DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: ARQUITECTURA DE COMPUTADORES.

GUSTAVO ADOLFO OCHOA BLANCO
NEDER JAIR MENDEZ FLOREZ

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2012

DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: ARQUITECTURA DE COMPUTADORES.

GUSTAVO ADOLFO OCHOA BLANCO
NEDER JAIR MENDEZ FLOREZ

Trabajo de grado presentado como requerimiento parcial para optar al título de:

Ingeniero Electrónico

Tesis desarrollada dentro del grupo de investigación CPS

Director:
MSc. Jorge H. Ramón Suarez

Co-Director:
MIE. William A. Salamanca B.
MIE(c). Carlos A. Angulo J.
MIE. Carlos A. Fajardo A.
MIE. Sergio A. Abreo C.

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2012

Agradecimientos

Queremos agradecer a nuestro director y a nuestros codirectores que nos apoyaron en todo el proceso y significaron para nosotros el recurso humano que fue de ayuda invaluable y desinteresada para desarrollo del proyecto. A los compañeros del proyecto conjunto, a los estudiantes voluntarios que hicieron del proceso de validación una experiencia de gran ayuda.

Dedico este trabajo a mi familia en especial a mi abuela Cleotilde quien me crio me dio todo el amor que una madre puede dar, sentó mis principios morales y humanos, a mi abuela Dulcelina que me apoyo de todos los modos posibles y me brindo afecto en todo momento, a mi tia Alba que estuvo siempre pendiente de mí, y a mi madre, quien he aprendido a querer estos últimos años y se ha convertido en uno de los pilares de mi vida, gracias a esas cuatro mujeres que significan todo para mí.

Neder Jair Mendez Florez

Le dedico este logro a mi familia, por la confianza depositada y por su apoyo en los momentos difíciles que me ayudan a sobrepasar las adversidades que se me van presentando en el camino. A mi madre Cecilia por siempre estar mi lado y brindarme ese amor y fortaleza que me motivan a realizar todos mis sueños. A mi Padre Ovidio que aunque no se encuentre presente, sé que me está acompañando y se encuentra orgulloso de esta meta alcanzada en mi vida. A diana por su entrega, apoyo y motivación que me ayudaron a creer en mí mismo. A todas y cada una de esas personas que me brindaron su tiempo y sus concejos para terminar este proyecto solo tengo palabras de agradecimiento.

Gustavo Adolfo Ochoa Blanco

Tabla de Contenido

Lista de Figuras

Lista de Anexos

Resumen

TÍTULO: DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: ARQUITECTURA DE COMPUTADORES.[*]

AUTORES: GUSTAVO ADOLFO OCHOA BLANCO, NEDER JAIR MENDEZ FLOREZ [**]

PALABRAS CLAVE: Plataforma SIE, *Driver*, periféricos, interfaz de comunicación, procesador, FPGA, arquitectura de computadores.

Este trabajo pretende ser una herramienta de apoyo académico para los estudiantes de la asignatura arquitectura de computadores e iniciar un proceso de renovación del enfoque pedagógico del área de sistemas digitales en la Universidad Industrial de Santander. Se diseñaron un conjunto de guías como medio de transferencia de conocimiento de la investigación hecha y las experiencias ligadas a esta, basadas en la plataforma SIE y las grandes ventajas académicas que representa. El propósito de cada guía es el de provocar en el estudiante una comprensión profunda y consiente de conceptos propios de la asignatura, a través de una filosofía de concepción a bajo nivel de los procesos que se llevan a cabo en cada experiencia y actividad propuesta. Ventaja que ofrece la plataforma SIE al ser totalmente libre (hardware copyleft, free software) debido a que el estudiante dispone entre muchas otras cosas de información que de otra manera estaría restringida. Además del conjunto de guías se diseñó un manual de usuario, que tiene como función complementar los temas y conceptos vistos en las guías refiriéndose en su mayoría a conceptos complementarios y detalles que deben ser tenidos en cuenta para el desarrollo de las mismas. Por último para asegurar la calidad del producto final de este trabajo, representado en las guías de laboratorio y el manual de usuario se realizó un proceso de validación que tuvo como principal recurso, un grupo de estudiantes voluntarios, que ayudaron a detectar errores y a enriquecer el contenido de de estas a través de su ejecución.

[*] Trabajo de grado

[**] **Facultad**: Facultad de Ingenierías Físico Mecánicas.**Escuela:** Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones.**Grupo de Investigación:** CPS. **Director:** MSc. Jorge H. Ramón Suarez. **Codirectores:** MIE. William A. Salamanca B., MIE(c). Carlos A. Angulo J., MIE. Carlos A. Fajardo A., MIE. Sergio A. Abreo C.

Abstract

TITLE: DEVELOPMENT OF EDUCATIONAL MATERIAL FOR DIGITAL SYSTEMS AREA BASED ON PLATFORM SIE. SUBJECT: COMPUTER ARCHITECTURE .[*]

AUTHORS: GUSTAVO ADOLFO OCHOA BLANCO, NEDER JAIR MENDEZ FLOREZ [**]

KEY WORDS: SIE Board, Driver, peripherals, comunication interface, processor, FPGA, computer architecture.

This degree project was done in order to support computer architecture students in their practical learning process, and begin a renewal process of the pedagogical approach in digital system education at UIS. A set of laboratory guides was designed to transfer the knowledge reached by the research based of the SIE platform and his pedagogical advantages. Each guide is intended to cause deep and conscious understanding of concepts related to the subject by a deep conception philosophy of the process related to the activities in each laboratory guide. This advantage of SIE is result of his free software and hardware condition since the student can get information that otherwise would be restricted. Furthermore a user manual was designed in order to complement the laboratory guides, describing extra topics and some details to get success in the laboratory guides. Finally in order to check quality in the laboratory guides and user manual a validation process was done, in which a group of volunteers students followed the steps in each guide and found some extra information in the user manual to finish the activities in each guide, and in the end of the experience the students gave some recommendations and advices to improve structure and concepts in the guide.

[*] Degree project

[**] **Faculty**: Physico-mechanical Engineering Faculty.**School:** School of Electrical, Electronics and Telecommunications Engineering.**Research group:** CPS. **Director:** MSc. Jorge H. Ramón Suarez. **Codirectors:** MIE. William A. Salamanca B., MIE(c). Carlos A. Angulo J., MIE. Carlos A. Fajardo A., MIE. Sergio A. Abreo C.

INTRODUCCIÓN

La necesidad de implementar circuitos digitales para realizar diferentes tareas donde la electrónica analógica tiene algunas restricciones, hacen de los sistemas embebidos una parte importante del desarrollo tecnológico que se perfila actualmente como uno de los puntos claves del progreso de la sociedad.

En el mercado mundial existe una variedad de sistemas de desarrollo de circuitos digitales con *software* privado, los cuales poseen costos elevados que dificultan el acceso al estudiante interesado en trabajar en el mundo digital, ya que implican una inversión de capital inicial elevado y posterior desinterés en el tema. Además, algunos de estos limitan al estudiante a proponer ideas de implementación de sus propios diseños y omiten información de cómo se conectan todos sus componentes ya que el *software* es el encargado de realizar toda esta comunicación con sus diferentes periféricos.

Los laboratorios de sistemas digitales que posee actualmente la Universidad Industrial de Santander UIS, se encuentran dotados con varios sistemas de desarrollo del fabricante *Xilinx*, como lo son el sistema *CoolRunner II, X-board versión* y el sistema *Spartan 3an*, éste ultimo utilizado en la asignatura arquitectura de computadores. Para darle al estudiante más opciones de encaminar sus ideas desde la teoría, al diseño y llegar a la implementación para afianzar estos conceptos , surge una posibilidad que reúne todos estos factores y es la tarjeta SIE. Esta tarjeta esta basada en *hardware copyleft* que por su filosofía de ofrecerle al usuario final un *hardware* estable, la utilización de *software* libre y toda la documentación necesaria para su reproducción hace que el estudiante realice sus diseños y los implemente llevando a la práctica varios aspectos como lectura de hojas de datos y lectura de esquemáticos afianzando sus conocimientos teóricos.

En este trabajo de grado se realizaron un manual de guías de laboratorio y un manual de usuario para la asignatura arquitectura de computadores, que van a introducir al estudiante en el manejo de la tarjeta SIE en la asignatura, comenzando por el manejo de las llamadas al sistema que permiten una relación entre memoria física y memoria virtual. Una parte importante de esta tesis va relacionada con la interfaz de comunicación entre el procesador y el FPGA de la SIE, que es la interacción directa entre *software* y *hardware* permitiendo al estudiante entender cómo el procesador se comunica con sus periféricos desde bajo nivel programando en lenguaje de alto nivel para el procesador y en VHDL en caso del FPGA. Para comprobar que se haya realizado esta comunicación correctamente se realiza un circuito de verificación.

Un capitulo está dedicado a la comunicación FPGA-procesador viendo el FPGA como un periférico interno de la tarjeta. El procesador se encarga de manipular las peticiones de los periféricos que necesiten ser atendidos porque terminaron de hacer un proceso y requieren permiso para seguir trabajando, generando así una interrupción en él, hecho esto la comunicación entre el procesador y el periférico ya se puede hacer en ambos sentidos.

Otro capitulo es la adición de un periférico externo a la tarjeta SIE y mas adelante se aborda el tema de interrupciones en el procesador completando así el ciclo de aprendizaje del estudiante en esta asignatura. Para consolidar los conocimientos y competencias adquiridas en el transcurso de la asignatura se propone un proyecto final.

Una parte importante de este proyecto es la validación por parte de algunos estudiantes a las guías, donde se realiza una aprobación tanto metodológica como contextual del contenido de cada una de ellas.

1. ESPECIFICACIONES DEL PROYECTO

1.1 Objetivo general

Desarrollar material didáctico de laboratorio para la asignatura de arquitectura de computadores apoyándonos en la plataforma SIE.

1.2 Objetivos especificos

- Crear un manual de usuario de la plataforma SIE orientado a su uso en la asignatura arquitectura de computadores.
- Implementar un mecanismo de comunicación entre el procesador y los periféricos desde el sistema operativo.
- Desarrollar guías que le permitan al alumno estudiar la comunicación entre un procesador y diferentes tipos de periféricos.
- Diseñar e implementar el hardware necesario para el correcto desarrollo de las guías.
- Validar las prácticas propuestas.

1.3 Alcances

Dentro de los alcances de este proyecto fueron:

- La creación de un manual de usuario que introduzca al estudiante al potencial de la herramienta de trabajo. Éste estará enfocado a las herramientas que se utilizarán a través del curso.
- Utilización de un *driver* genérico para establecer la comunicación entre el procesador y diversos periféricos, y permitir el control de estos.

- Las guías se orientarán hacia la comunicación procesador-periféricos, aprovechando el *driver* genérico mencionado anteriormente, tratando los siguientes temas:

  Interrupciones.

  Descripción del hardware de comunicación.

- Diseño e implementación de hardware de expansión como: leds, pulsadores, interruptores y cualquier otro que se necesite en el transcurso del proyecto

- Las guías serán validadas poniéndolas a consideración de los integrantes del proyecto conjunto, siguiendo el proceso indicado en éstas. Las sugerencias y correcciones arrojadas por la experiencia serán realimentadas al proyecto.

2. GENERALIDADES

2.1 Plataforma SIE.

La plataforma SIE nace como una herramienta para la enseñanza aunque también puede ser usada para implementar aplicaciones reales de la industria. Fue concebida por el profesor Carlos Camargo de la Universidad Nacional de Colombia y proviene del proyecto *open hardware* BEN NANONOTE a la cual básicamente se le retiró la carcasa, el teclado y la pantalla, liberando así puertos de entrada salida y también se le agregó un FPGA.

La plataforma SIE se muestra en la figura 2.1 y cuenta con las siguientes características :

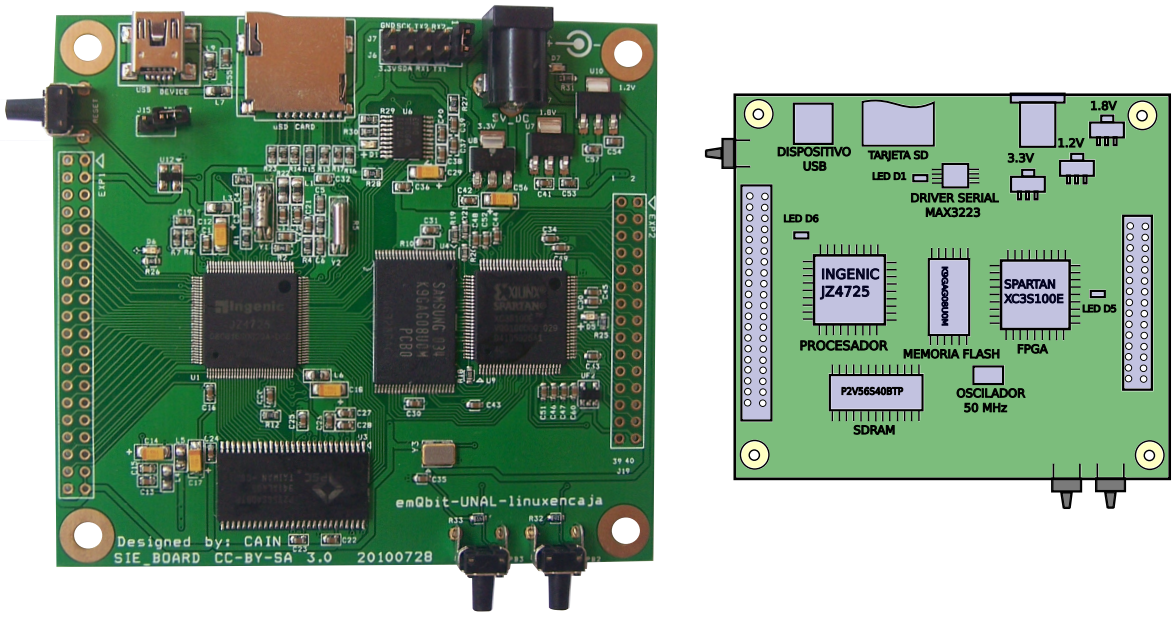


Figura 2.1: SIE

FUENTE: Los autores.

- Procesador Ingenic JZ4725.
- 64MB de memoria SDRAM.
- Memoria NAND de 2GB.
- FPGA XC3S100E_VQ100 que proporciona 25 puertos de entrada salida de propósito general con señales digitales (rango 0-3.3V).
- El Puerto USB puede ser usado como Ethernet, o dispositivo de consola serial.
- Puerto Micro SD.
- Líneas de entrada / salida de audio Estéreo.
- Señal de entrada de micrófono.
- Puerto I2C.
- Puerto RS-232 Serial UART.

A continuación se muestra el diagrama de bloques de la SIE:

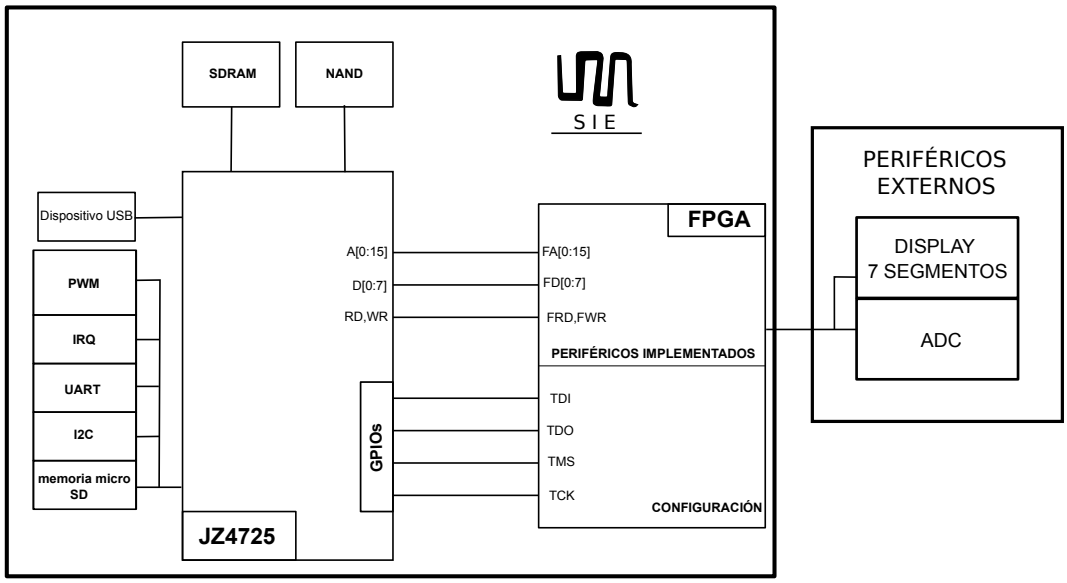


Figura 2.2: Diagrama de bloques de la SIE.

FUENTE: Los autores.

El diagrama del procesador se muestra en la figura 2.3.

2.2 Conceptos de Linux.

El procesador JZ4725 funciona bajo el sistema operativo OPENWRT, el cual es una distribución de Linux para sistemas embebidos. El papel del sistema operativo es fundamental para el desarrollo de las guías, ya que funciona como puente entre el estudiante y el *hardware*.

El *kernel* o núcleo es la parte más importante del sistema operativo y está compuesto por fragmentos de código llamados módulos, que se encargan de realizar alguna tarea. Cuando un módulo del *kernel* se encarga de administrar o manejar periféricos recibe el nombre de *driver* sin embargo no deja de ser un módulo del *kernel*. Los usuarios de Linux son libres de crear sus propios módulos o *drivers*, o de manipular los que posee

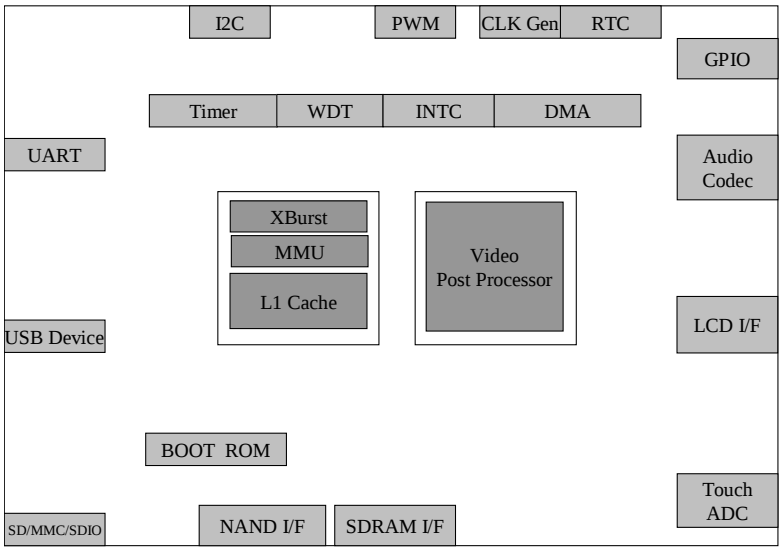


Figura 2.3: Diagrama del procesador

FUENTE: [7].

el sistema operativo, pero con ciertas políticas de manejo para garantizar su seguridad y estabilidad de éste.

El principal objetivo de este trabajo es diseñar guías que ayuden al estudiante a comprender los aspectos relacionados con el manejo y comunicación de periféricos desde un procesador, por lo cual se debe tener acceso al *hardware* por algún medio. El sistema operativo, al ser el que administra los recursos del procesador entre ellos el *hardware*, permite que el usuario tenga acceso y sea capaz de manipularlo a través de un *driver* propio o del sistema raíz.

En el sistema operativo Linux los *drivers* no pueden ser usados directamente, por lo cual se debe recurrir a los llamados al sistema. Los llamados al sistema son funciones de Linux que permiten al usuario comunicarse con el *kernel* de manera segura y hacer uso de las funcionalidades de los módulos que la componen.

En resumen, para administrar los periféricos (en el transcurso de las guías) se hará

uso de un *driver* nativo de Linux que fue diseñado para administrar la memoria física del procesador y así acceder a sus registros y buses, que entre otras, funciones permiten la comunicación con el exterior. Dicho *driver* será combinado con el uso de dos llamados al sistema: *open* y *mmap* que juntos permiten hacer uso del *driver* para las necesidades de cada guía.

La figura 2.4 muestra un diagrama de la interacción entre el sistema operativo, el usuario y los periféricos.

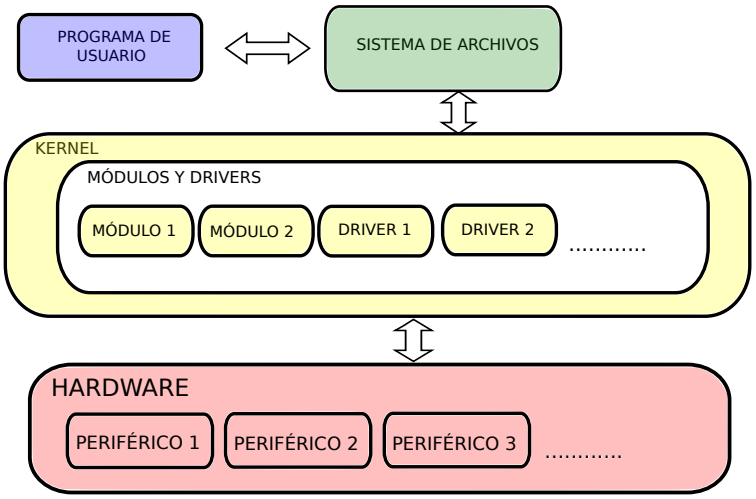


Figura 2.4: Interacción usuario-periféricos.

FUENTE: Los autores.

Adicionalmente, se debe tener en cuenta que el sistema de archivos de Linux y el *kernel* deben ser copiados a la memoria NAND del procesador este proceso es conocido como "flasheo" y es necesario para el funcionamiento de la plataforma.

2.3 Compilación de programas

Con el fin de crear aplicaciones para el procesador de la plataforma SIE es necesario tener una herramienta de *software* llamada compilador que se encarga de traducir programas o aplicaciones de usuario a lenguaje máquina. En este caso se encarga de transformar un código en lenguaje C a un ejecutable. Este compilador puede ser instalado en

el procesador de la plataforma, hecho que gastaría recursos y sería poco práctico, por lo que se prefiere instalar dicho compilador en un equipo externo que puede ser un computador personal. Lo anterior teniendo en cuenta que la versión de este compilador debe ser “cruzada”, es decir, instalada y usada en una arquitectura para compilar programas para otras arquitecturas diferentes. El compilador cruzado usado en esta oportunidad es GCC en su versión para la arquitectura de procesadores MIPS que corresponde a la del procesador JZ4725.

El proceso para generar un ejecutable a partir de un código C se muestra en la figura 2.5

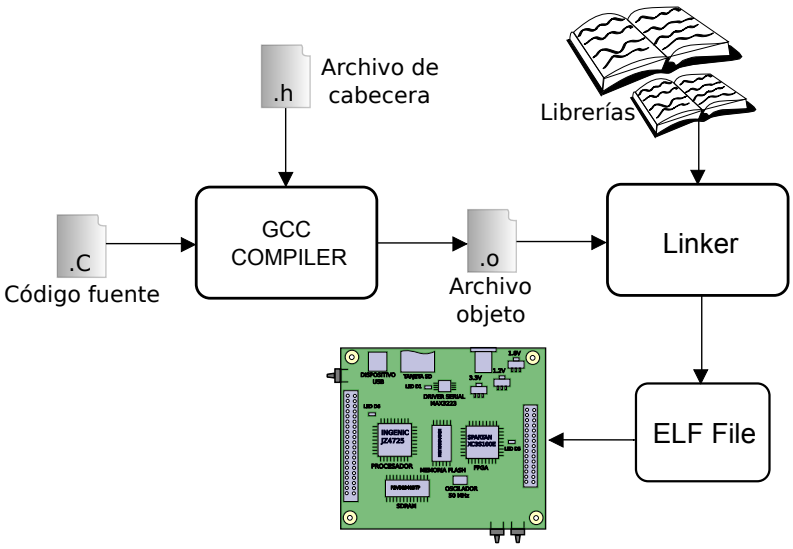


Figura 2.5: Proceso de compilación

FUENTE: Los autores.

En la primera etapa se crea un archivo objeto a partir del programa de usuario en lenguaje C y los archivos de cabecera usados en dicho programa, posteriormente el linker se encarga de relacionar el archivo objeto sólo con los fragmentos o partes de código de las librerías que en realidad necesita, finalizando el proceso. Los últimos pasos son enviar el ejecutable a la plataforma y posteriormente ejecutarlo.

3. METODOLOGÍA

Definición del tema y objetivos.

Siempre es importante darle sentido y un propósito a toda actividad, por lo tanto lo primero que se debate es la finalidad de las prácticas y qué temas serán tocados, Además cumpliendo con las necesidades propuestas en el plan de proyecto.

Investigación y estudio del tema.

Este proyecto requiere que los temas que se involucren en cada guía sean investigados, debido al desconocimiento parcial del funcionamiento y la poca experiencia con la plataforma SIE. La información que hay al respecto es suficiente e involucra muchos aspectos que pueden ser usados para el desarrollo de las guías, lo que refuerza la importancia de esta etapa.

Análisis de factibilidad

Posteriormente al estudio del tema, se concluye con la posibilidad de implementarlo utilizando los recursos que ofrece la plataforma SIE con el fin de no crear falsas expectativas y pérdida de tiempo.

Formulación de actividades y alcances.

Si se concluye con el paso anterior que es posible implementar alguna actividad que incluya el tema definido, se procede a proponer y definir más puntualmente posibles actividades.

Elección de actividad a realizar y sus alcances.

Con la lista de actividades definida, se pasa a elegir una que reúna la mayor cantidad de información al respecto del tema e involucre la necesidad de aprender nuevos conceptos

complementarios que entren a aumentar el grueso de competencias del estudiante.

Desarrollo de la actividad y documentación.
En este punto se está en capacidad de realizar la actividad propuesta y ganar experiencia en aspectos que servirán para el diseño de la guía. Todo esto debe ser documentado con el fin de evitar la pérdida de información y tener la base para el diseño de la guía.

Coherencia y ordenamiento de información.
La información documentada en el paso anterior, al ser solo información sin una secuencia explícita o al menos fácil de entender, debe ser organizada y clasificada con el fin de darle validez procedimental.

Diseño de la guía.

Teniendo la información organizada se procede a relacionarlos y disponerlos en las diferentes secciones que se definieron en la estructura de la guía.

Validación.
Las primera versión de la guía queda completa en este paso pero primero debe ser validada, con el fin de comprobar la coherencia procedimental y pedagógica de la práctica.

Correcciones.
Con el fin de comprobar la coherencia y validez de los conceptos empleados y mencionados en la guía se realizan correcciones con la ayuda de los docentes encargados de los proyectos.

Retos propuestos.
Por último se proponen actividades y preguntas que motiven al estudiante a reforzar y profundizar los conceptos empleados en el transcurso de la guía.

Estructura y metodología de las guías.

Las guías se componen de :

- Introducción.

  En donde se ubica al estudiante en el contexto de la guía y temas a tratar, además se menciona la actividad propuesta.

- Objetivos.

  Se mencionan los propósitos de la guía con el fin de establecer las metas y alcances para posteriormente evaluar la efectividad del proceso.

- Marco teórico.

  Se establece el fundamento teórico que da soporte y ayuda a entender de una mejor manera el proceso.

- Procedimiento general.

  Se describen de manera general los pasos o el método para realizar la guía y el porqué de éstos, con el fin de exponer una coherencia procedimental que permita al estudiante articular y relacionar conceptos.

- Desarrollo de la práctica.

  Los pasos descritos en la sección anterior son descritos detalladamente y justificados aquellos que lo necesiten.

- Preguntas y actividades de prueba.

  Por ultimo para consolidar, comprobar y extrapolar los conocimientos y experiencias adquiridas en el transcurso de la práctica se proponen actividades y preguntas que permitan el ejercicio y relación de conceptos.

4. GUÍAS DE ARQUITECTURA DE COMPUTADORES

4.1 Manipulación de periféricos por medio de mmap

Con esta guía se pretende preparar el escenario adecuado que posteriormente permita al estudiante acceder a periféricos y controlarlos. El acceso a los registros del procesador posibilita cambiar, configurar y verificar estados de sus módulos internos, periféricos externos y puertos siendo este paso indispensable para añadir nuevos periféricos y establecer comunicación con ellos.

Haciendo uso del sistema operativo es posible acceder a registros del procesador por medio de los llamados al sistema y un *driver* de Linux. Lo que implica el conocimiento de otros conceptos como mapa de memoria físico y virtual y manipulación de punteros.

La actividad propuesta es la manipulación de dos registros que pertenecen al procesador (PCDATS y PCDTAC) los cuales controlan los niveles lógicos del puerto PCDAT17. El funcionamiento de la práctica se puede verificar con el apagado y encendido de un led contectado al puerto.

4.2 Módulo PWM

Con los conocimientos adquiridos en la guía 1, se motiva al estudiante a aplicarlos en la configuración de un modulo interno del procesador. Con esto se busca que el estudiante conozca y use los recursos que el procesador ofrece. La actividad propuesta es la configuración de un modulo PWM, que involucra la selección de un reloj como entrada y que a su vez debe ser adecuado manipulando sus registros, implicando así el trabajo con diferentes componentes del procesador. Por otra parte se estimula a conocer las diferentes funciones que puede soportar un puerto del procesador y que se debe cambiar

de acuerdo a la necesidad del usuario, en este caso para la generación de una señal PWM.

4.3 Interfaz entre el procesador y el FPGA

En esta instancia se tiene un escenario propicio para pensar en la comunicación del procesador con un periférico. Este periférico se implementa en el FPGA y se controla a través del controlador de memoria externo (EMC) que posee el procesador y que provee las señales de control necesarias para esto. Además es necesario implementar una interfaz que interactúe directamente con el procesador y evite problemas de metaestabilidad y contención de datos. El principal objetivo de esta guía es mostrarle al estudiante cómo son los buses del sistema, cómo están conectados los periféricos, cómo es la interfaz y las señales de control, cómo son los dominios de reloj del procesador y el FPGA y cómo se puede establecer una comunicación robusta entre el procesador y un periférico.

4.4 Introducción al manejo de periféricos externos

Esta guía ocupa un espacio importante dentro de la filosofía del proyecto debido a que motiva al estudiante a diseñar e implementar circuitos externos a la SIE, lo cual implica tener en cuenta factores que aseguren que ese *hardware* sea compatible y funcione de manera adecuada, factores que anteriormente no entraban en el proceso de aprendizaje del estudiante debido a que los sistemas de desarrollo para el área de sistemas digitales facilitan la conexión de periféricos y delegan todas las tareas de bajo nivel a un software especializado.

La guía propone adicionar a la plataforma SIE un dispositivo de visualización compuesto de dos módulos siete segmentos y generar un conteo en el procesador mediante un programa de usuario en C que posteriormente enviará datos al FPGA que los reenviará al dispositivo siete segmentos. Además se implementará un periférico complementario en el FPGA que es necesario para el funcionamiento del dispositivo siete segmentos.

4.5 Interrupciones

Para completar lo referente a la comunicación procesador-periféricos se aborda el tema de la interrupciones que representan el mecanismo de atención de las peticiones de un periférico.

La actividad propuesta es la implementación de un periférico en el FPGA que es capaz de generar interrupciones, las cuales son controladas por el usuario a través de un pulsador. Dichas interrupciones son dirigidas al procesador que se encuentra ejecutando una rutina en particular, con el fin de mostrar cómo este detiene esta rutina para atender la interrupción y luego de terminar vuelve a la rutina original. Cabe resaltar que se hizo necesario el uso de un *driver* para la atención de la interrupción que no será descrito a profundidad debido a que el estudio de *drivers* está por fuera de los alcances de la asignatura y del proyecto.

4.6 Proyecto base final de laboratorio

Por ultimo para afianzar las competencias adquiridas en el transcurso de la asignatura se propone un proyecto base, que servirá de modelo para el desarrollo de proyectos y diseños propios del estudiante.

La actividad propuesta es la implementación de un sistema de adquisición de señales, que implique la implementación de *hardware* externo para la adecuación y conversión de señal y la implementación de *hardware* en el FPGA para recepción y comunicación con el procesador y por último la creación de un programa de usuario que permita la lectura y visualización de datos en consola.

Cabe destacar que este proyecto solo pretende ofrecer una idea al estudiante que sirva de referencia o ayuda para su proyecto propio, por lo tanto solo se documentó lo realizado para la conclusión del proyecto y se dejaron abiertos temas como: mejoras, uso de

interfaces de comunicación diferentes, selección de componentes, etc. No obstante se le dio una estructura y un diseño modular al proyecto con el fin de mantener un orden de trabajo.

5. VALIDACIÓN

El producto final de este trabajo de grado son el conjunto de guías y el manual de usuario que apoyan al estudiante en su proceso de aprendizaje y debido a tan importante tarea deben ser sometidas a un proceso de validación para mejorar la calidad de las mismas. La retroalimentación de conceptos y sugerencias arrojadas por el estudiante al realizar el procedimiento expuesto en cada guía se usará de referencia para dicho fin.

En el segundo semestre del 2011 se dispuso de la colaboración de los estudiantes del curso Diseño con Microprocesador y Microcontroladores a cargo del profesor William Alexander Salamanca Becerra para el proceso de validación. Se hicieron pruebas periódicas en orden ascendente de dificultad y temas, que consistieron en el desarrollo de la guía. A medida que se llevaba a cabo la prueba los estudiantes formulaban inquietudes, preguntas y sugerencias que eran documentadas y posteriormente realimentadas para corregir la guía.

Al terminar el proceso de validación se observó que las dificultades que los estudiantes presentaban al momento de desarrollar la guía eran de tipo procedimental y por ende se enfatizó en mejorar este aspecto. De hecho, la anterior conclusión representó el mayor aporte del proceso de validación y facilitó en gran manera correcciones y diseños posteriores. Otros aspectos tenían que ver con falencias conceptuales que inmediatamente se reforzaron y documentaron de una manera mas didáctica al terminar el proceso de validación. Por último cabe resaltar que también se realizaron procesos de validación con los docentes y compañeros del proyecto conjunto.

6. CONCLUSIONES Y RECOMENDACIONES PARA TRABAJOS FUTUROS

Todos los recursos que conforman la plataforma SIE permiten que ésta pueda ser usada como herramienta de aprendizaje en el curso de arquitectura de computadores, principalmente porque es un sistema basado en la ideología de *software* y *hardware* libre. Lo que implica que está disponible la información necesaria de todos sus aspectos funcionales, físicos y estructurales y por ende los procesos involucrados en cada guía son transparentes al estudiante, formando en éste una concepción profunda y consciente. Por otro lado la plataforma SIE puede ser modificada, reproducida y mejorada; la información necesaria para esto se encuentra disponible y la licencia que la rige lo permite siempre y cuando se otorgue crédito al autor original. Lo anterior posibilita que el contenido de la materia y sus guías de laboratorio, no solo de arquitectura de computadores sino de otras asignaturas afines puedan ser mejorados continuamente y actualizados de acuerdo a la evolución de la tecnología.

La importancia de la plataforma SIE para la asignatura se encuentra en la interacción entre el procesador y el FPGA que ésta posee, ya que el FPGA puede ser reconfigurado con el fin de implementar una gran variedad de periféricos. Este escenario ofrece al estudiante un grueso de posibilidades que van desde la implementación de un solo periférico, hasta un conjunto de varios de ellos trabajando como sistema.

El hecho de que la plataforma SIE tenga los componentes básicos de una arquitectura de computador y no incluya dispositivos extra o periféricos adecuados para trabajar fomenta en el estudiante la creación de diseños propios con el fin de suplir o incluir funciones que no están disponibles o no son soportadas por la plataforma. Este hecho representa uno de los pilares de este proyecto, estimular en el estudiante la competen-

cia de crear diseños de *hardware* propios con el fin de completar el ciclo de generación tecnológica.

La construcción de una metodología para el desarrollo de las guías en este trabajo de grado fue de gran importancia al establecer un orden en el proceso de investigación, generación de conocimiento y diseño de las guías, ya que permitió sentar una base procedimental que imprima coherencia en la consecución de pasos y posterior ahorro de tiempo.

El proceso de validación y corrección fue la base del mejoramiento del diseño de las guías y arrojó como resultado principal falencias en aspectos relacionados con el procedimiento y estructura de las mismas, principalmente la carencia de una descripción general del proceso originando confusión en el estudiante. El proceso de validación y realimentación de las guías, permitió diseñar preguntas y actividades de prueba con base en preguntas e inquietudes conceptuales de los estudiantes que fueron parte de este proceso.

Los resultados de este trabajo de grado, pueden ser tomados como referencia para trabajos posteriores; el conjunto de guías y el manual de usuario pueden ser mejorados o adaptados a las necesidades de la asignatura a medida que se crea conveniente. Por esto se motiva a cualquiera que lo desee a continuar con el proceso y la filosofía de renovación del plan de la asignatura arquitectura de computadores, pensando en la generación, independencia y transferencia tecnológica, con herramientas adecuadas como la plataforma SIE o cualquier otra basada en conceptos de *software* libre o *hardware copyleft*.

También se propone la modificación y mejoramiento de la plataforma SIE para trabajos posteriores, con el fin de acomodarla a las exigencias del estudiante UIS y así fomentar una cultura de pertenencia y generación tecnológica en nuestra universidad.

La plataforma SIE representa una herramienta de alto potencial para el área de sistemas digitales al contar con los elementos necesarios que propician el ejercicio y articulación de conceptos de una forma clara y profunda, y al ser libre facilita los procesos de generación y transferencia tecnológica, de los cuales el estudiante debe ser el principal protagonista con el fin de gestar bienestar y desarrollo a la comunidad.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

[1] Camargo, Carlos Ivan. *SIE: Plataforma Hardware copyleft para la ensenanza de sistemas digitales*. Universidad Nacional de Colombia. 2010. 6 p.

[2] Chu, Pong P. *FPGA PROTOTYPING BY VHDL EXAMPLES , 3rd Edition*. Jhon Wiley & Sons, INC, publication. 2008.

[3] Cobbaut, Paul. *Linux Fundamentals, 1st Edition*. 2010.

[4] Corbet, Jonathan and Rubini, Alessandro and Kroah-Hartman, Greg. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc. 2005.

[5] Datasheet JZ4725, *multimedia Application processor*, Ingenic Semiconductor Co. Ltd, Revision: 1.0,May 2009

[6] Página linux en caja. `http://linuxencaja.net/wiki/SIE/es`.

[7] Página qi-hardware. `http://en.qi-hardware.com/wiki/SIE`.

[8] Patteeson, David A. Hennessy, John L. *Organizacion y Diseno de Computadores. La interfaz hardware/software*. McGRAW-HILL. 2003.

[9] Xilinx Inc. *Xilinx ISE 10.1 Design Suite Software Manuals and Help - PDF Collection*. 2008.

ANEXO A. MANUAL DE GUIAS DE LABORATORIO ARQUITECTURA DE COMPUTADORES

LABORATORIO 1

MANIPULACIÓN DE REGISTROS POR MEDIO DE MMAP

Después de escalar una montaña muy alta, descubrimos que hay muchas otras montañas por escalar.

NELSON MANDELA

1 INTRODUCCIÓN

La plataforma SIE cuenta con un procesador XBURST JZ4725 de la empresa INGENIC que puede ser aprovechado de varias formas, una de ellas es accediendo a sus registros internos. Si se tiene acceso a los registros internos del procesador es posible configurar y controlar una gran parte de su funcionamiento, además de la posibilidad de controlar periféricos internos o externos ligados a dichos registros.

Una de las dificultades es que no se cuenta con herramientas especializadas que faciliten la tarea de manipular registros, ya que algunas ofrecen un conjunto de archivos de cabecera especiales,y manejan *drivers* necesarios a la hora de escribir y leer en éstos. He aquí el núcleo del problema, escribir y leer en registros. ¿Cómo se hace?.
Para leer o escribir en registros es necesario llegar a ellos mediante el sistema operativo que posee el procesador[1] quien administra los recursos de éste y es capaz de acceder a registros por medio del *driver* emph/dev/mem.
Por último los registros a los cuales se accederán son los que controlan el encendido del led D6 que es el más cercano físicamente al procesador, todo con el fin de comprobar la relación existente entre la memoria física del procesador y la memoria virtual generada por el sistema operativo.

[1] openwrt rama backfire

2 OBJETIVOS

- Manipular periféricos del procesador a través de registros.
- Identificar los conceptos de mapa de memoria físico y virtual.
- Interpretar el mapa de memoria físico del procesador.
- Comprender el concepto de llamados al sistema.

3 MARCO TEÓRICO

Openwrt tiene un *driver* que permite acceder a direcciones de memoria física. Este *driver* se ubica en */dev/mem* y es un fichero que administra la memoria física del procesador y a través de comandos y funciones soportadas por el sistema operativo es posible leerla y escribirla. El problema con el sistema operativo es que éste maneja direcciones de memoria virtual que están conectadas con una dirección física de memoria y no se ha establecido alguna relación entre las dos. Una de las soluciones es usar el llamado al sistema *mmap*, que ofrece un sistema de referencia entre las direcciones virtuales y las direcciones físicas, todo esto será explicado más adelante.

4 PROCEDIMIENTO GENERAL

Los pasos que se seguirán a medida que se desarrolla esta guía para lograr escribir o leer en los registros son:

- Abrir el fichero *mem* contenido en la carpeta */dev*.

  Antes de trabajar con cualquier fichero éste debe ser abierto.

- Crear un mapa de memoria desde la dirección inicial hasta una dirección final.

  El fichero *mem* es una abstracción de todas las direcciones de memoria físicas del procesador, con este paso se limita el espacio de direcciones con las que se van a trabajar a un rango determinado.

  La dirección en la que se desea leer o escribir debe estar dentro de éste rango.

- Acceder(leer o escribir) por medio de punteros al mapa de memoria hecho en el paso anterior.

  El rango de memoria o mapa de memoria hecho en el paso anterior es accesible mediante punteros por lo tanto el usuario es libre de realizar operaciones de escritura y lectura.

- Deshacer el mapa y cerrar el archivo con la función *close*.

  Con el fin de liberar los recursos utilizados en pasos anteriores se recomienda deshacer el mapa creado.

4.1 Como abrir y cerrar el fichero *mem*

El llamado al sistema *open* tiene como función abrir un fichero existente y devolver un descriptor de fichero el cual debe guardarse en una variable. El descriptor de fichero(*file_descriptor* en el programa) es una variable que referencia el archivo abierto. Es posible pensar en el descriptor de fichero como una variable que representa el archivo que se ha abierto previamente con *open* para así usarlo posteriormente.

Entonces el código empezaría de la siguiente manera, para mayor información ver manual de usuario:

```
1  #include <stdio.h>  /*librería para printf*/
2  #include <fcntl.h>  /* librería necesaria para Modos de apertura y función open()*/
3  int main()
4  {
5  int file_descriptor;
6  file_descriptor=open("/dev/mem",O_RDWR|O_SYNC); /*apertura del fichero*/
7   if (file_descriptor<0)
8        {
9           printf("error al abrir /dev/mem \n");
10       }
11
12 close(file_descriptor);     /* cierre del fichero*/
13 }
```

Script 1.1: Apertura del fichero */dev/mem*

NOTA: cada vez que se abra un fichero se debe cerrar antes de terminar el programa con la función **close**.

4.2 Creando el mapa de memoria que representan los registros del procesador

Al crear el mapa de memoria se le solicita al sistema operativo que relacione las direcciones de memoria virtual con las direcciones de memoria física dentro de un rango. El llamado al sistema que permite

esto es *mmap* y tiene la siguiente sintaxis, para mayor información ver manual de usuario:

```
1 void *mmap (void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

Script 1.2: Llamado al sistema con *mmap*

nota:para poder usar *mmap* es necesario agregar la librería $< unistd.h >$

Si no se produce ningún error, *mmap* devuelve un puntero al área de memoria mapeada, en cambio si se produce un error *mmap* devuelve -1.

ahora se procede a adicionar la parte de mapeo al código así:

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <sys/mman.h> /*contiene la función para obtener el tamaño de cada página*/
4  #include <unistd.h>     /*librerias para mmap*/
5  #include <asm/ioctl.h>  /*libreria para trabajar con I/Os*/
6  int main()
7  {
8  int tam = sysconf(_SC_PAGESIZE); /*tamaño de las páginas de memoria*/
9  int file_descriptor;
10 int N1,N2;
11 int *mapeo;   /*declaración del puntero mapeo*/
12
13 file_descriptor=open("/dev/mem",O_RDWR|O_SYNC);
14  if (file_descriptor<0)
15       {
16          printf("error al abrir /dev/mem \n");
17       }
18 /************************ proceso de mapeo  ***************************/
19 mapeo=mmap(NULL,N2*tam,PROT_READ|PROT_WRITE,MAP_SHARED,file_descriptor,N1*tam);
20 /**********************************************************************/
21
22 munmap(mapeo,0xN2*tam);/*deshace el proceso de mapeo,los argumentos de esta función
23                         son la dirección inicial de memoria del mapeo y la longitud
24                          del mapeo que se quiere deshacer */
25 close(file_descriptor);
26 }
```

Script 1.3: Realización mapa de memoria con */dev/mem*

4.3 Calculando los valores de *Offset* y *Len*.

Para decidir que direcciones se van a mapear es necesario referirse al mapa de memoria del procesador que se encuentra en el manual de programación del mismo:

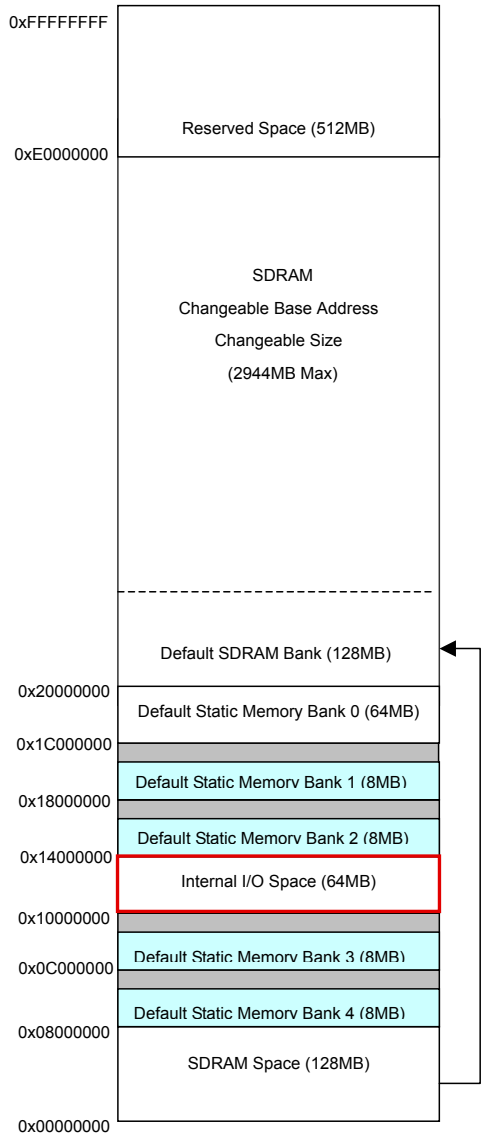


Figura 1.1: mapa de memoria del procesador.

FUENTE: Manual de programación del procesador.

Al ver el mapa de memoria, se nota que el rango de direcciones de los puertos de entrada salida I/O va desde la dirección **0x10000000** hasta la dirección **0x14000000**. Los registros PCDATS y PCDATC están dentro de este rango de direcciones, estos permiten establecer valores lógicos altos y

bajos respectivamente en el puerto C, la longitud de este registro es de 32-bits y cada bit tiene una tarea especifica. El led D6 esta conectado al procesador al puerto C bit 17 (PCDAT17) [2], por lo tanto los registros PCDATS y PCDATC controlan el encendido y apagado de dicho led, más exactamente en el bit 17. Por consiguiente solo este bit debe ser modificado estableciendo valores altos o bajos con el fin de prender o apagar el led. Una descripción más detallada de estos registros se puede encontrar en el manual de programación del procesador. La figura 1.2 muestra las direcciones en memoria de PCDATS y PCDATC.

Name	Description	RW	Reset Value	Address	Size
GPIO PORT C					
PCPIN	PORT C PIN Level Register	R	0x00000000	0x10010200	32
PCDAT	PORT C Data Register	R	0x00000000	0x10010210	32
PCDATS	PORT C Data Set Register	W	0x????????	0x10010214	32
PCDATC	PORT C Data Clear Register	W	0x????????	0x10010218	32
PCIM	PORT C Interrupt Mask Register	R	0xFFFFFFFF	0x10010220	32
PCIMS	PORT C Interrupt Mask Set Register	W	0x????????	0x10010224	32
PCIMC	PORT C Interrupt Mask Clear Register	W	0x????????	0x10010228	32
PCPE	PORT C PULL Enable Register	R	0x00000000	0x10010230	32
PCPES	PORT C PULL Enable Set Register	W	0x????????	0x10010234	32
PCPEC	PORT C PULL Enable Clear Register	W	0x????????	0x10010238	32
PCFUN	PORT C Function Register	R	0x00000000	0x10010240	32
PCFUNS	PORT C Function Set Register	W	0x????????	0x10010244	32
PCFUNC	PORT C Function Clear Register	W	0x????????	0x10010248	32
PCSEL	PORT C Select Register	R	0x00000000	0x10010250	32
PCSELS	PORT C Select Set Register	W	0x????????	0x10010254	32
PCSELC	PORT C Select Clear Register	W	0x????????	0x10010258	32
PCDIR	PORT C Direction Register	R	0x00000000	0x10010260	32
PCDIRS	PORT C Direction Set Register	W	0x????????	0x10010264	32
PCDIRC	PORT C Direction Clear Register	W	0x????????	0x10010268	32
PCTRG	PORT C Trigger Register	R	0x00000000	0x10010270	32

Figura 1.2: GPIO puerto C.

FUENTE: Manual de programación del procesador.

En base a lo dicho anteriormente de debe escoger un rango que contenga los registros PCDATS y PCDATC, una de estas posibilidades es mapear el rango completo de los registros internos de entrada y salida, Entonces la dirección inicial será **0x10000000** y la final será **0x14000000**.

La dirección inicial (0x10000000) corresponde al offset. Éste debe ser múltiplo del tamaño de la página, de lo contrario se incurrirá en un error denominado *segmentation fault*. El tamaño de la página es 0x1000 si se desea verificar esta información la función *sysconf* que utiliza el macro *_SC_PAGESIZE* es útil para esta tarea.

[2] ver hoja de datos de la plataforma SIE.

```
1 int tam; // contendra el valor del tamamño de la página
2 tam = sysconf(_SC_PAGESIZE);
3 printf("el tamaño de la página es es%x \n ",tam);
```

Script 1.4: Conocer tamaño de la página

Ahora se debe averiguar si el *offset*(dirección inicial) es múltiplo del tamaño de página para poderlo usar en *mmap* de esta forma.

$$\frac{\text{dirección inicial}}{\text{tamaño página}} = \text{número entero} \tag{1.1}$$

$$\frac{0x10000000}{0x1000} = 0x10000 \tag{1.2}$$

Ahora para escoger el parámetro *LEN* se hace se hace el siguiente procedimiento.

$$\text{dirección final} - \text{dirección inicial} = LEN \tag{1.3}$$

$$0x14000000 - 0x10000000 = 0x4000000 \tag{1.4}$$

El parámetro *LEN* también debe ser múltiplo del tamaño de página esto se verifica así:

$$\frac{\text{LEN}}{\text{tamaño página}} = \text{número entero} \tag{1.5}$$

$$\frac{0x4000000}{0x1000} = 0x4000 \tag{1.6}$$

Lo siguiente es poner estos argumentos en el código.

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <sys/mman.h>
4 #include <unistd.h>
5 #include <asm/ioctl.h>
6 int main()
7 {
8 int tam = sysconf(_SC_PAGESIZE);
9 int file_descriptor;
10 int *mapeo;
11 file_descriptor=open("/dev/mem",O_RDWR|O_SYNC);
12  if (file_descriptor <0)
13       {
14           printf("error al abrir /dev/mem \n");
15       }
16 mapeo=mmap(NULL,0x4000000 ,PROT_READ|PROT_WRITE,MAP_SHARED, file_descriptor ,0x10000000);
17
```

```
18  munmap(mapeo,0x4000000 );
19  close(file_descriptor);
20  }
```

Script 1.5: Adición de valores *offset* y *len* al código

Ahora es posible escribir y leer en los registros que se encuentran dentro del rango previamente mapeado, estas operaciones se harán con un puntero.

El descriptor de fichero(fd) devuelto por el llamado al sistema *open* representa el fichero *mem* que contiene el mapa completo de memoria física del procesador, ahora bien el llamado al sistema *mmap* se encarga de dos cosas:

- A partir del descriptor de fichero derivar un mapa de memoria virtual que representa el mapa de memoria físico del procesador y es asequible al usuario.
- Delimitar dicho mapa virtual a un rango que el usuario establece.

El llamado al sistema *mmap* devuelve un puntero a la primera dirección de memoria del rango que el usuario delimitó, esta primera dirección es virtual pero equivale a la primera dirección de memoria física que el usuario estableció en su rango. La segunda dirección de memoria virtual equivale a la segunda dirección de memoria física y así sucesivamente, es decir, se ha conseguido establecer un sistema de referencia entre el mapa de memoria virtual que maneja el sistema operativo y el mapa de memoria físico del procesador.

Por otro lado se podría pensar que si existe una equivalencia entre los mapas físicos y virtuales la utilización de *mmap* sería innecesaria, pero el principal problema es que esta relación no es fija, es decir las equivalencias entre ambas cambian en función al tiempo y los procesos que se estén ejecutando. Debido a esto la utilidad de *mmap* radica en que es una herramienta para construir un sistema de referencia o equivalencia fijo entre los mapas virtuales y físicos.

El puntero que devuelve *mmap* contiene información que se resume en dos aspectos, la dirección a la que apunta y el valor del registro que está en dicha dirección. Ya que el puntero que devuelve *mmap* se encuentra en la primera dirección de memoria virtual del rango y que ésta es equivalente a la primera dirección de memoria física, es posible escribir o leer en ella. Para escribir en el registro siguiente de memoria virtual que equivale al registro siguiente de memoria física se debe saber cuál es la dirección de memoria virtual, que no es más que la dirección de memoria inicial+1. Aprovechando

este hecho es posible moverse a través de todo el rango de memoria y leer (obtener los valores de los registros que componen ese rango) o por otro lado escribir (cambiar los valores de los registros que componen ese rango).

4.4 Escribir y leer registros

Para escribir en una posición de memoria, se define un puntero a esa dirección y luego se modifica el valor que contiene esta así:

```
1 int *puntero;
2 puntero=dirección_de_memoria_deseada;/*ubicamos el puntero en la dirección
3                                       que se quiere escribir*/
4 *puntero=valor_que_queremos_escribir; /* cambiamos el valor del dato al cual apunta
5                                          el puntero*/
```

Script 1.6: Definición puntero

4.4.1 Hallar la posición en la cual se desea escribir

Como se vio antes en la figura 1.2 las direcciones en las cuales se debe escribir son **0x10010214** y **0x10010218**, cada una representa un registro (PCDATS y PCDATC). El mapa de memoria virtual empieza en la dirección del puntero que devuelve *mmap* (mapeo) y termina en la dirección (**mapeo+length-1**), la relación entre el mapa virtual y el mapa físico sería:

DIRECCIÓN VIRTUAL	**DIRECCIÓN FÍSICA**
mapeo+0x0000000	0x10000000
mapeo+0x0000001	0x10000004
mapeo+0x0000002	0x10000008
mapeo+0x0000003	0x1000000C
mapeo+0x0000004	0x10000010
.	.
.	.
.	.
mapeo+0x1000000	0x14000000

Tabla 1.1: Direcciones memoria virtual y memoria física

Es decir, si se quiere acceder a la posición física **0x10010214** se debería sumarle a la dirección **mapeo** el valor **0x4085** porque:

$$\frac{\text{Dirección del registro} - \text{Dirección inicial del rango mapeado}}{4} = \text{N} \tag{1.7}$$

Donde.

- N = Espacios de memoria recorridos a partir de la dirección inicial del rango para llegar a la dirección que se desea escribir.

Por lo tanto reemplazando.

$$0x10010214 - 0x10000000 = 0x10214 \tag{1.8}$$

Se procede a dividir en 4 ya que el paso entre dirección y dirección de memoria física es de 4.

$$\frac{0x10214}{4} = 0x4085 \tag{1.9}$$

Y si se quiere acceder a la posición que le sigue **0x10010218** se debe sumar a la dirección **mapeo 0x4086**.

Por último para poder encender el led se va a colocar en 1 el bit 17 del regísto PCDAT, esto se hace escribiendo 1 en el regísto PCDATS y para apagarlo se escribe 1 en el mismo bit del registro PCDATC. Ahora que se sabe en qué posición se debe escribir, por consiguiente,se procede agregar esta parte al código de la siguiente manera:

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <sys/mman.h>
4  #include <unistd.h>
5  #include <asm/ioctl.h>
6  int main()
7  {
8  int tam = sysconf(_SC_PAGESIZE);
9  int file_descriptor;
10 int *mapeo;
11 int *escritura;
12 int i;
13 file_descriptor=open("/dev/mem",O_RDWR|O_SYNC);
14  if (file_descriptor<0)
15        {
16           printf("error al abrir /dev/mem \n");
17        }
18 mapeo=mmap(NULL,0x4000000  ,PROT_READ|PROT_WRITE,MAP_SHARED, file_descriptor ,0x10000000);
19 while(2>1){
20 for(i=0;i<1000000;i++){
21 escritura=mapeo+0x4085;
22 *escritura=0x00020000;
23 }
24 for(i=0;i<1000000;i++){
25 escritura=mapeo+0x4086;
26 *escritura=0x00020000;
27 }
```

```
28  }
29
30  munmap(mapeo,0x4000000  );
31   close(file_descriptor);
```

Script 1.7: Código completo

De esta forma se finaliza el programa.

5 PREGUNTAS Y ACTIVIDADES DE PRUEBA

1. Describa el procedimiento que seguiría usted para poder leer el valor que se escribió en LED D6.

2. Encienda el led D6 cambiando los parámetros *LEN* y *offset*.

LABORATORIO 2

MÓDULO PWM

Sólo es inmensamente rico aquel que sabe limitar sus deseos.

FRANÇOIS MARIE AROUET

1 INTRODUCCIÓN

Comprender el proceso de configuración de registros en el procesador JZ4725 es muy importante, ya que por medio de ellos es posible habilitar o deshabilitar funciones, periféricos y demás recursos que éste posee internamente.

En esta guía se dan a conocer los pasos necesarios para la configuración de un modulo PWM interno, y parte del bloque *clock generation unit (CGU)* necesario para generar una señal de reloj que servirá de fuente para el funcionamiento del modulo PWM ambos bloques internos al procesador. Algunos de los parámetros que se controlan en los procesos de configuración de los bloques PWM y CGU son el ciclo de trabajo, frecuencia y periodo. En primera instancia se necesita adecuar un el PLL (phase-locked loop) que compone el CGU y es el encargado de generar la frecuencia del reloj de entrada para la señal PWM, posteriormente se seguirá a modificar registros que activan la señal PWM y controlan el periodo y el ciclo de trabajo de ésta.

2 OBJETIVOS

- Habilitar el PWM interno del procesador mediante la manipulación de registros.
- Utilización adecuada de la configuración de los registros para generar una señal PWM.
- Conocer la arquitectura y el funcionamiento del *clock generation unit*.
- Comprender el mecanismo para configurar un periférico.

3 MARCO TEÓRICO

El procesador de la plataforma SIE posee un bloque funcional llamado *Timer/Counter Unit*, una de sus funciones es generar señales PWM. Dicho bloque necesita para su funcionamiento de una fuente de reloj ya sea interna o externa, dentro de éstas se tienen tres posibilidades PCLK, EXTAL y RTCCLK. Adicionalmente el procesador cuenta con una unidad generadora de reloj (CGU *Clock Generation Unit*) que puede ser utilizada para proveer la señal de reloj necesaria para el bloque *Timer/Counter Unit*.

El *clock generation unit* es el bloque encargado de administrar la fuentes de reloj para todos los circuitos internos del procesador. Internamente esta compuesto de dos osciladores, un circuito PLL y circuitos divisores de frecuencia, estos componentes permiten generar diferentes fuentes independientes de reloj a diferentes frecuencias.

El bloque *Timer/Counter unit* es un contador de frecuencia y limite de conteo controlable que puede ser usado además para generar señales PWM y modificar tanto su frecuencia como su ciclo de trabajo. Este bloque trabaja en base a un contador de 16 bits de frecuencia de entrada establecida por la configuración de la fuente de reloj.

Para que la señal PWM sea correctamente recibida a la salida, el puerto al cual se envía esta debe ser correctamente configurado para esa función, es decir debe ser establecido para trabajar en la función alterna PWM. Cada uno de los puertos que posee el procesador son usados como puertos de entrada/salida o interrupción. Adicionalmente es posible configurarlos para otras funciones distintas a estas, esta propiedad se llama *alternate function* y es controlada por el registro PXFUN.

4 PROCEDIMIENTO

Los pasos a seguir para el desarrollo de la guía son los siguientes:

- Configuración de la fuente de reloj de entrada para el bloque *Counter Unit*.
- Configuración del bloque *counter unit*.
- Configuración del puerto D25 como PWM.

A continuación se muestra el procedimiento de manera más detallada.

4.1 Configuración de la fuente de reloj de entrada para el bloque *Counter Unit.*

Para generar la fuente de reloj de entrada para el *Counter Unit* se hace uso del bloque de generación de reloj y para configurarlo se utilizan los registros relacionados con éste:

- Registro CPCCR.

  Encargado de establecer la relación de división de la fuente de reloj, todas las fuentes de reloj

  del procesador poseen un reloj externo el cual es posible disminuir su frecuencia dividiéndolo por un número determinado.

- Registro CPPCR.

  Encargado de la configuración del PLL, Establece parámetros que determinan la frecuencia de

  salida de PCLK.

4.2 Configuración del bloque *counter unit.*

Para configurar el bloque *counter unit* se debe hacer lo siguiente:

1. Configurar el registro TCSR:
   a. Escribir en el campo INITL para establecer el nivel de salida inicial del PWM.
   b. Escribir en el campo SD para establecer el modo de apagado.
   c. Escribir en el campo PRESCALE para establecer la frecuencia de conteo de TCNT.
2. Configurar los registros TCNT, TDHR y TDFR:
   a. TCNT: Para establecer el valor inicial del contador.
   b. TDHR: Para establecer el ciclo de trabajo.
   c. TDFR: Para establecer el periodo de la señal PWM, realizando una comparación con TCNT.
3. Terminar la configuración del registro TCSR.
   a. Escribir en el campo PWM_EN para habilitar el contador como salida PWM.
   b. Escribir en el campo EXT_EN, RTC_EN y PCK_EN para seleccionar la fuente de reloj de entrada, solo uno de ellos puede ser activado.

4. Habilitar el contador por medio del registro TESR.

4.3 Configuración del puerto D25 como PWM.

El puerto D pin 25 se encuentra configurado por defecto cómo GPIO, pero puede ser usado como PWM [1] para esto se debe:

1. Establecer el registro PXTRG en 0.
2. Seleccionar la función PWM2 estableciendo el registro PDSEL en 0.
3. Establecer el registro PDFUN25 en 1 para activar la propiedad *alternate funtion*.

A continuación se muestra una de las posibles configuraciones para los registros anteriormente mencionados siguiendo el procedimiento general. **Nota**: la fuente de reloj en este caso es PCLK.

4.3.1 registro CPCCR

Se modifica para establecer el factor de división de PCLK.
NOTA: los campos que no sean útiles para este fin no deben ser modificados y así no provocar algún fallo en el sistema.

Por otro lado en la primera guía se comprendió el proceso de manipulación de registros a través de el llamado al sistema *mmap*. Por lo tanto el estudiante debe ser capaz de identificar los registros, mapear una región donde se encuentren estos y acceder a ellos mediante punteros. EL mapeo hecho en el desarrollo de esta guía es el mismo que se hizo en la guía de manipulación de registros por medio de *mmap*(guía 1).

El led D1 esta conectado a al pin PWM2 (PUERTOD25), para comprobar la correcta configuración de los atributos de la señal PWM se usa el canal PWM2 aprovechando el led D1 para efectos de visualización de su intensidad lumínica en relación con el ciclo de trabajo de la señal.
Al leer el registro CPCCR el valor inicial de este en hexadecimal es 0x40642220 que equivale en binario a 0100 0000 0110 0100 0010 0010 0010 0000, esta información se interpreta de la siguiente forma:
Para establecer el factor de división de PCLK en 8, HCLK en 4, MCLK en 4 y CCLK en 2 se escribe el valor 0x40643531 en el registro CPCCR, es decir:

[1] para mayor información ver manual de programación

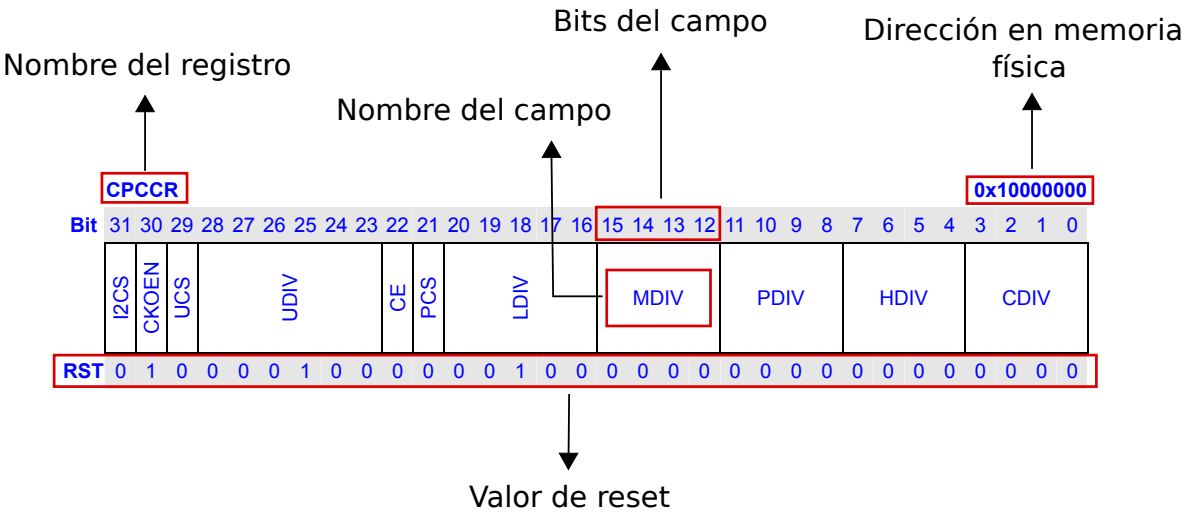


Figura 2.1: Registro CPCCR

FUENTE: Manual de programación del procesador.

I2CS=0
CKOEN=1
UCS=0
UDIV= 000000
CE=1
PCS=1
LDIV= 00100
MDIV=0010
PDIV=0010
HDIV=0010
CDIV=0000

Tabla 2.1: Interpretación de los valores iniciales de CPCCR

Se debe tener cuidado con la configuración de este registro ya que es obligatorio cumplir las siguientes condiciones:

- CCLK debe ser múltiplo entero de HCLK.
- La razón de división entre CCLK y HCLK no puede ser 24 ni 32.
- HCLK debe ser igual a MCLK o dos veces MCLK.
- HCLK y MCLK deben ser múltiplos enteros de PCLK.

Valores a escribir
I2CS=0
CKOEN=1
UCS=0
UDIV= 000000
CE=1
PCS=1
LDIV= 00100
MDIV=0011
PDIV=0101
HDIV=0011
CDIV=0001

Tabla 2.2: Valores a escribir

Valores leídos	Valores a escribir
I2CS=0	I2CS=0
CKOEN=1	CKOEN=1
UCS=0	UCS=0
UDIV= 000000	UDIV= 000000
CE=1	CE=1
PCS=1	PCS=1
LDIV= 00100	LDIV= 00100
MDIV=0010	MDIV=0011
PDIV=0010	PDIV=0101
HDIV=0010	HDIV=0011
CDIV=0000	CDIV=0001

Tabla 2.3: Registro CPCCR

El valor escrito en el registro CPCCR es 0x40643531 y con esto se configura la frecuencia del reloj PCLK. A continuación se procede a configurar el PLL.

4.3.2 registro de control del PLL(CPPCR)

El valor leído de este registro es:

0x1B000520 ⇒ 0001 1011 0000 0000 0000 0101 0010 0000

La dirección en mmap es: mapeo + 0x0004

La mínima relación entre M/N=9 entonces si M/N=10 y N=2 entonces M=20 y PLLM=18. Para mayor información referirse al manual de usuario. Entonces se configura el registro así:

Se escribe en CPPCR el valor 0x09000120 y con esto se termina de configurar el PLL.

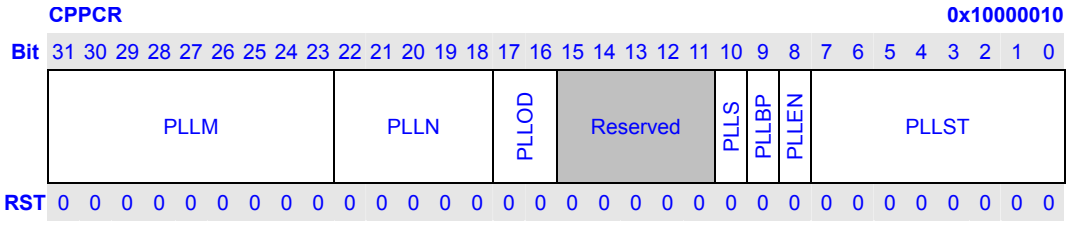


Figura 2.2: Registro CPPCR

FUENTE: Manual de programación del procesador.

Valores leídos	Valores a escribir
PLLM= 000110110	PLLM= 18=000010010
PLLN=00000	PLLN=00000
PLLOD=00	PLLOD=00
PLLS=1	PLLS=0
PLLBP=0	PLLBP=0
PLLEN=1	PLLEN=1
PLLST=00100000	PLLST=00100000

Tabla 2.4: Registro CPPCR

4.4 Configurando el timer/counter unit

4.4.1 Timer control register(TCSR2)

El valor leído de este registro es:

0x3000 $\Rightarrow$ 0011 0000 0000 0000

La dirección en mmap es: mapeo + 0x081B

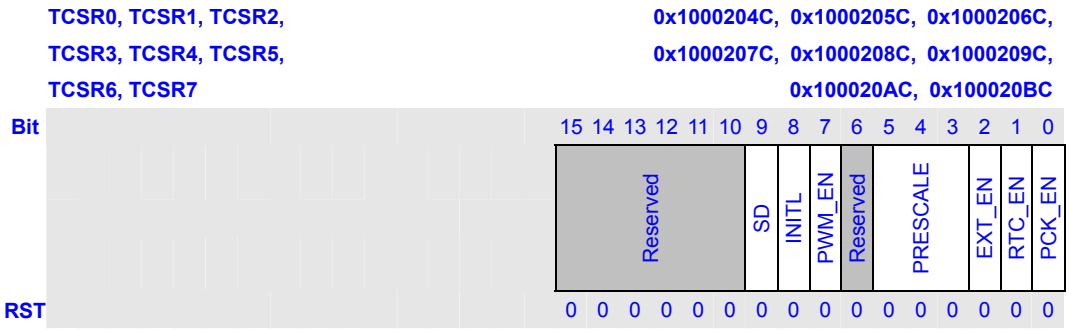


Figura 2.3: Registro TCSR2

FUENTE: Manual de programación del procesador.

Valores leídos	Valores a escribir
SD= 0	SD=0
INTL=0	INTL=0
PWM_EN=0	PWM_EN=0
reserved=0	reserved=0
PRESCALE=010	PRESCALE=101
EXT_EN=1	EXT_EN=1
RTC_EN=0	RTC_EN=0
PCK_EN=0	PCK_EN=0

Tabla 2.5: Registro TCSR2

Se escribe el valor 0x002C en el registro TCSR2.

4.4.2 Inicializando el *timer counter register*(TCNT2)

El valor leído de este registro es:

0x5f80 ⇒ 0101 1111 1000 0000 = 24448

La dirección en mmap es: mapeo + 0x081A

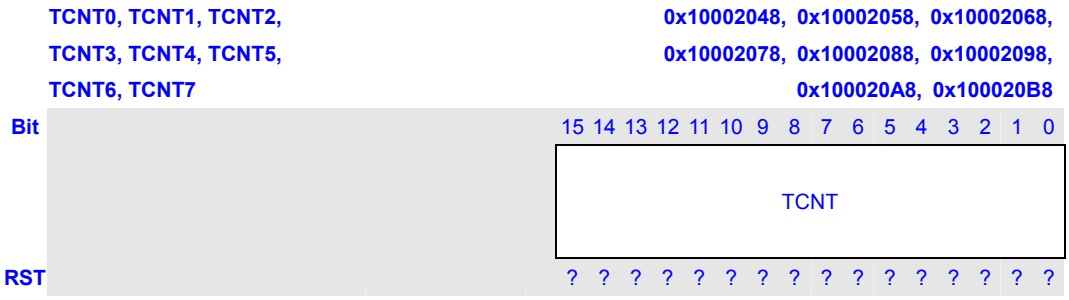


Figura 2.4: Registro TCNT2

FUENTE: Manual de programación del procesador.

Se estableció en 0 para que al iniciar la cuenta, en el primer periodo no se altere el ciclo de trabajo. El valor máximo de este registro es 65535.

4.4.3 Timer data full register(TDFR2)

El valor leído de este registro es:

0x059B ⇒ 0000 0101 1001 1011=1435

La dirección en mmap es: mapeo + 0x0818

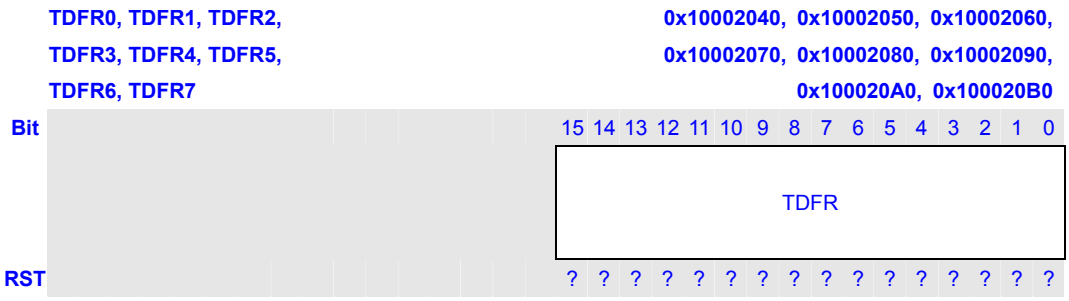


Figura 2.5: Registro TDFR2

FUENTE: Manual de programación del procesador.

Este registro se establece en el valor máximo para que el periodo de la señal PWM sea el periodo de la señal de reloj. A continuación se escribe en el registro TDFR2 el valor de 0xFFFF.

4.4.4 Timer data half register(TDHR2)

El valor leído de este registro es:

0x0000 ⇒ 0000 0000 0000 0000=0

La dirección en mmap es: mapeo + 0x0819

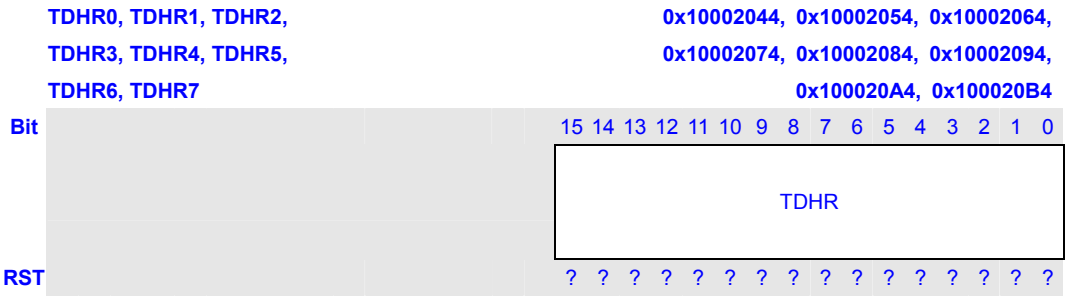


Figura 2.6: Registro TDHR2

FUENTE: Manual de programación del procesador.

Se escribe en este registro la mitad de lo que se escribió en TDFR2 es decir 0x8000.

4.4.5 Timer control register(TCSR2)

Este registro ya se había configurado pero debido a que en la hoja de datos del procesador se sugiere un orden determinado, es necesario volver a configurarlo. El valor leído de este registro es:

0x3000 ⇒ 0011 0000 0000 0000

La dirección en mmap es: mapeo + 0x081B

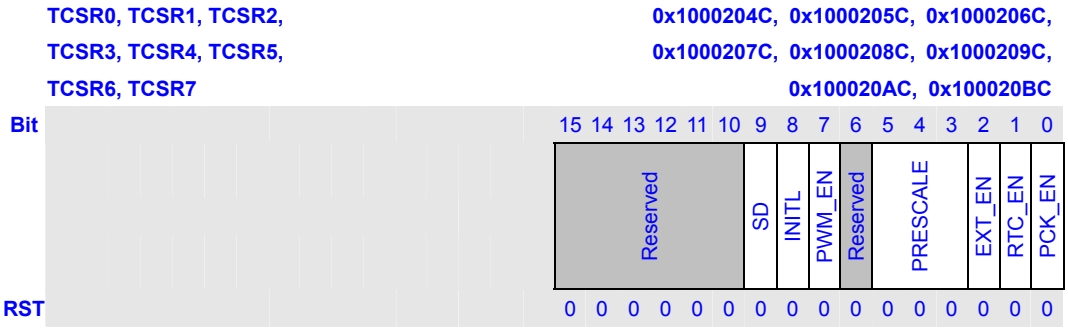


Figura 2.7: Registro TCSR2

FUENTE: Manual de programación del procesador.

Valores leídos	Valores a escribir
SD= 0	SD=0
INTL=0	INTL=0
PWM_EN=0	PWM_EN=1
reserved=0	reserved=0
PRESCALE=010	PRESCALE=101
EXT_EN=1	EXT_EN=0
RTC_EN=0	RTC_EN=0
PCK_EN=0	PCK_EN=1

Tabla 2.6: Registro TCSR2

Se escribe el valor 0x00A9 en el registro TCSR2.

4.4.6 Timer counter enable set register(TESR)

El valor leído de este registro es:

0x00 ⇒ 0000 0000

La dirección en mmap es: mapeo + 0x0805

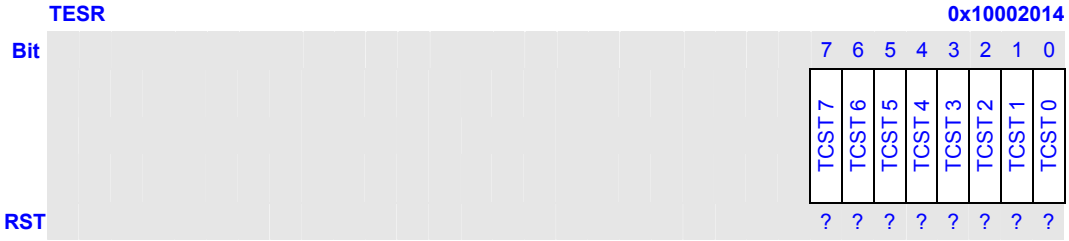


Figura 2.8: Registro TESR

FUENTE: Manual de programación del procesador.

Se escribe el valor 0x04 para habilitar el contador hacia arriba de TCEN2 en el registro TER.

4.5 Configuración de la función del puerto

Para configurar el puerto como salida PWM2 se necesita que se cumplan las siguientes condiciones:

PDFUN(25) = 1
PSEL(25) = 0
PTRG(25) = 0

4.5.1 Port function set register(PDFUNS)

El valor leído de este registro es 0x35fc0001= 0011 0101 1111 1100 0000 0000 0000 0001 y su dirección en mmap es: mapeo + 0x40D1

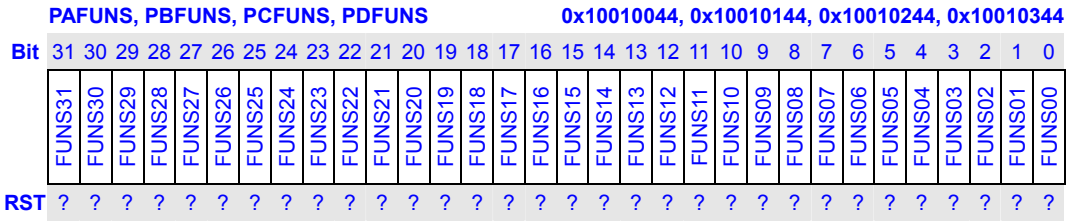


Figura 2.9: Registro PXFUNS

FUENTE: Manual de programación del procesador.

Se establece en el registro PDFUNS el valor 0x37FC0001.

4.5.2 Port select clear register(PDSELC)

El valor leído de este registro es 0x35fc0001= 0011 0101 1111 1100 0000 0000 0000 0001 y su dirección en mmap es: mapeo + 0x40D6.

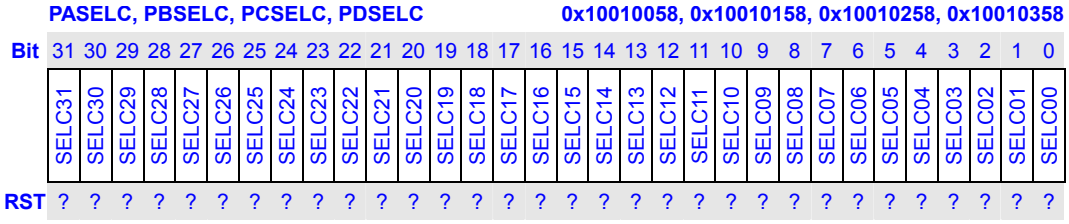


Figura 2.10: Registro PXSELC

FUENTE: Manual de programación del procesador.

Se escribe en PDSELC el valor 0x35fc0001.

4.5.3 Port trigger clear register(PDTRGC)

El valor leído de este registro es 0x35fc0001= 0011 0101 1111 1100 0000 0000 0000 0001 y su dirección en mmap es: mapeo + 0x40DE.

PATRGC, PBTRGC, PCTRGC, PDTRGC **0x10010078, 0x10010178, 0x10010278, 0x10010378**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TRIGC31	TRIGC30	TRIGC29	TRIGC28	TRIGC27	TRIGC26	TRIGC25	TRIGC24	TRIGC23	TRIGC22	TRIGC21	TRIGC20	TRIGC19	TRIGC18	TRIGC17	TRIGC16	TRIGC15	TRIGC14	TRIGC13	TRIGC12	TRIGC11	TRIGC10	TRIGC09	TRIGC08	TRIGC07	TRIGC06	TRIGC05	TRIGC04	TRIGC03	TRIGC02	TRIGC01	TRIGC00
RST	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Figura 2.11: Registro PXTRGC

FUENTE: Manual de programación del procesador.

Se escribe en PDTRGC el valor 0x35fc0001.

5 PREGUNTAS Y ACTIVIDADES DE PRUEBA

1. ¿ Que registros se deben modificar para variar el ciclo de trabajo ?
2. En esta practica se empleo la salida 2 del PWM. ¿ como cree usted que se deberían modificar los registros para habilitar cualquier otra salida ya que se dispone de 8 salidas ?
3. Genere una señal PWM con una fuente reloj diferente a PCLK

LABORATORIO 3

INTERFAZ DE COMUNICACIÓN ENTRE EL PROCESADOR Y EL FPGA

El hardware es lo que hace a una máquina rápida; el software es lo que hace que una máquina rápida se vuelva lenta

Craig Bruce

1 INTRODUCCIÓN

La plataforma SIE cuenta con un procesador(JZ4725) como componente central, y un dispositivo FPGA que ofrece la ventaja de poder implementar diversidad de periféricos en ésta, teniendo así una herramienta con un potencial didáctico importante.

El fin de implementar diversos periféricos en el FPGA es comunicarlos con el procesador y así poder ejercer control sobre ellos y administrar sus recursos, estimulando en el estudiante la concepción de este proceso. El procesador posee un bus lo conecta con el FPGA que sirve de canal para enviar o recibir datos y además necesita de un circuito que adecue la información antes de ser recibida por el FPGA. Se puede usar sin ningún circuito adicional pero no es aconsejable por dos razones:

- Metaestabilidad, porque el procesador y el FPGA poseen diferentes dominios de reloj.
- Posible choque entre datos provenientes del FPGA y salientes del procesador.

Para el primer inconveniente que es la metaestabilidad, se puede usar un circuito sincronizador de dominios de reloj, y para el segundo, debido a que el bus de datos es bidireccional y cabe la posibilidad que el procesador realice una operación de escritura de datos sobre el bus y el FPGA también, por lo tanto se producirá un choque de datos, se necesita un circuito que habilite el flujo de datos entrantes al procesador solo cuando éste realiza una operación de lectura. Esta práctica se concentrará en describir

e implementar el circuito de sincronización y habilitación de flujo de datos que va entre el procesador y el FPGA, es decir, en el sistema de comunicación en sentido procesador-FPGA.

2 OBJETIVOS

- Describir e implementar un circuito que posibilite la comunicación en el sentido procesador-FPGA sin problemas de metaestabilidad.
- Describir e implementar un circuito que evite el choque de datos en la interfaz de comunicación.

3 MARCO TEÓRICO

Una de las funciones más importantes del procesador es administrar periféricos siendo estos un recurso del sistema de cómputo. En la arquitectura de la plataforma SIE cada uno de los periféricos conectados al procesador le corresponde un rango de direcciones de memoria que lo identifican, un circuito especial llamado *address decoder* se encarga de asignar dichas direcciones. Cada periférico posee una señal llamada *chip select* (CS) que se encarga de indicar cuando el procesador quiere empezar a comunicarse con dicho periférico, la activación de esta señal le corresponde al *address decoder* quien actúa de acuerdo a la petición del procesador y a la dirección del periférico.

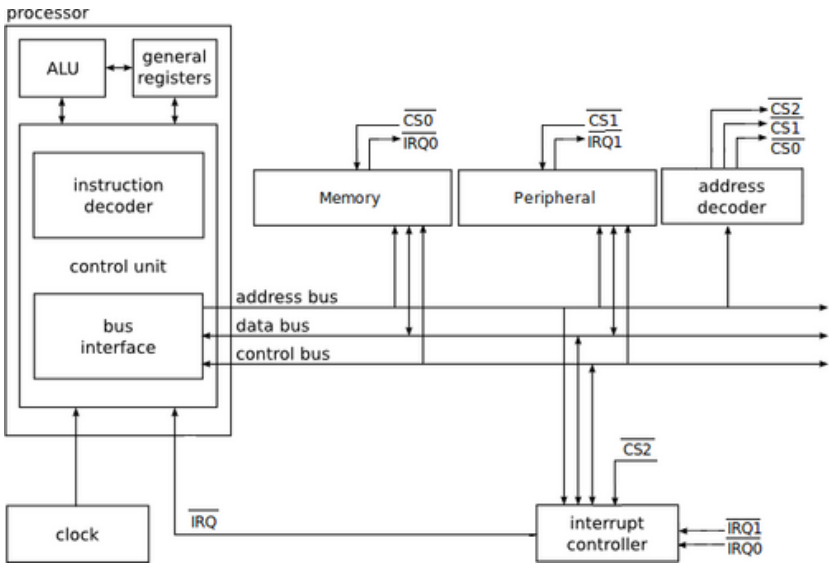


Figura 3.1: Esquema de comunicación del procesador.
FUENTE: [1].

3.1 Comunicación en sentido procesador-periférico

Cada periférico dentro de la plataforma SIE cuenta con su propia señal de *chip select*, para el caso del FPGA, ésta es *chip select* 2 (CS2). Si el procesador requiere comunicarse con un determinado periférico, debe enviar la dirección de memoria de dicho periférico a través del bus de direcciones al *address decoder* quien activa la señal *chip select* (CS) de éste. Solo un periférico puede establecer comunicación a la vez con el procesador, por lo tanto solo una señal de *chip select* puede estar activada al tiempo. Una vez la señal CS es activada el periférico puede empezar la comunicación e inmediatamente los demás pasan a un estado de alta impedancia para evitar daños físicos en los buses.

Ya establecida la comunicación, si el procesador requiere leer, éste activa la señal nRD y solo el periférico seleccionado toma el control del bus de datos. Si el procesador quiere escribir entonces éste activa la señal nRW y el procesador coloca la información que quiere en el bus de datos y el periférico la lee.

3.2 Comunicación en sentido periférico-procesador

Si el proceso de comunicación se hace en el sentido contrario, es decir, cuando un periférico requiere comunicarse con el procesador lo puede hacer de dos formas la primera es llamada *polling*, en ésta el

periférico en cuestión cambia alguno de sus registros internos para que el procesador al leerlo se de cuenta del cambio y atienda la petición de comunicación. Cabe destacar que este método requiere que el procesador esté preguntando periódicamente por este registro.

La segunda opción es usar interrupciones, la cual consiste en enviar una señal llamada IRQ(*interruption request*) al procesador con lo que se consigue que éste salga del flujo de ejecución del programa que esté ejecutando en ese instante y salte a una posición de memoria conocida como vector de interrupciones en la cual se encuentra una rutina llamada *interrupt service routine* (ISR) que contiene el código que se ejecutará de acuerdo al origen de la interrupción. Cabe anotar que la interrupción es solo un mecanismo para pedirle al procesador que se comunique con un periférico y que el control del bus de datos y el bus de control es del procesador, después de que la solicitud es aceptada por el procesador el proceso siguiente es el mismo de la comunicación procesador-periférico.

3.3 Uso del controlador de memoria externa (EMC)

El controlador de memoria externa EMC(*external memory controller*) es un bloque funcional interno del procesador que además de ser utilizado para administrar memorias, es usado para administrar periféricos diferentes. En el proceso de comunicación con un periférico son necesarias las señales nRD, nRW y CS2 que indican cuando se lee o escribe y cuando se escoge el periférico, es aquí donde el EMC es útil ya que provee esas señales facilitando una interfaz de control adecuada. En esta oportunidad no se entrará en detalle respecto al EMC, si se quiere revisar más a fondo remitase a el capítulo 3 del manual del programador del procesador xburst JZ4725.

Las señales que maneja el EMC y que se utilizarán son:
D[7:0] $\Longrightarrow$ señal de datos
RDWR $\Longrightarrow$ señal de escritura/lectura
WE0 $\Longrightarrow$ señal de escritura
CS2 $\Longrightarrow$ *chip select*

3.4 Arquitectura de comunicación

Además del EMC como proveedor de señales de control necesita un hardware de por medio que se encargue de tomar esas señales de control y de datos y someterlas a un protocolo o a unas reglas que permitan una comunicación segura y adecuada por las razones mencionadas anteriormente. El

hardware sugerido se muestra en la figura 3.2 y es implementado en el FPGA:

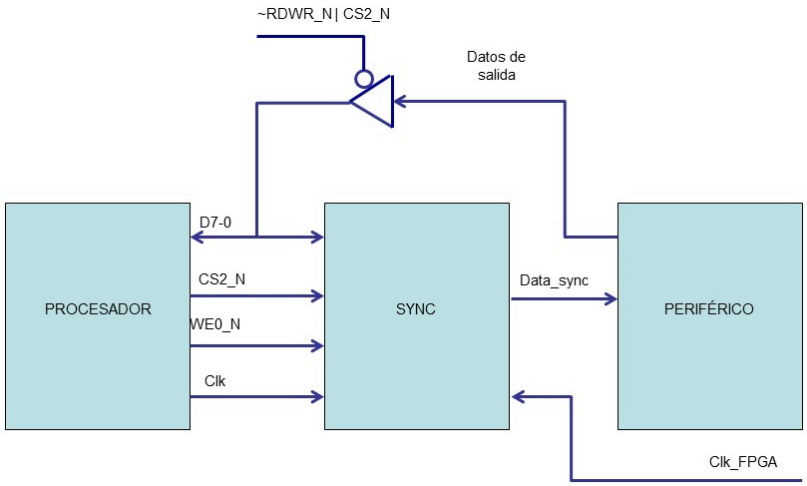


Figura 3.2: Interfaz de hardware entre el procesador y el FPGA.
FUENTE: Los autores.

Este circuito disminuye la posibilidad de meta-estabilidad al tener un sincronizador y evita el choque de datos al tener un *buffer* -triestado. Los detalles de esta figura serán expuestos más adelante.

3.5 Configuración del controlador de memoria

Para proveer las señales de control anteriormente mencionadas basta con configurar el EMC para que administre memorias estáticas. El EMC posee dos registros de configuración para la interfaz de memoria estática que son el *Static Memory Control Register* (SMCR1-SMCR4) y el *Static Bank Address Configuration Register* (SACR1-SACR4). Ambos se dejan en su valor por defecto ya que esta configuración permite administrar el periférico como si de una memoria estática normal se tratase dejándola en el modo normal, con ancho de bus de datos de 8 bits, y con los tiempos de espera máximos. Todas estas configuraciones son explicadas con detalle en el manual de usuario, en el capítulo referente a este laboratorio

Cabe aclarar que no se van a manejar memorias, simplemente se está usando la interfaz de memoria estática del EMC para proveer las señales de control(CS2, RDWR, WE0) a el periférico, ya que esta interfaz de control de memoria estática es fácil de manejar y ofrece lo necesario para administrar la interfaz de comunicación que se implementará en el FPGA. El EMC viene entonces listo para trabajar,

el siguiente paso es implementar el circuito sincronizador en el FPGA que permite la escritura y el *buffer* -triestado que habilita o deshabilita el flujo de datos hacia el procesador dependiendo de la situación.

4 PROCEDIMIENTO GENERAL

1. Implementación del circuito de sincronización para la comunicación en sentido procesador-FPGA.

El circuito de sincronización consta de:

- Bloque 1.

  Dos flip-flops tipo D encargados de generar una señal sincronizada de ambos relojes.

- Bloque 2.

  Representa un registro de salida para el domino de reloj del procesador.

- Bloque 3.

  Habilita el flujo de datos con el flanco de reloj del FPGA, pero siendo habilitado

  por la señal sincronizada de ambos relojes generada por el bloque 2.

2. Implementación de un circuito que evite el choque de datos.

- El circuito es un *buffer* triestado, que en las operaciones de escritura hechas por el procesador inhabilita el flujo de datos entrantes a éste.

3. Diseño e implementación de un circuito de prueba.

- Compuesto de un flip-flop para la entrada y un ejemplo de una posible descripción en VHDL de un circuito para verificar el funcionamiento de la interfaz.

4. Rutina de verificación.

- Encargado de habilitar la señal CS2, enviar datos al FPGA y recibir un dato de verificación.

A continuación se muestra de manera más detallada los pasos a seguir.

4.1 Implementación del circuito de sincronización para la comunicación en sentido procesador-FPGA.

El circuito de sincronización que se va a implementar es el equivalente al bloque SYNC en la figura 3.2 y se muestra en la figura 3.3. Con este circuito se soluciona el problema de meta-estabilidad.

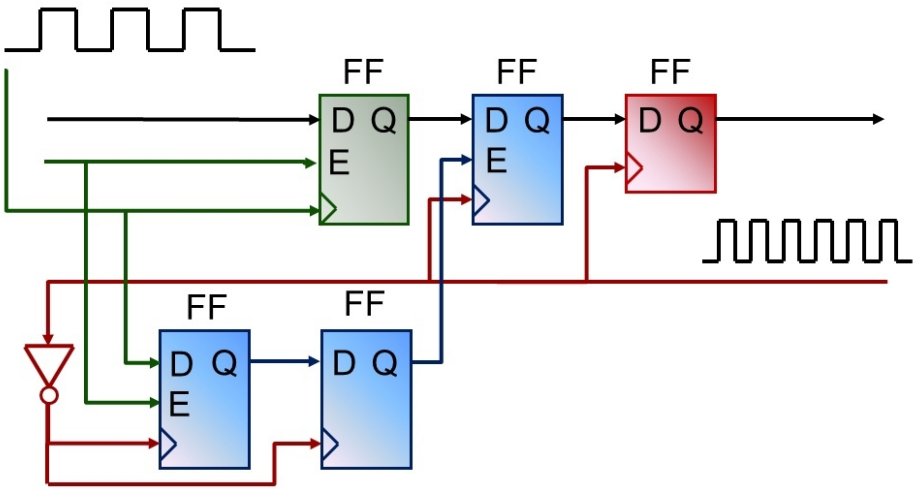


Figura 3.3: Circuito de sincronización de dominios de reloj.
FUENTE: [2]

4.1.1 Bloque 1

El bloque 1 se muestra en la figura 3.4. Éste consiste de dos flip-flops tipo D cuya entrada es NWE negada, debido a que esta señal indica cuando el procesador coloca datos en el bus, es decir, equivale a una señal sincronizada con las operaciones de escritura y por lo tanto actúa como señal de reloj del procesador en este caso.

NWE es activa baja, ésta indica cuando el procesador realiza una operación de escritura, pero estas operaciones pueden estar dirigidas a otros periféricos, la señal que asegura que una operación esta dirigida al FPGA es la señal de *chipselect* CS2, es por esto que se usa como señal negada de habilitación para asegurar que las operaciones de escritura son dirigidas al FPGA y evitando así el paso de datos no deseados al FPGA.

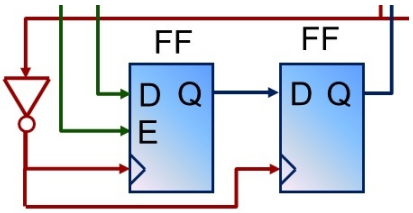


Figura 3.4: Bloque 1.

FUENTE: [2]

A continuación se presenta el código en VHDL que describe este bloque.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity bloque1 is
7      Port ( nwe0 : in  STD_LOGIC;
8             ncs2 : in  STD_LOGIC;
9             clk : in  STD_LOGIC;
10            E1 : out  STD_LOGIC);
11 end bloque1;
12
13 architecture Behavioral of bloque1 is
14 signal clk1: std_logic;
15 signal aux,aux1: std_logic;
16
17 begin
18 clk1<= not clk;
19 aux1<= not nwe0;
20 process(clk1,ncs2)
21 begin
22         if(clk1'event and clk1='1') then
23                 if(ncs2='0')then
24                         aux<=aux1;
25                 end if;
26         end if;
27 end process;
28
29 process(clk1)
30 begin
31         if(clk1'event and clk1='1') then
32                 E1<=aux;
33         end if;
34 end process;
35
36 end Behavioral;
```

Script 3.1: Descripción en VHDL del bloque 1

4.1.2 Bloque 2

El bloque 2 se muestra en la figura 3.5. como se puede observar este es un flip-flop D cuya entrada es la entrada de datos proveniente del procesador cuya señal de reloj y de *enable* son WE0 y CS2 respectivamente, por las razones expuestas en la descripción del bloque 1.

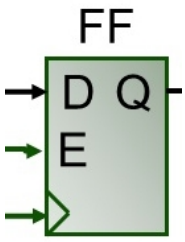


Figura 3.5: Bloque 2.

FUENTE: [2]

Su implementación en VHDL es la siguiente:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity bloque2 is
7      Port ( data : in  STD_LOGIC_VECTOR (7 downto 0);
8             nwe0 : in  STD_LOGIC;
9             ncs2 : in  STD_LOGIC;
10            buffer_data : out  STD_LOGIC_VECTOR (7 downto 0));
11 end bloque2;
12
13 architecture Behavioral of bloque2 is
14 begin
15
16 process(nwe0,ncs2)
17 begin
18         if(nwe0'event and nwe0='0') then
19                 if(ncs2='0')then
20                         buffer_data<=data;
21                 end if;
22         end if;
```

```
23  end process;
24
25  end Behavioral;
```

Script 3.2: Descripción en VHDL del bloque 2

4.1.3 Bloque 3

El bloque 3 se muestra en la figura 3.6. Como se puede ver este bloque consiste de un flip-flop tipo D cuya entrada es la salida de datos del bloque 2 y el reloj corresponde al reloj del FPGA, la particularidad de este bloque es que posee una entrada que sirve de habilitación, condicionando su funcionamiento a una señal sincronizada de ambos relojes. Esta señal sincronizada de los relojes del procesador y el FPGA corresponde a la salida del bloque 1.

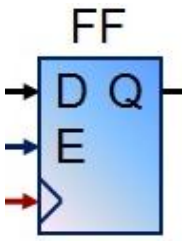


Figura 3.6: Bloque 3.

FUENTE: [2]

Su implementación en VHDL es la siguiente:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity bloque3 is
7       Port ( buffer_data : in  STD_LOGIC_VECTOR (7 downto 0);
8              E1 : in  STD_LOGIC;
9              clk : in  STD_LOGIC;
10             buffer_data1 : out  STD_LOGIC_VECTOR (7 downto 0));
11  end bloque3;
12
13  architecture Behavioral of bloque3 is
14
15  begin
16  process(clk,E1)
17  begin
```

```
18          if(clk'event and clk='1') then
19                  if(E1='1')then
20                          buffer_data1<=buffer_data;
21                  end if;
22          end if;
23  end process;
24
25  end Behavioral;
```

Script 3.3: Descripción en VHDL del bloque 3

4.2 Implementación de un circuito que evite el choque de datos.

Este circuito es un *buffer* -triestado mostrado en la figura 3.7, cuya señal de control es:

$$not(not(rdwr)or(cs2)) \tag{3.1}$$

Cuando se realiza una operación de lectura en el procesador la señal CS2 es activa-baja y la señal RDWR es activa-alta por lo tanto la señal de control es 1 y el *buffer* -triestado deja pasar los datos para ser leídos. Si en vez de esto se realiza una operación de escritura en el procesador la señal RDWR pasara automáticamente a activa-baja y CS2 también por lo que la señal de control es 0 y el *buffer* -triestado pasara a impedancia alta impidiendo así el paso de datos del FPGA hacia el procesador para evitar posibles choques de datos, y si se realizan operaciones de escritura o lectura dirigidos a otros dispositivos diferentes al FPGA la señal CS2 será 1 y la señal de control será siempre 0 y el *buffer* -triestado estará en alta impedancia.

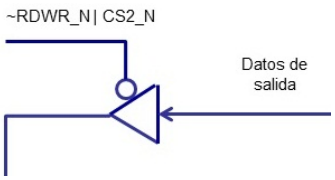


Figura 3.7: Buffer-triestado.

FUENTE: los autores.

su implementación en VHDL es la siguiente:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity tri_estate is
12     Port ( rd_bus : in  STD_LOGIC_VECTOR (7 downto 0);
13            ncs2 : in  STD_LOGIC;
14            rdwr : in  STD_LOGIC;
15            data : out  STD_LOGIC_VECTOR (7 downto 0));
16 end tri_estate;
17
18 architecture Behavioral of tri_estate is
19 signal enable:std_logic;
20 begin
21 enable<=(not(rdwr)) or (ncs2);
22 process(rd_bus,enable)
23 begin
24         if(enable='1') then
25                 data<="ZZZZZZZZ";
26         else
27                 data<=rd_bus;
28         end if;
29 end process;
30
31 end Behavioral;
```

Script 3.4: Descripción en VHDL del circuito buffer triestado

4.3 Diseño e implementación de un circuito de prueba.

El circuito de prueba se muestra en la figura 3.8.

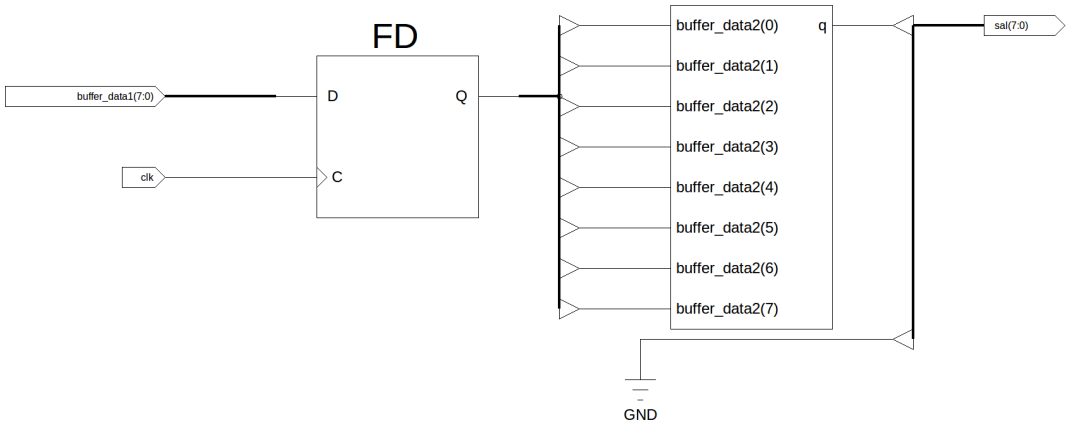


Figura 3.8: Circuito de prueba.

FUENTE: Los autores.

En la figura 3.8 se observa un *flip-flop* a la entrada junto con una descripción en VHDL que se muestra a continuación.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity circuito is
7      Port ( clk : in  STD_LOGIC;
8             buffer_data1 : in  STD_LOGIC_VECTOR (7 downto 0);
9             q : out  STD_LOGIC;
10            sal : out  STD_LOGIC_VECTOR (7 downto 0));
11 end circuito;
12
13 architecture Behavioral of circuito is
14 signal buffer_data2:std_logic_vector(7 downto 0);
15 begin
16
17 process(clk)
18 begin
19         if(clk'event and clk='1') then
20                 buffer_data2<=buffer_data1;
21         end if;
22 end process;
23
24 process(buffer_data2)
```

```
25 begin
26         case buffer_data2 is
27                 when "00000001" =>
28                         q<='1';
29                         sal<="00000101";
30                 when others =>
31                         q<='0';
32                         sal<="00000000";
33         end case;
34 end process;
35
36 end Behavioral;
```

Script 3.5: Descripción en VHDL del circuito de entrada y el circuito de prueba

El circuito toma la entrada de datos proveniente del procesador y si ésta coincide con el valor de 1 entonces la salida **q** es 1 y la salida **sal** es 5. Por otro lado si la entrada de datos es cualquier otro valor entonces ambas salidas son 0. la salida **q** es conectada al led del FPGA y la salida **sal** es conectada a la entrada del *buffer* -triestado de modo que si se escribe 1 el led prenderá, y si después de esto se realiza una operación de lectura, el valor leído será 5.

En esta instancia ya se ha completado la arquitectura de comunicación entre el procesador y el FPGA, lo siguiente es realizar unir cada uno de los bloques que se implementaron para comprobar el funcionamiento de la interfaz.

4.4 Interfaz completa

Luego de tener todos los bloques implementados se procede a unirlos en un solo proyecto por medio de *componets*. Una vez unidos los bloques se continua con la creación del archivo UCF.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7  entity sync_all is
8   Port ( clk: in  STD_LOGIC;
9          data:inout std_logic_vector(7 downto 0);
10         ncs2:in std_logic;
11         nwe0,rdwr:in  STD_LOGIC;
12         q: out  STD_LOGIC);
13
14 end sync_all;
15
```

```
16 architecture Behavioral of sync_all is
17
18 ---DECLARACION DE COMPONENTES
19
20 component bloque1
21 port( nwe0,ncs2 : in  STD_LOGIC;
22       clk : in  STD_LOGIC;
23       E1 : out  STD_LOGIC);
24 end component;
25
26 component bloque2
27 port(  data : in  STD_LOGIC_VECTOR (7 downto 0);
28        nwe0 : in  STD_LOGIC;
29        ncs2 : in  STD_LOGIC;
30        buffer_data : out  STD_LOGIC_VECTOR (7 downto 0));
31
32 end component;
33
34 component bloque3
35 port(  buffer_data : in  STD_LOGIC_VECTOR (7 downto 0);
36        E1 : in  STD_LOGIC;
37        clk : in  STD_LOGIC;
38        buffer_data1 : out  STD_LOGIC_VECTOR (7 downto 0));
39
40 end component;
41
42 component circuito
43 port(  clk:in std_logic;
44        buffer_data1 : in  STD_LOGIC_VECTOR(7 downto 0);
45        q: out  STD_LOGIC;
46        sal: out  STD_LOGIC_VECTOR(7downto 0));
47 end component;
48
49 component tri_state
50 port(  rd_bus:in std_logic_vector(7 downto 0);
51        ncs2:in std_logic;
52        rdwr:in  STD_LOGIC;
53        data : out  STD_LOGIC_VECTOR(7 downto 0));
54 end component;
55
56 signal buffer_data_signal,buffer_data1_signal:std_logic_vector(7 downto 0);
57 signal sal_signal:std_logic_vector(7 downto 0);
58 signal E1_signal:std_logic;
59 begin
60
61
62 instancia_bloque1: bloque1
63         port map(
64              clk=>clk,
65              nwe0=>nwe0,
```

```
66              ncs2=>ncs2,
67              E1=>E1_signal
68                  );
69
70  instancia_bloque2: bloque2
71                  port map (
72              data=>data,
73              ncs2=>ncs2,
74              nwe0=>nwe0,
75              buffer_data=>buffer_data_signal
76                          );
77
78  instancia_bloque3: bloque3
79                  port map (
80              buffer_data=>buffer_data_signal,
81              E1=>E1_signal,
82              clk=>clk,
83              buffer_data1=>buffer_data1_signal
84                          );
85
86  instancia_circuito: circuito
87                  port map (
88              buffer_data1=>buffer_data1_signal,
89              clk=>clk,
90              sal=>sal_signal,
91              q=>q
92                          );
93
94  instancia_tri_state: tri_state
95                  port map (
96              rd_bus=>sal_signal,
97              ncs2=>ncs2,
98              rdwr=>rdwr,
99              data=>data
100                         );
101 end Behavioral;
```

Script 3.6: Interfaz completa.

El circuito RTL es el siguiente.

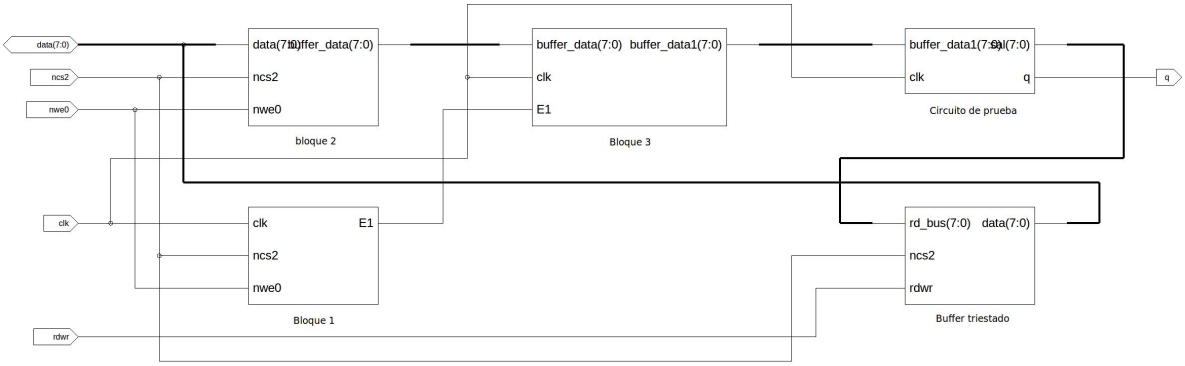


Figura 3.9: Circuito RTL de la interfaz completa

FUENTE: Los autores.

4.4.1 Asignación de pines

En este archivo se hace la asignación de pines del FPGA. La descripción de este archivo es la siguiente:

```
1  net clk loc=p38; # reloj de 50 MHz
2
3  net data(7) loc=p4;
4  net data(6) loc=p5;
5  net data(5) loc=p9;
6  net data(4) loc=p10;
7  net data(3) loc=p11;
8  net data(2) loc=p12;
9  net data(1) loc=p15;
10 net data(0) loc=p16;
11
12 net rdwr loc=p86;
13 net ncs2 loc=p69;
14 net nwe0 loc=p88;
15 net q loc=p44; # led D5
```

Script 3.7: Asignación de pines archivo.ucf

4.5 Rutina de verificación.

Además de la arquitectura de comunicación se necesita un programa de usuario que envíe datos y luego reciba un dato de verificación de acuerdo a la descripción del circuito de prueba. Antes de pasar

a la rutina de prueba es necesario tener en cuenta:

- El FPGA está conectado a la señal de habilitación CS2, si se accede al rango de direcciones de memoria física (manipulación de registros por medio de *mmap*) del banco de memoria *static memory bank* 2 se habilita esta señal y todos los demás periféricos pasan a impedancia alta. por lo tanto para escribir en el FPGA basta con escribir en cualquiera de las direcciones pertenecientes al rango *static memory bank* 2.
- Físicamente el bus escritura lectura es D7-D0 y se puede verificar en la hoja de datos de la plataforma SIE y el manual de programación del procesador.
- El pin que corresponde a la señal CS2 inicialmente se encuentra inhabilitada para esa función, por lo tanto se deben cambiar los registros necesarios con el fin de cambiar su función a *chip select*.

A continuación se muestra la rutina de verificación.

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4  #include <asm/ioctl.h>
5  #include <sys/mman.h>
6  int main()
7    {
8
9       int file_descriptor;
10      int *mapeo;
11      char *escritura;
12      int *lectura;
13      int a,opcion;
14
15 /*-------------------------------------------------------------------------------
16                                   CUERPO DEL PROGRAMA
17 -------------------------------------------------------------------------------*/
18          file_descriptor=open("/dev/mem",O_RDWR|O_SYNC);
19                  if (file_descriptor<0)
20                        {
21                          printf("error al abrir /dev/mem \n");
22                        }
23
24 mapeo=mmap(NULL,0x8000000,PROT_READ|PROT_WRITE,MAP_SHARED,file_descriptor,0x10000000);
25
26
27 /*-------------------------------------------------------------------------------
28                                   VERIFICACIÓN SEÑAL CS2
29 -------------------------------------------------------------------------------*/
```

```
30              lectura=mapeo+0x4050;
31                 a=*lectura & 0x4000000;
32                     if(a==0x0){
33                     printf("la señal CS2 esta deshabilitada ¿desea habilitarla? \n");
34                     printf("1.si                  \n");
35                     printf("2.no                 \n");
36
37           scanf("%d",&opcion);
38                     if(opcion==1){
39                     lectura=mapeo+0x4051;
40                     *lectura=0x4000000;
41                     printf("CS2 habilitada \n");
42                                                    }
43                     if(opcion==2){
44                     printf("CS2 no habilitada \n");
45                                }
46                                }
47  /*-------------------------------------------------------------------------------------------
48                                RUTINA DE ESCRITURA Y LECTURA
49  -------------------------------------------------------------------------------------------*/
50           escritura=(char *)(mapeo+0x1000004);
51           *escritura=0x1;
52           escritura=(char *)(mapeo+0x1000005);
53                     printf("el valor de  el registro es %x \n ",*escritura);
54  /*-------------------------------------------------------------------------------------------
55  -------------------------------------------------------------------------------------------*/
56           munmap(mapeo,0x8000000);
57           close(file_descriptor);
58       }
```

Script 3.8: Rutina de verificación

5 PREGUNTAS Y ACTIVIDADES DE PRUEBA

1. ¿ Porque cree que se debe utilizar una interfaz entre el procesador y el FPGA en la SIE ?.
2. Si el reloj del FPGA no fuera considerablemente mayor al del procesador, explique que consecuencias tendría en el desempeño del sincronizador.
3. Proponga un circuito o interfaz que haga posible la comunicación en el sentido FPGA-procesador.
4. Implemente un circuito en el FPGA para verificar que una secuencia de datos fue enviada correctamente.

Bibliografía

[1] Página qi-hardware. `http://en.qi-hardware.com/wiki/SIE`.

[2] Salamanca, William. *Tomada de la asignatura Procesadores incluida en el pénsum ingenieria electronica*. 2011.

LABORATORIO 4

INTRODUCCIÓN AL MANEJO DE PERIFÉRICOS EXTERNOS

Quien no ama su trabajo, aunque trabaje todo el día es un desocupado.

FACUNDO CABRAL

1 INTRODUCCIÓN

En esta instancia se han adquirido conocimientos en el trabajo con registros, el uso de éstos para controlar funciones en el procesador y periféricos externos a éste. También fue posible implementar un bloque que permitió la comunicación adecuada entre el procesador y periféricos implementados en el FPGA. Todo esto proporciona las herramientas necesarias para implementar y acoplar periféricos externos a la tarjeta SIE.
La ventaja de poder adicionar periféricos externos a la plataforma SIE radica en:

- Agregar funcionalidad a la plataforma, debido a que dichos periféricos cumplen tareas adicionales o complementarias que aumentan el rango de aplicaciones del sistema.
- Reforzar conceptos que en el momento de la implementación se deben tener en cuenta, entre ellos la capacidad de corriente, señales mixtas, interferencias magnéticas, señales de ruido, asignación de pines etc que deben ser revisados a la hora de agregar nuevos periféricos.
- Motivar al estudiante en un aspecto importante como el diseño e implementación de circuitos impresos, debido a que dichos periféricos necesitan de un soporte físico que permita las conexiones entre sus distintos bloques y etapas que garanticen su funcionamiento y conexión con la plataforma SIE.

Esta guía pretende establecer una base en el proceso de acople de periféricos externos a la plataforma SIE, usando el procesador como componente central de control, el FPGA como intermediario y circuito complementario.

2 OBJETIVOS

- Agregar *hardware* externo a la tarjeta SIE.
- Realizar un contador en un *display* de 7 segmentos.

3 MARCO TEÓRICO

El periférico externo a agregar en esta oportunidad es un *display* 7 segmentos (*pmod ssd* de la empresa *digilent*)que se muestra en la figura 4.1.

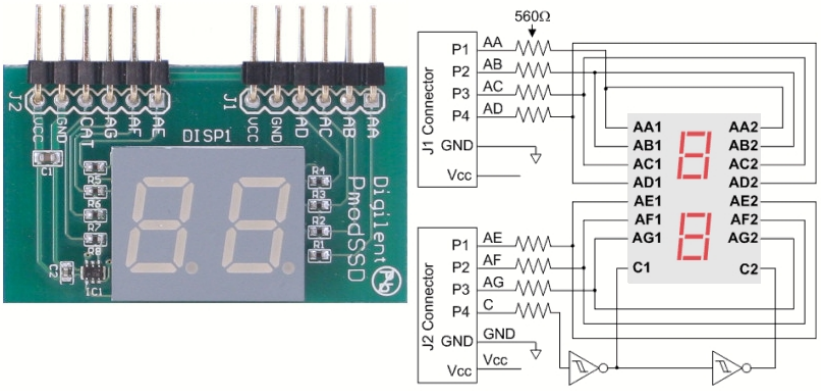


Figura 4.1: *display* de siete segmentos

FUENTE: hojas de datos del *display*.

Este dispositivo posee dos módulos siete segmentos que a su vez se componen de leds individuales. Los ánodos de dichos leds son compartidos por ambos módulos, y lo que decide cual módulo se enciende es un cátodo común. Dependiendo del estado lógico del cátodo se habilita el encendido de uno de los dos módulos siete segmentos.

Debido a que solo un módulo puede estar encendido a la vez, se debe hacer variar la señal del cátodo de un estado lógico alto a otro bajo rápidamente y así dar la impresión de que ambos están encendidos. La frecuencia de transición de un estado lógico a otro del cátodo debe ser controlada, debido a que si es muy baja el usuario notará los cambios de un módulo a otro, y si es muy alta se generaran transitorios indeseados y comportamientos impredecibles del dispositivo.

La señal del cátodo debe permanecer en los niveles lógicos alto y bajo la misma cantidad de tiempo del periodo siendo el periodo máximo 16 ms y el periodo mínimo 1 ms. Por lo tanto si se establece el periodo de la señal del cátodo en 16 ms la señal del cátodo debe estar en estado lógico alto 8 ms y en

estado lógico bajo 8 ms todo esto se muestra en la figura.

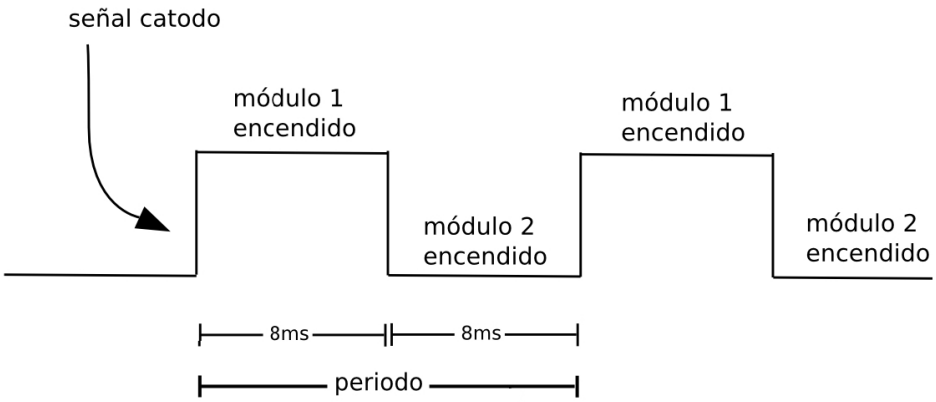


Figura 4.2: diagrama de tiempos de la señal del catodo

FUENTE: Los autores.

Por otra parte el FPGA de la plataforma SIE cuenta con 27 puertos digitales de entrada/salida que son asequibles externamente, es decir, estos puertos representan un canal de comunicación que puede ser usado para controlar el *display* siete segmentos. Por consiguiente el FPGA se usa para dos cosas:

1. Establecer un canal de comunicación entre el procesador y el *display*.
2. Implementar un circuito complementario cuya tarea es cambiar rápidamente de estado el cátodo para alternar el encendido de los módulos del *display* y así la visualización sea correcta.

La actividad propuesta consiste en generar un conteo de 0-99 visible en el *display* usando procesador y al FPGA para esta tarea.

4 PROCEDIMIENTO GENERAL

Para el desarrollo de esta guía se debe realizar el siguiente procedimiento.

1. Descripción del hardware necesario en el FPGA para el funcionamiento del periférico.

- Se va a implementar un divisor de frecuencia, un decodificador y un circuito para manejar el cátodo del *display*.

2. Adición del nuevo periférico a la tarjeta SIE.

- Selección de los pines a utilizar del FPGA para agregar el *display* siete segmentos.

3. Envío datos desde el procesador al FPGA.

- Ejecución de programa en el procesador para generar la secuencia de datos.

4.1 Descripción de hardware en el FPGA.

Con el fin de controlar el periférico que se implementará en el FPGA por medio del procesador, se necesita el circuito sincronizador de la figura 3.3 añadiendo después el bloque de circuito que se describe en el *script* 4.1 el nuevo diagrama de bloques es el siguiente.

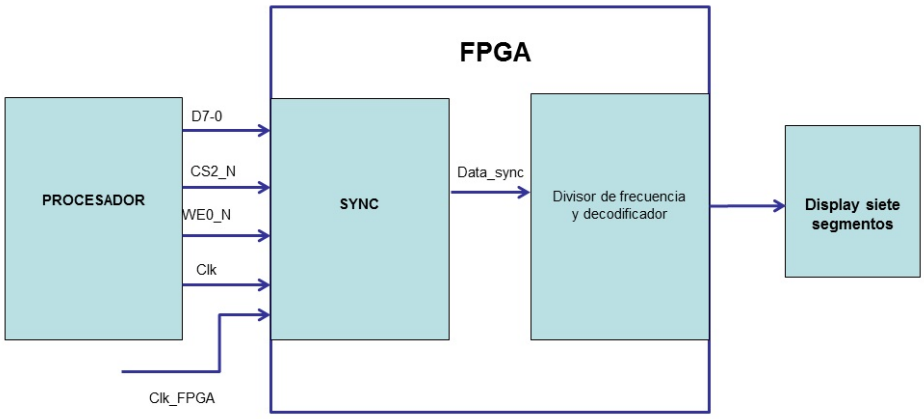


Figura 4.3: diagrama de bloques

FUENTE: Los autores.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity circuito is
7      Port (clk:in std_logic;
8            cat:out std_logic;
9            buffer_data1 : in  STD_LOGIC_VECTOR(7 downto 0);
10           salida : out  STD_LOGIC_VECTOR (6 downto 0));
11 end circuito;
12
13 architecture Behavioral of circuito is
14 signal buffer_data2:std_logic_vector(7 downto 0);
```

```
15  signal buffer_data3:std_logic_vector(3 downto 0);
16  signal cont:std_logic_vector(19 downto 0);
17  signal cat_signal,ent,buff_ent:std_logic;
18
19  begin
20  process(clk)
21  begin
22
23  if(clk'event and clk='1') then
24
25  buffer_data2<=buffer_data1;
26  end if;
27  end process;
28  ----------divisor de frecuencia-------------
29  process (clk,cont)
30  begin
31     if(clk'event and clk='1') then
32           if (cont="00000010011100010000") then
33           cont<="00000000000000000000";
34           else
35           cont<=cont+'1';
36           end if;
37     end if;
38  end process;
39  ------------------------------------------
40  -------------control del catodo-----------
41  cat_signal<='1' when cont="00000010011100010000" else
42                '0';
43
44  buffer_data3<=buffer_data2(7 downto 4) when buff_ent='1' else
45                buffer_data2(3 downto 0);
46
47  process (clk,cat_signal)
48  begin
49        if(clk'event and clk='1') then
50            if(cat_signal='1') then
51        buff_ent<=ent;
52            end if;
53        end if;
54  end process;
55
56  ent<=not buff_ent;
57  cat<=buff_ent;
58  ------------------------------------------
59  -------------decodificador----------------
60  with buffer_data3 select
61  salida<= "1111110" when "0000",
62      "0110000" when "0001",
63           "1101101" when "0010",
64           "1111001" when "0011",
```

```
65            "0110011" when "0100",
66            "1011011" when "0101",
67            "1011111" when "0110",
68            "1110000" when "0111",
69            "1111111" when "1000",
70            "1111011" when "1001",
71            "1000111" when others;
72  ------------------------------------------------
73  end Behavioral;
```

Script 4.1: bloque de la descripción del periférico

En el anterior *script* se describió un divisor de frecuencia, un decodificador y el control del cátodo para que este variando continuamente entre los dos *displays*.

4.2 Adición del nuevo periférico a la SIE

Observando el hoja de datos de la tarjeta se ve que pines se pueden utilizar para agregar el periférico.

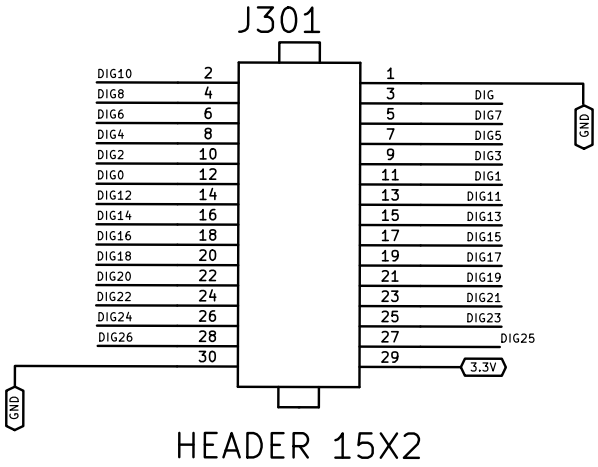


Figura 4.4: pines de salida FPGA

FUENTE: Hoja de datos plataforma SIE.

Se configura la distribución de pines para el periférico. Esta configuración es un ejemplo de una posible configuración del archivo UCF.

```
1  net clk loc=p38; # reloj de 50 MHz
2
3  net data(7) loc=p4;
4  net data(6) loc=p5;
5  net data(5) loc=p9;
```

```
net data(4) loc=p10;
net data(3) loc=p11;
net data(2) loc=p12;
net data(1) loc=p15;
net data(0) loc=p16;

net cat loc=p22;#dig25 catodo
net salida(0) loc=p24;#dig23
net salida(1) loc=p27;#dig21
net salida(2) loc=p33;#dig19
net salida(3) loc=p35;#dig17
net salida(4) loc=p40;#dig15
net salida(5) loc=p47;#dig13
net salida(6) loc=p49;#dig11

net rdwr loc=p86;
net ncs2 loc=p69;
net nwe0 loc=p88;
```

Script 4.2: Asignación de pines siete segmentos

4.3 Envío de datos desde el procesador al FPGA

Utilizando el script 3.8 visto en el tercer laboratorio se va a se añadir el siguiente *script* en la rutina de escritura y lectura, Para poder hacer el conteo en el procesador y ser luego visualizado en el *display*.

```
int i,n;
escritura=(char *)(mapeo+0x1000004);
for(n=0;n<10;n++){

for(i=0;i<10;i++){

*escritura=i+(16*n);
for(e=0;e<5000000;e++){;}
}
}
```

Script 4.3: contador para el siete segmentos

5 PREGUNTAS Y ACTIVIDADES DE PRUEBA

1. Diseñe e implemente una placa de circuito impreso que permita la conexión del *display* 7 segmentos .

2. ¿ Es posible manejar el cátodo desde el procesador ? y si es posible implementelo. ¿ Que concecuencias tendria en la visualización en el *display*?

LABORATORIO 5

INTERRUPCIONES

Sólo una cosa vuelve un sueño imposible: el miedo a fracasar.

PAULO COELHO

1 INTRODUCCIÓN

Como se habló en la guía 3(interfaz de comunicación procesador-FPGA) cuando un periférico requiere comunicarse con el procesador lo puede hacer de dos formas:

- Polling(encuesta) en donde el periférico cambia alguno de sus registros, luego el procesador nota este cambio y de acuerdo a este ejecuta una rutina. Este método involucra que el procesador debe preguntar constantemente por el registro y esto implica consumo de recursos.
- Interrupciones de hardware, en donde el periférico genera una señal eléctrica que va al procesador por un puerto de entrada específico dedicada a recibir interrupciones, esta señal comunica al procesador que se ha generado un petición por parte de un periférico.

La ventaja de las interrupciones de hardware radica en que el procesador es libre de atender otros procesos mientras la interrupción no se ha generado.

Las fuentes de interrupción de hardware van desde presionar una tecla hasta señales de un disco duro o una impresora. En el desarrollo de la guía se dará a conocer como se puede generar una interrupción y se mostrará como el procesador interrumpe el proceso que está atendiendo para atender la ISR.

2 OBJETIVOS

- Manejar interrupciones generadas desde cualquier periférico en este caso el FPGA.
- Conocer el funcionamiento, para qué son importantes, cómo se generan y para qué sirven las interrupciones en el procesador JZ4725 de la tarjeta SIE.

3 MARCO TEÓRICO

La señal de interrupción generada por un periférico es llamada IRQ(*iterrupt request*), Una vez ésta ha alcanzado el procesador, es identificada(quien la genero) y de acuerdo a su fuente es atendida, la rutina que el procesador este ejecutando en ese momento se interrumpe para atender la interrupción. Existe un código especial llamado rutina de servicio a la interrupción (*interrupt service routine* ISR) que determina que debe hacer el procesador de acuerdo a la interrupción, y una vez es atendida, el procesador retorna al proceso que fue interrumpido en un inicio.

El sistema operativo Linux maneja las interrupciones desde el kernel (núcleo). El kernel esta compuesto de módulos y los que se encargan de administrar periféricos son llamados drivers, éstos también se encargan de administrar las interrupciones. Luego por medio de un programa de usuario se puede utilizar dicho driver para tomar el control del comportamiento del procesador al atender interrupciones.

Las interrupciones se pueden generar por medio de una señal asíncrona indicando la necesidad de atención de un hardware o por un evento síncrono de software solicitando cambio en la ejecución. Las interrupciones por hardware hacen que el procesador guarde el estado en el que se encontraba antes de la interrupción y se dirija a la rutina del manejador de interrupciones (*interrupt handler*) encargado de manejar la solicitud de interrupción con una subrutina en el kernel.

4 PROCEDIMIENTO

Los pasos a seguir para el desarrollo de esta guía son:

1. Agregar y verificar el *driver* en el sistema operativo.
2. Implementar el periférico que va a generar la interrupción.
3. Generación y verificación de interrupciones.

5 Agregar y verificar el driver en el sistema operativo.

Lo primero es adecuar el procesador de la tarjeta SIE para que pueda manejar la interrupción que se va a generar desde el FPGA, Añadiendo un modulo al kernel. Para esto diríjase a la página de la asignatura y descargue los siguientes archivos a una misma carpeta.

- irq_fpga.c.
- Makefile.
- irq_fpga_main.
- irq.bit.
- montar.sh

Ya teniendo estos archivos en el computador se va a generar el archivo que corresponde al *driver*, para esto se utiliza el *Makefile* modificando el *PATH* donde se encuentra el compilador cruzado.

```
1  EXTRA_CFLAGS += -Wall -I.
2  CC              = mipsel-openwrt-linux-gcc
3  OPENWRT_BASE    = /home/PATH_TO_USER/trunk
4  KERNEL_SRC      = $(OPENWRT_BASE)/build_dir/linux-xburst_qi_lb60/linux-2.6.37.6/
5  CROSS_COMPILE = mipsel-openwrt-linux-
6
7  obj-m +=  irq_fpga.o
8  all: driver irq_fpga_main
9
10 driver:
11  make -C $(KERNEL_SRC) M=$(PWD) ARCH=mips CROSS_COMPILE=$(CROSS_COMPILE) modules
12 clean:
13    make -C $(KERNEL_SRC) M=$(PWD) ARCH=mips CROSS_COMPILE=$(CROSS_COMPILE) clean
14         rm -rf *.o main.o main irq_fpga.ko Modules.symvers irq_fpga_main
15
16 main: main.o
```

```
17
18 PREPROCESS.c = $(CC) $(CFLAGS) $(TARGET_ARCH) -E -Wp,-C,-dD,-dI
19 %.pp : %.c FORCE
20         $(PREPROCESS.c) $< > $@
```

Script 5.1: Makefile

Luego se ejecuta el *Makefile* para generar el archivo **irq_fpga.ko**.

```
1 make
```

Script 5.2: Generar archivo .ko

Ahora si se procede a copiar los archivos a la tarjeta.

```
1 scp irq_fpga.ko irq_fpga_main irq.bit montar.sh root@192.168.1.1:~
```

Script 5.3: subimos archivos a la SIE

Se agrega el driver al kernel de la tarjeta, utilizando un script que nos realiza esta tarea.

```
1 root@BenNanoNote:~# sh montar.sh
```

Script 5.4: adición del driver a la SIE

al ejecutar este script se verifica las siguientes rutas comprobando que se haya agregado el driver correctamente.

```
1 root@BenNanoNote:~# ls -l /dev/
2 root@BenNanoNote:~# cat /proc/devices
```

Script 5.5: verificación

en la ruta */proc/devices* se encontrará una lista de dispositivos dentro de los cuales debería estar el *driver* (irq_FPGA), es decir, deberá aparecer en la lista. En el siguiente *script* se muestra una lista de ejemplo.

```
1  Character devices:
2    1 mem
3    2 pty
4    3 ttyp
5    4 /dev/vc/0
6    4 tty
7    4 ttyS
8    5 /dev/tty
9    5 /dev/console
10   5 /dev/ptmx
11   7 vcs
12  10 misc
13  13 input
14  14 sound
```

```
15   29 fb
16   90 mtd
17  116 alsa
18  128 ptm
19  136 pts
20  252 irq_FPGA
21  253 ubi0
22  254 rtc
23
24  Block devices:
25  259 blkext
26   31 mtdblock
27  179 mmc
```

Script 5.6: verificación en */proc/devices*

Ya teniendo el driver listo para ser utilizado se va a cargar el circuito en el FPGA que va a generar la interrupción.

6 Implementar el periférico que va a generar la interrupción.

```
1 root@BenNanoNote:~# Xc3sprog irq.bit
```

Script 5.7: Adición de archivo al FPGA

Este circuito es importante por que representa el periférico que genera la interrupción, a continuación se muestra la hoja de datos de la SIE y se observa que pines son los necesarios para generar y para recibir la interrupción.

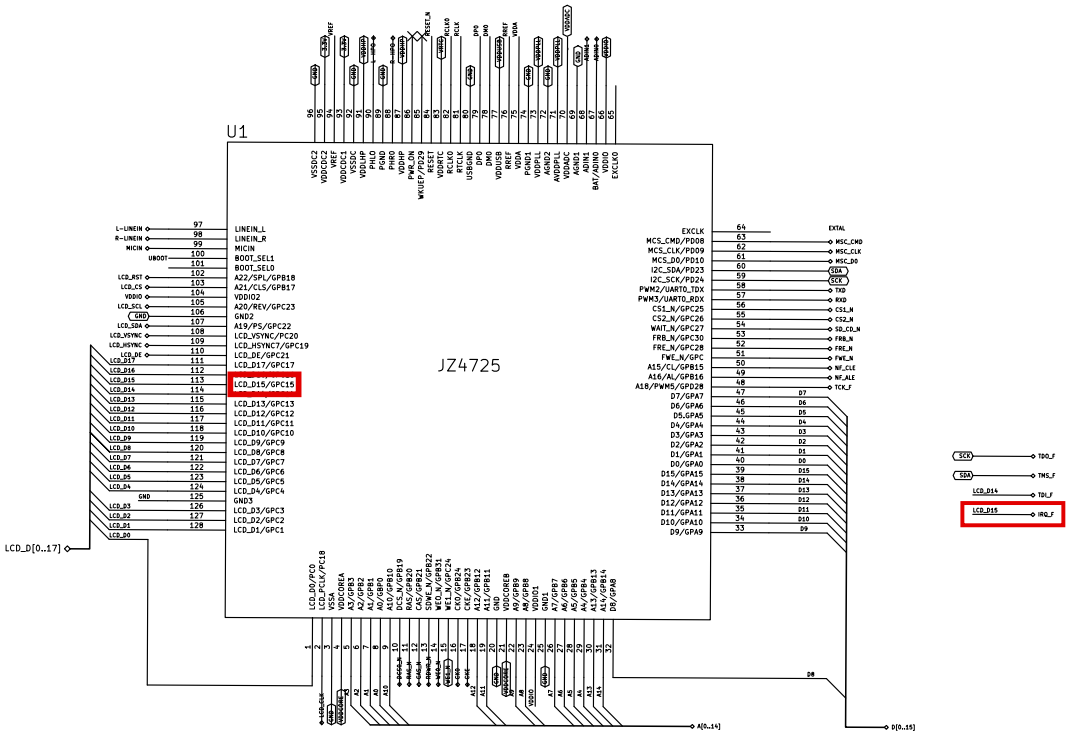


Figura 5.1: pin irq en FPGA

FUENTE: Hoja de datos plataforma SIE.

Apoyándose en el manual de programación del procesador se sabe que los puertos GPIO pueden ser configurados para recibir interrupciones pero solo uno de ellos está directamente conectado con el FPGA. El pin que va a recibir la interrupción es el LCD_D15 y se procede hacer lo mismo con el FPGA.

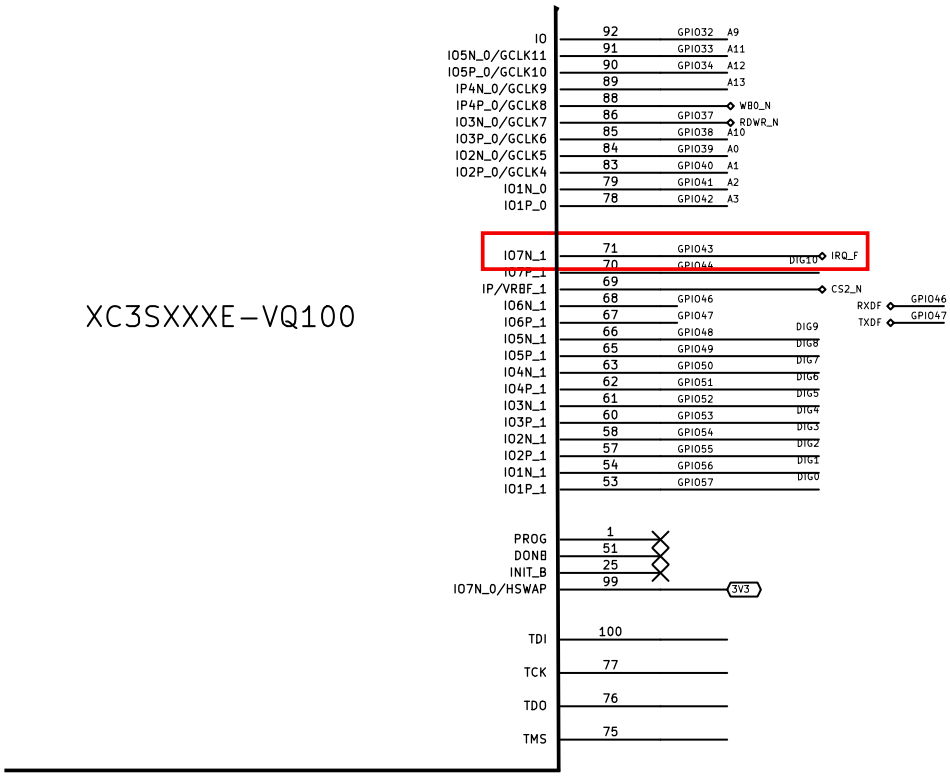


Figura 5.2: pin irq en JZ4725

FUENTE: Hoja de datos plataforma SIE.

Se observa que el pin GPIO43 va directamente al procesador para generar la interrupción. Ahora, ya se sabe cuales son los pines directamente involucrados en la interrupción.

Por último se implementa un circuito sencillo en el FPGA que consiste en una compuerta OR. Utilizando los pulsadores de la SIE es posible enviar una interrupción al procesador y enviar un 1 al led del FPGA, este último con el fin de verificar que cuando el led encienda se esta cambiando el estado a alto del pin IRQ.

7 Generación y verificación de interrupciones.

Ahora por último se revisa el archivo irq_fpga_main que tiene 3 opciones para manejar la interrupción. Conociendo las opciones de este ejecutable se procede a usarlo y en cada ejecución se mostrarán los

Comando	Descripción
./irq_fpga_main enable	Habilita la interrupción haciendo que el procesador este listo para recibirla desde el FPGA.
./irq_fpga_main disable	Deshabilita la interrupción, si se oprime cualquiera de los pulsadores, el procesador no cambiará hacia la subrutina del manejador de interrupciones, éste seguirá realizando su rutina habitual.
./irq_fpga_main read	Con este comando se leerá cuantas interrupciones se generaron al tener habilitada la interrupción, si está deshabilitada simplemente no contara ninguna interrupción.

Tabla 5.1: Opciones de habilitación de interrupción.

siguientes mensajes.

```
1 root@BenNanoNote:~# ./irq_fpga_main enable
2 Dispositivo disponible
3 Dispositivo habilitado para interrupciones
4 root@BenNanoNote:~#Ctrl + c
```

Script 5.8: irq_fpga_main enable

Al habilitar la interrupción se observa que en el led D6 del procesador se genera una intermitencia, la cual corresponde a su rutina habitual. Cuando se genera una interrupción (presionando el pulsador) el led D5 encenderá y el procesador saldrá de su rutina habitual para atender a la ISR que consiste en una rutina que enciende el led D6. El contador de interrupciones incrementará dependiendo del tiempo que se mantenga oprimido el pulsador, ahora se detiene el proceso y se verifica el conteo.

```
1 root@BenNanoNote:~#/.irq_fpga_main read
2 Dispositivo disponible
3 Interrupts = 346
4 root@BenNanoNote:~# ctrl + c
```

Script 5.9: ./irq_fpga_main read

Ahora con la opción para deshabilitar la interrupción se debe observar en el estado del contador antes y después de ejecutar el comando.

```
1 root@BenNanoNote:~# ./irq_fpga_main read
2 Dispositivo disponible
3 Interrupts = 346
4 root@BenNanoNote:~# ctrl + c
5
6 root@BenNanoNote:~#./irq_fpga_main disable
7 Dispositivo disponible
8 Dispositivo deshabilitado para interrupciones
```

```
9 root@BenNanoNote:~# ctrl + c
```

Script 5.10: ./irq_fpga_main disable

Después de haber deshabilitado las interrupciones se oprime el pulsador y se verifica que el conteo no haya aumentado.

```
1 root@BenNanoNote:~# ./irq_fpga_main read
2 Dispositivo disponible
3 Interrupts = 346
4 root@BenNanoNote:~# ctrl + c
```

Script 5.11: ./irq_fpga_main read para verificar

Independientemente del número de interrupciones se debe observar que no aumento ya que estaba deshabilitado.

8 PREGUNTAS Y ACTIVIDADES DE PRUEBA

1. ¿ Cuales son la razones de implementar las interrupciones en el procesador ?
2. ¿ Que otras fuentes de interrupción se pueden atender en el procesador ?
3. ¿ Que cree usted que ocurriría si se generaran varias interrupciones al mismo tiempo ?

LABORATORIO 6

PROYECTO BASE FINAL DE LABORATORIO ARQUITECTURA DE COMPUTADORES

1 INTRODUCCIÓN

Llegada esta instancia en el proceso de aprendizaje práctico de la asignatura arquitectura de computadores es conveniente consolidar los conocimientos y competencias adquiridas por el estudiante en guías anteriores, puntualmente en los campos de concepción, diseño e implementación. La mejor forma de lograrlo es a través de un proyecto que exija la aplicación de lo aprendido en el transcurso de la asignatura y además obligue a la implementación de *hardware* externo completando así la lista de competencias del estudiante.

El diseño e implementación de *hardware* externo es una tarea muy importante en el árbol de competencias de un estudiante, tarea que normalmente se delega a los sistemas de desarrollo que posee la universidad en los laboratorios de sistemas digitales. Por tal motivo este proyecto base quiere rescatar y exaltar la importancia de esta etapa del diseño para así contribuir a la formación cognitiva integral de los estudiantes.

Lo que se pretende es construir un proyecto base que sirva de modelo para proyectos posteriores y no confinar al estudiante a una idea en particular, incluso es posible que los proyectos individuales sean completamente diferentes a éste.

La idea principal de este proyecto es implementar un sistema de adquisición de datos. El trabajo se dividió en dos partes principales, tareas de *hardware* como un convertidor analógico-digital y tareas de software como una rutina que atienda los datos de llegada.

Por último, en este trabajo se diseñará el sistema de adquisición de datos, la interfaz de comunicación con el procesador y la rutina de atención de datos de llegada para visualizarlos en consola. Además

tareas complementarias cómo, la visualización de las formas de onda, mejoramiento de etapas, procesamiento de las señales y opciones de captura de datos se dejan abiertos para que el estudiante los aborde y desarrolle con la ventaja que este proyecto base le ofrece al brindarle una orientación inicial en su trabajo.

2 VISIÓN GENERAL DEL PROYECTO

Como se mencionó anteriormente el proyecto base consiste en un sistema de adquisición de datos, el diagrama general del proyecto se muestra en la siguiente figura:

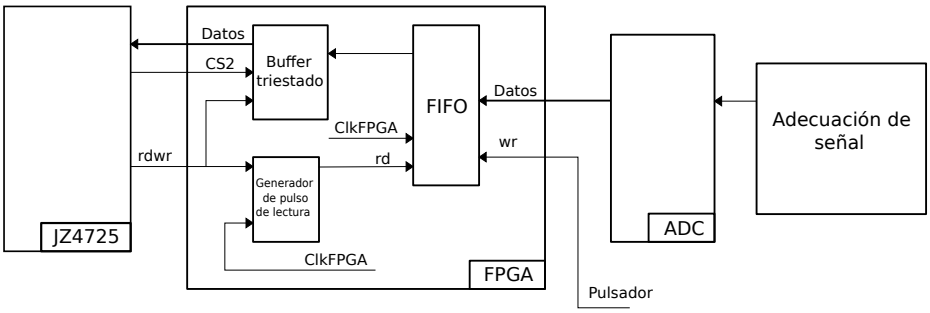


Figura 6.1: diagrama de bloques general

FUENTE: Los autores.

El desarrollo del proyecto se dividió en dos partes:

TAREAS DE *HARDWARE*:

- Interfaz de comunicación entre el convertidor de datos y el procesador de la tarjeta SIE.
- Implementación del convertidor de datos.
- Adecuación de señal.

TAREAS DE *SOFTWARE*:

- Lectura de datos de llegada desde el FPGA hacia el procesador a través de un programa de usuario en lenguaje C.

A continuación se procede al desarrollo de cada una de las tareas mencionadas anteriormente.

3 TAREAS DE HARDWARE

3.1 INTERFAZ DE COMUNICACIÓN ENTRE EL CONVERTIDOR DE DATOS Y EL PROCESADOR DE LA TARJETA SIE

Para este proyecto es necesario una interfaz que gestione el envío de datos desde el FPGA hacia el procesador, debido a dos problemas: la metaestabilidad y choque de datos.
Todos los requerimientos anteriores apuntan a un sistema de comunicación con un bloque de memoria FIFO como principal componente, debido a su capacidad de escritura y lectura a diferentes velocidades y por ser un componente de memoria, el ordenamiento y almacenamiento queda asegurado, evitando así posibles perdidas de datos.
En cuanto al choque de datos, la alternativa de solución es la implementación de un *buffer* triestado que controlará el paso de datos hacia el procesador. Los demás componentes de la interfaz son secundarios pero no menos importantes ya que ayudan al funcionamiento del sistema.

3.1.1 DESCRIPCIÓN DE LA INTERFAZ IMPLEMENTADA EN EL FPGA

La interfaz que se implementó con el fin de gestionar la llegada de datos del convertidor analógico-digital consta de:

I. Circuito generador de la secuencia de datos.

II. Circuito generador de pulso de lectura.

III. Memoria FIFO *single port*.

IV. *Buffer* triestado.

A continuación se explicarán con más detalle cada uno de los bloques que componen la interfaz.

I Circuito generador de la secuencia de datos

Este bloque simula el comportamiento del convertidor de datos analógico-digital con el fin de hacer pruebas preliminares y observar el comportamiento de la interfaz de manera más rápida y fácil.
El circuito en VHDL es una máquina de estados que cambia con el reloj de 50 MHz que posee el FPGA y una señal llamada **wr**(pulsador PB2) que permite controlar cuando se generan los datos.

El diagrama de estados de esta se muestra en la figura 6.2.

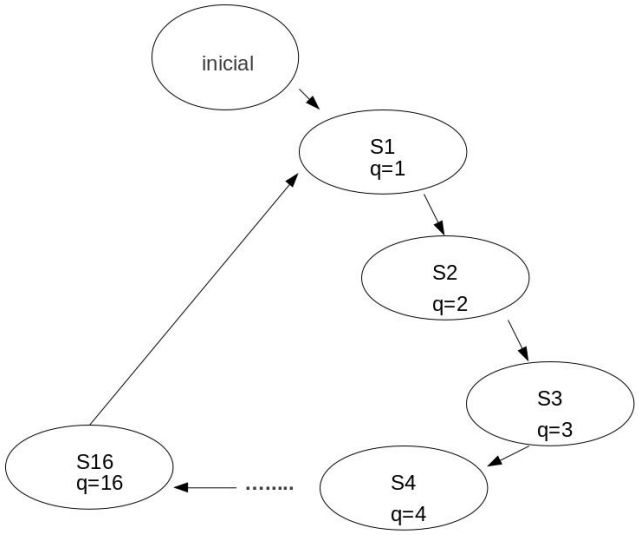


Figura 6.2: diagrama de estados del bloque generador de la secuencia de datos

FUENTE: Los autores.

El esquemático de este bloque es el siguiente:

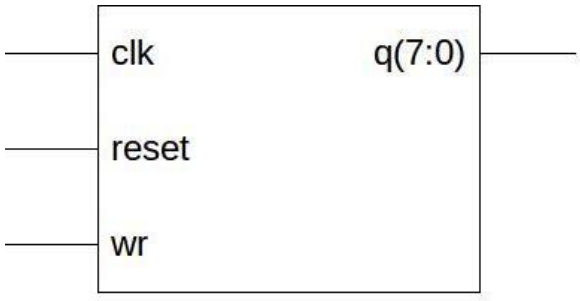


Figura 6.3: esquemático del bloque generador de secuencia

FUENTE: Los autores.

La descripción en VHDL es la siguiente:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity ramdom is
12
13 port(
14                 clk,wr,reset:in std_logic;
15                 q:out std_logic_vector(7 downto 0));
16 end ramdom;
17
18 architecture Behavioral of ramdom is
19 type estado is (inicial,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16);
20 signal presente, futuro:estado;
21 begin
22
23 --registro o memoria de estado
24 process(clk,reset)
25 begin
26   if(reset='0') then
27     presente<=inicial;
28   elsif(clk'event and clk='1') then
29   if (wr='0') then
30     presente<=futuro;
31          end if;
32   end if;
33 end process;
34
35
36 --logica del estado siguiente
37 process(presente)
38 begin
39 case presente is
40 when inicial=>
41      futuro<=s1;
42 when s1=>
43      futuro<=s2;
44 when s2=>
45      futuro<=s3;
46 when s3=>
47      futuro<=s4;
48 when s4=>
49      futuro<=s5;
```

```
50  when s5=>
51        futuro<=s6;
52  when s6=>
53        futuro<=s7;
54  when s7=>
55        futuro<=s8;
56  when s8=>
57        futuro<=s9;
58  when s9=>
59        futuro<=s10;
60  when s10=>
61        futuro<=s11;
62  when s11=>
63        futuro<=s12;
64  when s12=>
65        futuro<=s13;
66  when s13=>
67        futuro<=s14;
68  when s14=>
69   futuro<=s15;
70  when s15=>
71        futuro<=s16;
72  when s16=>
73        futuro<=s1;
74
75  end case;
76  end process;
77
78  --logica de salida
79  process(presente)
80  begin
81    case presente is
82     when s1=>
83          q<="00000001";
84  when s2=>
85          q<="00000010";
86  when s3=>
87          q<="00000011";
88  when s4=>
89          q<="00000100";
90  when s5=>
91          q<="00000101";
92  when s6=>
93          q<="00000110";
94  when s7=>
95          q<="00000111";
96  when s8=>
97          q<="00001000";
98  when s9=>
99          q<="00001001";
```

```
100 when s10=>
101         q<="00001010";
102 when s11=>
103         q<="00001011";
104          when s12=>
105         q<="00001100";
106          when s13=>
107         q<="00001101";
108 when s14=>
109         q<="00001110";
110          when s15=>
111         q<="00001111";
112          when s16=>
113         q<="00010000";
114    when others =>
115         q<="00000000";
116 end case;
117 end process;
118
119 end Behavioral;
```

Script 6.1: Descripción en VHDL bloque generador de la secuencia de datos

II circuito generador de pulso de lectura

La función de este bloque es generar un solo pulso por cada señal de lectura del procesador. El problema radica en que el FPGA posee un reloj más rápido que el procesador y por ende cada ciclo de lectura del procesador significa muchos ciclos de reloj del FPGA, y por consiguiente el bloque de memoria FIFO al trabajar a la frecuencia del FPGA, tomara cada flanco de subida de reloj como una lectura, por lo cual se producirán muchas de éstas en un solo ciclo de lectura de procesador y este comportamiento es indeseable.

El circuito generador de pulso de lectura es en sí una máquina de estados y se describe en la figura 6.4:

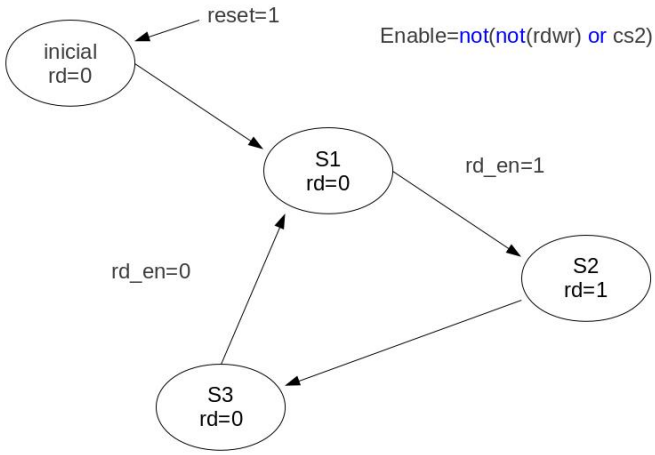


Figura 6.4: máquina de estados del bloque generador de pulso de lectura

FUENTE: Los autores.

Para una mejor comprensión de esta máquina de estados se muestra el diagrama de tiempos asociado en la figura 6.5:

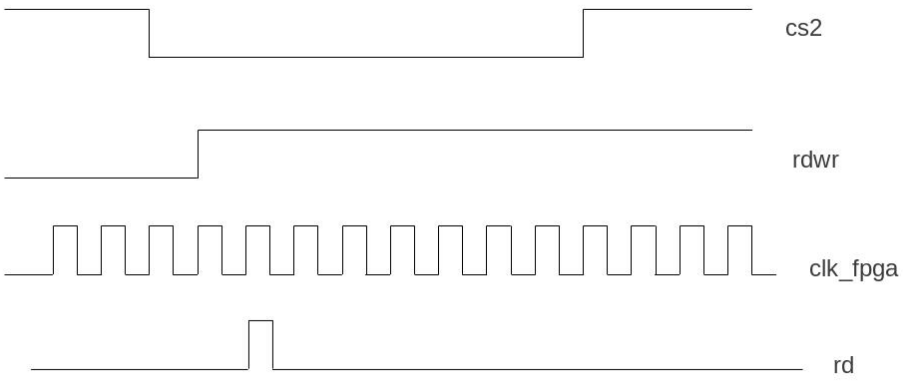


Figura 6.5: diagrama de tiempos del bloque generador de pulso de lectura

FUENTE: Los autores.

Como se puede observar en el diagrama de tiempos el circuito genera un pulso cuando el procesador realiza una operación de lectura evitando así lecturas repetidas y mal funcionamiento del bloque FIFO debido a que una lectura innecesaria provocaría la perdida de datos.

El esquemático del generador de pulso de lectura es el siguiente:

Figura 6.6: esquemático del bloque generador de pulso de lectura

FUENTE: Los autores.

A continuación la descripción en VHDL:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity lectura is
12 port(
13                 rd_en,clk,reset:in std_logic;
14                 rd:out std_logic
15                                                              );
16
17 end lectura;
18
19 architecture Behavioral of lectura is
20 type estado is (inicial,s1,s2,s3);
21 signal presente, futuro:estado;
22 begin
23
24 --registro o memoria de estado
25 process(clk,reset)
26 begin
27   if(reset='0') then
28     presente<=inicial;
29   elsif(clk'event and clk='1') then
30
31     presente<=futuro;
32
33   end if;
34 end process;
```

```
35
36 --logica del estado siguiente
37 process(presente,rd_en)
38 begin
39 case presente is
40 when inicial=>
41      futuro<=s1;
42 when s1=>
43 if (rd_en='1') then
44      futuro<=s2;
45           else
46           futuro<=s1;
47           end if;
48 when s2=>
49      futuro<=s3;
50 when s3=>
51 if (rd_en='0') then
52 futuro<=s1;
53 else
54 futuro<=s3;
55 end if;
56 end case;
57 end process;
58
59 --logica de salida
60 process(presente)
61 begin
62   case presente is
63
64
65 when s2=>
66        rd<='1';
67
68    when others =>
69        rd<='0';
70 end case;
71 end process;
72
73 end Behavioral;
```

Script 6.2: Descripción en VHDL bloque generador de pulso de lectura

III *buffer* triestado

La función de este bloque es asegurar que no se generen choque de datos de entrada y salida, es decir evitar el flujo de datos hacia el procesador cuando se realice una operación de escritura. Este circuito se implementa solo por seguridad y robustez, ya que cuando se implemente esta interfaz de comunicación

no se deberá escribir datos hacia el FPGA desde el procesador.

el esquemático de el *buffer* triestado es el siguiente:

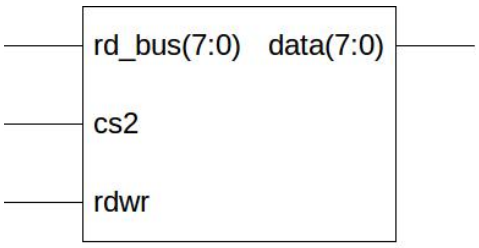


Figura 6.7: esquemático del *buffer* triestado

FUENTE: Los autores.

La implementación en VHDL es la siguiente:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity tri_state is
12     Port (
13
14             rd_bus:in std_logic_vector(7 downto 0);
15             cs2:in std_logic;
16             rdwr:in  STD_LOGIC;
17
18             data : out  STD_LOGIC_VECTOR(7 downto 0));
19 end tri_state;
20 architecture Behavioral of tri_state is
21 signal enable:std_logic;
22 begin
23 enable<=(not(rdwr)) or (cs2);
24 process(rd_bus,enable)
25 begin
26 if(enable='1') then
27 data<="ZZZZZZZZ";
28 else
29 data<=rd_bus;
30 end if;
31 end process;
32 end Behavioral;
```

Script 6.3: Descripción en VHDL *buffer* triestado

IV Memoria FIFO *single port*

El bloque de memoria FIFO es el encargado de la sincronización, ordenamiento y almacenamiento de datos, por lo tanto es la parte más importante de la interfaz.
Este funciona como un registro de datos en forma circular que contiene dos punteros, uno de lectura y otro de escritura destinados para sendos propósitos, ambos indican la posición en la que se va a leer o escribir respectivamente.

Cada vez que se realiza una operación de escritura el puntero de escritura avanza una posición, y cada vez que se realiza una operación de lectura el puntero de lectura avanza una posición, todo esto hasta que se alcanza el tamaño máximo de la memoria, en cuyo caso si no hay ninguna bandera activada (*full/empty*) el puntero se devuelve a la primera posición en la próxima operación.

Si ambos punteros alcanzan la misma posición una de las dos banderas se activa, si se hizo una operación de lectura para igualar la posición del puntero de escritura y lectura entonces la bandera de vacío (*empty*) se activa y no es posible leer más datos hasta que no se vuelva a escribir, por el contrario si se realizó una operación de escritura para igualar la posición del puntero de lectura y escritura la bandera de lleno se activa(*full*) y no es posible escribir más en la memoria hasta que se lea por lo menos un dato. Cada vez que se lee un dato, esa posición queda libre para albergar un nuevo dato.
El comportamiento del bloque de memoria FIFO permite leer los datos que van llegando en el momento o almacenarlos hasta que se esté listo para leer, en el mismo orden de llegada. A continuación se muestra un ejemplo del funcionamiento del bloque de memoria FIFO en la figura 6.8 adaptado del libro[1]. .

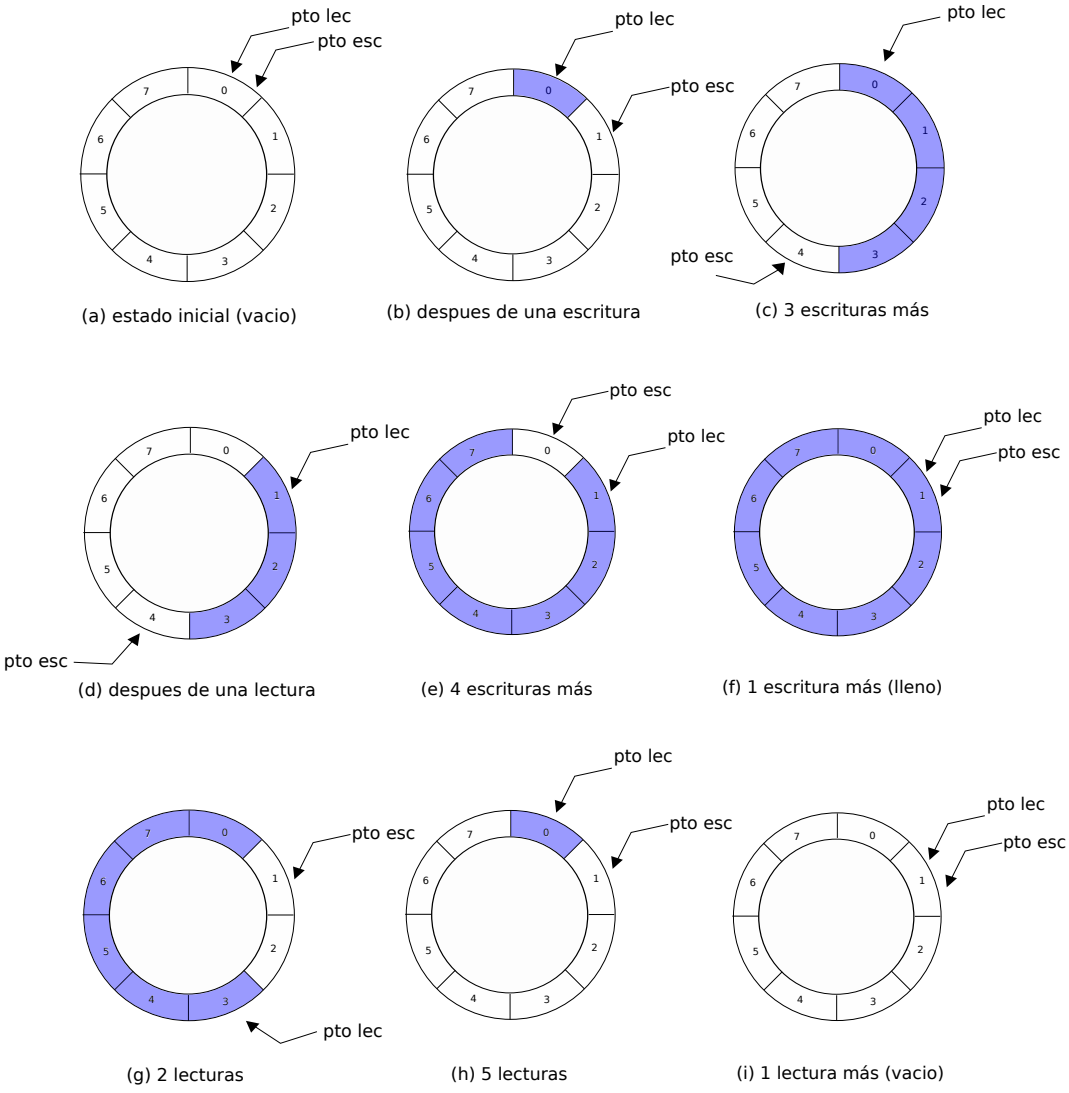


Figura 6.8: Ejemplo de funcionamiento de el bloque de memoria FIFO

FUENTE: Los autores.

El bloque FIFO se compone de:

- Circuito de control FIFO: éste se encarga de acceder a las direcciones de los registros de memoria, actualizar los punteros de lectura y escritura generar banderas de lleno y vacío.
- Registro de memoria: se encarga de guardar los datos provenientes del convertidor analógico-digital para luego ser leídos por el procesador, cabe resaltar que la lectura se puede dar a la vez que la FIFO recibe los datos evitando así esperar que la memoria se llene.

La descripción en VHDL es la siguiente:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity fifodual is
6   generic(
7        B: natural:=8; -- number of bits
8        W: natural:=4 -- number of address bits
9     );
10    port(
11         clk,reset: in std_logic;
12        rd,wr: in std_logic;
13        w_data: in std_logic_vector (B-1 downto 0);
14        empty, full: out std_logic;
15        r_data: out std_logic_vector (B-1 downto 0)
16    );
17
18 end fifodual;
19
20 architecture Behavioral of fifodual is
21  type reg_file_type is array (2**W-1 downto 0) of
22          std_logic_vector(B-1 downto 0);
23    signal array_reg: reg_file_type;
24    signal w_ptr_reg, w_ptr_next, w_ptr_succ:
25       std_logic_vector(W-1 downto 0);
26    signal r_ptr_reg, r_ptr_next, r_ptr_succ:
27       std_logic_vector(W-1 downto 0);
28    signal full_reg, empty_reg, full_next, empty_next:
29           std_logic;
30    signal wr_op: std_logic_vector(1 downto 0);
31    signal wr_en,wr1: std_logic;
32
33 begin
34 wr1<=not wr;
35  --=============================================
36    -- register file
37    --=============================================
```

```
38      process(clk,reset)
39      begin
40        if (reset='0') then
41           array_reg <= (others=>(others=>'0'));
42        elsif (clk'event and clk='1') then
43           if wr_en='1' then
44              array_reg(to_integer(unsigned(w_ptr_reg)))
45                    <= w_data;
46           end if;
47        end if;
48      end process;
49
50      r_data <= array_reg(to_integer(unsigned(r_ptr_reg)));
51      wr_en <= wr1 and (not full_reg);
52
53      --=============================================
54      -- fifo control logic
55      --=============================================
56      -- register for read and write pointers
57      process(clk,reset)
58      begin
59         if (reset='0') then
60            w_ptr_reg <= (others=>'0');
61            r_ptr_reg <= (others=>'0');
62            full_reg <= '0';
63            empty_reg <= '1';
64         elsif (clk'event and clk='1') then
65            w_ptr_reg <= w_ptr_next;
66            r_ptr_reg <= r_ptr_next;
67            full_reg <= full_next;
68            empty_reg <= empty_next;
69         end if;
70      end process;
71
72      -- successive pointer values
73      w_ptr_succ <= std_logic_vector(unsigned(w_ptr_reg)+1);
74      r_ptr_succ <= std_logic_vector(unsigned(r_ptr_reg)+1);
75
76      -- next-state logic for read and write pointers
77      wr_op <= wr1 & rd;
78      process(w_ptr_reg,w_ptr_succ,r_ptr_reg,r_ptr_succ,wr_op,
79              empty_reg,full_reg)
80      begin
81         w_ptr_next <= w_ptr_reg;
82         r_ptr_next <= r_ptr_reg;
83         full_next <= full_reg;
84         empty_next <= empty_reg;
85         case wr_op is
86            when "00" => -- no op
87            when "01" => -- read
```

```
88              if (empty_reg /= '1') then -- not empty
89                  r_ptr_next <= r_ptr_succ;
90                  full_next <= '0';
91                  if (r_ptr_succ=w_ptr_reg) then
92                      empty_next <='1';
93                  end if;
94              end if;
95          when "10" => -- write
96              if (full_reg /= '1') then -- not full
97                  w_ptr_next <= w_ptr_succ;
98                  empty_next <= '0';
99                  if (w_ptr_succ=r_ptr_reg) then
100                     full_next <='1';
101                 end if;
102             end if;
103         when others => -- write/read;
104             w_ptr_next <= w_ptr_succ;
105             r_ptr_next <= r_ptr_succ;
106     end case;
107   end process;
108   -- output
109   full <= full_reg;
110   empty <= empty_reg;
111
112 end Behavioral;
```

Script 6.4: Descripción en VHDL bloque de memoria FIFO

3.1.2 UNIÓN DE LOS BLOQUES PARA COMPLETAR LA INTERFAZ DE COMUNICACIÓN

Por último se debe hacer una conexión entre los bloques para completar la descripción del sistema, en donde se muestra cada uno de ellos, sus señales, respectivas conexiones y funciones. El primer circuito en orden es el generador de secuencia de datos, seguido del bloque generador de pulso de lectura cuya salidas se conectaran al bloque de memoria FIFO para indicarle a éste cuando leer y que datos escribir o guardar. Por último el bloque triestado encargado de pasar los datos leídos del bloque FIFO a el procesador solo cuando éste los requiera.

A continuación se muestra un diagrama que permitirá una concepción más fácil y didáctica del sistema:

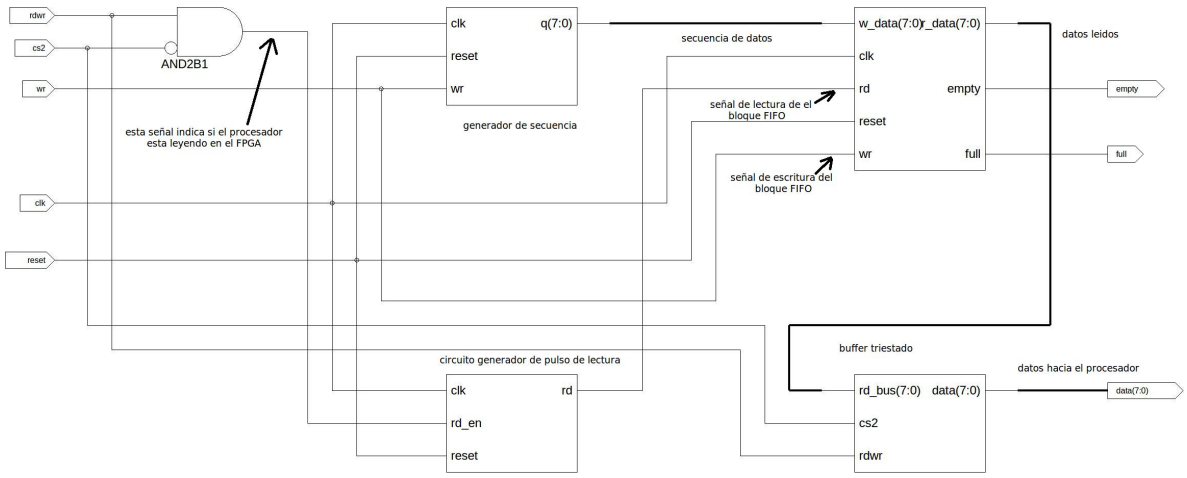


Figura 6.9: diagrama de conexiones del sistema

FUENTE: Los autores.

la descripción en VHDL de el circuito o el bloque que contiene y conecta el sistema es el siguiente:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity todo is
12 generic(
13       B: natural:=8; -- number of bits
14       W: natural:=4 -- number of address bits
15    );
16 port(
17 --clk1:out std_logic
18                 clk,reset,cs2,rdwr,wr:in std_logic;
19       data:out std_logic_vector(7 downto 0);
20            full,empty:out std_logic
21      );
22 end todo;
23 architecture Behavioral of todo is
24 --declaracion de señales
25 signal ren_signal,rd_signal:std_logic;
26 signal rd_bus_signal,w_data_signal:std_logic_vector(7 downto 0);
27 ----------------------------------
28 --componente lectura
29 component lectura
```

```
30  port(
31                  rd_en,clk,reset:in std_logic;
32                  rd:out std_logic
33                                                                  );
34
35  end component;
36
37  ------------------------------------------------
38
39  --componente tri_state
40  component tri_state
41  port(   rd_bus:in std_logic_vector(7 downto 0);
42              cs2:in std_logic;
43              rdwr:in  STD_LOGIC;
44
45              data : out  STD_LOGIC_VECTOR(7 downto 0));
46  end component;
47  ------------------------------------------------
48
49  --componente fifodual
50  component fifodual
51  port(
52          clk,reset: in std_logic;
53         rd,wr: in std_logic;
54         w_data: in std_logic_vector (B-1 downto 0);
55         empty, full: out std_logic;
56         r_data: out std_logic_vector (B-1 downto 0)
57      );
58  end component;
59  ------------------------------------------------
60  --componente ramdom
61  component ramdom
62
63  port(
64                  clk,wr,reset:in std_logic;
65                  q:out std_logic_vector(7 downto 0));
66  end component;
67
68  ------------------------------------------------
69  begin
70
71
72  instancia_tri_state: tri_state
73                  port map (
74              rd_bus=>rd_bus_signal,
75              cs2=>cs2,
76              rdwr=>rdwr,
77
78              data=>data
79                                  );
```

```
80
81  instancia_ramdom:ramdom
82
83  port map(
84                  clk=>clk,
85                  wr=>wr,
86                  reset=>reset,
87                  q=>w_data_signal
88
89                                  );
90
91  instancia_fifo_dual:fifodual
92          port map(
93                                   clk=>clk,
94
95                                   reset=>reset,
96                                   wr=>wr,
97                                   rd=>rd_signal,
98                                   w_data=>w_data_signal,
99                                   r_data=>rd_bus_signal,
100                                  full=>full,
101                                  empty=>empty);
102
103
104 instancia_lectura:lectura
105         port map(
106                                  clk=>clk,
107
108                                  reset=>reset,
109                                  rd=>rd_signal,
110                                  rd_en=>ren_signal
111                                  );
112
113
114
115
116         ren_signal<=not((not(rdwr)) or (cs2));
117 end Behavioral;
```

Script 6.5: Descripción en VHDL de conexión del sistema

Con esto se completa la interfaz de comunicación y se abre paso al siguiente escalón de diseño que compromete la elección adecuada de un convertidor analógico-digital y la implementación de una etapa de adecuación de señal.

3.2 Implementación del convertidor de datos

Para el diseño e implementación de la etapa de conversión de datos es recomendable seguir los siguientes pasos:

- Elección del convertidor analógico-digital.
- Generación del reloj para el convertidor analógico-digital y acoplamiento con el FPGA.

A continuación se describirán de manera más detallada cada uno de los pasos.

3.2.1 ELECCIÓN DEL CONVERTIDOR ANALÓGICO-DIGITAL

Para elegir el convertidor se tienen en cuenta los siguientes aspectos:

- Voltajes de alimentación.
- Consumo de potencia.
- Resolución.
- Frecuencia de muestreo.
- Facilidad de manejo e implementación.
- Compatibilidad con el FPGA.

En base a esto se definieron los siguientes requerimientos de diseño para el proyecto:

	Mínimo	**Máximo**
Voltaje de alimentación	3.3 [v]	5 [v]
Consumo de potencia		300 [mW]
Resolución	8-bits	20-bits
Frecuencia de muestreo	1 [MHz]	100 [MHz]

Tabla 6.1: Requerimientos de diseño

Basado en los requerimientos del proyecto el convertidor seleccionado fue el **THS1230** de **TEXAS INSTRUMENTS** que presenta las siguientes características:

Voltaje de alimentación	3.3 [v]
Consumo de potencia	220 [mW]
Resolución	12-bits
Frecuencia de muestreo	30 [MHz]

Tabla 6.2: Caracteristicas del convertidor

Como se puede apreciar en la tabla este dispositivo cumple con los requerimientos impuestos para el proyecto, además de ser de fácil manejo. Por último se debe hacer un análisis de la compatibilidad del convertidor seleccionado y el FPGA.

3.2.1.1 ANÁLISIS DE LA COMPATIBILIDAD DEL CONVERTIDOR THS1230 CON EL FPGA XC3S100E

Para llevar a cabo el análisis de compatibilidad entre el convertidor y el FPGA se debe tener en cuenta los siguientes aspectos:

- Voltaje máximo de las entradas del convertidor que éste considera como nivel lógico bajo.
- Voltaje mínimo de las entradas del convertidor que éste considera como nivel lógico alto.
- Voltaje máximo de las entradas del FPGA que éste considera como nivel lógico bajo.
- Voltaje mínimo de las entradas del FPGA que éste considera como nivel lógico alto.
- Voltaje máximo del reloj del FPGA en nivel lógico bajo.
- Voltaje mínimo del reloj del FPGA en nivel lógico alto.
- Voltaje máximo de las salidas del convertidor en nivel lógico bajo.
- Voltaje mínimo de las salidas del convertidor en nivel lógico alto.

Con las condiciones de operación recomendadas por la hoja de datos del convertidor se tiene la siguiente tabla:

Fuentes de alimentación	**AVDD=3.3v**	**DVDD=3.3v**	
Entradas digitales (EXTREF,OE,CON0,CON1,CLK)	*Mínimum high level input*=2.64v	*Maximum low level input*=0.66v	Ci=5pf
Salidas digitales (D0-D11,OVRNG)	*Mínimum high level input*=2.9v	*Maximum low level input*=0.4v	
Salidas digitales (D0-D11,OVRNG)	8-bits	20-bits	
Entradas analógicas (Ain+,Ain-,Ci)	Ain+=2.15v	Ain-=1.15v	Ci=6pf

Tabla 6.3: Niveles lógicos del convertidor **THS1230** bajo las condiciones de operación recomendadas

Entradas digitales	*Mínimum high level input*=2v	*Maximum low level input*=0.8v	Cimax=10pf
Salidas digitales(CLKfpga)	*Mínimum high level input*=2.9v	*Maximum low level input*=0.4v	

Tabla 6.4: Niveles lógicos del FPGA XC3S100E

Compatibilidad de la salida de voltaje del FPGA con la entrada de reloj del convertidor

La compatibilidad entre las entradas EXTREF,OE,CON0, CON1 está asegurada debido a que estas serán conectadas a DGND y DVDD cuyos niveles son 0v y 3.3v respectivamente y como se ve en la tabla 6.3 0 volt es menor que lo máximo que reciben las entradas mencionadas anteriormente como nivel lógico bajo (0.66v), y 3.3volts es mayor que lo mínimo que las entradas consideran como nivel lógico alto(2.64).

Ahora lo que se debe asegurar es la compatibilidad de la entrada de reloj del convertidor y el reloj que se va a usar para el convertidor que es un pin de salida del FPGA. Esto se comprueba refiriéndose a la tabla 6.4 en donde se observa que la salida mínima del FPGA en nivel lógico alto es de 2.9 volts y la entrada de reloj del convertidor recibe mínimo como nivel lógico alto 2.64 volts, y la salida máxima del FPGA en nivel lógico bajo es de 0.4volts y la entrada de reloj del convertidor recibe como máximo 0.66 volts con lo cual se comprueba que ambas están dentro del rango.

Compatibilidad de los niveles lógicos de salida del convertidor y los niveles lógicos de entrada del FPGA

Las salidas del convertidor que se van a conectar con las entradas del FPGA son las de datos D0-D11 y la salida que indica over-range. Como se puede observar en las tablas 6.3 y 6.4 el máximo voltaje que el FPGA recibe como nivel lógico bajo es 0.8 volts y el voltaje máximo que entrega el convertidor en nivel lógico bajo es 0.4 volts por lo cual ambas son compatibles.
Ahora en ambas tablas se puede observar que el voltaje mínimo que el FPGA acepta como nivel lógico alto es 2 volts y el voltaje máximo que entrega en convertidor en nivel lógico alto es 2.9 volts, por lo tanto son compatibles.

3.2.2 GENERACIÓN DEL RELOJ PARA EL CONVERTIDOR ANALÓGICO-DIGITAL Y ACOPLAMIENTO CON EL FPGA

El conversor analógico-digital **THS1230** de **TEXAS INSTRUMENTS** necesita un reloj para su funcionamiento que puede estar entre 5 MHz y 30 MHz, preferiblemente de ciclo de reloj 50% con un nivel lógico bajo máximo de 0.66 volts y un nivel lógico alto mínimo de 2.64 volts.
Como es sabido el FPGA cuenta con un reloj de 50 MHz que no es adecuado para el convertidor ya que excede el limite máximo de frecuencia, por esta razon es necesario implementar en VHDL un circuito divisor de frecuencia que la reducirá hasta 12.5 MHz es decir se dividira por 4.
El esquemático del circuito es el siguiente:

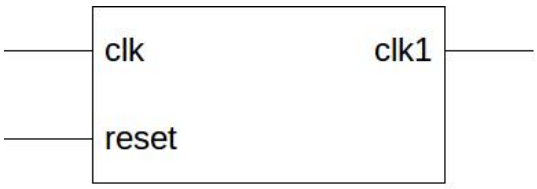


Figura 6.10: esquemático del circuito divisor de frecuencia
FUENTE: Los autores.

El circuito divisor de frecuencia se puede resumir en la siguiente máquina de estados:

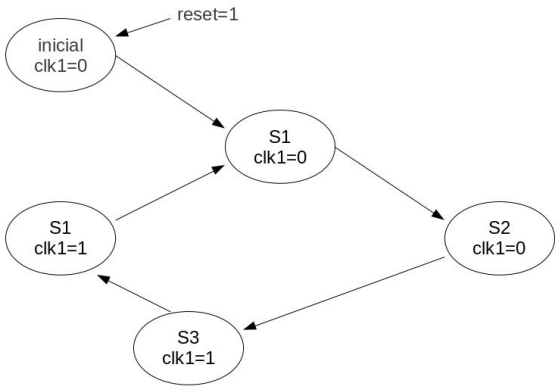


Figura 6.11: máquina de estados del divisor de frecuencia

FUENTE: Los autores.

Para una mejor comprensión de su funcionamiento se proporciona el diagrama de tiempos:

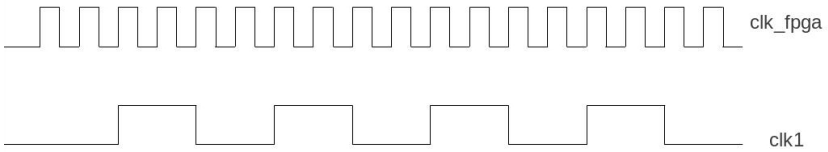


Figura 6.12: diagrama de tiempos circuito divisor de frecuencia

FUENTE: Los autores.

A continuación la implementación en VHDL:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity divisor is
12
13 Port (
14              clk : in  STD_LOGIC;
15            reset:in  STD_LOGIC;
16
17            clk1: out  STD_LOGIC);
18
19 end divisor;
```

```
20
21 architecture Behavioral of divisor is
22 type estado is (inicial,s1,s2,s3,s4);
23 signal presente,futuro:estado;
24 signal q_buffer:std_logic;
25 begin
26
27 --registro o memoria de estado
28 process(clk,reset)
29 begin
30   if(reset='0') then
31     presente<=inicial;
32   elsif(clk'event and clk='1') then
33     presente<=futuro;
34     --buffer_a<=a;
35   end if;
36 end process;
37 --logica del estado siguiente
38 process(presente)
39 begin
40 case presente is
41 when inicial=>
42      futuro<=s1;
43 when s1=>
44              futuro<=s2;
45 when s2=>
46         futuro<=s3;
47 when s3=>
48                    futuro<=s4;
49 when s4=>
50         futuro<=s1;
51 end case;
52 end process;
53 --logica de salida
54 process(presente)
55 begin
56   case presente is
57    when inicial=>
58        q_buffer<='0';
59    when s1=>
60        q_buffer<='0';
61 when s2=>
62       q_buffer<='0';
63    when others =>
64        q_buffer<='1';
65 end case;
66 end process;
67 clk1<=q_buffer;
68 end Behavioral;
```

Script 6.6: Descripción en VHDL de divisor de frecuencia

Se debe acoplar este circuito y las salidas de datos del convertidor analógico-digital a la interfaz de comunicación diseñada inicialmente como se muestra en la siguiente gráfica:

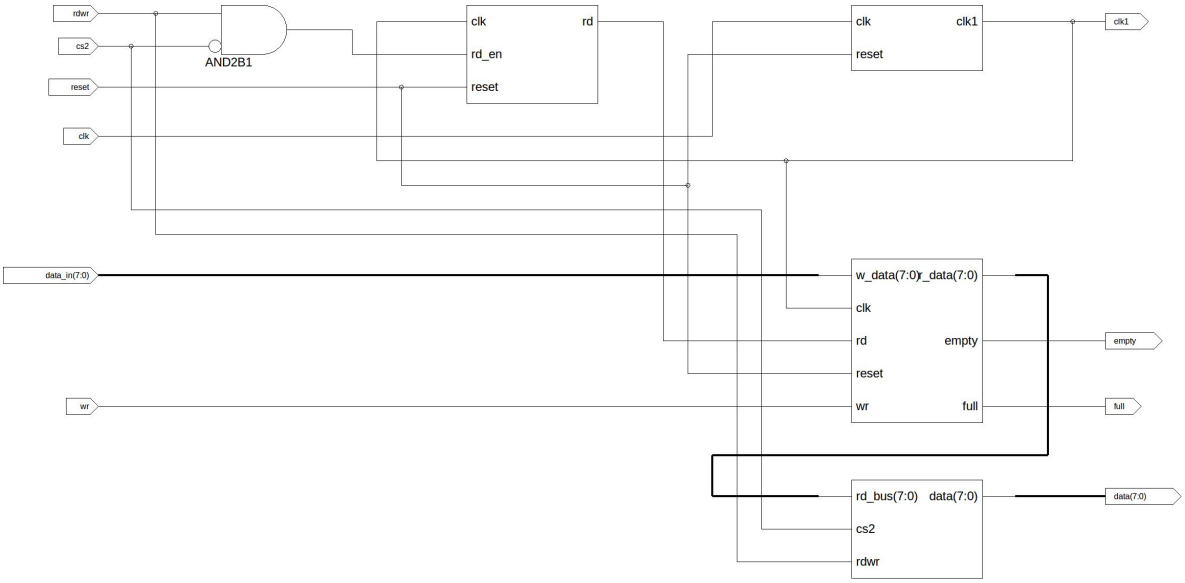


Figura 6.13: Esquemático del acoplamiento del circuito divisor de frecuencia y las salidas de datos del convertidor analógico-digital a la interfaz de comunicación

FUENTE: Los autores.

La implementación en VHDL es la siguiente:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ---- Uncomment the following library declaration if instantiating
7  ---- any Xilinx primitives in this code.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity todo is
12 generic(
13       B: natural:=8; -- number of bits
14       W: natural:=7 -- number of address bits
15    );
16 port(
17 --clk1:out std_logic
```

```
18                  clk,reset,cs2,rdwr,wr:in std_logic;
19                  data_in:in std_logic_vector(7 downto 0);
20        data:out std_logic_vector(7 downto 0);
21              full,empty,clk1:out std_logic
22       );
23 end todo;
24
25
26
27
28 architecture Behavioral of todo is
29 --declaracion de señales
30 signal ren_signal,rd_signal,clk_fifo:std_logic;
31 signal rd_bus_signal:std_logic_vector(7 downto 0);
32 ---------------------------------
33
34
35 --componente lectura
36 component lectura
37 port(
38                  rd_en,clk,reset:in std_logic;
39                  rd:out std_logic
40                                                                    );
41
42 end component;
43
44 ------------------------------------------------
45
46
47
48 --componente tri_state
49 component tri_state
50 port(   rd_bus:in std_logic_vector(7 downto 0);
51              cs2:in std_logic;
52              rdwr:in  STD_LOGIC;
53
54              data : out  STD_LOGIC_VECTOR(7 downto 0));
55 end component;
56 ------------------------------------------------
57
58 --componente fifodual
59 component fifodual
60 port(
61         clk,reset: in std_logic;
62        rd,wr: in std_logic;
63        w_data: in std_logic_vector (B-1 downto 0);
64        empty, full: out std_logic;
65        r_data: out std_logic_vector (B-1 downto 0)
66     );
67 end component;
```

```
68  ----------------------------------------------
69  --componente divisor
70  component divisor
71  Port (
72              clk : in  STD_LOGIC;
73             reset:in  STD_LOGIC;
74
75             clk1: out  STD_LOGIC);
76
77  end component;
78
79  ----------------------------------------------
80  begin
81
82
83  instancia_tri_state: tri_state
84                  port map (
85             rd_bus=>rd_bus_signal,
86             cs2=>cs2,
87             rdwr=>rdwr,
88
89             data=>data
90
91          );
92
93  instancia_divisor:divisor
94
95  port map(
96                  clk=>clk,
97                  reset=>reset,
98                  clk1=>clk_fifo
99
100                                  );
101
102 instancia_fifo_dual:fifodual
103         port map(
104                                  clk=>clk_fifo,
105
106                                  reset=>reset,
107                                  wr=>wr,
108                                  rd=>rd_signal,
109                                  w_data=>data_in,
110                                  r_data=>rd_bus_signal,
111                                  full=>full,
112                                  empty=>empty);
113
114
115 instancia_lectura:lectura
116         port map(
117                                  clk=>clk_fifo,
```

```
118
119                                     reset=>reset,
120                                     rd=>rd_signal,
121                                     rd_en=>ren_signal
122                                     );
123
124
125
126
127         ren_signal<=not((not(rdwr)) or (cs2));
128 clk1<=clk_fifo;
129 end Behavioral;
```

Script 6.7: Descripción en VHDL de divisor de frecuencia

Por último un ejemplo archivo UCF es el siguiente:

```
1  net clk loc=p38; #reloj de 50 MHz
2  net empty loc=p44; #led D5
3  net reset loc=p13;#pulsador PB3
4  net rdwr loc=p86;#señal rdwr
5  net cs2 loc=p69;#señal cs2
6  net wr loc=p30; #pulsador PB2
7  net clk1 loc=p70; #señal de reloj para el ADC
8
9  ##########señales de datos hacia el procesador###########
10 net data(7) loc=p4;
11 net data(6) loc=p5;
12 net data(5) loc=p9;
13 net data(4) loc=p10;
14 net data(3) loc=p11;
15 net data(2) loc=p12;
16 net data(1) loc=p15;
17 net data(0) loc=p16;
18 ##########################################################
19
20
21 ##########señales de datos de salida del ADC hacia el FPGA###########
22 net data_in(7) loc=p23;#dig24
23 net data_in(6) loc=p26;#dig22
24 net data_in(5) loc=p32;#dig20
25 net data_in(4) loc=p34;#dig18
26 net data_in(3) loc=p36;#dig16
27 net data_in(2) loc=p41;#dig14
28 net data_in(1) loc=p48;#dig12
29 net data_in(0) loc=p53;#dig0
30 ##########################################################################
```

Script 6.8: Asignación de pines

Cabe resaltar que para permitir la escritura de datos en la memoria FIFO se debe mantener presionado el pulsador PB2(p30 en el ucf) y que el led D5(led del FPGA) indica cuando la memoria esta vacía.

3.3 ADECUACIÓN DE SEÑAL

En la hoja de datos del convertidor analógico-digital **THS1230** de **TEXAS INSTRUMENTS** se especifica que el rango adecuado de voltaje de la señal de entrada depende de como se configure la generación de voltajes de referencia. El convertidor posee dos voltajes de referencia que son VREFT y VREFB, en este caso se configuró la generación de estos internamente, para más información de la configuración del convertidor **THS1230** referirse al manual de usuario.

La generación interna de los voltajes de referencia VREFT y VREFB implica que estos sean fijos e iguales a 2.15 volts y 1.15 volts respectivamente. Todo voltaje de entrada más grande que VREFT o más pequeño que VREFB esta fuera del rango de entrada del dispositivo y por lo tanto sera ignorado y provocará que la señal OVRNG se active, debido a esta restricción es indispensable el diseño de una etapa de adecuación de señal para ampliar el rango de entrada del convertidor **THS1230** y poder recibir señales de mayor amplitud.

La etapa implementada es la recomendada en la hoja de datos del convertidor y se muestra en la figura 6.14:

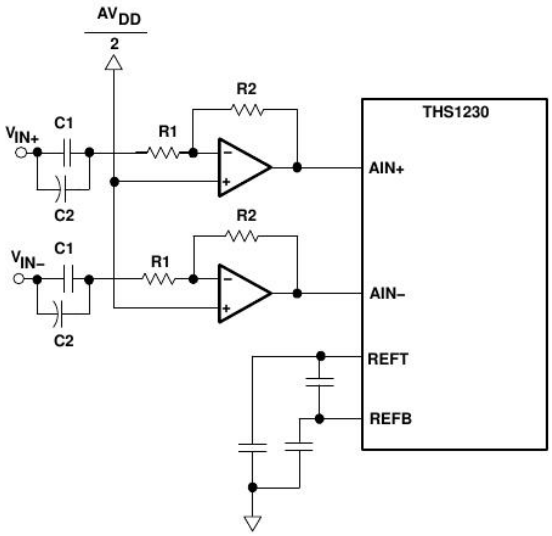


Figura 6.14: Etapa de adecuación de señal
FUENTE: Hoja de datos del convertidor.

En donde R2=10 kΩ, R1=120 kΩ, C1=0.1 uF y C2 no se tiene en cuenta para este diseño. Para más información acerca de esta esta etapa referirse al manual de usuario.

4 TAREAS DE SOFTWARE

Tiene como función leer los datos provenientes de la interfaz de comunicación, no obstante más tareas de *software* pueden ser creadas, como por ejemplo aquellas que permitan hacer algún tratamiento o análisis a la señal, pero esto se deja a consideración del educador y el estudiante para su desarrollo.

4.1 Lectura de datos de llegada desde el FPGA hacia el procesador a través de un programa de usuario en lenguaje C

La lectura de datos se logró haciendo uso de lo aprendido en las guías 1 y 3, es decir el acceso a registros y la comunicación con el FPGA. Este programa de usuario en C consiste de una rutina que habilita la señal de *chip select* para que pueda ser controlada por el EMC, otro código que apunta hacia la dirección del banco de memoria 2 y por último otra rutina que lee los datos permanentemente. A continuación el programa de usuario en C:

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4  #include <asm/ioctl.h>
5  #include <sys/mman.h>
6  int main()
7    {
8      /*DEFINICIONES*/
9      int file_descriptor;
10     int *mapeo;
11     char *escritura;
12     int *lectura;
13     int a,opcion,l;
14     char valor;
15
16
17                         /*CUERPO DEL PROGRAMA*/
18                         file_descriptor=open("/dev/mem",O_RDWR|O_SYNC);
19
20                         if (file_descriptor<0)
21                         {
22                           printf("error al abrir /dev/mem \n");
23                         }
24
25                            /*fin de la rutina de verificación*/
26
```

```
27          mapeo=mmap(NULL,0x8000000 ,PROT_READ|PROT_WRITE,MAP_SHARED, file_descriptor ,0x10000000);
28
29
30                      //verificacion de la señal CS2
31                              lectura=mapeo+0x4050;
32                              a=*lectura & 0x4000000;
33
34
35                          if(a==0x0){
36                          printf("la señal CS2 esta deshabilitada ¿desea habilitarla? \n");
37
38
39                          printf("1.si                \n");
40                          printf("2.no               \n");
41
42
43                          scanf("%d",&opcion);
44                          if(opcion==1){
45                          lectura=mapeo+0x4051;
46                            *lectura=0x4000000;
47                            printf("CS2 habilitada \n");
48                                      }
49
50                          if(opcion==2){
51                           printf("CS2 no habilitada \n");
52                                       }
53
54                                       }
55                          //fin de la verificacion
56
57          escritura=(char *)(mapeo+0x1000004);
58  //scanf("%d",&valor);
59
60  while(2>1){
61
62                            printf("el valor de  el registro es %d \n ",*escritura);
63
64                                    }
65
66        munmap(mapeo,0x8000000);
67        close(file_descriptor);
68      }
```

Script 6.9: Codigo completo

Para finalizar, es recomendable hacer uso de este proyecto como una guía inicial, es decir el estudiante deberá analizarlo y comprenderlo, para así tener una idea clara a la hora de afrontar un proyecto más grande y amplio.

Bibliografía

[1] Pong P. CHU, *FPGA PROTOTYPING BY VHDL EXAMPLES*, WILEY, United state of america, 3th edition, 2008

ANEXO B. MANUAL DE USUARIO DE LA TARJETA SIE LABORATORIO ARQUITECTURA DE COMPUTADORES

CAPÍTULO 1

GUIA DE INSTALACION DEL COMPILADOR CRUZADO PARA LA PLATAFORMA SIE

La tarjeta SIE posee un procesador *xburst* de la empresa *ingenic* en este caso un *Ingenic* JZ4725, debido a esto se debe usar una herramienta para que SIE entienda los programas hechos por el usuario mediante el PC. Esta herramienta es un compilador cruzado, el cual permite hacer ejecutables a partir de archivos .c .h para otras arquitecturas diferentes en donde se estan creando. El compilador cruzado a utilizar hace parte del sistema operativo *openwrt*, en especial se escogió una rama de éste llamada *backfire*, en esta rama se puede encontrar los elementos necesarios para producir los binarios del *software* que posteriormente irán a la tarjeta.

La guía se dividirá en los siguientes pasos:

1. Preparación.
2. Configuración.
3. Instalación.
4. Compilación de un programa ejemplo.
5. Pasar los archivos a la tarjeta.
6. Ejecutar el programa.

1.1 Preparación

Antes de empezar a instalar es necesario que el sistema operativo disponga de ciertos paquetes.Se procede a descargarlos colocando en la consola lo siguiente:

```
1 sudo aptitude install sed wget cvs subversion git-core coreutils unzip texi2html texinfo
2 libsdl1.2-dev docbook-utils gawk python-pysqlite2 diffstat help2man make gcc
3 build-essential g++ desktop-file-utils chrpath flex libncurses5 libncurses5-dev
```

```
4 libxml-simple-perl zlib1g-dev pkg-config gettext libxml-simple-perl guile-1.8 cmake
```

Script 1.1: Descarga de paquetes necesarios

El paso anterior descarga e instala automáticamente los paquetes necesarios. teniendo los paquetes instalados se procede a descargar el repositorio:

```
1  git clone git://projects.qi-hardware.com/openwrt-xburst.git
```

Script 1.2: Descarga de repositorio

Cuando finalice el proceso se entra a la carpeta que se descargo llamada *openwrt-xburst* así:

```
1  cd openwrt-xburst
```

Script 1.3: Entrar a la carpeta

```
1  git fetch origin
```

Script 1.4: Cambiar a repositorio descargado

Después de esto se procede a actualizar el repositorio descargado:

```
1   git checkout --track -b local_backfire origin/tracking_backfire
```

Script 1.5: Actualizar repositorio descargado

El paso anterior es posible que falle, una posible solución es entrar a la siguiente pagina `http://projects.qi-hardware.com/index.php/p/openwrt-xburst/source/tree/backfire/` a través de su navegador favorito y descargar el *branch* que está disponible haciendo *click* en *Download this version*.

Luego de esto se descomprime y se copian todos los elementos de la carpeta que se descargó y se pegan dentro de la carpeta *openwrt-xburst* pero teniendo cuidado de no reemplazar los archivos existentes solo darle en *skip* para agregar los archivos que no están en nuestra carpeta, ya con esto queda nuestro *openwrt* actualizado.

1.2 CONFIGURACIÓN

Ahora se configura la instalación, es decir elegir lo que se va a instalar así:

```
1   cp data/qi_lb60/conf/config.minimal .config
```

Script 1.6: Como elegir configuración mínima

Este comando realiza la configuración mínima, para poder compilar se coloca en consola lo siguiente :

```
1 make menuconfig
```

Script 1.7: configuración del menú

Se elige en el submenu *Languages* [1] lo que indica la figura.

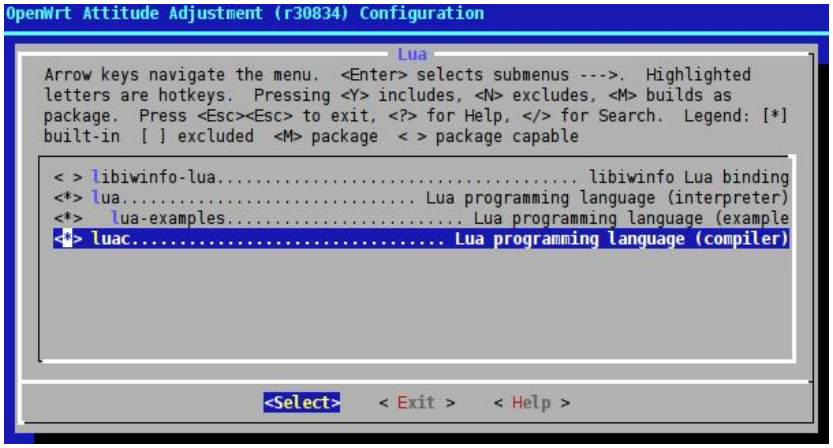


Figura 1.1: Configuración del *make*

FUENTE: Los autores.

Ahora en *Advanced configuration options (for developers)* se elige lo siguente.

[1] Con Y se selecciona un submenu y con ENTER se entra al submenu

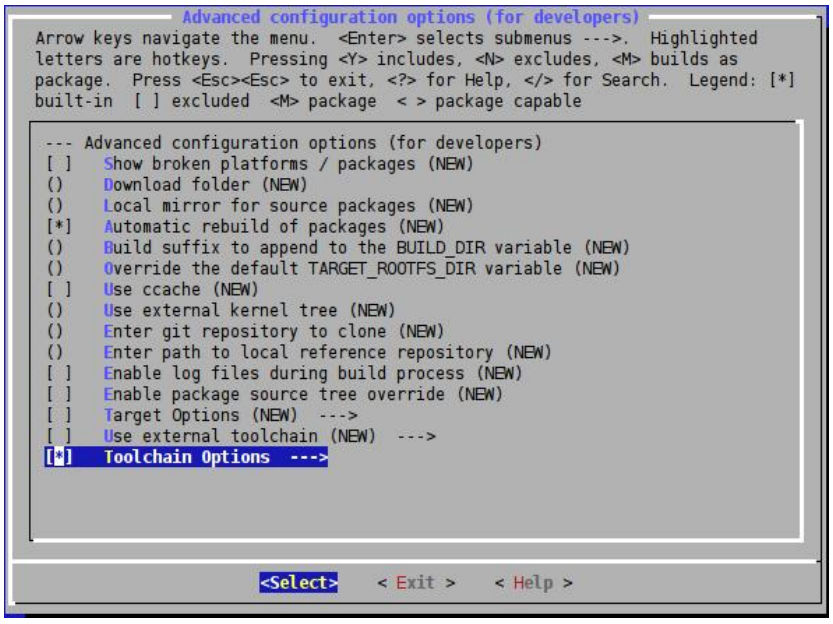


Figura 1.2: Configuración del *make*

FUENTE: Los autores.

1.3 INSTALACIÓN

Ahora solo hace falta compilar.

```
1 make -j3
```

Script 1.8: ejecutar *make*

Cuando el proceso se complete, normalmente esto toma alrededor de una hora u hora y media pero dependiendo del computador puede llegar a tomar hasta dos horas o mas. Habiendo terminado de instalar debe aparecer los siguientes archivos en la carpeta PATH_TO_OPENWRT/openwrt-xburst/bin/xburst:

- Carpeta packages.
- md5sums.
- openwrt-xburst-qi_lb60-root.ubi.
- openwrt-xburst-qi_lb60-rootfs.tar.gz.
- openwrt-xburst-qi_lb60-u-boot.bin.

- openwrt-xburst-qi_lb60-uImage.bin.

Con esto se finaliza la configuración e instalación del compilador cruzado.

1.4 Compilación de un programa ejemplo

Para hacer la compilación y generar los .elf y los .bin del programa ejemplo se busca el programa que viene dentro del *xburst-tool* que se puede descargar de `http://projects.qi-hardware.com/index.php/p/xburst-tools/source/tree/master/` o simplemente colocando en consola:

```
1 cd
2 git clone git://projects.qi-hardware.com/xburst-tools.git
```

Script 1.9: Descarga del programa ejemplo

No es necesario instalar esta herramienta para compilar, solo se hará uso del programa *hello world* que viene dentro de esta y se encuentra en /PATH_TO_xburst-tools/xburst-tools/usbboot/xburst_stage1/. Ahora se ingresa a la carpeta y se observan los siguientes archivos:

- board-jz4740.c.
- board-jz4760.c.
- board-jz4760.h.
- common.c.
- debug.c.
- head.s.
- main.c.
- Makefile.
- target.ld

Estos archivos son necesarios para la compilación y contienen la información del programa, desde que recursos se van a utilizar hasta como y cuando se utilizan. Ahora se procede a editar el archivo *Makefile* que es el que contiene el orden y reglas de compilación y creación de objetos:

```
1 gedit Makefile
```

Script 1.10: Modificación del *Makefile*

```
#
# Authors: Xiangfu Liu <xiangfu@sharism.cc>
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version
# 3 of the License, or (at your option) any later version.
#
#######################################################
###########Agregar la siguiente linea###############
#######################################################

CROSS_COMPILE=mipsel-openwrt-linux-

#######################################################


INFLASH_SRC_PATH = ../src
XBURST_INCLUDE_PATH = ../xburst_include

ifeq ($(CROSS_COMPILE),)
$(warning CROSS_COMPILE variable not set, should point to .../mipsel-openwrt-linux-)
endif

CFLAGS  := -O2 -fno-unit-at-a-time -fno-zero-initialized-in-bss -mips32 -fno-pic \
           -mno-abicalls -I$(INFLASH_SRC_PATH) -I$(XBURST_INCLUDE_PATH)
LDFLAGS := -nostdlib -EL -T target.ld

OBJS    = head.o main.o common.o board-jz4740.o board-jz4760.o debug.o

all: xburst_stage1.bin

xburst_stage1.bin: xburst_stage1.elf
        $(CROSS_COMPILE)objcopy -O binary $< $@+
        $(CROSS_COMPILE)objdump -D $< > xburst_stage1.dump
        $(CROSS_COMPILE)objdump -h $< > xburst_stage1.map
        $(CROSS_COMPILE)nm -n $< > System.map
        chmod -x $@+
        mv -f $@+ $@

xburst_stage1.elf: $(OBJS)
        $(CROSS_COMPILE)ld $(LDFLAGS) $(OBJS) -o $@
.c.o:
        $(CROSS_COMPILE)gcc $(CFLAGS) -c $< -o $@
.S.o:
        $(CROSS_COMPILE)gcc $(CFLAGS) -c $< -o $@
clean:
        rm -f xburst_stage1.bin xburst_stage1.elf xburst_stage1.dump xburst_stage1.map
        rm -f $(OBJS)
```

```
50          rm -f System.map
```

Script 1.11: *Makefile*

Despues de esto se le indica al sistema operativo donde se encuentran los comando propios al compilador cruzado para que el sea capaz de ejecutarlos. Dentro de la carpeta de usuario se ejecuta:

```
1  gedit .bashrc
```

Script 1.12: modificación *.bashrc*

En la última linea de este archivo se va agregar la siguiente linea, hay que tener precaución por que no siempre es la misma, esto depende de donde se guardaron los binarios, se recomienda primero buscar donde se encuentran y después agregar la linea. Un ejemplo de como puede ser la ruta es la siguiente.

```
1 export PATH=$PATH:/home/user/openwrt-xburst/staging_dir/toolchain-mipsel_gcc-4.3.3+cs_uClibc
2 -0.9.30.1/bin/
```

Script 1.13: modificación *.bashrc*

Después de esto se debe cerrar la consola actual y trabajar sobre una nueva consola para que se ejecute el .bashrc con la nueva ruta. Por último se accede a la ubicación del programa ejemplo y se ejecuta el *make*:

```
1 cd /PATH_TO_xburst-tools/xburst-tools/usbboot/xburst_stage1/
2 make
```

Script 1.14: ubicación carpeta ejemplo

Al hacer *make* se generan los archivos .elf y .bin entre otros que son los que se van a pasar a la tarjeta.

1.5 Pasar los archivos a la tarjeta

Habiendo generado los archivos con el compilador cruzado se procede a pasarlos al procesador para posteriormente ser ejecutados. Para esto se ingresa a la carpeta donde se encuentra el archivo a pasar, en este caso es *xburst_stage1*, antes de realizar esto se debe establecer la comunicación con la tarjeta y luego se copia el archivo y se entra a la tarjeta:

```
1  dmesg
2  sudo ifconfig usb0 192.168.1.2 up
3  scp xburst_stage1.bin root@192.168.1.1:~
4  ssh root@192.168.1.1
```

Script 1.15: pasos para configurar la tarjeta

1.6 Ejecutar el programa

Ya despues de haber ingresado en el procesador se ejecuta el programa con el siguiente comando.

```
1  ./archivo_que_queremos_ejecutar
```

Script 1.16: ejecutar programa en la SIE

Y con esto ya queda comprobado el compilador cruzado para la tarjeta SIE.

CAPÍTULO 2

LLAMADOS AL SISTEMA

2.1 Llamada al sistema mmap

El *kernel* de Linux se compone de módulos que agregan funcionalidad a éste, estos módulos son códigos con una estructura definida que pueden interactuar entre si y cumplen ciertas tareas, en particular los módulos encargados de administrar y controlar periféricos son llamados *drivers*.

Los *drivers* abstraen o representan el periférico que controlan en el sistema de archivos mediante un fichero, uno de estos ficheros es */dev/mem* que es la abstracción de los registros de el procesador que soporta el sistema operativo, es decir dicho *driver* puede controlar estos registros.

Los llamados al sistema son funciones que atienden tareas y procesos solicitados por el usuario, dandoles prioridad. El usuario del sistema operativo solo se puede comunicar con el *kernel* a través de los llamados al sistema,los cuales permiten el uso de los módulos pertenecientes a éste con el fin de aprovechar su funcionalidad. En particular el usuario puede hacer uso de los llamados al sistema para disponer de los *drivers* y así controlar periféricos determinados asociados a éste, la llamada al sistema *mmap* es una de ellas y es capaz de proyectar el contenido de el fichero */dev/mem* en memoria para que se pueda tener acceso a el.

```
1 #include<sys/mman.h>
2 void * mmap(void *addr ,
3        size_t len ,
4        int prot ,
5        int flags ,
6        int fd ,
7        off_t offset );
```

Script 2.1: llamada al sistema *mmap*

El llamado al sistema *mmap*() le solicita al *kernel* hacer un mapeo en la memoria de *len* bytes del objeto representado por el descriptor de archivo *fd*, comenzando a *offset* bytes dentro del archivo. Si se especifica *addr*, se utiliza este valor como la dirección inicial en la memoria. Las opciones de la llamada al sistema *mmap* se resumen en la siguiente tabla.

<table>
<tr><th colspan="3">OPCIONES MMAP</th></tr>
<tr><td>*addr</td><td colspan="2">Dirección en la que se va a alojar la proyección de memoria(el mapeo).Si se coloca NULL el sistema es el que elige.</td></tr>
<tr><td>len</td><td colspan="2">Cantidad del fichero a mapear, es decir la porción del fichero que se quiere mapear, esta tiene que ser múltiplo del tamaño de página de memoria del sistema operativo.</td></tr>
<tr><td rowspan="6">prot</td><td colspan="2">protección de memoria deseada</td></tr>
<tr><td>Opciones comando</td><td>Descripción</td></tr>
<tr><td>PROT_EXEC</td><td>Se permite ejecución.</td></tr>
<tr><td>PROT_READ</td><td>se permite leer.</td></tr>
<tr><td>PROT_WRITE</td><td>se permite escribir.</td></tr>
<tr><td>PROT_NONE</td><td>las páginas no pueden ser accedidas.</td></tr>
<tr><td rowspan="4">flags</td><td colspan="2">opciones de acceso.</td></tr>
<tr><td>Opciones comando</td><td>Descripción.</td></tr>
<tr><td>MAP_FIXED</td><td>No seleccionar una dirección diferente a la especificada. Si la dirección especificada no puede ser utilizada, mmap fallará. Utilizar esta opción no es aconsejable.</td></tr>
<tr><td>MAP_SHARED</td><td>varios procesos tienen acceso a el mapa que se quiere hacer.</td></tr>
<tr><td>MAP_PRIVATE</td><td>el mapa realizado es privado y solo un proceso tiene acceso a el.</td></tr>
<tr><td>fd</td><td colspan="2">descriptor del fichero al que se le va a hacer el mapeo,este debe abrirse previamente. La parte que describe cómo se abre el fichero de esta guía FD se llama file_descriptor.</td></tr>
<tr><td>offset</td><td colspan="2">el offset es la dirección en la cual se quiere que comience el archivo, este al igual que el LEN tiene que ser múltiplo del tamaño de la página.</td></tr>
</table>

Tabla 2.1: Opciones **MMAP**.

2.2 Llamada al sistema open

El llamado al sistema *open* tiene como función abrir un fichero existente y devolver un descriptor de fichero el cual debe guardarse en una variable. La sintaxis del llamado al sistema ***open*** es la siguiente:

```
1 int open(const char *pathname, int flags)
```

Script 2.2: llamada al sistema *open*

donde **pathname* es la ruta donde se ubica el fichero que se quiere abrir, El parámetro *flags* establece la forma en que se va a trabajar con el fichero. Dentro de estos se tiene:
Para poder usar los modos y la función *open* se debe incluir la cabecera $< fcntl.h >$ los modos pueden combinarse, haciendo un "or" lógico, como por ejemplo :

O_RDONLY | O_WRONLY

flags	DESCRIPCIÓN
O_RDONLY	Abre el archivo y permite leerlo.
O_WRONLY	Abre el archivo y permite escribir en el.
O_SYNC	Opción para trabajar con dispositivos I/O síncronos.
O_RDWR	Abre el fichero habilitando lectura y escritura.
O_CREAT	Primero crea el fichero y luego lo crea.

Tabla 2.2: Opciones parametro *flags*.

El parámetro **acceso** solo se usa cuando se incluye la opción **O_CREAT**,La función ***open*** retorna un descriptor de fichero válido si el fichero se ha podido abrir, y el valor -1 en caso de error .

Bibliografía

[1] The Open Group Base Specifications Issue 6, *IEEE Std 1003.1, 2004 Edition*. `http://pubs.opengroup.org/onlinepubs/009695399/functions/mmap.html`.

[2] die.net, *man page for reference to call systems*. `http://linux.die.net/man/2/syscalls`.

CAPÍTULO 3

MÓDULO PWM

La tarjeta SIE tiene un procesador con arquitectura MIPS de ingenic semiconductor el SoC JZ4725, que contiene una serie de módulos importantes como también salidas de propósito general (GPIO) que al configurarlas correctamente por medio de registros de lectura y escritura pueden ofrecer muchas ventajas en variedad de aplicaciones.
El PWM que contiene el JZ4725 es de 8 canales configurado a través de un *timer* que contiene las siguientes características.

- Canales independientes, cada uno contiene.
  - contador.
  - registro de datos.
  - registro de control.
- Reloj independiente para cada contador seleccionado por *software*.
  - PCLK,EXTAL Y EL RTCCLK que puede usado como el reloj del contador.
  - La relación de división puede ser seleccionada en 1,4,16,64,256 y 1024 por *software*.
- Usando un registro de datos para la comparación donde el *timer* 0 al 7 puede ser usado como PWM seleccionando un nivel con una señal inicial.

3.1 Unidad generación de reloj

La unidad de generación de reloj (CGU *clock generation unit*) contiene un PLL que es manejado por un oscilador externo y un circuito generador de reloj del que se derivan los siguientes relojes.

señal	**descripción**
CCLK	Reloj rápido para operaciones internas tales como ejecución de instrucciones de la cache.
HCLK	Reloj del sistema, aparece como entrada a la CPU. Este reloj siempre esta habilitado(cuando el sistema no esta en modo *sleep*)
PCLK	Reloj de periféricos,reloj del dispositivo *APB BUS*
MCLK	Reloj del controlador EMC
CKO	Reloj SDRAM
LDCLK	Reloj para dispositivo LCD
LPCLK	Reloj para el pixel del LCD
CIM_MCLK	Reloj de salida para el modulo CIM
CIM_PCLK	Reloj de entrada para el modulo CIM
I2SCLK	Reloj para el codec I2S
MSCCLK	Reloj MSC
SSICLK	Reloj SSI
EXCLK	Reloj de salida de 12M for UART I2C SSI TCU USB2.0-PHY

Tabla 3.1: UNIDAD DE GENERACIÓN DE RELOJ

3.2 Diagrama de bloques CGU

En la figura 3.1 se muestra el diagrama de bloques para CGU.

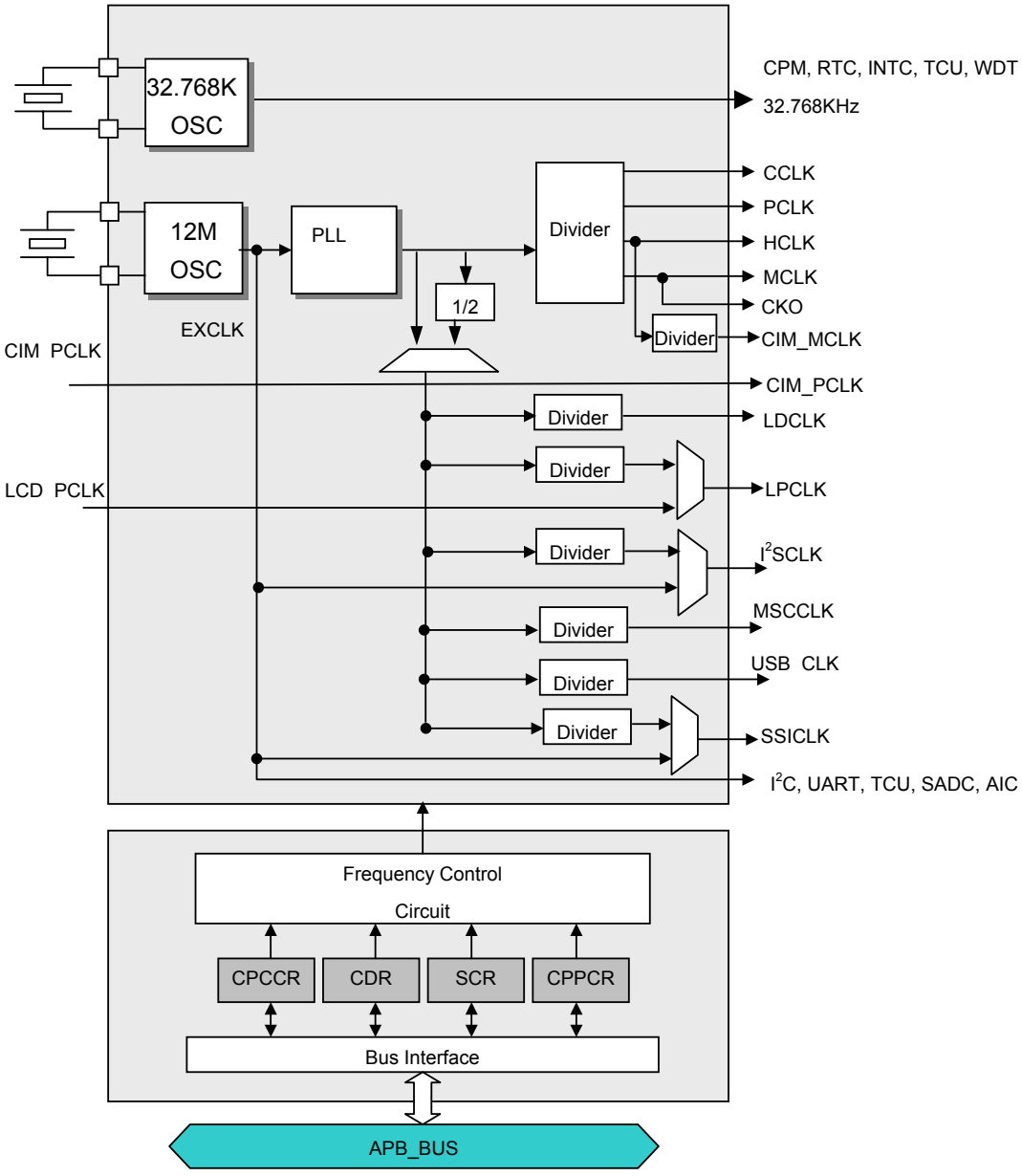


Figura 3.1: Diagrama de bloques CGU

FUENTE: Los autores.

3.3 Registros CGU

Todos los registros del CGU son de 32 bits las direcciones de acceso son direcciones físicas.

Nombre	Descripción	RW	Valor reset	Dirección	tam
CPCCR	Registro de control del reloj	RW	0x42040000	0x10000000	32
CPPCR	Registro de control PLL	RW	0x28080011	0x10000010	32
I2SCDR	Registro divisor de reloj I2S	RW	0x00000004	0x10000060	32
LPCDR	Registro divisor de pixel LCD	RW	0x00000004	0x10000064	32
MSCCDR	Registro divisor de reloj MSC	RW	0x00000004	0x10000068	32
UHCCDR	Registro divisor de reloj UHC 48M	RW	0x00000004	0x1000006C	32
SSICDR	Registro divisor de reloj SSI	RW	0x00000004	0x10000074	32

Tabla 3.2: Configuración de registros CGU

3.3.1 Registro CPCCR

El registro de control del reloj (CPCCR) es un registro de 32 bits de escritura/lectura que controla el divisor de CCLK, HCLK, PCLK,MCLK y LDCLK. Se inicializa a 0x42040000 por cualquier reset. Solo palabras de acceso pueden ser utilizadas en CPCCR.

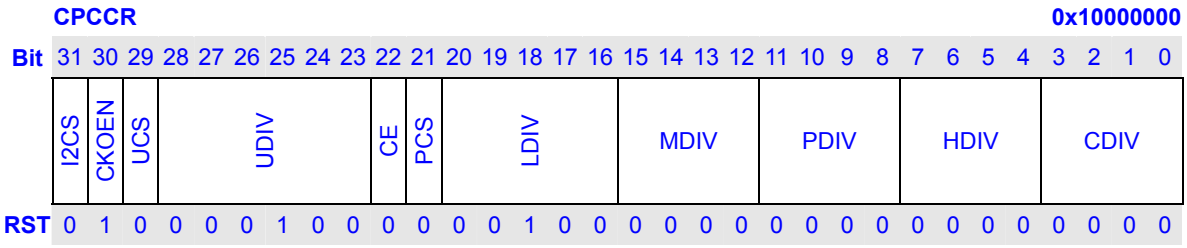


Figura 3.2: Registro CPCCR

FUENTE: [1].

bits	Nombre	Descripción	RW
31	I2CS	Selecciona la fuente de reloj de I2S entre el PLL y el pin EXCLK 0: la fuente de I2S es EXCLK. 1: la fuente de I2S es la salida del PLL dividida por I2SDIV	32
30	I2CS	Controla la salida de CKO permitiendo su habilitación 0: Deshabilita la salida de CKO, CKO es Hi-z(alta impedancia) 1: Habilita la salida de CKO	32
29	I2CS	Selecciona la fuente de reloj de bus USB entre la salida del PLL y el pin EXCLK 0: La fuente de reloj del USB es el pin EXCLK 1: La fuente de reloj del USB es la salida del PLL	32
28:23	UDIV	Divisor de frecuencia para el USB. Cuando la fuente de reloj del USB es el PLL (el bit UCS es 1), este campo especifica la razón de división la cual varia de 1 a 64 (razón de división=UDIV+1)	32
22	CE	Cambios habilitados. Cuando este campo está en 1 la secuencia de cambios de frecuencia provocados por la escritura en los campos CDIV,HDIV,PDIV,MDIV,UDIV,PXDIV o LDIV empieza inmediatamente. Cuando está en 0, la escritura en los campos de registro CDIV,HDIV,PDIV,MDIV,UDIV,PXDIV o LDIV no empieza inmediatamente, la razón o proporción de división es actualizada en la secuencia de proporción de cambios múltiple del PLL o se deshabilita la secuencia.	32
21	PCS	Seleccion de la fuente de reloj del PLL. Éste provee la fuente de reloj para el MSC, I2S, LCD,USB 0: Fuente de reloj para ser dividida es la salida del PLL dividida por 2 1: Fuente de reloj para ser dividida es la salida del PLL	32
20:16	LDIV	Divisor para la frecuencia de reloj para el dispositivo LCD. Especifica la razón de división, la cual varia de 1 a 32 (razón de división=LDIV+1). La frecuencia de LCLK debe ser igual o menor de 150 MHz.	32
15:12	MDIV	Divisor para la frecuencia de reloj de la memoria. Especifica la razón de división de MCLK 0000= X1 0001= X1/2	32

		0010= X1/3 0011= X1/4 0100= X1/6 0101= X1/8 0110= X1/12 0111= X1/16 1000= X1/24 1001= X1/32 Otro valor= Reserved	
11:8	PDIV	Divisor para la frecuencia de reloj de los periféricos.especifica la razón de división de el PCLK. 0000= X1 0001= X1/2 0010= X1/3 0011= X1/4 0100= X1/6 0101= X1/8 0110= X1/12 0111= X1/16 1000= X1/24 1001= X1/32 Otro valor= Reserved	32
7:4	HDIV	Divisor para la frecuencia de reloj del sistema. Especifica la razón de división de el HCLK. 0000= X1 0001= X1/2 0010= X1/3 0011= X1/4 0100= X1/6 0101= X1/8 0110= X1/12 0111= X1/16 1000= X1/24 1001= X1/32 Otro valor= Reserved	32
3:0	CDIV	Divisor para la frecuencia de reloj de la CPU. Especifica la razón de división de CCLK. 0000= X1 0001= X1/2 0010= X1/3 0011= X1/4 0100= X1/6 0101= X1/8 0110= X1/12 0111= X1/16 1000= X1/24 1001= X1/32 Otro valor= Reserved	32

Tabla 3.3: Configuración registro CPCCR

3.3.2 Registro I2SCDR

Es un registro de 32 bits de escritura/lectura que especifica el divisor de reloj de del dispositivo I2S. Este registro es inicializado a 0x00000004 por cualquier reset. Solo palabras de acceso pueden usarse en I2SCDR.

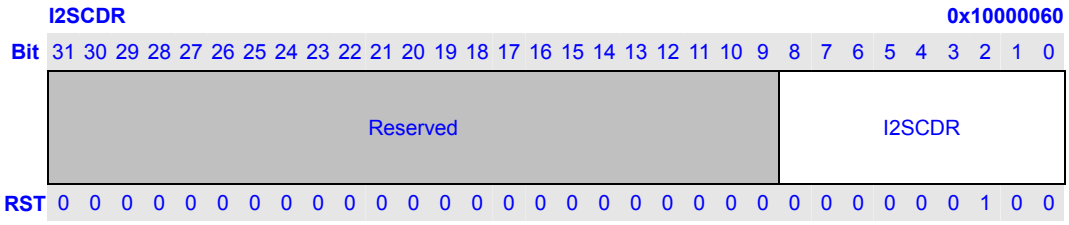


Figura 3.3: Registro I2SCDR

FUENTE: [1].

bits	Nombre	Descripción	RW
31:9	Reservado	Escribir en estos bits no tiene efecto y siempre se lee 0 de ellos.	R
8:0	I2SCDR	Divisor de frecuencia para el I2S. Especifica la razón de división de reloj del dispositivo I2C la cual varia de 1 a 512(razón de división=I2CDR+1)	RW

Tabla 3.4: Configuración registro I2SCDR

3.3.3 Registro LPCDR

Es un registro de 32 bits de escritura/lectura que especifica el divisor de reloj de pixel(LPCLK). Este registro es inicializado a 0x00000004 por cualquier reset. Solo palabras de acceso pueden usarse en LPCDR.

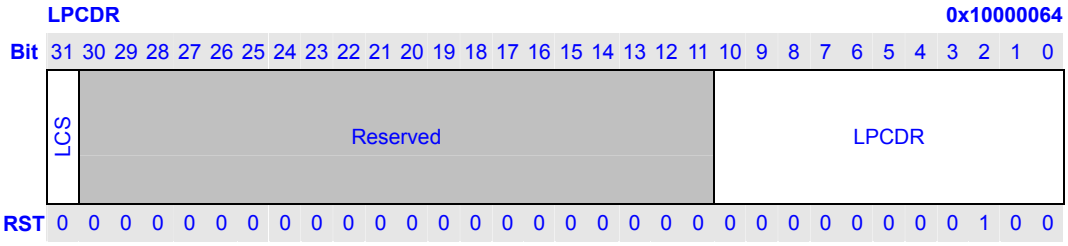


Figura 3.4: Registro LPCDR

FUENTE: [1].

bits	Nombre	Descripción	RW
31	LCS	Selección de la fuente de reloj del pixel LCD selecciona la fuente de reloj del pixel LCD entre el divisor y una entrada externa de reloj 0: La fuente es la salida del divisor. 1: La fuente es el pin LCD_PCLK.	RW
30:11	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0.	R
10:0	LPCDR	Divisor de frecuencia para los pixeles. Especifica la razón de división del reloj de los pixeles del LCD(LPCLK) la cual varia de 1 a 2048(razón de división=LPCDR+1)	RW

Tabla 3.5: Configuración registro LPCDR

3.3.4 Registro MSCCDR

Es un registro de 32 bits de escritura/lectura que especifica el divisor de reloj del dispositivo MSC. Este registro es inicializado a 0x00000004 por cualquier reset. Solo palabras de acceso pueden usarse en MSCCDR.

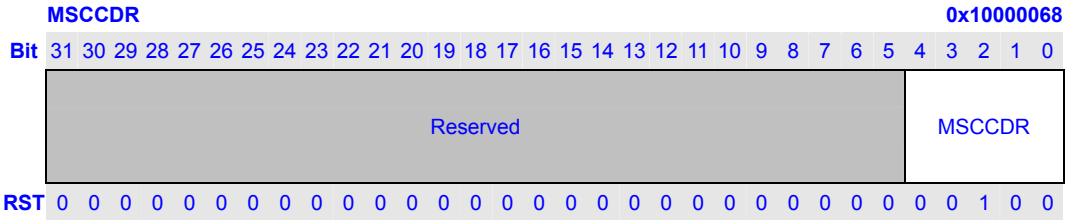


Figura 3.5: Registro MSCCDR

FUENTE: [1].

bits	Nombre	Descripción	RW
31:5	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0.	R
4:0	MSCCDR	Divisor de frecuencia para MSC. Especifica la razón de división para MSC la cual varia de 1 a 32(razón de división=MSCCDR+1)	RW

Tabla 3.6: Configuración registro MSCCDR

3.3.5 Registro UHCCDR

Es un registro de 32 bits de escritura/lectura que especifica el divisor de reloj del dispositivo UHC 48M. Este registro es inicializado a 0x00000004 por cualquier reset. Solo palabras de acceso pueden usarse en UHCCDR.

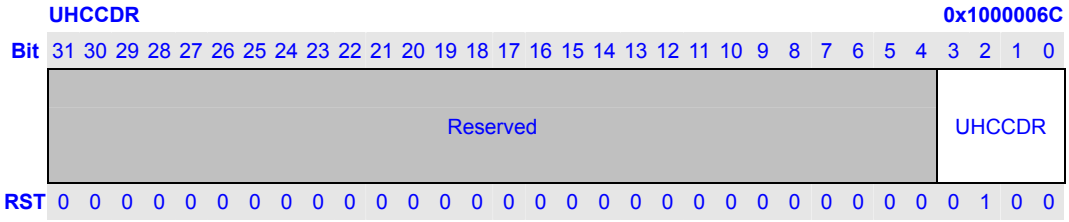


Figura 3.6: Registro UHCCDR

FUENTE: [1].

bits	Nombre	Descripción	RW
31:4	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0.	R
3:0	UHCCDR	Divisor de frecuencia para UHCCDR. Especifica la razón de división para UHC la cual varia de 1 a 16(razón de división=UHCCDR+1)	RW

Tabla 3.7: Configuración registro UHCCDR

3.3.6 Registro SSICDR

Es un registro de 32 bits de escritura/lectura que especifica el divisor de reloj del dispositivo SSI. Este registro es inicializado a 0x00000004 por cualquier reset. Solo palabras de acceso pueden usarse en SSICDR.

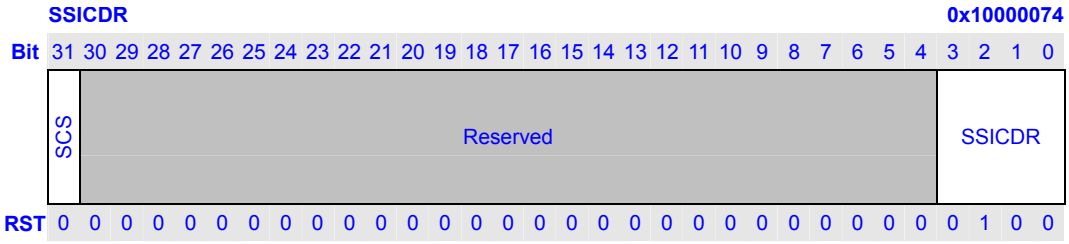


Figura 3.7: Registro SSICDR

FUENTE: [1].

bits	Nombre	Descripción	RW
31	SCS	Selecciona la fuente de reloj del SSI entre la salida del PLL y el pin EXCLK 0:la fuente de reloj del SSI es el pin EXCLK 1:la fuente de reloj del SSI es la salida del PLL	RW
30:4	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0.	R
3:0	SSICDR	Divisor de frecuencia para SSI. Especifica la razón de división para SSI la cual varia de 1 a 16(razón de división=SSICDR+1)	RW

Tabla 3.8: Configuración registro SSICDR

3.3.7 Registro CPPCR

Es un registro de 32 bits de escritura/lectura el cual controla el multiplicador del PLL, el estado on/off y el tiempo de estabilización. Este registro es inicializado a 0x28080011 por cualquier reset. Solo palabras de acceso pueden usarse en CPPCR.

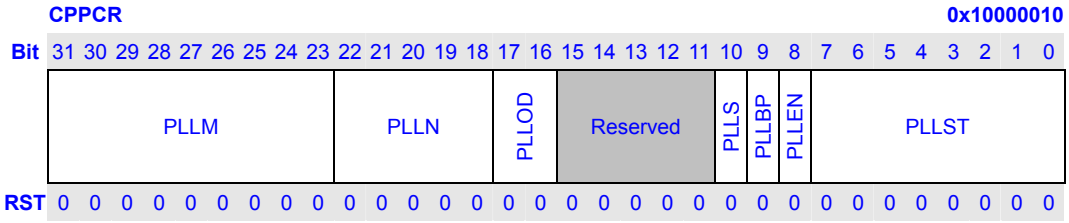


Figura 3.8: Registro CPPCR

FUENTE: [1].

bits	Nombre	Descripción	RW
31:23	PLLM	Realimentación de 9 bits del divisor del PLL	RW
22:18	PLLN	Entrada de 5 bits del divisor PLL	RW
17:16	SCS	00:dividir por 1 01:dividir por 2 10:dividir por 2 11:dividir por 4	RW
15:11	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0	R
10	PLLS	Bandera de estabilización de PLL 0: el PLL esta apagado o no se ha estabilizado. 1: el PLL esta encendido y estable.	R
9	PLLBP	Derivación PLL(PLL Bypass).si el campo PLLEN es 1, poner PLLBP en 1 provocara que se omita el paso por el PLL y por consiguiente la fuente de los divisores el el oscilador de 12 M. Si PLLEN es 0, poner PLLBP en 1 no tendrá ningún efecto. Si PLLEN es 1 entonces al establecer PLLBP en 0 provocara que la fuente de reloj de los divisores sea la salida del PLL	R
8	PLLEN	Habilitar PLL.cuando PLLEN es establecido en 1 el PLL empieza a funcionar, después de que se estabiliza el bit PLLS es 1.si el bit PLLBP es 0 la fuente de reloj de los divisores es la salida del PLL. Cuando PLLEN es establecido a 0 el PLL se encuentra apagado o inactivo, por lo tanto la fuente de reloj de los divisores es 12-MHz predominando sobre el valor del bit PLLBP	RW
7:0	PLLST	Tiempo de estabilización del PLL. Especifica el tiempo de estabilización por unidad de ciclos de RTCCLK(aproximadamente 32 KHz).Es usado al variar el multiplicador del PLL o cuando el PLL pasa de apagado a encendido. Es inicializado a H'11	RW

Tabla 3.9: Configuración registro CPPCR

3.4 Configuración del PLL

Hay tres diferentes divisores(N,M y NO) para establecer la frecuencia de salida de reloj del PLL llamada CLKOUT:

- Valor del divisor de entrada N.

  N=PLLN=CPPCR + 2

- Valor del divisor de realimentación M.

M=PLLM=CPPCR + 2.

- Valor del divisor de salida NO.

Configurando el divisor de salida	**Valor del divisor de salida**
0	1
1	2
2	2
3	4

Tabla 3.10: Configuración del PLL

Ahora con los datos ya calculados de M,N y NO obtenemos la frecuencia de salida.

$$CLK_{OUT} = X_{IN} \times (M \div N) \times (1 \div NO) \tag{3.1}$$

Donde.

- CLK_{OUT} representa la frecuencia de salida.
- X_{IN} representa la frecuencia de entrada del PLL.
- M representa el valor del divisor de entrada.
- N representa el valor del divisor de realimentación.
- NO representa el valor del divisor de salida.

Hay que tener cuidado con las mínimas y máximas frecuencias que se pueden obtener.

$$1MHZ \leq X_{IN} \div N \leq 15MHZ \tag{3.2}$$

$$100MHZ \leq CLK_{OUT} \times NO \leq 500MHZ \tag{3.3}$$

Las frecuencias de reloj principal pueden ser cambiadas separadamente o simultáneamente cambiando la razón de división. Las siguientes condiciones deben cumplirse:

- CCLK debe ser múltiplo entero de HCLK.
- La razón de división de CCLK y HCLK no puede ser 24 y 32.
- HCLK debe ser igual a MCLK o dos veces MCLK.
- HCLK y MCLK deben ser múltiplos enteros de PCLK.

No se deben violar las anteriores restricciones, de otra manera errores impredecibles pueden ocurrir.

3.5 Timer/Counter Unit TCU

El *Timer/Counter Unit* con salida PWM contiene 8 canales de salida de *timers* programables de 16-bits que pueden ser usados como *timer* o PWM.
El TCU tiene las siguientes características.

- 6 canales independientes, cada uno contiene.

  Contador.

  Registros de datos.

  Registros de control.

- Contador para cada Reloj independiente seleccionado por *software*.

  PCLK,EXTAL y RTCCLK pueden ser usados como contador de reloj.

  La proporción de división del reloj puede colocado a 1,4,16,64,256 y 1024 por *software*.

- Las interrupciones completas y medias pueden ser generadas en cada canal usando la comparación de registros de datos.

  timer 0 y el *timer* 1 tienen interrupciones por separado.

  timer 2-7 tienen una interrupción en común.

  timer 0-7 pueder ser usado como PWM seleccionando un nivel de señal inicial.

3.6 Descripción registros del Timer/Counter Unit TCU.

En esta sección se va a describir los registros del TCU. En la siguiente tabla se muestra la definición de todos los registros.

Nombre	**Descripción**	**RW**	**Valor Reset**	**Dirección**	**Tamaño de acceso**
TSR	Registro de parada del *timer*	R	0x00	0x1000201C	8
TSSR	Registro para establecer TSR	W	0x00	0x1000202C	8
TSCR	Registro para limpieza TSR	W	0x00	0x1000203C	8
TER	Registro para habilitar contador del *timer*	R	0x00	0x10002010	8
TESR	Registro para establecer TER	W	0x??	0x10002014	8
TECR	Registro para limpieza de TER	W	0x??	0x10002018	8
TFR	Registro bandera del *timer*	R	0x00000000	0x10002020	32
TFSR	Registro para establecer TFR	W	0x????????	0x10002024	32

TFCR	Registro para limpieza de TFR	W	0x????????	0x10002028	32
TMR	Registro de mascara del *timer*	R	0x00000000	0x10002030	32
TMSR	Registro para establecer TMR	W	0x????????	0x10002034	32
TMCR	Registro para limpieza de TMR	W	0x????????	0x10002038	32
TDFR0	Registro 0 de comparación completa del *timer*	RW	0x????	0x10002040	16
TDHR0	Registro 0 de comparación media del *timer*	RW	0x????	0x10002044	16
TCNT0	Contador *timer* 0	RW	0x????	0x10002048	16
TCSR0	Registro de control del *timer* 0	RW	0x0000	0x1000204C	16
TDFR1	Registro 1 de comparación completa del *timer*	RW	0x????	0x10002050	16
TDHR1	Registro 1 de comparación media del *timer*	RW	0x????	0x10002054	16
TCNT1	Contador *timer* 1	RW	0x????	0x10002058	16
TCSR1	Registro de control del *timer* 1	RW	0x0000	0x1000205C	16
TDFR2	Registro 2 de comparación completa del *timer*	RW	0x????	0x10002060	16
TDHR2	Registro 2 de comparación media del *timer*	RW	0x????	0x10002064	16
TCNT2	Contador *timer* 2	RW	0x????	0x10002068	16
TCSR2	Registro de control del *timer* 2	RW	0x0000	0x1000206C	16
TDFR3	Registro 3 de comparación completa del *timer*	RW	0x????	0x10002070	16
TDHR3	Registro 3 de comparación media del *timer*	RW	0x????	0x10002074	16
TCNT3	Contador *timer* 3	RW	0x????	0x10002078	16
TCSR3	Registro de control del *timer* 3	RW	0x0000	0x1000207C	16
TDFR4	Registro 4 de comparación completa del *timer*	RW	0x????	0x10002080	16
TDHR4	Registro 4 de comparación media del *timer*	RW	0x????	0x10002084	16
TCNT4	Contador *timer* 4	RW	0x????	0x10002088	16
TCSR4	Registro de control del *timer* 4	RW	0x0000	0x1000208C	16
TDFR5	Registro 5 de comparación completa del *timer*	RW	0x????	0x10002090	16
TDHR5	Registro 5 de comparación media del *timer*	RW	0x????	0x10002094	16
TCNT5	Contador *timer* 5	RW	0x????	0x10002098	16
TCSR5	Registro de control del *timer* 5	RW	0x0000	0x1000209C	16
TDFR6	Registro 6 de comparación completa del *timer*	RW	0x????	0x100020A0	16
TDHR6	Registro 6 de comparación media del *timer*	RW	0x????	0x100020A4	16
TCNT6	Contador *timer* 6	RW	0x????	0x100020A8	16

TCSR6	Registro de control del *timer* 6	RW	0x0000	0x100020AC	16
TDFR7	Registro 7 de comparación completa del *timer*	RW	0x????	0x100020B0	16
TDHR7	Registro 7 de comparación media del *timer*	RW	0x????	0x100020B4	16
TCNT7	Contador *timer* 7	RW	0x????	0x100020B8	16
TCSR7	Registro de control del *timer* 7	RW	0x0000	0x100020BC	16

Tabla 3.11: Descripción registros del TCU

3.6.1 Registro TCSR

Es un registro de 16-bits de lectura/escritura. Este contiene los bits de control para cada canal. Este es inicializado a 0x00 por un reset.

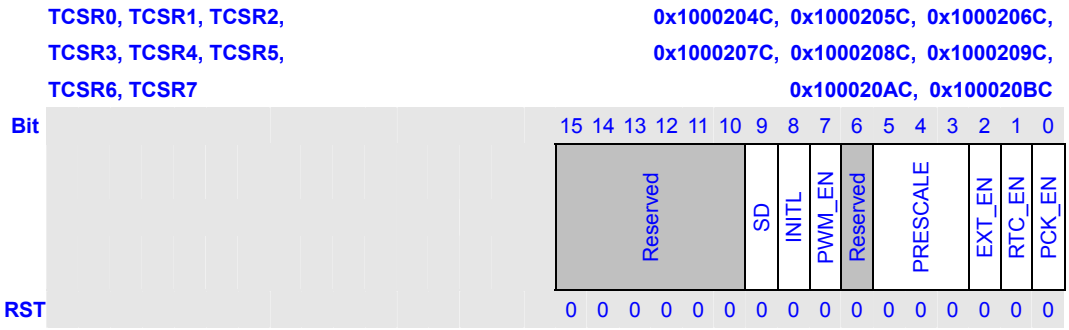


Figura 3.9: Registro TCSR

FUENTE: [1].

bits	Nombre	Descripción	RW
15:10	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0.	R
9	SD	Apagar la salida del PWM 0: Apagar correctamente 1: Apagar abruptamente	RW
8	INITL	Seleccionar un nivel de salida para la salida del PWM 1: alta 0: baja	RW
15:11	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0	R

7	PWM_EN	pin de salida como PWM 1: Habilita el pin de salida como PWM. 0: Deshabilita el pin de salida como PWM y el pin tendrá un valor inicial de acuerdo a INITL.	RW
6	Reservado	Escribir en estos campos no tiene efecto, y al leerlo siempre se lee 0.	R
5:3	PRESCALE	Estos bits seleccionan la frecuencia de reloj del contador TCNT. No cambiar este campo cuando el canal este siendo utilizado 000= reloj interno= CLK/1 001= reloj interno= CLK/4 010= reloj interno= CLK/16 011= reloj interno= CLK/64 100= reloj interno= CLK/256 101= reloj interno= CLK/1024 110-111= reloj interno= reservado	RW
2	EXT_EN	Selecciona EXTAL como reloj de entrada al *timer* 1: Habilitar 0: Deshabilitar	RW
1	RTC_EN	Selecciona RTCCLK como reloj de entrada al *timer* 1: Habilitar 0: Deshabilitar	RW
0	PCK_EN	Selecciona PCLK como reloj de entrada al *timer* 1: Habilitar 0: Deshabilitar	RW

Tabla 3.12: Configuración registro TCSR

3.6.2 Registro TDFR

Los registros de datos para la comparación completa TDFR son usados para almacenar los datos de la comparación con el contenido del contador ascendente TCNT. Este registro es de lectura/escritura.

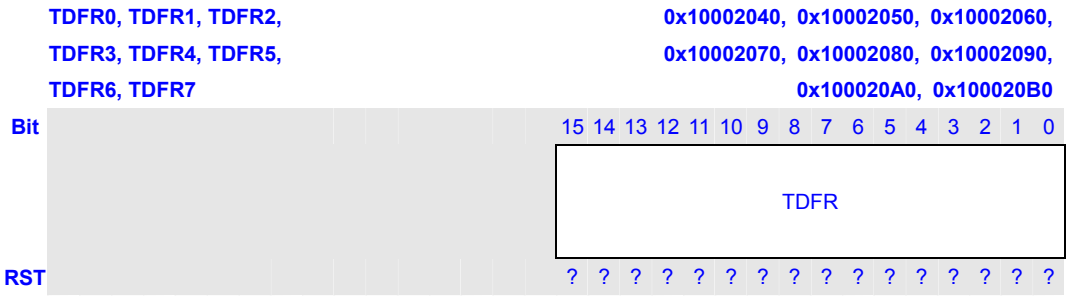


Figura 3.10: Registro TDFR

FUENTE: [1].

3.6.3 Registro TDHR

Los registros de datos para la comparación media TDFR son usados para almacenar los datos de la comparación con el contenido del contador ascendente TCNT. Este registro es de lectura/escritura.

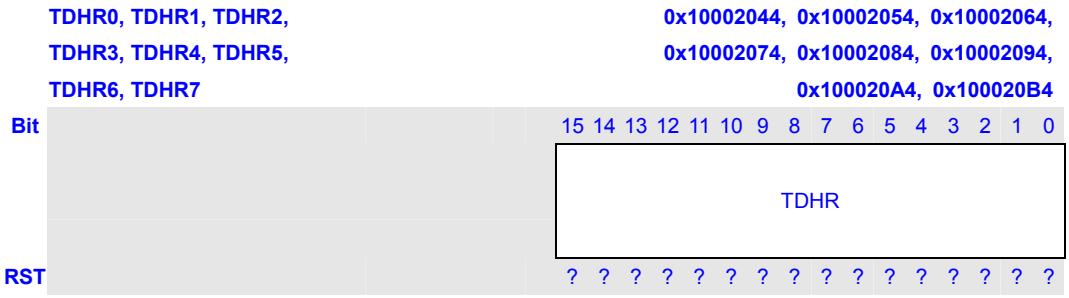


Figura 3.11: Registro TDHR

FUENTE: [1].

3.6.4 Registro TCNT

Es un registro de lectura/escritura de 16-bits. El contador ascendente TCNT puede ser inicializado a 0 por *software*.Cuando TCNT es igual a TDFR, este se inicializará a 0 y seguirá contando.

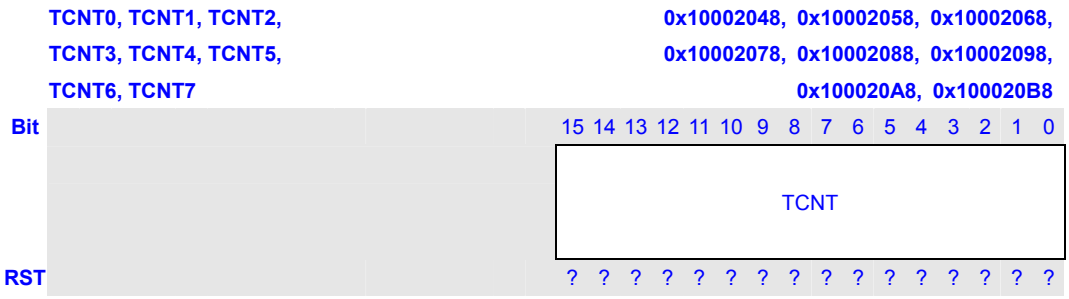


Figura 3.12: Registro TCNT

FUENTE: [1].

3.6.5 Registro TER

Registro de 8-bits de solo lectura. Este contiene los bits de control que habilitan el contador de cada canal, inicializado a 0x00 por cualquier reset.

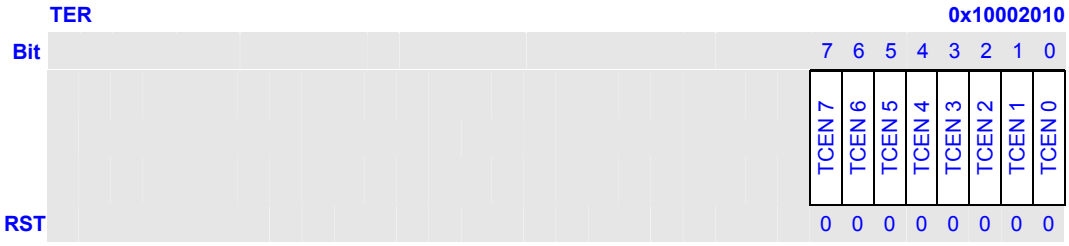


Figura 3.13: Registro TER

FUENTE: [1].

bits	Nombre	Descripción	RW
7	TCEN 7	Habilita el contador en el *timer* 7 1: Comienza conteo ascendente 0: Para el conteo	R
6	TCEN 6	Habilita el contador en el *timer* 6 1: Comienza conteo ascendente 0: Para el conteo	R
5	TCEN 5	Habilita el contador en el *timer* 5 1: Comienza conteo ascendente 0: Para el conteo	R
4	TCEN 4	Habilita el contador en el *timer* 4 1: Comienza conteo ascendente 0: Para el conteo	R
3	TCEN 3	Habilita el contador en el *timer* 3 1: Comienza conteo ascendente 0: Para el conteo	R
2	TCEN 2	Habilita el contador en el *timer* 2 1: Comienza conteo ascendente 0: Para el conteo	R
1	TCEN 1	Habilita el contador en el *timer* 1 1: Comienza conteo ascendente 0: Para el conteo	R
0	TCEN 0	Habilita el contador en el *timer* 0 1: Comienza conteo ascendente 0: Para el conteo	R

Tabla 3.13: Configuración registro TER

3.6.6 Registro TESR

Es un registro de 8-bits de solo escritura. Este contiene los bits que habilitan el contador en cada canal tendiendo en cuenta que se pueden hablitar varios canales al mismo tiempo.

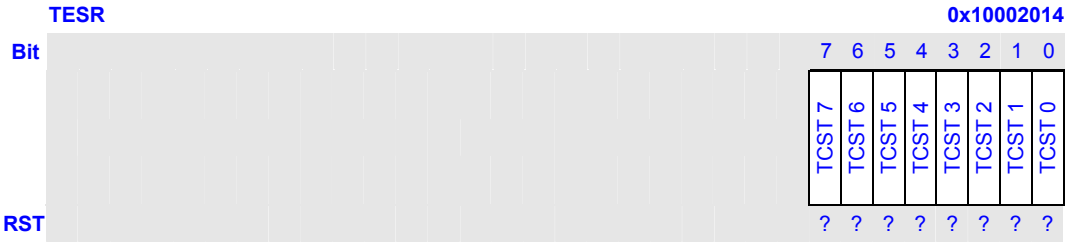


Figura 3.14: Registro TESR

FUENTE: [1].

bits	Nombre	Descripción	RW
7	TCST 7	Escribe en el bit TCEN 7 de TER 1: Fijar en 1 el bit TCEN 7 0: Ignora	R
6	TCST 6	Escribe en el bit TCEN 6 de TER 1: Fijar en 1 el bit TCEN 6 0: Ignora	R
5	TCST 5	Escribe en el bit TCEN 5 de TER 1: Fijar en 1 el bit TCEN 5 0: Ignora	R
4	TCST 4	Escribe en el bit TCEN 4 de TER 1: Fijar en 1 el bit TCEN 4 0: Ignora	R
3	TCST 3	Escribe en el bit TCEN 3 de TER 1: Fijar en 1 el bit TCEN 3 0: Ignora	R
2	TCST 2	Escribe en el bit TCEN 2 de TER 1: Fijar en 1 el bit TCEN 2 0: Ignora	R
1	TCST 1	Escribe en el bit TCEN 1 de TER 1: Fijar en 1 el bit TCEN 1 0: Ignora	R
0	TCST 0	Escribe en el bit TCEN 0 de TER 1: Fijar en 1 el bit TCEN 0 0: Ignora	R

Tabla 3.14: Configuración registro TESR

3.6.7 Registro TECR

Es un registro de 8-bits de solo escritura. Este contiene los bits que reinician el contador en cada canal tendiendo en cuenta que se pueden habilitar varios canales al mismo tiempo.

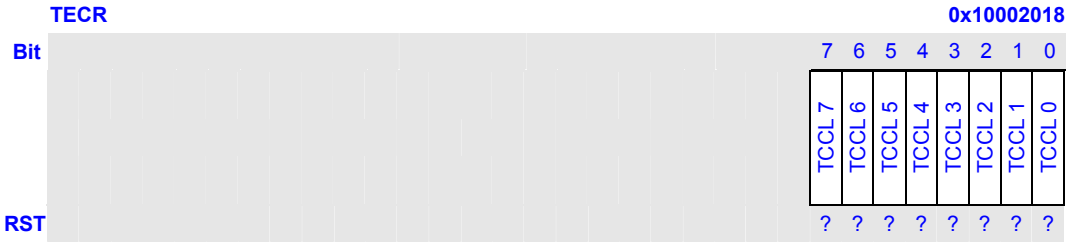


Figura 3.15: Registro TECR

FUENTE: [1].

bits	Nombre	Descripción	RW
7	TCCL 7	Escribe en el bit TCEN 7 de TER 1: Fijar en 0 el bit TCEN 7 0: Ignora	R
6	TCCL 6	Escribe en el bit TCEN 6 de TER 1: Fijar en 0 el bit TCEN 6 0: Ignora	R
5	TCCL 5	Escribe en el bit TCEN 5 de TER 1: Fijar en 0 el bit TCEN 5 0: Ignora	R
4	TCCL 4	Escribe en el bit TCEN 4 de TER 1: Fijar en 0 el bit TCEN 4 0: Ignora	R
3	TCCL 3	Escribe en el bit TCEN 3 de TER 1: Fijar en 0 el bit TCEN 3 0: Ignora	R
2	TCCL 2	Escribe en el bit TCEN 2 de TER 1: Fijar en 0 el bit TCEN 2 0: Ignora	R
1	TCCL 1	Escribe en el bit TCEN 1 de TER 1: Fijar en 0 el bit TCEN 1 0: Ignora	R
0	TCCL 0	Escribe en el bit TCEN 0 de TER 1: Fijar en 0 el bit TCEN 0 0: Ignora	R

Tabla 3.15: Configuración registro TECR

Bibliografía

[1] Datasheet JZ4725, *multimedia Application processor*, Ingenic Semiconductor Co. Ltd, Revision: 1.0,May 2009

CAPÍTULO 4

INTERFAZ DE COMUNICACIÓN ENTRE EL PROCESADOR Y EL FPGA

4.1 Controlador de memoria externa EMC

Habilita las conexiones de diferentes memorias al procesador como lo son la memoria estática y la memoria NAND. En éste capitulo se va hablar de La interfaz de memoria estática que soporta la selección de 4 señales de habilitación (*chip select*) (CS4-1) con configuración independiente en cada banco, ahora el tamaño y las direcciones base de los bancos de memoria estática son programables por *software* que pueden ser configurados de acuerdo a las memorias conectadas al procesador, en las siguientes tablas podemos ver las direcciones y los *chip select* por defecto después de oprimir algún reset.

dirección inicio	**dirección final**	**Espacio de memoria**	**capacidad**
H'0000 0000	H'07FF FFFF	SDRAM	128 MB
H'0800 0000	H'0FFF FFFF	Estática	128 MB
H'1000 0000	H'13FF FFFF	Interna entrada/salida	64 MB
H'1400 0000	H'1bFF FFFF	Estática	128 MB
H'1C00 0000	H'1FBF FFFF	Sin usar	60 MB
H'1FC0 0000	H'1FC0 0FFF	ROM	4 KB
H'1FC0 1000	H'1FFF FFFF	Sin usar	4095 KB
H'2000 0000	H'BFFF FFFF	SDRAM	2944 MB
H'D000 0000	H'FFFF FFFF	Espacio reservado	512 MB

Tabla 4.1: Espacio del mapa de direcciones fisícas

chip select	**Memoria conectada**	**capacidad**	**tamaño memoria**	**dirección inicio**	**dirección final**
CS1#	banco 1 memoria estática	8 MB	8,16,32	H'1800 0000	H'1BFF FFFF
CS2#	banco 2 memoria estática	8 MB	8,16,32	H'1400 0000	H'17FF FFFF
CS3#	banco 3 memoria estática	8 MB	8,16,32	H'0C00 0000	H'0FFF FFFF
CS4#	banco 4 memoria estática	8 MB	8,16,32	H'0800 0000	H'0BFF FFFF
DCS#	banco SDRAM	128 MB	16,32	H'2000 0000	H'27FF FFFF

Tabla 4.2: Señales de habilitación por defecto

Cada *chip select* puede acceder directamente a memorias o dispositivos de entrada/salida que son de 8-bits. 16-bits o 32-bits. El dispositivo conectado a una linea de *chip select* tiene 2 registros asociados que controlan esta operación y la sincronización de acceso al dispositivo externo. El registro de direcciones de la memoria estática SACRn especifica la dirección base y el tamaño de cada dispositivo, habilitando cualquier dispositivo localizado en cualquier lugar del rango de direcciones físicas.
La interfaz de memoria estática incluye las siguientes señales.

- cuatro *chip select*, CS4-1#.
- veintitrés señales de direcciones. A22-A0.
- una señal habilitación de lectura RD#.
- una señal habilitación de escritura WE#.
- cuatro bits de habilitación, BE3$\tilde{1}$#.
- un pin de espera, WAIT#.

4.2 Descripción de registros

Nombre	Descripción	RW	Valor reset	Dirección	Tamaño
SMCR1	Registro 1 de control	RW	0x0FFF7700	0x13010014	32
SMCR2	Registro 2 de control	RW	0x0FFF7700	0x13010018	32
SMCR3	Registro 3 de control	RW	0x0FFF7700	0x1301001C	32
SMCR4	Registro 4 de control	RW	0x000018FC	0x13010020	32
SACR1	Registro de configuración de direcciones del banco 1	RW	0x000018FC	0x13010034	32
SACR2	Registro de configuración de direcciones del banco 2	RW	0x000016FE	0x13010038	32
SACR3	Registro de configuración de direcciones del banco 3	RW	0x0FFF14FE	0x1301003C	32
SACR4	Registro de configuración de direcciones del banco 4	RW	0x00000CFC	0x13010040	32

Tabla 4.3: Descripción registros EMC

Las siguientes figuras muestran la sincronización de las señales en una memoria normal.

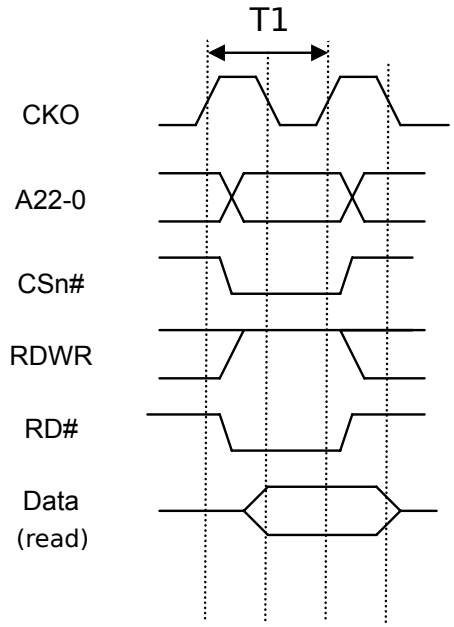


Figura 4.1: señales de reloj en modo lectura

FUENTE: [1]

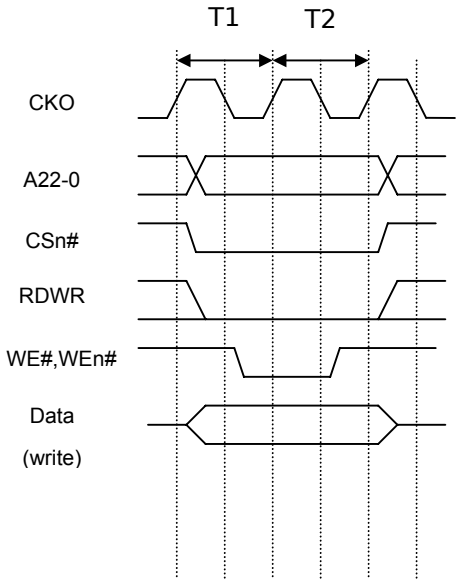


Figura 4.2: señales de reloj en modo escritura

FUENTE: [1]

Bibliografía

[1] Datasheet JZ4725, *multimedia Application processor*, Ingenic Semiconductor Co. Ltd, Revision: 1.0,May 2009

CAPÍTULO 5

CONFIGURACIÓN Y ADECUACIÓN DEL CONVERTIDOR ANALÓGICO-DIGITAL THS1230 DE TEXAS INSTRUMENTS

A continuación se describe como se configuro el convertidor analógico-digital con el fin de adaptarlo al proyecto, cabe anotar que la configuración se hizo en base a las condiciones de operación recomendadas en la hoja de datos de dicho convertidor.

5.1 CONDICIONES DE OPERACIÓN

Las condiciones de operación recomendadas en la hoja de datos del convertidor THS1230 son:

- AVDD=3.3v=DVDD=3.3v;
- fs= 30 MHz.
- 50% de ciclo de trabajo;
- MODE1(CON1=0 CON0=1);
- 1v de spam de entrada;
- referencia interna;
- Tmin a Tmax;

5.1.1 Configuración de las condiciones de operación.

Para poder configurar las condiciones de operación a las que se recomiendan en la hoja de datos se deben conectar ciertos pines a ciertos niveles, a continuación se muestra el procedimiento para la configuración de estas condiciones:

5.1.1.1 Conexión de EXTREF para generación interna de referencia.

El pin **EXTREF** que es el que controla si la generación de los voltajes de referencia VREFT y VREFB es interna o externa debe ser conectado a **DGND** para que la generación de dichas referencias sea interna.

5.1.1.2 Habilitación de los *buffers* de salida a través del pin OE

Para que los buffers de salida funcionen se debe conectar el pin OE a un 0 lógico, es decir conectarlo a **DGN**.

5.1.1.3 Conexión de los pines CON0 y CON1 para configurar el modo de operación.

Para que el convertidor trabaje bajo el modo 1 en el que la ganancia de este es 1 se debe conectar **CON0** a un 1 lógico y **CON1** a un cero lógico. Es decir conectamos *CON0* a DVDD y *CON1* a DGNG en la gráfica 1 se muestran los posibles modos de operación.

MODE	CON1	CON0	MODE OF OPERATION
0	0	0	Device powered down
1	0	1	Differential mode × 1
2	1	0	Differential mode × 0.5
3	1	1	Not used

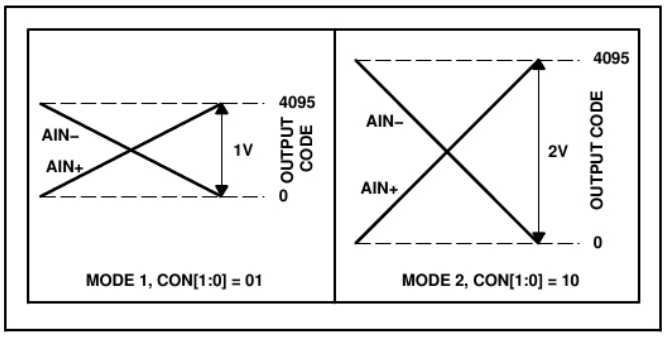


Figura 5.1: Modos de operación

FUENTE: Hojas de datos del convertidor.

5.1.2 Señal de entrada.

Sea cual sea el modo de operación y la naturaleza de la entrada esta debe estar limitada a un intervalo de va desde 1.15v hasta 2.15v (**VREFT=2.15v y VREFB=1.15v**), para esto se deben implementar

el acople AC con su respectiva adecuación que aseguren que la señal este dentro de este intervalo.

En la figura 5.2 se muestra el esquemático que corresponde a la configuración mencionada anteriormente, donde DVDD se refiere a la fuente de alimentación digital, AVDD a la fuente de alimentación analógica, AGND a la tierra analógica y DGND a la tierra digital, el par de leds se adicionaron con fines de prueba. algunos detalles faltantes de la figura 2 serán explicados mas adelante.

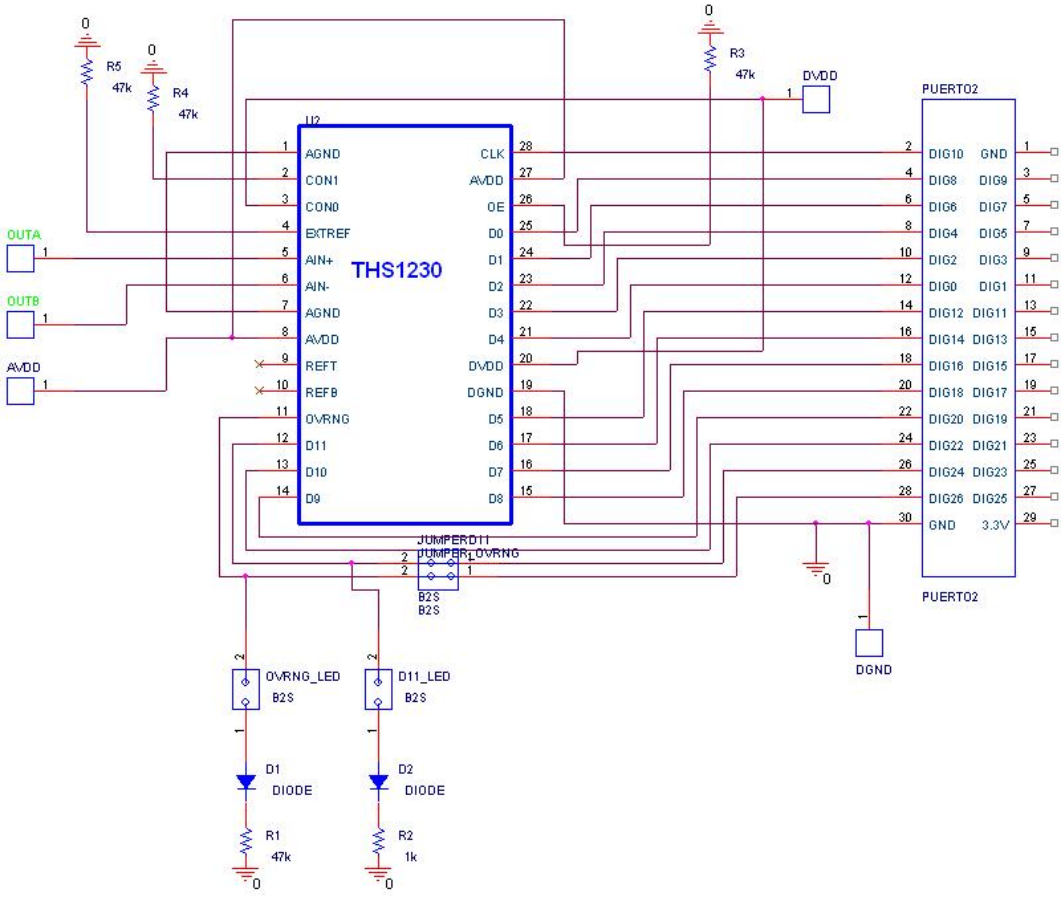


Figura 5.2: esquemático de la configuración del conversor

FUENTE: Los autores.

5.2 Implementación de la etapa de adecuación de señal.

La necesidad de implementación de una etapa de adecuación de señal se debe al rango de entrada limitado del convertidor THS1230 de *TEXAS INSTRUMENTS*. El rango de entrada de este conver-

tidor en particular es definido por los voltajes de referencia,todo voltaje de entrada mayor a VREFT o menor a VREFB están fuera del rango del convertidor y provocara la activación de la señal OVRNG. La configuración del convertidor establece los valores de VREFT y VREFB en 2.15v y 1.15v respectivamente por lo cual se debe implementar un mecanismo que asegure la ubicación de la señal de entrada dentro de este rango(1.15v-2.15v).

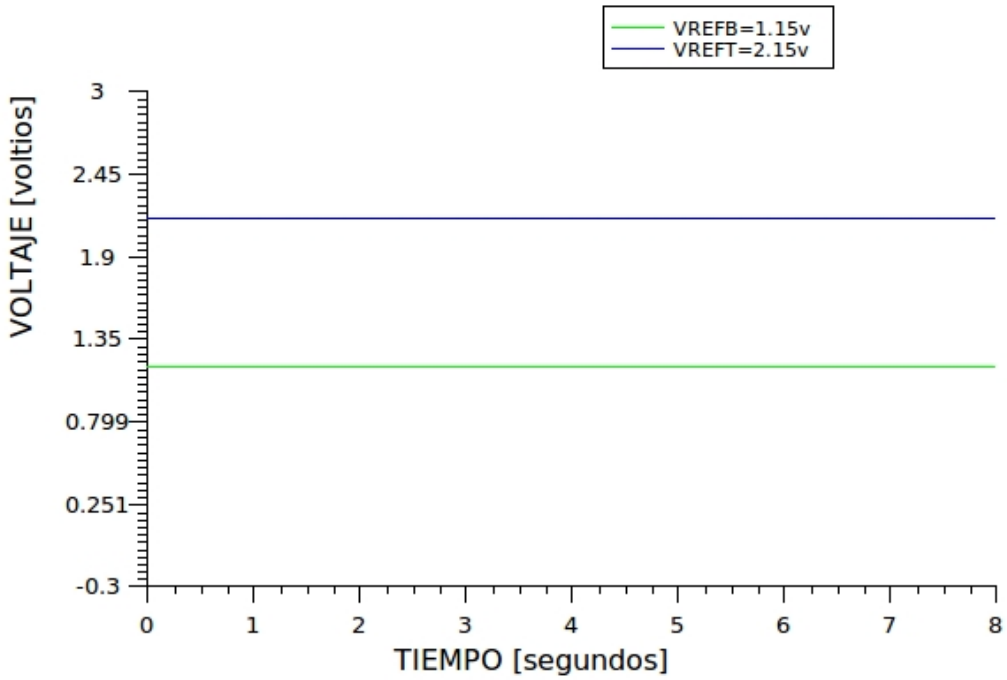


Figura 5.3: Rango de entrada del ADC

FUENTE: Los autores.

Hay que aclarar que esta etapa es para acople de señales alternas, por lo tanto y con el fin de mantener la señal de entrada en el rango adecuado se deben usar capacitores de bloqueo de DC y así evitar que cualquier componente DC desplace la señal de entrada. Por otro lado es necesario después de bloquear cualquier componente de DC que traiga la señal se debe inyectar un componente de DC conocido y así ubicarla en la mitad del rango permisible(1.65v) con el fin de que pueda variar en sentido ascendente y descendente con la misma libertad.

Como ejemplo En la figura B se muestra una señal senoidal de amplitud 2v con componente DC de 5v, y en la figura C se muestra la señal después

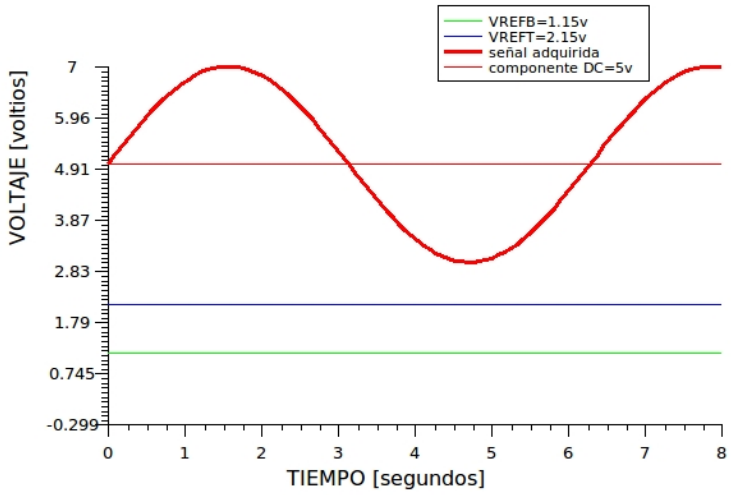


Figura 5.4: Señal senoidal con componente DC.

FUENTE: Los autores.

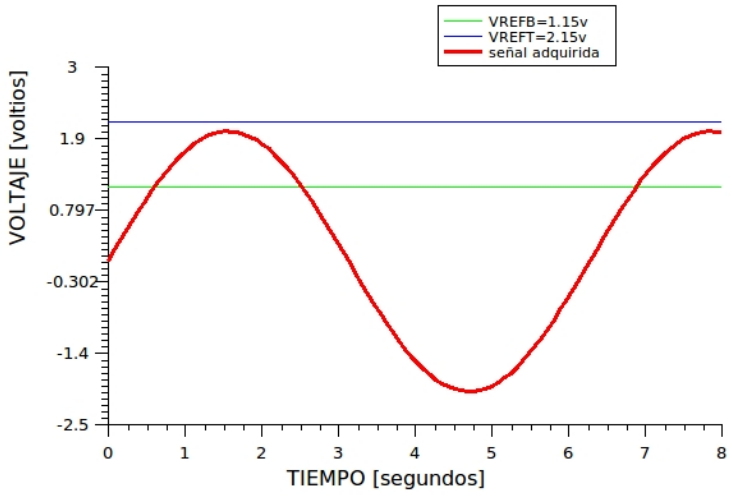


Figura 5.5: Señal senoidal pasada a través de los capacitores de bloqueo DC.

FUENTE: Los autores.

Por ultimo para aumentar el rango de amplitudes de las señales de entrada se procede a implementar un circuito atenuador de ganancia 1/12, con esto se gana una amplitud máxima de entrada de 6v, y una amplitud mínima de entrada de -6v. La escogencia de la ganancia del circuito atenuador se deja a consideración del diseñador dependiendo de las especificaciones o requerimientos de los proyectos individuales, además cabe resaltar que la inyección de DC se hace junto con la atenuación.

En la figura D se muestra la señal ejemplo de las figuras B y C luego de ser atenuada, y en la figura E e muestra la señal atenuada luego de ser inyectada con un componente de DC de 1.65 v(mitad del rango de entrada) donde se puede notar que la ubicación es la que se quería en un principio, es decir en el centro del rango de entrada y dentro de este.

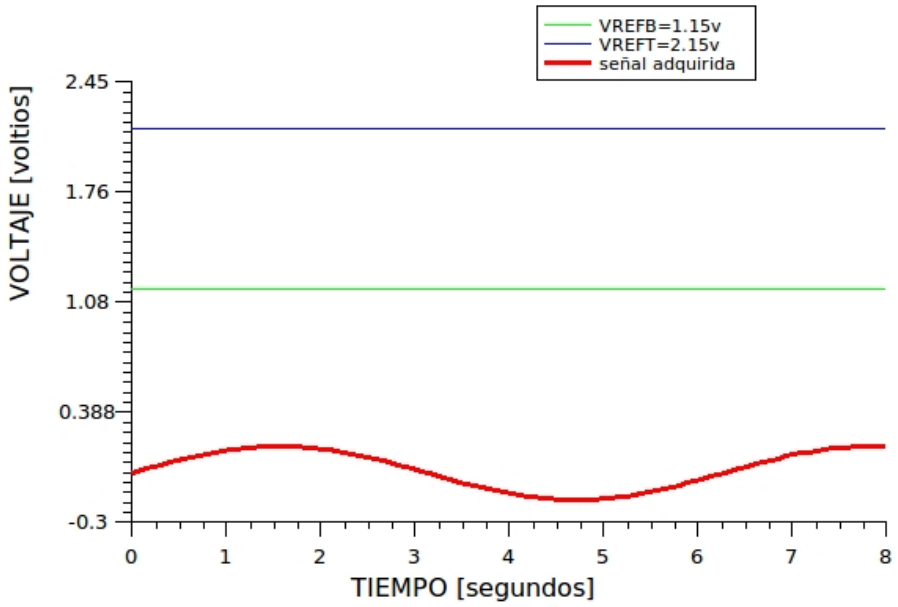


Figura 5.6: Señal senoidal atenuada.

FUENTE: Los autores.

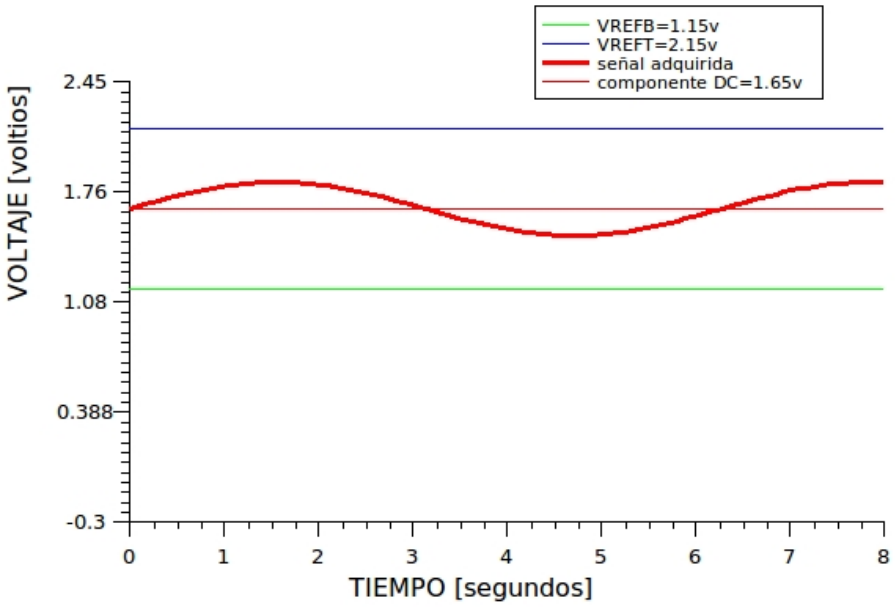


Figura 5.7: señal senoidal atenuada y con componente DC=1.65 v.

FUENTE: Los autores.

5.3 DESCRIPCIÓN DE LAS ETAPAS

Habiendo aclarado aspectos preliminares, se procede a describir las etapas implementadas. A continuación se muestra la figura F que corresponde a la implementación del sistema completo. En donde se pueden diferenciar la etapa de adecuación de señal y el montaje del conversor analógico-digital, además de algunos circuitos complementarios pero importantes como lo son reguladores y separación de fuentes.

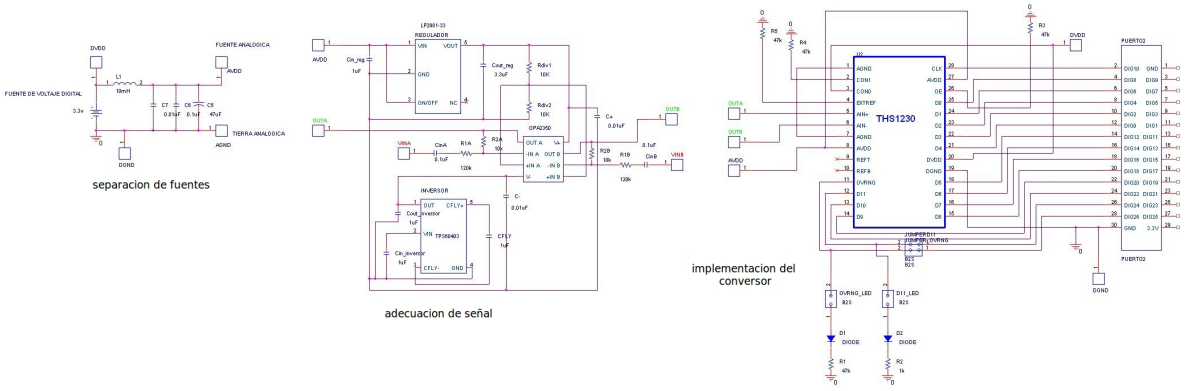


Figura 5.8: señal senoidal atenuada y con componente DC=1.65 v.

FUENTE: Los autores.

La etapa de adecuación recomendada en la hoja de datos del conversor es la siguiente:

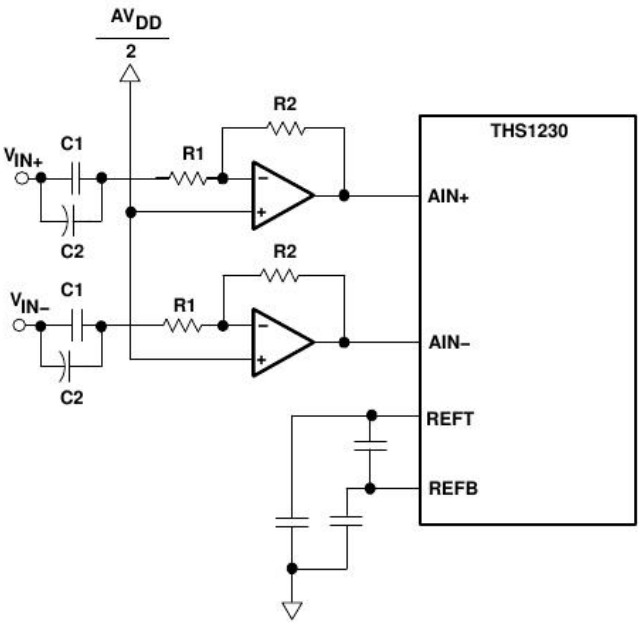


Figura 5.9: Etapa de adecuación recomendada

FUENTE: Hojas de datos del convertidor.

5.3.1 Etapa de adecuación de señal

La etapa implementada se muestra con mas detalle en la figura 5.10:

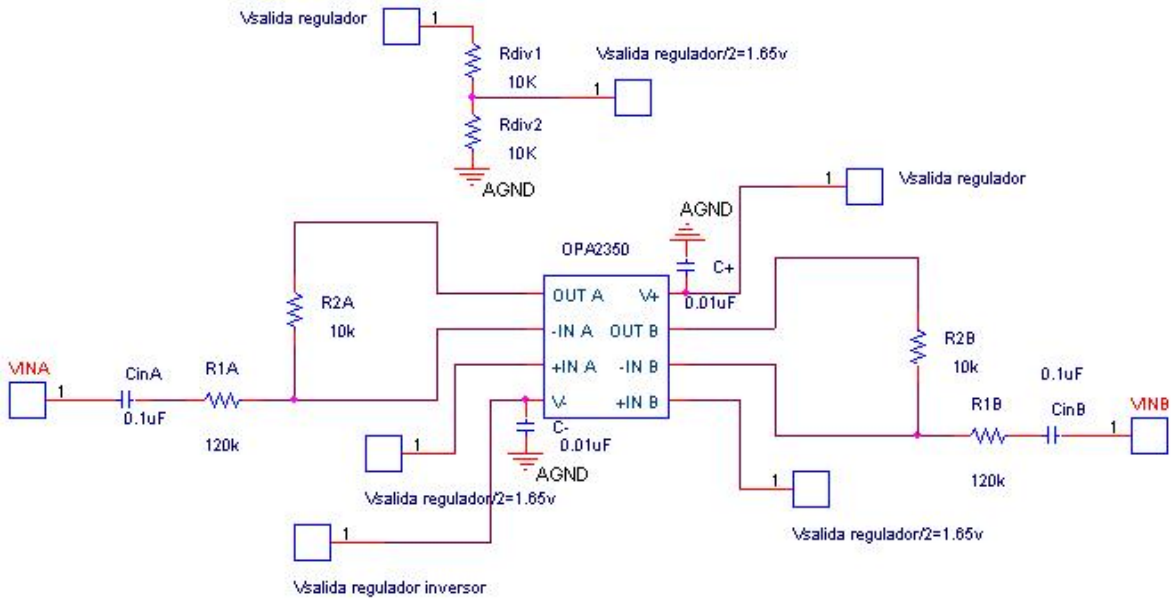


Figura 5.10: Etapa de adecuación implementada

FUENTE: Los autores.

Como se explico en un principio esta etapa se encarga de atenuar la señal para aumentar el rango de amplitudes permisibles y además de montarla sobre un nivel de DC igual a 1.65v que es el centro de el rango de entrada del convertidor. En el montaje se uso el dispositivo opa2350 debido a su buen rendimiento a altas frecuencias además de tener dos amplificadores operacionales y ser compatible con los voltajes de alimentación del convertidor analógico-digital.

Los capacitores CinA y CinB son los encargados de bloquear cualquier componente de DC de la señal adquirida, son capacitores cerámicos de valor 0.1 uF.

Por otro lado la ganancia de los dos amplificadores es la misma e igual a R2A/R1A=1/12 con el fin de aceptar una amplitud de señal máxima de 6v. El nivel de DC se crea partir de la salida de la etapa de regulación(explicada mas adelante), y un divisor de tensión compuesto por dos resistencias llamadas Rdiv1 y Rdiv2 cuya función es dividir el voltaje de salida del regulador(3.3v) por 2 para así obtener un voltaje igual a 1.65v.

El voltaje de DC obtenido es llevado a la entrada no inversora de cada amplificador operacional como

en el diagrama recomendado por la hoja de datos y así lograr adicionar este componente en la señal luego de pasar a través de los capacitores de bloqueo de DC y luego de ser atenuada.

5.3.2 Implementación de la fuente de alimentación dual.

El dispositivo opa2350 necesita para su funcionamiento ser alimentado de manera dual, es por esta razón que se debe implementar una etapa adicional que cumpla esta tarea. Esta etapa esta compuesta de un regulador y un regulador inversor, ambos alimentados por una sola fuente, el esquema de esta etapa se muestra en la figura 5.11.

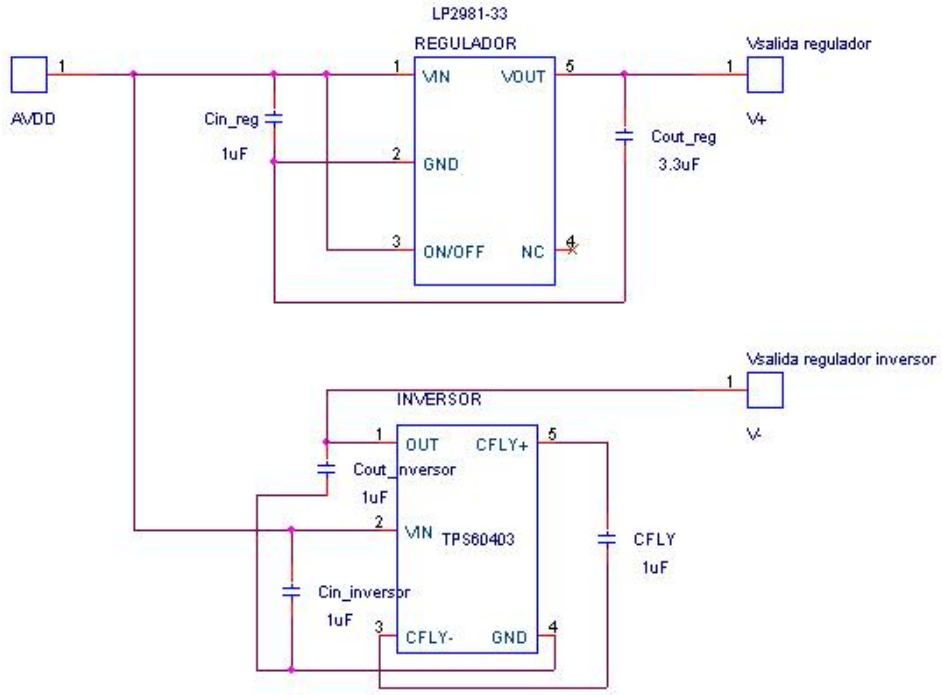


Figura 5.11: Circuito para alimentación dual

FUENTE: Los autores.

Los reguladores usados son LP2981-33 y TPS60403 de *TEXAS INSTRUMENTS* donde el ultimo es un regulador inversor, ambos son alimentados con una única fuente de voltaje establecida en 3.3v, las salidas son 3.3v y -3.3v respectivamente. Los capacitores usados son cerámicos fijos y se especifican en la hoja de datos de cada regulador.

5.3.3 Separación de fuentes analógica y digital.

Los circuitos de señales mixtas son aquellos que involucran señales de naturaleza analógica y digital, conllevando a problemas en la calidad de las señales analógicas. En los sistemas con señales mixtas los circuitos digitales demandan cambios de corriente y voltaje dependiendo de su frecuencia de trabajo, cuando esta frecuencia es alta los cambios de corriente en las fuentes que los alimentan son grandes, generando transitorios y perturbaciones que podrían afectar los circuitos analógicos si estos son alimentados con la misma fuente que se alimentan los circuitos digitales. Por esta razón es necesario alimentar de manera independiente las partes analógicas y digitales, el circuito que se propone en en la figura J es capaz de alimentar de manera independiente los circuitos analógicos(etapa de adecuación) y digitales(conversor THS1230).

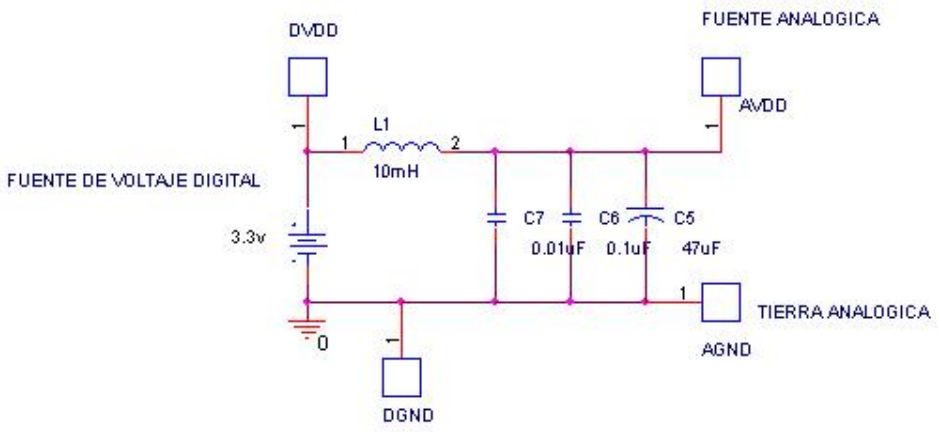


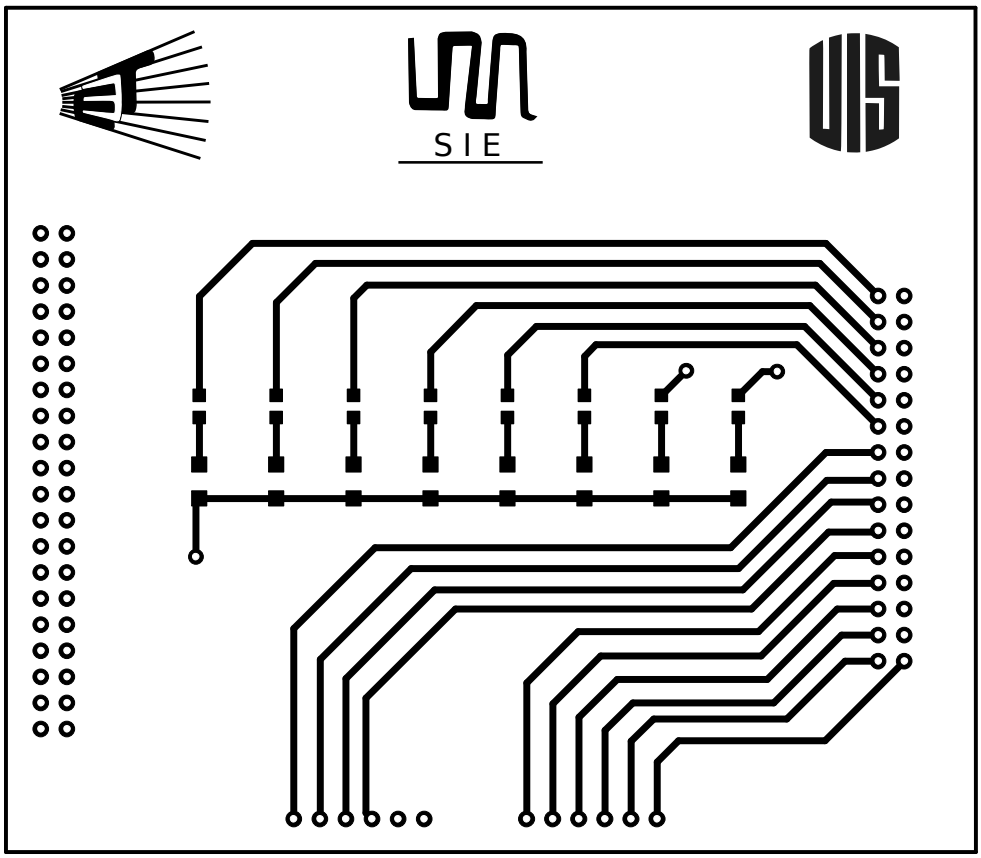
Figura 5.12: Circuito para separación de fuentes

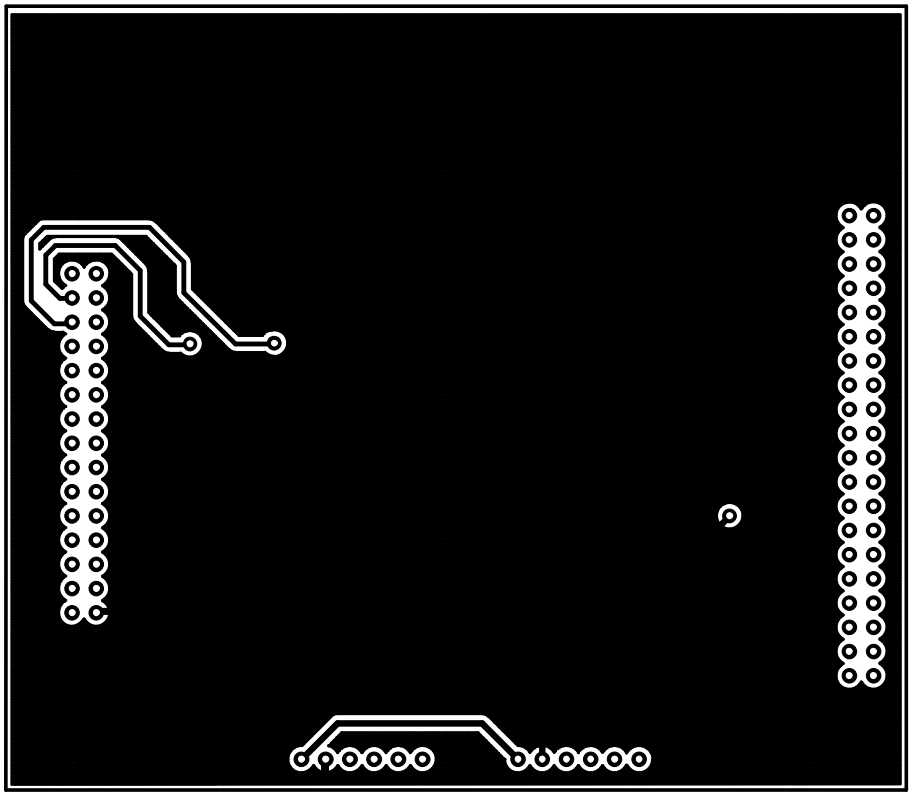
FUENTE: Los autores.

La inductancia L1 tiene como función servir de atenuación para señales en cualquier frecuencia mayor a 0, dejando pasar señales DC, los capacitores C7, C6 y C5 tienen como función filtrar señales o perturbaciones originadas por las señales digitales en frecuencias altas, medias y bajas respectivamente, C7 y C6 son cerámicos y C5 es electrolítico.

Con el diseño de este circuito se completan las etapas necesarias para la implementación del sistema de adquisición de datos, por otro lado es conveniente aclarar que la frecuencia de trabajo es de 12.5 MHz, los otros parámetros de diseño son tratados en las guías de la asignatura arquitectura de computadores, por lo que este capitulo solo trata aspectos de implementación y diseño en base a los requerimientos planteados en dichas guías.

ANEXO C. PLACA DE CIRCUITO IMPRESO DEL DISPLAY SIETE SEGMENTOS





ANEXO D. PLACA DE CIRCUITO IMPRESO DEL SISTEMA DE ADQUISICIÓN DE DATOS