INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

Diseño e implementación del bus local para el procesador Lagarto

T E S I S

Que para obtener el grado de:

Maestría en Ciencias en Ingeniería de Cómputo

P R E S E N T A :

Ing. Job Isaias Quiroz Mercado

Directores de Tesis:

Dr. Marco Antonio Ramírez Salinas

Dr. Luis Alfonso Villa Vargas

DICIEMBRE 2016

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México siendo las 16:00 horas del día 22 del mes de noviembre de 2016 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Diseño e implementación del bus local para el procesador Lagarto"

Presentada por el alumno(a):

Quiroz	**Mercado**	**Job Isaías**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	5	0	1	6	2

aspirante de: **MAESTRÍA EN CIENCIAS EN INGENIERÍA DE CÓMPUTO**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Marco Antonio Ramírez Salinas

Dr. Luis Alfonso Villa Vargas

Dr. Herón Molina Lozano

Dr. José Luis Oropeza Rodríguez

M. en C. Osvaldo Espinosa Sosa

Dr. Héctor Báez Medina

PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION

Dr. Marco Antonio Ramírez Salinas

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 7 del mes Diciembre del año 2016, el (la) que suscribe Job Isaias Quiroz Mercado alumno del Programa de Maestría en Ciencias en Ingeniería de Cómputo con número de registro A150162, adscrito al Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del Dr. Marco Antonio Ramírez Salinas y Dr. Luis Alfonso Villa Vargas y cede los derechos del trabajo intitulado Diseño e Implementación del bus local para el procesador Lagarto, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección jobquiroz@hotmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Job Isaias Quiroz Mercado

Nombre y firma

RESUMEN

El continuo desarrollo de la industria electrónica ha permitido un incremento en la densidad de los circuitos integrados, esto ha requerido el uso de nuevas metodologías de diseño para poder hacer un uso eficiente de esta tecnología. La metodología de diseño System On Chip permite la integración de bloques prediseñados, conocidos como IP cores, dentro de un mismo encapsulado. Estos componentes trabajan en conjunto para resolver una tarea específica. La interacción entre los IP cores se realiza sobre una arquitectura de comunicación *on-chip*, la cual asegura que la información se entrega de manera eficiente y confiable.

En el presente trabajo se describe el diseño e implementación de una arquitectura de comunicación *on-chip* mediante el uso de buses compartidos, la cual permite la comunicación del procesador Lagarto I con bloques de memoria y dispositivos de entrada/salida.

El bus diseñado tiene una topología de bus jerárquico el cual incluye un bus de alta velocidad para la comunicación del procesador con bloques de memoria y un bus de baja velocidad para la comunicación con los controladores de periféricos. El bus es compatible con el estándar de comunicación *on-chip* Wishbone. El diseño del mismo ha sido descrito en lenguaje Verilog HDL e implementado en un dispositivo FPGA de Altera.

ABSTRACT

The constant development of the electronic industry has allowed an increased in the density of the integrated circuits; this has required the use of new methodologies in order to use efficiently this technology. The System On Chip design methodology allows the integration of predesigned blocks, known as IP cores, within a single chip. These components work together in order to solve a specific task. The IP core's interaction is supported by the on-chip communication architecture, which allows the information to be efficiently and reliably transmitted.

This works presents the design and implementation of a shared bus on-chip communication architecture, which allows the communication between Lagarto I processor and peripherals controllers.

The designed bus has a hierarchical bus topology, which includes a high speed bus for the communication between the processor and memory blocks and also a low speed bus for the communication with the peripherals. The bus is based on the Wishbone on-chip communication standar. The design has been developed in Verilog HDL and has been implemented into an Altera FPGA device.

AGRADECIMIENTOS

En primer lugar me gustaría agradecer a mi familia, pues sin ellos no habría podido alcanzar muchas de las metas que me he propuesto a lo largo de la vida. Agradezco especialmente a mis padres, por los sacrificios que han realizado para tratar de darme siempre lo mejor.

Agradezco al Instituto Politécnico Nacional por ser como un segundo hogar desde hace ya más de 10 años. Agradezco el apoyo de mis asesores Dr. Marco Antonio Ramírez Salidas y Dr. Luis Alfonso Villa Vargas, por el tiempo que me dedicaron para la realización de este trabajo, a pesar de que en ocasiones sus agendas estaban muy saturadas.

Agradezco a los profesores del Centro de Investigación en Computación con quien tuve la oportunidad de colaborar, en especial a los profesores pertenecientes al laboratorio de investigación MICROSE. También me gustaría agradecer a los compañeros del mismo centro por su constante apoyo y por haber compartido conmigo esta etapa de la vida.

Por último me gustaría agradecer a Jeny, Slash y Princesa, por acompañarme en muchas noches de desvelo y por compartir conmigo su alegría cada vez que llego a casa.

Job Isaias Quiroz Mercado

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

CPU: Acrónimo de *Central Processing Unit*, en español Unidad Central de procesamiento. Es el hardware dentro de una computadora donde se interpretan y ejecutan las instrucciones de programa.

FPGA: Acrónimo de *Field Programmable Gate Array*. Es un circuito integrado programable diseñado para ser configurado generalmente mediante un lenguaje de descripción de hardware.

FSM: *Finite State Machine*. Máquina de estados finitos. Modelo matemático utilizado para el diseño de programas de computadoras y circuitos de lógica secuencial.

Handshaking. Proceso automático de negociación en el que dos entidades establecen los parámetros de un canal de comunicación previo a realizarse una transferencia de información.

IP core. Bloque prediseñado de hardware que puede ser reutilizado en diseños SoC, incluye bloques periféricos, de comunicación y de procesamiento.

Kernel (Sistema Operativo). Es un programa computacional que constituye la parte central del sistema operativo. Es el primer programa que ejecuta el procesador y donde se gestiona la inicialización (*startup*) y demás procesos de sistema.

LE. *Logic element*. En español Elemento lógico, es el bloque de interconexión más básico en un FPGA (de Altera, para dispositivos Xilinx se conoce como *Logic Cell*). De manera general está compuesto por una sección de lógica combinatoria (multiplexores y tablas de verdad) y una sección de lógica secuencial (registros).

LUT. *Lookup table*. Es un conjunto de circuitos de lógica combinatoria que permite reemplazar el procesamiento de una entrada, por una operación de indexación.

MIPS. Acrónimo de *Microprocessor without Interlocked Pipeline Stages*. Hace referencia a una arquitectura de procesador tipo RISC, desarrollada por MIPS Technologies®.

MMU. *Memory Management Unit*. En español Unidad de gestión de memoria. Es un bloque de hardware dentro del CPU donde se controlan las operaciones de lectura y escritura de las memorias de datos e instrucciones.

Polling. Es una técnica para averiguar el estado de un dispositivo de entrada/salida que consiste en hacer lecturas repetitivas hasta que se detecte un cambio de estado específico.

Procesador Escalar: Arquitectura capaz de ejecutar una instrucción por ciclo de reloj

RISC: Acrónimo de *Reduced Instruction Set Computer*. En español Conjunto reducido de instrucciones de computadora.

SoC: Acrónimo de *System on Chip*, en español: Sistema en un Chip. Metodología de diseño de un sistema computacional dentro de un mismo encapsulado.

Trade-off. Situación en la cual se debe sacrificar cierta cualidad a cambio de la mejora en alguna otra.

TLB. *Translation Lookaside Buffer*. Memoria cache administrada por la MMU que contiene partes de la tabla de paginación.

CAPÍTULO 1 INTRODUCCIÓN

1.1 Planteamiento del problema

El laboratorio de Microtecnología y Sistemas Embebidos del Centro de Investigación en Computación ha trabajado durante los últimos 5 años en el desarrollo del proyecto Lagarto, cuyo principal objetivo es el desarrollo de propiedad intelectual en materia de arquitectura de computadoras y sistemas operativos que sea de utilidad para la academia y la industria.

Actualmente el primer modelo de procesador, Lagarto I, ha concluido su primera etapa de desarrollo. Lagarto I es un procesador escalar segmentado de 32 bits, cuyo diseño se basa en la arquitectura MIPS-32[1], el procesador es sintetizable en dispositivos FPGA de Altera. Una segunda versión del procesador, Lagarto II, capaz de ejecutar dos instrucciones por ciclo de reloj actualmente sigue en desarrollo. Paralelamente al desarrollo de las dos versiones del procesador, se ha desarrollado la unidad de gestión de memoria (MMU) y la unidad de manejo de excepciones (Exception Handler).

Uno de los objetivos actuales es la integración de Lagarto I en un *System On Chip*, de tal forma que el procesador pueda comunicarse con otros dispositivos de hardware y comenzar a crear con esto un ambiente propicio para la implementación de un Sistema Operativo. Este trabajo pretende dar paso al desarrollo de dicho ambiente, al permitir la comunicación eficiente entre el procesador y otros componentes de hardware tales como temporizadores, puertos de entrada salida y controladores de memoria.

1.2 Justificación

El desarrollo del presente trabajo permitirá la creación de un System on Chip con el procesador Lagarto I y diversos periféricos, cuya interacción será soportada por una arquitectura de comunicación compatible con el estándar Wishbone.

Esta arquitectura de comunicación incrementará la funcionalidad del procesador Lagarto I, al permitirle hacer uso de dispositivos periféricos y de memoria, lo cual es un requisito para la implementación del Sistema Operativo uno de los objetivo del proyecto Lagarto.

Este trabajo permitirá el desarrollo de nuevas aplicaciones con el procesador Lagarto en las cuales se haga uso de dispositivos de entrada salida. También permitirá el diseño de IP cores compatibles con el estándar Wishbone, los cuales podrán ser añadidos al System On Chip.

1.3 Solución propuesta.

La arquitectura de comunicación propuesta consiste de un bus jerárquico de 32 bits diseñado bajo el estándar de comunicación *on-chip* Wishbone. El bus consiste de un bus de alta velocidad que conecta al procesador con bloques de memoria y de un bus de baja velocidad para la comunicación con dispositivos de entrada salida. En la Figura 1, se observa la estructura general del sistema.

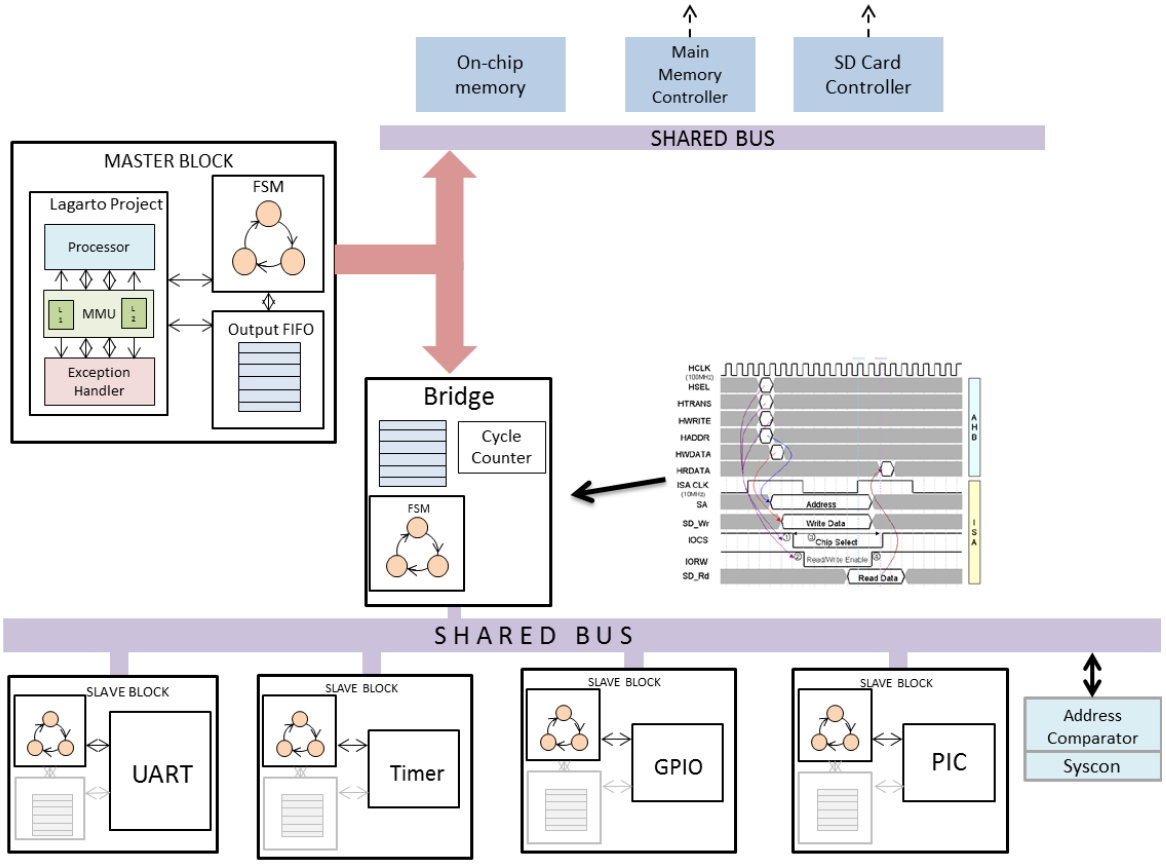


Figura 1. Estructura general del sistema

Tanto el procesador como los dispositivos de entrada salida requieren de un bloque interfaz el cual traduzca las señales del procesador en señales compatibles con el estándar Wishbone, el cual se basa en una comunicación Maestro-Esclavo.

Los dos buses de la arquitectura son conectados entre sí por medio de un puente (*bridge)* el cual permite la comunicación de bloques que trabajan a frecuencias de operación distintas.

1.4 Objetivos

Objetivo General

Diseñar e implementar un Bus Local de Intercomunicación que sirva como medio para la comunicación de datos entre el procesador y los bloques hardware que integran el proyecto Lagarto, según las especificaciones del estándar de comunicación *on-chip* Wishbone.

Objetivos Específicos

- Definición de las características del bus con base a las posibilidades del estándar Wishbone (ancho de palabra, velocidad, etc.)
- Diseñar mediante Verilog HDL una arquitectura de comunicación de bus compartido jerárquico acorde a la especificación Wishbone.
- Implementar en un dispositivo FPGA el bus de comunicación de alta velocidad que permite la comunicación entre el procesador y bloques de memoria.
- Implementar en un dispositivo FPGA el bus de comunicación de baja velocidad que permita la comunicación entre el procesador y periféricos (UART, temporizador, GPIO, etc.)

1.5 Organización del documento.

En el capítulo 1 se presenta la motivación y los objetivos del proyecto, además de una descripción general del problema y la solución propuesta. En el capítulo 2 se aborda el

marco teórico sobre el que se basa el trabajo, específicamente se trata sobre las características de las arquitecturas de comunicación mediante buses compartidos, además de realizar una breve revisión del Estado del Arte, destacando los estándares de comunicación on-chip más utilizados.

En el capítulo 3 se describen las características del procesador Lagarto I, así como de la unidad de gestión de memoria y el procesador de sistema. También se explica la relación que existe entre el Sistema Operativo y el bus del sistema.

En el capítulo 4 se explican algunas generalidades del estándar Wishbone y se presenta el diseño de las interfaces genéricas maestro y esclavo.

En el capítulo 5 se explica la implementación de cada uno de los dispositivos de entrada salida que conforman el sistema, mostrando su modelo de programación, modelo de hardware, imágenes de la simulación del mismo y estadísticas de implementación.

En el capítulo 6 se presenta un par de bloques adicionales para el control del bus y posteriormente se explica la integración de todos los bloques explicados en el capítulo 5, se incluye un programa ejemplo en ensamblador que hace uso de los dispositivos de entrada salida y se muestran imágenes de simulación del mismo.

En el capítulo 7 se encuentran las conclusiones del trabajo y las recomendaciones para trabajos futuro.

CAPÍTULO 2 ANTECEDENTES Y ESTADO DEL ARTE

En este capítulo se explica qué es la metodología de diseño System On Chip y cuáles son sus principales características. Además se explica la importancia de la arquitectura de comunicación dentro de un diseño SoC y se describen las características de las arquitecturas de comunicación *on-chip* basadas en buses compartidos. Por último se exponen los estándares de comunicación on-chip más utilizados: AMBA, IBM CoreConnect, Altera Avalon y Wishbone.

2.1 Metodología de diseño System On Chip

El continuo desarrollo de la industria de los semiconductores ha permitido un incremento en la densidad de los circuitos integrados. Para poder hacer un uso eficiente de esta nueva tecnología se han desarrollado distintas metodologías para abordar la creciente complejidad en el diseño de chips [3].

El diseño System on Chip (SoC) es una metodología en la cual bloques prediseñados, también conocidos como bloques *IP* (*Intellectual Property*), o *IP cores*, se combinan en un mismo chip. Estos *IP cores* pueden ser controladores de memoria, temporizadores, bloques de comunicación y procesadores embebidos [4].

Un System On Chip es un circuito electrónico que integra los elementos fundamentales de un sistema computacional dentro de un solo chip [24]. En general, los diseños SoC incorporan al menos un procesador programable, memoria *on-chip* y dispositivos periféricos, todo esto interconectado entre sí sobre una arquitectura de comunicación *on-chip* que permite transferir información entre dispositivos, Figura 2.

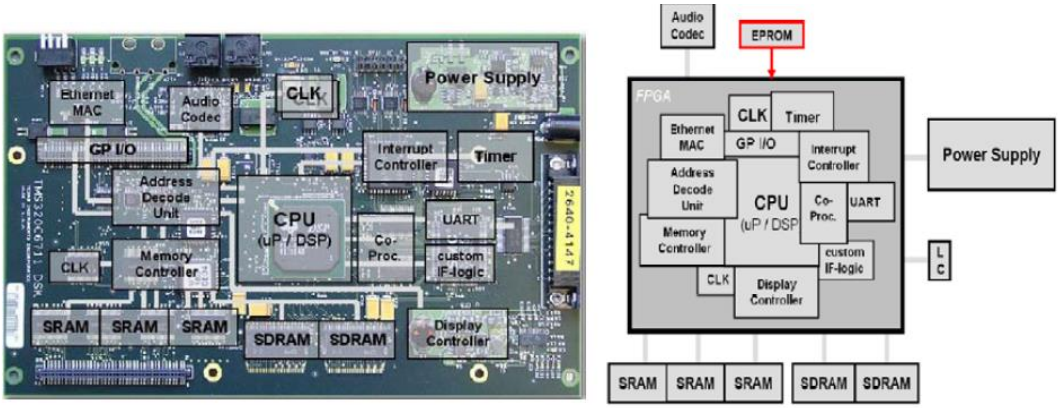


Figura 2. Ejemplo de un System on Chip implementado en un dispositivo FPGA

Dentro de los beneficios del diseño SoC se encuentran:

- Reducción en el número de chips del circuito
- Menores requerimientos de memoria
- Menores costos al consumidor
- Mejor desempeño y sistemas más confiables
- Reducción en el tiempo del diseño

La adopción de la metodología SoC ha demandado muchas de las mejoras tecnológicas de los circuitos integrados. La importancia de los diseños SoC también ha requerido de la estandarización de los buses, formatos de intercambio de IP cores, documentación, etc.

2.2 Arquitectura de Comunicación *on-chip*

La disponibilidad de una gran cantidad de dispositivos en un solo chip ha traído también nuevas posibilidades de conexión entre los mismos, la arquitectura de comunicación es un factor clave para diferenciar el desempeño de un sistema respecto de otro. La arquitectura de comunicación se refiere a todas aquellas características que conforman el medio de comunicación de un circuito digital tales como el protocolo de comunicación utilizado, la topología de conexión y el esquema de arbitraje.

La arquitectura de comunicación determinará tres factores muy importantes del sistema: la escalabilidad, desempeño y consumo de energía.

El uso de distintas arquitecturas de comunicación *on-chip* provee a los diseñadores la habilidad de explorar múltiples topologías, protocolos y esquemas de arbitraje que permitan destacar su producto del resto, tanto a nivel de desempeño, consumo de energía, costo, confiabilidad y disminución de tiempos para salir al mercado.

2.3 Arquitectura de Comunicación On-Chip mediante buses.

La comunicación mediante buses es la forma más común de comunicación entre dispositivos en un System on Chip. La simplicidad y eficiencia de la transferencia de datos mediante buses ha asegurado que se mantengan como el mecanismo de interconexión preferido.

Un bus es un conjunto de cables utilizados para transportar información entre dos o más componentes en un sistema digital. En la práctica, múltiples dispositivos son conectados a un bus para transferir mensajes a otros dispositivos.

Si bien mediante el uso de un bus un dispositivo puede transmitir información a todos los demás dispositivos conectados al mismo tiempo, en la mayoría de los casos la comunicación es de un solo dispositivo a otro, aquellos dispositivos no participantes de la transacción ignoran las señales del dispositivo remitente. Para llevar a cabo dicha transacción es necesario establecer un protocolo de comunicación.

El protocolo de comunicación define de manera explícita las características espaciales (tamaño del bus) y temporales (secuencia de los mensajes) de una transacción, además determina cuales componentes pueden tener acceso al bus compartido si múltiples componentes desean utilizarlo.

2.3.1 Terminología

A continuación se definen algunos términos que serán utilizados para describir las características de la comunicación mediante buses.

- Bloque Maestro: Es aquel componente capaz de iniciar una transferencia de datos, ya sea lectura y escritura. Usualmente se trata de Procesadores.
- Bloque Esclavo: Es aquel componente que únicamente es capaz de responder a las señales solicitadas por un bloque maestro.
- Interfaz: Los componentes Maestro y Esclavo interactúan con el bus a través de una interfaz, la cual consiste en un conjunto de cables conectados al bus, en estos cables se incluyen señales de dirección, datos y control.
- Bridge: Es un componente utilizado para conectar dos buses, los cuales por lo general tienen frecuencias de operación e incluso protocolos de comunicación distintos. Este tipo de componentes son un componente híbrido de bloques maestro y esclavo.

2.4 Características de las arquitecturas de comunicación

Las arquitecturas de comunicación tienen diversas características físicas y estructurales dentro de las que destacan las siguientes:

2.4.1 Tipos de señales

Las señales de los buses son clasificadas en tres categorías: señales de dirección, datos y control, Figura 3.

Figura 3. Tipos de señales dentro de un bus

Líneas de dirección

El número de señales (ancho del bus) usadas para transmitir la dirección es usualmente una potencia de 2 (valores comunes son 16, 32, 64). La mayoría de los sistemas tienen un solo bus de direcciones sobre el cual se realizan tanto lecturas como escrituras, sin embargo, es posible tener buses de dirección independientes, lo cual mejora el

desempeño del sistema puesto que muchas transferencias de datos pueden realizarse en paralelo, pero el consumo de energía y el área del diseño incrementan de manera considerable.

Líneas de datos

Estas señales se utilizan para enviar el valor de los datos a la dirección destino. El tamaño típico del bus de datos (ancho del bus) suele ser de 16, 32, 64, 128, 256, 512 y 1024 bits, este ancho depende de los requerimientos específicos del sistema. El número de señales del bus determina si es necesario implementar un sistema de empaquetado (packing) o desempaquetado (unpacking) de los datos. En los sistemas donde el tamaño del bus de datos es del mismo tamaño que la palabra del procesador (o la memoria) no es necesario hacer ninguna clase de empaquetado.

De manera similar al bus de direcciones, el bus de datos puede ser implementado con un solo bus para lecturas y escrituras o con buses independientes.

Líneas de control

Estas señales son utilizadas para enviar información acerca de la transferencia y son específicas de cada protocolo de comunicación. Las señales *request* y *acknowledge* son las más comunes, la primera proviene de un dispositivo maestro que solicita el uso del bus al sistema y la segunda proviene del esclavo e informa al dispositivo maestro que los datos han sido recibidos. Otras señales de control pueden ser señales de error, de identificación de dispositivos, de identificación de tipo de transmisión, etc.

2.4.2 Estructura física del bus

Una forma de conectar los dispositivos al bus es mediante el uso de buffers tri-estado, los cuales gestionan líneas bidireccionales de datos, Figura 4. Tiene la ventaja de utilizar poca área pues la cantidad de señales es reducida, pero tiene la desventaja de tener un alto consumo de energía, además de que los retrasos de la señal que pueden limitar el desempeño y la depuración puede ser problemática.

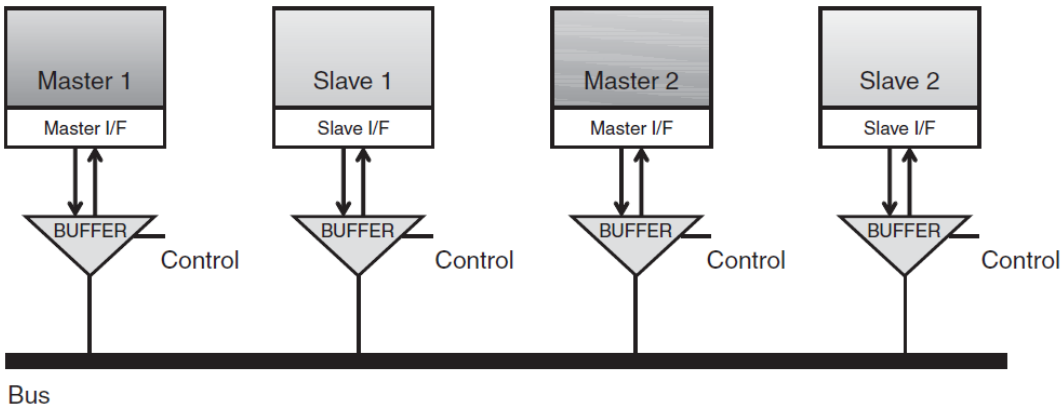


Figura 4. Bus compartido con buffer tri-estado

Una alternativa más eficiente es utilizar multiplexores la cual, a diferencia de la implementación basada en buffers, es sintetizable en cualquier dispositivo de lógica programable, Figura 5, además de tener un consumo de energía menor [3].

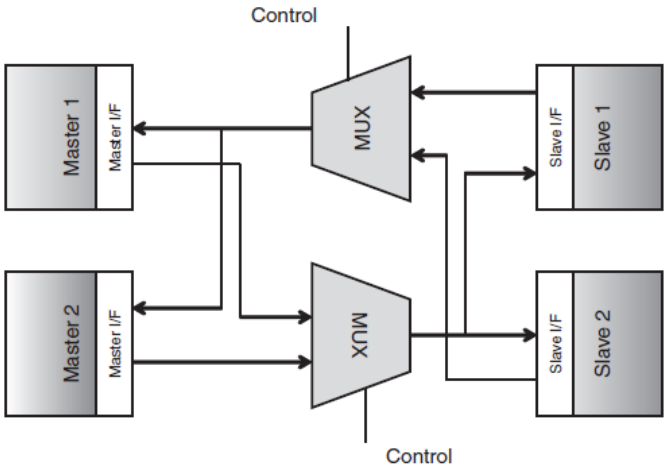


Figura 5. Bus compartido basado en multiplexores.

2.4.3 Estrategias de sincronía.

La estrategia de sincronía se refiere al hecho de que la señal de reloj forme parte de las señales de control del bus o no, de ser así se dice que el bus es síncrono, de lo contrario es un bus asíncrono.

Bus Síncrono

Los buses que incluyen una señal de reloj como parte de las señales de control son denominados buses síncronos. La señal de reloj coordina las distintas etapas de la transmisión de datos dentro del bus. En la Figura 6 se muestra un ejemplo de un bus síncrono en el cual el maestro solicita una dirección destino en un ciclo de reloj, y en el siguiente envía el dato, el tercer paso sería la recepción de alguna señal de reconocimiento (o de error) por parte del esclavo.

Los buses utilizados en la mayoría de los estándares de comunicación *on-chip* son síncronos, estos buses alcanzan altas velocidad de transmisión pero pueden requerir del uso de convertidores de frecuencia, puesto que no todos los componentes conectados al bus trabajan a la misma frecuencia. De hecho en diseños SoC modernos, procesador trabaja a una frecuencia de reloj de dos a cuatro veces más rápida que el reloj del bus [3].

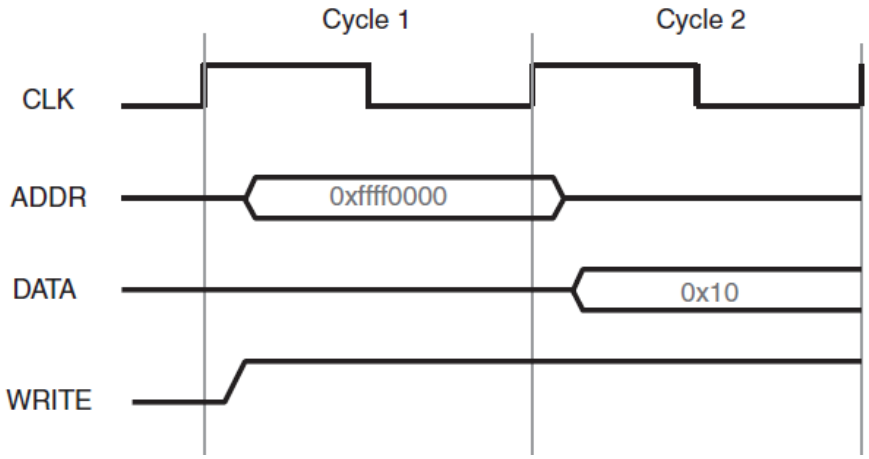


Figura 6. Ejemplo de bus síncrono

Bus Asíncrono

En los buses asíncronos no hay una señal de reloj dentro del bus de control. En este caso, sincronía del bus depende de un protocolo de *handshake* que usa señales de solicitud-reconocimiento (*request-acknowledgment*) para asegurar que una transferencia ha sido completada satisfactoriamente.

La Figura 7, muestra un ejemplo de una transmisión asíncrona, el maestro envía la dirección, el dato, la señal de escritura y una señal de solicitud de transacción (REQ), una vez el esclavo recibe dichas señales envía una señal de reconocimiento (ACK) de vuelta al maestro, el cual a su vez la reconoce al negar la señal de solicitud (REQ).

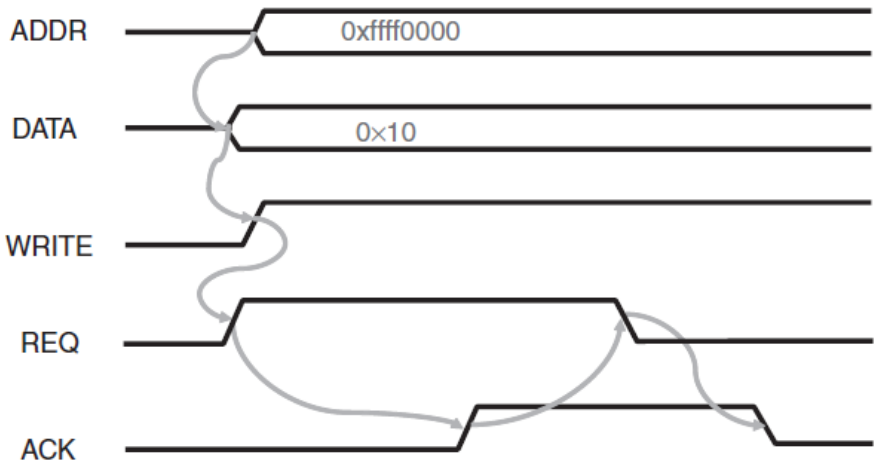


Figura 7. Ejemplo de bus asíncrono

Los procesos de *handshaking* en los buses asíncronos incluyen más señales y condiciones que en los buses síncronos, por lo que suelen ser más lentos, pero no requieren de convertidores de frecuencia por lo que consumen menos área de chip.

2.4.4 Arbitraje

En un bus compartido es posible que dos o más maestros requieran hacer uso del bus al mismo tiempo, por lo que es necesario tener un mecanismo de arbitraje que determine cual maestro tomará el control del bus primero.

Un árbitro es un componente en los buses compartidos que utiliza cierto criterio para terminar cual maestro obtiene el acceso al bus, a dicho criterio se le denomina esquema de arbitraje.

Los requerimientos básicos de un esquema de arbitraje es que garantice el acceso a todos los dispositivos, pero que al mismo tiempo asegure que las transferencias críticas se realizan tan rápido como sea posible. Los esquemas de arbitraje más comunes son:

Prioridad estática.

En este esquema cada maestro tiene un valor de prioridad fijo. Para decidir que maestro tiene acceso al bus basta con elegir aquel cuya prioridad sea más alta. Este esquema es bastante simple de implementar y alcanza un alto desempeño, puesto que las transferencias críticas acceden al bus rápidamente, sin embargo, solicitudes de maestros de prioridad baja podrían ser ignoradas por completo.

Round Robin.

En este esquema la prioridad de los maestros va cambiando de manera circular, esto es, cualquier dispositivo puede estar en la posición de prioridad más alta en un instante de tiempo y posteriormente tener la prioridad más baja. Este esquema asegura que todos los maestros tengan un acceso justo al bus. La principal desventaja es que el desempeño es limitado, puesto que las transferencias críticas no tienen un acceso inmediato al bus.

Time division multiple Access (TDMA)

En este esquema a cada maestro se le asigna un slot de tiempo cuya longitud depende de la importancia del maestro en el sistema. Una vez el maestro acceda al bus tendrá el control del mismo durante la duración de su slot, la longitud del mismo debe de ser lo suficientemente larga para completar la transferencia de al menos un conjunto de datos. Este esquema garantiza un acceso rápido de los bloques críticos sin ignorar por completo a los maestros de menor prioridad.

2.5 Topologías de arquitecturas On-Chip

2.5.1 Bus compartido

Existen diversas arquitecturas para la comunicación *on-chip*, la más simple de éstas es la de bus compartido simple, Figura 8, la cual consiste de un conjunto de cables que es compartido por diversos componentes. Únicamente un componente en el bus puede tener el control de los cables en un tiempo determinado, esta arquitectura es de las más utilizadas, y si bien, no es una arquitectura que pueda ser escalable para sistemas muy demandantes, es muy eficiente para diseños con pocos componentes [5].

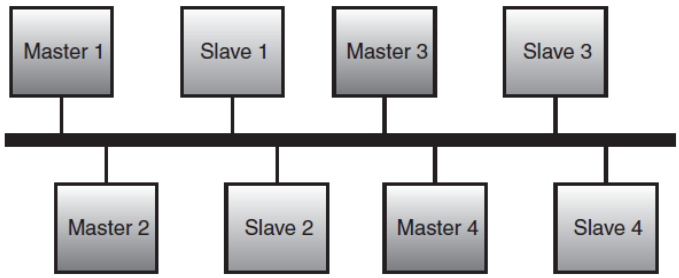


Figura 8. Topología de bus compartido.

2.5.2 Bus compartido jerárquico.

Una topología que permite múltiples transferencias de datos en paralelo es la topología de bus jerárquico, Figura 9. En esta topología los componentes están conectados a múltiples buses que se comunican entre sí por medio de puentes. Cada bus puede tener una frecuencia de reloj distinta, por lo que el diseño del puente puede ser bastante complejo, pues debe ser capaz de manejar las transacciones entre buses, buffer de datos, realizar conversiones de frecuencia, etc. Un ejemplo de SoC que hace uso de esta topología es ARM PrimeXsys SoCs[16], el cual tiene aplicaciones en dispositivos móviles.

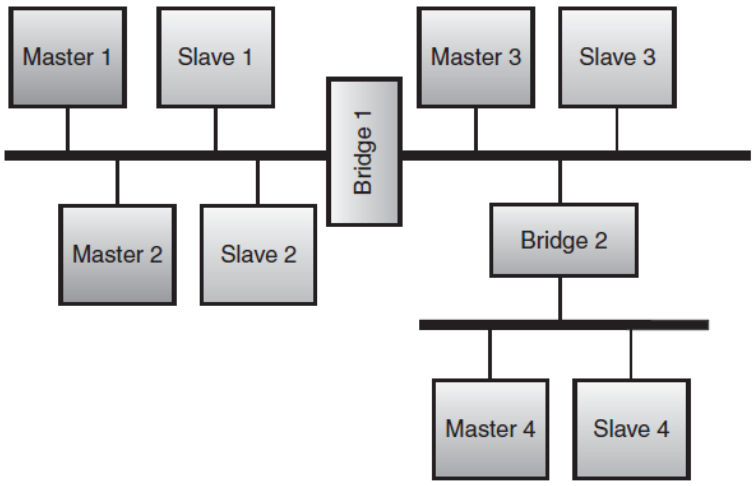


Figura 9. Topología de bus jerárquico

2.5.3 Full Cross Bar

Para sistemas con mayores requerimientos de desempeño es necesario utilizar topologías que permitan tener un mayor paralelismo, tal es el caso de la topología *full cross bar* (o *full bus matrix*). En esta topología cada maestro está conectado a cada bus mediante un bus independiente, Figura 10, por lo que se tiene un esquema de comunicación punto-punto.

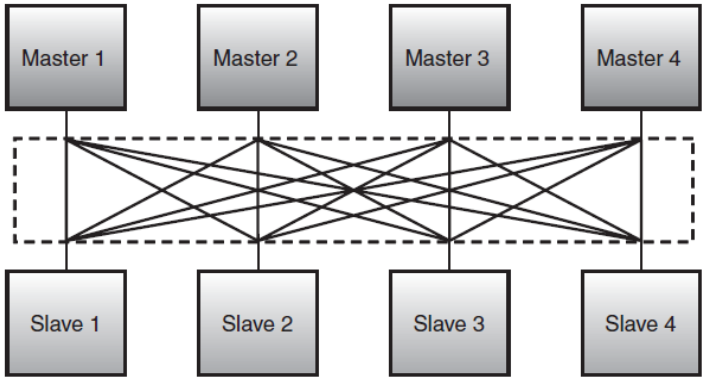


Figura 10. Topología *full cross bar*

La gran cantidad de buses en esta topología permiten la realización de distintas transferencias en paralelo. Cabe mencionar que a diferencia de las anteriores topologías descritas la topología *full cross bar* requiere de mecanismos de arbitraje para cada esclavo.

Si bien esta topología ofrece un desempeño superior al resto, el área consumida suele ser grande, lo cual incrementa el consumo de energía considerablemente, además de que el ruteo de las señales es complejo [6].

Está topología puede ser excesiva para sistemas pequeños, pero tiene presencia en sistemas de alto desempeño. Un ejemplo de está topología es el SoC Niagara Multiprocessor de la empresa SUN Systems [17], el cual conecta ocho procesadores SPARC a cuadro bancos de memoria cache L2, un bridge I/O y una FPU.

2.6 Estado del Arte: Estándares de comunicación *on-chip*

Desde principios de los años 1990s se han propuestos distintos estándares de comunicación on-chip como una manera de mantener compatibilidad entre los distintos IP cores desarrollados. Estos estándares son utilizados para facilitar la rápida interconexión de los IP cores dentro de un diseño SoC, el contar con interfaces bien definidas simplifica el proceso de integración y promueve el reuso del diseño [25].

Los estándares más populares son AMBA de los procesadores ARM, IBM CoreConnect, Altera Avalon y OpenCores. A continuación se describirá de manera breve las principales características de cada uno.

2.6.1 AMBA

AMBA es un estándar originalmente concebido para permitir la comunicación entre procesadores ARM, sin embargo, en la actualidad es posible utilizarlo para la interconexión de cualquier IP core [10]. La Figura 11 muestra un ejemplo de la configuración de este bus.

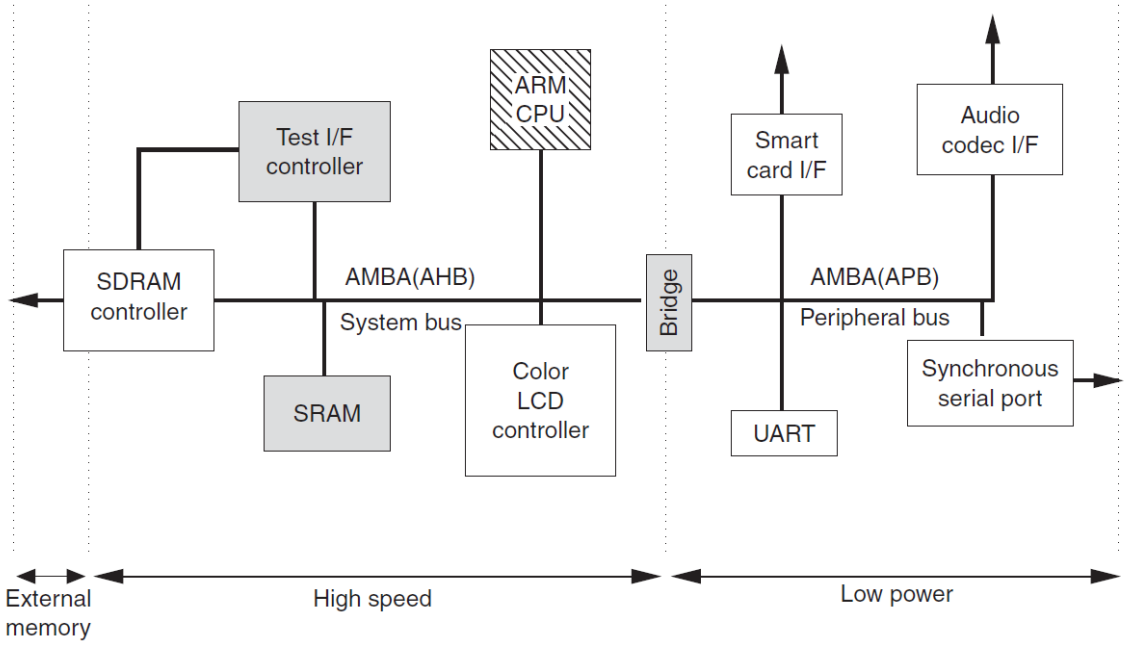


Figura 11. Ejemplo de configuración de bus AMBA

AMBA define tres distintos buses:

1) *Advanced high performance bus* (AHB), el cual es un bus de alta velocidad al cual se conectan los microprocesadores, controladores DMA, controladores de memorias off-chip y bloques de memoria on-chip
2) *Advanced System Bus* (ASB), el cual es una alternativa a AHB, también es un bus de alta velocidad pero que no tiene todas las características avanzadas de AHB.
3) *Advanced Peripheral Bus* (APB), es un bus optimizado para la operación con bajo consumo de energía, está diseñado para comunicar dispositivos de baja velocidad como Timers, UARTs, teclados, etc.

AMBA es por lo tanto un estándar basado en una comunicación de bus jerárquico aunque puede ser implementado con otras topologías. El bus AHB incluye características específicas para ser utilizado con topología Bus Matrix, Figura 12, tales como bloques de arbitraje distribuidos o realización de transacciones fuera de orden (*OO transaction completion*).

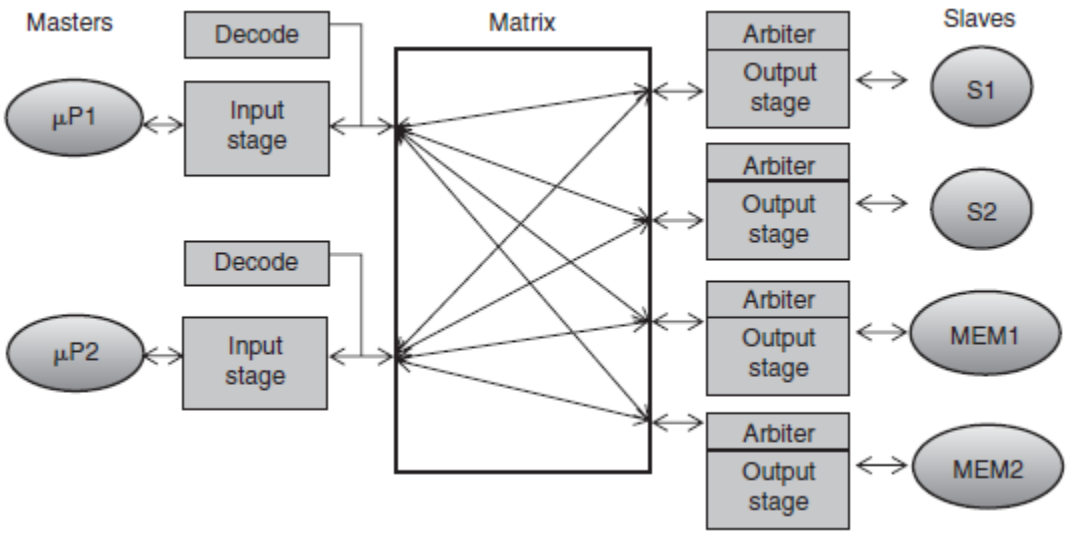


Figura 12. Bus AMBA con topología Bus Matrix

2.6.2 IBM CoreConnect

IBM CoreConnect ofrece un esquema de interconexión versátil orientado a sistemas de alto rendimiento, por lo que muchas de sus características principales resultan excesivas para sistemas embebidos simples [13].

La arquitectura IBM CoreConnect comparte diversas características con la especificación AMBA. CoreConnect también utiliza principalmente una configuración de bus jerárquico, Figura 13, con un bus principal de alta velocidad (PLB) y uno de baja velocidad (OPB) los cuales están enlazados por un puente (bus bridge) que permite la comunicación entre ambos buses.

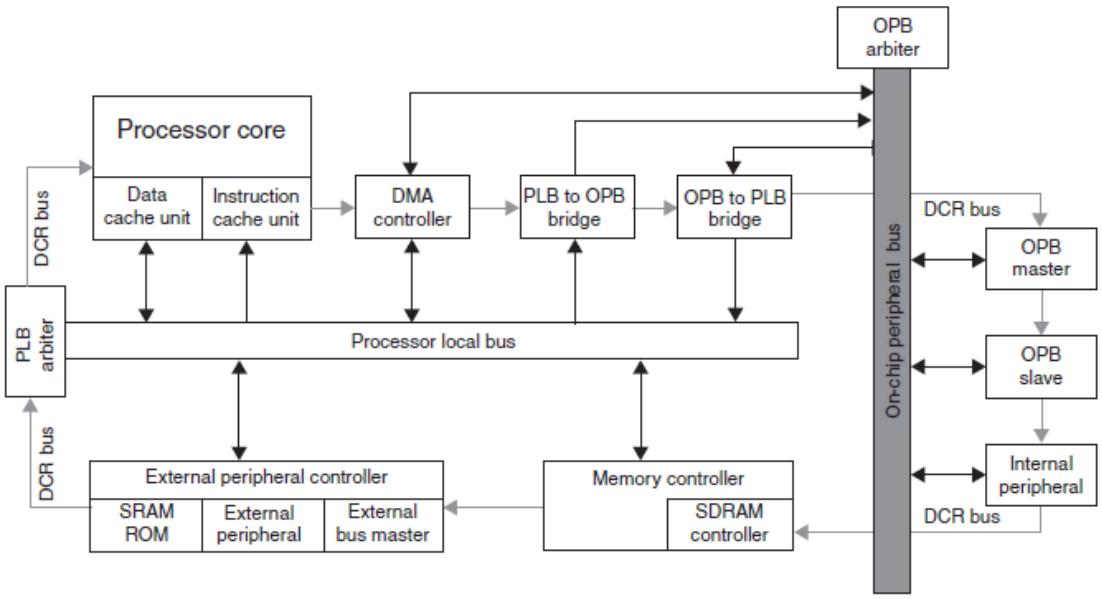


Figura 13. Configuración del bus CoreConnect

2.6.3 Altera Avalon

Esta arquitectura está enfocada en diseños System-on-programmable-chip (SoPC), y está compuesta por dos estándares: Avalon Memory Mapped (Avalon MM) y Avalon Streaming (Avalon-ST).

Avalon-MM define una interfaz para conectar maestros (principalmente procesadores) y esclavos (UARTs, memorias, Timers, etc) mediante un mapeo de memoria, la principal características de Avalon es la configurabilidad de las interfaces.

A diferencia de los estándares anteriores, la implementación de Avalon es del tipo bus *crossbar*, Figura 14. Esta topología tiene integrado un controlador de interrupciones y soporta lógica adicional para hacer transmisiones a otros dominios de reloj y entre anchos de palabra variables. Es utilizado por default como medio de interconexión de procesadores Nios [14].

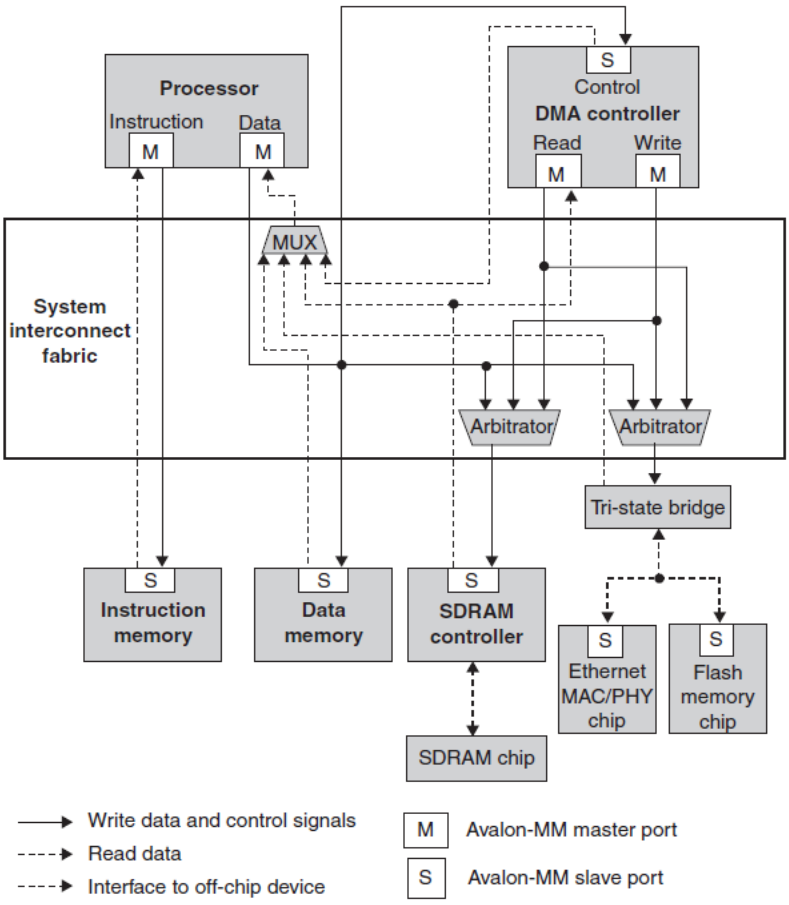


Figura 14. Estructura general del bus Avalon

2.6.4 Wishbone.

Wishbone es una interfaz portable para el uso con IP cores cuyo propósito es mitigar los problemas de integración en diseño SoC [7]. Esta interfaz de comunicación no requiere del uso de herramientas de desarrollo específicas ni va orientada a un hardware en particular, es una interfaz de propósito general. Wishbone define una serie de reglas para la comunicación de datos, dichas reglas implican una serie de características temporales y espaciales a las que el diseñador debe atenerse.

Algunas características importantes de esta interfaz son:

- Simple y compacta. El bloque de hardware para la interconexión requiere una poca cantidad de unidades lógicas.
- Ancho de bus de datos variable desde 8 a 64 bits.

- Ancho de bus de direcciones hasta 64 bits.
- Soporte de esquemas “Multiple master”
- Topologías Point-to-Point, data flow, shared bus, crossbar.
- Esquema de arbitraje flexible (seleccionado por el usuario)
- El esquema es altamente configurable, permitiendo al usuario la personalización de etiquetas (tags) o señales para cumplir con los requerimientos de la aplicación.

La interfaz de comunicación Wishbone permite implementar una metodología de diseño estructurado, en la cual cada miembro de un equipo de trabajo realiza el diseño de una entidad de hardware tomando en cuenta el estándar de conexión de Wishbone para posteriormente integrar todas las entidades. En la Figura 15 se puede observar un esquema de interconexión *shared bus* (bus compartido).

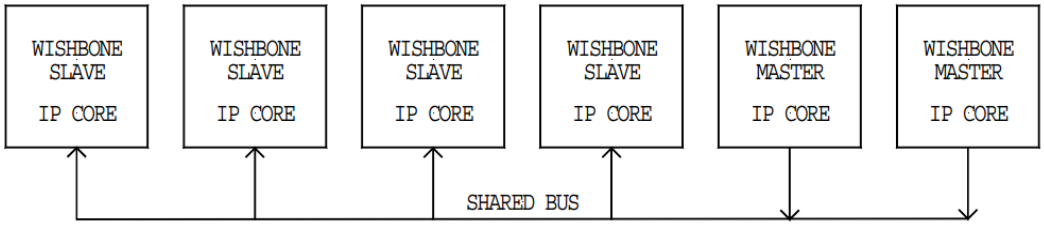


Figura 15. Esquema de bus compartido de Wishbone

2.6.5 Comparación entre estándares.

En [9-11] se ha realizado una comparación de Wishbone con otros estándares populares como ARM AMBA e IBM CoreConnect. Las conclusiones sugieren que el estándar Wishbone provee un esquema simple y flexible para la interconexión de bloques lo que en general facilita el diseño SoC. En términos de rendimiento el estándar Wishbone puede cumplir con todos requerimientos de un sistema embebido simple, y si bien no presenta algunas características relevantes para el desarrollo de sistemas de alto rendimiento como la topología de bus jerárquico la flexibilidad de Wishbone permite adaptar el estándar a sistemas más demandantes.

En [8] se describe el diseño e implementación de dos tipos de sistemas que utilizan bloques DMA maestro y bloques de memoria esclavos, utilizando los esquemas de comunicación Point-to-Point y Shared Bus de Wishbone, además de presentar algunas de las características de Wishbone que fueron de utilidad para el diseño, se concluye que la interfaz requiere de poca lógica para su implementación.

CAPÍTULO 3. PROCESADOR, SISTEMA OPERATIVO Y BUS DE SISTEMA

En este capítulo se describen las características del procesador Lagarto I, posteriormente se abordan conceptos relacionados con la gestión de los controladores de entrada salida por parte del sistema operativo, así como la relación del mismo con el bus del sistema.

3.1 Procesador Lagarto I

El procesador es el circuito electrónico dentro de una computadora que se encarga de ejecutar las instrucciones de un programa de computadora al realizar operaciones aritméticas, lógicas y de control de estrada salida.

Los componentes principales del procesador incluyen la unidad aritmética lógica (ALU) donde se realizan las operaciones, el banco de registros que suple los operandos a la ALU y almacena los resultados de la misma, y la unidad de control que dirige el flujo de los datos desde la lectura de instrucciones hasta la etapa de almacenamiento.

Lagarto I es un procesador escalar basado en arquitectura MIPS [1] que cuenta con un pipeline de seis etapas de profundidad en las que se realiza la emisión, decodificación y ejecución de las instrucciones: *Fetch*, *Decode*, *Read Registers*, *Execute*, *Memory Access* y *Write Back*. Cada una de estas etapas se encuentra separada por un registro interetapa (IF/ID, ID/IRR, RR/EX, EX/MEM, MEM/WB) en el que se almacenan los datos provenientes de cada etapa, mismos que serán emitidos en cada ciclo de reloj. En la Figura 16 se observa el pipeline del procesador.

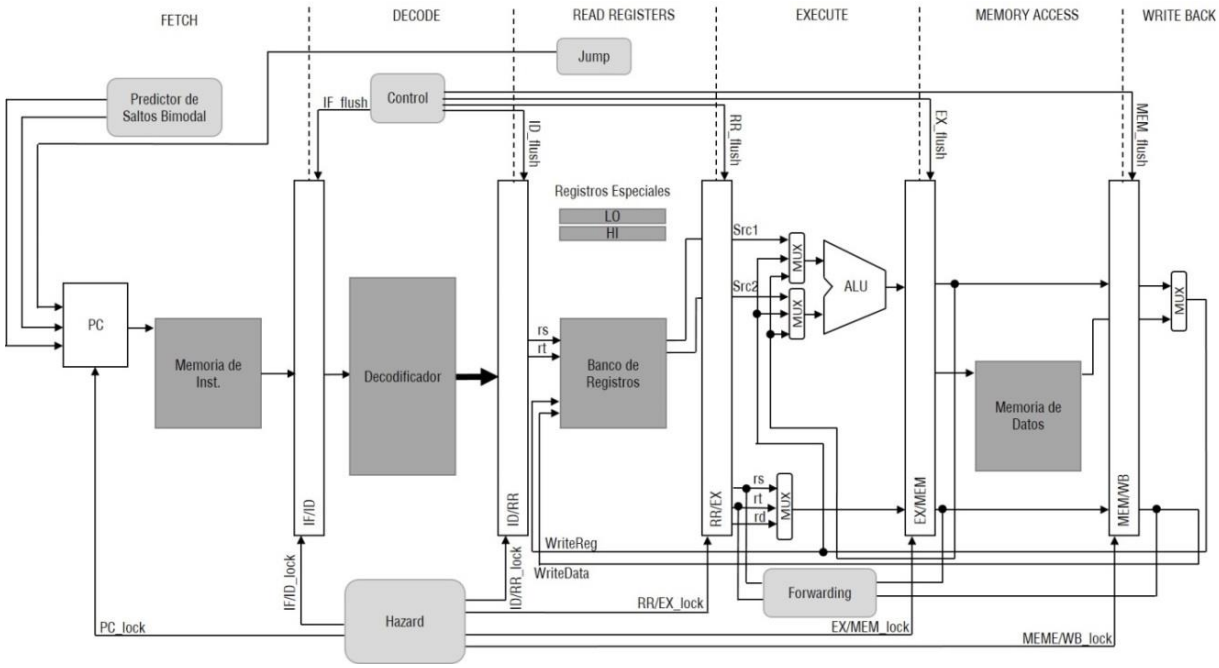


Figura 16. Etapas del procesador Lagarto I.

3.1.1 Búsqueda y emisión de Instrucciones.

La información que recorre el pipeline durante cada ciclo de reloj tiene su origen en esta etapa, la cual es conocida por su término en inglés *fetch*. Está conformada por un registro *Program Counter* (PC) el cual determina la dirección de la instrucción a ser leída de la memoria de instrucciones en cada pulso de reloj.

Lagarto I es un procesador de 32 bits, cada instrucción está formada por 4 bytes que se encuentran almacenados en registros contiguos de la memoria de instrucciones. Cuando un programa ejecuta instrucciones de forma ininterrumpida el Program Counter incrementa su valor en cuatro a cada ciclo de reloj, sin embargo, cuando se presentan instrucciones de salto el Program Counter interrumpe su conteo normal y toma un valor determinado por dichas instrucciones.

3.1.2 Decodificación

Esta etapa recibe la instrucción leída de la memoria de instrucciones. En la especificación del set de instrucciones de la arquitectura MIPS [1] se clasifica cada tipo de instrucción dependiendo de un código de operación embebido en la misma instrucción, Tabla 1.

Tabla 1. Códigos de operación de MIPS

Codificación	Opcode	Direcciónamiento
Special	000000	Function
RT-field	000001	rt
Special2	011100	Function
Opcode	xxxxxx	opcode

En la etapa de decodificación se realiza un análisis del código de operación con el objetivo de determinar el valor de bits de recursos, operación y control, mismos que serán utilizados por las etapas siguientes.

3.1.3 Lectura del Banco de Registros.

En esta etapa se accede al Banco de Registros del procesador, el cual es un conjunto de 32 registros de propósito general, cada uno de 32 bits, los cuales serán utilizados para la realización de operaciones aritmético-lógicas en la etapa de ejecución.

3.1.4 Ejecución.

En esta etapa se realizan tanto las operaciones aritmético-lógicas como operaciones especiales definidas en el conjunto de instrucciones. El procesador Lagarto I cuenta con los siguientes módulos de ejecución:

- Un sumador/restador.
- Un multiplicador/divisor.
- Un módulo para operaciones lógicas.

- Un módulo para detección y cálculo de datos condicionales.
- Un módulo para la transferencia de datos entre registros
- Un módulo para la detección de trampas
- Un contador de unos o ceros
- Un sumador específico para el cálculo de direcciones de acceso a memoria.

Cada módulo es capaz de operar con datos de 32 bits y definir si el tratamiento de la operación es con signo o no, según sea determinado por el vector de operación.

3.1.5 Acceso a memoria

En esta etapa se realizan las operaciones de carga y almacenamiento de información en la memoria de datos. El procesador es capaz de decodificar diez instrucciones de carga y siete instrucciones de almacenamiento. En ambos casos es posible el tratamiento de datos de 1 byte, 2 bytes y 4 bytes.

3.1.6 Escritura de retorno.

En esta etapa se almacenan los valores provenientes de la etapa de ejecución o de acceso a memoria en el Banco de Registros.

3.2 Unidad de Gestión de Memoria (MMU)

La unidad de gestión de memoria es un elemento de hardware que forma parte del CPU y cuya principal función es la traducción de direcciones de memoria virtual a direcciones físicas. Cuando el procesador intenta acceder a una dirección virtual, la MMU realiza una búsqueda en una memoria caché especial llamada Buffer de traducción de direcciones (TLB, *Translation Lookaside Buffer*). En esta memoria se mantienen las entradas de la tabla de páginas (PTE, *Page Table Entry*), donde pueden ser leídas las direcciones físicas correspondientes a algunas direcciones virtuales de forma directa.

En esta unidad son implementados todos los procesos requeridos para la interfaz entre el núcleo del procesador y los distintos niveles de memoria. Un beneficio fundamental de la

MMU es la posibilidad de implementar protección de memoria, evitando que los programas accedan a porciones de memoria prohibidas.

La unidad de gestión de memoria del procesador Lagarto utiliza TLBs de mapeo directo de 32 entradas. La caché de instrucciones es de conjunto asociativo con dos vías y bloques de 32 bytes, el tamaño total es de 64 KB. La caché de datos es de conjunto asociativo de 4 vías y bloques de 64 bytes, el tamaño total es de 128 KB.

3.3 Procesador de Sistema

El procesador de sistema tiene como tarea manejar las excepciones que ocurran dentro del procesador, de tal manera que el Sistema Operativo no tenga que realizar esta operación. Las excepciones son cualquier cambio inesperado en el flujo del programa. Existen excepciones síncronas y asíncronas. Ejemplo de las primeras son sobreflujos aritméticos, instrucciones indefinidas y errores en paginación. Las excepciones asíncronas incluyen interrupciones de dispositivos de entrada/salida, errores de memoria y fallos en la fuente de alimentación.

Cuando una excepción ocurre, el control es transferido a un programa llamado *Manejador de Excepciones*, escrito con el único propósito de lidiar con las excepciones. Una vez que se ha terminado de atender la excepción el control regresa nuevamente al programa que estaba siendo ejecutado.

Actualmente se tiene un diseño del procesador de sistema el cual consta de cuatro etapas de ejecución: *Decode, Format, Register File* y *Data Issue* [18].

3.4 Sistema Operativo y Bus de Sistema

Dentro de un sistema computacional existe una constante interacción entre el CPU y otros elementos de hardware, tales como memorias RAM, temporizadores, controladores de interrupciones, etc. La comunicación con dichos dispositivos puede ser gestionada mediante la programación a bajo nivel, o bien, mediante el uso de un sistema operativo.

El Sistema Operativo de la computadora provee una forma genérica, conveniente y confiable de acceder a los dispositivos de entrada salida (I/O) [19]. Actualmente las

computadores son capaces de interactuar con una gran cantidad de dispositivos, los cuales pueden clasificarse como dispositivos de almacenamiento (discos duros o cintas), dispositivos de transmisión (conexiones a redes, Bluetooth) y dispositivos con interfaz humana (monitor, teclado, mouse, audio).

Un dispositivo de comunica con la computadora al enviar señales a través de un sistema de cables. El punto de conexión de esta comunicación se denomina puerto. Comúnmente el dispositivo es conectado a un bus, el cual define un protocolo que especifica la manera en la que los mensajes serán entregados, ver Capítulo 2.

Los buses son ampliamente utilizados en arquitectura de computadoras y varían en sus métodos de señalización, velocidad y métodos de conexión. Una estructura de bus típico se presenta en la Figura 17, en donde se observa la conexión entre el procesador y los distintos subsistemas de memoria, así como un bus de expansión que conecta dispositivos con velocidades relativamente lentas, tales como teclados y puertos seriales. También es posible observar un bus SCSI (Small Computer System Interface) conectado al bus principal mediante un controlador SCSI. Existen otros buses usados para la interconexión de dispositivos como el PCI Express (PCIe) el cual suele tener tasas de envío de hasta 16 GB/segundo.

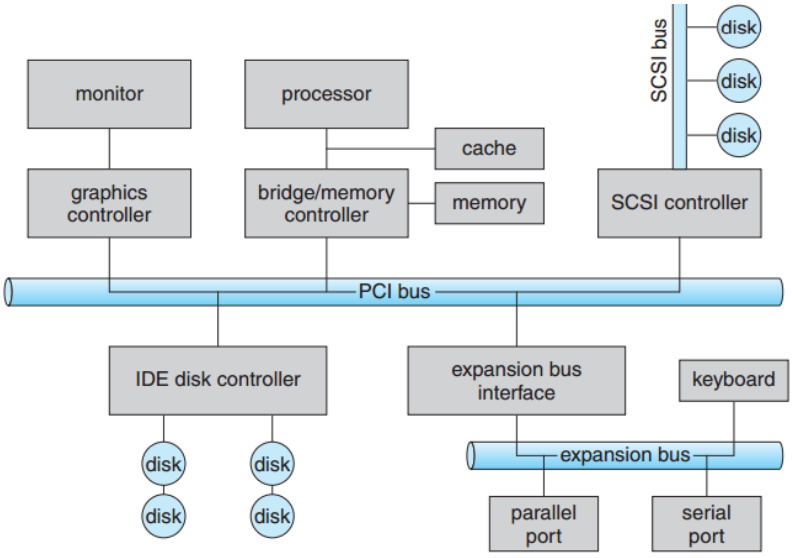


Figura 17. Configuración típica del bus de una PC

3.4.1 Controladores de entrada salida

Un controlador es un elemento que permite operar un puerto, un bus, o un dispositivo. Puede ser un chip dedicado, o en el caso de los System On Chip, un segmento del mismo. Un controlador puede contener dentro de sí un procesador, memorias y registros especiales. El procesador envía comandos y recibe datos desde el controlador a través de la lectura y escritura de registros de estado, de control y de datos:

- Registro de datos de entrada/salida. Son aquellos que pueden ser escritos o leídos por el Host.
- Registros de Estado. Contiene bits que pueden ser leídos por el Host. Estos bits indican el estado actual del controlador, algunos ejemplos incluyen: el bit de fin de inicialización, bit de ocupado, bits de error, etc.
- Registro de Control. Pueden ser escritos por el Host para iniciar un comando o para cambiar el modo de operación del dispositivo. Por ejemplo bits que habilitan el tamaño de palabra de un puerto serial, selección de velocidad, selección de paridad, etc.

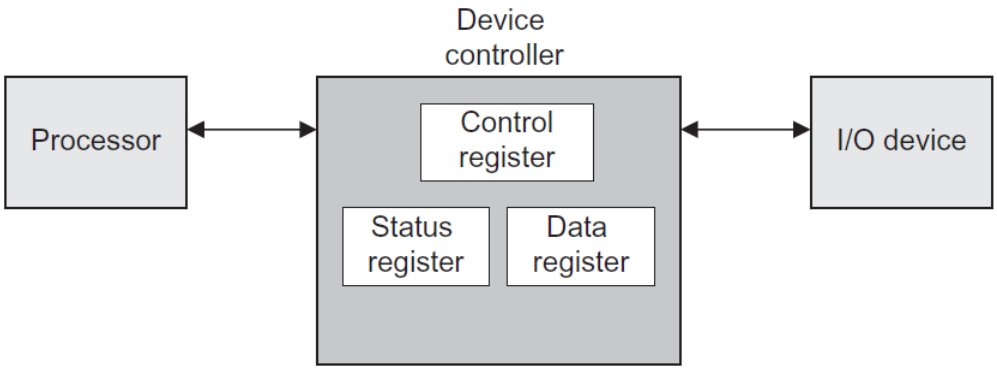


Figura 18. Estructura general de un controlador de dispositivo

Algunas arquitecturas de procesadores, como es el caso de MIPS, tienen los controladores de entrada salida mapeados en memoria, es decir, los registros de control tienen asignados una dirección específica del espacio de direcciones del procesador. El CPU ejecuta solicitudes I/O usando sus instrucciones estándar para leer o escribir registros en la memoria.

Existen ocasiones en las que los dispositivos de entrada/salida requieren de la atención del procesador, sin embargo, al ser dispositivos esclavos en el bus, no pueden

iniciar una transferencia de información en el mismo. Existen dos métodos para realizar el testeo del estado de los dispositivos: muestreo y por interrupciones.

Muestreo.

En este método, también conocido como *polling,* el procesador debe comprobar el estado de un dispositivo mediante la lectura de alguno de su registro de estado, una vez se compruebe que un determinado bit está activado se procede a dar atención al dispositivo.

En un caso ideal este proceso puede llevarse a cabo en tres ciclos de instrucción: lectura de registro, extracción del bit de estado y atención del dispositivo. Sin embargo, este método es sumamente ineficiente, pues el código principal del procesador se convertiría en una serie de bucles de testeo de dispositivos, lo cual tendría un efecto relevante en el desempeño del sistema.

Interrupciones.

El método de interrupciones permite al procesador ejecutar su programa principal y que únicamente sea interrumpido en el momento en que los dispositivos lo requieran. Para esto se requiere que el hardware del CPU tenga una entrada denominada *Interrupt-Request line*.

El mecanismo básico del método de interrupciones funciona de la siguiente manera: Cuando el procesador de sistema detecta que un controlador solicita su atención mediante la línea de *Interrupt Request,* el procesador almacena el estado actual del sistema y salta a una dirección fija en la memoria donde se ubica la rutina de manejo de interrupción o *interrupt-handler routine.* Dicha subrutina determina la causa de la interrupción, realiza el procesamiento necesario, restaura el estado del sistema y realiza un salto de vuelta al punto donde fue interrumpido el código principal. El diagrama de la Figura 19 resume los distintos procesos descritos.

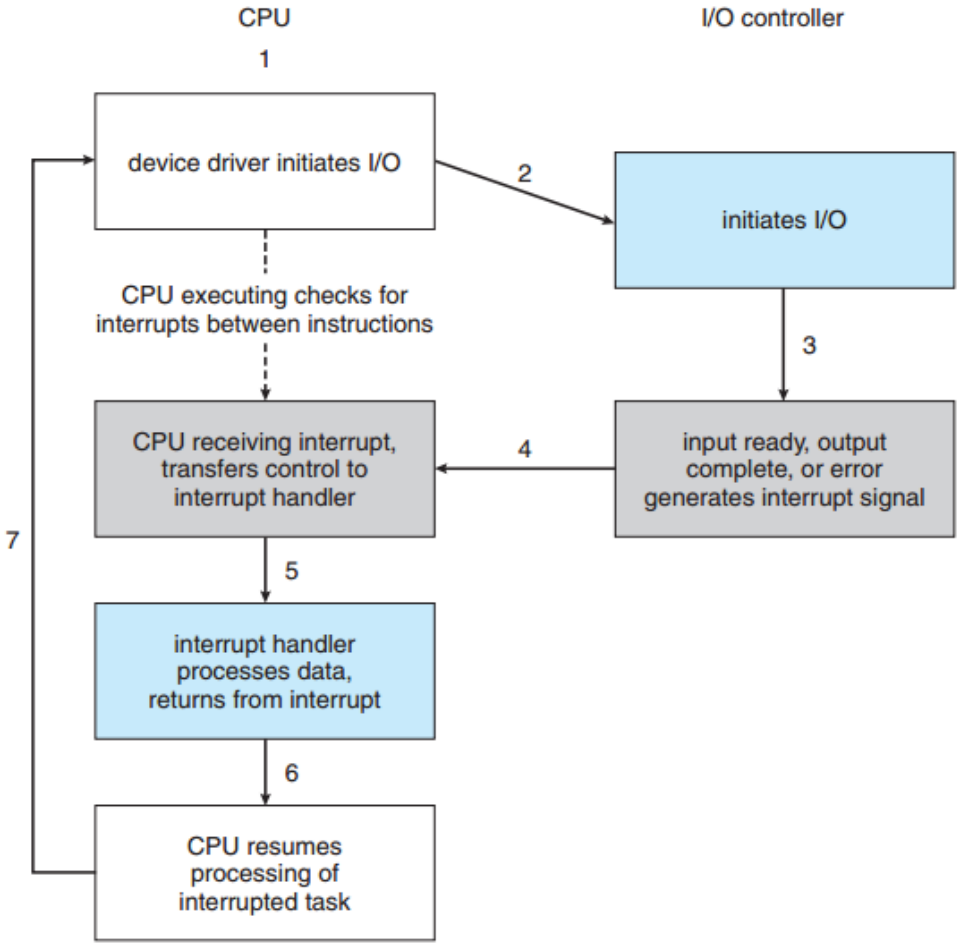


Figura 19. Diagrama de flujo del método de interrupciones.

Dentro de los principales dispositivos de entrada salida se encuentran:

- Controlador de Interrupciones
- Temporizador
- Módulo de comunicación serial UART
- Teclado
- Mouse
- Impresoras
- Controladores de Discos

3.4.2 Temporizador.

El temporizador es un dispositivo importante utilizado por el sistema operativo para implementar el esquema multi-tareas y otros trabajos. Es utilizado para que el CPU tenga un sentido del tiempo y habilitar con esto, una operación en un instante de tiempo determinado. Si bien el CPU cuenta con una señal de reloj de entrada, el hecho de que muchas veces el procesador detenga su funcionamiento requiere del uso de una señal de tiempo externa. Los temporizadores realizan las siguientes funciones:

- Generan interrupciones periódicas
- Se utilizan por el planificador del sistema para generar interrupciones cuando hay necesidad de hacer un cambio de procesos.
- Puede ser utilizado por el sistema de discos
- Se puede usar para cancelar operaciones que causan congestiones dentro de una red.

La frecuencia del temporizador de sistema (*tick rate*) es programa durante la inicialización del sistema, basándose en el valor de la directiva *HZ,* localizado en el archivo <asm/param.h> dentro del kernel. Este valor difiere de cada arquitectura y varía de 10 a 1,000 Hz. En el caso de arquitectura MIPS este valor es 100 Hz.

3.4.3 Controlador de Interrupciones

El controlador de interrupciones programable (Programmable Interrupt Controller, PIC) gestiona el ambiente de las interrupciones de dispositivos. Es el dispositivo que permite la implementación del método de interrupciones.

El controlador de interrupciones recibe las solicitudes de los dispositivos de entrada salida y determina cuál de todas tiene una mayor prioridad para posteriormente informar al procesador sobre la existencia de una interrupción de hardware, el CPU realiza una lectura al PIC para obtener la dirección del dispositivo causante de la interrupción, luego de lo cual se realiza un salto a la dirección correspondiente para atender al dispositivo.

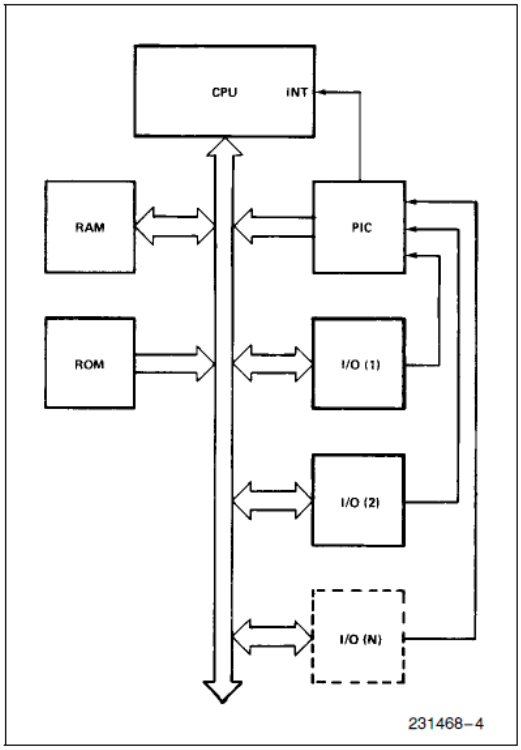


Figura 20. Conexión del controlador de interrupciones con el CPU.

3.5 Espacio de Memoria de MIPS.

Los procesadores MIPS utilizan dos niveles de privilegios para la gestión del espacio de memoria: usuario y kernel [20]. Para los procesadores de 32 bits el mapa de memoria está dividido en 4 secciones, Figura 21.

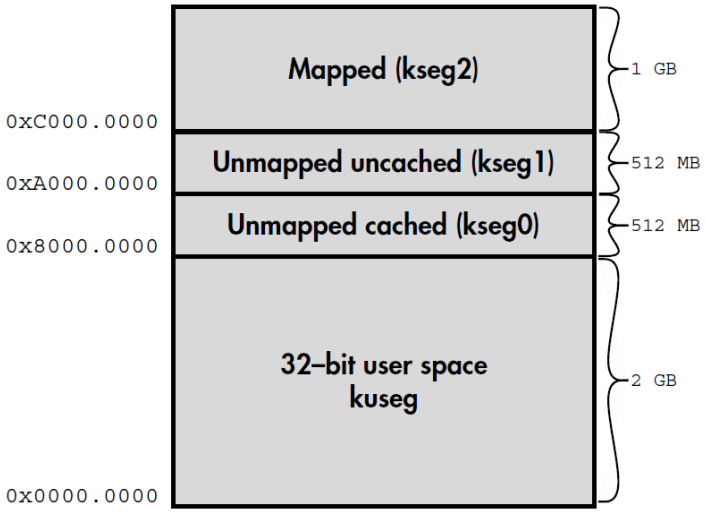


Figura 21. Mapa de memoria de MIPS 32

Tabla 2. Descripción de los segmentos del mapa de memoria de MIPS

Segmento	Función
kuseg 0x0000.0000 – 0x7FFF.FFFF	Es el conjunto de direcciones al que se puede accesar en modo usuario. En procesadores con MMU (unidad de gestión de memoria) las direcciones virtuales siempre serán traducidas.
kseg0 0x8000.0000 – 0x9FFF.FFFF	Estas direcciones son traducidas en direcciones físicas al omitir el bit más significativo de la dirección, por lo que se ocupan los primeros 512 MB de la memoria física. El direccionamiento a esta región se realiza a través de las caché, por lo que antes de acceder al segmento las caches deberán ser inicializadas.
kseg1 0xA000.0000 – 0xBFFF.FFFF	Estás direcciones se mapean a direcciones físicas al omitir los 3 bits más significativos, con lo que nuevamente se accederá a los primeros 512 MB de la memoria física, con la diferencia de que en esta ocasión no se hará uso de la cache. Esta región es la única en el mapa de memoria de MIPS que está garantizada a ser funcional tras el *reset* del sistema. Dentro reside el punto de inicio inicialización (*after-reset starting point*), localizado en la dirección 0xBFC00000. Dentro de esta región está el programa de ROM inicial, y es utilizado para acceder a los registros de los controladores de entrada salida.
kseg2 0xC000.0000 – 0xFFFF.FFFF	Esta área solo es accesible en modo kernel. Se llevan a cabo procesos relacionados con el Sistema Operativo, se accede a través de la traducción de la MMU.

Cada segmento está clasificado como “Mapeado” o “No mapeado”. Las direcciones mapeadas son traducidas por la unidad de gestión de memoria, mientras que las direcciones no mapeadas representan una dirección física. Los registros de los controladores de entrada salida están localizados en el segmento kseg1, por lo cual dichas direcciones no acceden a la MMU, sino que tienen un acceso directo al bus del sistema, Figura 22.

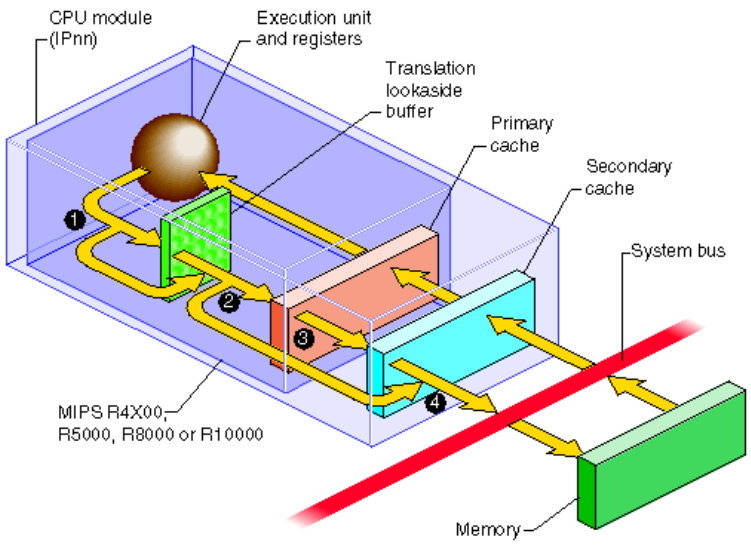


Figura 22. Esquema del núcleo del procesador, MMU y conexión al bus de sistema.

CAPÍTULO 4. DISEÑO DE INTERFACES MAESTRO ESCLAVO.

En este capítulo se describen las especificaciones del estándar de comunicación *on-chip* Wishbone. Primeramente se explica la terminología de las señales, los tipos de bloques que existen y los pasos a realizar para transferencia de información entre bloques.

Una vez expuestos todos los conceptos se explica el diseño de las interfaces mediante el uso de máquinas de estado finito. Finalmente se muestran imágenes de la simulación de las interfaces diseñadas.

4.1 Especificaciones de Wishbone

Wishbone utiliza una arquitectura Maestro/Esclavo, un bloque Maestro es aquel capaz de iniciar una transferencia de información en el bus, mientras que un bloque esclavo únicamente es capaz de responder ante las operaciones solicitadas por un bloque maestro. Un tercer elemento es el bloque SYSCON, el cual gestiona las señales de *reset* y reloj. Los bloques se comunican entre sí a través de una interface de interconexión denominada INTERCON, Figura 23.

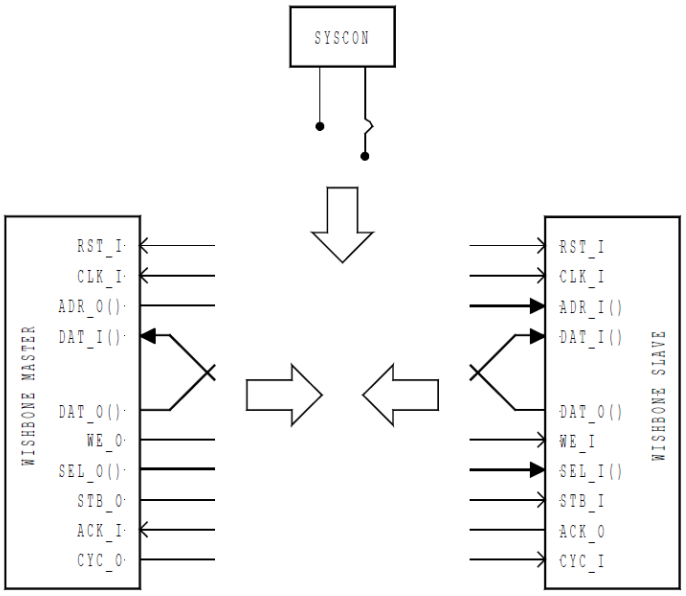


Figura 23. Elementos del estándar Wishbone.

El estándar de comunicación Wishbone define una serie de reglas y recomendaciones para la conexión entre dos o más dispositivos. Las reglas son enunciados definidos en el manual de especificaciones de Wishbone [15] que deben ser acatadas de manera obligatoria para que dicho bloque pueda ser denominado *Wishbone compatible*, las recomendaciones no deben seguirse de manera obligatoria.

En las Tablas 3 - 6 se describen las distintas señales que deben tener los bloques maestros y esclavos de la red de interconexión.

Tabla 3. Señales del Módulo SYSCON

Señal	**Descripción**
CLK_O	Es la señal de reloj del sistema, al ser Wishbone un protocolo de transferencia síncrono esta señal que coordina todas las actividades del bus. Es una señal de salida para SYSCON y es conectada a la señal CLK_I de los bloques MASTER y SLAVE.
RST_O	Es también una señal de salida generada por SYSCON. Forza a todos los elementos de Wishbone a reiniciar. Se conecta a la entrada RST_I de las mismas.

Tabla 4. Señales comunes para bloques maestro y esclavo.

Señal	Descripción
CLK_I	Esta señal de entrada coordina todas las actividades del bloque.
RST_I	Señal de *reset*, proveniente del bloque SYSCON. Restablece los valores de los registros internos a un valor predeterminado.
DAT_I [ ], DAT_O[ ]	Son arreglos de entrada y de salida, respectivamente, utilizados para transmitir las palabras binarias. El tamaño de las mismas está definido por el tamaño de puerto del bus, el cual puede tener un máximo de 64 bits. En el caso de este diseño estás señales son de 32 bits.

Tabla 5. Señales del bloque maestro

Señal	Descripción
ACK_I	Señal de reconocimiento de entrada, cuando está en estado alto, indica la terminación normal de un ciclo de bus.
ADR_O[ ]	Señal de direcciones, dependiendo del dispositivo puede indicar una dirección de memoria o el acceso a un registro específico. Es utilizada por el decodificador de direcciones para activar el esclavo correspondiente y para el caso de este diseño es de 32 bits.
CYC_O	Señal de ciclo, cuando está en estado alto indica que un ciclo de bus valido está en progreso. La señal se mantiene en estado alto durante la duración de todo el ciclo (pueden enviarse más de una palabra).
SEL_O	Señal de selección de byte. Indica en que parte de la palabra se espera el dato en la señal DAT_I().
STB_O	Señal de salida de strobe. Indica una transferencia de único dato.
WE_O	Señal de write enable, indica si la operación actual es de lectura (estado bajo) o escritura (estado alto).

Tabla 6. Señales específicas bloques esclavo

Señal	Descripción
ACK_O	Cuando está señal se pone en alto indica la terminación de un ciclo de escritura o lectura, se conecta a la entrada ACK_I del maestro.
ADR_I[ ]	Indica los registros a los cuales el MASTER desea leer o escribir.
CYC_I	Entrada de ciclo, cuando está en estado alto indica que un ciclo de bus está en progreso. En sistemas donde no existe un árbitro (sólo un MASTER) está señal se conecta con CYC_O del MASTER.
SEL_I	Indica donde se encuentra el dato válido dentro del arreglo DAT_I en ciclos de escritura, y donde se debe de presentar el dato válido en DAT_O para ciclos de lectura.
STB_I	Señal de *strobe*, cuando está en estado alto indica al esclavo que ha sido seleccionado para la transferencia, y en respuesta debe de poner en alto la señal ACK_O.
WE_I	Señal de *write enable*, indica si el presente ciclo es de lectura o escritura.

4.2 Ciclos de lectura y escritura de Wishbone

A continuación se describen las distintas etapas que se siguen al realizar lecturas y escrituras simples y en bloque, según las reglas de Wishbone.

4.2.1 Ciclo de lectura simple.

Flanco de Reloj 0:

- El MASTER presenta una dirección valida en el bus.
- El MASTER niega WE_O para indicar un ciclo de lectura
- El MASTER presenta un valor de SEL_O para indicar donde se espera el dato.
- El MASTER pone en alto la señal CYC_O para indicar el inicio del ciclo
- El MASTER pone en alto la señal STB_O para indicar el inicio de la primera fase.

Flanco de Reloj 1:

- El SLAVE decodifica las entradas y pone en alto la señal ACK_I
- El SLAVE presenta un dato valido en su salida DAT_O (arreglo DAT_I del MASTER)

Flanco de Reloj 2:

- El MASTER captura el dato en su entrada DAT_I
- El MASTER niega la señal STB_O y CYC_O para indicar que ha recibido DAT_I y finalizar el ciclo de lectura.

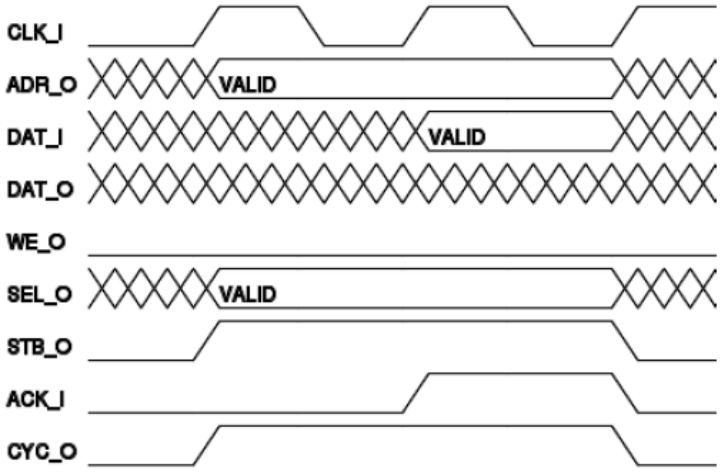


Figura 24. Ciclo de lectura simple

4.2.2 Ciclo de escritura simple.

Flanco de Reloj 0:

- El MASTER presenta una dirección valida en el bus.
- El MASTER pone en alto WE_O para indicar un ciclo de escritura
- El MASTER presenta un valor de SEL_O para indicar donde se presentará el dato a escribir.
- El MASTER pone en alto la señal CYC_O para indicar el inicio del ciclo
- El MASTER pone en alto la señal STB_O para indicar el inicio de la primera fase.

Flanco de Reloj 1:

- El SLAVE decodifica las entradas y pone en alto la señal ACK_I
- El SLAVE captura el valor de DAT_O

Flanco de Reloj 2:

- El MASTER captura el dato en su entrada DAT_I
- El MASTER niega la señal STB_O y CYC_O para indicar el fin de la operación y del ciclo.

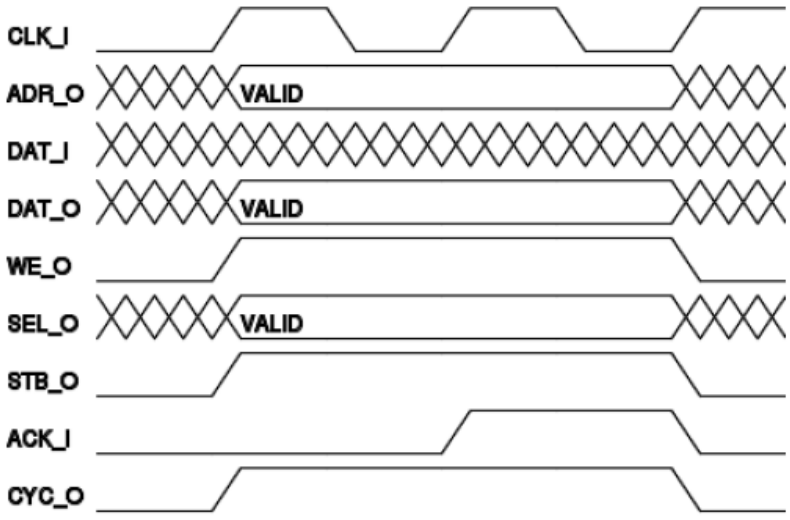


Figura 25. Ciclo de escritura simple

4.2.3 Ciclo de lectura/escritura en bloque

Hay ocasiones en las que el bloque MASTER requiere enviar más de un dato a uno o varios dispositivos. En estos casos se utiliza un ciclo de lectura/escritura de bloque cuya principal diferencia con los ciclos simples es el MASTER mantiene la señal CYC_O en alto hasta que termine todas las transferencias. En las Figuras 26 y 27 es posible observar un ejemplo.

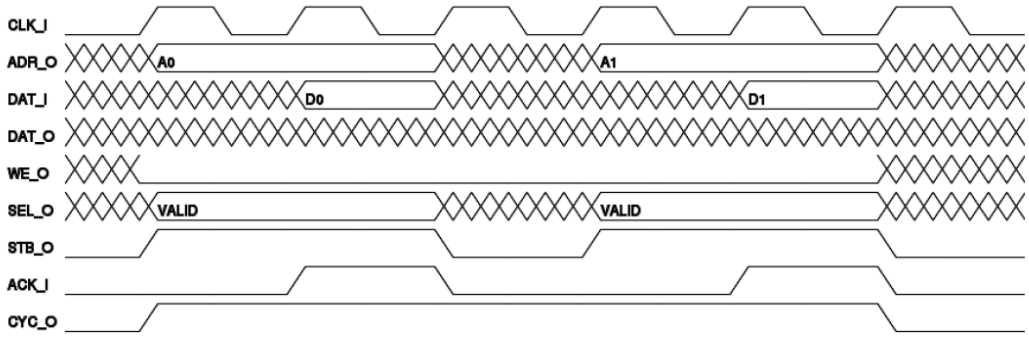


Figura 26. Ciclo de lectura en bloque

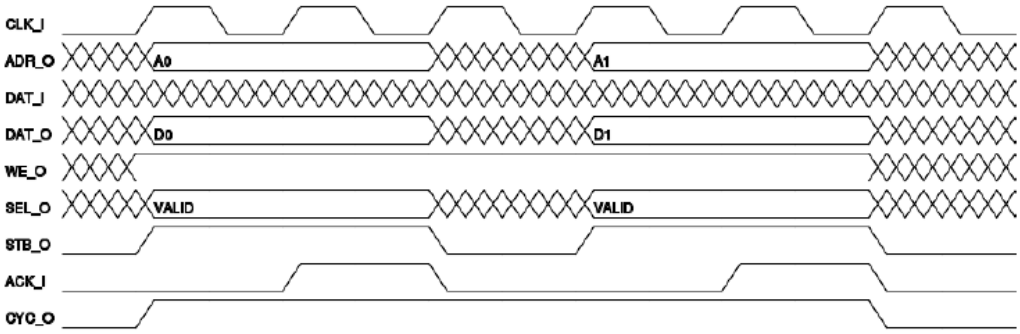


Figura 27. Ciclo de escritura en bloque

4.3 Diseño de las interfaces Maestro/Esclavo

A partir de las especificaciones de la sección 4.2 se realizó el modelado de máquinas de estados finitos para la generación de las señales de escritura y lectura. El modelado mediante máquinas de estado permite que se muestren de manera explícita las condiciones que habilitan el valor de cada salida a partir de las entradas.

4.3.1 Diseño de Interfaz Maestro.

Únicamente existen dos bloques maestros en el bus implementado: el procesador Lagarto y el bloque Bridge. Ambos bloques utilizan una estructura de datos FIFO donde son almacenadas todas las transacciones pendientes.

En la FIFO se almacenan la dirección, el dato de salida y el tipo de operación a realizar (lectura o escritura), la máquina de estado lee estos datos y los envía a través del bus, siguiendo las especificaciones de Wishbone.

La Figura 28 muestra la estructura de la máquina de estados finitos que modela el comportamiento del bloque maestro.

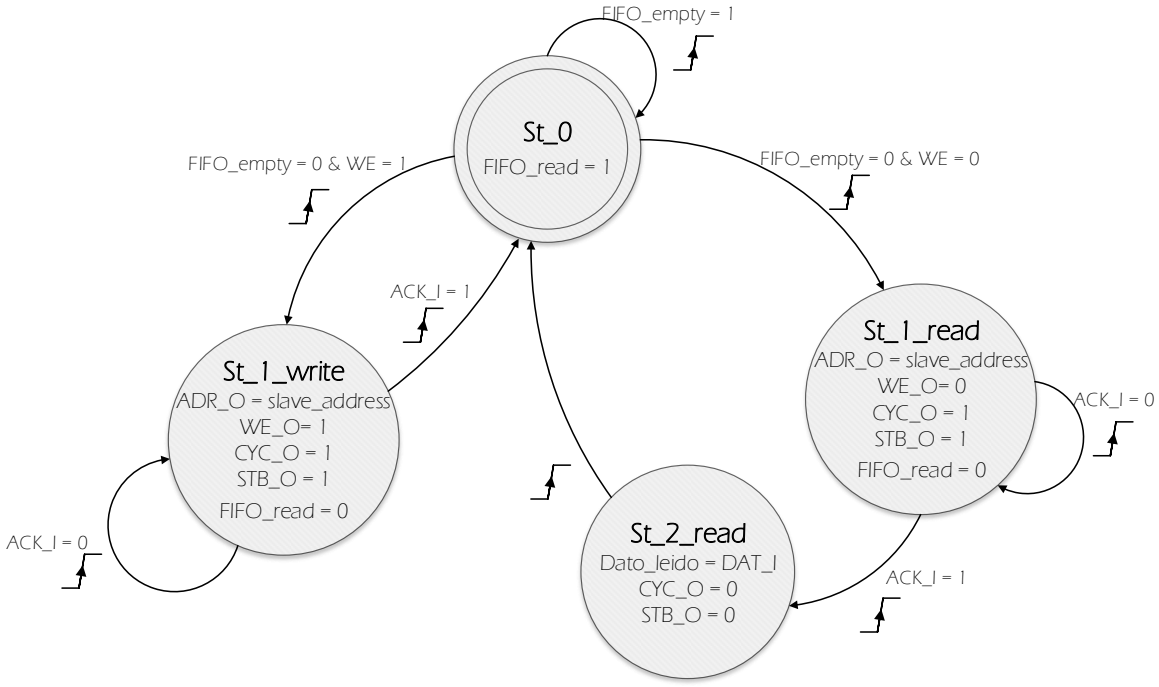


Figura 28. FSM de la interfaz Maestro

.

Descripción de estados del Maestro.

St0: La FSM se mantiene en este estado mientras la FIFO se encuentre vacía, la señal de lectura de la misma se mantiene en alto, una vez que la FIFO sea escrita el dato es inmediatamente leído y se traslada a St1_read o St1_write dependiendo del valor de WE.

St1_read: Aquí se inicia la comunicación con el bloque esclavo al enviar al bus las señales de inicio de ciclo, la dirección y la selección de operación WE = 0. El maestro se mantiene en este estado hasta que la señal de reconocimiento ACK_I es recibida.

St2_read: La señal de reconocimiento ACK_I ha sido recibida. Se ponen en bajo las señales de inicio de ciclo (CYC_O y STB_O) y se captura el dato de entrada. Posteriormente regresa al estado inicial St0 con el siguiente flanco de subida del reloj.

St1_write: Aquí se inicia la comunicación con el bloque esclavo al enviar al bus las señales de inicio de ciclo, la dirección y la selección de operación WE = 1. Una vez que se recibe la señal de reconocimiento el maestro regresa a su estado inicial.

4.3.2 Diseño de Interfaz Esclavo

La máquina de estado para los bloques esclavo es más sencilla. Consta únicamente de tres estados, mismos que se definen a continuación.

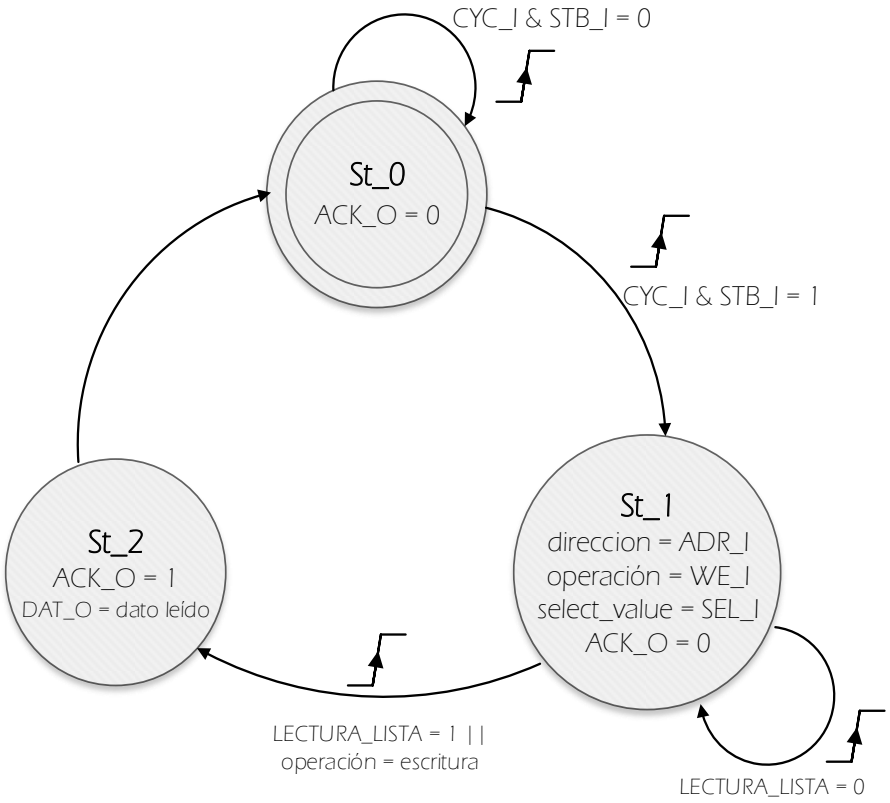


Figura 29. FSM de la interfaz esclavo

.

Descripción de estados del esclavo.

St0: La FSM se mantiene en este estado hasta que se recibe la señal CYC_O y STB_O

St1: Se reconoce el tipo de operación a realizar mediante la evaluación de WE_I. Si se trata de una escritura, se capturan el dato de entrada y la dirección y se hace la transición a St2.Si se trata de una lectura, la FSM permanece en St1 hasta que la señal interna LECTURA_LISTA se pone en alto.

St2: Se responde al Maestro la recepción de la información al poner en alto la señal ACK_O, en si se trató de una operación de lectura se establece también el valor de DAT_O.

Algunos bloques esclavo tienen entradas y salidas adicionales que no forman parte de las señales de Wishbone, por ejemplo la salida de interrupción del temporizador, o las señales RxD y TxD del bloque UART, ver capítulo 5.

4.4 Simulación de comunicación punto – punto.

La forma de comunicación más básica es la comunicación punto a punto, es un caso particular de la topología de bus compartido en donde solamente existe un bloque maestro y un bloque esclavo, en la Figura 30 es posible observar la conexión entre ambos bloques.

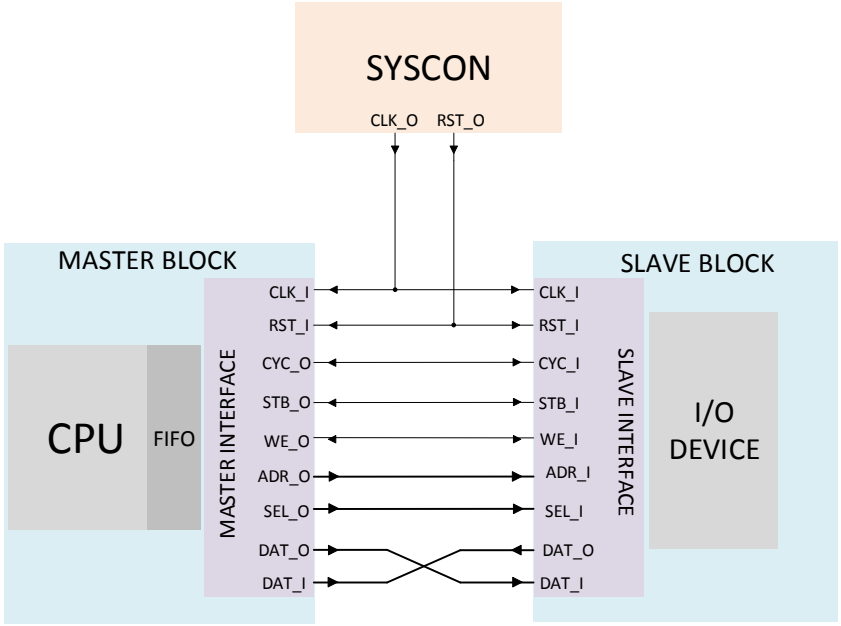


Figura 30. Conexión punto - punto

La Figura 31 muestra el resultado de la simulación de la comunicación punto-punto. Es posible observar las señales de entrada y salida de los bloques, en este caso se está realizando una operación de escritura de bloque (dos palabras), es posible observar como la señal CYC_O se mantiene en alto hasta que se han enviado los datos 0x41 y 0x42.

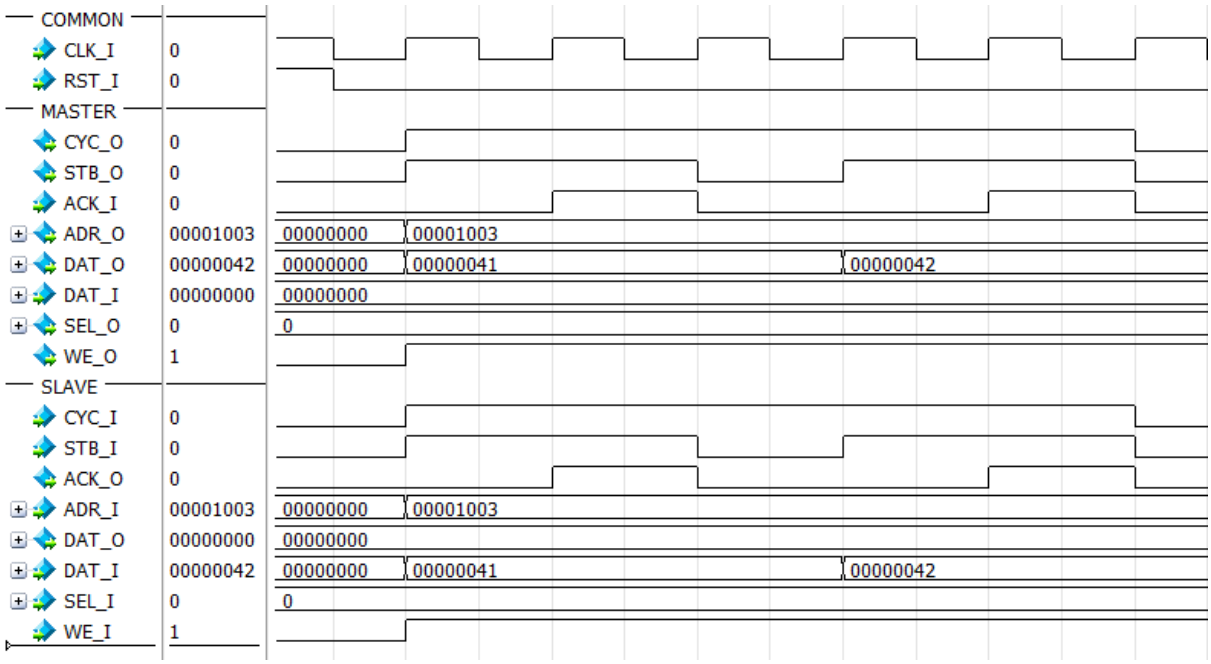


Figura 31. Operación de escritura en bloque en comunicación punto - punto

CAPÍTULO 5. DISEÑO DE LOS DISPOSITIVOS DE ENTRADA SALIDA.

En este capítulo se explica el diseño de los distintos bloques que forman parte del bus de comunicación implementado. En el caso de los controladores de dispositivo se explican las características importantes desde el punto de vista de hardware (bloques que integran el controlador y señales de entrada salida) y desde el punto de vista de software (registros de configuración y de estado).

Cada bloque se conecta a la interfaz de Wishbone diseñada en el capítulo 4. Si bien cada controlador tiene distintos modos de funcionamiento la interfaz con el bus es la misma. Una vez explicado el diseño de cada bloque se procede a mostrar algunas señales de simulación del mismo, así como las características de implementación del mimo, como los elementos lógicos y la frecuencia máxima.

5.1 Bloque Maestro Lagarto.

El núcleo del procesador, la unidad de gestión de memoria y procesador de sistema (coprocesador 0) se interconectan entre sí como se muestra en la Figura 32, juntos forman el CPU. La comunicación del CPU con dispositivos de entrada salida se realiza a través del bus de sistema, la conexión a dicho bus se realiza mediante una interfaz que convierte la convierte las señales provenientes del procesador en señales que cumplan un estándar de comunicación, en este caso, Wishbone.

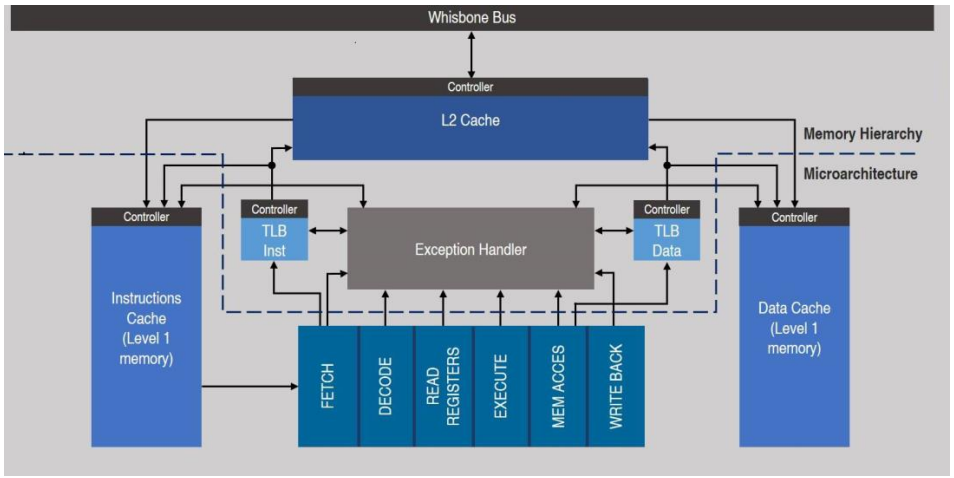


Figura 32. Estructura del CPU Lagarto I

5.1.1 Arquitectura de la interfaz.

En la Figura 33, es posible observar la interconexión entre los bloques *Core* y MMU, se muestran las señales de mayor importancia para las operaciones de lectura y escritura.

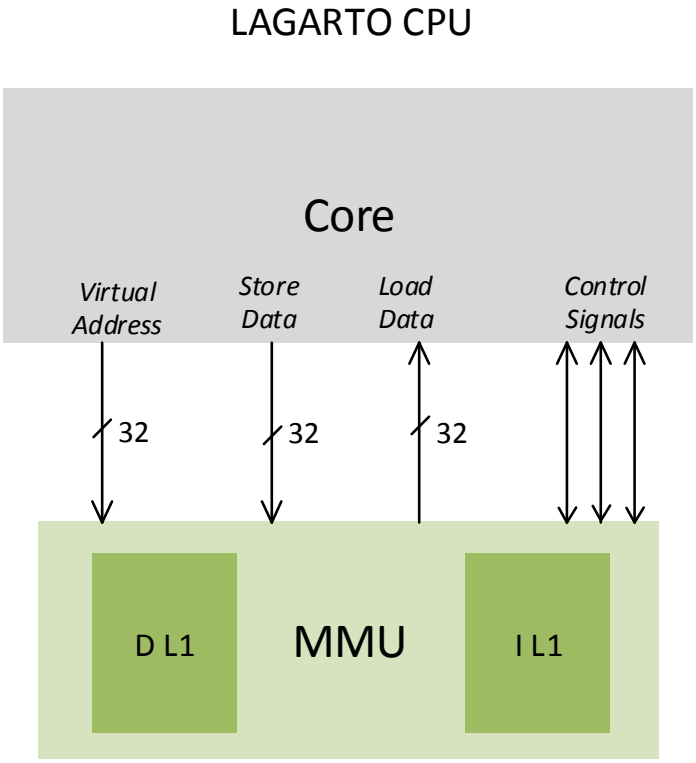


Figura 33. Conexión original entre el núcleo del procesador y la MMU

Debido a que en el set de instrucciones de MIPS no se tienen instrucciones especiales de entrada salida, el acceso a los registros de los controladores de hardware se realiza mediante instrucciones de carga y almacenamiento en memoria. Esto significa que no todas las operaciones de almacenamiento harán uso de la MMU, si no que en algunos casos se tendrá un acceso directo al bus del sistema, ver capítulo 3.

El factor que determina cuando se accede al bus o a la MMU es el segmento de memoria al que pertenece la dirección virtual indicada por la instrucción. El acceso a los registros de los controladores se ubica en el segmento kseg1 del mapa de memoria de MIPS por lo que fue necesario modificar la conexión entre el núcleo del procesador y la MMU como se muestra en la Figura 34.

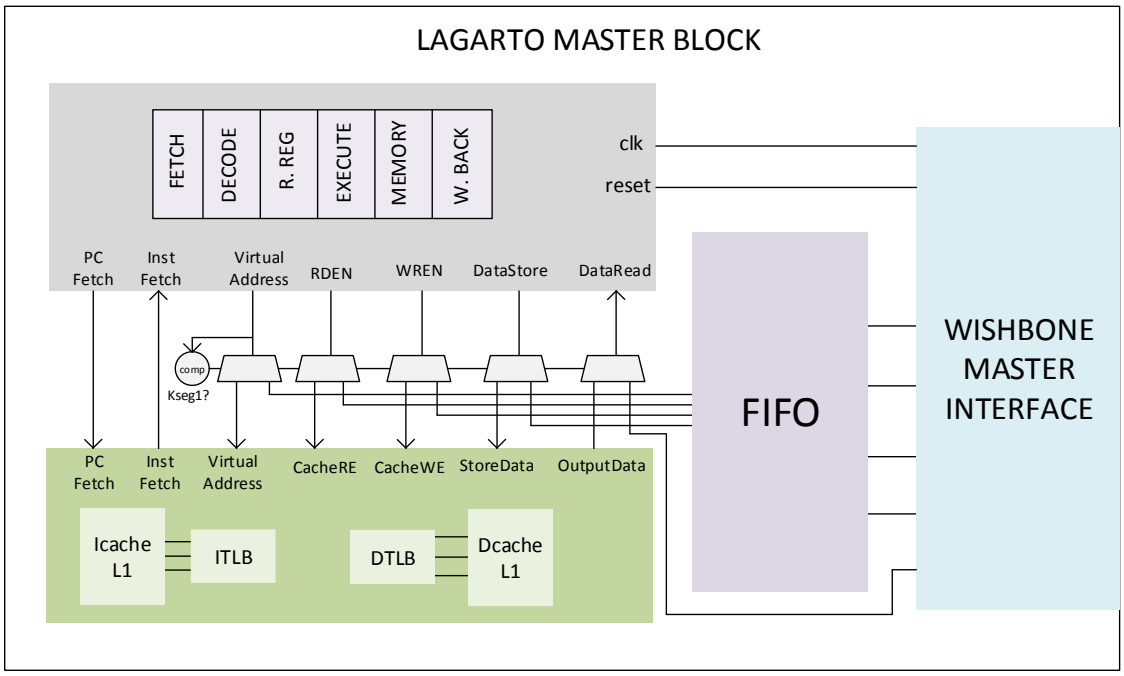


Figura 34. Configuración del bloque Maestro Lagarto

A partir de un comparador se identifica si la dirección virtual procedente del procesador pertenece al segmento kseg1, de ser así, las señales de escritura y lectura son enviadas a la interfaz Wishbone, la cual solicitará acceso al bus y enviará la información a un bloque esclavo.

La etapa de acceso a memoria (*memory access)* consume un ciclo de reloj en ser ejecutada, mientras que una operación de escritura simple en Wishbone toma al menos tres ciclos de reloj. Debido a esta disparidad fue necesario incluir una memoria FIFO en la cual se almacenen los comandos pendientes en lo que el procesador continúa su ejecución.

Las operaciones de almacenamiento no detienen al procesador[1], el comando se almacena en la FIFO y espera su turno a ser enviado por el bus. En el caso de las operaciones de carga el paro del procesador es imperativo, una vez se envía la dirección del registro que se desea leer el procesador debe permanecer inactivo hasta que la interfaz maestro le entregué el dato deseado.

[1] A no ser que exista un error que genere a su vez una interrupción de hardware.

5.1.2 Simulación y resultados de implementación

En la Figura 35 se muestra el comportamiento de las señales de salida del procesador, es posible observar los datos de salida en *StoreData* y la dirección destino *ALUVirtualAddress*, las señal WREN indica una operación de escritura y RDEN una operación de lectura. Estas señales son transformadas en las señales del estándar Wishbone (CYC, STB, ADR_O, etc) por medio de la interfaz maestro descrita en el capítulo 4. La Figura 36 muestra las señales de salida de la interfaz maestro.

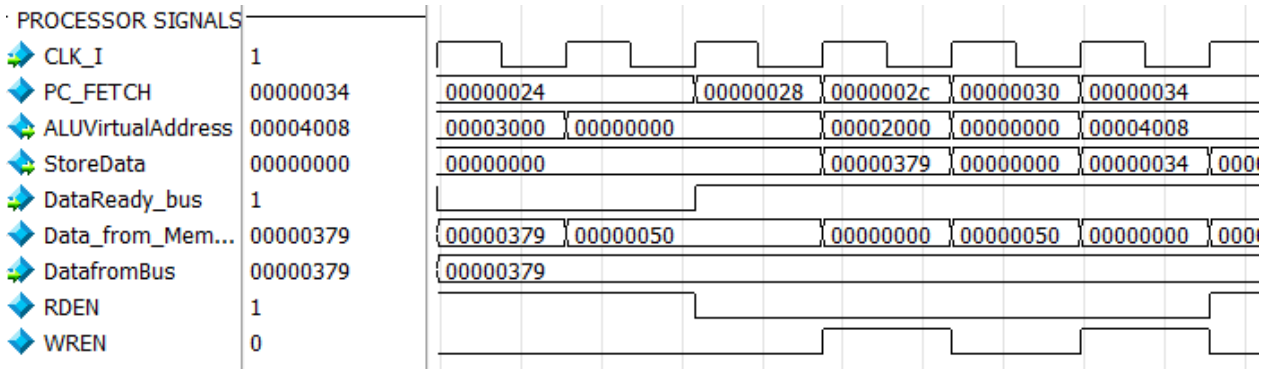


Figura 35. Señales de simulación del procesador.

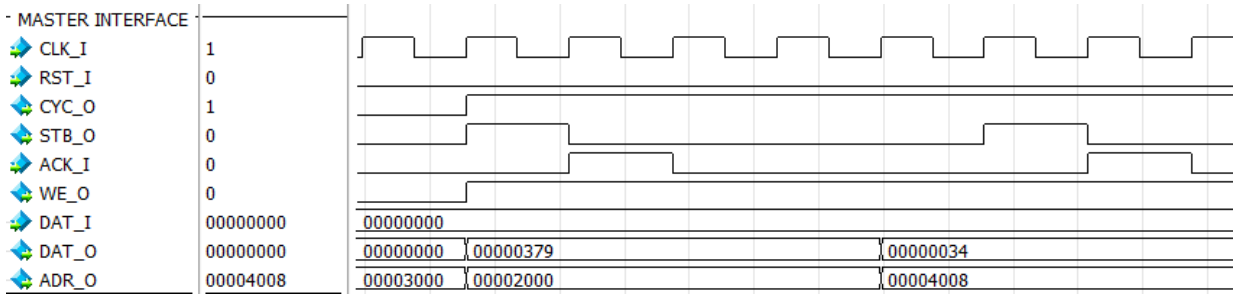


Figura 36. Señales de la interfaz Maestro del procesador

En la Tabla 7 se encuentran los recursos utilizados para la implementación del bloque y la frecuencia máxima del mismo.

Tabla 7. Recursos utilizados y frecuencia máxima

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	3711	79.4%
Dedicated Logic Registers	1201	64.6%
Memory Bits	785408	97.9%
LUT-Only LCs	2510	89.1%
Register-Only LCs	144	39.6%
LUT/Registers LCs	1057	70.7%
Frecuencia máxima	46.61 MHz	-

5.2 Temporizador.

5.2.1 Arquitectura del temporizador

Un temporizador consiste de tres componentes:

- Pre-scaler
- Contador de N bits.
- Registro de comparación de N bits.

El pre-scaler permite ajustar la frecuencia que alimenta al contador. Este componente toma la señal de entrada del temporizador, la divide por un valor (fijo o configurable) y envía dicha señal al contador. El contador aumenta su valor en cada pulso de entrada y cuando iguala el valor del registro de comparación se genera una señal de salida. El valor de comparación es establecido por el procesador.

El bloque temporizador implementado en este trabajo basa su funcionamiento en el temporizador 8253 de Intel [21]. Este dispositivo está compuesto por un contador de 16 bits pero la interface con el sistema es de 8 bits. Para el caso de este trabajo únicamente fue implementado el modo de operación "Interrupt on Terminal Count". En la Figura 37 se muestra la estructura del bloque temporizador.

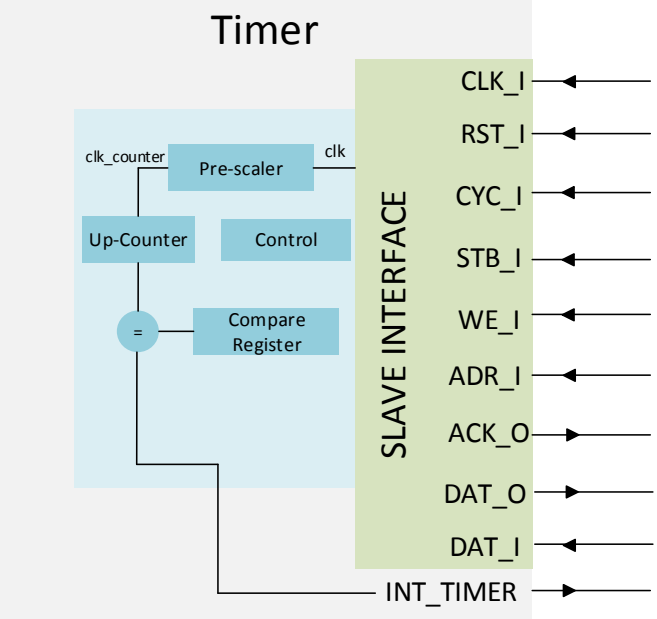


Figura 37. Temporizador con interfaz Wishbone

Para la configuración del temporizador es necesario escribir dos registros: el registro de comparación y el de modo. En la Tabla 8 se muestran el offset correspondiente para cada registro, desde el punto de vista de programación el temporizador equivale a dos registros en la memoria de datos del procesador.

Tabla 8. Registros del Temporizador

Dirección	Operación	Registro
0	R/W	MODE REGISTER
4	R/W	COMPARATOR REGISTER

5.2.2 Simulación y resultados de implementación.

En la Figura 38 se muestra una simulación del temporizador. Es posible observar que inicialmente se realizan tres operaciones de escritura: carga del modo de operación, carga del byte bajo y del byte alto. Posteriormente el contador comienza a incrementar con cada pulso de reloj.

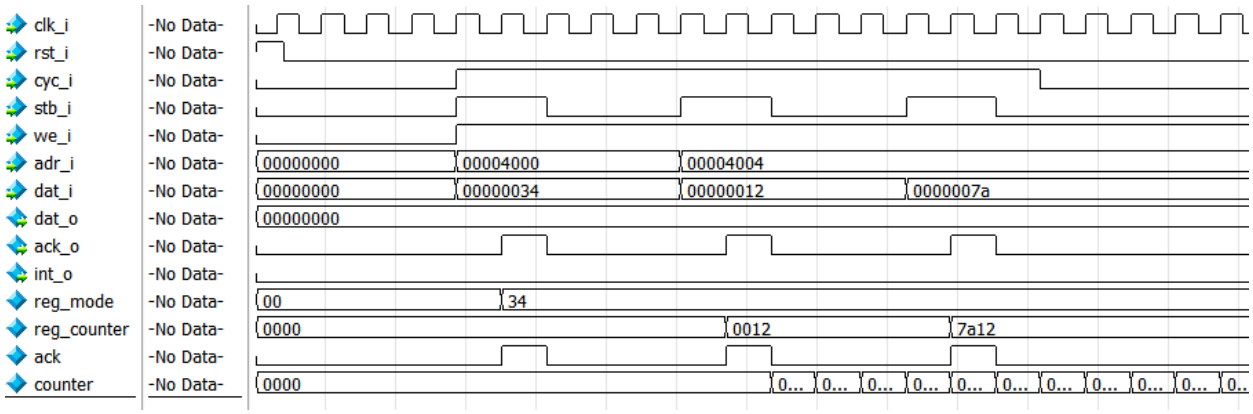


Figura 38. Simulación del temporizador con interfaz Wishbone

En la Tabla 9 se encuentran los recursos utilizados para la implementación de este bloque así como la frecuencia máxima del mismo.

Tabla 9. Recursos utilizados y frecuencia máxima

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	87	1.86%
Dedicated Logic Registers	69	3.7%
Memory Bits	0	0%
LUT-Only LCs	18	0.6%
Register-Only LCs	2	0.5%
LUT/Registers LCs	67	4.48%
Frecuencia máxima	265 MHz	-

5.3 Universal Asynchronous Receiver and Transmitter (UART)

La comunicación serial UART sigue siendo una de las manearas más ampliamente utilizadas para la transmisión y recepción de datos en sistemas embebidos. El diseño de este módulo se basa en el descrito en [22].

5.3.1 Arquitectura del UART

En la Figura 39 se muestra el modelo de hardware. Es posible observar la interfaz con las líneas de transmisión RxD y TxD, y con el CPU, señales de reloj, control de escritura y lectura y selección de chip.

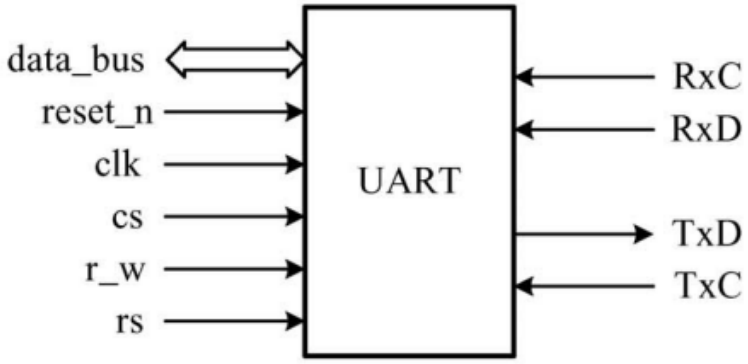


Figura 39. Modelo de hardware del bloque UART

La línea de transmisión de datos TxD es empleada para enviar los datos hacia otro dispositivo, y la línea RxD es utilizada para recibir datos, ambos de forma serial (bit a bit). Ambas líneas están controladas por dos señales de reloj locales: TxC y RxC, respectivamente. La frecuencia de reloj RxC is usualmente más alta que la de TxC, por un factor de 4, 16, o 64.

El modelo de programación, Figura 40, consiste de cuatro registros: el *receiver data Register (RDR), transmitter data Register (TDR), status Register (SR) y control Register (CR).* RDR y SR son registros de sólo lectura, mientras que TDR y CR son de sólo escritura. Las señales RS y RW permiten la selección y operación del registro a leer/escribir mientras que la señal se utiliza como señal de habilitación del chip. Los posibles valores de las señales RS y RW dieron lugar a las distintas direcciones de los registros como parte del modelo de programación del bloque de entrada salida, Tabla 10.

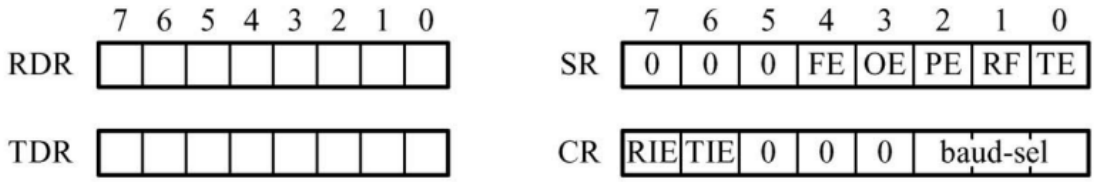


Figura 40. Modelo de programación del bloque UART

Tabla 10. Registros del bloque UART

Dirección	Operación	Registro
0	R	Status Register (SR)
4	W	Control Register (CR)
8	R	Read Data Register (RDR)
12	W	Transmitter Data Register (TDR)

Un dispositivo UART típico está compuesto de tres partes principales: un transmisor, un receptor, y un generador de tasa de baudios.

5.3.2 Transmisor

El componente central del transmisor es un registro de corrimiento que continuamente envía un valor de bit a través de la salida TxD. La Figura 41 muestra los componentes básicos de un transmisor típico. El transmisor se compone de un registro TDR (transmitter data Register), un TSDR (transmitter shift data Register), un bit bandera TE (TDR empty flag) un generador de paridad y un circuito de control.

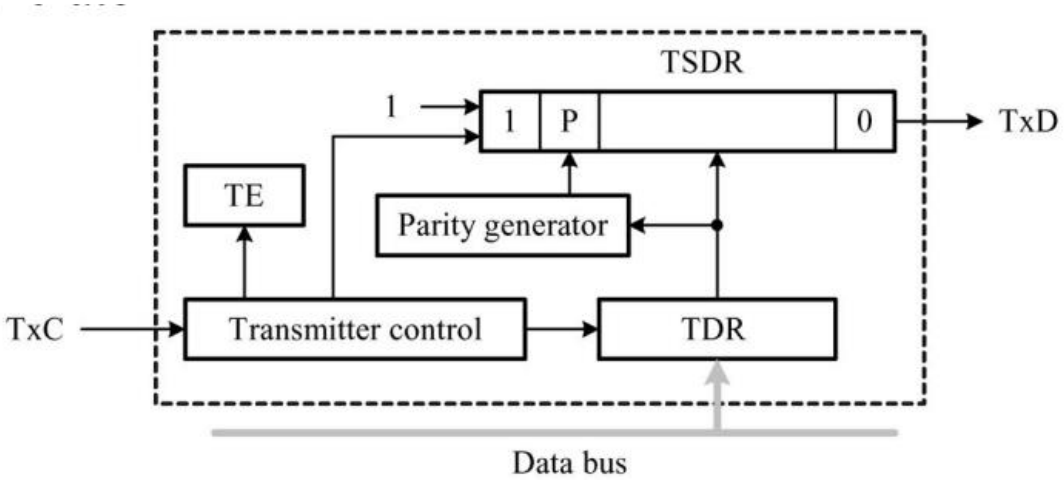


Figura 41. Transmisor del UART

El dato a transmitir se carga en el registro TDR, el cual posteriormente se transfiere al registro TSDR, en el cual es continuamente rotado a cada ciclo del reloj TxC. Los siguientes diagramas de flujo muestran los detalles de operación desde el punto de vista del UART y del CPU.

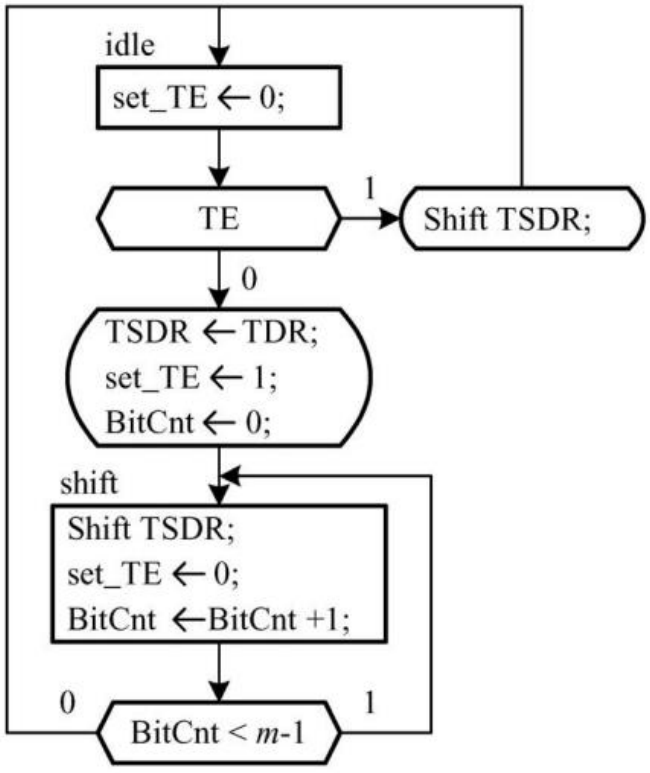


Figura 42. Diagrama de flujo del transmisor.

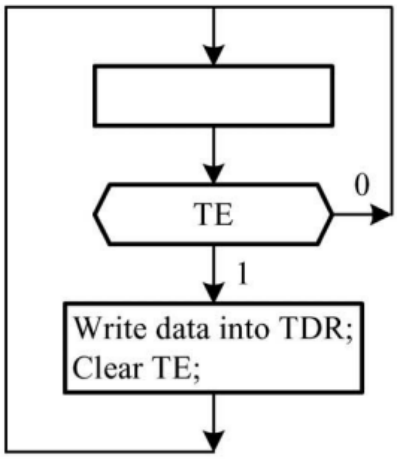


Figura 43. Operaciones a realizar por el CPU para la transmisión.

5.3.3 Receptor

Este componente también está centrado en un registro de corrimiento, el cual muestrea el valor de la línea de entrada RxD. Está compuesto por el registro RDR, el RSDR (receiver shift data Register), RDR full flag (RF), un circuito de control y otras banderas. La Figura 44 muestra la composición de este componente.

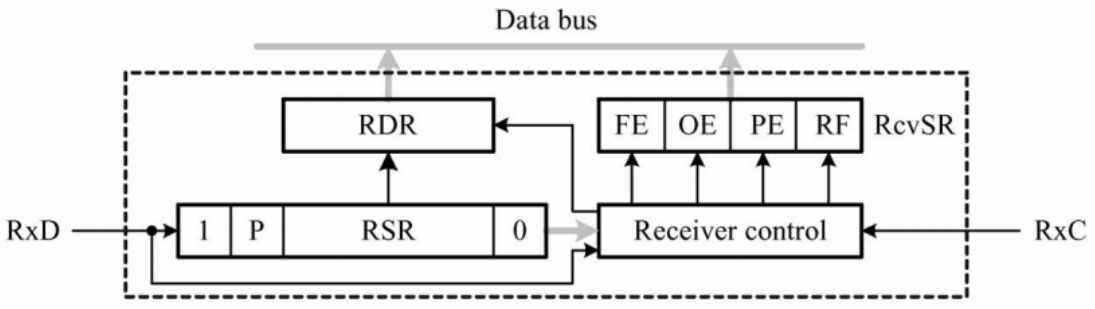


Figura 44. Receptor del UART

Hay cuatro banderas asociadas con el receptor: frame error (FE), overrun error (OE), parity error (PE) y RDR full (RF).

- La bandera FE indica que hay un error con la configuración de la palabra recibida, el bit de parada es 0 en lugar de 1.
- La bandera OE indica que un nuevo byte es recibido antes de que el dato anterior haya sido leído.

- La bandera PE indica que la paridad recibida es errónea
- La bandera RF se pone en alto cada que un nuevo byte se extrae del registro RSDR y se carga en RDR.

Los diagramas de flujo de la Figura 45 y 46 muestran el funcionamiento más detallado del receptor, desde el punto de vista del UART y del CPU.

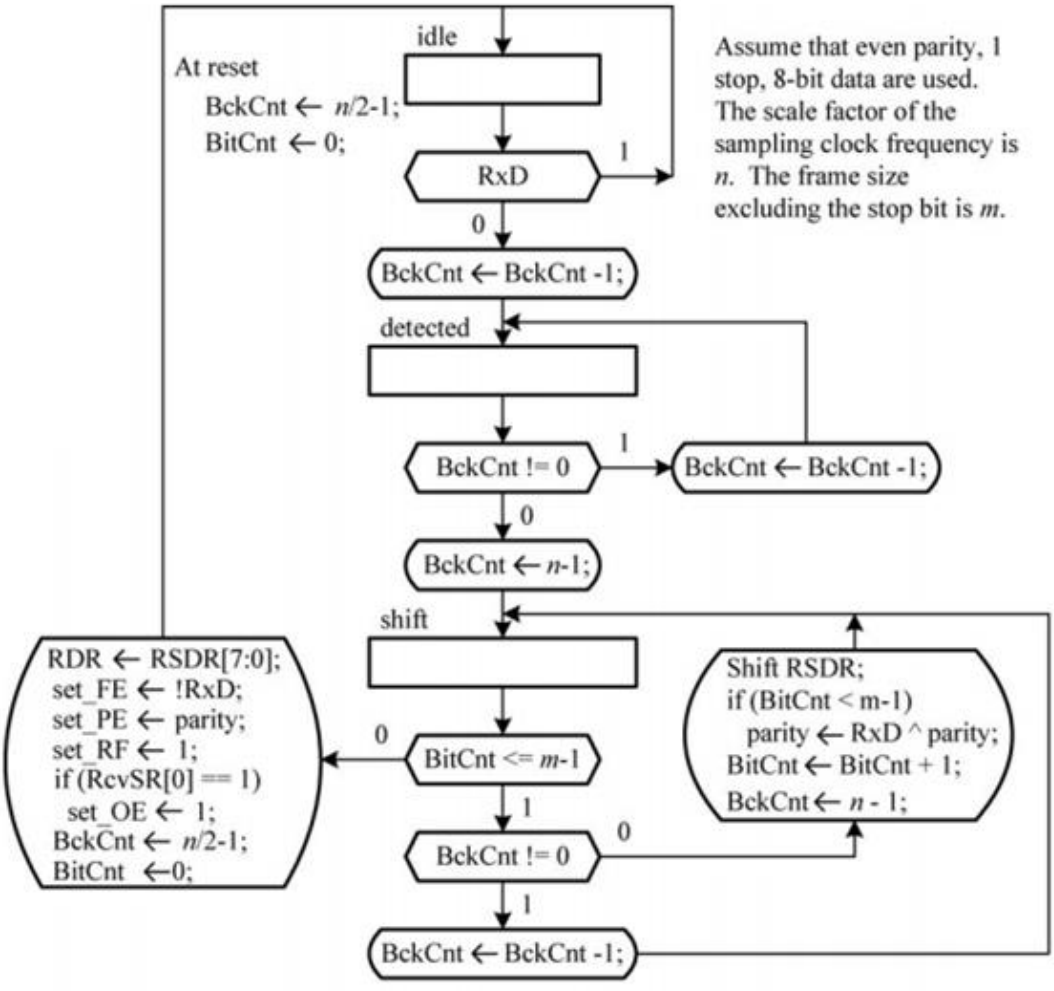


Figura 45. Diagrama de flujo del receptor del UART.

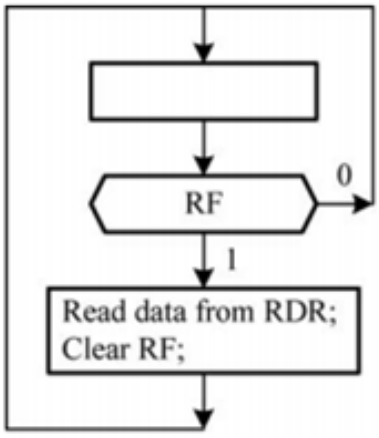


Figura 46. Operaciones del CPU para la recepción.

5.3.4 Generador de Baud-Rate.

Este módulo es muy importante en el módulo de comunicación porque genera las señales de reloj para el transmisor y receptor, TxC y RxC respectivamente.

La Figura 47 muestra la estructura del generador de baud-rate, el cual consiste de un contador prescaler modulo-M, un contador y un multiplexor, este último permite la selección de la velocidad de comunicación deseada a través del registro de control CR.

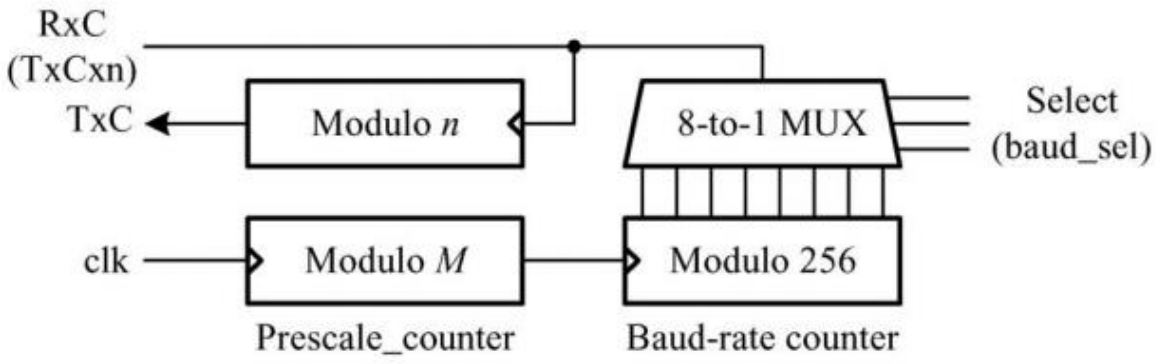


Figura 47. Estructura del generador de baud-rate.

5.3.5 Integración del UART al bus del procesador.

El proceso para generar un módulo UART compatible con Wishbone fue utilizar dos máquinas de estados finitos. Debido a que la transmisión de datos es considerablemente

más lenta que la comunicación interna del bus, se diseñó una máquina de estados finitos la cual recibe los datos del maestro y envía las correspondientes señales de reconocimiento, para posteriormente almacenar dichas palabras en una FIFO. Una segunda máquina de estados fue utilizada para ir cargando al registro TDR con los valores de la FIFO, cada que hubiera disponibilidad en el transmisor. La Figura 48 muestra la configuración interna del bloque UART.

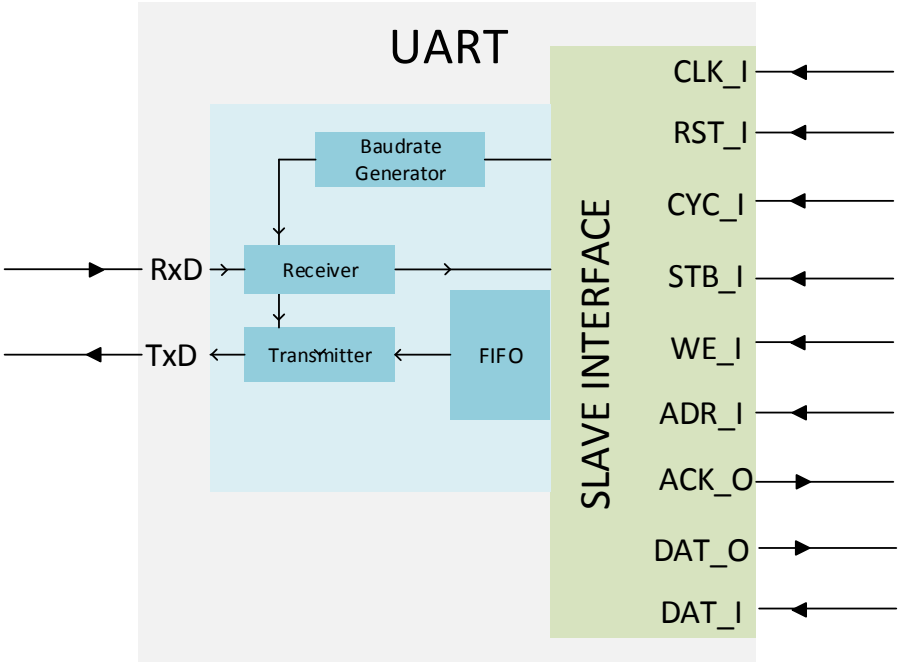


Figura 48. Estructura del bloque esclavo UART

5.3.6 Simulación y resultados de implementación.

En la Figura 49 se muestra el resultado de simulación tras realizar una escritura del carácter ASCII ‘A’ para posteriormente realizar una lectura del registro SR.

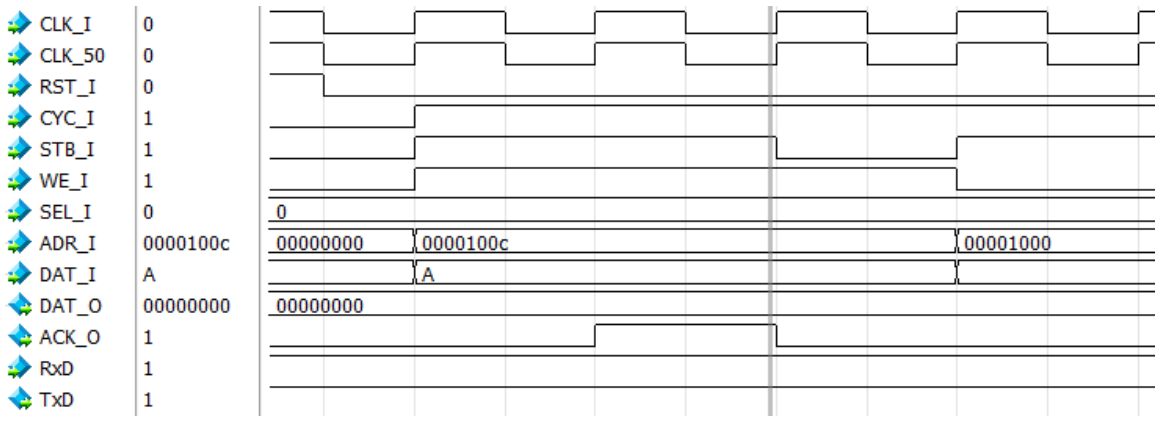


Figura 49. Simulación del bloque UART

En la Tabla 11, se muestran los recursos utilizados y la frecuencia máxima del bloque UART diseñado.

Tabla 11. Recursos utilizados y frecuencia máxima

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	176	3.7%
Dedicated Logic Registers	125	6.7%
Memory Bits	640	0.08%
LUT-Only LCs	51	1.8%
Register-Only LCs	25	6.9%
LUT/Registers LCs	100	6.7%
Frecuencia máxima	69 MHz	-

5.4 Controlador de Interrupciones Programable (PIC).

Este dispositivo basa su funcionamiento en el circuito 8259 de Intel [23]. Como se mencionó en el capítulo 3, el controlador de interrupciones es el componente que recibe las señales de interrupción de todos los dispositivos de entrada salida de tal manera que cuando uno o más dispositivos requieran de la atención del procesador el controlador de interrupciones indique al procesador la existencia de una interrupción.

5.4.1 Arquitectura del bloque

Los principales registros internos del 8259 son el IRR (Interrupt Request Register) y el ISR (In Service Register). El IRR almacena todas las peticiones de interrupción pendientes; el ISR almacena todas las interrupciones que están siendo atendidas en un momento dado.

La lógica de control gestiona la lectura y escritura de los registros internos, además de incluir el bloque que resuelve la prioridad, el cual es el encargado de decidir cuál de las interrupciones debe de atenderse primero. Este bloque sirve también para gestionar la inicialización del dispositivo.

La Figura 50 muestra el diagrama a bloques del sistema.

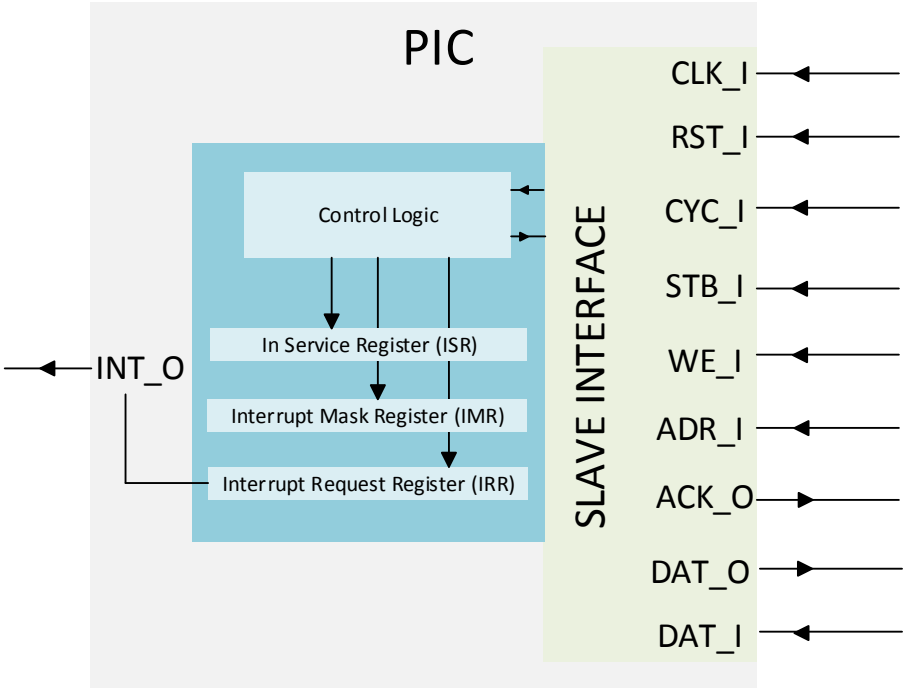


Figura 50. Diagrama a bloques del controlador de interrupciones

5.4.2 Funcionamiento general del PIC

A continuación se enumeran las etapas por las que el PIC se sigue ante una interrupción.

1. Una o más líneas de IR son activadas por los periféricos, lo que pone a 1 el correspondiente bit del IRR.
2. El PIC evalúa la prioridad de estas interrupciones y solicita la interrupción a la CPU.
3. Cuando la CPU reconoce la interrupción realiza una operación de lectura para saber cuál interrupción debe ser atendida.
4. El PIC envía el código de la interrupción por atender y regresa a su estado inicial

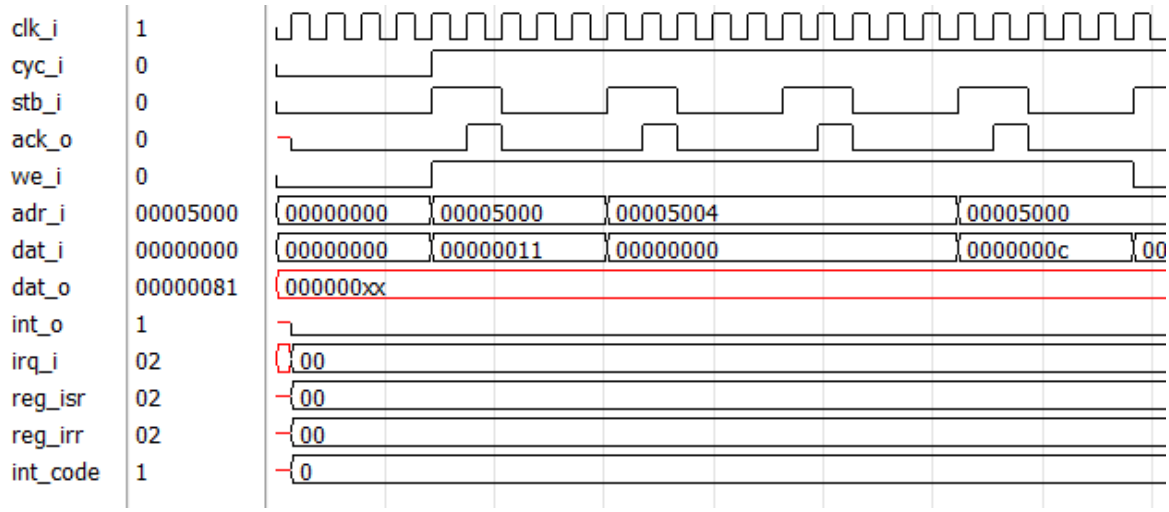
El modelo de programación del controlador de interrupciones consta de cinco registros: RC, IRR, ISR, IMR y el código de interrupción (INT_CODE) que es generado cuando se recibe una interrupción de Hardware, pero únicamente se cuenta con dos valores de offset, el acceso a cada registro dependerá no solamente del valor de dirección si no del modo en el que sea configurado el dispositivo, dicha configuración es dependiente del Registro de Comando (RC), Tabla 12.

Tabla 12. Registros internos del controlador de interrupciones

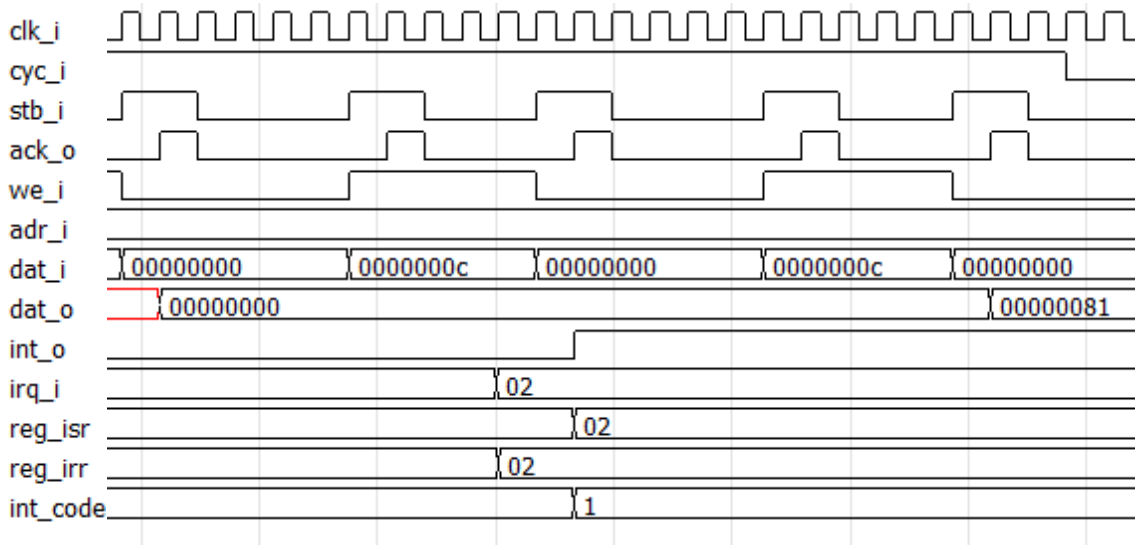
Modo	offset	Operación	Registro
IDLE	0	W	RC
	4	W/R	IMR
POLL	0	R	INT_CODE
	4	-	-
IRR	0	R	IRR
	4	-	-
ISR	0	R	ISR
	4	-	-

5.4.3 Simulación y resultados de implementación.

Las Figura 51 (a) y (b) muestran el resultado de la simulación del controlador de interrupciones. En (a) se muestran los comandos para la inicialización del dispositivo. En (b) se realiza la lectura del mismo para detectar si ha ocurrido alguna interrupción, una vez que se detecta una se procede a leer el código de interrupción correspondiente.



(a) Inicialización



(b) *muestreo*

Figura 51. Señales de simulación del controlador de interrupciones

La Tabla 13 muestra las estadísticas de la implementación del módulo.

Tabla 13. Recursos utilizados y frecuencia máxima de operación

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	113	2.4%
Dedicated Logic Registers	58	3.1%
Memory Bits	0	0%
LUT-Only LCs	55	2%
Register-Only LCs	1	0.3%
LUT/Registers LCs	57	3.8%
Frecuencia máxima	308.55 MHz	-

5.5 General-Purpose Input and Output

En cualquier sistema con un microprocesador es necesario tener puertos de entrada y salida para tener interacción con el mundo físico. Un puerto es un espacio físico sobre el cual una señal puede ser transferida entre dos módulos. Existen puertos paralelos, en los que un conjunto de líneas se escriben o leen al mismo tiempo, y puertos seriales, en los cuales la información es transmitida un bit a la vez. Un puerto puede también ser de entrada, de salida o bidireccional.

5.5.1 Arquitectura del GPIO.

Un puerto GPIO es un dispositivo que puede ser configurado como puerto de entrada o de salida. Consta de un conjunto de pines de entrada salida y dos registros: el registro de dirección de datos *DDR*, y el registro de puerto *PORT*. El primero determina cuales pines son entrada y cuáles de salida y el segundo establece los valores de los pines de salidas y permite la lectura de los pines de entrada.

El funcionamiento del GPIO es bastante simple, dependiendo del valor de DDR, se establece el valor del puerto PORT_PIN, si DDR[i] vale 1, entonces el pin PORT_PIN[i] será una salida, por lo que se realiza la asignación PORT_PIN[i] = PORT[i]. Si por el contrario, DDR[i] vale 0, entonces PORT_PIN[i] será utilizado como entrada, por lo que se realiza la asignación PORT_PIN[i] = ‘Z’, el software de diseño deduce entonces que ese pin se utilizará como entrada y procede a hacer las configuraciones adecuadas. Cuando se solicite la lectura de PORT_PIN[i] no se leerá el valor de ‘Z’ si no el valor de tensión al que está sometido el pin físico. En la Figura 52 se muestra un esquema lógico del circuito resultante.

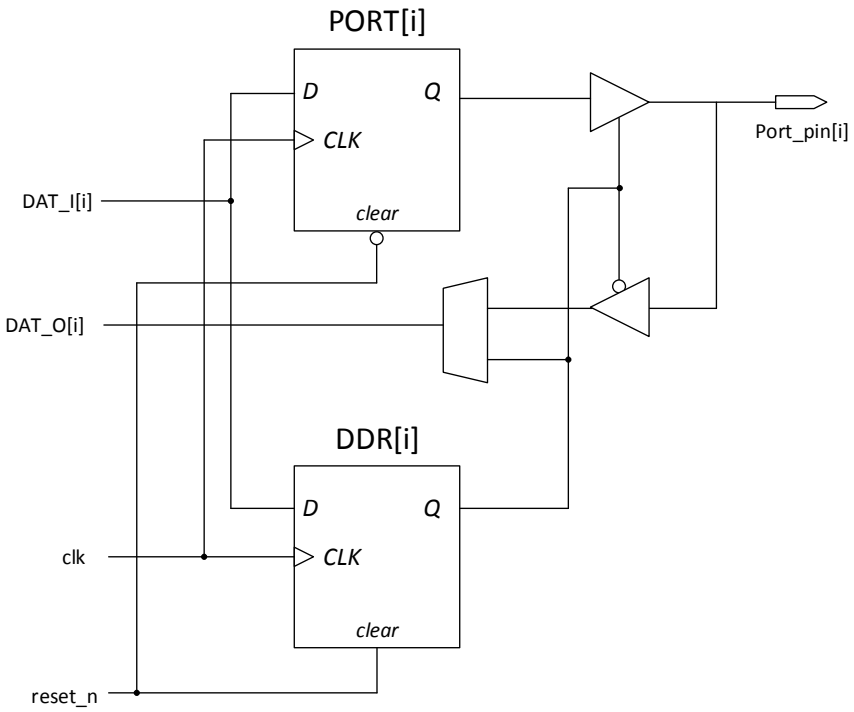


Figura 52. Circuito lógico para cada bit del GPIO.

En la Figura 53 se muestra el modelo de hardware del GPIO, es posible observar las señales de entrada y de salida del bloque.

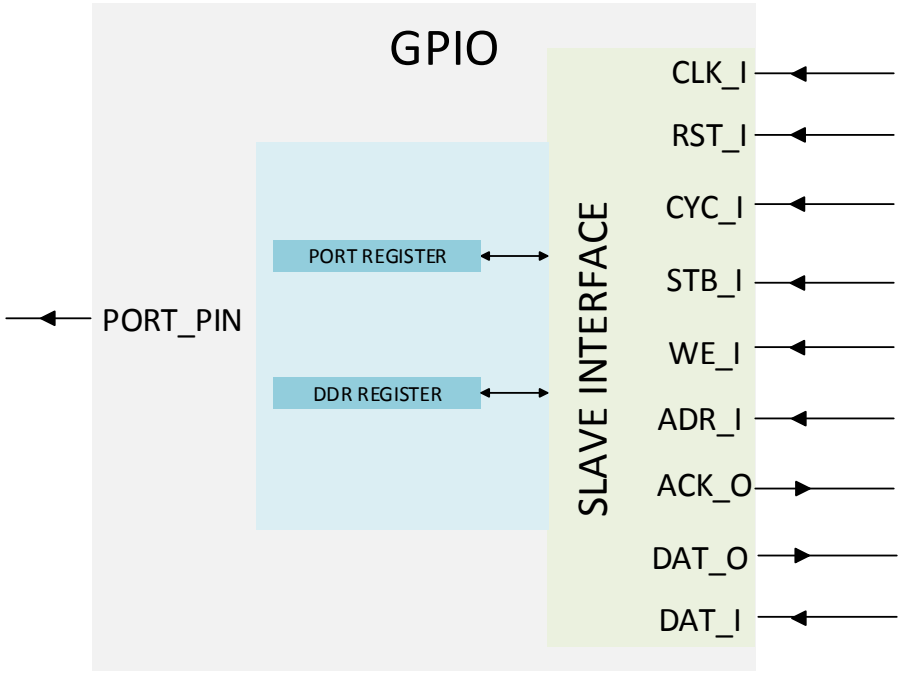


Figura 53. Bloque esclavo GPIO

El modelo de programación consta de los registros DDR y PORT, ambos de 32 bits, la Tabla 14 muestra el valor de offset correspondiente a cada registro.

Tabla 14. Registros del GPIO

Dirección	Operación	Registro
0	R/W	PORT
4	R/W	DDR

5.5.2 Simulación y resultados de implementación.

En la Figura 54 se muestra un ejemplo del uso del módulo GPIO, en el cual se realiza un ciclo de operaciones de lectura y escritura del GPIO. Primeramente se realiza una operación de escritura que modifica el valor del registro DDR, el cual se establece en 0x0000FFFF, con lo cual se configuran los 16 bits más significativos como entrada y los 16 bits menos significativos como salida.

Posteriormente se realiza una lectura del registro PORT, y el resultado obtenido se vuelve a enviar de vuelta al registro PORT, pero esta vez se escribe únicamente en los 16 bits menos significativos. El resultado final es que PORT_PIN tiene el valor replicado en los bits 31 a 16 y los bits 15 a 0.

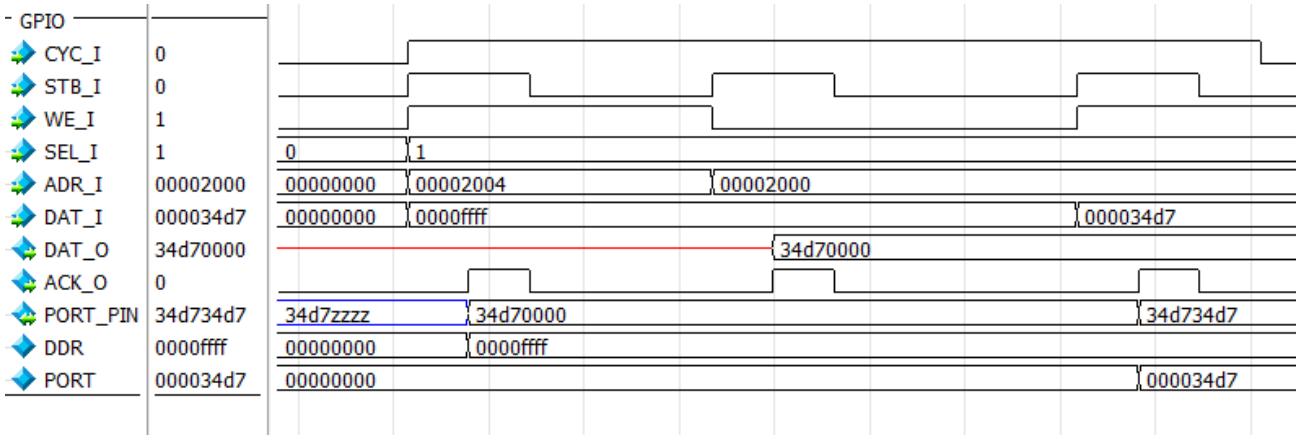


Figura 54. Señales de simulación del bloque GPIO, al finalizar el ciclo, los pines de salida son iguales a los pines de entrada.

En la Tabla 15 se encuentran los recursos utilizados para la implementación de este bloque así como la frecuencia máxima del mismo.

Tabla 15. Recursos utilizados y frecuencia máxima

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	101	2.1%
Dedicated Logic Registers	98	5.2%
Memory Bits	0	0%
LUT-Only LCs	3	0.1%
Register-Only LCs	64	17.6%
LUT/Registers LCs	34	2.2%
Frecuencia máxima	394 MHz	-

5.6 On-chip RAM memory.

Si bien el CPU cuenta con una unidad de gestión de memoria en donde están incluidas memorias de datos e instrucciones, es posible integrar en el bus más bloques de memoria, los cuales pueden ser accedidos mediante instrucciones de almacenamiento y carga.

Este bloque de memoria es el único bloque esclavo que se encuentra en el bus de alta velocidad. La conexión entre las señales de la memoria con las señales de Wishbone es bastante simple, pues no se requiere del uso de FIFO o máquinas de estado adicionales.

La Figura 55 muestra la configuración del bloque.

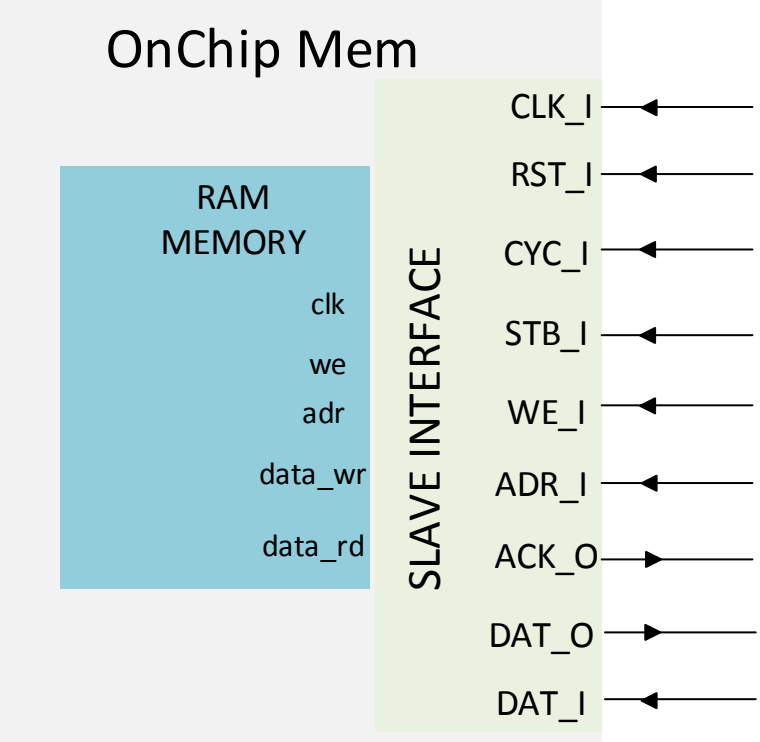


Figura 55. Bloque de memoria *on-chip*

5.6.1 Simulación y resultados de implementación

En la Figura 56 se muestra la simulación de las operaciones de lectura y escritura del módulo.

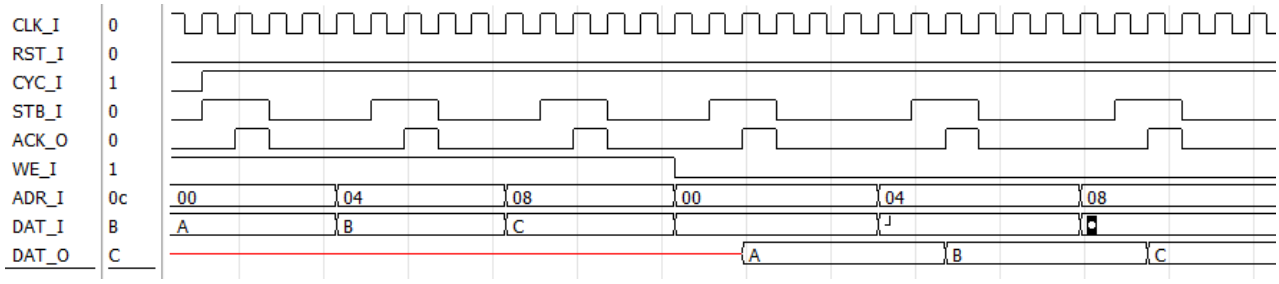


Figura 56. Escritura y lectura de los caracteres A, B y C en la memoria *on-chip*

Es posible configurar el tamaño de la memoria *on-chip* antes de compilar el diseño en Verilog, para esto es necesario asignar un valor de ancho de direcciones y ancho de palabra. Para los datos de la Tabla 16 se utilizó una memoria de 64 x 32 (64 localidades de 32 bits cada una).

Tabla 16. Recursos utilizados y frecuencia máxima

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	4	0.09%
Dedicated Logic Registers	2	0.1%
Memory Bits	2048	0.25%
LUT-Only LCs	2	0.07%
Register-Only LCs	1	0.3%
LUT/Registers LCs	1	0.06%
Frecuencia máxima	487 MHz	-

5.7 Bloque de interconexión Bridge

El bus local del procesador Lagarto tiene una topología de bus jerárquico, por lo que es necesario implementar una forma de comunicar dispositivos con frecuencias de operación distintas.

5.7.1 Diseños con múltiples dominios de reloj.

En muchos sistemas digitales complejos el uso de más de una señal de reloj para la sincronización de eventos, *multiple-clock domain* o *crossing clock domains*, es inevitable. En este tipo de diseños es necesario asegurarse de que las señales y datos que cambian de dominio de reloj están sincronizados de manera adecuada, esto es, que las transiciones de los datos obedezcan los flancos de reloj del dominio en cuestión. Existen tres técnicas que ayudan con este fin: uso de *synchronizers*, *handshaking protocols* y FIFOs[22].

Una FIFO es un arreglo de memoria que es utilizado comúnmente en el diseño de hardware y software como un buffer que permite coordinar el traspaso de datos cuando las tasas de entrada y de salida entre dos módulos son distintas. Puede ser asíncrona, sin señal de reloj, o síncrona, una o dos señales de reloj, como la que se observa en la figura. Este método suele ser el más utilizado cuando el ahorro de área del diseño no es la prioridad más importante.

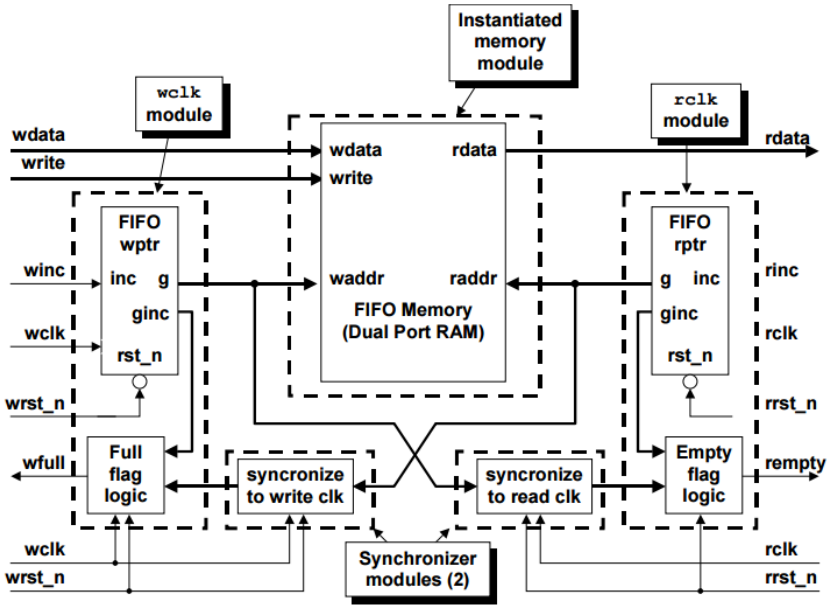


Figura 57. Ejemplo de una FIFO con sistema de reloj dual

5.7.2 Diseño del bloque utilizando FIFO dual.

El principal componente del Bridge es una estructura de datos FIFO, la cual trabaja con líneas de reloj independientes para la lectura y la escritura. El Bridge forma parte tanto del bus de alta velocidad, en donde es un bloque esclavo, como del bus de baja velocidad, en donde es un bloque maestro. Las señales de escritura y lectura están controladas por dos bloques FSM independientes, estos bloques también se encargan de generar las señales propias del estándar Wishbone para realizar las transferencias, Figura 58.

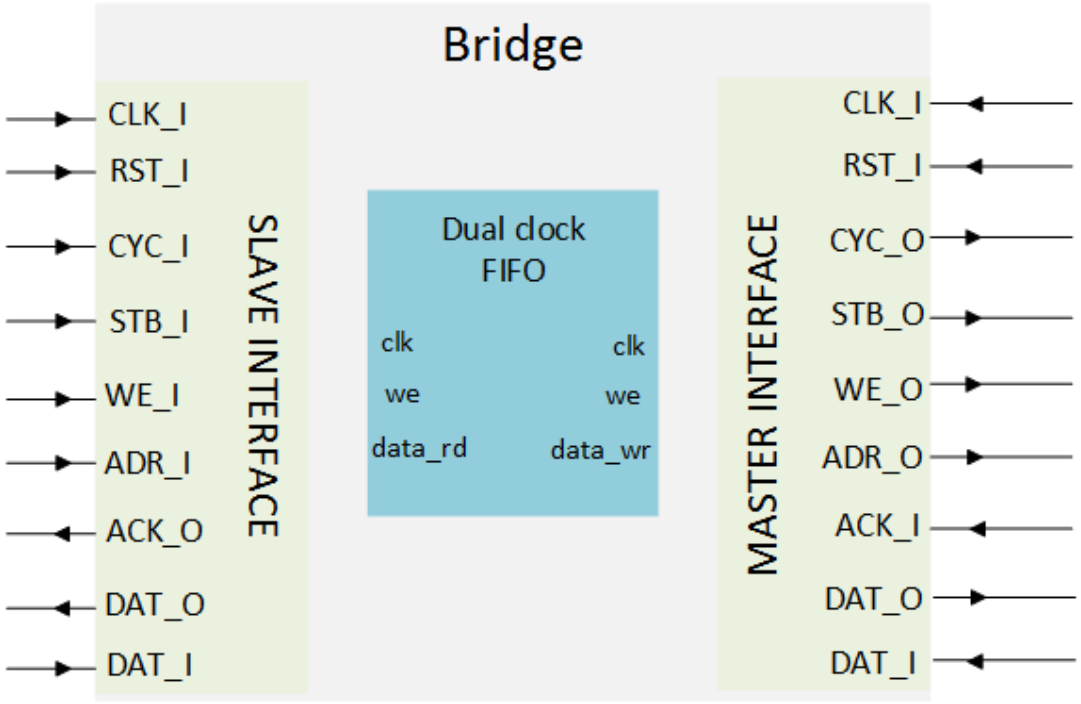


Figura 58. Configuración interna del bloque Bridge

Cabe mencionar que dentro de las especificaciones del estándar Wishbone no se menciona la topología de Bus Jerárquico, por lo que si bien este bloque no forma parte de un esquema típico de comunicación Wishbone, el bloque sigue obedeciendo las reglas del estándar para las interfaces maestro y esclavo.

5.7.3 Simulación y resultado de implementación.

En la Figura 59 se muestra un ejemplo de una operación de escritura y lectura por parte del procesador. El Bridge recibe los datos a operar y comienza a enviarlos por el bus de baja velocidad a una frecuencia más baja. En el caso de las operaciones de escritura, el procesador debe esperar a que el periférico envíe la respuesta, la cual pasa a través del bus y se entrega con la frecuencia de reloj del bus de alta velocidad.

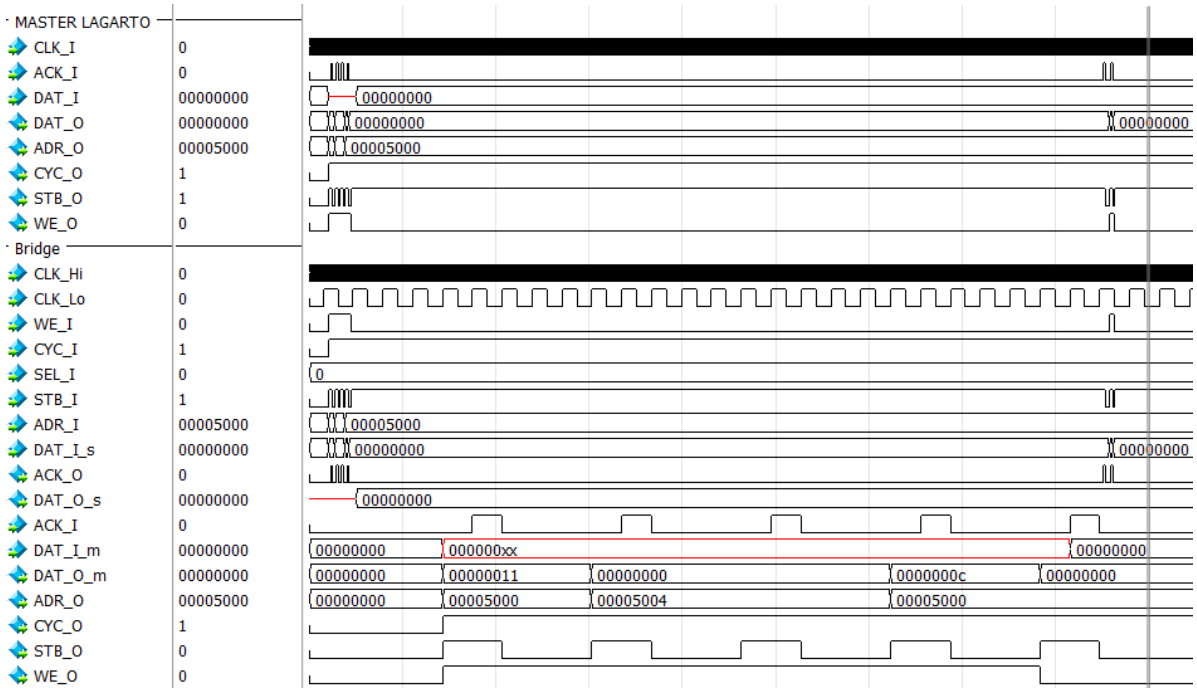


Figura 59. Señales de simulación del bloque Bridge. Es posible observar la interfaz de rápida velocidad y la de baja velocidad.

En la Tabla 17 se encuentran los recursos utilizados para la implementación de este bloque así como la frecuencia máxima del mismo.

Tabla 17. Recursos utilizados y frecuencia máxima

Parámetro	Cantidad	Porcentaje del total del diseño
Elementos Lógicos	209	4.5%
Dedicated Logic Registers	186	10%
Memory Bits	15872	2%
LUT-Only LCs	23	0.8%
Register-Only LCs	73	20%
LUT/Registers LCs	113	7.5%
Frecuencia máxima	160 MHz	-

CAPÍTULO 6. INTEGRACIÓN DEL BUS DE COMUNICACIÓN

En este capítulo se presenta la integración del procesador y los dispositivos de entrada salida descritos en el capítulo 5 en una arquitectura de comunicación mediante bus compartido. Primeramente se describe el diseño de dos elementos adicionales dentro del bus: un bloque comparador de direcciones y un bloque *switch*. Posteriormente se muestra un diagrama del bus jerárquico del sistema y se muestra un ejemplo de uso del mismo.

6.1 Bloques de interconexión adicionales

6.1.1 Comparador de direcciones.

Buses de computadora estándar como PCI y VMEbus, utilizan un sistema de decodificación de direcciones completo (full address decoding). En este método cada módulo esclavo constantemente decodifica la dirección existente en el bus, si por ejemplo, se utiliza un bus de 32 bits, cada esclavo decodifica toda la dirección para saber si debe o no capturar el dato de entrada.

El estándar de comunicación Wishbone propone un sistema de decodificación de direcciones parcial (partial address decoding). En este método cada esclavo decodifica únicamente el rango de direcciones que utiliza, por ejemplo, si el esclavo únicamente contiene cuatro registros, entonces la interfaz de Wishbone utiliza únicamente dos bits de direcciones. Con esta técnica se obtiene una velocidad de decodificación mucho más alta y se utiliza menos área en el chip al eliminar lógica redundante para la decodificación.

Dentro del estándar Wishbone existe un bloque denominado "Comparador de direcciones" (*address comparator)* el cual recibe la dirección de 29-bits proveniente del Maestro (el procesador utiliza direcciones de 32 bits, pero como se mencionó en el capítulo 3 los primeros tres bits son omitidos) y con base a esto pone en alto la señal de selección de esclavo ACMP*x*, donde *x* es el número del esclavo a utilizar. El bloque *Switch* utiliza el valor de las señales ACMP*x* para establecer el valor de la señal STB_O. En la Figura 60 se

muestra la configuración del comparador de direcciones para un bus con cinco esclavos. Las Tabla 18 y 19 muestran un ejemplo del funcionamiento de este bloque.

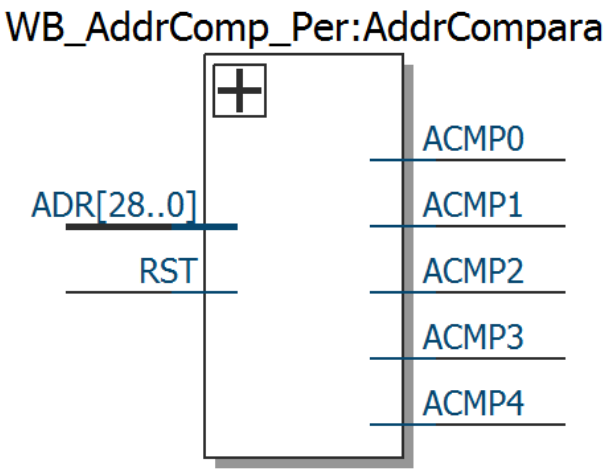


Figura 60. Modelo de hardware del comparador de direcciones

Tabla 18. Rangos de activación ejemplo de las señales ACMP.

Señal	Rango de Direcciones
ACMP0	0x0000.1000 - 0x0000.1FFC
ACMP1	0x0000.2000 - 0x0000.2FFC
ACMP2	0x0000.3000 - 0x0000.3FFC

Tabla 19. Valor de las señales ACMP ante distintas direcciones de entrada

Dirección de entrada	Salida
0x0000.1100	ACMP = { 0, 0, 1 }
0x0000.3200	ACMP = { 1, 0, 0 }
0x0000.0000	ACMP = { 0, 0, 0 }

El rango correspondiente a cada esclavo está definido en un archivo *include* (Direcciones.v). El rango de direcciones para cada dispositivo esclavo es muy importante para la posterior configuración del sistema operativo.

6.1.2 Bloque Switch

El bus implementado utiliza multiplexores para distribuir las señales, mismos que se integran dentro del bloque *Switch*, todos los bloques maestro y esclavo conectan las señales de su interfaz a este bloque, Figura 61.

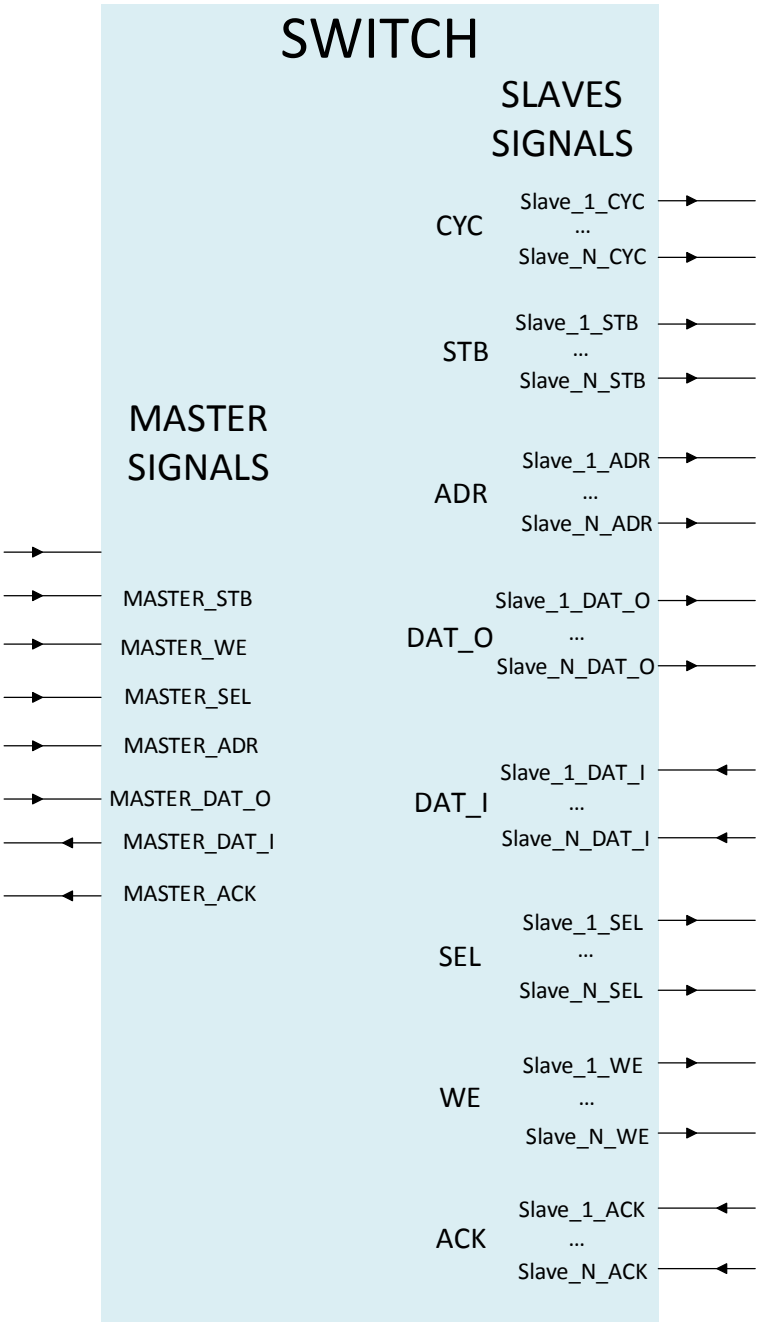


Figura 61. Entradas y salidas del bloque *Switch*

El funcionamiento de este bloque se basa en el siguiente conjunto de ecuaciones booleanas:

$$Master\_DAT\_I = \bigcup_{i=0}^{N} [\ 32\{ACMP_i\}\ \&\ Slave\_i\_DAT\_I\ ]$$

$$Master\_ACK = \bigcup_{i=0}^{N} [\ 32\{ACMP_i\}\ \&\ Slave\_i\_ACK\ ]$$

En donde $\bigcup_{i=0}^{N}$ representa la operación lógica OR acumulada para los esclavos desde 0 hasta N. La respuesta que se envíe al Maestro dependerá del esclavo que fue seleccionado, lo cual se ve reflejado en el valor de la señal ACMP.

Para las señales hacia el esclavo:

$$\begin{aligned} Slave\_i\_WE\_O &= Master\_WE\_I \\ Slave\_i\_ADR\_O &= Master\_ADR\_I \\ Slave\_i\_DAT\_O &= Master\_DAT\_I \\ Slave\_i\_CYC\_O &= Master\_CYC\_I \\ Slave\_i\_SEL\_O &= Master\_SEL\_I \\ Slave\_i\_STB\_O &= Master\_STB\_I\ \&\ Master\_CYC\_I\ \&\ ACMP_i \end{aligned}$$

Es posible observar que los esclavos siempre están recibiendo las señales provenientes del Maestro a excepción de la señal *strobe*, cuya habilitación depende del valor de la señal ACMP, que a su vez depende del rango de direcciones al cual se desee accesar.

6.2 Integración del bus jerárquico.

La Figura 62, muestra la integración de los distintos bloques descritos a lo largo del capítulo 5. El bus de alta velocidad está compuesto por el bloque CPU, que incluye el núcleo del procesador, la unidad de gestión de memoria y donde estará también integrado el procesador de sistema. Como dispositivos esclavo únicamente se encuentran una memoria

on-chip, en este bus puede ser añadido también controladores a otro tipo de memorias *off-chip* como SDRAM, SD-Card, Disco IDE, etc.

El bloque *Bridge* permite hacer la transferencia de datos entre los buses de alta y baja velocidad, es un bloque híbrido con interfaces Maestro y Esclavo, compuesto principalmente por una estructura de datos FIFO con sistema de reloj dual y un sistema de control.

El bus de baja velocidad está compuesto por cuatro distintos módulos esclavos: módulo de comunicación serial UART, un GPIO de 32 bits, un temporizador de 16 bits y un controlador de interrupciones. Cualquier transferencia de datos desde o hacia el bloque CPU debe ser gestionada por el bloque *Bridge*, con excepción de la señal de interrupción del controlador de interrupciones, la cual es conectada a un puerto específico del CPU.

Tanto el bus de baja velocidad como el de alta cuentan con un bloque *switch* y un comparador de direcciones, estos bloques facilitan la integración de nuevos bloques maestro y esclavo, pues únicamente se requiere modificar las ecuaciones booleanas anteriormente descritas.

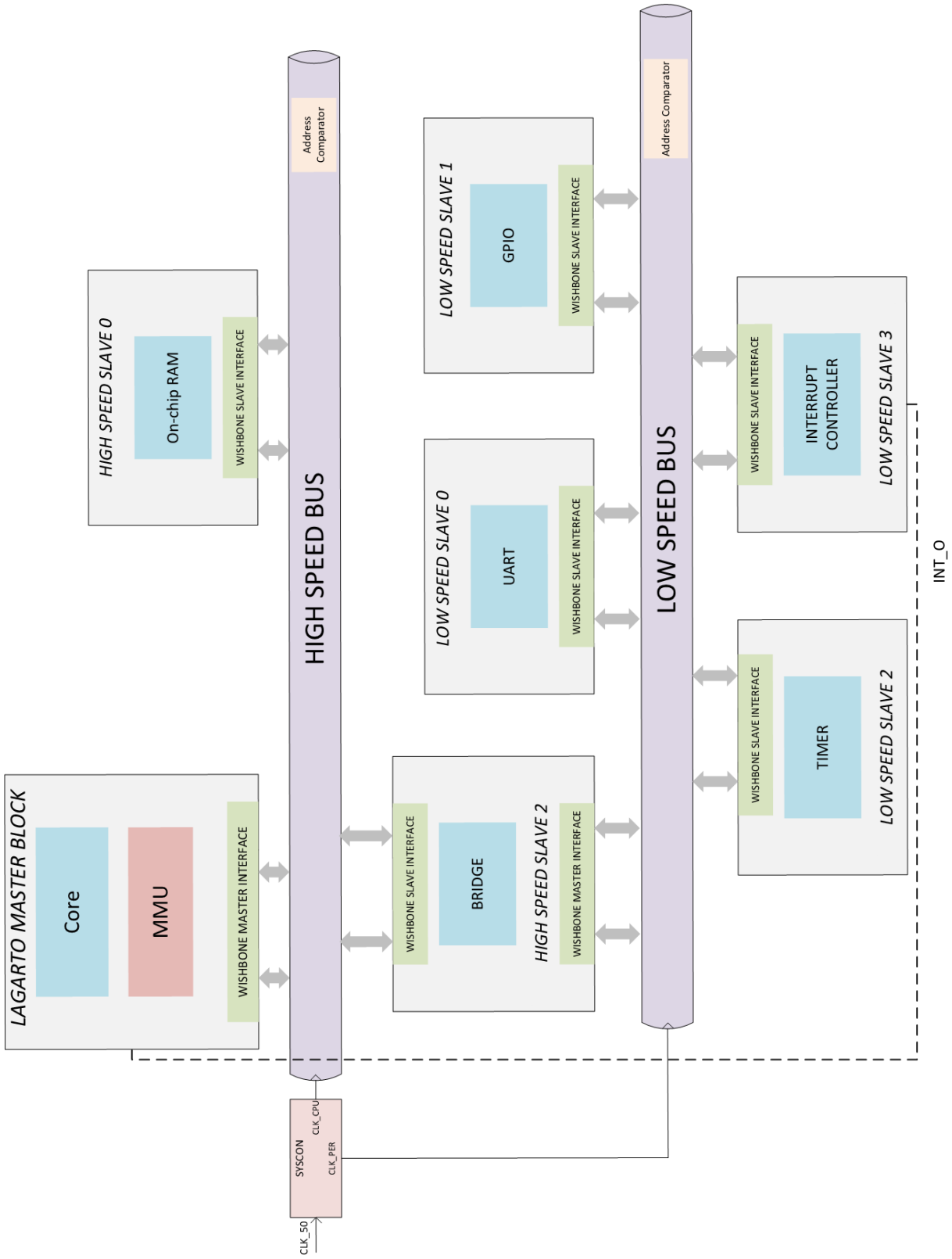


Figura 62. Bus compartido jerárquico diseñado

6.3 Ejemplo de uso del bus.

A continuación se describirá un ejemplo del funcionamiento de la implementación del bus. Se muestra primero la sección de código del manejo de dispositivos y posteriormente se muestran las señales de simulación correspondientes.

El programa realiza lo siguiente:

- Inicialización de variables, se cargan las direcciones base de la Tabla 20 a registros de propósito general.
- Configuración GPIO (primeros 16 pines como entrada, segundos 16 pines como salida).
- Se lee el valor de los pines de entrada y se carga dicho valor a los pines de salida.
- Se escribe y se lee de la RAM *on-chip* tres caracteres ASCII
- Se envían por el puerto UART los caracteres almacenados en la RAM on-chip
- Se realiza la inicialización del PIC y se configura en modo de muestreo (o *polling*).
- Se realiza la verificación de los registros del PIC hasta que se reciba la interrupción de fin de cuenta del Timer.
- Al recibirse la interrupción se realiza un echo del puerto UART (se envían de vuelta los caracteres recibidos).

Para el caso de este ejemplo se utilizaron los rangos de direcciones mostrados en la Tabla 20.

Tabla 20. Rango de direcciones de los dispositivos

Dispositivo	Rango de direcciones
UART	0xA0001000 - 0xA0001FFF
GPIO	0xA0002000 - 0xA0002FFF
Temporizador	0xA0004000 - 0xA0004FFF
Interrupt Controller	0xA0005000 - 0xA0005FFF
Memoria *on-chip*	0xA0007000 - 0xA0007FFF

Inicialización de Variables.

Es la primera sección del programa. Las direcciones base de los dispositivos se almacenan en registros de propósito general, para posteriormente ser utilizados en las operaciones de lectura y escritura.

```
.data

.text
   # Inicialización de variables
   addi $t0,$zero,10
   sll  $t0,$t0,28      # A0000000
   addi $t1,$t0,4096    # A0001000    UART
   addi $t2,$t0,8192    # A0002000    GPIO
   addi $t3,$t0,16384   # A0004000    Timer
   addi $t4,$t0,20480   # A0005000    PIC
   addi $t5,$t0,28672   # A0007000    RAM ON CHIP
```

Figura 63. Inicialización de variables

Uso del GPIO.

Se establecen los primeros 16 pines del puerto como entrada y los segundos como salida, esto se realiza al cargar el valor del registro DDR del dispositivo GPIO (offset 4) con el valor 0x000.FFFF. Posteriormente se procede a leer el valor del registro PORT, se realiza un corrimiento de 16 lugares hacia la derecha y se carga este valor de vuelta al registro PORT.

```
# Configuración GPIO
addi $s0,$zero,65535     #0x0000FFFF
sw   $s0,4($t2)          #Registro DDR
# Lectura GPIO (pines entrada)
lw   $s3,0($t2)
# Escritura GPIO (pines de salida)
srl  $s3,$s3,16
sw   $s3,0($t2)
```

Figura 64. Configuración y uso de GPIO (código)

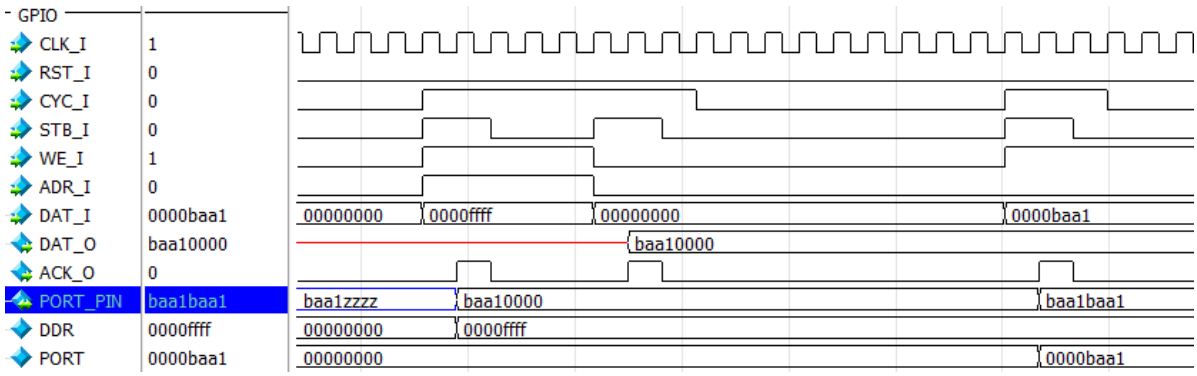


Figura 65. Configuración y uso de GPIO (simulación)

Memoria on-chip

Se almacenan en registros de propósito general tres caracteres ASCII: *A, B* y *C*. Posteriormente se almacenan en las direcciones 0, 4 y 8 de la RAM on-chip, para posteriormente ser leídos de vuelta. El objetivo de este proceso es verificar que la operación de almacenamiento y carga de la memoria en el bus es correcta.

```
# Carga de letras a enviar
addi $s0,$zero,65  # A
addi $s2,$zero,66  # B
addi $s4,$zero,67  # C

# Escribe letras a RAM,
# posteriormente se leen
sw   $s0,0($t5)
lw   $s1,0($t5)
sw   $s2,4($t5)
lw   $s3,4($t5)
sw   $s4,8($t5)
lw   $s5,8($t5)
```

Figura 66. Escritura y lectura de caracteres ASCII en la memoria on-chip (código)

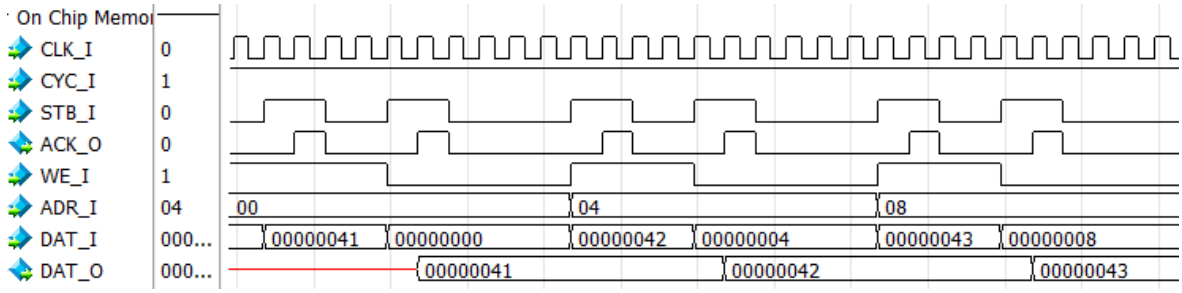


Figura 67. Escritura y lectura de caracteres ASCII en la memoria on-chip (simulación)

Puerto UART: escritura.

Los valores leídos de la memoria RAM (caracteres ASCII *A, B* y *C*) son enviados al módulo de comunicación serial UART mediante la instrucción *store word*.

```
# Envia letras puerto UART
sw    $s1,12($t1)
sw    $s3,12($t1)
sw    $s5,12($t1)
```

Figura 68. Envío de caracteres al UART (código)

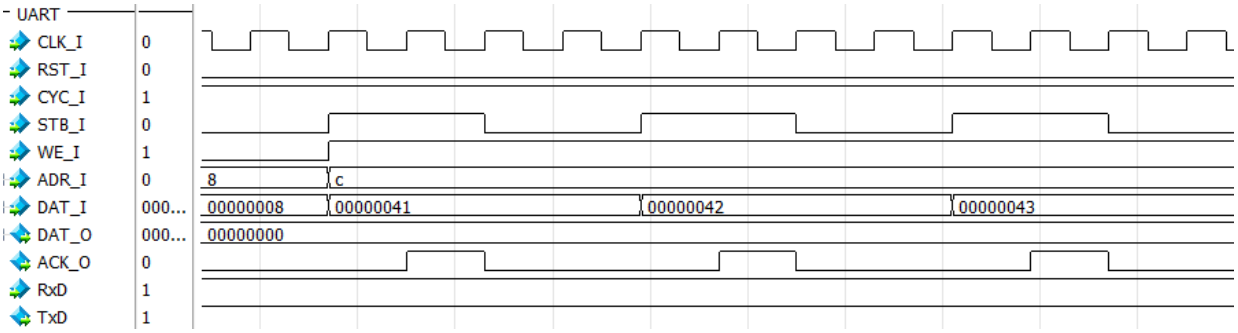


Figura 69. Envío de caracteres al UART (simulación).

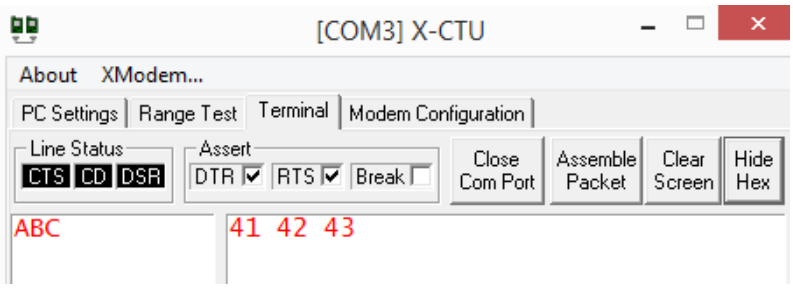


Figura 70. Recepción de caracteres por parte de la PC

Configuración del Temporizador

La inicialización del temporizador consiste en enviar el modo de operación (por default 0x34) y posteriormente enviar el valor de la cuenta, que en este caso es 20. El prescaler del temporizador es configurable, y para el caso de este ejemplo tiene un valor de división de 21, lo cual otorga una frecuencia de salida de 0.2 Hz.

El valor requerido por el sistema es de 100 Hz (ver capítulo 3), pero para el caso de este ejemplo se conectó la salida de interrupción a un LED para verificar el funcionamiento del temporizador de una manera más eficiente.

```
# Inicialización Timer
addi $s0,$zero,52    #Modo
sw   $s0,0($t3)
addi $s1,$zero,20   #Cuenta
sw   $s1,4($t3)     # Escribe parte baja
sw   $zero,4($t3)   # Escribe parte Alta
```

Figura 71. Configuración del Temporizador (código).

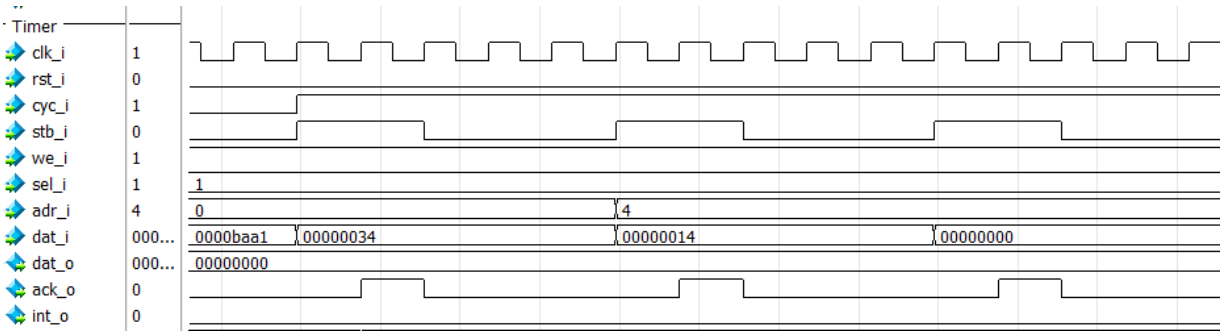


Figura 72. Configuración del Temporizador (simulación).

Controlador de Interrupciones.

El controlador de interrupciones requiere de tres palabras de inicialización denominadas ICW.

```
# Inicializacion PIC
addi $s0,$zero,17
sw   $s0,0($t4)
sw   $zero,4($t4)
sw   $zero,4($t4)
```

Figura 73. Inicialización del controlador de interrupciones (código).

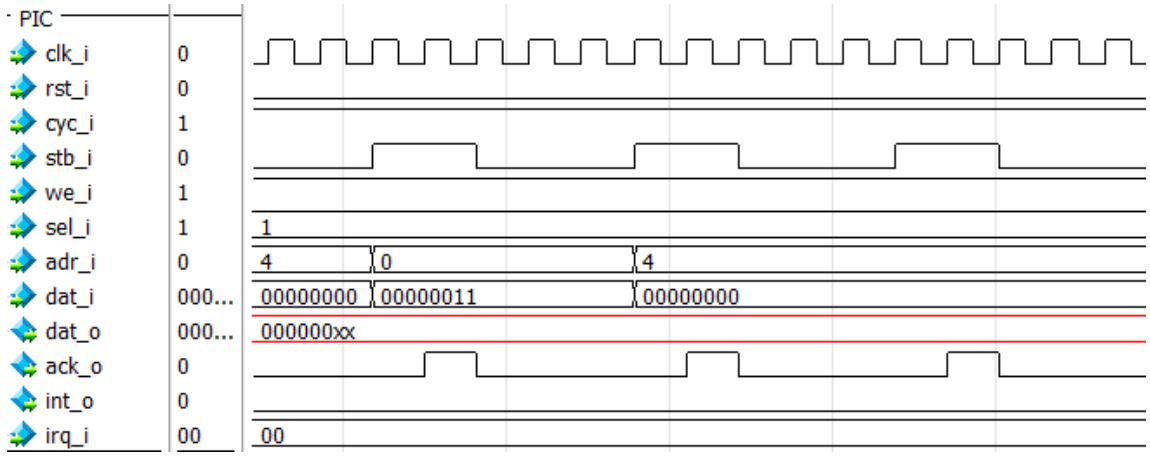


Figura 74. Inicialización del controlador de interrupciones (simulación).

También existen comandos de operación, denominados OCW. El comando OCW3 se envía al controlador para avisar que se desea hacer una lectura del registro de estado (para verificar si existe una interrupción). En la Figura 75 se presenta el bucle que realiza el envío del comando OCW3 y la posterior lectura del registro de IRR. Una vez que se recibe la interrupción del temporizador el código sale de este bucle.

```
  #Polling
poll:
   addi  $s1,$zero,12
   sw    $s1,0($t4)
   lw    $s0,0($t4)
   andi  $s0,$s0,128    #Evalua bit 7
   beq   $s0,$zero,poll
```

Figura 75. Muestreo de la interrupción del temporizador (código)

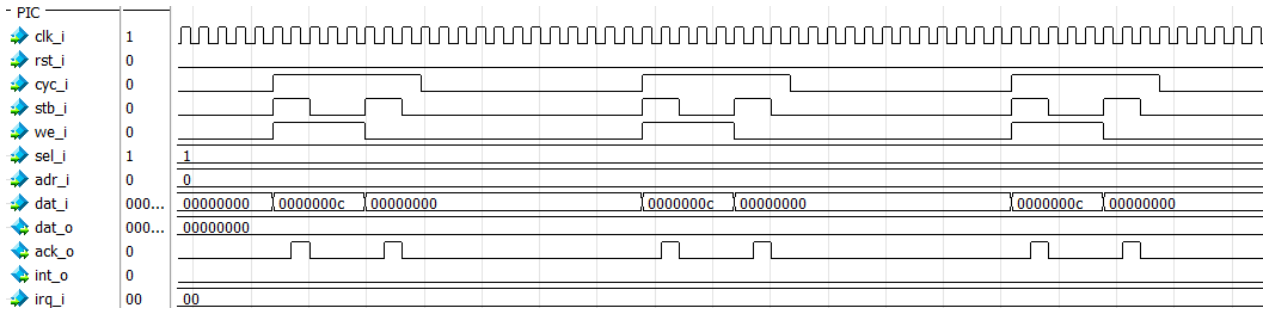


Figura 76. Muestreo de la interrupción del temporizador (simulación)

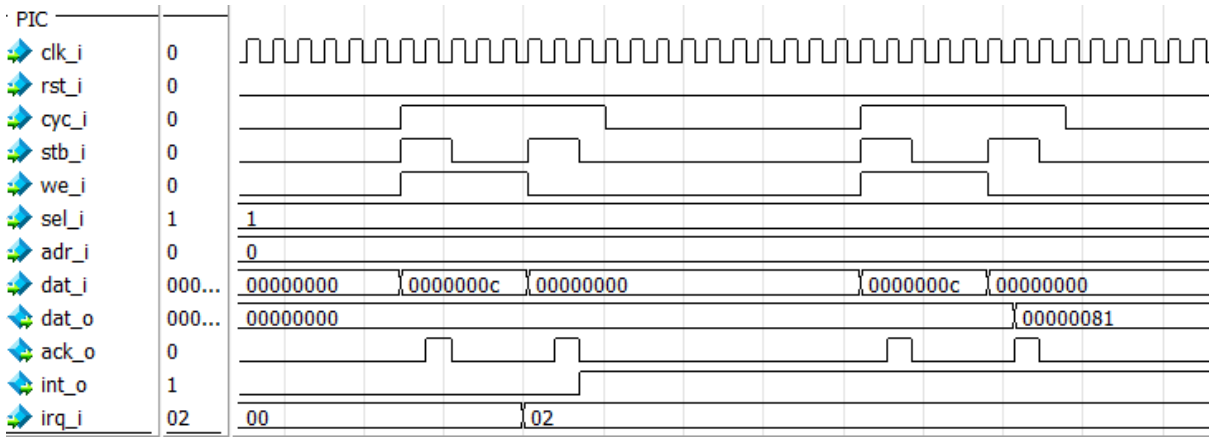


Figura 77. Recepción de la señal de interrupción del temporizador (simulación)

Puerto UART: lectura y escritura.

Una vez se rompe el bucle de muestreo, se ingresa a otro bucle infinito en el cual cada vez que se detecte la recepción de un carácter se procede a enviarlo de vuelta.

```
  # Lectura UART
recibe:
    lw  $s0,0($t1)
    andi $s0,$s0,2
    beq  $s0,$zero,recibe
    lw   $s1,8($t1)
    sw   $s1,12($t1)   #echo
    j    recibe
```

Figura 78. Rutina de lectura del puerto UART (código).

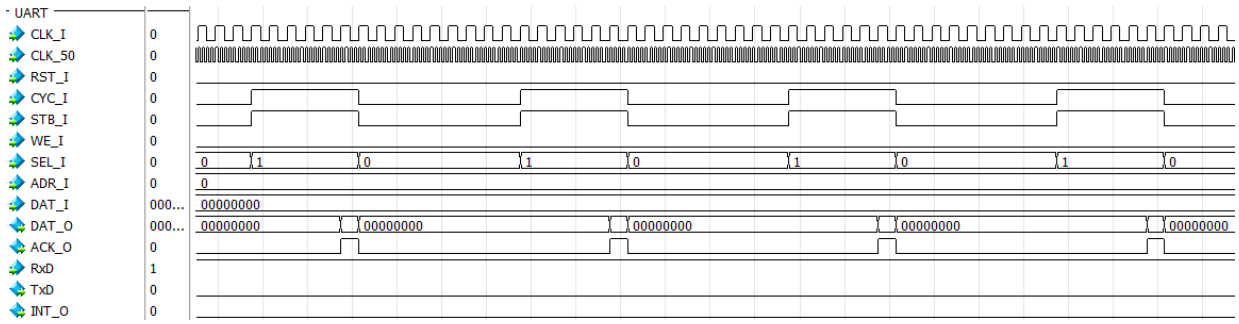


Figura 79. Recepción del valor del registro SR del puerto UART (simulación).

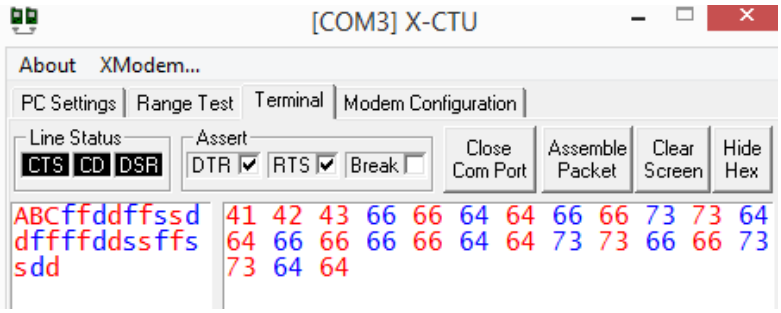


Figura 80. Comunicación Lagarto I - PC

6.4 Estadísticas de implementación del bus jerárquico.

El diseño fue probado en una tarjeta de desarrollo Terasic DE2-115 el cual tiene un dispositivo FPGA de Altera Cyclone IV (EP4CE115F29C7), este dispositivo cuenta con 114,480 elementos lógicos, 3,888 Kbits en elementos de memoria, multiplicadores embebidos y 4 PLLs.

La tarjeta DE2-115 cuenta con distintos periféricos tales como puerto RS-232, Ethernet, VGA, entrada y salida de audio, USB y PS/2. También cuenta con un switches, LEDs, displays de 7 segmentos, LCD 16x2, push botons, entre otras cosas. En la Figura 81 se muestra una imagen de la tarjeta que está ejecutando el ejemplo descrito.

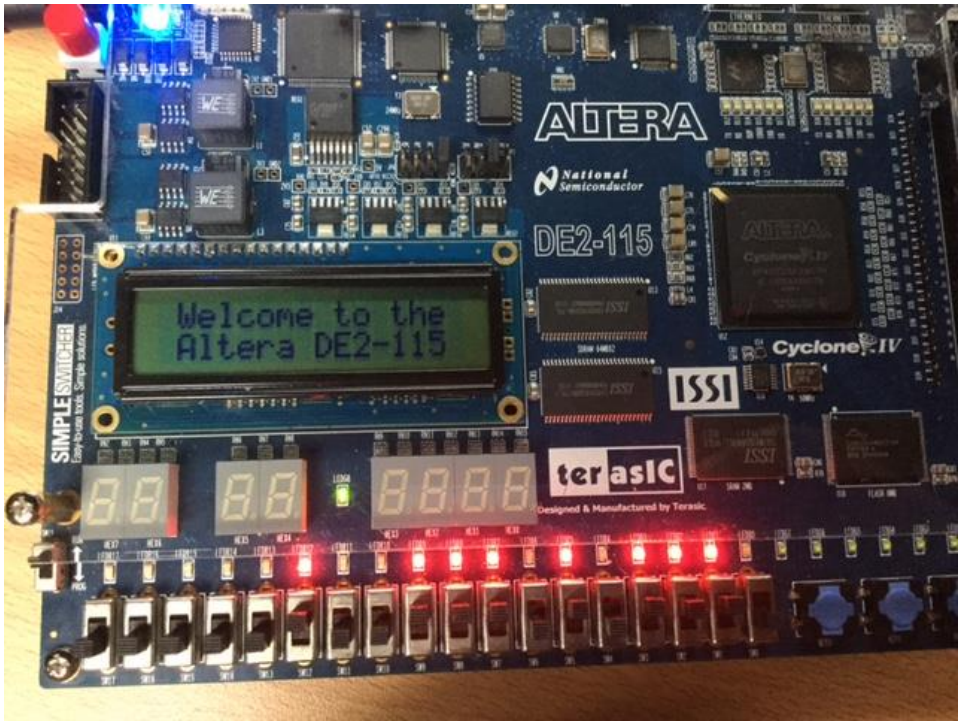


Figura 81. Tarjeta DE2-115 con el ejemplo de la sección 6.3 en funcionamiento.

En la Tabla 21 se observa el valor de algunos parámetros para el ejemplo de la sección 6.3, todos estos parámetros pueden ser modificados desde el proyecto de diseño de hardware. En la Tabla 6.5 se observan los recursos consumidos y la frecuencia máxima de todo el sistema en conjunto

Tabla 21. Parámetros de elementos del diseño

Parámetro	**Valor**
CLK_CPU	12.5 MHz
CLK_PER	6.25 MHz
Tamaño de RAM on-chip	32x1024
[UART] Tasa de baudios	19200 baudios
[UART] Tamaño de FIFO	64 localidades, 10 bits
[Timer] Frecuencia de salida del Pre-scaler	2.98 Hz
[Bridge] Tamaño de FIFO	65 bits x 255 localidades

Tabla 22. Recursos utilizados y frecuencia máxima de operación

Parámetro	**Cantidad**
Elementos Lógicos	4,675
Registros lógicos dedicados	1,859
Memory Bits	801,920
LUT-Only LCs	2,816
Register-Only LCs	364
LUT/Registers LCs	1,495
Frecuencia máxima	43.54 MHz

CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO.

7.1 Conclusiones

La arquitectura de comunicación on-chip es un factor diferenciador de un System on Chip. La manera en la que se interconectan los dispositivos tiene un impacto relevante en el desempeño y consumo de energía del sistema, si bien, actualmente las arquitecturas de comunicación *Network on Chip* (NOC) están teniendo un fuerte avance, la comunicación mediante buses compartidos sigue siendo la mejor opción para sistemas con número reducido de componentes.

La interconexión entre el procesador con elementos de entrada salida, así como el desarrollo de la unidad de gestión del memoria y el procesador de sistema, son indispensables para alcanzar un ambiente propicio para la implementación de un Sistema Operativo, la realización del presente trabajo es una contribución importante para el desarrollo de dicho ambiente.

La arquitectura de comunicación implementada sigue las especificaciones del estándar Wishbone, el cual suele ser utilizado con topologías de bus compartido simple, pero que en el presente trabajo fue extendida hacia una topología de bus compartido jerárquico. A pesar de esta modificación, aún es posible integrar al sistema cualquier tipo de IP core que sea compatible con Wishbone.

Tras la realización de este trabajo ahora es posible desarrollar aplicaciones con el procesador Lagarto I donde se haga uso de periféricos de entrada salida tales como puerto de comunicación serial UART, puertos GPIO y temporizador, además de poder gestionar las interrupciones de hardware mediante un controlador de interrupciones.

7.2 Trabajo Futuro

Es posible dividir el trabajo futuro en dos categorías: aquellas tareas relacionadas con el desarrollo de System On Chip, y las relacionadas con la implementación del Sistema Operativo.

7.2.1 System On Chip

Con el objeto de seguir mejorando el funcionamiento del bus es necesario el desarrollo de Benchmarks, mediante los cuales se evalué el desempeño del SoC, se encuentren errores y cuellos de botella para consecuentemente proponer mejoras y nuevas funcionalidades del bus.

Dentro de las posibles funcionalidades que pueden ser añadidas al bus se encuentran:

- Señales de paro y detección de errores.
- Señales de etiqueta (*Tag*), útiles para informar al árbitro la importancia de una transacción.
- Señales adicionales que no se consideren en el estándar Wishbone. El realizar esta acción haría que el bus dejara de ser compatible con IP cores Wishbone pero permitiría dar soluciones específicas para el proyecto Lagarto.

Otro aspecto importante para el desarrollo del SoC es la optimización de los bloques diseñados en este trabajo, ya sea para incrementar la velocidad de los mismos o para añadir nuevas funcionalidades. También es necesaria la integración de nuevos periféricos, dentro de los cuales se recomienda:

- Bloque DMA
- Puerto USB
- Reloj en tiempo real (RTC)
- Controladores de Memoria SDRAM
- Controladores de SD Card
- Controlador de Ethernet

7.2.2 Sistema Operativo

Es necesario realizar la integración completa del CPU Lagarto (núcleo, unidad de gestión de memoria y procesador de sistema). La sincronía de estos elementos es indispensable para continuar con la implementación del Sistema Operativo.

Realizar modificaciones a nivel hardware o software de acuerdo a las especificaciones del kernel de Linux, tales como asignación de direcciones base a cada dispositivo, asignación de vectores de interrupción, etc.

REFERENCIAS

[1] MIPS. *MIPS32 Architecture for Programmers.* MIPS Technologies, Inc., July 2005. Version 2.5

[2] Dally, W., "Computer architecture is all about interconnect", in *8th International Symposium on High-Performance Computer Architecture*, Cambridge, MA, 2002.

[3] S. Pasricha and N. Dutt. *On-Chip Communication Architectures*. Morgan Kaufmann, 2008.

[4] Saleh, R., Wilton, S., "System-on-chip: Reuse and integration", *Proceedings of the IEEE,* vol. 94, No. 6, pp. 1050-1069, June 2006.

[5] Flynn, M., Luk, W., *Computer System Design. System-on-chip.* New Jersey: Wiley, 2011, pp. 123-160

[6] Paricha, S, Dutt, N. and Ben-Romdhane, M., "Contraint-driven bus matrix synthesis for MPSoC", *Asia and South Pacific Design Automation Conference (ASPDAC 2006),* Yokohama, Japan, January 2006, pp. 30-35

[7] Herveille, R,. WISHBONE *System-on-chip Interconnection Architecture for Portable IP Cores,* rev. version: B4, 2010. By Open Cores Organization [Online]. Available: http://www.opencores.org/opencores,wishbone

[8] Swain, A., Mahapatra, K., "Design and Verification of WISHBONE Bus Interface for System-on-Chip Integration", *2010 Annual IEEE India Conference (INDICON)*, 2010.

[9] Sharma, M., Kumar, D. "Wishbone bus architecture. A survey and comparison", *International Journal of VLSI design & Communication Systems,* vol. 3, No. 2, April 2012.

[10] Ayala, J,. Lopez, M., "State-of-the-Art SoC Communication Architectures," in *Embedded Systems Handbook*, Zurawski, R., 1st ed. Taylor & Francis Group, 2006, pp: 521-542.

[11] Mitic, M., Stojcv, M., "*A survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone,"* ICEST 2006.

[12] ARM. "AMBA Specification" 2015. [Online] Available: http://www.arm.com/products/system-ip/amba-specifications.php

[13] IBM Microelectronics. "CoreConnect Bus Architecture Overview" 2015. [Online] Available: https://www.01.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture

[14] Altera Avalon Interface Specification. Abril 2006. [Online] Available: http://www.altera.com

[15] OpenCores. "Wishbone Specification B4" 2010. [Online] Available: http://opencores.org/cdn/downloads/wbspec_b3.pdf

[16] ARM PrimeXsys Platform, [Online] Available: http://www.arm.com.

[17] Phillips, S., "VictoriaFalls: Scaling highly-threaded processor cores", *Proceedings, HotChips,* 2007.

[18] Martínez, D. O., "Diseño de un procesador de sistema para arquitecturas RISC," M.S. thesis, Centro de Investigación en Computación, Instituto Politécnico Nacional, Distrito Federal, Junio 2015.

[19] Chauhan. *Principles of Operating Systems*. Oxford University Press, 2014

[20] Sweetman, D., *See Mips Run,* "Chapter 2. MIPS Architecture", San Francisco: Morgan Kaufmann Publishers, 2007, pp. 29 – 35.

[21] Intel. *8253 Programmable Interval Timer*. Intel, November 1986.

[22] Lin, M., *Digital Design and Practice: Using Verilog HDL and FPGAs*. "Design Examples", USA: CreateSpace Independent Publishing, 2015, pp. 510 – 580.

[23] Intel. *8259A Programmable Interrupt Controller.* Intel, December 1988.

[24] Martin, G., "The History of the SoC Revolution", in *Winning the SoC Revolution*, 1st ed. New York: Springer, 2003, pp. 21-46.

[25] McGroddy-Goetz, K., Devins, R., Hale, M., "SoC – The IBM microelectronics approach", in *Winning the SoC Revolution*, 1st ed. New York: Springer, 2003, pp. 119-140.