DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (I/2023)

Tarea 1

Respuestas sin desarrollo o justificación no tendrán puntaje.

Publicación: Miércoles 22 de Marzo 10:00
Entrega: Miércoles 5 de Abril 20:00

Instrucciones

- Lea atentamente el enunciado. Responda cada pregunta en un PDF por separado. Ponga su nombre y número de alumno/a en cada pregunta.
- La entrega de la tarea será a través de **canvas**.
- La tarea debe ser en formato **digital** y no manuscrito, a excepción de los diagramas de circuitos lógicos. Estos pueden ser imágenes de su dibujo, pero debe ser **claro** y **ordenado**. De no cumplir esto, **no** se revisará dicha pregunta.
- Cualquier duda que tengan respecto a la tarea, preguntar en el foro de clases.
- **Cualquier uso de material externo al curso debe ser citado**.
- Siga el código de honor.

Código de Honor de la UC

"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."

Pregunta 1: Representación de números (12 ptos.)

(a) (4 ptos.) El complemento de 2 para la representación de números en base 2 estudiado en clases cuenta con la limitante de ser desequilibrado al contar con más números negativos que positivos. Comente qué tipo de representaciones posicionales de números enteros, según su base, son equilibradas al aplicar su respectivo complemento. Señale, además, una desventaja que presenten frente a la representación binaria con complemento de 2.

(b) (4 ptos.) Describa una codificación para números enteros binarios distinta al complemento de 2, que utilice bit de signo y tenga una única representación para el cero. No es necesario que en esta codificación se cumpla que $A + (-A) = 0$.

(c) (4 ptos.) Dados números naturales K, N, T, donde $K \geq 2$ y $N > T > 0$, describa un algoritmo para transformar de manera **eficiente** números naturales codificados en base K^N a K^T. No es válido utilizar la fórmula general de transformación de bases $\sum_{k=0}^{n-1} s_k \times b^k$, dado que **no es eficiente** realizar una transformación intermedia para lograr el resultado esperado.

Pregunta 2: Operaciones aritméticas y lógicas (24 ptos.)

(a) (4 ptos.) Un comparador de números es un circuito que, dados dos números A y B en representación posicional, indica cuál es el mayor, o si estos son iguales. El circuito posee tres salidas, donde la primera entrega un 1 solo si A es el mayor, la segunda un 1 solo si ambos son iguales, y la tercera un 1 solo B es mayor. Haciendo uso de las compuertas lógicas vistas en clases, diseñe un comparador de números **enteros** de N bits, explicando la funcionalidad de cada uno de los circuitos que elabore.

(b) (4 ptos.) Construya un circuito que permita detectar la ocurrencia de *overflow* al sumar o restar dos números enteros de 8 bits en una ALU.

(c) (4 ptos.) Construya dos componentes *shift left* y *shift right* de 16 bits que permitan seleccionar la cantidad de *shifts* a realizar en un número. Considere que se admite un máximo de 16 *shifts* para cada componente y que son de tipo lógico, no aritmético.

(d) (12 ptos.) En la década de los 90, los videojuegos eran limitados tanto en aspectos gráficos como en sonido y funcionalidad. En este último aspecto, podemos tomar como ejemplo la generación de números aleatorios (RNG o *Random Number Generator*). Basándonos en videojuegos como *Super Mario World* de la consola *Super Nintendo*[1], podemos crear una función en pseudo-código que describe una forma de generar números al azar:

```
func get_rand(S, T):
  S, T = update_seed(S, T)
  K = S XOR T
  S, T = update_seed(S, T)
  J = S XOR T
  return J, K
end

func update_seed(S, T):
  S = 17S + 1
  if T[4] == T[7] then
    T = 2T + 1
  else
    T = 2T
  end
  return S, T
end
```

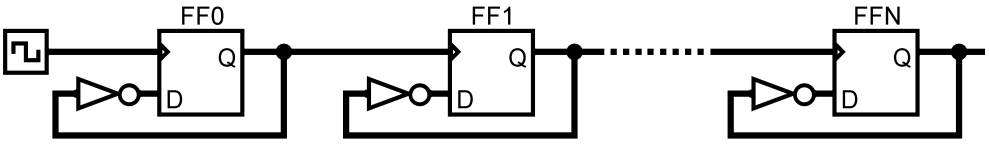
Digamos que queremos tener este mismo generador de *seeds*, pero ahora integrado al *hardware* de nuestro dispositivo. Cree el circuito lógico que permita hacer esto, teniendo como *inputs* S y T (ambos de 16 bits) y *outputs* J y K, también de 16 bits.

Restricciones: Solo se permiten circuitos combinacionales, no se permiten circuitos secuenciales ni unidades de almacenamiento de datos (latch, flip-flop, registros, contadores, memorias). Los únicos circuitos "pre-hechos" que se pueden utilizar (aparte de las compuertas lógicas) son el sumador-restador de N bits, *enablers*, multiplexores y compuertas *bitwise* vistas en clases. Puede, y se recomienda, crear componentes intermedias si así lo requiere su implementación.

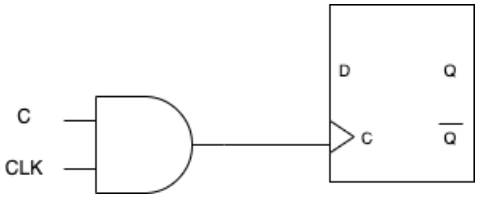
[1] Puede aprender más de esto en el siguiente video.

Pregunta 3: Almacenamiento de datos (24 ptos.)

(a) (4 ptos.) En la siguiente figura, si la frecuencia del *clock* que entra al flip-flop FF0 es de F Hz, ¿cuál es la frecuencia del *clock* del flip-flop FFN? Debe incluir una explicación en su respuesta.



(b) (4 ptos.) Una alternativa para bloquear el almacenamiento en un flip-flop D a partir de una señal de control es unir esta con la señal de flancos (*clock*) a través de una compuerta `AND`, tal como se ilustra en la figura. ¿Le parece esto una buena idea? Comente en base a las implicancias de esta implementación.



(c) (6 ptos.) A partir de la construcción de un registro de 4 bits con señales *Load* y *Reset*, realice un diagrama de esta componente con una modificación para que la combinación de señales `Load = 1, Reset = 1` **preserve el estado del registro**, es decir, que no sea sobreescrito con el bus de entrada ni reseteado.

(d) (10 ptos.) Extienda la idea de la pregunta anterior y elabore un diagrama del contador de 4 bits (con señales *Up* y *Down*) donde cualquier *input* de señales inválido preserve el estado del componente. Se consideran como *input* inválido cualquier combinación de señales que implique más de una modificación sobre el valor registrado. Por ejemplo, `Up = 1, Down = 1` es una combinación inválida ya que el registro no puede incrementar y decrementar su valor al mismo tiempo. Debe entregar una tabla de verdad para las combinaciones de señales *Up*, *Down*, *Load* y *Reset*.