Álvaro Montesano Martínez

ADAV

Memoria Práctica_A2

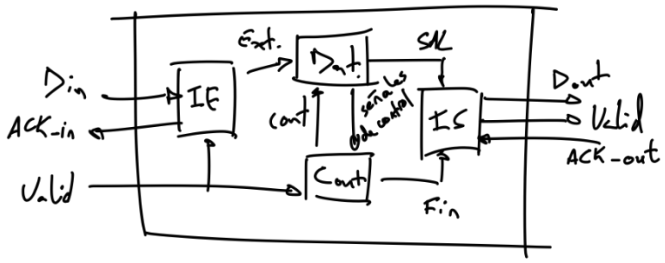
1.- Introducción

En esta práctica se implementará un sistema de filtrado utilizando dos operadores, un multiplicador y un sumador/restador implementado mediante una estructura RippleCarry. Este sistema contará con dos interfaces para la comunicación externa, **Interfaz de entrada** e **Interfaz de salida**, con una unidad de **Control** y un **Datapath** que hará uso de los operadores. Estos están implementados en dos unidades independientes **RippleCarry** y **Multiplicador**, los cuales importaremos al datapath para su uso. Todos estos módulos serán controlados por la unidad **Top**, el cual se encargará de unir todos estos módulos y obtener el sistema final.

2.- Interfaces de entrada y salida

Para adaptar el sistema a un control externo hemos incluido la señal ack_in. Esta señal transmite un pulso al control exterior cuando el sistema esta validado y los datos cargados en la entrada.

Haciendo algo similar en el interfaz de salida hemos creado la señal ack_out. Esta señal recibe el estado del sistema externo, '0' significa libre y '1' significa ocupado. Mediante esta señal, al confirmar que el sistema externo esta libre, procedemos cargar los valores de las salidas en el bus data_out y ponemos la señal valid_out a '1'.



3.- Planificación tabla de reserva

El sistema base contaba con 18 operaciones para implementar el sistema filtrado, teniendo en cuenta que cada operación ocupaba 1 ciclo de reloj, el número total de ciclos que tardaba en realizar un filtrado eran 18 ciclos de reloj.

01-04-2024

Ciclos	Multiplicador
1	tmp1 = m_tmp1 (39 downto 16) = tmp0 * b1
2	tmp2 = m_tmp2 (39 downto 16) = tmp0 * b2
3	tmp3 = m_tmp3 (39 downto 16) = tmp0 * b3
4	tmp4 = m_tmp4 (39 downto 16) = tmp0 * b4
5	tmp5 = m_tmp5 (39 downto 16) = tmp0 * b5
	Suma
6	tmp9 = tmp1 + sv1
7	tmp8 = tmp2 + sv2
8	tmp7 = tmp3 + sv3
9	tmp6 = tmp4 + sv4
	Multiplicador
10	tmp10 = m_tmp10 (39 downto 16) = tmp9 * inv_a1
11	tmp11 = m_tmp11 (39 downto 16) = tmp10 * neg_a2
12	tmp12 = m_tmp12 (39 downto 16) = tmp11 * neg_a3
13	tmp13 = m_tmp13 (39 downto 16) = tmp12 * neg_a4
14	tmp14 = m_tmp14 (39 downto 16) = tmp13 * inv_a5
	Suma
15	sv4 = tmp15 = tmp8 + tmp11
16	sv3 = tmp16 = tmp7 + tmp12
17	sv2 = tmp17 = tmp6 + tmp13
18	sv1 = tmp18 = tmp5 + tmp14

Con las operaciones simplificadas ordenadas procedemos a realizar la tabla de reserva.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
tmp0	X	X	X	X														
tmp1	X	X	X	X	X	X	X	X										
tmp2		X	X	X	X	X	X											
tmp3			X	X	X	X												
tmp4				X	X													
tmp5					X	X	X	X	X	X	X	X	X	X	X	X	X	
sv1	X	X	X	X	X	X	X	X										
sv2	X	X	X	X	X	X	X											
sv3	X	X	X	X	X	X												
sv4	X	X	X	X	X													
tmp6									X	X	X	X	X	X	X	X		
tmp7								X	X	X	X	X	X	X	X			
tmp8							X	X	X	X	X	X	X	X				
tmp9						X	X	X	X									
tmp10										X	X	X	X	X	X	X	X	X
tmp11											X	X	X	X				
tmp12												X	X	X	X			
tmp13													X	X	X	X		
tmp14														X	X	X	X	
tmp15															X	X	X	X
tmp16																X	X	X
tmp17																	X	X
tmp18																		X

Viendo la tabla de reserva observamos que podemos realizar la paralización de varias operaciones, pudiendo usar un multiplicador y un sumador en el mismo ciclo. A continuación, realizaremos la paralelización de dichas operaciones.

Álvaro Montesano Martínez

	Multiplicador	Suma
1	tmp1 = m_tmp1 (39 downto 16) = tmp0 * b1	
2	tmp2 = m_tmp2 (39 downto 16) = tmp0 * b2	tmp9 = tmp1 + sv1
3	tmp3 = m_tmp3 (39 downto 16) = tmp0 * b3	tmp8 = tmp2 + sv2
4	tmp4 = m_tmp4 (39 downto 16) = tmp0 * b4	tmp7 = tmp3 + sv3
5	tmp5 = m_tmp5 (39 downto 16) = tmp0 * b5	tmp6 = tmp4 + sv4
6	tmp10 = m_tmp10 (39 downto 16) = tmp9 * inv_a1	
7	tmp11 = m_tmp11 (39 downto 16) = tmp10 * neg_a2	
8	tmp12 = m_tmp12 (39 downto 16) = tmp11 * neg_a3	sv4 = tmp15 = tmp8 + tmp11
9	tmp13 = m_tmp13 (39 downto 16) = tmp12 * neg_a4	sv3 = tmp16 = tmp7 + tmp12
10	tmp14 = m_tmp14 (39 downto 16) = tmp13 * inv_a5	sv2 = tmp17 = tmp6 + tmp13
11	sv1 = tmp18 = tmp5 + tmp14	

Con esta paralelización obtenemos la siguiente tabla de reserva (reduciendo de 18 a 11 ciclos necesarios):

	1	2	3	4	5	6	7	8	9	10	11
tmp0	X	X	X	X							
tmp1	X										
tmp2		X									
tmp3			X								
tmp4				X							
tmp5					X	X	X	X	X	X	
sv1	X										
sv2	X	X									
sv3	X	X	X								
sv4	X	X	X	X							
tmp6					X	X	X	X	X		
tmp7				X	X	X	X	X			
tmp8			X	X	X	X	X				
tmp9		X	X	X	X						
tmp10						X	X	X	X	X	X
tmp11							X				
tmp12								X			
tmp13									X		
tmp14										X	
tmp15								X	X	X	X
tmp16									X	X	X
tmp17										X	X
tmp18											X

Con esta información, podemos saber que registros podemos agrupar, quedándonos la tabla de reserva con los registros agrupados, de la siguiente manera:

	1	2	3	4	5	6	7	8	9	10	11
tmp0	r1	r1	r1	r1							
tmp1	r2										
tmp2		r2									
tmp3			r2								
tmp4				r2							
tmp5					r2	r2	r2	r2	r2	r2	r2
sv1	r3										
sv2	r4	r4									
sv3	r5	r5	r5								
sv4	r6	r6	r6	r6							
tmp6					r1	r1	r1	r1	r1		
tmp7				r5	r5	r5	r5	r5			
tmp8			r4	r4	r4	r4	r4				
tmp9		r3	r3	r3	r3						
tmp10						r3	r3	r3	r3	r3	r3
tmp11							r6				
tmp12								r4			
tmp13									r4		
tmp14										r1	
tmp15								r6	r6	r6	r6
tmp16									r5	r5	r5
tmp17										r4	r4
tmp18											r1

De las posibles agrupaciones que hemos tenido en cuenta, esta es la óptima ya que solamente hace un cambio de registro, siendo este el de temp18 a sv1:

Carga de datos finales en operaciones iniciales	tmp15 -> sv4	r6	r6
	tmp16 -> sv3	r5	r5
	tmp17 -> sv2	r4	r4
	tmp18 -> sv1	r1	r3

De la siguiente manera será como quedaran los registros y las correspondientes operaciones asociadas a estos:

Ciclo	Operación con registros antiguos		Operación con registros nuevos	
0 (idle)	tmp0 = entradas		r1 = entradas	
1	tmp1 = m_tmp1 (39 downto 16) = tmp0 * b1		r2 = r1 * b1	
2	tmp2 = m_tmp2 (39 downto 16) = tmp0 * b2	tmp9 = tmp1 + sv1	r2 = r1 * b2	r3 = r2 + r3
3	tmp3 = m_tmp3 (39 downto 16) = tmp0 * b3	tmp8 = tmp2 + sv2	r2 = r1 * b3	r4 = r2 + r4
4	tmp4 = m_tmp4 (39 downto 16) = tmp0 * b4	tmp7 = tmp3 + sv3	r2 = r1 * b4	r5 = r2 + r5
5	tmp5 = m_tmp5 (39 downto 16) = tmp0 * b5	tmp6 = tmp4 + sv4	r2 = r1 * b5	r1 = r2 + r6
6	tmp10 = m_tmp10 (39 downto 16) = tmp9 * inv_a1		salidas = r3 = r3 * inv_a1	
7	tmp11 = m_tmp11 (39 downto 16) = tmp10 * neg_a2		r6 = r3 * neg_a2	
8	tmp12 = m_tmp12 (39 downto 16) = tmp11 * neg_a3	tmp15 = tmp8 + tmp11	r4 = r6 * neg_a3	r6 = r4 + r6
9	tmp13 = m_tmp13 (39 downto 16) = tmp12 * neg_a4	tmp16 = tmp7 + tmp12	r4 = r4 * neg_a4	r5 = r5 + r4
10	tmp14 = m_tmp14 (39 downto 16) = tmp13 * inv_a5	tmp17 = tmp6 + tmp13	r1 = r4 * inv_a5	r4 = r1 +r4
11	tmp18 = tmp5 + tmp14		r1 = r2 + r1	

4.- Implementación RippleCarry y Multiplicador

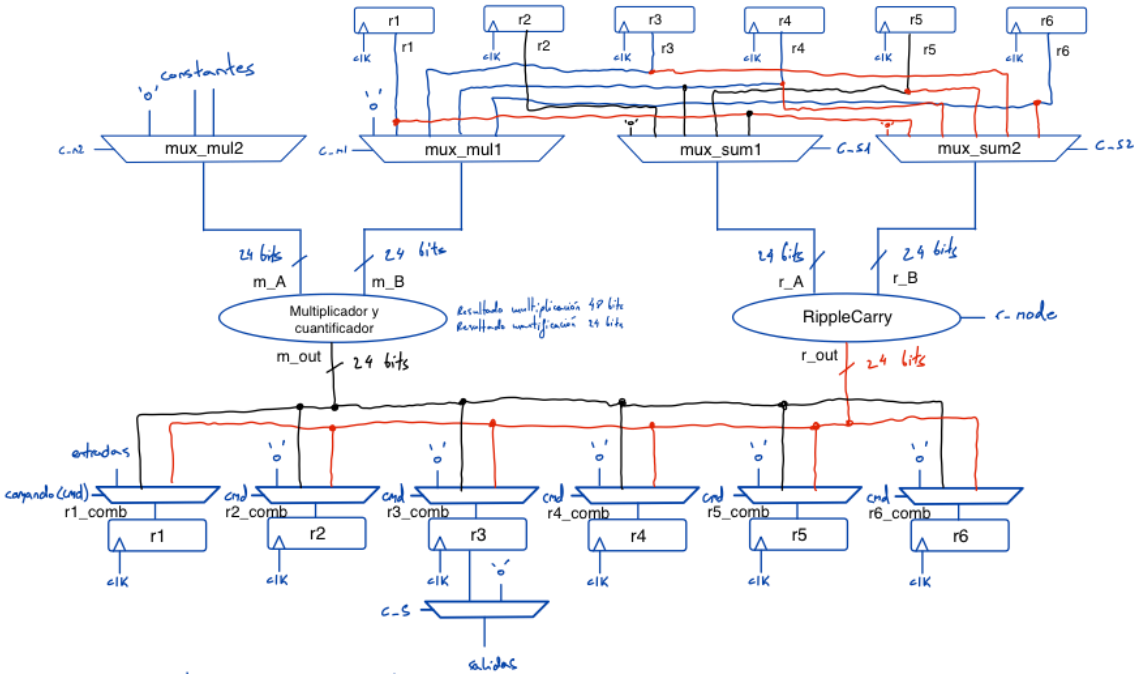
Para la implementación del RippleCarry hemos utilizado 24 FullAdder, poniéndolos en serie y obteniendo la estructura Ripple Carry. Mediante la señal de control r_mode seremos capaces de seleccionar el modo de funcionamiento del RippleCarry, '0' suma y '1' resta, en este caso al interesarnos el operando suma lo dejaremos a '0'.

Para la implementación del módulo hemos realizado la operación de multiplicación de dos vectores de 24 bits que se proporcionan por las entradas de este. Esta operación produce un vector de 48 bits el cual hemos decidido recortar en este propio módulo, para no tener que gestionar vectores de 48 bits en el datapath, a un vector de 24 bits el cual es el que se devuelve. Hemos usado la misma reducción de bits que la proporcionada.

5.- Diseño del datapath y planificación de las señales en módulo de control

Tras hacer la planificación del sistema, junto con la pertinente reducción de registros, procedemos a realizar el diseño del datapath:

Álvaro Montesano Martínez



Como se puede observar en la imagen anterior de 4 multiplexores para proporcionar los datos a los operadores.

Ciclo	Codificación c_m2 (4 bits)	Salida mux_mul2 (m_A)
0 (idle)	0000	cero
1	0001	b1
2	0010	b2
3	0011	b3
4	0100	b4
5	0101	b5
6	0110	inv_a1
7	0111	neg_a2
8	1000	neg_a3
9	1001	neg_a4
10	1010	neg_a5
11	0000	cero

Ciclo	Codificación c_m1 (3 bits)	Salida mux_mul1 (m_B)
0 (idle)	000	cero
1	001	r1
2	001	r1
3	001	r1
4	001	r1
5	001	r1
6	011	r3
7	011	r3
8	110	r6
9	100	r4
10	100	r4
11	000	cero

Ciclo	Codificación c_s2 (3 bits)	Salida mux_sum1 (r_A)
0 (idle)	000	cero
1	000	cero
2	011	r3
3	100	r4
4	101	r5
5	110	r6
6	000	cero
7	000	cero
8	110	r6
9	100	r4
10	100	r4
11	001	r1

Ciclo	Codificación c_s1 (3 bits)	Salida mux_sum1 (r_B)
0 (idle)	000	cero
1	000	cero
2	010	r2
3	010	r2
4	010	r2
5	010	r2
6	000	cero
7	000	cero
8	100	r4
9	101	r5
10	001	r1
11	010	r2

Disponemos de 1 multiplexor por cada registro, todos son controlados por la señal comando. Esta señal es la generada por el *process* del módulo control proporcionado, incrementando los estados desde clk0 hasta clk11.

clk	Comando (cmd)	Output mux_r1	Output mux_r2	Output mux_r3	Output mux_r4	Output mux_r5	Output mux_r6
0 (idle)	000000000001	entradas	cero	cero	cero	cero	cero
1	000000000010	entradas	m_out	cero	cero	cero	cero
2	000000000100	entradas	m_out	r_out	cero	cero	cero
3	000000001000	entradas	m_out	cero	r_out	cero	cero
4	000000010000	entradas	m_out	cero	cero	r_out	cero
5	000000100000	r_out	m_out	cero	cero	cero	cero
6	000001000000	cero	cero	m_out	cero	cero	cero
7	000010000000	cero	cero	cero	cero	cero	m_out
8	000100000000	cero	cero	cero	m_out	cero	r_out
9	001000000000	cero	cero	cero	m_out	r_out	cero
10	010000000000	m_out	cero	cero	r_out	cero	cero
11	100000000000	r_out	cero	cero	cero	cero	cero

Como se puede ver en la tabla anteriro hemos decidido cargar los valores de entrada en el registro r1 en ciclo 0 (idle). Esto se debe a que necesitamos que esten los datos en el registro r1 antes del ciclo 1 que es cuando comenzamos a realizar operaciones.

El último multiplexador es el que se encarga de cargar los valores del r3 (tmp10) en la salida, mediante la señal de control c_s. Esta señal se pone a '1' desde el ciclo 6 hasta el ciclo 11, cargando el valor de r3 a la salida.

Todas las señales de control se implementan en el módulo de **Control** y mediante el módulo **Top** se interconecta con el datapath.

Señales de control	Tamaño (bits)
c_m2	4
c_m1	3
c_s2	3
c_s1	3
r_mode	1
comando	12

Teniendo en cuenta estas tablas hemos implementado el **Datapath** mediante un *process* combinacional en el cual implementamos todas las estructuras de *case* necesarias para implementar los multiplexores. Para implementar los registros hemos utilizado un *process* secuencial, cargando así los valores al inicio del ciclo de reloj. El módulo de control lo hemos implementado mediante un *process* secuencial el cual se encarga de generar el estado del sistema y mediante un *process* combinacional, el cual depende de clk y de la señal estado, hemos determinado el valor de todas las señales de control dependiendo del estado en el que se encuentre el sistema.