Módulo 3:
División de frecuencias

Contenidos del módulo 3

- Implementación de circuitos de división de frecuencia, con relaciones:
  - 1/N
  - N/M (M fijo, 10 o 16)
  - Otros casos (*dual modulus*).
  - N/M con cualquier N y M enteros
- Análisis de distribución del ruido de fase en divisores por relaciones no enteras.

Divisores 1/M

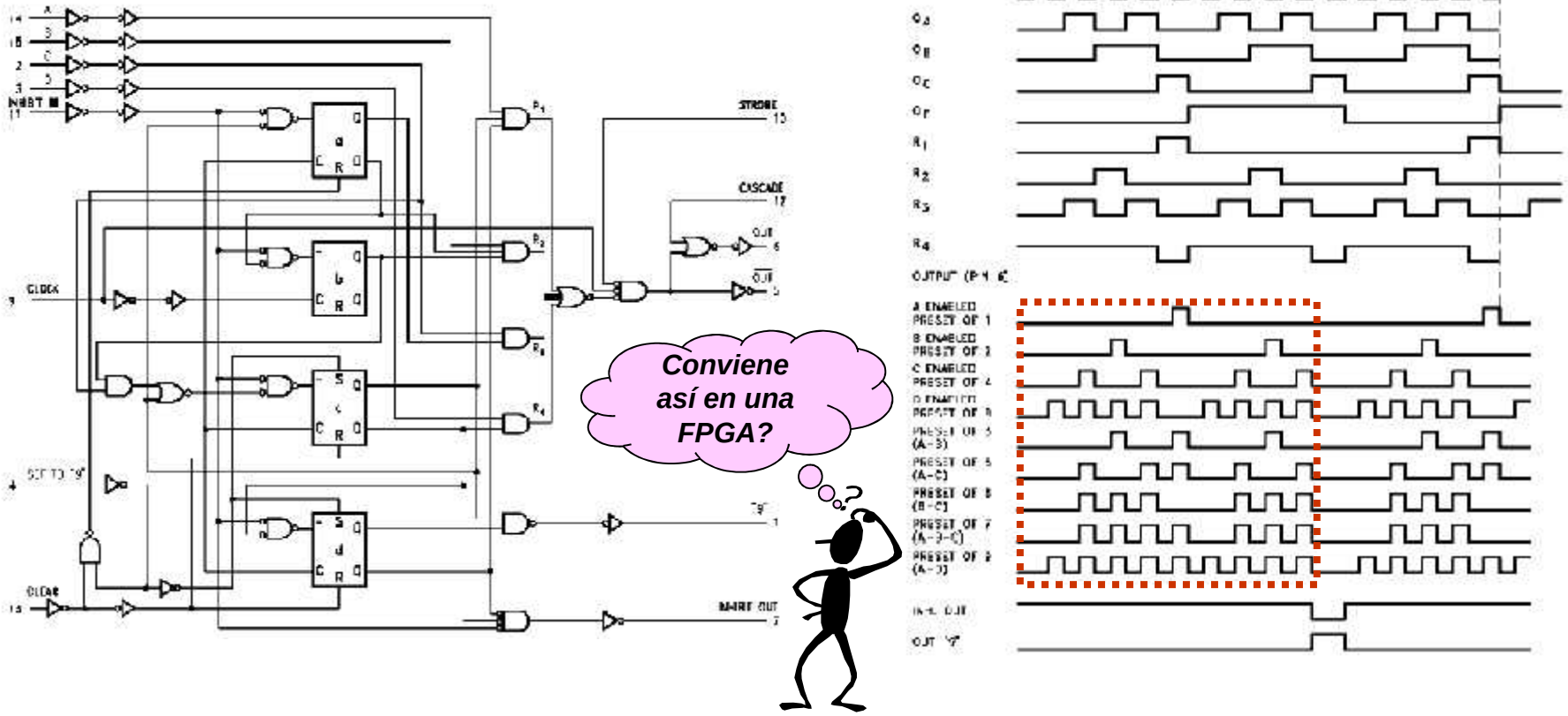
- Se basan en contadores que reciclan cada M pulsos de reloj
- De usar contadores binarios con $M=2^N$ el bit MSB tiene una frecuencia 1/M respecto a la frecuencia de entrada y relación de trabajo 50%
- Si $M<>2^N$ la relación de trabajo puede no ser 50%, aunque si M es par un divisor M/2 seguido de uno por 2 garantiza una relación de trabajo del 50%.
- Dado que el factor de división siempre el mismo, la frecuencia de la señal de salida es igualmente estable que la frecuencia de la señal de entrada.
- Para una frecuencia de entrada **F_{in}** el valor de frecuencia más próximo a **F_{in}** a la salida de un divisor entero es **$F_{out}=F_{in}/2$**, luego **$F_{out}=F_{in}/3$**, etcetera. Para bajos valores de **M** el salto de frecuencia **ΔF_{out}** entre sucesivos valores de **M** es muy grande, y **ΔF_{out}** disminuye a medida que **M** crece. Si se realizan gráficos de **$\Delta F_{out}=f(M)$** o de **$F_{out}=f(M)$** éstos son hiperbólicos.

Divisores N/M: los “rate multipliers”

- Un “***rate multiplier***” es un circuito sincrónico que permite obtener una frecuencia de salida proporcional a la frecuencia de entrada **F_{in}** pero que varía linealmente en función de **N**, es decir **F_{out} = Fin x N / M**, para un numerador variable **N**=0,1,..,M-1, y un divisor **M** fijo.
- La división de **F_{in}** se hace insertando o quitando pulsos para mantener un ritmo de pulsos de salida *lo más uniforme posible* (*esta es una desventaja respecto a los divisores enteros y parcialmente similar a lo que sucede en los divisores fraccionarios de doble módulo*)
- Es usual encontrar ***rate multiplers*** donde **$M=10^K$** llamados ***decimal rate multipliers*** y otros donde **$M=2^K$** llamados ***binary rate multipliers***. Por ejemplo:
  - CD4089 o SN7497 *cascadeable Binary Rate Multiplier*
  - CD4527 o SN74167 *cascadeable Decimal Rate Multiplier*
- En un CD4089 (M=16), para una dada **Fin** la **Fout** más próxima es **(15/16).Fin**, luego **(14/16).Fin**, luego **(13/16).Fin**, etc ... es decir un control lineal de la frecuencia
- En muchas aplicaciones digitales este “clock de período desparejo” no genera ningún problema, en tanto Fmax del circuito sea igual o mayor a Fin

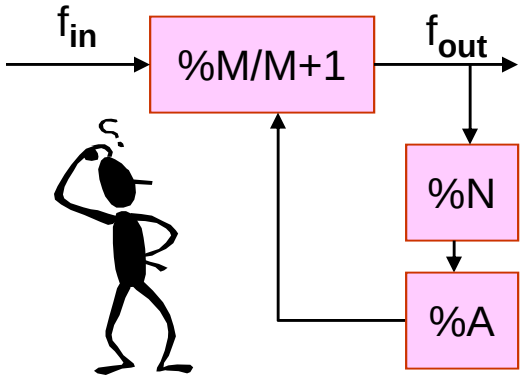
Divisores mediante "rate multipliers": el CD4527

- La figura muestra la arquitectura interna de un CD4527 y los circuitos adicionales para relizar su conexión en cascada. La figura de la derecha muestra cómo en la salida OUT es sacado uno o más pulsos de la señal de reloj de entrada



Divisores “dual modulus”

- Usados en RF para lograr mejor discriminación al dividir frecuencias que la de un divisor 1/M. Emplea 3 divisores:
  - un divisor de alta frecuencia que divide la frecuencia de entrada por dos posibles valores sucesivos (M o M+1)
  - un divisor operando a la frecuencia más lenta de salida, que divide la frecuencia de salida por un factor N
  - un divisor decreciente operando a la frecuencia más lenta de salida, que divide la frecuencia de salida por un factor A (donde A <=N) pero que al llegar a cero queda detenido para reiniciarse sólo cuando reinicia el divisor por N. Este divisor genera una señal de control hacia M/M+1 cuando está contando (caso en que el primer divisor divide por M+1) o en cero (caso en que el primer divisor divide por M)



Cada N ciclos de fout:

- A ciclos corresponden a M+1 pulsos de reloj de entrada por ciclo
- (N-A) ciclos corresponden a M pulsos de reloj de entrada por ciclo

$f_{in}/f_{out} = (A*(M+1)+(N-A)*M)/N$

$f_{in}/f_{out} = (N*M + A) /N$

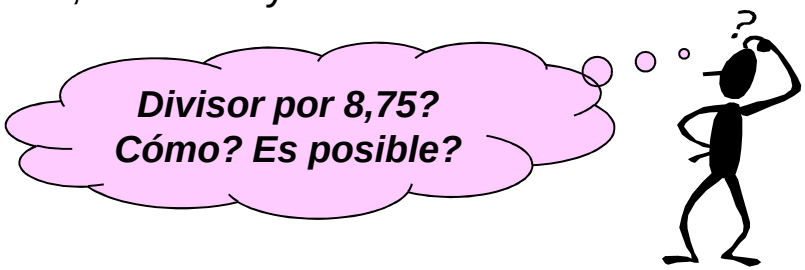
$f_{in}/f_{out} = M + A/N$

parte entera

parte fraccionaria

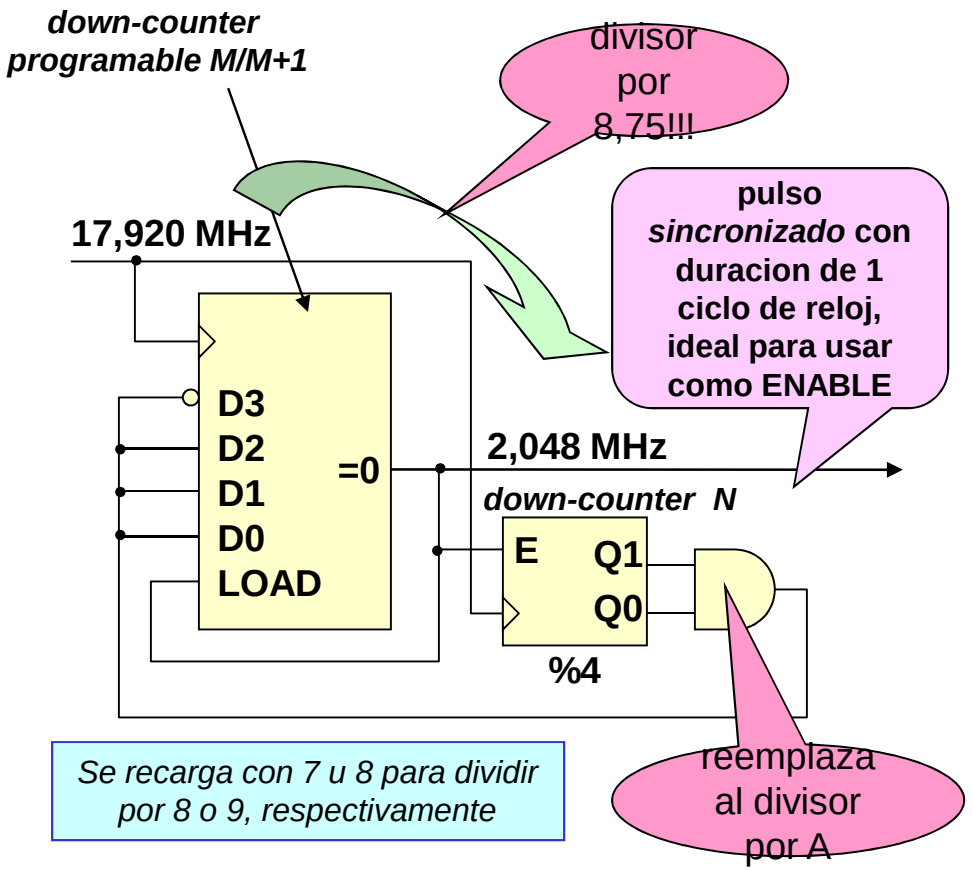
Otra visión del “dual modulus”

- En ciertas aplicaciones es necesario contar con más de una señal de reloj, y si esto es inevitable es ideal que ambos relojes sean coherentes, para evitar metaestabilidades
- Una posible solución es usar un reloj básico muy veloz cuya frecuencia sea el ***Mínimo Común Múltiplo (MCM)*** de las señales deseadas, y dividir esa señal con distintos divisores para lograr los relojes deseados, que operan como “enables”
- Por ejemplo, de desearse dos señales de 2,048MHz y de 2,240 MHz (cuya relación racional es 32/35) una opción es:
  - seleccionar un reloj maestro de 71,680MHz (MCM de 2048 y 2240)
  - dividirlo por 35 para generar 2,048 MHz
  - dividirlo por 32 para generar 2,240 MHz
- Si el reloj maestro requerido resulta de frecuencia demasiado alta, la alternativa puede ser usar divisores fraccionarios. Por ejemplo, usar 17,920 MHz y
  - ***dividirlo por 8,75*** para generar 2,048 MHz
  - dividirlo por 8 para generar 2,240 MHz



Otra visión del “dual modulus”

- Un divisor entero de frecuencias divide la señal de entrada *siempre* por el mismo número entero
- Un divisor fraccionario de frecuencias divide la señal de entrada por uno de dos posibles números enteros sucesivos (***dual modulus***)
- Cuántas veces divide por uno o por el otro determina el factor de división promedio
- Por ejemplo, para dividir por 8,75 basta de cada cuatro veces dividir tres veces por nueve y una vez por ocho
- (3x9+1x8)/4=8,75
- Para el ejemplo del dual modulus sería M=8, N=4, A=3



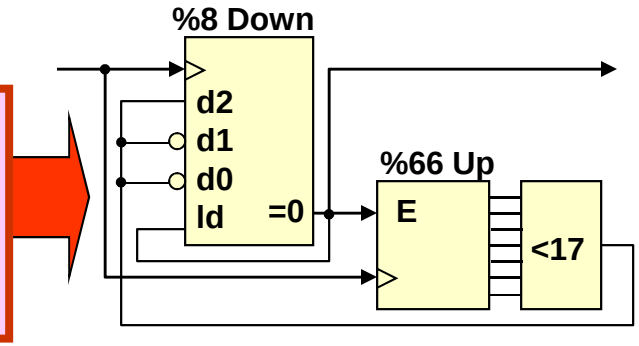
Similar al Dual Modulus
Diseñar el circuito en VHDL y simularlo

Divisores de frecuencia con factores fraccionarios arbitrarios

- Si se desea diseñar un divisor por un número fraccionario cabe como posibilidad que la parte fraccionaria sea o no periódica. Por ejemplo dividir por 8,75 tiene una parte fraccionaria (0,75) no periódica, pero dividir por **4,2575757575757..** tiene parte fraccionaria con una parte no periódica (el **2**) seguida de una parte periódica (**57 57 57 57 57..**)
- Dado un factor de la forma **N**,**A..BC..DC..DC..D..** (factor de división con una parte **A..B** no periódica seguida de una parte **C..D** periódica, es posible expresar la parte fraccionaria como un cociente de enteros (**A..BC..D** - **A..B**)/**M**, donde **M** es un número con tantos nueves como dígitos tenga el campo periódico **C..D** seguido a derecha por tantos ceros como dígitos tenga el campo no periódico **A..B**.
- El factor de división **4, 2 $\overline{57}$ 57 57 57..** puede ser expresado como:

4 + (257 - 2)/990 = 4 + 255/990 = 4 + 17/66

el divisor por 4,25757575 puede solucionarse con un divisor de doble módulo que, cada 66 veces que recicla, en 17 veces divida por 5 y en 49 por 4!!! Es decir, un divisor down programable de 3 bits y uno up fijo de 7 bits y modulo 66 (dual modulus con M=4, N=66, A=17)

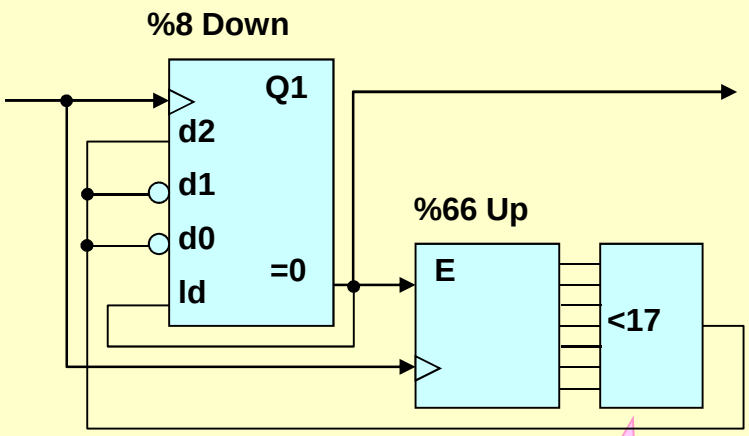


Un primer fractional divider en VHDL

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE ieee.std_logic_unsigned.ALL;

ENTITY fractdiv IS PORT (
  reloj : IN std_logic; sali : OUT std_logic);
END fractdiv ;

ARCHITECTURE a OF fractdiv IS
 SIGNAL div8  : std_logic_vector (2 DOWNTO 0);
 SIGNAL div66 : std_logic_vector (6 DOWNTO 0);
 SIGNAL men17 : std_logic;
BEGIN
 men17 <= '1' WHEN div66 < 17 ELSE '0';
 PROCESS (reloj) IS BEGIN
  IF (reloj = '1') THEN
   IF (div8 = 0) THEN
    IF (men17 = '1') THEN div8 <= "100"; ELSE div8 <= "011"; END IF;
    IF (div66 < 65) THEN div66 <= div66 + 1; ELSE div66 <= (OTHERS => '0'); END IF;
   ELSE div8 <= div8-1;
   END IF;
  END IF;
 END PROCESS;
 sali <= div8(1);
END a;
```



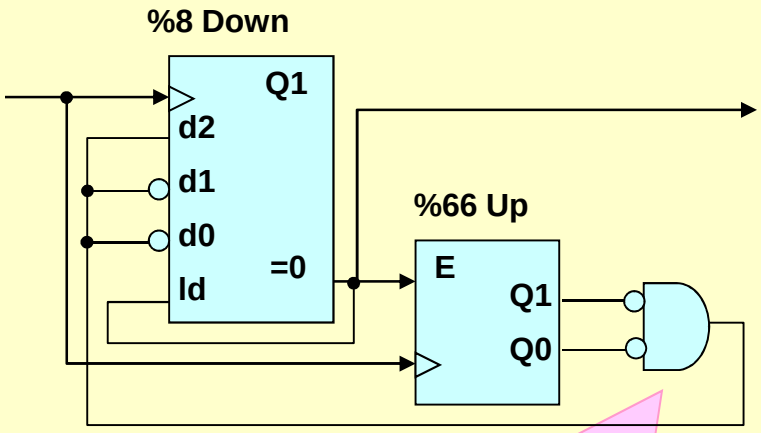
Mejora posible: como distribuir mejor las divisiones por 5 y por 4 ?

El fractional divider mejorado

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE ieee.std_logic_unsigned.ALL;

ENTITY fractdiv IS PORT (
  reloj : IN std_logic; sali : OUT std_logic);
END fractdiv ;

ARCHITECTURE a OF fractdiv IS
 SIGNAL div8  : std_logic_vector (2 DOWNTO 0);
 SIGNAL div66 : std_logic_vector (6 DOWNTO 0);
BEGIN
 PROCESS (reloj) IS BEGIN
  IF (reloj = '1') THEN
   IF (div8 = 0) THEN
    IF (div66(1 DOWNTO 0) = “00”) THEN div8 <= "100"; ELSE div8 <= "011"; END IF;
    IF (div66 < 65) THEN div66 <= div66 + 1; ELSE div66 <= (OTHERS => '0'); END IF;
   ELSE div8 <= div8-1;
   END IF;
  END IF;
 END PROCESS;
 sali <= div8(1);
END a;
```



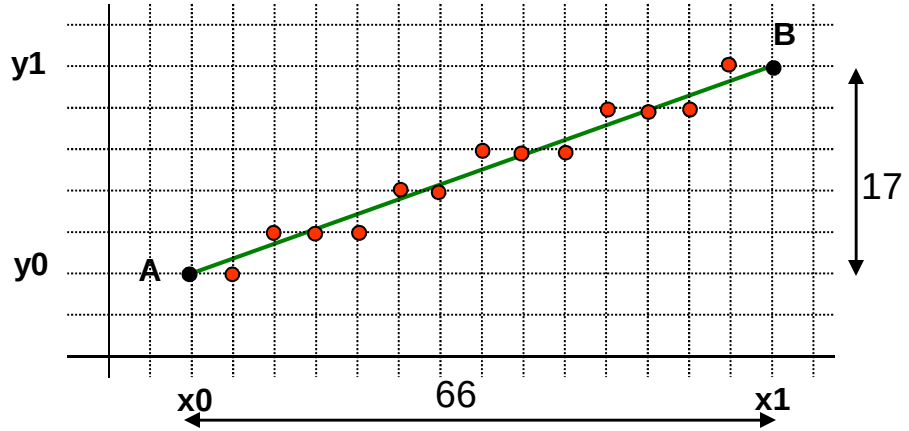
Si cuento los múltiplos de 4 que caben en un número de 0 a 65 tengo justo 17!!!: 0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60 ,64

*Aunque..entre **64..65..0** hay solo un periodo de separacion, y en todos los otros casos 3*

Este comportamiento es MUCHO mejor que el de un prescaler dual modulus!!

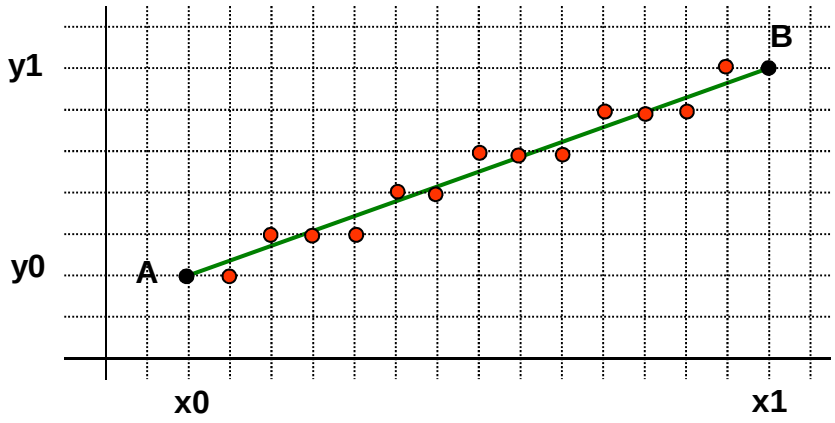
Se podrá mejorar aún más esta distribución?

Cómo no acumular error de fase y distribuir la división entre N y N+1?



- En un divisor de doble módulo (4 o 5 en el ejemplo previo) es inevitable el “ruido de fase” (*jitter*) debido a que ambos períodos son de distinta duración (N y N+1). Es deseable la distribución uniforme de los ciclos “N+1” entre los ciclos “N” para que el error de fase sea el mínimo posible (±½ ciclo de reloj).
- En el ejemplo del divisor fraccionario se vió que cada 66 pasajes por cero del divisor por N/N+1 en 17 casos había que dividir por N+1=5 y en los 49 casos restantes por N=4
- Si imaginamos tener que dibujar una recta que cada 66 incrementos enteros en el eje X realice 17 incrementos enteros en el eje Y, podríamos decir que cada vez que Y se incrementa nuestro divisor debe dividir por 5, y cuando no se incrementa el divisor debe dividir por 4
- Y en esta recta el error de la aproximación (puntos rojos) a la recta real (verde) debe estar entre ±½

El Algoritmo de Bresenham para optimizar divisores fraccionarios



- Para lograr la distribución uniforme de los ciclos “N+1” entre los ciclos “N” y hacer que el error de fase sea el mínimo posible (±½ ciclo de reloj), puede emplearse una variación de un algoritmo usado en “*computer graphics*” para dibujar líneas, llamado **Algoritmo de Bresenham**.

La ecuación ***y = f (x)*** de un segmento de recta entre dos puntos ***A*** y ***B***, definidos por sus coordenadas ***A=(x0,y0)*** y ***B=(x1,y1)*** está dada por:

y = y0 + m.(x-x0) para ***x0 <= x <= x1***

Donde la pendiente ***m = (y1 – y0)/(x1 - x0)***

Si ***x*** sólo toma valores enteros, una realización en C de una función que grafique el segmento es:

```
void drwseg (int x0, int y0, int x1, int y1) {
  int x = x0;
  float y = y0;
  float dx = x1 – x0;
  float dy = y1 – y0;

  set(x,(int)y);
  while(x != x1){
    x++;
    y += dy/dx;
    set(x,(int)(y + 0.5));
   }
}
```

División, Suma y Redondeo: Operaciones en punto flotante!!

Actualizo **y** en forma incremental sumándole **m**

El Algoritmo de Bresenham para optimizar divisores fraccionarios

- Inicializa Deltay=-0 y en cada incremento de “x” al valor de “y” se suma m (dy/dx).
- En vez de hacer eso se puede llevar por separado la parte entera de “y” y el error “Deltay” entre esa parte entera y el valor exacto de “y”. Si Deltay supera +0,5 se suma 1 a la parte entera de “y” y se resta 1.0 al error Deltay
- Pero …
- puedo inicializar Deltay=-1 y tomar decisiones si supera 0 .. O mejor aún
- Si al error Deltay lo multiplico por 2*dx:
  - *Se lo inicializa con –2*dx (en vez de –1)*
  - *Se lo incrementa en 2*dy (en vez de m)*
  - *Si corresponde, se le resta 2*dx (en vez de 1)*
- Efecto: todas las operaciones son sumas y restas con enteros. Este es el ***Algoritmo de Bresenham***

```
void drw1(int x0, int y0, int x1, int y1){
  float  dx, dy, Delta_y = -0.5;
  int x=x0,y=y0; dx=(x1 – x0); dy=(y1 – y0);
  set(x,y);
  while(x != x1){
    x++;
    Delta_y += dy/dx;
    if(Delta_y > 0){ Delta_y--; y++;}
    set(x,y);
   }
 }
```

Operaciones en punto flotante!!

```
void bres1(int x0, int y0, int x1, int y1){
  int dx, dy, Deltay, x = x0, y = y0;
  int dx2 = -2*(x1-x0), dy2 = 2*(y1-y0);
  Deltay = dx2;
  set(x,y);
  while(x != x1){
    x++;
    Deltay += dy2;
    if(Deltay > 0){ Deltay += dx2; y++;}
    set(x,y);
   }
 }
```

Todo es en punto fijo!!!!!!

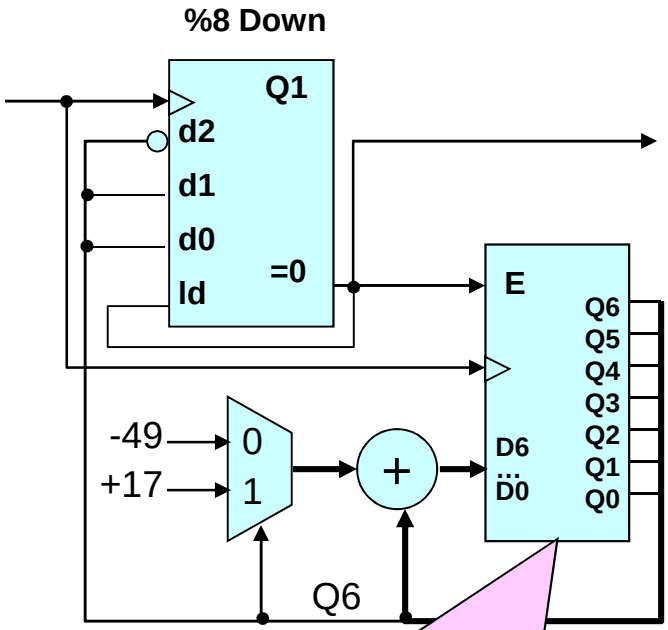
El Algoritmo de Bresenham para optimizar divisores fraccionarios

```
void bres2(int x0, int y0, int x1, int y1){
  int inct, incf, dx, dy, Dy, x = x0, y = y0;
  dx = x1 – x0; dy = y1 – y0;
  inct = 2*dy; incf = 2*(dy - dx);
  Dy = -dx + 2*dy;
  set(x,y);
  while(x != x1){ x++;
    if(Dy > 0){ Dy += incf; y++;}
    else        Dy += inct;
    set(x,y);
   }
}
```

Divido por N+1

Divido por N

- El incremento constante en cada ciclo de **Dy += 2*dy** puede ponerse dentro del IF generando un IF..ELSE que es un multiplexor que suma a **Dy** uno entre dos valores constantes
- En el divisor por **4,25757575 = 4 + 17/66** no importan los valores iniciales, ni es necesario multiplicar por 2 para el redondeo, entonces **dx=66**, **dy=17**, **inct = +17**, **incf = -49**



El acumulador puede ir desde -49 hasta +16
Con 7 bits tengo –64 a +63

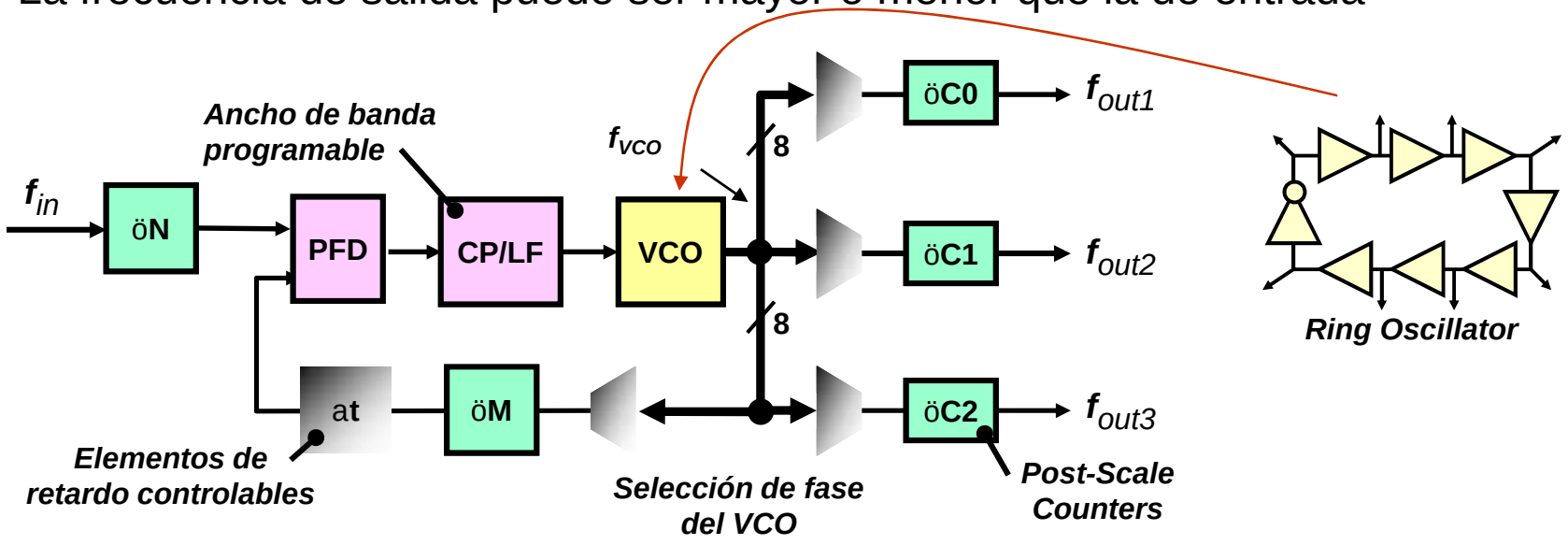
Diseñar el circuito en VHDL y simularlo

Uso de PLLs para obtener relaciones de frecuencias

- Un PLL (***Phase Locked Loop***) es un circuito realimentado que emplea un detector de frecuencia y fase (PFD), un filtro compensador pasabajos (LF), un oscilador controlado por voltaje (VCO) y varios divisores
- En condiciones estables la relación entre las frecuencias de salida y de entrada queda definida por

$$f_{out} = f_{in}\left(\frac{M}{N*C}\right)$$

- La frecuencia de salida puede ser mayor o menor que la de entrada

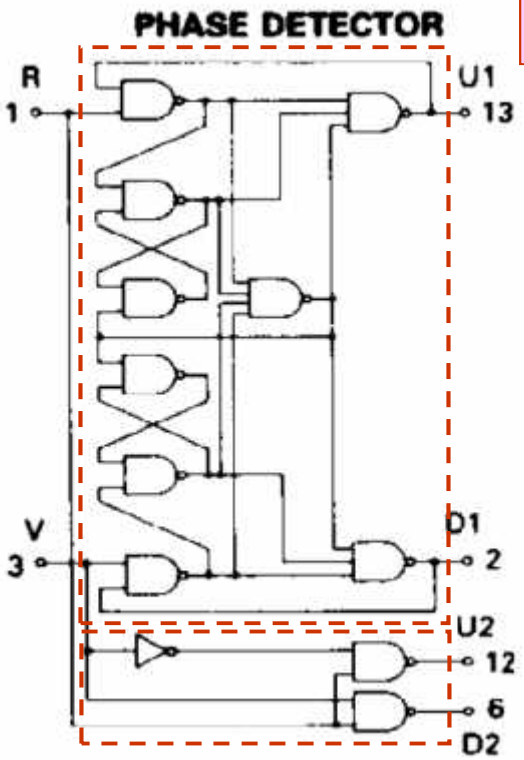


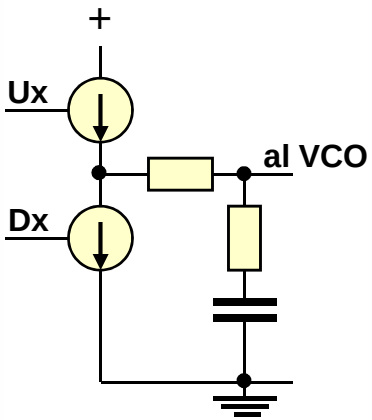
Circuitos PD/PFD para PLLs

- El PD (***Phase Detector***) más simple es una compuerta XOR, sirve para señales de relación de trabajo 50% y permite mantener 90% de diferencia de fase
- Un PFD (detector de fase y frecuencia) controla a un VCO para realizar el ajuste grueso de frecuencia. Con esto se lleva la frecuencia de recepción dentro de un margen de tolerancia de la del transmisor.

  Y una vez en frecuencia permite el control de fase

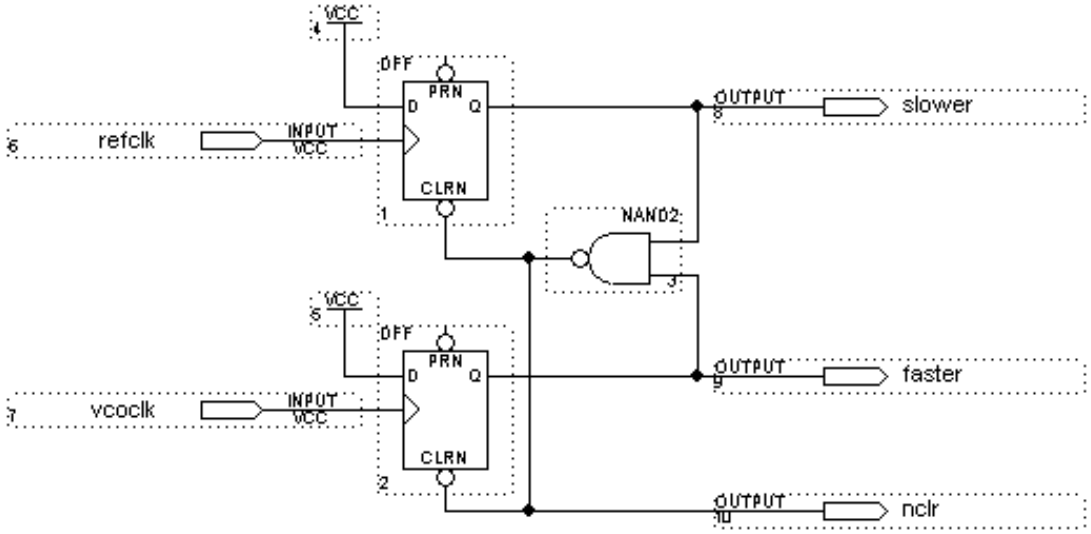
Diseñar el circuito en VHDL y simularlo





El venerable MC4044 incluía un simple detector de fase (salidas U2 y D2) y un detector de frecuencia/fase (salidas U1 y D1)

Circuitos PFD para PLLs aptos para FPGAs

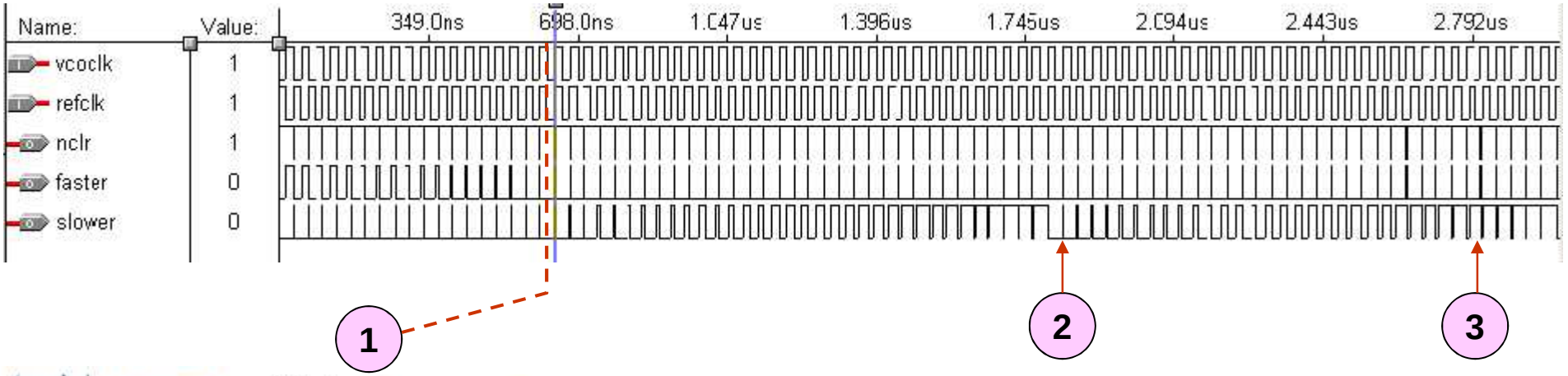


- El esquema básico de un PFD, es el de la figura
- Pasa a un estado conocido a partir del momento en que dos flancos de la señal más rápida entran dentro de un ciclo de la más lenta
- Si ***refclk > vcoclk*** en ***slower*** aparecen pulso mínimos, en cambio en ***faster*** aparecen pulsos de mayor duración, que se hacen minimos mientras que ambos relojes están en fase

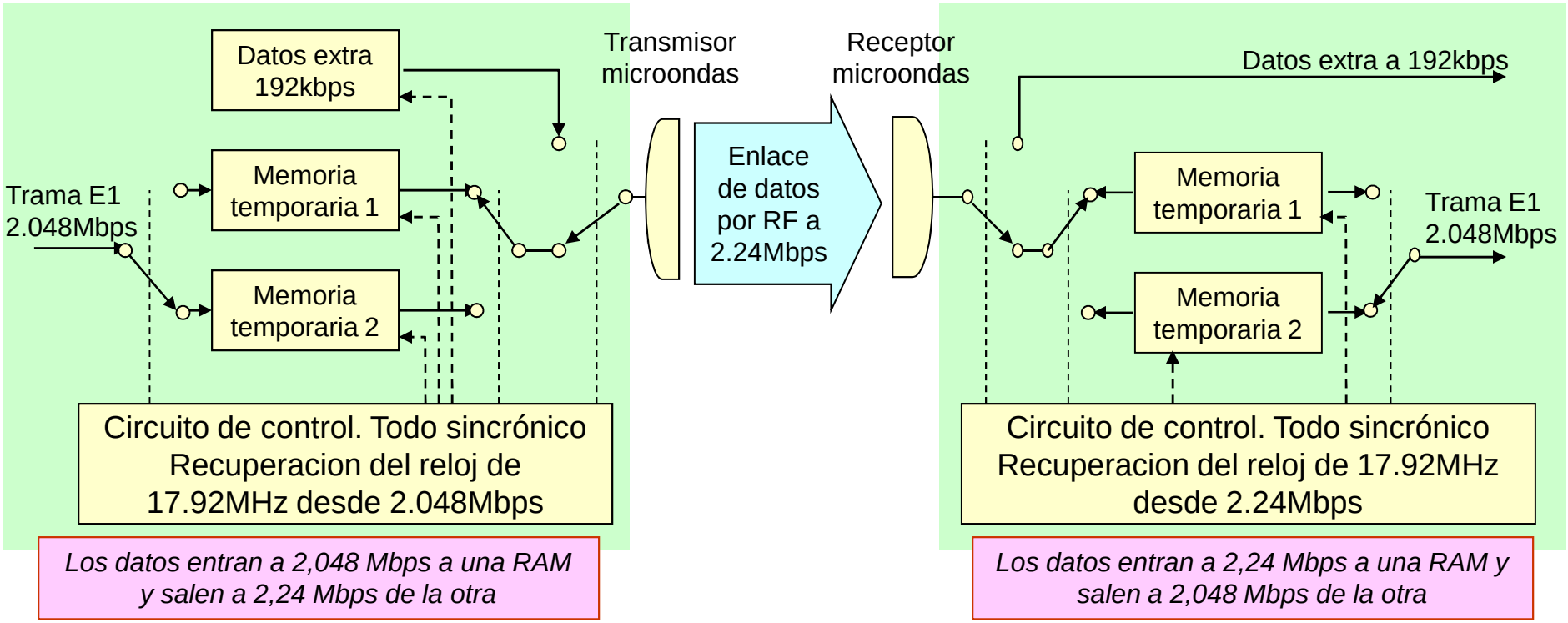
Circuitos PFD para PLLs

Se muestra la sincronización y posterior funcionamiento de detección de frecuencia y fase

- Hasta (1) el proceso está desincronizado
- En (1) se produce que dos flancos de la señal ***refclk***, de mayor frecuencia, entran dentro de un período de ***vcoclk***, más lenta, y el circuito se sincroniza
- A partir de allí ***faster*** sólo genera pulsos mínimos, mientras que ***slower*** genera señales de valor medio significativo, indicando que el vco opera a menor frecuencia que la referencia
- La señal ***slower*** se hace mínima cuando ambos relojes están con error de fase positivo mínimo (2) y máxima cuando el error de fase positivo es máximo (3), con lo que no sólo controla frecuencia sino enganche de fase entre ambos relojes



Ejemplo: uso de relojes coherentes con divisores fraccionarios



- Corresponde a un problema real de un enlace de telefonía (una trama digital E1) a través de microondas
- El canal auxiliar de datos de 192kbps permite agregar información de servicio (voz y datos) para el mantenimiento del sistema, en forma transparente al enlace telefónico
- El uso de memorias internas de las FPGAs, de divisores coherentes fraccionarios, y de sistemas de sincronización facilita el diseño en la FPGA

Ejercicio práctico

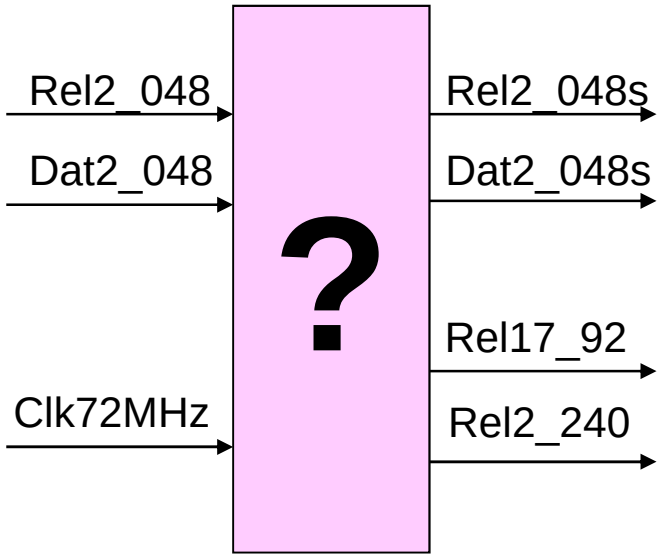
El problema a considerar consiste en el diseño de un circuito que recibe dos señales del mundo externo:

- Un reloj de 2,048MHz
- Datos a 2,048Mbps sincronizados con ese reloj

Usando un reloj interno con una frecuencia aproximada a 72 MHz se deben diseñar los sincronizadores y recuperadores de reloj que permitar generar:

- Las señales de Datos y Reloj de entrada sincronizados con el reloj interno, a fin de evitar posteriores metaestabilidades
- Un reloj de 17,92MHz (en realidad 35/4 veces la frecuencia del reloj de entrada)
- Un reloj de 2,240MHz (relacion 35/32 veces la frecuencia del reloj de entrada)

Todo el circuito debe operar en modo sincrónico usando el reloj de 72MHz



Ejercicio práctico: una posible arquitectura