ENVIO DE DATOS POR GIGABIT ETHERNET DESDE UNA INTERFAZ LVDS USANDO SOC (SINGLE BOARD COMPUTER + FPGA)

DATA TRANSMISSION THROUGH GIGABIT ETHERNET FROM A LVDS INTERFACE USING A SOC (SINGLE BOARD COMPUTER + FPGA)

DAVID GEORGE SHATWELL PITTALUGA

Universidad de Ingeniería y Tecnología - UTEC

Departamento de Ingeniería Electrónica

david.shatwell@utec.edu.pe

Asesor: Joaquín Verástegui

Área de I&D+I – Radio Observatorio de Jicamarca

Resumen

En el presente trabajo se hace una descripción del diseño e implementación de un sistema capaz de transmitir datos a alta velocidad desde JARS 2.0 hacia un PC por medio de Gigabit Ethernet usando un SoC. El sistema consta principalmente de dos partes: adquisición de datos de la interfaz LVDS y envío hacia la PC por medio de un protocolo de comunicación. Para la adquisición de datos de la interfaz LVDS fue necesario diseñar un sistema en el FPGA que demultiplexe y copie los datos a una memoria compartida con el procesador. Luego, para el envío de datos se diseñó un software que lea los datos de la memoria compartida y los envíe hacia la PC por medio del protocolo UDP.

Palabras clave: JARS 2.0, Gigabit Ethernet, SoC, LVDS, UDP

Abstract

The objective of this project is to design and implement a system capable of transmitting data at high speeds from JARS 2.0 to a remote PC through Gigabit Ethernet using a SoC. The system has two main parts: data acquisition from the LVDS interface and transmission of the data to the PC through a communication protocol. To acquire the data from the LVDS interface, it was necessary to design a system in the FPGA that multiplexes and copies the data to a memory shared with the processor. Then, a software was designed to read the data from the shared memory and send it to the PC through the UDP protocol.

Keywords: JARS 2.0, Gigabit Ethernet, SoC, LVDS, UDP

1. Introducción

El sistema JARS adquiere los datos del radar principal y los transmite por medio de una interfaz LVDS. Para que los datos sean recolectados y posteriormente procesados por una computadora, es necesario algún tipo de sistema que adquiera los datos de la interfaz LVDS y los envíe por un protocolo de comunicación estándar. En la versión 2.0 de JARS se usa un FPGA Spartan 6 en el cual se ha implementado la pila del protocolo UDP para poder enviar los datos por Ethernet. Sin embargo, este sistema tiene las siguientes desventajas: la tarjeta de evaluación y el software que se usa para programarla (ISE Design Suite) están descontinuados, el conector LVDS es propietario, el ancho de banda de la comunicación está limitado por el FPGA y el protocolo está limitado únicamente a UDP.

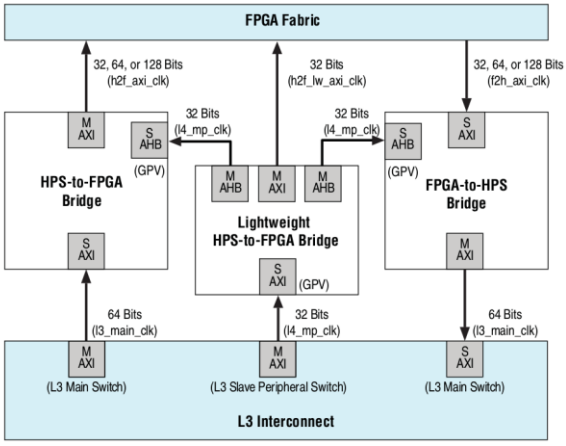
Actualmente existen mejores alternativas como los SoC (System-on-Chip), los cuales poseen un FPGA y un procesador. La principal ventaja de un SoC para la implementación de una interfaz LVDS-Ethernet es que permite el uso de sistemas operativos Linux que manejan toda la carga de la red. Es decir, no es necesario implementar la pila de los protocolos TCP/IP y UDP en el FPGA. Otras ventajas de los SoC son su bajo costo (alrededor de $100), velocidad elevada (600 MHz), conectores comunes con opción a LVDS y tamaño compacto.

Envío de Datos de JARS 2.0

El sistema JARS envía los datos adquiridos del radar principal como words de 32 bits. Cuando se usan los 8 canales de la antena, JARS tiene la capacidad de enviar datos a una frecuencia de 1 MHz, lo cual representa una velocidad de 256 Mbps.

DE0-Nano-SoC

El DE0-Nano-SoC es una tarjeta de desarrollo de Terasic que contiene un SoC Cyclone V de Intel (antiguamente Altera). El Cyclone V está compuesto por un FPGA y un procesador ARM A9 de dos núcleos. La comunicación entre el FPGA y el procesador se



da a través de tres buses AXI, uno de control (32 bits) y dos de datos (hasta 128 bits).

Figura 1. Sistema de comunicación HPS-FPGA

DMA

Un DMA (Direct Memory Access) es un circuito que permite a los periféricos acceder a la memoria principal del sistema independientemente del CPU. Sin un DMA, el CPU está totalmente ocupado durante las operaciones de lectura/escritura y por lo tanto, no puede realizar otras operaciones en paralelo.

2. Desarrollo

Para desarrollar el sistema del SoC, lo primero que se establece es el flujo de datos del FPGA al procesador. La figura 2 muestra el sistema propuesto, donde las flechas delgadas son señales de control y las gruesas el flujo de datos.

Con el flujo de datos establecido, se puede separar el diseño del sistema en tres partes: creación de bloques del FPGA escritos en VHDL y Verilog, instanciación de IP cores y conexiones entre el FPGA y el procesador con la herramienta

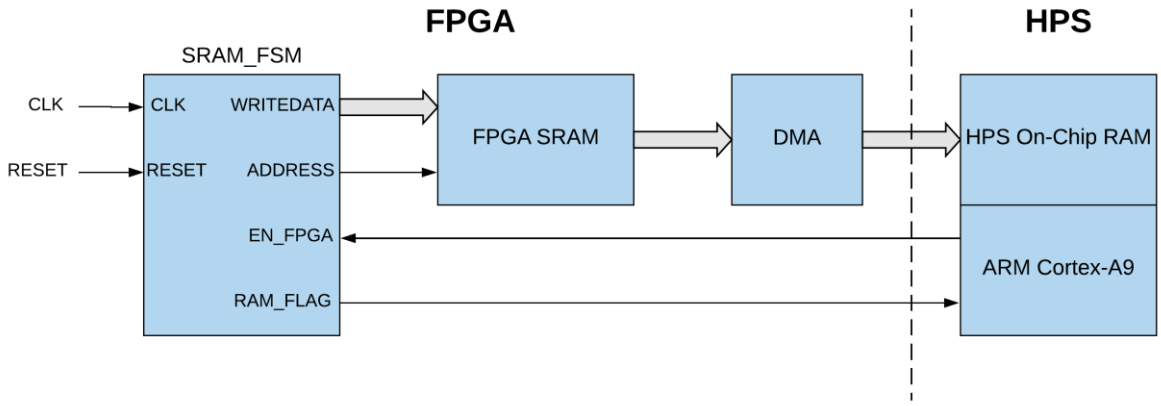


Figura 2. Flujo de datos del sistema

Platform Designer y la creación del software en lenguaje C.

2.1. FPGA

En el FPGA se creó una máquina de estados llamada SRAM_FSM cuya función es generar una secuencia de datos y direcciones de memoria para la SRAM a una frecuencia de 8 MHz.

Cabe resaltar que, en el sistema real, los datos del JARS llegan en paquetes de 8 bytes y deben ser demultiplexados antes de ser guardados en memoria. Sin embargo, para hacer pruebas resulta más fácil usar un generador de números conectado directamente a la memoria SRAM.

2.2. SoC

En la parte del SoC, se instanciaron los siguientes IP cores de Intel.

2.2.1. PLL

Se usó un PLL que genera 2 señales de reloj a partir de una referencia de 50 MHz. El primer reloj generado es de 100 MHz y está conectado a los buses HPS-FPGA, la SRAM y a los PIO. El segundo reloj es de 400 MHz y se usa exclusivamente para que el bloque "NUM_GENERATOR" pueda generar números a una velocidad de 8 MHz.

2.2.2. HPS

El bloque HPS (Hard Processor System) se refiere al procesador ARM Cortex-A9. A pesar de la gran cantidad de entradas y salidas que tiene, el procesador solo utiliza dos de ellas. El bus "h2f_axi_slave" está conectado al puerto "write_master"

del DMA y sirve para adquirir los datos del FPGA. Por otro lado, el bus "f2h_axi_master" está conectado al puerto de control del DMA y a los PIO "en_fpga" y "sram_flag", las cuales sirven como señales de control entre el FPGA y el procesador.

2.2.3. DMA

El bloque DMA se usa usa para copiar los datos de la memoria SRAM en el FPGA a la memoria On-Chip RAM del procesador. Este bloque lee los datos de la SRAM por el puerto "read_master" y escribe en la On-Chip RAM por el puerto "write_master". Además, tiene un puerto controlado por software que se usa para indicar las posiciones de memoria que se van a transferir del FPGA al procesador.

2.2.4. SRAM

El bloque SRAM es una memoria de 64 KB con words de 32 bits instanciada en el FPGA. Esta memoria tiene dos puertos esclavos: uno para lectura y otro para escritura. Como se mencionó anteriormente, el puerto de lectura está conectado al DMA mientras que el puerto de escritura es exportado al FPGA y controlado por el bloque "SRAM_FSM ".

2.2.5. PIO: Habilitador de FPGA y Bandera de SRAM

El bloque "en_fpga" funciona como una señal de control de 1 bit que se usa para habilitar los bloques del FPGA desde el programa principal. Su puerto "s1" está

conectada al puerto "h2f_axi_master" del procesador.

Por otro lado, el bloque "RAM_flag" funciona como una señal de control de 6 bits que se usa para indicarle al procesador qué sección de la memoria SRAM está siendo escrita. Su puerto "s1" también está conectada al puerto "h2f_axi_master" del procesador.

2.3. Procesador ARM Cortex-A9

Para recolectar los datos de la SRAM y enviarlos por Ethernet se escribió un programa en C cuyo diagrama de flujo se muestra en la figura 3.

1.1.1. Adquisición de Direcciones de Memoria Virtual

Lo primero que se debe hacer en el programa es definir las macros con las direcciones de memoria a la cual están mapeados los componentes del FPGA (tabla 1). Debido a que Linux no permite al usuario acceder directamente a la memoria física, también se deben definir punteros que almacenan las direcciones virtuales de memoria.

Luego, dentro del función principal se abre el archivo /dev/mem para obtener un descriptor de archivo y se adquiere la dirección de memoria virtual del bus LW HPS-to-FPGA Bridge con la función mmap(). Los punteros que apuntan a la dirección de memoria virtual del puerto de control del DMA y los componentes "sram_flag" y "en_fpga" se definen como la dirección del bus sumados con un desfase. En este caso, el desfase de memoria de los componentes se muestra en la tabla 1.

Componente	Nombre del puerto	Dirección base	Desfase
F2H AXI Slave	hps_0.f2h_axi_slave	0xFFFF0000	0x00000000
DMA Control Port	dma.control_port_slave	0xC0000000	0x00000000
SRAM	SRAM_0.s1	0xC0000000	0x00000000
Habilitador de FPGA	en_fpga.s1	0xC0000000	0x00000020
Bandera SRAM	sram_flag.s1	0xC0000000	0x00000030

Tabla 1. Direcciones de memoria virtual usadas en el programa

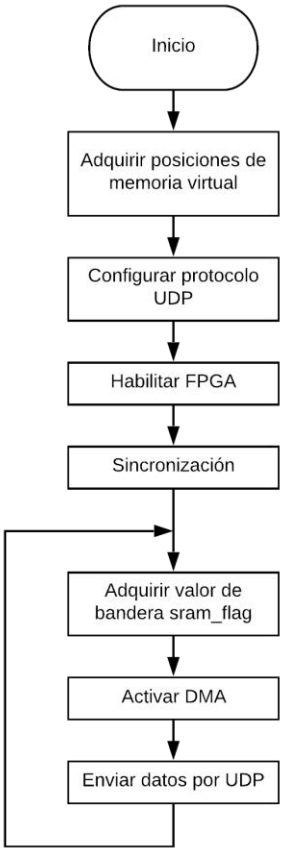


Figura 3. Diagrama de flujo del programa del SoC

1.1.2. Configuración de UDP

Para hacer las configuraciones de UDP lo primero que se necesita es abrir un socket para transmitir datos (Tx) con la función socket() y especificar que se va a usar IPv4 y el protocolo UDP. Además, se crea una estructura "talker" con la dirección IP y el puerto al cual se envían los datos.

1.1.3. Sincronización

Antes del lazo principal, se utiliza la señal de control "en_fpga" para habilitar el FPGA de modo que el primer dato que llega se encuentre en la posición de memoria uno. Luego, el programa espera a que suceda un cambio en la bandera "sram_flag" para que el DMA empiece a transferir los datos hacia la On-Chip RAM. Este proceso de sincronización asegura que en todo momento se ejecuten los siguientes tres procesos en paralelo:

- El procesador envía por UDP los datos en la sección "n" de la memoria On-Chip RAM.

- El DMA copia la sección "n+1" de la memoria SRAM a la memoria On-Chip RAM.
- La máquina de estados guarda los datos que vienen del JARS en la sección "n+2" de la memoria SRAM.

1.1.4. Lazo principal

En el lazo principal, lo primero que se hace leer nuevamente la bandera de la SRAM y empezar a transferir la sección de memoria correspondiente con el DMA. A la misma vez que se transfieren los datos con el DMA, el programa determina las direcciones de memoria necesarias de la sección "n" y utiliza la función sendto() para enviar los datos de dicha sección por UDP. La función sendto() tiene como argumentos el descriptor de archivos del puerto, el puntero que apunta hacia el arreglo la estructura "server" que se creó anteriormente.

1.2. PC de Destino

Para adquirir los datos enviados por UDP desde el SoC, también se creo que un programa en C similar al del servidor UDP. El diagrama de flujo del programa se muestra en la figura 4.

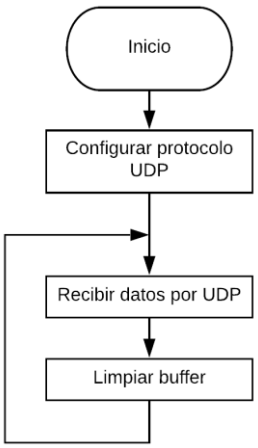


Figura 4. Diagram de flujo del programa de la PC

1.1.1. Configuración de UDP

Para hacer las configuraciones de UDP lo primero que se necesita es abrir un socket para recibir datos (Rx) con la función socket() y especificar que se va a usar IPv4 y el protocolo UDP. Además, se crea una estructura "listener" con la dirección IP y el puerto por el cual se reciben los datos y un buffer de 1024 bytes para almacenarlos.

1.1.2. Lazo Principal

En el lazo principal del cliente UDP lo único que se hace recibir paquetes con la función receivefrom(), la cual tiene como argumentos el descriptor de archivo generado por la función socket(), el buffer para almacenar los datos y la estructura "listener" creada anteriormente. Una vez que recibe el paquete, se limpia el buffer y se repite el proceso.

2. Resultados

2.1. Validación del Hardware

Para verificar el funcionamiento de la máquina de estados creada en el FPGA, se hizo un testbench con la herramienta Modelsim. Los resultados del testbench se muestran en la figura 5.

Se puede observar que, con una señal de reloj de 350 MHz, se genera un dato y una dirección de memoria cada 125 ns (8 MHz). Esto demuestra que la máquina de estados conectada a la memoria SRAM funciona correctamente.

2.2. Validación del Software

1.1.1. Validación del Contenido de los Paquetes

En la primera prueba que se hizo para validar el software, se verificó que el contenido de los paquetes recibidos por la PC sean iguales a aquellos enviados por el SoC. Para ello, se hizo algunas modificaciones al código del SoC y de la PC.

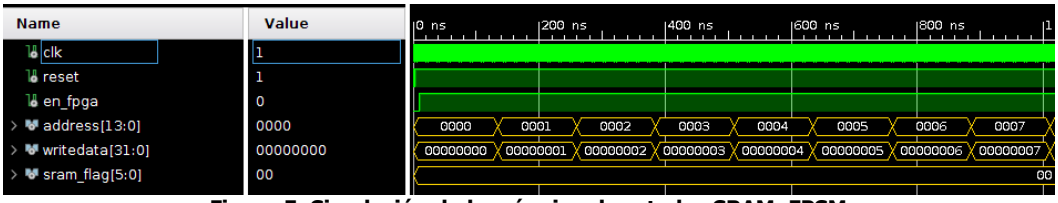


Figura 5. Simulación de la máquina de estados SRAM_FPSM

En el SoC, se creó un vector para almacenar el contenido de 10 paquetes consecutivos, lo cual equivale al contenido total de la memoria. Una vez que el lazo principal se ejecuta 10 veces, el programa imprime el contenido del vector junto con el estado de la bandera de la SRAM en la terminal.

A partir de esta prueba se determinó que el contenido de todos los paquetes en la PC de destino es idéntico a aquellos mandados por el SoC.

1.1.2. Validación del Porcentaje de Paquetes Recibidos

El objetivo de la segunda prueba fue verificar el porcentaje de paquetes recibidos por la PC. Para ello, se hizo 10 pruebas que consistían en que el SoC envíe 100 paquetes y, con un contador, verificar cuántos paquetes llegan a la PC de destino.

Con esta prueba se determinó que el porcentaje de paquetes recibidos por la PC es de aproximadamente 96.5%.

2. Discusión

Para lograr que el sistema de envío de datos funcione correctamente, es necesario tener en cuenta la frecuencia de reloj conectada a los buses de datos y componentes del sistema, especialmente del DMA. Una frecuencia muy baja (<50 MHz) hace que la transferencia de datos entre el FPGA y el procesador sea lenta, lo cual a su vez hace que se pierdan paquetes, mientras que una frecuencia muy alta (>150 MHz) hace que el procesador del SoC se cuelgue.

En cuanto al porcentaje de paquetes recibidos, se pudo notar que no hubo variación entre las frecuencias intermedias (50 a 100 Mhz). Se cree que esto sucede debido a que el cuello de botella no se encuentra en los buses AXI o en el DMA, sino en el programa del SoC que envía los datos por UDP.

3. Conclusiones

En el desarrollo de este proyecto se logró trabajar con la tarjeta de desarrollo DE0-Nano-SoC para enviar datos generados en el FPGA por Ethernet a una velocidad de 256 Mbps. El sistema que se probó escribe los datos que llegan al SoC en una memoria SRAM, para luego leer y enviar dichos datos por Ethernet.

El uso de la herramienta Platform Designer del programa Quartus Prime resultó ser muy útil para el desarrollo del proyecto. Esta herramienta no solo es necesaria para diseñar los buses que conectan el FPGA con el procesador, sino también para instanciar IP cores de manera sencilla en el FPGA. Sin embargo, para poder crear los buses es necesario entender bien cómo funciona el flujo de datos dentro del SoC. Para ello resultó muy útil leer el manual del Cyclone V, el cual explica de manera detallada cómo funciona el SoC.

4. Recomendaciones

Como trabajo futuro se recomienda buscar la manera de optimizar el código del SoC para poder mejorar el porcentaje de paquetes recibidos. Además, se recomienda realizar un shield para el DE0-Nano-SoC donde se pueda conectar los cables LVDS del JARS y hacer pruebas más exactas.

5. Agradecimientos

Agradezco al Radio Observatorio de Jicamarca y al Instituto Geofísico del Perú por haberme brindado la facilidad de usar los equipos necesarios para este proyecto, el cual me ayudó mucho en mi formación académica y

profesional.

Así mismo, quiero agradecer especialmente a mi asesor Joaquín Verástegui y a John Rojas por su apoyo y orientación a lo largo del proyecto.

6. Bibliografía

(2012). Cyclone V Device Handbook, Volume 3: Hard Processor System Technical Reference Manual. Editado por Altera Corporation, San Jose, California.

(2015). DE0-Nano-SoC User Manual. Editado por Terasic.

Land, B. (2019). DE1-SoC: ARM HPS and FPGA, Addresses and Communication, Cornell ECE 5760. [En línea]. Disponible en https://people.ece.cornell.edu/land/cources/ece5760/DE1_SOC/HPS_peripherials/FPGA_addr_index.html