Diseño y Simulación de una Unidad de Punto Flotante con estructura Pipeline para procesadores de propósitos generales de alta performance

Eduardo Balliriain, Martín Ignacio Falcón Faya, Pablo Slavkin
Tutor: Norberto M. Lerendegui, MEE

Resúmen

En este trabajo se presenta el diseño completo y los resultados de las simulaciones de una unidad de punto flotante (FPU) compatible con la norma IEEE 754 para operaciones en simple precisión. La arquitectura de la FPU propuesta, que ha sido denominada PFPU, hace uso de un núcleo pipeline que es capaz de incrementar hasta un 600% el flujo de operaciones. Esto permitió alcanzar, en las simulaciones realizadas, velocidades de procesamiento de al menos un 50% superiores a las de sistemas comerciales de primera línea, tales como el Pentium IV 2600MHz.

El diseño de la PFPU propuesta permitiría su aplicación en un rango extenso de aplicaciones: DSPs, PCs, dispositivos portátiles (por poseer un módulo “Power-Save” para reducir el consumo entre un 48% y un 82%).

1. La Norma IEEE 754

En investigación científica y en ingeniería es muy común trabajar con magnitudes que presentan valores numéricos muy pequeños o muy grandes. Para poder operar con estas magnitudes el investigador o el ingeniero utilizan la difundida notación de punto flotante o científica, que permite representar un número real mediante una mantisa y un exponente asociado a una potencia de base 10. De este modo, la masa del electrón puede escribirse como 9,109 $*10^{-31}$ Kg. y el Número de Avogadro como 6,022 * 10^{23}, evitándose tener que utilizar para representar ambos números, 35 dígitos en el primer caso y 24 en el segundo. Este sistema de representación numérica permite abarcar un rango dinámico muy grande utilizando pocos dígitos, fundamentalmente debido al hecho de que la mantisa no es muy grande debido a que las magnitudes físicas sólo pueden medirse con una precisión acotada.

Los primeros procesadores fueron desarrollados con una capacidad matemática limitada basada fundamentalmente en operaciones de suma de enteros en simple y doble precisión. A medida que los procesadores fueron evolucionando, se les incorporó mejoras en su Unidad Aritmético-Lógica (ALU, en inglés) a los efectos de implementar instrucciones de multiplicación y división de enteros, operaciones en múltiple precisión, etc. En la difundida línea de computadoras personales (PC: personal computer, en inglés), las Operaciones de Punto Flotante (FPO: floating-point operations, en inglés) se implementaron inicialmente en un dispositivo externo al procesador conocido como Coprocesador (Ej.: Intel 8087) que operaba a partir de órdenes del Procesador del sistema (Ej.: Intel 8088). En el caso particular de las PCs, a partir del procesador INTEL 486, las operaciones matemáticas de punto flotante fueron realizadas por módulos internos de los procesadores (FPU: Floating-Point Units, en inglés)**.**

Independientemente de las arquitecturas de FPU que un particular fabricante haya diseñado para sus procesadores, resulta claro la conveniencia de establecer un sistema de codificación binario que utilice pocos bits y provea un rango dinámico suficientemente grande. Asimismo este sistema debe ser compatible con el tamaño de los buses de datos de los procesadores, generalmente de 32 ó 64 bits. Esta tarea fue realizada por William Kahan, profesor de Berkeley, Liam Kahan, y su trabajo dio origen a la norma IEEE-754 ([NRM1], [NRM2], [NRM3])**.**

Esta norma estandariza dos formatos numéricos de punto flotante; simple precisión (32 bits) y doble precisión (64 bits). Establece que un número de simple precisión consta de 3 sectores: Signo, Exponente y Mantisa (o Fracción, correspondiente a un número positivo). Estos sectores tienen una longitud de 1, 8 y 23 bits respectivamente para simple precisión, y 1, 11 y 52 bits respectivamente para doble precisión. La expresión genérica sería: $N = S\ M * 2^{EXP}$

Signo	Exponente	Mantisa (Fracción)	Precisión
1	8	23	Simple (32 bits)
1	11	52	Doble (64 bits)

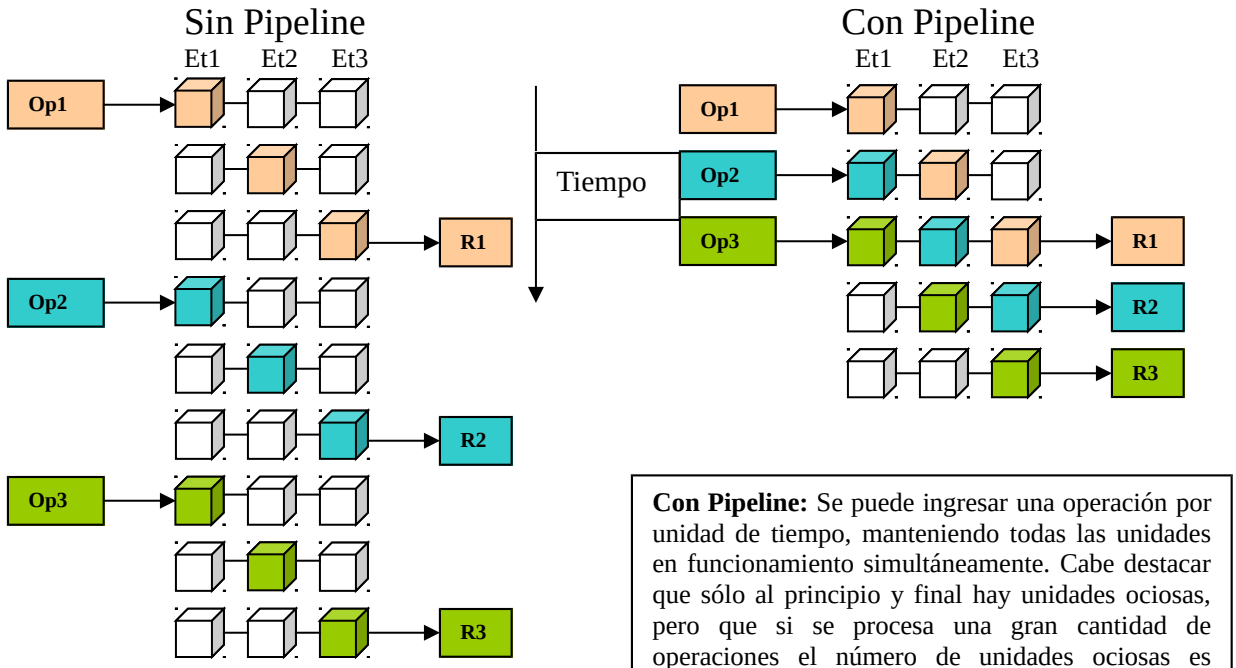
El exponente se utiliza en la modalidad ‘Exceso 128’ para simple precisión y ‘Exceso1023’ para doble precisión. Es decir, al número guardado en *Exponente* se le debe restar 128 ó 1023 para conocer el número que representa. En principio esto parecería responder a un capricho extraño del diseñador de la norma, pero, como se verá más adelante, esta decisión en la implementación permite simplificar notablemente la circuitería a implementar, reduciéndose el tiempo de ejecución y la cantidad de compuertas.

La norma IEEE-754 especifica además que todos los números están *Normalizados*, debiendo estar la mantisa comprendida entre 1 y 2 ($1 \leq M < 2$). El primer bit en el sector *Mantisa / fracción* corresponde en realidad al dígito binario inmediato siguiente al punto / coma de fracción, quedando implícito el valor 1 que precede al punto / coma. A título de ejemplo, el número:
$N = -10 = -1{,}25*2^3$ se representaría en simple precisión como:

Signo	Exponente	Mantisa	Hexadecimal (32 bits)
$1_{|2}$	$10000010_{|2} = (127+3)$	01000000000000000000000	C1200000

3. Pipeline

Uno de los grandes avances en la arquitectura de los procesadores lo constituye la incorporación de Pipelining entre etapas. Este concepto es ilustrado en la figura siguiente para tres operaciones que deben ser ejecutadas en un sistema compuesto de 3 etapas seriales.



Sin Pipeline: Una operación no puede comenzar hasta que no finaliza la operación que la precede, por lo que gran parte del hardware permanece ocioso todo el tiempo (cubos blancos).
En este ejemplo, cursar 3 operaciones involucra nueve unidades de tiempo.

Con Pipeline: Se puede ingresar una operación por unidad de tiempo, manteniendo todas las unidades en funcionamiento simultáneamente. Cabe destacar que sólo al principio y final hay unidades ociosas, pero que si se procesa una gran cantidad de operaciones el número de unidades ociosas es despreciable frente al de ocupadas. Para este ejemplo el tiempo de ejecución es de 5 unidades, es decir la mitad del tiempo sin pipeline. Si se ejecuta una gran cantidad de operaciones entonces prácticamente todas las unidades estarán ocupadas,

La técnica de pipelining consiste en identificar las etapas secuenciales involucradas en la ejecución de una instrucción y organizar el hardware de modo que todas las etapas del proceso se encuentren ocupadas, de forma de aprovechar al máximo el hardware del sistema.

3.1. Selección del número óptimo de etapas de pipeline

El presente diseño intenta maximizar la velocidad de procesamiento. La relación lineal entre velocidad y numero de etapas presentada hasta el momento implicaría implementar infinitas etapas; sin embargo en la practica aparecen 2 factores:

1) Cada vez que se agrega una etapa de pipeline hay que agregar un conjunto de celdas (latches) que permitan retener los resultados de esa etapa para ingresarlos en la próxima cuando ésta se desocupe. El tiempo que tardan estas celdas hace que la relación entre velocidad de procesamiento y numero de etapas se degrade ,como se observara mas adelante.

2)Dependiendo del lugar elegido para segmentar el sistema se deberá almacenar mas o menos líneas de datos por etapa, lo que implica una fuerte variación del costo de hardware asociado.

A continuación se muestra el análisis que determinó el número óptimo de etapas del pipeline. Se estudió la performance del sistema cuando se lo separa de 1 a 10 etapas de pipeline.

A.- Se recolectó información sobre los posibles lugares donde producir los 'cortes' para introducir latches para cada una de estas 10 posibilidades. Como los tiempos de las etapas formadas no son iguales entre sí nace el concepto de 'camino crítico' de la etapa de pipeline, que se entiende como el máximo tiempo que tarda cualquiera de las etapas de pipeline definidas. Este tiempo es el que se le asigna luego a todas las etapas por igual y se mide en tiempos de retardo de transistor (Td) para facilitar la comparación con sistemas comerciales.

B.- Se estimó la cantidad de compuertas que ocuparía cada una de las 10 posibilidades

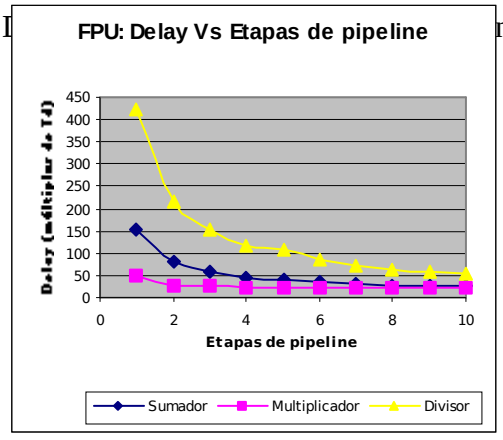
C.- Se utilizaron celdas de almacenamiento con un retardo de 10 Td.

D.- Se normalizaron los datos para poder utilizarlos en alguna fórmula de optimización del diseño (se hace que los datos estén comprendidos entre 0 y 1), obteniendo:

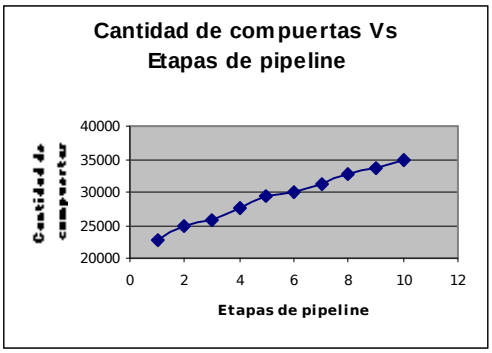
Estas curvas se obtuvieron de los datos recogidos ,que se volcaron en la tabla que sigue:

Nro de etapas de Pipeline	Operación	Delay de la Etapa Crítica	Delay del latch *(múltiplos de Td)*	Delay Total Crítico	Compuertas *(sin latches)*	Compuertas *(latches)*	Compuertas por operación	Cantidad total de compuertas
1	SUMA/RESTA	142	10	152	2840	272	3112	
	MULTIPLICACIÓN	38	10	48	6175	272	6447	22710
	DIVISIÓN	412	10	422	12879	272	13151	
2	SUMA/RESTA	72	10	82	2840	640	3480	
	MULTIPLICACIÓN	19	10	29	6175	936	7111	24879
	DIVISIÓN	207	10	217	12879	784	13663	
3	SUMA/RESTA	49	10	59	2840	1265	4105	
	MULTIPLICACIÓN	15	10	25	6175	1408	7583	25863
	DIVISIÓN	144	10	154	12879	1296	14175	
4	SUMA/RESTA	37	10	47	2840	1504	4344	
	MULTIPLICACIÓN	12	10	22	6175	2456	8631	27662
	DIVISIÓN	109	10	119	12879	1808	14687	
5	SUMA/RESTA	30	10	40	2840	2584	5424	
	MULTIPLICACIÓN	12	10	22	6175	2728	8903	29526
	DIVISIÓN	96	10	106	12879	2320	15199	
6	SUMA/RESTA	24	10	34	2840	2408	5248	
	MULTIPLICACIÓN	12	10	22	6175	3000	9175	30134
	DIVISIÓN	77	10	87	12879	2832	15711	
7	SUMA/RESTA	22	10	32	2840	2832	5672	
	MULTIPLICACIÓN	12	10	22	6175	3272	9447	31342
	DIVISIÓN	61	10	71	12879	3344	16223	
8	SUMA/RESTA	19	10	29	2840	3488	6328	
	MULTIPLICACIÓN	12	10	22	6175	3544	9719	32782
	DIVISIÓN	55	10	65	12879	3856	16735	
9	SUMA/RESTA	18	10	28	2840	3550	6390	
	MULTIPLICACIÓN	12	10	22	6175	3816	9991	33628
	DIVISIÓN	49	10	59	12879	4368	17247	
10	SUMA/RESTA	15	10	25	2840	4088	6928	
	MULTIPLICACIÓN	12	10	22	6175	4088	10263	34950
	DIVISIÓN	45	10	55	12879	4880	17759	

I nclusiones:



1.- El tiempo de las etapas de pipeline queda enteramente definido por el divisor para cualquier cantidad de etapas de pipeline que se quiera implementar. Esto es debido a que el divisor es mucho mas lento que el multiplicador y sumador/restador. Luego tanto el sumador/restador como el multiplicador tendrán tiempos ociosos mientras realizan las operaciones correspondientes.
2.- La caída del tiempo de etapa **no es lineal** respecto al aumento en la cantidad de etapas de pipeline. Esto es debido a que a medida que las etapas se hacen mas pequeñas empieza a aparecer el efecto del retardo constante que agregan las celdas (Notar en la tabla que para 10 etapas de pipeline, el 22 % del tiempo de etapa es fijado por las celdas).



3.- La cantidad de compuertas crece en forma lineal con la cantidad de etapas de pipeline a implementar, por lo que el pipeline no presenta problemas graves de escalabilidad.

Como del análisis anterior surge que el cuello de botella del sistema es el bloque Divisor, se puede pensar en aplicaciones donde no se lo requiera, típicamente en DSPs, de forma que ahora el bloque limitante sea el Sumador / restador, lo que implica que se debe considerar en los cálculos un nuevo camino critico correspondiente a dicho bloque. Llamaremos a este caso implementación en modalidad DSP.

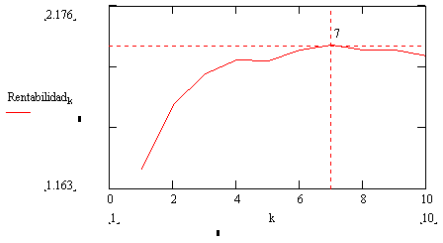
E.- A partir de las curvas anteriores se definió una fórmula (Rentabilidad) que permite obtener la cantidad de etapas óptima de acuerdo a como se prioricen la velocidad de procesamiento y la cantidad de compuertas. La formula empleada es:

$$\text{Rentabilidad}_k := \frac{1}{(1 - P)\cdot \text{Compnorm}_k + P\cdot \text{Delaynorm}_k}$$

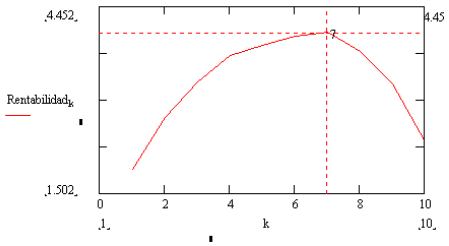
Donde el parámetro k es la cantidad de etapas de pipeline [1, 2 .. 10] y el parámetro P es el peso que se le quiere dar a la velocidad de procesamiento [0 .. 1]. Para este diseño se consideró como criterio priorizar un 60% la velocidad de procesamiento y un 40% el numero de compuertas empleadas. Luego la fórmula queda:

$$\text{Rentabilidad}_k := \frac{1}{0.4\cdot \text{Compnorm}_k + 0.6\cdot \text{Delaynorm}_k}$$

Graficando la fórmula anterior queda:



Rentabilidad PFPU Completa



Rentabilidad PFPU en modo DSP

Es evidente que la máxima rentabilidad ocurre para *7 etapas de pipeline* tanto para la PFPU como para la PFPU en modo DSP, por lo tanto se elige este valor para el diseño.

Con estos datos se ve que la frecuencia máxima posible para la PFPU será el inverso del retardo de la etapa limitante incluyendo éste un delay debido a la etapa de almacenamiento. Para fines de comparación se utilizó tecnología de 130nm con tiempos de 1.5ps de retardo de transistor, logrando una velocidad de procesamiento máxima :

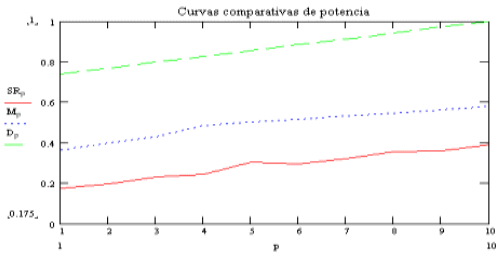
1/(Cantidad de transistores del camino critico * Td) = 9.009 GHz para la PFPU

y 26.975 GHz para la PFPU en modo DSP.

6. Potencia Disipada

Una característica importante de la PFPU es la potencia que disipa. El hecho de aumentar etapas de pipeline implica agregar compuertas que disipan energía al conmutar a la frecuencia del reloj de sistema .

Se muestra a continuación una serie de curvas que permiten ver el aumento de la potencia disipada a medida que se agregan etapas de pipeline para cada operación:



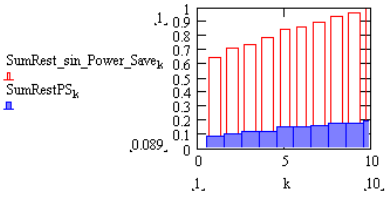
Curva normalizada respecto de la potencia del divisor
Verde (- -): Divisor
Azul (. .) multiplicador
Rojo(__) : Sumador / Restador

Se puede ver que las curvas obtenidas son prácticamente rectas con pendientes similares, por lo que no representan un límite importante para el crecimiento de la cantidad de etapas de pipeline.

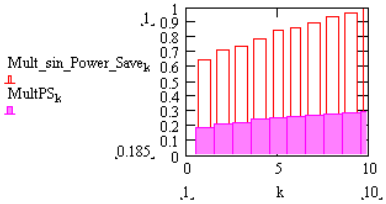
De la grafica surge la idea que si se realizara un solo tipo de operación varias veces seguidas (la misma cantidad de veces que justifican el aumento de performance al implementar el pipeline) entonces convendría 'apagar' los bloques que realizan las otras dos operaciones.

La siguiente tabla muestra valores porcentuales de ahorro de potencia cuando se ejecutan en forma independiente las operaciones correspondientes.

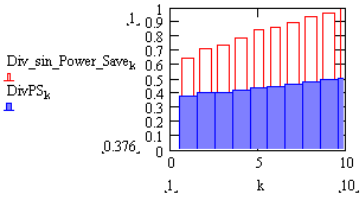
Por lo que el consumo de cada bloque con esta mejora respecto del mismo bloque sin ella para todas las posibles elecciones de pipeline queda:



Mejora del Sumador / Restador



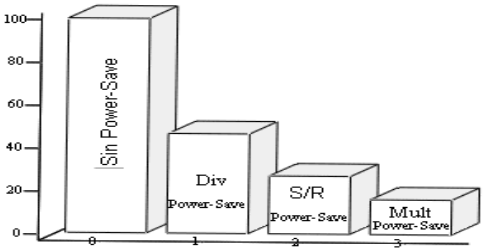
Mejora del Multiplicador



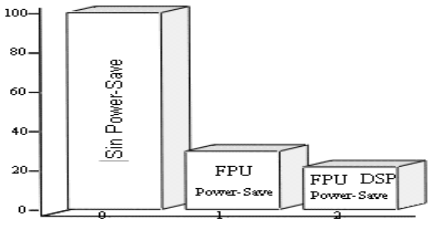
Mejora del Divisor

Para el caso concreto del presente trabajo, que utiliza un pipeline de 7 etapas, se presentan gráficas que muestran:

1.- Consumo de cada tipo de operación *en ráfaga*. Se muestra como porcentaje del consumo de la PFPU sin la implementación del modo Power-Save.



en ráfaga, ada bloque



Multiplicador

Del análisis completo, se puede ver que este bloque es el mas eficientes de los 3, gracias a que al emplear un sumador híbrido y por la geometría del algoritmo hace que sus tiempos totales sean menores comparados con los necesarios para las demás bloques. Aun así se podrían estudiar otras alternativas no consideradas como las propuestas en [ML1] y [ML2].

Divisor

Este bloque demostró ser el menos eficiente respecto del conjunto, ya que represento el cuello de botella del desarrollo. Es por eso que seria muy conveniente que se intenten nuevos algoritmos, como los propuesto en [DV1], [DV2], [DV3], [DV4], para minimizar su efecto.

Como una mejora alternativa se podría proponer que los tiempos entre etapas de pipeline no sean iguales para todas las etapas, pero esto probablemente complicaría en gran forma el hardware de sincronismo, ya que obligaría a establecer un orden dinámico en que los datos son tomados por el procesador.

Si bien el presente trabajo demuestra que el sistema propuesto funciona con una cierta performance, se debe tener en cuenta que parte del análisis se hizo suponiendo que la forma en que se implementan los latch para las etapas de pipeline es a base de una configuración de compuertas, pero se debe tener en cuenta que al momento de implementarse posiblemente el numero de transistores y los tiempos sean inferiores a los calculados, dado que existen técnicas a nivel litográfico que no fueron posible demostrar por simulación. Por lo tanto posiblemente al sistema real funcione con una performance algo mayor a la calculada.

10. Conclusiones

En este trabajo se presenta el diseño completo y los resultados de las simulaciones de una unidad de punto flotante (FPU) compatible con la norma IEEE 754 para operaciones en simple precisión. Para el alcanzar el objetivo, se plantearon algoritmos básicos para resolver operaciones de punto flotante. Dado que algunos de estos algoritmos no eran totalmente eficientes para alcanzar un gran flujo de operaciones, se ensayaron alternativas desde el punto de vista de la arquitectura para mejorar la eficiencia haciendo uso de un pipeline. Si bien este enfoque en un principio pareció incierto, haciendo un estudio comparativo de la estructuras se consiguió obtener altas performances, teniendo en cuenta el bajo costo de hardware expresado en números de compuertas.

Las simulaciones realizadas permitieron comprobar el correcto funcionamiento del diseño, y determinar que la eficacia medida concuerda con la esperada.

La PFPU propuesta en este trabajo es capaz de incrementar el flujo de operaciones hasta un 600%. Esto permitió que un sistema con bloques de operación no del todo optimizados, alcance velocidades de procesamiento al menos un 50% superiores a las de sistemas comerciales de primera línea (Pentium IV 2600MHz).

El diseño de la PFPU propuesta permitiría cubrir un rango extenso de aplicaciones, desde la integración de sistemas portátiles de bajo consumo (ya que la PFPU incluye un módulo "Power-Save" que permite disminuir la potencia disipada entre un 48% y un 82% para operaciones en ráfaga), hasta aplicaciones de alta potencia de cálculo (donde si bien el consumo de energía no hace posible la portabilidad del equipo, la relación Potencia de cálculo / Potencia disipada es muy superior a la de sistemas comerciales de primera línea,). Cabe aclarar que la comparación de la potencia total de cálculo con la de dispositivos comerciales no es estricta, ya que se compara una

PFPU con procesadores completos que tienen una cantidad mucho más elevada de funciones. No obstante, los resultados obtenidos proveen un índice suficientemente adecuado de la performance alcanzada.

*Dado que el bloque divisor es el limitante del desempeño de la PFPU, se plantea como caso especial una modalidad DSP (Digital Signal Processor), que no utiliza dicha etapa (ya que los DSP están optimizados para realizar operaciones del tipo A*B+C*D). Con esta premisa, los resultados de las simulaciones realizadas indican que la potencia de procesamiento (medida en MFLOPS) de la PFPU alcanzaría valores impensados para sistemas monoprocesadores, dado que sería equivalente a la de 8 procesadores Pentium IV 2600 MHz trabajando en paralelo.*

La siguiente tabla ilustra las características más sobresalientes del diseño realizado:

Potencia de cálculo: hata 8 veces (en modo DSP) la de un procesador Pentium IV 2600MHz utilizando la misma tecnología.
Consumo PFPU: desde 41 mW (a 1,5 GHz) hasta 233 mW (9 GHz)
DSP: desde 30 mW (a 1,5 GHz) hasta 531 mW (30 GHz)
(Referencia: Pentium IV disipa 54700 mW a 2,4 GHz)
Sistema Power-Save: desde 48% hasta 82% de ahorro de energía para operaciones en ráfaga.
Pipeline : 700% de aumento en la velocidad de operación + velocidad de conmutación de tareas muy superior a la de los sistemas operativos actuales.

11. Referencias

Sumador/Restador:

[SM1]:Peter M.Seidel and Guy Even , ***An IEEE floating Point Adder Design Optimized For Speed***, September 1, 1999.

[SM2]: Yariv Levin, ***Supporting De-normalized Numbers in an IEEE Compliant Floating-Point Adder Optimized For Speed***, July 2001

[SM3]: J.R. Hoff and G.W. Foster, ***A Full Custom, High Speed Floating Point Adder, Fermi National Accelerator Laboratory***, Fermilab-Pub-92, 1992

[SM4]: Martin Horauer, Dietmar Loy, ***ADDER SYNTHESIS***, University of Technology Vienna Insitute of Computer Technology

[SM5]: Stuart F. Oberman and Michael J. Flynn, ***A variable latency pipelined floating point adder***, Technical report, Fbruary 1996.

[SM6]:Nhon T. Quach and Michael Flynn, ***An Improved Algorithm For High-Speed Floating Point Addition***, Technical report, August 1990.

[SM7]: Nhon T. Quach and Michael Flynn, ***Design and Implementation of the SNAP Floating Point Adder***, Technical report, December 1991.

Siguiente Página

[SM8]: Reto Zimmermann, ***Binary Adder Architecture for Cell-Based VLSI and their Syntesis***, A dissertation submitted to the SWISS FEDERAL INSTITUTE OF TECHNOLOGY, Zurich, 1997.

[SM9]: Montek Singh and Steven M. Nowick, ***Fine-Grain Pipelined Asynchronous Adders for High-Speed DSP Applications***, In Proceedings of the IEEE Computer Society Annual Workshop on VLSI, April 27–28, 2000, Orlando, Florida.

Multiplicador:

[ML1]: Gary W. Bewick, ***Fast Multiplication: Algorithms and implementatio***n, A dissertation submitted to the department of electrical engineering and the committee of graduate study of Stanford University, February 1994.

[ML2]: ***ECEN 6 2 6 3 Advanced VLSI Design***, Carry Save Adder Trees in Multipliers

[ML3]:Hesham and Flynn, ***Technology Scaling Effects on Multipliers***, Stanford University.

[ML4]: Guy Even, Peter M. Seidel, ***A comparisson of three rounding algorithm for IEEE Floating Point Multiplication***, August 29, 1998

Divisor:

[DV1]: Stuart F. Oberman, Student Member, IEEE, and Michael J. Flynn, Fellow, IEEE, ***Division Algorithms and Implementations***, IEEE TRANSACTIONS ON COMPUTERS, VOL. 46, NO. 8, AUGUST 1997.

[DV2]: Stuart F. Oberman, Member, IEEE, and Michael J. Flynn, Life Fellow, IEEE, **Minimizing the Complexity of SRT Tables**, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 6, NO. 1, MARCH 1998.

[DV3]:Stuart Oberman, Nhon Quach and Michael Flynn, ***The design and implementation of a high performance floating point divider***, Technical report, January 1994.

[DV4]: Stuart Oberman, and Michael Flynn, ***An Analysis of division algorithm and implementation***, Technical report, July 1995.

FPU:

[FP2]: Stuart Franklin Oberman, D***esign Issues In High Performance Floating Point Units***, Technical Report, December 1996.

[FP3]: Oberman, Hesham and flynn, ***The SNAP Proyect: Design of Floating Point Arithmetic Units***, Stanford University.

Norma IEEE:

[NRM1]: "***IEEE Standard for Binary Floating-Point Arithmetic***", ANSI/IEEE Standard 754, 1985.

[NRM2]: DAVID GOLDBERG, ***What Every Computer Scientist Should Know About Floating-Point Arithmetic***, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto.

[NRM3]: W. Kahan, ***IEEE Standar 754 for Binary Floating Point Arithmetic, Lecture notes on status of IEEE 754***, May 31, 1996.

Low Power:

[PW1]:Etienne Jacobs, ***Using Gate Sizing to Reduce Glitch Power.***

[PW2]: Luke Seed, ***Power Consumption in Circuits***, University of Sheffield E.E.E. Department Electronic Systems Group, Low Power Circuit Design Course 25 May 2001.

Multi-Thread:

[MT1]: Burton Smith, ***"A pipelined, shared resource MIMD computer",*** Proc. 1978 Int. Conf. Parallel Processing, Aug. 1978, pp. 6-8

General:

[VR1]: William Stalling, ***Organización y Arquitectura de computadoras: Principios y aplicaciones***, Ed. Megabyte, 1995.

Software:

[SW1]: Sisoft Sandra, ***Benchmarking Software***, www.sisoftware.demon.com.uk