Diseño e Implementación de un ASIP para la generación de gráficos 2-D por computador

Jorge Agüero Zamora, Arturo Chinchilla Sánchez, Crisptofer Fernández Fernández
georgeaz56@gmail.com, mchinchilla11@gmail.com, caffernandez@estudiantec.cr
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—El presente documento trata sobre el diseño e implementación de un ASIP (Application Specific Instruction Set Processor) para la generación de gráficos 2D mediante el uso de un computador. Para ello se utiliza un procesador con *pipelining*, en el cual se utiliza como base el procesador MIPS (Microprocessor without Interlocked Pipeline Stages) de arquitectura RISC desarrollados por MIPS Technologies, con modificaciones como: reducción de instrucciones que no se utilizan, creación de nuevas instrucciones especializadas, que permiten la creación de gráficos de una manera más eficiente en términos de procesamiento y consumo de recursos, para lo cual se debe modificar la microarquitectura original. Se utiliza una memoria de vídeo, en la cual se almacenarán los datos generados (que serán la salida) y un controlador VGA para mostrar mediante una pantalla los gráficos creados al usuario.**

Por último se creó un compilador de código ensamblador a lenguaje máquina, para las instrucciones implementadas y las nuevas agregadas a la arquitectura.

***Palabras clave*—ASIP, Gráficos, Pipeline, VGA.**

I. Introducción

En la computación, la diferencia principal entre un procesador de propósito general y un ASIP (Application-specific instruction set processor) es el dominio de la aplicación que estos pueden tener. Un procesador de propósito general tiene la capacidad de abordar un gran dominio de aplicaciones, pero sin embargo su eficiencia en términos de velocidad se reduce dado que no está optimizado para realizar dichas tareas dentro de este dominio de aplicaciones. Por otra parte, un ASIP tiene un dominio de aplicaciones limitado, pero está especializado en cumplir tareas específicas, lo cual tiene ventajas con respecto a tiempo de procesamiento, consumo de espacio y energía; un procesador especializado en una tarea tiene mayor desempeño en esta que un procesador de propósito general [1]. El problema concreto para resolver se basa en la aplicación de un ASIP para generación de gráficos en dos dimensiones por computador, para lo cuál es necesario diseñar e implementar en hardware el dispositivo basado en la arquitectura MIPS, además de esto, se debe implementar una interfaz de salida para los gráficos generados por el ASIP y un compilador especializado para el set de instrucciones asociado y definido a este.

Para la generación de gráficos se cuenta con una memoria de video externa e instrucciones especializadas que sirven para pintar y navegar en esta, además se implementa el algoritmo de Bresenham que sirve para pintar figuras básicas en la memoria de video. En este articulo se mostrarán algunas referencias a trabajos y resultados previos utilizando ASIP, esto para tener una linea base relativa, además se realiza la propuesta conceptual de la solución al problema para que pueda ser reproducible, también se analizaran y discutirán los resultados obtenidos mediante simulaciones y pruebas físicas sobre el sistema implementado.

II. Antecedentes

En [2] los autores implementan un ASIP para el filtrado en tiempo real de imágenes y vídeos mediante un filtro Retinex, esto porque ellos mencionan que un ASIP es flexible para un dominio especifico de aplicaciones específico y además es seleccionado por su rendimiento, flexibilidad y criterios físicos como el área y el consumo de energía. Para los autores, en la teoría Retinex una imagen es producto de la iluminación ambiental y la reflectancia de los objetos en la escena, estas variables son procesadas independientemente. Los objetivos específicos de esta implementación según los autores incluye la corrección de imágenes y vídeos adquiridos en malas condiciones de luz, mejoras de contraste y el control dinámico de sensores logarítmicos.

Según el los autores en [3] entre los Circuitos integrados de aplicaciones especificas y los procesadores programables de propósito general existe una brecha en términos de rendimiento, potencia, costo y flexibilidad y mencionan que los ASIP unen esta brecha. En este proyecto, los autores tratan de demostrar los beneficios clave de la implementación de un ASIP para aplicaciones de video. Además, se basan en el diseño de una plantilla ME, que es útil para varias aplicaciones de video como codificación de video, detección de obstáculos, conversión ascendente de imagen, conversión de video 2-D a 3-D. El ASIP se basa en una plantilla de procesador de palabras de instrucción muy largas (VLIW) y cumple con los requisitos de bajo consumo de energía y bajo costo, y proporciona la flexibilidad necesaria para el dominio de la aplicación.

III. Propuesta

III-A. Arquitectura

La base del ASIP es arquitectura MIPS, entonces a partir de esto se tomó información de la hoja de datos de MIPS para set de instrucciones del procesador MIPS, las cuales son:

Add, Add Immediate, And, Branch On Equal, Branch On Not Equal, Jump, Jump And Link, Jump Register, Load Word, Nor, Or, Set Less Than, Set Less Than Immediate, Shift Left, Logical, Shift Right Logical, Store Word y Substract. Esto para ahorrar trabajo y no "reinventar la rueda". Para que realmente el procesador sea un ASIP, es necesario especializarlo para una tarea específica que en este caso es la generación de gráficos en 2D. El sistema solo operará con dos tipos de datos, enteros y píxeles, estos tienen un tamaño de 32 bits o 4 bytes, pero los píxeles están organizados en formato RGB de la siguiente forma: R[23:16],G[15:8],B[7:0]. Donde el espacio faltante para completar 4 bytes es padding. Cabe destacar que el Byte Ordering de la arquitectura es BigEndian.

También se menciona el renombramiento de registros para propósitos definidos como por ejemplo el pixel-pointer ($pp), frame buffer pointer ($fp), vertical resolution($vr) y horizontal resolution ($hr). Teniendo estos registros, es posible definir las instrucciones especializadas necesarias para navegar a través de memoria de vídeo y pintar píxeles sobre esta, las cuales pueden observarse en el cuadro III-A, también se definieron dos pseudoinstrucciones que permiten pintar figuras más grandes como puntos y puntos medianos, estas también pueden observarse en el cuadro III-A.

Cuadro I
CUADRO DE INSTRUCCIONES Y PSEUDOINSTRUCCIONES AGREGADAS AL ASIP

Instrucción	Mnémonico
Move pixel pointer	mpp
Move pixel pointer immediate	mppi
Paint Pixel	ppxl
Paint and Move up	ptmu
Paint and Move left	ptml
Paint and Move Down	ptmd
Paint and Move Rigth	ptmr
Paint Pixel with color register	ppxlc
Paint and Move Pixel	pmpxl
Small point	spnt
Medium Point	mpnt

Las pseudoinstrucciones están compuestas por rutinas de instrucciones más simples las cuales pintan pixel por pixel hasta formar las respectivas figuras.

III-B. Micro-arquitectura

En la figura 1, se esboza un diagrama de tercer nivel de la solución en hardware del problema.

En la figura 1 puede destacarse el uso de dos memorias distintas, memoria de datos para guardar datos de procesamiento u otros y la memoria de vídeo, que es donde el controlador VGA tomará la información para mostrar en pantalla y en donde las instrucciones especializadas operarán para realizar figuras y demás.

Para la implementación especifica del ASIP, como se mencionó anteriormente, se utiliza como base una arquitectura MIPS reducida, esto quiere decir que no cuenta con todas las instrucciones que esta arquitectura posee, además de esto,

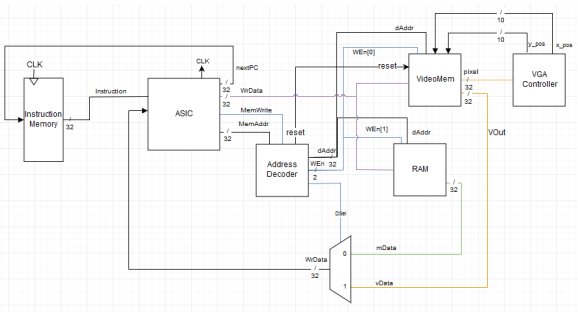

Figura 1. Diagrama de tercer nivel de la implementación en hardware de la solución.

cuenta con pipeline de 5 etapas con adelantamiento implementado, esto para reducir las penalizaciones que puedan ocurrir por riesgos de datos. Sin embargo, igual se tiene que recurrir a stalls en algunas ocasiones por lo que además de unidad de adelantamiento, se cuenta con unidad de stall.

Para la ejecución de las nuevas instrucciones fue necesario agregar módulos extras en la etapa de decodificación de la instrucción, esto para poder realizar multiplicaciones, sumas y shifts lógicos hacia la izquierda antes de la etapa de ejecución, dado que en esta etapa, se emplea la ALU para el calculo de direcciones a memoria donde se pintará o se modificará un píxel dado.

El set de registros cuenta con salidas especiales para los registros $hr, $cr y $fp, esto para agilizar la ejecución de las instrucciones especiales porque así puede operarse usando el valor de más registros a la vez. Cabe destacar tambien que la unidad de control cuenta con 17 señales de control y la mayoría de estas deben moverse a etapas posteriores por lo que los registros del pipeline también cuentan con entradas y salidas para guardar estas señales de forma temporal, en la figura 2 puede observarse el diagrama de bloques de la microarquitectura.

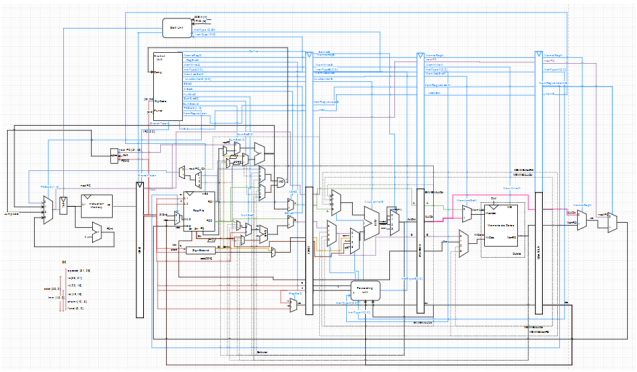

Figura 2. Diagrama de bloques de la microarquitectura.

Se creó un nuevo compilador, que realice el proceso de traducción del código ensamblador a lenguaje máquina (binario y/o hexadecimal) correspondiente para a las instrucciones y así poder cargarlas en memoria para su posterior ejecución. Este

programa fue implementado utilizando el lenguaje de alto nivel Python, con la ayuda de Lex y Yacc. Lex para la generación de analizadores léxicos y Yacc como analizador sintáctico. El proceso consiste en analizar la cadena de caracteres (asm) e ir traduciendo a su binario/Hexadecimal correspondiente.

Se implementó una rutina en lenguaje ensamblador equivalente al algoritmo de Bresenham con la cual se pueden trazar líneas rápidamente sobre la memoria de vídeo, en cualquier dirección y de tamaño variable.

IV. Resultados

El proyecto quedó inconcluso para el tiempo asignado. A pesar de que se hizo todo el diseño y los módulos de HDL fueron completados, se logró llegar hasta la etapa de simulación del diseño, donde se encontraron diferentes problemas de funcionamiento o de conexiones. El arreglo de estos problemas tardó más tiempo de lo esperado y no se logró poder quemar el diseño en la FPGA.

V. Conclusiones

No podemos dar conclusiones válidas de un proyecto inconcluso.

Algunas observaciones del porqué no se logró fue a los constantes cambios que se descubría que se debían tomar y como para incluir estos cambios y probarlos se debían modificar muchos archivos de testbench para comprobar la funcionalidad. Al final, el obviar estos cambios causo más errores en las etapas de validación.

Python en conjunto con Lex y Yacc proporcionaron un ambiente sencillo para la implementación del compilador a código máquina.

Referencias

[1] D. Liu, *Embedded DSP processor design: Application specific instruction set processors*. Elsevier, 2008, vol. 2.

[2] S. Saponara, L. Fanucci, S. Marsi, G. Ramponi, D. Kammler, and E. M. Witte, "Application-specific instruction-set processor for retinex-like image and video processing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 7, pp. 596–600, 2007.

[3] H. Peters, R. Sethuraman, A. Beric, P. Meuwissen, S. Balakrishnan, C. A. A. Pinto, W. Kruijtzer, F. Ernst, G. Alkadi, J. Van Meerbergen *et al.*, "Application specific instruction-set processor template for motion estimation in video applications," *IEEE transactions on circuits and systems for video technology*, vol. 15, no. 4, pp. 508–527, 2005.