CENTRO REGIONAL UNIVERSITARIO CÓRDOBA
IUA

ARQUITECTURA DE COMPUTADORAS I

PROFESOR

Ing. Luis E. Toledo

INTEGRANTES

Gazagne, Alejo (agazagne026@alumnos.iua.edu.ar)

Llamosas, Simón (sllamosas806@alumnos.iua.edu.ar)

14 de noviembre del año 2022

Índice General

Introducción

El presente trabajo fue llevado a cabo mediante el uso de la placa FPGA DE0_Nano, implementando la herramienta Quartus, y los conocimientos adquiridos en la materia Arquitectura De Computadores I, utilizando assembler como lenguaje de desarrollo.

Objetivos

a) Realice un programa a fin de que el usuario pueda seleccionar desde un menú, una de cuatro secuencias de luces posibles. Dos de ellas serán comunes para todos los proyectos y son: “El auto fantástico” y “El choque”. Las otras dos serán propias de cada grupo y se deberán implementar una de ellas con algoritmo y la restante por medio de la técnica de tablas de datos.

b) Cada vez que el usuario seleccione una secuencia el programa deberá dar algún indicativo para señalar cuál secuencial está ejecutándose. Al optar por abandonar la actual, el programa deberá regresar al menú principal inmediatamente sin completar la secuencia que se está desarrollando y apagando todas las luces.

c) Permite la posibilidad de controlar la velocidad de cada secuencia. Introduzca la comprobación de las teclas oprimidas en el lugar apropiado de su programa a fin de percibir la reacción del sistema en forma inmediata, independiente de la velocidad actual. La velocidad ajustada en cada secuencia deberá conservarse entre llamadas a diferentes secuencias.

Especificaciones Técnicas

Para la comprensión del informe vemos necesario explicar los componentes de la placa con los que vamos a trabajar y hablar durante el desarrollo del mismo. La placa utilizada es una FPGA DE0_Nano:

En la FPGA se programó un software desarrollado en SystemVerilog, versión reducida del microprocesador ARM (32-bit), que cuenta con menos funcionalidades.

La placa cuenta con varios componentes, pero nosotros solamente utilizamos los pulsadores, switches, y leds.

A cada uno de ellos le asignamos funcionalidades y puertos para su implementación:

OUTPUTS:

Las salidas que presenta nuestro programa son los 8 leds pertenecientes a la placa, adoptando el estándar **L**east **S**ignificant **B**it (LSB) para el puerto de salida del programa (8 bits).

INPUTS:

Las entradas que utiliza nuestro programa son 6, 2 pulsadores y 4 switches.

- pulsador-KEY[1]: este se programó para aceptar la selección de las secuencias.
- pulsador-KEY[0]: este está asignado a la operación de reset.
- DIP Switch[0]: selección secuencia 1
- DIP Switch[1]: selección secuencia 2

- DIP Switch[2]: selección secuencia 3
- DIP Switch[3]: selección secuencia 4
- Clock R8: clock de 5o MHz propio de la placa

La versión del ARM reducido utilizada tiene un INport de 8 bits, por lo tanto nos quedan 3 bits sin utilizar. Tomamos los primeros 4 bits menos significativos para los switches y el 5to para la operación de aceptar. El pulsador restante está asignado al "reset" que presenta la versión del ARM.

El FPGA utilizado implementa lógica negativa, por lo tanto los pulsadores son activos por bajos y las entradas sin asignaciones son igual a 1.

A continuación se pueden ver las asignaciones de los puertos en Quartus para cada componente de la placa:

Node Name	Direction	Location
INport[7]	Input	
INport[6]	Input	
INport[5]	Input	
INport[4]	Input	PIN_E1
INport[3]	Input	PIN_M15
INport[2]	Input	PIN_B9
INport[1]	Input	PIN_T8
INport[0]	Input	PIN_M1
OUTport[7]	Output	PIN_L3
OUTport[6]	Output	PIN_B1
OUTport[5]	Output	PIN_F3
OUTport[4]	Output	PIN_D1
OUTport[3]	Output	PIN_A11
OUTport[2]	Output	PIN_B13
OUTport[1]	Output	PIN_A13
OUTport[0]	Output	PIN_A15
clk	Input	PIN_R8
resetE	Input	PIN_J15

Nota: Imagen tomada desde la pestaña pin planner, dentro de Quartus, con la asignación de los pines utilizados en la placa

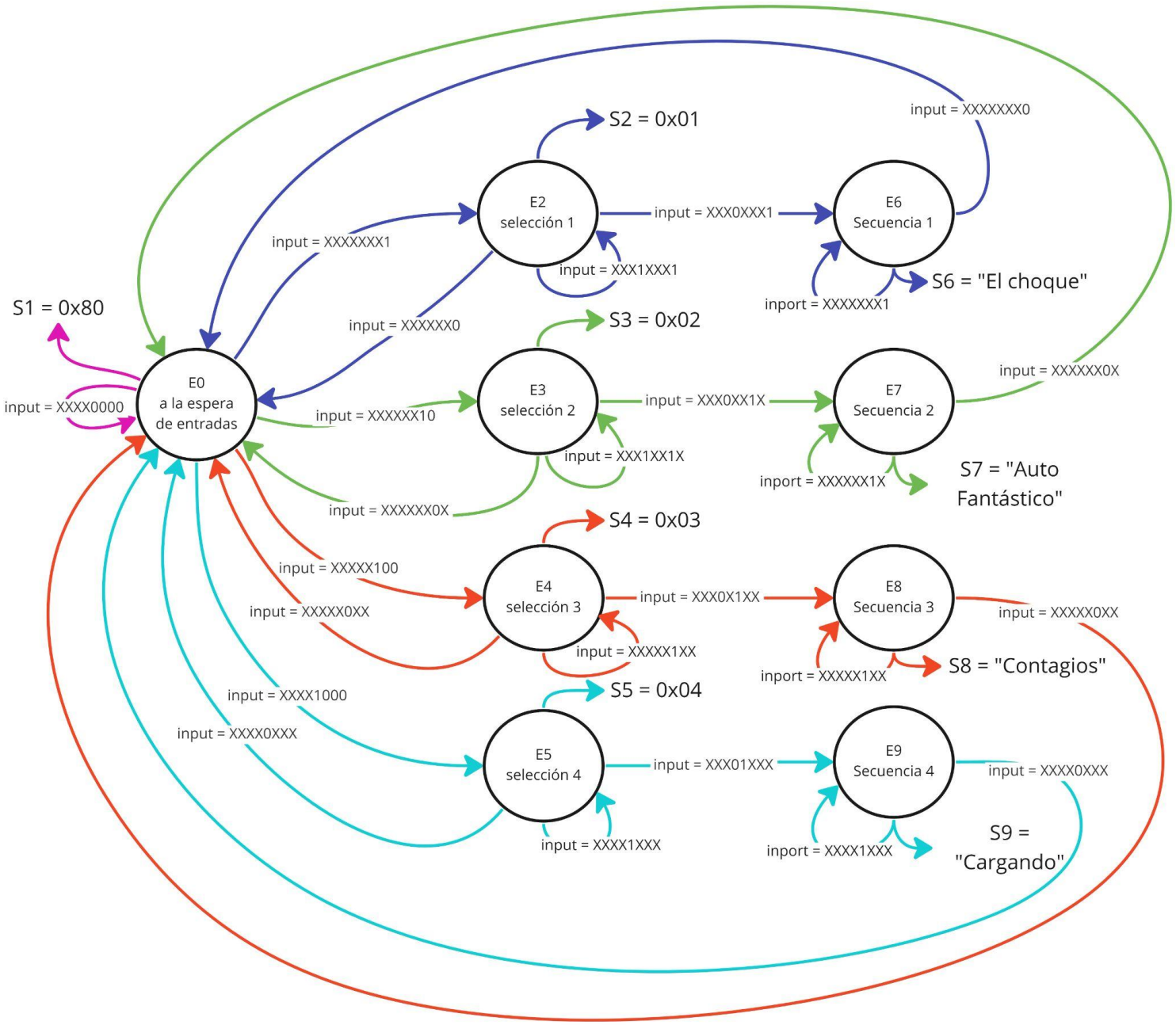
Desarrollo

Uno de nuestros objetivos era realizar un menú interactivo con el que podamos seleccionar alguna de las 4 secuencias desarrolladas. Este fue pensado y desarrollado de la forma que podamos hacerlo mediante los switches propios de la placa. El primer switch (DIP Switch[0]) es para la primer secuencia;el siguiente(DIP Switch[1]) es para la secuencia 2 y así con los 2 restantes. Cada vez que accionamos un switch este activará un led para que podamos ver la selección, y esperará a que apretemos el botón “aceptar” secuencia, si nosotros queremos cancelar la selección podremos desplazar nuevamente,a su posición inicial, el switch para anular la selección. Pero si estamos de acuerdo con la opción podremos apretar el botón que está asignado para la aceptación de las secuencias y se iniciará la secuencia selecta. Una vez iniciada la secuencia podremos salir de ella llevando al switch activo al estado de reposo.

Como fue planteado en los objetivos, se realizaron 4 secuencias, dos de ellas fueron planteadas por el profesor, estas fueron “Auto fantástico” y “Choques”. Ahora bien, los códigos de estas fueron creados por nosotros. Una condición fue que, “Auto fantástico” fuese desarrollada solamente por algoritmo y “Choques” implementada mediante el uso de tablas leídas de un archivo. Las otras dos secuencias fueron pensadas por los integrantes del grupo. Estas fueron “Cargando” que fue implementada por algoritmo y “Contagios” desarrollada con el uso de tablas. Cargando se trata de una secuencia que simula una “barra de carga” acumulativa; Contagios consiste de encender uno de los leds del medio, donde los adyacentes va a ir copiando el estado de activo (encendido), cuando todos se encuentren prendidos, se apagaran cómo se encendieron.

Los algoritmos, tanto de lectura de tabla como de las secuencias, se implementan en el archivo “imemfile.dat”,donde se guardan las instrucciones (o palabra de control) de 32-bits; y los valores,que corresponden a las secuencias por tabla, serán leídos desde el archivo “dmemfile.dat”, ambos archivos se encuentran en hexadecimal.

Diagrama de estados



Menú inicial

A continuación se muestra la implementación del estado inicial del programa:

```
        SUB R0, R15, R15
INICIO: ADD R2, R0, #0x80
        STR R2, [R0, #0x800]
INICIAL:ADD R3, R0, #2
INPUT:  SUBS R3, R3, #1
        LDR R1, [R0, #0x800]
        BNE INPUT
        ANDS R2, R1, #0x0F
        BEQ INICIAL
        ANDSNE R2, R1, #0x01
        STRNE R0, [R0, #0x800]
        BNE SELECCION1
        ANDSEQ R2, R1, #0x02
        STRNE R0, [R0, #0x800]
        BNE SELECCION2
        ANDSEQ R2, R1, #0x04
        STRNE R0, [R0, #0x800]
        BNE SELECCION3
        ANDSEQ R2, R1, #0x08
        STRNE R0, [R0, #0x800]
        BNE SELECCION4
        B INICIO
```

En esta etapa el programa enciende el led más significativo para que tengamos feed back de que la placa está encendida y que el programa se cargó correctamente, mientras que no se haya seleccionado ninguna de las secuencias disponibles, el programa se mantendrá en espera. En el momento que cambie de estado alguno de los switches se prenderá el respectivo led de la selección y hará el salto al estado de "aceptación" de la opción seleccionada.

Menú De Aceptación

A continuación se muestra la implementación del estado de aceptación :

```
        AND R2, R1, #condicionLuz
LOOP1: STR R2, [R0, #0x800]
```

```
        ADD R3, R0, #2
        SUBS R3, R3, #1
INPUT:  LDR R1, [R0, #0x800]
        BNE INPUT
        ANDS R2,R1, #condicionSecuencia
        BEQ SECUENCIA
        ANDSNE R2, R1, #condicionSeguir
        BEQ Menu inicial
        BNE LOOP1
```

Este estado es autónomo para cada una de las secuencias, por lo tanto las etiquetas son independientes y propias para cada una de las opciones. Nos da la posibilidad de cancelar la selección, si no es la que se desea, o en caso contrario aceptarla.

Explicación de etiquetas:

- condicionLuz: se encarga de limpiar la entrada y dejar solo el bit que nos interesa para prender el led correspondiente a la secuencia.Por ejemplo, si seleccionamos la opción 1, condición = 0x1, los primeros 7 bits no nos interesan y podemos encender la luz correspondiente a la selección, en este caso es la menos significativa.
- condicionSecuencia: verifica si realmente se quiere acceder a la secuencia, chequeando si se acepta la opcion.Es común para todas las secuencias(0x10).
- condicionSeguir: nos da la opción de volver al menú inicial, chequeando si el bit correspondiente a la selección es puesto en cero.

Secuencias

A continuación se muestra la implementación para leer datos del archivo dmem:

```
        LDR r5, [r0,#delay]
        ADD r7, r0, #punteroInicial
        ADD r8,r0,#cantPalabras
INICIO: LDR r9, [r7]
        STR r9, [r0, #0x800]
        ADD r6,r0,r5
RETARDO: SUBS r6, r6, #1
        BNE RETARDO
```

```
        ADD r7, r7, #4
        SUBS r8,r8,#1
        ADDEQ r7,r0,#punteroInicial
        ADDEQ r8,r8,#cantPalabras
        ADD R3,R0,#0x2
INPORT: SUBS R3,R3,#1
        LDR R1,[R0,#0x800]
        BNE INPORT
        ANDS R1,R1,#condicionSwitch
        BEQ FIN
        BNE INICIO
```

Este algoritmo es el que se encarga de hacer la lectura del archivo “dmemfile.dat” y sacar los valores de la tabla por los outputs .Primero se hace lectura del valor de tabla que vamos a utilizar como retardo para mantener las luces prendidas o apagadas el tiempo que nosotros indiquemos. Después empieza a leer el archivo números de 8 bits, estos son cargados en el puertos de salida “outputs” donde están asignados los leds, después se incrementa el valor de la palabra para que en la siguiente iteración lea la palabra que sigue a continuación y así sucesivamente, hasta que se recorra toda la tabla de valores, por lo tanto,el algoritmo debe comenzar nuevamente.

Explicación de etiquetas:

- delay:puntero a la tabla donde se encuentran la cantidad de iteraciones del bucle.
- punteroInicial: puntero que indica donde arranca la lectura de datos del dmem.
- cantPalabras: cantidad de señales a sacar del dmem, destinadas a las salidas de las luces.
- codicionSwitch: depende en qué secuencia nos encontremos, si estamos en Choques es igual a 0x01; si estamos en Contagios es igual 0x02

A continuación se muestra el algoritmo de la secuencia “Auto Fantastico” :

```
INICIO: ADD R4, R0, #7
        ADD R8, R0, #0
        ADD R6, R0, #0x80
LOOP1:  STR R6, [R0, #0x800]
        ADD R5, R0, #6
```

```
        ADD R7, R0, #1
        SUBS R5, R5, R8
        BEQ SALTO
DIVISION:SUBS R5, R5, #1
        ADD R7, R7, R7
        BNE DIVISION
SALTO:  ADD R8, R8, #1
        SUB R6, R6, R7
        LDR R9, [R0, #0x4]
DELAY1: SUBS R9, R9, #1
        LDR R1,[R0,#0x800]
        ANDS R1,R1,#0x01
        BEQ FIN
        BNE DELAY1
        SUBS R4, R4, #1
        BNE LOOP1
        STR R6, [R0, 0x800]
        ADD R4, R0, #7
LOOP2:  ADD R6, R6, R6
        LDR R9, [R0, #0x4]
DELAY2: SUBS R9, R9, #1
        LDR R1,[R0,#0x800]
        ANDS R1,R1,#0x02
        BEQ FIN
        BNE DELAY2
        STR R6, [R0, 0x800]
        SUBS R4, R4, #1
        BNE LOOP2
        BNE INICIO
```

Este algoritmo da salida de la secuencia de luces “Auto Fantastico”, el cual consiste en arrancar un valor inicial potencia de 2 (128), en base 10, e irle restando las potencias anteriores al número con el que se inició para obtener el siguiente valor que corresponde al led que le da continuación a la secuencia, hasta converger en 1. La vuelta de la secuencia, se logró por medio de la suma acumulada del valor 1, hasta converger en 128.

A continuación se muestra el algoritmo de la secuencia “Cargando” :

```
INICIO: SUB R0,R15,R15
```

```
        ADD R9,R0,#1
        ADD R2,R0,#2
        ADD R10,R0,#8
        ADD R5,R0,#0
        ADD R6,R0,#1
        ADD R7,R0,#2
        ADD R8,R0,#7
LOOP 2: ADD R9,R0,R6
        ADD R2,R0,R7
        ADD R6,R9,R2
        ADD R7,R2,R2
LOOP 1: STR R9,[R0,0x800]
        ADD R9,R9,R2
        ADD R2,R2,R2
        LDR R4,[R0,#0x4]
RETARDO 1: SUBS R4,R4,#1
        LDR R1,[R0,#0x800]
        ANDS R1,R1,#0x08
        BEQ FIN
        BNE RETARDO 1
        ADD R5,R5,#1
        SUBS R10,R10,#1
        BNE LOOP 1
        SUB R10,R5,#1
        ADD R5,R0,#0
        SUBS R8,R8,#1
        BNE LOOP 2
        B INICIO
```

Este algoritmo da la sida a la secuencia de luces “Cargando”, este comienza con un valor inicial 1, en base decimal, al que se le va a ir sumando potencias de dos arrancando del mismo, hasta converger en 256, el cual corresponde al valor de todos los leds encendidos, una vez que pasa esto, se vuelve a comenzar la secuencia pero esta vez del primer número obtenido a partir de la suma del valor inicial y la primer potencia de 2 (2+1), y se le aplica el mismo procedimiento de sumatorias con múltiplos de 2 hasta converger nuevamente en 255. Esto ocurrirá hasta que el valor inicial sea igual al último, es decir valor inicial igual a 256, esto indica que la secuencia ya ha sido completada y debe volver a comenzar.

Program Files:

PROGRAMA ARM-32 BITS REDUCIDA
CÓDIGO FUENTE
imemFile
dmemFile
especificaciones dmemFile

Observaciones

Hay algunas observaciones que podemos resaltar que nos dimos cuenta durante el desarrollo del proyecto, una de estas fue que los botones y los switches son activos por bajo, esto significa que mientras que los botones no sean accionados la señal de entrada de estos es 1. Por esto último, tuvimos que hacer una modificación de la entrada “reset” ,en el código del ARM reducida, negando la misma para adaptarla a un activo por alto, que es lo que más cómodo nos quedaba a la hora de interactuar con la placa.

A la hora de realizar los delay para que las luces permanezcan encendidas tuvimos un problema, este era que cuando queríamos guardar un valor demasiado grande, con la intención de utilizarlo como contador, el valor máximo que se guarda en los registro es de 8 bits ya que la ALU de la versión reducida solo puede generar numeros de 8 bits, la solución a esto fue guardar los valores que utilizaremos como contador en el archivo “dmemfile.dat”, y luego en el programa guardarlos en un registro a medida que se vayan necesitando.

Un punto adicional con el que nos surgieron algunos inconvenientes fue con respecto al INport de 8 bits, al cual se le debe aplicar la operación de doble lectura para obtener el valor correspondiente a la entrada, y así continuar con la ejecución del programa.

Bibliografía

- Harris, S. L., & Harris, D. (2015). *Digital design and computer architecture*. Morgan Kaufmann.
- Terasic DE0-Nano User Manual