LIBRERÍAS PARA LOS PERIFÉRICOS

CLASE 14
MAR 21 2024

Un Ejemplo → la librería para los GPIO de la RPP2040 → la está haciendo el PROFE!

→ Muestra todos ~~la librería de~~ los registros del GPIO

→ Por c/GPIO se tiene un registro de STATUS y uno de CONTROL

→ Se tienen 30 pines del 0 al 29 GPIO4 / STATUS / CTRL

→ Luego se tienen 4 Registros para interrupciones ya que se pueden configurar 4 eventos: NIVEL ALTO, NIVEL BAJO, FLANCO SUBIDA, FLANCO BAJADA

→ ~~Son~~ 30 pines → 30×4 = 120 EVENTOS y en 4 registros de 32 bits → hay 128 bits

Y es la razón por la que hay 4 Regs para INTERRUPCIONES

En R4 → para los 1ros 8 GPIO (0 al 7)
R1 → " los del 8 al 15
R2 → " los del 16 al 23
R4 → " " del 24 al 30
} va a corresponder un bit por c/evento

- Luego tenemos otros 4 registros para activar los mismos eventos! En el SDK uno no manipula esos registros directo! sino que hay una función que lo hace por mí! Yo paso la máscara de cuál es el evento que yo quiero activar y él lo habilita!
Estos serían entonces "los habilitadores de interrupción."
- Luego tenemos los de "FORZAR INTERRUPCIÓN" → Cuál es el objetivo? Un setup para esto podría ser: En un "ambiente de desarrollo tener el MCU conectado al PC y que éste emule un display o unos pulsadores en una I/F gráfica y que a través de un software debug el pueda generar una interrupción cuando yo presione un "botón en la pantalla"
- En el diag. de Bloques del GPIO hay varios puntos donde uno puede ir a agarrar la señal, este es uno de esos "después de enmascarar y después de forzar" (y esto aparece para el proc 0 y el proc 1)
"INTERRUP-STAT_AFTER MASKING AND FORCING"
- DORMANT WAKE INTERRUPT ENABLE → Para activar un GPIO para que saque al MCU del modo DORMANT.
Y allí aparecen todos los registros que se tienen → Muchos!
Pero GPIO0_STATUS → va a ser igual al GPIO1_STATUS

Y yo no lo tengo que declarar las 60? veces que aparece!
sino que lo voy a declarar una sola vez (real! 30 → una por c/GPIO!)
Qué estructura tiene ese registro? el Reg STATUS → da info del estado!

Ej: Reg STATUS → ese bit dice cuál es el estado de la interrupción del procesador después de aplicar el override y dice que es un bit de solo lectura
bit 26

Reg CTRL
ó Cuál es la señal que entra al periférico después de que se aplica override!
— Cuál es la señal en el PAD, antes que se aplique override
— Ahí está y se puede chequear con o sin override!

Hay unos bits asociados con las entradas y otros con las salidas

—//—

Uno lo que tiene que hacer es mapear cada registro a la librería

TAREA
Se selecciona un módulo de un MCU y se va a
c/u de los registros y se va a mapear a la librería.

Reg. STATUS (para los GPIOs)

```
typedef union {
    uint32_t WORD;          // Whole Reg Access
    struct BITS{
        uint32_t  : 8;      // no se para un nombre
        uint32_t OUTFROMPERI : 1;   // el bit 8
        uint32_t OUTTOPAD    : 1;
        uint32_t  : 2;      // Not implemented bits
        ...
        uint32_t  : 5;
    } BITS;
} GPIO_STATUS -> ???
```

"En blanco"

Porque unión?
- Para tener 2 tipos de acceso a c/registro:
- uno por medio de word a todos los 32 bits del registro y otro
- por medio de campos de bits
- Aquí declaramos una estructura de bits equivalente al registro status de un GPIO de la RPP
→ Porque es reservado!
lo primero declarado ocupa los lsb y se va llenando hasta ocupar los 32 y nos encontramos que los bits del 5 al 7 no están implementados!
Y por tanto no los deberíamos usar! por eso en la librería ni siquiera le ponemos un nombre!

En algún momento yo voy a operar con esto, y le voy a decir que le sumo/resto, qué si es "=" a tanto?"
El cuando saque esto lo va a cargar en un reg de 32 bits

- Así voy a describir c/u de los bits en el registro de estado de los GPIOs en la RPP2040

- Se habló de MÁSCARAS, MACROS, CONSTANTES, en la clase pasada
  - ¿Cómo es son estas? → es poner un "1" en ese bit o poner todos "1" en ese campo de bits
  - Y después me permiten hacer la manipulación de poner un bit en "1" o en "0" en...
- Aunque se supone que con la estructura de bits yo no necesito usar máscaras (porque puedo acceder directamente con esas declaraciones)
- Pero la librería la estoy haciendo de forma genérica para yo trabajar "como quiera trabajar"
- Recordar → hay varias formas de trabajar los registros y sus bits.
  - Con valores constantes que yo calculo en mi mente (no es lo normal)
  - Con máscaras o con... } estas librerías (permiten trabajar en estas dos formas.) → tal cual están diseñadas...
  - Estructuras de bits

→ Cuando se trabaja con máscaras, se usa la parte "word" (o el miembro) de la estructura union
- Allí es donde se va a aplicar las máscaras y no a los bits/campos de bits individuales.

Se declara una máscara por c/bit

```
/* MASKS definition for GPIOs status register */
```

→ Es un "1" corrido el # de bits donde arranca ese bit!

```
#define mGPIO_status_OUTFROMPERI 0x00000100UL
```

Si son 5 bits en un campo de bits! → se pone 0x0000001F y corrido el # de bits donde arranca ese campo de bits

|| En este caso pues solamente son de 1 un bit y no ocupan más ||

GPIO_STATUS 20 ——"——

Como los bits todos son "Read Only" entonces no se les declaran constantes.
- No tiene sentido porque c/bit o está en 1 o está en 0 → y la de aquí no tiene sentido!
- El significado sería → está en 0 o está en 1 → entonces no tiene sentido

——"——

Registro de control de los GPIO
- Aquí sí son campos de bits.
- Y se pueden declarar constantes → que serían los 3 valores que puede tomar, ej:

```
#define kGPIO0_FUNSEL_SPI0RX
```

```
typedef union {
  uint32_t WORD;                    // Whole Register ACCESS
  struct BITS {
    uint32_t FUNCSEL   : 5;         // Function select (RW) -> RESET 0x1F
    uint32_t           : 3;
    uint32_t OUTOVER   : 2;
    uint32_t           : 2;
    uint32_t OEOVER    : 2;
    uint32_t           : 2;
    uint32_t INOVER    : 2;
    uint32_t           : 10;
    uint32_t IRQOVER   : 2;
    uint32_t           : 2;
  } BITS;
} GPIO_CTRL;
```

MÁSCARA 0x1F

SU MÁSCARA 0x11UL y desplazando 8 bits.

va a completarse con "ceros"

Por defecto los cte's son de 32 bits. Si quiero que sean de 64 bits => 0x11ULL

MASCARA 0x11UL → corrido 12 bits!

MASCARA 0x11UL → corrido 28 bits!