Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (II/2018)

Tarea 3

Fecha de entrega: lunes 3 de septiembre de 2018 a las 11:59 AM

Parte programación

Llamamos **máquina o circuito combinacional** a la que tiene la característica de que su salida depende solo de los valores de sus entradas (por ejemplo, un *half-adder*). En contraste a este, existe la **máquina o circuito secuencial**, cuya salida no solo depende de su combinación de señales, sino que también del último estado de esta (por ejemplo, un *flip-flop*).

Forma A

Escriba un programa que, dada la descripción de una máquina secuencial, retorne la descripción de una máquina combinatorial de salidas equivalentes.

El programa debe ser escrito en Python 3.5, debe tener como nombre `sequential_to_combinational.py` y debe tomar dos argumentos: el primero correspondiente a la ruta a un archivo `JSON` que describe el circuito a traducir, y el segundo, la ruta a un nuevo archivo `JSON` que será escrito por su programa y corresponderá al nuevo circuito equivalente.

Forma B

Escriba un programa que, dada la descripción de una máquina secuencial y una secuencia de entradas, simule su ejecución y retorne el resultado correspondiente en cada iteración.

El programa debe ser escrito en Python 3.5, debe tener como nombre `sequential_simulator.py` y debe tomar dos argumentos: el primero correspondiente a la ruta a un archivo `JSON` que describe la máquina secuencial, y el segundo, la ruta a un nuevo archivo `JSON` que será escrito por su programa y corresponderá a los valores de los estados y señales de salida en cada iteración.

Formato

A continuación, se muestra un ejemplo del archivo de entrada `JSON` que podría recibir su programa, tanto para la forma A como para la forma B:

```
{
    "sequential": [
    {"from": "input1", "to": "ffd1", "gate": "C"},
    {"from": "ffd1", "gate": "Q", "to": "not1"},
    {"from": "not1", "to": "ffd1", "gate": "D"},
    {"from": "not1", "to": "out1"}
    ],

    "input":{
        "input1": [0,1,0,0,0,1]
    }
}
```

Este es correspondiente a la componente de la figura 1:

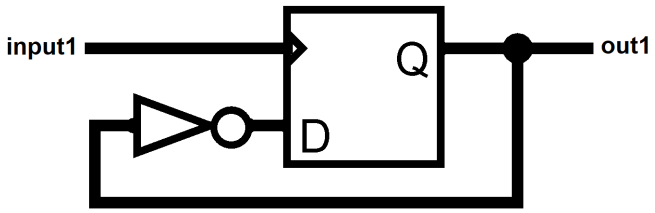

Figura 1: Componente equivalente al `JSON` anterior.

Resultado - Forma A

En este caso, es necesario retornar un archivo `JSON` con la misma nomenclatura, pero con atributo "*combinational*":

```
{
    "output": [
    ...
    ]
}
```

Puede ignorar el atributo "*input*" del `JSON` de entrada, ya que no hará uso de este.

Resultado - Forma B

En este caso, es necesario que simule la ejecución de la componente según el *input* en cada iteración. Podrá notar que es un *hash* donde cada atributo es una lista conteniendo los valores de cada señal de *input* en cada iteración. Siguiendo este mismo formato, debe retornar un *hash* con la lista de valores de cada *output* y estado en cada iteración. Esto es:

```
{
    "output": {
        "out1": [0,1,1,1,1,0],
        "ffd1": [0,1,1,1,1,0]
    }
}
```

La convención de entradas y salidas será "`input`{i}" y "`out`{j}" respectivamente, siendo i y j sus numeraciones para diferenciar las señales entre sí. Por otra parte, la convención de las compuertas es simplemente el uso de sus nombres con su respectiva numeración: "`and`{k}", "`or`{l}", "`xor`{m}", "`not`{n}". Finalmente, los *flip-flops* (que son exclusivamente de tipo D) siguen la nomenclatura "`ffd`{s}", siendo s la numeración correspondiente. **Esta convención es equivalente para ambas formas de esta tarea.**

Parte práctica

Utilizando el proyecto base de Vivado llamado **Proyecto Base**, que puede encontrar en el Syllabus, construya tres componentes en VHDL correspondientes a diferentes tipos de *flip-flops*, para luego utilizarlos en un caso práctico.

Descripción de los componentes de *flip-flop*

Para comenzar, deberán modelar 3 componentes que correspondan a *flip-flops* SR, JK y D. En la figura 2 se ve un ejemplo del circuito del *flip-flop* SR, en la figura 3 se ve un ejemplo del circuito del *flip-flop* JK y por último, en la figura 4 se ve un ejemplo del circuito del *flip-flop* D. Es importante recalcar que estos circuitos son ejemplos y que se pueden modelar usando otras compuertas además de `nand`. Para esta entrega deberán generar componentes para cada uno de los *flip-flops* mostrados.

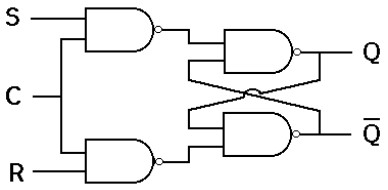

Figura 2: *Flip-flop* SR.

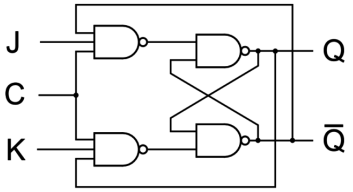

Figura 3: *Flip-flop* JK.

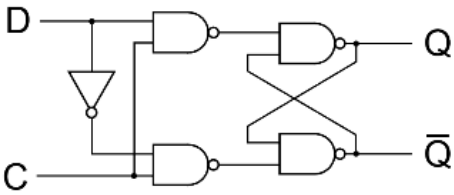

Figura 4: *Flip-flop* D.

Componente Serial

A continuación, deberán crear un componente para poder corroborar el funcionamiento correcto de sus *flip-flops* D. Deberá utilizarlos para conectar 16 de estos en serie como se muestra en la figura 5. Esto quiere decir que, la salida `Q` del primer *flip-flop* corresponderá a la señal de control del segundo, siguiendo el mismo comportamiento de forma sucesiva hasta el último. Además, el estado de cada uno se irá alternando según su propia frecuencia para así simular una señal de control.

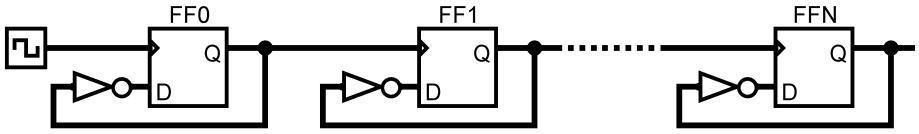

Figura 5: Serie de *flip-flops* D conectados en serie. En este caso, $N = 16$.

Descripción del componente general

Después de crear los componentes de los *flip-flops* y el serial, se debe generar un componente que los ocupe para lograr el siguiente comportamiento:

- Se usará el *switch* 0 para definir si se usa la señal de control del sistema (llamada `clock`) o se usa el botón central como señal de reloj. Si el *switch* está abajo, se usa la señal de control del sistema. Si el *switch* está arriba se usa el botón central.
- Al recibir la señal del botón superior, debe trabajar con el *flip-flop* SR. Debe recibir los *switches* 15 y 16 para los valores de S y R respectivamente. Debe mostrar en los *leds* 0 y 1 los valores de `Q` y $\overline{\texttt{Q}}$ respectivamente.

- Al recibir la señal del botón derecho, debe trabajar con el *flip-flop* JK. Debe recibir los *switches* 15 y 16 para los valores de J y K respectivamente. Debe mostrar en los *leds* 0 y 1 los valores de `Q` y $\overline{\texttt{Q}}$ respectivamente.
- Al recibir la señal del botón izquierdo, debe trabajar con el *flip-flop* D. Debe recibir el *switch* 16 para el valor de D. Debe mostrar en los *leds* 0 y 1 los valores de `Q` y $\overline{\texttt{Q}}$ respectivamente.
- Al recibir la señal del botón inferior, deben trabajar con una conexión serial de 16 *flip-flops* D, siguiendo lo mostrado en la figura 5. Debe conectar el estado `Q` de cada *flip-flop* de la secuencia a un *led* de la placa en el mismo orden, de forma que se pueda observar la frecuencia de cada uno de estos

Este componente debe estar instanciado en `Basys3.vhd`. Debe recibir las señales `clock`, `sw` y `btn`. Debe tener señales de output que se conecten a los *leds*. Por último, al apretar un botoón, debe mantener el comportamiento inlcuso cuando no está siendo apretado. El comportamiento cambia sólo cuando se apreta un botón diferente.

Funcionalidades de la placa

A continuación, se muestra un diagrama con todas las funcionalidades de los elementos de la placa para esta tarea:

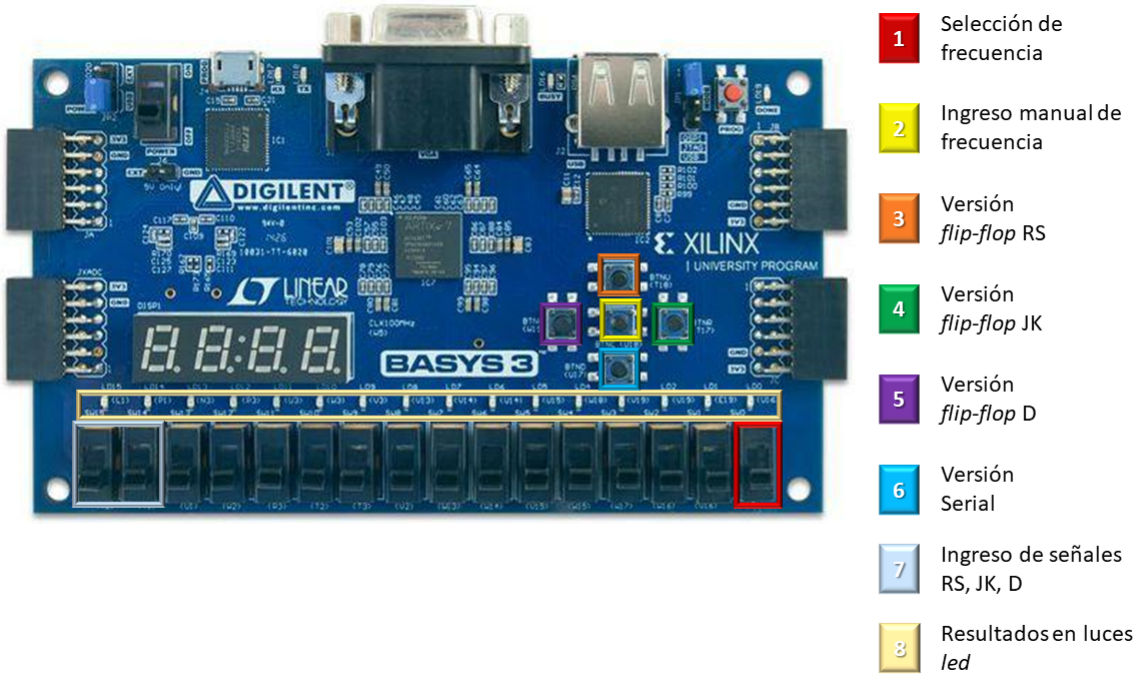

Figura 6: Diagrama de uso de la placa.

Entrega y evaluación

La tarea se debe realizar de **manera individual** y la entrega se realizará a través de GitHub. El formato de entrega para cada modalidad será:

- **Programación:** Archivo `sequential_to_combinational.py` o `sequential_simulator.py` y los extras necesarios para ejecutar su programa correctamente.
- **Práctica:** Archivo `.bit` sintetizado para programar la placa, además del proyecto de Vivado que pueda generar la síntesis.

El repositorio debe contar con un `README.md` escrito en *Markdown* que identifique sus datos, la versión de la tarea realizada y consideraciones que el corrector deba tomar en cuenta, tales como número de placa utilizado, sistema operativo usado para realizar la tarea, paquetes de LaTeX empleados, etc.

Archivos que no compilen o que no cumplan este formato de entrega implicarán nota **1.0** en la tarea. En caso de atraso, se aplicará un descuento de **1.0** punto por cada 6 horas o fracción.

Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno (grupo) copia un trabajo, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir reprobación del curso y un procedimiento sumario. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.