Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Escuela de Ingeniería Electrónica

Sistemas Digitales II

Trabajo Práctico N° 2

Comunicaciones on-board: Protocolo I2C. Protocolo SPI.

Acelerómetro. OLED. Modos de bajo consumo.

Autor/es:

|  |  |
| --- | --- |
| Grupo N° 3 | |
| Nombres y Apellido | N° de Legajo |
| Bellini Valentin | B-6127/1 |
| Iván Saitta | S-5435/6 |

|  |  |
| --- | --- |
| Corrigió | Calificación |
|  |  |

2024

Índice

## **Introducción**

Este trabajo práctico aplica los contenidos temáticos de la asignatura al desarrollo de un controlador implementado en un dispositivo de la familia KL43 de la placa de desarrollo FRDM-KL43Z. El funcionamiento del sistema se modela utilizando el formalismo de Máquina de Estado Finito / Statecharts UML y el código C debe reflejar el modelo propuesto. El desarrollo se apoya en las funciones de biblioteca provistas por el fabricante.

El controlador se comunicará con el acelerómetro incluido en la placa de desarrollo mediante el protocolo I2C y presentará resultados en un display OLED que se encuentra conectado a la placa comunicado por SPI. Se aplicarán conocimientos referidos a los modos de bajo consumo del microcontrolador.

La aplicación se programará y depurará utilizando el ambiente MCUXpresso y las bibliotecas asociadas.

## **Objetivos**

Objetivos cognitivos:

Se espera que los alumnos sean capaces de:

1. Especificar el comportamiento del sistema utilizando el modelo de Máquina de Estado Finito / Statecharts UML.

2. Aplicar los conocimientos adquiridos sobre la arquitectura de la familia de microcontroladores KL46 para desarrollar una aplicación basada en la placa FRDM-KL46Z.

3. Incorporar distintos modos de trabajo del microcontrolador para reducir el consumo del sistema.

4. Aplicar los conocimientos adquiridos sobre el protocolo on-board I2C.

5. Incorporar en la implementación periféricos de la placa de desarrollo (acelerómetro).

6. Incorporar un display gráfico externo a la placa comunicado por SPI.

7. Utilizar las funciones de biblioteca suministradas por el fabricante para soportar el desarrollo de la aplicación software.

8. Aplicar el criterio de reutilización de código al definir la estructura del proyecto. Dividiendo las distintas funcionalidades o MEFs en archivos separados.

Objetivos actitudinales:

1. Promover el trabajo en equipo para obtener la solución a un problema.

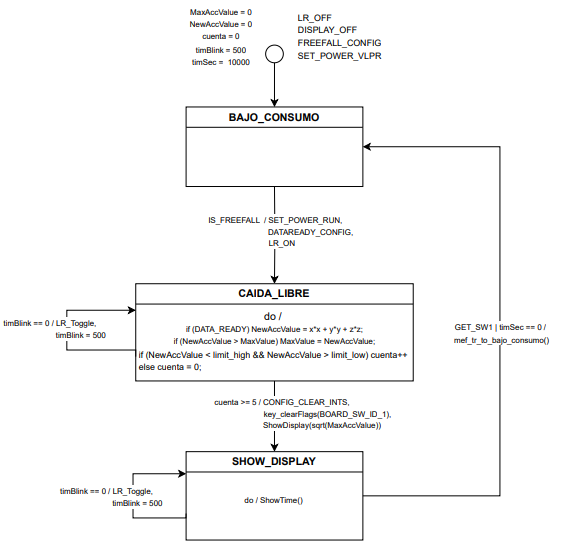
2. Promover la habilidad de realizar una defensa de la solución propuesta para el problema planteado.

3. Promover la habilidad de elaborar un reporte escrito sobre el trabajo realizado.

## **Tareas Desarrolladas**

Para la resolución del trabajo práctico planteado, se partió desde el diseño de la MEF que modela una posible solución al problema. Tres estados marcados:

* “Modo de Bajo Consumo”: Estado donde se encuentra el programa cuando es inicializado. En este estado no hay ninguna acción mas que poner el micro en bajo consumo y esperar una interrupción dada por la caída libre del acelerómetro.
* “Caída Libre”: Una vez que se da la caída libre, el microprocesador pasa a estado Run, con el clock en 48MHz. Se configura el acelerómetro para leer las aceleraciones correspondientes e ir almacenando el valor máximo medido.
* “Show Display”: En este último estado se entra cuando la caída libre ha finalizado, determinando esto mediante un algoritmo que se detallará más adelante. En este momento, se hace la comunicación por SPI con el display OLED para informar en el mismo la aceleración máxima obtenida. Cuando pasen 10 segundos o se presiones el SW1, el programa vuelve al estado inicial: “Modo de Bajo Consumo”.



Para el análisis de la solución planteada vamos a abordar cada tema por separado.

### **Acelerómetro.**

La comunicación con el acelerómetro en la placa de desarrollo FRDM-KL43Z se realiza mediante el protocolo I2C. Para Inicializar el módulo I2C contamos con un archivo de la catedra (SD2\_I2C.c) con una función que, libera el bus en caso de que se haya reiniciado el micro, pero no el acelerómetro, configura los puertos y pines correspondientes a las líneas de SDA y SCL, y carga una configuración para el I2C0.

Para la lectura y escritura de los registros del acelerómetro a través de I2C, se utilizan dos funciones de la cátedra y una implementada por el grupo. Las dos primeras permiten leer o escribir un registro particular del acelerómetro al pasarle la dirección del mismo. La tercera función implementada por el equipo, permite leer múltiples registros en un buffer de tamaño variables. Esta última se utiliza para leer las aceleraciones en una sola transferencia y así hacerlo mas rápido. Todas las funciones hacen una transferencia de tipo “Blocking”, que si bien no es lo mejor para el rendimiento, es más sencilla de implementar.

Todos los registros del acelerómetro se encuentran traducidos a estructuras de datos de bytes en el archivo de cabecera “mma8451\_registers.h” junto con las direcciones de cada uno de estos registros.

Se cuenta con 3 configuraciones principales para los registros de los acelerómetros:

* mma8451\_freefall\_config(): Implementa una configuración para que el acelerómetro interrumpa en el pin INT1 cuando detecta caída libre. Para esto debe setearse un threshold, un debounce y habilitar las interrupciones, entre otros parámetros que visualizamos en el código con sus comentarios correspondientes.
* mma8451\_dataReady\_config(): Implementa una configuración para que el acelerómetro interrumpa en el pin INT1 pero ahora cuando termine la conversión de un nuevo dato. En este momento el acelerómetro está continuamente conviertiendo mientras leemos las aceleraciones correspondientes con la función para leer múltiples registros anteriormente mencionada. Para la configuración se setea el format scale en 4G, un ODR de 200 Hz para tener una conversión cada 5 ms y se habilita el fast-Read mode para lograr una conversión de 8 bits que pierde resolución pero se realiza con mayor velocidad.
* mma8451\_clearInterruptions\_config(): Esta configuración deshabilita las interrupciones tanto por freefall como por dataready y se utiliza cuando se quiere mostrar el resultado por display ya que en dicho momento no se necesita ningún servicio del acelerómetro.

Por último tenemos el handler de la interrupción del acelerómetro. Dado que ambas interrupciones en uso (freefall y dataready) se han ruteado al pin INT1, es necesario verificar de donde viene la interrupción a partir de leer el registro INT\_SOURCE\_t. Si la interrupción viene por freefall, se setea una bandera en true (que se limpia cuando es leída) y se pasa el microprocesador a modo run. Si la interrupción viene por dataready, se setea su bandera correspondiente en true (que también se limpia cuando es leída), y se leen los registros del acelerómetro para guardar las 3 lecturas y poder utilizarlas en la lógica del programa.

La lectura de las conversiones del acelerómetro en sus registros se hace mediante la función readAccsFromRegisters(void)

|  |
| --- |
|  |
| void fastReadAcelerationsFromRegister(void){  // Operate in Fast Read Mode (resolution: 8 bits)  uint8\_t buffer[3];  mma8451\_read\_multiple\_reg(0x01,buffer, sizeof(buffer));  read.X = (int8\_t)buffer[0];  read.Y = (int8\_t)buffer[1];  read.Z = (int8\_t)buffer[2];  } |

Donde, al trabajar con FREAD en 1, el autoincremento del mapa de direcciones se saltea las partes menos significativas de la conversión, teniendo solo en cuenta 8 bits por conversión que son asignados a una estructura en la función mma8451\_read\_multiple\_reg(uint8\_t addr, uint8\_t \*pbuffer, uint8\_t size).

### **Modos de Consumo**

Para el intercambio entre los modos de consumo utilizados en el trabajo (Run y VLPR), se utiliza una función encontrada en las resoluciones planteadas por la cátedra. En el archivo “power\_mode\_switch.c” nos encontramos con una función que permite intercambiar el modo de consumo del micro entre dos estados. Esta función se utiliza en dos ocasiones, para cambiar de modo Run a modo VLPR y viceversa.

Además, contamos con una función para chequear el estado en el cuál nos encontramos y poder imprimirlo por pantalla junto con la frecuencia del clock que se está utilizando.

Es importante el llamado en el main a las inicializaciones correspondientes para poder configurar los clocks a máxima frecuencia y obtener el modo run a 48MHz. Para el caso de VLPR, la configuración setea el micro a una frecuencia de 2MHz.

### **SSD1306**

SSD1306 es el controlador para el display OLED por el cuál se muestran los resultados de la aceleración máxima leída. El mismo funciona con SPI.

La configuración de SPI se puede ver en “SD2\_board.c”, precisamente en la función “board\_configSPI1()”. La placa de desarrollo utilizada tiene 2 módulos de SPI y en este caso se utiliza SPI1 debido a las comodidades del equipo para la utilización de los pines. La configuración para SPI es más sencilla que para I2C, ya que se utiliza una configuración por defecto y se le agrega 3 parámetros que deben coincidir con el display OLED para una correcta configuración: Polaridad del clock (1), Fase del clock (1) y Baud Rate (4MBps).

Para enviar los datos por SPI se utiliza una transferencia de tipo Blocking indicando el dato del transmisor (buffer) y la longitud del mismo. Para el manejo del controlador se utiliza una librería de terceros y sobre esta se utiliza un archivo de la cátedra que tiene funciones de utilidad, entre ellas: limpiar la pantalla del display, dibujar un rectángulo, colocar un string en la pantalla, etc.

Estas funciones se utilizan para poder colocar el dato entero medido de la conversión del acelerómetro, como un string en el display.

### **Lógica**

Ahora que se describió como funciona cada uno de los drivers utilizados para el desarrollo del trabajo, se desarrolla la lógica implementada para la medición de aceleración correcta.

Para entrar al estado de caída libre, solo basta con habilitar las interrupciones por freefall y entonces el acelerómetro por si solo, sabiendo el threshold y el debounce, arrojará una interrupción cuando esto suceda. Aquí es donde debe variarse estos parámetros según el requerimiento de la implementación. Bajar el threshold hará el dispositivo menos sensible, mientras que subirlo puede hacer que se detecte más fácil la caída libre como también algún movimiento brusco. Un debounce alto nos ayudará a no leer falsos positivos en una caída libre, pero también nos limitará el mínimo de altura que se requiere para que se atienda la interrupción antes que el micro llegue al suelo.

Una vez que el acelerómetro detecta la caída libre, el microprocesador se pone en modo run, para trabajar con mayor velocidad, y se pone el ODR en 200 Hz para realizar lecturas cada 5ms. Además, se pone el F\_READ (del registro 1) en alto, para habilitar el fast-read mode, que limita la resolución a 8 bits, pero hace que las lecturas sean más rápidas.

Durante la caída libre, el micro realizará las conversiones y al leerlas, son comparadas con un valor máximo (inicialmente en 0) para ir sobrescribiendo el nuevo máximo, si es necesario, en cada lectura.

**If** (NewAccValue >= MaxAccValue) MaxAccValue = NewAccValue;

Cuando la lectura se encuentra dentro de un 5% de los límites del estado de reposo (medición correspondiente a 1G), se suma una variable de cuenta. Si la lectura está por fuera de estos límites planteados, se setea la variable cuenta nuevamente en cero. Así, cuando la variable cuenta llegue al valor seteado encontrado por simulación (cinco), se entiende que el micro ya se encuentra en la superficie y se cambia al siguiente estado para mostrar la aceleración máxima por el display OLED.

**if** (NewAccValue < limit\_high && NewAccValue > limit\_low{

cuenta++;

} **else** {

cuenta = 0;

}

## **Equipamiento Utilizado**

Para la realización del trabajo práctico fue necesario el siguiente equipamiento:

* Placa de desarrollo FRDM-KL43Z
* Display OLED 128x64 SPI
* 7 cables de conexión de tipo hembra-macho

## **Resultados Obtenidos**

Los resultados obtenidos fueron favorables, el programa cumple con la función requerida por el trabajo y responde a las configuraciones que han sido seteadas. Se realizaron diversidad configuraciones para diferentes escenarios, variando threshold, debounce, modos de operación, entre otros, con el objetivo de verificar el correcto entendimiento del programa. En todos estos casos, la solución era adecuada con la correspondiente configuración.

## **Conclusiones**

Logramos concluir en que el problema presentado era un trabajo integral, que requería conocimiento de todos los temas que aborda: comunicaciones ob-board (I2C y SPI), acelerómetro, display OLED y modos de bajo consumo.

Si bien los resultados obtenidos fueron los esperados, la mayor parte del tiempo fue dedicada a la configuración del acelerómetro y la calibración del mismo para evitar falsos positivos en la detección de caída libre y optimizar el proceso de lectura de datos. La calibración se hizo a partir de reiteradas pruebas con distintas configuraciones, pero nos ayudó a aprender a leer y comprender todos los registros necesarios de un módulo a partir de su hoja de datos: el acelerómetro.

Otro gran inconveniente que tuvimos fue que, al elegir rutear ambas interrupciones del acelerómetro al pin INT1, se nos imposibilitó utilizar las funciones de la cátedra para la comunicación con el OLED por el SPI0 debido a que estas dos implementaciones compartían el mismo puerto y pin del microprocesador. Si bien la solución a este problema no es algo complejo (implementamos una nueva función para el SPI1), encontrar el problema fue un desafío, ya que no sabíamos por qué motivo se estaba interrumpiendo la comunicación SPI con el display.

## **Bibliografía**

[1] Gunther Gridling, Bettina Weiss, “Introduction to Microcontrollers”, Vienna University of Technology, Institute of Computer Engineering, Embedded Computing Systems Group February 26, 2007, Version 1.4.

[2] NXP Semiconductors, “KL43 Sub-Family Reference Manual”, Document Number: KL43P64M48SF6RM, July 2016, Rev. 5.1.

[3] NXP Semiconductors “MCUXpresso IDE User Guide”, 26 October, 2023, Rev. 11.8.0.

[4] Freescale Semiconductor. Application Note: “Motion and Freefall Detection Using the MMA8451, 2, 3Q”. Document Number: AN4070 Rev 1, 10/2011.

[5] NXP Semiconductors Data sheet: Technical data. “MMA8451Q, 3-axis, 14-bit/8-bit digital accelerometer”. Document Number: MMA8451Q Rev. 10.1, 05/2016.

## **Anexo**

|  |
| --- |
| Main.c |
| **int** **main**(**void**) {  /\* Inicialización de clocks a máxima frecuencia y uc en modo RUN a 48MHz \*/  init\_clocks\_and\_power\_mode();  /\* Inicialización FSL debug console. \*/  BOARD\_InitDebugConsole();  /\* Inicialización de funciones de la placa \*/  board\_init();  /\* Inicialización de SPI y display OLED \*/  board\_configSPI1();  oled\_init();  oled\_setContrast(16);  oled\_clearScreen(*OLED\_COLOR\_BLACK*);  /\* Inicialización del I2C \*/  SD2\_I2C\_init();  /\* Inicialización MEF de pulsadores\*/  key\_init();  /\* Se configura interrupción de systick \*/  SysTick\_Config(SystemCoreClock / 1000U);  /\* Inicialización de la MEF del programa --> VLPR\*/  mef\_init();  **while**(1) {  mef();  }  **return** 0;  } |

|  |
| --- |
| **SysTick\_Handler** |
| **void** **SysTick\_Handler**(**void**){  key\_periodicTask1ms();  mef\_task1ms();  } |

|  |
| --- |
| Macros MEF.C |
| /\* ------------------------- Macros IO ------------------------- \*/  **#define** LR\_ON board\_setLed(BOARD\_LED\_ID\_ROJO, BOARD\_LED\_MSG\_ON)  **#define** LR\_OFF board\_setLed(BOARD\_LED\_ID\_ROJO, BOARD\_LED\_MSG\_OFF)  **#define** LR\_TOGGLE board\_setLed(BOARD\_LED\_ID\_ROJO, BOARD\_LED\_MSG\_TOGGLE)  **#define** GET\_SW1 key\_getPressEv(BOARD\_SW\_ID\_1)  /\* ------------------------- Macros Timers ------------------------- \*/  **#define** T\_BLINK 500  **#define** T\_SEC 10000  /\* ------------------------- Macros Acelerómetro ------------------------- \*/  **#define** CONFIG\_CLEAR\_INTS mma8451\_clearInterruptions\_config()  **#define** IS\_FREEFALL mma8451\_getFreefallInterruptStatus()  **#define** IS\_DATA\_READY mma8451\_getDataReadyInterruptStatus()  **#define** GRAVITY\_BOUND 0.05 |

|  |
| --- |
| **mef\_init**() y **mef\_Tr\_To\_Bajo\_Consumo**() |
| **extern** **void** **mef\_init**(){  /\* Set Variables \*/  MaxAccValue = 0;  NewAccValue = 0;  cuenta = 0;  timBlink = T\_BLINK;  timSec = T\_SEC;  state = *BAJO\_CONSUMO*;  /\* Settings \*/  LR\_OFF;  mma8451\_freefall\_config(); // Freefall interrupt configured  APP\_PowerModeSwitch(*kSMC\_PowerStateRun*, *kAPP\_PowerModeVlpr*);  oled\_clearScreen(*OLED\_COLOR\_BLACK*);  oled\_putString(30, 29, (uint8\_t\*)"Bajo Consumo" , *OLED\_COLOR\_WHITE*, *OLED\_COLOR\_BLACK*);  /\* Verification and Print \*/  currentPowerState = SMC\_GetPowerModeState(SMC);  APP\_ShowPowerMode(currentPowerState);  freq = CLOCK\_GetFreq(*kCLOCK\_CoreSysClk*);  PRINTF(" Core Clock = %dMHz \r\r", freq/1000000);  }  **extern** **void** **mef\_Tr\_To\_Bajo\_Consumo**(){  mef\_init();  } |

|  |
| --- |
| **Mef()** |
| **extern** **void** **mef**(){  **switch**(state){  **case** *BAJO\_CONSUMO*:  **if**(IS\_FREEFALL){  state = *CAIDA\_LIBRE*;  LR\_ON;  mma8451\_dataReady\_config();  /\* uC is set in run mode at the mma8451 IRQ\_Hanlder \*/  }  **break**;  **case** *CAIDA\_LIBRE*:  **if** (timBlink == 0){  LR\_TOGGLE;  timBlink = T\_BLINK;  }  **if**(IS\_DATA\_READY){ // Si la conversión finalizó  int16\_t x = mma8451\_getAcX();  int16\_t y = mma8451\_getAcY();  int16\_t z = mma8451\_getAcZ();  NewAccValue = x\*x + y\*y + z\*z;  **if**(NewAccValue >= MaxAccValue) MaxAccValue = NewAccValue;  **if** (NewAccValue < limit\_high && NewAccValue > limit\_low){  cuenta++;  } **else** {  cuenta = 0;  }  }  **if**(cuenta >= 5){  state = *SHOW\_DISPLAY*;  MaxAccValue = sqrt(MaxAccValue);  CONFIG\_CLEAR\_INTS;  key\_clearFlags(*BOARD\_SW\_ID\_1*);  **char** max\_acc\_str[16];  oled\_clearScreen(*OLED\_COLOR\_BLACK*);  oled\_fillRect(8, 16, 56+64, 16+32, *OLED\_COLOR\_WHITE*);  oled\_fillRect(12, 20, 56+64-4, 16+32-4, *OLED\_COLOR\_BLACK*);  sprintf(max\_acc\_str, "Max Acc: %d.%d g",MaxAccValue/100,MaxAccValue-(MaxAccValue/100)\*100); // Similar a un string copy  oled\_putString(20, 29, (uint8\_t \*)max\_acc\_str , *OLED\_COLOR\_WHITE*, *OLED\_COLOR\_BLACK*);  }  **break**;  **case** *SHOW\_DISPLAY*:  **if** (timBlink == 0){  LR\_TOGGLE;  timBlink = T\_BLINK;  }  **char** char\_time[16];  sprintf(char\_time, "%d.%03d S", timSec / 1000, timSec % 1000);  oled\_putString(46, 2, (uint8\_t \*)char\_time , *OLED\_COLOR\_WHITE*, *OLED\_COLOR\_BLACK*);  **if**(!timSec | GET\_SW1){  mef\_Tr\_To\_Bajo\_Consumo();  }  **break**;  }  } |

|  |
| --- |
| **mef\_task1ms** |
| **extern** **void** **mef\_task1ms**(**void**){  **if**(state != *BAJO\_CONSUMO* && timBlink) timBlink--;  **if**(state == *SHOW\_DISPLAY* && timSec) timSec--;  } |

|  |
| --- |
| **mma8451\_read\_multiple\_reg**(uint8\_t addr, uint8\_t \*pbuffer, uint8\_t size) |
| **static** **void** **mma8451\_read\_multiple\_reg**(uint8\_t addr, uint8\_t \*pbuffer, uint8\_t size){  // La funcino carga los datos leidos en el buffer  // Ej: Leer desde 0x01 a 0x06 (size 6 desde address inicial: 0x01)  i2c\_master\_transfer\_t masterXfer;  memset(&masterXfer, 0, **sizeof**(masterXfer));  masterXfer.slaveAddress = MMA8451\_I2C\_ADDRESS;  masterXfer.direction = *kI2C\_Read*;  masterXfer.subaddress = addr;  masterXfer.subaddressSize = 1;  masterXfer.data = pbuffer;  masterXfer.dataSize = size;  masterXfer.flags = *kI2C\_TransferDefaultFlag*;  I2C\_MasterTransferBlocking(I2C0, &masterXfer);  } |

|  |
| --- |
| **mma8451\_freefall\_config**(**void**) |
| **void** **mma8451\_freefall\_config**(**void**){  /\* Registers definitions \*/  CTRL\_REG1\_t ctrl\_reg1;  CTRL\_REG4\_t ctrl\_reg4;  CTRL\_REG5\_t ctrl\_reg5;  FF\_MT\_CFG\_t FF\_MT\_CFG\_reg;  FF\_MT\_THS\_t FF\_MT\_THS\_reg;  FF\_MT\_COUNT\_t FF\_MT\_COUNT\_reg;  XYZ\_DATA\_CFG\_t data\_cfg;  /\* Put the device in Standby Mode: Register 0x2A CTRL\_REG1 \*/  ctrl\_reg1.ACTIVE = 0;  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  FF\_MT\_CFG\_reg.XEFE = 1 ;  FF\_MT\_CFG\_reg.YEFE = 1 ;  FF\_MT\_CFG\_reg.ZEFE = 1 ;  FF\_MT\_CFG\_reg.OAE = 0 ;  FF\_MT\_CFG\_reg.ELE = 1 ;  mma8451\_write\_reg(FF\_MT\_CFG\_ADDRESS,FF\_MT\_CFG\_reg.data);  /\* Format Scale (2g, 4g u 8g) \*/  data\_cfg.FormatScale = *FS\_4G*;  data\_cfg.HPF\_out = 0;  mma8451\_write\_reg(XYZ\_DATA\_CFG\_ADDRESS, data\_cfg.data);  FF\_MT\_THS\_reg.THS = 0x04; // 0.15g/31.25mg = 4.8  FF\_MT\_THS\_reg.DBCNTM = 1;  mma8451\_write\_reg(FF\_MT\_THS\_ADDRESS,FF\_MT\_THS\_reg.data);  FF\_MT\_COUNT\_reg.data = 0x1B; // Note: 135 ms/5 ms (steps) = 27 counts  mma8451\_write\_reg(FF\_MT\_COUNT\_ADRESS,FF\_MT\_COUNT\_reg.data);  ctrl\_reg4.INT\_EN\_DRDY = 0;  ctrl\_reg4.INT\_EN\_FF\_MT = 1; //Activo la interrupcion por FreeFall  mma8451\_write\_reg(CTRL\_REG4\_ADDRESS, ctrl\_reg4.data);  ctrl\_reg5.INT\_CFG\_DRDY = 0; // Interrupt Data Ready en el pin INT2  ctrl\_reg5.INT\_CFG\_FF\_MT = 1; // Interrupt Freefall en el Pin INT1  mma8451\_write\_reg(CTRL\_REG5\_ADDRESS, ctrl\_reg5.data);  ctrl\_reg1.DR = *DR\_200hz*;  ctrl\_reg1.ASLP\_RATE = 0B00;  ctrl\_reg1.F\_READ = 1;  ctrl\_reg1.LNOISE = 1;  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  ctrl\_reg1.ACTIVE = 1;  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  config\_port\_int1();  } |

|  |
| --- |
| **mma8451\_dataReady\_config**() |
| **void** **mma8451\_dataReady\_config**(**void**){  CTRL\_REG1\_t ctrl\_reg1;  CTRL\_REG4\_t ctrl\_reg4;  CTRL\_REG5\_t ctrl\_reg5;  FF\_MT\_CFG\_t FF\_MT\_CFG\_reg;  XYZ\_DATA\_CFG\_t data\_cfg;  /\* Put the device in Standby Mode: Register 0x2A CTRL\_REG1 \*/  ctrl\_reg1.ACTIVE = 0;  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  /\* REGISTROS DE FREEFALL, valores default \*/  FF\_MT\_CFG\_reg.XEFE = 0 ;  FF\_MT\_CFG\_reg.YEFE = 0 ;  FF\_MT\_CFG\_reg.ZEFE = 0 ;  FF\_MT\_CFG\_reg.OAE = 0 ;  FF\_MT\_CFG\_reg.ELE = 0 ;  mma8451\_write\_reg(FF\_MT\_CFG\_ADDRESS,FF\_MT\_CFG\_reg.data);  /\* Enable Data Ready Interrupt Function in the System (CTRL\_REG4) \*/  ctrl\_reg4.INT\_EN\_DRDY = 1; // Interrupciones activas por DataReady  ctrl\_reg4.INT\_EN\_FF\_MT = 0; // Interrupciones desactivadas por FreeFall  mma8451\_write\_reg(CTRL\_REG4\_ADDRESS, ctrl\_reg4.data);  /\* Route the Dara Ready Interrupt Function to INT1 hardware pin (CTRL\_REG5) \*/  ctrl\_reg5.INT\_CFG\_DRDY = 1; // Interrupt Data Ready en pin INT1  ctrl\_reg5.INT\_CFG\_FF\_MT = 0; // Interrupt Freefall en pin INT2  mma8451\_write\_reg(CTRL\_REG5\_ADDRESS, ctrl\_reg5.data);  /\* Format Scale (2g, 4g u 8g) \*/  data\_cfg.FormatScale = *FS\_4G*;  data\_cfg.HPF\_out = 0;  mma8451\_write\_reg(XYZ\_DATA\_CFG\_ADDRESS, data\_cfg.data);  /\* Put the device in Active Mode, 200 Hz, Fast-Read mode \*/  ctrl\_reg1.DR = *DR\_200hz*; // Dr 010, 200HZ 5ms Hz output data rate  ctrl\_reg1.ASLP\_RATE = 0B00;  ctrl\_reg1.F\_READ = 1;  ctrl\_reg1.LNOISE = 0; //  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  ctrl\_reg1.ACTIVE = 1; // En 0 es stanby, en 1 es active.  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  config\_port\_int1();  } |

|  |
| --- |
| **mma8451\_clearInterruptions\_config**(**void**) |
| **void** **mma8451\_clearInterruptions\_config**(**void**){  CTRL\_REG1\_t ctrl\_reg1;  CTRL\_REG4\_t ctrl\_reg4;  isInterruptionByFreeFall = false;  isInterruptionByDR = false;  ctrl\_reg1.ACTIVE = 0; // Lo pone en standby  mma8451\_write\_reg(CTRL\_REG1\_ADDRESS, ctrl\_reg1.data);  // Deshabilito ambas interrupciones  ctrl\_reg4.INT\_EN\_DRDY = 0;  ctrl\_reg4.INT\_EN\_FF\_MT = 0;  mma8451\_write\_reg(CTRL\_REG4\_ADDRESS, ctrl\_reg4.data);  config\_port\_int1();  } |

|  |
| --- |
| Flags Getters |
| bool **mma8451\_getFreefallInterruptStatus**(**void**) {  bool ret = false;  **if**(isInterruptionByFreeFall){  isInterruptionByFreeFall = false;  ret = true;  }  **return** ret;  }  bool **mma8451\_getDataReadyInterruptStatus**(**void**) {  bool ret = false;  **if**(isInterruptionByDR){  isInterruptionByDR = false;  ret = true;  }  **return** ret;  } |

|  |
| --- |
| Aceleration Getters |
| int16\_t **mma8451\_getAcX**(**void**){  **return** (int16\_t)(((int16\_t)read.X \* 100) / (int16\_t)32);  }  int16\_t **mma8451\_getAcY**(**void**){  **return** (int16\_t)(((int16\_t)read.Y \* 100) / (int16\_t)32);  }  int16\_t **mma8451\_getAcZ**(**void**){  **return** (int16\_t)(((int16\_t)read.Z \* 100) / (int16\_t)32);  } |

|  |
| --- |
| **fastReadAcelerationsFromRegister()** |
| **void** **fastReadAcelerationsFromRegister**(**void**){  // Operate in Fast Read Mode (resolution: 8 bits)  uint8\_t buffer[3];  mma8451\_read\_multiple\_reg(OUT\_X\_MSB\_ADDRESS, buffer, **sizeof**(buffer));  read.X = (int8\_t)buffer[0];  read.Y = (int8\_t)buffer[1];  read.Z = (int8\_t)buffer[2];  } |

|  |
| --- |
| IRQ Handler |
| **void** **PORTC\_PORTD\_IRQHandler**(**void**){  INT\_SOURCE\_t intSource;  STATUS\_t status;  intSource.data = mma8451\_read\_reg(INT\_SOURCE\_ADDRESS);  /\* Chequeamos de donde viene la interrupción \*/  **if** (intSource.SRC\_DRDY){  status.data = mma8451\_read\_reg(STATUS\_ADDRESS);  **if** (status.ZYXDR){  fastReadAcelerationsFromRegister();  isInterruptionByDR = true; // Set in false by reading  }  } **else** **if** (intSource.SRC\_FF\_MT) {  isInterruptionByFreeFall = true; // Set in false by reading  APP\_PowerModeSwitch(*kSMC\_PowerStateVlpr*, *kAPP\_PowerModeRun*);  }  mma8451\_read\_reg(FF\_MT\_SRC\_ADDRESS);  mma8451\_read\_reg(INT\_SOURCE\_ADDRESS);  PORT\_ClearPinsInterruptFlags(INT1\_PORT, 1<<INT1\_PIN);  } |

|  |
| --- |
| **board\_configSPI1** |
| **void** **board\_configSPI1**(){  **const** port\_pin\_config\_t port\_spi\_config = {  /\* Internal pull-up resistor is disabled \*/  .pullSelect = *kPORT\_PullDisable*,  /\* Fast slew rate is configured \*/  .slewRate = *kPORT\_FastSlewRate*,  /\* Passive filter is disabled \*/  .passiveFilterEnable = *kPORT\_PassiveFilterDisable*,  /\* Low drive strength is configured \*/  .driveStrength = *kPORT\_LowDriveStrength*,  /\* Pin is configured as SPI0\_x \*/  .mux = *kPORT\_MuxAlt2*,  };  PORT\_SetPinConfig(SPI1\_SS\_PORT, SPI1\_SS\_PIN, &port\_spi\_config);  PORT\_SetPinConfig(SPI1\_SCK\_PORT, SPI1\_SCK\_PIN, &port\_spi\_config);  PORT\_SetPinConfig(SPI1\_MOSI\_PORT, SPI1\_MOSI\_PIN, &port\_spi\_config);  CLOCK\_EnableClock(*kCLOCK\_Spi1*);  spi\_master\_config\_t userConfig;  SPI\_MasterGetDefaultConfig(&userConfig);  userConfig.polarity = *kSPI\_ClockPolarityActiveLow*;  userConfig.phase = *kSPI\_ClockPhaseSecondEdge*;  userConfig.baudRate\_Bps = 4000000U;  SPI\_MasterInit(SPI1\_MASTER, &userConfig, SPI1\_MASTER\_CLK\_FREQ);  } |

|  |
| --- |
| **board\_SPI1Send** |
| **void** **board\_SPI1Send**(uint8\_t\* buf, size\_t len){  spi\_transfer\_t xfer;  xfer.txData = buf;  xfer.rxData = NULL;  xfer.dataSize = len;  SPI\_MasterTransferBlocking(SPI1\_MASTER, &xfer);  } |