Diseño de bloques básicos para simulación de sistemas asincrónicos en Matlab

Guadalupe Ayala Lomeli, Gonzalo Isaac Duchén Sánchez.

Sección de Estudios de Posgrado e Investigación (SEPI)
Escuela Superior de Ingeniería y Mecánica y Eléctrica (ESIME)
Instituto Politécnico Nacional (IPN)
Unidad Profesional Culhuacán
Av. Santa. Ana No.1000., Col. San Francisco Culhuacán, C.P. 04430,
México, DF.

lomeli79@calmecac.esimecu.ipn.mx

Abstract. The use of a central clock has begun to have some fundamental limitations like: overheat of the circuit, inefficiency in the use of the energy, emission of electromagnetic noise and synchronization failures[1,2]. One solution to this problem is the asynchronous design, reducing the limitations of the synchronous circuits. So, it is necessary to begin to handle these circuits, but the problem now is that there are no tools for the asynchronous design as in the case of the synchronous design, for this reason, in this paper is described the development of a library that contains a set of asynchronous blocks designed in Matlab to make simulations of totally asynchronous circuits.

Resumen. El uso de un reloj central ha comenzado a tener algunas limitaciones fundamentales como: sobrecalentamiento del circuito, ineficiencia en el consumo de la energía, emisión de ruido electromagnético y fallas de sincronía [1,2]. Una solución a este problema es el diseño asincrónico, que reduce en gran medida las limitaciones de los circuitos sincrónicos. Como se ha observado es necesario comenzar a manejar estos circuitos, pero existe el problema de que no hay herramientas para el diseño asincrónico como es el caso del diseño sincrónico, por tal motivo, en este artículo se describe el desarrollo de una librería que contiene un conjunto de bloques asincrónicos diseñados en Matlab para poder realizar simulaciones de circuitos totalmente asincrónicos.

Palabras clave: handshaking, circuito asincrónico, comunicación, latch, elemento muller c, simulink.

Introducción

En el diseño de circuitos asincrónicos no es necesario esperar un cambio en la señal de reloj para ejecutar alguna acción, en vez de esto, se emplea un protocolo (handshaking) entre sus componentes para llevar a cabo la sincronización necesaria, la comunicación y secuencia de operaciones, evitando el sobrecalentamiento del circuito y consumo de energía, ya que con este protocolo, los componentes solo trabajan cuando se les requiere y además elimina la emisión de ruido electromagnético debido a que no existe una señal de reloj para controlar acciones, como es el caso del diseño sincrónico.

La mayoría de los circuitos asincrónicos están basados en un protocolo que está conformado por una señal de petición y una señal de reconocimiento (Fig.1), en donde la señal de petición la genera el emisor y es la principal iniciadora de cualquier transferencia, mientras que la señal de reconocimiento la emite el receptor e indica la terminación de la transferencia [3]. Estas señales proporcionan el control de secuencia necesario para llevar a cabo el correcto funcionamiento del sistema asincrónico.

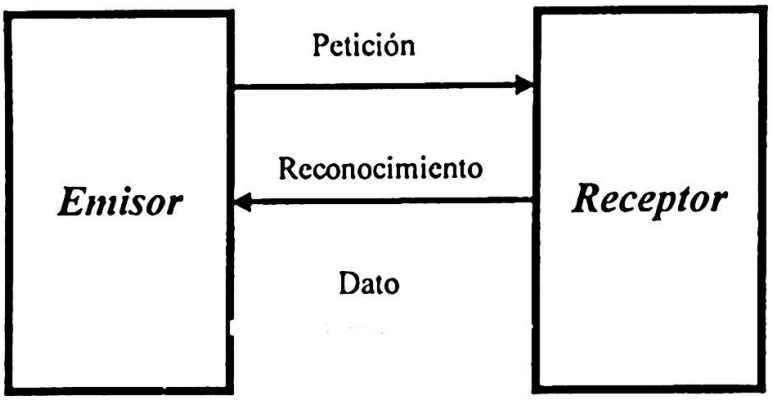


Fig. 1. Interfaz asincrónica

La codificación más popular para las señales de handshaking (petición y reconocimiento) son: el protocolo de 2 fases y 4 fases bundled data, también conocido como NRZ (sin retorno a cero) y esta se caracteriza por una señal alta (generalmente +5 ó +3,3 voltios) para un 1 binario y una señal baja (0 voltios) para un 0. El protocolo de 4 fases bundled data también conocido como RTZ (con retorno a cero) [4]. En el protocolo de 2 fases los eventos están indicados por solo 2 transiciones, ya sea de 1→0 ó de 0→1, en la figura 2 se observa el esquema de este protocolo.

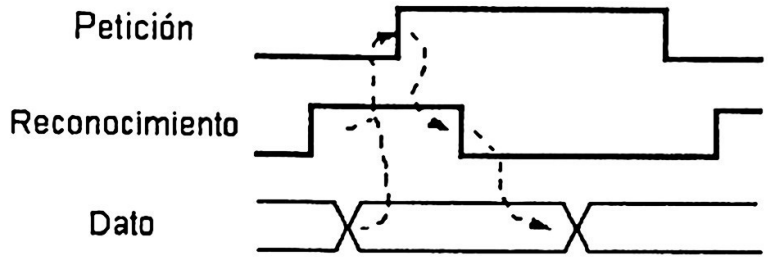


Fig. 2. Protocolo de 2 fases Bundled Data

En cambio, en el protocolo de 4 fases, los eventos están indicados por 4 transiciones, en donde la primera transición se lleva a cabo cuando el emisor necesita enviar un dato y la señal de petición se coloca en 1, la segunda transición es cuando el receptor recibe el dato y la señal de reconocimiento se coloca en 1, la tercera transición es cuando el emisor responde colocando la señal de petición en 0 y por ultimo la cuarta transición es cuando el receptor reconoce la acción realizada colocando la señal de reconocimiento en 0. En este momento, el emisor puede iniciar el siguiente ciclo de comunicación. En la figura 3 se observa el esquema de este protocolo de comunicación

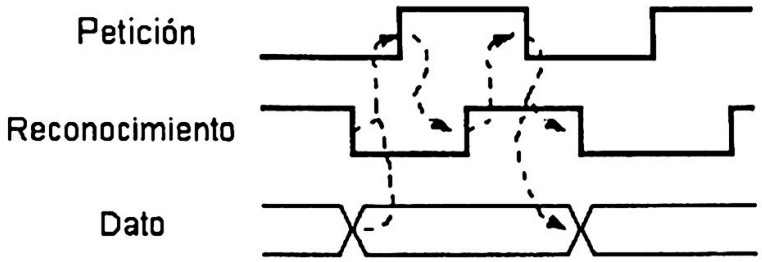


Fig. 3. Protocolo de 4 fases Bundled Data

Como se explicó arriba, en los circuitos asincrónicos la sincronización no es por medio de una señal de reloj, en vez de está, se emplean las señales de handshaking entre registros vecinos; ahora vamos a considerar que los registros son un conjunto de latches.

Las latches son los únicos componentes que inician y toman parte activa en el handshaking, mientras que los demás componentes son "transparentes" al handshaking. Para representar las latches, emplearemos el símbolo de una caja con doble línea vertical.

Para comenzar a familiarizarnos con el funcionamiento de las latches, en la figura 4 se muestra un pipeline compuesto por cinco latches. Los arreglos de cajas representan enlaces que consisten de petición, reconocimiento y señales de dato. El valor valido en L1 tiene que copiarse en L2 y el valor vacío en L3 tiene que copiarse en L4. Esto significa que L1 y L3 contienen ahora los duplicados viejos de los valores que han

sido almacenados en L2 y L4. Los duplicados viejos son llamados "bubbles" y los nuevos valores válidos y vacíos se llaman "tokens" [5].

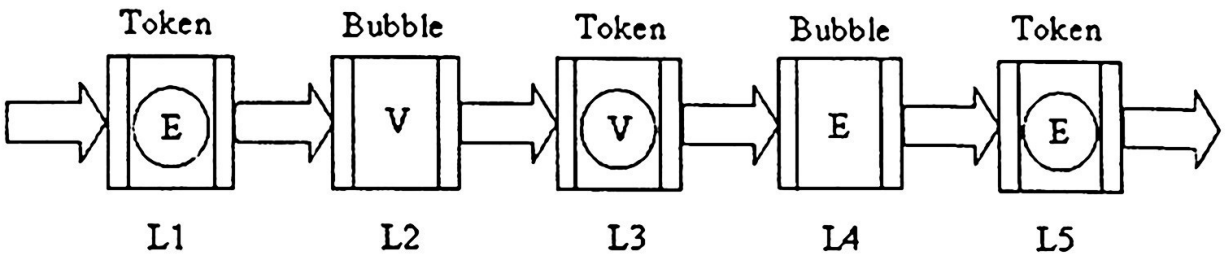


Fig. 4. Estado pipeline de 5 etapas

Ahora, para poder realizar la implementación de circuitos asincrónicos, es necesario contar con un conjunto mínimo de componentes, los cuales se pueden agrupar en cuatro categorías: Latch, Bloque de función, Control de flujo condicional y Control de flujo incondicional. (Fig.5).

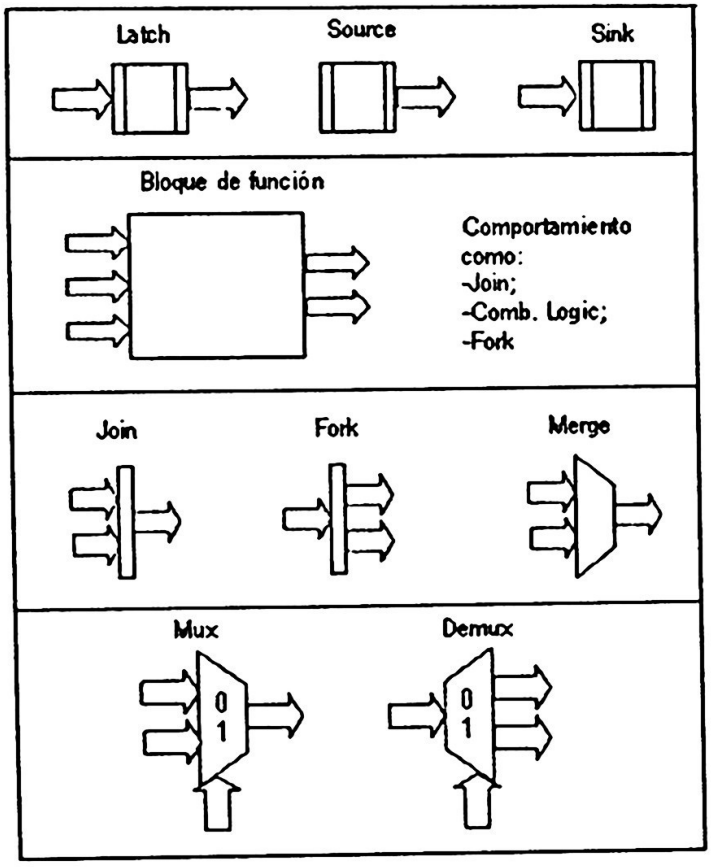


Fig. 5. Conjunto de componentes asincrónicos

El componente latch nos proporciona el almacenamiento de variables e implementa el handshaking. Una latch con solo un canal de salida es una fuente (*source)* que produce tokens (con el mismo valor constante) y una latch con solo un canal de entrada es un consumidor (*sink)* que consume tokens.

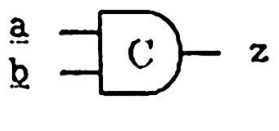
El bloque de función realiza lo siguiente: (1) Espera tokens en sus entradas (un join implícito), (2) realiza la función combinacional necesaria y (3) pone en circulación los tokens en sus salidas.

Los componentes de control de flujo incondicional son: fork, join y merge, donde un fork se emplea cuando la salida de un componente es la entrada para más componentes; un join se emplea cuando el dato de varios canales independientes necesita ser sincronizado, normalmente por que son entradas (independientes) para un circuito. El salir en todas direcciones de un canal implica un fork, y el entrar en todas direcciones de varios canales implica un join. El componente merge tiene dos o más canales de entrada y un canal de salida., el handshake en la entrada del canal es mutuamente exclusivo.

Los componentes de control de flujo condicional son el mux y demux, los cuales realizan la función usual de seleccionar entre varias entradas para una salida y varias salidas para una entrada. Un mux sincronizará el control del canal. Similarmente el demux sincronizará el control y los canales de entrada de datos y dirige la entrada al canal seleccionado de salida.

Implementación de bloques

La implementación de estos componentes se realizó con la ayuda de S-Function Builder, que es una utilería de simulink que permite construir bloques por medio de código C [6]. Para comenzar la implementación de los circuitos mencionados arriba, primero fue necesario analizar y programar el elemento Muller C, ya que es un componente fundamental para la mayoría de los circuitos asincrónicos y permite sincronizar los procesos. Este elemento consta de 2 entradas y 1 salida, en donde la salida será 1 si las entradas son 1, la salida será 0 si la entrada es 0 y en los demás casos la salida no cambia. En la figura 6 se muestra la tabla de verdad y la simbología del elemento muller C.



a	b	z
0	0	0
0	1	no cambia
1	0	no cambia
1	1	1

Fig. 6. Elemento Muller C

El siguiente paso fue realizar la implementación del conjunto latch (source, sink y latch) conforme a la figura 7. El funcionamiento del conjunto latch es el siguiente: para que el dato se copie de una latch a otra, es necesario que la petición sea igual a un 1 lógico y el reconocimiento sea un 0 lógico, implicando esto que la siguiente latch está lista para copiar el dato, y así sucesivamente. Como se observa, por cuestiones de

diseño del elemento Muller C, es necesario invertir la señal de reconocimiento, ya que el elemento Muller C permite pasar el dato solo si la petición y el reconocimiento es igual a 1.

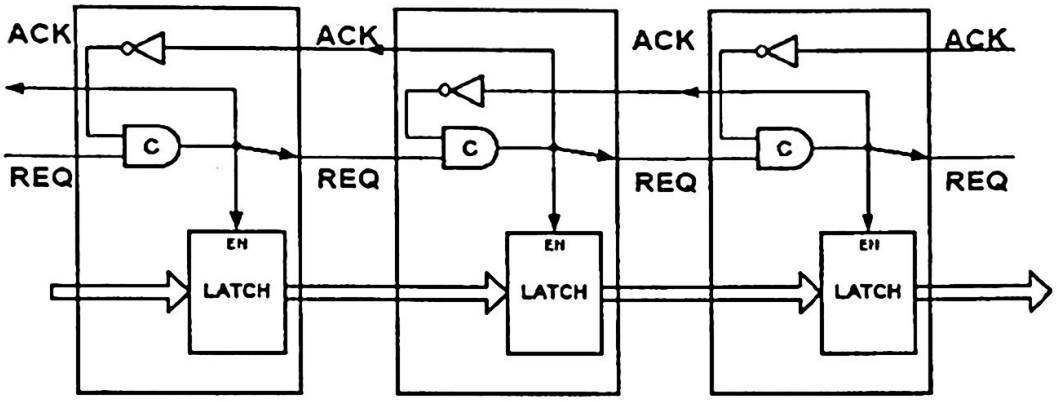


Fig. 7. Implementación de latch

La implementación del componente fork se llevó a cabo conforme a la figura 8. El fork simplemente duplica el dato de entrada y está compuesto por un elemento Muller C que controla las señales de reconocimiento.

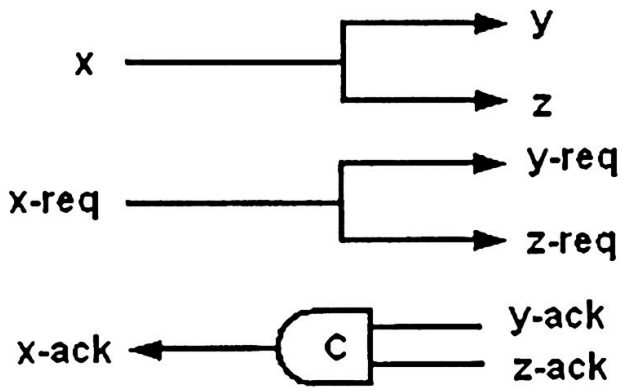


Fig. 8. Implementación de fork.

La implementación del componente merge, se realizó conforme a la figura 9. Este componente es un poco más complicado, ya que a diferencia de los demás, en los canales de entrada las señales de handshake son mutuamente exclusivas. En general la función del merge es sincronizar la transmisión de la entrada activa a los canales de salida.

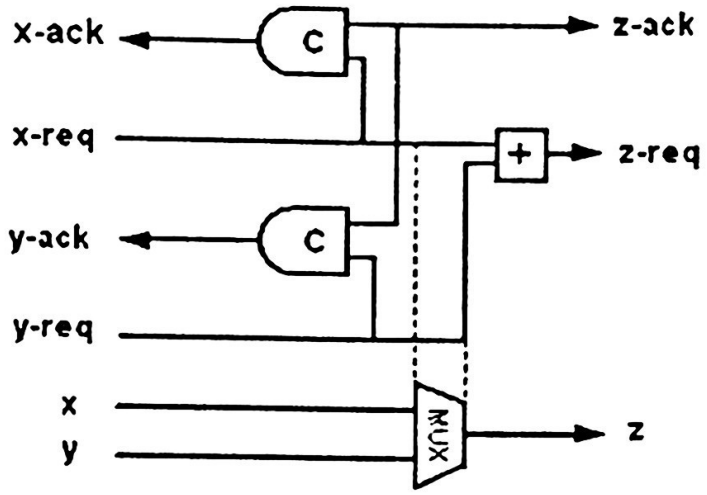


Fig. 9 . Implementación de merge

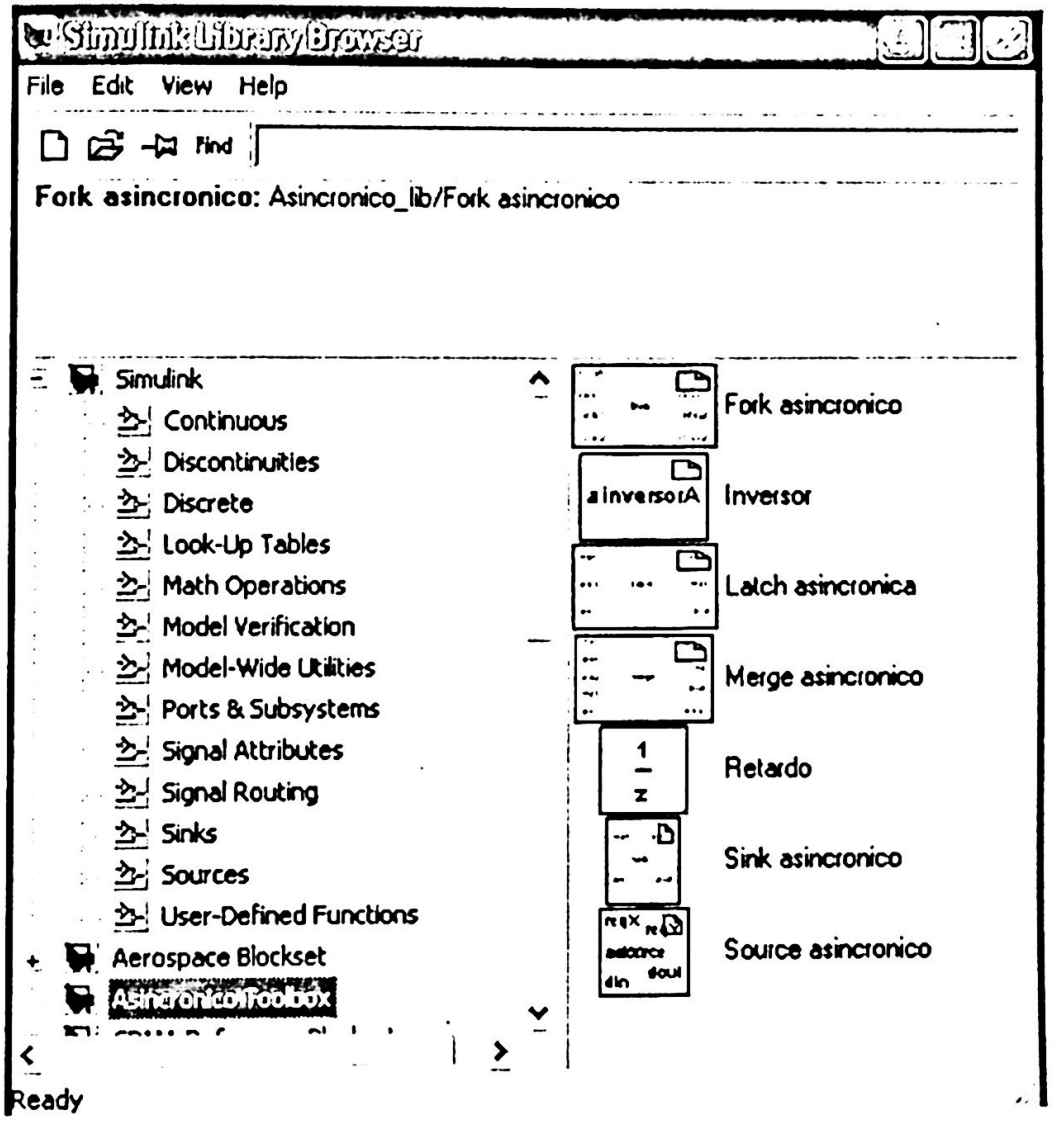


Fig. 10. Librería Asincrónica en Simulink

Ejemplos y resultados

Una vez que los bloques fueron implementados, el siguiente paso fue crear una librería dentro de Simulink, que se realizó con la ayuda de la función slblocks de matlab. La librería lleva el nombre de Asincrónico Toolbox y contiene los siguientes elementos: fork, latch, merge, retardo, inversor, sink y source. En la figura 10 se observa la librería y sus componentes.

En la figura 11 se muestra el ejemplo del funcionamiento de latch, sink y source, en el cual se realiza la transferencia del dato 999 en modo asincrónico. Primero es necesario realizar la petición por medio de un valor constante para comenzar la comunicación entre las latches. Como se observa, es necesario colocar un retardo por cada uno de los componentes, ya que este tipo de circuitos requieren retroalimentar la señal de reconocimiento y por cuestiones de simulación es necesario colocar el retardo para poderse llevar a cabo correctamente.

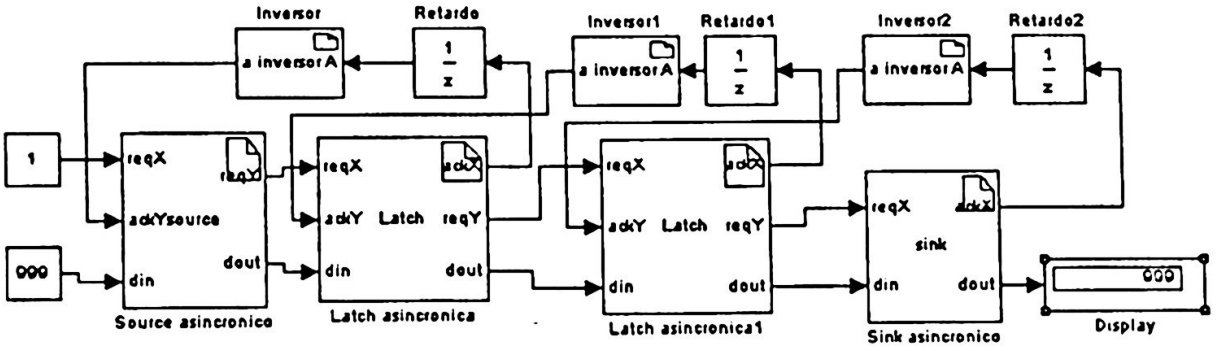


Fig. 11. Transferencia de datos asincrónico.

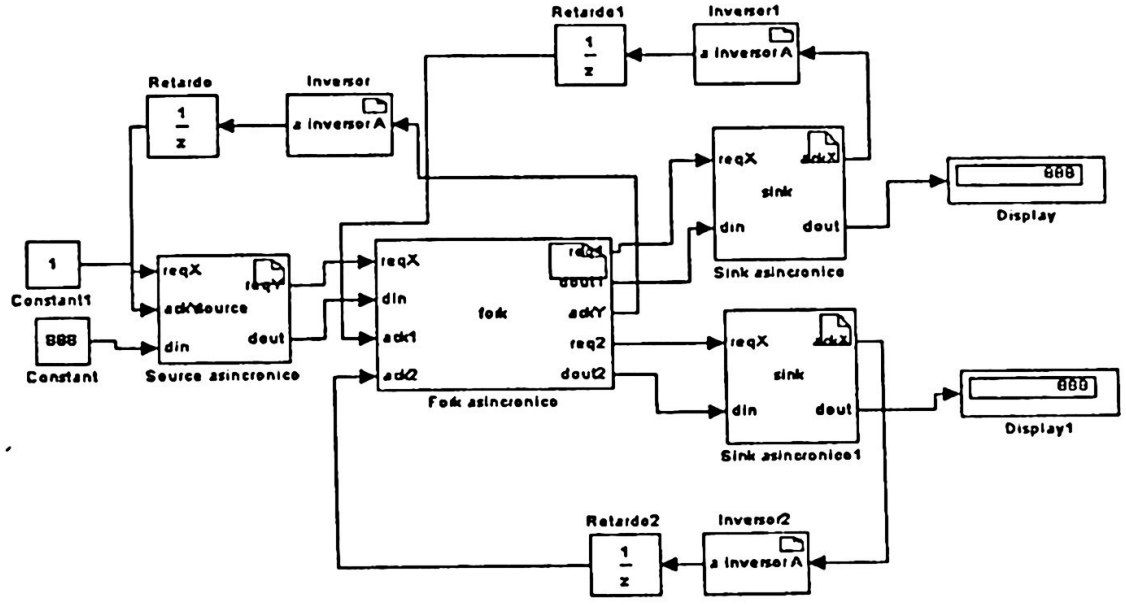


Fig. 12. Ejemplo del funcionamiento de fork

En la figura 12 se muestra el ejemplo del funcionamiento de sink, fork y source, en donde la función del fork es duplicar el dato 888 que le proporciona el source y ambos sinks consumen los datos duplicados.

Conclusiones

Los circuitos sincrónicos hoy en día tienen muchas limitaciones y es necesario comenzar a cambiar de visión y retomar el diseño asincrónico para contrarrestar esas limitaciones.

Un buen comienzo hacia el diseño asincrónico es la implementación de este conjunto de componentes que proporciona una herramienta muy útil para el usuario interesado en simular y diseñar circuitos asincrónicos.

Referencias

1.Bradley J. Benschneider, et. al., A 300-MHz, 64-b Quad-Issue CMOS RISC Microprocessor, IEEE Journal of Solid-State Circuits, vol. 30, no. 11, November 1995, pp. 1203-1214.

2.D.W. Dopperpuhl et. al., A 200-MHz 64-b dual-issue CMOS microprocessor, IEEE Journal of Solid-State Circuits, vol. 27, no. 11, November 1992, pp. 1555-1565.

3.A. Davis and S.M Nowick, "An introduction to asynchronous circuit design", Technical Report UUCS-97-013, Deparment of Computer Science, University of Utah, septiember 1997.

4.D.W. Lloyd and J.D. Garside, "A practical comparison of Asynchronous Design Styles", IEEE Computer Society Press, 2001.

5. J. Sparsø and S. Furber, "Principles of a synchronous circuit design" -A systems perspective, Kluwer Academic Publishers, 2001.

6 .Building S-Functions Automatically (Construcción automática de funciones S), Ayuda en línea de Matlab.