Diseño Automático de sistemas

Dominios de reloj: Metaestabilidad

Prof. Pablo Sarabia Ortiz

UNIVERSIDAD NEBRIJA

Objetivos Clase

- Entender los problemas derivados de usar más de un dominio de reloj en un diseño digital.
- Detectar y mitigar los riesgos de la metaestabilidad.
- Aprender a seleccionar entre las soluciones disponibles para evitar la metaestabilidad.

Bibliografia

- **Capítulo 16**

  Pong P. Chu (2006), RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability, Willey, 1st Edition

Contenidos

1. Dominios de Reloj
2. Relojes síncronos derivados (clock enable)
3. GALS (Global Asynchronous Locally Synchronous)
4. Metaestabilidad (repaso)
5. Handshake
6. Memoria Compartida
7. Resumen

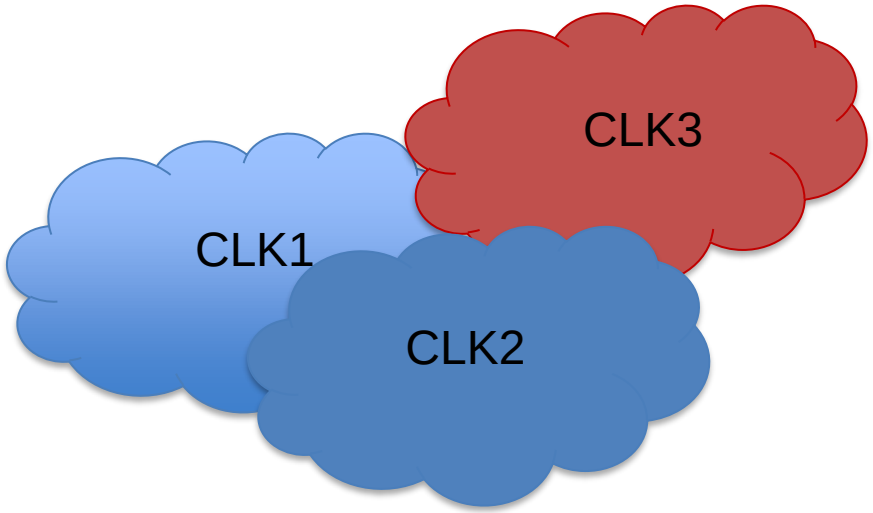
Diseño síncrono

Hasta ahora todos los diseños que hemos realizados tienen un único reloj. Es la forma más sencilla de diseñar pero desafortunadamente no es lo habitual.

Habitualmente tenemos diferentes relojes dentro de un mismo diseño.

Dominio de reloj

Se llama dominio de reloj a todos aquellos circuitos síncronos que tienen la misma señal de reloj.



¿Cuando sucede?

- **Interfaces externas**, funcionan a una frecuencia fija diferente al diseño interno de la FPGA, práctica 2.
- **Tamaño del circuito,** cuando el circuito se hace muy grande no se puede tener una única señal de reloj.
- **Complejidad del diseño,** diferentes partes de nuestro circuito funcionan a diferentes frecuencias, es habitual que ciertas partes de nuestro diseño tengan una frecuencia limite diferente al resto.
- **Consumo**, mayor frecuencia implica mayor consumo. Muchas veces, sólo es necesario tener un subsistema funcionando a una frecuencia elevada.

¿Qué tipos de reloj hay?

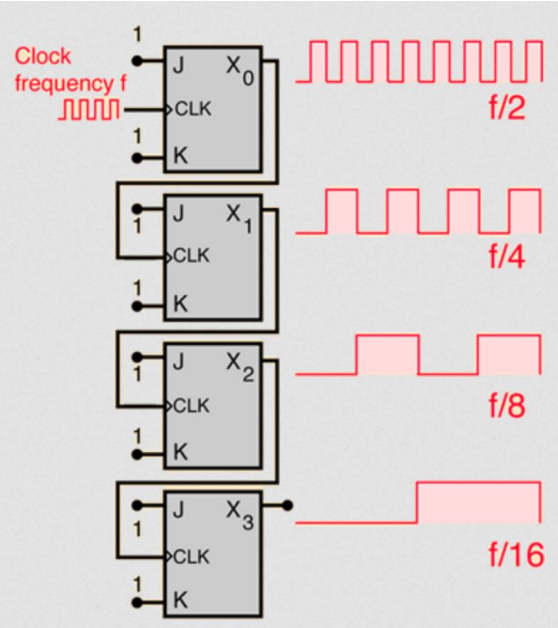
- Señales de reloj síncronas derivadas: Gate Clocking
- Señales de reloj asíncronas: GALS (Globally Synchronous Locally Synchronous)

Reloj síncrono derivado

- Todos los relojes del sistema dependen de uno de mayor frecuencia.
- Es la técnica **más sencilla,** pero obliga al sistema a funcionar a una elevada frecuencia. Muchas veces solo necesaria en una parte del diseño.
- Elevado consumo. Elevado área.
- **Nunca** usar una señal de clk derivada para los registros (siguiente diapo). Usar gate clocking.

Reloj síncrono derivado: Divisor de reloj

- Idea intuitiva para generar un reloj:
  Divisor de reloj, encadenar flipflops.



Reloj síncrono derivado: Divisor de reloj

- No se debe conectar esta salida de reloj (SCLK) a otra lógica

```
1  -- Dos señales de reloj CLK y SCLK (reloj síncrono derivado)
2  if rising_edge(SCLK)
3      A <= B;
4  end if;
5
```

INCORRECTO

- La solución: Clock enable es siempre usar el mismo reloj del sistema. Y generar señales que nos indiquen el flanco de subida y bajada del reloj derivado.

Reloj síncrono derivado: Clock Enable

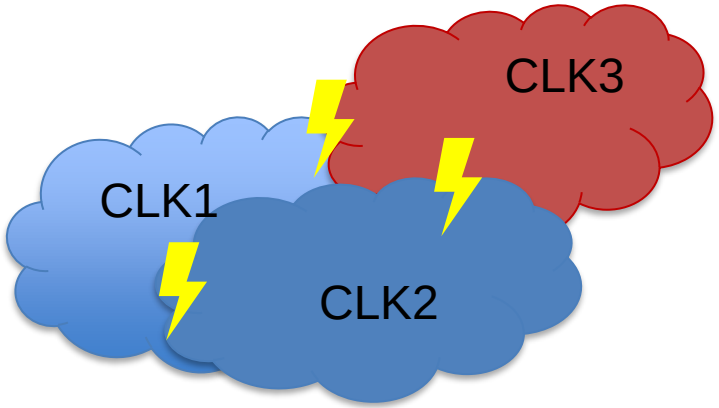
- Generamos la señal de SCLK y las señales que nos indican el flanco de subida y de bajada.
- El flanco de subida/bajada lo ponemos como un enable del clock principal.
- Problema: Necesitamos que los relojes secundarios sean divisiones de 2^n donde n debe ser al menos 2.
  Consecuencia: la máxima frecuencia de un reloj derivado es f_sys/4.

```
1  -- Dos señales de reloj CLK y SCLK (reloj síncrono derivado)
2  if rising_edge(CLK) and SCLK_rise = '1'
3      A <= B;
4  end if;
5
```

Relojes Asíncronos: GALS

- Consecuencia de los problemas de derivar una señal de reloj del reloj principal.
- Muchas veces es inevitable tener relojes asíncronos.

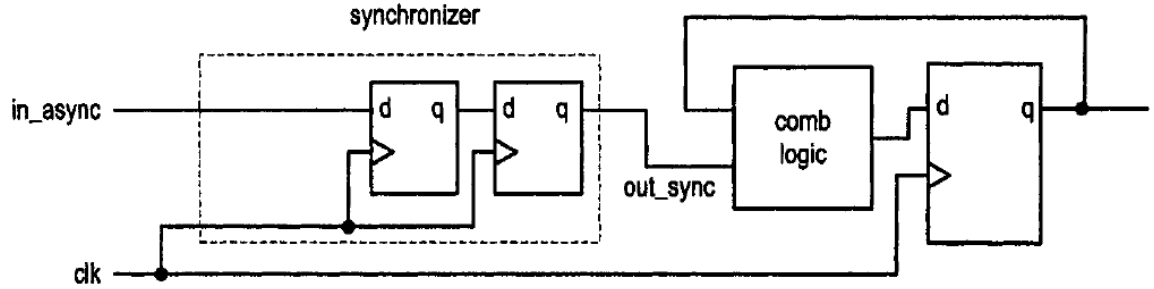
Reloj Asíncrono: GALS



- Cada uno de los relojes no conoce el estado del resto de los relojes.
- Al cambiar de dominio tenemos que tener especial precaución.
- No es sencillo implementar los cambios de dominio, pero es necesario.

Metaestabilidad (repaso)

- Sucede cuando un flipflop no respeta los tiempos de setup/hold.
- El resultado no es predecible.
- En este caso se da porque las señales que cruzan de un dominio de reloj a otro no son síncronas.
- Solución Sincronizadores (encadenar flipflops), regla general usar 3 flipflops.



(c) Two-FF synchronizer

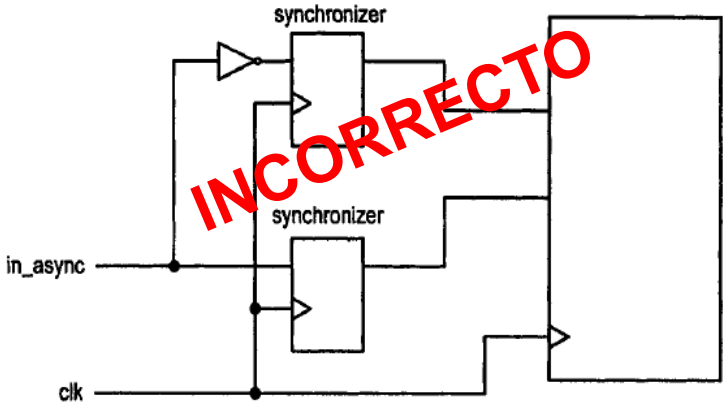
Soluciones

- Es inevitable que tengamos señales que deban cruzar los dominios de reloj.
- Existen diferentes soluciones para evitar la metaestabilidad, todas se basan en reducir al mínimo las señales que cruzan los dominios.
- Vamos a ver en detalle: Sincronizador, Detector de flancos, Handshake, Memoria compartida.

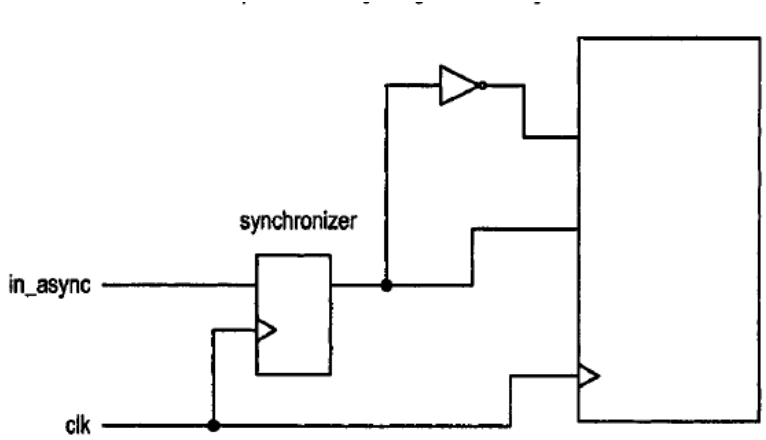
Sincronizador: Consejos

- Usar señales de entrada sin glitches (es decir registradas).
- Sincronizar una señal en único sitio.
- Evitar sincronizar señales relacionadas.
- Cuidado con las señales de un ciclo que al cambiar de dominio pasan a ser de varios ciclos (edge detector)
- Revisar que el MTBF (Tiempo Medio entre Fallos) al cambiar el diseño sigue siendo razonable (Fuera del objetivo de este curso).

Sincronizar en un único punto



(a) Synchronizing a signal in two places



(b) Synchronizing a signal in one place

Evitar sincronizar señales relacionadas

- Dado un contador (2 bits) si lo deseamos sincronizar, va a cambiar de 01 a 10. Este cambio de dos bits de forma simultanea da más problemas que un único cambio de un bit.
- Solución: Contador en código Gray

Contador en código Gray

- Dado un contador (2 bits) si lo deseamos sincronizar, va a cambiar de 01 a 10. Este cambio de dos bits de forma simultanea da más problemas que un único cambio de un bit.
- Solución: Contador en código Gray, todos los cambios en bit consecutivos es de un bit.

Decimal Value	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100

Ejemplo crear un contador Gray

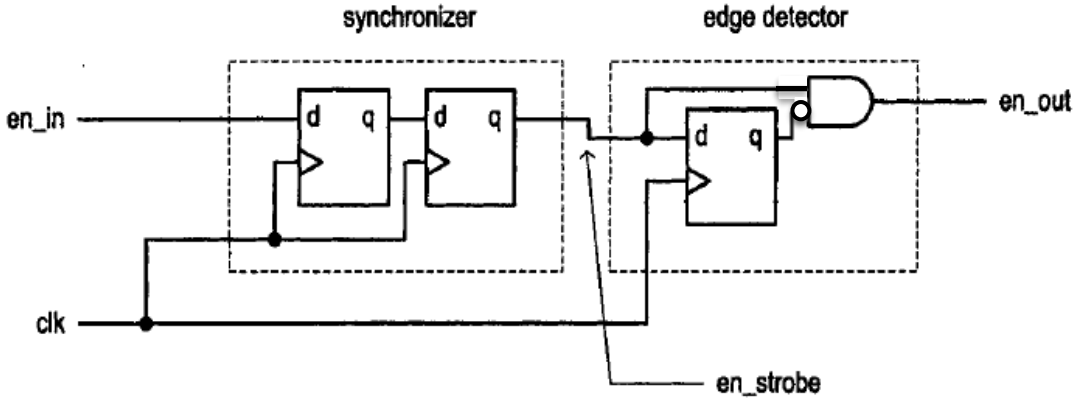
```
18      IF rising_edge(clk) THEN
19        IF (Rst = '1') THEN
20          Currstate <= (OTHERS =>'0');
21        ELSIF (En = '1') THEN
22          Currstate <= Nextstate;
23        END IF;
24      END IF;
25    END PROCESS;
26      -- a_i+1
27    hold <= Currstate XOR ('0' & hold(N-1 DOWNTO 1));
28    next_hold <= std_logic_vector(unsigned(hold) + 1);
29    Nextstate <= next_hold XOR ('0' & next_hold(N-1 DOWNTO 1));
30    output <= Currstate;
```

Señales anchas (edge detector)

- Al pasar de un reloj lento a uno rápido necesitamos un detector de flanco.

- **Ejercicio: Diseña el edge detector.**

Señales anchas (falling edge detector)




```
1  process(clk, rst)
2  begin
3      if (rst = '1') then
4          delay_reg <= '0';
5      elsif rising_edge(clk) then
6          delay_reg <= sig_in;
7      end if;
8  end process;
9   output <= (not delay_reg) and sig_in;
```

¿Qué pasa cuando vamos de un reloj más rápido a uno más lento?

- Si una señal de un pulso de duración va de un reloj más rápido a uno más lento lo más probable es que se pierda.
- Solución: Handshake, Mantenemos la señal hasta que el otro sistema nos lo indique.

Handshake

- Los dos subsistemas (en diferentes dominios de reloj), acuerdan la transacción.
- El más común es el Handshake de 4 etapas.
  - Talker y Listener
  - Talker activa el req y espera el ack
  - Listener manda el ack talker envia desactiva el req
  - Listener desactiva el ack

Handshake 4 etapas

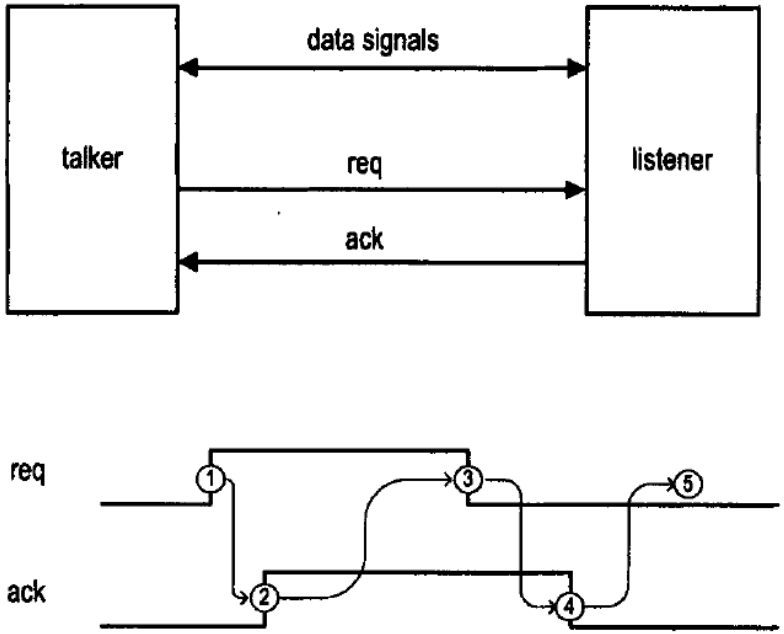
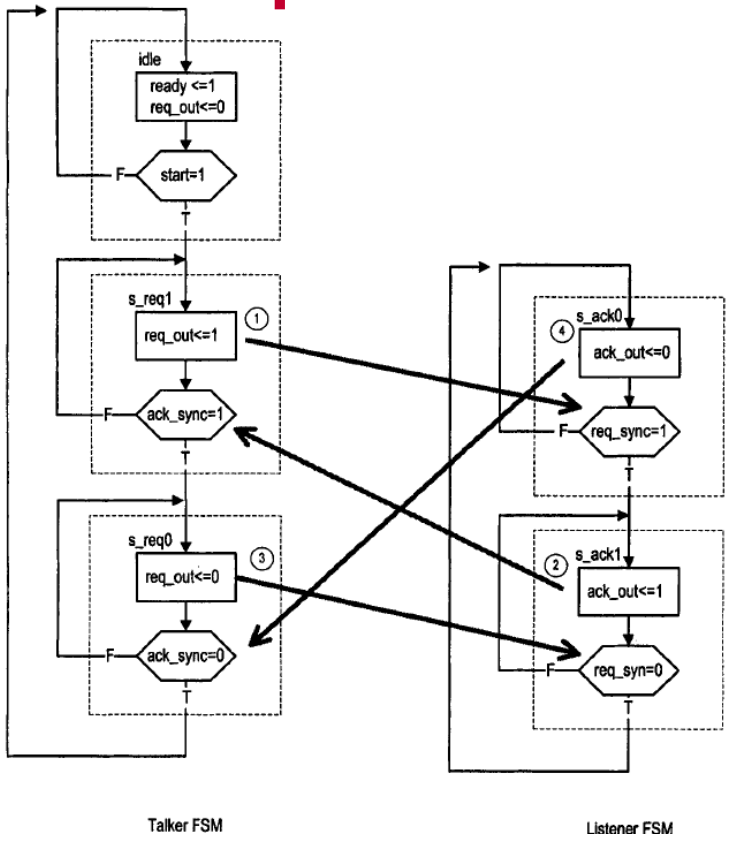


Figure 16.14 Basic conceptual and timing diagrams of the four-phase handshaking protocol.

Handshake 4 etapas FSM

- **Ejercicio: Diseña el asm del handshake de 4 etapas**

Handshake 4 etapas ASM: Solución



Transferencia de datos

Hasta ahora hemos visto transferencia de señales, pero ¿Qué pasa cuando tenemos que comunicar datos?.

Podemos simplemente **añadir más líneas al protocolo** de handshake que comuniquen los datos. El talker mantiene el dato hasta que el receiver manda el ACK.

O bien usar una **memoria compartida**.

Memoria FIFO asíncrona

Repaso: Una FIFO es un array de memoria con dos punteros de escritura y lectura.

Si el sistema que escribe y el que lee están en diferentes dominios de reloj, los punteros deben de ser sincronizados.

Para los punteros se usa un contador, como ya hemos mencionado es necesario usar un contador que solo cambie un bit por incremento (Gray code counter).

Memoria FIFO asíncrona: Esquema

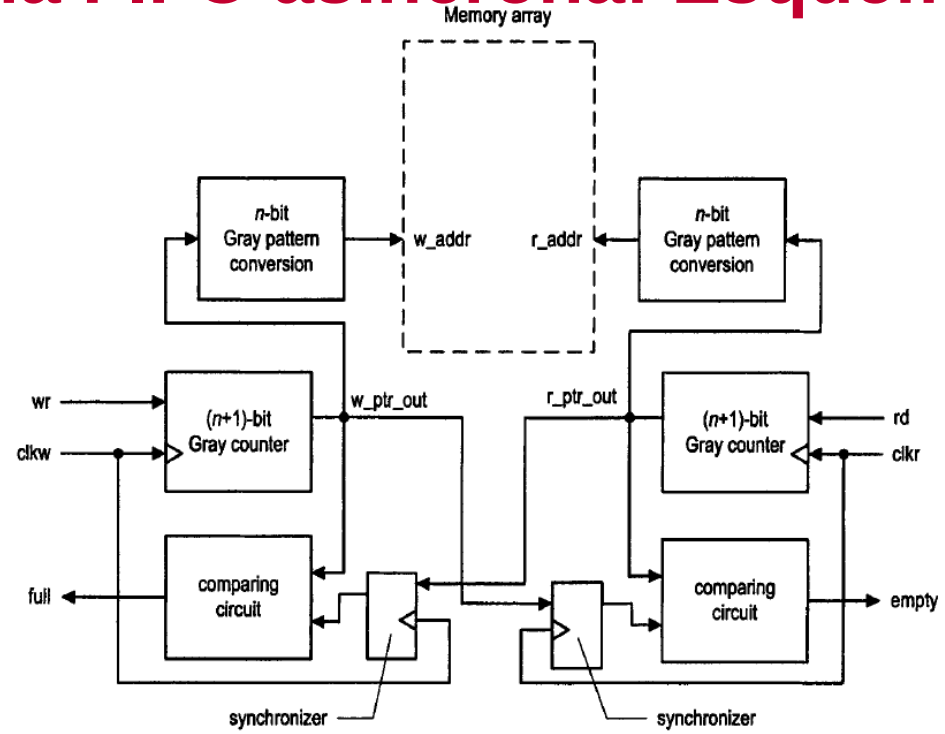


Figure 16.26 Block diagram of an asynchronous FIFO controller.

Diseño de un sistema con varios relojes

- Lograr un diseño que satisfaga de forma **sistemática** las condiciones de reloj.
- En simulación aparecerán X, para indicar que se ha violado una condición de tiempos, pero en síntesis se convertirán en 1s o 0s aleatoriamente.
- Aproximación
  1. Dividir el sistema en los bloques síncronos
  2. Diseñar y verificar cada uno de los bloques síncronos
  3. Desarrollar un protocolo para la comunicación entre los bloques
  4. Analizar que se cumplen los tiempos y las condiciones de reloj
  5. Verificar el sistema completo

Resumen

- No usar señales de reloj generadas para lógica.
- **Minimizar** el número de relojes , usar relojes síncronos cuando se pueda (clock enable).
- Sincronizar las señales en **único lugar**.
- Evitar sincronizar señales relacionadas (Gray Code)
- **Usar señales registradas** (sin glitches) para sincronizar.
- Identificar claramente las **regiones de reloj**.
- Analizar el protocolo de dominio para las posibles combinaciones de reloj: mas lentas o más rapidas (edge detector, handshake, memoria compartida, etc…).

En la próxima hora

- Resolución de problemas.

¿Preguntas?

UNIVERSIDAD
NEBRIJA