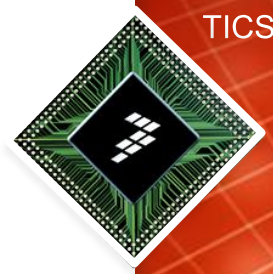
Freescale MQX Tareas y GPIO

Carlos Musich
Soledad Godínez

TICS - Technical Information & Commercial Support



Traducida por:
Erick y Ulises

Jun 2014

Agenda de modulo

- **Tareas**

- **GPIOs**

Tareas

Task Template List

- *Task template List*, es una lista de plantillas de tareas (TASK_TEMPLATE:STRUCTURE), esta estructura define un conjunto inicial de plantillas, de las tareas que se pueden crear en el procesador

```
typedef struct task_template_struct
{
_mqx_uint                TASK_TEMPLATE_INDEX;
void _CODE_PTR_          TASK_ADDRESS)(uint_32);
_mem_size                TASK_STACKSIZE;
_mqx_uint                TASK_PRIORITY;
char _PTR_               TASK_NAME;
_mqx_uint                TASK_ATTRIBUTES;
uint_32                  CREATION_PARAMETER;
_mqx_uint                DEFAULT_TIME_SLICE;
} TASK_TEMPLATE_STRUCT, _PTR_ TASK_TEMPLATE_STRUCT_PTR;
```

Task Template List

```
typedef struct task_template_struct
{
_mqx_uint               TASK_TEMPLATE_INDEX;
void _CODE_PTR_         TASK_ADDRESS)(uint_32);
_mem_size               TASK_STACKSIZE;
_mqx_uint               TASK_PRIORITY;
char _PTR_              TASK_NAME;
_mqx_uint               TASK_ATTRIBUTES;
uint_32                 CREATION_PARAMETER;
_mqx_uint               DEFAULT_TIME_SLICE;
} TASK_TEMPLATE_STRUCT, _PTR_ TASK_TEMPLATE_STRUCT_PTR;
```

TASK_INDEX es usualmente un DEFINE con un número de índice de cada tarea

Se refiere al nombre de la función; la documentación la llama puntero de dirección tarea, pero cuando se utiliza el nombre de la tarea C toma la dirección de la función

El tamaño de stack o pila definido

Indica la prioridad de la tarea. A menor número la prioridad es mayor. El núcleo del SO utiliza prioridades del 0 al 8, por lo que a las tareas se les asigna a partir de la prioridad 9.

El identificador de la tarea.

Atributos de la tarea. Puedes utilizar más de un atributo por cada tarea de la lista. Los atributos permitidos son:

- **Auto start.: Cuando MQX inicia, se crea una instancia de la tarea.**
- **DSP: MQX guarda los registros del Coprocesador DSP como parte del contexto de la tarea.**
- **Floating point: MQX guarda los registros floating point como parte del conexto de la tarea.**
- **Time Slice: MQX utiliza la programación round roubin para la tarea. (El default es FIFO).**

Es el parámetro a pasar a la tarea cuando se crea.

Es el segmento de tiempo (usualmente en milisegundos) utilizado para la tarea cuando se usa round robin.

Creando una tarea

Cuando creas una tarea tienes que:

- Hacer el prototipo y la definición del índice de la tarea.
- Añadir la tarea alTask Template List
- Hacer la definición de la tarea.

Creando una tarea

- Cualquier tarea (creador) puede crear otra tarea (hijo) llamando:
  - **_task_create(),**
  - **_task_create_at()**
  - **or _task_create_blocked()**

Creando una tarea

- **_task_create(),**

Esta función pone el hijo en la cola de tareas listas para la prioridad de la tarea . Si el hijo es de mayor prioridad que el creador, el hijo se vuelve la tarea activa, porque es la tarea lista con mayor prioridad. Si el creador es de mayor o igual prioridad que el hijo, entonces se mantiene activo.

```
void Main_task(uint_32 initial_data)
{
    _task_id task_id;
    printf("\n Hello World \n");
    task_id = _task_create (0,OTRA_TASK, 0);
 }
```

Creando una tarea

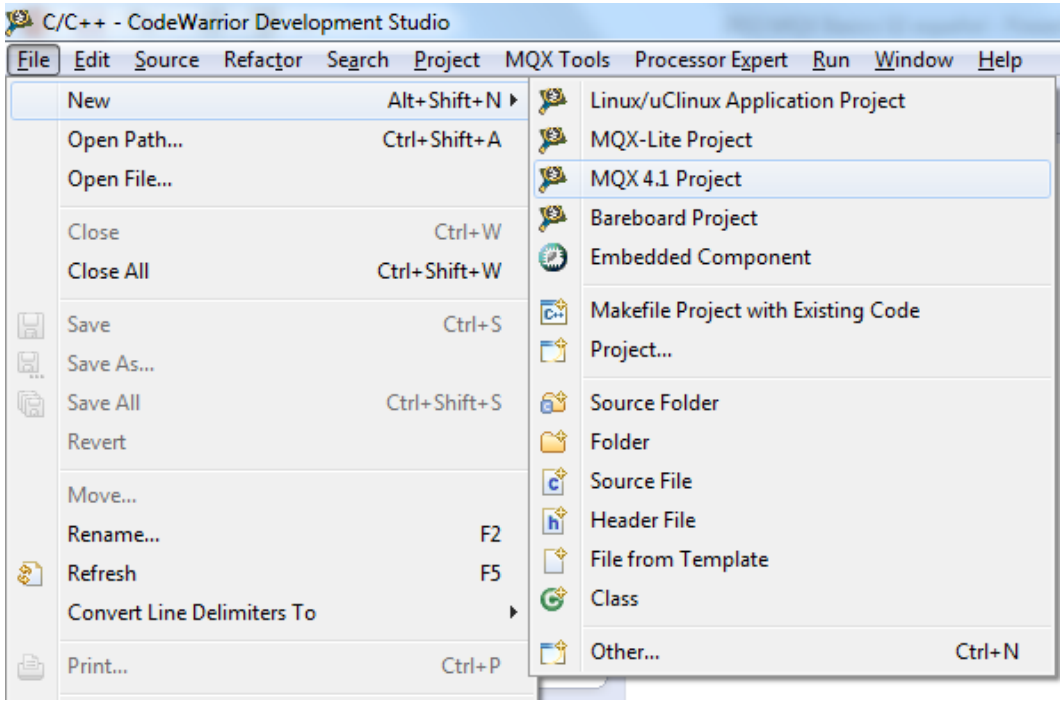
- **_task_create_blocked()**

  Crea una tarea que se encuentra bloqueada. La tarea no se encuentra lista para correr hasta que otra tarea llame

_task_ready().

Ejemplo de Creación de tareas

- En este ejemplo una tarea (creador) se encargara de crear una tarea (hijo).
- Primero se crea un nuevo proyecto para MQX 4.1 en code warrior.



Ejemplo: creación de una tarea

- Una vez creado, añadimos al task template list en el archivo main.c, las tareas del programa; en este ejemplo solo serán 2, una con inicio automático y otra sin iniciar, esto con el fin de que la primera tarea (creador) cree la segunda (hijo).

```
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
/*  Task number, Entry point, Stack, Pri, String, Auto? */
   {CREADOR,   creador,   1500,  9,   "main", MQX_AUTO_START_TASK},
   {HIJO   ,   hijo   ,   1500,  9,   "main", 0                   },
   {0,           0,           0,     0,   0,      0,              }
};
```

Ejemplo: creación de una tarea

- Posteriormente creamos el prototipo de función de las tareas y sus respectivos defines; todo esto en el main.h

```
#define creador 1
#define hijo 2

extern void CREADOR(uint32_t);
extern void HIJO(uint32_t);
```

- Después se crean las tareas en el main.c en donde el creador va a utilizar la función de _task_create(); para crear la segunda tarea (hijo).

Ejemplo: creación de una tarea

- Creador:

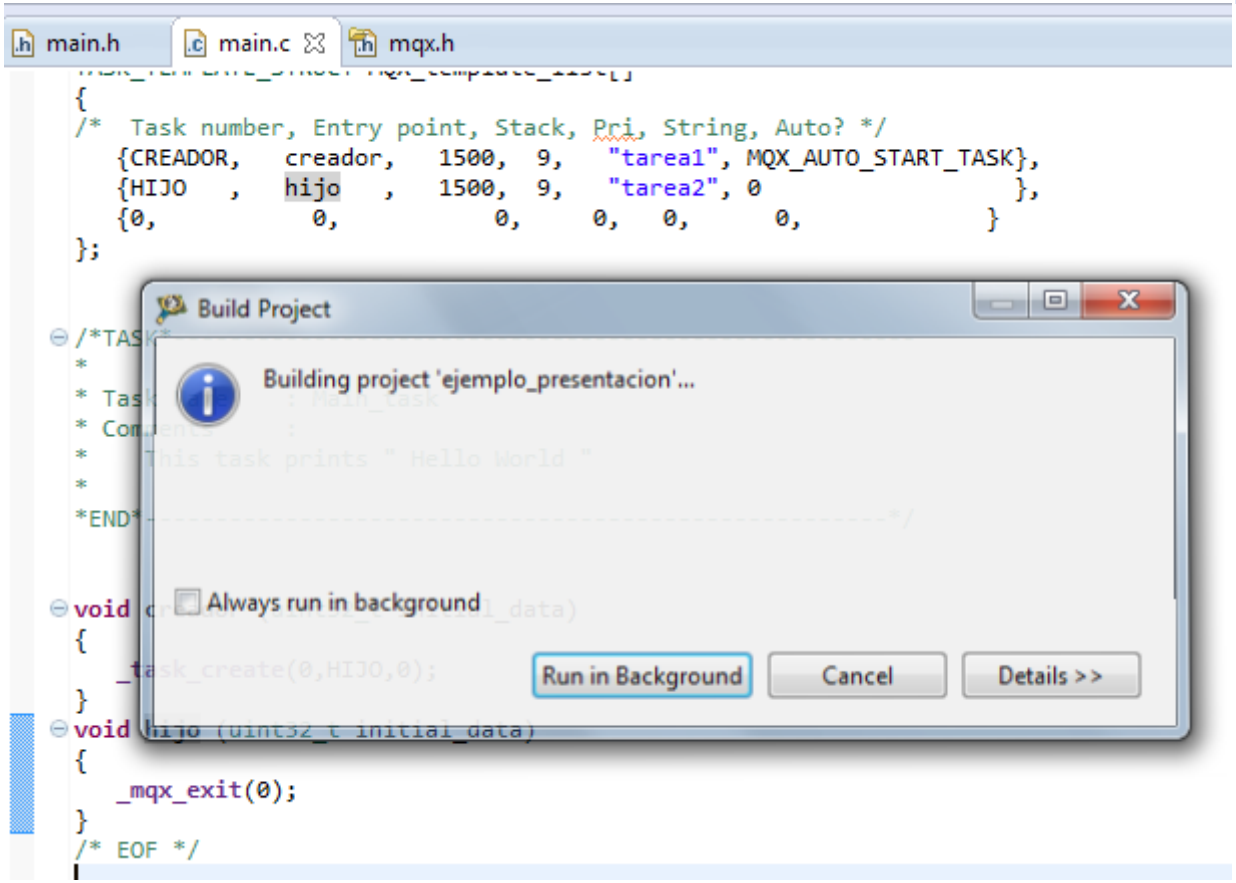
```
void creador(uint32_t initial_data)
{
    _task_create(0,hijo,0);
}
```

Hijo:

```
void hijo(uint32_t initial_data)
{
    _mqx_exit(0);
}
```

Ejemplo: creación de una tarea

- Una vez terminado el código se compila para verificar que no tenga ningún error. Para compilar se da click en 



Ejemplo: creación de una tarea

main.h | main.c | mqx.h

```
{
/*  Task number, Entry point, Stack, Pri, String, Auto? */
   {CREADOR,   creador,   1500,  9,   "tarea1", MQX_AUTO_START_TASK},
   {HIJO   ,   hijo   ,   1500,  9,   "tarea2", 0                   },
   {0,           0,           0,     0,   0,      0,              }
};


/*TASK*-------------------------------------------------------
*
* Task Name    : Main_task
* Comments     :
*    This task prints " Hello World "
*
*END*-------------------------------------------------------*/


void creador (uint32_t initial_data)
{
   _task_create(0,HIJO,0);
}
void hijo (uint32_t initial_data)
{
   _mqx_exit(0);
}
/* EOF */
```

Problems | Console | Memory

0 items

Description

GPIOs

Controlador LWGPIO

- El **Light-Weight GPIO (LWGPIO**) es una interface común para los módulos GPIO.

- El controlador **LWGPIO** es diseñado como un controlador por pin, lo que significa que la llamada del **LWGPIO API** puede manejar solamente un pin.

- Los archivos fuente del controlador LWGPIO se encuentran localizados en el directorio *source\io\lwgpio.*

Controlador LWGPIO

- El capítulo 20 de **Freescale MQX™ I/O Drivers User Guide** explica todas las funciones de referencia para el modulo de LWGPIO

- El siguiente ejemplo utiliza el modulo LWGPIO para alternar un pin de estado.

Controlando un pin

- La función lwgpio_init() tiene que ser llamada antes de llamar cualquier otra función API del controlador LWGPIO. Esta función inicializa la estructura LWGPIO_STRUCT. El puntero a la estructura LWGPIO_STRUCT es pasado como un parámetro manejador. Para identificar el pin, es usado el número LWGPIO_PIN_ID especifico de la plataforma. Es necesario tener una variable para poder asignar el puntero a la LWGPIO_STRUCT; para hacer esto es necesario agregar la siguiente línea.

```
LWGPIO_STRUCT led1;
```

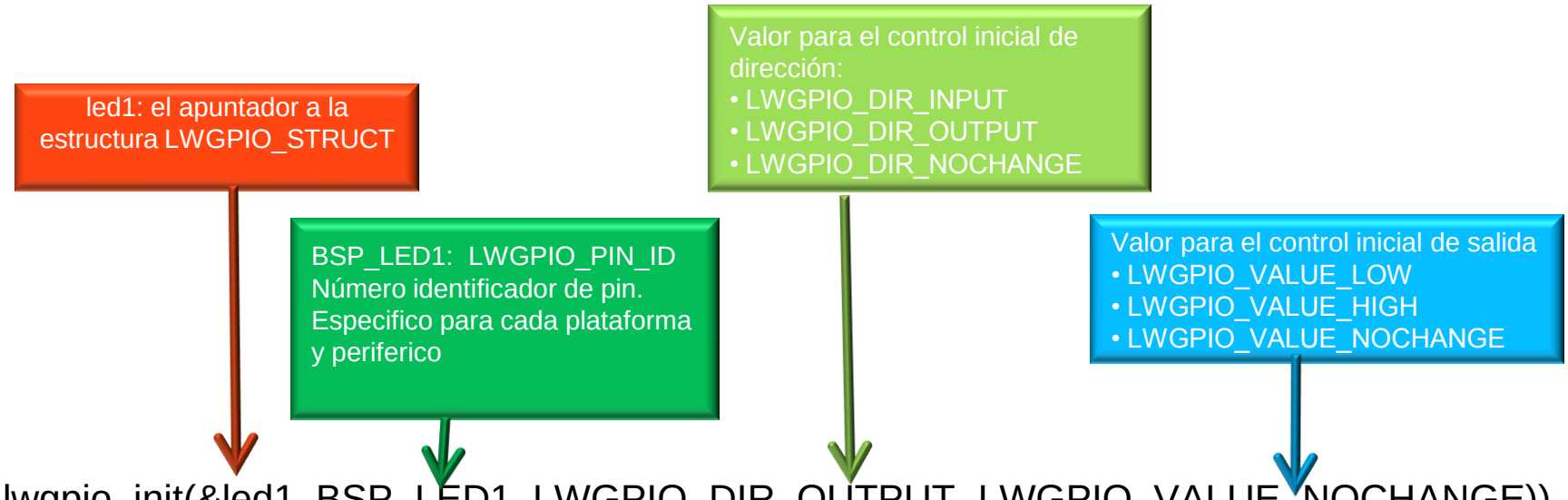
Controlando un pin

- La siguientes líneas inicializan el LWGPIO handle para el pin **LWGPIO_MUX_xxx_GPIO**. El código es el mismo para todos los proyectos; El BSP ya sabe la dirección y el número para cada puerto GPIO y cada pin. Esto esta definido en el archivo *mqx/source/bsp/<bsp_name>/<bsp_name>.h.*

```
if (!lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT,
   LWGPIO_VALUE_NOCHANGE))
  {
    printf("Initializing Port TC pin0 GPIO of the MCF52259 as output failed.\n");
    _task_block();
  }
```

Controlando un pin

- Los parámetros para lwgpio_init() son:



Controlando un pin

- Una vez que es inicializado, es necesario asignar la funcionabilidad deseada al pin, como el modo GPIO o cualquier otro modo de periférico. La función lwgpio_set_functionality() define la funcionabilidad del pin. El valor del parámetro de funcionabilidad representa el número almacenado en el registro multiplexor, el cual, selecciona la funcionabilidad deseada. Para el modo GPIO puede utilizar las macros predefinidas, que se pueden encontrar en el archivo ***lwgpio_<mcu>.h***. Añada las siguientes líneas cambian al modo GPIO de funcionabilidad.

```
lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);
```

Controlando un pin

- La función lwgpio_set_value() define el estado del pin especificado (alto o bajo). Para poder alternar el estado del pin indicado, escriba las siguientes líneas:

```
while (TRUE)
  {
    _time_delay(3000);
    lwgpio_set_value(&led1, value); /* toggle pin value */
    value = value^1;
  }
```

Ejemplo de prender y apagar 1 led

Para este ejemplo utilizaremos el que se realizó anteriormente; la única diferencia será que la tarea 2 (hijo) va a ser la encargada de prender y apagar el LED.

Primero creamos la estructura para almacenar los parámetros con la siguiente línea:

```
LWGPIO_STRUCT led1;
```

Después inicializamos el pin:

```
lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_HIGH);
```

Por último definimos su modo de funcionalidad

```
lwgpio_set_functionability(&led1, BSP_LED1_MUX_GPIO);
```

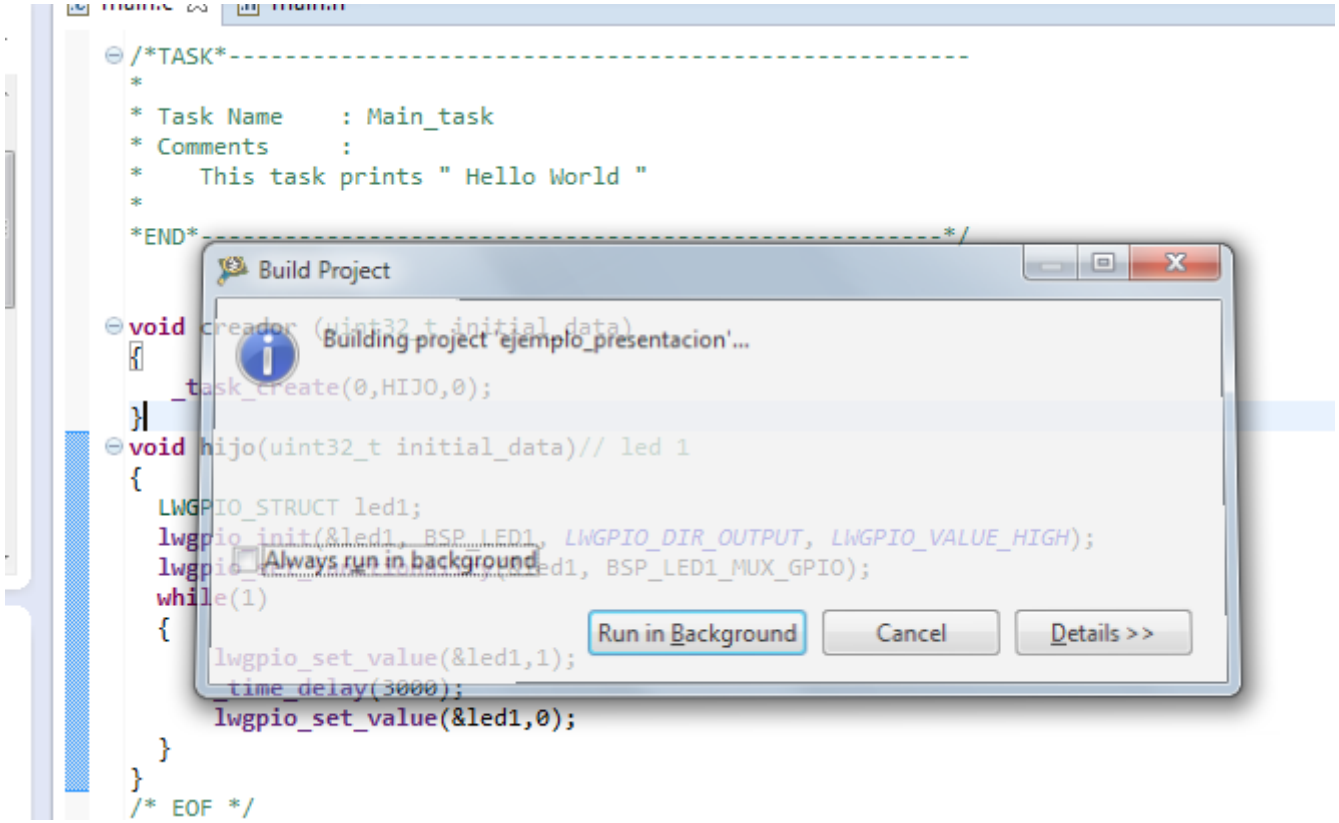
Ejemplo de prender y apagar 1 led

- Por último, también dentro de la tarea 2, realizamos el cambio de estado (toggle) del pin para que prenda y apague:

```
void hijo(uint32_t initial_data)// led 1
{
  LWGPIO_STRUCT led1;
  lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_HIGH);
  lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);
  while(1)
  {
      lwgpio_set_value(&led1,1);
      _time_delay(3000);
      lwgpio_set_value(&led1,0);
  }
}
```

Ejemplo de prender y apagar 1 led

- Compilamos y verificamos que no haya errores.



Ejemplo de prender y apagar 1 led

main.c | main.h

```
/*TASK*-----------------------------------------------------
*
* Task Name    : Main_task
* Comments     :
*    This task prints " Hello World "
*
*END*-----------------------------------------------------*/


void creador (uint32_t initial_data)
{
   _task_create(0,HIJO,0);
}
void hijo(uint32_t initial_data)// led 1
{
  LWGPIO_STRUCT led1;
  lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_HIGH);
  lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);
  while(1)
  {
      lwgpio_set_value(&led1,1);
      _time_delay(3000);
      lwgpio_set_value(&led1,0);
  }
}
/* EOF */
```

Problems | Console | Memory

0 items

Description


freescale™