Práctica 1: Desarrollo de un sistema de sensado y actuación controlado por USB. Tercera parte (1,5 puntos).

1. Objetivos

En este documento se recogen las especificaciones correspondientes a la tercera fase del proyecto práctico. Este es el último documento de especificaciones a implementar durante el desarrollo del proyecto. El sistema de sensado y actuación desarrollado a lo largo de las especificaciones anteriores se ampliará para incorporar al sistema dos dispositivos que se conectarán a través de un interfaz I^2C al microcontrolador. Este interfaz es un bus serie síncrono que permite la interconexión de diversos dispositivos esclavos al microcontrolador utilizando una señal de datos y otra de control. Este tipo de interfaces es muy utilizado en el diseño de sistemas empotrados basados en microcontrolador, ya que permite la interconexión del microcontrolador con diversos tipos de dispositivos (tanto de control como sensores) utilizando únicamente dos pines.

2. Aplicación de partida

El código de partida será el desarrollado por el estudiante en la fase anterior. Además, para facilitar el desarrollo de esta tercera parte se ha dejado un código de ejemplo adicional que contiene una implementación parcial de las funcionalidades que va a ser necesario desarrollar al implementar estas especificaciones. El estudiante deberá en primer lugar **integrar** la nueva funcionalidad en su código, y posteriormente modificar los nuevos componentes del proyecto suministrados para cumplir con las especificaciones que se piden en este documento. Estos componentes se describen con más detalle en próximas secciones.

3. Prerrequisitos

Para poder implementar las modificaciones indicadas en este documento es necesario que el estudiante tenga conocimiento acerca del funcionamiento del bus I2C y del uso que del mismo realizan los dispositivos comerciales. Por ello **en el campus virtual se ha dejado un documento introductorio** al funcionamiento de este bus, en el que se detallan las posibles transacciones que pueden realizarse sobre el mismo, y se describe la implementación que más habitualmente se sigue en dispositivos comerciales (y que también siguen los dispositivos que se van a utilizar en esta actividad, el ACME 1623 y el Bosch BM160**). Este documento debe leerse** y analizarse antes de continuar con la realización de estas especificaciones.

4 Requisitos.

Los clientes que nos han encargado el desarrollo del sistema de sensado y actuación controlado por USB y basado en el TMC123G nos han pedido extender la funcionalidad del sistema mediante la incorporación al mismo de dos dispositivos con interfaz I2C: un dispositivo extensor de puertos, que permitiría ampliar en

un futuro el número de señales GPIO que el sistema puede controlar, y un sensor inercial de 6 ejes (acelerómetro y giróscopo).

En un principio, los dispositivos que se han seleccionado para formar parte del diseño son el ACME 1623 (extensor de puertos y conversor A/D de 14 bits y 4 canales) y el Bosch BM160 (sensor inercial de "6 ejes" con acelerómetro y giróscopo). Ambos se conectarán al microcontrolador, que actuará como maestro, a través de uno de los interfaces I2C del TMC123G (El I2C1, accesible a través de los pines PA6 y PA7 si se configuran adecuadamente). El diagrama de conexiones puede verse en la figura

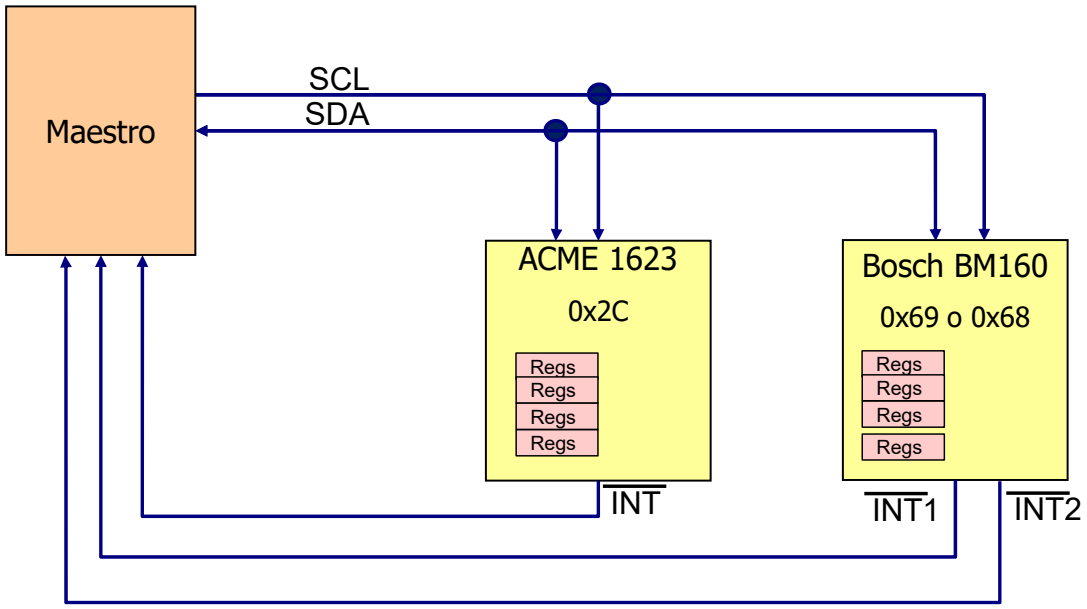

Figura 1. Diragrama de conexiones.

Además de las líneas de datos y reloj del bus I2C, ambos dispositivos disponen de líneas de "salida de interrupción" que le permiten solicitar servicio al microcontrolador de forma asíncrona, cuando se den determinadas condiciones. Para poder detectar estas interrupciones en el microcontrolador, estas líneas pueden conectarse a entradas GPIO del mismo.

Además de las líneas de señal, para que el sistema funcione correctamente será necesario alimentar los dispositivos adecuadamente conectando las señales de masa (GND) y alimentación.

4.1 IMPORTANTE: Disponibilidad de los sensores.

Debido a la crisis de componentes provocada por la pandemia de COVID19, nuestro equipo de diseño no cuenta actualmente con suministros del dispositivo ACME 1623. Afortunadamente, el fabricante del dispositivo suministra un emulador del mismo para el TM2C123, por lo que **en el desarrollo del proyecto vamos a utilizar un dispositivo ACME 1623 emulado en lugar de un dispositivo real.** Para activar y poder utilizar el emulador hay que añadir algunos componentes al proyecto, tal y como se comentará en la siguiente sección.

En el caso del acelerómetro Bosch BM160, sí que vamos a utilizar el dispositivo real, que **se os suministrará en clase** como material de préstamo.

4.2 Código de partida.

Nuestro equipo de diseño ha desarrollado de forma parcial nuevos componentes de código para implementar la funcionalidad adicional que se quiere incorporar al sistema. En esta sección se describen dichos componentes y su utilización en el nuevo proyecto de ejemplo que se utiliza. Estos componentes no están del todo completos, y deberán ser finalizados por vosotros para poder implementar las funcionalidades pedidas.

Antes de comenzar a desarrollar las nuevas funcionalidades es necesario que incorporéis e integréis de forma adecuada en vuestro proyecto estos nuevos componentes.

4.2.1 Componentes adicionales.

El nuevo código de ejemplo suministrado incluye tres nuevas bibliotecas: una biblioteca i2c_driver.c, que implementa un driver de alto nivel I2C capaz de realizar diferentes tipos de transacciones por el bus I2C (realizando internamente llamadas a la biblioteca de bajo nivel i2c.c que forma parte de la biblioteca TIVAWARE), y las bibliotecas BMI160.c y ACME_1623.c, que implementan una API para la configuración, lectura y control de cada dispositivo ("drivers" de los dispositivos).

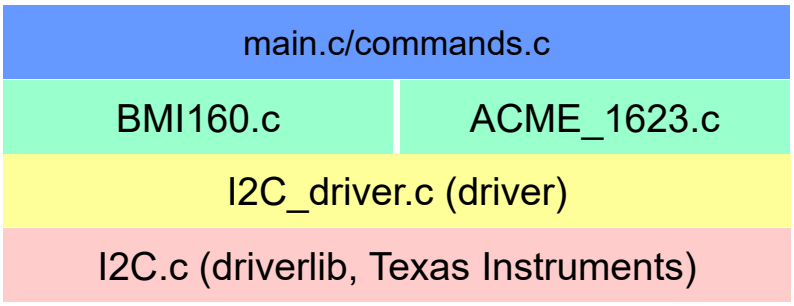

Figura 2. Nuevos componentes del proyecto

La biblioteca i2c_driver.c se ha implementado de forma que esté integrada con FreeRTOS y pueda utilizarse de forma eficiente desde las tareas. Sólo pueden utilizarse desde el nivel de tarea porque son funciones bloqueantes. Las funciones de las bibliotecas BMI160 y ACME_1623 utilizan a su vez a la de la biblioteca i2c_drivers.c para acceder a los diferentes registros de configuración y control de los dispositivos correspondientes, y por tanto tienen la misma limitación.

4.2.2 Inicialización del sensor I²C.

La inicialización de los dispositivos I²C sólo es necesaria hacerla una única vez al arrancar el código, y por los motivos anteriormente mencionados, debe hacerse desde una tarea (no debe hacerse desde una rutina de interrupción, pero tampoco puede hacerse en la función main()). En el ejemplo de partida el código de la inicialización se ha introducido en la tarea UARTTask del fichero commands.c, entre las líneas 554 a 566. Esta inicialización no tendría por qué hacerse en dicha tarea y podría ser movida a otro lugar si se considera más adecuado**. El código de inicialización del sensor BMI160 está desactivado** a la espera de que completéis las primeras especificaciones que se piden en este documento y tengáis el sensor disponible para conectarlo a la placa. Aún así debes incorporarlo a tu proyecto.

4.2.3 Tabla de vectores.

Por motivos de eficiencia, las operaciones I2C son gestionadas por el fichero i2c_driver.c mediante interrupciones. Al integrar el código en tu proyecto es necesario tener en cuenta que la rutina de

interrupción implementada en dicho fichero (*I2CDriver_ISR*) debe ser agregada a la tabla de vectores, en la posición correspondiente al periférico I2C1.

4.2.4 Ejemplo de operación del sensor I²C.

Para mostrar un ejemplo de cómo funcionan los dispositivos I2C y facilitar el desarrollo de las especificaciones se han implementado en el ejemplo suministrado dos comandos adicionales en el intérprete de comandos accesible por la consola serie. Estos comandos están implementados en el propio fichero commands.c (líneas 336 a 508). Si en el intérprete de comandos tecleamos "readacc", el intérprete de comandos ejecutará la función Cmd_readAcc e intentará leer los datos de aceleración del sensor BM160 (¡ojo!, **no va a funcionar hasta que se hayan completado** parte de las funcionalidades que se piden en la sección 4.3). Por otra parte, el comando "acme", proporciona acceso a las diversas funcionalidades del dispositivo ACME1623, de forma que sea posible la realización de pruebas de validación de los componentes desarrollados para el proyecto, ofreciendo varios "subcomandos" que se relacionan a continuación:

- El comando "acme dir <valor>" permite escribir un valor de 8 bits en el registro DIRECTION del ACME1623
- El comando "acme input" permite leer el valor del registro INPUT del ACME1623.
- El comando "acme output <valor>" permite escribir un valor de 8 bits en el registro OUTPUT del ACME1623
- El comando "acme istat" permite leer el valor del registro INT_STATUS del ACME1623.
- El comando "acme clear <valor>" permite escribir un valor de 8 bits en el registro INT_CLEAR del ACME1623
- El comando "acme itype <word>" permite escribir sendos valores en los registros INT_TYPE y ADC _INT_TRIGGER del ACME1623 (el byte alto de <word> se escribe en ADC _INT_TRIGGER y el bajo en INT_TYPE).
- El comando "acme disconnect" permite simular la "desconexión" del dispositivo ACME emulado al bus I2C (para poder comprobar qué ocurre si el dispositivo no está conectado).
- El comando "acme connect" permite simular la "reconexión" del dispositivo ACME emulado al bus I2C.
- El comando "acme adc" lee los valores de los registros ADC_DATA[0:3] del ACME 1623 donde se almacenan los valores de conversión de las muestras y muestra los valores por la pantalla.
- El comando "acme read" lee los 16 registros internos del dispositivo ACME. Los registros se leen en orden consecutivo, pero el primero que se lee depende de las operaciones que se hayan realizado previamente. El principal propósito de este comando es poder probar la implementación de parte de las funcionalidades que se piden al principio de la sección 4.3.

La "guía rápida de programación" del dispositivo ACME 1623 en la que se describe el funcionamiento del dispositivo y la funcionalidad de estos registros, está disponible en el campus virtual.

4.2.5. IMPORTANTE: Emulación del ACME 1623.

Cuando se activa la emulación del ACME 1623, el sistema se comporta como si tuviese un dispositivo real conectado al bus I2C, aunque sin necesidad de conectar realmente el dispositivo. La desconexión (o no conexión del dispositivo) puede simularse ejecutando el comando "acmé disconnect" comentado en la sección anterior.

Como ya se ha mencionado, la "guía rápida de programación" del dispositivo ACME 1623 está disponible en el campus virtual. Cuando se activa la emulación, el dispositivo emulado por el microcontrolador se comporta exactamente como se describe en dicha guía. Los **pines del dispositivo emulado son implementados por el propio microcontrolador mediante el puerto B** (los pines PB0 a PB3 se comportan como las líneas GPIO0 a GPIO3 del dispositivo ACME), **y la señal de salida /INT del ACME es implementada por el pin PA5 del microcontrolador**. Por ello, con la emulación activada se debe **evitar el uso de los pines PB0:3 y del pin PA5**.

Una vez completada la implementación, si nuestro sistema tuviese conectado el dispositivo ACME real podría desactivarse la emulación y liberarse estos recursos, en cuyo caso nuestro sistema final contaría con un mayor número de líneas de entrada salida gracias a la incorporación de este dispositivo.

Para integrar la emulación del dispositivo ACME en nuestro proyecto, es necesario **copiar la carpeta "emul" del proyecto de ejemplo a nuestro proyecto**. Esta carpeta contiene una biblioteca que es llamada a su vez desde la biblioteca i2c_driver.c para realizar la emulación.

4.3 Funcionalidad a implementar (EP31): Completar la biblioteca I2C_driver para soportar todas las operaciones requeridas. (0,4 puntos)

Nuestro equipo de diseño ha implementado parcialmente, pero no de forma completa la funcionalidad de la biblioteca i2c_driver.c que se encarga de realizar transacciones de escritura, lectura e híbridas (escritura-lectura) por el bus I2C. Sólo está implementada la escritura, por lo que las operaciones de lectura o híbridas fallan. Por este motivo no es posible habilitar en el proyecto el código de inicialización del acelerómetro, no se pueden realizar lecturas del mismo y no se pueden realizar operaciones que impliquen lecturas de valores de los registros del dispositivo ACME 1623 (por eso al arrancar, el código indica que no reconoce el ACME 1623, ya que no puede leer el registro ID para comprobar que el dispositivo conectado en la dirección 0x2C corresponde con él). Tú te encargarás de completar la funcionalidad.

Completa el funcionamiento de la biblioteca i2c_driver.c, de forma que se soporten también las operaciones de lectura y las operaciones híbridas de escritura-lectura por el bus I2C. Para ello debes analizar el funcionamiento de la biblioteca i2c_driver.c y completar de forma correcta la rutina de interrupción. Para probar esta funcionalidad dispones del comando "acme" del intérprete de comandos (sección 4.2.4). Una vez completado correctamente el código de la biblioteca i2c_driver, el sensor ACME 1623 debería ser correctamente detectado por el sistema, y deberías poder realizar las diferentes operaciones desde el intérprete de comandos.

Para realizar esta implementación debes tener en cuenta que al igual que las transacciones de escritura ya implementadas, las de lectura podrían constar de un único byte o de varios. De la misma forma, las transacciones híbridas pueden constar de la escritura de uno o más bytes seguidos de la lectura de uno o más bytes.

Es por ello que para gestionar adecuadamente los diversos tipos de transacciones I2C, el manejador de interrupción del I2C debe implementar una máquina de estados como la que se representa en la figura 2 (ya está parcialmente implementada, debes extender lo que hay implementado para soportar las operaciones de lectura y escritura-lectura):

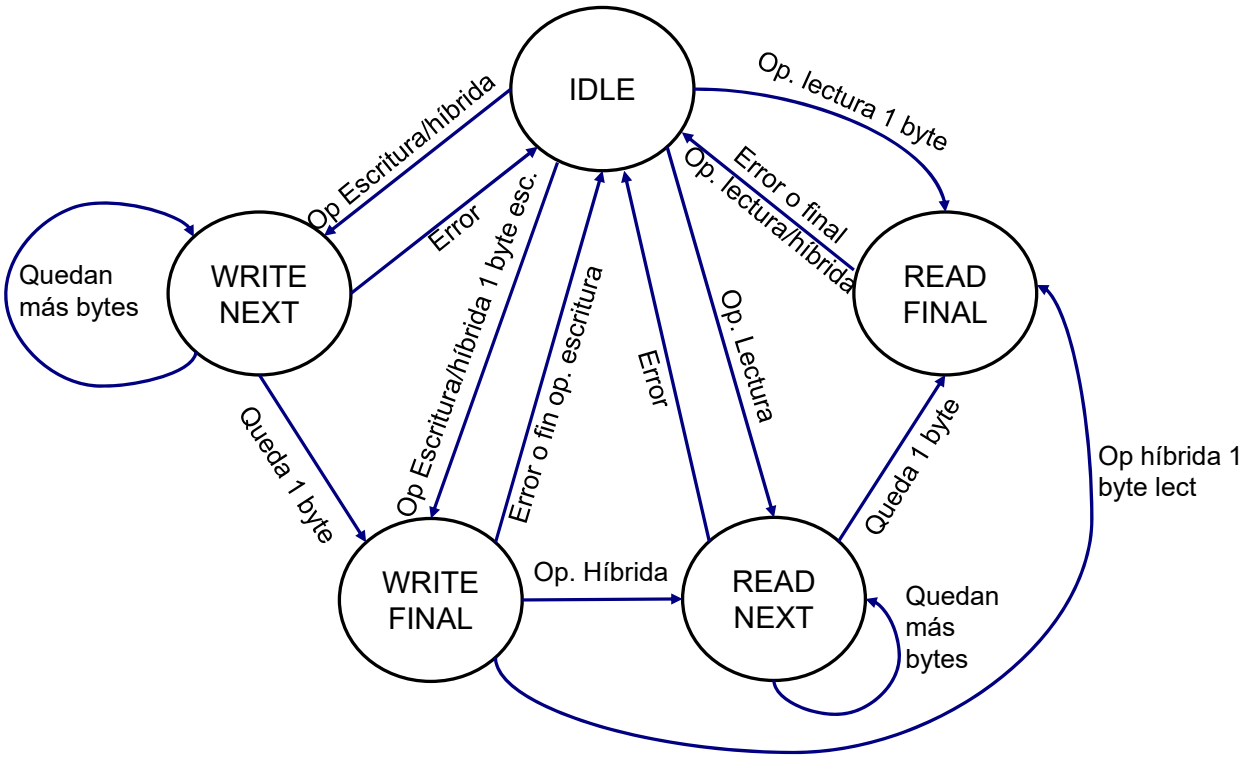

Figura 3. Máquina de estados que debe implementar la rutina de interrupción I2C.

El estado IDLE es el estado en el que no hay ninguna transacción en marcha (el estado inicial, en el que el bus está en reposo). En el código suministrado, cuando se arranca una transacción de escritura, se transmite el primer byte y se cambia de estado al estado WRITE_NEXT o al estado WRITE_FINAL en función de si quedan más bytes que enviar o no. Cuando se produce la interrupción (porque se ha completado el envío del byte), si estamos en el estado WRITE_NEXT enviamos el siguiente byte y cambiamos al estado WRITE_FINAL si era el último byte a enviar, o permanecemos en WRITE_NEXT si hay más bytes que enviar. Si cuando se produce la interrupción de transmisión (de nuevo porque se ha completado el envío de un byte) y estamos en el estado WRITE_FINAL, se detiene la operación y volvemos al estado IDLE. Cualquier error de operación provoca la vuelta al estado IDLE y el descarte de la operación en curso.

Al implementar las transacciones de lectura y escritura-lectura se deben implementar el resto de estados y transiciones:

- Al arrancar una operación de lectura pasaremos desde el estado IDLE al estado READ_FINAL o READ_NEXT en función de si queda pendiente la lectura de un único byte o de varios. Cuando se produzca la interrupción de recepción, podremos obtener el byte recibido. Si estamos en el estado READ_FINAL, debemos detener la transacción y volver al estado IDLE. Si estamos en el estado

READ_NEXT, tendremos que cambiar al estado READ_FINAL si sólo queda 1 byte pendiente de recibir, o permanecer en READ_NEXT en caso contrario.

- Si la transacción es de escritura-lectura, se ejecutará primero la parte de escritura y luego la de lectura. En este tipo de transacción, cuando finaliza el estado WRITE_FINAL, en lugar de volver al estado IDLE debe cambiarse al estado READ_NEXT o READ_FINAL en función de si hay que recibir un único byte o más de uno

Para la comunicación entre las funciones utilizadas por las tareas y la rutina de interrupción se utilizan dos mecanismos de IPC: una cola de mensaje para almacenar los tipos de operaciones a realizar y sus parámetros (datos a enviar y recibir), y las notificaciones directas a tarea de FreeRTOS que permite que las tareas con una operación i2c en curso o encolada esperen a que se realice y complete la misma. Al finalizar la transacción (o en caso de error), la rutina de interrupción eliminará la operación de la cola de mensajes, dará paso a la tarea utilizando el correspondiente flag de notificación (operación completada o error), mediante las notificaciones directas a tarea y volverá al estado IDLE.

En caso de haber varias tareas intentando realizar una operación I2C, sus peticiones se irán encolando. Sólo una de ellas tendrá acceso al I2C (la primera de la cola) y el resto quedarán a la espera de que se vayan cursando las peticiones precedentes. Una vez que termine la transacción en curso, se pasará a la siguiente de la cola.

Para obtener los datos recibidos o escribir los datos a enviar por el bus I2C, así como para poder controlar la operación realizada por el periférico I2C como maestro del bus, se utilizan las funciones del módulo I2C.c de las bibliotecas TIVAWARE, implementada por Texas Instruments y documentadas en el manual del programador de TIVAWARE. En concreto las funciones son las siguientes:

- **I2CMasterDataPut**: Escribe un dato en el buffer de salida del periférico para proceder a su envío. Una vez se complete el envío saltará la interrupción de transmisión.
- **I2CMasterDataGet**: Lee el último dato recibido del buffer de recepción del periférico. En recepción, los datos recibidos estarán disponible para su lectura cuando salte la interrupción, y no antes.
- **I2CMasterSlaveAddrSet**: Establece la dirección del dispositivo esclavo con el que se va a realizar la transacción (debe hacerse antes de iniciarla).
- **I2CMasterControl**. Establece el estado del periférico I2C como maestro del bus, controlando si se arranca, continúa o finaliza los diferentes tipos de transacciones.

La documentación de estas funciones deberá consultarse para poder implementar la funcionalidad pedida.

Antes de intentar implementar la funcionalidad que debe tener esta biblioteca se recomienda **analizar el funcionamiento del código de partida** suministrado.

Como ya se ha mencionado, para comprobar la correcta implementación de esta funcionalidad dispones del comando “acme” del intérprete de comandos (sección 4.2.4). Una vez completado correctamente el código de la biblioteca i2c_driver, el sensor ACME 1623 debería ser correctamente detectado por el sistema, y deberías poder realizar las diferentes operaciones desde el intérprete de comandos (excepto la

operación de configuración del tipo de interrupción escribiendo en los registros INT_TYPE_A e INT_TYPE_B, que aún no está implementada en el driver el ACME 1623).

4.4 Funcionalidad a implementar (EP32): Control del sensor ACME.

4.4.1 EP32.A. Completar la biblioteca ACME_1623.c (0,25)

Implementa la función ACME_setIntTriggerType del driver del ACME 1623, de forma que permita la escritura de los registros INT_TYPE y ADC_ INT_TRIGGER del dispositivo ACME en una **única transacción I2C de escritura**.

Implementa la función ACME_readADC del driver del ACME 1623, de forma que permita leer las muestras convertidas del ADC (los 4 canales) en una única una **única transacción I2C de escritura-lectura.**

Estas funciones las puedes probar con el intérprete de comandos de terminal serie.

4.4.2 EP32.B. Uso del sensor ACME desde el interfaz de usuario Qt (0,3 puntos)

Una vez implementada la modificación anterior implementa las modificaciones necesarias que permitan utilizar la funcionalidad del dispositivo ACME 1623 desde el interfaz de usuario Qt. En particular queremos que:

- Los pines de salida GPIO2 y GPIO3 se puedan controlar (encender o apagar) desde el interfaz de usuario mediante los controles adecuados (por ejemplo, *checkboxes*).
- Los pines GPIO0 y GPIO1 del ACME1623 se configurarán como entradas y se activarán sus interrupciones. El estado de los pines de entrada se podrá monitorizar desde el interfaz Qt mediante indicadores de tipo LED de la biblioteca *analogwidgets*.
- Se realizará una lectura de los datos del ADC cuando éstos se actualicen debido a la activación de la señal de disparo TRG. Esto también conllevará el control de las interrupciones generadas por el dispositivo ACME 1623. Los valores leídos (cuando se actualicen) se enviarán a Qt y se representarán en indicadores numéricos.

Nótese que, aunque esta funcionalidad se va a probar mediante el dispositivo ACME emulado (en la que los pines PB0-3 del microcontrolador se corresponden con los pines GPIO0-3 del dispositivo ACME, y los pines PF0-PF3 del microcontrolador con los pines ADC3-ADC0 del ACME), la funcionalidad debe ser implementada como si se estuviese utilizando el dispositivo ACME real (es decir, **está totalmente prohibido acceder directamente a los pines del puerto B** mediante las funciones del módulo GPIO de TIVAWARE o mediante acceso directo a los registros del periférico). La configuración y control de los pines deben hacerse mediante operaciones I2C con el dispositivo ACME emulado.

Para detectar las interrupciones generadas por el dispositivo mediante su señal de salida /INT (pin PA5 del microcontrolador cuando se usa el dispositivo emulado), dicha señal deberá conectarse con una entrada GPIO del microcontrolador que sí deberá configurarse y controlarse mediante las funciones del módulo GPIO de TIVAWARE para ser capaz de detectar y atender las interrupciones generadas por el dispositivo emulado. Para este efecto puede utilizarse, por ejemplo, el pin PA3.

Para notificar al interfaz de usuario el estado de los pines de entrada debe crearse una tarea que se encargue de esperar las interrupciones generadas por el dispositivo ACME a través de su pin /INT, lea el

estado de los pines de entrada y borre los flags de interrupción del dispositivo (ambas cosas mediante operaciones I2C utilizando el driver del dispositivo ACME), y notifique al interfaz de usuario el estado de dichos pines.

4.5 Funcionalidad a implementar (EP33): Sensor inercial BMI160.

Una vez implementadas las especificaciones anteriores, la última parte del proyecto se va a dedicar a trabajar con el sensor inercial BMI160, que se os va a dejar en préstamo.

Para conectarlo al microcontrolador y a la placa launchpad, debéis de tener en cuenta que el sensor que se os deja en préstamo ya va montado en una PCB junto con un regulador, por lo que la tensión a la que hay que alimentarlo es 5V (el sensor BMI160 se alimenta con 3.3v, pero la placa hay que alimentarla a 5 y el regulador interno se encarga de suministrar 3.3 al BMI160). Además de alimentación y masa hay que conectar las señales SDA y SCL a los pines del microcontrolador que se han configurado para dicha funcionalidad (PA7 y PA6, respectivamente). La siguiente tabla muestra las conexiones que se deben realizar entre la placa del sensor y la placa launchpad TM4C123:

Pin placa BM160	Pin placa TIVA Lauchpad	Funcionalidad
VIN	5V (VBUS)	Alimentación
GND	GND	Alimentación
SDA	PA7	Señal de datos I^2C
SCL	PA6	Señal de reloj I^2C

Una vez conectado el dispositivo puedes descomentar las líneas que se encargan de inicializarlo y que estaban desactivadas (líneas 574 a 579 del fichero commands.c del ejemplo de partida, y llamada a la función *I2CLoopbackEnable* en i2c_driver.c, aunque en tu proyecto han podido quedar colocadas en otro lugar), y se puede utilizar el comando "*readacc*" del intérprete de comandos para probar la lectura de datos de aceleración.

Tras la conexión del dispositivo se piden implementar la funcionalidad indicada en las subsecciones 4.5.1.1 y 4.5.2.

4.5.1 EP33.A. Completar la biblioteca BMI160.c

La biblioteca suministrada permite la inicialización del dispositivo y la lectura de datos de aceleración, pero no permite la lectura de datos de los giróscopos ni la configuración del rango de medida de los acelerómetros. Se pide:

- Completar la función que permite obtener los datos de los giróscopos
- Completar la función que permite configurar el rango de aceleración a +-2G, +-4G, +-8G y +-16G
- Completar la función que permite configurar el rango del acelerómetro.

4.5.2 EP33.B. Envío de datos de aceleración y giro hacia el interfaz QT y representación en el mismo.

Esta funcionalidad será posible detenerla o arrancarla desde el interfaz de usuario. Cuando esté activa, se leerán de forma periódica los valores de aceleración y giro en los 3 ejes y se enviarán al interfaz de usuario Qt, donde se representarán en una gráfica (2 gráficas, una para la aceleración y otra para la velocidad angular, cada una con tres curvas, correspondientes a los ejes X, Y y Z de cada magnitud). La frecuencia con la que se realizará la medida será de 25Hz, controlada por un *timer software* de FreeRTOS. Los valores de aceleración representados deben mostrarse en "unidades g", independientemente del rango de aceleración seleccionado. En el caso del giróscopo, se representará en grados por segundo.

Esta funcionalidad debe poder funcionar "en paralelo" con otras funcionalidades del sistema desarrolladas durante esta tercera parte del proyecto o en partes anteriores.

4 Puntuaciones (1,5p en total).

- EP31 Integración de los nuevos componentes software en el código del proyecto propio (resultante de la segunda parte de las especificaciones) y completar la biblioteca i2c_driver con el soporte de las transacciones de lectura y escritura-lectura. **0,4 puntos**
- EP32. Control del dispositivo ACME: **0,55 puntos**
- EP33: Control del BMI160 y envío de datos a Qt: **0,55 puntos**