Laboratorio 12:

Multitarea cooperativa

Programación de sistemas y dispositivos

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

J.M. Mendías
2021

Presentación

- Objetivo: Desarrollar una aplicación multitarea con distintas arquitecturas de múltiples tareas cooperativas.
- El punto de partida es una aplicación que realiza de 7 tareas:
  1. Cada 500 ms, alterna el led que se enciende.
  2. Cada 50 ms, lee el keypad y, si hay una tecla pulsada, envía su scancode a otras tareas.
     - Usa un maibox (variable global + flags) como mecanismo de comunicación.
  3. Cada segundo, muestra por la UART0 la hora leída del RTC.
  4. Cada 10 s, muestra por la UART0 el número de ticks transcurridos desde el inicio.
     - Lleva localmente la cuenta de ticks.
  5. Muestra por la UART0 cada una de las teclas pulsadas.
  6. Muestra en el display 7-segmentos cada una de las teclas pulsadas.
  7. Indica por la UART0 la detección (por interrupción) de la presión de un pulsador.
     - La RTI activa un flag cada vez que detecta presión.
- Adicionalmente, todas las tareas envían un mensaje a la UART0 a su inicio.
- Esta aplicación deberá ampliarse con 2 tareas adicionales:
  8. Muestra en el LCD cada una de las teclas pulsadas.
  9. Cada segundo, muestra en el LCD los segundos transcurridos desde el inicio.

Arquitectura fore/background

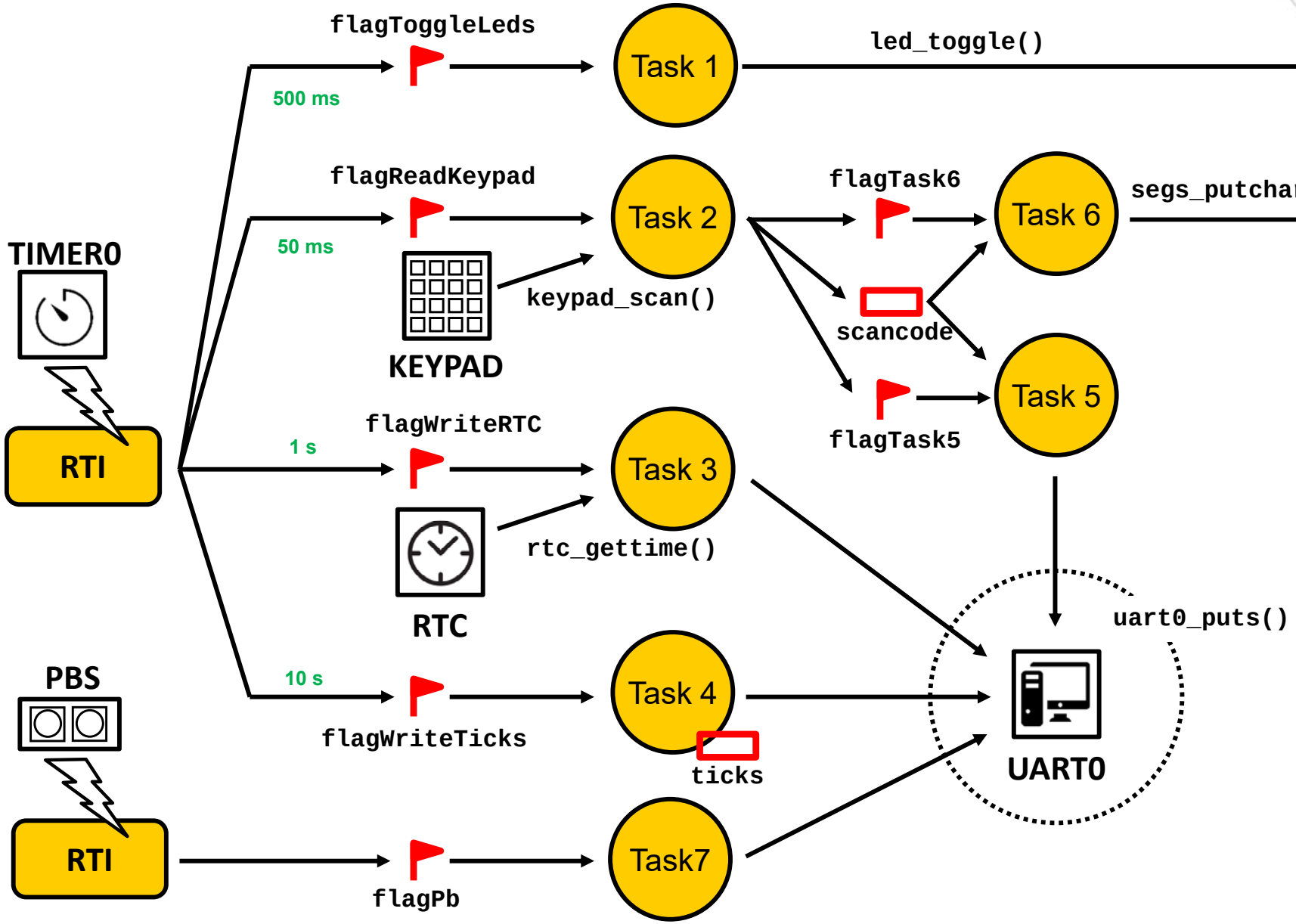
arquitectura software

- Las tareas se ejecutan en una arquitectura foreground/background.
  - Una hebra en foreground (RTI por temporizador) se encarga de señalizar mediante distintos flags el transcurso de cada periodo de tiempo (50 ms, 500 ms, 1s y 10 s).
    - Su disparo, además, despierta al procesador.
  - Otra hebra en foreground (RTI por pulsador) se encarga de señalizar mediante un flag la presión de cualquier pulsador.
    - Su disparo, también despierta al procesador.
  - Una única hebra en background (función main) chequea secuencialmente los flags y ejecuta las tareas que correspondan en cada instante.
    - Dado que las tareas son ejecutadas sin expropiación, los recursos compartidos son accedidos secuencialmente y no es necesario protegerlos.
    - Tras chequear todos los flags y ejecutar las tareas que correspondan, suspende al procesador.

J.M. Mendias
2021

Arquitectura fore/background

arquitectura software



J.M. Mendías
2021

Arquitectura fore/background

declaraciones

```
#define TICKS_PER_SEC   (100)
```
Entre tick y tick transcurren 10 ms

```
uint8 scancode;
boolean flagTask5;
boolean flagTask6;

volatile boolean flagPb;
volatile boolean flagToggleLeds;
volatile boolean flagReadKeypad;
volatile boolean flagWriteRTC;
volatile boolean flagWriteTicks;
```
Declara los recursos de sincronización y comunicación

```
void Task1( void );
void Task2( void );
...
void Task7( void );
```
Declara tareas

```
void isr_pb( void ) __attribute__ ((interrupt ("IRQ")));
void isr_tick( void ) __attribute__ ((interrupt ("IRQ")));
```
RTI

Arquitectura fore/background

programa principal (hebra background)

```
void main( void )
{
  sys_init();
  timers_init();
  ...
  keypad_init();

  uart0_puts("\n\n Ejecutando una aplicación foreground/background\n");
  uart0_puts(" ----------------------------------------\n\n");

  flagTask5      = FALSE;
  ...
  flagWriteTicks = FALSE;

  Task1();
  Task2();
  ...
  Task7();

  pbs_open( isr_pb );
  timer0_open_tick( isr_tick, TICKS_PER_SEC );
  ...
```

Inicializa dispositivos

Inicializa flags

Ejecuta por primera vez las funciones para inicializarlas

Instala RTI

J.M. Mendías
2021

Arquitectura fore/background

programa principal (hebra background)

```
...
while( 1 )
{
  sleep();                          Entra en estado IDLE, sale por interrupción
  if( flagToggleLeds )
  {
    flagToggleLeds = FALSE;
    Task1();
  }
  if( flagReadKeypad )              Chequea el estado del flag
  {
    flagReadKeypad = FALSE;         Las tareas se ejecutan por orden de aparición en el código
    Task2();
  }
  ...
  if( flagPb )
  {
    flagPb = FALSE;
    Task7();
  }
}
}
```

Arquitectura fore/background

RTI (hebras foreground)

```
void isr_tick( void )
{
  static uint16 cont5ticks  = 5;
  static uint16 cont50ticks = 50;
  ...

  if( !(--cont5ticks) )
  {
    cont10ticks = 5;
    flagReadKeypad = TRUE;
  }
  if(!(--cont50ticks) )
  {
    cont50ticks = 50;
    flagToggleLeds = TRUE;
  }
  if(!(--cont100ticks) )
  ...
  I_ISPC = BIT_TIMER0;
};
```

Contadores modulares de ticks

Activa flag cada 5 ticks (50 ms)

Activa flag cada 50 ticks (500 ms)

```
void isr_pb( void )
{
  flagPb = TRUE;
  EXTINTPND = BIT_RIGHTPB | BIT_LEFTPB;
  I_ISPC = BIT_PB;
}
```

Activa flag cada vez que se pulsa cualquier pulsador

Arquitectura fore/background

tarea 1: alternancia de leds

- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagToggleLeds
  - La RTI del timer0 (hebra en foreground) activa este flag cada 50 ticks (500 ms)

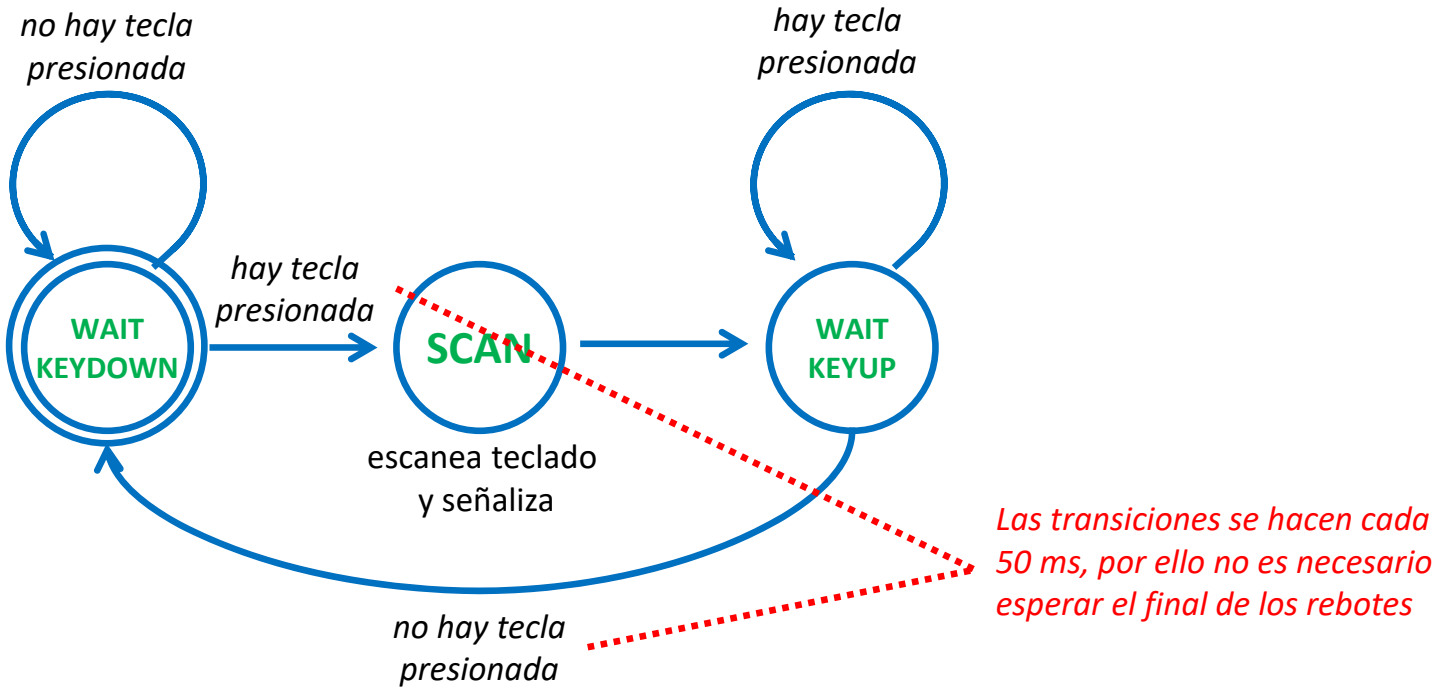
```
void Task1( void )
{
  static boolean init = TRUE;  ····· Testigo que permite distinguir si es la primera vez que se ejecuta

  if( init )  ····· Si es la primera vez que se ejecuta...
  {
    init = FALSE;  ····· ... desactiva el testigo
    uart0_puts( " Task 1: iniciada.\n" );  ····· ... muestra un mensaje por la UART0
    led_on( LEFT_LED );
    led_off( RIGHT_LED );  } ... inicializa leds
  }
  else
  {
    led_toggle( LEFT_LED );
    led_toggle( RIGHT_LED );  } Conmuta el estado de los leds
  }
}
```

J.M. Mendías
2021

Arquitectura fore/background

tarea 2: lectura del keypad y difusión del scancode

- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagReadKeypad
  - La RTI del timer0 (hebra en foreground) activa este flag cada 5 ticks (50 ms)
- La lectura del keypad la realizará una FSM por pooling periódico
  - No puede usarse keypad_getchar() porque el resto de tareas (ejecutadas en la misma hebra) no podrían avanzar a la espera de que el usuario leyera una tecla.



J.M. Mendías
2021

Arquitectura fore/background

tarea 2: lectura del keypad y difusión del scancode

```
void Task2( void )
{
  static boolean init = TRUE;
  static enum { wait_keydown, scan, wait_keyup } state;

  if( init )
  {
    init  = FALSE;
    uart0_puts( " Task 2: iniciada.\n“);
    state = wait_keydown;
  } else switch( state ) {
      case wait_keydown:
        if( keypad_pressed() )
          state = scan;
        break;
      case scan:
        scancode = keypad_scan();
        if( scancode != KEYPAD_FAILURE )
          { flagTask5 = TRUE; flagTask6 = TRUE; }
        state = wait_keyup;
        break;
      case wait_keyup:
        if( !keypad_pressed() )
          state = wait_keydown;
        break;
  }
}
```

Estados de la FSM

Estado inicial de la FSM

Cambia de estado solo si detecta presión

Escanea el teclado 50 ms después de haber detectado la presión: no es necesario esperar el fin de los rebotes de presión

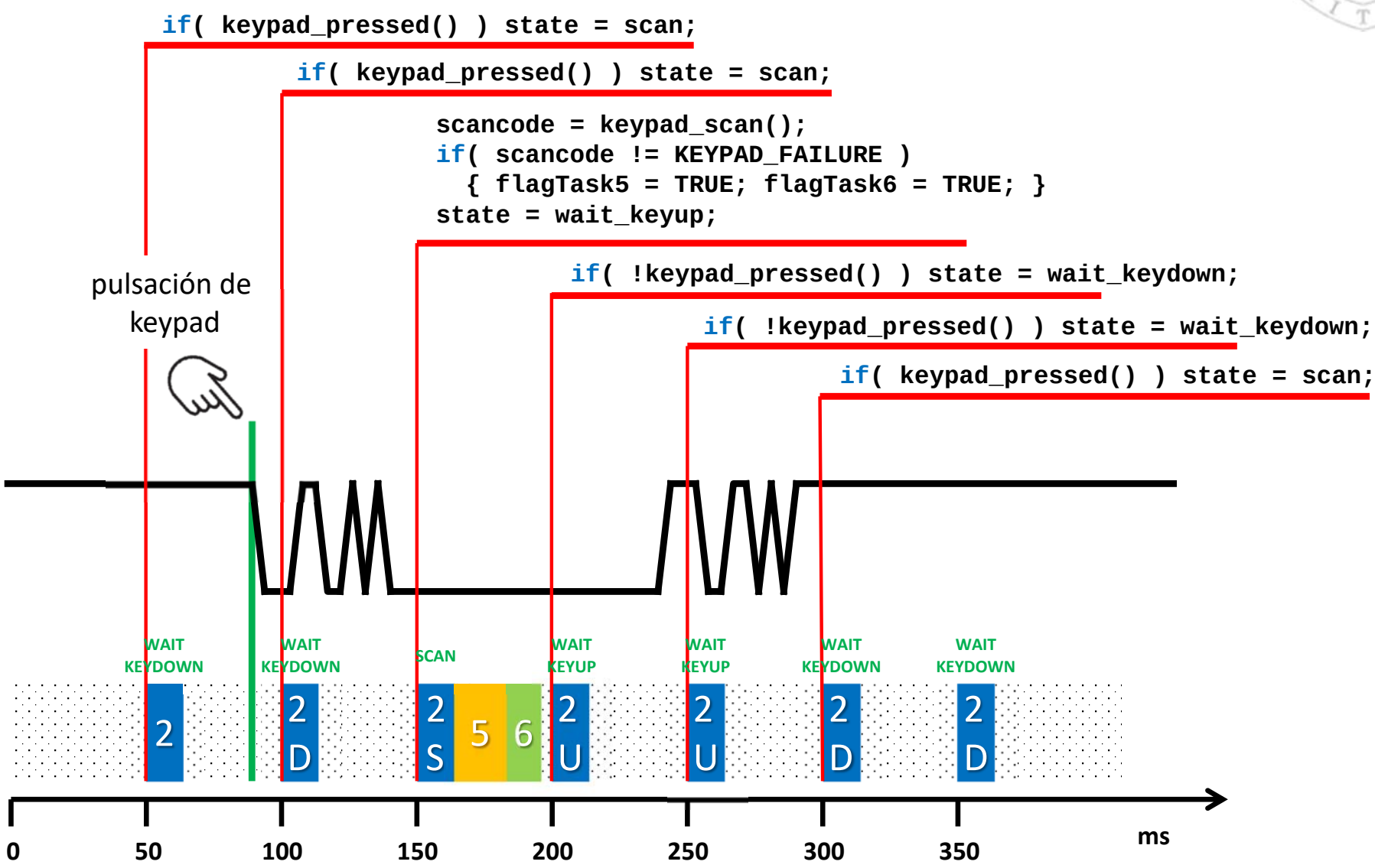
Activa flags si la lectura es válida

Cambia de estado solo si detecta depresión

J.M. Mendías
2021

Arquitectura fore/background

tarea 2: lectura del keypad y difusión del scancode



Arquitectura fore/background

tarea 3: informe de la hora

- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagWriteRTC
  - La RTI del timer0 (hebra en foreground) activa este flag cada 100 ticks (1 s)

```
void Task3( void )
{
  static boolean init = TRUE;
  rtc_time_t rtc_time;

  if( init )
  {
    init = FALSE;
    uart0_puts( " Task 3: iniciada.\n" );
  }
  else
  {
    rtc_gettime( &rtc_time );
    uart0_puts( "  (Task 3) Hora: " );
    uart0_putint( rtc_time.hour );
    uart0_putchar( ':' );
    uart0_putint( rtc_time.min );
    uart0_putchar( ':' );
    uart0_putint( rtc_time.sec );
    uart0_puts( "\n" );
  }
}
```

Lee la hora del RTC

Muestra la hora del RTC por la UART0

Arquitectura fore/background

tarea 4: informe de los ticks transcurridos

- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagWriteTicks
  - La RTI del timer0 (hebra en foreground) activa este flag cada 1000 ticks (10 s)

```
void Task4( void )
{
  static boolean init = TRUE;
  static uint32 ticks;                          Almacena el numero de ticks transcurridos

  if( init )
  {
    init = FALSE;
    uart0_puts( " Task 4: iniciada.\n" );
    ticks = 0;                                  Inicializa el numero de ticks transcurridos
  }
  else
  {
    ticks += TICKS_PER_SEC * 10;                Actualiza el numero de ticks transcurridos
    uart0_puts( "  (Task 4) Ticks: " );
    uart0_putint( ticks );                      Muestra los ticks transcurridos por la UART0
    uart0_puts( "\n" );
  }
}
```

Arquitectura fore/background

tarea 5: informe de la tecla pulsada

- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagTask5
  - La tarea 2 (hebra en background) activa este flag tras leer el keypad

```
void Task5( void )
{
  static boolean init = TRUE;

  if( init )
  {
    init = FALSE;
    uart0_puts( " Task 5: iniciada.\n" );
  }
  else
  {
    uart0_puts( "  (Task 5) Tecla pulsada: " );
    uart0_puthex( scancode );
    uart0_puts( "\n" );
  }
}
```

Muestra el scancode por la UART0

J.M. Mendías
2021

Arquitectura fore/background

tarea 6: visualización en display de la tecla pulsada

- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagTask6
  - La tarea 2 (hebra en background) activa este flag tras leer el keypad

```
void Task6( void )
{
  static boolean init = TRUE;

  if( init )
  {
    init = FALSE;
    uart0_puts( " Task 6: iniciada.\n" );
  }
  else
  {
    segs_putchar( scancode );  ····· Muestra el scancode por el display 7-segmentos
  }
}
```

Arquitectura fore/background

tarea 7: aviso de pulsador presionado

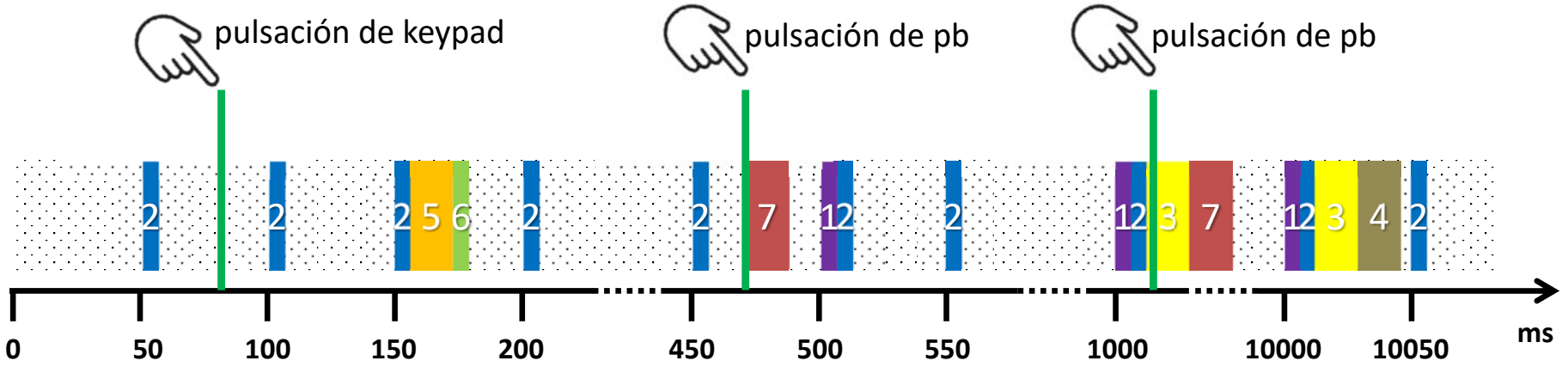
- Esta tarea es ejecutada desde la función main (hebra background) cuando por pooling detecta la activación de flagPb
  - La RTI del pushbuttom (hebra en foreground) activa este flag cuando detecta presión

```
void Task7( void )
{
  static boolean init = TRUE;

  if( init )
  {
    init  = FALSE;
    uart0_puts( " Task 7: iniciada.\n“);
  }
  else
  {
    uart0_puts( "  (Task 7) Se ha pulsado algún pushbutton...\n" );
  }
}
```

Aplicación multitarea

análisis temporal

- El envío de datos por la UART0 es la causa principal de retardo
  - Enviar con espera un carácter por la UART0 tarda 87 µs.
    - (115200 baudios) / 10 b/char = 11520 char/s
- Los tiempos de cómputo del las tareas son:
  - Task1, Task2 y Task6: despreciable
  - Task3: 2.0 ms
  - Task4: 1.9 ms
  - Task5: 2.3 ms
  - Task7: 3.5 ms



J.M. Mendías
2021

Arquitectura cola de funciones

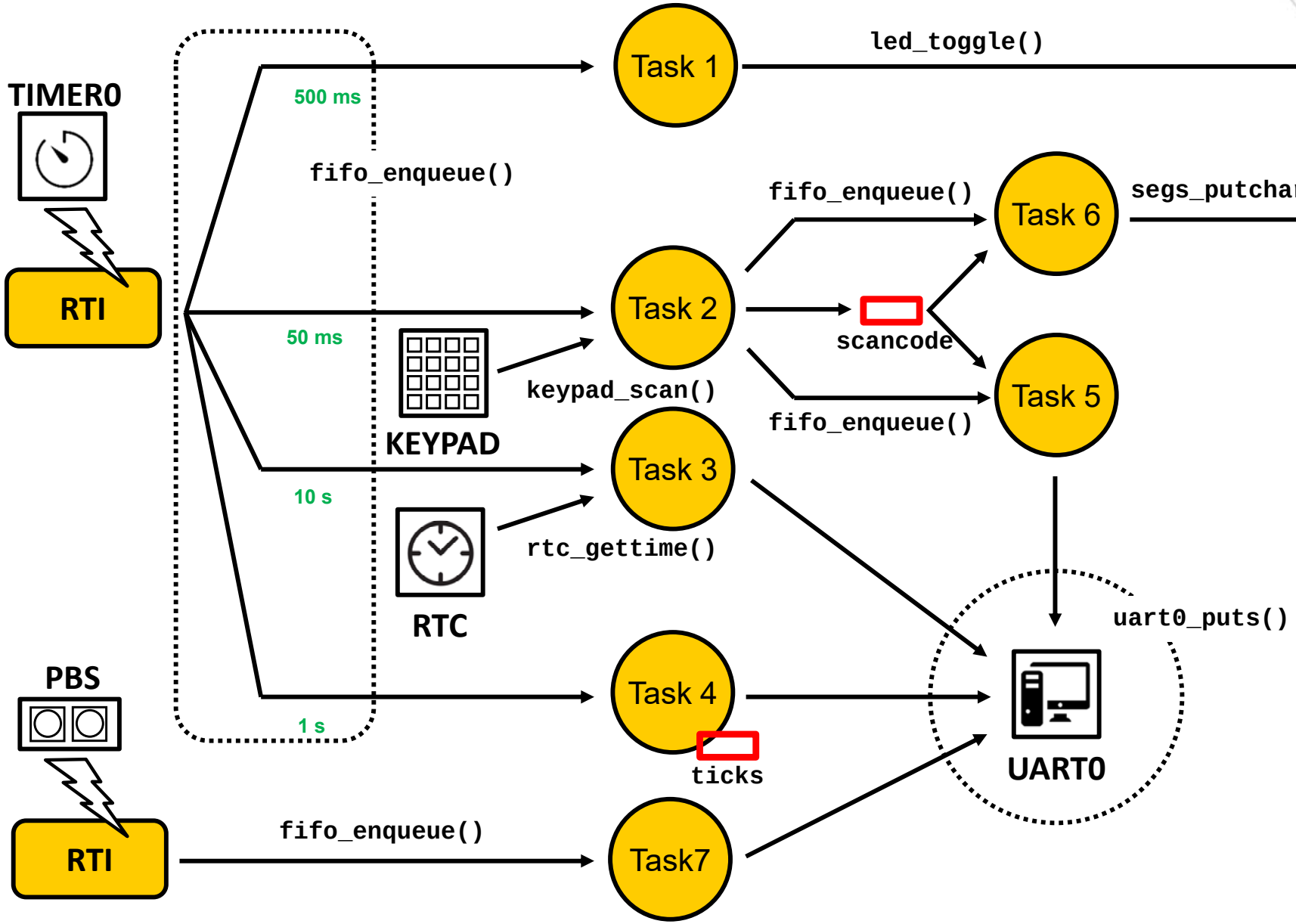
arquitectura software

- Las tareas se ejecutan en una arquitectura cola de funciones.
  - Una hebra en foreground (RTI por temporizador) se encarga de encolar las tareas periódicas tras el transcurso de cada periodo de tiempo (50 ms, 500 ms, 1s y 10 s).
    - Su disparo, además, despierta al procesador.
  - Otra hebra en foreground (RTI por pulsador) se encarga de encolar la tarea 7 cada vez que detecta la presión de cualquier pulsador.
    - Su disparo, también despierta al procesador.
  - Una única hebra en background (función main) desencola las tareas y las ejecuta secuencialmente.
    - Dado que las tareas son ejecutadas sin expropiación, los recursos compartidos (UART0 y mailbox) son accedidos secuencialmente y no es necesario protegerlos.
    - Cuando la cola queda vacía, suspende al procesador.
  - El encolado/desencolado de tareas reemplaza a la señalización por flags.

J.M. Mendias
2021

Arquitectura cola de funciones

arquitectura software



Arquitectura cola de funciones

declaraciones

```
#define TICKS_PER_SEC   (100)
```
Entre tick y tick transcurren 10 ms

```
#define BUFFER_LEN   (512)
```
Tamaño máximo de la cola de tareas

```
typedef void (*pf_t)(void);
```
Tipo tarea (puntero a función)

```
typedef struct fifo
{
  uint16 head;
  uint16 tail;
  uint16 size;
  pf_t buffer[BUFFER_LEN];
} fifo_t;
```
Tipo cola de tareas (implementada como buffer circular)

```
void fifo_init( void );
void fifo_enqueue( pf_t pf );
pf_t fifo_dequeue( void );
boolean fifo_is_empty( void );
boolean fifo_is_full( void );
```
Funciones para la gestión de la cola de tareas

```
...
```

Arquitectura cola de funciones

declaraciones

```
...

volatile fifo_t fifo;          ····· Cola de tareas

uint8 scancode;                ····· Declara los recursos de sincronización y comunicación

void Task1( void );
void Task2( void );            } Declara tareas
...
void Task7( void );

void isr_pb( void ) __attribute__ ((interrupt ("IRQ")));
void isr_tick( void ) __attribute__ ((interrupt ("IRQ")));   } RTI
```

J.M. Mendias
2021

Arquitectura cola de funciones

programa principal (hebra background)

```
void main( void )
{
  pf_t pf;

  sys_init();
  timers_init();
  ...
  keypad_init();

  uart0_puts("\n\n Ejecutando una aplicación como cola de funciones\n");
  uart0_puts("  ----------------------------------------\n\n");

  fifo_init();

  Task1();
  Task2();
  ...
  Task7();

  pbs_open( isr_pb );
  timer0_open_tick( isr_tick, TICKS_PER_SEC );
  ...
```

Inicializa dispositivos

Inicializa la cola de tareas

Ejecuta por primera vez las funciones para inicializarlas

Instala RTI

J.M. Mendías
2021

Arquitectura cola de funciones

programa principal (hebra background)

```
  ...
  while( 1 )
  {
    sleep();                          Entra en estado IDLE, sale por interrupción
    while( !fifo_is_empty() )         Mientras la cola no esté vacía...
    {
      pf = fifo_dequeue();            ... desencola la tarea
     (*pf)();                         ... y la ejecuta (en el orden de encolado)
    }
  }
}
```

Arquitectura cola de funciones

RTI (hebras foreground)

```
void isr_tick( void )
{
  static uint16 cont5ticks  = 5;
  static uint16 cont50ticks = 50;
  ...

  if( !(--cont5ticks) )
  {
    cont10ticks = 5;
    fifo_enqueue( Task2 );
  }
  if(!(--cont50ticks) )
  {
    cont50ticks = 50;
    fifo_enqueue( Task1 );
  }
  if(!(--cont100ticks) )
  ...
  I_ISPC = BIT_TIMER0;
};
```

Contadores modulares de ticks

En lugar de activar un flag, encola la tarea cada 5 ticks (50 ms)

En lugar de activar un flag, encola la tarea cada 50 ticks (500 ms)

```
void isr_pb( void )
{
  fifo_enqueue( Task7 );
  EXTINTPND = BIT_RIGHTPB | BIT_LEFTPB;
  I_ISPC = BIT_PB;
}
```

En lugar de activar un flag,
encola la tarea cuando se presiona cualquier pulsador

J.M. Mendías
2021

Arquitectura cola de tareas

tareas

- Las tareas 1, 3, 4, 5 y 6 son idénticas a la arquitectura fore/background.
- La tarea 2 reemplaza por encolados la señalización con flags.

```
void Task2( void )
{
  static boolean init = TRUE;
  static enum { wait_keydown, scan, wait_keyup } state;

  if( init )
  {
    ...
  } else switch( state ) {
    ....
      case scan:
        scancode = keypad_scan();
        if( scancode != KEYPAD_FAILURE )
          { fifo_enqueue( Task5 ); fifo_enqueue( Task6 ) }
        state = wait_keyup;
        break;
        ...
  }
}
```

En lugar de activar flas, encola tareas si la lectura es válida

- Todas ellas son ejecutadas desde la función main (hebra background) cuando se desencolan.

J.M. Mendías
2021

Arquitectura cyclic executive

arquitectura software

- Las tareas se ejecutan en una arquitectura cyclic executive.
  - Fuera de línea se realiza una planificación de las tareas.
  - Una hebra en foreground (RTI por temporizador) se encarga de señalizar mediante un flag el transcurso de un ciclo de la planificación.
    - Su disparo, además, despierta al procesador.
  - Otra hebra en foreground (RTI por pulsador) se encarga de señalizar mediante un flag la presión de cualquier pulsador.
    - Su disparo, también despierta al procesador.
  - Una única hebra en background (función main) ejecuta secuencialmente las tareas que corresponden en cada ciclo según la planificación estática precalculada.
    - Dado que las tareas son ejecutadas sin expropiación, los recursos compartidos son accedidos secuencialmente y no es necesario protegerlos.
    - Tras ejecutarlas todas, suspende al procesador.

- Esta arquitectura requiere que las tareas sean periódicas:
  - Las tareas 1, 2, 3 y 4 lo son con periodos de activación 500 ms, 50 ms, 1 s y 10 s.
  - Las tareas 5 y 6 no, lo serían haciendo un chequeo periódico (100 ms) del mailbox.
  - La tarea 7 tampoco, pero lo sería haciendo un chequeo periódico (50 ms) del flag.

J.M. Mendías
2021

Arquitectura cyclic executive

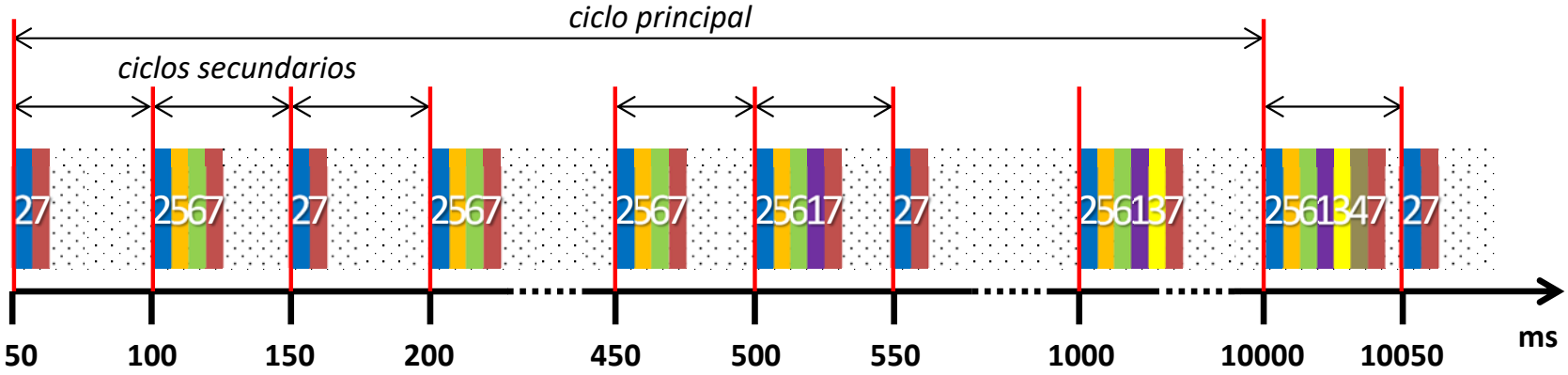
planificación off-line

- Las tareas de este sistema son fáciles de planificar:
  - El ciclo principal es de 10 s y el secundario de 50 ms
  - La planificación consta de 200 ciclos secundarios, con 5 trabajos distintos.

Task1: $t_1 = 500$ ms, $c_1 = 0$ ms
Task2: $t_2 = 50$ ms, $c_2 = 0$ ms

Task3: $t_3 = 1000$ ms, $c_3 = 2.0$ ms
Task4: $t_4 = 10000$ ms, $c_3 = 1.9$ ms

Task5: $t_5 = 100$ ms, $c_5 = 2.3$ ms
Task6: $t_6 = 100$ ms, $c_6 = 0$ ms
Task7: $t_6 = 50$ ms, $c_6 = 3.5$ ms



JobA:
$t_A = 50$ ms, $c_A = 3.5$ ms

JobC:
$t_C = 500$ ms, $c_C = 5.8$ ms

JobE:
$t_E = 10000$ ms, $c_2 = 9.5$ ms

JobB:
$t_B = 100$ ms, $c_B = 5.8$ ms

JobD:
$t_D = 1000$ ms, $c_D = 7.6$ ms

Arquitectura cola de funciones

declaraciones

```
#define MAYOR_PERIOD (10000)
#define MINOR_PERIOD (50)

#define NUM_JOBS     (MAYOR_PERIOD/MINOR_PERIOD)

typedef void (*pf_t)(void);

uint8 scancode;
boolean flagTask5;
boolean flagTask6;

volatile boolean flagPb;
Volatile boolean flagTimer;

void Task1( void );
void Task2( void );
...
void Task7( void );

void isr_pb( void ) __attribute__ ((interrupt ("IRQ")));
void isr_tick( void ) __attribute__ ((interrupt ("IRQ")));
...
```

Entre tick y tick transcurren 50 ms

Tipo tarea/trabajo (puntero a función)

Declara los recursos de sincronización y comunicación

Declara tareas

RTI

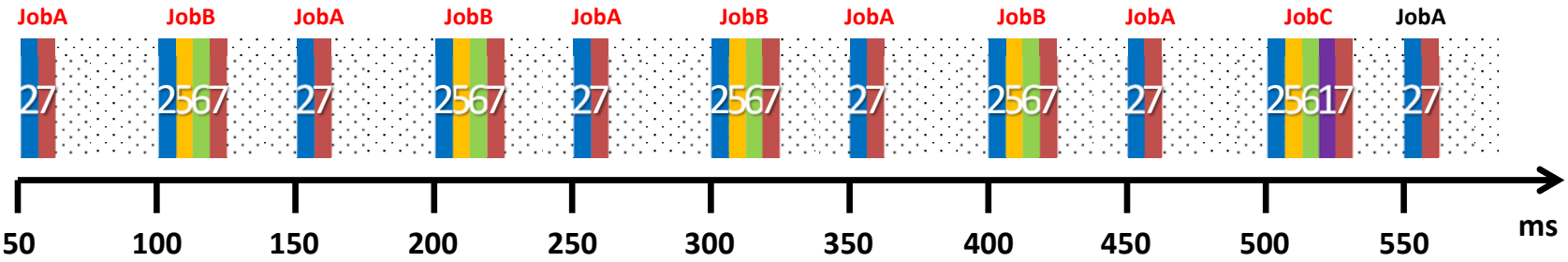
Arquitectura cyclic executive

declaraciones

```
...

void JobA( void );
void JobB( void );
void JobC( void );
void JobD( void );
void JobE( void );

const pf_t pjobs[NUM_JOBS] =
{
  JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobC,
  JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobD,
  JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobC,
  ...
  JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobB, JobA, JobE
};
```

Declara trabajos

Planificación: array estático de trabajos



J.M. Mendias
2021

Arquitectura cyclic executive

programa principal (hebra background)

```
void main( void )
{
  uint8 i;

  sys_init();
  timers_init();
  ...
  keypad_init();

  uart0_puts("\n\n Ejecutando una aplicación cyclic executive\n");
  uart0_puts(" ----------------------------------------\n\n");

  flagTask5 = FALSE;
  flagTask6 = FALSE
  flagPb    = FALSE;
  flagTimer = FALSE;

  Task1();
  Task2();
  ...
  Task7();

  pbs_open( isr_pb );
  timer0_open_ms( isr_tick, MINOR_PERIOD, TIMER_INTERVAL );
  ...
```

Inicializa dispositivos

Inicializa flags

Ejecuta por primera vez las funciones para inicializarlas

Instala RTI

Arquitectura cyclic executive

programa principal (hebra background)

```
  ...
  while( 1 )
  {
    sleep();                    Entra en estado IDLE, sale por interrupción
    if( flagTimer )             Si ha transcurrido un ciclo de la planificación...
    {
      flagTimer = FALSE;
      (*pjobs[i])();            ... ejecuta el trabajo
      i = ( i==NUM_JOBS-1 ? 0 : i+1 );   y recorre cíclicamente el array de trabajos
  }
}
```

J.M. Mendias
2021

Arquitectura cola de funciones

RTI (hebras foreground)

```
void isr_tick( void )
{
  flagTimer = TRUE;  ····· Activa flag cada vez que transcurre un ciclo de la planificación
  I_ISPC    = BIT_TIMER0;
};
```

```
void isr_pb( void )
{
  flagPb    = TRUE;  ····· Activa flag cada vez que se pulsa cualquier pulsador
  EXTINTPND = BIT_RIGHTPB | BIT_LEFTPB;
  I_ISPC    = BIT_PB;
}
```

J.M. Mendias 2021

Arquitectura cyclic executive

trabajos

- Todos los trabajos son ejecutados desde la función main (hebra background) en según la planificación estática definida en el array.
  - Cada trabajo define una lista de tareas que se ejecutan secuencialmente.
  - La RTI del timer0 (hebra en foreground) despierta a la hebra en background cada 50 ms para que ejecute el trabajo que corresponda

```
void JobA( void )
{
    Task2(); Task7();
}

void JobB( void )
{
    Task2(); Task5(); Task6(); Task7();
}

void JobC( void )
{
    Task2(); Task5(); Task6(); Task1(); Task7();
}

...
```

JobA:
$t_A = 50$ ms, $c_A = 3.5$ ms

JobB:
$t_B = 100$ ms, $c_B = 5.8$ ms

JobC:
$t_C = 500$ ms, $c_C = 5.8$ ms

J.M. Mendías
2021

Arquitectura cyclic executive

tareas

- Las tareas 1, 2, 3 y 4 son idénticas a la arquitectura fore/background.
- Las tareas 5, 6 y 7 solo añaden un chequeo por flags

```
void Task6( void )
{
  static boolean init = TRUE;

  if( init )
  {
    init = FALSE;
    uart0_puts( " Task 6: iniciada.\n" );
  }
  else if( flagTask6 )
  {
    flagTask6 = FALSE;
    segs_putchar( scancode );
  }
}
```

- Todas ellas son ejecutadas desde la función main (hebra background) cuando se llaman desde el trabajo correspondiente.

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)

  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/