Universidad de Costa Rica | Facultad de Ingeniería | Escuela de Ingeniería Eléctrica

Bitácora de Laboratorio 4

Roberto Sánchez Cárdenas - B77059
Gabriel Jiménez Amador - B73895

San José, 31 de agosto de 2021
Laboratorio de Circuitos Digitales

Resumen

Se presentan los resultados de habilitar la interrupción IRQ del core PicoRV32 y como este la maneja al surgir una. También, se presenta un módulo capaz de leer datos de aceleración provenientes del sensor acelerómetro ADXL362 de la tarjeta Nexys 4 DDR, en donde los datos de la aceleración del eje Y y Z se muestra en una pantalla de 7 segmentos. Al módulo lector de aceleración, luego se le añadió la funcionalidad de lectura sólo cuando detecta un cambio pronunciado en uno de sus ejes, aprovechando el concepto de interrupciones.

Índice

1. Introducción

Este laboratorio tiene como fin realiza una introducción al manejo de interrupciones del core PicoRV32 y la comunicación serial SPI con módulos de la tarjeta Nexys 4 DDR. En este caso se habilita la interrupción irq, la cual es únicamente manipulada por medio del del PicoRV32, se espera poder ver el comportamiento y los retrasos que ocurren cuando se da una interrupción. Además se va a utilizar el sensor de acelerómetro para medir el movimiento espacial de la tarjeta.

Para el manejo adecuado del sensor se debe tener claro el funcionamiento del protocolo de comunicación SPI, pues la comunicación este se da por una vía que utiliza dicho protocolo.

2. Correcciones realizadas

2.1. Ejercicio 1

En el ejercicio 1 se tuvo que corregir el firmware a utilizar, pues inicialmente no se tenía claro el uso de la función `irq`, por lo que esta había sido borrada. Se utilizó el firmware original provisto por el profesor.

2.2. Ejercicio 2

En el ejercicio 2 se tuvo que cambiar tanto el firmware como el diseño del lector del acelerómetro.

2.2.1. Cambios al firmware

El principal cambia al firmware consistió en que la escritura del valor leído del acelerómetro para los ejes Y y Z se combinó ambos valores de 16 bits en uno solo de 32 bits (`z_val` como MSBs y `y_val` como LSBs) para escribir a la dirección de lectura del display de 7 segmentos.

```
#include <stdint.h>

#define LOOP_WAIT_LIMIT 2000000

#define LEDS_7SEG   0x10000000
#define Y_ADDR      0x20000000
#define Z_ADDR      0x30000000

// Put in memory
static void putuint(uint32_t i, int addr) {
*((volatile uint32_t *)addr) = i;
}

// Get from memory
static uint32_t getuint(int addr) {
return *((volatile uint32_t *)addr);
}

// Delay
static void delay() {
    uint32_t counter = 0;
    while (counter < LOOP_WAIT_LIMIT){
        counter++;
    }
}

void main() {
uint32_t y_val = 0;
uint32_t z_val = 0;

while (1) {
        // Get Y and Z axis accel values
        y_val = getuint(Y_ADDR);
        z_val = getuint(Z_ADDR);

        // Combine and write to 7 segment register
        putuint(((z_val << 16) | y_val), LEDS_7SEG);

        delay();
}
}
```

Figura 1: Cambios al firmware del ejercicio 2.

2.2.2. Cambios al diseño

Se cambiaron los estados de la máquina de estados como está descrito en la Figura 2.

En cada estado se realiza lo siguiente:

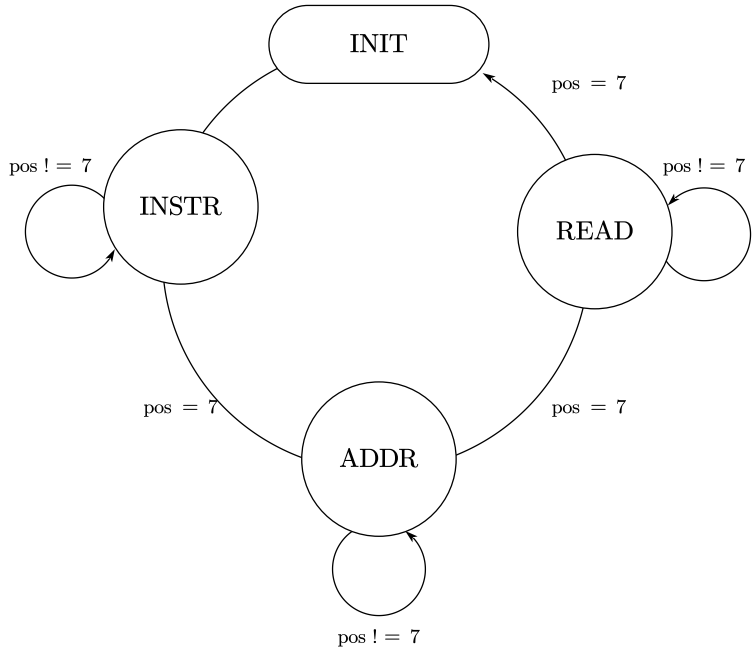

Figura 2: Diagrama de estados del Ejercicio 2

- INIT: Estado inicial
  - Se levanta `CS`.
  - Se alista el byte con la instrucción de lectura `0x0B`.
- INSTR: Se envía la instrucción de lectura al acelerómetro.
  - Se baja `CS` para iniciar comunicación con el acelerómetro ADXL362.
  - Se envía el byte con la instrucción de lectura `0x0B`.
  - La señal `pos` es un contador que señala cuantos bits se han enviado serialmente por la señal `MOSI` al ADXL362. Cuando este llega a su máximo valor de 7, pasa al estado ADDR.
- ADDR: Se envían las direcciones a leer.
  - `CS` se mantiene en bajo.
  - Se envía cíclicamente sólo una de las siguientes direcciones que corresponden a los registros de los datos de los ejes con signo extendido:
    1. YDATA_H: `0x11`
    2. YDATA_L: `0x10`
    3. ZDATA_H: `0x13`
    4. ZDATA_L: `0x12`

  - Cuando `pos` llega a 7, pasa al estado ADDR.

- READ: Se procesa el byte recibido desde el acelerómetro.
  - `CS` se mantiene en bajo.
  - No se envía nada, sólo se recibe el byte del acelerómetro según la dirección enviada.
  - Cuando `pos` llega a 7, pasa al estado INIT para reiniciar la lectura y leer de otra dirección.

2.3. Ejercicio 3

2.3.1. Cambios al firmware

Para este firmware se añadió la función para el manejo de interrupciones `irq` el cual activa una bandera si se detecta una interrupción del parámetro `irqs`. Si la bandera se encuentra activa, se procede a leer valores de los ejes, combinarlos y escribirlos al registro del display de 7 segmenos. En caso contrario, solo se despliegan 0s.

```
#include <stdint.h>
#include <stdbool.h>

#define LOOP_WAIT_LIMIT 2000000

#define LEDS_7SEG   0x10000000
#define Y_ADDR      0x20000000
#define Z_ADDR      0x30000000

bool active = false;

// Put in memory
static void putuint(uint32_t i, int addr) {
*((volatile uint32_t *)addr) = i;
}

// Get from memory
static uint32_t getuint(int addr) {
return *((volatile uint32_t *)addr);
}

// Delay
static void delay() {
    uint32_t counter = 0;
    while (counter < LOOP_WAIT_LIMIT){
        counter++;
    }
}

// Interrupt handling
uint32_t *irq(uint32_t *regs, uint32_t irqs) {
    if((irqs & 0x00000004) == 0x00000004){
        active = true;
    } else{
        active = false;
    }
    return regs;
}

void main() {
uint32_t y_val = 0;
uint32_t z_val = 0;

while (1) {
        if (active){
            // Get Y and Z axis accel values
            y_val = getuint(Y_ADDR);
            z_val = getuint(Z_ADDR);

            // Combine and write to 7 segment register
            putuint(((z_val << 16) | y_val), LEDS_7SEG);

            delay();
        } else{
            putuint(0, LEDS_7SEG);
        }
}
}
```

Figura 3: Cambios al firmware del ejercicio 3.

2.3.2. Cambios al diseño

Se cambiaron los estados de la máquina de estados como está descrito en la Figura 4.

En cada estado se realiza lo siguiente:

- INIT: Estado inicial
  - Se levanta `CS`.
  - Se alista el byte con la instrucción de escritura `0x0A` si estamos realizando la configuración inicial del ADXL362 lectura, sino se alista la instrucción de lectura `0x0A`.

- INSTR: Se envían instrucciones al acelerómetro
  - Se baja `CS`.
  - Se envía el byte con la instrucción de escritura `0x0A` si estamos realizando la configuración inicial del ADXL362 lectura, sino se alista la instrucción de lectura `0x0A`.

- Cuando `pos` llega a 7, pasa al estado ADDR.

- ADDR: Se envían las direcciones a leer.
  - `CS` se mantiene en bajo.
  - Si se encuentra configurando el acelerómetro (escritura), se envían uno por uno las siguientes direcciones [1]:
    - THRESH_ACT_L: `0x20`
    - THRESH_ACT_H: `0x01`
    - THRESH_INACT_L: `0x23`
    - THRESH_INACT_H: `0x24`
    - TIME_INACT_L: `0x25`
    - ACT_INACT_CTL: `0x27`
    - INTMAP1: `0x2A`
    - INTMAP2: `0x2B` (No es necesario pero es útil para propósitos de depuración.)
    - POWER_CTL: `0x2D`
  - Si se encuentra leyendo del acelerómetro, envía cíclicamente sólo una de las siguientes direcciones que corresponden a los registros de los datos de los ejes con signo extendido:
    1. YDATA_H: `0x11`
    2. YDATA_L: `0x10`
    3. ZDATA_H: `0x13`
    4. ZDATA_L: `0x12`
  - Cuando `pos` llega a 7, pasa al estado RW.

- RW: Se lee o escribe al acelerómetro.
  - Si se encuentra configurando el acelerómetro (escritura), se envían uno por uno las siguientos datos:
    - THRESH_ACT_L <- `0xFA`

1 Se basa en la configuración de ejemplo dado por la hoja de fabricante del acelerómetro ADXL362 para un switch con detección de movimiento autónomo (pag. 36)

- THRESH_ACT_H <- `0x00`
- THRESH_INACT_L <- `0x96`
- THRESH_INACT_H <- `0x00`
- TIME_INACT_L <- `0x1E` (Define plazo de 5 s de gracia para inactividad)
- ACT_INACT_CTL <- `0x3F` (Loop mode)
- INTMAP1 <- `0x10` (Mapea el bit de `ACTIVE` a INT1)
- INTMAP2 <- `0x20` (Mapea el bit de `INACTIVE` a INT2, no es necesario pero es útil para propósitos de depuración.)
- POWER_CTL <- `0x0A`

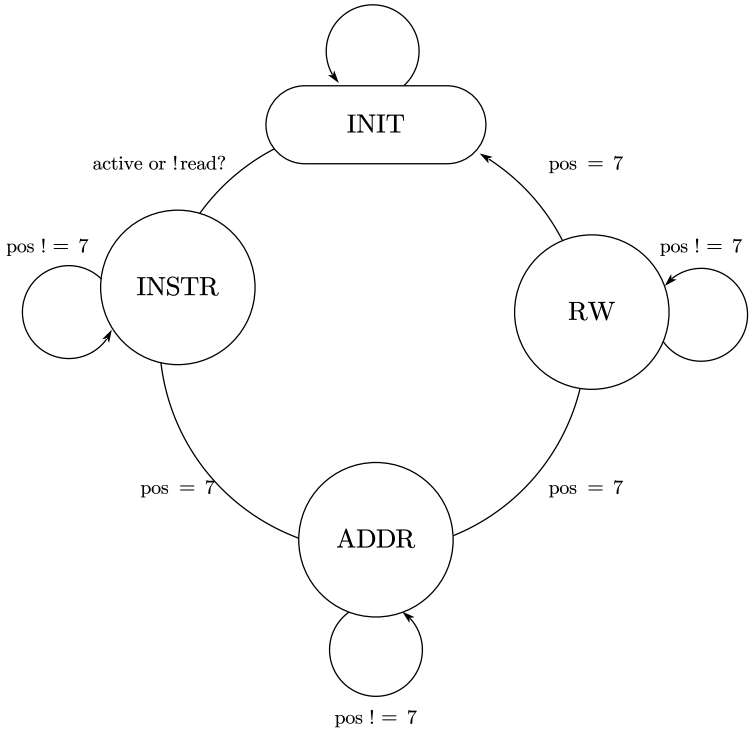

Figura 4: Diagrama de estados del Ejercicio 3

2.4. Archivo de restriccion de pines

Sólo se cambiaron los nombres de algunas señales y añadieron mas conexiones a los LEDs para efectos de depuración.

```
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
# clk
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 [get_ports {clk}];
# switches
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { resetn }];
# leds
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { out_byte[0] }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { out_byte[1] }];
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { out_byte[2] }];
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { out_byte[3] }];
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { out_byte[4] }];
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { out_byte[5] }];
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { out_byte[6] }];
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { out_byte[7] }];
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { out_byte[8] }];
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { out_byte[9] }];
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { out_byte[10] }];
set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { out_byte[11] }];
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { out_byte[12] }];
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { out_byte[13] }];
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { out_byte[14] }];
set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { out_byte[15] }];

set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVCMOS33 } [get_ports { out_byte_en }];
set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports { trap  }];

#7 segment display
# cathodes
set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { c_out[7] }];
set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { c_out[6] }];
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { c_out[5] }];
set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { c_out[4] }];
set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { c_out[3] }];
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { c_out[2] }];
set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { c_out[1] }];
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { c_out[0] }];
# anodes
set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { an_out[7] }];
set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { an_out[6] }];
set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { an_out[5] }];
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { an_out[4] }];
set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { an_out[3] }];
set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { an_out[2] }];
set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { an_out[1] }];
set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { an_out[0] }];
# accelerometer
set_property -dict { PACKAGE_PIN E15 IOSTANDARD LVCMOS33 } [get_ports { miso }];
set_property -dict { PACKAGE_PIN F14 IOSTANDARD LVCMOS33 } [get_ports { mosi }];
set_property -dict { PACKAGE_PIN F15 IOSTANDARD LVCMOS33 } [get_ports { sclk }];
set_property -dict { PACKAGE_PIN D15 IOSTANDARD LVCMOS33 } [get_ports { cs }];
set_property -dict { PACKAGE_PIN B13 IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[0] }];
set_property -dict { PACKAGE_PIN C16 IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[1] }];
```

Figura 5: Archivo de restricciones de localización de pines utilizado.

3. Resultados y análisis

3.1. Ejercicio 1

Este ejercicio tiene como fin realizar una introducción al manejo de interrupciones tanto en el core PicoRV32. Para ello se generan interrupciones por medio del test bench y se modifica el módulo para contabilizarlas. En la siguiente figura se observan las interrupciones irq, los tiempos que tardan en procesarse y la cuenta.

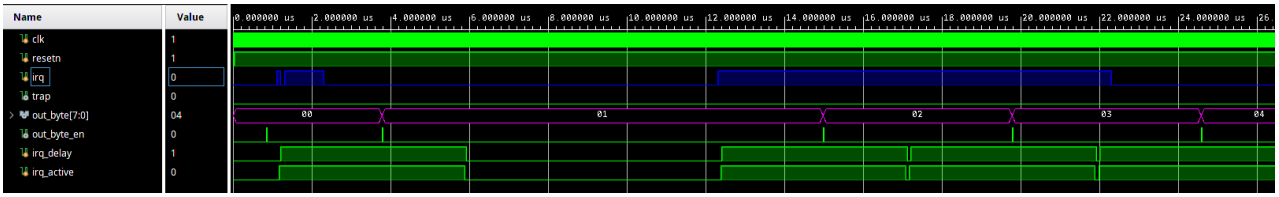

Figura 6: Resultados de activación de interrupción irq

En la figura previa se puede observar que cuando la señal irq se levanta, toma un tiempo en procesarse la interrupción, por lo que aunque la señal se active mientras que se procesa, esta es ignorada. El caos mencionado se observa en la primer interrupción levantada, pues esta se desactiva y luego se levanta de nuevo, mas solo se contabiliza una.

Además, se observa que se la señal irq se mantiene levantada por mucho tiempo, solo se van a contabilizar una nueva interrupción hasta que la primera detectada haya terminado de contabilizarse. El comportamiento presentado en ambas situaciones es el esperado, pues las interrupciones detienen el flujo normal del código hasta que la misma haya sido procesada por completo, una vez termina puede tomar nuevas interrupciones.

3.2. Ejercicio 2

Para este ejercicio se implementó un módulo lector del acelerómetro. Este se encuentra en constante lectura de los registros `YDATA_L`, `YDATA_H`, `ZDATA_L`, `ZDATA_H` del acelerómetro ADXL362 del Nexys 4 DDR. Estos registros contienen los datos de aceleración para el eje Y y Z con extensión de signo que mide el módulo.

Se creó un acelerómetro *dummy* para utilizarlo de prueba en el testbench que recibe las señales SPI del lector y responderá alternando entre las siguientes valores de los ejes:

- Eje Y: `0xDEAD`, `0xBEEF`
- Eje Z: `0xCAFE`, `0xBABE`

Conectando este acelerómetro al lector diseñado se obtienen las formas de onda de la Figura 7.

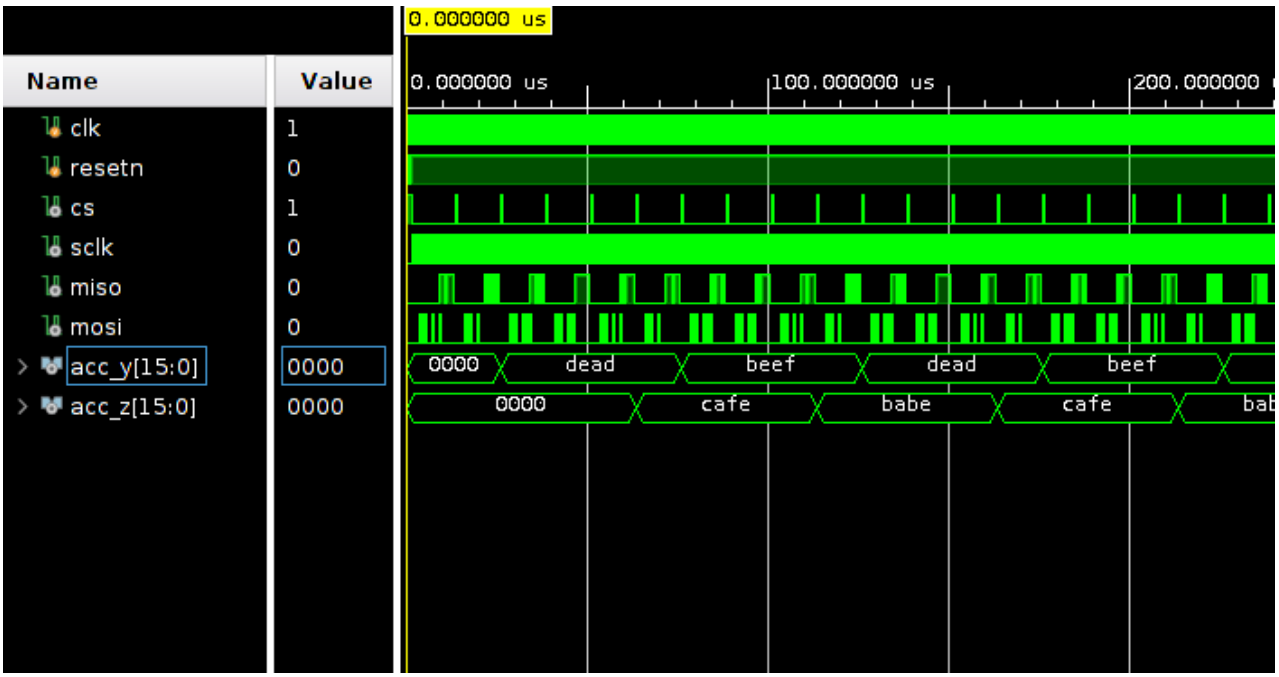

Figura 7: Simulación del acelerómetro

Las transacciones seriales que transportan las señales `miso` y `mosi`, donde `mosi` sale del lector (Master) y entra al accelerómetro de prueba (Slave) y `miso` sale del accelerómetro de prueba (Slave) y entra al lector (Master), se visualizan más detalladamente en la Figura 9. Decodificando cada transmisión binaria se observa que el comportamiento experimental es idéntico al descrito por la hoja de datos del acelerómetro ADXL362 de la Figura 8.

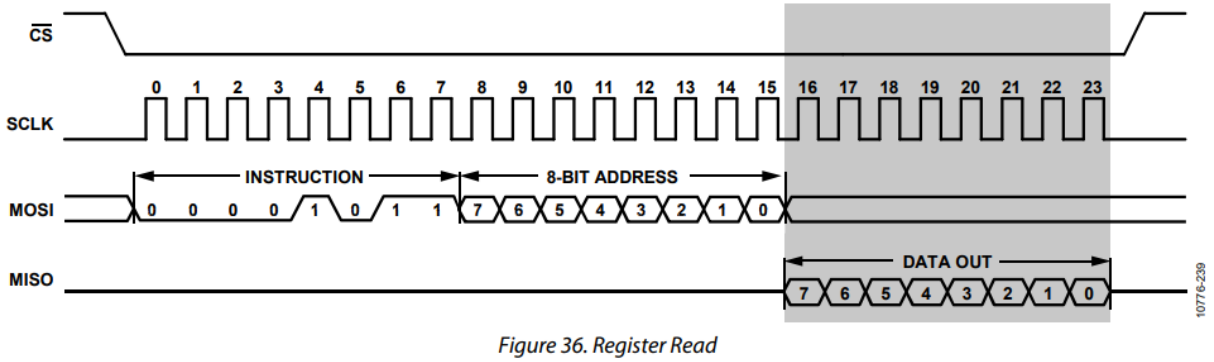

Figura 8: Estructura de comandos del acelerómetro ADXL362 para la lectura de registros. Tomado de https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf

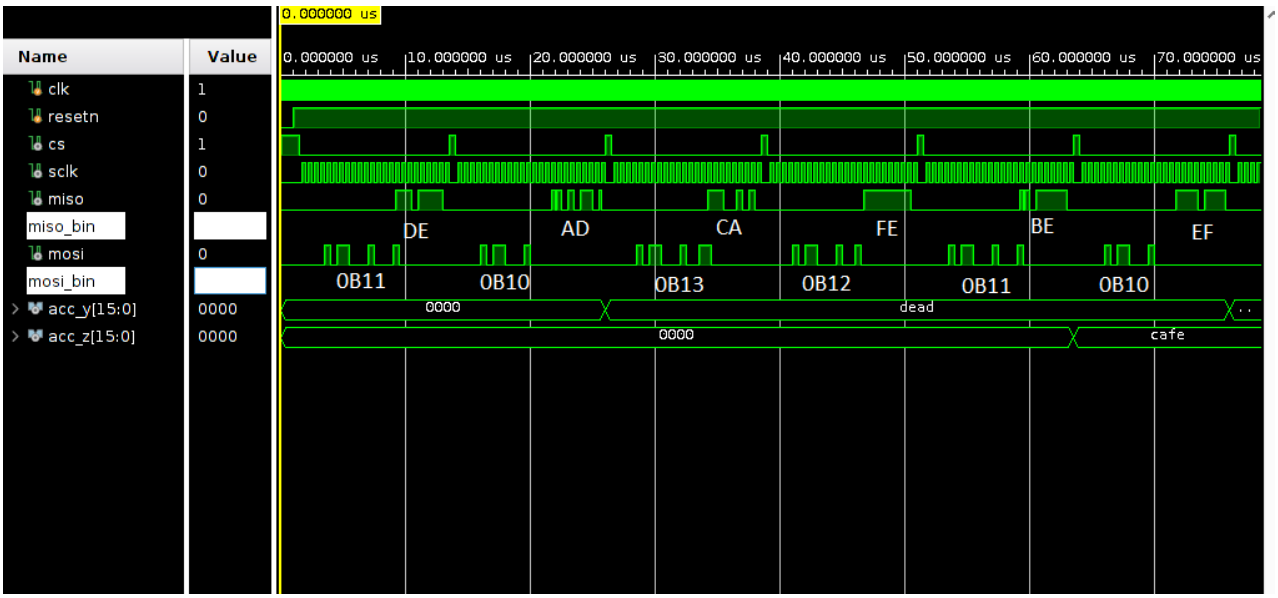

Figura 9: Decodificación de las señales `miso` y `mosi`.

Teniendo demostrado la funcionalidad experimental con una simulación por comportamiento, se pasó a sintetizar e implementar con Vivado en la tarjeta Nexys 4 DDR cuyo comportamiento se observó y analizó en el siguiente video:

https://youtu.be/VP7rNyrh7w0

3.3. Ejercicio 3

En este ejercicio se modificó el lector del acelerómetro para que:

- Configure inicialmente el acelerómetro para que detecta actividad dado un *threshold* específico, cuando detecte actividad levanta una interrupción.
- Siga leyendo (enviando señales SPI) el acelerómetro sólo si este se encuentre activo.

Con el mismo *dummy* accelerometer usado de prueba en el ejercicio anterior se procede a simular con el nuevo lector de acelerómetro. Los resultados de la simulación se observan en la Figura 10. En donde se observa el comportamiento esperado por parte del lector, sólo cuando se encuentra activo el acelerómetro (simulado con la señal `ACL_INT`) enviará señales SPI a este y leerá de sus registros.

Además, en la Figura 11 se observa la configuración inicial del acelerómetro, compuestos por 9 transacciones SPI (más detalles de la configuración referirse al apartado de las correciones del ejercicio). Luego de estas 9 transacciones el lector procede a la secuencia principal de lectura.

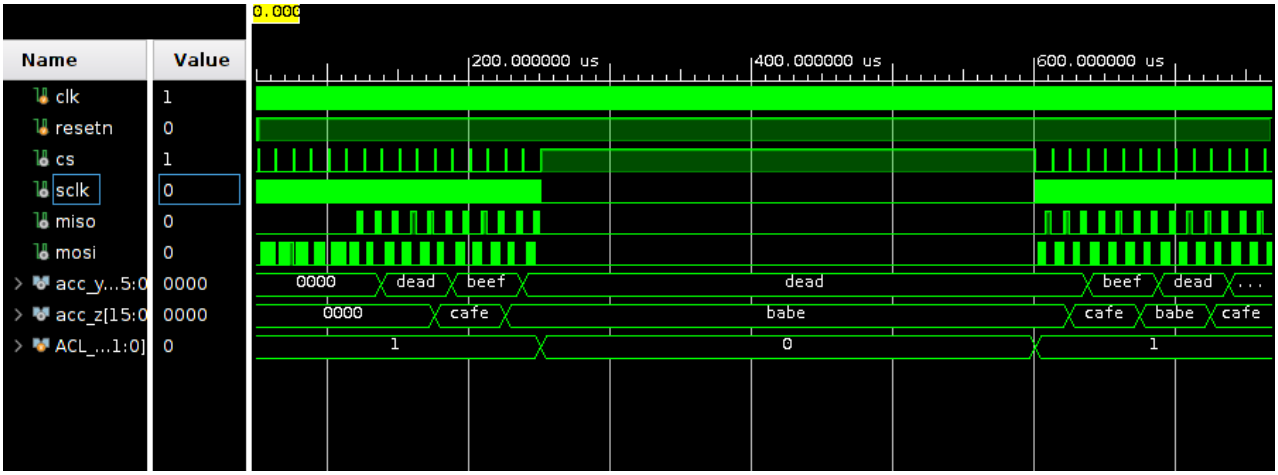

Figura 10: Simulación del acelerómetro con interrupciones

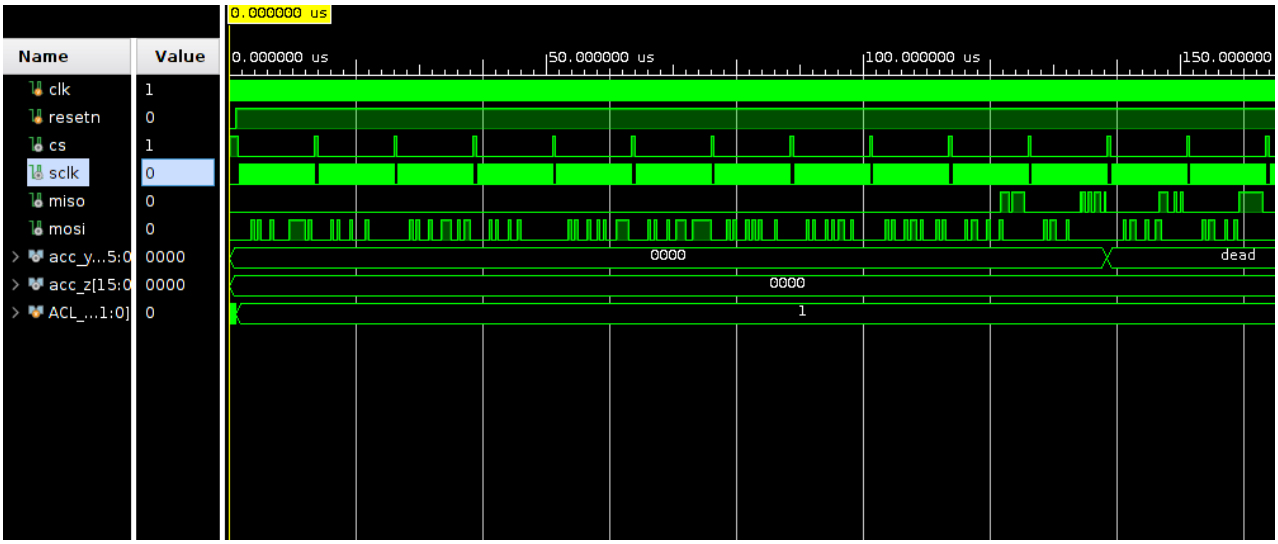

Figura 11: Configuración inicial del acelerómetro

Teniendo demostrado la funcionalidad experimental con una simulación por comportamiento, se pasó a sintetizar e implementar con Vivado en la tarjeta Nexys 4 DDR cuyo comportamiento se observó y analizó en el siguiente video:

https://youtu.be/uAdoe27X0u4

4. Repositorio

El código necesario para recrear los resultados de este laboratorio se pueden encontrar en GitHub utilizando el siguiente enlace:

https://github.com/ucr-ie-0424/laboratorio-4-jimenez-sanchez

El ID del último commit es: de7eca1602ce9a70ad84f46bfc44b7bb53e67d81

5. Conclusiones y recomendaciones

5.1. Conclusiones

- En el ejercicio 1 se pudo observar el manejo de las interrupciones IRQ de manera exitosa, además de que se logró ver las señales que muestran los retrasos que ocurren por el procesamiento de señales. En este caso, cuando ocurre una interrupción, se debe esperar un tiempo a que el core realice las operaciones que debe hacer con dicha interrupción, durante este periodo solo hace esto. Una vez culmina, puede leer más interrupciones.

- En el ejercicio 2 se pudo obtener una comunicación exitosa entre el core PicoRV32 y el módulo del acelerómetro del Nexys 4 DDR, el ADXL362. Mediante un módulo lector que pasa leyendo los registros de datos de los ejes Y y Z con el protocolo de comunicación serial SPI. Se pudo desplegar este valor el display de 7 segmentos de la tarjeta.

- En el ejercicio 3 se pudo desplegar exitosamente los valores de los registros de los ejes Y y Z del acelerómetro ADXL362 del Nexys 4 DDR, sólo cuando este haya detectado actividad, es decir movimiento de la tarjeta. Utilizando las interrupciones que envía el módulo hacia el core y luego de que este lo haya procesado, tanto el módulo lector como desde el firmware se pausa la lectura.

5.1.1. Recomendaciones

- Antes de generar el bitstream y enviarlo a la tarjeta Nexys 4 DDR, se recomienda simular de forma exhaustiva. La pérdida de tiempo, en general, que se toma sintetizando e implementando es mucho mayor en total de la que se lleva confeccionar un acelerómetro de prueba y probarlo exhaustivamente.

- Se recomienda verificar bien la configuración física del Nexys 4 DDR, esta tarjeta tiene unos *jumpers* que aveces limitan o bien cortan completamente la comunicación SPI.

- Se debe tener claro que el manejo de interrupciones toma cierto tiempo y durante este periodo, no se ejecuta el programa principal.