Diseño y Verificación de un Interpolador Lineal

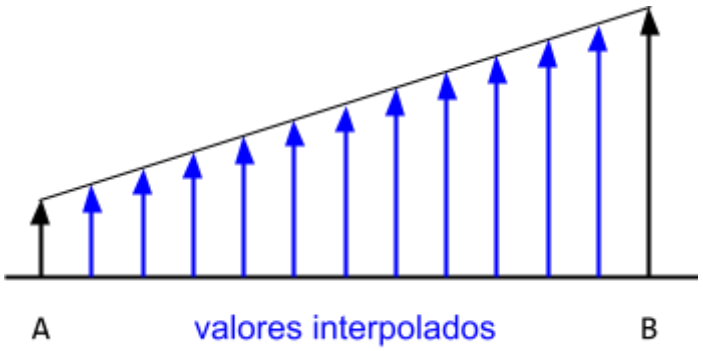
Introducción al diseño y verificación de sistemas de comunicaciones digitales complejos.

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 (es decir, de manera ordenada y en distintos ciclos de reloj) por el mismo puerto.

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 : signed (9 downto 0);
end record;
```

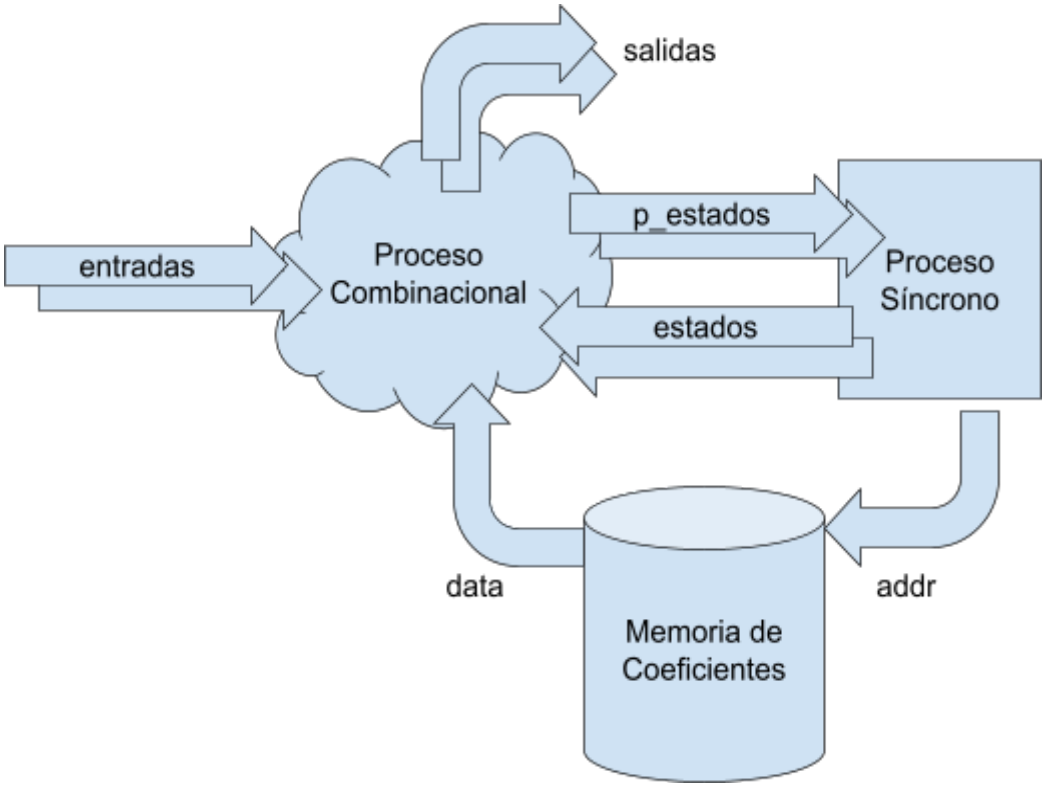
No es obligatorio seguir las recomendaciones de esta sección, pero en caso de no seguirlas es importante plantearse bien la entidad antes de continuar, ya que la definición de la entidad nos puede condicionar la arquitectura de la misma.

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

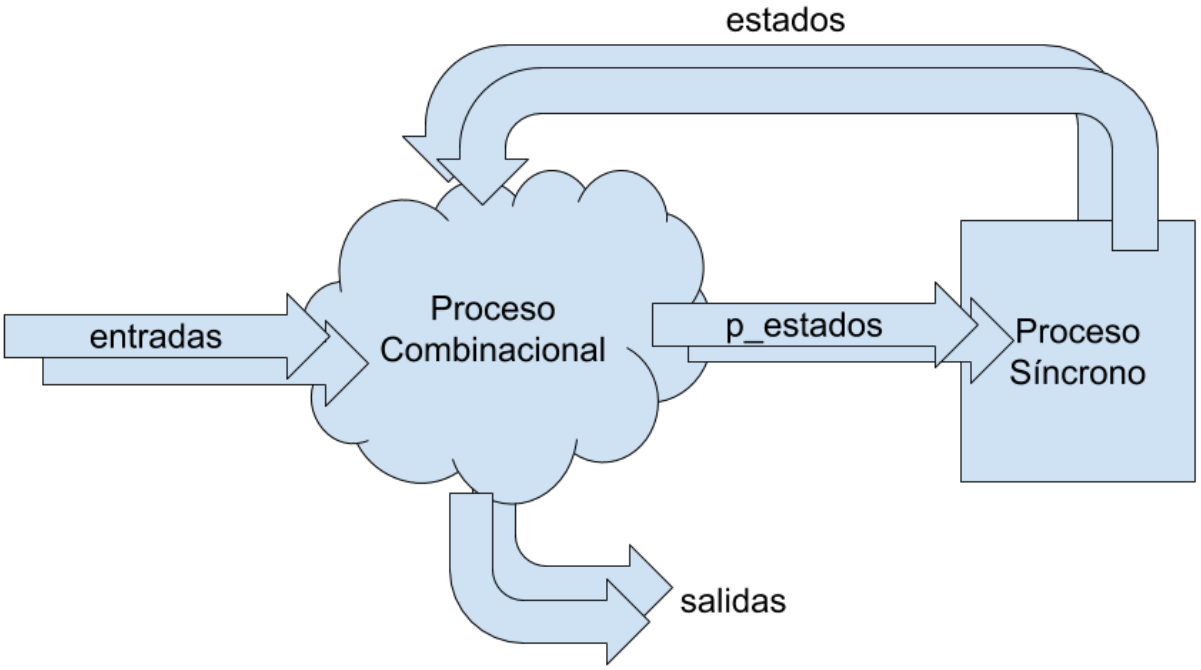
La arquitectura más genérica sería aquella en la que se utilizaran unos coeficientes de interpolación almacenados en una memoria ROM, que indicarían cómo se debe realizar dicha interpolación:



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.

Consideraciones para el diseño del interpolador

El interpolador consistirá de un proceso combinacional y un proceso síncrono, además de las sentencias concurrentes que sean necesarias.

Recordemos que puede demostrarse que, si estimamos 12 veces el valor de canal, el cálculo de la salida puede simplificarse:

```
salida = 12*estim(i) = sup * i + inf * (12-i)
```

Estados internos

Como estado interno que será almacenado por el proceso síncrono, necesitaremos al menos el valor de `i`. Ya que tendremos que multiplicar dicha `i` por los valores tipo `signed` de las partes reales e imaginaria de `sup` e `inf`, necesitaremos que el tipo de dato de `i` sea `signed`, en particular de 5 bits ya que necesitamos poder almacenar en él valores entre 0 y

11. *(Nota: cuando estimamos el canal, realmente `i` va entre 1 y 11, no obstante se recomienda estimar el canal también en la posición del piloto inferior, `i = 0`, ya que tener esta información será muy útil a la hora de depurar, porque nos permite distinguir el fallo tipo "tenemos un valor incorrecto en el piloto" del fallo tipo "el piloto es correcto pero estamos interpolando mal").*

Otro estado interno que necesitaremos almacenar será si estamos interpolando o no. Lo más directo es definir un tipo de dato que tenga dos estados posibles (`reposo, interpola`), pero también es posible utilizar un valor de `i` que no se esté usando (por ejemplo `i=12`) para indicar el estado de reposo.

Descarte de bits

Al realizar la multiplicación de, por ejemplo, `sup * i`, el multiplicador hardware nos devolverá 10 + 5 bits, pero nosotros a la salida del interpolador debemos quedarnos únicamente con 10. Ya que `i` es de tipo `signed` pero no toma valores negativos, esto implica que el bit más significativo de `i` no se estará utilizando realmente, por lo que no debemos descartar directamente los 5 bits menos significativos de la multiplicación, sino comprobar si podemos descartar primero alguno de los más significativos.

A la hora de comprobar que el funcionamiento del interpolador es correcto, se debe tener en cuenta que tenemos un factor de escala de 12/(2^`Ndesc`), el 12 viene de que estamos estimando 12 veces el canal, y el 1/(2^`Ndesc`) viene de eliminar los `Ndesc` bits menos significativos.

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.

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 en el enlace http://heimdall.us.es/docs/docencia/EDC1MIT/edc.tar.gz unos ficheros de ejemplo con los que trabajar. Entre ellos, se proporciona un interpolador con su testbench básico, aunque cada alumno debe trabajar en su propio diseño.

También se proporciona un interpolador con el testbench avanzado, junto con Makefiles para automatizar la síntesis y simulación en el mismo fichero. 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/src/` : carpeta donde el alumno debe poner su código fuente
- Códigos de ejemplo:
  `edc/interpolator/src/interpolator.vhd` ,
  `edc/interpolator/src/edc_common.vhd` ,
  `edc/interpolator/src/tb_interpolator_simple.vhd` ,
  `edc/interpolator/src/tb_interpolator_nocover.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. `tb_interpolator_simple` es el testbench simple, `tb_interpolator_nocover` es el testbench con todas las funcionalidades avanzadas salvo el alcance funcional, y `tb_interpolator_coverage` es el testbench anterior con el alcance funcional añadido. *Nota: ya que el Makefile busca un testbench que se llame tb_interpolator.vhd, se debe renombrar el testbench que se quiera utilizar, ya que si no, no encontrará ningún fichero con el nombre esperado por lo que dará un error.*

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

Configuración de la máquina virtual

Para disponer de la máquina virtual con el software que utilizaremos para la asignatura, debemos seguir los siguientes pasos:

1. Descargar la máquina virtual del siguiente enlace y seguir las instrucciones: http://trajano.us.es/~fjfj/mv/maq_virtual.html

   Esta máquina virtual es una copia de la distribución de linux existente en el Centro de Cálculo. Si ya la tenéis en vuestro ordenador, podéis omitir este primer paso.

   Se debe utilizar el usuario salas, cuyo password es también salas. Este usuario tiene permisos para hacer sudo.

2. Instalar las dependencias del software, ejecutando el siguiente comando en un terminal:

   ```
   sudo apt install gnat zlib1g-dev lcov
   sudo apt install libcanberra-gtk-module gtkwave
   ```

3. Descargar el software para la asignatura:

   ```
   cd ~
   wget https://heimdall.us.es/sw/fosshdl-20210406.tar.gz
   tar xzvf fosshdl-20210406.tar.gz
   ```

   Se puede comprobar que la descarga ha sido correcta utilizando el siguiente comando:
   `md5sum fosshdl-20210406.tar.gz`

   La salida de dicho comando debe incluir este código (si el código es distinto, significa que ha habido un error en la descarga):
   `508ae568d8d6a8bba0214c4e6491610c  fosshdl-20210406.tar.gz`

4. En la consola en la que vayamos a utilizar el software, cargar las variables de entorno:

   ```
   source ~/fosshdl/env.rc
   ```