Laboratorio No. 1

Diseño de Sistemas Digitales

Transmission Control Protocol (TCP) – Client

Autor: Santiago Correa Marulanda

C.C: 1033486496

Arquitectura de Computadores y Laboratorio

Fredy Alexander Rivera Velez

22/09/2024

Contenido

INTRODUCCIÓN

El informe presenta el proceso de diseño e implementación de un protocolo de control de transmisión (Transmission Control Protocol, TCP) del lado del receptor, utilizando el simulador de diseño de sistemas digitales “Logisim Evolution”. El funcionamiento del TCP comienza con una gestión de transmisión de mensajes codificados en 16 bits en distintos paquetes utilizando un componente que permite “sacar” palabras por medio de su aleatoriedad, planteado así para comenzar la simulación del lado del cliente, posteriormente estos paquetes serán reorganizados en paralelo, a continuación los datos serán almacenados en otros paquetes para poder evitar la perdida de estos, luego son ordenados para obtener

los datos que son necesarios, es decir, los primeros 2 bits son usados para saber el orden de llegada y los 14 restantes representarán 2 caracteres de 7 bits cada uno, este proceso es para seleccionar los caracteres de los mensajes, poder ser presentados en la terminal o TTY.

OBJETIVO

El objetivo de este laboratorio, es llevar a la practica los conocimientos y teoría adquirida en la asignatura de Arquitectura de Computadores y Laboratorio, para implementarlos en un Sistema de Transmission Control Protocol, haciendo uso de las herramientas que nos brinda el software presentado por el profesor, durante su acompañamiento en el desarrollo de este sistema digital en las clases de laboratorios.

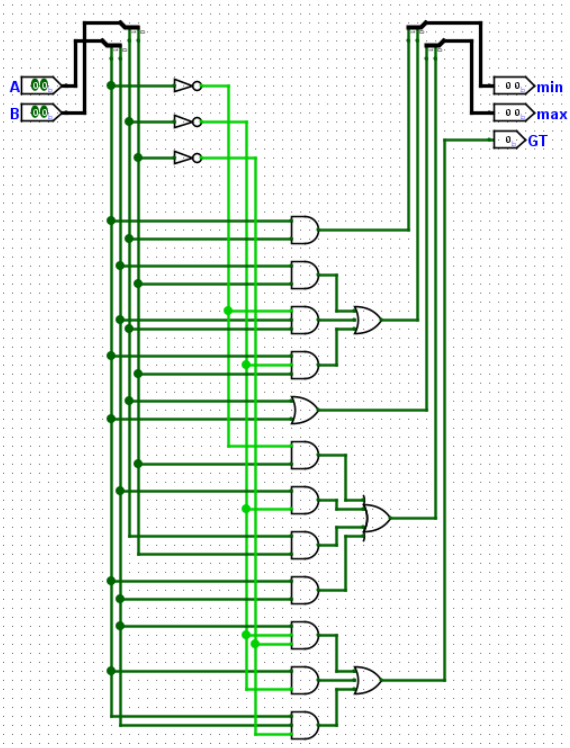
DESCRIPCIÓN

Binary Arithmetic Operation Sorter (BAOS)

Para el comienzo del diseño de este sistema digital, se decide iniciar con un ordenamiento de números binarios, para este caso se tienen 2 entradas de 16 bits por red de ordenamiento, con un proceso que haremos posteriormente, se tomarán únicamente los 2 primeros bits o bits más significativos, que son los que indican él orden de cada carácter del mensaje, cada red lanza tres datos de salida, el dato menor, el dato mayor y el dato que nos indicará cual de los dos anteriores es el mayor. Para poder llevar a cabo este circuito se decidió hacer uso de una tabla de verdad que luego se implementará en el simulador

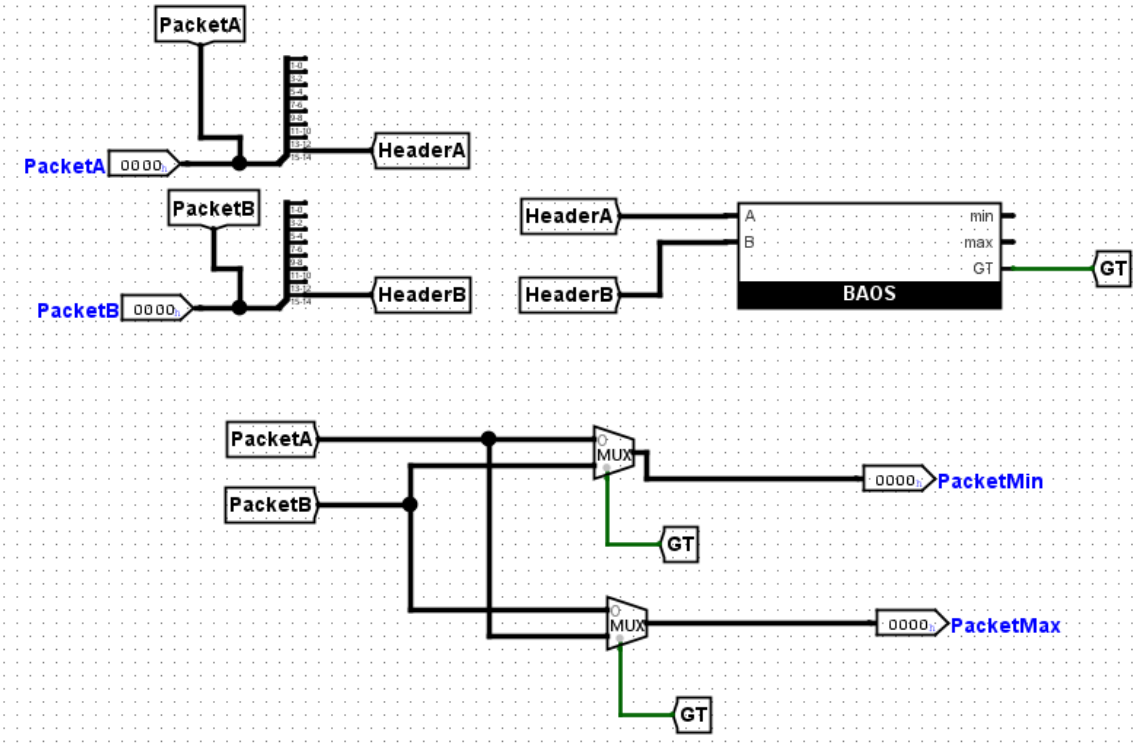
A	B	min	max	GT
0 0	0 0	0 0	0 0	0
0 0	0 1	0 0	0 1	0
0 0	1 0	0 0	1 0	0
0 0	1 1	0 0	1 1	0
0 1	0 0	0 0	0 1	1
0 1	0 1	0 1	0 1	0
0 1	1 0	0 1	1 0	0
0 1	1 1	0 1	1 1	0
1 0	0 0	0 0	1 0	1
1 0	0 1	0 1	1 0	1
1 0	1 0	1 0	1 0	0
1 0	1 1	1 0	1 1	0
1 1	0 0	0 0	1 1	1
1 1	0 1	0 1	1 1	1
1 1	1 0	1 0	1 1	1
1 1	1 1	1 1	1 1	0

La tabla refleja lo anteriormente indicado, 2 entradas A y B, de dos bits cada una, estos bits representan el orden de cada carácter, es decir, si es 00 este será el primer carácter, si es 01 será el segundo carácter y funciona igual para las entradas 10 y 11. En la tabla veremos las 16 diferentes combinaciones de las entradas, las variables o datos de salida son min, max y GT, el dato min tendrá el valor menor entre A y B, max indicará el mayor valor de los mismos, mientras que GT tendrá el valor 1 cuando A sea mayor a B y 0 en caso contrario. Una vez planteada nuestra tabla de verdad, resta implementar esto en el simulador.



Ordenar paquetes como entradas (PacketOrder2Inputs)

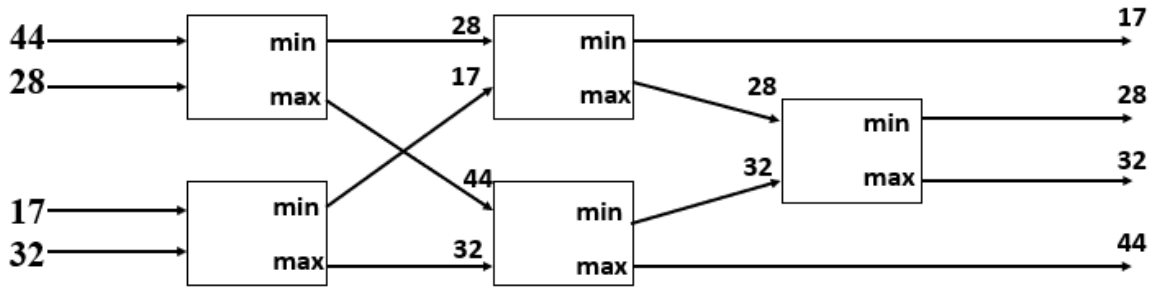
Ya tenemos diseñado el circuito que nos permite ordenar 2 números de 2 bits de forma ascendente, recordar que estos 2 bits son los bits mas significativos de cada cadena de 16 bits que llega, aun falta ordenar los paquetes que contienen los caracteres de un mensaje, para esto hay que diseñar un circuito que tome lo bits más significativos de cada uno de los 2 paquetes de datos de 16 bits, para que estos entren al circuito BAOS, Una vez hecho el proceso debe dar el dato de salida de cual paquete es el mayor y el menor y con esto implementar dos datos de salida, uno para el paquete que va primero y otro para el segundo, llevando esto al simulador obtendríamos algo así.



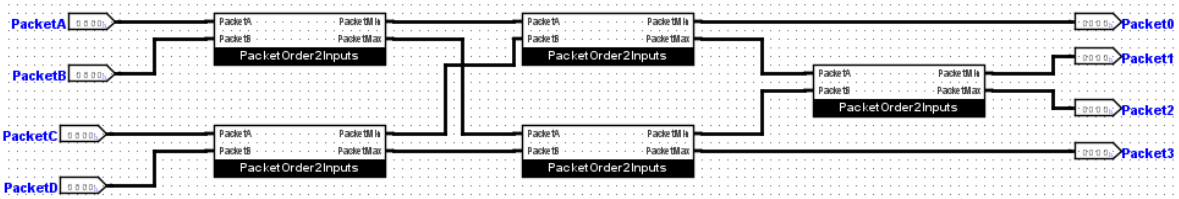
El flujo del circuito sigue el siguiente orden: Recibe 2 paquetes de 16 bits cada uno, los dos primeros bits serán usados para ordenar cual paquete será el primero y cual el segundo, para separar estos bits, haremos uso de dos Splitters o Separadores, conectaremos cada uno a los bits más significativos de cada paquete (14-15) y estos serán llevados al circuito BAOS que ya se diseñó, retornará GT, recordar que este dato nos dice cual paquete es el mayor, si es 1 será el PacketA, si es 0 será el PacketB, luego este valor de GT se conectará a dos multiplexores 2:1 como entrada de control, a estos mismos conectaremos los paquetes A y B como entradas de datos y también unos datos de salida, llamados PacketMax y PacketMin, observe que si GT es 1, el Multiplexor de la parte inferior llevará al PacketA a PacketMax y el multiplexor de la parte superior, llevará al PacketB como PacketMin.

Ordenamiento de paquetes (Packet_Sorter)

Se han diseñado dos sistemas de ordenamiento, pero aún no se han ordenado los caracteres de una palabra completa, para esto se usará un modelo de redes de ordenamiento ascendente basándonos en el siguiente ejemplo.



La anterior imagen es un modelo básico de red de ordenamiento, observe que nuestro circuito nombrado “PacketOrder2Inputs” cumple la función de los cuadrados de la imagen que contienen “min, max”, únicamente debemos conectar cada paquete de entrada a estos circuitos.



Finalmente, tener 4 datos de salida que tendrán los paquetes de 2 caracteres cada uno, totalmente ordenados.

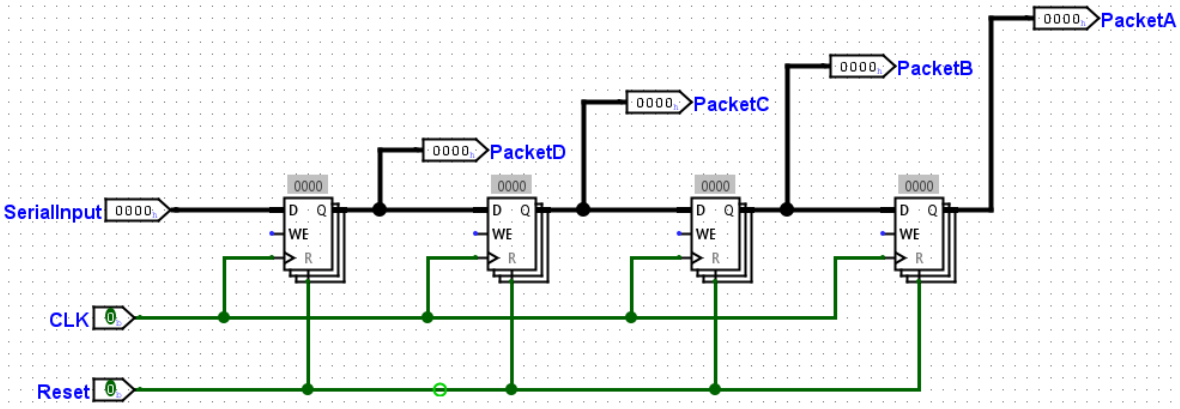
Des encapsulamiento (Decaps)

Esta parte del circuito es la menos compleja, es el circuito que le sigue al anterior (Pakcet_Sorter) y su función es recibir lo 4 paquetes ya ordenados para simplemente descartar lo dos bits mas significativos de cada paquete y únicamente rescatar los 14 restantes, recordar que estos bits son los que tienen los caracteres de las palabras ya que cada carácter es de 7 bits.

Datos en serie a paralelo (Serial2Paral)

En pasos anteriores realizamos el circuito que se encarga de ordenar los caracteres de las palabras, pero aun falta recibir estos, además de esto, recordar que estos datos vienen en serie, es decir, bit a bit y deben ser transformados a datos en paralelo y que podamos recibir la secuencia de bits en un solo dato para así dar uso a el circuito “Packet_Sorter”.

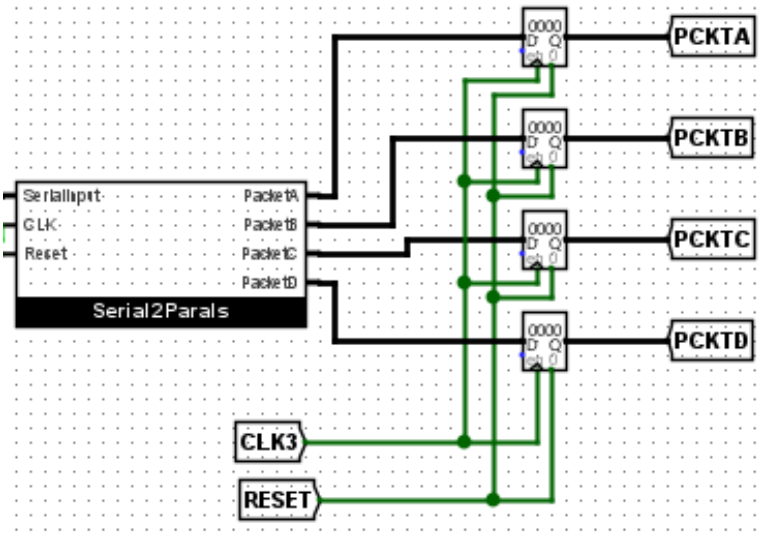
Para esto debemos diseñar un circuito que reciba un dato en serie y lo distribuya a cuatro diferentes paquetes para poder pasarlos al siguiente circuito.



En este circuito, recibimos un dato de entrada SerialInput que pasará por un primer Registro o Register en la entrada Data, que se encargará de ingresar el valor que este Registro debe almacenar, en la entrada Clock de cada registro se conectó el mismo reloj y se hizo lo mismo para la entrada Clear de cada Registro conectando en cada uno la entrada Reset. Ahora la salida Output del registro se conectó con un dato de salida PacketD y a la entrada Data del segundo registro, este proceso lo repetiremos hasta llegar a el ultimo paquete PacketA, de esta manera cada flanco de reloj hará que cada paquete almacene un valor distinto, esto se podrá ver mejor luego durante la simulación.

Almacenamiento seguro de los datos

Luego del proceso anterior, necesitamos de un circuito que se encargue de almacenar los datos de manera segura, manteniéndolos en unos registros para ser usados posteriormente en el circuito ya creado "Packet_Sorter", así evitamos la perdida de datos, permitiendo que el sistema procese la información correctamente, de lo contrario, se perderían algunos datos y se verían las palabras incompletas en la terminal.



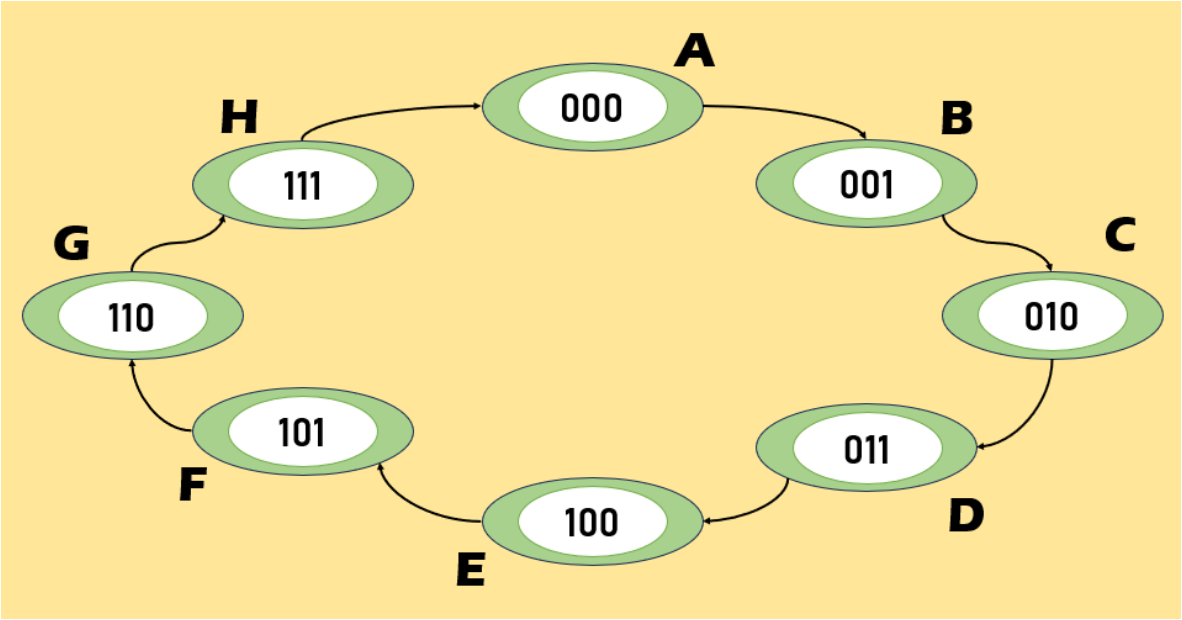
Como se ve en el circuito, está conectado al anteriormente diseñado, cada paquete se almacena en un registro y este dará a los datos de salida PCKTA, PCKTB, PCKTC y PCKTD el dato almacenado, así, estos llegan hasta el final del circuito mostrándose en la terminal y en el momento que la palabra sea escrita, estos registros almacenan los nuevos paquetes de la nueva palabra. Este circuito fue proporcionado por el profesor. (Velez, Adenda #2, 2024)

Contador (FF_Counter)

Luego de des encapsular los paquetes, llega la hora de poder mandar cada carácter a la terminal, ya se tiene el sistema que nos ordenó los paquetes, ahora nos queda crear un circuito que indique cual paquete debe pasar primero y cual le sigue, para esto es necesario un contador modulo 8 que cuente los numero del 0 al 7, ya que son 8 caracteres los que serán impresos esto para que indique a un multiplexor que paquete debe ser impreso, esta idea será esclarecida más adelante.

Para la creación de este contador, primero se ha realizado un diagrama de estados.

Diagrama de estados



Como se nota en el diagrama, el contador inicia en 0 y avanza hasta 7 para luego volver al estado de inicio A.

Teniendo en cuenta que cada flip_flop en un contador es utilizado para representar un bit en un numero binario, para saber cuántos flip_flops necesitamos, basta con saber que tenemos 8 estados diferentes, es necesario saber que exponente de 2 es igual o mayor a 8, este numero es 3 ($2^3 = 8$), por lo tanto, necesitamos de 3 flip_flops.

Para el siguiente paso, se nombró a cada flip_flop como Q2, Q1 y Q0, siendo el bit más significativo Q2, seguido de Q1 y finalmente Q0 como el menos significativo, igualmente, se nombra el estado siguiente de estos como Q2n, Q1n y Q0n respectivamente.

En este punto se puede realizar la tabla de estados y la tabla de estados codificada, en este caso, se realizó ambas al mismo tiempo.

Tabla de estados codificada

	ESTADO ACTUAL			ESTADO SIGUIENTE		
	Q_2	Q_1	Q_0	Q_{2n}	Q_{1n}	Q_{0n}
A	0	0	0	0	0	1
B	0	0	1	0	1	0
C	0	1	0	0	1	1
D	0	1	1	1	0	0
E	1	0	0	1	0	1
F	1	0	1	1	1	0
G	1	1	0	1	1	1
H	1	1	1	0	0	0

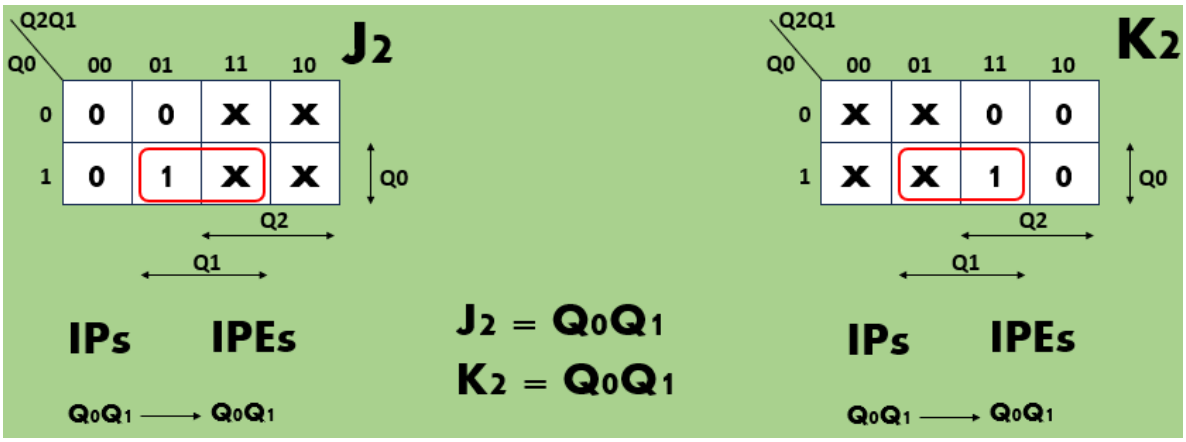
En esta tabla se puede ver el contador con los flip_flops, dando paso a la siguiente tabla de estados codificada y extendida, para realizar esta tabla debemos tener en mente la tabla de excitación del flip_flop a utilizar, en este caso, flip_flops JK.

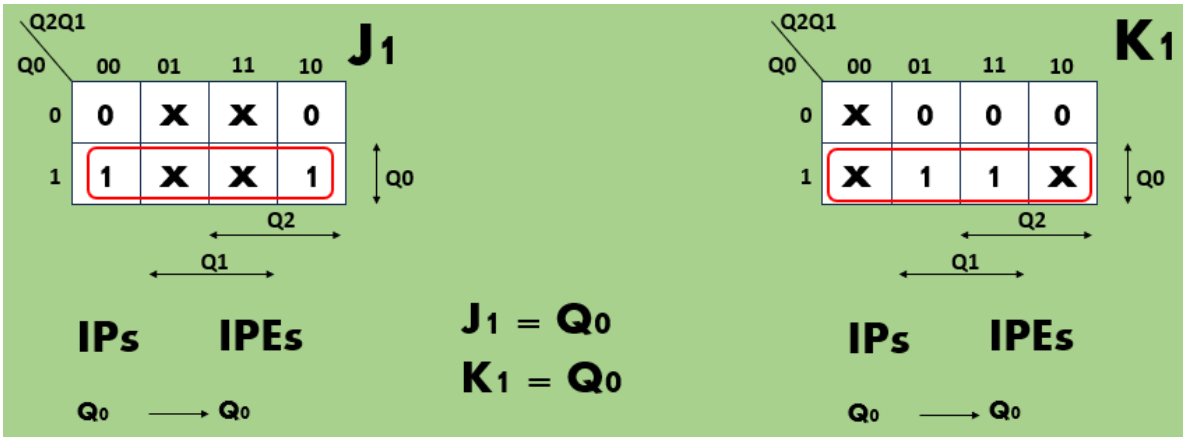
Tabla de estados codificada y extendida

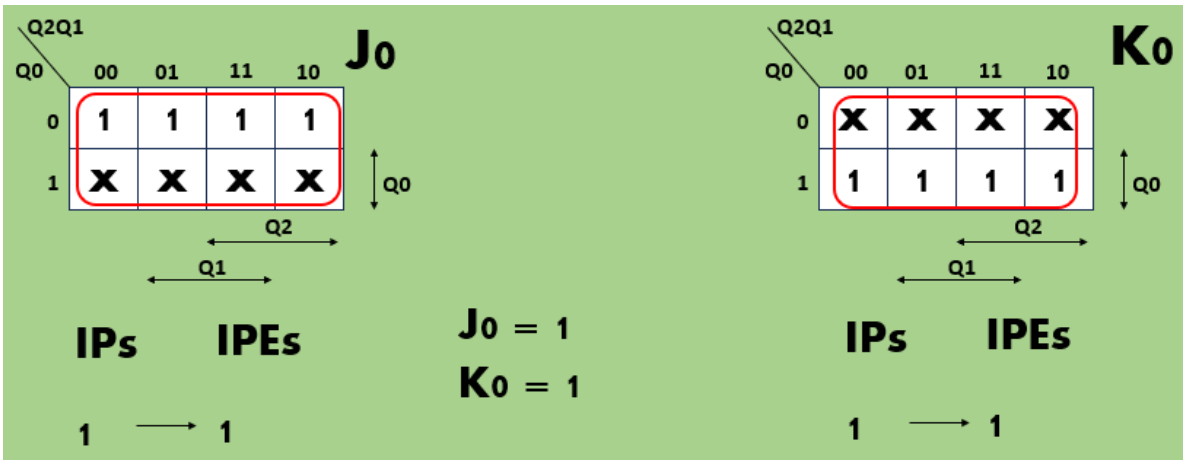
Q	Qn	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

	ESTADO ACTUAL			ESTADO SIGUIENTE								
	Q_2	Q_1	Q_0	Q_{2n}	Q_{1n}	Q_{0n}	J_2	K_2	J_1	K_1	J_0	K_0
A	0	0	0	0	0	1	0	x	0	x	1	x
B	0	0	1	0	1	0	0	x	1	x	x	1
C	0	1	0	0	1	1	0	x	x	0	1	x
D	0	1	1	1	0	0	1	x	x	1	x	1
E	1	0	0	1	0	1	x	0	0	x	1	x
F	1	0	1	1	1	0	x	0	1	x	x	1
G	1	1	0	1	1	1	x	0	x	0	1	x
H	1	1	1	0	0	0	x	1	x	1	x	1

Con esta tabla, se pueden realizar los mapas de Karnaugh para cada J y K, donde indicaremos los implicantes primos y sus implicantes primos esenciales.

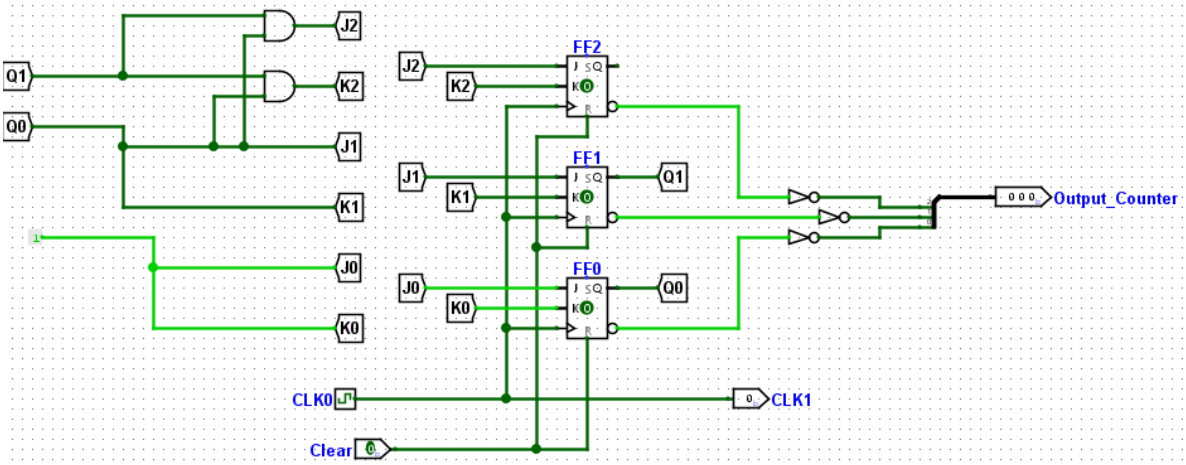






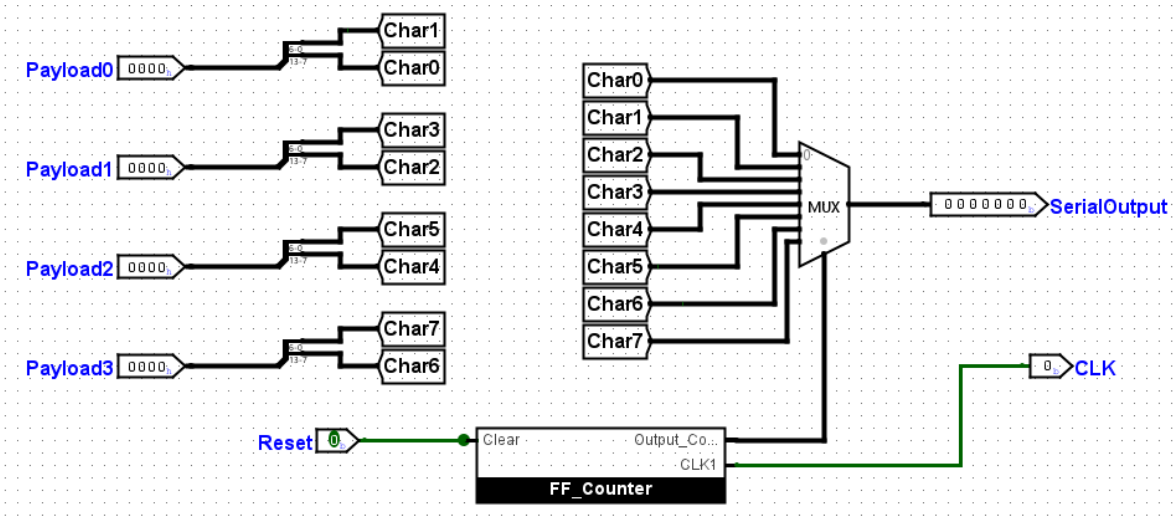
A partir de los mapas de Karnaugh, se ha determinado que la entrada Q2 no es necesaria, ya que la simplificación que aplicamos, revela que agregarla no aporta al funcionamiento del circuito, además, J0 y K0 serán conectados con una constante 1, ya que vemos que en ambos, el mapa K puede ser representado por completo como un IPE, resta implementar el circuito, usando 3 flip_flops JK.

Las conexiones son sencillas: En la entrada Clock conectamos un reloj, en Clear un Reset y en J, K y Q conectamos sus respectivos datos, y como dato de salida, con ayuda de un Splitter llevamos los bits dados por los Flip_flops a un Output_Counter, dando como resultado el siguiente circuito



Paralelo a serie (Paral2Serial)

En este circuito daremos uso al contador anteriormente implementado. Cabe recordar que el circuito “Decaps” nos ha entregado 4 paquetes de 14 bits cada uno, conteniendo cada uno 2 caracteres, estos ya están ordenados, necesitamos diseñar un circuito que separe cada paquete de palabras en dos partes, cada parte tendrá un carácter, luego de esto, el contador “FF_Counter” le indicará al un multiplexor, que carácter debe mostrarse en la terminal TTY.



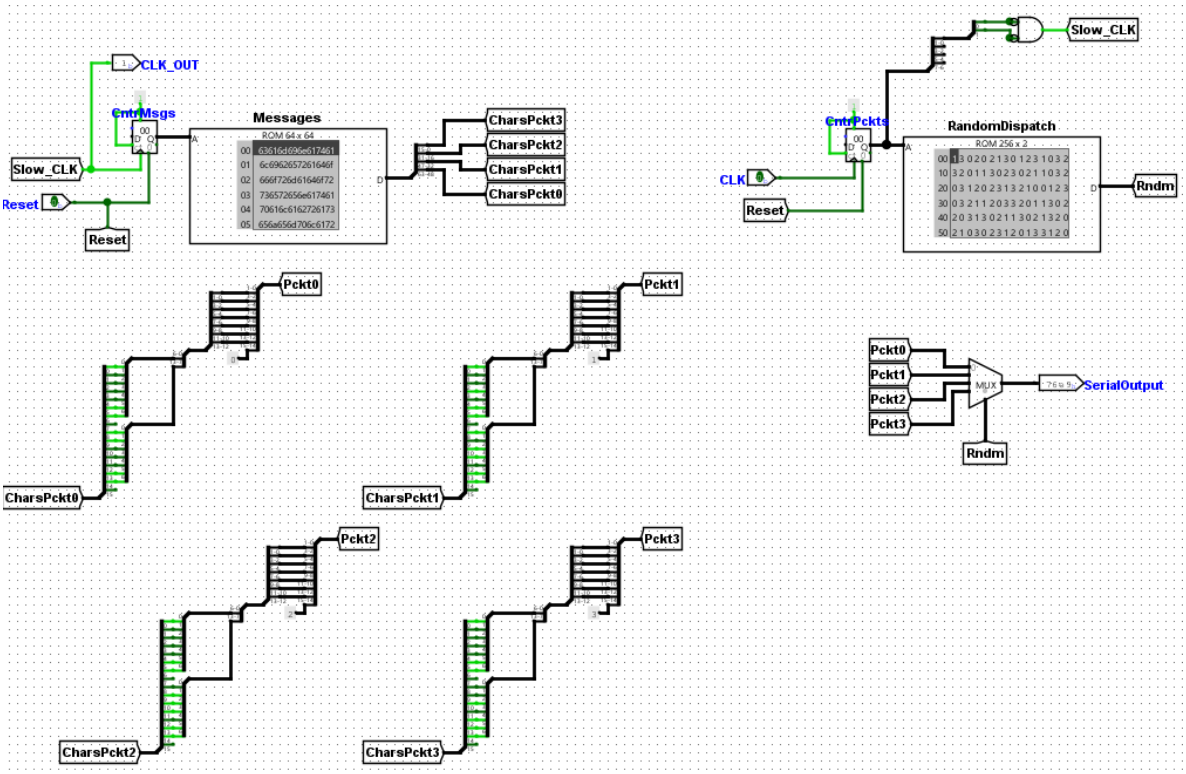
En el circuito se observa el uso del contador diseñado. Para comenzar, los cuatro paquetes o Payloads que se reciben del circuito “Decaps”, con ayuda de un Splitter por Payload, separamos los dos carácter que cada uno tiene en dos distintos datos de entrada que llamaremos Char0, Char1,…, Char7, siendo Char0 los 7 bits mas significativos del payload0 y los 7 restantes serán Char1, el proceso es similar para cada Payload.

A continuación, conectamos el contador como dato de entrada de control en el multiplexor que igualmente tiene como conexiones los caracteres de las palabras. El funcionamiento del contador hará que pasé al dato de salida el caracter0, seguido de caracter1 y así sucesivamente, este dato de salida será conectado a la terminal.

Emisor de mensajes (Message_Dispatcher)

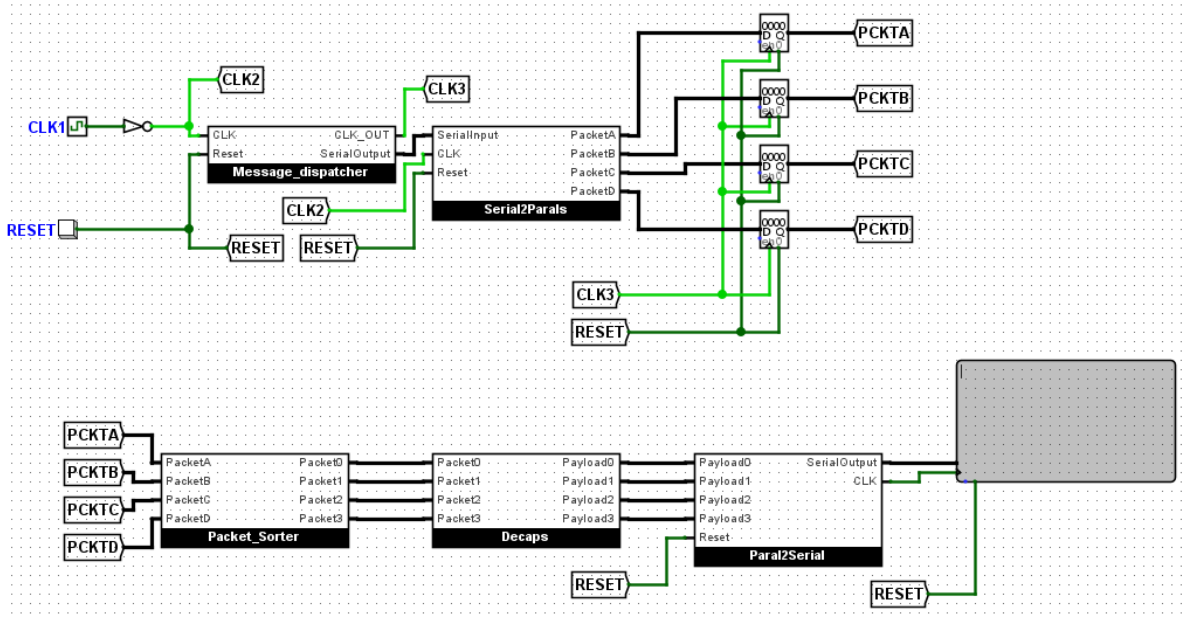
El siguiente, es el circuito que se encarga de transmitir los mensajes, se almacenan en código Hexadecimal en una memoria ROM, luego con otra de las mismas, se encarga de enviar estos mensajes aleatoriamente como funciona en una situación real, para luego ser recibido, por nuestro circuito “Serial2Parals” y Comenzar el proceso que hemos diseñado.

Este circuito fue proporcionado por el profesor. (Velez, Adenda #1, 2024)



Menú de ejecución (Main)

Resta únicamente unir las conexiones correctas entre cada circuito para que todo funcione correctamente.



Message_Dispatcher cuenta con dos entradas principales: un reloj CLK y una señal de reinicio Reset. En cuanto a sus salidas, además del reloj de salida, proporciona datos en serie. El reloj de entrada debe estar conectado al complemento de otro, ya que lo necesitamos inicializado con un flanco de subida, a este lo denominaremos CLK1, configurado con una duración de 2 ticks tanto en nivel alto como en nivel bajo. Esta señal de reloj también se conectará al CLK Serial2Parals.

La señal de Reset estará asociada a un botón denominado RESET, que se conectará a las entradas de reinicio (Reset o Clear) de todos los circuitos diseñados. El reloj de salida se llamará CLK3, y se generará a partir de la señal Slow_CLK del Message_Dispatcher. Este se conectará a los registros encargados de almacenar correctamente los paquetes de datos procesados por el circuito Serial2Parals.

Para la conexión de los circuitos, los paquetes de datos se conectarán entre las conexiones correspondientes. Finalmente, la salida del reloj de Paral2Serial, que proviene del circuito FF_Counter y tiene una duración de 1 tick en nivel alto y 1 tick en nivel bajo, se conectará a la terminal en su entrada de Clock. Como último paso, la salida SerialOutput de Paral2Serial se conectará a la entrada Data del módulo TTY.

Luego de esto, el circuito puede ser simulado correctamente. Descargue el circuito y compruebe su funcionamiento.

Conclusiones

En este laboratorio se logró implementar con éxito un sistema digital que emula un Transmission Control Protocol (TCP) del lado del cliente utilizando el simulador Logisim Evolution. A través del diseño de sistemas digitales, métodos de simplificación, la utilización de diferentes circuitos y los convertidores de serie-paralelo y paralelo-serie, se consiguió simular la transmisión, reordenamiento y desencapsulación de mensajes en paquetes.

El proceso incluyó la optimización del diseño mediante la simplificación de funciones lógicas utilizando mapas de Karnaugh, lo que permitió reducir la complejidad del diseño de los circuitos. Además, el uso de contadores y registros garantizó la correcta sincronización

y almacenamiento de los datos y por último, el buen manejo de los flip_flops permitió una sincronización con los distintos circuitos implementados en el sistema.

En conclusión, este ejercicio permitió aplicar de manera práctica los conocimientos adquiridos en clase y laboratorios de la asignatura Arquitectura de Computadores, facilitando una mayor comprensión del TCP, caso similar con los diferentes elementos y herramientas estudiados durante estas semanas, con los que pudimos experimentar gracias al simulador usado para este laboratorio.

Exposición y simulación

Puede ver la exposición junto con la simulación del circuito aquí, allí también estará la explicación de cada módulo.

Bibliografía y referencias

Bibliografía

Velez, F. A. (2024). *Adenda #1.* Medellin.

Velez, F. A. (2024). *Adenda #2.* Medellin.

Referencias

Velez, F. A. (2024). *Adenda #1.* Medellin.

Velez, F. A. (2024). *Adenda #2.* Medellin.