Laboratorio 8:

Diseños monociclo vs. multiciclo

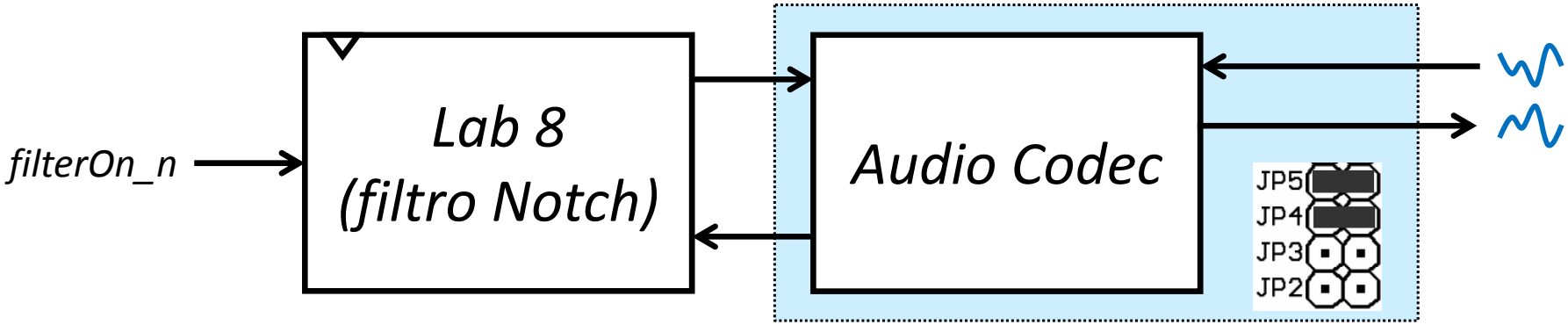
transmisión y procesado de sonido muestreado por bus IIS

Diseño automático de sistemas

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Presentación

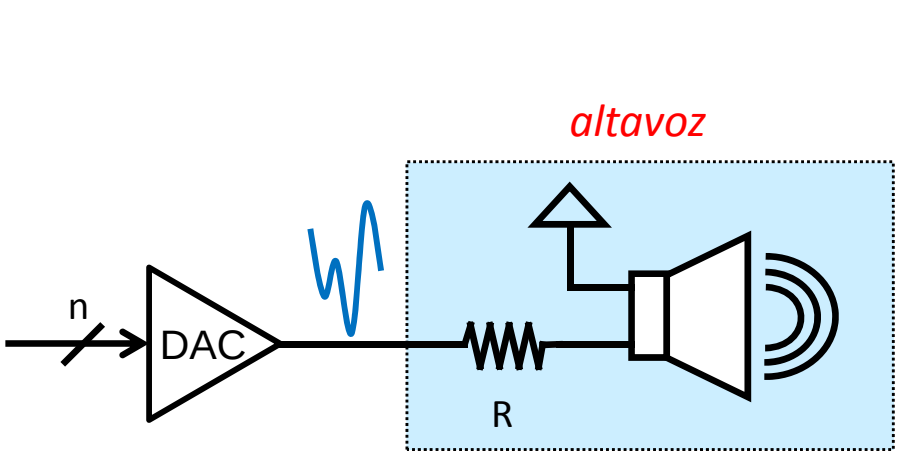
- Diseñar un procesador digital de sonido muestreado elemental.
  - En tiempo real eliminará una frecuencia parásita de una señal de audio digital.
    - Mediante un filtro notch de características configurables
  - Para recibir/enviar muestras de sonido usará el audio CODEC de la placa XST.
  - Para probar su funcionamiento, podrá seleccionarse usando un pulsador si:
    - Se envían las mismas muestras que se reciben (escuchándose en audio original)
    - Se envían las muestras recibidas una vez filtradas (escuchándose en audio procesado)

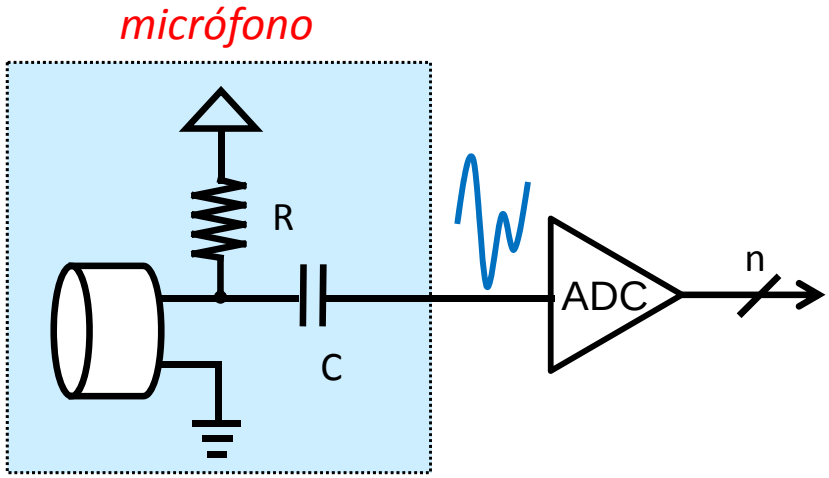


J.M. Mendías
2017

Sonido muestreado

- Un sistema digital puede generar sonidos a través de un altavoz.
  - Generando una señal digital periódica a través de uno de sus pines.
  - Sin embargo, con un bit sólo se pueden generar ondas cuadradas que varíen entre 2 niveles eléctricos.
  - De esta manera sólo puede controlar el tono (frecuencia) del sonido, pero no el volumen (amplitud) ni el timbre (forma) del mismo.
  - Para poder generar ondas con mayor riqueza es necesario disponer de un conversor digital/analógico.
- Un sistema digital puede captar sonidos a través de un micrófono.
  - Leyendo la señal eléctrica que genera usando un conversor analógico/digital

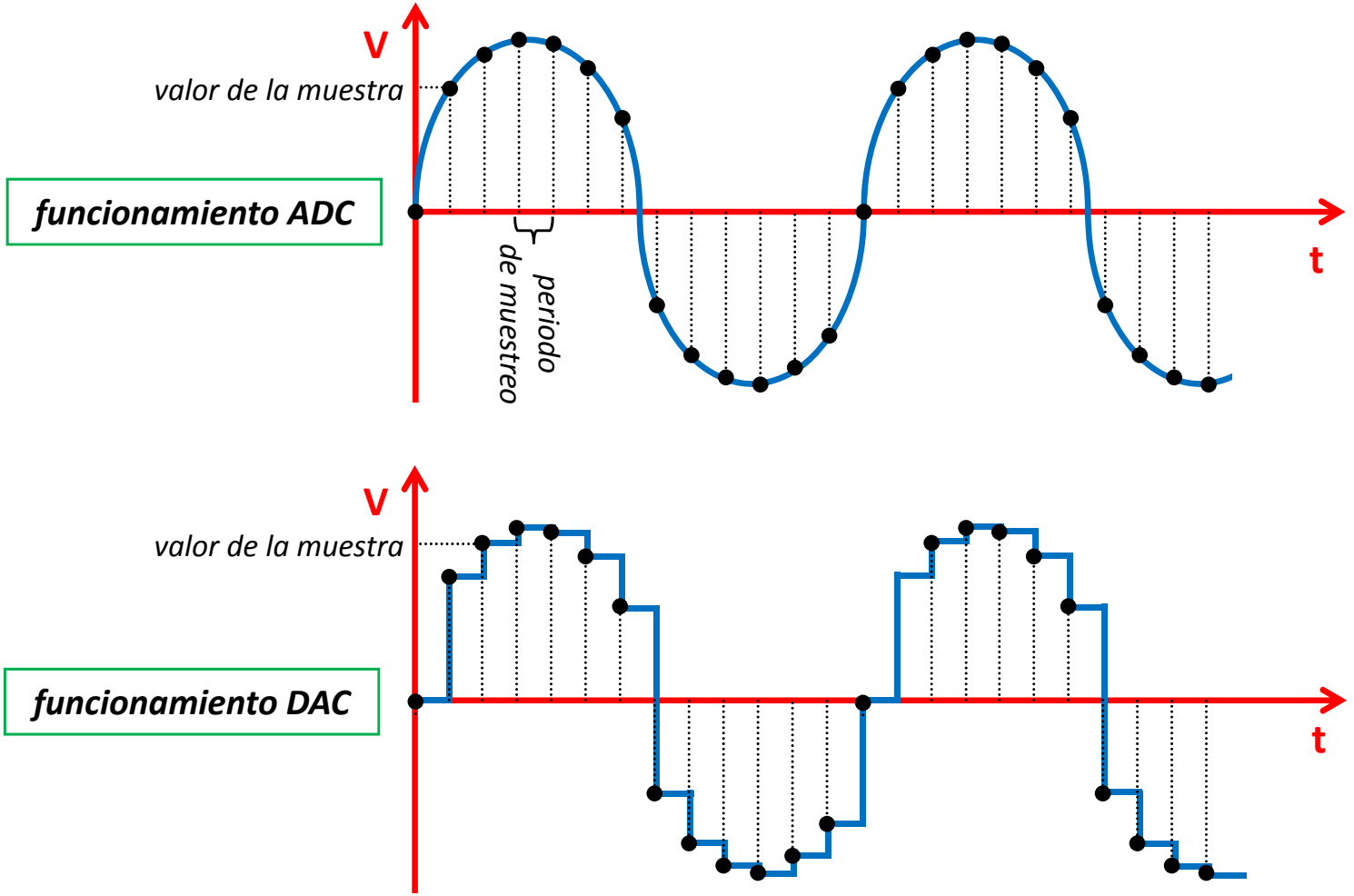




J.M. Mendías
2017

Sonido muestreado

- Generalmente los DAC/ADC se escriben/leen a una frecuencia fija denominada frecuencia de muestreo (fs)



Audio CODEC

- Un audio CODEC es un DAC/ADC especializado en el muestreo y generación de señales dentro del rango de frecuencias audibles.
  - Las señales que muestrea son señales eléctricas analógicas generadas a partir de ondas sonoras captadas por un micrófono, o generadas por un sistema de audio.
  - Las señales que genera son señales eléctricas analógicas que pueden transformarse en ondas sonoras a través de un altavoz, o ser procesadas por un sistema de audio.
- Elementos de un CODEC:
  - Frecuencia de muestreo (fs): su inversa define el intervalo que transcurre entre 2 muestras consecutivas.
    - 8.000 Hz Teléfono
    - 16.000 Hz Voz IP
    - 32.000 Hz mini DV
    - 44.100 Hz audio CD
    - 48.000 Hz DVD
    - 96.000 Hz HD-DVD
  - Número de canales: número de señales que es capaz de muestrear/generar simultáneamente.
    - 1 canal (codec mono) o 2 canales (codec stéreo).
  - Resolución: número de bits usados para codificar el valor de cada muestra.
    - Típicamente representados en C2

Audio CODEC

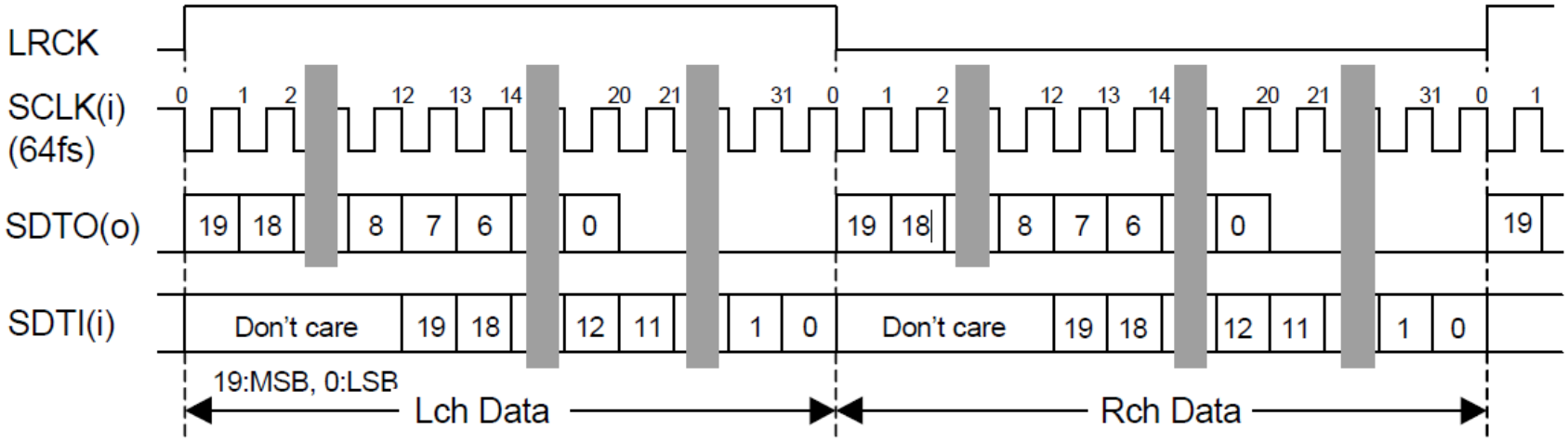
transmisión de muestras por bus IIS

- IIS (Integrated Interchip Sound) es un bus serie síncrono para la transmisión de audio stéreo entre dispositivos digitales.
  - Tiene 2 líneas unidireccionales de datos serie: SDTI (entrada) y SDTO (salida).
    - Permite el envío/recepción simultánea de muestras de sonido.
  - Tiene 3 líneas de reloj: SCLK (transmisión de datos serie), LRCK (selección de canal izquierdo/derecho), MCLK (reloj principal).
  - Todos los datos transmitidos están codificados en C2 (MSB first) con un número de bits no definido (depende del emisor/receptor).
    - Los datos serán truncados o extendidos con 0 según convenga en el emisor/receptor.
  - Las frecuencias de los relojes se definen relativas a la frecuencia de muestreo (fs)
    - $f_{LRCK} = fs$
    - $f_{SCLK} > (\text{bits/muestra}) \times (\text{número de canales}) \times fs$
    - $f_{MCLK} = \{ 256 \cdot fs / 384 \cdot fs / 512 \cdot fs \}$
- Protocolo
  - La transmisión de datos es continua.
  - El maestro genera las señales de reloj.
  - Emisor y receptor generan las señales de datos de audio alternando muestras del canal izquierdo y derecho.

Audio CODEC

AK4551

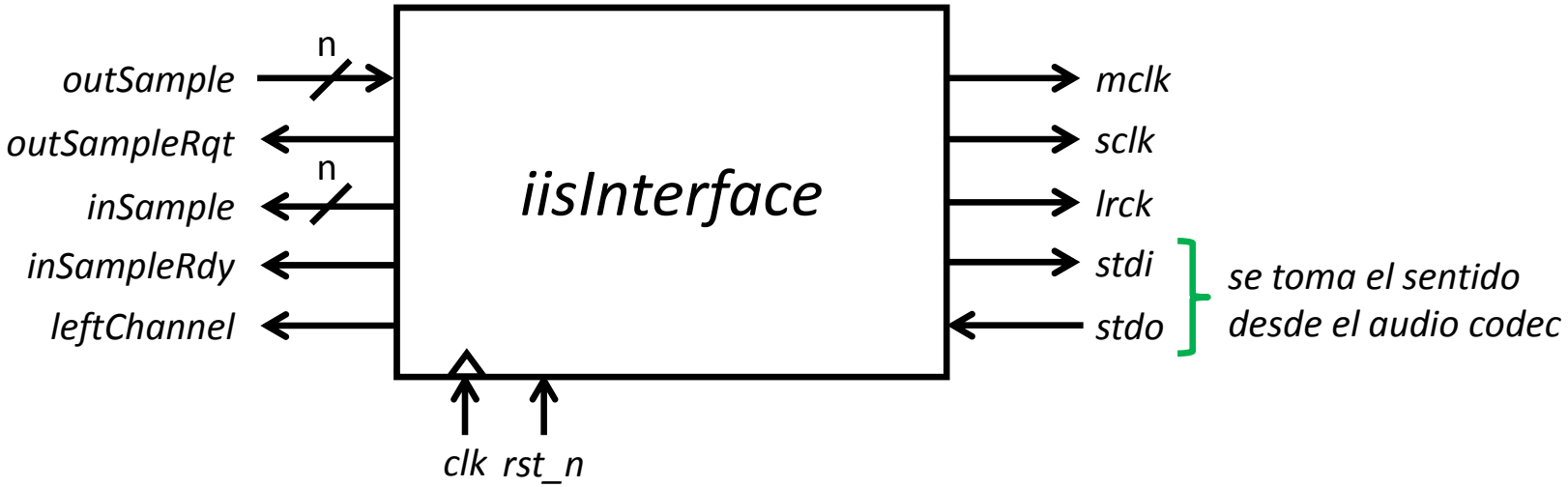
- El audio codec de la placa XST tiene las siguientes características:
  - Las muestras se codifican en C2 de 20 bits
  - Las muestras transmiten, primero MSB, según un cronograma fijo
    - Las muestras que salen del codec justificadas al comienzo del ciclo de transmisión
    - Las muestras de entran al codec justificadas al final del ciclo de transmisión



  - La relación entre las frecuencias de los relojes puede ser:
    - f_{LRCLK} = fs, donde 8 KHz < fs < 50 KHz
    - $40 \cdot fs \leq f_{SCLK} \leq 96 \cdot fs$
    - f_{MCLK} = { 256·fs / 384·fs / 512·fs }

Interfaz ISS
presentación

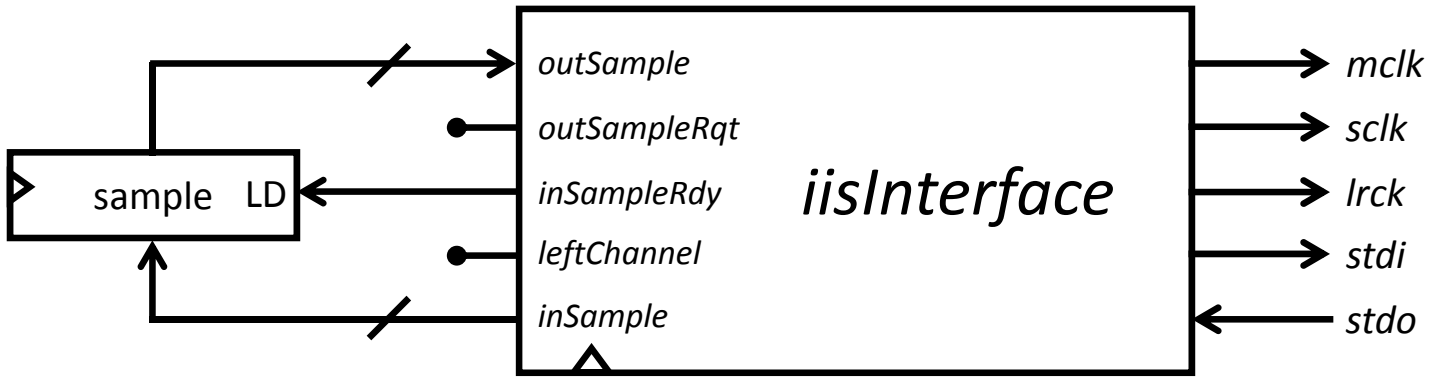
- Diseñar un interfaz IIS elemental que constantemente:
  - Convierta de paralelo a serie cada muestra a transmitir por el bus IIS.
  - Cada vez que la señal outSampleRqt se active, comenzará a transmitir el dato leído de data.
  - Convierta de serie a paralelo cada dato muestra recibida por el bus IIS.
  - Cada vez que reciba una muestra, la volcará en inSample.
  - Por cada nueva muestra activará durante un ciclo la señal de strobe inSampleRdy.
  - Generará las señales de sincronización a las siguientes frecuencias:
    - f_{LRCLK} = fs = 48,8 KHz, f_{SCLK} = 64·fs, f_{MCLK} = 256·fs



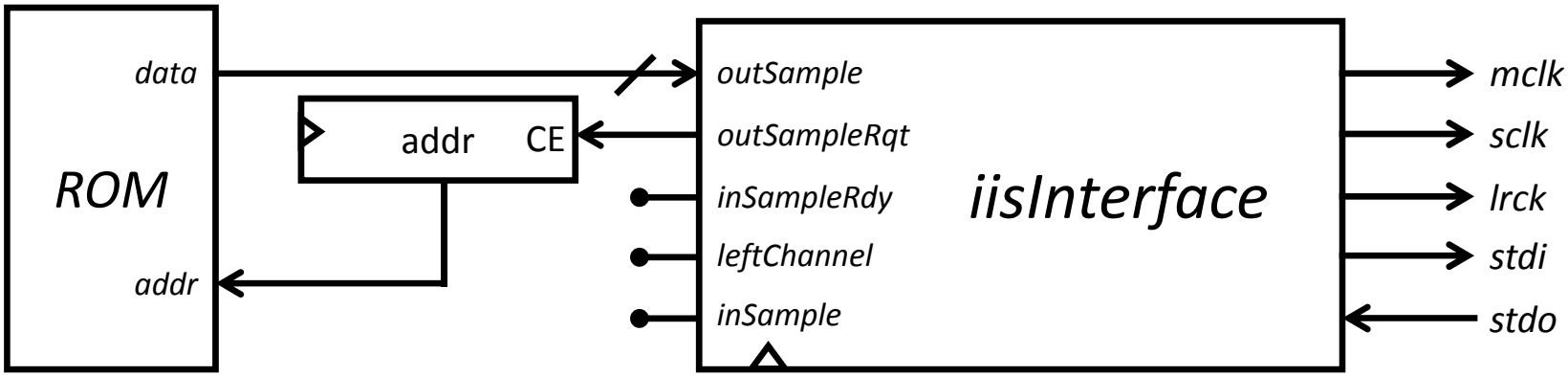
Interfaz IIS

aplicación al diseño

- Para realizar un loopback bastaría:



- Para reproducir un sonido fijo almacenado bastaría:



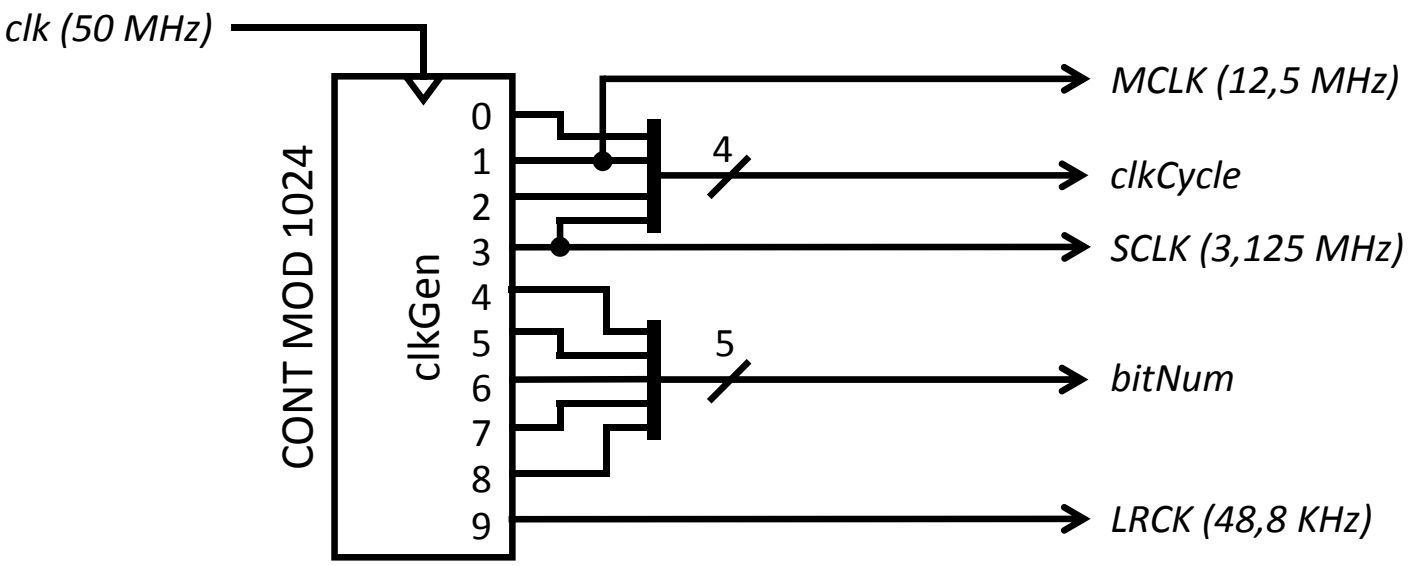
J.M. Mendías 2017

Interfaz IIS

generación de relojes

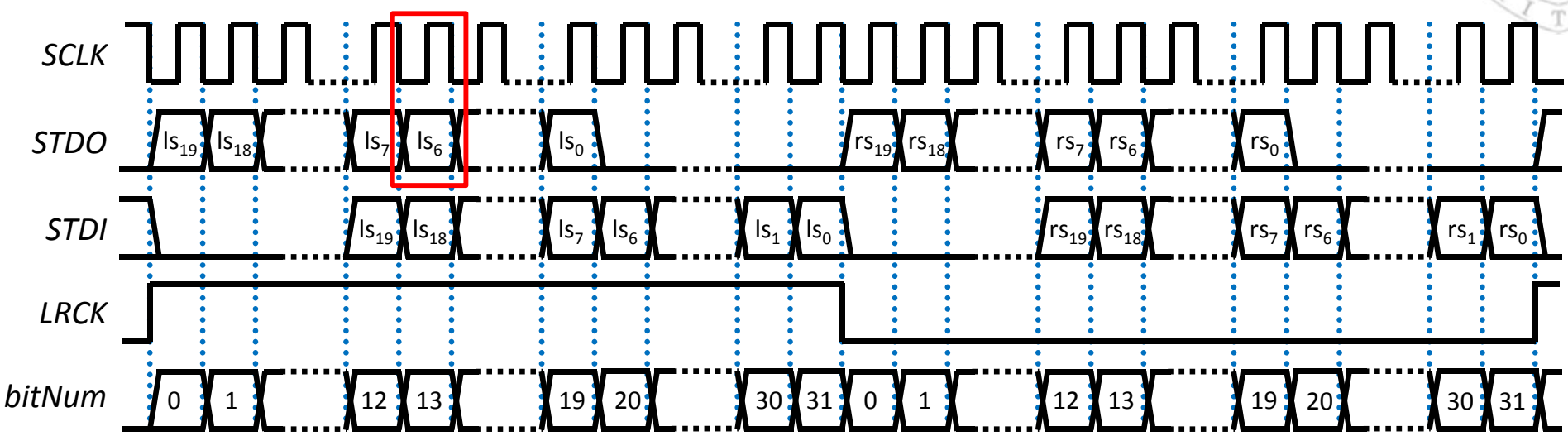
- Dado que la relación entre las frecuencias de los relojes es fija, pueden derivarse a partir de la señal de reloj principal:
  - fs = 48,8 KHz $\Rightarrow$ f_{MCLK} = 12,5 MHz = 50 MHz ÷ 4 $\Rightarrow$ $f_{MCLK} = f_{CLK} \div 4$
  - f_{MCLK} = 256·fs
  - f_{SCLK} = 64·fs
  - f_{LRCLK} = fs

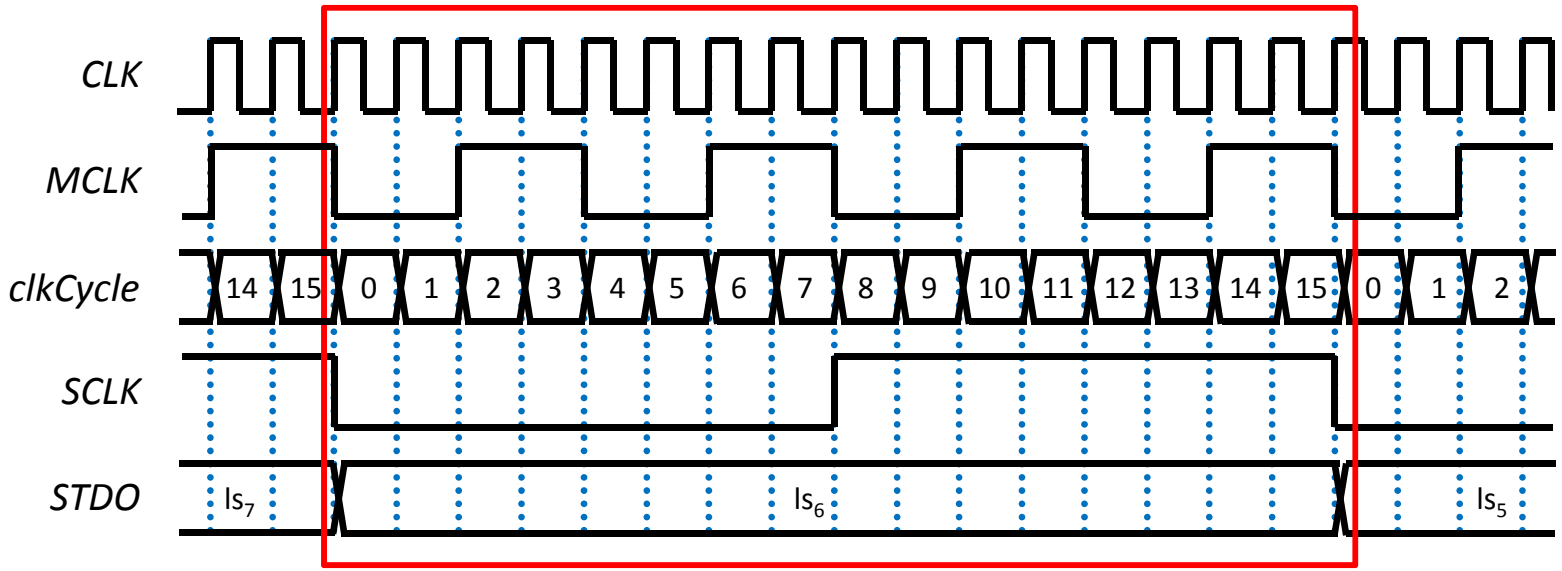
$$\Rightarrow \begin{cases} f_{SCLK} = f_{MCLK} \div 4 \\ f_{LRCLK} = f_{MCLK} \div 256 \end{cases} \Rightarrow \begin{cases} f_{SCLK} = f_{CLK} \div 16 \\ f_{LRCLK} = f_{CLK} \div 1024 \end{cases}$$



Interfaz IIS

análisis del comportamiento (i)





Interfaz IIS

recepción de muestras

- Se requiere un registro de 20 bits que deberá desplazar cuando:
  - Por *STDO* se estén transmitiendo datos válidos:
    - *bitNum* < 20
  - Los datos que transmite STDO estén estabilizados:
    - *clkCycle* = 7 (el centro de la muestra)

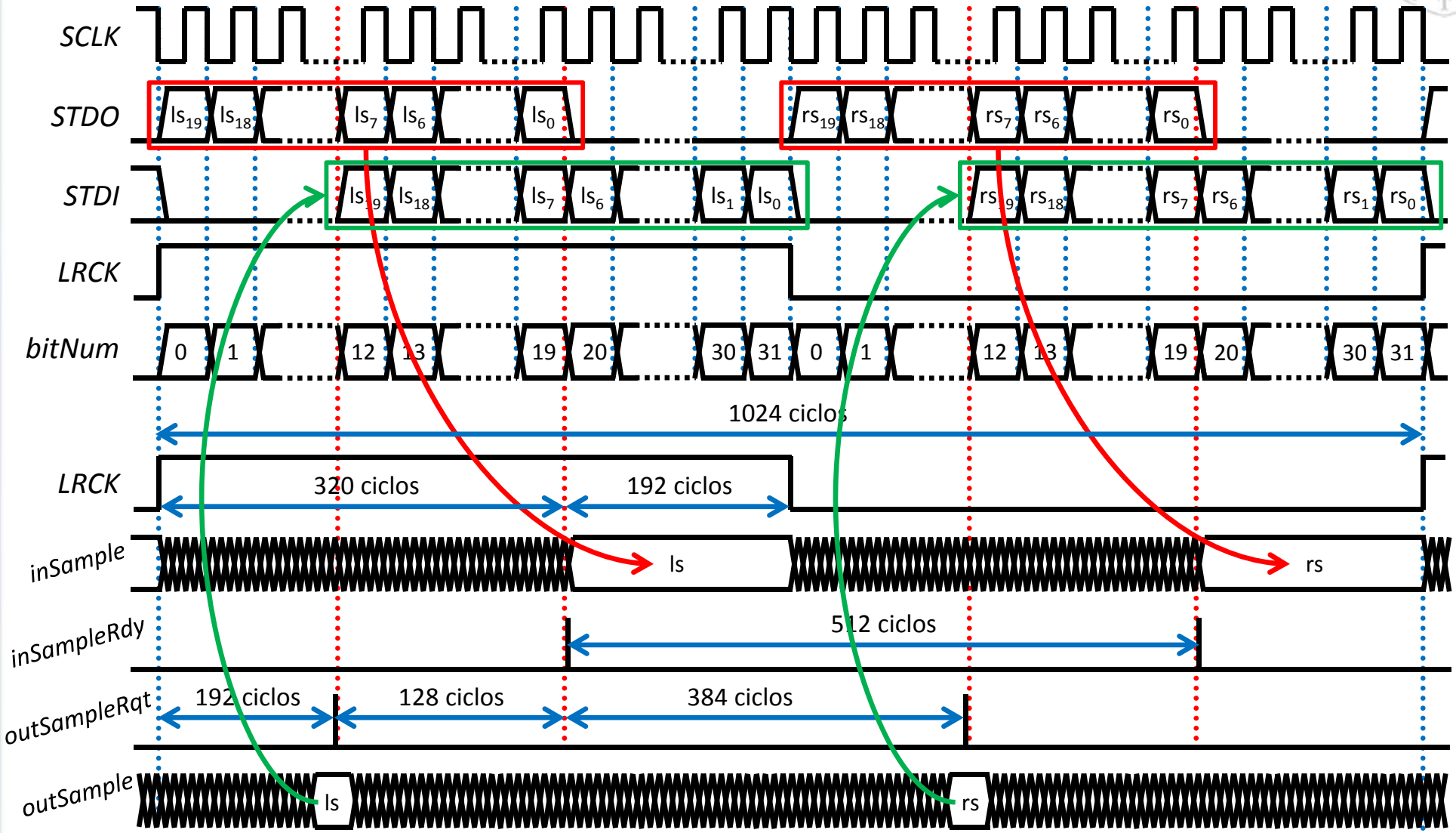
- La señal inSampleRdy se activará durante un ciclo cuando:
  - Haya finalizado la conversión serie a paralelo :
    - *bitNum* = 20 y *clkCycle* = 0 (justo en el ciclo de reloj siguiente)
    - Aunque lo podría hacer en cualquier clkCycle con bitNum ≥ 20

Interfaz IIS
transmisión de muestras

- Se requiere otro registro de 20 bits que deberá desplazar cuando:
  - Por *STDI* se deban transmitir datos válidos:
    - *bitNum* > 11
  - De modo que los datos a transmitir por *STDI* cambien a flanco de bajada de *SCLK*:
    - *clkCycle* = 15
- El registro deberá cargar un nuevo dato
  - Antes de que comience la conversión paralelo-serie
    - *bitNum* = 11 y *clkCycle* = 15 (justo en el ciclo de reloj anterior)
    - Aunque lo podría hacer en cualquier clkCycle con bitNum ≤ 11
- La señal outSampleRqt se activará durante un ciclo cuando:
  - El registro de desplazamiento vaya a cargar un nuevo dato:
    - *bitNum* = 11 y *clkCycle* = 15

Interfaz IIS

análisis del comportamiento (ii)

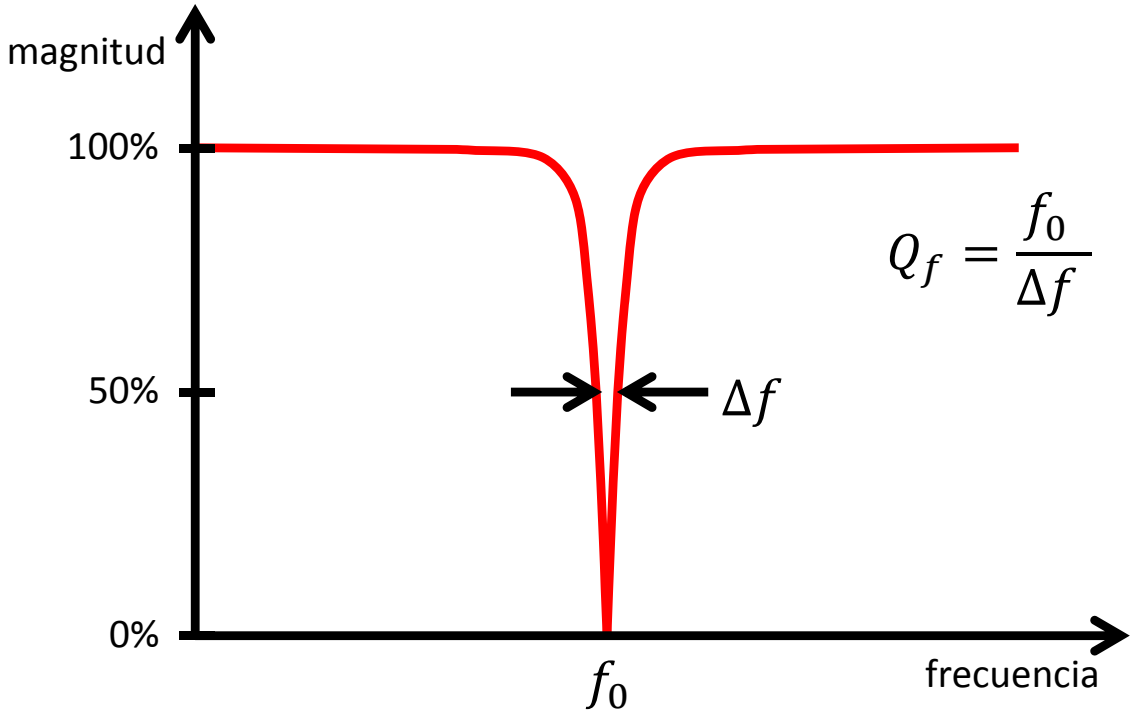


J.M. Mendías 2017

Filtro notch

presentación

- Un notch filter es un filtro que atenúa las frecuencias en un rango muy estrecho dado y deja inalteradas el resto de frecuencias de una señal.
  - Se usa para eliminar interferencias (p.e. la causada por la alimentación alterna)
- Su comportamiento suele definirse en base a 2 parámetros
  - Frecuencia de corte (f_0): frecuencia en la que se consigue la máxima atenuación.
  - Factor de calidad (Q_f): medida de la estrechez de la banda que filtra.



Filtro notch

especificación (i)

- Un notch filter puede implementarse como un filtro IIR de 2do. orden con la siguiente función de transferencia.

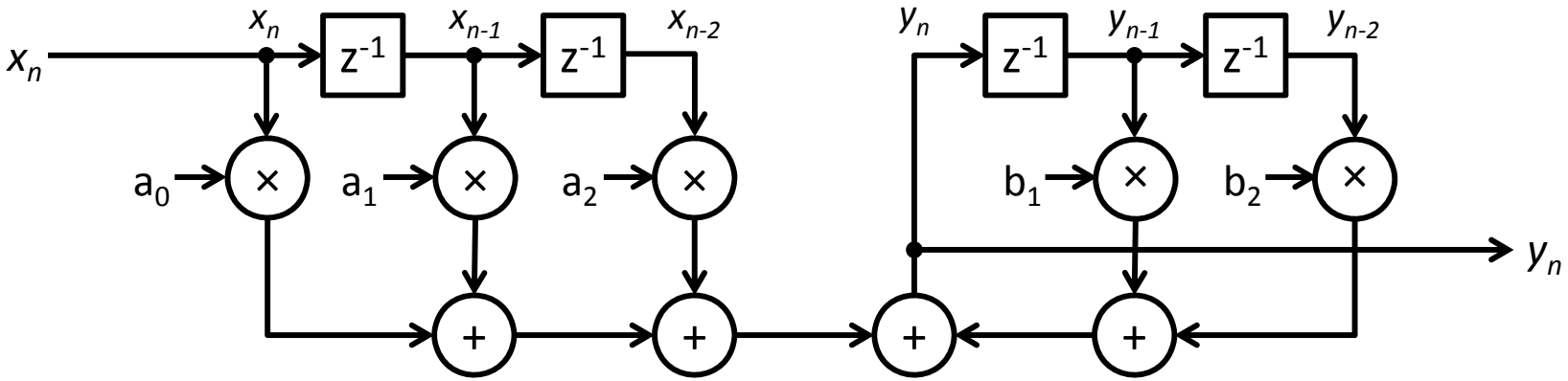
$$H(z) = k\frac{1 - 2\cos(\omega_0)\,z^{-1} + z^{-2}}{1 - 2k\cos(\omega_0)\,z^{-1} + (2k-1)z^{-2}} \qquad donde: \quad \omega_0 = 2\pi\frac{f_0}{f_s} \qquad k = \frac{1}{1 + \tan(\frac{\omega_0}{2 \cdot Q_f})}$$

- Que para su implementación se expresa como:
  - Ecuación de diferencias:

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} + b_1 y_{n-1} + b_2 y_{n-2}$$

$$a_0 = k$$
$$a_1 = -2k\cos(\omega_0)$$
$$a_2 = k$$
$$b_1 = 2k\cos(\omega_0)$$
$$b_2 = -(2k-1)$$

  - Diagrama de bloques:



Filtro notch

especificación (ii)

- Así, por ejemplo, para eliminar una frecuencia de 800Hz en una señal de audio muestreado a 48.8 KHz:

$$\left.\begin{array}{l} Q_f = 60 \\ f_0 = 800\ Hz \\ f_s = 48800\ Hz \end{array}\right\} \quad cos\left(2\pi\frac{800}{48800}\right) = 0.9945 \quad \frac{1}{1+\tan(\pi\frac{14}{48800})} = 0.9991$$

$$a_0 = k = 0.9991 \qquad a_1 = -2k\cos(\omega_0) = -1.9872 \qquad a_2 = k = 0.9991$$

$$b_1 = 2k\cos(\omega_0) = 1.9872 \qquad b_2 = -(2k-1) = -0.9982$$

- Podría usarse un filtro IIR con la siguiente ecuación de diferencias:

$$y_n = 0.9991 \cdot x_n + (-1.9872) \cdot x_{n-1} + 0.9991 \cdot x_{n-2} + 1.9872 \cdot y_{n-1} + (-0.9982) \cdot y_{n-2}$$

- Sin embargo, el tipo real de VHDL no es sintetizable. Hay 2 alternativas para implementar la ecuación:
  - Codificar los datos en punto flotante y diseñar a mano todas las FU punto flotante.
  - Codificar los datos en punto fijo y usar las FU enteras facilitadas por la herramienta.

J.M. Mendías 2017

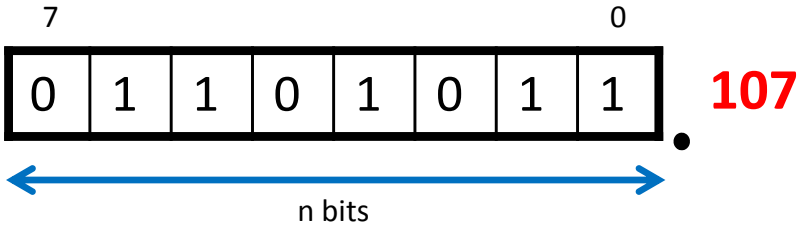
Aritmética en punto fijo

representación de datos (i)

- En punto fijo, la posición del punto en la cadena de bits es fija y queda implícita en el código.
  - En los números enteros se asume que el punto está situado a la derecha del LSB

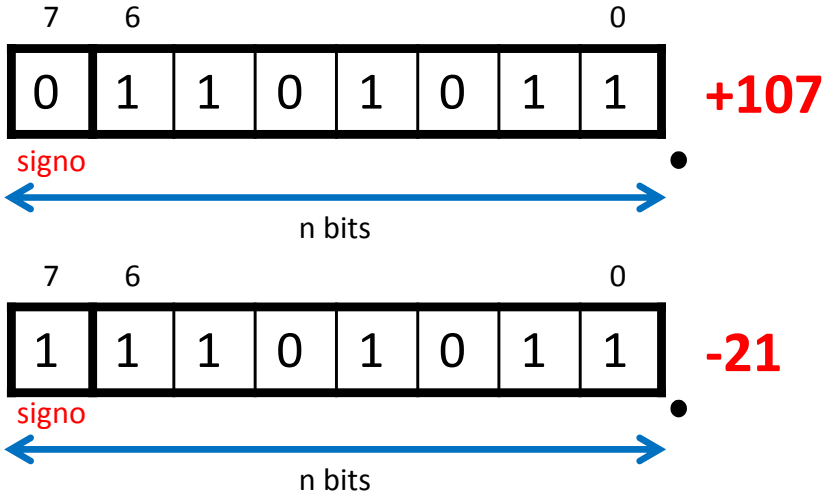
- Los números enteros sin signo se representan en binario

$$v(\underline{x}) = \sum_{i=0}^{n-1} 2^i \cdot x_i$$



- Los números enteros con signo se representan en C2

$$v(\underline{x}) = -2^{n-1} \cdot x_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot x_i$$



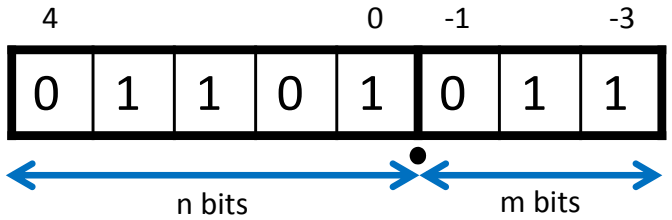
J.M. Mendías 2017

Aritmética en punto fijo

representación de datos (ii)

- Los números reales sin signo se representan en punto fijo en binario

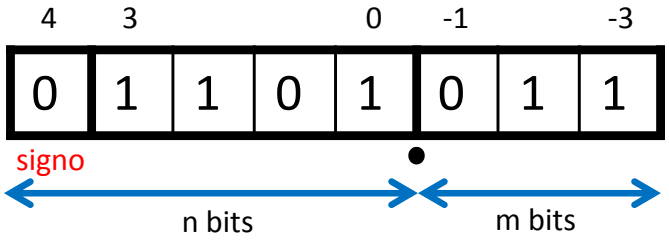
$$v(\underline{x}) = \sum_{i=0}^{n-1} 2^i \cdot x_i + \sum_{i=0}^{m-1} 2^{-i} \cdot x_{-i}$$



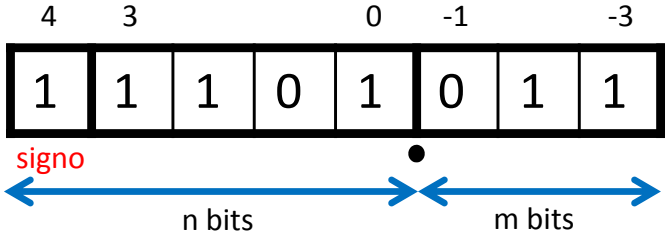
13.375

- Los números reales con signo se representan en punto fijo en C2

$$v(\underline{x}) = -2^{n-1} \cdot x_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot x_i + \sum_{i=0}^{m-1} 2^{-i} \cdot x_{-i}$$



+13.375



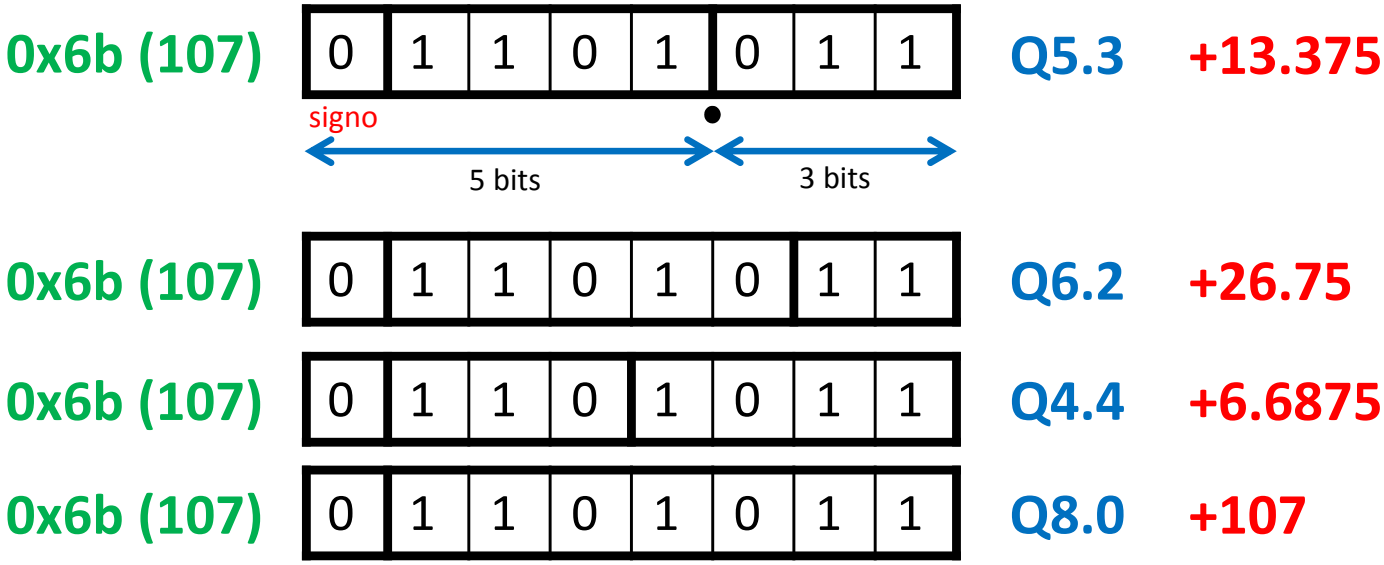
-2.625

J.M. Mendías 2017

Aritmética en punto fijo

notación Q

- Para indicar la representación concreta de punto fijo (típicamente con signo) utilizada en un tipo de dato, se usa la notación Qn.m, donde:
  - n es el número de bits enteros (excluyendo, o no, el bit de signo según convenio)
  - m es el número de bits decimales



- En ocasiones se usa únicamente Qm
  - si la anchura de la representación se sobrentiende (8/16/32 bits)

Aritmética en punto fijo

rango vs. resolución

- El rango representable en punto fijo Qn.m es $[-2^{n-1}, 2^{n-1}-2^{-m}]$
  - Por ejemplo, el rango representable en Q5.3 es $[-2^4, 2^4-2^{-3}] = [-16, +15{,}875]$
- La resolución en punto fijo Qn.m es 2^{-m} y el error máximo de representación $\pm 2^{-(m+1)}$
  - Por ejemplo, en Q5.3 la resolución es $2^{-3} = 0.125$ y el error $\pm 2^{-4} = \pm 0.0625$
- Así, a tamaño de palabra fijo:
  - Si m es muy pequeño, aumenta el error al representar números con decimales
  - Si m es muy grande, aumenta el riesgo de overflow

representación	rango	resolución	error
Q5.3	[-16, +15.875]	0.125	± 0.0625
Q6.2	[-32, +31.75]	0.25	± 0.125
Q4.4	[-8, +7.9375]	0,0625	± 0.03125
Q8.0	[-128, +127]	1	± 0.5
Q1.7	[-1, +0.9921875]	0,0078125	± 0.00390625

J.M. Mendías 2017

Aritmética en punto fijo

suma y resta (i)

- Para sumar/restar 2 números con la misma representación Qn.m
  - Al estar los puntos alineados, puede utilizarse la aritmética entera
  - Sin embargo, en general, el resultado requiere ser representado en Qn+1.m

	0	0	0	0	1	1	1	0	Q5.3	+1.75
+	0	1	1	0	1	0	1	1	Q5.3	+13.375
	0	1	1	1	1	0	0	1	Q5.3	+15.125

	0	1	1	0	0	1	0	0	Q5.3	+12.5
+	0	1	1	0	1	0	1	1	Q5.3	+13.375
CORRECTO	0	1	1	0	0	1	1	1	Q6.3	+25.875
INCORRECTO	1	1	0	0	1	1	1	1	Q5.3	-6.125

J.M. Mendías 2017

Aritmética en punto fijo

suma y resta (ii)

- Cuando la representación del resultado de la suma/resta debe ser la misma que la de los operandos, existen 2 alternativas:
  - Wrap: descartar siempre el bit mas significativo del resultado

	0	1	1	0	0	1	0	0	Q5.3 +12.5
+	0	1	1	0	1	0	1	1	Q5.3 +13.375
	~~0~~	1	1	0	0	1	1	1	Q5.3 -6.125

  - Saturar: en caso de producirse overflow/underflow devolver el máximo/mínimo valor representable.

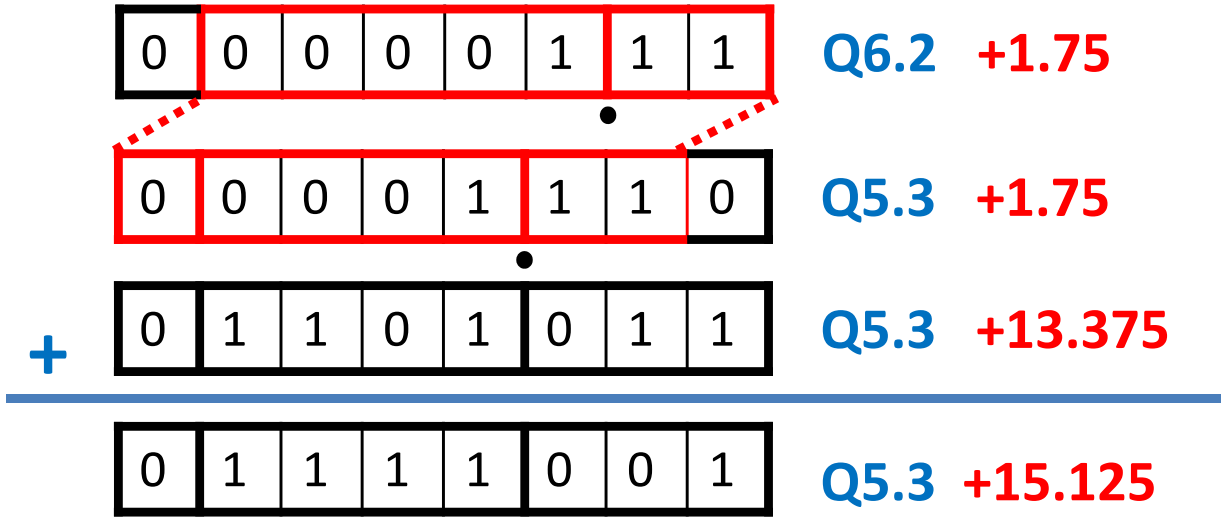
	0	1	1	0	0	1	0	0	Q5.3 +12.5
+	0	1	1	0	1	0	1	1	Q5.3 +13.375
	0	1	1	1	1	1	1	1	Q5.3 +15.875

J.M. Mendías 2017

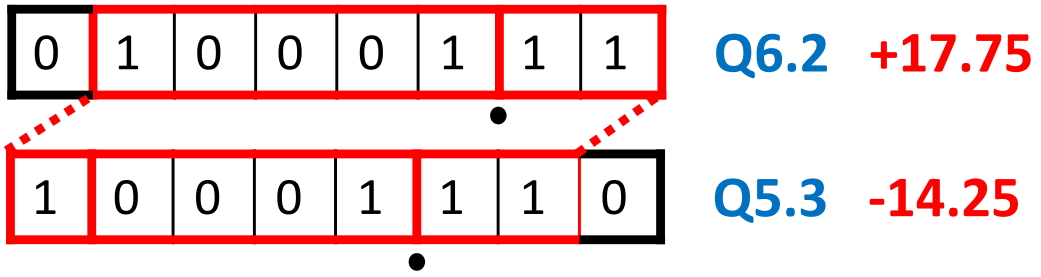
Aritmética en punto fijo

suma y resta (iii)

- Si los operandos a sumar/restar tienen representaciones distintas
  - Previamente deben alinearse los puntos mediante desplazamiento (reescalado)
  - Estos desplazamientos pueden provocar overflow o pérdida de resolución



INCORRECTO

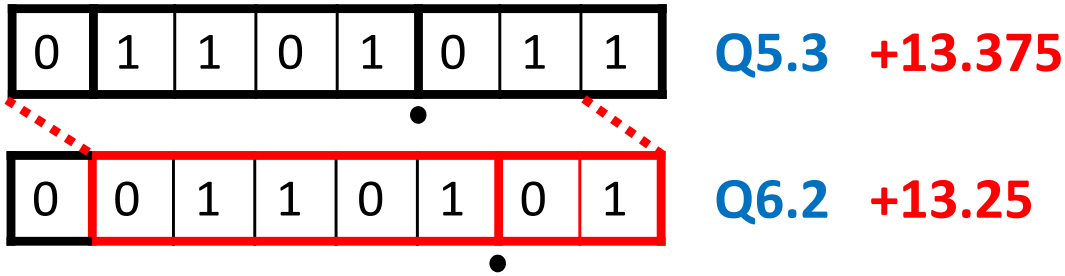


J.M. Mendías 2017

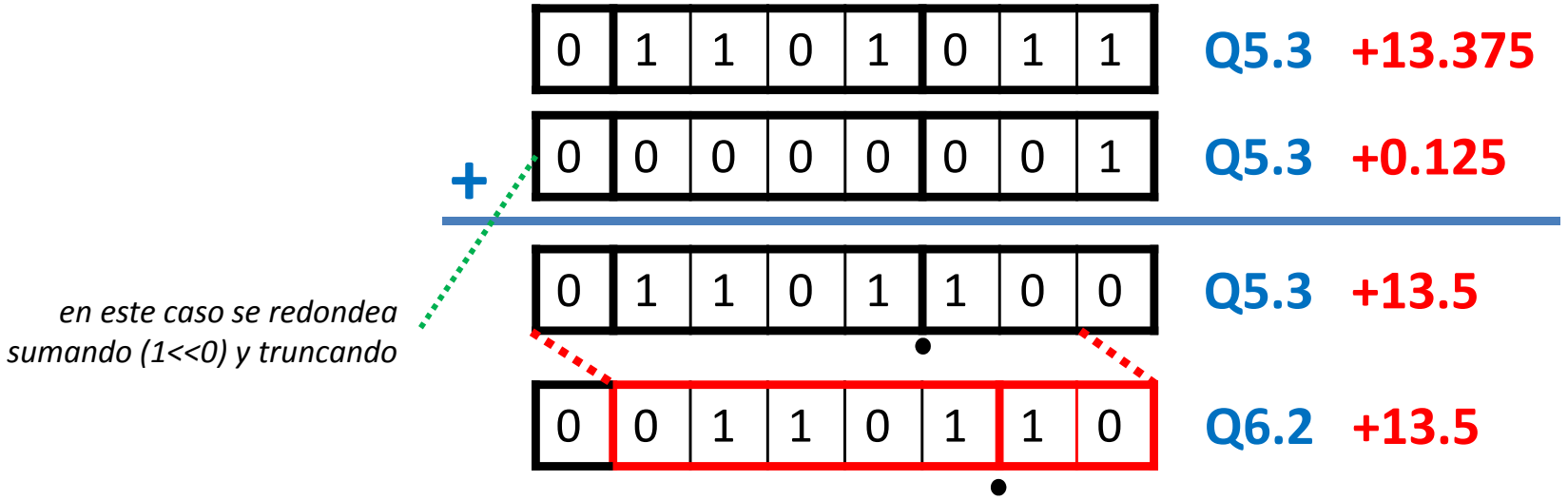
Aritmética en punto fijo

suma y resta (iv)

- Cuando el reescalado de un dato supone una pérdida de resolución, existen 2 alternativas:
  - Truncar: descartar siempre los bits menos significativos



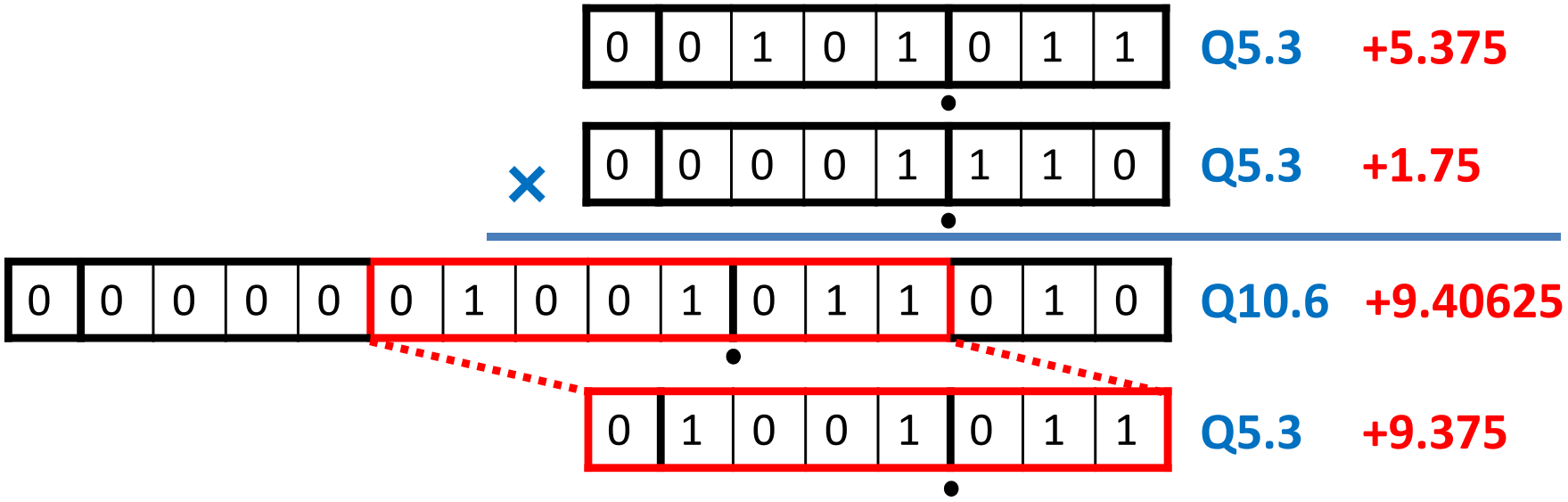
  - Redondear: Aproximar al número representable más cercano



Aritmética en punto fijo

multiplicación y división (i)

- Para multiplicar/dividir 2 números con la misma representación Qn.m
  - Se utiliza aritmética entera pero deben realizarse correcciones de escala
    - Multiplicación: $Z = (X \cdot Y) \div 2^m \Rightarrow Z = (X \cdot Y) >> m$
    - División: $Z = (X \cdot 2^m) \div Y \Rightarrow Z = (X << m) / Y)$
  - Puede producirse overflow y perdida de precisión
    - Se compensan con las mismas técnicas aplicadas con la suma

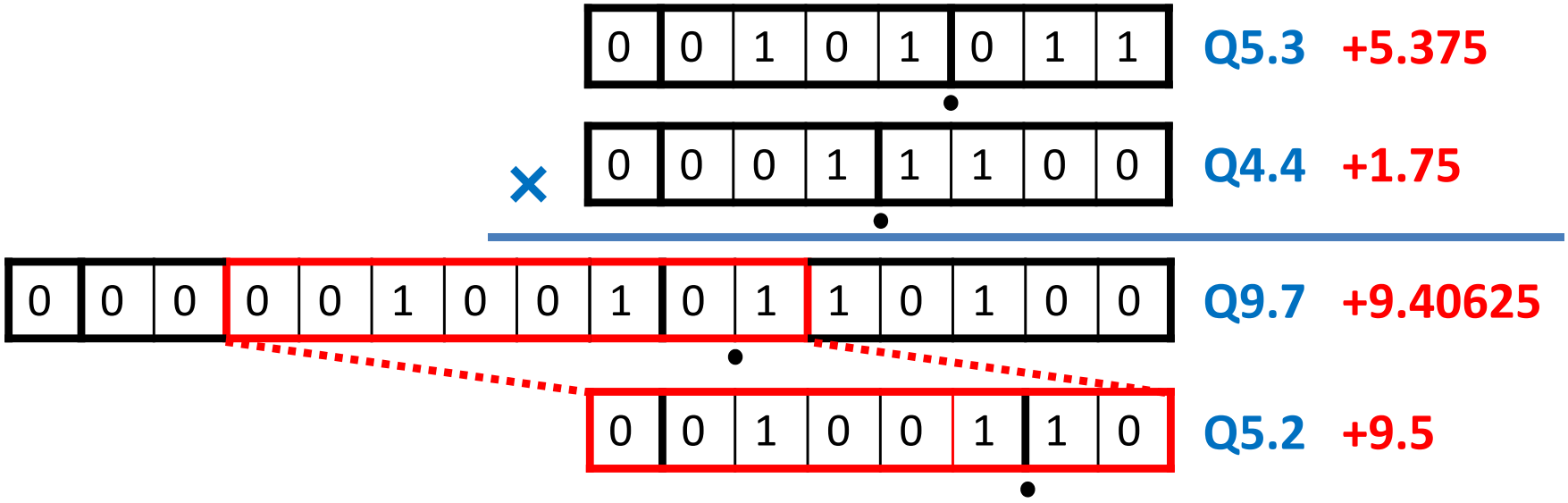


J.M. Mendías 2017

Aritmética en punto fijo

multiplicación y división (ii)

- Si los operandos tienen representaciones distintas
  - La corrección de escala se ajusta según el número de bits decimales de cada uno.
  - Por ejemplo, para multiplicar 2 números en formato $Qn_1.m_1$ y $Qn_2.m_2$ de manera que el resultado quede en formato $Qn_3.m_3$
    - $Z = (X \cdot Y) >> m_1+m_2-m_3$

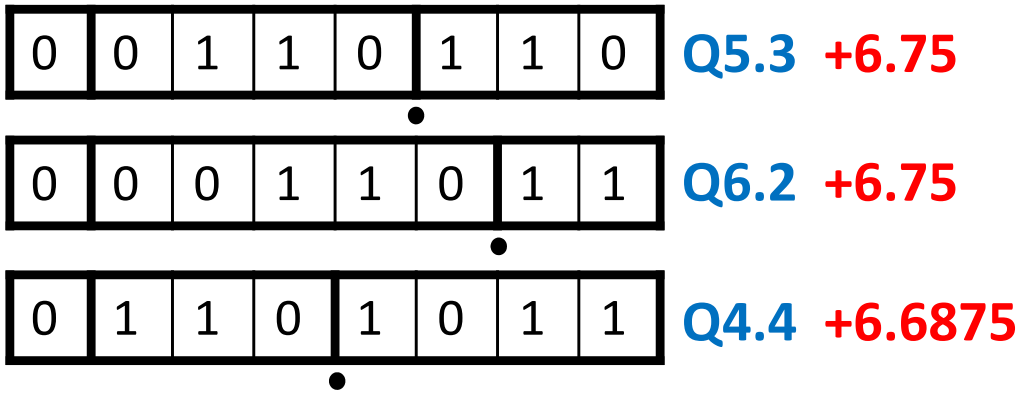


Aritmética en punto fijo

uso en VHDL (i)

- VHDL'87/93 no soporta nativamente el punto fijo, por ello es necesario:
  - Declarar las señales, variables y constantes de tipo signed/unsigned.
    - El tipo elegido deberá ser los suficientemente ancho para contener los datos.
    - La posición del punto quedará implícita y podrá ser diferente para cada dato
  - Usar los operadores convencionales (definidos en el paquete numeric_std)
    - Teniendo en cuenta los factores de escala a usar en cada operación aritmética.
    - Redondeando / truncando / saturando los resultados explícitamente en el código.
    - Gestionando los potenciales overflows.
  - Convertir manualmente los literales reales en sus correspondientes signed/unsigned.
  - Reglas de conversión de literales (real ↔ punto fijo Qn.m) :
    - $literal_{int} = redondeo(literal_{real} \cdot 2^m)$
    - $literal_{real} = literal_{int} \div 2^m$

literal real	repr.	literal entero
6.7	Q5.3	$6.7 \cdot 2^3 = 53.6 \approx 54$
	Q6.2	$6.7 \cdot 2^2 = 26.8 \approx 27$
	Q4.4	$6.7 \cdot 2^4 = 107.2 \approx 107$



J.M. Mendías
2017

Aritmética en punto fijo

uso en VHDL (ii)

```
library ieee;
use ieee.numeric_std.all;
use ieee.math_real.all;
use work.common.all;

architecture syn of notchFilter is

  ...

  type signedFixArray is array (0 to 2) of signed(WL-1 downto 0);

  constant QF : real := 60.0;
  constant w0 : real := 2.0*MATH_PI*F0/FS;
  constant k  : real := 1.0 / (1.0 + tan(w0/(2.0*QF)));

  constant a : signedFixArray := (
    toFix( k, QN, QM ),
    toFix( -2.0*k*cos(w0), QN, QM ),
    toFix( k, QN, QM )
  );

 ...

begin
  ...
end syn;
```

biblioteca de funciones matemáticas de tipo real

constantes de tipo real definidas usando funciones de la biblioteca math_real

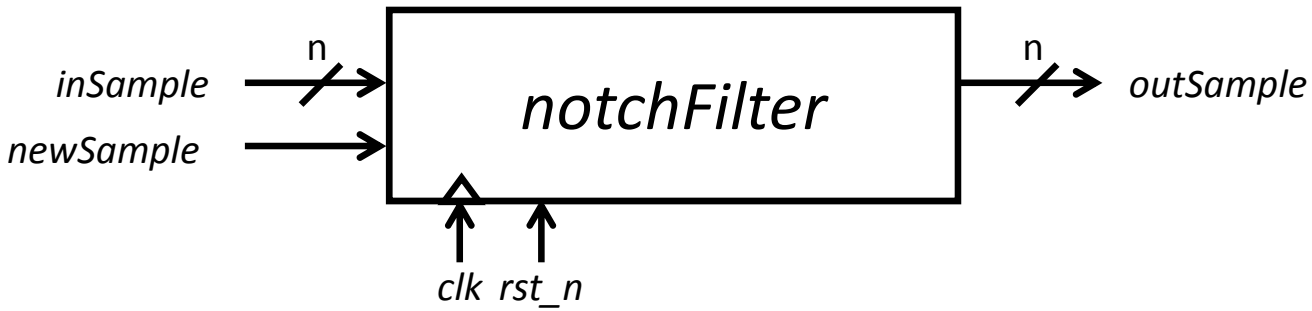
las constantes se convierten a punto fijo

```
function toFix( d: real; qn : natural; qm : natural ) return signed is
begin
  return to_signed( integer(d*(2.0**qm)), qn+qm );
end function;
```

Filtro Notch

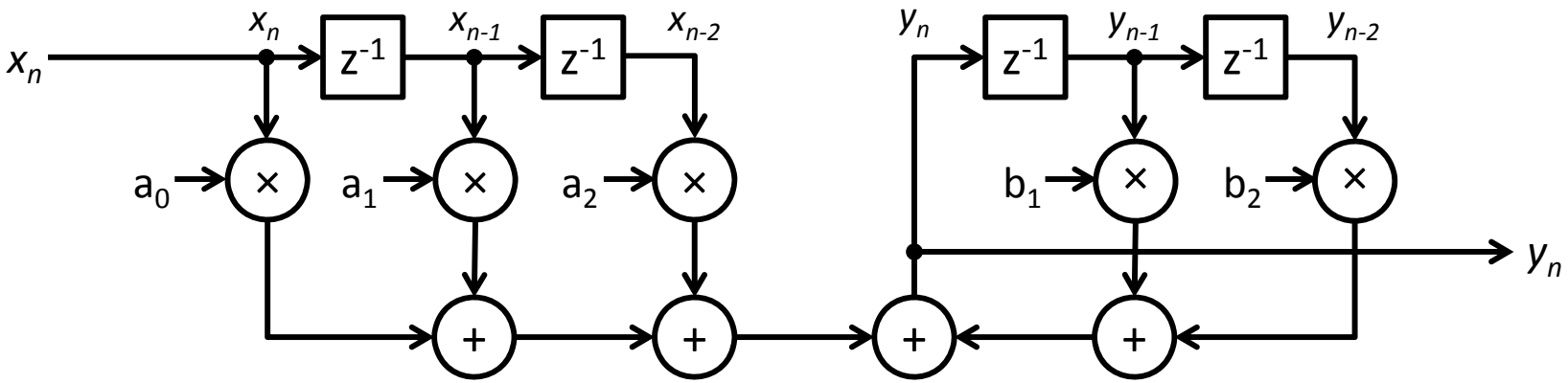
implementación

- Diseñar un notch filter de caracteristicas configurables:
  - Se implementará como un filtro IIR (infinite-impulse response) de 2do. orden.
  - Las muestras serán interpretadas como números reales codificados en punto fijo.
  - Serán configurables la anchura y formato de datos, la frecuencia de muestreo y la frecuencia de corte.
  - Por cada nueva muestra deberá activarse durante un ciclo la señal de strobe newSample.



Filtro Notch

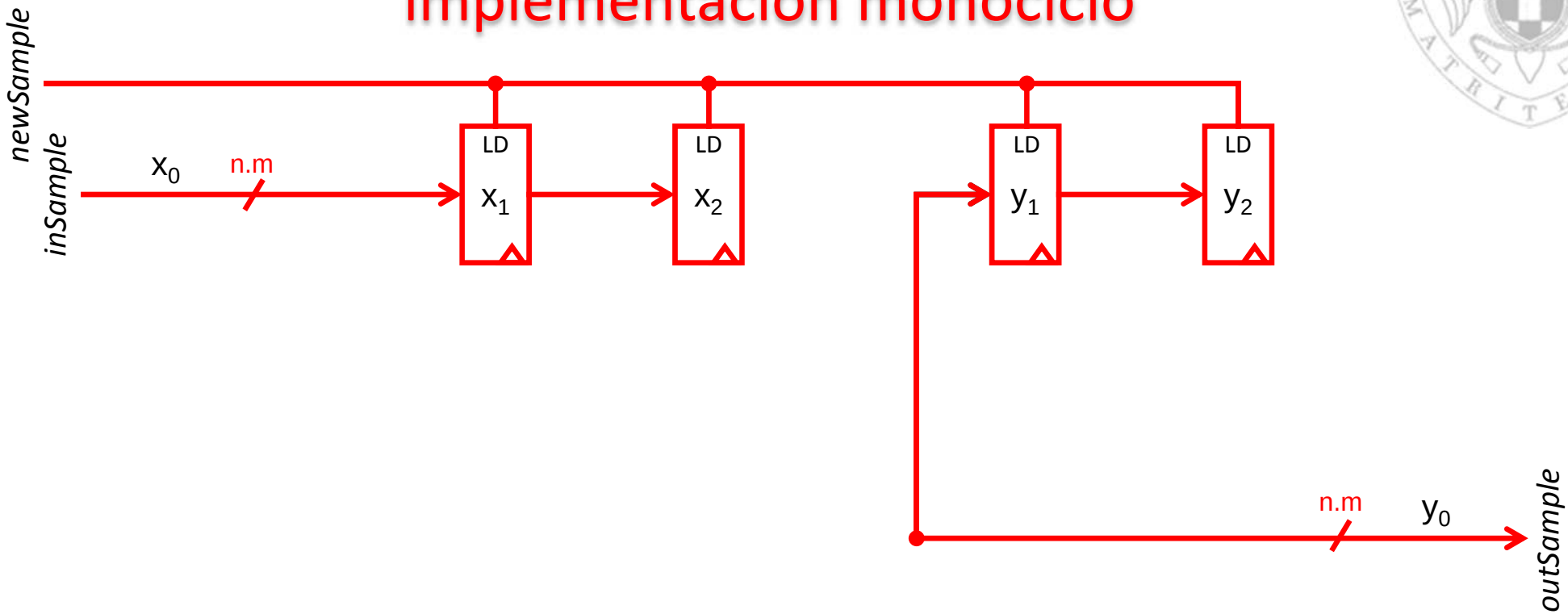
implementación monociclo

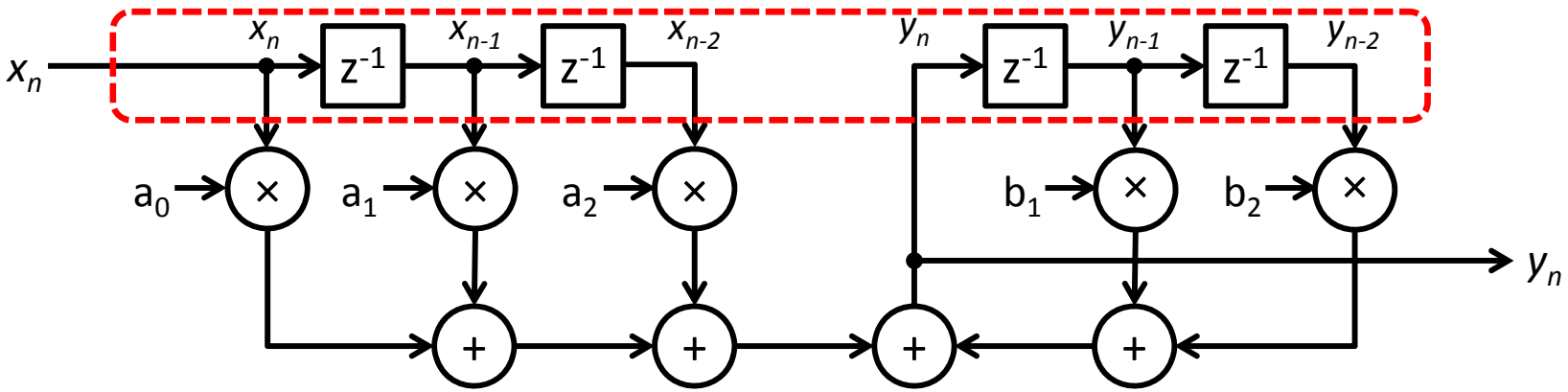


J.M. Mendías 2017

Filtro Notch

implementación monociclo

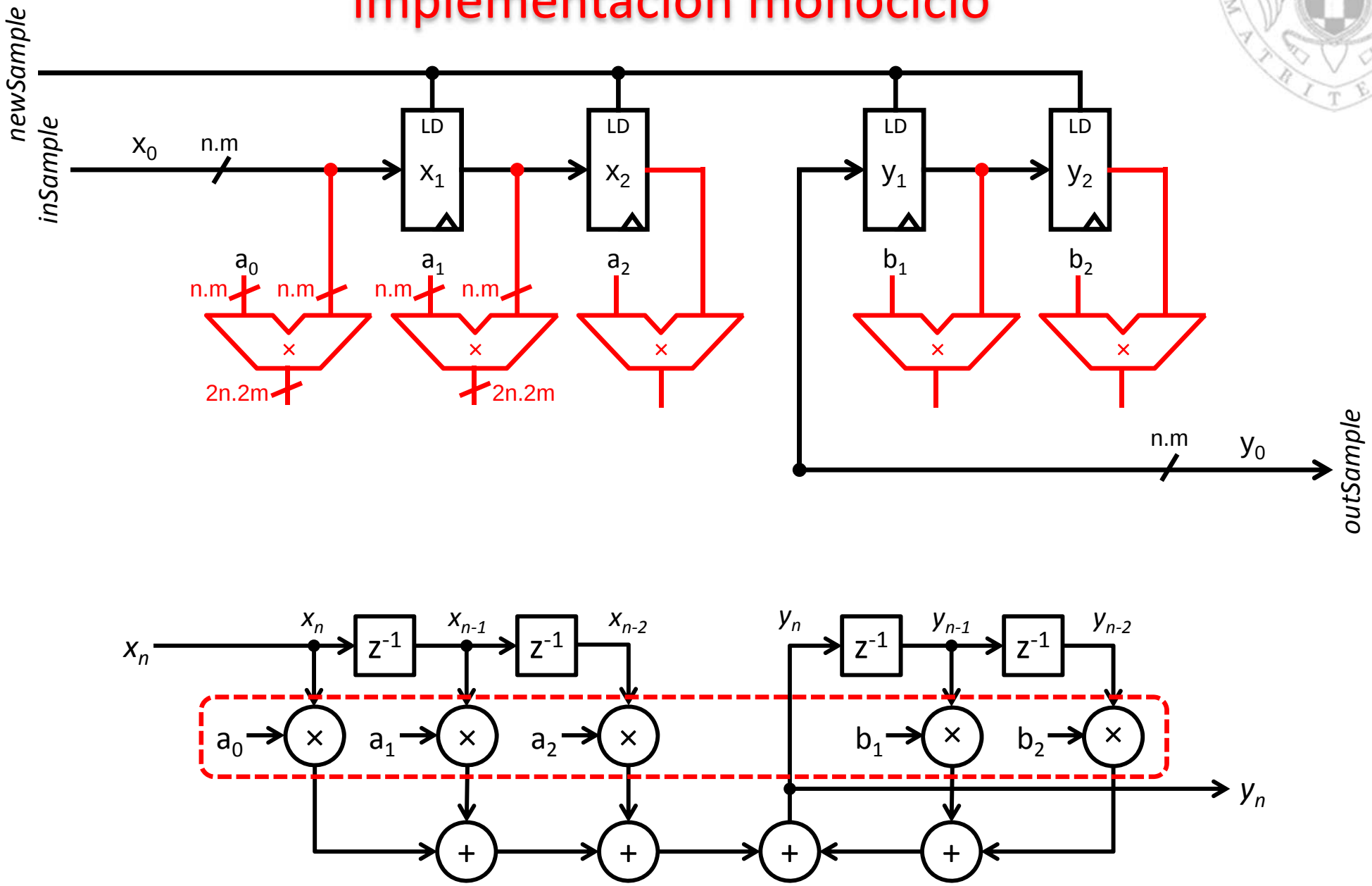




J.M. Mendías
2017

Filtro Notch

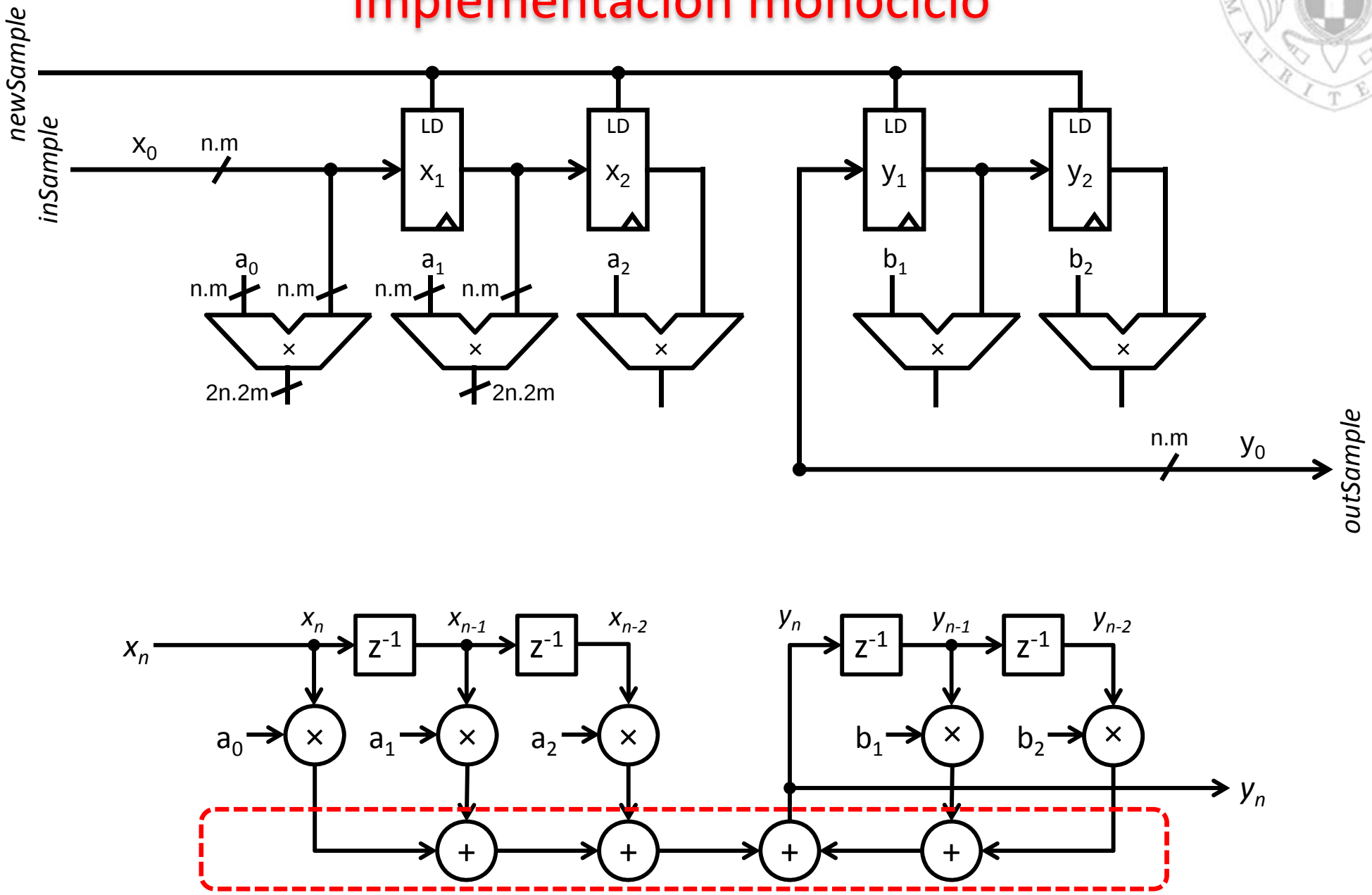
implementación monociclo



J.M. Mendías
2017

Filtro Notch

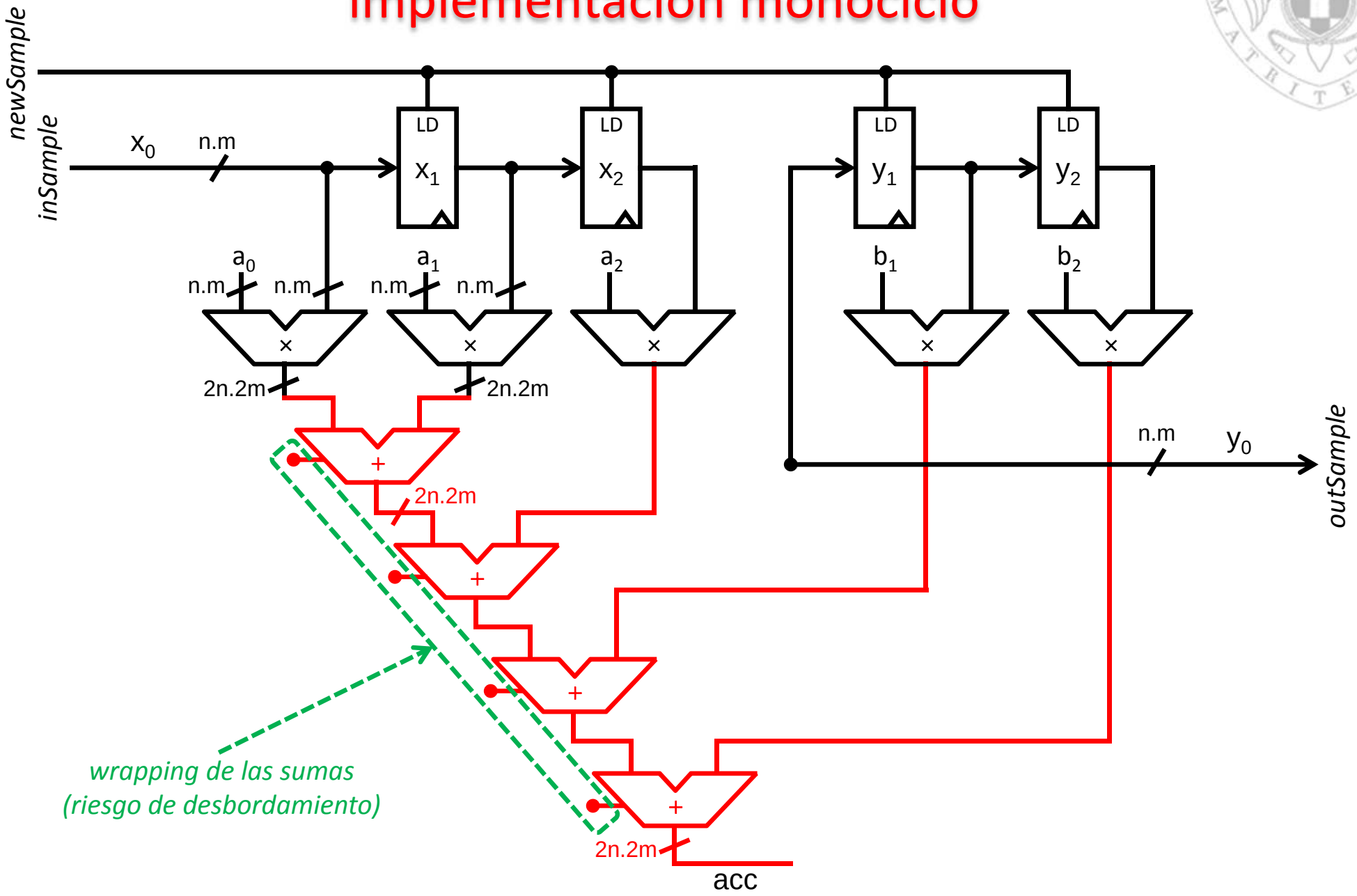
implementación monociclo



J.M. Mendías
2017

Filtro Notch

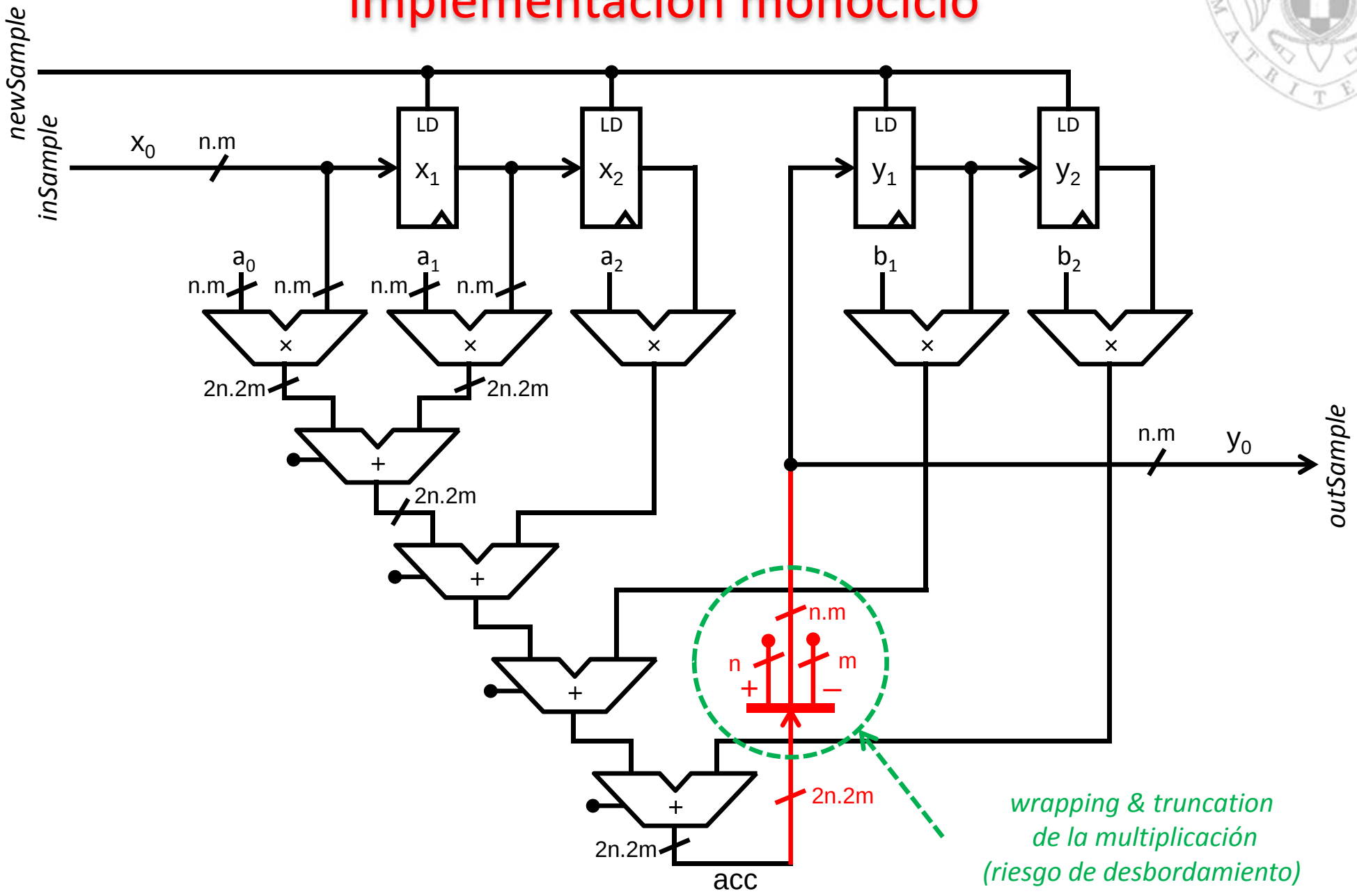
implementación monociclo



J.M. Mendías 2017

Filtro Notch

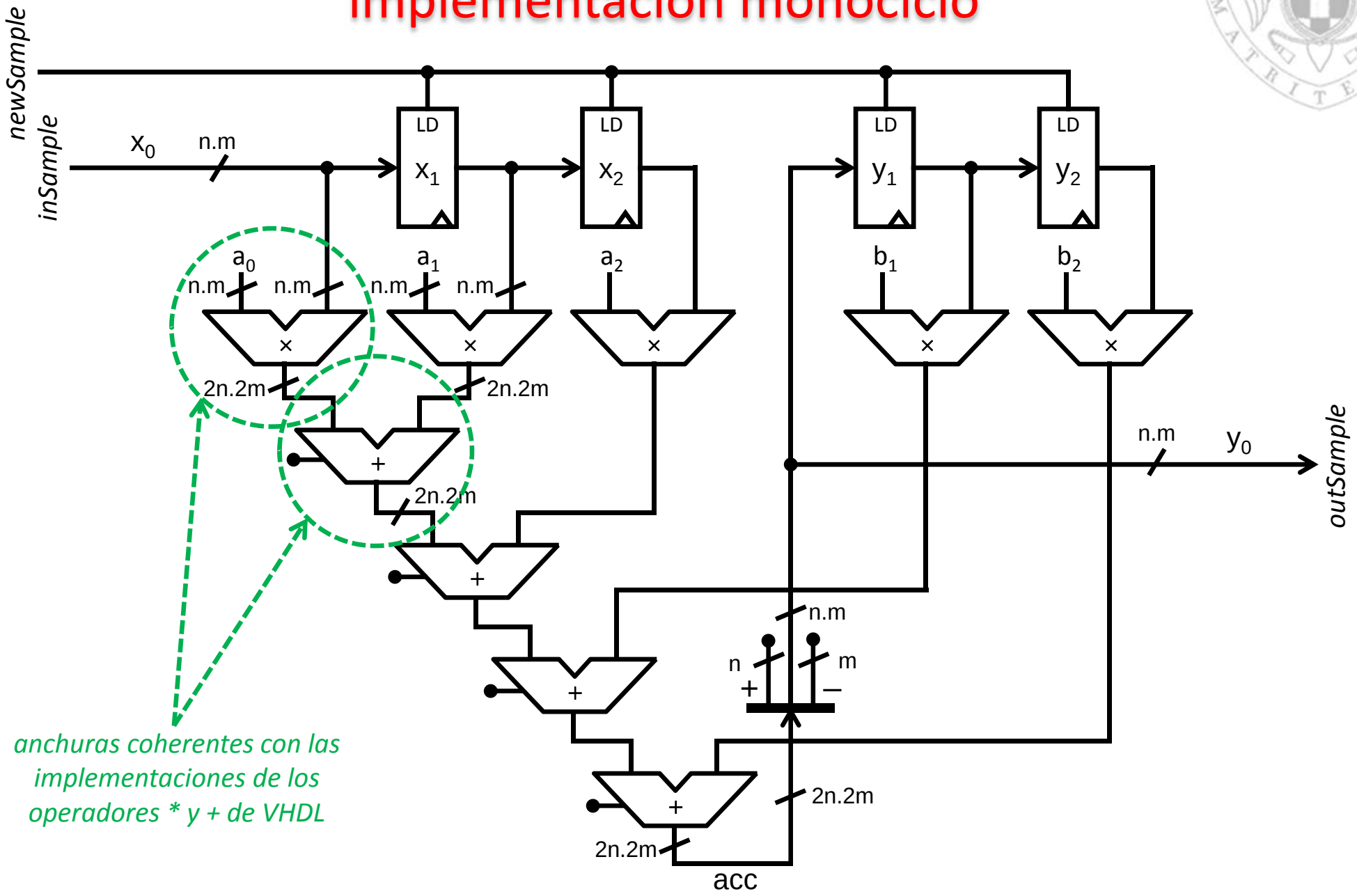
implementación monociclo



J.M. Mendías
2017

Filtro Notch

implementación monociclo



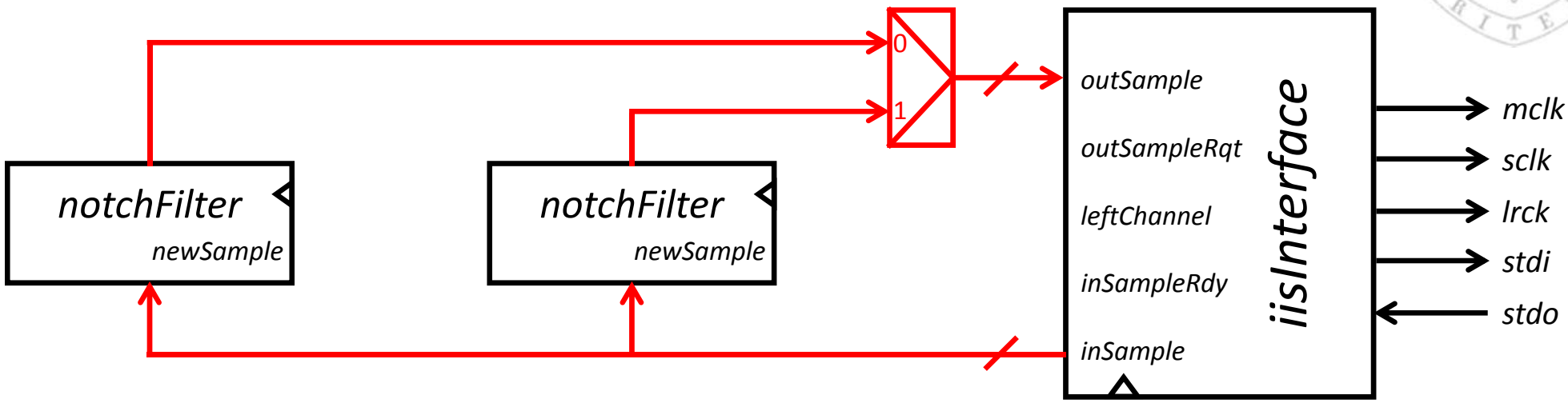
Filtro Notch

implementación monociclo: conexión estéreo

J.M. Mendías
2017

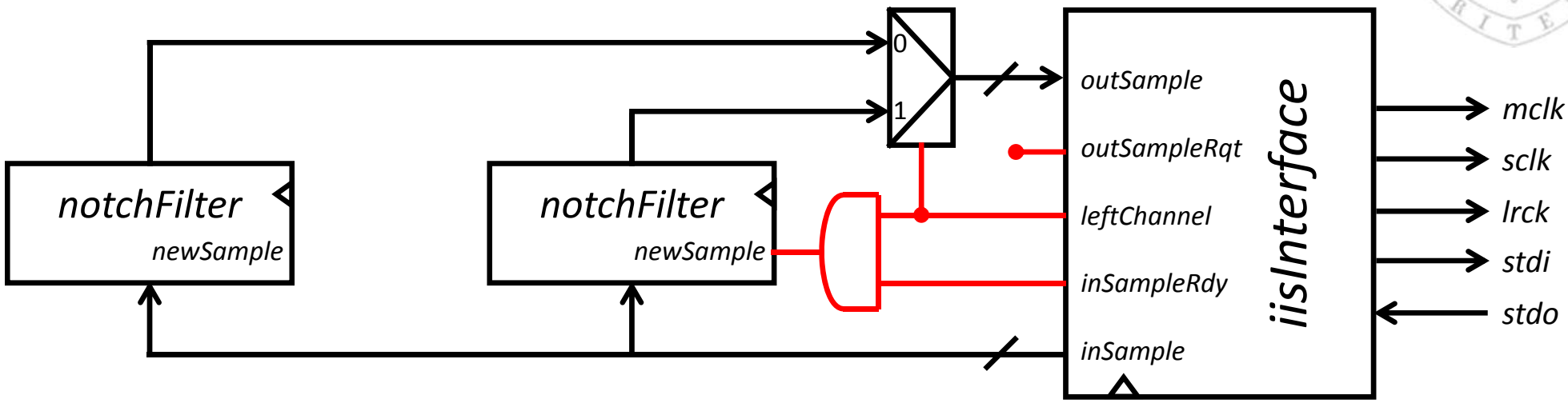
Filtro Notch

implementación monociclo: conexión estéreo



Filtro Notch

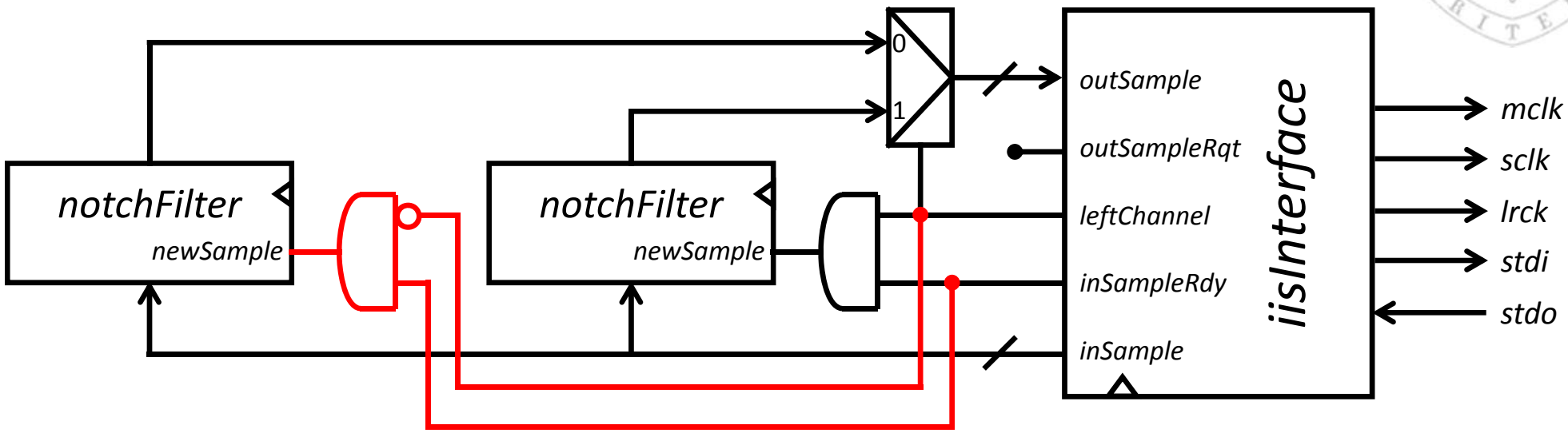
implementación monociclo: conexión estéreo



J.M. Mendías
2017

Filtro Notch

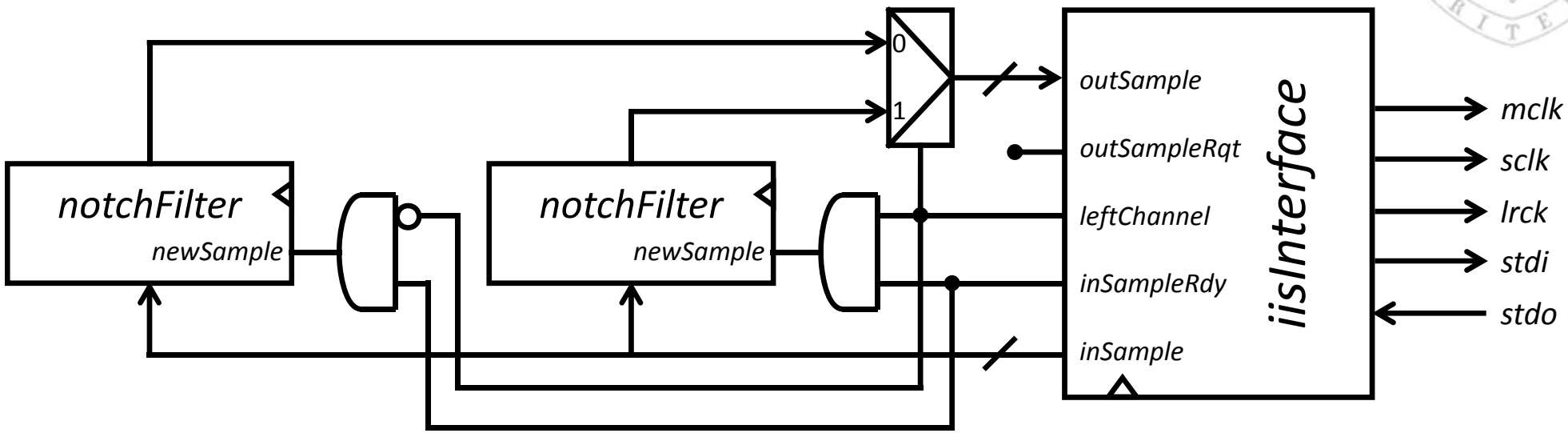
implementación monociclo: conexión estéreo

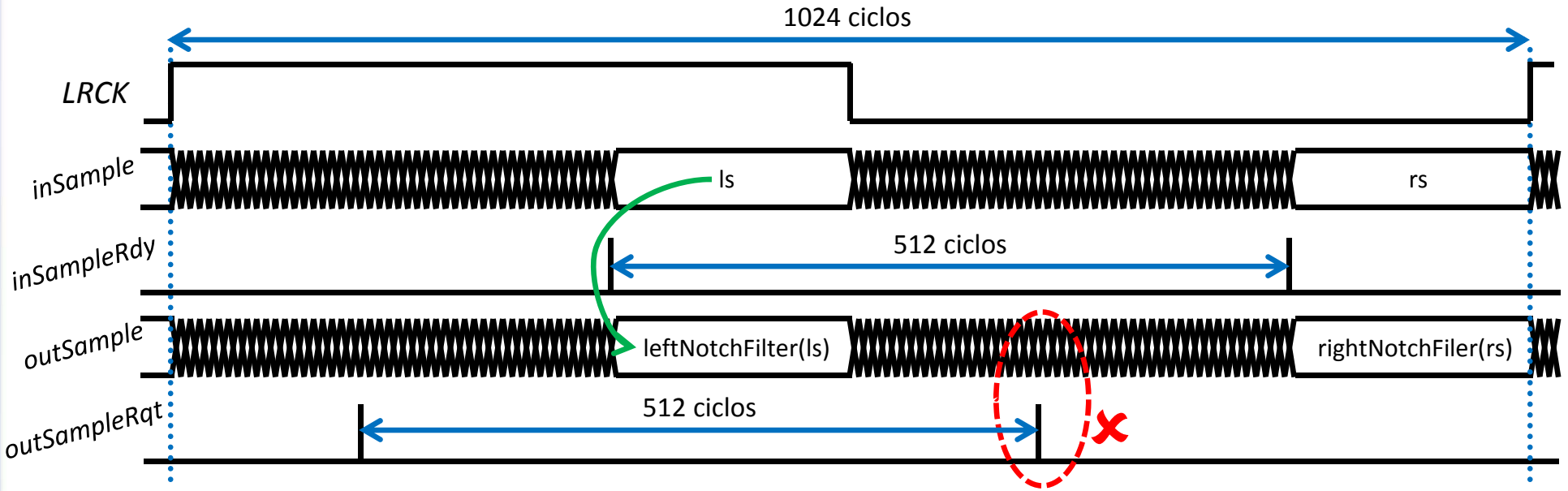


J.M. Mendías
2017

Filtro Notch

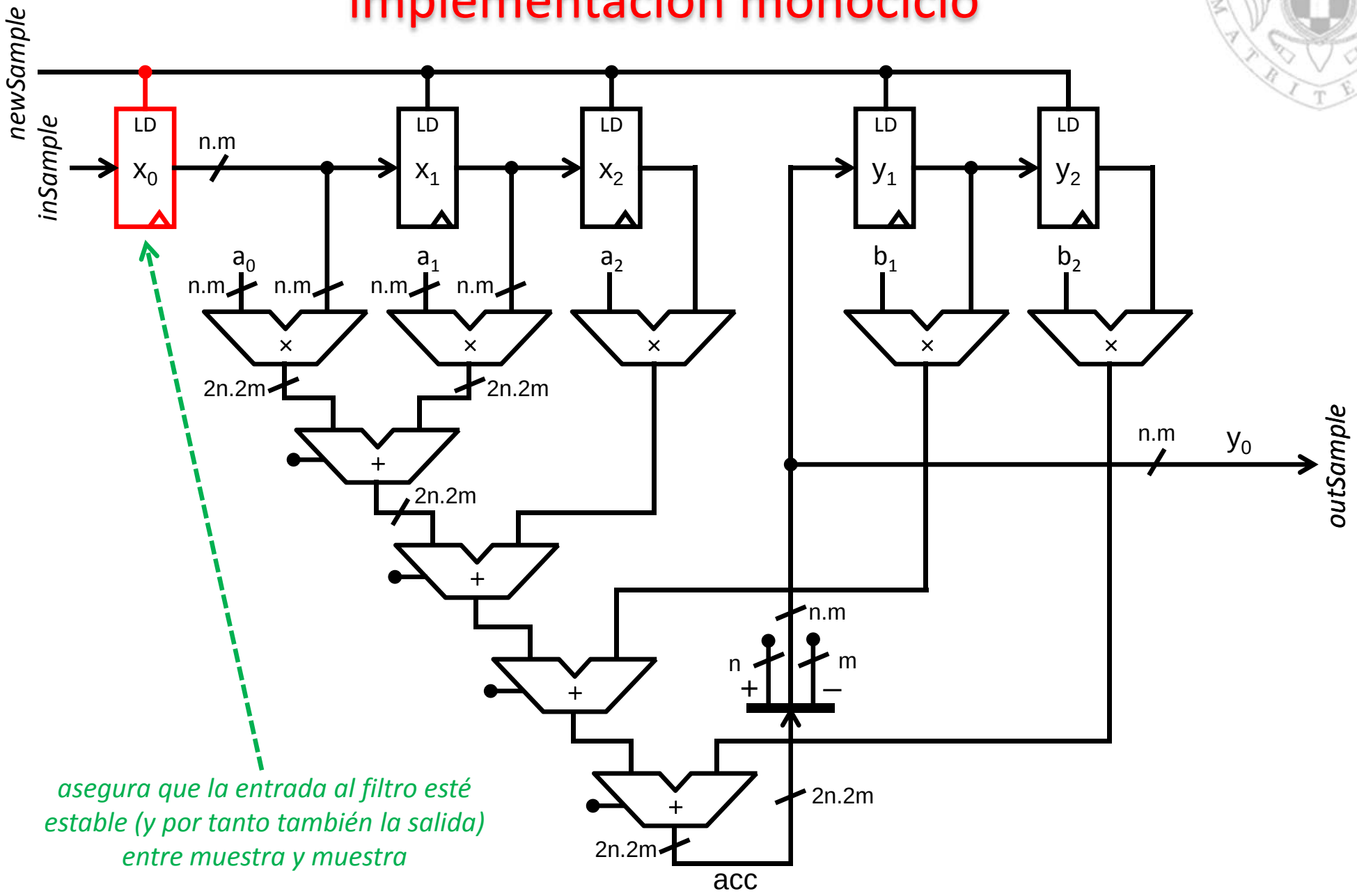
implementación monociclo: conexión estéreo





Filtro Notch

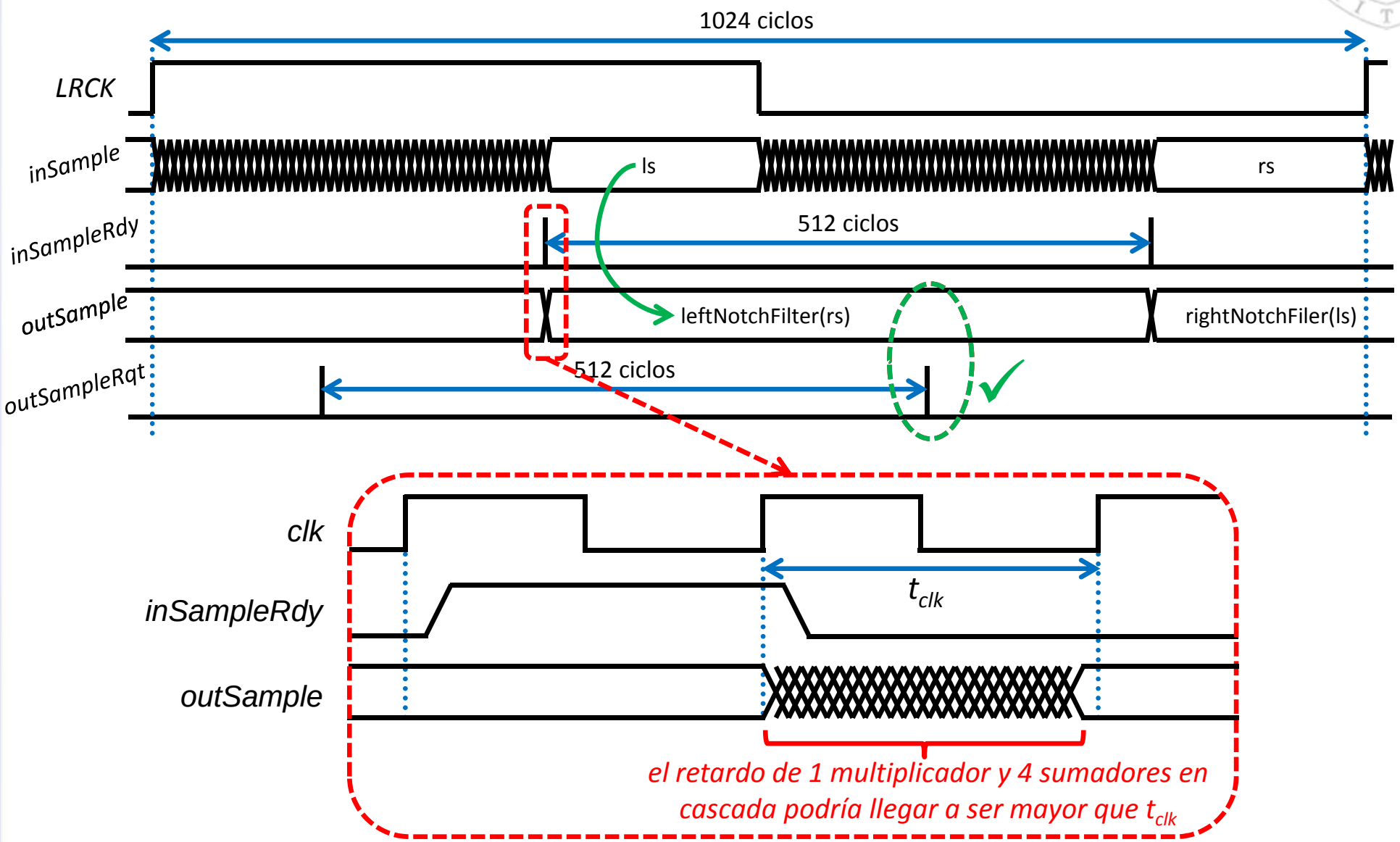
implementación monociclo



J.M. Mendías 2017

Filtro Notch

implementación monociclo: conexión estéreo



Filtro Notch

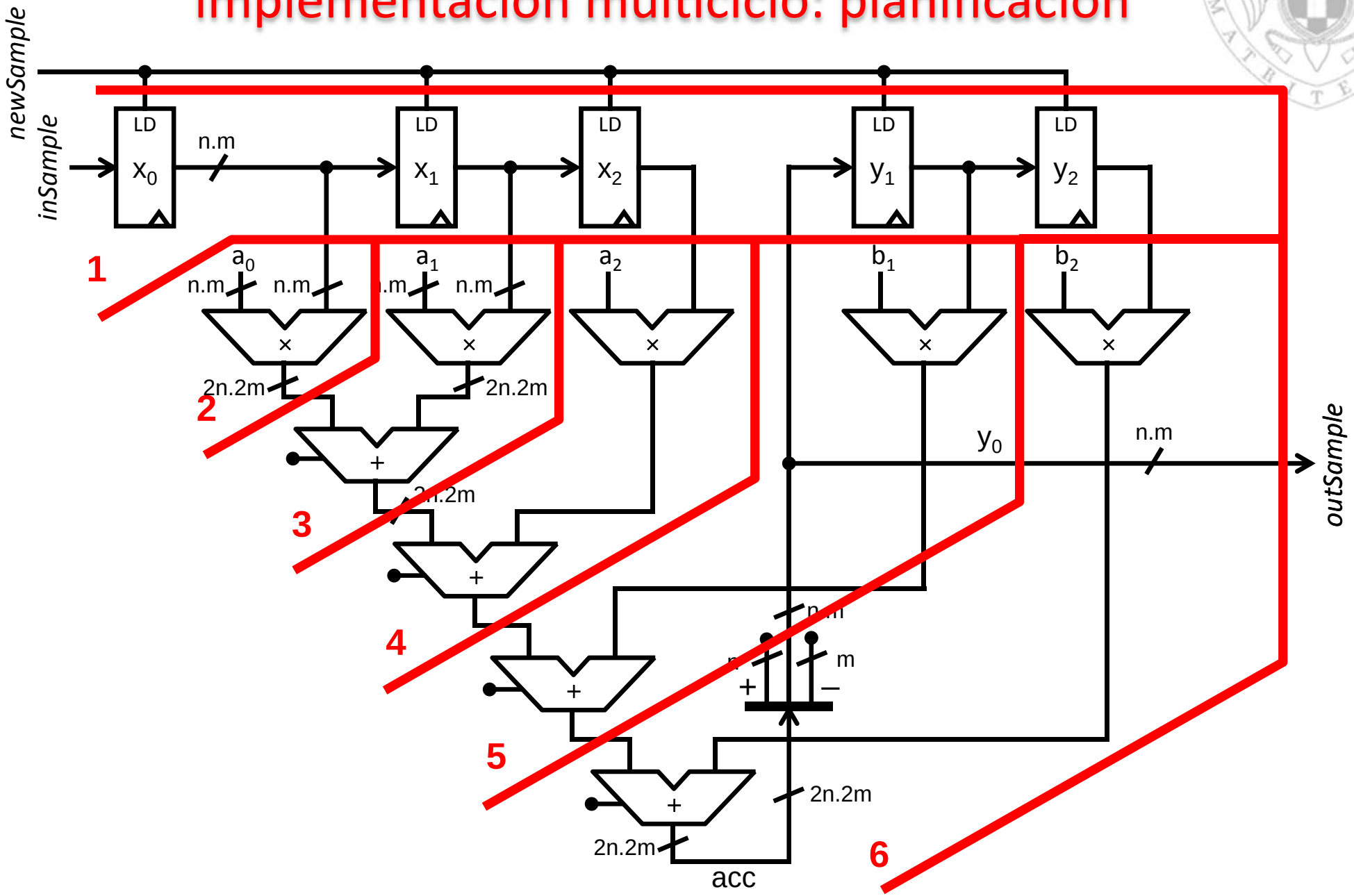
implementación monociclo: discusión

- La implementación monociclo del filtro es simple de especificar, pero:
  - Es cara: Requiere 5 multiplicadores y 4 sumadores.
    - Si aumenta en orden del filtro, el coste aumenta (2 multiplicadores y 2 sumadores por orden)
  - Es ineficiente: Realiza todos los cálculos en 1 ciclo de reloj aunque reciba una nueva muestra cada 1024 ciclos.
    - Si aumenta el orden del filtro, el retardo del camino crítico aumenta (el retardo de 2 sumadores por orden) pudiendo obligar a reducir la frecuencia de reloj para ser funcional.

- Cuando $f_{clk} \gg f_s$ y el número de recursos es limitado es mejor realizar implementaciones multiciclo:
  - Las operaciones se planifican para repartir su ejecución equilibradamente entre los ciclos disponibles entre muestra y muestra.
  - Los recursos se reutilizan para ejecutar en distintos ciclos más de una operación del algoritmo especificado.
  - Una FSM gobierna la ejecución ordenada de las operaciones en los recursos.

Filtro Notch

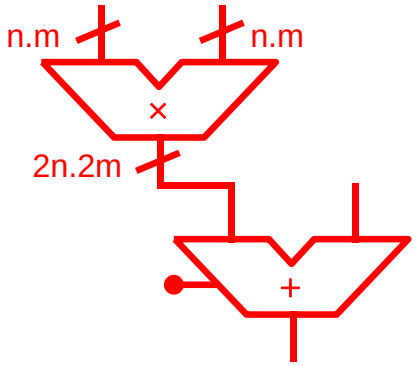
implementación multiciclo: planificación



J.M. Mendías
2017

Filtro Notch

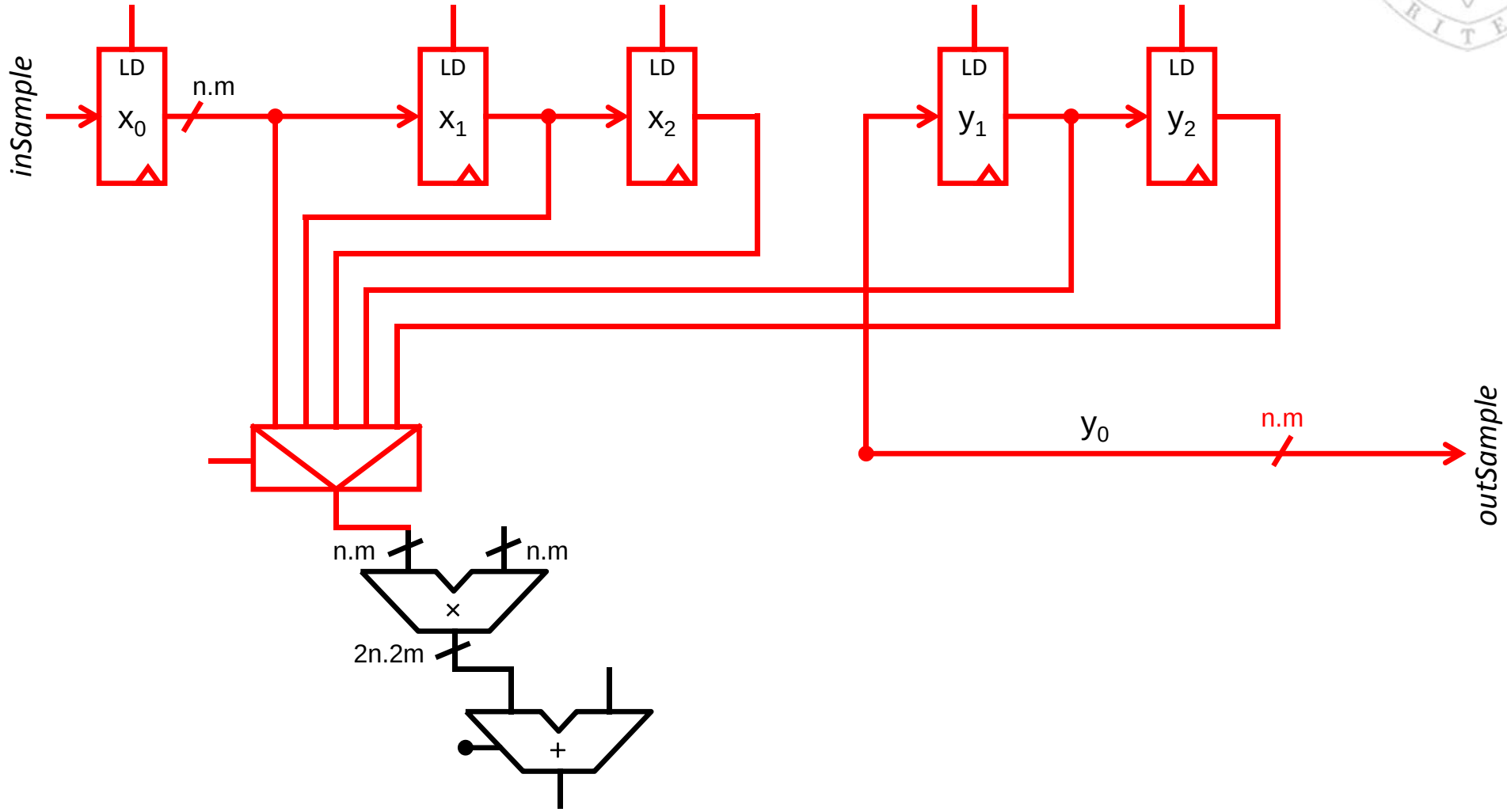
implementación multiciclo: ruta de datos



J.M. Mendías
2017

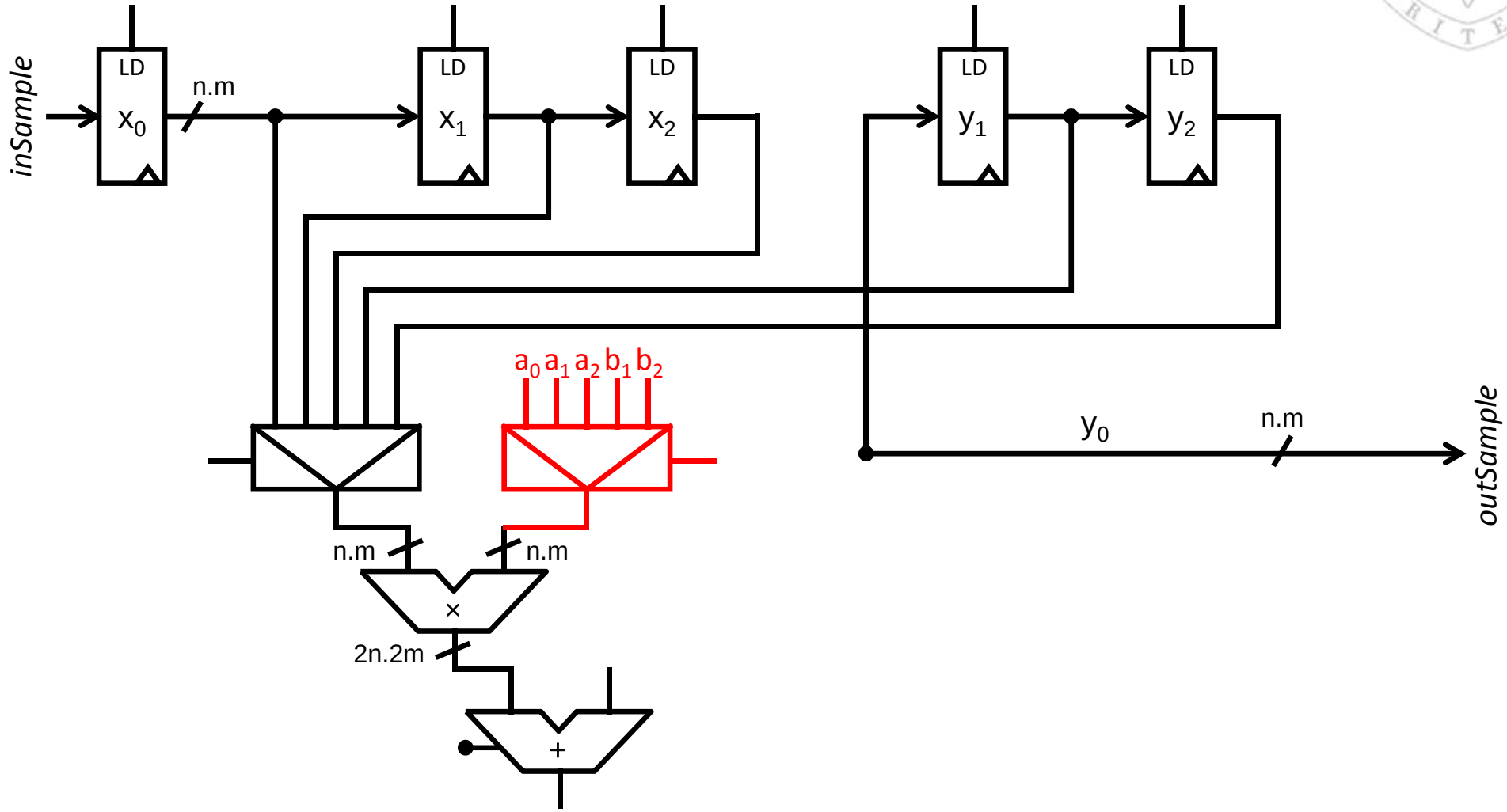
Filtro Notch

implementación multiciclo: ruta de datos



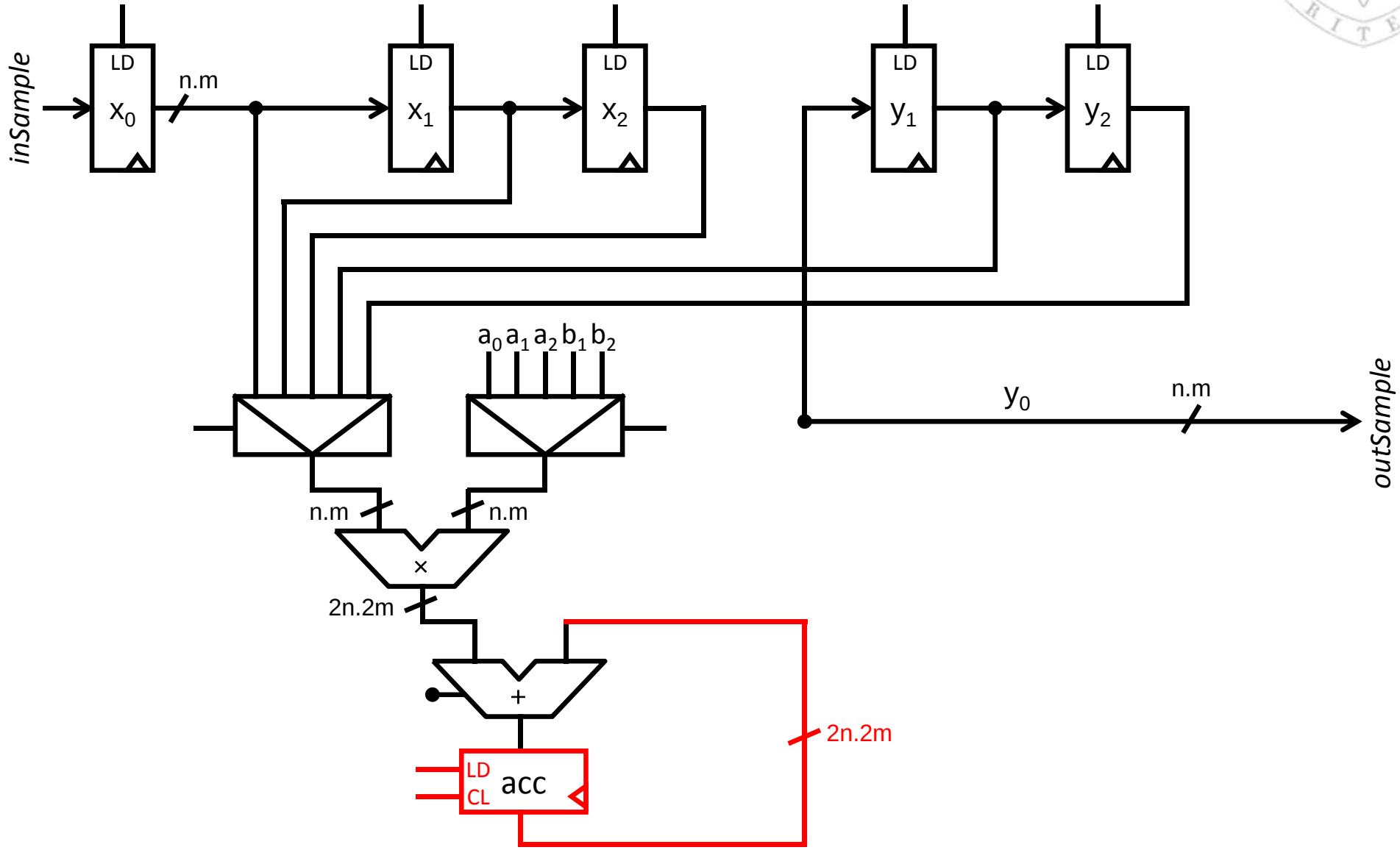
Filtro Notch

implementación multiciclo: ruta de datos



Filtro Notch

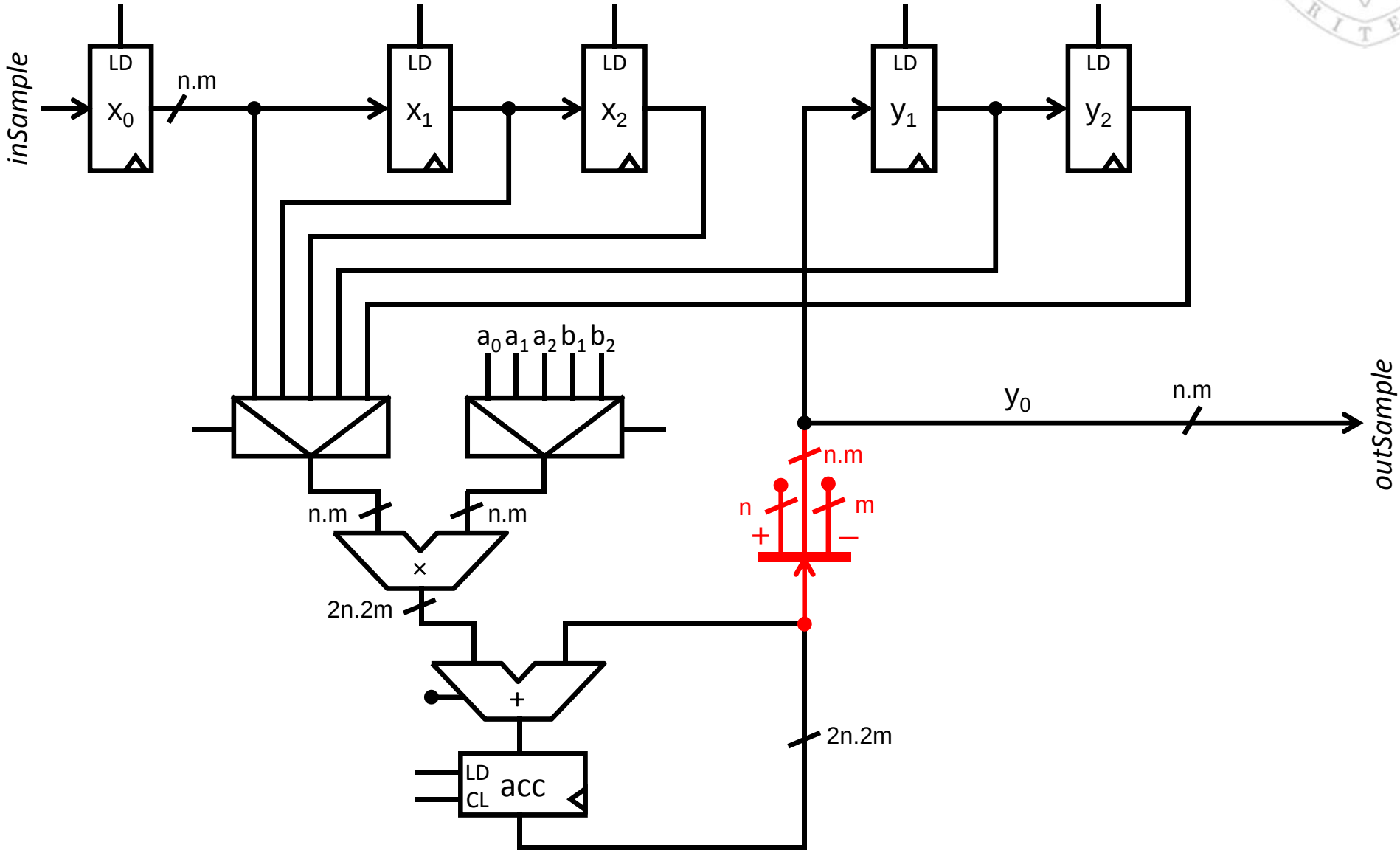
implementación multiciclo: ruta de datos



J.M. Mendías
2017

Filtro Notch

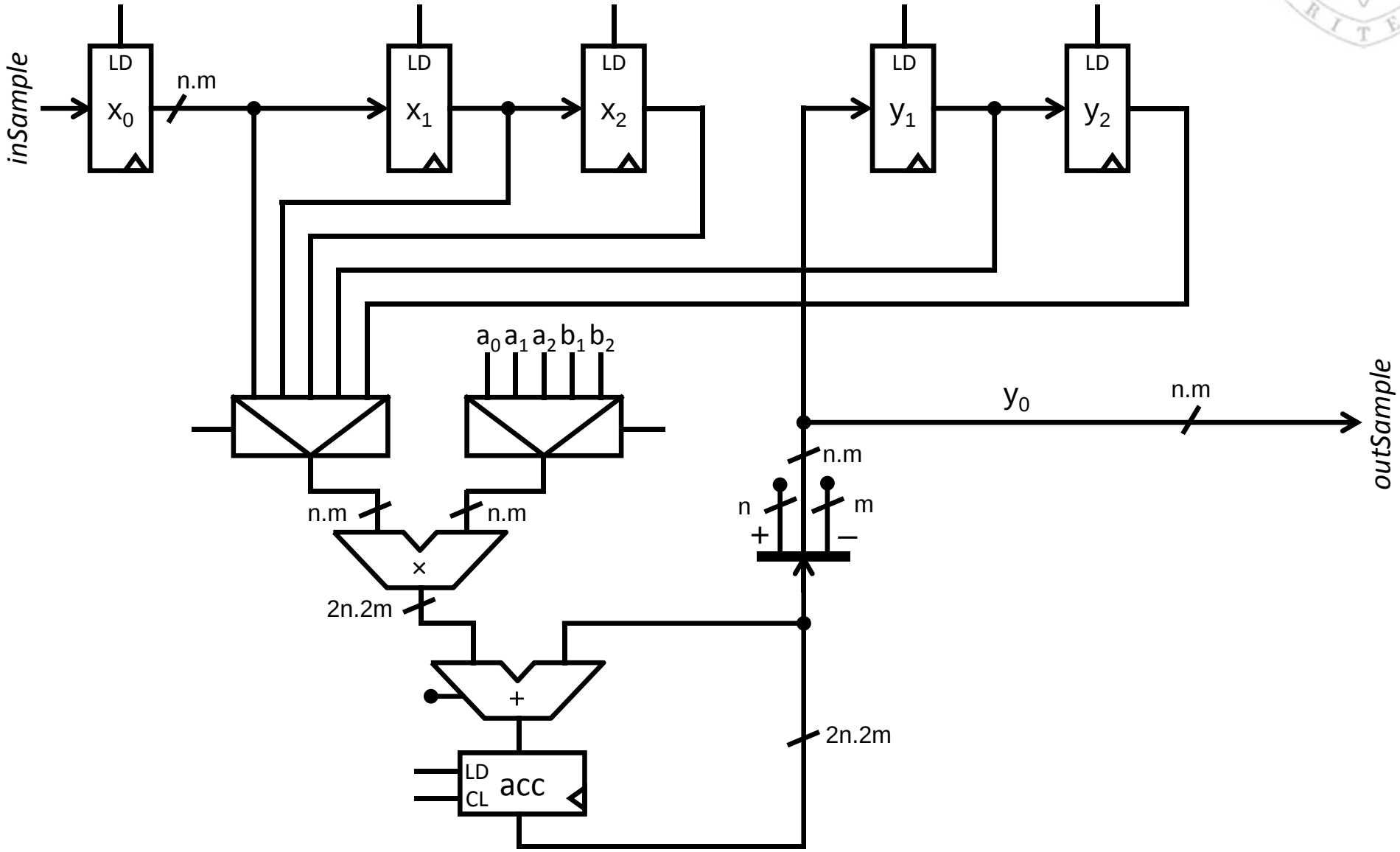
implementación multiciclo: ruta de datos



J.M. Mendías
2017

Filtro Notch

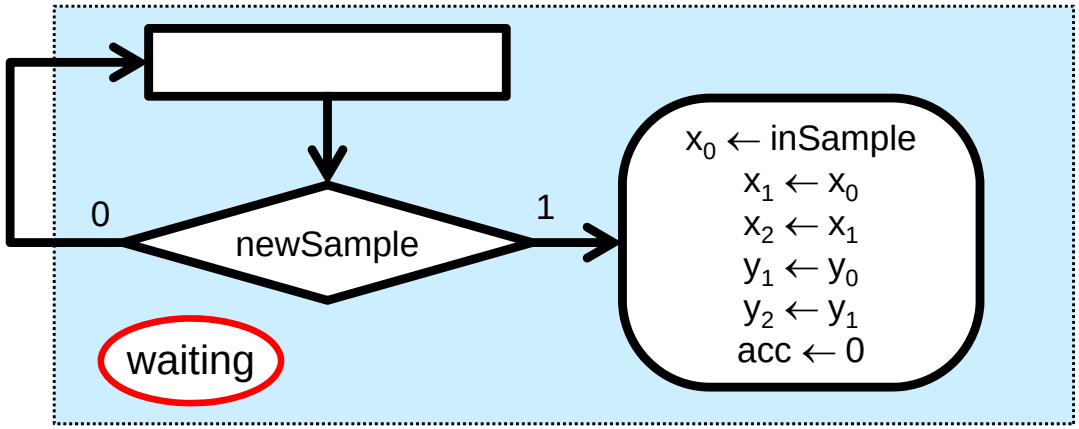
implementación multiciclo: ruta de datos



J.M. Mendías
2017

Filtro Notch

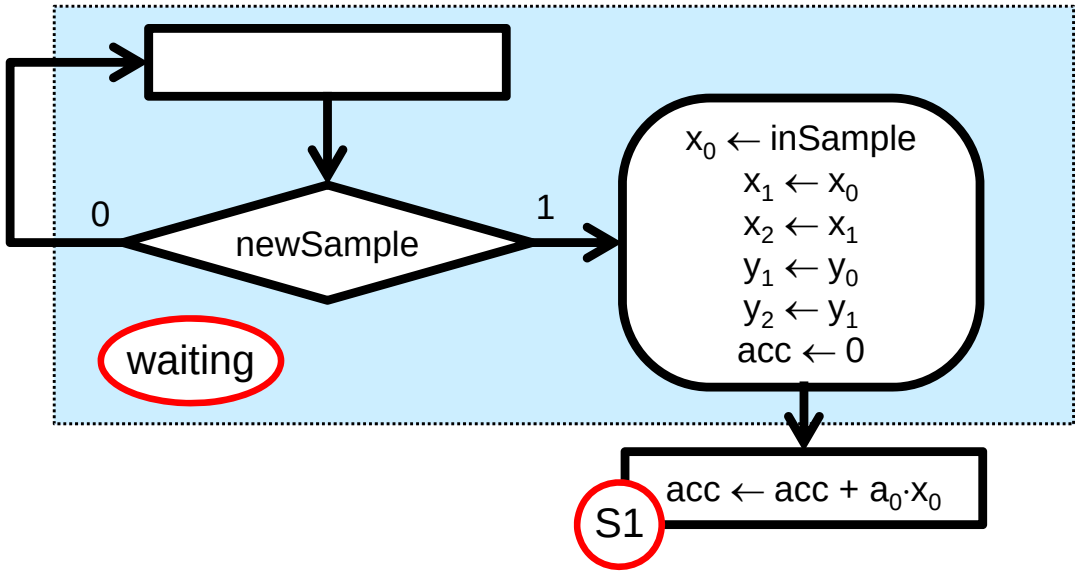
implementación multiciclo: controlador



J.M. Mendías
2017

Filtro Notch

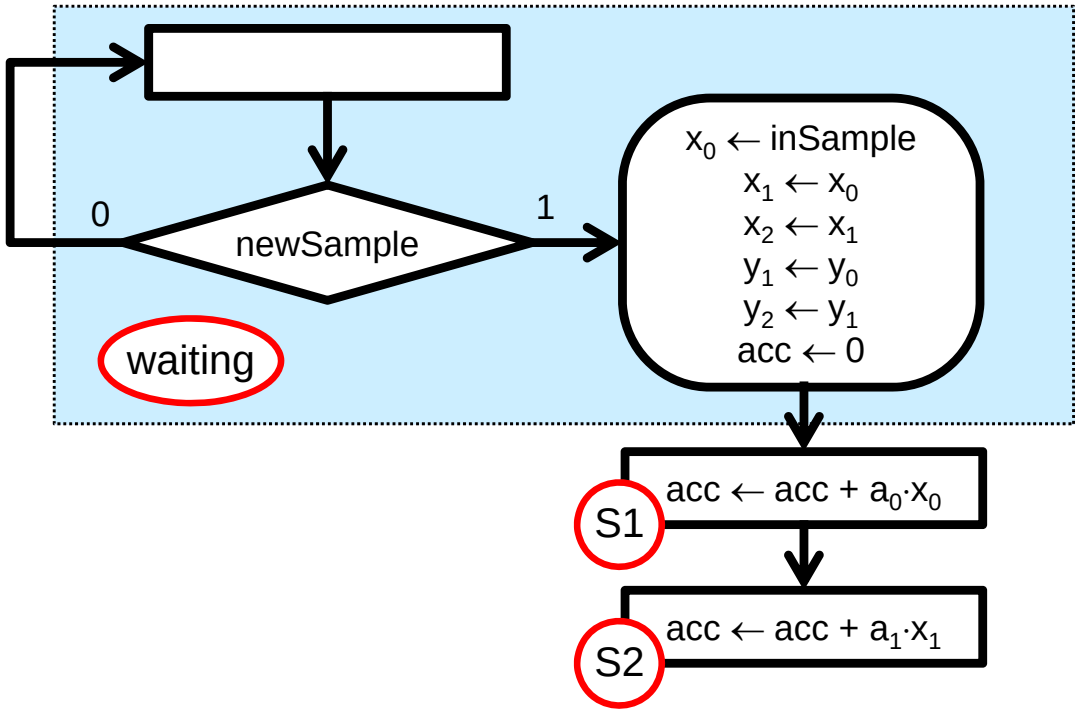
implementación multiciclo: controlador



J.M. Mendías
2017

Filtro Notch

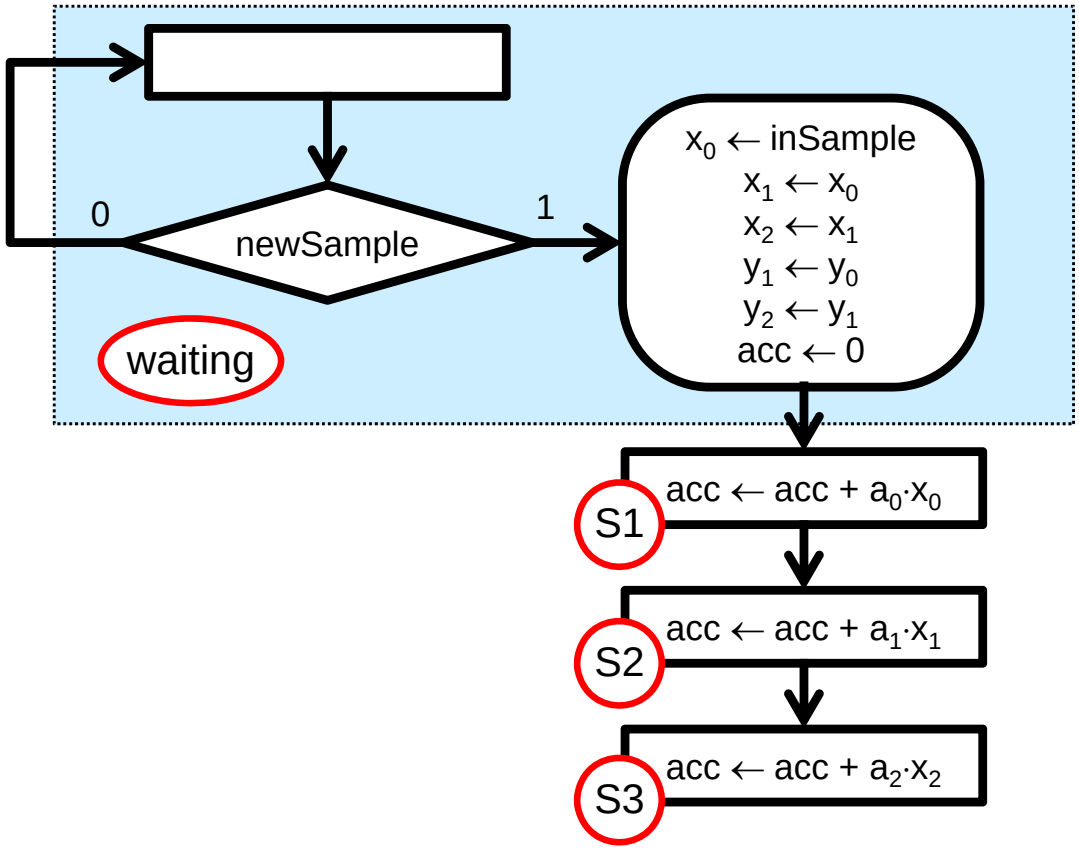
implementación multiciclo: controlador



J.M. Mendías
2017

Filtro Notch

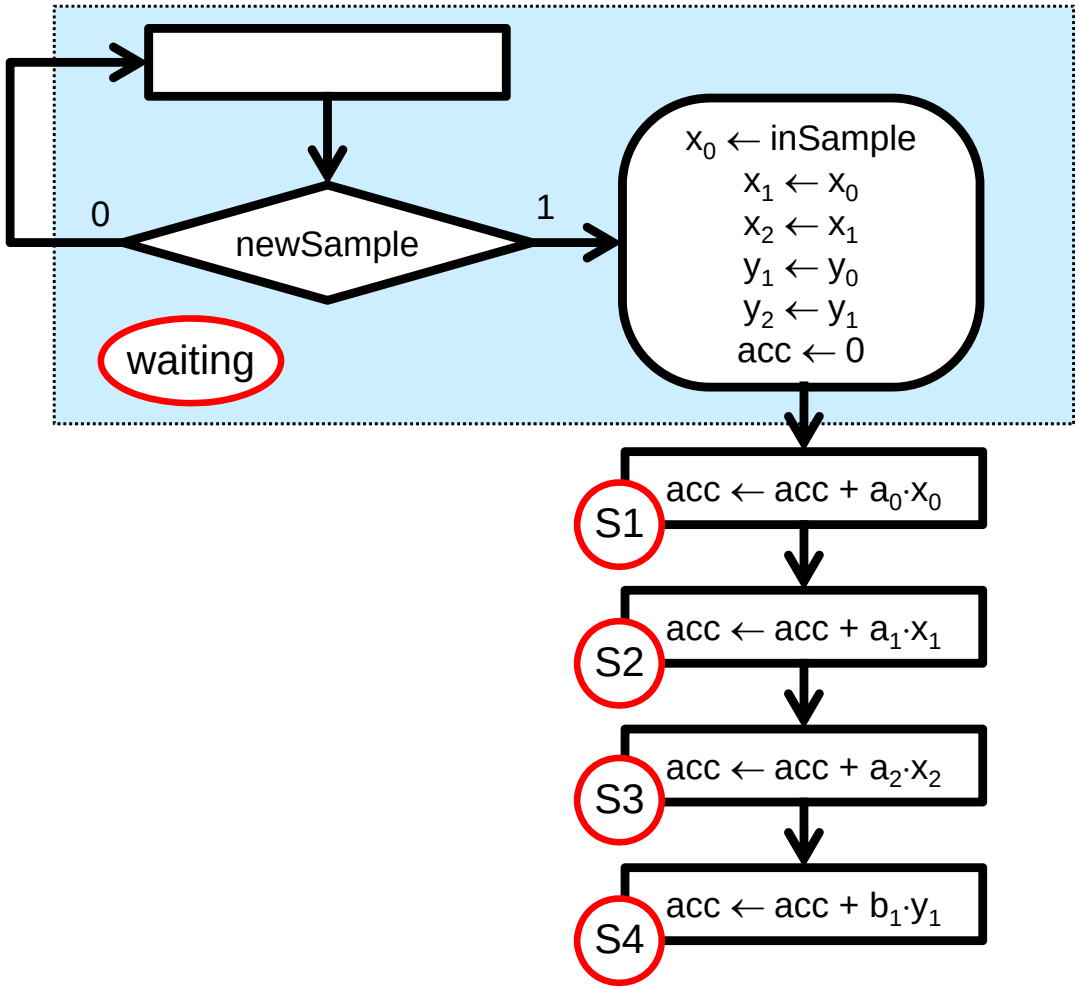
implementación multiciclo: controlador



J.M. Mendías
2017

Filtro Notch

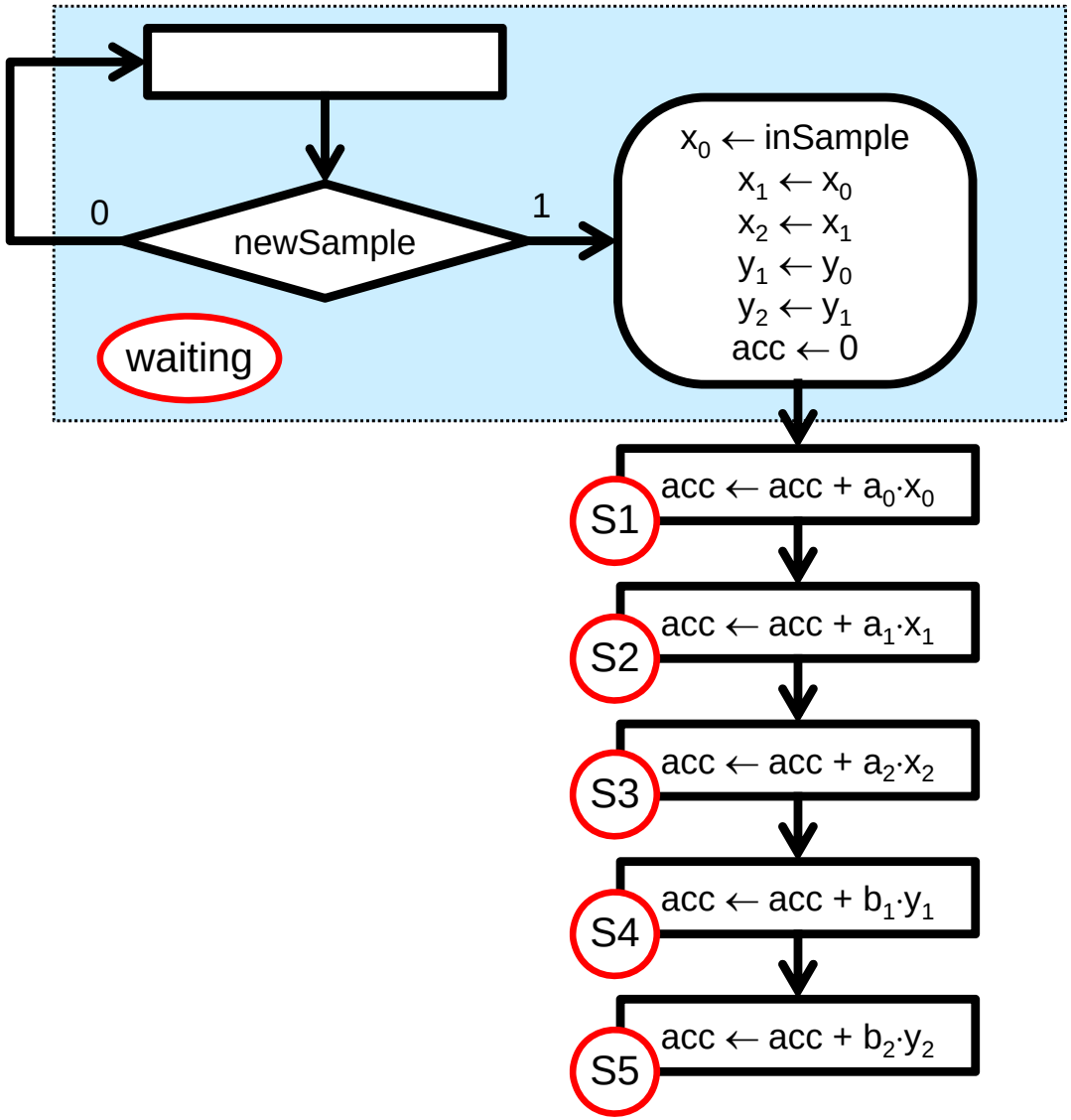
implementación multiciclo: controlador



J.M. Mendías
2017

Filtro Notch

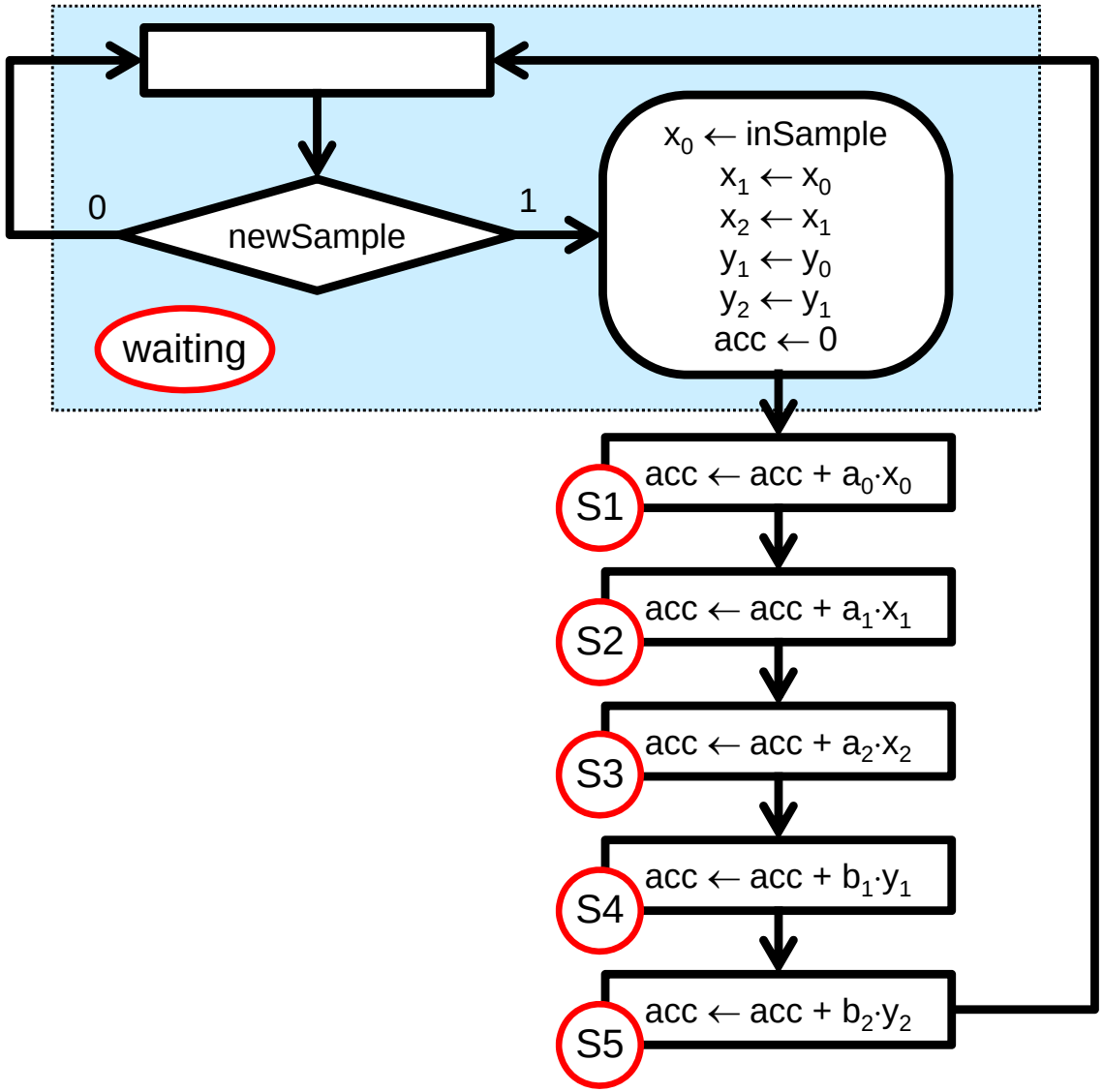
implementación multiciclo: controlador



J.M. Mendías
2017

Filtro Notch

implementación multiciclo: controlador



Filtro Notch

implementación monociclo: discusión

- La implementación multicliclo del filtro es más compleja de especificar, sin embargo:
  - Es barata: Requiere 1 multiplicador y 1 sumador.
    - Si aumenta en orden del filtro, el coste permanece casi fijo (aumenta solamente el coste de añadir estados a la FSM y de ampliar los multiplexores)
  - Es eficiente: Realiza todos los cálculos en varios ciclos de reloj (hasta un máximo de 1024 ciclos).
    - Si aumenta el orden del filtro, el retardo del camino crítico permanece casi fijo (aumenta solamente el retardo de ampliar los multiplexores).
  - Existen herramientas de diseño que automatizan este proceso (aunque con Xilinx ISE todo haya que hacerlo a mano).
    - Obteniendo trade-offs entre el coste del circuito y su latencia (número de ciclos en que computa el algoritmo).

Tareas

implementación monociclo

1. Crear el proyecto **`lab8monocycle`** en el directorio **`DAS`**
2. Descargar de la Web en el directorio **`common`** el fichero:
   - **`iisInterface.vhd`** y **`test-800Hz.wav`**
3. Descargar de la Web en el directorio **`lab8monocycle`** los ficheros:
   - **`monocycleNotchFilter.vhd`**, **`lab8monocycle.vhd`** y **`lab8monocycle.ucf`**
4. Completar el fichero **`common.vhd`** con la declaración del nuevo componente reusable.
5. Completar el código omitido en el ficheros:
   - **`iisInterface.vhd`** y **`monocycleNotchFilter.vhd`**
6. Añadir al proyecto los ficheros:
   - **`common.vhd`**, **`synchronizer.vhd`**, **`debouncer.vhd`**, **`issInterface.vhd`**, **`monocycleNotchFilter.vhd`**, **`lab8monocycle.vhd`** y **`lab8monocycle.ucf`**
7. Sintetizar, implementar y generar el fichero de configuración.
8. Conectar el cable de audio, unos auriculares, arrancar la reproducción continua del audio **`test-800Hz.wav`** en el PC y encender la placa.
9. Descargar el fichero **`lab8multicycle.bit`**
   - Si desborda, disminuir en el PC el volumen de reproducción del audio.

J.M. Mendías
2017

Tareas

implementación multiciclo

1. Crear el proyecto **`lab8multicycle`** en el directorio **`DAS`**
2. Descargar de la Web en el directorio **`lab8multicycle`** los ficheros:
   - **`multicycleNotchFilter.vhd`**, **`lab8multicycle.vhd`** y **`lab8multicycle.ucf`**
3. Completar el código omitido en el fichero:
   - **`multicycleNotchFilter.vhd`**
4. Añadir al proyecto los ficheros:
   - **`common.vhd`**, **`synchronizer.vhd`**, **`debouncer.vhd`**, **`issInterface.vhd`**, **`multicycleNotchFilter.vhd`**, **`lab8multicycle.vhd`** y **`lab8multicycle.ucf`**
5. Sintetizar, implementar y generar el fichero de configuración.
6. Conectar el cable de audio, unos auriculares, arrancar la reproducción continua del audio **`test-800Hz.wav`** en el PC y encender la placa.
7. Descargar el fichero **`lab8multicycle.bit`**
   - Si desborda, disminuir en el PC el volumen de reproducción del audio.

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/