Entrega Lab 1 ADAV Bloque A

Daniel Muñiz Zurrunero

Explicación de la práctica

El objetivo de esta práctica es la modificación del circuito realizado en la práctica anterior, alterando el datapath y control que realizan el filtro para usar solo un sumador y un restador y cambiando las interfaces de entrada y salida para que estas sean asíncronas.

Optimizaciones

Convención seguida para la tabla

Para una mayor compresión de la tabla de reserva, se ha decidido seguir el siguiente sistema para denominar a las celdas.

En primer lugar, en lugar de poner una X para indicar que un registro está ocupado, se pone el nombre del registro tmp correspondiente, para así luego, al reducir el número de registros, saber qué rol del código original tiene cada nuevo registro en cada momento.

Por último, para marcar el momento en el que se asigna un valor al registro se ha decidido poner el prefigo g al nombre del registro con fondo rojo, así, la celda de la tabla en la que se genera tmp1 sería:

gtmp1

Proceso de optimización

En primer lugar, se ha planteado el filtro de manera teórica utilizándose solo un multiplicador y un sumador sin ninguna optimización:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
tmp0	gtmp0	tmp0	tmp0	tmp0																
tmp1		gtmp1	tmp1	tmp1	tmp1	tmp1	tmp1	tmp1	tmp1											
tmp2			gtmp2	tmp2	tmp2	tmp2	tmp2	tmp2	tmp2											
tmp3				gtmp3	tmp3	tmp3	tmp3													
tmp4					gtmp4	tmp4														
tmp5						gtmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5	tmp5		
tmp6							gtmp6	tmp6	tmp6	tmp6	tmp6	tmp6	tmp6	tmp6	tmp6	tmp6	tmp6			
tmp7								gtmp7	tmp7	tmp7	tmp7	tmp7	tmp7	tmp7	tmp7	tmp7				
tmp8									gtmp8	tmp8	tmp8	tmp8	tmp8	tmp8	tmp8					
tmp9										gtmp9										
tmp10											gtmp10	tmp10	tmp10	tmp10						
tmp11												gtmp11	tmp11	tmp11	tmp11					
tmp12													gtmp12	tmp12	tmp12	tmp12				
tmp13														gtmp13	tmp13	tmp13	tmp13			
tmp14															gtmp14	tmp14	tmp14	tmp14		
tmp15																gtmp15	tmp15	tmp15	tmp15	
tmp16																	gtmp16	tmp16	tmp16	
tmp17																		gtmp17	tmp17	
tmp18																			gtmp18	
sv1	sv1	sv1	sv1	sv1	sv1	sv1	sv1	sv1	sv1											gsv1
sv2	sv2	sv2	sv2	sv2	sv2	sv2	sv2													gsv2
sv3	sv3	sv3	sv3	sv3	sv3	sv3														gsv3
sv4	sv4	sv4	sv4	sv4	sv4															gsv4

A continuación, se ha observado que como tmp0 proviene de la interfaz de entrada, su valor ya está registrado, además, sv1-4 se corresponden con gtmp15-18, por lo que no hace falta crear registros nuevos para ellos. Por último, se han paralelizado los procesos de suma y multiplicación, obteniéndose el siguiente resultado (representando las líneas de código paralelas operaciones en paralelo):

```
tmp1 = tmp0 * b(1);
tmp2 = tmp0 * b(2); tmp9 = tmp1 + tmp15;
tmp3 = tmp0 * b(3); tmp8 = tmp2 + tmp16;
tmp4 = tmp0 * b(4); tmp7 = tmp3 + tmp17;
tmp5 = tmp0 * b(5); tmp6 = tmp4 + tmp18;

tmp10 = tmp9 * (-1/neg_a(1));
tmp11 = tmp10 * neg_a(2); y = [y tmp10];
tmp12 = tmp10 * neg_a(3); tmp15 = tmp8 + tmp11;
tmp13 = tmp10 * neg_a(4); tmp16 = tmp7 + tmp12;
tmp14 = tmp10 * neg_a(5); tmp17 = tmp6 + tmp13;
tmp18 = tmp5 + tmp14;
```

	1	2	3	4	5	6	7	8	9	10	11
tmp1	gtmp1										
tmp2		gtmp2									
tmp3			gtmp3								
tmp4				gtmp4							
tmp5					gtmp5	tmp5	tmp5	tmp5	tmp5	tmp5	
tmp6					gtmp6	tmp6	tmp6	tmp6	tmp6		
tmp7				gtmp7	tmp7	tmp7	tmp7	tmp7			
tmp8			gtmp8	tmp8	tmp8	tmp8	tmp8				
tmp9		gtmp9	tmp9	tmp9	tmp9						
tmp10						gtmp10	tmp10	tmp10	tmp10		
tmp11							gtmp11				
tmp12								gtmp12			
tmp13									gtmp13		
tmp14										gtmp14	
tmp15	tmp15							gtmp15	tmp15	tmp15	tmp15
tmp16	tmp16	tmp16							gtmp16	tmp16	tmp16
tmp17	tmp17	tmp17	tmp17							gtmp17	tmp17
tmp18	tmp18	tmp18	tmp18	tmp18							gtmp18

Harían falta 11 ciclos. Por último, se optimizaron los registros utilizando la tabla:

	1	2	3	4	5	6	7	8	9	10	11
reg1	gtmp1	gtmp2	gtmp3	gtmp4	gtmp5	tmp5	tmp5	tmp5	tmp5	tmp5	
reg2	tmp15	gtmp9	tmp9	tmp9	tmp9		gtmp11	gtmp15	tmp15	tmp15	tmp15
reg3	tmp16	tmp16	gtmp8	tmp8	tmp8	tmp8	tmp8	gtmp12	gtmp16	tmp16	tmp16
reg4	tmp17	tmp17	tmp17	gtmp7	tmp7	tmp7	tmp7	tmp7	gtmp13	gtmp17	tmp17
reg5	tmp18	tmp18	tmp18	tmp18	gtmp6	tmp6	tmp6	tmp6	tmp6	gtmp14	gtmp18
reg6						gtmp10	tmp10	tmp10	tmp10		

```
reg1 = tmp0 * b(1);
reg1 = tmp0 * b(2); reg2 = reg1 + reg2;
reg1 = tmp0 * b(3); reg3 = reg1 + reg3;
reg1 = tmp0 * b(4); reg4 = reg1 + reg4;
reg1 = tmp0 * b(5); reg5 = reg1 + reg5;

reg6 = reg2 * (-1/neg_a(1));
reg2 = reg6 * neg_a(2); y = [y reg6];
reg3 = reg6 * neg_a(3); reg2 = reg3 + reg2;
reg4 = reg6 * neg_a(4); reg3 = reg4 + reg3;
reg5 = reg6 * neg_a(5); reg4 = reg5 + reg4;
reg5 = reg1 + reg5;
```

Con 6 registros se podría realizar el algoritmo en 11 ciclos.

Modificaciones realizadas

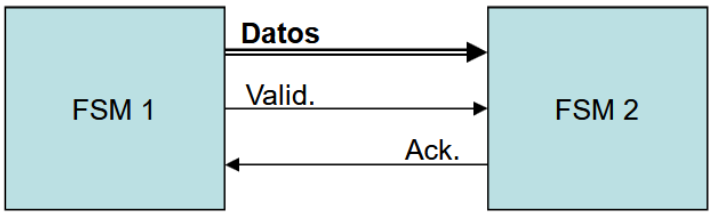
Interfaces de entrada y salida

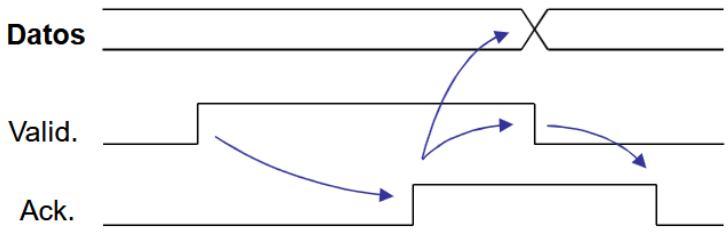
Las interfaces de entrada y salida se han adaptado para ser asíncronas.

A la interfaz de entrada se le ha añadido una salida extra correspondiente al ACK que debe devolver al recibir la señal de validación por parte de una interfaz de salida externa. Este ACK se encuentra a nivel bajo hasta que recibe la señal de validación a 1, momento en el cual se activa a nivel alto hasta que la señal de validación pasa a ser 0. El dato de entrada se registra con la señal de validación.

A la interfaz de salida se le ha añadido una entrada extra correspondiente también al ACK que recibe por parte de una interfaz de entrada externa tras mandar su señal de validación. De manera homónima a la interfaz de entrada, la señal de validación se activará a nivel alto al finalizarse las operaciones del datapath (momento marcado por la activación a nivel alto de la señal fin) y se mantendrá hasta recibir el ACK de la interfaz de entrada, momento en el que la señal de validación volverá a nivel bajo. El dato de salida se registra al finalizarse las operaciones del datapath y se mantiene hasta recibirse el ACK.

El funcionamiento de las interfaces queda ilustrado en la siguiente imagen extraída de las transparencias:





Datapath

El Datapath se ha modificado completamente con respecto a su versión original, creándose cuatro procesos (que podrían haberse simplificado a tres):

- **Asignación de entradas de operadores:** Proceso combinacional que multiplexa las entradas del sumador y multiplicador para introducir las correspondientes al estado en el que se encuentra el datapath.
- **Operadores:** Proceso (en este caso es proceso para una mejor explicación, aunque debido a su simplicidad en el código son dos líneas fuera de un proceso) que realiza la suma y multiplicación de sus entradas.
- **Asignación de salidas:** Proceso combinacional (para asignación de señales _next) seguido de un proceso secuencial (para asignar las señales _next a sus registros de salida) que multiplexa las salidas del datapath para asignar las correspondientes al estado en el que este se encuentra. Este proceso se puede modelar como un solo proceso secuencial si se quiere. Se añade un estado idle que reinicia los registros para el próximo cálculo.

Control

La unidad de control se ha modificado para hace que el datapath pase por todos los estados por los que debe pasar la ejecución del filtro y genere la señal de fin tras 17 (16+1) secuencias de cálculo del filtro (cuenta llevada a cabo por una señal counter), este valor se ha determinado mediante observación, viendo que entorno a ese número de cálculos del filtro la señal de salida se estabiliza (como se verá en la simulación). Cuando llega la señal de validación a nivel alto, se inicia la secuencia de transición por todos los estados para volverse al estado de reposo tras acabar.

top

En el módulo top simplemente se han añadido las señales de ACK y se han modificado las entradas y salidas de los módulos en su instanciación para que se correspondan con sus nuevas versiones modificadas.

Simulación y resultados

Modificaciones en la simulación

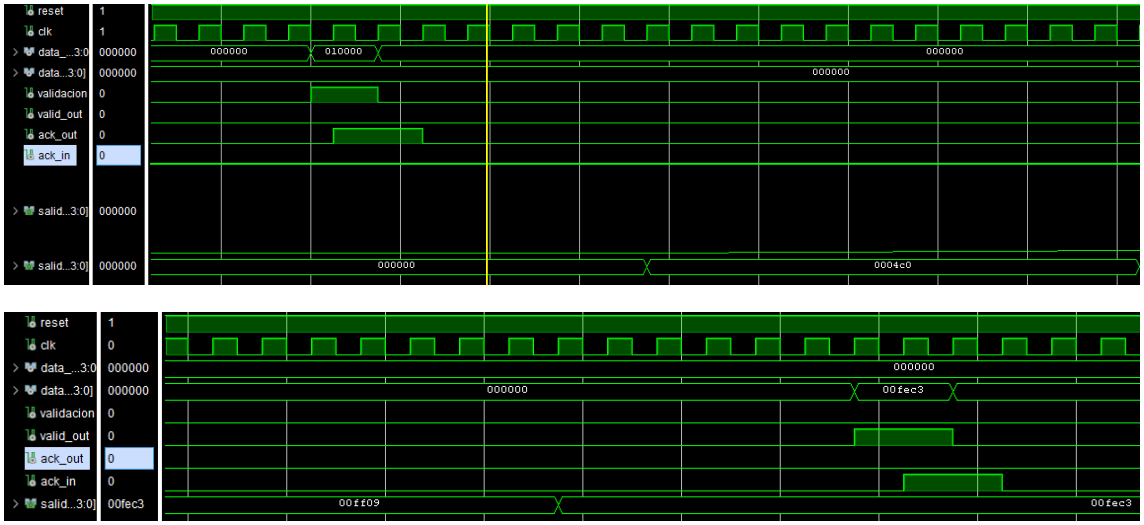
Para la simulación del nuevo sistema se ha decidido utilizar como base el testbench ya existente de la práctica anterior, añadiendo las señales de ACK y modificando el comportamiento del testbench para que este mande una señal de validación esperando un ACK de la interfaz de entrada y mande un ACK como respuesta a la señal de validación de la interfaz de salida, ambas señales del testbench no reaccionan inmediatamente, sino que esperan un pulso de reloj, para comprobar que las señales de validación y ACK

del módulo reaccionan correctamente. De esta forma se ha comprobado el envío de dos datos para el cálculo del filtro.

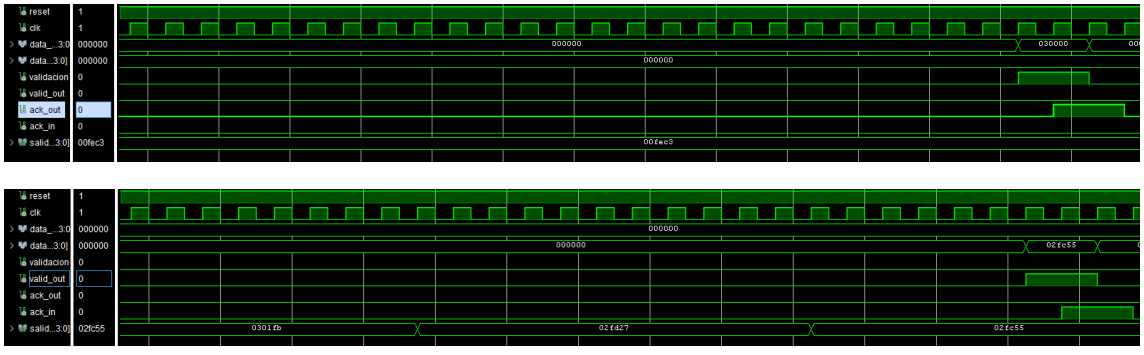
Por último, se ha modificado la escritura en fichero para que este escriba cuando tanto la señal de validación del módulo como el ACK del testbench estén a 1 (es decir, en el pulos exacto en el que la comunicación termina), para así evitar datos repetidos de un mismo cálculo.

Resultados

A continuación, se muestran los resultados de la simulación para una transmisión, los cuales se pueden ver que son correctos, esperando el módulo para reaccionar al ACK o señal de validación del testbench:



Por último, se puede observar que la segunda transición funciona de manera similar a la anterior:



Las salidas de ambas transmisiones se guardan en el fichero f_out.txt de la carpeta xsim del proyecto.