Diseño y Verificación de un Interpolador Lineal

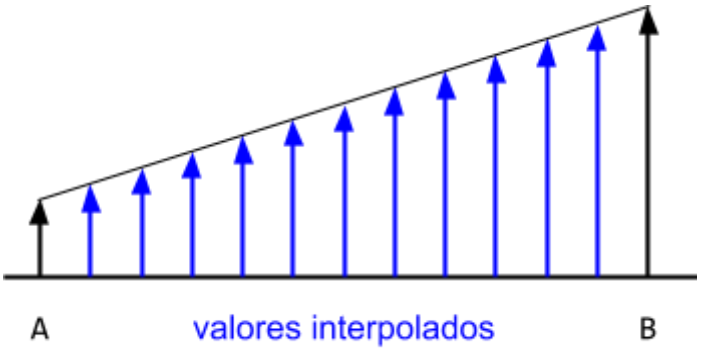
Introducción al diseño y verificación de sistemas de comunicaciones digitales complejos.

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hguzman@us.es

Interpolador lineal

Uno de los submódulos del estimador de canal que tendremos que realizar como trabajo de la asignatura es un Interpolador Lineal. Un interpolador lineal es simplemente una entidad que realiza una interpolación de primer orden entre dos muestras.

En nuestro caso queremos generar por interpolación 11 valores entre dos entradas A y B. Estas entradas A y B corresponden al valor conocido del canal en dos pilotos contiguos (pilotos inferior y superior), y los once valores intermedios corresponden al canal estimado en las 11 frecuencias utilizadas entre ambos pilotos:



Hay que tener en cuenta que, ya que el canal es complejo, tendremos que realizar interpolaciones tanto de la parte real como de la imaginaria.

Debido a que no sería eficiente tener 11 puertos de salida, el Interpolador Lineal sacará las 11 salidas interpoladas de manera secuencial por el mismo puerto.

Se implementará el diseño para la FPGA XC6SLX9-2CSG324C, que es la que está disponible en la tarjeta LX9 Microboard.

Planteamiento de la entidad

Se propone aquí una descripción de entidad para el interpolador lineal:

Puerto	Dirección	Tipo de Dato	Descripción
`clk`	`in`	`std_logic`	Señal de reloj. Activa por flanco de subida.
`rst`	`in`	`std_logic`	Reset global asíncrono. Activo a nivel alto.
`inf`	`in`	`complex10`	Piloto inferior.
`sup`	`in`	`complex10`	Piloto superior.
`valid`	`in`	`std_logic`	Señal que indica que las entradas `inf` y `sup` son válidas.
`estim`	`out`	`complex10`	Valor interpolado entre `sup` e `inf`.
`estim_valid`	`out`	`std_logic`	Señal que indica que la salida `estim` es válida.

Se recomienda definir el tipo de dato `complex10` en un `package` de definiciones comunes al trabajo de la asignatura:

```
type complex10 is record
    re : signed (9 downto 0);
    im : std_logic_vector (9 downto 0);
end record;
```

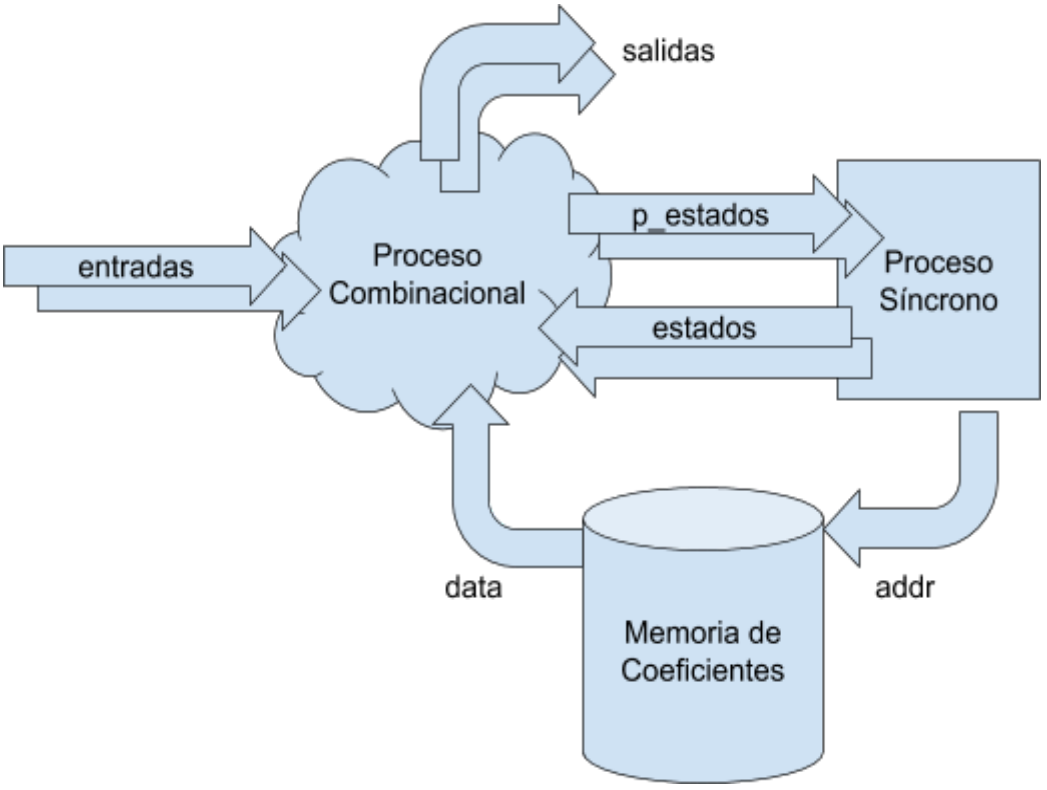
No es obligatorio seguir las recomendaciones de esta sección, pero en caso de no seguirlas es importante plantearse bien la entidad antes de continuar, ya que la definición de la entidad nos puede condicionar la arquitectura de la misma.

Caso de querer realizar algún cambio en la entidad, puede ser interesante considerar las siguientes ampliaciones, que incrementan la dificultad del diseño pero lo harían reutilizable en otros proyectos:

- Anchura de los datos de entrada y salida parametrizable a través de un `Generic` (pero no podremos utilizar `record`, ya que éstos no son parametrizables)
- Número de pasos en la interpolación parametrizable a través de un `Generic`

Diseño de la arquitectura

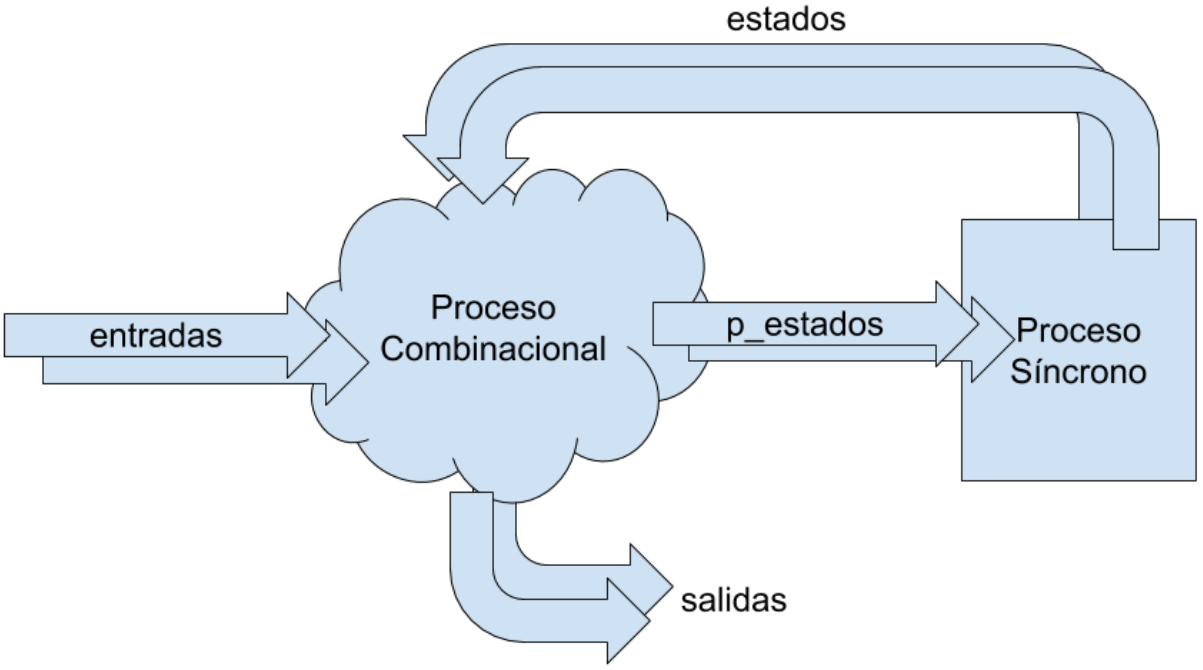
La arquitectura más genérica sería aquella en la que se utilizaran unos coeficientes de interpolación almacenados en una memoria ROM, que indicarían cómo se debe realizar dicha interpolación:



Si bien esta arquitectura es interesante, ya que permite la reconfiguración del filtrado de interpolación simplemente reescribiendo valores nuevos en la memoria, en el caso de realizar una interpolación lineal resulta innecesariamente complicada.

Es por ello por lo que se propone una arquitectura simplificada, sin memoria de coeficientes, en la que se implemente la ecuación del filtro utilizando valores constantes:

```
estim(i) = coef_inf(i) * inf + coef_sup(i) * sup
```



Se deja como ejercicio el determinar las señales necesarias (y su tipo) para almacenar el estado interno necesario para la entidad. Se recomienda realizar una versión del diagrama anterior en el que aparezcan explícitamente todas las señales necesarias, ya que esto ayudará a la realización de la implementación.

Hay que tener en cuenta que tras la multiplicación, se deben truncar o redondear los datos para poder tener una salida de 10 bits.

Testbench básico

Es necesario realizar un testbench básico que haga una comprobación básica de la funcionalidad del interpolador, por ejemplo, indicando al interpolador que interpole entre (0,0) y (500,500), comprobando en el visor de forma de onda que los valores interpolados coincidan con los esperados.

Este testbench nos servirá de base sobre la que trabajar en la siguiente parte de la práctica, en la que incorporaremos capacidades avanzadas a dicho testbench.

Asimismo se debe realizar la síntesis del interpolador y comprobar que no tiene ni listas de sensibilidad incompletas ni latches, ya que estos pueden causar un “simulation mismatch”, es decir, que la simulación no refleje correctamente el funcionamiento del circuito. Cuando se realizan cambios en un circuito, es muy recomendable comprobar esto antes de trabajar sobre la simulación.

Verificación usando los paquetes UVVM y OSVVM

En esta segunda parte de la práctica añadiremos funcionalidades avanzadas al testbench del interpolador, como son:

- Ejecución por línea de comandos del test
- Chequeo de valores dentro del testbench
- Generación de estímulos aleatorios
- Logging y chequeo de errores
- Drivers y monitores de protocolo
- Functional coverage

UVVM son las siglas de Universal VHDL Verification Methodology, mientras que OSVVM son las siglas de Open Source VHDL Verification Methodology. Ambos son libraries en VHDL (conjuntos de paquetes) que implementan funcionalidad interesante para verificación:

UVVM
- Logging y chequeo de valores, con impresión por pantalla de los valores recibidos y esperados, para diferentes tipos de datos
- Generación de reloj configurable con señal de habilitación (muy útil para parar automáticamente la simulación)

OSVVM
- Generación de estímulos aleatorios
- Functional coverage e “intelligent coverage” (generación/restricción inteligente de estímulos aleatorios, orientados a cubrir huecos en el functional coverage)

El objetivo de la segunda parte de esta práctica es convertir un testbench tradicional, de los de “proceso de reloj + proceso de estímulos + comprobación manual de formas de onda”, en un testbench automatizado en el que se generen estímulos aleatorios y se comprueben automáticamente los resultados.

Descarga del software en los PCs del Centro de Cálculo

Durante el curso 2018/2019, para poder utilizar el software en los PCs del CdC, tenemos que seguir los siguientes pasos:

Instalar GHDL, yosys y otras herramientas:

Ir al directorio donde instalaremos las herramientas:

- `cd /opt/salas`

Descargar los binarios:

- `wget http://woden.us.es/docs/docencia/EDC1MIT/fosshdl-03042018.zip`

Descomprimir (el password es “edc_1MIT2018” - sin las comillas):

- `unzip fosshdl-03042018.zip`

Descargar lcov para poder generar informes de code coverage:

Como no tenemos permisos de super usuario en los PCs del CdC, simplemente descargaremos el paquete de los repositorios de ubuntu, pero en lugar de instalarlo, lo descomprimiremos y posteriormente lo añadiremos al PATH.

Si estamos instalando esto en un PC del que tengamos permisos de superusuario, podemos simplemente instalar lcov con el gestor de paquetes (apt o yum).

Ir al directorio donde instalaremos la herramienta:

- `cd /opt/salas`

Crear una carpeta para lcov y entrar en dicha carpeta:

- `mkdir lcov`
- `cd lcov`

Descargar el paquete, sin instalarlo:

- `apt-get download lcov`

Descomprimir el .deb descargado:

- `ar vx lcov_1.13-3_all.deb`

Descomprimir el data.tar.gz que viene dentro del .deb y que contiene los binarios:

- `tar xf data.tar.xz`

Añadir los binarios descargados al PATH

En cada consola desde la que queramos trabajar con estas herramientas, tendremos que añadir las rutas en las que hemos descargado los binarios a la variable de entorno PATH:

- `export PATH=/opt/salas/fosshdl/bin:/opt/salas/lcov/usr/bin:$PATH`

También podemos añadir esta línea a nuestro fichero ~/.bashrc para que se modifique el PATH automáticamente cada vez que abramos una nueva consola

Configuración de la máquina virtual

Para trabajar desde casa, podemos utilizar la máquina virtual.

Descargar la máquina virtual. Se proporcionan dos opciones. Ambas alternativas incluyen el software necesario para realizar la práctica:

- http://woden.us.es/docs/docencia/EDC1MIT/cdc.ova (7.7GB). (md5sum: 33ae522c8f9ae090dca54f1cfb0f28b9). Esta imagen es una copia de la instalada en los ordenadores del Centro de Cálculo, añadiéndole el software necesario para la práctica.

- http://woden.us.es/docs/docencia/EDC1MIT/cdc-lubuntu.ova (2.8 GB). (md5sum: b1a8ee337fd9089a7866e5d7548961c6). Esta alternativa utiliza Lubuntu, que ocupa menos espacio, pero tiene un entorno de escritorio un poco diferente.

Instalar Virtualbox si no lo tenemos instalado:

- En sistemas Debian/Ubuntu: como root, ejecutar: `apt install virtualbox`
- En sistemas Centos/Redhat: como root, ejecutar: `yum install virtualbox`
- En sistemas windows, descargar los binarios de: https://www.virtualbox.org/wiki/Downloads

Abrir Virtualbox y seguir los siguientes pasos:

- File -> Import Appliance
- Seleccionar el fichero `.ova` anteriormente descargado
- Seleccionar la configuración de los recursos hardware (RAM, CPUs) dedicados a la máquina virtual. En caso de duda, mantener los valores por defecto propuestos por el programa.
- Esperar a que se realice la importación. Cuando termine, podemos arrancar la máquina seleccionándola y haciendo click en el botón “Start”

Si estás usando la máquina virtual, el software a utilizar (el simulador GHDL y el sintetizador Yosys) ya está configurado. La contraseña del usuario “salas” es “salas” y tiene habilitado “sudo” por si necesitas instalar algún paquete.

Funcionalidades a añadir al testbench

Partiendo del interpolador y su testbench básico, queremos añadir las siguientes funcionalidades:

Interpolador:

- Firewall assertions en el interpolador:
  - Dar un error/failure si la entrada `valid` se activa mientras el interpolador está ocupado
  - Dar un error/failure si los datos de entrada (`sup`/`inf`) cambian mientras el interpolador está ocupado

Testbench del interpolador:

- Uso del generador de reloj de UVVM:
  - Utilizando el procedure clock_generator en la arquitectura
  - `clock_generator (clk, clk_enable, CLK_PERIOD, "Global clock", CLK_PERIOD/2);`
- Uso del procedure `log()` proporcionada por UVVM
- Driver de protocolo utilizando un procedure

    - Se recomienda definir el procedure dentro del process que genera los estímulos (antes del begin), para que quede una invocación más sencilla del mismo: si se define un procedure dentro de un process, éste puede acceder a las señales movidas por el process sin que éstas sean parte de sus argumentos
- Monitor de protocolo en un process. Se recomienda muestrear las salidas en el flanco de bajada de reloj (`falling_edge(clk)`).
- Cálculo de las salidas esperadas utilizando el tipo de dato `real`. Esto se puede hacer tanto en el process del monitor como utilizando sentencias concurrentes. Se recomienda el uso del tipo de dato `real` de forma que se calcule la salida esperada de manera diferente a como lo hace el interpolador.
- Chequeo automático, en el monitor, de las salidas del interpolador, comparándolas con las salidas esperadas, dejando un margen de 1 LSB para tener en cuenta los redondeos (en Matlab estamos redondeando mientras que en VHDL estamos truncando):
    - `check_value_in_range (señal_a_comprobar, rango_inferior, rango_superior, ERROR, "Mensaje de error");`
- Generación de estímulos aleatorios. La función `RV.RandInt(MIN, MAX)` devuelve un entero aleatorio contenido entre estos dos valores (inclusive)
- Functional coverage
    - Functional coverage para una de las entradas del interpolador
    - Functional coverage para las 4 entradas del interpolador
    - Cross coverage entre las 4 entradas

Se deben añadir poco a poco las distintas funcionalidades al testbench, haciendo "`make test`" frecuentemente para comprobar el correcto funcionamiento del mismo. Se recomienda añadir las funcionalidades en el orden en el que se nombran en este apartado.

Ficheros de ejemplo

Se proporciona un interpolador con su testbench básico en: http://woden.us.es/docs/docencia/EDC1MIT/interpolator_simple/ , aunque cada alumno debe trabajar en su propio diseño.

Se proporciona un interpolador con el testbench avanzado, junto con Makefiles para automatizar la síntesis y simulación en: http://woden.us.es/docs/docencia/EDC1MIT/edc.tar.gz . Se recuerda que el comando para descomprimir este fichero es: `tar xzvf edc.tar.gz` . Los contenidos de este fichero son:

- `edc/README.md` : Instrucciones
- `edc/Makefile` : Makefile para automatizar la descarga y compilación de UVVM y OSVM. En el README.md se explican los 'targets' del Makefile
- `edc/interpolator/Makefile` : Makefile para automatizar la compilación y síntesis del interpolador. El alumno debe modificar este fichero para adaptarse a los

nombres de sus propios ficheros y entidades. En el README.md se explican los 'targets' de este Makefile

- `edc/interpolator/syn.ys` : script para realizar la síntesis con Yosys Open Synthesis Suite
- `edc/interpolator/src/` : el alumno debe poner aquí su código fuente
- `edc/interpolator/src/interpolator.vhd` , `edc/interpolator/src/interpolator_common.vhd` , `edc/interpolator/src/tb_interpolator.vhd` , `edc/interpolator/src/tb_interpolator_coverage.vhd` : se proporcionan estos ficheros como ejemplos de una posible realización de la práctica completa, con comentarios incluidos. El alumno debe hacer su propia implementación, pero puede consultar este código si tiene dudas al respecto de las funcionalidades a añadir.

Se recomienda que el alumno descomprima el fichero en dos carpetas diferentes, de forma que mantenga una con la solución para consultar dudas, y en la otra elimine los ficheros de la carpeta `interpolator/src` y los sustituya por los suyos, y edite `interpolator/Makefile` e `interpolator/syn.ys` para ajustarse a los nombres de ficheros y entidades de su propia implementación