ELO212: Laboratorio de Sistemas Digitales
Guía de Actividades Sesión 5

8 al 12 de mayo de 2023

1. Requisitos de entrada.

- Haber **completado y entendido** las actividades de las sesiones anteriores.
- Entender las diferencias fundamentales entre circuitos combinacionales y secuenciales, identificar cada tipo a partir de un diagrama, y como describirlos en SystemVerilog utilizando los bloques `always` correspondientes y sus convenciones de uso (ver slides de clases asociadas).

2. Objetivos

- Revisar diseños previos y comparar con módulos de referencia.
- Entender en profundidad la descripción de circuitos combinacionales y secuenciales, y como separar ambos tipos de lógica en SystemVerilog.
- Describir y experimentar con configuraciones básicas de bancos de registros para almacenamiento de información.
- Estudiar propiedades de señales de entrada provenientes de componentes electromecánicas análogas y como tratarlas digitalmente.

3. Actividades Guiadas.

3.1. Revisión ALU de referencia y diseños previos.

Esta actividad está asociada a los resultados de la sesión evaluada, y se dará a conocer posterior a que todos los paralelos y grupos hayan recibido su calificación.

3.2. Identificación y separación de lógica combinacional y secuencial.

Para esta actividad, revisar el video cápsula_sesion5 disponible en el repositorio de videos del curso.

La gran mayoría de los circuitos digitales modernos de mediana complejidad, por no decir casi todos, utilizan una configuración de lógica combinacional conectada mediante componentes secuenciales sincrónicas[1]. La lógica combinacional en general se encarga de generar salidas a partir de la combinacion de entradas mediante compuertas lógicas o primitivas de más alto nivel, mientras que las componentes secuenciales sincrónicas (básicamente FF-Ds) se encargan de propagar las señales en instantes determinados definidos por los cantos de una señal de reloj. En otras palabras, la lógica combinacional define el valor al que debe cambiar una señal, y la lógica secuencial determina el cuando las señales deben actualizarse a sus nuevos valores. Todo sistema digital sincrónico, desde controladores de segmentos a microprocesadores de alto desempeño se diseñan en base a estos principios.

Desde el punto de vista de diseño con HDLs, es importante entender cuando se debe trabajar con lógica combinacional y secuencial, seguir las reglas y convenciones para describir cada tipo de lógica, e identificar a que tipo de lógica corresponde un código. En general es una buena práctica separar ambos tipo de lógica, dejando bloques exclusivamente combinacionales a cargo de los cálculos (`assign` y `always_comb`), y bloques secuenciales a cargo de propagar las señales en los cantos de reloj (`always_ff`). Hacer esta separación explícita facilita seguir las convenciones de uso para cada tipo de bloque y reduce las potenciales fuentes de error (como por ejemplo, confundir el uso de o mezclar asignaciones bloqueantes y no-bloqueantes).

Hasta ahora hemos trabajado con circuitos secuenciales relativamente simples, como contadores sencillos, en que esta separación explícita en tipos de lógica no ha sido tan necesaria. Sin embargo, a medida que aumenta la complejidad de los sistemas, como veremos en las siguientes sesiones asociadas a máquinas de estado, realizar esta separación se vuelve necesario y hoy en día es una regla no escrita (una convención ampliamente aceptada pero que no es una regla debido a temas de compatibilidad).

Para ilustrar este tema, considere los siguientes ejemplos de módulo que describen tres distintas versiones para un contador simple:

```
1
2  module counter1#( parameter N = 8 )(
3    input logic clock, reset,
4    output logic [N-1:0] counter
5  );
6    always_ff @(posedge clock) begin
7      if(reset)
8        counter <= 'd0;
9      else
10       //Esta linea incluye logica combinacional cuyo resultado se actualiza en el
       canto de reloj (mezcla en un solo bloque logica combinacional y secuencial,
       lo que es aceptable para logica sencilla.)
11       counter <= counter + 'd1;
12   end
13 endmodule
14
15
16 module counter2#( parameter N = 8 )(
17   input logic clock, reset,
18   output logic [N-1:0] counter
19 );
```

[1] También hay circuitos asincrónicos que no veremos en este curso.

```
20   logic [N-1:0] counter_next;
21
22   always_comb begin
23       //La logica combinacional determina el valor al que debe actualizarse la
       senal en el siguiente ciclo
24     counter_next = counter + 'd1;
25   end
26
27   always_ff @(posedge clock) begin
28   //El valor calculado por la logica solo se propaga al llegar el canto de reloj.
       Notar que esta logica corresponde a un FF-D que solo determina el cuando, no
       el que.
29     if(reset)
30       counter <= 'd0;
31     else
32       counter <= counter_next;
33   end
34 endmodule
35
36
37 module counter3#( parameter N = 8 )(
38   input logic clock, reset,
39   output logic [N-1:0] counter
40 );
41
42   logic [N-1:0] counter_next;
43
44   always_comb begin
45   // En este caso la logica combinacional determina TODOS los valores al que debe
       actualizarse la senal en el siguiente ciclo, incluyendo el reset
46     if(reset)
47       counter_next = 'd0;
48     else
49       counter_next = counter + 'd1;
50   end
51
52   always_ff @(posedge clock) begin
53   //En esta seccion solo se indica la actualizacion del valor y nada mas
54     counter <= counter_next;
55   end
56 endmodule
```

Analice cuidadosamente cada código y dibuje en *papel y lápiz* el circuito que se está describiendo con los nombres de señales correspondientes. Haga, también en *papel y lápiz*, un diagrama de tiempo para ver la evolución de las señales en cada caso. Discuta con sus compañeros de grupo. Una vez entiendan bien los pasos anteriores, observen los esquemáticos generados por los elaborated designs y post-síntesis en cada caso, y comparen con sus interpretaciones. Verifique la equivalencia funcional instanciando en un solo testbench los tres contadores, sométalos a los mismos estímulos, y compare las salidas.

Se reitera que para circuitos simples como el contador planteado, todas las formas de descripción son equivalentes y válidas, siempre que se verifique la equivalencia funcional. Sin embargo, saber separar la lógica combinacional de la secuencial le puede ahorrar bastante dolores de cabeza y frustraciones en las siguientes etapas del curso. A medida que vaya adquiriendo experiencia, irá definiendo su propio estilo y reutilizando diseños previamente realizados.

Se enfatiza que entender estos conceptos es crítico para las siguientes actividades de esta sesión y para las sesiones siguientes. Plantee cualquier duda al staff hasta asegurarse de que entiende muy bien estos conceptos.

3.3. Modelos y análisis de registros básicos.

En la sección de bibliografía de la página principal de Aula se encuentra disponible un enlace llamado *HDL Design using Vivado*, el cual dirige a una página con referencias y actividades entregadas por Xilinx para buenas prácticas en la descripción de bloque lógicos fundamentales para varias tareas. En general, se recomienda revisar en detalle esa página y tenerla como referencia para refinar sus diseños previos y también para explorar cosas que no se verán directamente en clases. Notar que el material entregado contiene ejemplos en *Verilog clásico* [2], pero deberían ser capaces de entender su uso y actualizar a SystemVerilog reemplazando los bloques `always` clásicos por `always_comb` y `always_ff`, según corresponda, y reemplazando `wire` y `reg` por `logic` (ver slides de clases).

En particular, para esta actividad enfóquese en el Lab *Modeling registers and counters*, parte 1-Registers. En base a los ejemplos, implemente las estructuras planteadas a continuación. Para cada caso, asegúrese de entender la descripción, dibuje un esquemático a mano de la funcionalidad descrita, compare su dibujo con el entregado por elaborated design, y verifique la funcionalidad mediante simulación:

- Registro simple de 1 bit con carga condicional y señales `reset` y `load` sincrónicos, con `reset` teniendo prioridad sobre `load`.
- Registro de desplazamiento de 1 bit de entrada y un bit de salida, implementando una linea de retardo de 32 ciclos de reloj. El módulo debe tener una señal de `reset` y una señal `enable`. Si `enable` está activa al ocurrir un canto positivo, entonces se registra el valor de entrada y se desplaza el contenido interno de la linea de retardo en una posición. Notar que si `enable` se mantiene activo, un bit capturado en la entrada demorará 32 ciclos en aparecer en la salida.

En caso de dudas, se recomienda buscar referencias adicionales y consultar al staff.

3.4. ALU con entradas registradas.

Utilizando los módulos desarrollados en etapas anteriores, implemente el sistema mostrado en el diagrama de la Figura 1. En este caso, los operandos de entrada a la ALU provienen de bancos de registros con carga condicional. Adicionalmente, se debe incorporar un banco de registros a la salida de la ALU para almacenar el resultado y flags para la última operación realizada.

El comportamiento esperado se especifica a continuación (el cual también deberían ser capaces de inferir directamente desde el diagrama):

- Existe un único bus de entrada `data_in` que está directamente conectado a las señales `A`, `B`, y `OpCode`. Para efectos prácticos, y usando como referencia el datasheet de la tarjeta

[2] Se enfatiza que se menciona *Verilog clásico* para referirse en forma coloquial a formas de descripción que ya se encuentran en estado *deprecated*, pero que siguen siendo válidas en SystemVerilog. Recordar que técnicamente SystemVerilog es un superconjunto que incluye sintáxis de Verilog por temas de compatibilidad para facilitar la reutilización de módulos desarrollados previamente, pero que no se recomienda su uso en sistemas nuevos.

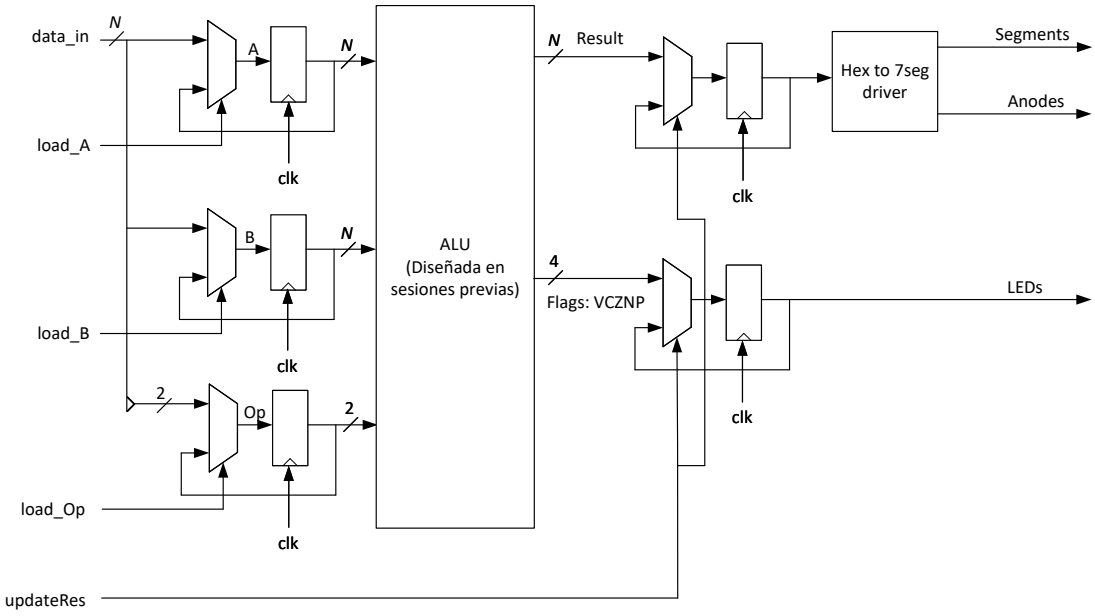

Figura 1: Diagrama de alto nivel para ALU con entradas registradas de carga condicional.

Nexys4DDR/A7, considere que los datos de entrada son seteados por medio de los switches disponibles. Para esto, considere que el parámetro N tiene un valor por defecto de 16. Notar que la señal `OpCode` requiere solo 2 bits de los 16 disponibles.

- Las entradas de la ALU provienen de bancos de registros. El valor a almacenar en los bancos de registros se actualiza en base al valor que tenga el bus de datos al momento de activar la señal de carga correspondiente. Si las señales de carga no están activadas, entonces los cambios en el bus de datos no se ven reflejados en las entradas de la ALU. Para efectos prácticos, y usando como referencia la tarjeta Nexys4DDR/A7, considere que cada señal de `load` proviene de uno de los botones pulsadores disponibles, el cual el usuario presiona solo una vez que ha terminado de configurar los datos de entrada [3]. De esta forma, solo se propaga el valor estable de los datos, evitando propagar cambios espúreos generados mientras se están actualizando los datos de entrada para calcular una nueva operación.

- Notar que lo que se está controlando por medio de los registros es el momento de actualizar los datos de entrada a la ALU. Recordar que la ALU es un circuito combinacional, por lo que está activa en todo momento calculando la operación especificada en sus datos de entrada (que en esta ocasión provienen de registros). Además, al utilizar un banco de registros en el resultado y los flags, estos se propagan a los pines de salida solo cuando se activa la señal `UpdateRes`.

- Incluya un botón de reset sincrónico global que permita llevar la salida de todos los bancos de registros a un valor deseado en cualquier momento. Este reset además es global para todo el sistema, incluyendo la lógica para el control de los displays.

- Al igual que como lo hizo en la guía 4, asuma que el reloj de entrada viene ajustado a la

[3] En general, estas señales no necesariamente deben ser manejadas por usuarios humanos, también pueden provenir de otros módulos.

frecuencia necesaria para que el número se vea estático en los displays al iterar la activación de los ánodos en cada ciclo de reloj.

Una vez descrito su diseño, observe, analice y compare los resultados obtenidos mediante el Elaborated Design y la síntesis lógica. Haga todas las consultas que considere necesarias para clarificar sus dudas. Revise cuidadosamente los mensajes entregados por el procesos de síntesis, y verifique que está todo en orden. Debe aprender a diferencias los Warnings que son informativos de los que son críticos para el diseño. Verifique la funcionalidad mediante simulación.

¿Que ventajas y desventajas observa del nuevo diseño comparado con la ALU diseñada para la sesión anterior? Por ejemplo, la tarjeta Nexys4DDR/Nexys A7 tiene 16 switches que puede utilizar para ingresar 16 bits. ¿Cual seria el máximo ancho de bits de los operandos para la ALU anterior sin entradas registradas y la diseñada para esta actividad? Piense, analice y discuta otras propiedades de cada diseño.