Trabajo Práctico Final

- Objetivo

Implementar en FPGA un ecualizador adaptivo utilizando los conocimientos adquiridos en el curso Diseño Digital Avanzado.

- Descripción

El trabajo final práctico consiste en implementar un ecualizador adaptivo en la FPGA utilizando el entorno de desarrollo planteado en las clases de laboratorio. En el archivo zip (**DDAfinal.zip**) encontraremos los elementos necesario para el desarrollar el práctico, los cuales son

- **cpp:** Software para el micro-procesador.
- **python:** Script para comunicar por el puerto UART la PC con la FPGA.
- **rtl:** Entorno de verificación.
- **sim:** Simulador en python que se utiliza para entender el comportamiento del filtro adaptivo.

A continuación se describe en detalle el entorno de trabajo.

- Ecualizador Lineal Adaptivo

Uno de los principales efectos que degrada la performance de un sistema de comunicaciones digitales es la ISI (Inter-Symbol Interference), la cual afecta todos los sistemas de modulación de pulso. Este fenómeno se origina por la distorsión de amplitud y fase que experimenta la señal transmitida durante su propagación por el canal de comunicaciones. Para hacer frente a las imperfecciones del canal, se han propuesto diferente técnicas de ecualización. El método de ecualización más atractivo por su baja complejidad en la implementación es el ecualizador lineal (Lineal Equalizer - LE), el cual consiste en un filtro que aplana la respuesta del canal pero su desempeño decrece frente a alto ruido. El ecualizador lineal consiste en un filtro transversal directo cuya salida del ecualizador se puede representar por

$$y[n] = \sum_{i=0}^{L-1} x[n-i]c[i] \tag{1}$$

donde L es el número de coeficientes del filtro, $c[i]$ los coeficientes y $x[n]$ las muestras de entrada. En la mayoría de los casos, el canal no se conoce, por lo tanto en necesario aplicar alguna técnica de adaptación de coeficientes del filtro. Una de las técnicas mas utilizadas en la industria por su simpleza en la implementación es el LMS (Least Mean Square), el cual minimiza el error cuadrático medio, cuya representación es

$$c[n+1]^{(i)} = c[n]^{(i)} + \mu.e[n].x[n-i]; \quad i = 0, \ldots, L-1 \tag{2}$$

donde μ es el paso de adaptación y $e[n]$ el error entre la salida del filtro ($y[n]$) y la salida del slicer ($\hat{y}[n]$), el cual toma valores ± 1. Por lo tanto, el error queda definido como

$$e[n] = y[n] - \hat{y}[n] \tag{3}$$

- Implementación

El sistema que se implementa consiste en un bloque de control (Pc) y un bloque de procesamiento de señales (FPGA) (Fig. 1).

- **PC:** Envía comandos a la FPGA utilizando el puerto UART, a través de la ejecución de script en python. Los scripts se encuentran en la carpeta **Python**.
- **FPGA:** Es la plataforma de desarrollo que realiza el procesamiento de las señales.

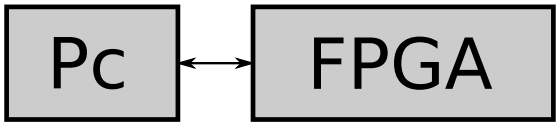

Figura 1: Esquemático Top level

La Fig. 2 muestra el esquemático general de los módulos que se implementan en la FPGA. Los módulos grises son entregados por la cátedra, los cuales se describen a continuación

- **uP:** Microprocesador MicroBlaze que ejecuta las tareas ordenadas desde la PC. La comunicación hacia los módulos dentro de la FPGA se realiza por medio de los puertos de entrada y salida de propósito general (GPIO). La frecuencia de trabajo es $100MHz$.
- **RegisterFile (RF):** Interference entre el **uP** y los módulos de procesamiento de señales.
- **PRBS:** Generador de secuencias binarias (Pseudorandom binary sequence), en donde símbolo "0" representa $+1$ y el "1" representa el -1.
- **Canal (Ch):** Es un filtro FIR que se configura con cuatro (4) tipos de canales seleccionados a través del puerto **i_switch[1:0]** (Fig. 3).
- **Step:** Determina el paso de adaptación del algoritmo LMS, teniendo la posibilidad de elegir entre cuatro (4) valores a través del puerto **i_switch[3:2]**.
- **DSP:** Bloque a desarrollar.

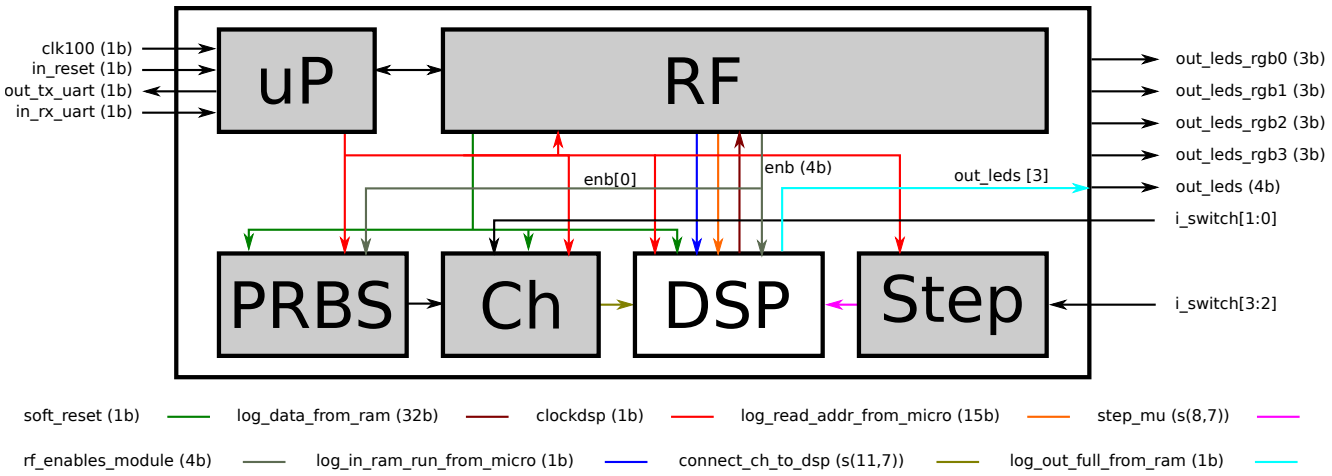

Figura 2: Diagrama en bloques del sistema implementado en la FPGA.

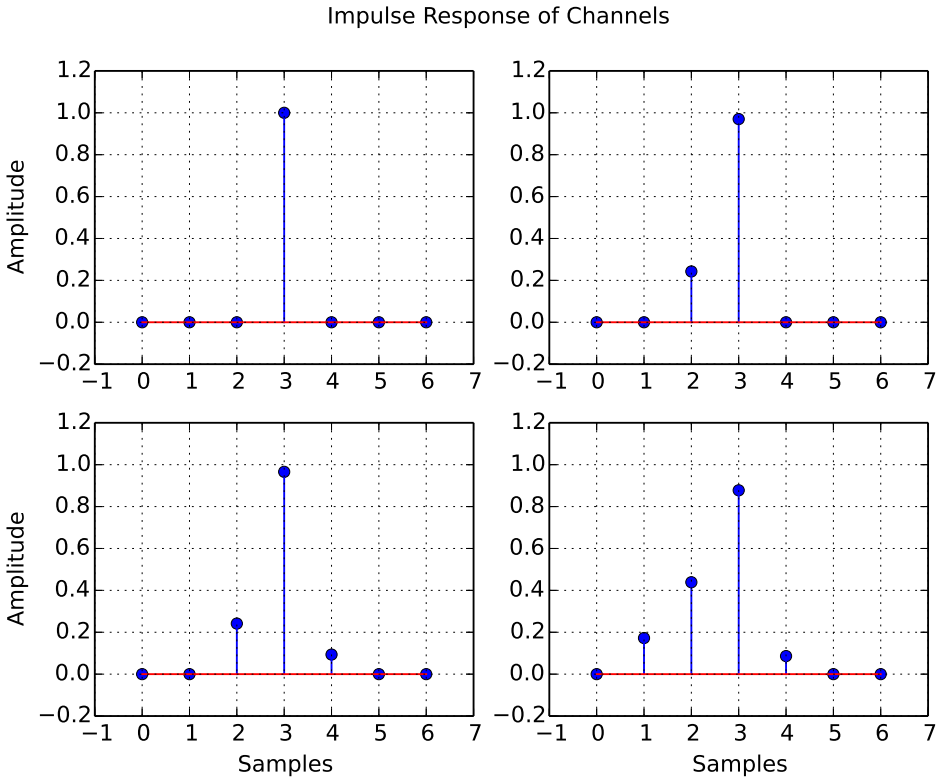

Figura 3: Canales disponibles en la FPGA.

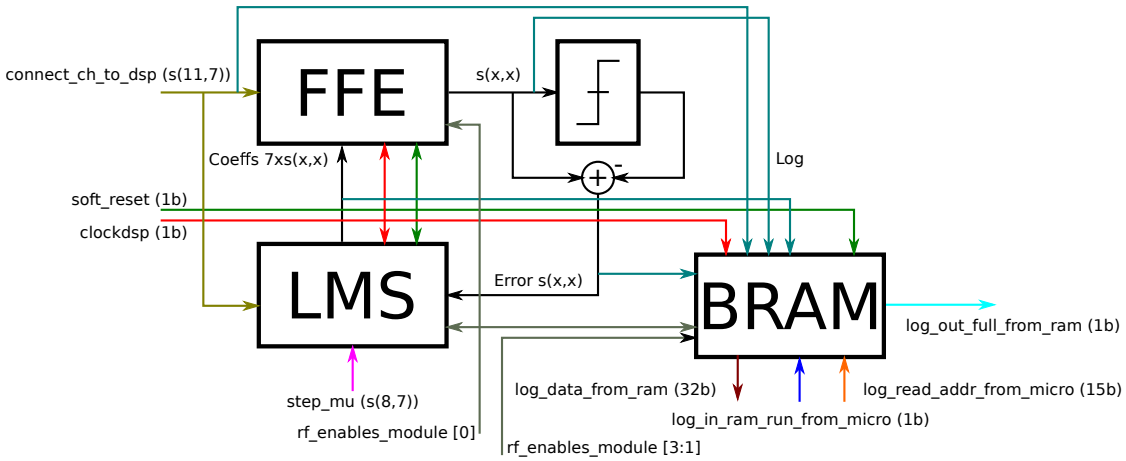

Figura 4: Diagrama en bloques del DSP.

El DSP consta de los siguientes módulos

- **FFE:** Es el ecualizador es un filtro FIR directo de 7 coeficientes que filtra las muestras del canal, cuyo comportamiento se representa en la Ec. (1). Las muestras se procesan en forma continua y los coeficientes se inicializa con el vector $[0, 0, 0, 1, 0, 0, 0]$ cuando la señal de *reset* esta en alto. El procesamiento se inicia cuando la señal de *enable* esta en alto, cosa contrario todo el procesamiento se detiene sin perder los últimos valores de los registros a menos que el *reset* se active nuevamente. Las operaciones aritméticas se realizan en máxima resolución y se aplica el criterio de *Saturación* y *Truncado* a la señal de salida. La resolución de los bits de salida se deben definir en base a los resultados obtenidos en el simulador (sinFfe.py).

- **Slicer:** Retorna el valor 0bit (+1) o 1bit (-1) si la salida del FFE es mayor o menor a cero, respectivamente.

- **Error:** Se estima el error tomando como referencia la diferencia de la salida del FFE y la salida del Slicer (Ec. (3)). El resultado se utiliza para adaptar los coeficientes del filtro. La resolución de los bits del error se deben definir en base a los resultados obtenidos en el simulador (sinFfe.py).

- **LMS:** Algoritmo de adaptación de los coeficientes que utiliza las muestras de entrada al FFE, el paso de adaptación, el error y el valor anterior de cada coeficientes en máxima resolución para realizar la adaptación de los coeficientes. El comportamiento del algoritmo se describe en la Ec. (2). El módulo retorna los coeficientes adaptados que se utilizarán en el filtro pero con una resolución menor a la utilizada en la adaptación, por lo tanto se deberá aplicar el criterio de *Saturación* y *Truncado* en el bus **Coeffs** (utilizar el simulador sinFfe.py para determinar la resolución óptima).

- **BRAM:** Memoria de logueo de $32kx32b$ que utiliza tres bits para seleccionar la fuente a loguear. Las señales que se loguean son la entrada y salida del FFE, el error y los coeficientes. En el caso de los se necesita aplicar algún criterio de ordenamiento de los mismos para guardarlos en la memoria.

Los puertos y conexiones internas de los módulos mostrados en la Fig. 2 y Fig. 4 se detallan a continuación

- **Puertos**
  - **clk100 (1b):** Reloj de referencia de 100MHz para el **uP**.
  - **in_reset (1b):** Hard reset en la placa de desarrollo.
  - **out_tx_uart (1b):** Puerto de salida UART.
  - **in_rx_uart (1b):** Puerto de entrada UART.
  - **out_leds_rgb0 (3b):** Led RGB.
  - **out_leds_rgb1 (3b):** Led RGB.
  - **out_leds_rgb2 (3b):** Led RGB.
  - **out_leds_rgb3 (3b):** Led RGB.
  - **out_leds (4b):** Leds.
  - **i_switch (4b):** Switchs.
- **Conexiones**
  - **connect_ch_to_dsp (s(11,7)):** Muestras de entrada al bloque ecualizador con resolución $s(11, 7)$.
  - **Coeffs:** Bus de siete (7) coeficientes adaptados de resolución a definir por el diseñador.
  - **Error:** Error estimado entre la salida del ecualizador y la salida del Slicer de resolución a definir por el diseñador.
  - **soft_reset (1b):** Señal de reset comandada desde el usuario.
  - **clockdsp (1b):** Señal de reloj.
  - **step_mu (s(8,7)):** Paso de adaptación del algoritmo LMS de resolución $s(8, 7)$.
  - **rf_enables_module (4b):** Señal de habilitación y control de logueo.
    - **rf_enables_module [0]**: Enable
    - **rf_enables_module [3:1]**: Selecciona que señal se desea loguear en la memoria BRAM.
  - **log_data_from_ram (32b):** Señales logueadas en la memoria BRAM.
  - **log_in_ram_run_from_micro (1b):** Inicio de logueo.
  - **log_read_addr_from_micro (15b):** Dirección de lectura de la memoria BRAM.
  - **log_out_full_from_ram (1b):** Se activa cuando la memoria BRAM esta llena.

- Archivos a entregar

Se deberá entregar un PDF con los esquemáticos de los módulos diseñados y una breve descripción de los mismos. Además, se deben adjuntas los archivos verilog utilizados en el proyecto.