Proyecto Final: CPU Uniciclo – Crono Audio

1 Decisiones de diseño tomadas (instrucciones)

En este apartado vamos a comentar las decisiones que como grupo hemos tomado a la hora de diseñar nuestra unidad de control.

En primer lugar, debemos destacar que partimos del diseño proporcionado por el profesor procedente de la actividad anterior. A partir del diseño original hemos realizado una serie de modificaciones con el objetivo de conseguir los nuevos requisitos que hemos implementado posteriormente.

Una de las decisiones más importantes que tomamos a la hora de diseñar fue la codificación de las instrucciones, ya que tuvimos que modificarlas porque el número de instrucciones aumentó y por lo tanto, tuvimos que transformar el número de bits del opcode. Asimismo, respecto a las instrucciones diferenciamos entre pertenecientes a la ALU y no pertenecientes. Entre dichas diferencias podemos destacar que si encontramos un cero en el cuarto bit empezando por los menos significativos estaremos ante una instrucción de ALU, mientras que si lo que encontramos es un uno no pertenecerán a esta. Nuestras instrucciones constan de 16 bits, todo esto se puede encontrar especificado tanto en el fichero **uc.v** de nuestro código, así como en el pdf que incluimos en el envío llamado **CodificacionInstruciones.pdf.**

Muchas de estas decisiones fueron tomadas con respecto a la E/S y la lectura de las nuevas instrucciones (IN,OUT, OUT_INM, salto relativo, salto a subrutina y vuelta de subrutina). Respecto a la entrada y salida hablaremos en el siguiente apartado y con ello, de las decisiones tomadas.

2 E/S y decisiones tomadas al respecto

A continuación, vamos a hablar de la implementación de la E/S, así como de todas y cada una de las decisiones y cambios que tuvimos que llevar a cabo para conseguir que funcionara.

Con respecto a la entrada acordamos implementar un multiplexor 4 a 1 (MUXENTRADA) donde cada entrada externa consta de 8 bits al igual que la salida. Además, este multiplexor tiene una señal de control para seleccionar cuál de las entradas existentes será la que activamos. Esta señal será la salida de la memoria de instrucciones (DATOS) que corresponderá con el conjunto de bits 6 y 7 que en nuestras instrucciones nos indican el puerto que cogeremos. La salida de estas entradas (salEntrada) irán a un multiplexor 2 a 1 (MUX2), el cual a través de la señal de control s_in selecciona

entre la entrada externa o la salida de la ALU que posteriormente introduciremos en el banco de registro.

En lo referido a la salida implementamos un decodificador que consta 2 entradas que proporcionan el número de puerto codificado en la instrucción y que elegirán cuál de los 4 registros de salida se activarán, y gracias a las señales s_out y out_in elegiremos si se cargarán en los puertos de salida tanto un inmediato contenido en la instrucción o el valor que contenga un registro de nuestro banco de registros.

Una vez decidimos la manera en la que realizar e implementar los módulos de entrada y salida tuvimos que cambiar algunos aspectos de nuestro módulo microc. Algunas de las modificaciones que hemos tenido que realizar son las siguientes. Para controlar tanto los saltos a subrutina, como el relativo y la vuelta de subrutina en la parte donde encontramos el PC tuvimos que añadir dos multiplexores 2 a 1 . El primero de ellos (MUX3) tiene como entradas la salida de la memoria de instrucciones (DATOS[15:6]) o la instrucción siguiente a la llamada a subrutina, calculada a través de un sumador que suma 1 a la actual instrucción que ejecuta el salto a la subrutina y la almacena en un registro que también hemos implementado y se activa a través de la señal s_subrutina.

Para la vuelta de la subrutina, utilizaremos un multiplexor que escogerá como entrada al PC bien la instrucción almacenada en el registro ra o la siguiente instrucción del PC, todo ello controlado por la señal s_ra.

En cuanto al salto relativo, se han de tener en cuenta varias consideraciones, entre las que debemos destacar sí el salto se realiza hacia delante o hacia atrás, para ello se ha elaborado un sumador/restador que o bien suma o resta un inmediato provisto en la instrucción al PC diferenciando por el bit más significativo (DATOS[15]) el tipo de operación a realizar (0 suma, 1 resta). Para elegir si se desea realizar un salto relativo, se ha establecido una señal s_rel que activará el multiplexor indicado (MUX5) y se cargará en el PC la siguiente instrucción a ejecutar o la instrucción calculada en el salto relativo.

3 PARTES OPCIONALES

Para el apartado de esta actividad hemos implementado un ensamblador para codificar nuestro código “MIPS”, además de la codificación 7 segmentos para poder visualizar las pruebas correctamente, en un led 7 segmentos con punto incorporado.

En cuanto al ensamblador decidimos realizarlo en código C++, lo que hace que este programa reciba un fichero con un código parecido a MIPS. Posteriormente, traducimos de este fichero tanto los mnemónicos de las instrucciones, registros, etiquetas e inmediatos para obtener como salida otro fichero con un código donde las instrucciones se muestran en 16 bits de 0 y 1.

Un ejemplo de traducción podría ser la siguiente:

LI $1, 0 → 0001_0000_0000_1000

ADD $10, $1, $2 → 1010_0010_0001_0010

NOTA:Como se ha dicho anteriormente, las codificaciones de las instrucciones están especificadas en **CodificacionInstrucciones.pdf**

En cuanto al código 7 segmentos lo que hemos realizado es lo siguiente. Para la codificación de la salida en binario puro, hemos tenido que llevar a cabo una serie de módulos que se encargarán de transformar el binario puro de la salida a codificación BCD. Para conseguirlo, hemos empleado dos módulos (add3 y BinaryToBCD) que generarán tres salidas a partir de un binario de 8 bits. Esas salidas (HUNDREDS, TENS, ONES) corresponderán con cada posible dígito que se podrá codificar con 8 bits (255) para la codificación a BCD. Para ello, se ha utilizado el exceso 3. Una vez obtenidos en binarios de 4 bits los dígitos del número de nuestra salida procedimos a la creación de módulos que codifican a 7 segmentos el código bcd y con todo ello se genera un módulo que tiene como entrada el binario puro de 8 bits y se obtiene como salida la codificación en 7 segmentos de cada uno de los 3 dígitos del binario.

4 Implementación en Quartus II

Para el testeo del software en un dispositivo hardware físico, hemos utilizado la fpga Altera Cyclone II EP2C20F484C7 para ello hemos utilizado diferentes herramientas proporcionadas por la herramienta Quartus II como puede ser el ModelSim (entorno de simulación) o el Pin Planner para la asignación de nuestras entradas o salidas a los pines de la placa.

En nuestro caso, el proyecto que hemos desarrollado, está basado en la salida de audio, para ello hemos utilizado módulos previamente definidos por otros proyectos para el controlador del audio, para ello hemos utilizado el módulo **VGA_Audio_PLL**, **I2C_AV_Config** y el **AUDIO_DAC** o **audio_codec**. Además de nuevos módulos que hemos diseñado para llevar a cabo nuestra tarea, módulos que nos ayudarán a emitir una señal (para descontar el acumulador) aproximadamente cada segundo o módulos que permitirán generar sonidos diferentes según avance el conteo.

Todo ello, se especificará a continuación:

a. Asignación de PINES

Para llevar a cabo la asignación de pines, fue necesario conocer el nombre de los pines y el dispositivo físico en el que se aplica, en nuestro caso, hemos hecho uso tanto de los switches, los keys y los displays 7 segmentos, además del uso del oscilador de 50 MHz del que dispone la fpga para asignarlo como entrada a la salida del reloj.

Cabe destacar que para el funcionamiento de los keys asignados en nuestro programa al botón para la suma, fue necesario negar la entrada para el correcto funcionamiento ya que cuando se pulsa éste tipo de pulsador, el circuito se cierra con un valor de 0 y en nuestro caso, el valor deberá de ser 1.

La asignación de pines estará un poco mejor detallada en el fichero **pines.txt** que se encuentra dentro del proyecto.

b. Módulos a implementar

Para la nueva implementación, únicamente hemos implementado tres nuevos módulos para nuestro proyecto, estos módulos son: **tiempos, sound** y **sound_continuado.**

- **tiempos:** Este módulo se encarga de generar una señal cada vez que transcurra un segundo respecto al reloj de 50MHz del que está provisto la fpga, para ello se utiliza un contador y una señal de activación que corresponderá con el switch que está asignado para el inicio del contador.
- **sound:** El módulo se encarga de generar el rate para el módulo que se encarga de transformar la conversión de digital a analógico (audio_codec) en base al número por el que vaya el conteo, generando sonido desde más graves a más agudos.
- **sound_continuado:** El módulo se encarga de activar una señal que pondrá siempre en 1 la señal que activa el módulo de sonido para producir al final de la cuenta, cuando la salida del contador se encuentre en 0, un sonido prolongado.

NOTA: Todos los códigos del proyecto se pueden ver bien en el directorio donde se encuentra el proyecto Quartus o en el directorio Código de la memoria.

5 Pseudocódigo del programa

```
acumulador <- 0

iniciocrono <- false

suma <- false


func sumar(){
```

```
                acumulador += 5;
}

func restauno(){
                acumulador = acumulador - 1;
}

while(!iniciocrono){
                if(suma){
                        sumar();
                }
                elif(iniciocrono){
                        return;
                }
}

while(acumulador != 0){
                //Cada Segundo
                if(pasa_1_segundo){
                        restauno();
                }
}
```

6 Dificultades

En el apartado de dificultades, debemos mencionar el uso de módulos externos para interactuar con la salida de audio, que al comienzo se nos hizo un poco extraño el código pero que con la búsqueda de información en proyectos más o menos similares, nos ayudó a llevar a cabo las conexiones necesarias y a asignar los valores indicados para generar el sonido que deseábamos en la salida.

También destacar que en el desarrollo del programa, se ha utilizado la herramienta ModelSim para su depuración, permitiendo la corrección de algunos fallos en la primera parte del desarrollo del programa, sobre todo fallos relacionados con los pulsos que se deben generar en cada segundo.

7 Video Demostración

https://www.youtube.com/watch?v=vMPMKnrXnk8