2011/
2012

Diseño VHDL de Procesadores con Técnicas de Paralelismo de Instrucciones

Práctica 1

Alberto Manuel Mireles Suárez
David Guillermo Morales Sáez

ULPGC
Arquitectura de Computadores
2011/2012

Contenido

Introducción

En esta práctica, el primer paso ha sido comprender los diseños VHDL del procesador DLX32p suministrados. Una vez hecho esto, hemos creado una rutina que lleva a cabo la multiplicación rusa y la hemos utilizado para realizar el producto escalar de dos vectores. Tras esto, hemos realizado la correspondiente simulación en Xilinx con el fin de analizar la eficiencia del sistema implementado.
Una vez hecho esto, hemos modificado la instrucción de multiplicación de manera que permita realizar multiplicaciones de dos vectores de cuatro elementos cada uno. También hemos modificado el programa anterior para adaptarlo a esta nueva implementación, tomando los resultados de eficiencia obtenidos. A su vez, hemos intentado optimizar el código mediante la reorganización del mismo para evitar penalizaciones por dependencia de datos y comparado los datos de eficiencia con el resto de implementaciones.

Análisis de prestaciones previo.

Inicialmente el procesador está formado por un repertorio de instrucciones muy reducido (ADD, SUB, AND, ORI, SRA, SLL, LW, SW, BEQZ, J) por lo que para implementar el producto escalar se requiere un gran número de instrucciones, principalmente debido a la falta de una instrucción de multiplicación. Debido a esto hemos necesitado más de 200 instrucciones para llevar a cabo el producto escalar de dos vectores, con 10 valores cada uno.
Para esta implementación, el número de ciclos de reloj es de **786** a una frecuencia máxima de 11,244 MHz y un tiempo de ejecución de **7860 ns**.

Familia	XC4000XV	
Dispositivo	40110XVBG560	
	DLX32p	Main32p
Número de CLBs	719 (17%)	1012 (24%)
Frecuencia máxima	10,655 MHz	11,244 MHz

Propuesta de modificación de la microarquitectura

Con el fin de mejorar el rendimiento del programa y aportar una instrucción que permita tener programas más pequeños que ocupen menos espacio en memoria, además de facilitar el diseño de los mismos, hemos creado una instrucción de tipo vectorial que lleva a cabo el producto escalar de dos vectores, cada uno con cuatro valores.
Esto nos permitiría reducir el número de operaciones ya que, hasta ahora, teníamos que realizar 10 productos y 10 sumas independientes mientras que con la propuesta se requerirán solamente tres productos y tres sumas.
Esperamos ver una reducción considerable en tiempo de ejecución ya que al ser capaces de disminuir el número de instrucciones, aun aumentando el tiempo de ciclo, se requerirán menos ciclos para obtener el resultado.

Descripción de la microarquitectura mejorada

Para llevar a cabo la modificación de la arquitectura hemos modificado 3 archivos: dlx_pack.vhd, ex.vhf y el fichero de sintaxis del compilador suministrado.

En el archivo de sintáxis se ha indicado que al introducir una instrucción "mult" se codificará con el código 101111 que corresponde con la nueva instrucción que se añadirá al hardware.

Dlx_pack.vhd

Hemos modificado uno de los códigos no definidos de función de las operaciones de la ALU para añadir la función de multiplicación vectorial (mult). Esto se indica de la siguiente forma:

```
Constant cALUFunc_mult : TypeDLXFunc := "101111";
```

Ex.vhd

En este fichero se especifican las operaciones que se llevan a cabo en la etapa de ejecución (ALU), y es donde hemos añadido la rutina de ejecución de la función mult. Para ello hemos añadido el siguiente código:

```
when cAluFunc_mult =>
      iAlu_out :=CONV_SIGNED(
            MUX_A_out(7 downto 0)*MUX_B_out(7 downto 0) +
            MUX_A_out(15 downto 8)*MUX_B_out(15 downto 8) +
            MUX_A_out(23 downto 16)*MUX_B_out(23 downto 16) +
            MUX_A_out(31 downto 24)*MUX_B_out(31 downto 24),32);
```

Programa

El programa inicial que hemos diseñado para la realización del producto escalar de dos vectores empleando la multiplicación rusa es el siguiente:

PC	Destino Salto	Código ensamblador	Código máquina
0x0000		ori r30,r0,0	00110100000111100000000000000000
0x0004		ori r3, r0, 0	00110100000000011000000000000000
0x0008		ori r10, r0, 0x1F20	00110100000010100001111100100000
0x000C		ori r12, r0, 0x0720	00110100000011000000011100100000
0x0010		ori r11, r0, 1	00110100000010110000000000000001
0x0014		ori r14, r0, 0xFF	00110100000011100000000011111111
0x0018		ori r13, r0, 0x8	00110100000011010000000000001000
0x001C		nop	00000000000000000000000000000000
0x0020		and r1, r10, r14	00000001010011100000100000100100
0x0024		and r2, r12, r14	00000001100011100001000000100100
0x0028	0x004C	j mrusa	00001000000000000000000000100000
0x002C		sra r10, r10, r13	00000001010011010101000000000111
0x0030		sra r12, r12, r13	00000001100011010110000000000111
0x0034		nop	00000000000000000000000000000000
0x0038	0x0094	beqz r10, siguientep1	00010001010000000000000001011000
0x003C		nop	00000000000000000000000000000000
0x0040		nop	00000000000000000000000000000000
0x0044	0x0020	j bucle	00001011111111111111111111011000
0x0048		nop	00000000000000000000000000000000
0x004C		nop	00000000000000000000000000000000
0x0050		and r4, r2, r11	00000000010010110010000000100100
0x0054		nop	00000000000000000000000000000000
0x0058		nop	00000000000000000000000000000000
0x005C	0x006C	beqz r4, desplazar	00010000100000000000000000001100
0x0060		nop	00000000000000000000000000000000
0x0064		nop	00000000000000000000000000000000
0x0068		add r3, r3, r1	00000000011000010001100000100000
0x006C		sra r2,r2,r11	00000000010010110001000000000111
0x0070		sll r1,r1,r11	00000000001010110000100000000100
0x0074		nop	00000000000000000000000000000000
0x0078		nop	00000000000000000000000000000000
0x007C	0x002C	beqz r2, sig	00010000100000011111111110101100
0x0080		nop	00000000000000000000000000000000
0x0084		nop	00000000000000000000000000000000
0x0088	0x004C	j mrusa	00001011111111111111111111000000
0x008C		nop	00000000000000000000000000000000
0x0090		nop	00000000000000000000000000000000
0x0094		ori r10, r0, 0x2120	00110100000010100010000100100000
0x0098		ori r12, r0, 0x0D08	00110100000011000000110100001000
0x009C		ori r11, r0, 1	00110100000010110000000000000001
0x00A0		and r1, r10, r14	00000001010011100000100000100100

0x00A4		and r2, r12, r14	00000001100011100001000000100100
0x00A8	0x00CC	j mrusa1	00001000000000000000000000100000
0x00AC		sra r10, r10, r13	00000001010011010101000000000111
0x00B0		sra r12, r12, r13	00000001100011010110000000000111
0x00B4		nop	00000000000000000000000000000000
0x00B8	0x0114	beqz r10, siguientep2	00010001010000000000000001011000
0x00BC		nop	00000000000000000000000000000000
0x00C0		nop	00000000000000000000000000000000
0x00C4	0x00A0	j bucle1	00001011111111111111111111011000
0x00C8		nop	00000000000000000000000000000000
0x00CC		nop	00000000000000000000000000000000
0x00D0		and r4, r2, r11	00000000010010110010000000100100
0x00D4		nop	00000000000000000000000000000000
0x00D8		nop	00000000000000000000000000000000
0x00DC	0x00EC	beqz r4, desplazar1	00010000100000000000000000001100
0x00E0		nop	00000000000000000000000000000000
0x00E4		nop	00000000000000000000000000000000
0x00E8		add r3, r3, r1	00000000011000010001100000100000
0x00EC		sra r2,r2,r11	00000000010010110001000000000111
0x00F0		sll r1,r1,r11	00000000001010110000100000000100
0x00F4		nop	00000000000000000000000000000000
0x00F8		nop	00000000000000000000000000000000
0x00FC	0x00AC	beqz r2, sig1	00010000010000001111111110101100
0x0100		nop	00000000000000000000000000000000
0x0104		nop	00000000000000000000000000000000
0x0108	0x00CC	j mrusa1	00001011111111111111111111000000
0x010C		nop	00000000000000000000000000000000
0x0110		nop	00000000000000000000000000000000
0x0114		ori r10, r0, 0x1F20	00110100000010100001111100100000
0x0118		ori r12, r0, 0x192C	00110100000011000001100100101100
0x011C		ori r11, r0, 1	00110100000010110000000000000001
0x0120		and r1, r10, r14	00000001010011100000100000100100
0x0124		and r2, r12, r14	00000001100011100001000000100100
0x0128	0x014C	j mrusa2	00001000000000000000000000100000
0x012C		sra r10, r10, r13	00000001010011010101000000000111
0x0130		sra r12, r12, r13	00000001100011010110000000000111
0x0134		nop	00000000000000000000000000000000
0x0138	0x0194	beqz r10, siguientep3	00010001010000000000000001011000
0x013C		nop	00000000000000000000000000000000
0x0140		nop	00000000000000000000000000000000
0x0144	0x0120	j bucle2	00001011111111111111111111011000
0x0148		nop	00000000000000000000000000000000
0x014C		nop	00000000000000000000000000000000
0x0150		and r4, r2, r11	00000000010010110010000000100100
0x0154		nop	00000000000000000000000000000000
0x0158		nop	00000000000000000000000000000000
0x015C	0x016C	beqz r4, desplazar2	00010000100000000000000000001100
0x0160		nop	00000000000000000000000000000000
0x0164		nop	00000000000000000000000000000000
0x0168		add r3, r3, r1	00000000011000010001100000100000

0x016C		sra r2,r2,r11	00000000010010110001000000000111
0x0170		sll r1,r1,r11	00000000001010110000100000000100
0x0174		nop	00000000000000000000000000000000
0x0178		nop	00000000000000000000000000000000
0x017C	0x012C	beqz r2, sig2	00010000010000001111111110101100
0x0180		nop	00000000000000000000000000000000
0x0184		nop	00000000000000000000000000000000
0x0188	0x014C	j mrusa2	00001011111111111111111111000000
0x018C		nop	00000000000000000000000000000000
0x0190		nop	00000000000000000000000000000000
0x0194		ori r10, r0, 0x2120	00110100000010100010000100100000
0x0198		ori r12, r0, 0x020B	00110100000011000000001000001011
0x019C		ori r11, r0, 1	00110100000010110000000000000001
0x01A0		and r1, r10, r14	00000001010011100000100000100100
0x01A4		and r2, r12, r14	00000001100011100001000000100100
0x01A8	0x01CC	j mrusa3	00001000000000000000000000100000
0x01AC		sra r10, r10, r13	00000001010011010101000000000111
0x01B0		sra r12, r12, r13	00000001100011010110000000000111
0x01B4		nop	00000000000000000000000000000000
0x01B8	0x0214	beqz r10, siguientep4	00010001010000000000000001011000
0x01BC		nop	00000000000000000000000000000000
0x01C0		nop	00000000000000000000000000000000
0x01C4	0x01A0	j bucle3	00001011111111111111111111011000
0x01C8		nop	00000000000000000000000000000000
0x01CC		nop	00000000000000000000000000000000
0x01D0		and r4, r2, r11	00000000010010110010000000100100
0x01D4		nop	00000000000000000000000000000000
0x01D8		nop	00000000000000000000000000000000
0x01DC	0x01EC	beqz r4, desplazar3	00010000100000000000000000001100
0x01E0		nop	00000000000000000000000000000000
0x01E4		nop	00000000000000000000000000000000
0x01E8		add r3, r3, r1	00000000011000010001100000100000
0x01EC		sra r2,r2,r11	00000000010010110001000000000111
0x01F0		sll r1,r1,r11	00000000001010110000100000000100
0x01F4		nop	00000000000000000000000000000000
0x01F8		nop	00000000000000000000000000000000
0x01FC	0x01AC	beqz r2, sig3	00010000010000001111111110101100
0x0200		nop	00000000000000000000000000000000
0x0204		nop	00000000000000000000000000000000
0x0208	0x0ACC	j mrusa3	00001011111111111111111111000000
0x020C		nop	00000000000000000000000000000000
0x0210		nop	00000000000000000000000000000000
0x0214		ori r10, r0, 0x1F20	00110100000010100001111100100000
0x0218		ori r12, r0, 0x0911	00110100000011000000100100010001
0x021C		ori r11, r0, 1	00110100000010110000000000000001
0x0220		and r1, r10, r14	00000001010011100000100000100100
0x0224		and r2, r12, r14	00000001100011100001000000100100
0x0228	0x024C	j mrusa4	00001000000000000000000000100000
0x022C		sra r10, r10, r13	00000001010011010101000000000111
0x0230		sra r12, r12, r13	00000001100011010110000000000111

0x0234		nop	00000000000000000000000000000000
0x0238	0x0294	beqz r10, fuera	00010001010000000000000001011000
0x023C		nop	00000000000000000000000000000000
0x0240		nop	00000000000000000000000000000000
0x0244	0x0220	j bucle4	00001011111111111111111111011000
0x0248		nop	00000000000000000000000000000000
0x024C		nop	00000000000000000000000000000000
0x0250		and r4, r2, r11	00000000010010110010000000100100
0x0254		nop	00000000000000000000000000000000
0x0258		nop	00000000000000000000000000000000
0x025C	0x026C	beqz r4, desplazar4	00010000100000000000000000001100
0x0260		nop	00000000000000000000000000000000
0x0264		nop	00000000000000000000000000000000
0x0268		add r3, r3, r1	00000000011000010001100000100000
0x026C		sra r2,r2,r11	00000000010010110001000000000111
0x0270		sll r1,r1,r11	00000000001010110000100000000100
0x0274		nop	00000000000000000000000000000000
0x0278		nop	00000000000000000000000000000000
0x027C	0x022C	beqz r2, sig4	00010000100000011111111110101100
0x0280		nop	00000000000000000000000000000000
0x0284		nop	00000000000000000000000000000000
0x0288	0x024C	j mrusa4	00001011111111111111111111000000
0x028C		nop	00000000000000000000000000000000
0x0290		nop	00000000000000000000000000000000
0x0294		nop	00000000000000000000000000000000

A continuación, se muestra el código ensamblador del nuevo programa con la instrucción vectorial:

PC	Código ensamblador	Código máquina
0x0000	ori r7, r0, 16	00110100000001110000000000010000
0x0004	ori r1, r0, 0x1F20	00110100000000010001111100100000
0x0008	ori r2, r0, 0x0720	00110100000000100000011100100000
0x000C	ori r3, r0, 0x1F20	00110100000000110001111100100000
0x0010	sll r1, r1, r7	00000000001001110000100000000100
0x0014	sll r2, r2, r7	00000000001000111000100000000100
0x0018	ori r4, r0, 0x192C	00110100000001000001100100101100
0x001C	ori r1, r1, 0x2120	00110100001000010010000100100000
0x0020	ori r2, r2, 0x0D08	00110100010000100000110100001000
0x0024	sll r3, r3, r7	00000000011001110001100000000100
0x0028	sll r4, r4, r7	00000000100001110010000000000100
0x002C	ori r5, r0, 0x1F20	00110100000001010001111100100000
0x0030	ori r3, r3, 0x2120	00110100011000110010000100100000
0x0034	ori r4, r4, 0x020B	00110100100001000000001000001011
0x0038	ori r6, r0, 0x0911	00110100000001100000100100010001
0x003C	mult r8,r1,r2	00000000001000100100000000101111
0x0040	mult r9,r3,r4	00000000011001000100100000101111
0x0044	mult r10,r5,r6	00000000101001100101000000101111
0x0048	nop	00000000000000000000000000000000
0x004C	add r11, r8, r9	00000001000010010101100000100000
0x0050	nop	00000000000000000000000000000000
0x0054	nop	00000000000000000000000000000000
0x0058	add r11, r11, r10	00000001011010100101100000100000

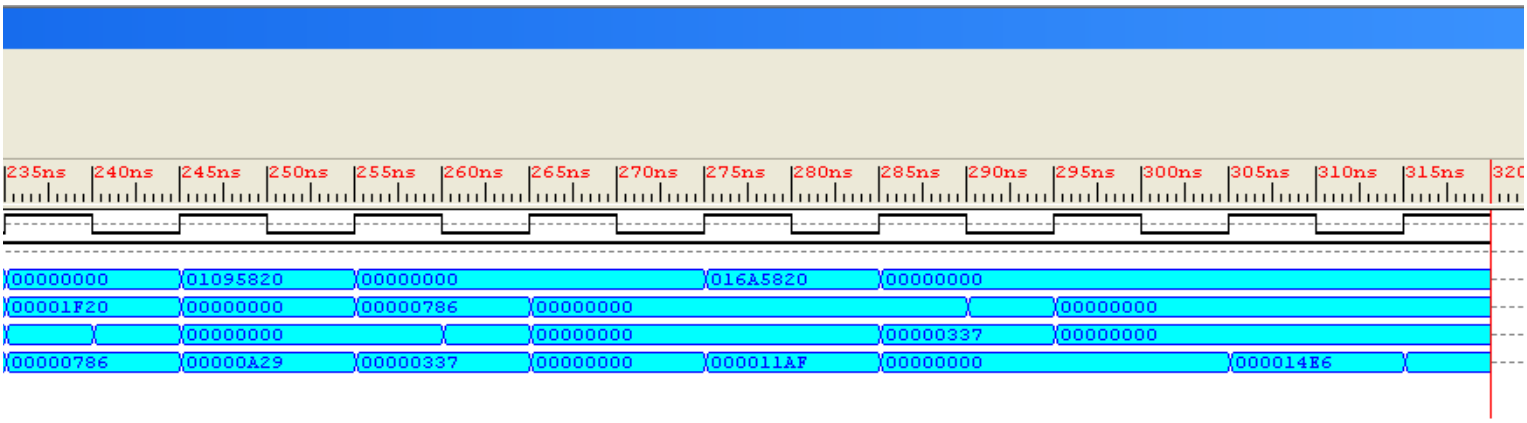
Como podemos observar, este es el programa optimizado mediante la reorganización del código para reducir las penalizaciones por dependencias de datos.

Para realizar el producto escalar de dos vectores lo subdividimos en 6 subvectores, ya que la multiplicación solo admite vectores de 4 valores como máximo. Además, para cargar cada subvector, primero cargamos los bits más significativos, los desplazamos y luego cargamos el resto de valores. Esto se debe a que la instrucción ori solo admite cargas de 16 bits.

Finalmente, llevamos a cabo las multiplicaciones y sumamos los resultados.

Simulación

Tras implementar el programa, se ha de llevar a cabo la simulación del mismo para comprobar su correcto funcionamiento siguiendo la traza a lo largo de su ejecución. Como podemos comprobar el resultado final es 0x14E6, que corresponde a 5350 en decimal.
Sabemos que se trata del resultado correcto ya que a priori sabemos el resultado del producto escalar y a su vez, podemos comprobar las entradas a la ALU para asegurarnos que los datos que cargamos son correctos.



Resultados de la síntesis e implementación

A continuación se muestran los resultados obtenidos del análisis del diseño propuesto para el programa que multiplica los vectores mediante la multiplicación rusa y el programa con la instrucción vectorial.

Resultados de la ejecución del programa sin instrucción vectorial:

Familia	XC4000XV	
Dispositivo	40110XVBG560	
	DLX32p	Main32p
Número de CLBs	719 (17%)	1012 (24%)
Frecuencia máxima	10,655 MHz	11,244 MHz

Resultados de la ejecución del programa con instrucción vectorial:

Familia	XC4000XV	
Dispositivo	40110XVBG560	
	DLX32p	Main32p
Número de CLBs	994 (24%)	1006 (24%)
Frecuencia máxima	18,01 Mhz	10,627 MHz

Como podemos observar, al añadir la instrucción de multiplicación vectorial, la frecuencia máxima del diseño ha disminuído, al igual que la complejidad del mismo, aunque no de manera significativa. A pesar de la disminución de la frecuencia, se espera que a la hora de analizar las prestaciones se observe una mejora en la instrucción vectorial ya que debería ejecutarse más rápido que el programa original.

Resultado del análisis de prestaciones

Para analizar las prestaciones de cada una de las implementaciones, se analizará la complejidad del diseño, al igual que la frecuencia máxima a la que funciona el sistema. A su vez, se realiza la comparativa del número de instrucciones que se ejecutan, los ciclos de reloj necesarios para completar la ejecución y el tiempo que se tarda en obtener el resultado.

Modelo escalar:

Familia	XC4000XV	
Dispositivo	40110XVBG560	
	DLX32p	Main32p
Número de CLBs	719 (17%)	1012 (24%)
Frecuencia máxima	10,655 MHz	11,244 MHz
Número de instrucciones	187	187
Ciclos de reloj	786	786
Tiempo de ejecución	7860 ns	7860 ns

Modelo vectorial:

Familia	XC4000XV	
Dispositivo	40110XVBG560	
	DLX32p	Main32p
Número de CLBs	994 (24%)	1006 (24%)
Frecuencia máxima	18,01 Mhz	10,627 MHz
Número de instrucciones	23	23
Ciclos de reloj	25	25
Tiempo de ejecución	250 ns	250 ns

Como se observa, el modelo escalar ha de ejecutar muchas más instrucciones que el modelo vectorial, lo cual afecta mucho al tiempo de ejecución del programa.
Si analizamos las prestaciones de las dos implementaciones, obtenemos que la implementación con la instrucción vectorial es 31,44 veces más rápida que la inicial. A su vez, por cada instrucción que realiza la implementación vectorial, se obtiene un promedio de 7,48 instrucciones del modelo escalar.