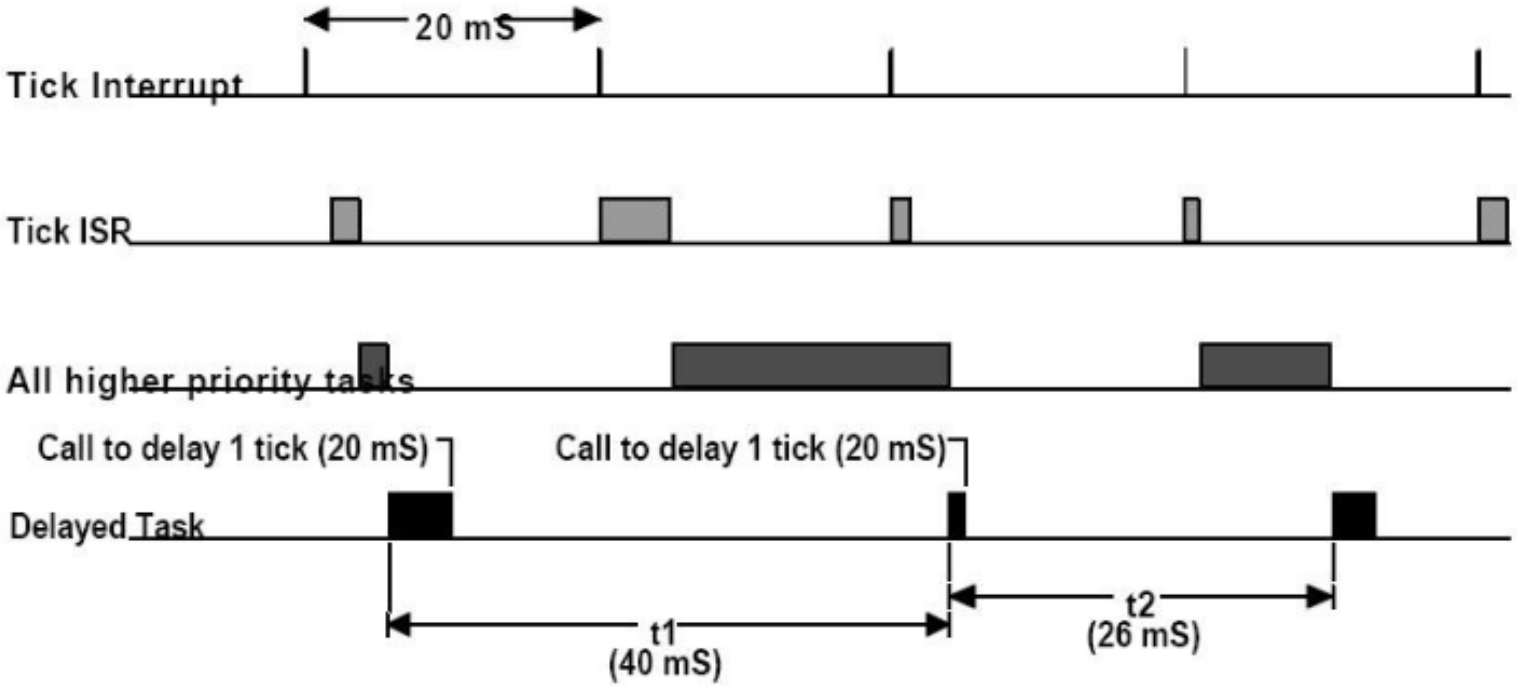
Resolución de Tiempo



Resolución de Tiempo



Resolución de Tiempo



Aspectos Relevantes

- Se debe disponer de una plataforma que posea RAM y ROM acorde al problema.
- Capacidad de administrar en forma dinámica el STACK
- Requerimos un dispositivo que permita controlar el tiempo del sistema.
  - TIMER
- Se debe crear una tarea antes de lanzar el proceso multitarea.

Aspectos Relevantes

- Se deben realizar un análisis del sistema para la correcta definición de las Tareas.
- Los Tareas definidas requieren de asignación de prioridades.
- Es posible reasignar prioridades en forma dinámica.
- Establecer el modo de trabajo del RTOS.
  - Preemptive - Non Premtive

Aspectos Relevantes

- El uso de Eventos para Sincronización requiere de recursos de ROM y RAM.
- Se debe configurar el RTOS antes de comenzar
  - Cantidad de Tareas, Eventos, etc.

Preguntas?

Contacto: MSc. Carlos Centeno
ccenteno@gmail.com

G.In.T.E.A – FRC UTN
http://www.investigacion.frc.utn.edu.ar/gintea/

Fuentes utilizadas

- MPLAB 8.53
- C18 3.02
- uCOS-II
- freeRTOS 8.0.0
- https://www.freertos.org
- https://doc.micrium.com/display/osiidoc
- https://www.st.com/en/development-tools/sw4stm32.html