Laboratorio de Circuitos Digitales 15 de Mayo del 2022
IE-0424 Jordi Louzao von Breymann - B53993
Laboratorio #2:Bitácora Dualok Fonseca - B42629

1. Introducción

En el presente laboratorio se realizo un acercamiento mas detallado al funcionamiento del hardware de la tarjeta Nexys4 para realizar operaciones aritméticas. Para comprender los resultados es necesario haber repasado conceptos sobre síntesis lógica, RTL(Verilog), FPGA, LUTs. Para implementar correctamente cada ejercicio se modificó el archivo RTL system.v para agregar módulos con diferentes algoritmos de multiplicación digital y se modificó el firmware varias veces con el fin de escribir los factores de las multiplicaciones y leer datos de memoria donde se guardaron los productos.

2. Ejercicios

2.1. Ejercicio 1

Para implementar el calculo de factoriales simplemente se implemento una función recursiva en el firmware llamada *factorial*. Al llamar esta función con el parámetro siendo el numero al cual se le calculara su factorial, esta función la retorna como un entero sin signo de 32 bits. A esta funcion se le pasaron los valores indicados y se noto la siguiente respuesta:

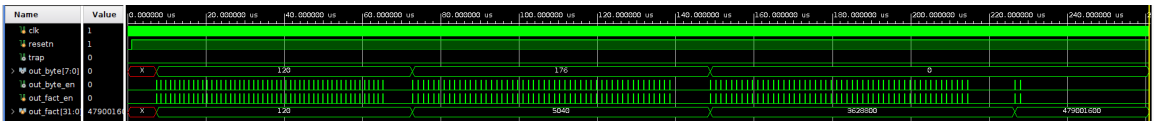

Figura 1: Calculo de factoriales

Notamos que para el out_byte los resultados mayores a 255 no se despliegan correctamente. Esto se debe a que el limite de valor que se puede mostrar usando 8 bits es de 2^8, es decir 255. Notamos que el resultado en el out_fact si se despliega de manera correcta, aun para valores más grandes. La razón que parece tomar tantos ciclos de reloj los cálculos de cada factorial es que estos se espaciaron usando un for que itera varias veces un mismo factorial, esto con el fin de desplegar todos los valores de manera legible. Eliminando este elemento, en promedio se tarda n ciclos de reloj para un calculo.

2.2. Ejercicio 2

Utilizando la simulación post-síntesis de Vivado, obtenemos el siguiente resultado:

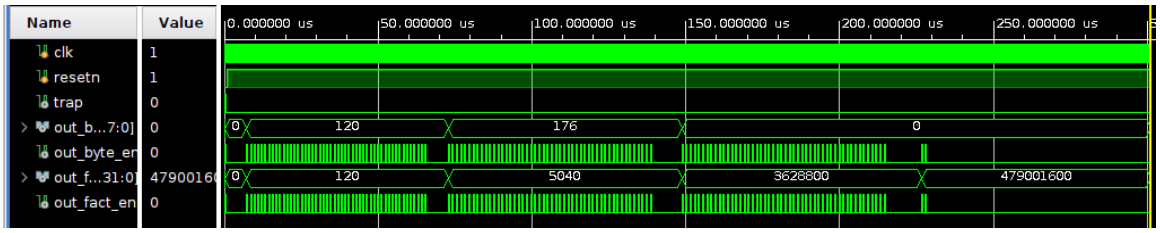

Figura 2: Calculo de factoriales sintetizado

Podemos notar que la respuesta es exactamente la misma que la simulación conductual, demostrando la funcionalidad del programa. Para lograr esta implementación se utilizaron 386 slices, 1178 LUTs, 796 flip-flops, y se calculo una frecuencia máxima de 1337.51MHz para la ejecución.

2.3. Ejercicio 3

Para esta sección se implemento un LUT que recibe dos números de 4 bits y los multiplica, dando un resultado de 8 bits. Para lograr esto se utilizaron dos mux de 4x1. Cada mux va a recibir 0 en una entrada, el numero a multiplicar en otra, el numero desplazado dos bits a la izquierda, y el numero desplazado dos bits a la izquierda mas el mismo numero. Esto equivale a entradas de cero, el numero, el numero por dos, y el numero por tres. La diferencia es que un mux va a tener de selector los dos bits mas significativos y el otro tendrá los dos bits menos significativos del segundo factor. Finalmente la salida de el primer mux se desplaza dos posiciones a la izquierda y se le suma la salida del segundo mux. Esto retorna un numero de 8 bits que es equivalente a la multiplicación de los dos números. Realizando las simulaciones obtenemos el siguiente resultado:

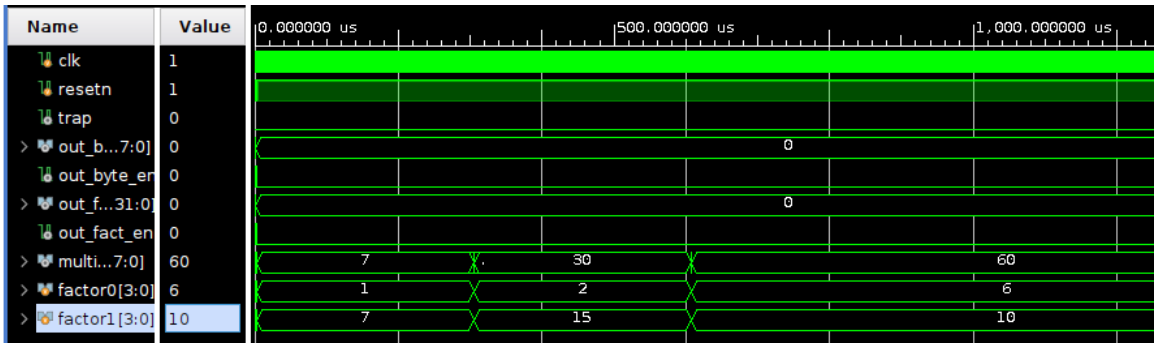

Figura 3: Calculo de multiplicaciones

Notamos que los resultados de las multiplicaciones son los correctos.

2.4. Ejercicio 4

Para generar una salida de 64 bits se decidió instanciar una gran cantidad de LUTs pequeños, esto con el fin de tener una respuesta rápida. Básicamente se instanciaron 4 módulos del multiplicador del ejercicio 3 para crear un multiplicador de dos números de 8 bits, se instanciaron 4 módulos de este ultimo multiplicador para crear uno de dos números de 16 bits, y finalmente se instanciaron 4 módulos de este multiplicador para crear uno de dos números de 32 bits. Esto en esencia creo 256 instancias de el modulo del ejercicio 3, permitiendo una búsqueda rápida de resultados. Se obtuvieron los siguientes resultados tras simular:

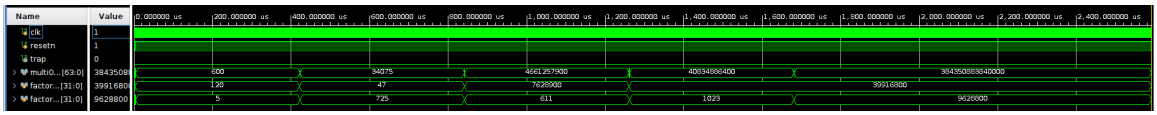

Figura 4: Calculo de multiplicaciones de 32 bits

Se nota la respuesta esperada de parte del multiplicador.

2.5. Ejercicio 5

Se implementó un multiplicador de cuatro bits de tipo array multiplier, para ello se creó un modulo sumador con tres entradas y dos salidas, donde las tres entradas son dos bits correspondientes a dos and de un bit del factor 1 y un bit del factor 2 y un carry de entrada, mientras que las salidas son un carry de salida y el resultado de la suma de las entradas. Luego se creó un modulo multiplicador tipo array de cuatro bits donde se conectaron 2 sumadores en cascada para obtener el resultado de una multiplicación de dos números de cuatro bits.

Se verificó este diseño escribiendo números específicos en el firmware en las posiciones 0xFFFFFFF0 y 0xFFFFFFF4 y leyendo el resultado de la posicion 0xFFFFFFF8, luego se escribieron todos los números del 1 al 15 en cada una de las posiciones de memoria correspondientes a los factores 1 y 2 de la multiplicación para generar todos los posibles resultados de multiplicación en 4bits

2.5.1. Resultados

Figura 5: Resultados de la multiplicación de 4bits

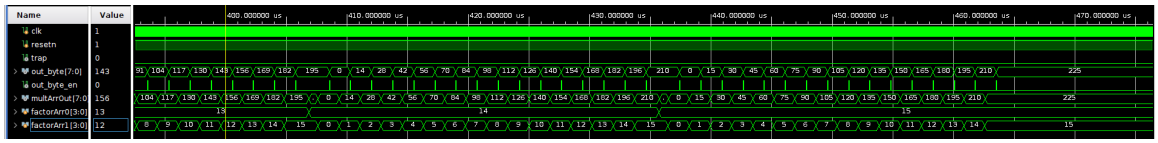

Figura 6: Resultados de la multiplicación de 4bits

Las figuras muestran los resultados de todas las multiplicaciones posibles y el multiplicador funciona perfectamente

2.5.2. Frecuencia, LUTs, slices y flip-flops

Este multiplicador utiliza 3523 LUTs, 985 slices y 828 flip-flops y la frecuencia máxima que la herramienta de síntesis estima es de 1333,33Mhz

2.6. Ejercicio 6

Con el mismo sumador del ejercicio 5 se creó un modulo que acepte dos números N bits de entradas, donde N es un parámetro de verilog. A partir de un bloque generate y utilizando arreglos de bits para guardar los resultados y los carry de cada sumador como si fuese una matriz de números, de modo que se puede iterar sobre este bloque, se hicieron 3 bloques if para los casos especiales donde las salidas o entradas no eran genéricas y se hizo para los sumadores cuyas entradas y salidas eran genéricas. De este modo se evitó el procedimiento tedioso de instanciar una exagerada cantidad de sumadores para lograr un multiplicador de 32bits (32x31 = 992 instancias).

Luego en el firmware se escriben datos en las posiciones de memoria 0xFFFFFFF0 y 0xFFFFFFF4 correspondientes al factor0 y factor1 de la multiplicación y se escribe el resultado en las posiciones 0xFFFFFFF8 y 0xFFFFFFFC, los 32 LSB y 32 MSB respectivamente y en un registro llamado out_fact se lleva la cuenta de cuantos bits en 1 tiene el resultado de cada multiplicación

2.6.1. Resultados

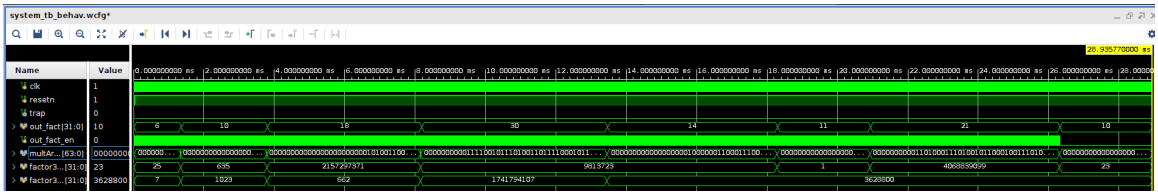

Figura 7: Resultados de la multiplicación de 32bits

El multiplicador y el contador de bits out_en funcionan a la perfección y el diseño del modulo no genera warnings

2.6.2. Frecuencia, LUTs, slices y flip-flops

Este multiplicador utiliza 3523 LUTs, 985 slices y 828 flip-flops y la frecuencia máxima que la herramienta de síntesis estima es de 1333,33Mhz

2.6.3. ¿Que son bloques generate sintetizables?

Un bloque generate permite multiplicar instancias de módulos o realizar instancias condicionales de cualquier módulo, brinda la capacidad de que el diseño se construya en función de los parámetros de verilog, toda instanciacion de un bloque generate se escribe entre los keywords *generate* y *endgenerate* y existen 3 tipos el generate for loop, generate if else y generate case.[1]

2.6.4. ¿Cuántas etapas tiene este nuevo circuito de multiplicación? ¿Qué podría ocurrir con el periodo del reloj si se añaden más y más etapas de lógica combinatoria?

Este circuito tiene N-2 Etapas para realizar la multiplicación, donde N es el numero de bits que tiene cada factor. Por lo tanto este multiplicador consta de 30 etapas, en caso de querer implementar un multiplicador de mayor numero de bits, puede llegar a aumentar la frecuencia máxima de operación del circuito

2.6.5. ¿Qué podría ocurrir con la frecuencia del circuito si se añaden latches entre cada etapa de sumadores?

Si se agregan latches entre cada etapa, se disminuye el tiempo muerto, ya que se puede ir calculando los resultados de una fila siguiente con los datos requeridos antes de que se haya terminado de calcular la fila anterior por completo, lo cual agiliza el proceso y aumenta la frecuencia de operación del circuito

3. Conclusiones

1. Se logró implementar de forma exitosa los multiplicadores de 4, 16 y 32 bits utilizando diferentes diseños(de tipo arreglo y de tipo LUT), implementando modulos programados de RTL y modificando el firmware del riscV32 en estudio

2. Utilizar un bloque generate para instaciar varios módulos cuyas entradas y salidas son genéricas es una opción mas practica que instanciando cada modulo uno por uno, siempre y cuando existan patrones para iterar.

3. Agregar un pipeline al multiplicador utilizando latches disminuye el tiempo máxima de espera y segmenta el proceso siendo una técnica útil a tener en casos donde la cantidad de etapas del circuito afecte el rendimiento.

Referencias

[1] *Verilog Generate Block*. Chip Verify. URL: https://www.chipverify.com/verilog/verilog-generate-block.