# ELO212: Laboratorio de Sistemas Digitales
# Guía de Actividades Sesión 6

29 de mayo al 2 de junio de 2023

## 1. Requisitos de entrada.

- Haber **completado y entendido** las actividades de la sesión anterior.
- Haber revisado el material entregado en la *capsula_sesion_6* disponible en el repositorio de videos del curso en Aula.
- Estudiar la sección 3.4.1 del libro *Digital Design and Computer Architecture*, de Harris, para tener una idea general de los conceptos de diseño de máquinas de estado. Estos contenidos se ven en detalle en el curso ELO211, y por ahora, para el contexto del curso, basta con tener una idea general sobre las representaciones, formalidades, y terminología utilizada en los diagramas de estado.
- Revisar los contenidos del repositorio GitHub del curso para códigos de referencia (`https://github.com/gcarvajalb/ELO212-reference-modules`).

## 2. Objetivos

- Experimentar con los conceptos de descripción y simulación de máquinas de estado utilizando SystemVerilog y Vivado.
- Experimentar con la descripción de circuitos sincronizadores y anti-rebotes.
- Preparar material base para la siguiente sesión evaluada.

## 3. Actividades guiadas

### 3.1. Revisión ALU de referencia y diseños previos.

En el repositorio Github del curso se encuentra disponible un archivo de SystemVerilog que describe una ALU como la solicitada para la sesión evaluada previa.

Revise el código de referencia y compárelo con la descripción desarrollada y entregada por Ud. en la sesión anterior. Analice y compare el resultado del Elaborated Design de ambas implementaciones. Genere un testbench en donde instancie el módulo de referencia y el desarrollado por Ud. operando en paralelo y bajo el mismo set de estímulos, y compare los resultados.

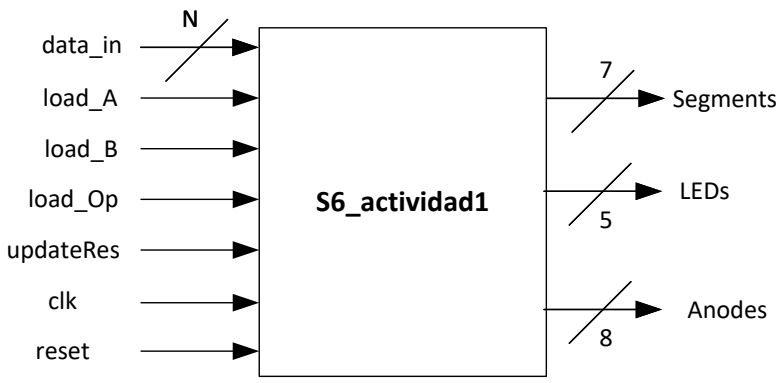


Figura 1: Diagrama de alto nivel para entregable de ALU con entradas y salidas registradas para someter a test exhaustivo.

Notar que el código de referencia es solo eso, una referencia desarrollada por alguien con más experiencia en diseño digital y que ha aprendido los sucios detalles (en base a sudor y lágrimas), y ya ha desarrollado un estilo de codificación ordenado y eficiente. Se enfatiza que esto no representa **la forma correcta de hacerlo**, pero si una forma limpia, eficiente, y autodocumentada. Hay literalmente infinitas formas de describir el comportamiento para llegar a un resultado funcionalmente equivalente, y finalmente la herramienta de diseño es la que se encarga de inferir el hardware necesario para la implementación.

La ALU que estaremos usando en el curso es una versión bastante simplificada de una ALU real. Para conocer más detalles, se recomienda revisar material adicional, para lo cual pueden usar el libro de referencia de Harris, o hacer búsquedas en Internet. Por ejemplo, una posible implementación para una ALU en una CPU se muestra en el siguiente enlace: `https://zipcpu.com/zipcpu/2017/08/11/simple-alu.html`. En particular, la ALU seguirá siendo una pieza fundamental en asignaturas posteriores.

### 3.2. Revisión ALU con entradas registradas de sesión previa.

En esta actividad el staff evaluará mediante un test exhaustivo su diseño para la actividad 4 de la sesión anterior, correspondiente a la ALU con entradas y salidas registradas.

Para esta ocasión, se requieren algunas modificaciones menores al diseño planteado en la guía anterior para cumplir con requerimientos específicos de diseño. Considere como base el diagrama de la Figura 1. La entrada de datos *data_in* es de 16 bits, en base a los switches disponibles en la tarjeta de desarrollo (ver datasheet). A su vez, dado este número de bits de datos, solo se requiere utilizar cuatro displays de 7 segmentos. Es decir, de los ocho displays disponibles en la tarjeta, solo los cuatro menos significativos deben utilizarse mediante multiplexación temporal, con cada display encendiéndose por un ciclo cada cuatro ciclos de reloj. En todo momento, los cuatros displays más significativos deben mantenerse apagados y debe haber a lo más un display encendido (configuración one-hot). La señal LEDs de cinco bits está conectada a los flags $VCZNP$, siendo $V$ el más significativo y $P$ el menos significativo.

Suba los archivos de diseño a Aula siguiendo el procedimiento para la última sesión evaluada. Debe subir una carpeta con el nombre *GPPXX_Sesion6_2023/Actividad1*, donde $PP$ indica el paralelo

(LU, MA, MI, VI) y $XX$ el número de grupo (01, 02, 03, etc.). La carpeta debe contener un top module llamado *S6_Actividad1*, además de todos los archivos con extensión *.sv* asociados a su diseño (ver Figura 1 ). **NO** debe incluir archivos de simulación, constraints, o archivos temporales generados por Vivado. Cuide el uso de mayúsculas y minúsculas en el nombre y señales del módulo top.

Debe subir los archivos dentro del horario de su sesión. Posterior a esto, el staff correrá un test exhaustivo para determinar si la actividad da FAIL o PASS y entregar el feedback correspondiente. Si bien esta actividad no es evaluada, es importante que la complete ya que la correcta implementación de este módulo será parte del trabajo a realizar en la siguiente sesión evaluada. Además, esta actividad no se marcará como terminada hasta que su diseño no genere un PASS. Dicho de otra forma, el PASS en esta actividad es un requisito para ser evaluado en la siguiente sesión evaluada. El plazo de entrega es un deadline, pero puede incluso entregarlo antes de la sesión para tener feedback antes.

### 3.3. Descripción y simulación de conversor level-to-pulse en SystemVerilog.

A partir del ejemplo de diseño y descripción de la máquina de estado de un conversor level-to-pulse entregado en la cápsula introductoria, estudie, describa, y simule la descripción de dicha máquina. Para la simulación, utilice distintos estímulos para verificar y entender el correcto comportamiento de salida. Considere señales de distinta duración (pulsos cortos, pulsos largos, etc.), distintos valores de entrada, etc. Como ejercicio, dibuje en papel y lápiz y previo a la simulación en Vivado, un diagrama de tiempo de estímulos de entrada y calcule los correspondientes valores de salida. Luego puede aplicar los estímulos en la simulación y comparar las salidas de la simulación con las que Ud. obtuvo en su análisis.

Ejecute el proceso de síntesis lógica y analice los reportes para verificar la interpretación que hace la herramienta de diseño de la descripción de su máquina de estados y la codificación utilizada para los estados.

Asegúrese de entender muy bien el proceso de descripción, análisis, y funcionamiento de la máquina descrita. ¿Que aplicación práctica le ve a dicha funcionalidad y en que tareas del día a día podría resultar útil?

### 3.4. Descripción y simulación de controlador de semáforo en SystemVerilog.

Repita las tareas de la actividad anterior para el ejemplo del semáforo descrito en la cápsula. Este ejemplo también lo puede encontrar descrito en la Seccion 3.4.1 del libro *Digital Design and Computer Architecture*, de Harris.

Muestre la simulación, responda las preguntas del staff, y también resuelva sus dudas con el staff.

### 3.5. Revisión de ejemplos y templates.

Como pueden haber notado de la cápsula introductoria y de las actividades anteriores, partiendo de una estructura genérica es posible obtener una descripción particular ajustando los nombres de señales y condiciones para cambios de estados y salidas.

En la práctica, y una vez asimiladas algunas convenciones y buenas prácticas de diseño con SystemVerilog, la complejidad del diseño con máquinas de estado estará en obtener un diagrama de estados funcional y formalmente correcto (ver slides de clases asociadas), el cual después se escribe en forma textual en HDL.

En el repositorio Github del curso se encuentran disponibles distintos ejemplos de descripción de máquinas de estado, incluyendo ejemplos completos y templates con *snippets* de código en SystemVerilog que puede utilizar como base para describir sus propias máquinas de estado. Los snippets entregan estructuras genéricas que incorporan las convenciones de buenas prácticas que hemos estado reforzando, y los puede utilizar para describir sus máquinas de estado. Los snippets incluyen máquinas de Moore, Mealy (no las vemos en detalle en el curso, pero se estudian en ELO211), y un tipo de estructura llamada *timed state machine*.

Revise cuidadosamente estos códigos y haga las consultas que considere necesarias sobre su utilización. Enfoquese en particular en las *timed state machines*, obteniendo un diagrama de bloques estructural que describa el funcionamiento a partir del snippet de código.

## 3.6. Sincronizador y debouncer para pulsadores electromecánicos.

Normalmente, los sistemas digitales con interfaz humana cuentan con componentes electromecánicas que permiten la interacción entre el usuario y el sistema digital (botones, switches, etc.). Por muy simple que esto parezca, existen dos problemas fundamentales con este tipo de interfaces: (i) los sistemas electromecánicos son análogos y por lo tanto bastante susceptibles al ruido, siendo incapaces de generar ”pulsos limpios” como los que esperamos en un sistema digital; y (ii) el ser humano, incluso en sus mejores momentos, opera mucho más lento que un sistema digital (Hz versus MHz/GHz) y no tiene un reloj que le permita definir en que momento presionar las componentes, por lo que desde el punto de vista del sistema digital las entradas pueden cambiar en cualquier momento y tener una duración arbitraria.

Investigue sobre los fenómenos asociados a la captura de señales generadas por componentes electromecánicos que se utilizan como entradas a sistemas digitales. En general, la captura de estas señales requiere de etapas de sincronización y filtro de ruido o rebotes. En particular, puede buscar por el término *push button debouncer*. Analice las causas, efectos, y como resolver este problema de forma digital, asumiendo que el sistema recibe una señal ruidosa [1].

Luego de investigar y tener claros las causas y efectos de los fenómenos asociados a la interacción de pulsadores electromecánicos como entradas a un circuito digital sincrónico, obtenga desde el repositorio Github del curso los archivos de la carpeta `PushButton_debouncer`, la cual contiene un proyecto con módulos que implementan la función de sincronización y filtro de señal para las señales digitales generadas por un pulsador mecánico típico.

Analice cuidadosamente el código basado en descripción por FSM. A partir del código, identifique las entradas y salidas, obtenga el diagrama de estados (el diagrama debe estar formalmente correcto), y analice la funcionalidad de los distintos parámetros del módulo. Considerando que el reloj base de la máquina de estados es de 100 Mhz, determine un valor adecuado del parámetro `DELAY` para que pueda capturar correctamente la presión de un botón de la tarjeta de desarrollo (el valor

[1] Se hace esta diferencia pues hay formas de mitigar estos efectos por medio de electrónica externa acoplada al periférico, de tal forma que el sistema digital reciba un pulso limpio. Como referencia práctica, pueden buscar la diferencia de características y precios entre botones pulsadores y switches de distinta calidad.

por defecto de `DELAY` es intencionalmente pequeño para facilitar la simulación funcional, pero no funcionará en la práctica).

La máquina de estados para este sistema incorpora el concepto de *timed FSM*, en la cual la transición de estados se realiza luego de cumplirse un tiempo predefinido (ver actividad anterior). Esto también muestra el concepto de *factorización de máquinas de estado*, con el cual el diseño de una máquina de estados potencialmente compleja se puede simplificar interconectando dos máquinas mucho más simples. En este caso se tiene una máquina de estados principal que determina la secuencia a seguir en base a valores de los pulsos de entrada, y un contador secundario (que también puede representarse por una máquina de estados) cuya salida se conecta como entrada a la máquina principal y permite especificar tiempos en los que se debe mantener cierto estado. Tratar de integrar ambas funcionalidades en un solo diagrama de estados se vuelve bastante complejo (haga la prueba).

Como referencia, también se provee una descripción alternativa funcionalmente equivalente para un debouncer, pero que no utiliza máquinas de estado. Analice y compare ambos códigos. Realice una simulación funcional para verificar que ambas descripciones son equivalentes (mismas salidas para mismos valores de entrada). Notar que no necesariamente una forma de describir la funcionalidad requerida es mejor que otra en términos de la implementación final, pero si hay una que es más intuitiva y auto-descriptiva que la otra. Pruebe distintos estímulos y valores de parámetros, y analice, interprete y discuta con sus compañeros los resultados de las simulaciones. Entender muy bien el funcionamiento de estos módulos será fundamental para la siguiente sesión evaluada.

Se enfatiza la importancia de entender muy bien los conceptos asociados, ya que este simple problema para dispositivos de uso cotidiano es representativo de conceptos fundamentales al diseño de todo sistema digital moderno para sistemas a todo nivel de complejidad y campos de aplicación, los cuales se verán más en detalle en este y otros cursos asociados.