



UNIVERSIDAD DE BUENOS AIRES  
FACULTAD DE INGENIERÍA

ESPECIALIZACIÓN EN MICROELECTRÓNICA

ARQUITECTURA DE SISTEMAS DIGITALES

# Implementación y Verificación de un Filtro IIR Bicuadrático con Resource Sharing

Alumno: Agustín Galdeman  
Fecha: 21 de junio de 2025

Buenos Aires, Argentina

# 1. Introducción

En el presente informe se resume la implementación de un filtro IIR biquadrático pasabajos con compartición de recursos (resource sharing), junto a su respectivo testbench.

Se propuso la implementación de un filtro con resource sharing, ya que se quiso optimizar el área utilizada, reduciendo la cantidad de multiplicadores y sumadores del filtro. En general, el sistema puede operar mucho más rápido que la frecuencia de generación de muestras; por ende, utilizar más ciclos de reloj para las operaciones básicas del filtro —en lugar de realizarlas todas en paralelo— resulta factible.

Para el desarrollo del trabajo, se utilizaron las implementaciones propuestas, con su interfaz y coeficientes dados. Durante la clase se implementó el sistema de control que maneja el orden de las operaciones, y posteriormente se mejoró el testbench para comprobar el correcto funcionamiento del filtro.

Como adicional a lo pedido en clase, se expandió el testbench y se generaron los resultados de la simulación en un archivo de texto. Se dejó abierta la posibilidad de agregar un script en Python que compare los resultados del RTL con un modelo de referencia (golden model). Dado que en la VM de la cátedra solo se cuenta con Python 2.7.5, se optó por no completar el archivo de verificación.

## 2. Interface

La interfaz del módulo es fija y está especificada en el enunciado. La señal `enable` se implementa de forma estática y sirve para encender o apagar el bloque. Gracias a esto, si se diseña adecuadamente, se puede prender o apagar el clock, lo cual permite reducir el consumo total de energía. Este tipo de bloque se conoce como celda *ICG* (Integrated Clock Gating).

El `reset` original es un *reset* asíncrono, aislado, pensado principalmente para eventos de *power-on reset*. En cambio, el `reset` que se agregó en esta implementación es un `reset` síncrono, que se utiliza como mecanismo de control, alineado con el clock del módulo.

La señal `req` funciona como una petición para comenzar a procesar datos. Si nadie activa esta señal, el sistema no hace nada. Quien levanta esa señal de `request` es el CIC (Clock Interface Controller), que se encarga de iniciar el procesamiento cuando corresponde.

Las señales del sistema incluyen:

- `x`: entrada de datos
- $a_1, a_2, b_0, b_1, b_2$ : coeficientes del filtro
- `y`: salida de datos

Todos los coeficientes se dejan como entradas programables, para poder ajustar el filtro dinámicamente en función de los valores calculados previamente.

Además, hay tres salidas adicionales:

- `ynew`: indica al siguiente bloque que hay un nuevo dato de salida disponible. Se activa cada vez que se genera una nueva salida `y`.
- `ysatlo` y `ysathi`: señales de monitoreo que indican si hubo saturación negativa o positiva en el filtro IIR.

Este tipo de señales de monitoreo son comunes en sistemas reales, ya que permiten reportar fallas, y también son útiles en arquitecturas con redundancia para reasignar prioridades o recursos ante errores.

Dado que los coeficientes e inputs pueden tener diferentes anchos de bit, se implementó una unidad de extensión. Su función es normalizar los tamaños y alinear todos los datos en punto fijo. Esto se hace agregando ceros o aplicando desplazamientos de bits. Por ejemplo, si un coeficiente tiene solo 4 bits, se lo extiende a 32 bits para unificar el formato de procesamiento.

### 3. Testbench

Se propusieron una serie de cambios el testbench para mejorar la metodología de testeо. Hay dos diferencias fundamentales:

- **Estimulos :**

- **testbench original:** seno de 27,5 kHz durante 10 000 ciclos.
- **testbench modificado:** escalón fijo de amplitud STEP\_INPUT = 0,5

- **Verificación:**

- **testbench original:** sin chequeo real; sólo se observan *waveforms*.
- **testbench modificado:** cálculo del error muestra a muestra y decisión *PAS-S/FAIL* si el error cae debajo de una tolerancia antes de llegar a un número máximo arbitrario de clocks.

- **Registro de resultados:**

- **testbench original:** sin registro de resultados.
- **testbench modificado:** abre *verilog\_outputs.txt* y *loguea* cada muestra, facilitando el post processing y la verificación mediante scripts.

#### 3.1. Explicación de cambios

A continuación se explicará como se implementaron las modificaciones mencionadas:

Se añadieron los siguientes parámetros para ayudar imprimir resultados del testeо del DUT. Se limitó la cantidad de clocks a analizar, para evitar caer en loops infinitos que pudieran haber sido causado por errores en el modulo del filtro o del testbench. El número de clocks elegidos fue previamente chequeado en los waveforms, y garantiza que la señal llegue a su valor estacionario con holgura.

```
localparam real STEP_INPUT      = 0.5;    // Escalón ( $\pm 1 = FS$ )
localparam real TOLERANCE       = 0.02;   // 2 % de error permitido
localparam int  MAX_SETTLING_CLOCKS = 100000;
```

##### 3.1.1. Transformación a float

Otro cambio importante es el agregado de las siguientes líneas:

```
always @(*) begin
    // Convertir de formato fijo a real (asumiendo formato Q15)
    current_output = $signed(o_y) / (2.0**(IIR_LPF_WLX-1));
end
```

El bloque de código se utiliza para convertir la salida del filtro digital (*o\_y*) desde formato de punto fijo a número real durante la simulación.

Esta conversión es necesaria ya que el filtro trabaja internamente en formato de punto fijo (por ejemplo, Q15, que representa números con 1 bit de signo y 15 bits de fracción),

mientras que si se quisiese utilizar un el modelo de referencia (golden model) para verificación, por ejemplo en Python, se debería operar en punto flotante.

La expresión `$signed(o_y)` convierte la salida binaria a un número entero con signo, mientras que la división por  $2.0^{(IIR\_LPF\_WLX-1)}$  aplica el factor de escala correspondiente para obtener el valor real. Este resultado se guarda en la variable `current_output` de tipo `real`, lo que permite:

- Mostrar el valor por consola con `$display`.
- Guardarlo en un archivo .csv o .txt para comparación posterior con el modelo de referencia.
- Representarlo gráficamente en herramientas como GTKWave, etc.

### 3.1.2. Módulo monitor de settling time

Por otro lado, se agregó un bloque monitor de settling time, es decir, un medidor de cuánto tiempo (en ciclos de reloj) tarda la salida del filtro en estabilizarse dentro de una tolerancia definida luego de la aplicación de un escalón:

```
// _____
// Monitor de settling time - CORREGIDO
// _____
always @(posedge tb_clk) begin
    if (o_y_new && step_applied && !settling_done) begin
        error = (current_output - TARGET_VALUE);
        if (error < 0) error = -error; // Valor absoluto

        $display("Clock - %0d : - Output = -%f , - Target = -%f , - Error = -%f" ,
                settling_clocks , current_output , TARGET_VALUE, error);

        if (error <= TOLERANCE) begin
            settling_done = 1'b1;
            $display("\n*** - ANALISIS - DE - SETTLING - TIME - ***");
            $display("Valor - Target : -%f" , TARGET_VALUE);
            $display("Valor - Final - Output : -%f" , current_output);
            $display("Error : -%f" , error);
            $display("Tolerancia : -%f" , TOLERANCE);
            $display("Settling - Time : -%0d - clocks" , settling_clocks);
            $display("Settling - Time : -%0.2f - us" , settling_clocks * TS / 1us);
            $display("*** - FIN - ***\n");
        end else begin
            settling_clocks++;
            // Protección contra bucle infinito
            if (settling_clocks >= MAX_SETTLING_CLOCKS) begin
                $display("\n*** - ERROR: - SETTLING - TIME - EXCEDIDO - ***");
                $display("Máximo - de - clocks - alcanzado : - %0d" , MAX_SETTLING_CLOCKS);
                $display("El - filtro - no - converge - en - el - tiempo - esperado" );
                $display("Output - actual : -%f , - Target : -%f , - Error : -%f" ,
                        current_output , TARGET_VALUE, error);
            end
        end
    end
end
```

```

        current_output , TARGET_VALUE, error );
$display ("***-TERMINANDO-SIMULACION-***\n");
settling_done = 1'b1; // Forzar salida del loop
end
end
end

```

Este bloque funciona de la siguiente manera:

- Escucha en flanco ascendente del reloj de testbench (tb\_clk).
- Solo se activa si se detecta una nueva salida (o\_y\_new), ya se aplicó el escalón (step\_applied) y aún no se alcanzó el establecimiento (!settling\_done).
- Calcula el error absoluto entre la salida actual (current\_output) y el valor objetivo (TARGET\_VALUE).
- Si el error está dentro de una tolerancia predefinida (TOLERANCE), considera que el sistema se stabilizó, imprime resultados y detiene la medición.
- Si no está dentro de tolerancia, incrementa el contador de ciclos (settling\_clocks).
- Si se excede un límite máximo de ciclos permitidos (MAX\_SETTLING\_CLOCKS), se reporta error y se termina la simulación.

### 3.1.3. Bloque de testing modificado

Por último, pero no menos importante, la parte de testing fue modificada ampliamente para lograr generar pruebas más significativas y dejar acentadas las bases para poder añadir un script de verificación. Los pasos del nuevo bloque de testing son los siguientes:

- Se inicializan variables de simulación y se aplican las condiciones iniciales: activación del enable, reinicio del DUT y aplicación del escalón (STEP\_INPUT).
- Se crea un archivo de salida `verilog_outputs.txt`, donde se guarda cada muestra generada por el filtro. El archivo incluye: índice de muestra, salida en punto flotante, salida en hexadecimal e input aplicado.
- La simulación principal se organiza con un bloque `fork-join-any`, que permite ejecutar en paralelo dos procesos:
  1. **Proceso 1:** realiza la simulación completa, ejecutando un ciclo por muestra, evaluando si hay nueva salida disponible (o\_y\_new) y almacenando los resultados.

A su vez, si aún no se alcanzó el establecimiento, calcula el error con respecto al valor objetivo y verifica si se encuentra dentro de la tolerancia especificada. Si se cumplen las condiciones, reporta el tiempo de establecimiento en cantidad de muestras y en microsegundos.

También incluye protección contra bucles infinitos: si se superan las muestras permitidas sin convergencia, se reporta error pero se continúa grabando.

2. **Proceso 2:** timeout de seguridad. Si por alguna razón la simulación se extiende demasiado tiempo (por ejemplo, por una mala configuración del filtro), este proceso forzado reporta un mensaje de error y permite finalizar la simulación limpiamente.

- Al finalizar la simulación (cuando cualquiera de los procesos concluye), se cierra el archivo de salida y se muestra el total de muestras guardadas.
- Finalmente, se dejó comentada la línea que ejecutaría un script en Python (`compare_with_golden.py`) que podría comparar automáticamente las salidas del filtro con el modelo de referencia.
- Se concluye la simulación con un mensaje de finalización.

Este nuevo bloque reemplaza los métodos de prueba anteriores por un enfoque más robusto y automatizado, integrando monitoreo en tiempo real, almacenamiento estructurado de datos, análisis de tiempo de establecimiento, y permite añadir verificación automática contra un modelo de referencia.

## 4. Resultados

En esta sección se dejan por último los resultados de varios pasos del trabajo práctico. En primer lugar se muestran los resultados de la simulación utilizando una entrada senoidal, se puede ver el retardo que existe en el bloque

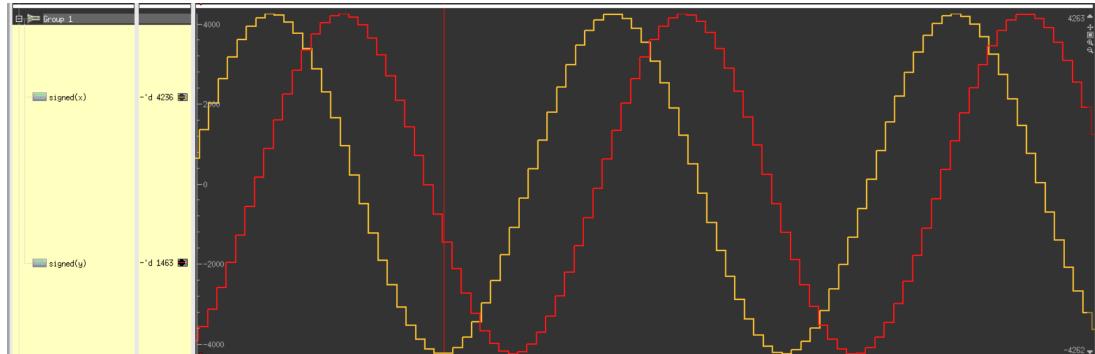


Figura 1: Entrada y Salida - señal senoidal

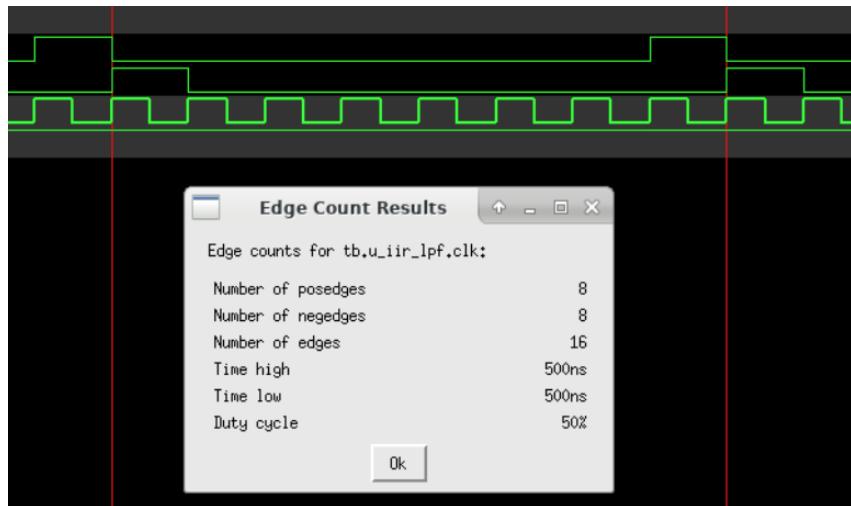


Figura 2: Verificaicon de numero de edges - señales de control

A continuación se muestra la respuesta del filtro a un escalón de módulo 0.5. Se puede ver bien el overshoot del filtro y como se estabiliza en el valor final:

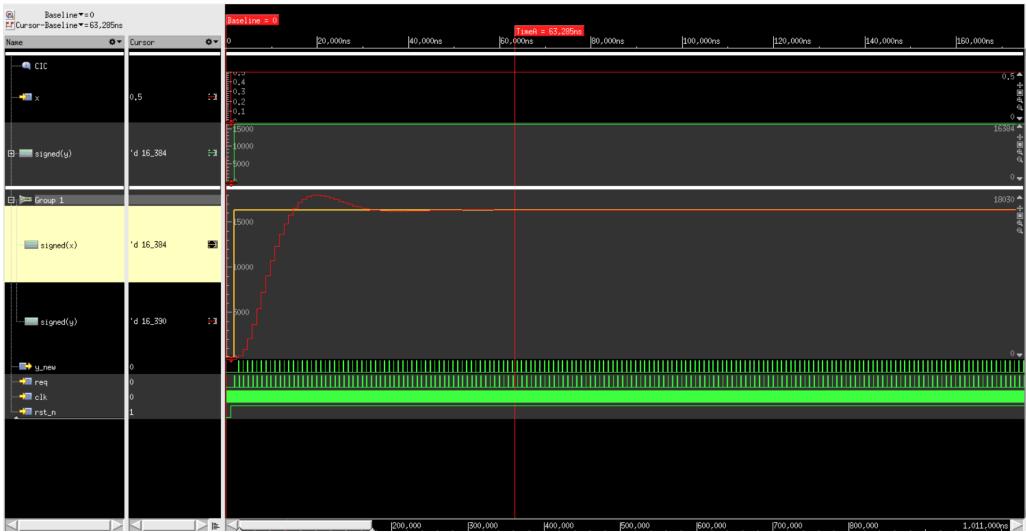


Figura 3: Respuesta al escalón

Por último se deja la impresión en consola de la simulación:

```
xmsim: +W,MTDYNUSE: SHM probe is configured with dynamic object dumping enabled. This is likely to degrade probing performance.
Created probe 1
xcelium> run

ENABLE
async reset
ENABLE
async reset

==== COMENZANDO TEST DE RESPUESTA AL ESCALON ====
Valor de Escalon Input: 0.500000
Target Output: 0.500000
Tolerancia: 0.020000 (2.0%)
Cantidad máxima de Clocks: 100000
=====

Clock 0: Output = 0.005524, Target = 0.500000, Error = 0.494476
Clock 1: Output = 0.026993, Target = 0.500000, Error = 0.473907
Clock 2: Output = 0.062805, Target = 0.500000, Error = 0.437195
Clock 3: Output = 0.110291, Target = 0.500000, Error = 0.389709
Clock 4: Output = 0.163940, Target = 0.500000, Error = 0.336066
Clock 5: Output = 0.219971, Target = 0.500000, Error = 0.280029
Clock 6: Output = 0.275330, Target = 0.500000, Error = 0.224670
Clock 7: Output = 0.327728, Target = 0.500000, Error = 0.172272
Clock 8: Output = 0.375549, Target = 0.500000, Error = 0.124451
Clock 9: Output = 0.417755, Target = 0.500000, Error = 0.082245
Clock 10: Output = 0.453766, Target = 0.500000, Error = 0.046234

*** ANALISIS DE SETTLING TIME ***
Valor Target: 0.500000
Valor Final Output: 0.483429
Error: 0.016571
Tolerancia: 0.020000
Settling Time: 23 samples
Settling Time: 2.30 us
*** CONTINUANDO GRABACION ***

Datos guardados en: verilog_outputs.txt
Total de muestras guardadas: 1998

==== SIMULACION COMPLETADA ====
Simulation complete via $finish() at time 1012 US + 0
/home/agaldeman/Documentos/iir_practica_19_5_25/tb/tb.sv:289      $finish();
xcelium> exit
TOOL: xrun(64)      23.09-s006: Exiting on Jun 20, 2025 at 22:46:46 -03 (total: 00:00:00)
[agaldeman@fiubarh01 verification]$
```

Figura 4: Resultado testing del rtl.