ELO212: Laboratorio de Sistemas Digitales
Guía de Actividades Sesión 4 (Evaluada)

2 al 6 de mayo de 2023

1. Requisitos de entrada.

Para las actividades de esta sesión debe cumplir con los siguientes requerimientos:

- Haber **completado y entendido** las actividades de las sesiones anteriores, y manejar todos los conceptos fundamentales asociados.
- Manejar los conceptos básicos de aritmética binaria con número finito de bits y representaciones de números sin signo y con signo.
- Haber revisado previo a la sesión el video *capsula_sesion_4.mp4*, disponible en el repositorio de clases. Este video contiene información crítica para la correcta entrega de su diseño.

2. Objetivos.

Los objetivos para esta sesión son:

- Consolidar el trabajo realizado en sesiones anteriores para la descripción y síntesis de circuitos y sistemas digitales, considerando especificaciones y requerimientos técnicos concretos, tanto funcionales como no funcionales.
- Experimentar con la implementación de circuitos aritméticos y las consideraciones necesarias al trabajar con operandos enteros con representación limitada por el número de bits.

3. Indicaciones generales.

Las actividades planteadas en esta guía serán revisadas durante la sesión de laboratorio correspondiente a su paralelo. Se recomienda encarecidamente que adelante su trabajo y llegue al menos con las partes fundamentales probadas. La estadística histórica indica que no alcanzará a terminar el trabajo si no llega con algo avanzado a la sesión.

Recuerden que todo diseño debe empezar con un plan y una descripción sobre qué es lo que deben implementar. En el contexto de la asignatura, esto implica realizar un diagrama de alto nivel de los módulos que necesitarán (cajitas), y como se conectarán entre ellos (flechitas que conectan

las cajitas). Una vez tengan eso, su problema está en gran parte resuelto. No toque la herramienta de diseño ni escriba una línea de código sin haber realizado este paso previo. Se recuerda que **las actividades no se revisarán y tampoco se responderán consultas si no tiene un diagrama de bloques de su diseño** sobre el que se pueda discutir. El pensar que *"tengo todo en mi cabeza, solo necesito unos minutos para escribirlo"*, es un salto directo al fracaso.

3.1. Sobre el proceso de revisión de actividades.

Para el proceso de revisión consideren lo siguiente:

- La revisión y evaluación de cada una de las actividades se realizará en dos etapas: (i) una validación básica de requerimientos funcionales mínimos durante la sesión, y (ii) una verificación exhaustiva de requerimientos funcionales y no funcionales posterior a la sesión.
- Durante la sesión, avise al staff cuando complete una o varias actividades para que muestre y explique sus resultados, y responda las preguntas que se le realizarán sobre sus diseños y conceptos asociados. Recuerde que no se atenderán consultas ni se revisará si no cuenta con un diagrama de alto nivel del sistema que está describiendo.
- Como parte de las preguntas se le puede solicitar que muestre los reportes del proceso de síntesis lógica. Debe entender y poder explicar los mensajes entregados por la herramienta para estas etapas. Procure contar con la síntesis actualizada antes de llamar a alguien del staff.
- Cualquier integrante del grupo debe ser capaz de responder las preguntas. Si las respuestas no son satisfactorias, no se revisará la actividad.
- Las actividades serán revisadas en el orden en que están planteadas. No se responderán consultas o revisará una actividad si no ha completado la actividad anterior.
- El staff solo responderá consultas técnicas específicas en base a lo que Ud. está mostrando para asegurarse que cumpla con requisitos mínimos para ser evaluado. No se les corregirán sus diseños ni responderán preguntas del tipo *¿me pueden decir si está bien lo que hice para poder entregarlo?* Ustedes deciden cuando su código esté listo para ser entregado, y es su responsabilidad verificar que su entrega cumple con todas las especificaciones indicadas en la guía.
- Durante la sesión, el staff priorizará la revisión de actividades e interrogaciones por sobre la atención de consultas generales. Para hacer un uso eficiente del tiempo durante la sesión, prefiera agrupar las dudas y tenga el material asociado listo al momento de llamar al staff.
- Durante la sesión, el staff marcará en una planilla cuando Ud. considere una actividad terminada y haya pasado la interrogación. El tener la actividad marcada como revisada en la sesión no entrega puntaje, y solo asegura que cumple con requerimientos mínimos para que entregue sus archivos de diseño y pasar a la siguiente etapa de revisión mediante tests exhaustivos posterior a la sesión. El formato de entrega de sus archivos se indica en la Sección 3.2.
- La verificación exhaustiva se revisará en base al comportamiento de entrada-salida del módulo principal (*top module*) de cada actividad. **Siga cuidadosamente las especificaciones para el módulo principal, usando exactamente los mismos nombres de entrada y salida para los**

pines especificados. Si la guía especifica que la entrada es P, no entregue su diseño con entrada A o p. Si considera que hay requerimientos que no están explícitamente definidos en la guía, puede tomar las decisiones que estime conveniente. Estas decisiones deben quedar documentadas como comentarios en el código, y deben ser consecuentes con ellas. Notar que una decisión de diseño, como su nombre lo indica, es una *decisión*, y como tal debe considerar sus efectos y consecuencias. Notar que esto es distinto a dejar cosas inespecificadas en el diseño para dejar que la herramienta lo interprete como estime conveniente.

- El test exhaustivo se realizará mediante testbenches avanzados que verifican el correcto comportamiento de su diseño en forma automatizada. Para que esto funcione, su diseño debe estar correcto en términos de funcionalidad lógica, apegarse a las especificaciones, y tener las salidas con las polaridades correctas en base a lo indicado en la guía. El procedimiento de revisión incluyen un paso por herramientas de detección de similaridad y plagio en códigos.

- La salida del test exhaustivo es una señal de `PASS` o `FAIL`. Una salida `PASS` indica que cumple con las especificaciones, y la actividad estará correcta y recibirá el puntaje correspondiente. Una salida `FAIL` indica que no se cumple alguna de las especificaciones funcionales o no funcionales (nombre de puertos y señales distintos a los indicados, lógica opuesta a los requerimientos eléctricos, inferencia de latches, error en la entrega de códigos, etc.), y su actividad estará incorrecta. Se reitera que la revisión preliminar realizada por el staff durante la sesión es para comprobar funcionalidad mínima y verificar que entiende lo que está entregando, y es su responsabilidad verificar que se cumplen las especificaciones.

- Una actividad marcada como `FAIL` en el test exhaustivo contará como no entregada a tiempo, cualquiera sea la razón del fallo. Esto enfatiza la revisión **todo o nada**. **No hay puntaje parcial por tener partes de la actividad** o algo que está *"casi funcionando"*, *"que le falte poquito"*, o la favorita del staff *"pero si está todo bien solo que las polaridades están al revés"*.

- La revisión exhaustiva se realizará sobre los archivos que entregó dentro de la sesión. Es su responsabilidad asegurarse de que se enviaron todos los archivos necesarios en sus versiones correctas. No se aceptan reclamos del tipo *"me equivoqué en los archivos que subí, pero ahora puedo enviar los correctos"*, o *"el ayudante o el profe me habían dicho durante la sesión que estaba bueno"*.

- Si una de sus actividades es evaluada como `FAIL` en el test exhaustivo, esta actividad contará como no entregada a tiempo. Este resultado le será informado y deberá entregar la actividad corregida en la siguiente sesión, sea esta evaluada o guiada, aplicando los descuentos por atraso como están definidos en el reglamento.

3.2. Indicaciones de formato para entrega.

Como se mencionó previamente, la evaluación constará de dos partes. La segunda parte constará de una verificación exhaustiva de sus diseños, la cual se realizará sobre los archivos fuente que deberá entregar vía Aula USM, los cuales deben seguir estrictamente las especificaciones que se le indican a continuación:

1. Debe crear una carpeta con el nombre *GPPXX_Sesion4_2023*, donde PP indica el paralelo

(LU, MA, MI, VI) y *XX* el número de grupo en decimal con dos dígitos (01, 02, 03, etc.). Por ejemplo, la carpeta para el grupo 5 del paralelo del Lunes se llamaría `GLU05`.

2. Dentro de dicha carpeta debe crear una subcarpeta para cada actividad con el nombre *ActividadY*, donde Y indica el número de la actividad.

3. Cada subcarpeta debe subir únicamente los archivos denominados *"Design Sources"* (Archivos de Diseño) asociados a cada una de las actividades. Para esta sesión, esto solo considera archivos con extensión *.sv*. Notar que **NO** debe incluir archivos de simulación, constraints, o archivos temporales generados por Vivado.

4. Cada una de las actividades debe contar con un módulo principal (top module) con el nombre *S4_actividadY*, donde *Y* es el número de la actividad correspondiente. Recordar que los nombres de las entradas y salidas deben ser los mismos que se indican en la presente guía (tener en cuenta que SystemVerilog es sensible a las letras mayúsculas, por ende $P \neq p$).

En el repositorio de clases encontrarán una cápsula que expone de manera práctica cada uno de los pasos mencionados anteriormente. Para facilitar su trabajo, en Aula quedará disponible una carpeta base con la estructura y snippets de módulo top para cada actividad. Puede usar esta carpeta para agregar sus archivos fuente y hacer la entrega. Recordar que **será responsabilidad del grupo entregar de manera correcta los archivos**. Ante cualquier duda sobre las indicaciones de entrega, contacte al staff o consulte por Aula **antes de su sesión**.

4. Actividades evaluadas.

Para cada una de las siguientes actividades, se solicita que registren el ***tiempo de trabajo efectivo*** aproximado en horas invertido por su grupo para cada actividad. Esto considera actividades como búsqueda de información, discusión grupal sobre aspectos de diseño, descripción de código, simulaciones, etc. Tiempos de ocio o distracciones en el medio de sesiones de trabajo, si bien son necesarias, no cuentan como tiempo efectivo de trabajo.

4.1. Actividad 1: Multiplexación temporal para múltiples displays. (25 puntos)

Considere un módulo que permita controlar ocho displays de 7 segmentos para mostrar el valor en hexadecimal de un número de 32 bits. La Figura 1 muestra un diagrama de entrada-salida de alto nivel para el controlador a diseñar. Diseñe, describa, sintetice, y verifique mediante simulación la funcionalidad de dicho módulo para controlar los ocho displays incluidos en la tarjeta Nexys4DDR/NexysA7. Las salidas del módulo corresponden a las señales necesarias para mostrar el valor hexadecimal del número de entrada en los ocho displays disponibles en la tarjeta, la cual se obtiene agrupando los bits de entrada en grupos de 4 bits, desde el menos significativo al más significativo, y generando un símbolo hexadecimal por cada grupo que se mapea a uno de los displays. Para esto, extienda la representación estándar BCD 8421 para mostrar los siguientes símbolos hexadecimales A, b, C, d, E, F (debe usar esta codificación de mayúsculas y minúsculas para mostrar en los displays).

Para que los números se muestren correctamente en el display múltiple con entradas compartidas, debe utilizar el concepto de *Time Division Multiplexing* (TDM), ya revisado en sesiones anteriores.

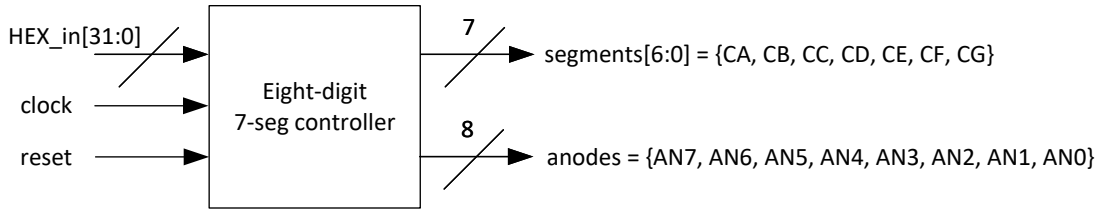

Figura 1: Diagrama de alto nivel para el controlador de display de 7 segmentos incluido en la tarjeta Nexys4DDR/NexysA7. Las señales de salida concuerdan con las señales descritas en el datasheet de la tarjeta.

Su módulo para esta actividad debe contener las señales de entrada y salida indicadas en la Figura 1, **utilizando exactamente los mismos nombres de señales y el mismo orden en la declaración de las señales multibit**. Si considera necesario agregar señales adicionales, agréguelas y explíquelas en su código. La señal de reset es una señal sincrónica global que al activarse lleva la salida de todas las componentes reseteables a un valor por defecto, que en este caso será 0. Notar que el reset no se usa solo al "encenderse" el sistema, y puede activarse en cualquier momento. Asuma que el reloj de entrada `clock` viene ajustado a la frecuencia necesaria para que el número se vea estático en los displays al iterar la activación de los ánodos en cada ciclo de reloj.

Revise cuidadosamente el datasheet para determinar la polaridad necesaria en las señales para encender los displays (notar que no siempre un 1 significará que algo está encendido).

4.2. Actividad 2: Contador de N-bits. (25 puntos)

Describa un módulo parametrizado para un contador de N-bits (por defecto $N = 32$), donde N es un parámetro que es posible especificar al momento de instanciar el módulo en un módulo de más alto nivel. La idea es que utilizando el mismo código, sea posible instanciar contadores con distinto ancho de bits.

El contador debe cumplir con las siguientes especificaciones:

- Contar con pines de entrada para las señales `clock` (reloj) y `reset`, y pines de salida para la señal `counterN`. El reset del contador es sincrónico, activo en alto, y lleva el valor del contador a 0.
- Contar con un pin de entrada para la señal `dec` de un bit, la cual especifica el modo de operación del contador. Si la entrada `dec` está en alto al momento de llegar un canto de reloj, entonces el contador opera en forma decremental (restando una unidad al valor actual). Si la entrada `dec` está en bajo al momento de llegar un canto de reloj, entonces el contador opera en forma incremental.
- Contar con un pin de entrada llamado `enable` de un bit. Si `enable` está en alto al momento de llegar un canto positivo de reloj, entonces el contador opera incrementando/decrementando el valor de la cuenta; en caso contrario, el contador mantiene el valor anterior.
- Contar con pines de entrada para la señal `Load_Ref_value` de N bits.

- Contar con un pin de entrada `load` de un bit. Si la entrada `load` está en alto al llegar un canto positivo de reloj, entonces el contador se actualiza con el valor seteado en la entrada `Load_Ref_value`. Si la entrada `load` está en bajo al llegar un canto positivo de reloj, entonces el contador funciona normalmente.
- Contar con un pin de salida `threshold` de un bit. La salida `threshold` toma un valor alto solo si el valor actual del contador es mayor al valor seteado en la entrada `Load_Ref_value`, manteniendo un valor bajo en cualquier otro caso.
- La señal de `reset` siempre tiene prioridad sobre cualquier otra entrada. Además, la señal `load` tiene prioridad sobre `enable`, es decir, la carga de un valor para el contador se aplica independiente del valor de `enable`. Notar que el sistema es completamente sincrónico, por lo tanto, todos los cambios se deben evaluar y aplicar en cantos positivos de reloj.

Al finalizar esta actividad solo debe entregar la descripción que abarque todas las especificaciones indicadas. Sin embargo, para el proceso de diseño se recomienda planear, describir, y verificar en forma incremental, partiendo con las especificaciones básicas de un contador *free-run* genérico, como lo analizado en las sesiones anteriores, e ir agregando funcionalidad por partes, verificando que cada característica funciona (y la entiende) antes de agregar nuevas funcionalidades. Recuerde verificar los warnings y errores en cada paso, asegurándose que no hayan latches inferidos.

Para la interrogación, debe ser capaz de reconocer las señales involucradas en cada uno de dichos pasos y responder de manera correcta las preguntas que se le realicen a partir de esto. Además, esté preparado para responder las diferencias entre un reset sincrónico y un reset asincrónico.

4.3. Actividad 3: Unidad Aritmética y Lógica básica. (50 puntos)

Una *Arithmetic and Logic Unit* (ALU) [1] es un circuito combinacional que permite realizar distintas operaciones aritméticas enteras y lógicas sobre dos operandos de entrada. Las ALUs son una parte fundamental de cualquier arquitectura de microprocesadores modernos, y trabajarán con ella en diversos cursos de especialidad. La Figura 2 muestra un diagrama de alto nivel de una ALU genérica, con sus principales entradas y salidas. Una ALU básica tiene como entradas dos operandos de M bits y una señal de selección `OpCode` que indica la operación a realizar sobre los operandos ingresados (el número S de bits del `OpCode` dependerá del número de operaciones distintas que puede realizar la ALU). La salida de la ALU es el resultado de la operación, que tiene el mismo número de bits que los operandos. Además, normalmente se incluyen bits de *status flags* que entregan información de excepciones para la última operación realizada de acuerdo a la siguiente descripción (para mayores detalles, puede revisar el link anterior, cualquier libro de sistemas digitales o arquitectura de computadores, o hacer una búsqueda simple en la web):

- V entrega información cuando los operandos de entrada representan números con signo y se realiza una operación aritmética (suma o resta). Si V=1, entonces el resultado excede la capacidad de representación y es incorrecto. A modo de ejemplo, la Figura 4 ilustra las condiciones para la activación del flag V para números de 4 bits. Si al moverse desde A hacia B se cruza la línea roja, entonces se activa el flag de overflow. En este caso el valor del flag C no entrega información y es irrelevante.

[1] https://en.wikipedia.org/wiki/Arithmetic_logic_unit

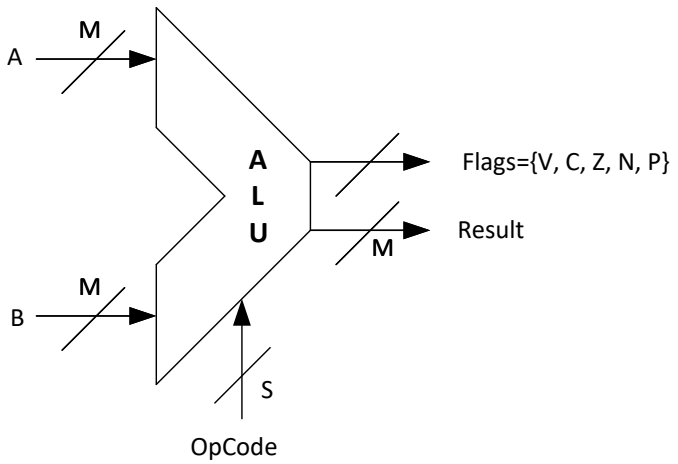

Figura 2: Diagrama de alto nivel de una ALU. Los operandos y resultado son de M bits. La señal `OpCode` especifica la operación a realizar sobre los operandos, la cual viene especificada por medio de una tabla (el número de bits S depende del número de operaciones que la ALU puede realizar). La ALU también tiene una salida de bits de flags que indica propiedades de la operación (por ejemplo, si el resultado es cero, si hubo overflow, etc.).

- C entrega información cuando los operandos de entrada representan números sin signo y se realiza una operación aritmética (suma o resta). Si C=1, entonces hubo un carry o borrow en el bit M (fuera del bit más significativo del resultado) y el resultado de la operación es incorrecto. A modo de ejemplo, la Figura 3 ilustra las condiciones para la activación del flag C para números de 4 bits. La operación $A + B$ indica moverse B pasos desde A en sentido horario, y la operación $A - B$ representa moverse B pasos desde A en sentido antihorario. Si al moverse desde A hacia B se cruza la línea roja, entonces se activa el flag de carry. En este caso el valor del flag V no entrega información y es irrelevante.
- Z es 1 si el resultado de la operación realizada es cero.
- N es 1 si el resultado de la operación realizada es negativo.
- P es 1 si el resultado de la operación realizada contiene un número par de unos.

Notar que la ALU opera sobre números binarios, sin saber si los operandos representan números con signo o sin signo (en realidad, la ALU solo propaga señales eléctricas, no números). Los flags están físicamente cableados a la lógica y siempre generarán un valor de acuerdo a la operación realizada. El usuario es responsable de interpretar el resultado y los flags en base a como está operando.

En esta actividad debe implementar un core de una ALU básica, con ancho de bits de datos parametrizable, de acuerdo a las siguientes especificaciones:

- La ALU tiene dos operandos de entrada A y B con ancho de bits configurables por medio de un parámetro al momento de instanciar el módulo. El ancho por defecto es de 8 bits, pero debe soportar un ancho arbitrario.
- Las operaciones a ejecutar sobre los operandos A y B se configurarán por medio de la señal `OpCode` de dos bits, de acuerdo a lo siguiente:

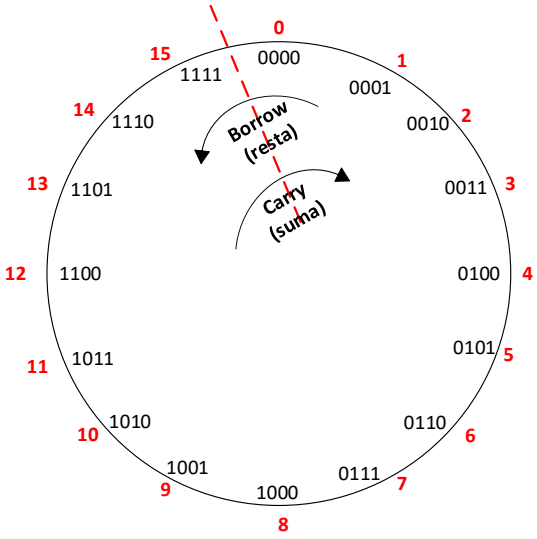

Figura 3: Representación de condiciones para activar el flag de carry para suma y resta en números de 4 bits representando números sin signo. Al sumar se mueve en sentido horario y al restar en sentido antihorario.

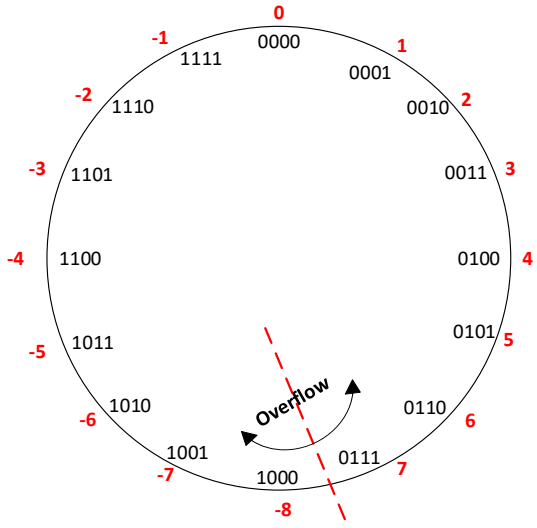

Figura 4: Representación de condiciones para activar el flag de overflow para suma y resta en números de 4 bits representando números con signo en codificación complemento a 2. Al sumar se mueve en sentido horario y al restar en sentido anti-horario.

- Si `OpCode`=00, la ALU ejecutará el NOR bit a bit entre A y B ($\sim$(A|B)).
- Si `OpCode`=01, la ALU ejecutará el NAND bit a bit entre A y B ($\sim$(A&B)).
- Si `OpCode`=10, la ALU ejecutará la suma de A y B.
- Si `OpCode`=11, la ALU ejecutará la resta entre A y B.
- El resultado de todas las operaciones realizadas por la ALU debe tener el mismo número de bits que los operandos.
- La ALU es totalmente combinacional, es decir, debe reevaluar instantáneamente el resultado ante cualquier cambio en alguna de sus señales de entrada, y su salida solo depende de los valores de entrada actuales y no de los anteriores.
- Los flags C y V siempre se mantienen en bajo si la operación no es una suma o resta. Los otros flags siempre muestran su valor independiente de los datos de entrada.

Verifique el funcionamiento de la ALU realizando operaciones de referencia. Busque ejemplos de operaciones que generen condiciones de excepción para verificar el funcionamiento de los flags. La interrogación asociada a la revisión considerará preguntas sobre estas operaciones.