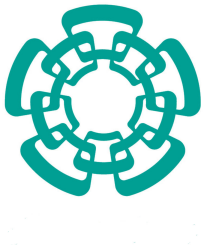


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Computación

Soporte en hardware para el control de acceso de dispositivos de E/S en un ambiente virtualizado

Tesis que presenta:

Brisbane Ovilla Martínez

Para obtener el grado de:

Maestro en Ciencias en Computación

Director de la Tesis:
Dr. Arturo Díaz Pérez

Cd. de México, Distrito Federal, México. Diciembre, 2009

Esta investigación fue parcialmente apoya mediante el proyecto No. 51623 del Fondo Mixto Conacyt-Gobierno del Estado de Tamaulipas.

This research was partialy funded by project number 51623 from 'Fondo Mixto Conacyt-Gobierno del Estado de Tamaulipas'

La tesis presentada por Brisbane Ovilla Martínez fue aprobada por:

Dr. Luis Gerardo de la Fraga

Dr. Cesar Torres Huitzil

Dr. Arturo Díaz Pérez, Director

Cd. de México, Distrito Federal, México., 16 de Diciembre de 2009

Hay hombres que luchan un día y son buenos.
Hay otros que luchan un año y son mejores.
Hay quienes luchan muchos años y son muy buenos.
Pero hay los que luchan toda la vida:
esos son los imprescindibles.

Bertolt Brecht

Para ti mamá, que has sido un ejemplo de lucha y siempre has estado a mi lado a pesar de la distancia. Te amo eres lo más grande que Dios me ha dado.

Agradecimientos

- A mi papá, mis hermanos, mis abuelitas, mis tíos y primos mil gracias por siempre confiar en mí y estar conmigo incondicionalmente, los quiero mucho.
- Gracias a mis amigos por ser mis pilares, sus porras siempre fueron de gran ayuda.
- Al Dr. Arturo Díaz por hacerse el tiempo para orientarme en el trabajo de tesis, mostrándome siempre que la investigación debe hacerse con gusto y responsabilidad.
- A todos mis compañeros del CINVESTAV-Zacantenco por brindarme su amistad.
- A todas aquellas personas que me brindaron su cariño durante mis estancias en Guadalajara y Cd. Victoria.
- Gabriel te agradezco tu apoyo incondicional en las buenas y en las malas, sin los momentos de sonrisas contigo no hubiera sido posible terminar este ciclo.
- Angelina gracias por ser mi cómplice en todas travesuras en Cd. Victoria ;).
- Pedro gracias por todo lo que has hecho por mí, por darme la oportunidad de conocerte y por permitirme ser parte de tu vida.
- Gracias a Amalia y Victor por recibirme en su casa durante mi estancia en Tamaulipas y hacerme sentir parte de su familia
- Gracias a Sofy por apoyarme siempre, ya sea de cerca o de lejos.
- Irasema y Fabiola gracias por las largas platicas que me otorgaron,
- Al CONACYT por el apoyo económico para la realización mis estudios de maestría

Índice General

Índice de Figuras

Índice de Tablas

Resumen

Soporte en hardware para el control de acceso de dispositivos de E/S en un ambiente virtualizado

por

Brisbane Ovilla Martínez
Maestro en Ciencias del Departamento de Computación
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2009
Dr. Arturo Díaz Pérez, Director

En la actualidad las computadoras han adquirido gran capacidad de procesamiento lo cual ha permitido ejecutar aplicaciones cada vez más complejas y costosas en recursos. Sin embargo, es importante considerar que estas capacidades muchas veces no son aprovechadas al máximo, es por esta razón que en los últimos años han surgido nuevas tecnologías que permiten aprovechar de mejor manera los recursos de cómputo.

Una de estas herramientas es la virtualización, que en términos computacionales, es un método para crear una versión virtual de un dispositivo o recurso mediante el uso de las llamadas Máquinas Virtuales (MV). Las MV nos permiten una combinación y repartición más eficiente de los recursos de cómputo con la finalidad de ofrecer una mayor versatilidad en la capacidad de resolución de tareas como el soporte a múltiples sistemas operativos. Debido al gran aumento de poder de cómputo en la actualidad resulta más ventajoso tener soporte de máquinas virtuales que tener físicamente máquinas separadas. Ya que algunos estudios afirman que los servidores normalmente no trabajan utilizando el $100\,\%$ los recursos, sino alrededor de un 10 a un $20\,\%$. Es en estos casos que mediante el uso de la virtualización permite aprovechar hasta un $80\,\%$ la capacidad del CPU y sus recursos. Entonces el virtualizar una plataforma permite lograr un mejor aprovechamiento del hardware, así como también reducir el consumo de energía y espacio y facilitar la administración gracias a la reducción del número de servidores físicos.

En años recientes una línea de investigación ampliamente abordada es la seguridad en ambientes virtualizados, ya diferentes VM's pueden tener acceso a dispositivos de hardware en un mismo entorno físico. Las labores de coordinación de acceso a los recursos de hardware son hechas por un software llamado monitores de máquinas virtuales (MMV), algunas de las versiones existentes de MMV's ocupan un módulo de control de acceso (MCA) para hacer más eficiente los accesos a los recursos físicos. Sin embargo, existen dos grandes problemas en las MMV's uno de ellos es el mal manejo de la seguridad ya que en ocasiones, por la comunicación entre VM's ocurren accesos no autorizados a recursos los físicos como la memoria. El otro problema es que al virtualizar algunas aplicaciones como sistemas operativos su rendimiento decrece, principalmente a debido a que el MMV debe regular las peticiones de las MV al hardware.

Una solución a estos dos problemas es la implementación del MCA directamente en hardware, y es que realizar la transferencia de control de acceso mediante software a una implementación hardware se traduce en una menor carga de trabajo al MMV y por consiguiente en una mejora en el rendimiento. Además la seguridad también se ve beneficiada debido a que el control de acceso a los dispositivos se realiza a en un nivel más bajo que el software haciendo más complicado lograr un acceso no autorizado a los dispositivos físicos. En este documento se plantean los principios de diseño del módulo de control de acceso en hardware. Los requerimientos y objetivos que deben ser cubiertos por el MCA son descritos, al igual que la arquitectura del MCA en hardware y las consideraciones que fueron tomadas para su integración a una arquitectura de cómputo.

Abstract

Soporte en hardware para el control de acceso de dispositivos de E/S en un ambiente virtualizado

by

Brisbane Ovilla Martínez
Master of Science from the Information Technology Labotory
Research Center for Advanced Study from the National Polytechnic Institute, 2009
Dr. Arturo Díaz Pérez, Advisor

Nowadays computers have acquired a huge processing capacity, which has allowed to create much more complex applications. However, something important to consider is that the capacities are often not completely used, as a consequence new technologies have been created to improve the usage of the computational resources.

One of this applications is the virtualization, which in computational terms, is a method to create a virtual version of a hardware device by using the so called Virtual Machines (VM). The VM allows a more efficient combination and assignment of the hardware resources aiming to offer a better versatility in the task solving capacities, such as supporting multiple operative systems. Due to the increment of computational capacities, today it is more convenient having support of VM rather than physically separated computers. Moreover, there are some research which affirms that typically servers don't use a $100\,\%$ of the whole hardware resources but just about a $10\,\%$ to $20\,\%$ of them. It Is in these cases where virtualizing allows to improve the performance by using up to $80\,\%$ of the CPU capacities, including its resources. Therefore, virtualizing allows not only to improve the usage of the hardware, but also to reduce the energy and space consumption besides, it the resources administration by decrementing the number of physical servers.

To take the best of physical resources, multiple VM's can run in the same computer, they also communicate and share the computer resources between them, and as a result, security has become an important research field. The access coordination tasks to hardware are done by a virtual machines

monitor (VMM), some of them use a control access module (CAM) to make more efficient the access to the physical resources. However, there are two main issues with the VMM's, one of them is the security, due to communications between VM's not allowed access to physical resources like memory are likely to occur. The other issue is that virtualizing applications like operative systems reduce their performance, mainly because VMM has to regulate the hardware petitions of the VM.

A plausible solution to these two problems is the implementation of the CAM directly into hardware, taking the hardware access control away from the VMM results into a reduction of tasks demands, and consequently into a improvement in the performance. Besides, the security is also improved since the hardware control access is done in a lower level than in software and so difficulting the not allowed access to the hardware devices. This document states the design principles of a CAM in hardware and describes the requirements and objectives that must be full filled by the CAM, as well as the architecture in hardware and the considerations that were necessary to its integration into a computer architecture.

Nomenclatura

AES	Advantage Encription Standard.
ACL	Access Control List.
ACP	Access Control Policy.
AMD	Advanced Micro Devices.
CAM	Content Addrees Memory.
DES	Data Encription Standard.
DMA	Direct Memory Access.
E/S	Entrada/Salida.
FPGA	Field Programmable Gate Array.
FSL	Fast Simplex Link.
GPA	Guest Physical Address.
HPA	Host Physical Address.
ISR	Interrupt Service Routine.
IOMMU	Input/Output Memory Manage Unit.
JVM	Java Virtual Machine.
LBBRAM	Lockable Battery-Backed Random Access Memory.
LMB	Local Memory Bus.
MEF	Máquina de Estados Finitos.
MCA	Módulo de Control de Acceso.
MCA-RAM-4	Módulo de Control de Acceso implementado con memoria RAM y 4 procesos de análisis.
MCA-CAM-n	Módulo de Control de Acceso implementado con memoria CAM.
MMV	Monitor de Máquinas Virtuales.
MV	Máquina Virtual.
PIC	Por sus siglas en inglés Programmable Interrupt Controller.
PKI	Infraestructura de Llave Pública.
PLB	Processor Local Bus
PP	Plataforma de Pruebas.
OPB	Por sus siglas en inglés On-Chip Peripheral Bus.

RAM	Random Access Memory.
RTL	Register Transfer Language.
RBAC	Role-based Access Control.
SO	Sistema Operativo.
SOA	Sistema Operativo Anfitrión.
SoC	System on Chip.
SOH	Sistema Operativo Huésped.
TI	Tecnología de la Información.
TLB	Translation Lookside Buffer.
TVMM	Trusted virtual machine monitor.
FPU	Float Point Unit .
XCL	Xilinx cacheLink.

1

Introducción

1.1 Antecedentes

Aunque hoy en día la virtualización es un tema con mucho auge los inicios de esta tecnología datan desde la década 1960, en esos años IBM diseño un computadora específicamente para la virtualización llamada IBM S/360 modelo 67, aunque su uso se mantuvo sólo 20 años y después se creía totalemnte desaparecido el concepto de virtualización. Pero en los años de lo 1990 volvió a resurgir la virtualización de plataformas y de muchos más recursos computacionales [4].

La virtualización se trata de una técnica que permite crear versiones lógicas de dispositivos o recursos computacionales, las cuales puede ser ocultadas a otros sistemas, aplicaciones o usuarios que interactúen con ellos [30]. Es posible hacer ver que un recurso físico, como un sistema operativo, un dispositivo de red, un dispositivo de almacenamiento o un servidor, aparezca como si se trataran de varios recursos lógicos a la vez o por el contrario, que un grupo de servidores, dispositivos de almacenamiento o dispositivos de red, parezcan como si fueran uno solo. Existe diferentes tipos de virtualización, como por ejemplo: virtualización de recursos, de sistema operativo, de plataforma.

En particular la virtualización de plataforma se trata de la simulación de una máquina real junto

con todos sus recursos. Esto es posible gracias al uso de una aplicación de software que permite hacer la separación del hardware y el software, lo cual posibilita a su vez que múltiples sistemas operativos, aplicaciones o plataformas de cómputo se ejecuten simultáneamente en un solo servidor o PC. Dentro de este esquema caben la mayoría de las formas de virtualización más conocidas, incluidas la virtualización de sistemas operativos, la virtualización de aplicaciones y la emulación de sistemas operativos [1].

La aplicación de software encargada de la separación entre el hardware y el software de una máquina real se le conoce hipervisor o MMV (Monitor de Máquinas Virtuales). El MMV se trata de un sistema operativo que se encarga de administrar los recursos de la máquina física (CPU, Memoria, Red, Almacenamiento) y presentárselos a cada MV una visión del hardware que le haga creer que cada una es dueña única del hardware físico del sistema.

Como todos los sistemas operativos, un MMV carga aplicaciones. En el 00caso de los MMV las aplicaciones que sobre él ejecutan son otros sistemas operativos. El objetivo inicial del MMV es ejecutarlos al mismo tiempo compartiendo recursos disponibles en la plataforma. Actualmente los MMV se han vuelto muy complejos debido a que tienen que llevar a cabo tareas adicionales, como la seguridad de los datos o recursos de cada MV.

A pesar que la virtualización parece tener grandes ventajas para el aprovechamiento de los recursos de cómputo tiene como desventaja que el tiempo de ejecución de una VM es mucho más lento que el de una máquina ejecutándose sin virtualización [20], una simple instrucción en un ambiente virtualizado se puede traducir a miles de instrucciones reales.

Una de las tareas que más afecta el rendimiento de un MMV es el manejo de E/S a los periféricos del sistema físico. Esta pérdida de rendimiento se ve incrementada con técnicas de virtualización de emulación o interpretación para los dispositivos de entrada/salida (E/S). Con el fin de disminuir y apoyar a los MMV en la virtualización, los principales fabricantes de procesadores han estado trabajando en desarrollar un soporte arquitectural en hardware que cumpla esta tarea, en años recientes han sido incorporadas a computadoras de uso común, Intel (INTegrated ELectronics)[19] y AMD (Advanced Micro Devices)[2] cuentan en sus procesadores con hardware de soporte para virtualizar la plataforma. El soporte en hardware se ha dado principalmente para las siguientes

plataformas: virtualización de CPU, dispositivos de E/S y algunos dispositivos específicos.

1.2 Motivaciones

El uso de la virtualización de plataforma va en aumento día con día sobre todo en empresas de tecnología de la información (TI), en donde se desea que la virtualización ofrezca beneficios económicos y tecnológicos. Por ello hoy en día se está investigando como lograr que el tiempo de ejecución de una MV no difiera tanto con la de una máquina no virtualizada. Una de las soluciones encontradas es aligerar el trabaja al MMV, por ejemplo: los principales fabricantes de procesadores como Intel y AMD ofrecen hardware que facilita la virtualización del CPU, dispositivos de E/S y otros dispositivos específicos, lo cual permite quitarle carga de trabajo al MMV.

Otra de las tareas que realiza el MMV y que degrada su rendimiento es la aplicación de seguridad en el sistema virtualizado. Las tecnologías de hardware actuales pueden realizar la separación física de los recursos del hardware pero no cuentan con esquemas de compartición de recursos entre las MV.

Es de vital importancia proteger la información y los procesos que ejecuta cada MV, contra el robo y la manipulación. El control de acceso, monitores de referencia o autentificación de dispositivos, suelen ser medidas de seguridad [31]. Actualmente existen arquitecturas de seguridad para ambientes virtualizados[15, 32]. Este control se realiza a nivel de software dentro del núcleo de hipervisor, estableciendo algunas políticas de seguridad y de control de acceso.

1.3 Planteamiento del problema

A pesar que estas tecnologías pueden realizar la separación física de los recursos del hardware, actualmente no siempre es necesario tener un aislamiento puro de cada MV. Por el contrario es deseable que las MV puedan compartir recursos entre sí. Para poder llevar a cabo lo anterior es necesario establecer esquemas de seguridad que controlen la interacción entre las MV y sus recursos.

Existe arquitecturas de seguridad para ambientes virtualizados [15] [32]. Pero este control es a

nivel de hardware dentro del núcleo de hipervisor.

Con el propósito de que el soporte de seguridad para ambientes virtualizados no contribuya a una pérdida en el rendimiento de las aplicaciones, es necesario proponer un mecanismo de hardware dedicado al control de acceso para dispositivos de E/S para arquitecturas virtualizadas que se encargue de forma eficiente de las operaciones en las que el software consume mucho tiempo de ejecución.

De lo anterior nos surge la pregunta **¿Es posible agregar un MCA en hardware que no degrade el rendimiento del sistema y que permita compartir dispositivos de E/S de forma segura entre las MVs?**

En este trabajo de tesis se propone diseñar e implementar en hardware un MCA (Módulo de Control de Acceso), es decir, un módulo que sirva de apoyo al MMV para facilitar el control de acceso las MV, implementándolo en hardware reconfigurable. Este módulo debe estar situado entre el procesador y los dispositivos de E/S con el fin que sirva de árbitro. El MCA será implementado con una lista de control de acceso (ACL por sus siglas en inglés *Access Control List*). El MCA interpretará cada política incluida en la ACL y manejará la decisión de control de acceso.

1.4 Objetivo de tesis

El objetivo principal de esta tesis es diseñar e implementar un módulo en hardware para el control de acceso dispositivos de E/S entre máquinas virtuales. Para alcanzar el objetivo principal, fueron definidos algunos objetivos particulares:

1. Desarrollar un módulo en hardware para el control de acceso de dispositivos de E/S implementándolo en un FPGA (por sus siglas en ingles Field Programmable Gate Array) para ambientes virtualizados.

2. Hacer la implementación de un procesador en FPGA e integrarlo con el módulo de control de acceso antes diseñado.

3. Desarrollar una plataforma de pruebas en una plataforma de FPGA con el fin de poder validar los resultados.

1.5 Metodología

En la figura 1.1 se presenta la metodología que se siguió para alcanzar los objetivos de este trabajo.

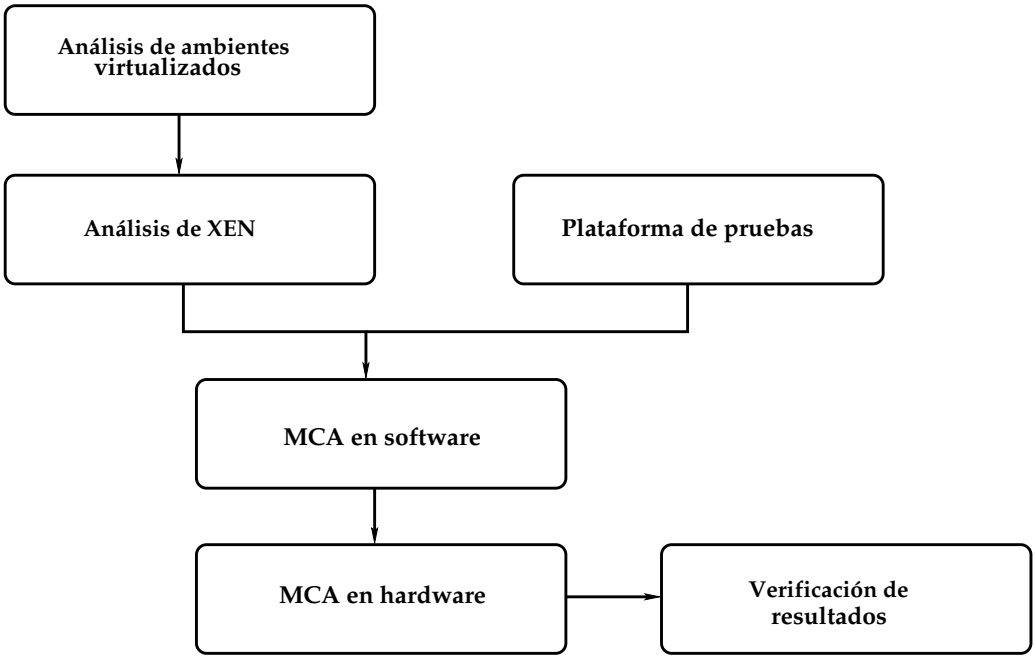


Figura 1.1: Esquema de la metodología seguida en la tesis.

1. **Análisis de ambientes virtualizados**. Se realizó una revisión de la literatura relacionada con virtualización y control de acceso a dispositivos de E/S. Lo importante de este punto fue conocer teóricamente como es llevada a cabo la virtualización y sus parámetros de desempeño, definir las principales demandas de seguridad en un sistema virtualizado y las características básicas para un módulo de control de acceso.

2. **Análisis de un hipervisor de la literatura**. Se instaló el hipervisor XEN al núcleo de Linux con le fin de analizar el comportamiento de las máquinas virtuales y poder también definir parámetros de desempeño. Se implementó el supervisor de seguridad de XEN y se aplicaron políticas de seguridad.

3. **Plataforma de pruebas**. Fue diseñada e implementada en una tarjeta de desarrollo de FPGA con 5 periféricos de E/S, esto con el fin de poder realizar la verificación de resultados.

4. **MCA en hardware**. A partir del diseño en implementaciones en software de módulos de control de acceso se realizó la descripción del módulo en RTL (por sus siglas en inglés Register Transfer Language) e integración a la plataforma de pruebas.

5. **Verificación de resultados**. Se evaluaron y obtuvieron los resultados del módulo de control, comparando la implementación en software contra la de hardware.

1.6 Organización de la tesis

La tesis está organizada de la siguiente manera: El capítulo 2 se hace una introducción al concepto de virtualización describiendo los modelos principales de virtualización, así como los métodos utilizados para compartir los recursos de E/S. En el capítulo 3 se explora el concepto de seguridad en plataformas virtualizadas, vulnerabilidades de seguridad, así como, la manera en la que le hacen frente a estos problemas. El capítulo 4 describe el diseño general del módulo de control de acceso en hardware, presentado los objetivos de seguridad que cubrir el módulo y cuáles serían sus limitantes. Para finalizar, en el capítulo 5 se presentan los resultados comparativos de MCA en hardware contra un MCA en software ejecutados en nuestra plataforma de pruebas.

2

Virtualización

En términos de computación, virtualización es un método para crear una versión virtual[1] de un dispositivo o recurso. Un hecho tan sencillo como particionar un disco es considerado una virtualización. En este capítulo se presenta una revisión breve de virtualización y su estado del arte.

Existen diferentes tipos de virtualización, como por ejemplo:

- Virtualización de recursos
- Virtualización de sistema operativo
- Virtualización de plataforma

La virtualización de recursos es básicamente la combinación de múltiples recursos con la finalidad de obtener un recurso de mayor capacidad o bien varias entidades virtuales con las mismas características. La segmentación de una red o partición lógica de una única red física es un claro ejemplo de este tipo de virtualización.

[1] Virtual: existe solo aparentemente y no es real.

La virtualización de aplicaciones convierte las aplicaciones en servicios virtuales gestionadas y administradas por un entorno virtual que actúa como una capa entre la aplicación y el sistema operativo (SO), y elimina los conflictos con otras aplicaciones o con el SO. Ofrece compatibilidad y portabilidad entre distintos SO y arquitecturas. Algunos API[2] que realizan la virtualización de aplicaciones son por ejemplo: JVM [35] (Java Virtual Machine), Wine [40] (Wine Is Not an Emulator), etc. JVM es una máquina virtual, es decir una arquitectura de computadora desarrollada por Sun Microsystems que sirve como intérprete del código que produce un compilador de Java. La ventaja es que el ejecutable compilado en cualquier arquitectura puede ejecutarse en cualquier otra arquitectura siempre que cuente con el intérprete de JVM.

2.1 Virtualización de plataforma

Virtualización de plataformas se trata de la abstracción de todos los recursos de una plataforma mediante software de control conocido como anfitrión o "host", que simula un entorno computacional para su software huésped o "guest". Este software huésped, que generalmente es un SO, se ejecuta en la plataforma como si fuera el único ejecutándose en ella. Debido al gran aumento de poder de cómputo de las computadoras actuales resulta más ventajoso tener soporte de máquinas virtuales que tener físicamente máquinas separadas. Los servidores nunca trabajan utilizando a un 100 % los recursos, de hecho el porcentaje de uso es de 10-20 %, con virtualización se puede aprovechar hasta un 80 % la capacidad del CPU y sus recursos. Virtualizar una plataforma permite lograr una utilización de los recursos significativamente alta, así como, en el ahorro de energía, espacio y facilita la administración debido que se reduce el número de servidores físicos. Las máquinas virtuales son totalmente independientes entre sí. Por lo tanto, algún fallo en un MV no afecta a las otras. Además cada MV tiene un acceso privilegiado (administrado) independiente. De tal forma que un ataque de seguridad a una MV solo afectara a dicha máquina, en teoría.

En este caso el tipo de virtualización en que estamos interesados es la de plataforma [30].Esta es una tecnología que fue desarrollada por IBM en los años 60s. La primera computadora diseñada específicamente para virtualización fue el mainframe IBM S/360 Modelo 67. Esta característica de

virtualización ha sido un estándar de la línea que siguió (IBM S/370) y sus sucesoras, incluyendo la serie actual.

Durante la década 1960 y 1970 fueron muy populares pero las máquinas virtuales desaparecieron prácticamente entre los años 1980 y 1990. Hasta el final de la década 1990 volvió a resurgir la tecnología de las máquinas virtuales y no solamente en el área de servidores sino también en muchas otras áreas del mundo de la computación.

Las máquinas virtuales son de gran utilidad ya ofrecen ciertas ventajas con respecto a las implementaciones en un servidor físico. Dentro de estas ventajas podemos considerar:

- Particionamiento, esta característica permite que diferentes procesos puedan correr de manera paralela gracias a que los recursos físicos son divididos entre las máquinas virtuales creadas.

- Independencia, debido a la manera en que se reparten los recursos del servidor físico, el funcionamiento de las máquinas virtuales es independiente. Esto se traduce en una reducción de los costos de parada por error en el funcionamiento de un sistema.

- Encapsulación, el sistema operativo junto con las aplicaciones instaladas son guardadas en archivos lo cual otorga un alto grado de portabilidad a la aplicación para su distribución entre máquinas virtuales.

- Seguridad, el acceso a cada máquina virtual es independiente de las demás, por lo tanto un ataque a una de ellas no afectaría directamente al resto.

- Flexibilidad, las máquinas virtuales pueden ser creadas con las especificaciones de disco duro, memoria o red que sean necesarias para una aplicación en específico. Así, es mucho más eficiente que comprar diferentes CPU's con las características que se requieren.

Aunado a los beneficios mencionados anteriormente las máquinas virtuales tienen como principal objetivo la reducción de los costos de mantenimiento y la optimización de los recursos empleados para tareas específicas. Un ejemplo podría ser la activación de alguna nueva aplicación sin la necesidad de detener las ya existentes en el servidor físico.

En términos técnicos la virtualización se refiere a la abstracción de los recursos de una computadora (ver fig 2.1), lo que se conoce como hipervisor o MMV (Monitor de Máquinas Virtuales). Se trata de una capa de software entre el hardware de la máquina física (host) y el sistema operativo (OS Operative System) de la MV (Máquina Virtual), también conocido como sistema operativo huésped o SOH (Sistema Operativo Huésped).

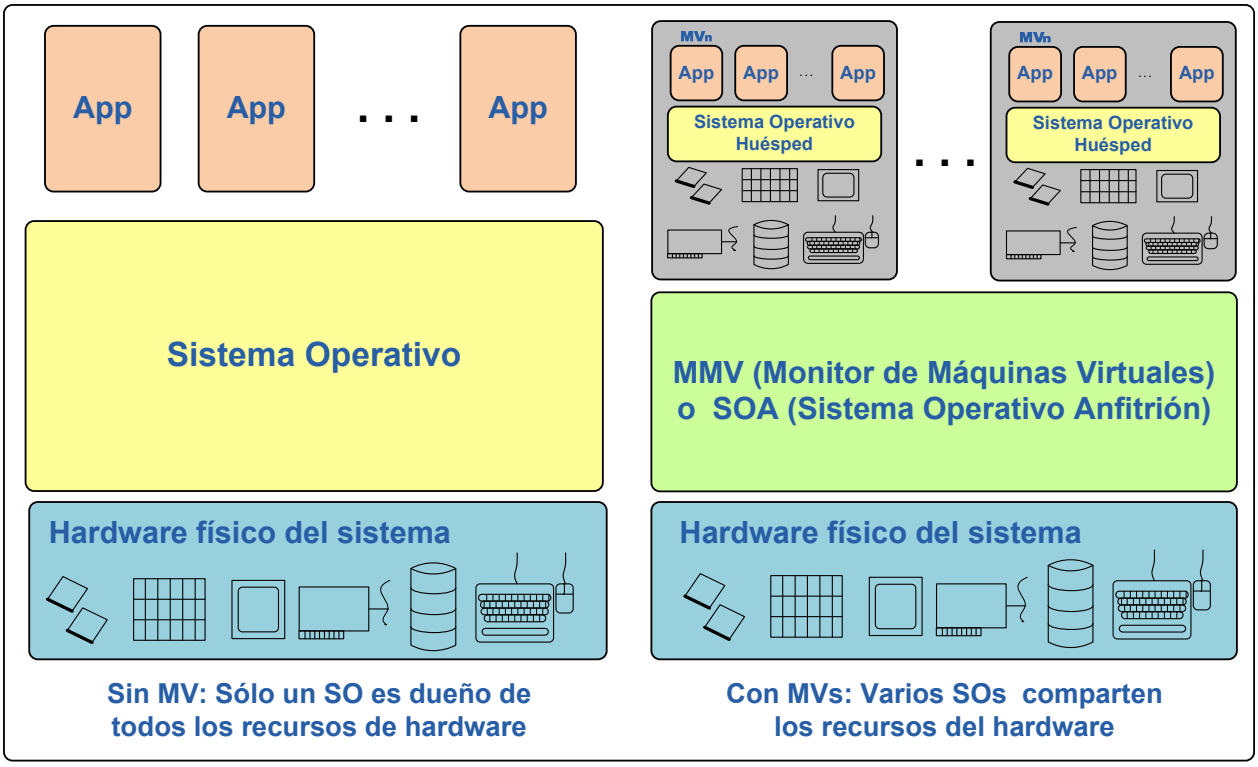


Figura 2.1: Arquitecturas de MMV.

Una MV es una ilusión de una máquina real. Se trata de un ambiente creado por un MMV que hace creer a otro sistema operativo que tiene acceso a todo el hardware de una computadora para sí mismo. A pesar que un OS por definición debe ser el único administrador del hardware de la plataforma, en entornos virtualizados el uso de los recursos de hardware es administrado por el MMV creando ilusiones de hardware sobre las cuales se podrá ejecutar otro sistema.

El MMV se encarga de manejar los recursos de cada MV implementada en el sistema, presentando a cada interfaz del hardware y haciendo creer que son las dueñas absolutas del hardware físico. Esta capa de software maneja, gestiona y arbitra los cuatro recursos principales de una computadora (CPU, Memoria, Red, Almacenamiento) y así podrá repartir dinámicamente dichos recursos entre todas las MVs definidas.

Como todos los sistemas operativos, un MMV carga aplicaciones. En el caso de los MMV, las aplicaciones que se ejecutan sobre éste son otros sistemas operativos. El objetivo del MMV es ejecutarlos al mismo tiempo compartiendo recursos. Se pueden encontrar tres formas de compartir recursos [30]:

- Algunos recursos se comparten por tiempo, como puede ser el procesador, el cual es utilizado por cada MV de acuerdo a algún tipo de programación de procesos.

- Algunos recursos pueden no ser compartidos, por lo cual se les da el acceso exclusivo a un SOH específico, e.g. una tarjeta de red la cual se defina pueda ser utilizada por una única MV.

- Otra forma de compartir o dividir los recursos entre las MVs, es dividiéndolos de forma virtual en varios recursos. De este modo, el MMV manipula transparentemente el canal de E/S (Entrada/Salida) del dispositivo, e.g. los discos, el MMV destina una parte del disco para cada MV y direccionamiento las llamadas de E/S del sistema operativo anfitrión o SOA (Sistema Operativo Anfitrión) a la porción del disco que le corresponde, asegurando que otro SOH acceda a un disco que no le pertenece.

2.2 MMV o hipervisor

En esta sección se dará una breve explicación sobre el tipo de arquitecturas de software que siguen algunas implementaciones de MMV o hipervisor. Además de la lista de MMVs revisaremos más a detalle la implementación de Xen, el cual es un código abierto que es frecuentemente utilizado para realizar casos de estudio en el área académica.

2.2.1 Arquitecturas de MMV

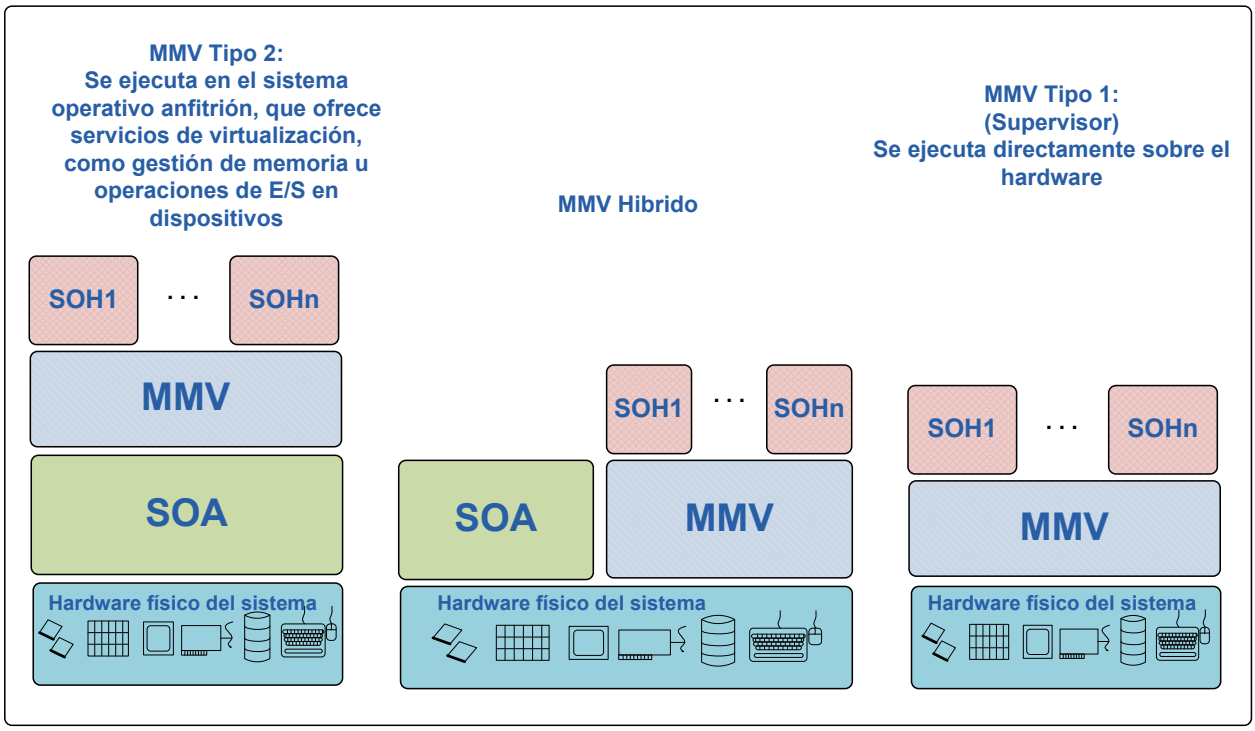


Figura 2.2: Arquitecturas de MMV.

Existen tres arquitecturas de implementación de MMVs como se presenta en la Figura 2.2:

1. Hipervisor de MMV solitario o tipo 1

2. MMV alojado en un Sistema Operativo o tipo 2

3. MMV híbrido

En la Tabla 2.1 se presentan algunos ejemplos de ambientes virtualizados para cada tipo los cuáles se discutirán en las secciones siguientes.

Arquitectura	Ejemplos
Tipo 1	VMware Server, VMware Workstation, Parallels Workstation, QEMU, VirtualBox, VMware Fusion y Parallels Desktop
Tipo 2	Oracle VM , VMware ESX Server, IBM POWER, Xen, , Hyper-V Hypervisor (PR/SM)
Tipo 3	Virtual PC y Virtual Server 2005 R2

Tabla 2.1: Ejemplos de los principales desarrollos MMV

2.2.1.1. Hipervisor MMV o tipo 1

Un hipervisor es una capa de software que reside justo arriba del hardware y abajo de los OS's, su propósito principal es proveer ambientes de ejecución separados llamados "particiones", desde los cuales las máquinas virtuales que contengan SOH puedan ejecutar. A cada partición se le provee una serie de recursos de hardware y el hipervisor es responsable de controlar y arbitrar el acceso al hardware físico.

El hipervisor debería tener sus propios controladores de dispositivos y programador de procesos. Con esto, se puede lograr un control total de los recursos de hardware que permita garantizar calidad de servicio para los sistemas huésped. Una ventaja es que el camino que realiza una solicitud de E/S de un sistema huésped es mucho más corta, ya que se realiza a través del los controladores del propio hipervisor y no a través de los del sistema anfitrión. Sin embargo, se pueden presentar problemas de implementación, ya que se deben reescribir controladores para cada dispositivo de hardware. Algunos ejemplos de este tipo de MMV son: VMware Server [38], VMware Workstation [39], VMware Fusion [37], QEMU, VirtualBox, así como Parallels Workstation [29] y Parallels Desktop [28].

2.2.1.2. MMV alojado en un Sistema Operativo o tipo 2

Se inicia con un SOA, es decir, el OS instalado directamente al hardware físico. Encima del SOA ejecuta un MMV, cuya función es crear y manejar máquinas virtuales, asignar recursos a estas máquinas y mantener estas MV separadas unas de las otras. El MMV se apoya de los controladores con que cuenta el SOA para obtener acceso a los recursos. Lo cual puede ser una ventaja o desventaja,

según el SOA utilizado.

A pesar de la facilidad con que puede ser implementado, el sistema es tan confiable, seguro y disponible como el sistema operativo del cual depende. Los más recientes ejemplos de este tiempo de MMV son: Oracle VM [27], VMware ESX Server [36], IBM POWER Hypervisor (PR/SM) [17], Hyper-V [23], Xen. Una variación de aplicación de este tipo de MMV pueden ser JVM [34] y CLR [13] para .NET.

2.2.1.3. MMV híbrido o tipo 3

Aquí tanto el SOA como el MMV se ejecutan directamente en el hardware y el SOH se ejecutan encima de la capa de virtualización. Un MMV híbrido busca lograr lo mejor de los 2 mundos; la confiabilidad y seguridad de un sistema de hipervisor solitario así como las facilidades que ofrece un sistema operativo existente. El MMV aún debe de pasar por el SOA para acceder al hardware, sin embargo tanto el SOA como el MMV se ejecutan en modo núcleo[3], así que esencialmente es una eterna batalla por el consumo de CPU; aún así el modo Híbrido es mucho más rápido debido a que esta ejecutándose en modo núcleo.

En este método, un micro hipervisor controla la CPU y la memoria, mientras que los recursos de I/O son controlados por los controladores de dispositivos de un sistema operativo de servicio sin privilegios, el cual no ejecuta otras aplicaciones y solo trabaja para el MMV para mejorar la seguridad y confiabilidad del sistema. Permite a cierto tipo de componentes de la computadora acceder a la memoria del sistema para leer o escribir independientemente de la CPU principal.

Un sistema híbrido ofrece lo mejor del MMV alojado y del MMV solitario, sin embargo introduce nuevos retos a resolver. Entre ellos, se incluye un problema de desempeño ocasionado por la sobrecarga de las transiciones de privilegios entre los SOH y el SOA. Pero todas las ventajas de un sistema operativo de servicio solo se logran cuando se agregan nuevos soportes de hardware para controlar el acceso a la memoria del sistema para leer o escribir sin pasar a través del hipervisor,esto se logra usualmente a través de DMA (por sus siglas en ingles Direct Memory Access). Ejemplos de este tipo de MMV son Virtual PC [25] y Virtual Server 2005 R2 [24].

2.2.2 Hipervisor Xen

Xen [9] es un ejemplo de hipervisor tipo 2 diseñado para poder ejecutar instancias de sistemas operativos con todas sus características, de forma completamente funcional en un equipo sencillo. Xen proporciona aislamiento seguro, control de recursos, garantías de calidad de servicio y migración de máquinas virtuales en caliente. Los sistemas operativos deben ser modificados explícitamente para ejecutar Xen (aunque manteniendo la compatibilidad con aplicaciones de usuario). Esto permite a Xen alcanzar virtualización de alto rendimiento sin un soporte especial de hardware.

Xen funciona actualmente en sistemas basados en x86 y se está portando a las plataformas AMD64, IA64 y PPC.

Xen es la capa de software que está situada directamente entre el hardware del plataforma y cualquier OS. Es el responsable de la planificación del CPU y la de particionar la memoria de varias máquinas virtuales ejecutándose en la plataforma. El hipervisor no solo abstrae el hardware de las MVs también controla los recursos compartidos entre las MVs. De acuerdo con la Figura 2.3, los componentes más importantes de Xen son:

- Xen hipervisor.
- Dominio 0: Es un núcleo de linux modificado que contiene permisos para acceder a dispositivos físicos de E/S, así como mantener interacción con los dominios U.
- Dominio de administración y control: es una serie de demonios de Linux, sirven para mantener la administración y control entre el dominio 0 y los dominios U
- Dominio U: Dominios donde pueden estar los SOH.

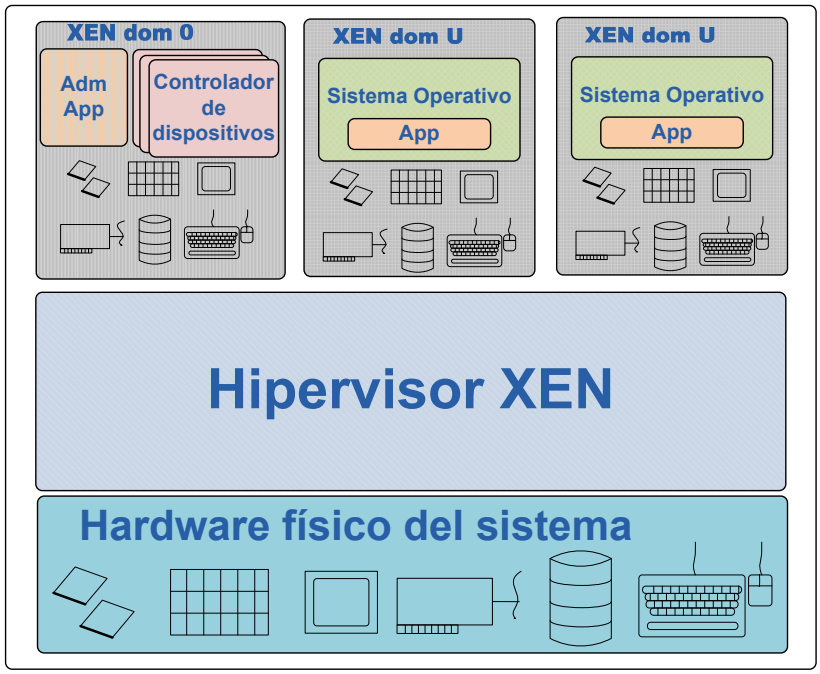


Figura 2.3: Arquitecturas de Xen.

2.3 Técnicas de virtualización de E/S con soporte de software

En el inicio del capítulo se habló de las formas de compartir recursos en sistemas virtualizados. Aquí se presentan diferentes técnicas utilizadas actualmente para virtualizar el acceso a dispositivos de entrada/salida (E/S), los cuales constituyen una de las plataformas susceptibles de ser virtualizadas.

Al momento de virtualizar un dispositivo de E/S debe tenerse en cuenta los tipos de operaciones que deben brindarse por parte del software de virtualización:

- Descubrimiento del dispositivo: un mecanismo para descubrir, consultar y configurar un dispositivo
- Control del dispositivo: un mecanismo de software para comunicarse con el dispositivo e iniciar una operación de E/S
- Transferencia de Datos: un mecanismo para que se pueda transferir datos desde y hasta la

memoria del sistema

- Interrupciones de E/S: un mecanismo de hardware para notificar al software de eventos y cambios de estado

Las técnicas de virtualización para dispositivos de E/S que se han utilizado son: emulación, paravirtualización y asignación directa. Cada una de ellas se describirá con mayor detalle en las secciones subsecuentes.

2.3.1 Emulación

La emulación, también conocida como virtualización completa, se refiere a la implementación completamente en software del hardware, La figura 2.4 ilustra el esquema general de esta técnica. Su gran ventaja es que no requiere modificaciones al SOH, ya que éste se ejecutará como lo haría de forma nativa sobre el hardware para el que fue construido interactuando con el emulador del MMV, tal cual lo haría con hardware real. El SOH no sabría que está sobre en un ambiente virtualizado. En esta solución, el MMV debe exponer los dispositivos de hardware de manera que puedan ser descubiertos por el sistema huésped, por ejemplo en un espacio de configuración PIC (por sus siglas en inglés Programmable Interrupt Controller) , indicando los dispositivos y los espacios de memoria a través de los cuales se puede interactuar con ellos.

De la misma forma, el MMV debe proveer métodos para capturar lecturas y escrituras para que el sistema huésped crea que está interactuando con hardware real. De la misma forma deben ser tratadas las interrupciones, emulando por software un PIC. Aunque es importante mencionar que esta emulación tiene algunas desventajas, una de ellas es que reduce el rendimiento de la aplicación que se está trabajando. Otra desventaja es que el modelo del dispositivo necesita emular de manera muy exacta el dispositivo de hardware, lo cual en algunos sistemas limita su desempeño.

2.3.2 Paravirtualización

El hecho de modificar el software dentro del huésped, es una técnica conocida como paravirtualización. La ventaja de paravirtualizar la E/S es mejorar el desempeño, pero tiene como desventaja que se

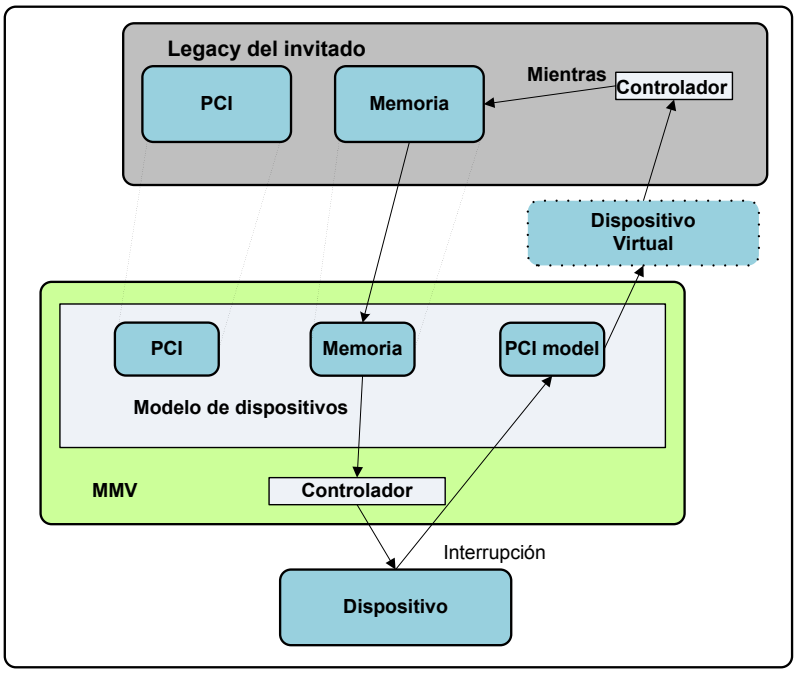


Figura 2.4: Técnica de Emulación

deben modificar los drivers y otros componentes de software del SOH, lo cual limita la solución si los SOH son heredados, es decir, tienen características y funcionalidades de algún otro sistema operativo, como sucede en el caso de migración de sistemas. La Figura 2.5 muestra el esquema general de paravitualización.

El funcionamiento se basa en una nivel de abstracción más alto en el cual el sistema huésped se comunica con una API del MMV para realizar las operaciones de E/S en vez de acceder directamente a un dispositivo de hardware (fuera real o virtual). De esta forma se reduce la interacción entre el MMV y el sistema huésped, resultando en mejor desempeño.

Las interrupciones, en vez de usar un mecanismo emulado, la paravirtualización utiliza un mecanismo de eventos o retrollamadas. De esta forma se elimina la comunicación con un PIC y se realiza a través de un ISR (Interrupt Service Routine) el cual acepta las interrupciones y genera una tarea para cada una de ellas en el contexto correspondiente. Este método también tiene la desventaja que los mecanismos de manejo de interrupciones del núcleo del sistema huésped deben ser modificados.

De la misma forma que la emulación, la paravirtualización soporta la migración de MV a cualquier plataforma que tenga las mismas APIs del MMV requerido por el SOH.

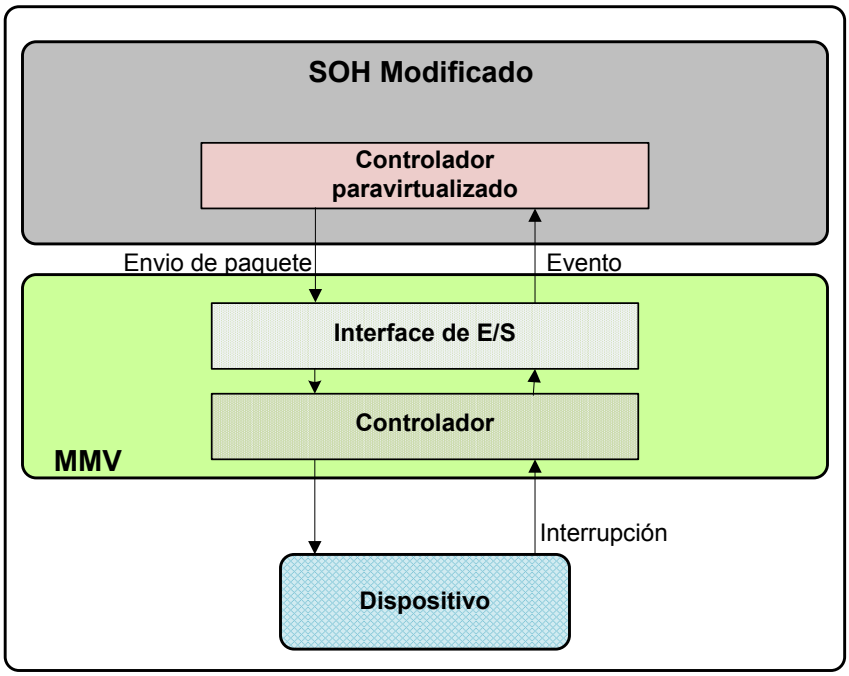


Figura 2.5: Técnica de Paravirtualización

2.3.3 Asignación directa

En el caso de la asignación directa como se muestra en la Figura 2.6, el MMV permite que el SOH acceda directamente al hardware que necesite para las operaciones de E/S. Estos recursos de hardware deben ser propiedad de la MV en cuestión. Como en la emulación, el sistema huésped tiene la posibilidad de interactuar directamente con la interfaz estándar de un dispositivo. Sin embargo, con asignación directa, el SOH puede reutilizar sus controladores ya existentes para comunicarse directamente con el dispositivo. Con asignación directa, se mejora el desempeño con respecto a la emulación, aunque se tiene la desventaja que solo se pueden asignar tantos recursos físicos como tenga el hardware, ya que no se comparten entre diferentes máquinas virtuales. También se pierde en capacidad de migración, ya que para llevar una máquina virtual de una plataforma a otra se

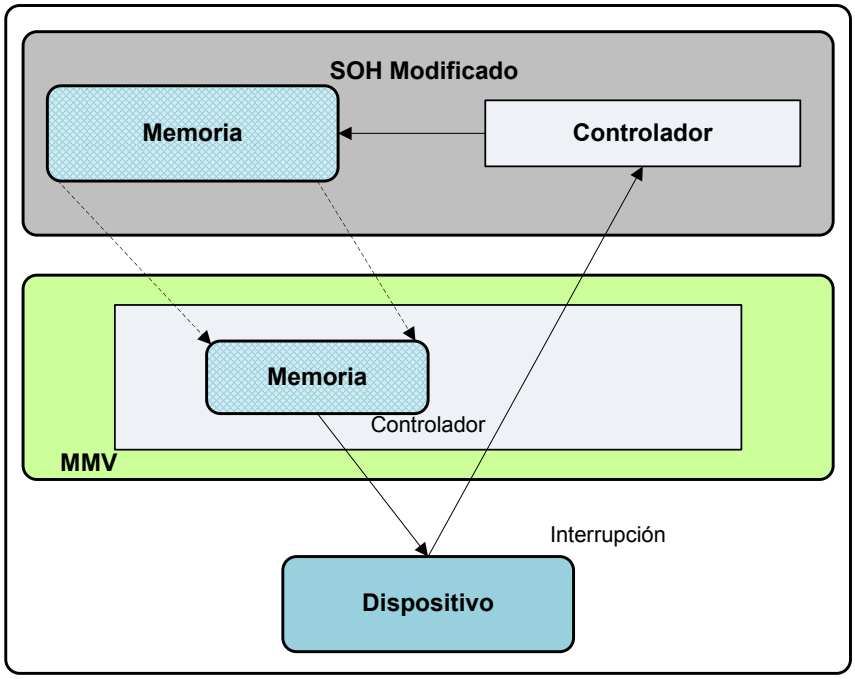


Figura 2.6: Técnica de Asignación Directa

requiere que en ambas existan los mismos dispositivos. Más allá de eso, el hardware a ser asignado directamente debe soportar ese método de trabajo y las interrupciones puede que aun sea necesario manejarlas a través del MMV, donde se perdería en la práctica lo que en teoría se gana en desempeño.

Finalmente la Tabla 2.2 contiene la comparación entre cada una de las técnicas de virtualización de E/S. La técnica de asignación directa es la única que no necesita intervención de MMV lo cual la hace una fuerte candidata para modelo de implementación de nuestro módulo de control de acceso.

Características	Emulación	Paravirtualización	Asignación directa
Modificación del SOH	no	no	si
Soporta la migración	si	si	no
Comparte recursos	si	si	no
Costo computacional	alto	medio	bajo
Flexibilidad	alto	medio	bajo
Interviene el MMV	si	si	no

Tabla 2.2: Comparación de técnicas de virtualización de E/S

2.4 Virtualización de E/S con soporte de hardware

La mayor desventaja de las virtualización es la pérdida considerable de rendimiento que presenta el sistema, especialmente en caso donde se aplican técnicas de emulación o interpretación. Una simple instrucción en un ambiente virtualizado se puede traducir a miles de instrucciones reales. Con el fin de disminuir y apoyar a los MMV en la virtualización, los principales fabricantes de procesadores han estado trabajando en desarrollar un soporte arquitectural en hardware que cumpla esta tarea. El soporte en hardware se ha dado principalmente para las siguientes plataformas: virtualización de CPU, E/S y algunos dispositivos específicos.

Este tipo de tecnologías solo había estado disponible para servidores con un alto poder de cómputo. En años recientes han sido incorporadas a computadoras de uso común. Intel y AMD (Advanced Micro Devices) especialmente cuentan en sus procesadores con hardware que ayuda a la virtualización de E/S.

2.4.1 IOMMU

Cuando un sistema se encuentra virtualizado, cada máquina virtual está provista de una vista de memoria física que en realidad se trata sólo de un rango de dirección en la memoria física del sistema anfitrión. Las direcciones de las MV son llamadas direcciones físicas del huésped GPA (por sus siglas en inglés Guest Physical Address), mientras que las del anfitrión se conocen como direcciones físicas del anfitrión HPA (Host Physical Address).

Cuando una MV quiere realizar una solicitud de E/S, ésta típicamente pasa por el MMV, el cual es el encargado de traducir esa dirección GPA a una HPA. La traducción genera una disminución del rendimiento del sistema, porque se pierden varios ciclos de reloj en realizar la traducción. La otra alternativa es quitarle la tarea al MMV del traducir las direcciones y agregar un módulo en hardware, como el IOMMU (Input/Output Memory Manage Unit), que realice esta tarea.

IOMMU es el nombre que recibe el hardware que sirve de soporte para la virtualización de E/S. EL IOMMU permite al sistema de software la creación de múltiples dominios de protección DMA,

estos dominios de protección son definidos como entornos aislados a los cuales se asigna una porción de memoria física del anfitrión. Este hardware provee la traducción de DMA y accesos seguros a los dispositivos físicos desde el software que se ejecuta sobre el SOH. Con IOMMU un controlador del SOH puede tener acceso directo a los dispositivos sin tener que ser modificado, además elimina los retrasos en la ejecución de las instrucciones.

En la Figura 2.7 se muestra la estructura interna típica de un IOMMU, el cual está compuesto de un dispositivo de remapeo DMA, éste a su vez se compone de una estructura de asignación de dispositivos y de estructuras de traducción. La estructura de asignación de dispositivos sirve para direccionar hacia la tabla de página que corresponda, en caso de detectar que el dispositivo no se encuentra asignado a esa máquina virtual entonces no podría tener acceso a él. Cada vez que se hace el recorrido de las tablas de página y se obtiene la traducción del GPA, ambas direcciones son colocadas en una memoria cache llamada TLB (Translation Lookside Buffer) para agilizar el proceso de búsqueda.

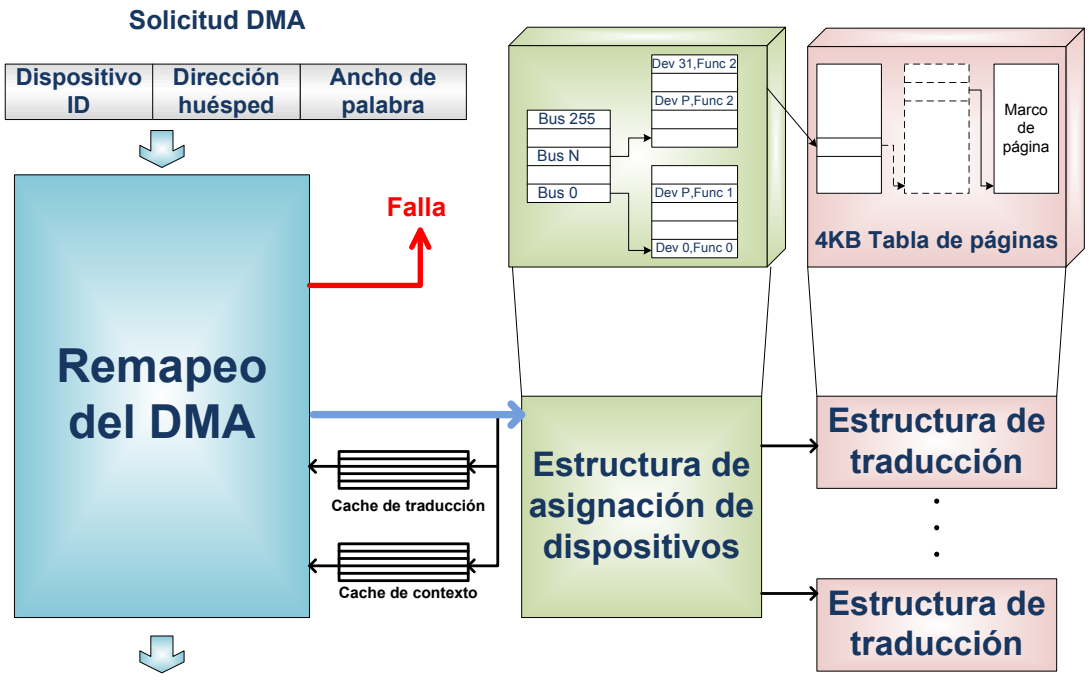


Figura 2.7: Estructura interna de un IOMMU

Un IOMMU maneja el acceso de dispositivos a la memoria del sistema. Está situado entre los dispositivos periféricos y el SOA, traduciendo direcciones desde la solicitud de dispositivos y verificando apropiadamente los permisos para cada acceso.

Un requisito general para todas las implementaciones de hardware para el soporte de virtualización de E/S, es la capacidad para aislar y restringir acceso de dispositivos a los recursos propiedad de un dominio. Esto le otorga al MMV las siguientes capacidades:

- Asignación de dispositivos de E/S a un dominio.
- Remapeo de DMA sin utilizar recursos del MMV.
- Remapeo de interrupción para apoyar el aislamiento y el encaminamiento de las interrupciones de dispositivos externos e interrupción de los controladores apropiados para máquinas virtuales.
- Fiabilidad por el hecho de poder crear dominios protegidos para cada MV.

El remapeo de DMA proporciona un soporte para el aislamiento de los accesos a memoria de los dispositivos, y permite a cada dispositivo ser asignado hacia un dominio gracias a la creación de tablas de páginas de E/S para cada dominio. Cuando el dispositivo intenta acceder a la memoria del sistema, el remapeo de DMA intercepta el acceso y utiliza las tablas de páginas de E/S para determinar si el acceso puede ser permitido, y cuál es la dirección física del acceso. La ventaja es que no se tiene que pasar por el MMV o hipervisor para hacer el acceso al dispositivo, y eso beneficia el rendimiento del sistema. La Figura 2.8 ilustra un ejemplo del acceso a dispositivos pansado por el hardware de soporte para la E/S.

2.4.2 Traducción y protección

Con IOMMU cada dispositivo es asignado a un dominio protegido. El dominio los definen las páginas de traducciones que van a ser utilizadas por cada dispositivo en el dominio. Así también, se definen los permisos para E/S en cada página. Las tablas de página son atrapadas por el IOMMU en el TLB[4] (Translation Lookside Buffer). Las entradas del TLB son etiquetadas por el dominio protegido y por la dirección de solicitud del dispositivo.

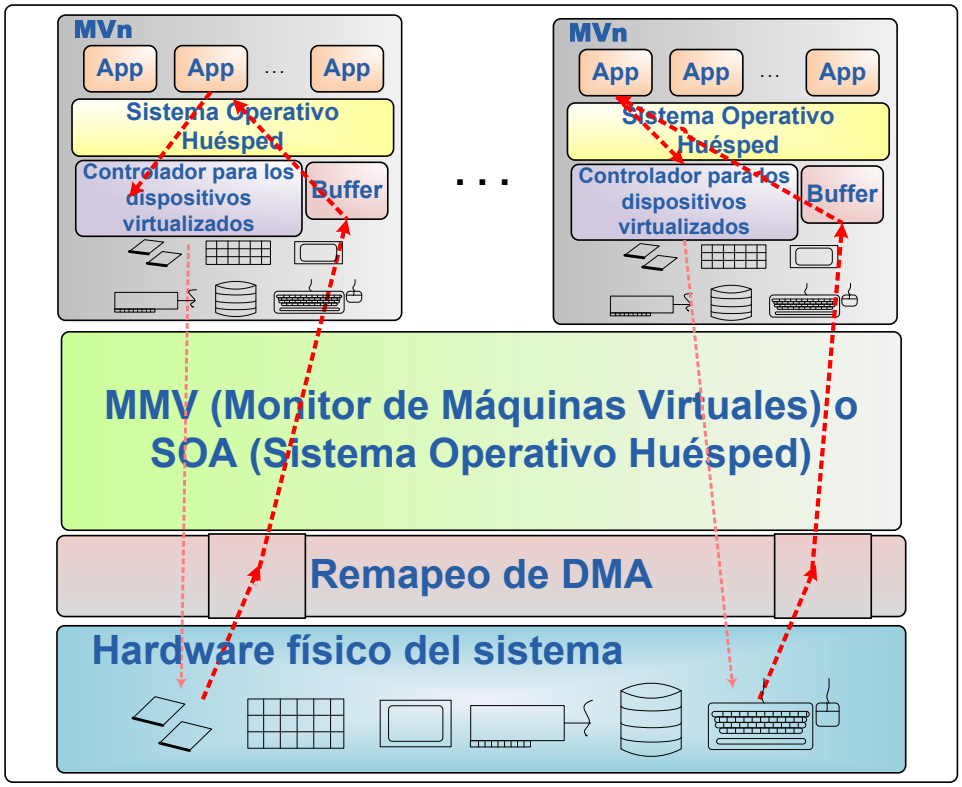


Figura 2.8: Acceso directo a dispositivos sin pasar por el MMV con hardware de soporte para E/S

El IOMMU determina cual es el dominio al que pertenece el dispositivo y después utiliza el dominio y la solicitud del dispositivo para buscar en el TLB. Las entradas de TLB contienen las banderas de lectura/escritura así como la traducción de la dirección.

El MMV configura la tabla de E/S para mapear hacia la dirección física del sistema desde la dirección física del huésped.

2.4.3 Modos de uso del IOMMU

El DMA es uno de los módulos más importantes del IOMMU, los modos de uso más comunes de este módulo se describen a continuación:

- Protección de un sistema operativo: Un sistema operativo puede definir un dominio que contenga código crítico y estructuras de datos, y restringir el acceso a este dominio a todos los

dispositivos de E/S en el sistema. Esto permite al sistema operativo reducir errores o corrupción de datos que se pueden generar a partir de una programación incorrecta de los controladores de los dispositivos, mejorando así la fiabilidad y robustez del sistema operativo.

- Función de apoyo: Puede ser utilizado para una mejor gestión del DMA para los dispositivos de aplicaciones antiguas ("legacy") hacia la memoria alta. Esto se logra por medio de la programación de las tablas de página de E/S, de tal forma que realice el remapeo de DMA hacia esas direcciones de memoria.

- Aislamiento: Esta función se puede ejemplificar muy bien a través de la figura 2.9. Aquí se muestra como un sistema operativo puede manejar la E/S mediante la creación de múltiples dominios asignando uno o más dispositivos de E/S a cada dominio. De esta manera, cada controlador de dispositivo registra su buffer de E/S con el sistema operativo y éste asigna ese buffer hacia un dominio específico utilizando el hardware para hacer el DMA y cumplir la protección del dominio.

2.4.4 Ventajas y desventajas del IOMMU

Hoy en día se cuenta con hardware que apoya directamente al MMV con la tarea de virtualizar una plataforma, y así poder tener varios sistemas independientes en una misma máquina sin que el MMV o el hipervisor tenga que intervenir en el proceso.

En caso específico de los IOMMU, hardware que sirve de soporte para la virtualización de E/S. Provee la traducción de DMA, es decir, maneja el acceso de dispositivos a la memoria del sistema, permite a cada dispositivo ser asignado a un dominio gracias a la creación de tablas de página multinivel de E/S para cada dominio y utiliza estas tablas para determinar la traducción de cada dirección virtual hacia la dirección física del acceso. El MMV es el encargado de realizar la programación de las tablas de página, lo cual es muy sencillo y flexible.

A pesar de los beneficios que a simple vista podrían surgir de esta tecnología, también existen desventajas que provocan que a gran cantidad de transacciones el rendimiento del sistema se vea perjudicado. Debido a que el tiempo de lectura/escritura de un dispositivo debe ser considerablemente

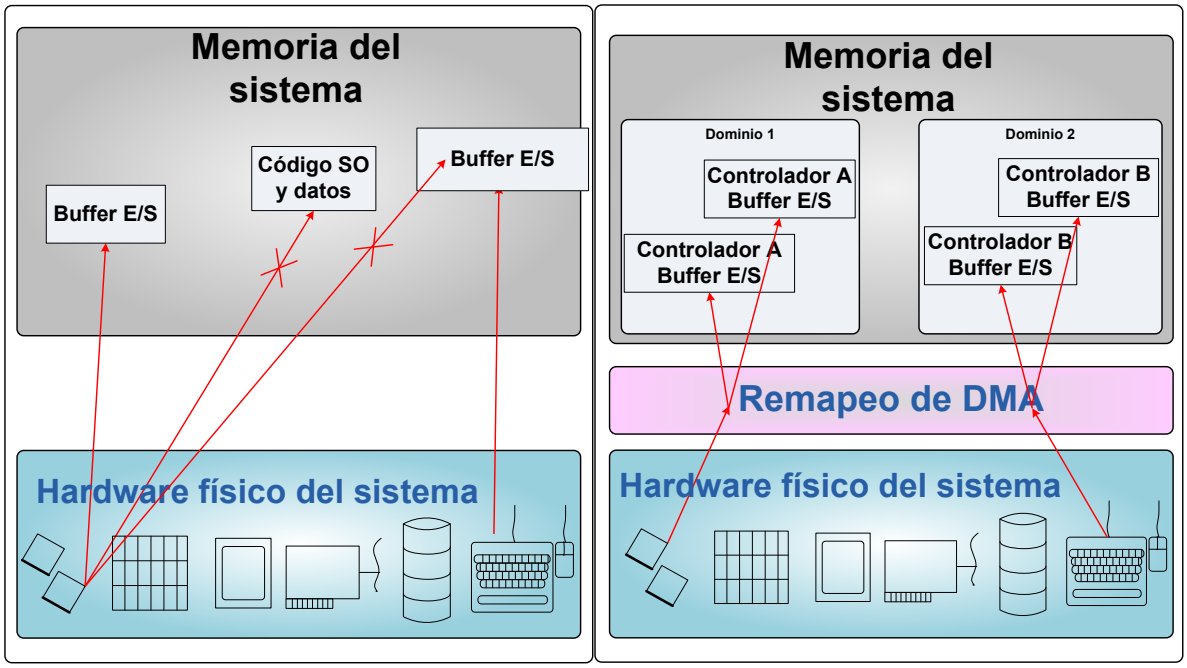


Figura 2.9: Ejemplo del uso del remapeo del DMA

rápido, la estructura de las tablas de página que contienen la traducción de las direcciones virtuales para cada dominio deben estar siempre cargadas en memoria principal, para poder entregar la traducción de la dirección de huésped, por lo tanto el IOMMU no soporta la demanda de página, porque eso le consumiría varias ciclos de reloj. Además, se cuenta con una caché que almacena la información de una traducción exitosa, la cual no se le podría obtener provecho si en cada modificación de la estructura de datos por parte del MMV, la cache tiene que ser invalidada.

2.5 Resumen

Las tecnologías de virtualización tienen una importante aplicación en diversas áreas de interés, pero debido a que el componente principal de la virtualización es el MMV y este es una aplicación que toma recursos directos del sistema, impacta en gran medida con el rendimiento de los SOH.

El hardware de soporte para E/S de virtualización todavía tiene muchas deficiencias en cuanto a los recursos de memoria principal que consume, debido al almacenamiento de las tablas de paginación.

Otro problema del IOMMU es que los dispositivos deben estar asignados de forma directa hacia alguna MV, lo que provoca que el MMV tenga que reprogramar continuamente las tablas de asignación e invalidar el TLB. Algún error por parte del MMV en este proceso puede dar como resultado, traducciones incoherentes de las direcciones GPA a HPA.

3

Arquitectura de Seguridad

La virtualización es una herramienta tecnológica muy utilizada en nuestros días, por lo que en años recientes una línea de investigación ampliamente abordada es la seguridad en ambientes virtualizados. Debido a que diferentes máquinas virtuales pueden tener acceso a dispositivos de hardware en un mismo entorno físico es importante proporcionar niveles de seguridad adecuados para el manejo de estos recursos. Para lograr este fin se han desarrollado algunas metodologías que intentan brindar estos niveles de seguridad, estableciendo políticas y niveles de confianza que existen entre diferentes capas del diseño tanto de software como de hardware.

A lo largo de este capítulo se mostrará el funcionamiento algunos mecanismos de seguridad existentes así como también el estado del arte respecto a estos mecanismos.

3.1 Seguridad en MV

La virtualización de plataformas es una tecnología que su uso va aumentando rápidamente. Los principales fabricantes de procesadores como Intel [19] y AMD [2] ofrecen hardware que facilita la virtualización del CPU, E/S y otros dispositivos específicos. A pesar de que estas tecnologías pueden

realizar la separación física de los recursos del hardware, actualmente no siempre es necesario tener una separación pura de cada MV, al contrario es deseable que las MVs puedan compartir recursos entre sí. Para poder llevar acabo esto es necesario establecer esquemas de seguridad que controlen la interacción entre las MV y sus recursos.

Es de vital importancia proteger la información y los procesos que ejecuta cada MV, contra el robo y la manipulación. Actualmente existen arquitecturas de seguridad para ambientes virtualizados [15] [32], este control es a nivel de software dentro del núcleo de hypervisor estableciendo algunas políticas de seguridad y de control de acceso.

Para crear una arquitectura de software que sea confiable en términos de seguridad no solo se debe tener en cuenta los mecanismos de seguridad, la implementación y administración sino también, es necesario hacer un análisis de las políticas de seguridad que debe soportar el sistema.

Una arquitectura típica divide a un sistema de software en diferentes capas dependiendo de su función y del nivel de confianza que se necesita tener. De tal forma que las capas superiores confíen en las capas inferiores, es decir, las aplicaciones deben confiar en las funciones del núcleo y el sistema operativo debe confiar en el hardware que ejecuta las instrucciones. Los tres niveles básicos para una arquitectura de seguridad son entonces, como se puede en la Figura 3.1, el nivel de software o aplicación, el nivel de software incrustado o firmware y el nivel de soporte de hardware. La administración de diferentes capas de confianza debe ser lo suficientemente flexible para soportar múltiples políticas de seguridad.

Incluir módulos de seguridad en hardware, es otra tendencia para implementar mecanismos de protección y control. Estos módulos ofrecen una protección física y lógica. Las aplicaciones más comunes de este estilo son de algoritmos criptográficos que implican generación, almacenamiento y procesamiento de claves y certificados [8] [3].

En las siguientes subsecciones veremos con más detalle de qué tratan estos conceptos.

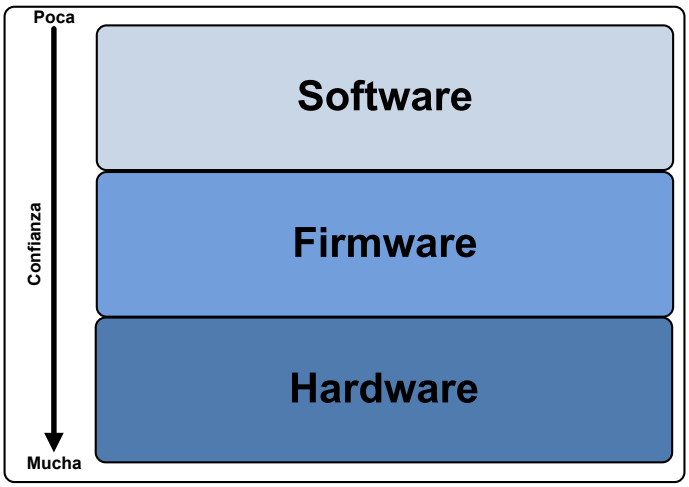


Figura 3.1: Capas de confianza.

3.2 Conceptos básicos de seguridad

Cuando se habla de seguridad de la información, se busca el cumplimiento básicamente de tres aspectos básicos que se deben preservar o garantizar, conocidos como la triada de seguridad: confidencialidad, integridad y disponibilidad (ver Fig 3.2).

Confidencialidad significa que la información procesada tiene carácter reservado, solo tienen acceso a ella las personas autorizadas. Las herramientas de seguridad informática deben proteger al sistema de invasiones, intrusiones y accesos, por parte de personas o programas no autorizados. Integridad se refiere a la validez y consistencia de los elementos de información almacenados y procesados en un sistema informático. Las herramientas de seguridad informática deben asegurar que los procesos de actualización estén sincronizados y no se dupliquen, de forma que todos los elementos del sistema manipulen adecuadamente los mismos datos. Disponibilidad se refiere a la continuidad de acceso a los elementos de información almacenados y procesados en un sistema informático. Es mantener el sistema informático en condiciones adecuadas para que el usuario pueda ingresar con la frecuencia y dedicación que requiera.

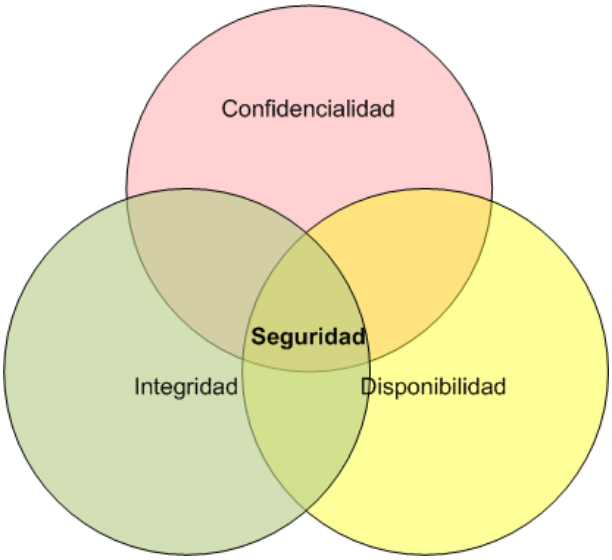


Figura 3.2: Triada de seguridad

Uno de los objetivos más importante de los sistemas de cómputo actuales es dar protección y seguridad a su información. Para ello es necesario establecer las normas, políticas y estándares de seguridad para los sistemas distribuidos que procesan, almacenan y transmiten información, a fin de minimizar riesgos en su integridad, confidencialidad y disponibilidad.

Las ventajas que ofrece el plantear objetivos o misiones en la organización, garantiza que la información manejada dentro y fuera del sistema central de la organización, cuente con los elementos necesarios para asegurar su protección contra alteración, divulgación, malversación o acceso no autorizados, permitiendo la continuidad de las operaciones en las áreas de negocio principalmente o en áreas donde se maneja información sensible.

Un modelo de seguridad es la presentación formal de las necesidades y prestaciones de seguridad otorgadas por un sistema. Lo primero que se debe de hacer para garantizar un nivel adecuado de seguridad en un sistema de información es identificar los aspectos que pueden afectarlo. Contar con un modelo de seguridad formal es vital para poder tener un esquema de seguridad fuerte que regule cómo un sistema maneja, protege y distribuye información delicada.

En particular los modelos de control de acceso identifican las reglas necesarias para que un sistema lleve a cabo el proceso que asegura que todo acceso a los recursos, sea un acceso autorizado. Estos

modelos refuerzan el principio fundamental de seguridad de autorización, ya que éste protege tanto a la confidencialidad como a la integridad. Básicamente existen dos tipos de modelos de control acceso: discrecionales y obligatorios.

Los controles de acceso discrecionales exigen un proceso de identificación y verificación del usuario a la entrada al sistema. Una vez verificado que se trata de un usuario autorizado, este podrá acceder a toda la información que no tenga definidas restricciones de acceso. Un ejemplo de este tipo de controles es el mecanismo de permisos de UNIX: cuando un usuario crea un objeto, puede o no definir un conjunto de permisos de manipulación para él, su grupo y los restantes usuarios.

Los controles de acceso obligatorios requieren múltiples niveles de seguridad. Los objetos y sujetos del sistema deben, obligatoriamente, tener un nivel de seguridad asociado. El sistema no debe permitir la creación de objetos o usuarios sin definir su nivel de seguridad, o que sean duplicados total o parcialmente con diferente nivel de seguridad. Cuando un usuario desea acceder a un objeto, el sistema comprueba los niveles de ambos y evalúa si se permite o no el acceso.

Dada la importancia que tiene el control de acceso en la seguridad de un sistema, se han desarrollado varias implementaciones en modelos formales. Las más comunes son el modelo Bell-LaPadula [6], el Bilba [7] y el Clark-Wilson [10]. A continuación una pequeña descripción de estos modelos:

- Bell-LaPadula [6]: Es un modelo de máquina de estados que captura los aspectos de confidencialidad. Así, si podemos probar que todas las transiciones son seguras y que nos llevan a estados considerados seguros, podemos mediante inducción demostrar que el sistema es seguro. Los permisos de acceso están definidos en una matriz de control de acceso y por medio de niveles de seguridad, las políticas de seguridad previenen que la información fluya hacia abajo desde el nivel más alto de seguridad hasta el nivel más bajo de seguridad. Clasifica las entidades en niveles de seguridad: desclasificado, confidencial o secreto, por ejemplo. La regla fundamental de este modelo es la que ninguna entidad debe tener acceso de lectura a un objeto clasificado por encima de su propio nivel de seguridad. Es decir, el nivel de seguridad de una entidad debe ser igual o superior al del objeto al que se va a acceder. Sin embargo este modelo permite la escritura en objetos de un nivel superior al de la propia entidad basándose en la regla de que para escribir en un objeto la entidad debe tener un nivel de seguridad igual o inferior al del

objeto. Por tanto, si no existen los mecanismos de control adecuados de la entidad hacia el objeto se puede llegar a la violación de la integridad del objeto.

- Biba [7]: Modelo enfocado para asegurar la integridad de la información. Maneja restricciones de lectura y escritura basados en el nivel de integridad para el acceso de sujetos y objetos. Los objetos son asignados a un nivel de integridad de acuerdo al daño que sufrirían si fueran modificados de manera inapropiada. Los usuarios son asignados a niveles de integridad basados en su veracidad. Los compartimentos de integridad son interpretados como compartimentos de confidencialidad. El nivel de integridad de un sujeto está basado en el nivel de integridad del usuario que representa y en sus necesidades, de acuerdo al principio del último privilegio. Esto se lleva a cabo permitiendo solo la escritura hacia abajo, es decir, se asegura que las entidades solo puedan escribir en objetos con nivel de seguridad igual o inferior al suyo. De esta forma se evita que una entidad altere objetos con clasificaciones superiores, y por tanto se impide la violación de la integridad. Sin embargo este modelo permite la lectura de objetos con clasificaciones igual o superior a la de la entidad, lo que puede llevar a cabo la violación de la confidencialidad. La solución proviene de la adecuada combinación de ambos modelos en un mismo sistema.

- Clark-Wilson [10]: Es un modelo que se basa en la jerarquización de aplicaciones para el manejo de información de parte de los usuarios, se encuentra orientado a proteger la integridad de la información. Este modelo propone dos categorías de mecanismos para prevenir la modificación irrecuperable e incorrecta de la información, intencionada o no. El mecanismo de transacciones correctas busca garantizar que un usuario no pueda modificar información arbitrariamente. Solamente permite la modificación en determinadas formas, restringiendo los posibles cambios incorrectos. Por ejemplo, un sistema en que solamente a un conjunto cerrado de programas se les permita modificar la información. El mecanismo de separación de obligaciones, por otra parte, trata de mantener la consistencia de la información separando todas las operaciones en diferentes partes que deben ser realizadas por diferentes sujetos. De esta manera, un usuario autorizado a iniciar una transacción no estará autorizado a ejecutarla o validarla.

Los modelos de seguridad son la representación formal de las políticas de seguridad, las cuales son un conjunto de leyes, reglas y prácticas que regulan la manera de dirigir, proteger y distribuir recursos para llevar a cabo los objetivos de seguridad.

La política define la seguridad de un sistema de cómputo. La política especifica qué propiedades de seguridad el sistema debe proveer. De manera similar, la política define la seguridad informática para una organización, especificando tanto las propiedades del sistema como las responsabilidades de seguridad de las personas.

La política de seguridad involucra papeles que se repiten en muchas situaciones y también en aplicaciones específicas. Se aplican los siguientes roles genéricos:

Primero, la política de seguridad debe reflejar fielmente el mundo real. Esto significa que deber ser especificada sin ambigüedades. Segundo, las políticas seleccionadas deben ser hechas sobre la situación actual de los sistemas conectados en red. Una política de seguridad debe estar especificada en un documento especial para tal propósito redactada en un lenguaje natural, claramente y sin ambigüedades posibles. El documento deberá especificar qué propiedades de seguridad se pretenden cubrir con la aplicación de las políticas y la manera de usarlas.

3.3 Mecanismos de seguridad

Los mecanismos de seguridad informática son herramientas que se utilizan para fortalecer la disponibilidad, confidencialidad y la integridad del sistema informático. Algunos mecanismos de seguridad son: control de acceso, monitores de referencia. Dichos servicios se describirán en las secciones subsecuentes.

Control de acceso

Los mecanismos de control de acceso [33] son representados conceptualmente mediante el uso de una matriz de control de acceso, la cual lista por renglones a los sujetos activos (usuarios de un sistema) y en las columnas se listan los objetos pasivos (archivos u otros recursos del sistema). Si se

guardara la matriz completa se requeriría de mucho espacio para un caso real con una gran cantidad de usuarios y recursos, debido a ello los sistemas reales utilizan los renglones o las columnas o una combinación parcial de estos para decisiones de control de acceso.

Uno de los mecanismos más utilizados son las listas de control de acceso (ACL, por sus siglas en inglés). Cada objetivo o recurso tiene asociada una ACL que contiene los nombres (identificadores) de los usuarios que pueden acceder a éste. A lado de cada nombre se coloca una lista de acciones que le son permitidas al usuario.

Las ACL son usualmente usadas en el campo de redes y se implementan dentro de switches, ruteadores y firewalls. Existen varios tipos de ACL, pero las más usadas en estos casos son las Listas de Acceso estándar y las extendidas [26].

El ACP (Access Control Policy) define las reglas de acceso y las etiquetas de seguridad. Cuando los dominios solicitan acceso de comunicación, el Módulo de Control de Acceso MCA interpreta la política y maneja la decisión de control de acceso. Luego, el ACP activa la etiqueta de seguridad y las reglas de acceso y las asigna a los dominios y recursos.

Monitor de Referencia

Un monitor de referencia [14] es el mecanismo utilizado para controlar el acceso de un conjunto de sujetos hacia una serie de objetos. El monitor es el subsistema que se encarga de verificar la legitimidad de los intentos de acceso de los sujetos hacia cierto objeto, y representa la abstracción para el control entre los sujetos y los objetos. Debe tener ciertas propiedades: debe ser a prueba de manipulaciones, siempre debe estar invocado, y debe ser lo suficientemente simple como para estar abierto a un análisis de la seguridad. Un monitor de referencia implementa el "mecanismo" para la separación de políticas requeridas como se puede ver en la Fig 3.3.

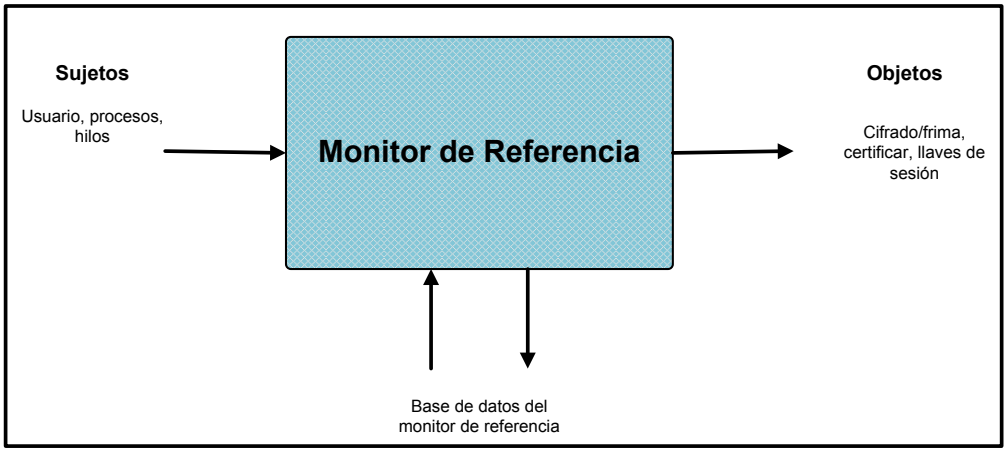


Figura 3.3: Esquema de un monitor de referencia.

Coprocesador de seguridad

Existen co-procesadores de seguridad como el 4758 de IBM [12], los cuales facilitan la implementación de aplicaciones seguras en una plataforma de cómputo, y apoyan a los procesadores de propósito general en la tarea de proveer seguridad.

El principal objetivo de este tipo de implementaciones es tener un mecanismo para poder dar los principales servicios de seguridad, mencionados en 1.6.1, como la autenticación y una infraestructura de llave pública (PKI).Los principales componentes de un coprocesador de seguridad deben ser por ejemplo los que se muestra en la Figura 3.4.

Bajo este esquema, los bloques de módulo criptográfico, módulo aritmético y generador de números aleatorios son componentes que contienen métodos criptográficos y que además son de gran apoyo para la infraestructura de llave pública, ya que ayudan a proveer de confidencialidad, autenticidad e integridad de datos. En el módulo criptográfico se pueden tener implementados

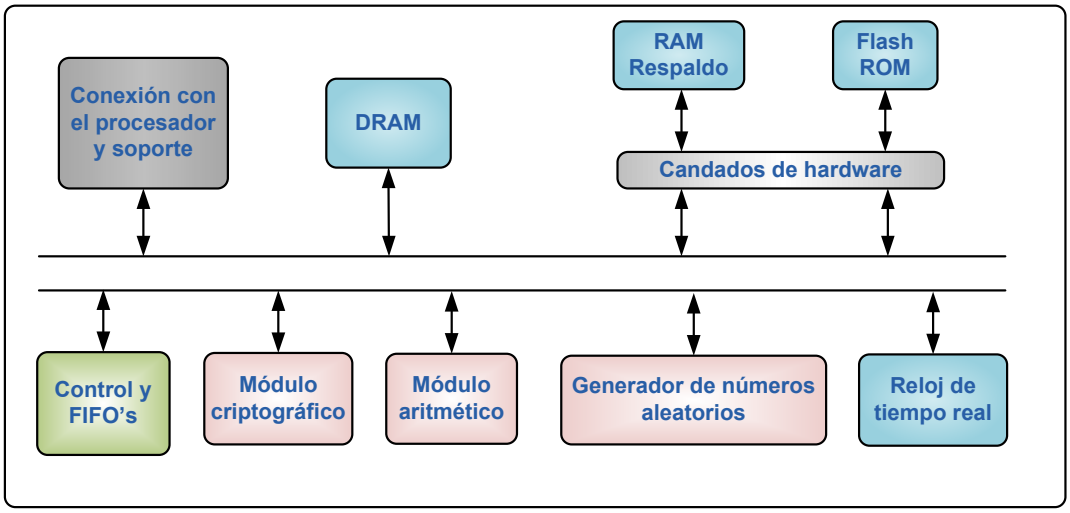


Figura 3.4: Esquema general de un coprocesador de seguridad.

algoritmos de llave simétrica como DES (Data Encription Standard), AES (Advantage Encription Standard), RC4, etc. En el módulo aritmético su pueden incluir operaciones eficientes de suma, elevar al cuadro, multiplicación, inversión y exponenciación. El bloque aleatoriedad se trata de un hardware que genera números aleatorios, estos son utilizados por el procesador para ayudar a generar su secreto inicial y para aplicaciones que validan el número generado o llave o el secreto inicial.

Los candados en hardware hacen que área de la memoria flash se conviertan en solo lectura y que zonas de LBBRAM (por sus siglas en inglés Lockable Battery-Backed Random Access Memory) que son inaccesibles. Un proceso es el encargado del control de los candados. Los cuales deben ser implementados de forma que solo puedan ser configurados en una ocasión y para quitarlos es necesario que el procesador sea reiniciado. En la RAM de respaldo en es donde se almacenan los secretos, como llaves, firmas, etc.

3.4 Seguridad en hypervisor

Las implementaciones actuales de hypervisor aseguran una protección contra aplicaciones ejecutándose en las máquinas virtuales, además de dar la garantía de ambientes a protegidos para cada MV y de sus recursos. Algunas de las prestaciones que un hypervisor con seguridad brinda son [18]:

- **Separación forzosa**. El intercambio y la comunicación entre máquinas virtuales tiene que ser mediada a través de hypervisor. Las operaciones pueden ser controladas aplicando políticas de seguridad como la seguridad multinivel (MLS), basada en las funciones de control de acceso RBAC (por sus siglas en inglés Role-based Access Control), entre otros.
- **Certificación y garantías de integridad para el hipervisor y sus máquinas virtuales.** El objetivo aquí asegurar que el arranque de la plataforma sea a través de un análisis de integridad y autenticación, así como el arranque de las máquinas virtuales y, opcionalmente, los sistemas operativos huéspedes y aplicaciones que se ejecutan en máquinas virtuales.
- **Control de recursos.** Se encarga de garantizar la limitación de los recursos, como la memoria o ciclos de CPU. Más elaborado es la gestión de los recursos necesarios para el control de ancho de banda de red, por ejemplo, para limitar el ancho de banda de red a una máquina virtual.
- **Servicios seguros.** Se proporciona una infraestructura de base para el desglose de los servicios, tales como la gestión de políticas de seguridad o auditoría distribuido, en partes más pequeñas y más manejables entornos de ejecución protegidos, permitiendo así que su uso sea en todo el sistema y, potencialmente, mejorar la seguridad de estos servicios.

3.4.1 IOMMU para protección

Como vimos en la sección 2.4.1 (en la pág. 21) IOMMU (por sus siglas en inglés Input/Output Memory Manage Unit) es la alternativa en hardware para apoyar al MMV en la virtualización de E/S, se encarga de hacer la traducción de direcciones entre la dirección física del huésped GPA (por sus siglas en inglés Guest Physical Address) y las direcciones físicas del anfitrión HPA (Host

Physical Address). Desde el punto de vista de seguridad, este dispositivo no únicamente es útil para disminuirle la carga de trabajo al MMV en el momento de un operación de E/S, sino que también permite crear ambientes aislados, los cuales son comúnmente llamados dominios, que no es otra cosa más que el entorno de configuración y rango de direcciones de las máquinas virtuales del sistema.

Cada MV es representada como un dominio, y a su vez cada dominio puede tener asignado con cualquier dispositivos de E/S pero éste no puede ser designado a otro dominio mientras esté concedido a otro dominio, todo ese proceso se hace mediante la configuración del IOMMU. La manera en la que IOMMU configura cada dominio es a través de la programación de la estructura de asignación de dispositivos (tabla de contexto) y de la estructura de traducción (tablas de paginación) las cuales son almacenadas en memoria principal. Estas estructuras son empleadas para garantizar que la información de cada dominio se encuentra en rangos diferentes de memoria y que un dominio no puede tener acceso a un recurso no asignado. En otras palabras, el aislamiento del DMA es llevado a cabo mediante la restricción de acceso a dominios de protección de memoria física.

Los dispositivos de E/S asignados a un dominio de protección pueden ser provistos de un mapa de memoria física diferente al mapa original manejado por el anfitrión. Un IOMMU maneja la dirección especificada mediante una solicitud de DMA como una dirección virtual. Dependiendo del modelo de uso del software puede ser una dirección GPA de alguno de los dispositivos de E/S asignados. Después, el hardware transforma la dirección en una solicitud DMA solicitada por un dispositivo de E/S hacia su HPA correspondiente. La figura 3.5 ejemplifica un esquema con múltiples dominios, los dispositivos 1 y 2 son asignados a los dominios 1 y 2 respectivamente, así cada domino ve solamente su propio espacio de direcciones que cuando se realiza la solicitud de DMA la dirección GPA es traducida a la HPA.

3.5 Trabajos relacionados

En los principios las máquinas virtuales eran esencialmente para computadoras con alto poder de cómputo que simulaba o emulaban el hardware de la plataforma, hacia los SOH. En los 1960s y 1970s existía el IBM VM/370 el cual fue el primer MMV.

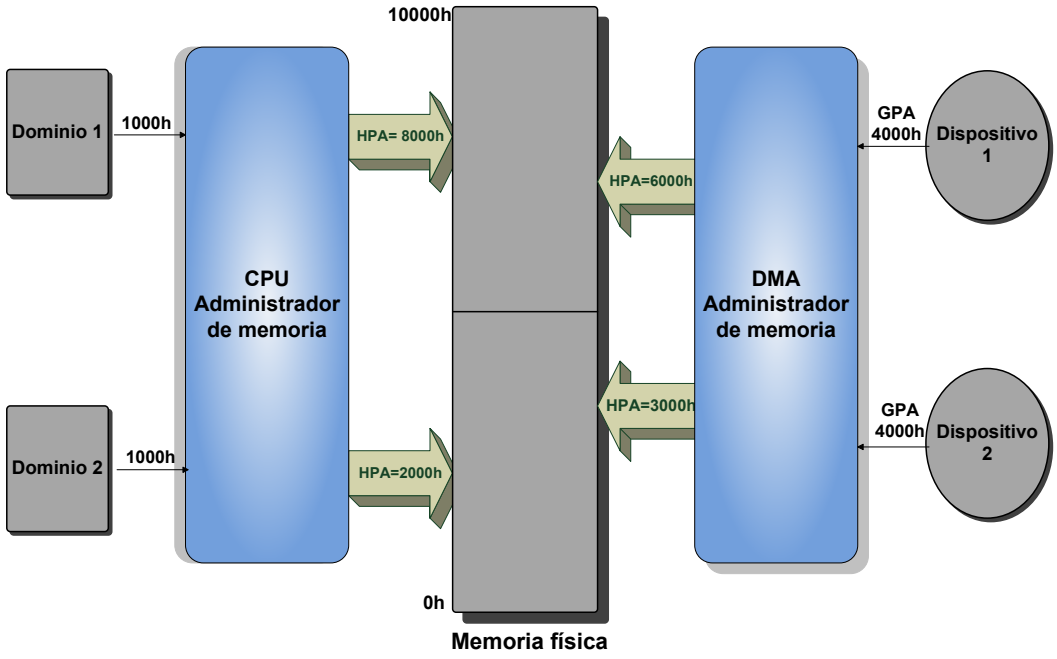


Figura 3.5: Ejemplificación del remapeo DMA realizado por el IOMMU

El KVM/370 [16] era una versión de seguridad del monitor virtual de la máquina de la IBM VM/370. Este sistema proporcionó las máquinas virtuales para sus usuarios, y una de sus metas era prevenir comunicaciones entre las máquinas virtuales de diversas clases de la seguridad, así que los usuarios en diversas clases de la seguridad podrían utilizar el sistema en el mismo tiempo. Como VM/370, las máquinas virtuales los minidisks y permitió que los sistemas compartieran algunas áreas del disco. De manera semejante VM/370, utilizó una política de la seguridad para mediar el acceso a las áreas compartidas del disco para limitar comunicaciones entre los sistemas.

Karger et al., en la Digital Equipment Corporation. Desarrollaron un monitor virtual de la máquina para la DEC VAX [21] fue el primer hypervisor con servicios de seguridad entre las máquinas virtuales con el fin de soportar el uso de comportado de recursos entre las MVs. Contenía algunas políticas de seguridad y etiquetas asociadas con cada MV y sus recursos. Su estructura es típica de las máquinas virtuales diseñadas para proporcionar seguridad. El VAX tiene cuatro niveles de privilegio: usuario, supervisor, ejecutivo, y modos del núcleo. Para proporcionar una máquina virtual compatible,

las máquinas virtuales deben también tener cuatro niveles de privilegio. Sin embargo, el modo del núcleo permite que un proceso tenga acceso a instrucciones privilegiadas en el hardware de VAX directamente. Solamente el MMV se permite hacer esto. Las máquinas virtuales no pueden tener acceso a modo del núcleo. La solución es proporcionar modos virtuales. Estos modos son usuario de la MV (que corresponde al modo del usuario), modo del supervisor de la MV, y ejecutivo de la MV y los modos del núcleo de la MV (ambos el modo realmente ejecutivo).

T. Garfinkel et al., en el 2003, presentaron Terra [15] otro enfoque de virtualización y técnicas de computación para crear sistemas de protección. Para ello proponen el uso de una partición del hardware de soporte disponible en la plataforma a múltiples MVs aisladas. En este trabajo se presenta una arquitectura flexible para la plataforma segura. Su arquitectura se basa en crear particiones denominadas "closed box" que son las que ofrecen aspectos de seguridad. Esta arquitectura propone una metodología mediante el uso de TVMM (trusted virtual machine monitor) que permite que las estrategias de seguridad puedan correr de manera paralela con aplicaciones instaladas en hardware de propósito general. El objetivo general es permitir que el hardware y las TVMM actúen de manera conjunta para realizar tareas de identificación criptográfica sobre el software que se está ejecutando. Para probar la eficiencia de este diseño se propone un conjunto de pruebas con el objetivo de probar la resistencia y las limitaciones de esta arquitectura.

R. Sailer et el. en el 2005 publica el artículo "Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor" [32], muestra el diseño y la implementación de sHyper. Esta aproximación a diferencia de otras existentes en la literatura pretende atacar los problemas de seguridad a nivel comercial, donde al objetivo principal es hacer aplicaciones de seguridad que no sean invasivas con las aplicaciones ya instaladas. Una arquitectura de seguridad para ambientes virtualizados que controla los recursos compartidos entre las MVs de acuerdo a unas políticas de seguridad. La implementación fue hecha dentro del hypervisor de Xen [5] pero la arquitectura no es específica para un solo hypervisor. De esta manera, el objetivo es transformar la capa hypervisor como un componente estándar en el sistema. Empleando los recursos de grano grueso para brindar protección contra las sobrecargas de trabajo.

3.6 Discusión

Si bien es cierto que las ventajas que se obtienen de la virtualización son amplias con respecto a las aplicaciones corriendo en servidores físicos, es necesario también mencionar que el uso de las máquinas virtuales presenta ciertas desventajas a tomar en cuenta. Entre ellas podemos mencionar las siguientes:

- En ocasiones el rendimiento es inferior debido a que hypervisor introduce una capa intermedia en la gestión de las peticiones de acceso al hardware y a la concurrencia a sí mismo, en consecuencia el rendimiento se ve afectado.

- No es posible utilizar hardware que no esté gestionado o soportado por el hypervisor.

- La portabilidad está condicionada a la solución de virtualización adoptada (GNU/Linux, Mac OS o Windows, etc) así que la decisión sobre el tipo de virtualización debe hacerse considerando las necesidades futuras de la empresa.

La mayor desventaja de la virtualización es la pérdida considerable de rendimiento que presenta el sistema virtualizado, una simple instrucción en un ambiente virtualizado se puede traducir a miles de instrucciones reales. Con el fin de disminuir y apoyar a los MMV en la virtualización. Para reducir los efectos de la virtualización algunas de las principales marcas de procesadores (Intel, AMD, etc) han desarrollado un soporte arquitectural en hardware que cumpla con el denominado IOMMU (Input-Output Memory Manage Unit).

En el caso del IOMMU, el principal problema es la cantidad de memoria física que consumen para definir la estructura de datos y las tablas de páginas que definen los dominios (término genérico utilizado para las máquinas virtuales). Además, es necesario tomar en cuenta que debido a que varias máquinas virtuales tienen acceso a la memoria cache, por lo que cada cambio que se realiza en él debe ser actualizado para todas las máquinas virtuales de lo contrario podría presentarse inconsistencia en los datos.

En el caso de la implementación de AMD para el IOMMU, también verifican los permisos de lectura/escritura de cada transacción de E/S. Si lo que se desea es tener control absoluto de lo que hacen esos dispositivos, algo importante sería poder autentificar que los dispositivos conectados a la plataforma son confiables.

De igual forma al implementar la virtualización es deseable que las MVs puedan compartir recursos entre sí. Para poder llevar acabo esto es necesario establecer esquemas de seguridad que controlen la interacción entre las MV y sus recursos, sin que esto afecte el rendimiento del sistema. El control de acceso, aislamiento de máquinas virtuales o autentificación de dispositivos, suelen ser medidas de seguridad que se le agregan al sistema a nivel de software.

Si hablamos de implementaciones de hypervisors con módulos de seguridad, como en el caso de sHyper y Terra, estos ofrecen típicamente monitores de referencia que tienen la función de un control de acceso por mandato. Esto es implementado a nivel de núcleo.

Una mejora a esta propuesta podría ser implementar el módulo de control de acceso directamente en hardware, porque a pesar de ser implementado dentro de la primera capa de software del sistema, la consulta de las lista tiene que ser secuencial, y en hardware se podría realizar la lectura de las reglas en forma paralela.

3.7 Resumen

Es necesario garantizar que las máquinas virtuales sean ejecutadas en un ambiente seguro, que ofrezca los servicios básicos de seguridad. Las propuestas del integración de módulos de seguridad dentro del hypervisor son muy flexibles pero le roban tiempo de procesamiento a las tarea comunes del hypervisor.

Otras propuestas más generales de seguridad son los coprocesadores de seguridad, los cuales proporcionan módulos en hardware que son mucho más rápido y seguros pero que no pueden ser fácilmente modificados como en el caso de software. Sin embargo, estos coprocesadores de seguridad en sus principios no fueron pensados para ambientes virtualizados, pero podrían ser utilizados para administrar los recursos entre máquinas virtuales.

Por otro lado, el hardware de soporte para E/S de virtualización todavía tiene muchas deficiencia en cuanto a los recursos de memoria principal que consume, debido al almacenamiento de las tablas de paginación.

Otro problema del IOMMU es que los dispositivos deben estar asignados de forma directa hacia alguna MV, lo que provoca que el MMV tenga que reprogramar continuamente las tablas de asignación e invalidar el TLB. Algún error por parte del MMV en este proceso puede dar como resultado, traducciones incoherentes de las direcciones GPA a HPA.

4

Módulo de control de acceso en hardware

Las aplicaciones de seguridad para el uso de hardware en una plataforma virtualizada es en los últimos años un tema de vital importancia. Implementaciones de VMM con módulos de seguridad ayudan a enfrentar el problema de la seguridad, sin embargo el rendimiento de los GOS se ve afectado porque se incrementan las tareas que realiza el VMM. El IOMMU podría ser la solución no sólo para evitarle al VMM el procesamiento de la traducción de DMA, sino también para crear ambientes aislados entre cada máquina sin ocupar recursos de VMM, pero tiene la desventaja que para realizar estos dominios utilizan una gran sección de la memoria principal. Otro inconveniente del IOMMU es el aislamiento total de los dominios, hoy en día es deseable que las VMs puedan compartir recursos de hardware entre sí. El módulo de control de acceso directamente en hardware, podría ser una mejora a estas dos alternativas presentadas, porque se puede implementar en hardware una lista de control de acceso para administrar el acceso a los dispositivos y la consulta de las reglas puede hacerse de forma paralela en lugar que secuencial como en el caso de VMM con seguridad.

En este capítulo se plantean los principios de diseño del módulo de control de acceso en hardware. Así como también una descripción de los conceptos básicos del procesador Microblaze y del bus local del procesador PLB. Los requerimientos y objetivos que deben ser cubiertos por el MCA son descritos,

al igual que la arquitectura del MCA en hardware y las consideraciones que fueron tomadas para su integración al bus de periféricos PLB.

4.1 Arquitectura general del procesador Microblaze

Microblaze es un procesador blando de 32-bit, es decir, no se trata de un procesador físico sino más bien de una descripción de hardware hecha en lenguaje HDL (por sus siglas en inglés Hardware Description Language), el cual es implementado mediante herramientas de apoyo en la lógica interna del FPGA [22]. Cuenta con una arquitectura RISC Harvard, un conjunto de instrucciones optimizadas para aplicaciones embebidas y su implementación en FPGAs de Xilinx tales como la familia Virtex y Spartan [43]. La figura 4.1 muestra un diagrama funcional del núcleo de Microblaze, dentro de sus características están treinta y dos registros de propósito general de 32-bit, palabra de instrucción de 32-bit con tres operandos y dos modos de direccionamiento, un bus de direcciones de 32-bit, entre otras.

Una de sus principales ventajas es la flexibilidad que ofrece para seleccionar algunas de sus características con lo cual es posible obtener el procesador justo a la medida de las aplicaciones.

Existen dos versiones de Microblaze, con pipeline de 3 etapas el cual está optimizado en área y el de 5 etapas de pipeline para obtener un desempeño más alto. Las dos versiones pueden ser implementadas en cualquier dispositivo Virtex o Spartan y el rendimiento en cada uno se muestra en la Tabla 4.1.

Arquitectura	Rendimiento	Máxima frecuencia de reloj
3- etapa de pipeline	1.19 Dhrystone-MIPs/MHz	235 MHz con Virtex5 FXT
5- etapa de pipeline	0.95 Dhrystone-MIPs/MHz	106 MHz con Spartan3A DSP

Tabla 4.1: Versiones de Microblaze y sus desempeños.

Una de las nuevas características que se ofrece con Microblaze es una unidad de punto flotante (FPU por sus siglas en inglés) que puede ser incluida opcionalmente en caso de ser necesaria. Cuenta con compatibilidad IEEE-754 y es conectada directamente a la instrucción de ejecución del pipeline

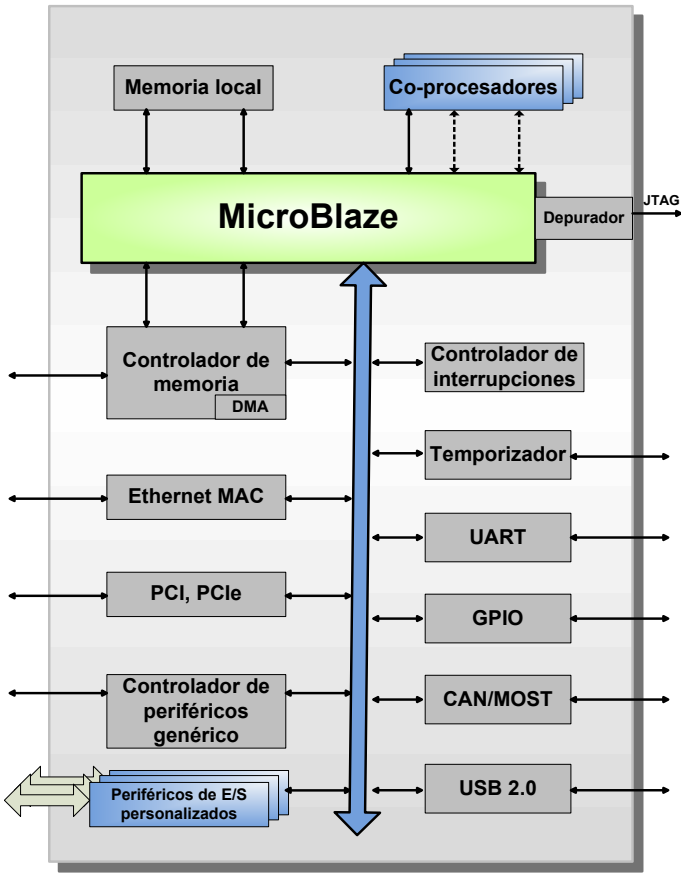


Figura 4.1: Esquema general de un sistema embebido con el procesador Microblaze.

con el fin de asegurar un alto rendimiento, una latencia baja y un diseño compacto; sin embargo, para este trabajo de tesis no fue incluida en la implementación. Además, cuenta con múltiples núcleos IP que permiten darle la flexibilidad deseada en las aplicaciones de sistemas embebidos. Tiene también más de 70 funciones configurables que pueden ser seleccionadas e implementadas si así se desea. Las características configurables son:

- Unidad de administración de memoria
- Unidad de punto flotante
    - Compatibilidad IEEE 754

    - Precisión simple
- Soporte de excepciones de hardware
- Cache de instrucción y de datos
- Divisor de enteros en hardware
- Multiplicador en hardware
- Extensiones al conjunto de instrucciones
    - Comparador de instrucciones
    - Estado de la máquina por registros
    - Acceso atómico
- Múltiples buses ajustables a las diferentes necesidades de rendimiento
    - Bus de memoria local (LMB por sus siglas en inglés Local Memory Bus)
    - Bus local del procesador (PLB Processor Local Bus)
    - 16 FSL (por sus siglas en inglés Fast Simplex Link)
- Señales de interrupción
- Lógica de depuración

Instrucciones

En Microblaze todas las instrucciones son de 32 bits y se definen como tipo A o tipo B. Las tipo A pueden tener de hasta dos operandos como registro fuente y un operando como registro destino. Mientras que las tipo B tienen un operando como registro fuente, un operando inmediato de 16 bits y solo tienen un operando como registro destino.

Las instrucciones de Microblaze están catalogadas por su funcionalidad: aritmética, lógica, saltos, carga-almacenamiento y especiales.

Pipeline

La ejecución de las instrucciones de Microblaze es por medio de segmentación (*pipelining*), aquí para la mayoría de las instrucciones cada etapa toma un ciclo de reloj. Consecuentemente, el número de ciclos necesarios para que una instrucción sea completada es igual al número de etapas que formen el pipeline y cada instrucción es completada cada ciclo de reloj. Son pocas las instrucciones que requieren múltiples ciclos de reloj en la etapa de ejecución para completarse y esto es arreglado mediante un estancamiento de pipeline.

Cuando la lectura de la instrucción es a partir de una memoria lenta la etapa de "fetch" puede tomar varios ciclos de reloj. Esto puede provocar una latencia que afecta directamente la eficiencia del pipeline, por este motivo Microblaze implementa una instrucción de prefetch buffer que reduce el impacto de la latencia de la memoria. Mientras el pipeline está estancado por la instrucción de múltiple ciclos en la etapa de ejecución, el prefetch buffer continua cargando la siguiente instrucción. Cuando el pipeline termina su ejecución la etapa de fetch puede carga la nueva instrucción directamente del buffer de prefetch en lugar de esperarla de la memoria.

En un salto, el pipeline está lleno de instrucciones que no corresponden con el flujo de ejecución. Este tipo de problemas están tomados en cuenta por Microblaze haciendo que cada ocasión que se produce un salto el pipeline sea vaciado. En el procesador se hace uso de la técnica de *delay slots* la cual se trata de un conjunto de instrucciones de salto que permiten la ejecución de la instrucción que les precede, con el objetivo de reducir la penalización de vaciado.

Microblaze cuenta con dos opciones para implementar el pipeline: el de tres etapas y el de cinco etapas. El pipeline de tres etapas es implementado cuando la optimización de área está habilitada, el pipeline es dividido en tres etapas con el fin de minimizar el costo en hardware: extraer, decodificación, ejecución. El pipeline de cinco etapas se implementa cuando la optimización de área se encuentra deshabilitada entonces el pipeline se divide en cinco etapas para maximizar el desempeño: extraer,

decodificación, ejecución, acceso a memoria, reescritura.

Arquitectura de memoria

Microblaze es implementado con una arquitectura de memoria tipo Harvard, los accesos a las instrucciones y los datos se realizan en espacios de direcciones distintos. Cada espacio de dirección tiene un rango de 32 bits, es decir, maneja hasta 4GB de memoria de instrucción y datos respectivamente.

Ambas interfaces (instrucciones y datos) en Microblaze son de 32 bits de ancho y utilizan el formato de *big endian*. Microblaze soporta accesos a la memoria de datos de una palabra, media palabra y un byte. El acceso a datos debe ser alineado a una palabra, a menos que el procesador esté configurado para admitir excepciones de alineación.

Microblaze utiliza dispositivos de E/S mapeados a memoria para realizar sus operaciones. Por tanto, cada operación de E/S se traduce a escribir o leer a los espacios de direccionamiento en los cuales está mapeado el dispositivo. El procesador tiene hasta tres interfaces de acceso a memoria:

- Bus de memoria local (LMB por sus siglas en inglés Local Memory Bus)
- Bus de local del procesador (PLB Processor Local Bus) u OPB (por sus siglas en inglés On-Chip Peripheral Bus)
- Xilinx cacheLink(XCL)

4.2 Descripción general del PLB

El bus local del procesador se trata de un bus de direcciones de 64-bits y un bus datos de 128 bits [11]. El PLB proporciona una interfaz estándar entre los núcleos del procesador y los controladores de bus integrados. PLB es utilizado en aplicaciones específicas de circuitos integrados y en el diseño de sistemas en un chip (SoC por sus siglas en inglés).

Existen dispositivos maestros y esclavos conectados al bus. PLB soporta transferencia de lecturas

y escrituras entre dispositivos maestros y esclavos que están equipados con una interfaz de bus PLB y conectados a través de las señales PLB.

Las señales del PLB están catalogadas de la siguiente forma:

- Señales del sistema: Son tomadas en cuenta por todos los componentes del PLB, así como por los maestros y esclavos que tenga conectados.
- Señales del arbitraje: Estas señales son utilizadas para completar los ciclos de transferencia, con el fin llevar a cabo la entrega de propiedad del bus durante la fase de solicitud o bien indicar el inicio y fin de la etapa de transferencia.
- Señales de estado: Son controladas por el árbitro del PLB y reflejan el estado de propiedad que tienen un maestro. Estas señales son utilizadas por los maestros y esclavos para resolver el arbitraje sobre el bus PLB y otros buses que esten conectados.
- Señales de calificadores de transferencia: Estas señales se utilizan como banderas para la verificación de la calidad en la transferencia de información entre un PLB maestro y un dispositivo esclavo.
- Señales de escritura del bus de datos: Son utilizadas durante el ciclo de escritura. En la transferencia, el dispositivo maestro posiciona la información que será leída en el bus. Posteriormente, el dispositivo esclavo espera a recibir la información para después emitir una señal de verificación.
- Señales de lectura del bus de datos: Son utilizadas en el ciclo de escritura. Durante la transferencia, el esclavo posiciona la información que será leída en el bus. Posteriormente, el maestro espera a recibir la información para después emitir una señal de verificación.
- Señales adicionales de salida para los esclavos: Consiste en un conjunto de señales adicionales definidas para el control de la comunicación entre los dispositivos maestros y esclavos.

Las señales que fueron tomadas en cuenta en el diseño del MCA son mostradas en la tabla 4.2.

Cada dispositivo maestro se añade al PLB a través de buses separados de direcciones, lectura de datos, escritura de datos y por la transferencia de señales de clasificación. Los esclavos se adjuntan

al PLB por medio de la compartición de buses de direcciones, lectura de datos, escritura de datos, y la transferencia de señales de control y estado para cada bus de datos.

El PLB otorga acceso a través de un mecanismo de arbitraje central que permite a los maestros competir por la propiedad del bus. Este mecanismo de arbitraje es lo suficientemente flexible como para prever la aplicación de diferentes esquemas de prioridad. Además, el PLB es un bus totalmente sincronizado, es decir, una sola fuente de reloj proporciona sincronización tanto para todos los módulos del bus como para los maestros y esclavos que son agregados al sistema.

Como se muestra en la Figura 4.2 las interconexiones de PLB proporcionan un vínculo entre el núcleo del procesador y otros periféricos. El sistema consiste en dispositivos maestros PLB, dispositivos maestros OPB y dispositivos esclavos.

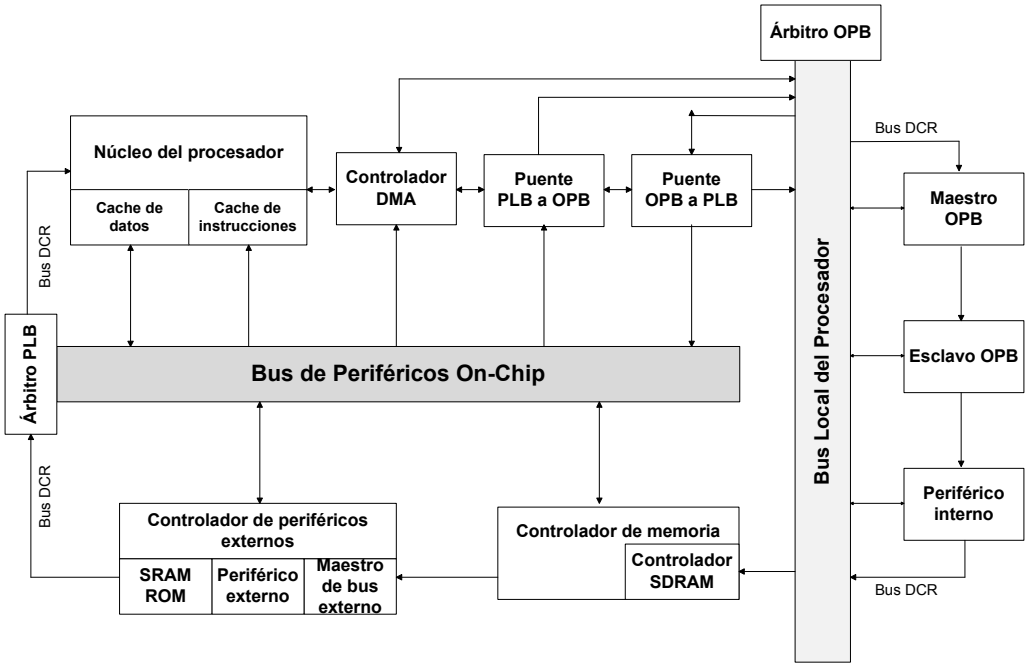


Figura 4.2: Estructura del sistema con PLB.

Nombre de la señal	Descripción
sPLB_Clk	La señal provee la sincronización para el PLB y es entrada para todos los maestros y esclavos, así como del árbitro del bus.
sPLB_Rst	Es la señal para activar el reinicio del árbitro del PLB y llevar a PLB a un estado de reposo.
PLB_PAValid	El árbitro activa esta señal para indicar que esta presente una señal de Mn_request, es decir, que fue entregada la propiedad del bus a un maestro y éste ahora puede hacer una operación hacia un esclavo. Además señala que la dirección principal y la calificación de transferencia es válida.
PLB_abus	Indica la dirección del bus PLB, los 32 bits menos significativos.
PLB_size	Señal manejada por el maestro e indica el tamaño de la transferencia solicitada.
PLB_type	Se usa para que el maestro le indique al esclavo cual es el tipo de transferencia. Existen dos categorías de transferencias: las de transferencia de memoria o las de acceso directo a memoria.
PLB_be	Señal controlada por el maestro. Para una transferencia *nonline* y una *nonburst* la señal identifica que bytes del objetivo que está direccionado son para lectura o para escritura. Cada bit corresponde a una línea del bus de lectura o escritura de datos.
PLB_RNW	El maestro controla esta señal e indica cuando la solicitud se trata de una transferencia de lectura o de escritura. Si PLB_RNW='1' es lectura y cuando PLB_RNW='0' entonces es escritura hacia el esclavo.
PLB_WrDBus	Bus que sirve para realizar la transferencia de datos entre el maestro y el esclavo durante una transferencia tipo escritura.
Sl_addrAck	Señal activada cuando el esclavo ha reconocido la dirección y tomado la dirección, así como los clasificadores de transferencia al final del ciclo de reloj actual. La señal sólo se habilita cuando PLB_PAValid esta activada.
Sl_WrDAck	Indica cuando el esclavo ya no requiere el dato que está actualmente en el bus PLB_WrDBus. El dato ya no es necesario porque el esclavo lo ha almacenado o lo hará al final del ciclo actual de reloj.
Sl_WrComp	El esclavo activa esta señal cuando desea indicar que la transferencia de escritura a sido completada.
Sl_RdDAck	Señal utilizada por el esclavo para advertir que los datos en el bus Sl_RdDBus son válidos y lo serán hasta el fin del ciclo actual de reloj
Sl_RdComp	El esclavo controla esta señal para avisar al árbitro del PLB que la transferencia de lectura ha sido completada correctamente o bien que será completada al final de actual ciclo de reloj.

Tabla 4.2: Descripción de las señales del PLB utilizadas en el diseño del MCA.

PLB es un bus de alto desempeño que es utilizado para el acceso a memoria a través de unidades de interfaz de bus. El controlador externo de periféricos y el controlador de memoria son catalogados como esclavos del PLB, estos esclavos como se muestra en la figura 4.2. El núcleo del procesador posee dos conexiones PLB tipo maestro, una para el cache de instrucciones y otra para la cache de datos. El controlador de DMA (por sus siglas en inglés Direct Memory Access) se trata de un dispositivo maestro PLB.

Los periféricos de bajo desempeño, en donde también existen maestros y esclavos que son adjuntados al OPB. Un puente entre el PLB y el OPB permite a los maestros PLB transferir datos hacia los esclavos OPB. En la figura 4.2 se muestran dos puentes. El primero es PLB-OPB el cual es esclavo en PLB. El otro puente es OPB-PLB que funciona como esclavo en OPB y maestro en PLB.

El dispositivo de registro de control DRC es utilizado principalmente para el acceso a los registros de estado y control de varios dispositivos PLB y OPB.

4.2.1 Implementación del PLB

La implementación del PLB es un núcleo en donde todos los maestros y esclavos son agregados. La lógica del núcleo del PLB consiste de un árbitro central del bus, así como de unidades de trayectoria de direcciones, escrituras, lecturas, un módulo de control de bus, temporizador de vigilancia, entre otros.

La figura 4.3 provee un diagrama a bloques del PLB. Algunos componentes importantes en el diagrama son la unidad de direccionamiento (Address Path Unit), contiene la información necesaria para seleccionar una dirección maestra la cual es dirigida hacia los dispositivos esclavos en el PLB. La unidad de escritura, contiene la lógica necesaria para la escritura tanto para los dispositivos maestros como para los esclavos. La unidad de lectura, contiene la lógica necesaria para la lectura de información del bus. Finalmente, tenemos la unidad de control del bus, se encarga del control y direccionamiento del flujo de información a través del PLB y DCRs.

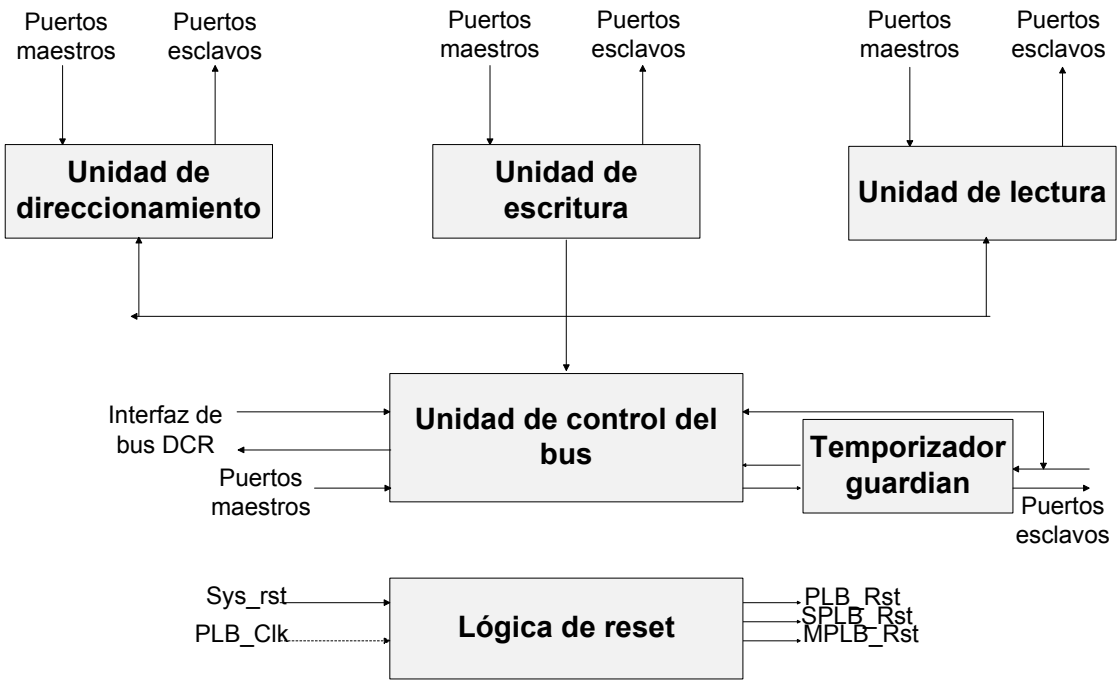


Figura 4.3: PLB diagrama a bloques

Por otra parte la arquitectura del PLB soporta hasta 16 dispositivos maestros y 8 esclavos. La Figura 4.4 es un ejemplo de las conexiones al PLB [42]. En la figura se pueden observar claramente 3 bloques, el de la izquierda se trata de los maestros del bus que se comunican con el núcleo del PLB (descrito en la figura anterior) mediante señales de arbitraje, calificadores de transferencia, escritura del bus de datos y control. El PLB (bloque central) se comunica con los maestros por medio de las señales de escritura de datos, estado y control. Mientras tanto los esclavos (bloque derecho) conectados al PLB reciben información mediante las señales de calificadores de transferencia, control y escritura de datos. Los esclavos pueden enviar señales tanto al núcleo del PLB como a los maestros, realizan escritura de datos al bus y envian señales del control y estado.

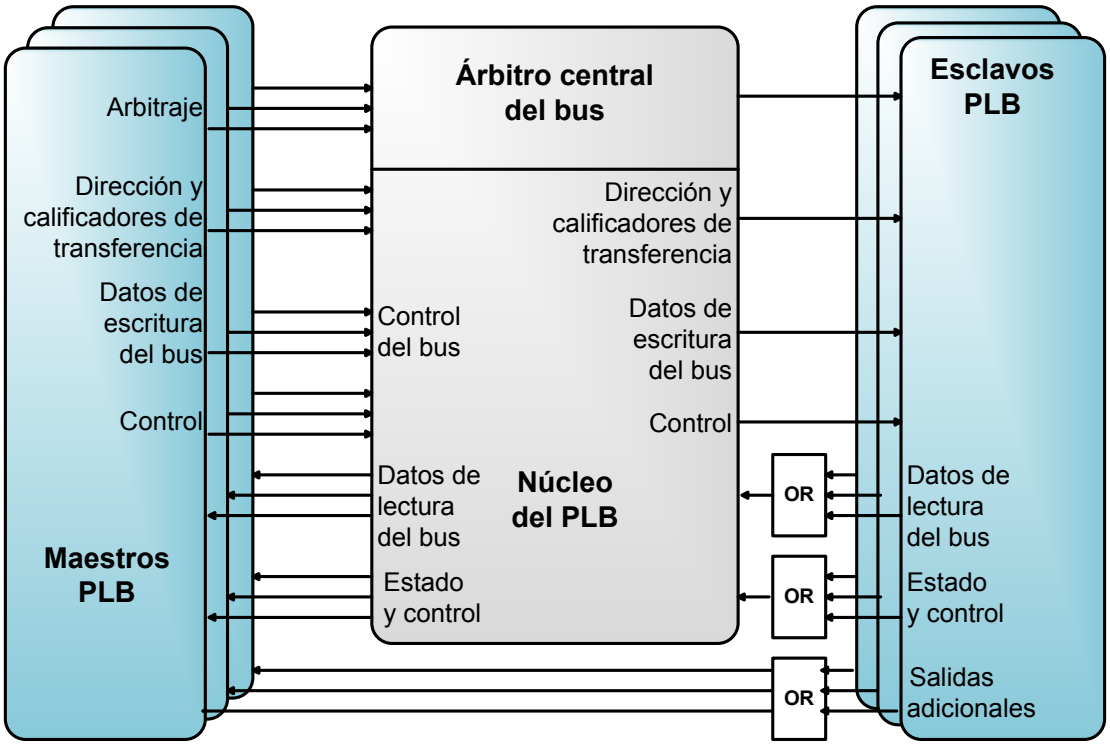


Figura 4.4: Digrama de interconexiones del PLB con maestros y esclavos.

4.2.2 Protocolo de transferencia del PLB

Una transacción de PLB se compone del ciclo de direccionamiento y el ciclo de datos, como se muestra en la Figura 4.5. El ciclo de direccionamiento tiene tres fases: solicitud, transferencia, y reconocimiento de dirección. Una transacción de PLB comienza cuando un maestro maneja una dirección, transfiere señales de clasificación y solicita propiedad del bus durante la fase de solicitud del ciclo. Después de que el árbitro del PLB concede la propiedad del bus, la dirección y las señales de clasificación son pasadas al dispositivo esclavo, esto en la fase de transferencia. Por último, el ciclo es concluido en la fase de reconocimiento de dirección cuando el esclavo toma la dirección y transfiere clasificadores al maestro.

El ciclo de datos consiste en dos fases denominadas de transferencia y de reconocimiento de datos. La fase de transferencia se trata de que el maestro maneje el bus de escritura de datos para una transferencia de escritura o bien el de lectura de datos para la transferencia de lectura. Las señales de reconocimiento de datos son requeridas para ser por concluida la fase de reconocimiento de datos.

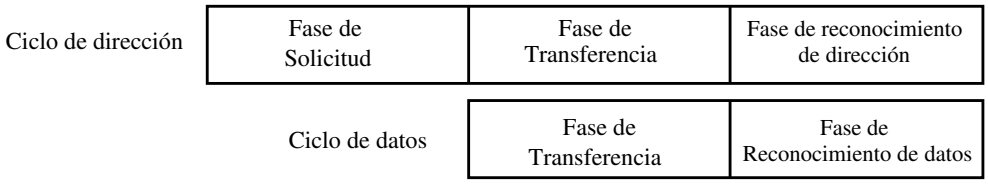


Figura 4.5: Ciclos de transferencia para el PLB.

4.3 Descripción general de la arquitectura del MCA

Con objetivo de realizar el diseño en hardware del módulo de control de acceso, el primer paso fue realizar un análisis del VMM Xen para verificar cuáles son las políticas que son ejercidas en una implementación en software y así conocer cuáles son más factibles de implementar a nivel de hardware. Para la especificación del diseño del MCA, se trató de definir los datos necesarios para que el módulo pueda identificar correctamente hacia que dispositivos se está haciendo una solicitud de E/S.

A partir del análisis realizado a las políticas de seguridad se encontró que los mecanismos de seguridad más utilizados son las matrices de control de acceso y las listas de control de acceso (ver capítulo 3). Para el caso de las matrices de control de acceso es posible tener dentro de ella todos los casos que podrían presentarse en el sistema y así poder tomar una decisión; sin embargo, la desventaja es que necesitan una memoria muy grande para su implementación, aunque a nivel de software el acceso a la decisión es relativamente rápido. En cambio, las listas de control de acceso solucionan el problema de uso excesivo de memoria cargando en cada localidad de la lista una regla

que contenga la información necesaria para poder tomar una decisión. Sin embargo, considerando lo anterior el acceso a cada regla tendría que ser secuencial y el tiempo para poder obtener una regla sería, en el peor de los casos, igual al número de reglas cargadas num_{reglas}.

El mecanismo que utilizado en esta tesis es las listas de control de acceso, para poder realizar un diseño correcto del MCA se tomaron en cuenta los siguientes aspectos:

- Cómo cargar y almacenar las reglas al MCA.
- Encontrar una ubicación dentro del sistema de buses, que fuera adecuada para el módulo.
- Tomar provecho de paralelismo que puede ser implementado a nivel de hardware.
- Como realizar la codificación de las reglas.
- Envío de señales y su control, para detectar la decisión del MCA.

4.3.1 Ubicación del MCA

Un módulo en hardware para el manejo del control de acceso de E/S, ha sido incrustado dentro de la arquitectura del sistema, básicamente entre el procesador, el bus o los dispositivos. En la figura 4.6 se presenta un diagrama a bloques que contiene la arquitectura general de un sistema de cómputo.

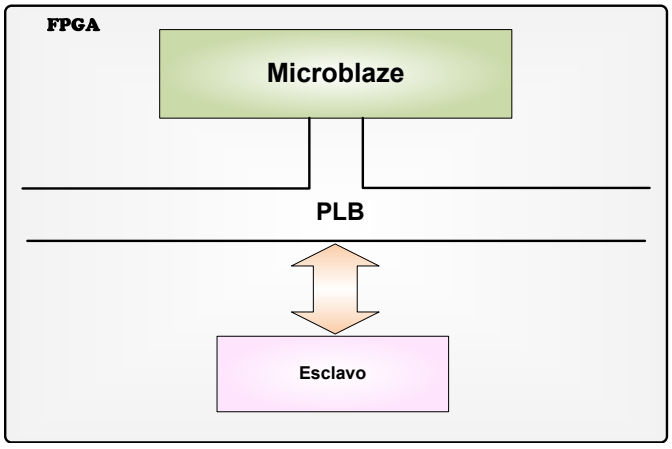


Figura 4.6: Diagrama de bloques de un sistema de cómputo típico

La ubicación del MCA dentro de la arquitectura del sistema es una decisión importante, de ello depende las señales que se tomarán en cuenta, el diseño de las reglas y el nivel de protección que el MCA ofrecerá al sistema. Las tres opciones iniciales para la ubicación del MCA se muestran en la Figura 4.7.

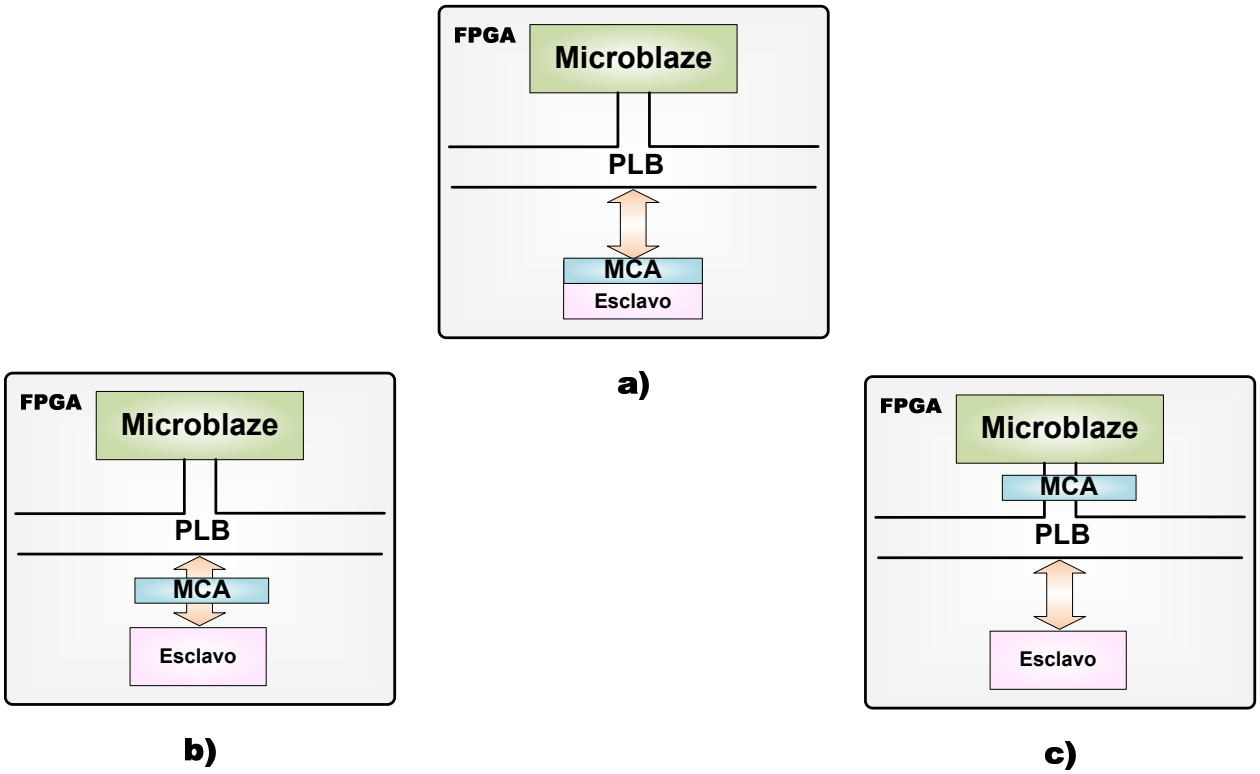


Figura 4.7: Esquemas de las alternativas de ubicación del MCA

En la Figura 4.7 inciso a) se muestra la alternativa de colocar el MCA internamente en el controlador de hardware del dispositivo, lo que permitiría proteger el acceso hacia el dispositivo modificado, la desventaja es que el diseño del MCA sería específico para cada controlador de hardware. El inciso b) representa la forma de colocar el MCA en el acoplador de señales entre el bus PLB y el controlador del dispositivo. Esta opción tiene la facilidad de que cada dispositivo que esté conectado a través del acoplador podrá ser controlado por el MCA pero su desventaja es que existen varios tipos de acopladores y se tendría que incluir un módulo para cada uno de ellos. Por último, la tercera alternativa c) propone colocar el MCA entre el procesador y el bus PLB, en esta ubicación

el módulo puede interceptar las instrucciones que viene desde el procesador y analizar cuales sí están autorizadas. La principal ventaja en esta tercer alternativa es que no es necesario diseñar un MCA para un dispositivo o acoplador específico, basta con analizar las instrucciones enviadas por el procesador para poder brindar el control de acceso a los dispositivos de una manera más genérica.

En este trabajo se implementaron las tres alternativas, aunque en la documentación de la tesis únicamente se detallará el diseño de la implementación del MCA entre el procesador y el bus PLB, debido a que es la solución que otorga más generalidad.

4.3.2 Estructura de la regla

La estructura de la regla es un aspecto importante para el diseño del MCA debido a que la regla debe de contener la información suficiente para poder detectar los casos que se quiere bloquear o permitir una operación hacia un esclavo. La regla debe contar con la instrucción que se quiere analizar y con la decisión que será ejecutada sobre esa instrucción. La figura 4.8 muestra la estructura de la regla general para las reglas de la lista de control de acceso.

Tipo de operación	**Dirección de acceso**	**Máquina virtual**	**Decisión**
4 bits	16 bits	8 bits	1 bit

Figura 4.8: Estructura de la regla.

Debido a que en Microblaze las entradas y salidas son mapeadas en memoria cada dispositivo esclavo en el sistema cuenta con un rango de dirección que puede ser accedido por cualquier maestro propietario del bus.

El campo de **Dirección de acceso** permite distinguir entre un dispositivo u otro, ya que cada uno está en un rango de diferente, se puede poner como valor la dirección base de cualquiera de los registro con acceso por software del dispositivo. En el campo de **Tipo de operación** sólo están las opciones de escritura y lectura. Para indicar una lectura se pone el campo en '1' y en '0' para señalar escritura. Para configurar la regla con la **Máquina virtual**, se debe de colocar ahí el identificador de la máquina virtual que está haciendo la solicitud. El campo de decisión consiste en un sólo bit, el cual cuando tiene un valor de '1' significa que la operación es admitida y en '0' la operación es

denegada.

El módulo de control de acceso se encargará de formar con las señales de entrada la instrucción de manera combinacional, la cual será comparada con las reglas cargadas. En la Figura 4.9 se encuentra el diagrama de tiempos para formar la instrucción de entrada, en donde las señales involucradas están etiquetadas. A continuación se da una descripción a detalle:

- El maestro que para el caso de nuestro sistema se trata típicamente del procesador, solicita al árbitro propiedad sobre el bus. Esta solicitud siempre va acompañada de la dirección a la que quiere acceder, cuando el árbitro de control valida la dirección y pone en activo la señal `plb_PAValid` (`etiqueta 1`) entonces la señal `plb_abus` contiene la dirección a la que se quiere tener acceso, y es justo en ese momento cuando el nibble alto es tomado para conformar la instrucción de entrada.

- El tipo de operación también debe ser especificado por el maestro a la hora de solicitar la propiedad del bus. El protocolo de PLB sólo maneja dos tipos de operación: lectura y escritura. Esta señal es la que fue denominada como `plb_RNW` (`etiqueta 2`) en la sección. Como el campo reservado para el tipo de operación es de 4 bits, el bit más significativo corresponde al valor de `plb_RNW` y los otros tres permanecen en cero.

- En el caso del campo del identificador de máquina virtual, se hace la suposición de que el procesador y el PLB cuentan con un transacción del tipo "máquina virtual" (`etiqueta 3`) en la cual se incluye la información de identificación de la máquina virtual que está haciendo la operación.

- En el momento de la señal `plb_PAValid`, el MCA toma el resto de la señales y forma la `instruccion` (`etiqueta 4`) de forma combinacional, durante el mismo ciclo que `plb_PAValid` se activó.

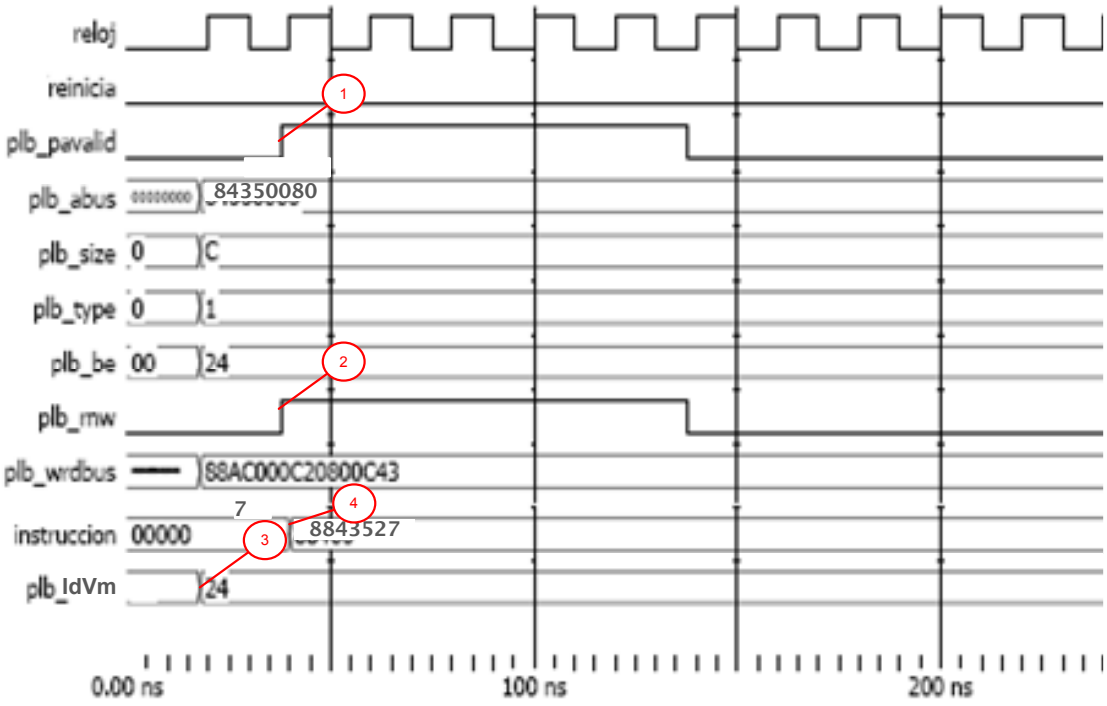


Figura 4.9: Diagrama de tiempos para formar la instrucción de entrada

4.4 Diseño general del MCA

La figura 4.10 contiene un diagrama funcional del módulo de control de acceso, el cual ilustra que el MCA está compuesto de una ACL (Access Control List) de donde se extraen las reglas para los equipos de cómputo. El MCA va estar situado entre las señales que vienen del procesador y las señales que van hacia los dispositivos esclavos del sistema, ya que en esta posición el MCA logra identificar todas las operaciones hacia los dispositivos de E/S. Las entradas de este módulo son las solicitudes de operación y datos en el bus maestro, mientras que por parte del esclavo el MCA recibe los datos generados en sus procedimientos y señales de reconocimiento, de dirección, escritura o lectura. Las salidas también van en dos sentidos, las del grupo de solicitud y datos al esclavo que son señales que su valor siempre es ceros al menos que el módulo de control después de analizar las señales de entrada determine que la operación es permitida, en ese caso las señales de salida hacia el esclavo toman el valor de las de entrada provenientes del PLB.

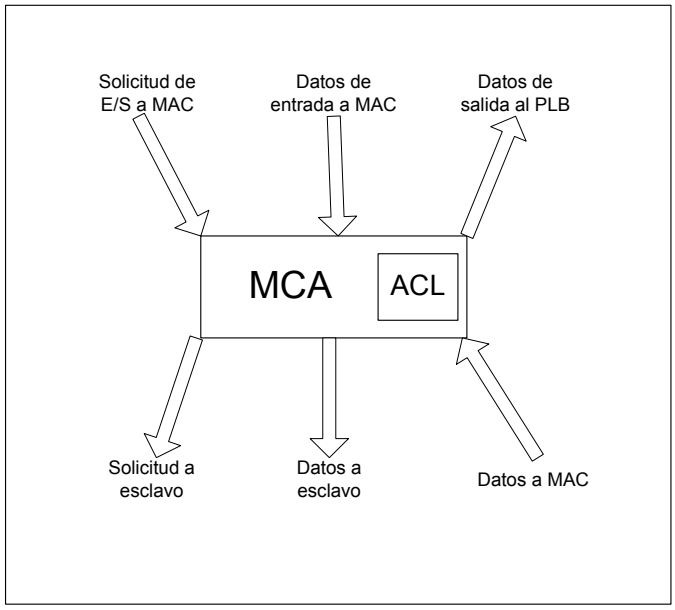


Figura 4.10: Diagrama a funcional del MCA

Las ACL son implementadas en memorias. Una parte fundamental en la mayoría de los sistemas digitales es el almacenamiento y recuperación de la información que está procesando en dicho sistema. En general, los principales objetivos que se persiguen en el momento de diseñar o utilizar una memoria son:

- Alta velocidad de acceso
- Bajo precio
- Gran capacidad de almacenamiento
- Bajo consumo de recursos y energía

Cada uno de estos objetivos se conseguirá en mayor o menor medida dependiendo del medio físico empleado, su organización, tecnología, étc. Para este trabajo lo importante la velocidad de acceso a las reglas almacenadas en memoria, ya que el MCA no conoce si la memoria contiene una regla que coincide con la instrucción de entrada y mucho menos en que posición de la memoria está.

Para resolver este problema fueron diseñadas e implementadas dos versiones del MCA, las cuales se diferencian por el tipo de lectura de la información almacenada.

- MCA-RAM-4. Recibe este nombre debido a que está diseñado con una memoria de acceso aleatorio (RAM por sus siglas en inglés), y la lógica que utiliza para hacer la búsqueda de la regla sobre la memoria no depende del número de reglas almacenadas sino de cuatro procesos que se encargan de leer una localidad de memoria diferente en el mismo ciclo de reloj.

- MCA-CAM-n. Este se trata de un diseño donde la ACL está implementada con una memoria CAM. La lógica utilizada por este diseño para realizar la búsqueda de una regla que coincida con la instrucción de entrada, depende directamente con el número de reglas que contenga el sistema.

En las siguientes sub-secciones se describen con más detalle ambos diseños del MCA, mostrando la arquitectura para cada uno, así como sus ventajas y desventajas.

4.4.1 MCA-RAM-4

Este diseño trata de proporcionar un equilibrio entre el tiempo de ejecución y el área utilizada para la implementación del MCA-RAM-4, además de explotar el paralelismo implícito en el hardware. En la Figura 4.11 se muestra el diagrama general de la arquitectura de este diseño.

El diseño del módulo de control de acceso consta de tres bloques principales que son la unidad de control, la memoria y una unidad de lógica de apoyo. La unidad de control es la encargada de la coordinación entre la memoria y la lógica de apoyo, la memoria RAM es necesaria para el almacenamiento de las reglas y los procesos de análisis son para hacer todas las operaciones adicionales que sean necesarias para obtener un correcto funcionamiento del MCA-RAM-4.

En seguida da una descripción del diseño de cada uno de los bloques presentados en el diagrama 4.11. Adicionalmente la Tabla 4.3 contiene una descripción de las señales utilizadas en este diseño.

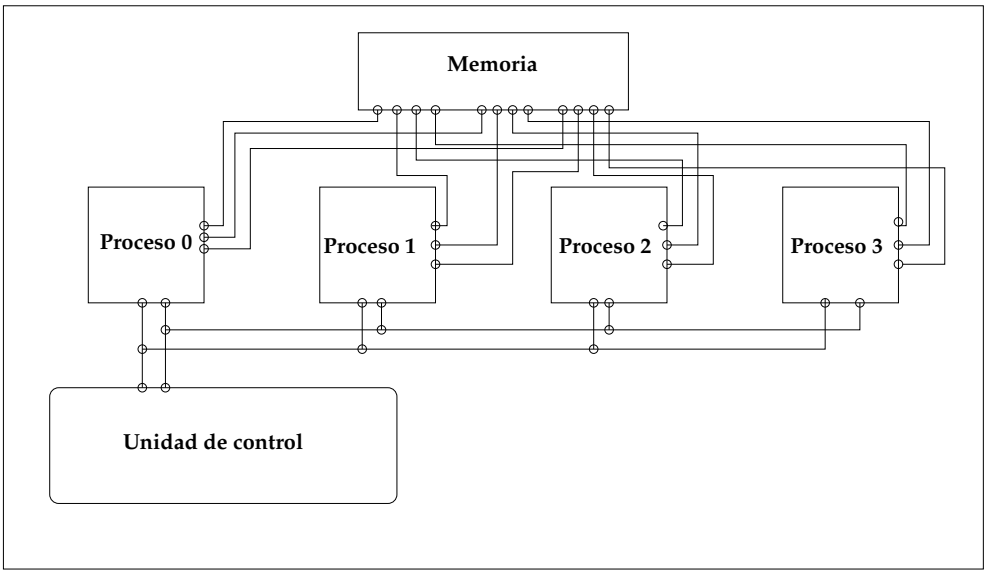


Figura 4.11: Organización de la arquitectura del MCA-RAM-4

Unidad de control

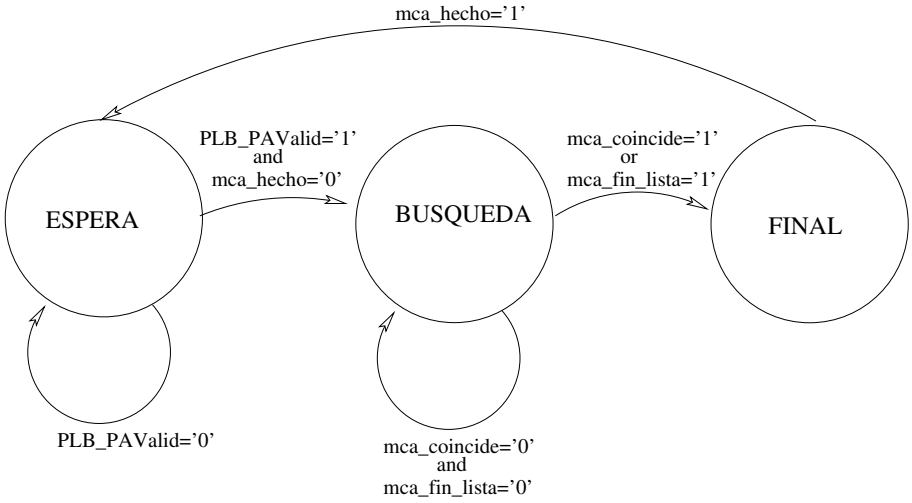


Figura 4.12: Diagrama de estados de la unidad de control del MCA-RAM-4

El MCA-RAM-4 está diseñado con una unidad de control que consta de una máquina de estados finitos (MEF) con tres estados denominados **ESPERA**, **BUSQUEDA** y **FINAL**. Esta máquina

MCARAM4_reloj	Es la señal de sincronización del proceso del MCA-RAM-4
plb_pavalid	Señal de entrada al MCA-RAM-4. Cuando se encuentra activa sirve como señal de inicio para el MCA-RAM-4 .
plb_abus	Señal tomada de PLB_abus, es entrada al módulo y sirve para construir la instrucción que el módulo de control comparará con las reglas que tenga almacenadas. Cuando la decisión del MCA-RAM-4 sea permitir la operación señal es conectada a la señal de salida plb_abus_out.
plb_rnw	Señal de entrada que indica que tipo de operación se realizará (lectura o escritura). Además es utilizada para formar la instrucción del MCA-RAM-4.
MCARAM4_state	Señal auxiliar para conocer en que estado se encuentra la MEF.
MCARAM4_next_state	Señal auxiliar para conocer cual será el estado de la MEF en el siguiente ciclo de reloj.
MCARAM4_hecho	Cuando la señal se activa significa que el MCA-RAM-4 ha terminado su proceso de búsqueda de regla. La señal regresa en valor '0' cuando el MCA-RAM-4 termina completamente el ciclo de transferencia del PLB.
MCARAM4_instruccion	Es conformada por las señales plb_rnw y plb_abus, se trata de la instrucción de entrada del MCA-RAM-4
MCARAM4_fin_lista	Señal que se activa cuando se llego al final de la lista de reglas.
MCARAM4_accion_in	Es una señal de cuatro bits que contiene el campo de decisión para cada regla examinada por uno de los cuatro procesos. Cuando se encuentra una coincidencia la señal es analizada para determinar cual es la acción que corresponde a la regla que coincidió.
MCARAM4_accion_al	Esta señal es la acción que va ser ejecutada por el MCA-RAM-4, después de que se hizo el análisis de accion_in
MCARAM4_coincide	Es un señal de cinco bits, en donde los cuatro bits menos significativos tienen correspondencia directa con el número de proceso que realiza el análisis, en cada bit el proceso almacena si en ese ciclo encontró alguna coincidencia o no.

Tabla 4.3: Descripción de las señales utilizadas en este diseño MCA-RAM-4

de estados se encuentra totalmente sincronizada con la señal MCARAM4_reloj la cual es la fuente de reloj que alimenta al PLB conocida como sPLB_CLK. Cada cambio de estado puede llevarse a cabo con flanco de subida del reloj. La única señal a la que MEF responde de forma asíncrona es mca_reinicia, cuando esta señal se encuentra activa la máquina se reinicia y se va a su estado inicial **ESPERA**, las señales de control son inicializadas. Un diagrama de estados es presentado en la figura 4.12.

Básicamente la operación que debe realizar el MCA-RAM-4 es esperar una señal de inicio para que con las señales de entrada forme una instrucción que será comparada con todas las reglas cargadas en el sistema. Cuando el campo de instrucción en la regla coincide con la instrucción de entrada entonces se extraerá de la regla la decisión para que sea aplicada a la solicitud, en caso que no sea encontrada entonces el MCA-RAM-4 debe contar con una decisión por defecto que indique que acción tomar con la operación.

El estado de **ESPERA** se encarga de inicializar todos los registros del MCA-RAM-4, así como formar la instrucción a comparar con las reglas almacenadas, cuando se detecta que la señal de `plb_pavalid` tiene valor de '1' entonces se pasa la instrucción válida a `MCARAM4_instruccion`.

BUSQUEDA es el estado donde se realiza la búsqueda de la instrucción formada en el estado anterior con alguna de las reglas almacenadas en el MCA-RAM-4. Este estado puede durar de uno a varios ciclos de reloj.

FINAL se encarga de identificar cual es la acción que se debe de tomar. Toma en cuenta las señales generadas en el estado anterior para conocer si la instrucción de entrada coincidió con algunas de las reglas almacenadas. Esta información permite tomar la decisión en cuanto a si debe o no extraer el contenido del campo de decisión y ejecutar la acción que ésta indique o bien tomar la acción por defecto.

En caso que el campo de decisión indique que la solicitud de E/S es aceptada, es decir, el bit sea igual a '1' la MEF genera las señales de control para permitir transferir las señales de entrada hacia las de salida del MCA-RAM-4, y así permitir que la operación se ejecute transparentemente tanto para el maestro como para el esclavo. Cuando el campo de decisión sea igual a '0' la MEF genera señales de control para no permitir la transferencia de las señales de entrada hacia las de salida. En este caso, genera internamente las señales que el esclavo respodería en caso que le hubiera llegado una instrucción. Estas señales son enviadas a la unidad de control de PLB, el cual continua con su proceso creyendo que los datos recibidos son cero.

En el caso que se tome la acción por defecto las señales generadas por la MEF dependerían de cual sea la acción a tomar: aceptar o rechazar la operación. Para todos los casos el estado de **FINAL** se encarga de verificar que el protocolo de transferencia sea finalizado correctamente.

Por otro lado existen señales de control que son tomadas en cuenta como condiciones de cambio de estado. Básicamente las señales consideradas en la MEF son:

- `plb_pavalid`. La señal es tomada en cuenta en el estado de **ESPERA** e indica que existe una dirección válida para ser analizada por el MCA-RAM-4 y verifica si la operación está permitida. Mientras que el valor de esta señal no sea '1' la MFE permanecerá en el estado **ESPERA**.

- `MCARAM4_fin_lista`. Indica que han sido analizadas todas las reglas del MCA-RAM-4, en el caso de que la señal se ponga en activo y `mca_coincide` tiene un valor de '0' entonces el MCA-RAM-4 tomará la acción por defecto para la operación ya que esto significa que la instrucción de entrada no se encuentra contemplada en las reglas. Esta señal es generada en el estado de **BUSQUEDA** y tomada en cuenta en **FINAL**.

- `MCARAM4_coincide`. Señala que durante la búsqueda en la lista de control se encontró una regla que coincide con la instrucción de entrada. En caso que el valor de la señal sea '1' entonces el módulo toma la regla que coincidió y la analiza para saber la acción sobre la operación solicitada.

- `MCARAM4_hecho`. La señal es generada durante el estado **FINAL** y su función es mostrar que se ha tomado una decisión sobre la instrucción de entrada, ya sea permitir la operación hacia el esclavo o bien bloquearla.

Memoria

La sincronización para la lectura de la memoria está dada por la señal `MCARAM4_reloj` que está conectada a `sPLB_Clk`. Existen señales de entrada a la memoria las cuales son controladas por el MCA-RAM-4, la activación de estas señales depende del estado en que se encuentre la MEF.

La Figura 4.13 ilustra el diagrama funcional de la memoria utilizada en este diseño. La memoria fue diseñada con cuatro puertos de lectura que están conformados de cuatro señales de control que son las señales de habilitación de lectura para cada puerto, cuando cualquiera de estas señales

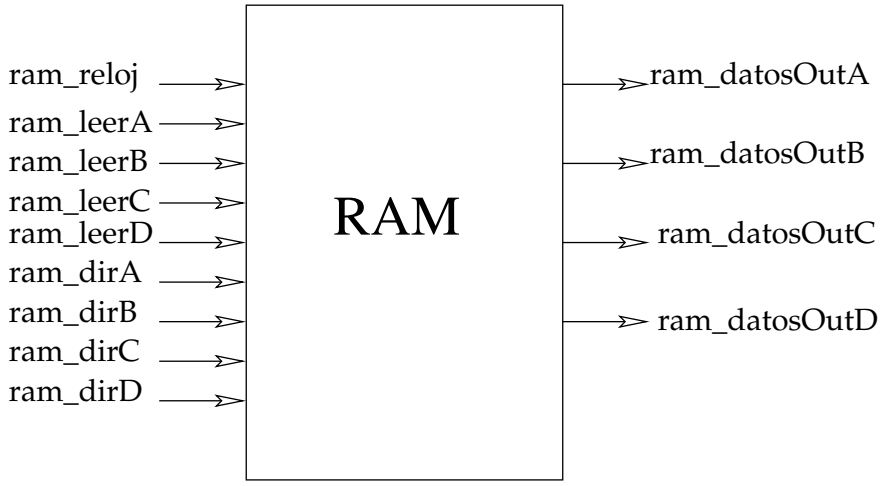


Figura 4.13: Diagrama funcional de la memoria RAM

esté activada significa que está permitido hacer la operación de lectura en ese puerto. Las señales son nombradas `ram_leerA`, `ram_leerB`, `ram_leerC`, `ram_leerD`. Otras señales de entrada son las de la dirección de lectura: `ram_dirA`, `ram_dirB`, `ram_dirC`, `ram_dirD`. Las salidas de la memoria son los datos leídos de cada puerto `ram_datosOutA`, `ram_datosOutB`, `ram_datosOutC`, `ram_datosOutD`.

La lógica interna de la memoria se muestra en la Figura 4.14. La instrucción formada por la unidad de control consiste de cuatro decodificadores, cada uno es para una dirección diferente de entrada. Cada decodificador apunta a la dirección de lectura y cuando la entrada de control lo indique la memoria pasa los datos almacenados a los registros `DatosOut_X`. Los datos de salida son tomados por los procesos del MCA-RAM-4.

Procesos de análisis

Son cuatros los procesos de análisis y son denominados `proceso_0`, `proceso_1`, `proceso_2`, `proceso_3`. Éstos se ejecutan concurrentemente y son sensibles al cambio de la señal `MCARAM4_reloj`, esto es con el fin de tenerlos sincronizados. La búsqueda de la regla que coincide con la instrucción de entrada se realiza de forma simultánea para 4 reglas secuenciales. Así, en cada ciclo de reloj está analizando una regla diferente.

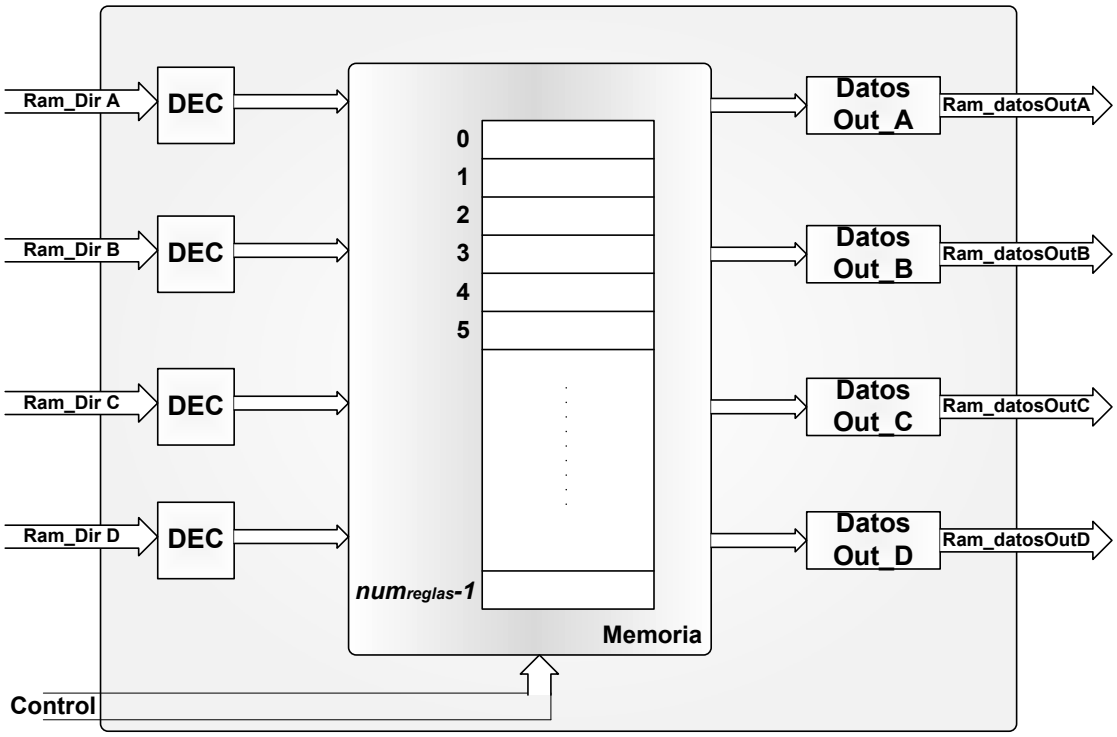


Figura 4.14: Diagrama del diseño interno de la memoria RAM

El MCA-RAM-4 cuenta con dos registros de suma importancia: `mca_regCoincide`, `mca_regAccion_in`. El registro `mca_coincide` consta de 5 bits, los 4 bits menos significativos tienen correspondencia directa a cada uno de los procesos. Cuando la regla que se está analizando coincida con la instrucción de entrada entonces el proceso pone en activo el bit que le corresponde del registro. El quinto bit es simplemente una OR de los demás bits, y es el bit que el proceso principal (`proceso_0`) verifica para saber si hubo una coincidencia.

El registro de `mca_regAccion_in` es de 4 bits e igualmente cada bit tiene correspondencia directa con los procesos de análisis de regla. El objetivo de este registro es que cuando el proceso detecte la coincidencia entre la instrucción y la regla que esta analizando coloque en el bit correspondiente la información extraída del campo de decisión de la regla.

Los procesos de análisis de reglas se encargan de mandar y recibir las señales a la memoria, cada proceso tiene un número diferente con el fin de que el índice de la regla que analicen siempre

sea diferente. Durante el estado de **ESPERA** todas las señales son reinicializadas y cada proceso tiene control de la señal para habilitar la lectura del puerto que le corresponde. Cuando la MEF se encuentre en el estado de **BUSQUEDA** cada proceso hace la lectura de su puerto, dándole como entrada el índice de la regla que debe analizar. En el caso de la que regla analizada coincida con la instrucción de entrada el proceso modifica los registros `mca_regCoindice` y `mca_regAccion_in`, sino, cada proceso de análisis incrementa en 4 el valor de su índice para que en el siguiente ciclo de reloj analice la siguiente regla que le corresponde. En la figura 4.15 entrada un diagrama de la arquitectura interna de los procesos de análisis.

Como se vio en la sección 4.4.1 la condición para el cambio del estado **ESPERA** al estado **FINAL** es que cualquiera de las señales `mca_coincide` o `fin_lista` estén activas. La señal `MCARAM4_fin_lista` se activa cuando el contador de $num_{lecturas}$ es igual al número total de reglas y la de `MCARAM4_coincide` está conectado al bit más significativo del registro `mca_regCoincide`. Adicionalmente, cuando el proceso principal llega al estado **FINAL** se tienen dos alternativas, la primera es que la instrucción de entrada no coincidía con ninguna de las reglas almacenada, en ese caso el MCA-RAM-4 debe de entregar como resultado la acción por defecto, ésta puede ser permitir la operación o bien bloquearla. La segunda alternativa es cuando Si se llegó a estado **FINAL** debido a que la señal `mca_coincide` se activó entonces, se analiza el registro `mca_regCoincide` para seleccionar cual de los procesos encontró la regla y poder tomar la decisión que se indicaba en la regla encontrada.

El valor del índice está dado por la ecuación ($num_{proceso} + 4 * num_{lectura}$); donde $num_{proceso}$ se trata del identificador numérico del proceso que está haciendo el análisis de la regla y el num_{reglas} se trata del valor de un contador que lleva el proceso principal, el cual su valor inicial es cero y es incrementado en 4 cada ciclo de reloj que se realiza la lectura de los 4 puertos.

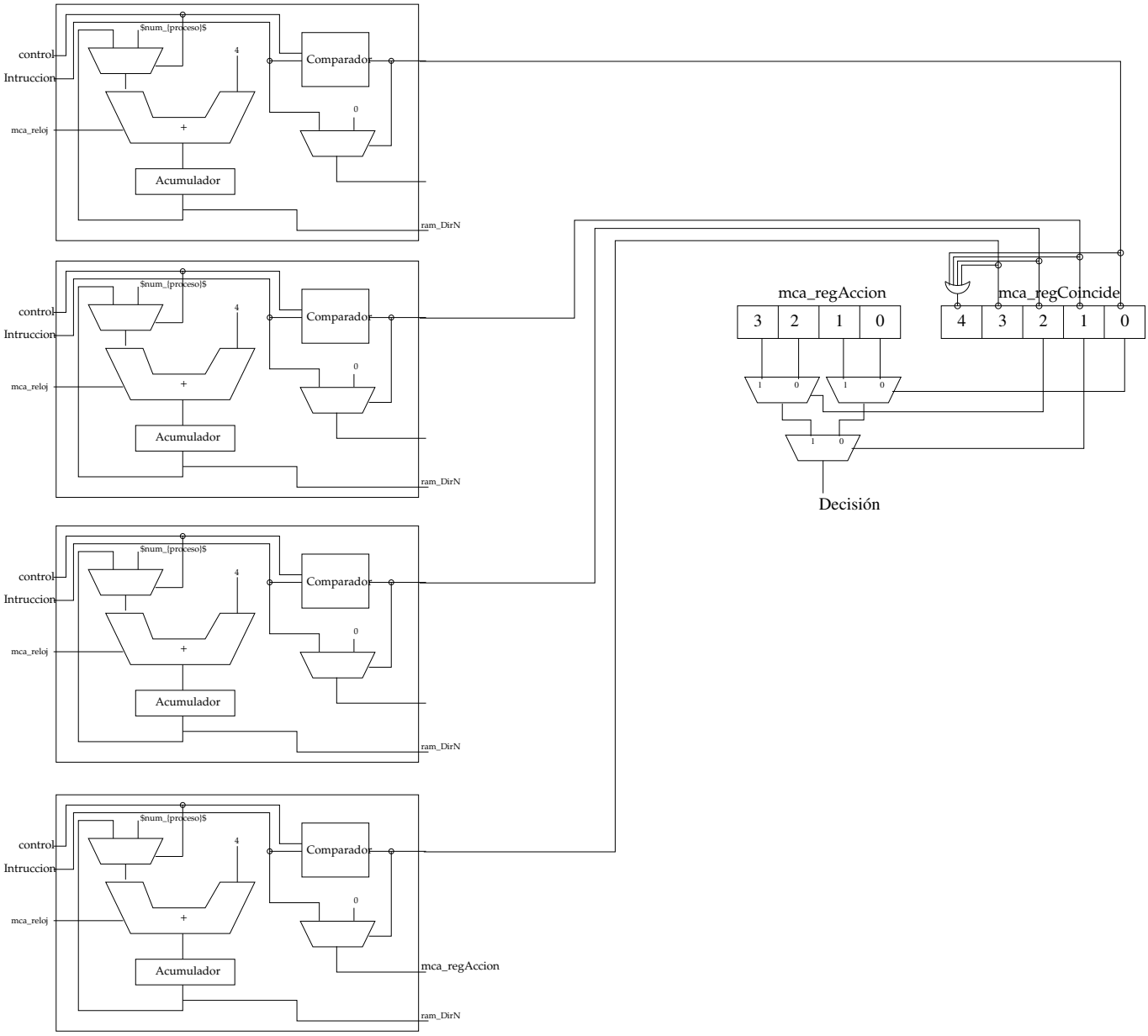


Figura 4.15: Esquema interno de los procesos de análisis

4.4.2 MCA-CAM-n

La figura 4.16 muestra el esquema general del arquitectura del MCA-CAM-n.

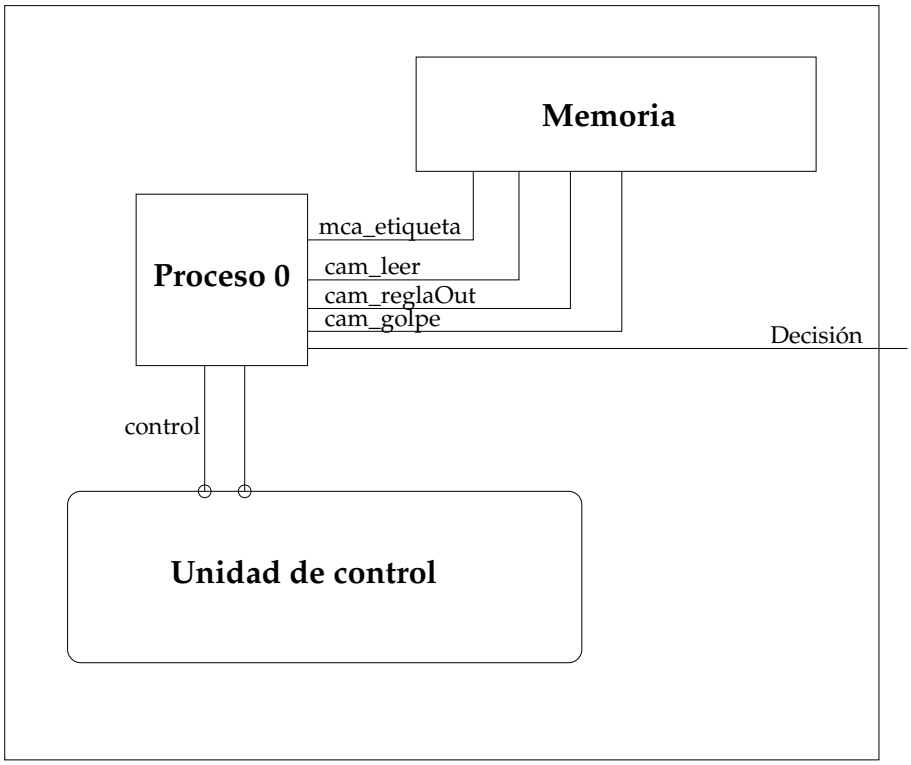


Figura 4.16: Diagrama general de la arquitectura de MCA-CAM-n

El diseño del MCA-CAM-n también consta de tres bloques: memoria, unidad de control, y un proceso de coordinación. La unidad de control va ser la encargada de coordinar las señales de control entre los bloques que componen el MCA-CAM-n, la memoria es necesaria para el almacenamiento de las reglas y recuperación de la decisión y el proceso de coordinación del MCA-CAM-n.

En las siguientes sub-secciones se da una descripción del diseño de cada uno de los bloques presentados en el diagrama 4.16. Adicionalmente la Tabla 5.5 contiene una descripción de las señales utilizadas en este diseño.

plb_pavalid	Señal de entrada al MCA-CAM-n, además de la funcionalidad de la señal PLB_PAValid, cuando se encuentra activa sirve como señal de inicio para el MCA-CAM-n .
plb_abus	Señal tomada de PLB_abus, es entrada al módulo y sirve para construir la instrucción que el módulo de control comparará con las reglas que tenga almacenadas. Cuando la decisión del MCA-CAM-n sea permitir la operación , esta señal es conectada a la señal /se de salida plb_abus_out.
plb_rnw	Señal de entrada que indica que tipo de operación se realizará (lectura o escritura). Además es utilizada para formar la instrucción del MCA-CAM-n.
CAM_state	Señal auxiliar para conocer en que estado se encuentra la MEF.
CAM_next_state	Señal auxiliar para conocer cual será el estado de la MEF en el siguiente ciclo de reloj.
CAM_instruccion	Es conformada por las señales plb_rnw y plb_abus, se trata de la instrucción de entrada del MCA-CAM-n
CAM_hecho	Cuando la señal se activa significa que el MCA-CAM-n ha terminado su proceso de búsqueda de regla. La señal regresa en valor '0' cuando el MCA-CAM-n termina completamente el ciclo de transferencia del PLB.
CAM_etiqueta_in	Tiene el mismo valor que la señal instruccion pero esta es la que entra a la memoria CAM para ser buscada dentro de sus reglas.
CAM_reglas_out	En caso de que la regla sea encontrada dentro de la memoria esta señal contiene el campo de decisión de la regla.
CAM_golpe	Cuando al señal tiene valor '0' y state=final, significa que la regla no fue encontrada en la memoria. Si por el contrario el valor de la señal es '1', significa que una de las reglas almacenadas coincidió con la señal

Tabla 4.4: Descripción de las señales utilizadas en este diseño MCA-CAM-n

Unidad de control

Consta de una máquina de estados finitos (MEF) con tres estados denominados **ESPERA**, **BUSQUEDA** y **FINAL**. Esta máquina de estados es sincronizada mediante la señal CAM_reloj también conocida como sPLB_CLK. Cada cambio de estado se lleva a cabo con flanco de subida del reloj. La única señal a la que MEF responde de forma asíncrona es CAM_reinicia, cuando esta señal se encuentra activa la máquina se reinicia y se va a su estado inicial **ESPERA**, las señales de control son inicializadas. Un diagrama de estados es presentado en la figura 4.12.

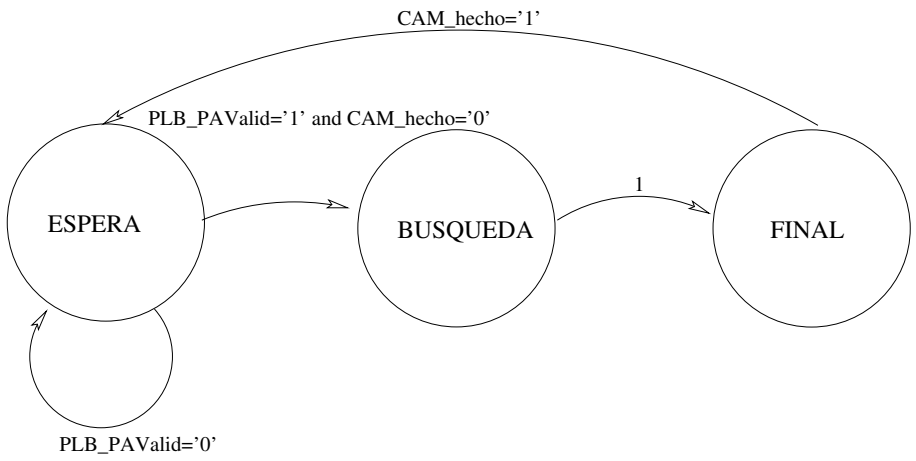


Figura 4.17: Diagrama de estados de unidad de control del MCA-CAM-n

La operación que realiza el MCA-CAM-n es esperar a que la señal `PLB_PAValid` sea '1' para tomar las señales de entrada y formar la instrucción que será comparada con las reglas almacenadas en la memoria, en este caso el proceso de análisis dura sólo un ciclo de reloj lo que significa que la MEF cambiará de estado sin ninguna condición más que transcurrir un ciclo de reloj.

El estado de **ESPERA** se encarga de inicializar todos los registros del MCA-CAM-c, así como formar la instrucción a comparar con las reglas almacenadas, cuando se detecta que la señal de `PLB_PAValid` tiene valor de '1' entonces se pasa la instrucción valida al registro `CAM_regInstruccion`.

BUSQUEDA es el estado donde se realiza la búsqueda de la instrucción formada en el estado anterior con alguna de las reglas almacenadas en el MCA-CAM-n. Este estado solo dura un ciclo de reloj debido a que la búsqueda sólo necesita un ciclo de reloj para determinar si alguna de las reglas almacenadas en la memoria coincide.

FINAL se encarga de ejecutar la decisión de la regla que coincidió o llevar a cabo la operación por defecto.

En caso que el campo de decisión indique que la solicitud de E/S es aceptada, es decir, el bit sea igual a '1', la MEF genera las señales de control para permitir transferir las señales de entrada hacia las de salida del MCA-CAM-n, y así permitir que la operación se ejecute transparentemente tanto para el maestro como para el esclavo.

Cuando el campo de decisión sea igual a '0' la MEF genera señales de control para no permitir la transferencia de las señales de entrada hacia las de salida. Por el contrario genera internamente las señales de el esclavo respondería en el caso que le hubiera llegado una instrucción, y estas señales son enviadas a la unidad de control de PLB, el cual continúa con su proceso creyendo que los datos recibidos son cero.

En el caso que se tome la acción por defecto las señales generadas por la MEF dependerían de cual sea la acción a tomar: aceptar o rechazar la operación. Para todos los casos el estado de **FINAL** se encarga de verificar que el protocolo de transferencia sea finalizado correctamente.

Por otro lado existen señales de control que son tomadas en cuenta como condiciones de cambio de estado. Básicamente las señales consideradas en la MEF son:

- `PLB_PAValid`. La señal es tomada en cuenta en el estado de **ESPERA** e indica que existe una dirección válida para ser analizada por el MCA-CAM-n y verificar si la operación está permitida. Mientras que el valor de esta señal no sea '1' la MFE permanecerá en el estado **ESPERA**. Un cambio de estado ocurrirá siempre y cuando el valor de esta señal sea '1' y la señal `MCA_hecho` sea '0'.

- `CAM_hecho`. La señal es generada durante el estado **FINAL** y su función es mostrar que se ha tomado una decisión sobre la instrucción de entrada, ya sea permitir la operación hacia el esclavo o bien bloquearla.

Memoria

La memoria de acceso aleatorio tiene el atributo de que la memoria es únicamente un dispositivo de almacenaje y cada palabra de la memoria se localiza por una dirección. Una memoria asociativa, además de ser un dispositivo de almacenamiento, tiene el atributo de que las palabras de la memoria son direccionables por el contenido y la memoria tiene la capacidad de búsquedas en paralelo. El direccionamiento por contenido significa que puede accederse a una palabra de memoria por comprobación de un campo seleccionable de una palabra de búsqueda dada, denominada argumento, en vez de por un dirección en una memoria de acceso al azar. La capacidad de búsqueda en paralelo

se traduce en que la palabra de búsqueda puede compararse con todas las palabras de la memoria. Por estas razones, la memoria asociativa se conoce como memoria direccionable por contenido.

El módulo de control de acceso implementado con una memoria direccionable por el contenido se trata de una arquitectura que consta de la unidad de control general que se ha sido descrita y de una memoria tipo CAM.

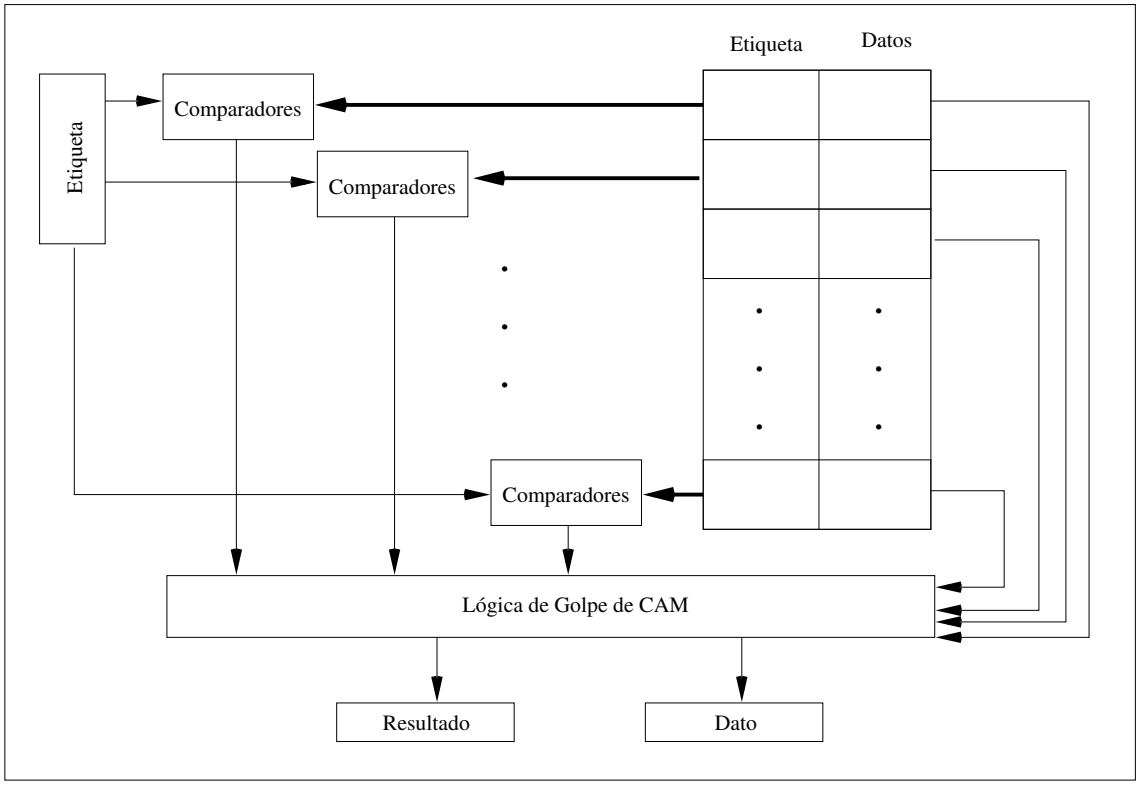


Figura 4.18: Estructura interna de la memoria CAM

La memoria está dividida en dos secciones: etiqueta y regla, la Figura 4.18 muestra la estructura interna del la memoria CAM. También se encuentra sincronizada por la fuente de reloj `CAM_reloj`, las operaciones que se puede realizar en la memoria son lectura y escritura. La escritura se hace de manera secuencial y substituye completamente los valores que se encuentran almacenados tanto en el campo de *etiqueta* como el de la *regla* solamente se puede hacer una escritura cada ciclo de reloj. Las entradas `CAM_reglas_in` y `CAM_etiqueta_in` deben de tener un valor válido para ser almacenado.

La Lectura cumple con la definición de una memoria CAM porque se hace en paralelo, haciendo num_{reglas} comparaciones de la etiqueta de entrada con todas la etiquetas almacenadas en un único ciclo de reloj. El número de niveles de multiplexores que son utilizados por la memoria CAM está dado por la siguiente fórmula: $log_2(num_{reglas})$. Cuando la etiqueta de entrada coincida con alguna de las etiquetas de la memoria se activa la señal `CAM_golpe` y se actualiza el valor de la señal `CAM_reglas_out` con el de la regla correspondiente a la etiqueta que coincidió. Es importante que después de hacer una lectura o escritura se le mande una señal activa de `CAM_reinicia` para que sean inicializados tanto el contador que indica el índice de escritura como la señal `CAM_golpe`. El MCA-CAM-n es el encargado de activar estas señales. Toda la búsqueda se hace en un ciclo, sin embargo, dado que la lógica adicional es un poco más sofisticada, entonces, el ciclo de reloj es más largo que con la versión de la memoria secuencial. La Figura 4.19 ilustra la lógica del bloque del análisis de golpe de la CAM, se trata solo de un ejemplo para una memoria con ocho reglas.

Procesos de coordinación

La tarea de la MCA-CAM-n es muy sencilla ya que prácticamente todo el trabajo lo realiza la memoria. Ésta consta de un sólo proceso que hace la función de coordinador, cuando está en el estado **ESPERA** los valores de la CAM y de control del MCA-CAM-n son inicializadas, si recibe la señal de `PLB_PAValid`=‘1’ forma la etiqueta de entrada de la misma manera en que el diseño formaba la instrucción. El valor de la etiquetada de entrada se pasa a la memoria de manera combinacional. En **CHECAR** combinacionalmente se general la señal `CAM_leer` que indica a la CAM que se va realizar una consulta, el MCA-CAM-n no realiza ninguna otra operación; ese estado es sólo de transición para esperar la respuesta de la memoria. En **FINAL** el MCA-CAM-n verifica el valor de la señal`CAM_golpe` en caso de ser ‘1’ compara `CAM_reglas_out` con ‘1’, si la comparación da positivo entonces permite que la operación solicitada sea ejecutada hacia el esclavo, de lo contrario la operación es bloqueada. Otra alternativa en **FINAL** es que cuando se verifique la señal `CAM_golpe` su valor sea ‘0’ en ese caso el MCA-CAM-n tendrá que tomar la decisión por defecto ya que significa que la etiqueta de entrada no coincidió con ninguna de las almacenadas.

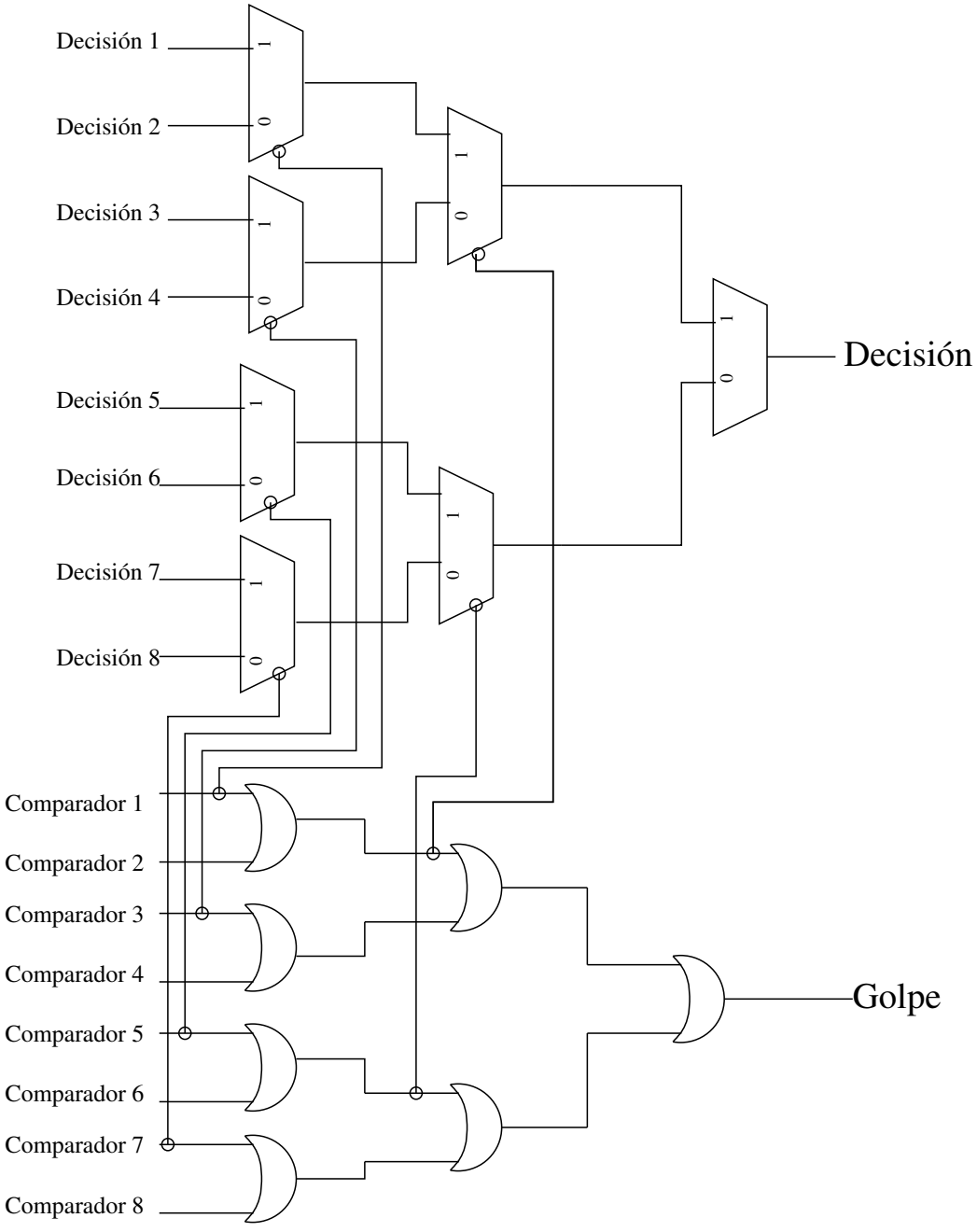


Figura 4.19: Diagrama interno del bloque de lógica de golpe de CAM

4.5 Integración del MCA al bus PLB

El bus PLB fue escogido para integrarle el MCA porque justamente es en él donde hay comunicación directa con todos los esclavos del sistema como se muestra en la Figura 4.20. Es importante considerar que colocar dentro del bus nuestro módulo no afecte la operación típica, sino por el contrario añada la función de poder controlar el acceso a los dispositivos a través de reglas que consideren la virtualización de la plataforma.

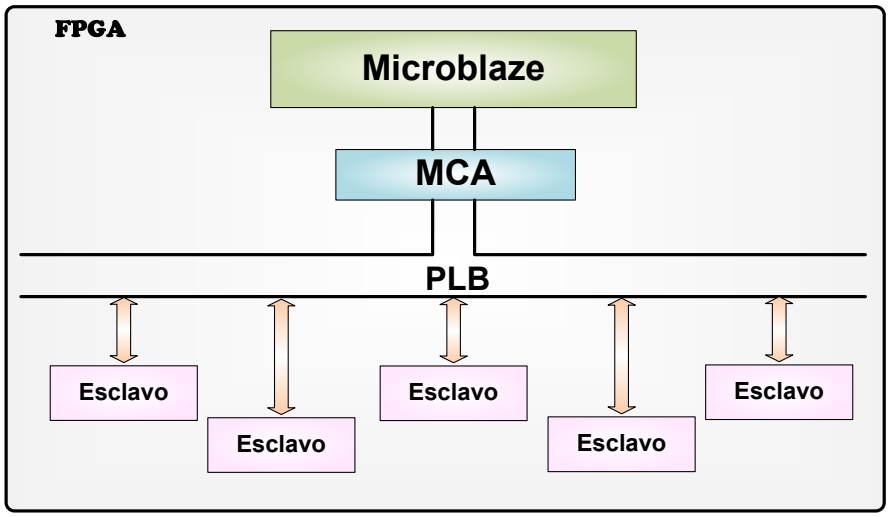


Figura 4.20: Esquema de integración del MCA

Para poder añadir el módulo de control al PBL se estudió tanto el protocolo de transferencia, así como también la forma en la que los esclavos implementan la comunicación con el PLB. Todos los esclavos del sistema utilizan una interfaz de comunicación llamada **PLB_slave_single**, la cual proporciona una comunicación bi-direccional e individual entre el PLB y el núcleo IP del esclavo [41]. Lo más importante es que implementa el protocolo y los tiempos de transferencia entre el PLB y cualquier dispositivo de E/S. La figura 4.21 presenta un diagrama a bloques para ilustrar la funcionalidad de ésta interfaz.

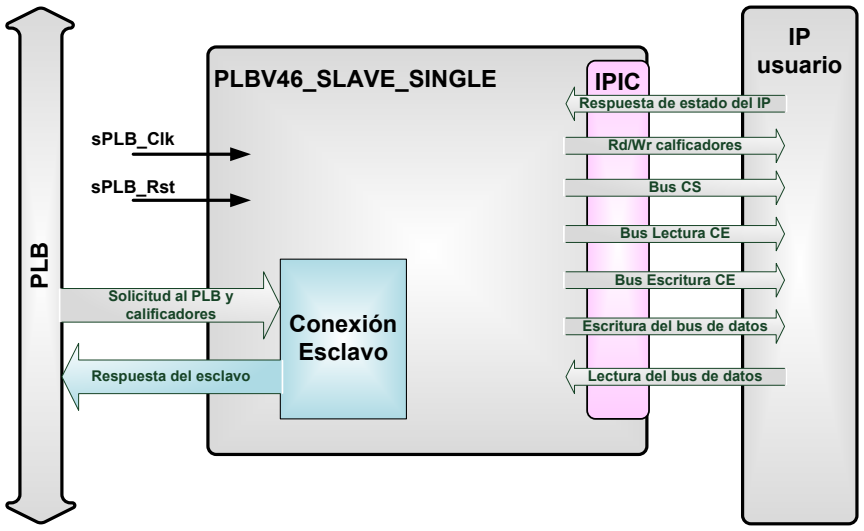


Figura 4.21: Diagrama a bloques de plb_slave_single

Para poder bloquear una operación hacia el dispositivo o permitirla basta con controlar las señales que recibe la interfaz **PLB_slave _single** . Como vimos en la sección 4.2 sobre las señales del PLB sólo debemos prestar atención de los clasificadores de solicitud: `PLB_PAValid`, `PLB_abus`, `PLB_size`, `PLB_type`, `PLB_be`, `PLB_RNW`, `PLB_WrDBus`. Al igual que algunas señales de reconocimiento por parte de la interfaz del esclavo **PLB_slave_single**: `Sl_addrAck`, `Sl_WrDAck`, `Sl_WrComp`, `Sl_RdDAck`, `Sl_RdComp`. Ver Figura 4.22.

El MCA debe de saber manejar dos situaciones con el protocolo de trasferencia del PLB, una es cuando la operación que se está solicitada es admitida y deja pasar todas las señales hacia el dispositivo esclavo y la otra es cuando es denegada. La figura 4.23 muestra un diagrama de tiempos para ilustrar escritura y lectura sin el módulo de control de acceso incrustrado.

Todas las señales del `PLB` son señales enviadas directamente al esclavo. En la Figura 4.23 se encuentra el diagrama de tiempos para formar la instrucción de entrada, en donde las señales involucradas están etiquedas. La (`etiqueta 1`) muestra el momento en que la señal `PLB_PAValid` es activada, a partir de ese ciclo y mientras la señal se mantenga activa las otras señales: `PLB_abus`,

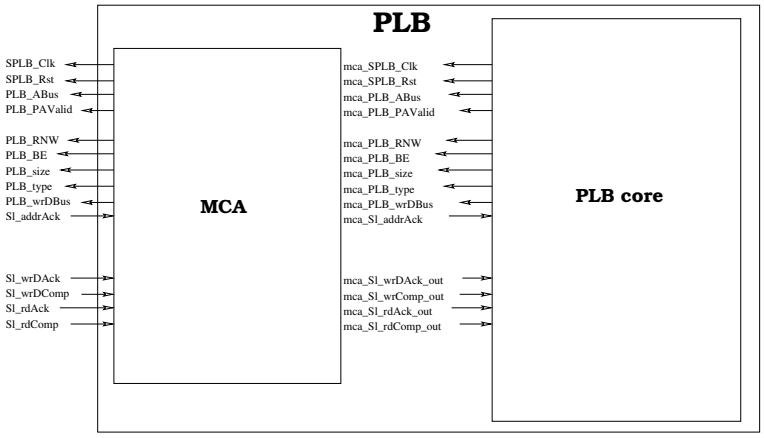


Figura 4.22: Diagrama de la integración del módulo de control al PLB

PLB_size, PLB_type, PLB_be y PLB_RNW son válidas para la interfaz del esclavo. La señal PLB_abus (etiqueta 2) indica la dirección del esclavo al que se está haciendo la solicitud y PLB_RNW ='1' (etiqueta 3) señala que se trata de una operación de lectura.

Una vez que el esclavo ha recibido las señales puede pasar varios ciclos para contestarle al bus el reconocimiento de la dirección y finalizar el ciclo de dirección. Para este caso la señal Sl_addrAck (etiqueta 4) es activada tres ciclos después de que fue indicada una dirección válida. Cuando esta señal es recibida la unidad de control del PLB pone en cero el valor de todas las señales que son enviadas al esclavo. Una de las especificaciones del bus PLB es que una vez iniciado el protocolo de transferencia no toma encuentra ningún cambio de señal hasta que se concluya el ciclo de datos.

El ciclo de datos también puede llevar varios ciclos de reloj y eso depende del tiempo que le lleve al esclavo terminar su proceso. Las señales Sl_RdDAck (etiqueta 6) y Sl_RdComp (etiqueta 7) que indican el reconocimiento de lectura y que la lectura fue completada respectivamente son activadas cuatro ciclos de reloj después y con eso finaliza el ciclo de datos. En este momento el bus está listo para realizar otra operación.

Después de dos ciclos de reloj el bus plb recibe otra operación porque la señal PLB_PAValid (etiqueta 8) está nuevamente activa pero en esta ocasión es el mismo ciclo que fue activada la señal PLB_RNW ='0' lo que significa que se trata de una operación de escritura. Al igual que la operación anterior las señales PLB_abus, PLB_size, PLB_type, PLB_be y PLB_RNW son pasadas

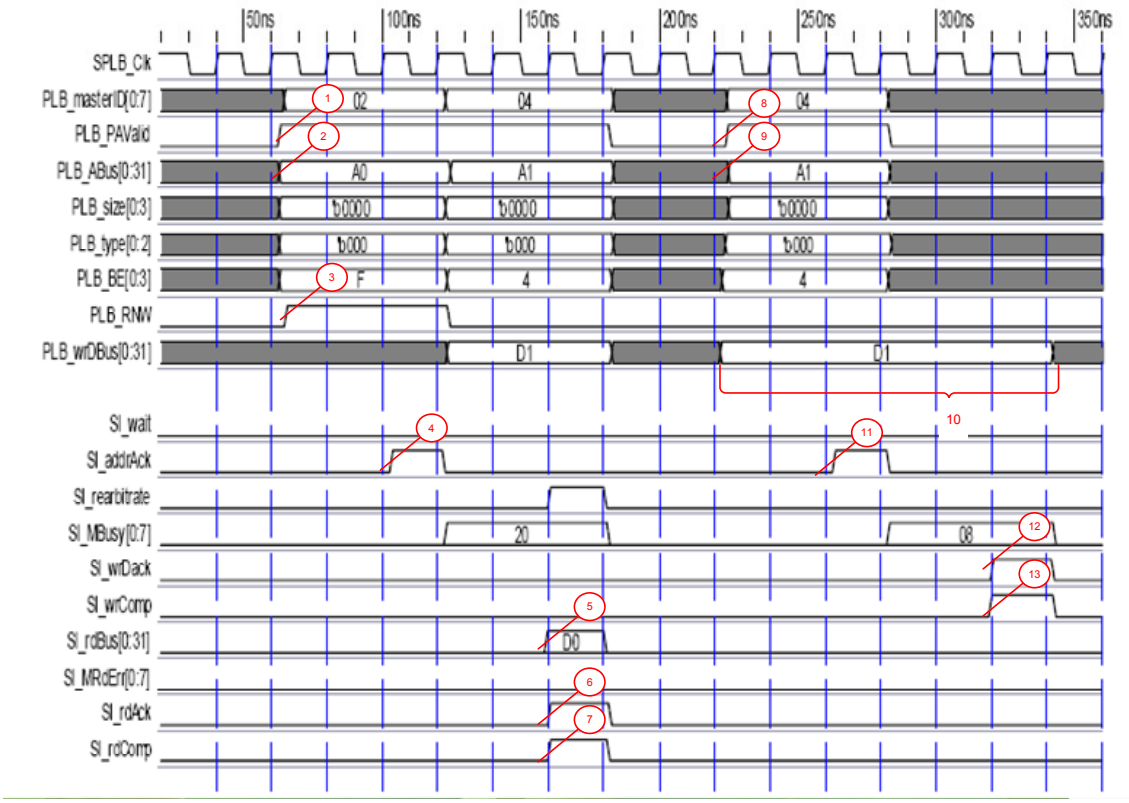


Figura 4.23: Diagrama de tiempos y señales generadas en una lectura y escritura normal.

directamente al esclavo que esté mapeado en la dirección indicada en (`etiqueta 9`). El esclavo responde con el reconocimiento de dirección `Sl_addrAck` (`etiqueta 10`) dos ciclos después, y pasan de nueva cuenta dos ciclos para que el esclavo responda con las señales `Sl_WrDAck`, `Sl_WrCompque` ha finalizado el proceso de escritura y que se encuentra listo para manejar otra operación.

En la Figura 4.24 se encuentra el diagrama de tiempos para formar la instrucción de entrada, en donde las señales involucradas están etiquetadas muestra el diagrama de tiempos con el módulo de control incluido. La franja representa la latencia que el MCA añade al bus PLB, el diagrama es un ejemplo de la ejecución del MCA-CAM-n Las señales de salida (`etiqueta 8`) del MCA se mantienen en cero a no ser que la máquina cambie del estado **FINAL** al estado **ESPERA** (`etiqueta 11`) y la señal `mca_accion='1'` (`etiqueta 7`), en ese caso el MCA hace la conexión entre la señales que normalmente irían al esclavo con las salidas del módulo. El otro caso es que se haya llegado al estado de **FINAL** porque no se encontró una regla que coincidiera. En este caso, el MCA deja las

señales de salida en cero y el dispositivo jamás se entera que había una solicitud para él pero como el protocolo indica que cuando manda una señal activa de `plb_PAValid` (`etiqueta 2`) se espera hasta 15 ciclos de reloj para que el dispositivo conteste un reconocimiento de dirección. Para que no se genere un error el módulo de control cuando detecta una situación así genera internamente la señal de `mca_Sl_addrAck`='1' (`etiqueta 9`) y se tome como entrada al PLB, el protocolo de PLB permite que los reconocimientos del esclavo (`etiqueta 12`) puedan ser generados en el mismo ciclo que se genera la solicitud. En consecuencia el módulo no tiene que esperar ningún ciclo de reloj para generar la señal y de esa manera terminaría correctamente el ciclo de direccionamiento, pero hace falta el ciclo de reconocimiento de datos.

Para concluir el ciclo de reconocimiento de datos, también el MCA es el encargado de generar las señales `mca_Sl_WrDAck_out`, `mca_Sl_WrComp_out` para cuando la operación es escritura y `mca_Sl_RdDAck_out`, `mca_Sl_RdComp_out` (`etiqueta 13`) en caso de lectura. El MCA se debe esperar dos ciclos de reloj para poner estas señales en activo, por que el protocolo se supone que gasta mínimo un ciclo para que el esclavo responda y otra en que la interfaz **PLB_slave_single** active las señales.

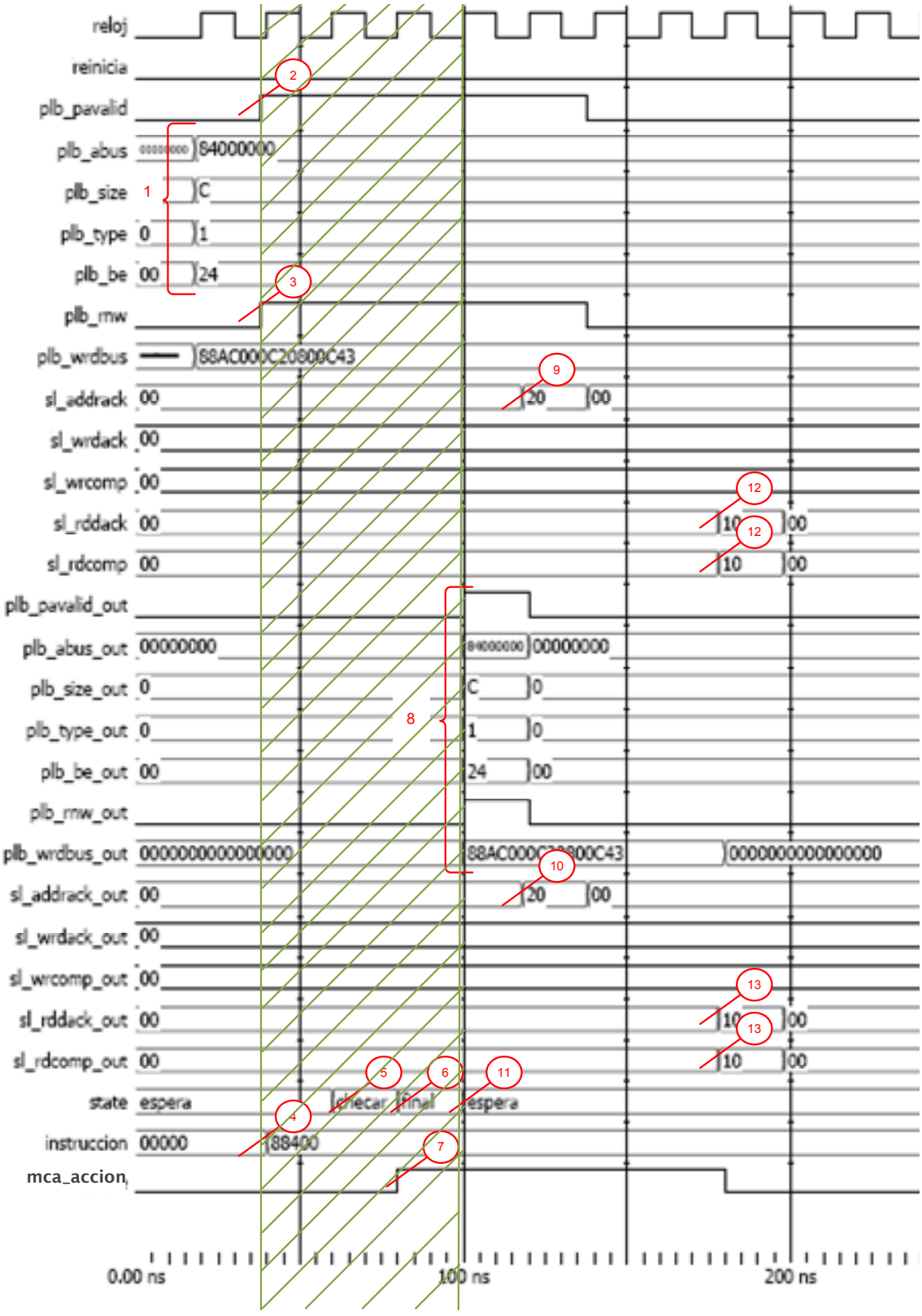


Figura 4.24: Diagrama de tiempos y señales generadas con el módulo integrado.

4.6 Resumen

Para realizar el diseño del MCA fue necesario tomar en cuenta las características generales del procesador Microblaze al igual que la estructura y señalización del PLB. El diseño del MCA fue derivado a partir del análisis que se realizó de los mecanismos de seguridad más comunes en la literatura. Las reglas que son cargadas al MCA fueron diseñadas de tal forma que pudieran controlar el acceso de operaciones tomando en cuenta: la máquina virtual que quiere hacer una operación de E/S, el dispositivo a operar y el tipo de operación sobre él.

Las arquitectura del MCA se basan en una unidad de control que se encarga de suministrar las señales de control a los diversos componentes conectados a este módulo, una memoria para el almacenamiento de las reglas y de registros auxiliares. Se presentaron dos alternativas de implementación del módulo: MCA-RAM-4 y MCA-CAM-n. El uso de la memoria RAM es el más apropiado cuando el área de hardware es limitada pero aún así se requiere controlar el acceso a los dispositivos, con la desventaja que aumenta una latencia de $2+num_{reglas}/4$ ciclos. Por otro lado, tanto la implementación con memoria CAM muestra el lado en donde no importa el área disponible de hardware y sí el tiempo de ejecución, ya que sólo da una latencia de 3 ciclos de reloj para poder dar el resultado.

5

Análisis y evaluación de resultados

En capítulo anterior se mostraron los detalles utilizados para la implementación de MCA como un soporte en versión hardware para controlar los dispositivos periféricos con las máquinas virtuales. Cabe mencionar que aunque la implementación en hardware del MCA presenta ventajas con respecto a la versión software (sobre todo la velocidad de procesamiento) se realizaron dos diferentes diseños para el módulo de control de acceso uno empleando memoria RAM y el otro empleando memoria CAM. El desempeño de ambos diseños fue evaluado mediante dos factores importantes, uno de ellos es la medida de área (los slices requeridos para soportar una de las versiones), el otro es el tiempo (ciclos de reloj) que se requiere para su ejecución. La plataforma de pruebas empleada para la implementación de nuestro diseño se muestra en la siguiente subsección.

5.1 Plataforma de pruebas

La plataforma de pruebas (PP) consiste en una tarjeta de desarrollo starter kit Spartan-3E (ver Figura 5.1), la cual consta de un dispositivo FPGA XC3S500E Spartan-3E. Algunas de las características básicas de este FPGA es que contiene hasta 232 pines de entrada y salida y más

de 10000 celdas lógicas. La tarjeta de desarrollo consta de una pantalla LCD de 2 líneas por 16 caracteres, puerto para el teclado PS/2, puerto VGA, puerto Ethernet, dos puertos serial RS-232 DTE y DCE, así como también dispositivos básicos de entrada y salida como switches, leds y push buttons.

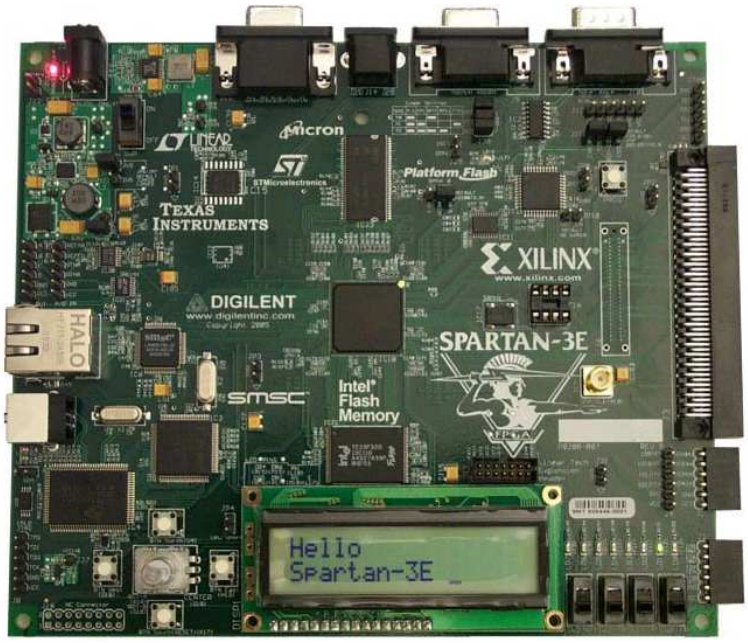


Figura 5.1: Tarjeta de desarrollo Starter Kit Spartan-3E.

La implementación de la plataforma de pruebas fue dividida en dos etapas: etapa de hardware y etapa de software. La etapa de hardware engloba la descripción de la arquitectura de la computadora junto con los controladores de hardware de los periféricos deseados. La selección de los dispositivos que integran a la plataforma de pruebas está en función al tipo de operaciones que se puede realizar con cada dispositivo. La Tabla 5.1 presenta los periféricos que se implementaron y las operaciones que se pueden realizar con cada uno de ellos. El símbolo (E) significa que es una operación de escritura, (L) indica que la operación es de lectura.

La etapa de software comienza cuando el diseño y la implementación en hardware de la PP es finalizada, en esta etapa se dotará a la PP de un sistema operativo Linux para poder realizar pruebas

Dispositivos	Operaciones
RS-232-DTE	Trasmisión(E), recepción(L), control(E) y estado(L)
RS-232-DCE	Trasmisión(E), recepción(L), control(E) y estado(L)
LCD	Desplegar(E), control(E) y estado(L)
VGA (PS/2)	Desplegar(E), control(E) y estado(L)
Teclado	Adquirir datos (L), control(E) y estado(L)
Push bottons	Adquirir datos (L), control(E) y estado(L)
Leds	Desplegar(E), control(E) y estado(L
Ethernet	Trasmisión(E), recepción(L), control(E) y estado(L)
Switches	Adquirir datos (L), control(E) y estado(L)

Tabla 5.1: Operaciones que realiza cada dispositivo

funcionales del módulo de control de acceso. La plataforma de pruebas, tanto para hardware como para software, fueron desarrolladas con la ayuda de las herramientas de Xilinx, ISE y EDK. En las siguiente subsecciones veremos con más detalle cada una de las etapas.

5.1.1 Etapa Hardware

La plataforma de pruebas consiste en diseñar e implementar los módulos de hardware para cada puerto disponible en la tarjeta de desarrollo, así como con el bus PLB y el procesador Microblaze. Aunque cabe mencionar que la integración de la PP no es una tarea fácil ya que es necesario realizar varios pasos para su configuración. La figura 5.2 contiene un diagrama funcional del sistema propuesto, el recuadro que enmarcar a la figura representa el FPGA y cada uno de los recuadros representan módulos de hardware que fueron implementados con lo CLBs del FPGA.

En la actualidad la virtualización de plataforma es una tecnología cuyo uso ha aumentado, sobretodo en empresas de tecnología de la información (TI). Mediante el uso de los VMM's nos permite la creación de múltiples VM lo cual facilita el aprovechamiento de los recursos físicos de la computadora. La principal ventaja es que cada máquina virtual puede ejecutar diferentes sistemas operativos, y a pesar de las ventajas que esto representa es también necesario considerar que con lleva ciertos inconvenientes principalmente el decremento en el rendimiento debido a la emulación.

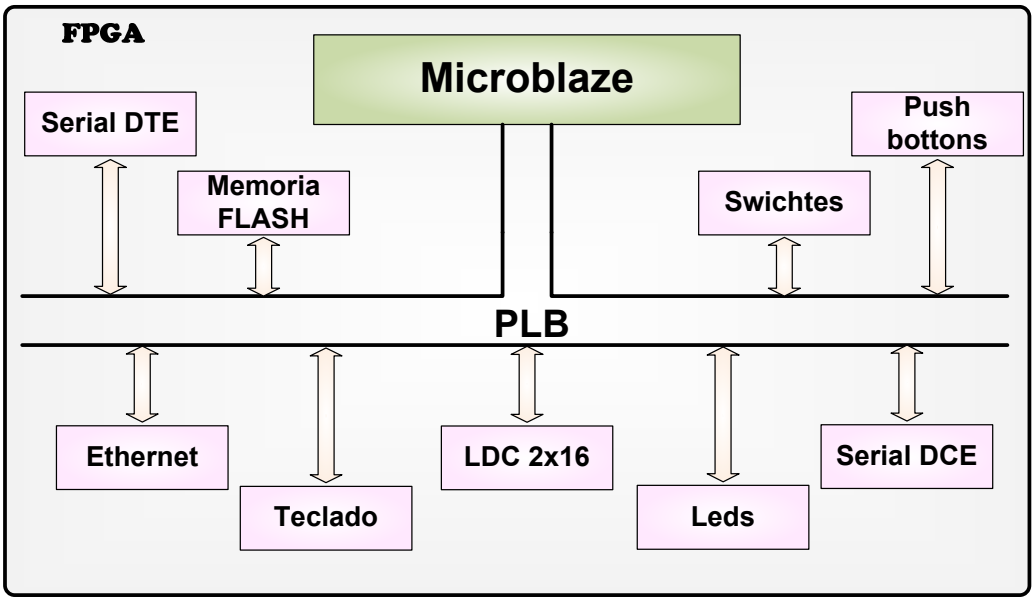


Figura 5.2: Diagrama a bloques de la estructura de interna de la plataforma de pruebas.

Además cabe mencionar que la seguridad se ve afectada de igual manera debido a la comunicación entre las diferentes VM que en algunas ocasiones es requerida.

Si consideramos que una de las grandes preocupaciones en el sistemas de cómputo actuales es la seguridad del sistema, un sistema virtualizado debería garantizar que la protección de la información, las máquinas virtuales contienen sistemas operativos que tiene como definición que son los únicos dueños del hardware, la virtualización debe hacer que las VM son únicas en el sistema administrando todos los recursos de hardware de la plataforma física.

Nuestra principal aportación es el desarrollo en hardware del módulo que permite el control de acceso a dispositivos físicos, lo cual permite tener un control de las peticiones de las VM a los diferentes dispositivos de E/S sin tener que agregar una carga más a las funciones que realiza el VMM. Con este fin se diseñó e implementó en hardware un MCA (Módulo de Control de Acceso), el cual sirve de apoyo al VMM para facilitar el control de acceso las VM, implementado en hardware reconfigurable. Este módulo fue situado entre el procesador y los dispositivos de E/S con el fin de que sirva de árbitro. El MCA se implementó en hardware reconfigurable, empleando una lista de

control de acceso (ACL por sus siglas en inglés *Access Control List*). El MCA interpreta cada una de las política incluidas en la ACL y maneja la decisión de control de acceso a los dispositivos.

En esta tesis fueron diseñadas e implementadas dos versiones de un módulo de control de acceso en hardware, las cuales difieren en el tipo de estructura de memoria que cada una contiene. Para este caso son dos la MAC-RAM-4 y MAC-CAM-n. El primero consta de una memoria RAM y de acuerdo a el análisis realizado este modelo es el más apropiado cuando el área de hardware es limitada pero aún así se requiere controlar el acceso a los dispositivos, la desventaja es que aumenta una latencia de $2 + num_{reglas}/4$ ciclos. La segunda consiste un una memoria del tipo CAM la cual tiene la característica de que sin importar el área disponible de hardware obtiene el mismo tiempo de ejecución ya que sólo da una latencia de 3 ciclos de reloj para poder dar el resultado. Como puede verse en gráfica 5.8 el comportamiento, en cuanto a rendimiento en tiempo, de las dos memorias no es el mismo, en los tres posibles casos (mejor, promedio y peor casos) la memoria CAM se comporta de manera más estable, debido principalmente a que mediante comparadores puede realizar la búsqueda de las instrucciones almacenadas en un solo ciclo de reloj. En cambio, como puede verse en la gráfica 5.9, la memoria RAM utiliza menos recursos en memoria por lo que es más adecuado para plataformas con menor espacio de almacenamiento.

Fue realizado el diseño de la estructura de reglas para el módulo de control de acceso, tomando en cuenta los requerimientos que debía cumplir el MCA. Esta regla se compuso de dos campos, instrucción y decisión. EL campo de instrucción fue formado por el tipo de operación, la dirección del esclavo y el identificador de la máquina virtual. Por otro lado el campo de decisión constaba de los mecanismos de validación para permitir o no el acceso a los dispositivos físicos en la computadora. Una condición para el diseño de la regla es la suposición de la existencia de un identificador de VM, el cual nos permitiría no sólo controlar el tipo de operación que se quiere realizar sino también que VM tiene acceso a ese dispositivo.

Además se realizó el diseño e implementación de una plataforma de pruebas, que permitía hacer la verificación del funcionamiento del MCA. La PP está compuesta de una tarjeta de desarrollo que cuenta con un FPGA Spartan-3E el cual fue programado con un archivo de configuración que describe la arquitectura interna del sistema (procesador, bus y periféricos). Con esta plataforma

permitió la evaluación de pruebas obtuvieron los resultados del módulo de control, comparando las implementaciones de los dos diseños propuestos. Las Pruebas funcionales de comportamiento realizadas mostraron resultados positivos, esto se reflejo a través de los diagramas de estado, los cuales permitieron verificar si el módulo se comportaba tal cual había sido descrito en la etapa de diseño.

Aunque los objetivos establecidos al inicio de este trabajo de tesis fueron alcanzados y se cumplió con la funcionalidad prometida hay algunos aspectos que podrían ser mejorados. Estos puntos se discuten en la siguiente subsección.

Dentro del FPGA fue construida la arquitectura completa de nuestra plataforma de pruebas. Para implementar los controladores de hardware de los dispositivos disponibles, se utilizaron los IPcores disponibles para los periféricos de la tarjeta de desarrollo (Ethernet, RS-232, switches, leds y push buttons) o bien diseñando los controladores de hardware para los periféricos faltantes (teclado PS/2, LCD). El procesador (Microblaze), junto con el bus PLB fueron también tomados de los IPcores. A continuación se describe el procedimiento desde cero para añadir un dispositivo a la PP:

- **Implementación en VHDL de los módulos de cada dispositivo**. Se trata de desarrollar el controlador de hardware para el dispositivo
  - Estudiar la funcionalidad y protocolo del dispositivo.
  - Realizar a partir de la funcionalidad y el protocolo, una descripción en hardware.
  - Sintetizar e implementar.
  - Hacer la simulación lógica.
- **Interfaz de conexión con el procesador.** Es el proceso para crear la interfaz de conexión entre el dispositivo y el procesador consiste en:
  - Escoger un bus para conectar el dispositivo.
  - Encapsular en el dispositivo.
  - Indicar las interrupciones y su tipo.
  - Indicar los registros de usuario.

- Conexión de las señales del dispositivo con las del protocolo del bus escogido.

- **Integración al sistema mínimo.** En este paso ya se cuenta un dispositivo en hardware que tiene completa la interfaz para ser conectado al procesador, el siguiente paso es añadir el nuevo periférico al sistema mínimo. La integración se logra siguiendo los pasos:

  - Se conecta al bus para el que fue diseñado.
  - Se conectan las señales externas del dispositivo.
  - Se hace el mapeo en memoria del dispositivo.
  - Con la ayuda del EDK se crean las bibliotecas para el manejo en software del periférico.

5.1.2 Etapa software

Una vez que se ha concluido la implementación en hardware de la plataforma de pruebas continuamos con la tarea de instalar un sistema operativo Linux a nuestro sistema embebido. Los pasos requeridos para la instalación del sistema operativo se describen a continuación:

- **Escoger una distribución de Linux adecuada al procesador seleccionado.** Entre los principales sistemas operativos que pueden ser implementados para el procesador Microblaze están:

  - Montavista: Distribución no gratuita que actualmente puede usarse en boardsXUPV2P, Virtex4, Virtex5 (Xilinx), basada en GNU/Linux. Soporte empresarial.
  - uCLinux : Distribución gratuita, se ha portado a diferentes plataforma. Para FPGA soporta tanto tarjetas de Xilinx como de Altera. Está basada en GNU/Linux, es un proyecto destinado a portar Linux a dispositivos sin unidades de manejo de memoria (Linux Empotrado). El soporte lo da la misma comunidad a través de la lista de distribución.
  - PetaLinux : Distribución gratuita basada en GNU/Linux. Soportado únicamente para tarjetas de desarrollo de Xilinx, es un trabajo derivado de los desarrolladores de uClinux.

  La distribución de Linux utilizada es PetaLinux, basada en la distribución anterior uClinux, la cual fue elegida porque se trata de una versión destinada para proveer de Linux a dispositivos sin unidades de manejo de memoria y de recursos limitados, como nuestro sistema.

- **Indicar el hardware con el que cuenta la plataforma y sus características.** Una vez que la plataforma de hardware ha sido definida se requiere generar un conjunto de parámetros de software sobre la base de la plataforma de hardware. Estos parámetros son utilizados para iniciar y comunicar el hardware.
- **Compilar el núcleo de Linux.** La compilación de Petalinux consiste en:
  - Compilar el núcleo de Linux
  - Compilar las herramientas de GNU
  - Construir el sistemas de archivos raíz (rootfs)
  - Compilar el sistema de arranque U-Boot
- **Cargar el sistema operativo al FPGA.** Por medio de las herramientas que proporciona Petalinux, se hace la descarga de la imagen del núcleo de Linux hacia el FPGA, en realidad, esta imagen es almacenada dentro de la memoria FLASH de la tarjeta de desarrollo.

Para más información acerca de ambas etapas véase el apéndice A, donde está documentado el proceso que debe seguirse para dotar de Linux a un sistema embebido.

5.2 Verificación del MCA

Para el diseño de la plataforma de pruebas fueron creados y/o implementados controladores de hardware para cada uno de los periféricos que la integran. Los controladores de hardware fueron desarrollados con lenguaje de descripción de hardware (VHDL). Estos controladores describen la lógica necesaria para poder conectar el dispositivo a alguno de los buses del sistema. El objetivo es otorgar a estos dispositivos la capacidad de manejar las señales correspondientes al protocolo del bus, tanto para las señales recibidas como para las señales de datos que el periférico responde al bus. Para la verificación del funcionamiento del sistema fue desarrollado un conjunto de pruebas, las cuales se dividen en: pruebas de comportamiento y funcionales, éstas se describen en la siguiente subsección.

5.2.1 Pruebas de comportamiento

Las pruebas de comportamiento consisten en simulaciones a nivel lógico del módulo de control conectado con el PLB, su principal objetivo es la verificación del comportamiento descrito en el diseño. El objetivo de estas pruebas fue darle al MCA las señales que el PLB generaría en el caso de querer realizar alguna operación de escritura o lectura hacia un periférico, así como las señales que los esclavos generarían para reconocer las operaciones y verificar que el MCA tenga un comportamiento correcto. Las señales que se modificaban manualmente por parte del bus PBL eran las de `PLB_abus`, `PLB_RNW`, `PLB_PAValid`, `PLB_size`, `PLB_type`, `PLB_be`. Para las señales de los esclavo se manipulaban `sl_addrAck`, `sl_WrDAck`, `sl_WrComp`, `sl_RrDAck`, `sl_RrComp`. El MCA previamente era cargado con las reglas deseadas, éstas operaciones eran modificadas dependiendo del caso que se quiera probar.

En la tabla 5.2 se muestran los casos que fueron probados, variando el valor de las señales de entrada al MCA.

Caso	Operación	Instrucción en las reglas	Acceso permitido
1	Escritura	Si	Si
2	Escritura	Si	No
3	Escritura	No	No importa
4	Lectura	Si	No
5	Lectura	Si	Si
6	Lectura	No	No importa

Tabla 5.2: Casos de pruebas para la simulación de comportamiento

A pesar de que fueron realizados los seis casos de prueba para ambos diseños, en esta sección de la tesis sólo se presentarán los casos uno y cinco. El resto de las pruebas serán presentadas el apéndice B.

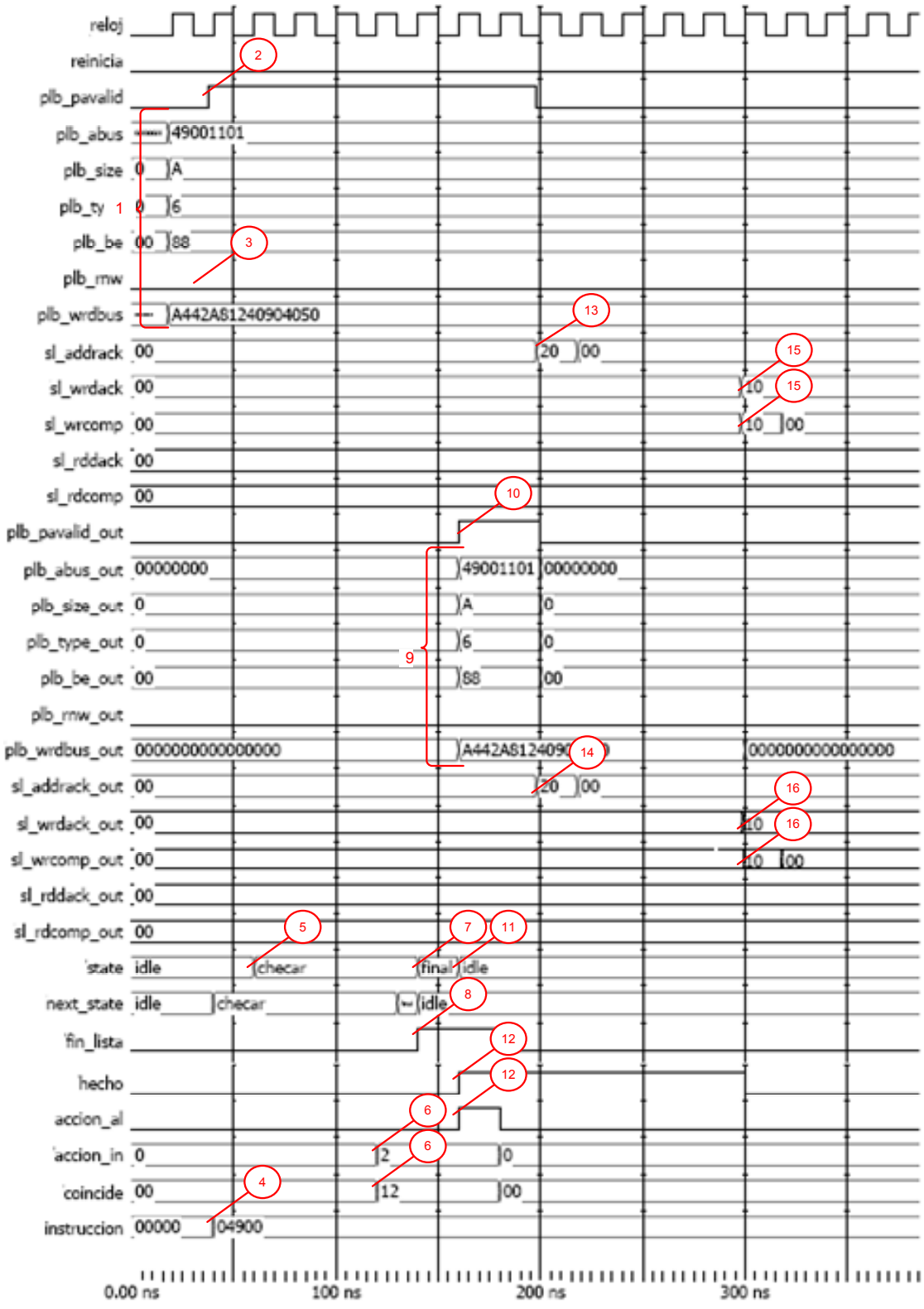


Figura 5.3: Diagrama de señales para permitir la escritura a un dispositivo con el MCA-RAM-4.

Las Figuras 5.3 y 5.4 muestran los diagramas de señales generadas por las simulaciones de la operación de lectura (MCA-CAM-n) y escritura (MCA-RAM-16) respectivamente. En ambas simulaciones la instrucción formada a partir de las señales de entrada coincide en alguna de la reglas almacenada en la memoria del MCA, además que la acción de las reglas es permitir el acceso al dispositivo.

En la Figura 5.3 se observa que las señales de salida válidas se obtienen después de seis ciclos de reloj, lo que indica que la regla fue encontrada dentro de las últimas cuatro reglas analizadas. La descripción de las señales utilizadas para esta prueba estan en las Tablas 5.3 y 5.4, la interpretación del diagrama que se muestra en la Figura 5.3 es descripta a continuación:

1. Las señales que se encuentran dentro de la `etiqueta 1` son las enviadas por el maestro que realiza la solicitud hacia el PLB, éste a su vez lo retransmite hacia el esclavo solicitado.
2. En este momento `PLB_PAValid` (`etiqueta 2`) la señal se activa significa que las señales recibidas en el punto anterior son tomadas como válidas por el bus y pueden ser leidas por el esclavo.
3. La señal `PLB_RnW` (`etiqueta 3`) se encuentra en cero en el momento en que `PLB_PAValid` fue activado (`etiqueta 2`), lo que indica que es una señal de escritura.
4. Con `PLB_PAValid` activado el módulo de control de acceso inicia su tarea creando una instrucción (`etiqueta 4`) que servirá de entrada para ser comparada con las reglas almacenadas.
5. La señal de `state` (`etiqueta 5`) muestra el cambio de estado después de un ciclo de que el MCA-RAM inició su operación.
6. Después de cuatro ciclos de reloj las señales `coincide` y `accion_in` (`etiqueta 6`) indican que existió una coincidencia entre la instrucción de entrada y las reglas almacenadas.
7. Un ciclo después de haber encontrado una regla nuevamente la máquina de estados finitos cambia al estado final (`etiqueta 7`).
8. A pesar de que `fin_lista` (`etiqueta 8`) se pone en activo la señal no es tomada en cuenta porque hubo una coincidencia, la razón por de su activación es debido a que esta coincidencia

fue encontrada durante las últimas reglas analizadas.

9. EL módulo de control de acceso da como salida los valores de entrada de las señales agrupadas en la `etiqueta 9`, lo cual significa que la regla encontrada especifica que la operación solicitada es permitida.

10. La señal `accion_al` (`etiqueta 12`) ratifica el comportamiento de la `etiqueta 9` dado que se encuentra activada justo en el ciclo en el que el módulo de control de acceso emitió la operación.

11. Como consta en la especificación del MCA después del estado final debe aguardar un ciclo de reloj para realizar el cambio al estado de espera (`etiqueta 11`).

12. Cuando la señal `hecho` (`etiqueta 12`) es igual a '1' lógico indica que el MCA ha terminado el proceso de búsqueda de la regla y que ha tomado una decisión, en este caso `accion_al` también se activa, es decir, la operación es permitida.

13. Después que el MCA ha enviado las señales hacia el esclavo, sólo le falta esperar que el esclavo mande un reconocimiento de dirección, lo cual se realiza activando la señal `sl_addrack` como es indicado en la `etiqueta 13`.

14. El módulo recibe la señal `sl_addrack` (`etiqueta 13`) y la conecta directo a su salida `sl_addrack_out` (`etiqueta 14`), como se muestra en el diagrama. Esta señal concluiría el ciclo de dirección y será recibida por el PLB como si en realidad fuera el esclavo quien se la enviará y no el MCA.

15. Para finalizar el ciclo de datos el MCA espera que el esclavo envíe la señal de reconocimiento de lectura y de completado de lectura, como se muestra en el diagrama.

16. Una vez que el MCA detecta un cambio en las señales `sl_rddack` y `sl_rdcomp` (`etiqueta 15`), como está haciendo el manejo de una operación permitida entonces envía los datos recibidos sus salidas `sl_rddack_out` y `sl_rdcomp_out`. La señal `hecho` (`etiqueta 12`) cambia su valor a '0', lo que indica que todo el protocolo del PLB ha sido concluido y que ahora se puede atender otra solicitud.

Señal	Descripción
reloj	Se trata de la extensión de la señal sPLB_Clk, sirve para la sincronización del MCA.
reinicia	Es una extensión de la señal sPLB_Rst y provoca que la señal esté activada todos los registros del MCA sean reinicializados
plb_pavalid	Señal de entrada al MCA, además de la funcionalidad de la señal PLB_PAValid, cuando se encuentra activa sirve como señal de inicio para el MCA .
plb_abus	Señal tomada de PLB_abus, es entrada al módulo y sirve para construir la instrucción que el módulo de control comparará con las reglas que tenga almacenadas. Cuando la decisión del MCA sea permitir la operación , esta señal es conectada a la señal de salida plb_abus_out .
plb_size	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_size_out
plb_type	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_type_out
plb_plb_be	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_be_out
plb_rnw	Señal de entrada que indica que tipo de operación se realizará (lectura o escritura). Además es utilizada para formar la instrucción del MCA.
plb_wrdbus	No tiene función para el proceso del MCA, pero en caso de que ladecisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_wrdbus_out
sl_addrack	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de solicitud y acoplar esta señal con sl_addrack_out
sl_wrdack	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_wrdack_out
sl_wrcomp	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_wrcomp_out
sl_rddack	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_rddack_out
sl_rdcomp	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_rdcomp_out

Tabla 5.3: Tabla de la señales de utilizadas en las pruebas escritura del MCA-RAM-4

plb_pavalid_out	Señal de salida que en caso que la operación solicitada sea permitida tendrá el mismo valor que la señal plb_pavalid, en caso contrario su valor será ceros.
plb_abus_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_abus, en caso contrario su valor será ceros.
plb_rnw_out	Señal de salida que en caso que la operación solicitada sea permitida tendrá el mismo valor que la señal plb_rnw, en caso contrario su valor será ceros.
plb_wrdbus_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_wrdbus, en caso contrario su valor será ceros.
sl_addrack_out	Toma el valor de sl_addrack cuando la operación solicitada fue permitida. Si la operación es rechazada el MCA genera un valor de X"20" para hacer creer a la unidad de control del bus PLB que el esclavo realizó el reconocimiento de dirección
sl_wrdack_out	Toma el valor de sl_wrdack cuando la operación solicitada fue permitida. Si la operación es rechazada el MCA genera un valor de X"10" para hacer creer a la unidad de control del bus PLB que el esclavo realizó el reconocimiento de dirección
sl_wrcomp_out	Toma el valor de sl_wrcomp cuando la operación solicitada fue permitida. Si la operación es rechazada el MCA genera un valor de X"10" para hacer creer a la unidad de control del bus PLB que el esclavo realizó el reconocimiento de dirección
state	Señal auxiliar para conocer en que estado se encuentra la MEF.
next_state	Señal auxiliar para conocer cuál será el estado de la MEF en el siguiente ciclo de reloj.
hecho	Cuando la señal se activa significa que el MCA ha terminado su proceso de búsqueda de regla. La señal regresa en valor '0' cuando el MCA termina completamente el ciclo de transferencia del PLB.
instruccion	Es conformada por las señales plb_rnw y plb_abus, se trata de la instrucción de entrada del MCA
fin_lista	Señal que se activa cuando se llego al final de la lista de reglas.
accion_in	Es la señal de cuatro bits que contiene el campo de decisión para cada regla examinada por uno de los cuatro procesos. Cuando se encuentra una coincidencia la señal es analizada para determinar cuál es la acción que corresponde a la regla que coincidió.
accion_al	Esta señal es la acción que va ser ejecutada por el MCA, después de que se hizo el análisis de accion_in
coincide	Es un señal de cinco bits, en donde los cuatro bits menos significativos tienen correpondendia directa con el número de proceso que realiza el análisis, en cada bit el proceso almacena si en ese ciclo encontró alguna coincidencia o no.

Tabla 5.4: Continuación de la señales de utilizadas en las pruebas escritura del MCA-RAM-4

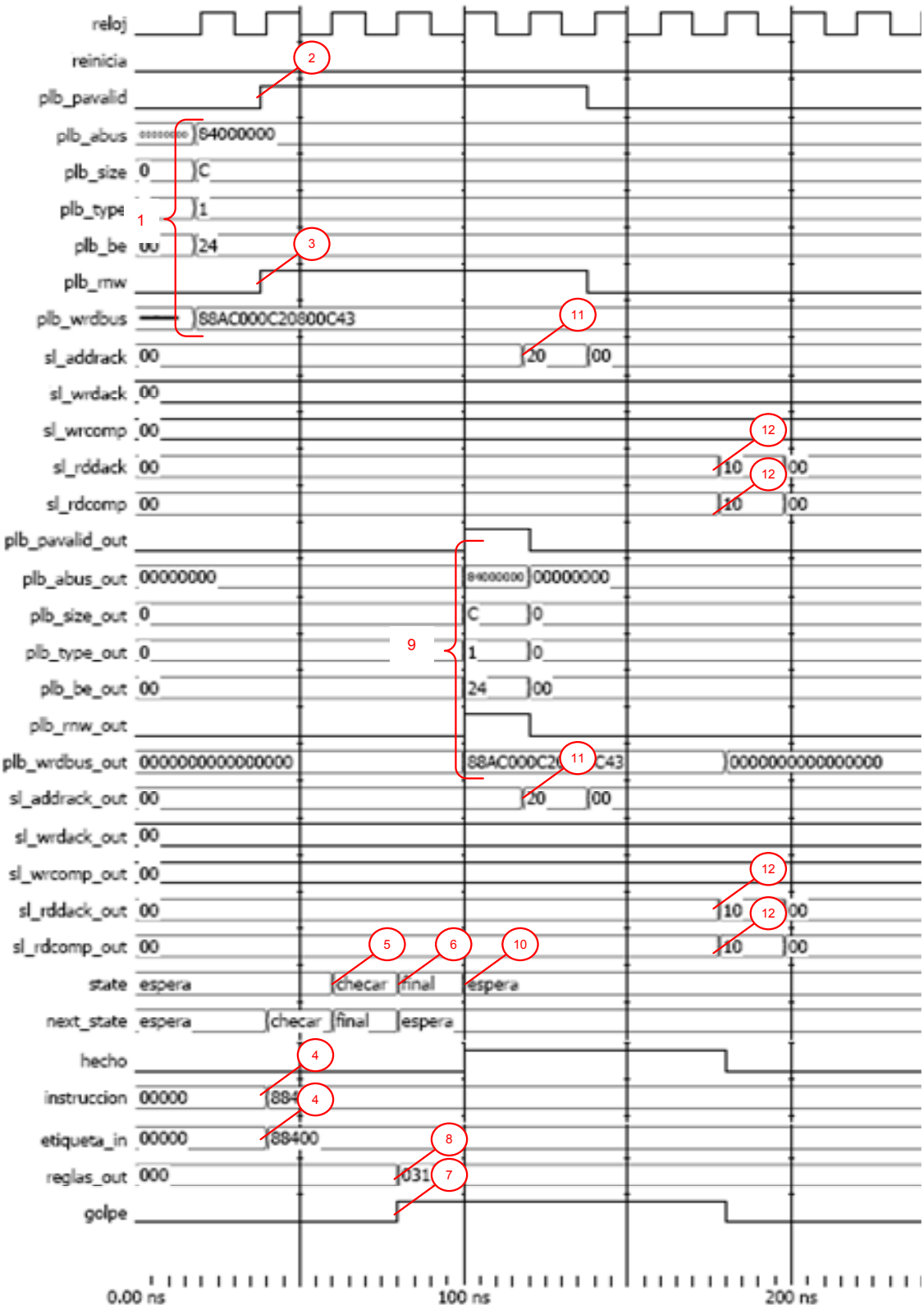


Figura 5.4: Diagrama de señales para permitir la lectura a un dispositivo con el MCA-CAM-n.

En las Tablas 5.5 y 5.6 presenta una breve descripción de las señales que son analizadas para el diseño del MCA-CAM.

A continuación se da la interpretación del diagrama que se muestra en la Figura 5.4, para esta explicación se colocaron marcas en la figura las cuales se relacionan con los números aquí mostrados:

1. La señales que se encuentran agrupadas en la dentro de la `etiqueta 1` son las enviadas por el maestro que realiza la solicitud hacia el PLB, son las señales que serán analizadas por el MCA-CAM-n.
2. Cuando la señal `PLB_PAValid` (`etiqueta 2`) es activada el MCA-CAM-n inicia su proceso y a partir de ese momento toma en cuenta las señales en la `etiqueta 1`) para iniciar el análisis.
3. La señal `PLB_RnW` (`etiqueta 2`) tiene valor de '1', esto significa que la operación solicitada se trata de una lectura a un esclavo.
4. Con `PLB_PVAalid` activado el módulo de control de acceso inicia su tarea creando una instrucción que servirá de entrada para ser comparada con las reglas almacenadas. La señal de `instruccin` (`etiqueta 4`) tiene un valor de X"88400" que fue construido a partir de las señales en la `etiqueta 1`.
5. La señal `state` muestra el cambio de estado de **ESPERA** a **CHECAR** (`etiqueta 5`) después de un ciclo en el que el MCA-CAM-n recibió `plb_pavalid`='1' e inicio su operación. Durante el estado **CHECAR** la `etiqueta_in` se envía a la memoria, ésta la analiza y entrega un resultado en el mismo ciclo de reloj.
6. La MEF cambia de estado al **FINAL** (`etiqueta 6`) porque en el ciclo anterior obtuvo la información de la CAM y ahora conoce dependiendo de los datos devueltos por la CAM ejecutará la decisión de la regla.
7. `golpe` (`etiqueta 7`) es la señal que devuelve la CAM para indicar que hubo una coincidencia entre la `etiqueta_in` y la regla almacenada.
8. `reglas_out` (`etiqueta 8`) es otro de los datos devueltos por la CAM, cuando la señal de `golpe` esta activa, el valor de la señal es el dato que esta almacenado en la memoria, cuando `golpe`='0'

, la señal devuelve puros ceros. En este caso el bit menos significativo de `reglas_outes` '1', esto significa que la operación es permitida.

9. Las señales englobas en la `etiqueta 9` son las salidas que genera el MCA-CAM-n una vez que se aseguro de que la operación era permitida. Las señales en (1) son pasadas directamente hacia el esclavo, esto en forma transparente.

10. Después que el MCA-CAM-n ha enviado las señales hacia el esclavo, sólo le falta esperar que el esclavo mande un reconocimiento de dirección, esto lo realiza activando la señal `sl_addrack` como es indicado aquí.

11. El módulo recibe la señal `sl_addrack` (`etiqueta 11`) y la conecta directo a su salida `sl_addrack_out` (`etiqueta 11`), como se muestra en el diagrama. Esta señal concluiría el ciclo de dirección y será recibida por el PLB como si en realidad fuera el esclavo quien se la enviará y no el MCA-CAM-n.

12. Para finalizar el ciclo de datos el MCA-CAM-n espera que el esclavo envíe la señal de reconocimiento de lectura y de completado de lectura, como se muestra en el diagrama.

13. Una vez que el MCA-CAM-n detecta un cambio en las señales `sl_rddack` y `sl_rdcomp` (`etiqueta 12`), como está haciendo el manejo de una operación permitida entonces envía los datos recibidos sus salidas `sl_rddack_out` y `sl_rdcomp_out`. La señal `hecho` cambia su valor a '0', lo que indica que todo el protocolo del PLB ha sido concluido y que ahora se puede atender otra solicitud.

Señal	**Descripción** a
reloj	Se trata de la extensión de la señal sPLB_Clk, sirve para la sincronización del MCA.
reinicia	Es una extensión de la señal sPLB_Rst y provoca que la señal esté activada todos los registros del MCA sean reinicializados
plb_pavalid	Señal de entrada al MCA, además de la funcionalidad de la señal PLB_PAValid, cuando se encuentra activa sirve como señal de inicio para el MCA .
plb_abus	Señal tomada de PLB_abus, es entrada al módulo y sirve para construir la instrucción que el módulo de control comparará con las reglas que tenga almacenadas. Cuando la decisión del MCA sea permitir la operación, esta señal es conectada a la señal de salida plb_abus_out.
plb_size	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_size_out
plb_type	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_type_out
plb_plb_be	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_be_out
plb_rnw	Señal de entrada que indica que tipo de operación se realizará (lectura o escritura). Además es utilizada para formar la instrucción del MCA.
plb_wrdbus	No tiene función para el proceso del MCA, pero en caso de que la decisión sea permitir la operación analizada, esta señal será conectada a la señal de salida plb_wrdbus_out
sl_addrack	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de solicitud y acoplar esta señal con sl_addrack_out
sl_wrdack	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_wrdack_out
sl_wrcomp	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_wrcomp_out
sl_rddack	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_rddack_out
sl_rdcomp	Señal de entrada, proveniente del esclavo. Si la operación fue permitida, el MCA la toma en cuenta para finalizar la fase de reconocimiento y acoplar esta señal con sl_rdcomp_out

Tabla 5.5: Tabla de la señales de utilizadas en las pruebas del MCA-CAM-n.

plb_pavalid_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_pavalid, en caso contrario su valor será ceros.
plb_abus_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_abus, en caso contrario su valor será ceros.
plb_size_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_size, en caso contrario su valor será ceros.
plb_rnw_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_rnw, en caso contrario su valor será ceros.
plb_wrdbus_out	Señal de salida que en caso que la operación solicitada sea permitida tedrá el mismo valor que la señal plb_wrdbus, en caso contrario su valor será ceros.
sl_addrack_out	Toma el valor de sl_addrack cuando la operación solicitada fue permitida. Si la operación es rechazada el MCA genera un valor de X"20" para hacer creer a la unidad de control del bus PLB que el esclavo realizó el reconocimiento de dirección
sl_rddack_out	Toma el valor de sl_rddack cuando la operación solicitada fue permitida. Si la operación es rechazada el MCA genera un valor de X"10" para hacer creer a la unidad de control del bus PLB que el esclavo realizó el reconocimiento de dirección
sl_rdcomp_out	Toma el valor de sl_rdcomp cuando la operación solicitada fue permitida. Si la operación es rechazada el MCA genera un valor de X"10" para hacer creer a la unidad de control del bus PLB que el esclavo realizó el reconocimiento de dirección
state	Señal auxiliar para conocer en qué estado se encuentra la MEF.
next_state	Señal auxiliar para conocer cuál será el estado de la MEF en el siguiente ciclo de reloj.
hecho	Cuando la señal se activa significa que el MCA ha terminado su proceso de búsqueda de regla. La señal regresa en valor '0' cuando el MCA termina completamente el ciclo de transferencia del PLB.
instruccion	Es conformada por las señales plb_rnw y plb_abus, se trata de la instrucción de entrada del MCA
etiqueta_in	Tiene el mismo valor que la señal instruccion pero esta es la que entra a la memoria CAM para ser buscada dentro de sus reglas.
reglas_out	En caso de que la regla sea encontrada dentro de la memoria esta señal contiene el campo de decisión de la regla.
golpe	Cuando al señal tiene valor '0' y state=final, significa que la regla no fue encontrada en la memoria. Si por el contrario el valor de la señal es '1', significa que una de las reglas almacenadas coincidió con la señal etiqueta_in

Tabla 5.6: Continuación de las señales de utilizadas en las pruebas del MCA.

5.2.2 Pruebas funcionales

Las pruebas funcionales consisten de programas desarrollados en lenguaje C que sirven con interfaz de software entre el procesador y el usuario final. Debido a que la comunicación entre los dispositivos y el procesador se realiza en forma de mapeo de memoria, entonces los programas desarrollados se encargan de acceder al rango de memoria en donde se encuentra mapeado el dispositivo. Estos programas utilizan las bibliotecas que son proporcionadas por la herramienta EDK de Xilinx cuando se construye todo el sistema, estas bibliotecas son en lenguaje C y contienen funciones encapsuladas que facilitan el manejo de E/S y abstraen el uso de las instrucciones del procesador.

Sabemos que el módulo de control de acceso es capaz de interceptar cualquier instrucción de escritura o lectura hacia alguno de los periféricos mapeados en memoria. Para realizar la verificación del sistema, se desarrolló una prueba en donde se utilizó alguno de los periféricos de la plataforma de pruebas. El periférico seleccionado es el puerto serial (RS-232) y se utilizó el módulo UartLite como controlador de hardware. La razón por la que se escogió el puerto serial es debido a que este dispositivo cuenta con las dos instrucciones que el MCA toma en cuenta: Escritura (Tx) y Lectura (Rx), en la Figura 5.5 se muestra un diagrama a bloques del módulo UartLite, en donde se muestran los registros internos que son accesible por software.

Los pasos para realizar la prueba funcionar se pueden reducir a continuación:

1. Configuración de las reglas dentro del MCA.
2. Integrar el programa en lenguaje C, para la prueba.
3. Programar el FPGA.
4. Configuración de la computadora auxiliar.
5. Realizar la comunicación entre la computadora auxiliar y la plataforma de pruebas.
6. Llevar acabo de prueba de puerto serial.

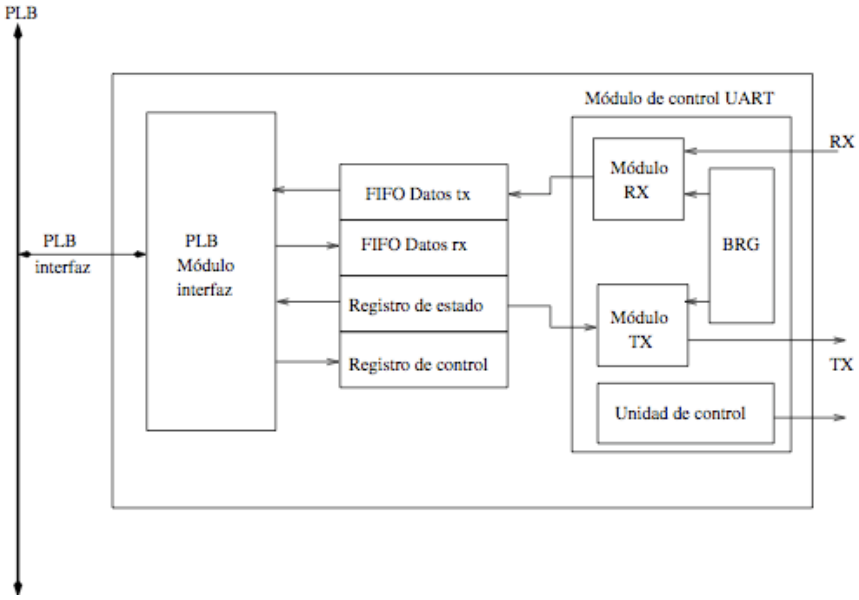


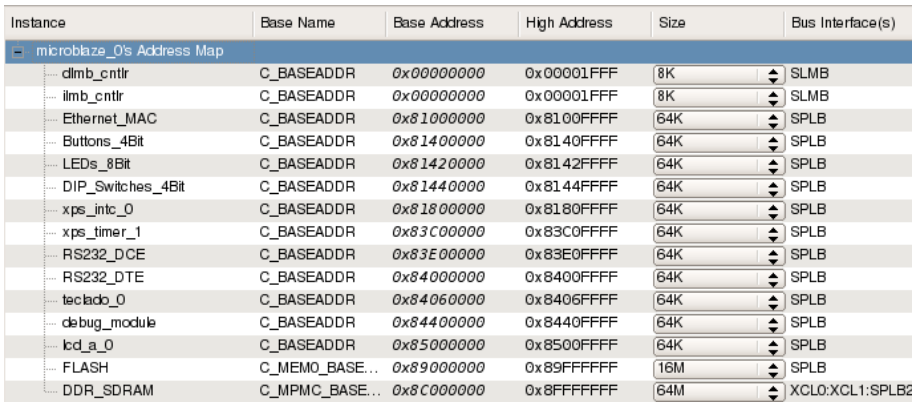
Figura 5.5: Diagrama a bloques de UartLite

A continuación se hace una descripción más a detalle de los pasos para realizar la prueba funcional del puerto serial.

1. Configuración de las reglas dentro del MCA.

Debido a que por el momento la configuración dinámica de las reglas para el MCA no fue implementada, es necesario hacer la configuración de la memoria antes de sintetizar el hardware de la PP, lo cual se logra haciendo la configuración correcta de las reglas. En este punto es necesario que el usuario defina las reglas que serán cargadas al sistema, para ello es necesario que tenga conocimiento acerca del mapa de direcciones del sistema. En la Figura 5.6 se muestra el mapa de direcciones de la plataforma de pruebas, particularmente nos interesa el rango de dirección asignado para la instancia del puerto serial DTE que es la que se utiliza para esta prueba.

Se puede observar en la Figura 5.6 que la dirección base para la instancia del DTE es 0x84000000, con este valor se puede formar una regla para ese dispositivo, en donde se permitan las dos operaciones



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)
microblaze_0's Address Map					
dlmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB
Ethernet_MAC	C_BASEADDR	0x81000000	0x8100FFFF	64K	SPLB
Buttons_4Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB
LEDs_8Bit	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB
DIP_Switches_4Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB
xps_intc_0	C_BASEADDR	0x81800000	0x8180FFFF	64K	SPLB
xps_timer_1	C_BASEADDR	0x83C00000	0x83C0FFFF	64K	SPLB
RS232_DCE	C_BASEADDR	0x83E00000	0x83E0FFFF	64K	SPLB
RS232_DTE	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB
teclado_0	C_BASEADDR	0x84060000	0x8406FFFF	64K	SPLB
debug_module	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB
lcd_a_0	C_BASEADDR	0x85000000	0x8500FFFF	64K	SPLB
FLASH	C_MEM0_BASE...	0x89000000	0x89FFFFFF	16M	SPLB
DDR_SDRAM	C_MPMC_BASE...	0x8C000000	0x8FFFFFFF	64M	XCL0:XCL1:SPLB2

Figura 5.6: Mapa de direcciones del sistema de pruebas

básicas, lectura y escritura. En la Tabla 5.7 se muestran las reglas que fueron incluidas a memoria CAM. Recordemos que para el caso del MCA con CAM no influye la posición de las regla en la memoria, en este caso la reglas que corresponde a la operación que se quiere admitir, se encuentran en la posición 14 y 15.

Indice	Etiqueta	Regla
0	X"88100"	X"001"
1	X"88140"	X"001"
2	X"08142"	X"001"
3	X"88180"	X"001"
4	X"00000"	X"001"
5	X"80000"	X"001"
6	X"08180"	X"001"
7	X"883C0"	X"001"
8	X"083C0"	X"001"
9	X"883E0"	X"001"
10	X"08402"	X"001"
11	X"88406"	X"001"
12	X'88500"	X"001"
13	X"00000"	X"001"
14	X"08400"	X"001"
15	X"88400"	X"001"

Tabla 5.7: Contenido de la memoria CAM, reglas pre-cargadas del sistema

Una vez terminada la configuración de las reglas es necesario sintetizar el hardware que describe

la PP para que se concluya la configuración de las reglas del MCA. El siguiente paso es la integración del programa

2. Integrar el programa en lenguaje C, para la prueba.

Una vez que se han creado los componentes de hardware y se generan las bibliotecas para el sistema sólo resta integrar el código del programa para la prueba funcional.

Haciendo uso de las bibliotecas creadas por la herramienta EDK se implementó un programa en lenguaje C que llevará a cabo el protocolo del puerto serial, realizando la lectura y escritura a los registros de Tx, Rx, control y estado del controlador de hardware del UartLite. El programa se encarga de manejar el escrutinio del registro de estado para saber si existe un dato nuevo en el registro Rx, y maneja las colas correspondientes para la transmisión de los datos. Todo caracter que es recibido en la plataforma es trasmitido de la misma forma. Una vez concluida la integración del programa en el sistema es necesario unir la descripción de hardware con la descripción de software para la plataforma de prueba, lo cual se realiza también con la herramienta EDK.

3. Programar el FPGA.

Una vez implementado el diseño, se debe generar un archivo para configuración de la FPGA, llamado *download.bit*. El archivo BIT puede ser utilizado directamente con la FPGA o puede ser convertido a un archivo PROM para almacenar la información de programación Para poder programar el FPGA con la arquitectura descrita en la PP, es necesario conectar la tarjeta de desarrollo Starter Kit Spartan-3E con una computadora que contenga el archivo de configuración, a través de un puerto USB de la computadora hacia la tarjeta, por medio de un cable. EDK cuenta con herramientas para la programación del FPGA.

4. Configuración de la computadora auxiliar.

Se trata de una computadora de escritorio con sistema operativo Debian 5 (Lenny), debe contar con un puerto disponible de USB para conectar el cable convertidor de USB-Serial. La computadora

debe contar con un programa para la trasmisión serial, existe varios programas de este estilo, como lo son Kermit, minicom, cutecom , etc. En este caso utilizaremos **Kermit** realizar la comunicación.

La configuración de los parámetros de trasmisión y del protocolo RS-232 es un aspecto importante para lograr una comunicación correcta entra la computadora auxiliar y la PP. E puerto serial debe ser configurado con los siguientes parámetros:

- Tarifa de banda= 9600
- Bits de datos= 8
- Paridad impar= 0
- Paridad = 0

5. Realizar la comunicación entre la computadora auxiliar y la plataforma de pruebas.

Para iniciar el programa de `kermit` es suficiente con que en línea de comando se ejecute la siguiente instrucción:

```
$ kermit -c -l /dev/ttyUSB0
```

El parámetro `-c` es para indicar que inicie la conexión, mientras que el `-l /dev/ttyUSB0` indica desde que dispositivo va hacer la conexión, en nuestro caso el sistema operativo de la computadora inicial identifico el cable serial como `/dev/ttyUSB0` pero éste puede cambiar dependiendo de hardware con que cuente la máquina.

Aquí hay un ejemplo de ejecución y de la salida normal del programa.

```
$ kermit -c
Connecting to /dev/ttyUSB0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
```

---

6. Llevar acabo de prueba de puerto serial.

Para verificar el funcionamiento fue necesario conectar la plataforma de pruebas por medio del puerto serial hacia una computadora, con el fin de que ambas pudieran realizar trasmisión y recepción de cadena de caracteres entre ellas (ver Figura 5.7).

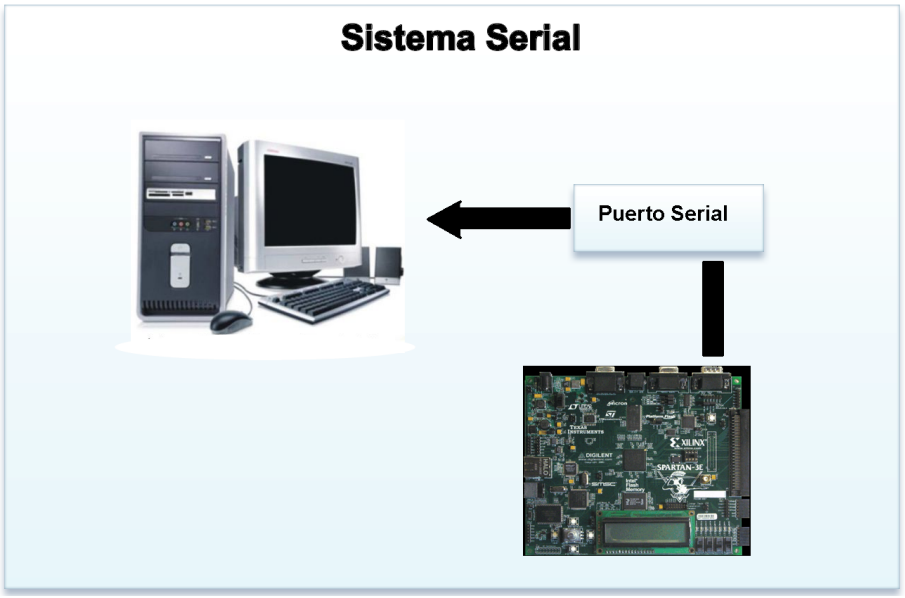


Figura 5.7: Diagrama a bloques de UartLite

Con la terminal de `Kermit` se pudo visualizar las cadenas de caracteres que eran enviadas desde la plataforma de pruebas, así como también enviar caracteres hacia ella. Los caracteres recibidos por la plataforma de pruebas eran enviados hacia la computadora con el fin de que visualizar las cadenas que habían sido enviadas y así verificar el funcionamiento.

Se comprobó que cuando el MCA contenía reglas que permiten el acceso al dispositivo el funcionamiento de la trasmisión-recepción era normal y el retardo que el MCA agrega al sistema es imperceptible. Cuando el MCA fue configurado con reglas que no permitían el acceso de lectura y escritura al puerto serial, el comportamiento fue el deseado, dado que la terminal de `Kermit` no

desplegaba ninguna de las cadenas que la plataforma de pruebas y tampoco permitía el envío de las cadenas.

A continuación un ejemplo del funcionamiento de la prueba:

```
<F2>$ kermit -c
Connecting to /dev/ttyUSB0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
Computadora envio: Hola Mundo
Plataforma de pruebas recibio: Hola Mundo
```

5.3 Parámetros de desempeño

Los parámetros de desempeño nos permiten tener una aproximación métrica del comportamiento de un sistema, empleando algunas variables relacionadas con aspectos como el espacio físico, que en hardware determinan la eficiencia de una implementación. Los parámetros de desempeño empleados en este trabajo son los slices, periodos de reloj y la frecuencia (MHz). El análisis de estos parámetros se muestra a continuación.

5.3.1 Parámetros de espacio y tiempo

En la Tabla 5.8 se muestran los datos obtenidos del análisis al módulo de control de acceso utilizando una memoria RAM. Se observa que los recursos utilizados, hablando de slices son muy pocos y éstos se mantienen constantes de acuerdo al incremento de número de reglas. El periodo de ciclo de reloj se encuentra entre 6 y 7 ns para los tres diseños, lo que significa que el número de reglas no afecta ninguno de estos parámetros. Esto se debe que en el diseño del MCA-RAM lo único que incrementa de tamaño es la memoria ya que se tiene que almacenar más reglas, pero el control

sigue siendo el mismo. Mientras la máquina de estados finitos del MCA se encuentre en el estado **CHECAR**, el MCA analiza cuatro reglas por ciclo, lo que significa que entre más reglas se tenga, la MEF permanecerá más ciclos en el estado **CHECAR**.

Reglas	Slices	Periodo de reloj (ns)	Frecuencia (Mhz)	Caso Promedio (No. de ciclos)	Peor caso (No. de ciclos)	Mejor caso (No. de ciclos)
16	186	6.975	143.363	4	6	3
32	187	6.107	163.734	7	10	3
64	210	6.180	161.820	11	18	3

Tabla 5.8: Recursos ocupados por el diseño de MCA con RAM

En cambio en el diseño del MCA-CAM los datos obtenidos muestran otro comportamiento. La cantidad de recursos ocupados (slices) incrementa proporcionalmente al número de reglas, al igual que el periodo de ciclo de reloj que va de $9.5\ ns$ para el de menor número de reglas hasta $12.5\ ns$ para el 64 reglas. En la Tabla 5.9 se muestran los resultados obtenidos.

El uso de recursos empleados en este diseño se justifica debido al uso de una memoria CAM, la cual conforme el número de reglas que contenga la demanda de recursos incrementa, ya que utiliza para cada regla una unidad de búsqueda que entre otra cosas contiene un comparador de 20 bit.

Reglas	Slices	Periodo de reloj (ns)	Frecuencia (Mhz)	Caso Promedio (No. de ciclos)	Peor caso (No. de ciclos)	Mejor caso (No. de ciclos)
16	541	9.561	103.831	3	3	3
32	1012	11.694	85.512	3	3	3
64	1911	12.522	79.859	3	3	3

Tabla 5.9: Recursos ocupados por el diseño de MCA con CAM

El comportamiento descrito acerca del espacio utilizado por los dos modelos de memoria puede observarse en la gráfica 5.9, de acuerdo a los experimentos el modelo que emplea la memoria RAM describe un mejor comportamiento ya que el incremento del espacio requerido es mucho menor al descrito por el modelo que emplea la memoria CAM. Este comportamiento se ve reflejado de igual manera al momento de considerar el tiempo en ciclos de reloj empleado por ambos modelos de memoria utilizando, obteniendo mejores resultados con la memoria RAM, ver Figura 5.8.

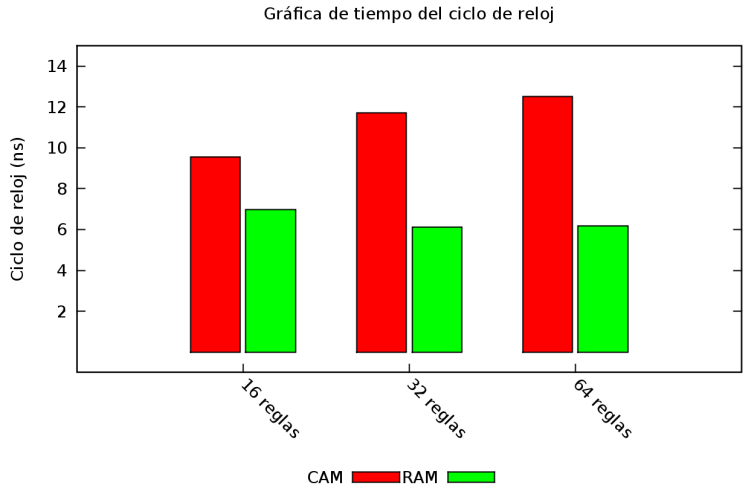


Figura 5.8: Gráfica de la duración del ciclo de reloj para cada diseño

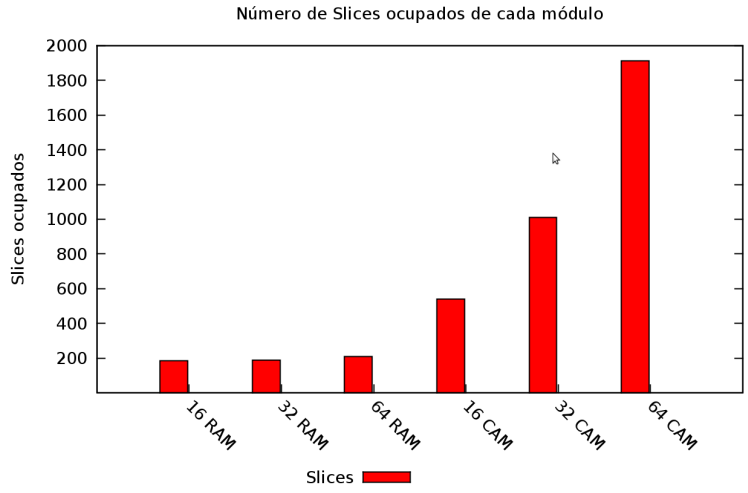


Figura 5.9: Gráfica de los recursos demandados por cada versión del MCA

Sin embargo, es importante tomar en cuenta que los dos modelos implementados varían drásticamente en la cantidad de ciclos de reloj empleados para encontrar una regla que coincide con la instrucción de entrada. En el capítulo anterior se determinó que el total de ciclos de reloj para el MCA-RAM

es a lo más de $2 + num_{reglas}/4$, mientras que el del MCA-CAM es de 3. Este parámetro es muy importante porque refleja el retardo real que será puesto al PLB.

5.3.2 Tiempo de ejecución

El tiempo de ejecución se refiere al tiempo real que se emplea para realizar una corrida del modelo empleado. Para tener una idea acerca del comportamiento del tiempo requerido se calculan tres casos, el Mejor Caso, el Caso Promedio y el Peor Caso. Estos tres casos describen las tres principales circunstancias que podrían presentarse durante la ejecución.

Para obtener los tiempos de ejecución de los dos modelos, se tomó en cuenta una simulación con las herramientas ISE y MoldelSim. Al realizar estas simulaciones se ve claramente que ambas implementaciones cumplen con el comportamiento esperado en el diseño.

En la Figura 5.10 se muestra el comportamiento de ambos diseños para el mejor caso, caso promedio y peor caso. Los datos graficados son obtenidos a partir de multiplicar el número de ciclos que ocupa el MCA para dar una decisión con el periodo de reloj en cada uno de los diseño. A pesar que en el mejor caso el comportamiento del MCA-RAM-4 es más estable que el tiempo requerido para el MCA-CAM-n, puede apreciarse en las barras del caso promedio el comportamiento del MCA-CAM-n es mas estable que el del MCA-RAM-4 debido a que la memoria CAM realiza una comparación en un solo ciclo de reloj para hacer el escaneo del contenido.

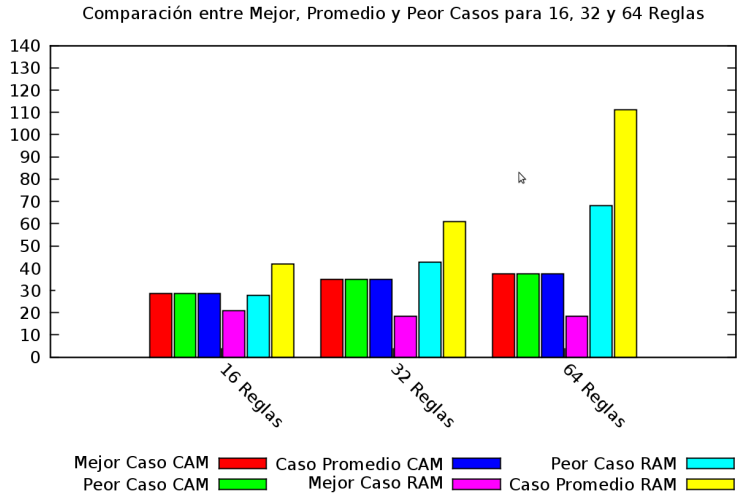


Figura 5.10: Gráfica de tiempo de ejecución en el mejor caso

6

Conclusiones y trabajo futuro

En la actualidad la virtualización de plataforma es una tecnología cuyo uso ha aumentado, sobretodo en empresas de tecnología de la información (TI). Mediante el uso de los VMM's nos permite la creación de múltiples VM lo cual facilita el aprovechamiento de los recursos físicos de la computadora. La principal ventaja es que cada máquina virtual puede ejecutar diferentes sistemas operativos, y a pesar de las ventajas que esto representa es también necesario considerar que con lleva ciertos inconvenientes principalmente el decremento en el rendimiento debido a la emulación. Además cabe mencionar que la seguridad se ve afectada de igual manera debido a la comunicación entre las diferentes VM que en algunas ocasiones es requerida.

Si consideramos que una de las grandes preocupaciones en el sistemas de cómputo actuales es la seguridad del sistema, un sistema virutalizado debería garantizar que la protección de la información, las máquinas virtuales contienen sistemas operativos que tiene como definición que son los únicos dueños del hardware, la virtualización debe hacer que las VM son únicas en el sistema administrando todos los recursos de hardware de la plataforma física.

Nuestra principal aportación es el desarrollo en hardware del módulo que permite el control de acceso a dispositivos físicos, lo cual permite tener un control de las peticiones de las VM a los

diferentes dispositivos de E/S sin tener que agregar una carga más a las funciones que realiza el VMM. Con este fin se diseñó e implementó en hardware un MCA (Módulo de Control de Acceso), el cual sirve de apoyo al VMM para facilitar el control de acceso las VM, implementado en hardware reconfigurable. Este módulo fue situado entre el procesador y los dispositivos de E/S con el fin de que sirva de árbitro. El MCA se implementó en hardware reconfigurable, empleando una lista de control de acceso (ACL por sus siglas en inglés *Access Control List*). El MCA interpreta cada una de las políticas incluidas en la ACL y maneja la decisión de control de acceso a los dispositivos.

En estas tesis fueron diseñadas e implementadas dos versiones de un módulo de control de acceso en hardware, las cuales difieren en el tipo de estructura de memoria que cada una contiene. Para este caso son dos la MAC-RAM-4 y MAC-CAM-n. El primero consta de una memoria RAM y de acuerdo a el análisis realizado este modelo es el más apropiado cuando el área de hardware es limitada pero aún así se requiere controlar el acceso a los dispositivos, la desventaja es que aumenta una latencia de $2 + num_{reglas}/4$ ciclos. La segunda consiste en una memoria del tipo CAM la cual tiene la característica de que sin importar el área disponible de hardware obtiene el mismo tiempo de ejecución ya que sólo da una latencia de 3 ciclos de reloj para poder dar el resultado. Como puede verse en gráfica 5.8 el comportamiento, en cuanto a rendimiento en tiempo, de las dos memorias no es el mismo, en los tres posibles casos (mejor, promedio y peor casos) la memoria CAM se comporta de manera más estable, debido principalmente a que mediante comparadores puede realizar la búsqueda de las instrucciones almacenadas en un solo ciclo de reloj. En cambio, como puede verse en la gráfica 5.9, la memoria RAM utiliza menos recursos en memoria por lo que es más adecuado para plataformas con menor espacio de almacenamiento.

Fue realizado el diseño de la estructura de reglas para el módulo de control de acceso, tomando en cuenta los requerimientos que debía cumplir el MCA. Esta regla se compuso de dos campos, instrucción y decisión. EL campo de instrucción fue formado por el tipo de operación, la dirección del esclavo y el identificador de la máquina virtual. Por otro lado el campo de decisión constaba de los mecanismos de validación para permitir o no el acceso a los dispositivos físicos en la computadora. Una condición para el diseño de la regla es la suposición de la existencia de un identificador de VM, el cual nos permitiría no sólo controlar el tipo de operación que se quiere realizar sino también que

VM tiene acceso a ese dispositivo.

Además se realizó el diseño e implementación de una plataforma de pruebas, que permitía hacer la verificación del funcionamiento del MCA. La PP está compuesta de una tarjeta de desarrollo que cuenta con un FPGA Spartan-3E el cual fue programado con un archivo de configuración que describe la arquitectura interna del sistema (procesador, bus y periféricos). Con esta plataforma permitió la evaluación de pruebas obtuvieron los resultados del módulo de control, comparando las implementaciones de los dos diseños propuestos. Las Pruebas funcionales de comportamiento realizadas mostraron resultados positivos, esto se reflejo a través de los diagramas de estado, los cuales permitieron verificar si el módulo se comportaba tal cual había sido descrito en la etapa de diseño.

Aunque los objetivos establecidos al inicio de este trabajo de tesis fueron alcanzados y se cumplió con la funcionalidad prometida hay algunos aspectos que podrían ser mejorados. Estos puntos se discuten en la siguiente subsección.

6.1 Trabajo futuro

Debido a que la implementación en versión hardware del módulo de control de acceso es un diseño único aun quedan algunos puntos por refinar. Los principales puntos que podrían ser mejorados se describen a continuación:

El principal punto que podría ser inspeccionado es agregar la configuración dinámica de las reglas del MCA, esto permitiría que el proceso de configuración de las reglas fuera más flexible ya que no sería necesario hacer la síntesis de todo el sistema. Esto permitiría una gran flexibilidad en la implementación ya que un usuario cualquiera podría realizar la configuración de los parámetros sin tener que hacerlo a nivel hardware.

Otro punto importante es que podrían ser exploradas otras arquitecturas para el diseño del MCA, para optimizar el uso de los recurso o bien el disminuir la latencia inherente por incluir el MCA dentro de la lógica del bus PLB. Las cuales podrían hacer uso de otras herramientas, principalmente de memoria, para intentar mejorar el rendimiento al momento de realizar la búsqueda de las instrucciones

almacenadas.

Por último se podría obtener una mejora significativa al incluir dentro de la arquitectura del procesador una señal de calificación de transferencia que proporcione el identificador de la máquina virtual que está solicitando el dispositivo físico. Este procedimiento permitiría poder manejar a nivel de completamente en hardware el acceso de E/S hacia estos dispositivos.

A

Integrar sistema operativo Linux a la plataforma de pruebas

El principal objetivo de esta apendice es implementar un sistema operativo Linux sobre el procesador Microblaze. Para poder hacer es necesario antes llevar acabo los siguientes pasos:

- Asignar los parámetros de la plataforma necesarios para la implementación de Linux
- Compilar el núcleo de Linux para un diseño especifico y cargar la imagen al FPGA.

A.1 Instalación de Petalinux

El primer paso es descargar la distribución de Petalinux, se trata de una descarga libre será suficiente con registrase en la página principal de Petalogix.

Ya que este descargado el archivo que contiene la distribución, que en este caso será el de la versión 4-rc3 simplemente se coloca en el directorio que nos convenga, no es necesario ser `root` para llevar a cabo la instalación de Petalinux. En este ejemplo lo haremos en el `home` de uno usuario.

```
$ tar xvf petalinux-v0.40-rc3.tar.gz
```

El comando anterior nos creará un directorio llamado `petalinux-v0.40-rc3`. Dentro de éste directorio se encuentran otros 3 directorios que básicamente están divididos en su funcionalidad: el de herramientas que contiene el compilador `gcc` y scripts de petalinux, el de hardware que consiste en proyectos ejemplos del EDK, y el de software en donde se encuentras las fuentes del núcleo de Linux para el `2.4` y `2.6`, y el entorno principal de `petalinux-dist`. La estructura de directorio de **Petalinux** se ilustra a continuación.

```
petalinux
   + tools
   |   + common
   |   |   + petalogix
   |   |       + bin
   |   + linux-i386
   |       + microblaze-uclinux-tools
   |
   + software
   |   + petalinux-dist
   |   + linux-2.6.x-petalogix
   |   + uClinux-2.4.x
   |   + user-apps
   |   + user-modules
   |
   + hardware
       + reference-designs
       + user-platforms
       + edk_user_repository
```

A.1.1 Configuración de variables de ambiente

Otro punto importante de la instalación es la configuración de las variables de ambiente, las cuales son necesarias para la ejecución de scripts que vienen junto con la distribución. Dentro del directorio raíz de la instalación, es decir, `petalinux-v0.40-rc3` se encuentran dos scripts,`settings.csh` y `settings.sh`, se tiene que ejecutar el que corresponda a la consola que se este utilizando.

```
$ echo $SHELL
```

Si la salida del comando arroja `/bin/bash` significa que se utiliza **Bash**, la extensión típica es `*.sh` y si da `/bin/csh` entonces se utiliza **C Shell** su extensión típica es `*.csh`. El siguiente es un ejemplo para el caso de que se esté utilizando **Bash** como consola.

```
 $ source settings.sh
PetaLinux environment set to '/home/usuario/petalinux-v0.40-rc3'
```

Debe de ejecutar el script de configuración **cada vez** que se abra una nueva terminal en la que se quiera trabajar con los archivos de la distribución, para evitar el hacerlo constantemente, se puede modificar el archivo `~/.bashrc` e incluir las configuraciones de las variable de ambiente.

A.1.2 Configuración de programa de comunicación serial

Esta sección se muestra los pasos necesarios para configurar el programa de comunicación serial. Existe varios programas de este estilo, como lo son Kermit, minicom, cutecom , etc. En este caso utilizaremos **Kermit** y los pasos para instalación y configuración son los siguientes:

```
# apt-get install ckermit
```

Después de haberlo instalado es necesario crear o modificar sobre la ruta del `home` del usuario el archivo `.kermrc`, el cual es un script que se ejecutará por defecto cuando se ejecute el programa `kermit`.

```
set line /dev/ttyUSB0
set speed 115200
set carrier-watch off
set handshake none
set flow-control none
robust
set file type bin
set file name lit
set rec pack 1000
set send pack 1000
set key \127 \8
set key \8 \127
set window 5
```

Lo que se indica con este script es que el dispositivo que se utilizará es el `/dev/ttyUSB0`, para el caso que estemos emulando el puerto serial con una usb, en caso de que el dispositivo que se quiera utilizar solo tenemos que modificar el script con la ruta correcta. También se pueden modificar el resto de los parámetros como la velocidad, paridad, etc.

Para iniciar el programa de `kermit` es necesario estar en el directorio `home` del usuario o bien indicarle la ruta del script de configuración (`~/.kermrc`), así como también indicarle que inicie en como "conectado" con el parametro `-c`. Aquí hay un ejemplo de ejecución y de la salida normal del programa.

```
$ kermit -c
Connecting to /dev/ttyUSB0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
```

A.1.3 Configuración de redes

U-boot una de las herramientas incluidas con petalinux permite descargar/recuperar imágenes o archivos a través de un servidor TFTP (por su sigles en inglés Trivial File Trasfer Protocol), esto ayuda a reducir significativamente el tiempo de configuración de las imágenes en los FPGA. Para instalar el servidor de TFTP es necesario tener instalados los paquetes: `xinetd`, `tftp`, `tftpd`. En caso de no tenerlos instalados, lo podemos hacer de la siguiente manera:

```
# apt-get install xinetd tftp tftpd
```

A.1.3.1. Configuración del directorio de transferencia

Es recomendable crear un `/tftpboot` en el directorio raíz /, el cual es un directorio de transferencia comúnmente usado en los sistemas de Linux embebidos para actualizar las imágenes y otros archivos de arranque de manera automática.

Para crear éste directorio es necesario tener privilegios de `root` y para que pueda ser utilizado por cualquier usuario del sistema se modifican los permisos de lectura, escritura y ejecución.

```
# mkdir /tftpboot
# chmod -R 777 /tftpboot
# chown -R nobody /tftboot
```

A.1.3.2. Configuración del servidor TFTP

Se crea o modifica el siguiente archivo `/etc/xinetd.d/tftp` incluyendo lo siguiente:

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
```

```
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /tftpboot
disable         = no
}
```

Para iniciar el demonio de `xinetd` se ejecuta el siguiente comando:

```
$ sudo /etc/init.d/xinetd start
```

Para comprobar que el servicio este establecido podemos ejecutar el siguiente comando y la salida debe ser muy similar a la presentada a continuación:

```
$ netstat -l -u ¿ grep tftp
udp        0      0 *:tftp                      *:*
```

A.2 Configurar Petalinux

La distribución de PetaLinux está diseñada para complementar el proceso de diseño con la herramienta de Xilinx EDK. Esto permite la plataforma de hardware que hemos creado en usando estas herramientas, puedan ser integradas a la estructura de directorios de código fuente de PetaLinux.

A.2.1 Agregar proyecto de hardware

Los proyectos definidos por los usuarios pueden ser colocados en el directorio `$PETALINUX/hardware/user-platforms`, para eso copiamos el directorio que contiene la descripción de nuestro proyecto de hardware. En este ejemplo el directorio que `plataforma-de-pruebas-Spartan3E500-RevD-edk101`.

```
$ cp -rf /ruta_de_ubicacion/plataforma-de-pruebas-Spartan3E500-RevD-e
dk101 $PETALINUX/hardware/user-platforms
```

El comando anterior copia recursivamente el directorio /ruta_de_ubicacion/plataforma-de-pruebas-Spartan3E500-RevD-edk101 en la ruta $PETALINUX/hardware/user-platforms, recordemos que antes fueron configuradas de ambiente y por lo tanto las variable $PETALINUX ya contiene la ruta absoluta del directorio de instalación de PetaLinux.

Una vez que la plataforma de hardware ha sido definida se requiere generar un conjunto de parámetro de software sobre la base de la plataforma de hardware. Estos parámetros son utilizados para iniciar y comunicar el hardware. Esto es conocido como BSP (por su siglas en inglés Board Support Package). PetaLinux include dentro de su distribución un BSP exclusivo, el cual se requiere para que funcionen el U-boot de arranque y el sistema operativo Linux. Dentro del directorio del proyecto de hardware se encuentra el archivo system.mss que contiene la descripción de los parametros de software del proyecto, es necesario modificar los parametros de la instancia de OS e indicarlos de la siguiente manera PARAMETER OS_NAME=petalinux y PARAMETER OS_VER=1.00.b.

```
BEGIN OS
 PARAMETER OS_NAME = petalinux
 PARAMETER OS_VER = 1.00.b
 PARAMETER PROC_INSTANCE = microblaze_0
    ⋮
```

Es necesario hacer unas modificaciones a nuestro proyecto de hardware (system.xmp), para indicar que utilizará el BSP de PetaLinux, dentro del directorio de $PETALINUX/hardware se encuentra el repositorio de EDK que contiene el BSP de PetaLinux (edk_user_repository). Con el fin de que EDK pueda tener acceso al BSP se tiene que indicar en el parametro ModuleSearchPath la ruta en donde se encuentra el BSP que en este caso es dos niveles arriba del directorio del proyecto, es decir ../../edk_user_repository/. A continuación un ejemplo del archivo.

```
XmpVersion: 10.1.03
VerMgmt: 10.1.03
IntStyle: default
```

```
ModuleSearchPath: ../../edk_user_repository/
MHS File: system.mhs
MSS File: system.mss
NPL File: projnav/system.ise
Architecture: spartan3e
Device: xc3s500e
Package: fg320
    ⋮
```

Para añadir nuestra plataforma al menú de configuración del núcleo de Linux, la distribución de PetaLinux tiene un script llamado `petalinux-new-platform` que toma los datos necesarios de la descripción del hardware y mapeo de memoria de cada dispositivo descrito en el proyecto para meterlo como archivo de configuración del núcleo. Los parámetros de entrada de este script son:

- -v Nombre del vendedor de la plataforma
- -p Nombre de la plataforma
- -k La versión de núcleo de Linux

Nos cambiamos hacia el directorio `$PETALINUX/software/petalinux-dist` y ejecutamos el script de la siguiente manera:

```
$ cd $PETALINUX/software/petalinux-dist
$ petalinux-new-platform -v Xilinx -p plataforma-de-pruebas-Spar
tan3E500-RevD-edk101 -k 2.6
New platform for Xilinx plataforma-de-pruebas-Spartan3E500-RevD-
edk1
01 MyVendorName successfully created
Please use petalinux-copy-autoconfig tools with the same paramet
ers (-v, -p)
```

Después de ejecutar este script ya es posible seleccionar la nueva plataforma dentro del menú de configuración del núcleo. En este momento deberían de aparecer en el "Vendor" y en "Select the Product".

```
--------------------------------------------------------------------------------
 ________________________ Vendor/Product Selection ________________________
|  Arrow keys navigate the menu.  <Enter> selects submenus --->.           |
|  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,  |
|  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.      |
|  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable      |
|  ____________________________________________________________________   |
| |--- Select the Vendor you wish to target                              | |
| | (Xilinx) Vendor                                                      | |
| | --- Select the Product you wish to target                            | |
| | (plataforma-de-pruebas-Spartan3E500-RevD-edk101) Xilinx Products     | |
| |                                                                      | |
| |______________v(+)_____________________________________________________| |
|_____________________________________________________________________________|
|                     <Select>    < Exit >    < Help >                     |
|_____________________________________________________________________________|
```

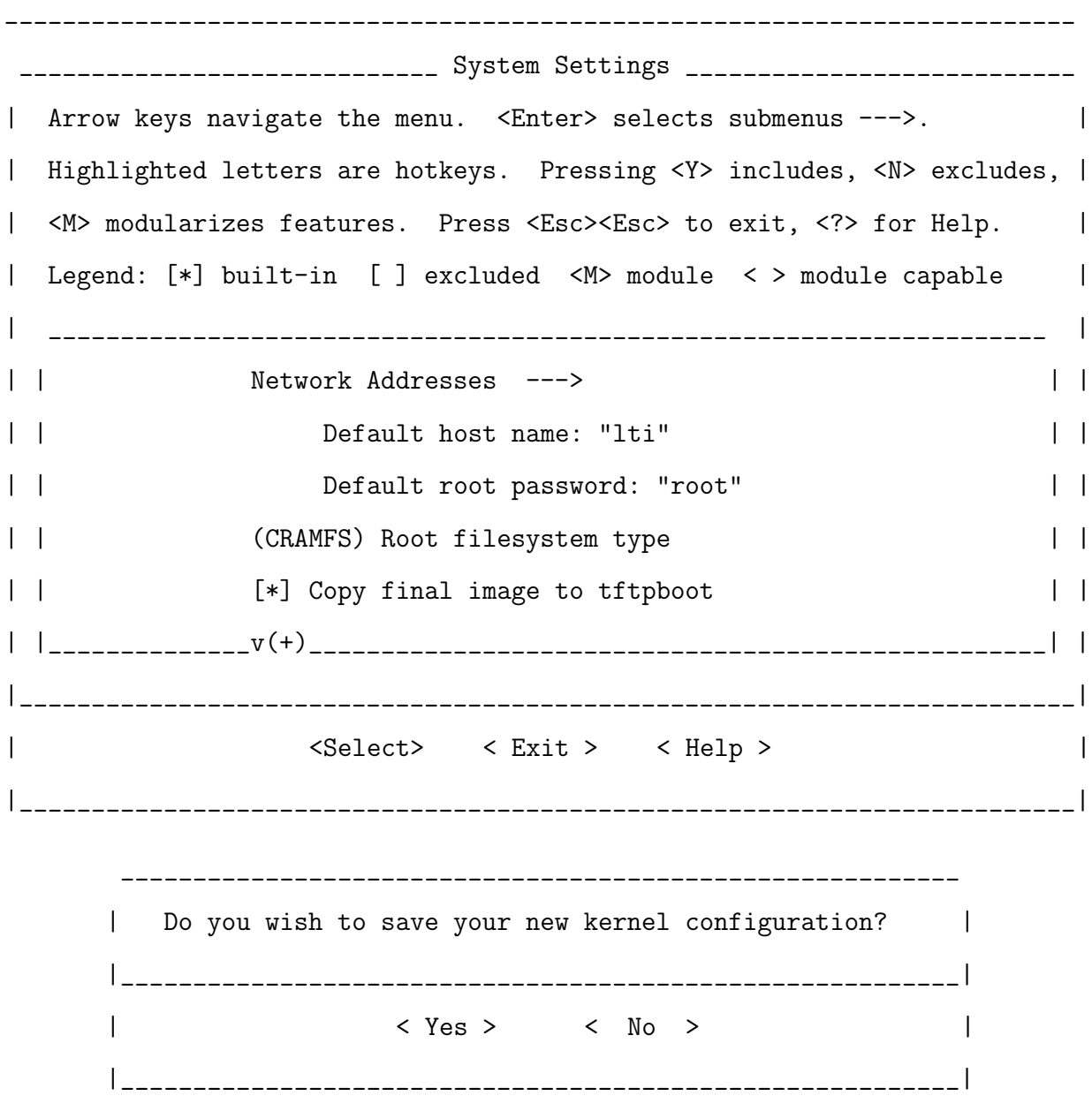
Ya que seleccionamos nuestro vendedor de plataforma y el nombre de la plataforma que acabamos de agregar damos enter en `Exit` y seleccionamos la siguiente opcion `Kernel/Library/DefaultsSelection` para poder personalizar las opciones del núcleo de Linux y del usuario, eso es seleccionando `CustomizeKernelSet` y `CustomizeVendor/UserSettings`.

```
--------------------------------------------------------------------------
 ____________________ Kernel/Library/Defaults Selection _________________
|  Arrow keys navigate the menu.  <Enter> selects submenus --->.          |
|  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, |
|  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.     |
|  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable     |
|  ___________________________________________________________________    |
| |              (linux-2.6.x) Kernel Version                         | |
| |              (None) Libc Version                                  | |
| |              [ ] Default all settings (lose changes)              | |
| |              [x] Customize Kernel Settings                        | |
| |              [x] Customize Vendor/User Settings                   | |
| |_____________v(+)__________________________________________________| |
|_________________________________________________________________________|
|                    <Select>    < Exit >    < Help >                     |
|_________________________________________________________________________|
```

Ahora seleccionamos `Exit` y de nueva cuenta `Exit`, en la siguiente pantalla damos `Yes` para guardar la configuración y en seguida nos aparecen los menus para personalizar las opciones del núcleo de Linux y del usuario.

```
        ___________________________________________________________
       |   Do you wish to save your new kernel configuration?      |
       |____________________________________________________________|
       |                   < Yes >      <  No  >                    |
       |____________________________________________________________|
```

El primer menú que saldrá será el de las opciones del núcleo.



A.3 Preparar el hardware

Ya antes se dieron los pasos para agregar nuestro proyecto de hardware a la estructura de directorio que utiliza el PetaLinux, también se agregado el ejecutable de FS-Boot a nuestro proyecto, ahora es necesario construir de nueva cuenta nuestro proyecto para que todos los cambios que hemos realizado

se ven reflejados en el archivo `dowload.bit` FS-arranque de Apoyo FS-arranque es la primera etapa de arranque que se utiliza para arrancar el sistema de destino. En esta sección se asume que el FS-arranque el gestor de arranque ya está agregado a la meta de hardware proyecto. Consulte la Sección FS-arranque para obtener más informac ión sobre el FS-arranque. FS-para incluir a su gestor de arranque de arranque de proyectos de hardware, consulte el FS-Inc luye la sección de arranque. La construcción de su hardware En esta sección se asume que su hardware proyecto ya está añadido a la PetaLinux árbol de código fuente. Si no ha añadido el hardware proyecto. Se refieren a la adición de nuevo hardware Sección proyec to para añadir la información sobre su hardware proyecto para PetaLinux. Cambie el directorio a su directorio en el proyecto de hardware PetaLinux.

```
cd $PETALINUX/hardware/user-platforms/plataforma-de-pruebas-Spartan3E
   500-RevD-edl101
```

Ahora se tienen que generar los Makefiles de nuestro proyecto.

```
$ xps -nw system.xmp
% save make
% exit
```

El comando anterior genera el archivo `system.make` que puede tener de entrada los siguientes argumentos:

1. `bits`. Genera el bitstream de hardware.
2. `libs`. Genera las bibliotecas del proyecto.
3. `program`. Hace la compilación de las aplicaciones que contenga el proyecto, en este caso hará la compilación del FS-Boot.
4. `init_bram`. Parámetro para incializar el BRAM
5. `download`. Realiza la descarga del archivo .bit al FPGA.

La forma en la que se van a ejecutar los comandos es la siguiente:

```
$ make -f system.make bits
$ make -f system.make libs
$ make -f system.make program
$ make -f system.make init_bram
```

Se debe de tomar en cuenta que este proceso toma varios minutos en concluirse, en caso de que exista algún problema durante alguno de estos comandos, se tiene que solucionar el problema y volver a ejecutarlos.

El siguiente paso es ejecutar el script de PetaLinux que sirve para crear/modificar archivos de configuración del núcleo en donde se le indican los parámetros básicos de del hardware, así como los del software.

El script `petalinux-copy-autoconfig` realiza este proceso de forma automática, solo es necesario estar en el directorio que contenga nuestro proyecto de hardware que para este ejemplo es `$PETALINUX/hardware/user-platforms/plataforma-de-pruebas-Spartan3 E500-RevD-edk101`. Es recomendable indicarle al comando los parametros que son opcionales pero que definen el nombre del proyecto de EDK, la versión del núcleo y el fabricante de la plataforma, basta con ejecutarlo de la siguiente manera:

```
$ petalinux-copy-autoconfig -v Xilinx -p plataforma-de-pruebas-
  Spartan3E500-RevD-edl101 -k 2.6
INFO: Attempting vendor/platform auto-detect
INFO: Auto-detected Xilinx/plataforma-de-pruebas-Spartan3E500-RevD
-edk101 combination.
Auto-config file successfully updated for Xilinx plataforma-de-pru
ebas-Spartan3E500-RevD-edk101
```

Después de este paso la plataforma nueva debe de estar configurada de forma correcta para ser compilar una imagen del núcleo para ese hardware.

El menú principal de configuración del núcleo de Linux es el siguiente:

A.4 Compilación de PetaLinux

La compilación de Petalinux consiste en:

- Compilar el núcleo de Linux
- Compilar las herramientas de GNU
- Construir el sistemas de archivos raíz (rootfs)
- Compilar el sistema de arranque U-Boot

Hay que asegurar de estar situado sobre el directorio `$PETALINUX/software/petalinux-dist`.

```
$ cd $PETALINUX/software/petalinux-dist
$ yes "" | make oldconfig dep all
```

El comando anterior va realizar la compilación de la imagen del kernel correspondiente a nuestro hardware definido en el proyecto de EDK. El tiempo de ejecución es un poco alto, en el caso de que se produzca algún error se debe de arreglar y volver a ejecutar el comando anterior hasta que marque que las imágenes fueron creadas.

Dependiendo de las opciones de entrada los archivos que se van a crear los archivos descriptos en la tabla A.1

A.5 Iniciando el sistema

Existen diversas formas de arrancar un sistema empotrado, y depende en gran medida del gestor de arranque utilizado y de los tipos de configuración de arranque. En el caso de PetaLinux tiene doble proceso de arranque con el fin de lograr una óptima configuración del sistema. A continuación se da una breve descripción de las fases para inicar el sistema.

1. **Cargar la primera fase de arranque**. El arranque inicial, Se requiere que el archivo `dowloand.bit` sea descargado al FPGA. Cuando este proceso se concluya se inciará el FS-Boot que es el gestor de arranque.

Nombre de la imagen	Descripción
Linux Kernel	
image.bin	El núcleo de Linux y el sistema de archivos raíz en formato binario
image.elf	El núcleo de Linux y el sistema de archivos raíz en formato ELF
image.srec	El núcleo de Linux y el sistema de archivos raíz en formato SREC
image.ub	El núcleo de Linux y el sistema de archivos raíz en formato U-Boot
linux.bin	Imagen del del núcleo de Linux sin sistema de archivos
romfs.img	La imagen de ROMFS en formato binario
U-Boot	
u-boot.bin	La imagen de U-Boot en formato binario
u-boot.srec	The U-Boot image en formato SREC
u-boot-s.bin	La imagen reubicable de U-Boot en formato binario
u-boot-s.elf	La imagen reubicable de U-Boot en formato ELF
u-boot-s.srec	La imagen reubicable de U-Boot en formato SREC
ub.config.img	Plataforma de secuencia de comandos de configuración de U-Boot en formato binario

Tabla A.1: Descripción de la imágenes creadas durante la compilación de Petalinux

2. **Descargar la imagen U-Boot SREC**. Para descargar esta imagen al FPGA se tiene que interrumpir el proceso del FS-Boot tecleando la letra 's'. Recordemos que las imagenes creadas en el proceso de comilación fuera depositadas en el directorio `/tftpboot` entonces ahi podemos encontrar el archivo `u-boot.srec`. FS-Boot iniciará automáticamente U-Boot cuando la descarga de la imagen haya sido completada.

3. **Cargar núcleo de Linux**. Una vez que U-Boot se inicia es posible utilizar el U-Boot para cargar el la imagen del núcleo de Linux en la memoria SDRAM que se encuentra en la tarjeta de desarrollo. Para descargar la imagen se puede utilizar el puerto serie o bien la conexión Ethernet, el archivo es el `image.bin` y debe ser cargado en la dirección base de la memoria SDRAM

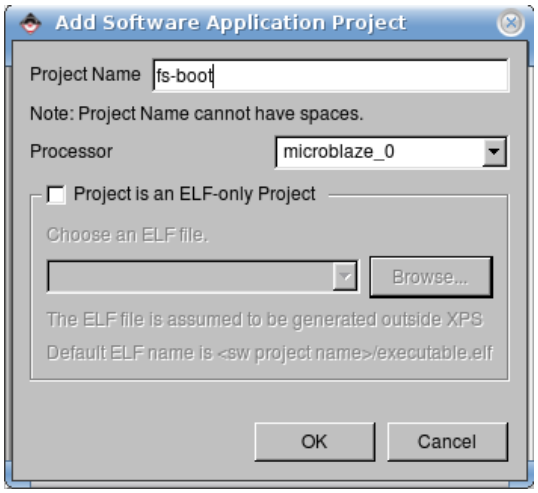
A.5.1 FS-Boot

A.5.1.1. Configuración

La distribución de PetaLinux trae preinstalado el FS-Boot y se puede encontrar dentro del directorio `$PETALINUX/hardware/fs-boot`. Los archivos que componen esta aplicación son:

1. Fuentes
    - fs-boot.c
    - srec.c
    - time.c
2. Encabezados
    - fs-boot.h
    - srec.h
    - time.h

Para agregar esta aplicación a nuestro proyecto en la ventana principal del XPS, en el menú `Software->AddSoftwareApplicationProyect` se introduce como nombre del proyecto: **fs-boot** y escogemos como procesador : **microblaze_0**, como se muestra en la siguiente imagen.



Ya que la aplicación fue creada nos dirijimos hacia el área de las aplicaciones que se encuentra en `ProjectInformationArea->Applications->fs-boot` ahi se puede añadir al proyecto de software los archivos fuentes y de encabezados que mencionamos anteriormente y también es conveniente asegurarnos que las opciones de compilación sean las mostradas en la Tabla A.2.

Opción	**Valor**
Environment	
Application Mode	executable
Output ELF file	default value
Linker Script	Use default Linker Script
Stack Size	1K
Debug and Optimisation	
Optimization Level	Size Optimized (-Os)
Paths and Options	
Other Compiler Options to Append	-Wall

Tabla A.2: Opciones de compilación.

El siguiente paso es marcar el proyecto de `fs-boot` como el que va ser inicializado en la BRAM, ningún otro proyecto debe ser marcado para iniciar en el BRAM.

Por último necesitamos volver a generar las bibliotecas del proyecto y compilar la aplicación.

```
$ make -f system.make libs
$ make -f system.make program
$ make -f system.make init_bram
```

A.5.1.2. Uso del FS-Boot

Como se muestra en la figura **??**, debemos de tener al menos dos consolas diferentes, una donde que ejecute la consola de **kermit**, si recordamos cuando se realizó la configuración del programa se creo un archivo `.kermrc` en el home del usuario, entonces ahora es solo suficiente con ejecutar `$kermit-c`.

En la otra terminal debemos de estar situados en el directorio de nuestro proyecto y realizamos la descarga del `download.bit`.

```
$ make -f system.make download
```

Una vez que la descarga al FPGA ha concluido debe de aparecer en la consola de Kermit la siguiente leyenda:

```
================================================
FS-BOOT First Stage Bootloader (c) 2006 PetaLogix
Project name: Xilinx-Spartan3E500-RevD-lite-edk101
Build date: May 29 2009 03:18:48  FS
Serial console: Uartlite
================================================
FS-BOOT: System initialisation completed.
FS-BOOT: Booting from FLASH. Press 's' for image download.
```

Como esta se trata de nuestro primer arranque, es necesario teclear 's' para poder cargar la imagen de U-Boot. El método que se utiliza para cargar la imagen es el seria y eso se hace sencillamente con el siguiente comando.

```
$ cat /tftpboot/u-boot.srec > /dev/ttyUSB1
```

El proceso de descarga dura un par de minutos o menos, en cuanto se termine la descarga de la imagen va aparecer una leyenda parecida a la siguiente:

```
FS-BOOT: Waiting for SREC image....
FS-BOOT: Image download successful.
FS-BOOT: Warning image location differ from default boot location. Image will not boot
FS-BOOT: Press 'n' to boot old image.
FS-BOOT: Use new image.
FS-BOOT: Booting image...
SDRAM :
        Enabling caches :
```

```
Icache:OK
Dcache:OK
U-Boot Start:0x8dfc0000
Malloc Start:0x8df80000
Board Info Start:0x8df7ffd0
Boot Parameters Start:0x8df6ffd0
FLASH: 16 MB
ETHERNET: MAC:00:0a:35:00:22:01

Hit any key to stop autoboot:  0
U-Boot>
```

A.5.2 U-Boot

U-Boot es un gestor de arranque de código abierto para sistemas embebidos orientados plataforma a través de múltiples arquitecturas incluyendo ARM, PPC, m68k, MIPS y Mic roBlaze. El proyecto U-Boot es mantenido por DENX Ingeniería del Software y está alojado en Sourceforge. U-Boot de arranque compatible con una amplia gama de herramientas e instalaciones específicas para los sistemas integrados.

El proceso de compilación de U-Boot se hace directamente desde el menú de configuración del PetaLinux y se hace únicamente con seleccionar que se compile U-Boot.

```
--------------------------------------------------------------------------
 ------------------------------ System Settings ----------------------------
|  Arrow keys navigate the menu.  <Enter> selects submenus --->.           |
|  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,  |
|  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.      |
|  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable      |
|  ------------------------------------------------------------------  |
```

```
| |               (CRAMFS) Root filesystem type                          | |
| |               [*] Copy final image to tftpboot                       | |
| |                           tftpboot directory: "/tftpboot"            | |
| |                     [*] Build u-boot                                 | |
| |                     Flash Partition Table  --->                      | |
| |_____________v(+)_____________________________________________________| |
|_____________________________________________________________________________|
|                     <Select>    < Exit >    < Help >                     |
|_____________________________________________________________________________|
```

U-Boot ofrece un pequeño conjunto de instrucciones en la Tabla A.3 se hace una breve descripción de cada comando. El proceso de condifuración de U-boot es moestra a continuación.

```
U-Boot> printenv
baudrate=115200
console=console=ttyUL0,115200
printenv=ethaddr
filesize=655
fileaddr=8C000000
autoload=no
ethaddr=00:0a:35:00:22:01
ipaddr=192.168.0.10
serverip=192.168.0.1
mtdparts=physmap-flash.0:256K(boot),128K(bootenv),128K(config),
         4M(image),11M(spare)
mtdkargs=mtdparts=physmap-flash.0:256K(boot),128K(bootenv),
         128K(config),4M(image),11M(spare)
clobstart=0x8c000000
netstart=0x8c002000
```

```
bootsize=0x40000
bootstart=0x89000000
bootenvsize=0x20000
bootenvstart=0x89040000
eraseenv=protect off $(bootenvstart) +$(bootenvsize);
         erase $(bootenvstart) +$(bootenvsize)
kernsize=0x400000
kernstart=0x89080000
make_cmdline=setenv netkargs macaddr=$(ethaddr);
             setenv bootargs $(mtdkargs) $(netkargs)
bootcmd=run make_cmdline; bootm $(kernstart)
bootdelay=4
load_kernel=tftp $(clobstart) image.ub
install_kernel=protect off $(kernstart) +$(kernsize);
               erase $(kernstart) +$(kernsize);
               cp.b $(fileaddr) $(kernstart) $(filesize)
update_kernel=run load_kernel install_kernel
load_uboot=tftp $(clobstart) u-boot-s.bin
install_uboot=protect off $(bootstart) +$(bootsize);
              erase $(bootstart) +$(bootsize);
              cp.b $(clobstart) $(bootstart) $(filesize)
update_uboot=run load_uboot install_uboot
netboot=run make_cmdline;tftp ${netstart} image.ub; bootm

Environment size: 1276/131068 bytes
```

Comando	Descripción
?	alias for 'help'
autoscr	run script from memory
base	print or set address offset
bdinfo	print Board Info structure
boot	boot default, i.e., run 'bootcmd'
bootd	boot default, i.e., run 'bootcmd'
bootm	boot application image from memory
bootp	boot image via network using BootP/TFTP protocol
cmp	memory compare
coninfo	print console devices and information
cp	memory copy
crc32	checksum calculation
dcache	enable or disable data cache
dhcp	invoke DHCP client to obtain IP/boot params
echo	echo args to console
erase	erase FLASH memory
flinfo	print FLASH memory information
go	start application at address 'addr'
help	print online help
icache	enable or disable instruction cache
nm	memory modify (constant address)
ping	send ICMP ECHO_REQUEST to network host
printenv	print environment variables
protect	enable or disable FLASH write protection
rarpboot	boot image via network using RARP/TFTP protocol
reset	Perform RESET of the CPU
run	run commands in an environment variable
saveenv	save environment variables to persistent storage
setenv	set environment variables
sleep	delay execution for some time
tftpboot	boot image via network using TFTP protocol
version	print monitor version

Tabla A.3: Comandos disponile en el U-Boot

A.5.2.1. Descargando archivos

En muchas ocasiones es necesario es necesario descargar archivos a U-Boot, ya sea a través de conexión serial o de red. Varios de los scripts du U-Boot son de ayuda para realizar esta tarea. Típicamente todos los comandos necesitan una dirección de memoria en la que los datos del archivo va a ser almacenados, esta dirección depende de a cual dispositivos queremos asignar ese archivo.

La memoria flash es una forma de memoria no volátil que se puede borrar y reprogramar. Normalmente, el flash se utiliza para almacenar la imagen U-Boot, las variables de ambiente de U-Boot y la imagen del núcleo. Con el fin de actualizar el contenido de Flash, el diseñador debe saber cómo reprogramar el flash. U-Boot ofrece varios comandos de programación, y la protección de borrado de la memoria flash.

En el siguiente ejemplo se muestra como actualizar la imagen de U-Boot y se esta sera guardada de una vez a la memoria FLASH. El comando `run` es utilizado para ejecutar las variables de ambiente que esta definidas en U-Boot.

```
U-Boot> tftp 0x8c000000 u-boot-s.bin
TFTP from server 192.168.0.1; our IP address is 192.168.0.10
Filename 'u-boot-s.bin'.
Load address: 0x8c000000
Loading: ####################
done
Bytes transferred = 104116 (196b4 hex)
.. done
U-Boot> protect off 0x89000000 + 0x40000
Un-Protected 2 sectors

.. done
U-Boot> erase  0x89000000 + 0x20000
Erased 2 sectors
```

```
U-Boot> cp.b 0x8c000000 0x89000000 0x196b4
Copy to Flash... done
```

En este caso utilizamos update_uboot que es equivalente a ejecutar los comandos anteriores:

```
U-Boot> run update_uboot
TFTP from server 192.168.0.1; our IP address is 192.168.0.10
Filename 'u-boot-s.bin'.
Load address: 0x8c000000
Loading: ####################
done
Bytes transferred = 104116 (196b4 hex)
.. done
Un-Protected 2 sectors

.. done
Erased 2 sectors
Copy to Flash... done

U-Boot> run update_kernel
TFTP from server 192.168.0.1; our IP address is 192.168.0.10
Filename 'image.ub'.
Load address: 0x8c000000
Loading: ##################################################################
 ################################################################
 ################################################################
 ################################################################
 ################################################################
 ################################################################
 ################################################################
```

```
 #################################################################
 #################################################################
 #################################################################
 #################################################################
 #################################################################
 ########################################
done
Bytes transferred = 4072581 (3e2485 hex)
.............................. done
Un-Protected 32 sectors


.............................. done
Erased 32 sectors
Copy to Flash... done
```

Una vez que la inicialización es completado con éxito, la plataforma de destino puede ser reiniciada, y deberá arrancar con todos los parámetros configurados. La salida debe ser parecida a lo siguiente.

```
## Booting image at 89080000 ...
   Image Name:   PetaLinux Kernel 2.6-MMU
   Image Type:   Microblaze Linux Kernel Image (uncompressed)
   Data Size:    4198544 Bytes =  4 MB
   Load Address: 8c000000
   Entry Point:  8c000000
   Verifying Checksum ... OK
```

B

Diagramas de tiempos para los casos de prueba

En la tabla se muestra los casos que fueron probados variando el valor de las señales de entrada al MCA.

Caso	Operación	Instrucción en las reglas	Acceso permitido
1	Escritura	Si	Si
2	Escritura	Si	No
3	Escritura	No	No importa
4	Lectura	No	No
5	Lectura	Si	Si
6	Lectura	No	No importa

En el capítulo 5 fueron explicados dos casos de pruebas, el caso 1 y el caso 5. A continuación se describen los casos faltantes.

Para la explicación de cada diagrama de tiempo se colocaron etiquetas en las imágenes las cuales son relacionada durante cada descripción.

B.1 Caso 2

El caso 2 se trata de una operación denegada por el MCA, la Figura B.1 muestra el diagrama de tiempos para esta situación.

1. La señales que se encuentran agrupadas en la dentro de la `etiqueta 1` son las enviadas por el maestro que realiza la solicitud hacia el PLB, son las señales que serán analizadas por el MCA.

2. Cuando la señal `PLB_PAValid` (`etiqueta 2`) es activada el MCA inicia su proceso y a partir de ese momento toma en cuenta las señales en la `etiqueta 1`) para iniciar el análisis.

3. La señal `PLB_RnW` (`etiqueta 3`) tiene valor de '0', esto significa que la operación solicitada se trata de una escritura al esclavo.

4. Con `PLB_PVAalid` activado el módulo de control de acceso inicia su tarea creando una instrucción que servirá de entrada para ser comparada con las reglas almacenadas. La señal de `instruccin` (`etiqueta 4`) fue construida a partir de las señales en la `etiqueta 1`.

5. La señal `state` muestra el cambio de estado de **ESPERA** a **CHECAR** (`etiqueta 5`) después de un ciclo en el que el MCA-CAM-n recibió `plb_pavalid`='1' e inicio su operación. Durante el estado **CHECAR** la `etiqueta_in` se envía a la memoria, ésta la analiza y entrega un resultado en el mismo ciclo de reloj.

6. La MEF cambia de estado al **FINAL** (`etiqueta 6`) porque en el ciclo anterior obtuvo la información de la CAM y ahora conoce dependiendo de los datos devueltos por la CAM ejecutará la decisión de la regla.

7. `golpe` (`etiqueta 8`) es la señal que devuelve la CAM para indicar que hubo una coincidencia entre la `etiqueta_in` y la regla almacenada. En este caso `golpe` es igual a '1', es decir, se encontró una regla.

8. la señal `accion_al` (`etiqueta 9`) a pesar de que `golpe` esta activada `accion_al` se mantiene en cero lo que significa que la regla encontrada no permite el acceso a dispositivo.

9. Las señales englobas en la `etiqueta 10` son las salidas que genera el MCA una vez que se tomó una decisión de que la operación. Como la operación es negada entonces el módulo entrega todas las señales en cero.
10. La señal `hecho` se activa para indicar que el módulo ha tomado un decisión y que todas las señales han sido establecidas.
11. El módulo se debe de encargar de finalizar con el protocolo de transferecia, entonces general una señal `sl_addrack_out` (`etiqueta 12`) para simular que el dispositivo respondió. Esta señal concluiría el ciclo de dirección y será recibida por el PLB como si en realidad fuera el esclavo quien se la enviará y no el MCA-CAM-n.
12. Para finalizar el ciclo de datos el MCA genera las señales reconocimiento de escritura y de completado de escritura (`etiqueta 13`)
13. La señal `hecho` cambia su valor a '0', lo que indica que todo el protocolo del PLB ha sido concluido y que ahora se puede atender otra solicitud.

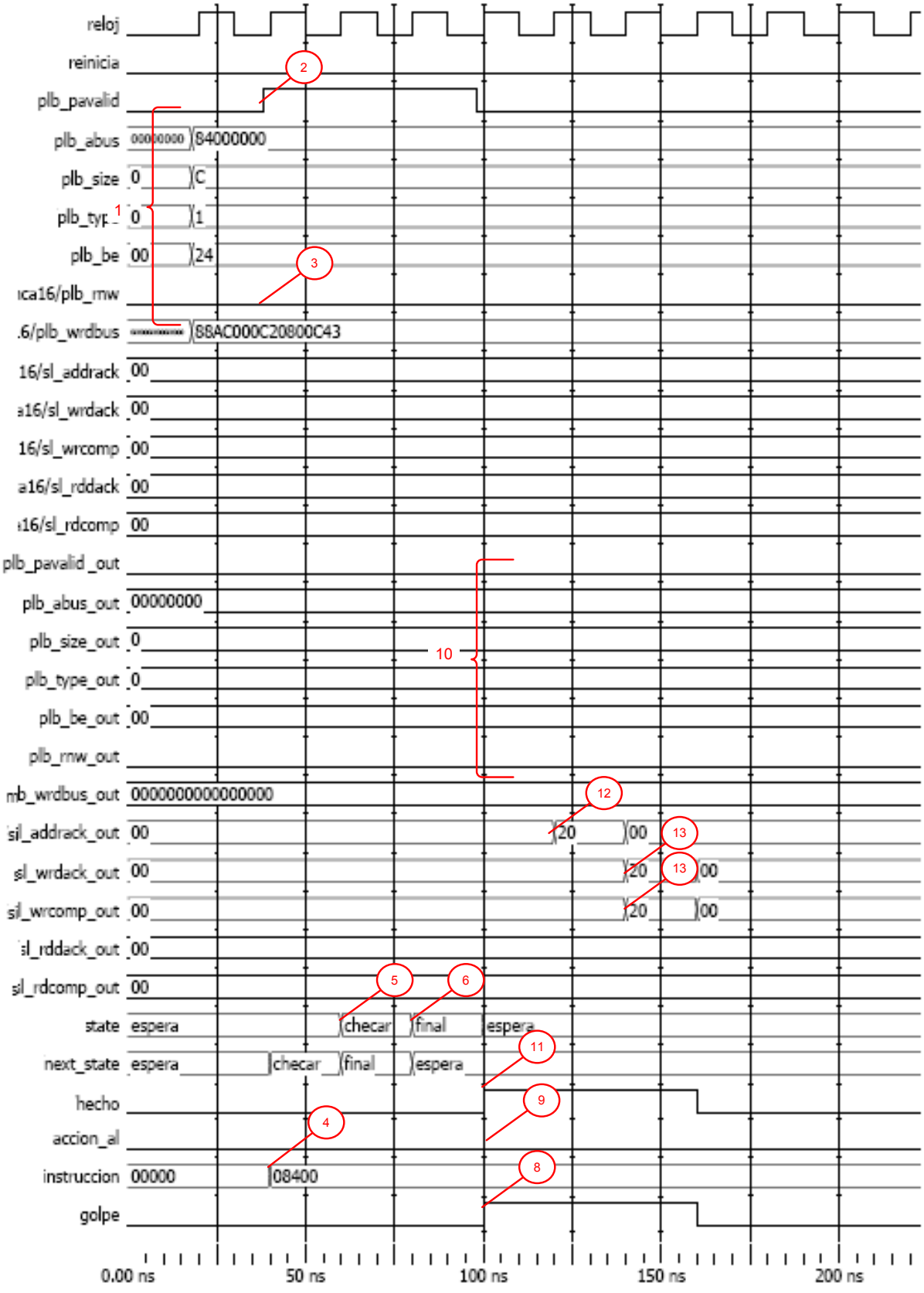


Figura B.1: Diagrama de tiempo para el caso 2 escritura negada

B.2 Caso 3

A continuación se da la interpretación del diagrama que se muestra en la Figura B.2, para esta explicación se colocaron marcas en la figura las cuales se relacionan con los números aquí mostrados:

1. La señales que se encuentran agrupadas en la dentro de la `etiqueta 1` son las enviadas por el maestro que realiza la solicitud hacia el PLB, son las señales que serán analizadas por el MCA.

2. Cuando la señal `PLB_PAValid` (`etiqueta 2`) es activada el MCA inicia su proceso y a partir de ese momento toma en cuenta las señales en la `etiqueta 1`) para iniciar el análisis.

3. La señal `PLB_RnW` tiene valor de '0', esto significa que la operación solicitada se trata de una escritura al esclavo.

4. Con `PLB_PVAalid` activado el módulo de control de acceso inicia su tarea creando una instrucción que servirá de entrada para ser comparada con las reglas almacenadas. La señal de `instruccin` (`etiqueta 3`) fue construida a partir de las señales en la `etiqueta 1`.

5. La señal `state` muestra el cambio de estado de **ESPERA** a **CHECAR** (`etiqueta 4`) después de un ciclo en el que el MCA-CAM-n recibió `PLB_PAValid`='1' e inicio su operación. Durante el estado **CHECAR** el módulo analiza en un sólo ciclo de reloj todas la reglas almacenadas.

6. Cuando la MEF cambia de estado al **FINAL** (`etiqueta 5`) el MCA-CAM debe de contar con la información necesaria para tomar un decisión.

7. `golpe` es la señal que devuelve la CAM para indicar que hubo una coincidencia entre la `etiqueta_in` y la regla almacenada. En este caso `golpe` es igual a '0', es decir, se no encontró una regla.

8. la señal `accion_al` no se modifica.

9. Las señales englobas en la `etiqueta 10` son las salidas que genera el MCA una vez que se tomó una decisión de que la operación. Como la operación por defecto es negar la operación entonces el módulo entrega todas las señales en cero.

10. La señal `hecho` se activa para indicar que el módulo ha tomado un decisión y que todas las señales han sido establecidas.
11. El módulo se debe de encargar de finalizar con el protocolo de transferecia, entonces general una señal `sl_addrack_out` (`etiqueta 8`) para simular que el dispositivo respondió. Esta señal concluiría el ciclo de dirección y será recibida por el PLB como si en realidad fuera el esclavo quien se la enviará y no el MCA-CAM-n.
12. Para finalizar el ciclo de datos el MCA genera las señales reconocimiento de escritura y de completado de escritura (`etiqueta 9`)
13. La señal `hecho` cambia su valor a '0', lo que indica que todo el protocolo del PLB ha sido concluido y que ahora se puede atender otra solicitud.

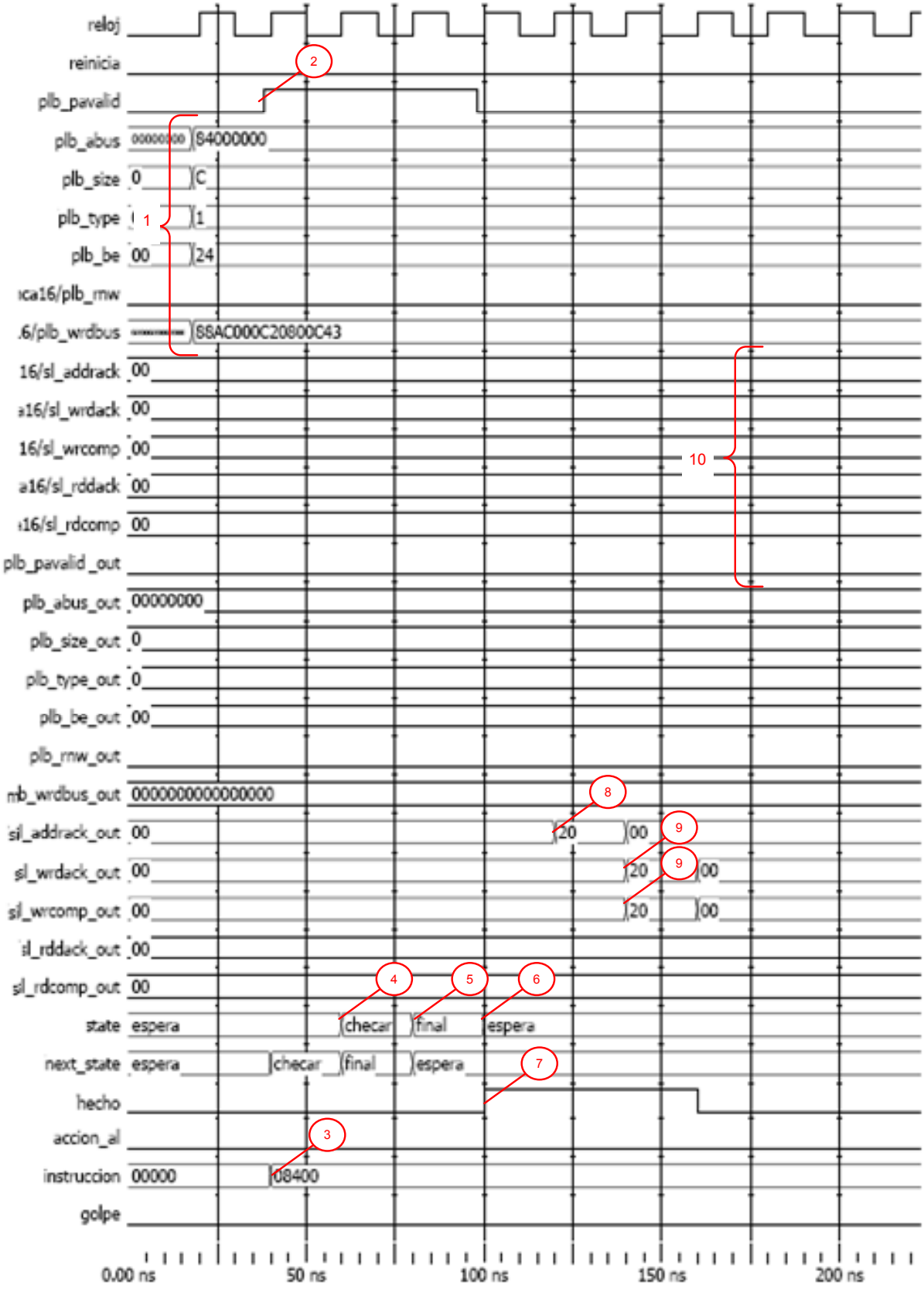


Figura B.2: Diagrama de señales para caso 3 escritura no está la regla en memoria

B.3 Caso 4

A continuación se da la interpretación del diagrama que se muestra en la Figura B.3, para esta explicación se colocaron marcas en la figura las cuales se relacionan con los números aquí mostrados:

1. La señales que se encuentran agrupadas dentro de la `etiqueta 1` son las enviadas por el maestro que realiza la solicitud hacia el PLB, son las señales que serán analizadas por el MCA.

2. Cuando la señal `PLB_PAValid` (`etiqueta 2`) es activada el MCA inicia su proceso y a partir de ese momento toma en cuenta las señales de la `etiqueta 1`) para iniciar el análisis.

3. La señal `PLB_RnW` (`etiqueta 3`) tiene valor de '1', esto significa que la operación solicitada se trata de una lectura al esclavo.

4. Con `PLB_PVAalid` activado el módulo de control de acceso inicia su tarea creando una instrucción que servirá de entrada para ser comparada con las reglas almacenadas. La señal de `instruccin` (`etiqueta 4`) fue construida a partir de las señales de la `etiqueta 1`.

5. La señal `state` muestra el cambio de estado de **ESPERA** a **CHECAR** (`etiqueta 5`) después de un ciclo que el MCA-CAM-n recibió `plb_pavalid`='1' e inicio su operación. Durante el estado **CHECAR** la `etiqueta_in` se envía a la memoria, ésta la analiza y entrega un resultado en el mismo ciclo de reloj.

6. La MEF cambia de estado al **FINAL** (`etiqueta 6`) porque en el ciclo anterior obtuvo la información de la CAM y ahora conoce dependiendo de los datos devueltos por la CAM ejecutará la decisión de la regla.

7. `golpe` (`etiqueta 8`) es la señal que devuelve la CAM para indicar que hubo una coincidencia entre la `etiqueta_in` y la regla almacenada. En este caso `golpe` es igual a '1', es decir, se encontró una regla.

8. La señal `accion_al` (`etiqueta 9`) a pesar de que `golpe` esta activada se mantiene en cero lo que significa que la regla encontrada no permite leer el dispositivo.

9. Las señales englobas en la `etiqueta 10` son las salidas que genera el MCA una vez que se tomó una decisión de que la operación. Como la operación es negada entonces el módulo entrega todas las señales en cero.
10. La señal `hecho` se activa para indicar que el módulo ha tomado una decisión y que todas las señales han sido establecidas.
11. El módulo se debe de encargar de finalizar con el protocolo de transferecia, entonces general una señal `sl_addrack_out` (`etiqueta 12`) para simular que el dispositivo respondió. Esta señal concluiría el ciclo de dirección y será recibida por el PLB como si en realidad fuera el esclavo quien se la enviará y no el MCA-CAM-n.
12. Para finalizar el ciclo de datos el MCA genera las señales reconocimiento de escritura y de completado de lectura (`etiqueta 13`)
13. La señal `hecho` cambia su valor a '0', lo que indica que todo el protocolo del PLB ha sido concluido y que ahora se puede atender otra solicitud.

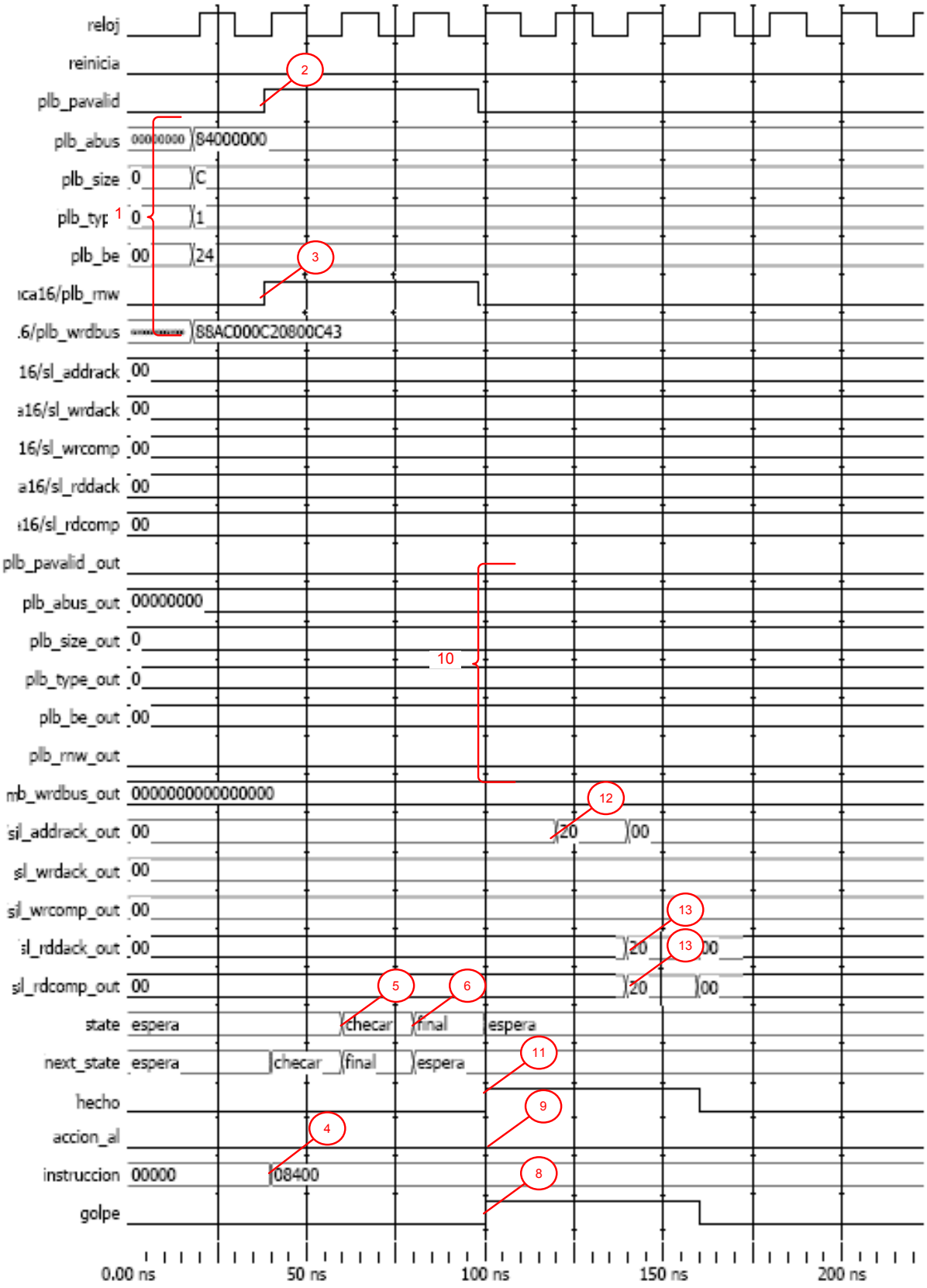


Figura B.3: Diagrama de tiempo de las señales para el caso 4 lectura denegada

B.4 Caso 6

A continuación se da la interpretación del diagrama que se muestra en la Figura B.4.

1. La señales que se encuentran agrupadas en la dentro de la `etiqueta 1` son las enviadas por el maestro que realiza la solicitud hacia el PLB, son las señales que serán analizadas por el MCA.

2. Cuando la señal `PLB_PAValid` (`etiqueta 2`) es activada el MCA inicia su proceso y a partir de ese momento toma en cuenta las señales en la `etiqueta 1`) para iniciar el análisis.
3. La señal `PLB_RnW` tiene valor de '1', esto significa que la operación solicitada se trata de una lectura al esclavo.
4. Con `PLB_PVAalid` activado el módulo de control de acceso inicia su tarea creando una instrucción que servirá de entrada para ser comparada con las reglas almacenadas. La señal de `instruccin` (`etiqueta 3`) fue construida a partir de las señales en la `etiqueta 1`.
5. La señal `state` muestra el cambio de estado de **ESPERA** a **CHECAR** (`etiqueta 4`) después de un ciclo en el que el MCA-CAM-n recibió `PLB_PAValid`='1' e inicio su operación. Durante el estado **CHECAR** el módulo analiza en un sólo ciclo de reloj todas la reglas almacenadas.
6. Cuando la MEF cambia de estado al **FINAL** (`etiqueta 5`) el MCA-CAM debe de contar con la información necesaria para tomar un decisión.
7. `golpe` es la señal que devuelve la CAM para indicar que hubo una coincidencia entre la `etiqueta_in` y la regla almacenada. En este caso `golpe` es igual a '0', es decir, se no encontró una regla.
8. la señal `accion_al` no se modifica.
9. Las señales englobas en la `etiqueta 10` son las salidas que genera el MCA una vez que se tomó una decisión de que la operación. Como la operación por defecto es negar la operación entonces el módulo entrega todas las señales en cero.
10. La señal `hecho` se activa para indicar que el módulo ha tomado un decisión y que todas las señales han sido establecidas.

11. El módulo se debe de encargar de finalizar con el protocolo de transferecia, entonces general una señal `sl_addrack_out` (`etiqueta 8`) para simular que el dispositivo respondió. Esta señal concluiría el ciclo de dirección y será recibida por el PLB como si en realidad fuera el esclavo quien se la enviará y no el MCA-CAM-n.
12. Para finalizar el ciclo de datos el MCA genera las señales reconocimiento de escritura y de completado de lectura (`etiqueta 9`)
13. La señal `hecho` cambia su valor a '0', lo que indica que todo el protocolo del PLB ha sido concluido y que ahora se puede atender otra solicitud.

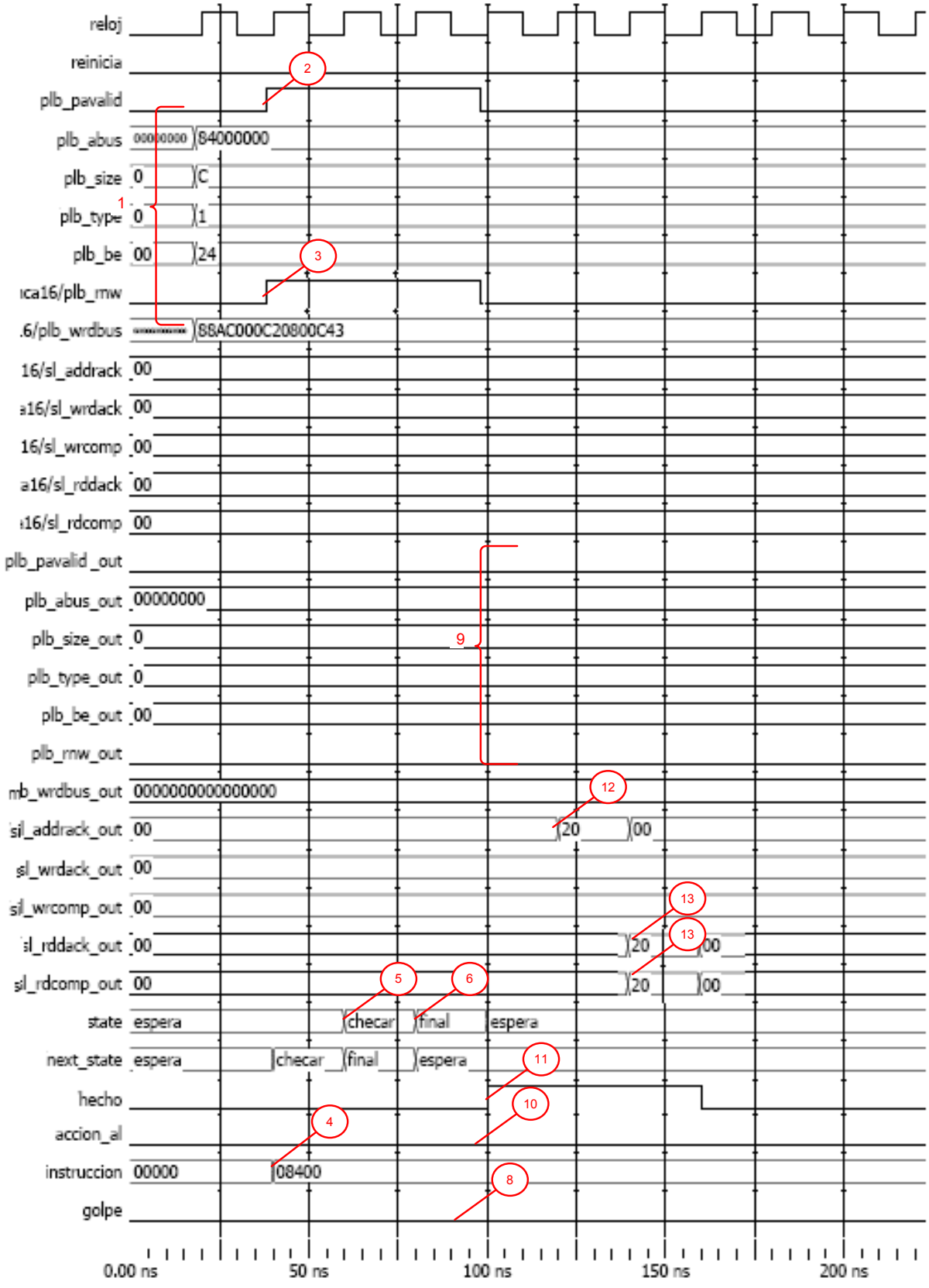


Figura B.4: Diagrama de tiempo para el caso 6 lectura con la regla no está en memoria

Bibliografía

[1] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. Intel virtualization technology for directed I/O. *Intel Technology Journal*, 10(03):179–192, August 2006.

[2] AMD. Advanced Micro Devices, AMD, 2009. http://www.intel.com.

[3] Jim Attridge. An overview of hardware security modules. *SANS Institute, InfoSec Reading Room*, vol. 1, Enero 2002.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[6] Elliot Bell and Leon LaPadula. Secure computer systems: Mathematical foundations. *MITRE Co., technical report MITRE-2547, Vol. 1*, 1973.

[7] Biba. Integrity considerations for secure computer systems. *MITRE Co., technical report ESD-TR 76-372*, 1977.

[8] Michael K. Bond. *Understanding Security APIs*. M. eng. thesis, University of Cambridge, Cambridge, UK, January 2004.

[9] Citrix Systems. About Xen.org, 2009. http://www.xen.org/about/.

[10] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. *Intel Technology Journal*, vol. 10(03):184–193, Mayo 1987.

[11] International Business Machines Corporation. *Processor Local Bus (128-bit)*. IBM systems and techonology group, New York, USA, primera edition, 2007.

[12] J.G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, and S.W. Smith. Building the IBM 4758 secure coprocessor. *Computer*, 34(10):57–66, Oct 2001.

[13] ECMA International. Standard ECMA-335: Common language infrastructure (CLI), 2006. http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[14] J.P. Anderson et al. Computer security technology planning study. Technical Report ESD-TR-73-51, Air Force System Command, USAF, 1972.

[15] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206, New York, NY, USA, 2003. ACM.

[16] B. D. Gold, R. R. Linde, and P. F. Cudney. KVM/370 in retrospect. *Security and Privacy, IEEE Symposium on*, 0:13, 1984.

[17] IBM. System z PR/SM, 2009. http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=/eicaz/eicazzlpar.htm.

[18] IBM Research . sHype - Secure Hypervisor, 2009. http://www.research.ibm.com/secure_systems_department/projects/hypervisor/.

[19] Intel. Intel corporation, 2009. http://www.intel.com.

[20] P.A. Karger and D.R. Safford. I/O for virtual machine monitors: Security and performance issues. *Security & Privacy, IEEE*, 6(5):16–23, Sept.-Oct. 2008.

[21] P.A. Karger, M.E. Zurko, D.W. Bonin, A.H. Mason, and C.E. Kahn. A retrospective on the VAX VMM security kernel. *Software Engineering, IEEE Transactions on*, 17(11):1147 – 1165, Nov. 1991.

[22] Roman Lysecky and Frank Vahid. A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 18–23, Washington, DC, USA, 2005. IEEE Computer Society.

[23] Microsoft. Microsoft Hyper-V server 2008, 2008. http://www.microsoft.com/servers/hyper-v-server/default.mspx/.

[24] Microsoft. Microsoft Virtual Server 2005 R2, 2008. http://www.microsoft.com/windowsserversystem/virtualserver/default.aspx.

[25] Microsoft. Microsoft Virtual PC, 2009. http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx.

[26] Wendell Odom. *CCIE # 1624, CCNA Exam 640-607 Certification Guide*. Cisco Press, 201 West 103rd Street Indianapolis IN 46290 USA, first edition, 2002.

[27] Oracle. Oracle VM, 2009. http://www.oracle.com/technologies/virtualization/index.html.

[28] Parallels. Parallels desktop, 2009. http://www.parallels.com/products/desktop/.

[29] Parallels. Parallels Workstation 2.2 for Windows & Linux - Virtual Machine for Linux, 2009. http://www.parallels.com/products/workstation/.

[30] Ronald Perez, Leendert van Doorn, and Reiner Sailer. Virtualization and Hardware-Based Security. *IEEE Security and Privacy*, 6(5):24–31, 2008.

[31] Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ramon Caceres, Ronald Perez, Stefan Berger, John Linwood Griffin, and Leendert van Doorn. Building a MAC-Based security architecture

for the Xen open-source hypervisor. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 276–285, Washington, DC, USA, 2005. IEEE Computer Society.

[32] Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ramon Caceres, Ronald Perez, Stefan Berger, John Linwood Griffin, and Leendert van Doorn. Building a MAC-Based security architecture for the Xen open-source hypervisor. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 276–285, Washington, DC, USA, 2005. IEEE Computer Society.

[33] Pierangela Samarati and Sabrina De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *FOSAD '00: Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design*, pages 137–196, London, UK, 2001. Springer-Verlag.

[34] Sun Microsystems, Inc. Learn about java technology, 2009. http://java.com/en/about/.

[35] Andrew S. Tanenbaum. *Sistemas Operativos Modernos*. Pearson Educación, México, D.F., segunda edition, 2003.

[36] VMware, Inc. Vmware esx build the foundation of a responsive data center, 2009. http://www.vmware.com/products/vi/esx/.

[37] VMware, Inc. Vmware fusion, 2009. http://www.vmware.com/products/fusion/.

[38] VMware, Inc. Vmware server: Get started with virtualization risk-free, 2009. http://www.vmware.com/products/server/.

[39] VMware, Inc. Vmware workstation: Run windows, linux, and more side by side on the same computer, 2009. http://www.vmware.com/products/ws/.

[40] WineHQ. Winehq - run windows applications on linux, bsd and mac os x, 2009.

[41] Inc Xilinx. Plbv46 slave single (v1.00.a) product specification. Technical report, Xilinx, Inc, Disponible en la documentación del EDK, 2007.

[42] Inc Xilinx. Processor local bus plb v4.6 (v1.00.a) product specification. Technical report, Xilinx, Inc, Disponible en la documentación del EDK, 2007.

[43] Xilinx, Inc. *MicroBlaze Processor Reference Guide*, 2008.