INTRO

MÓDULO DE CUADRATURA

El tivaC launchpad con el que contamos tiene implementados 2 módulos de quadratura (qei0 y qei1), permitiendo estos poder ser usados con motores para calcular la velocidad y posición actual de un motor que posea un encoder. (Claro está que debe ser uno diferencial, de los que devuelven un tren de pulsos desfasados por 2 canales A y B).

CONFIGURACIÓN DEL MÓDULO DE CUADRATURA

Primero, debemos decidir que módulo de cuadratura usaremos y/o que pines usaremos (el primer caso si disponemos de todos los pines, y lo segundo si es que hay pines que ya estamos usando y que puedan ser compartidos por el módulo de cuadratura que elijamos).

Para esto, checando el datasheet podemos ver una tabla en la que se explica que pines se usan para cada módulo (Tabla 21-1, página 1307):

Table 21-1. QEI Signals (64LQFP)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
IDX0	5 64	PF4 (6) PD3 (6)	I	TTL	QEI module 0 index.
IDX1	16	PC4 (6)	I	TTL	QEI module 1 index.
PhA0	28 53	PF0 (6) PD6 (6)	I	TTL	QEI module 0 phase A.
PhA1	15	PC5 (6)	I	TTL	QEI module 1 phase A.
PhB0	10 29	PD7 (6) PF1 (6)	I	TTL	QEI module 0 phase B.
PhB1	14	PC6 (6)	I	TTL	QEI module 1 phase B.

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

En nuestro caso usaremos los pines PC5 y PC6 con el QEI1, ya que no los estamos usando como pines de pwm. En el caso de estar usando estos, se puede observar que tenemos el QEI0, el cuál usa los pines PF0-1 o PD6-7, según sea el caso. Tenemos que tener cuidado si usamos este módulo, ya que los pines PF0 y PD7 están bloqueados por defecto, y se tienen que desbloquear escribiendo sobre registros adecuados, como veremos más adelante en otro ejemplo (checar la página 650 del datasheet, donde explican los pines con funciones por defecto en el tivaC).

Se puede observar también que no estamos usando el pin IDX1, ya que este es el index del módulo de cuadratura (un pin del encoder que bota un pulso cada vez que se completa una vuelta) ya que, para nuestro ejemplo, nuestro motor sólo tiene 2 pines en su encoder (canal A y canal B). Si tuvieras el Index, pues CHEVERE!!!, ayuda bastante para control de posición :D .

CONFIGURANDO EL MÓDULO

Pasemos a configurar el módulo QEI usando la librería, lo cuál haremos en los siguientes pasos:

1. Incluímos la librería a usar, siendo esta la del QEI:

```
#include "tivaCppLibrary/devLibraries/QEI.hpp"
```

Estas librerías se encuentran en la carpeta de librerías en desarrollo, ya que todavía no tienen el formato de las librerías normales. En si es algo más de detalles, pero ya las cambiaré a la carpeta de librarías principales.

2. Configuramos los pines del módulo a usar (en nuestro caso son PC5 y PC6):

```
Gpio::enableClock(Gpio::peripheral::GPIOC);

PC5::enableAsDigital();
PC5::setMode(Gpio::options::mode::alternate);
PC5::setAlternateMode(Gpio::options::altModes::alt6);

PC6::enableAsDigital();
PC6::setMode(Gpio::options::mode::alternate);
PC6::setAlternateMode(Gpio::options::altModes::alt6);
```

Como siempre, primero habilitamos el clock para el puerto C (sino, no podríamos configurar sus registros). Luego, procedemos a configurar los pines del QEI1 con su modo alterno 6 (el la tabla del GPIO y modos alternos, este es el modo alterno para modo QEI).

3. Configuramos el módulo en si:

```
qei1::enableClock();
qei1::config(qei::options::stallOnDebugger::stall,
             qei::options::velocityPredivider::div_1,
             qei::options::capVelocity::captureVel,
             qei::options::captureMode::mode1);
qei1::setTimer(8000000);// set the timer to 100 ms
qei1::enableQEI();
```

Primero, habilitamos el clock del módulo QEI que estamos usando (QEI1 en nuestro caso).
Seguidamente configuramos el QEI con la función config de la librería, pasando los siguientes parámetros:

- `stallOnDebugger::stall` : Para el conteo y el módulo cuando se pausa la ejecución en depuración (pausa por el usuario, o se llega a un breakpoint).
- `velocityPredivider::div_1` : Este es un parámetro que sirve como preescaler para la cantidad de pulsos que se toman en el registro de velocidades, siendo en nuestro caso "`div_1`" o no usar preescaler (el valor del registro de velocidad es exactamente la diferencia entre los valores de los registros de conteo entre dos eventos de timer del módulo de cuadratura).
- `qei::options::capVelocity::captureVel` : Para habilitar la captura de velocidad.
- `qei::options::captureMode::mode1` : Para usar el modo 1 del QEI (sólo contar los pulsos del canal A)

Luego, configuramos el timer del QEI para la captura de velocidad a 100 milisegundos, tomando así las diferencias entre los registros de conteo automáticamente cada 100 milisegundos y guardando este valor en el registro de velocidades.

Finalmente, habilitamos el QEI para que empiece el conteo de los pulsos y la toma de velocidad.

4. Finalmente, creamos 2 variables globales para leer nuestros valores del registro de conteo y del de velocidad, y los usamos en el loop principal.

```
// Variables globales ( antes del main )
u32 countReg = 0;
u32 speedReg = 0;

// Toma de los registros ( dentro del loop principal )
countReg = qei1::getCount();
speedReg = qei1::getSpeedReg();
```

Probaremos nuestra configuración con el siguiente programa ejemplo:

```
#include "tivaCppLibrary/include/Gpio.hpp"
#include "tivaCppLibrary/include/clock.hpp"
#include "tivaCppLibrary/delays.hpp"

#include "tivaCppLibrary/include/Gptimer.hpp"
#include "tivaCppLibrary/devLibraries/QEI.hpp"

// Variables globales ( antes del main )
u32 countReg = 0;
u32 speedReg = 0;

float duty = 0.2;

int main()
{
	clock::clock::config(clock::configOptions::clockSource::mainOscilator,

clock::configOptions::clockRate::clock_80Mhz);

	Gpio::enableClock(Gpio::peripheral::GPIOF);
	Gpio::enableClock(Gpio::peripheral::GPIOC);

	PF2::enableAsDigital();
	PF2::setMode(Gpio::options::mode::gpio);
	PF2::setIOmode(Gpio::options::IOmode::output);

	PC4::enableAsDigital();
	PC4::setMode(Gpio::options::mode::alternate);
	PC4::setAlternateMode(Gpio::options::altModes::alt7);

	PC5::enableAsDigital();
	PC5::setMode(Gpio::options::mode::alternate);
	PC5::setAlternateMode(Gpio::options::altModes::alt6);

	PC6::enableAsDigital();
	PC6::setMode(Gpio::options::mode::alternate);
	PC6::setAlternateMode(Gpio::options::altModes::alt6);
```

```
	qei1::enableClock();
	qei1::config(qei::options::stallOnDebugger::stall,
			qei::options::velocityPredivider::div_1,
			qei::options::capVelocity::captureVel,
			qei::options::captureMode::mode1);
	qei1::setTimer(8000000);// set the timer to 100 ms
	qei1::enableQEI();

	pwmW_0A::enableClock();
	pwmW_0A::config(50,0.2);

	while(1)
	{
		countReg = qei1::getCount();
		speedReg = qei1::getSpeedReg();

		pwmW_0A::setDuty(duty);

		PF2::toogle();
		delays::delay_ms(250);
	}

}
```

¿Qué hace este programa?, pués sólo setea la pwm necesaria al pin PC4 (salida CCP0 del WideTimer0) para poder controlar en lazo abierto la velocidad del motor usando una variable que podemos manejar en depuración llamada “duty”. Esta pwm está configurada a 50us de periodo (20 Khz de frecuencia de pwm) y con un duty inicial de 0.2 o 20%.