Facultad de Ingeniería y Ciencias Hídricas

I. Consigna

DISEÑAR un sistema integrador (SI) capaz de realizar una operación de multiplicación de números de 2 dígitos, el cual tiene como interfaz con el usuario un emulador de terminal en una pc. Dicho sistema estará compuesto de un hardware y software. El módulo hardware (MHW) incluye el procesador RV32I y la UART desarrollados en clase y demás componentes diseñados por el grupo de alumnos, a fin de establecer la comunicación entre la pc y el SI. El módulo software (MSW) incluye el código de manejo de e/s y la rutina de multiplicación.

II. Introducción

El sistema integrador diseñado se basa en la arquitectura de conjunto de instrucciones RISC-V, una ISA libre y de código abierto. Siguiendo la filosofía RISC (conjunto de instrucciones reducido), la arquitectura se centra en la implementación de las instrucciones más utilizadas para optimizar el rendimiento global.

Se construyó en Verilog un procesador RISC-V monociclo con la capacidad de ejecutar diversas instrucciones, incluyendo sumas, restas, salto incondicionales, saltos condicionales e instrucciones de carga y almacenamiento de datos en memoria. El presente informe se centra en detallar las implementaciones adicionales necesarias para este procesador RISC-V, conforme a la consigna que establece el diseño de un sistema integrador (SI) capaz de realizar operaciones de multiplicación de números de dos dígitos.

III. Descripción del SI propuesto

El SI propuesto, tal y como indica la consigna, consta de dos módulos. El módulo de hardware, que incluye el procesador RV32I construido a partir de las guías prácticas 6 y 7 y de los diagramas y material teórico propuesto por la cátedra; y de la UART también desarrollada en clases a partir del material de práctica. Y el módulo de software, realizado en lenguaje ensamblador en RARS que incluye la rutina de carga y decodificación de las entradas, y la muestra en pantalla del resultado.

A. Módulo de Hardware

El procesador diseñado consta de 9 módulos detallados a continuación:

PC: Este módulo, como su nombre lo indica, es el encargado de manejar el contador de programa. A él ingresan la señal del reloj (clk) y el siguiente pc, para así devolver como salida el pc actual, que se actualiza con cada pulso de reloj.

Adder16: Es un sumador de 16 bits, utilizados para incrementar el contador de programa.

Br: Es el encargado de controlar los registros, en él ingresan la señal de reloj, los operandos a1, a2, los cuales son las direcciones a leer, el operando a3, la señal wd3 y la bandera we, que indica si hay que escribir en un registro, donde si dicha bandera está activa, se escribirá en el registro con la dirección a3 el dato ingresado en la señal wd3. Este módulo devuelve las señales rd1 y rd2, que son los datos guardados en las direcciones de entrada a1 y a2.

Mu16: Es un multiplexor de 16 bits, que devuelve la entrada e1 o e2 en base a la señal de control sel. El mismo se utiliza para escoger la dirección del contador de programa al módulo PC (si se utiliza el PCjump o el PCnext). Tiene un rol importante en la implementación de las instrucciones de salto.

Mu: Es un multiplexor de 32 bits, también con dos opciones y la señal de control sel. El mismo se utiliza para elegir el segundo operando de la ALU, el cual puede ser un inmediato o el dato de registro rd2.

Mu4: Es un multiplexor que posee 4 entradas. Se utiliza para seleccionar el dato a escribir en el registro, que puede ser el resultado de la ALU, un dato de memoria o el PCnext. Si bien el multiplexor tiene cuatro entradas, al

tener solo 3 opciones, la cuarta entrada no se conecta a nada. Se tiene el cuidado de que la señal de control correspondiente a este multiplexor solo va de 00 a 10 y nunca cambia a 11.

SE: Es el generador de los números inmediatos. Dado que en las instrucciones los datos inmediatos no están siempre en el mismo lugar (en algunos casos el dato está todo en los últimos bits y en otros está segmentado en distintos sectores de la misma y se deben unir para conseguir el dato completo). El módulo recibe la instrucción y la señal de control src que indica que tipo de instrucción es, y devuelve el número inmediato correspondiente.

ALU: La unidad aritmético lógica, que realiza las operaciones aritméticas suma y resta, y las operaciones lógicas AND y OR. La misma recibe los operandos y una señal de control (ALUControl) que selecciona la operación a realizarse. Devuelve el resultado en la señal res. VER DE LA SEÑAL CERO

Dichos módulos se conectan dentro del datapath, el cual utiliza además las señales utilizadas

UC: Que es la unidad de control, la cual produce las señales de control mencionadas en los módulos anteriores, como ALUControl y las señales sel de los multiplexores. Recibe los códigos f3, f7 y opcode de la instrucción y utiliza dos módulos extra (mainDeco y aluDeco) para generar las señales de control mencionadas.

mainDeco: recibe el código de operación y devuelve los valores de las señales branch, jump, resSrc, memWrite, aluSrc, immSrc, regWrite y aluOp correspondientes según el tipo de operación (load, store, branch, entre otras).

aluDeco: recibe el código de operación, f7, f3 y aluop y, si la última está en alto (es decir, si se debe utilizar la alu), devuelve en ALUControl la operación que debe realizar la ALU.

El datapath y la unidad de control constituyen el procesador RV32I, cuyo circuito puede observarse en la figura 1.

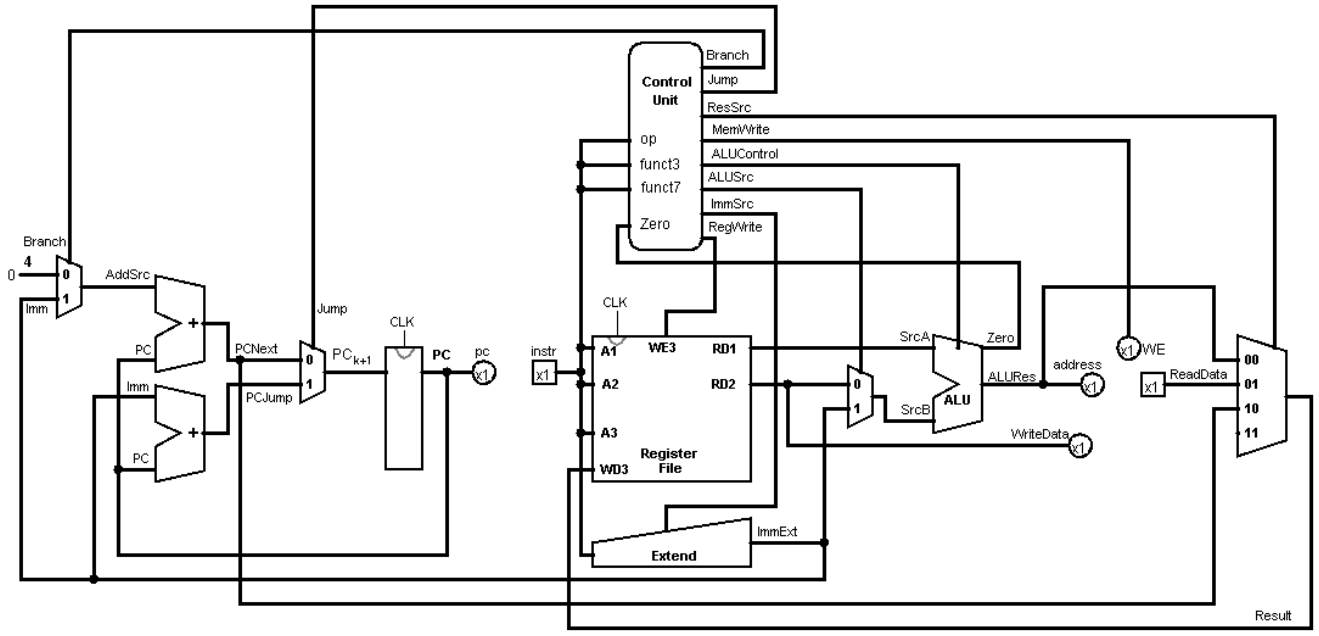

Fig. 1: Diagrama de bloques del procesador RISCV32I

Se tiene también un módulo de **memoria**, que contiene la memoria de instrucciones, la memoria de datos, la UART y la MMU que es un controlador que según la dirección lógica, controla la lectura y escritura de la memoria de datos y la UART. El diagrama del módulo de memoria puede observarse en la figura 2.

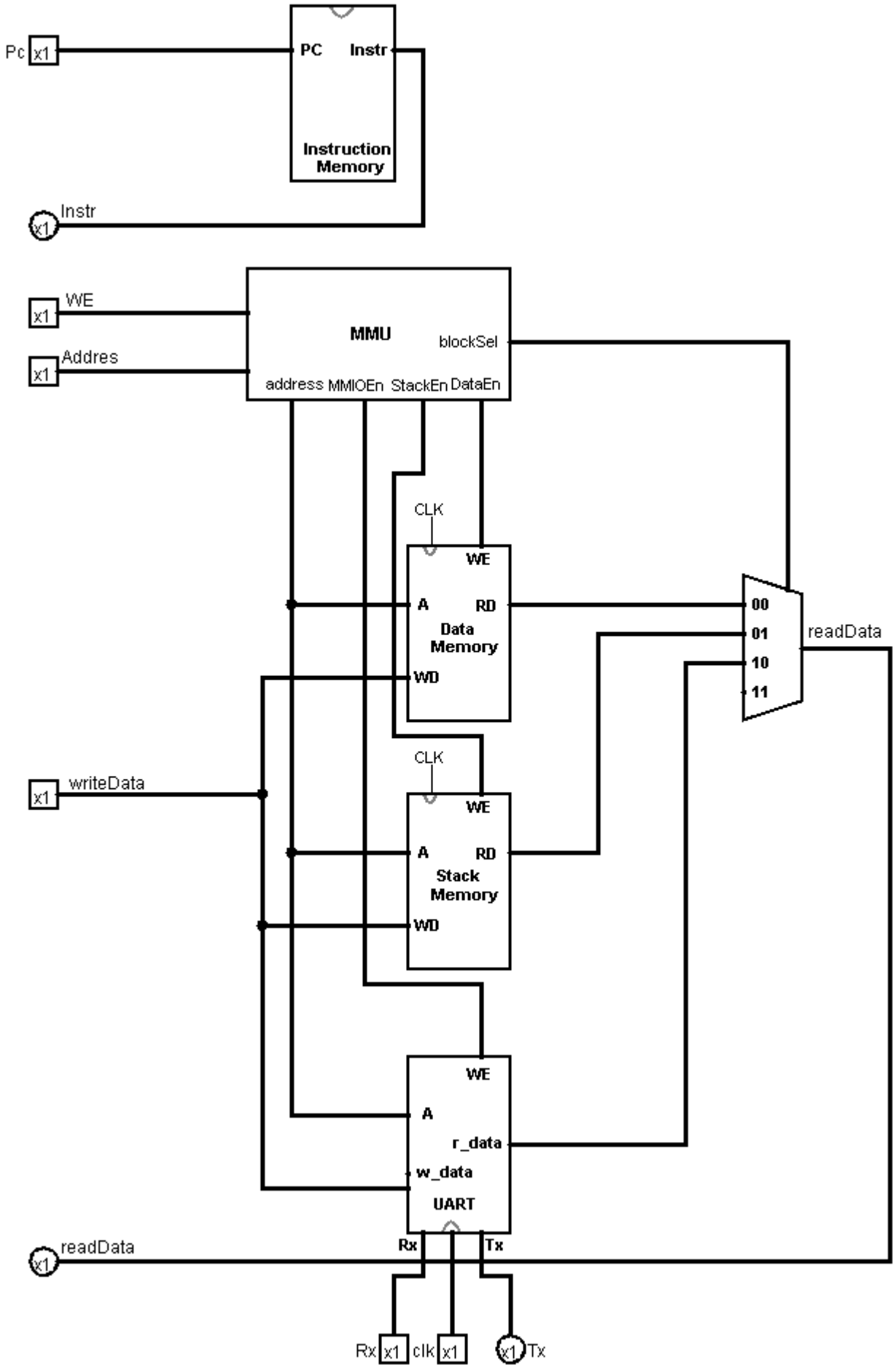

Fig. 1: Diagrama de bloques del módulo de memoria

El módulo UART mencionado anteriormente consta de un generador de baudios, un receiver, un transmitter y dos buffer. El mismo se encarga de la conversión serie-paralelo y paralelo-serie para manejar la entrada y transmisión de datos. Trabaja a 19200 Baudios, pensado para el reloj de 12MHz de la FPGA.

Baud Rate Generator: Es un contador módulo 39. Se modifica la velocidad de transmisión y recepción con este módulo. Este módulo genera un tick cada 39 ciclos de reloj

Receiver: Este módulo recibe los bits en serie y los guarda en registros en paralelo. Consta de 4 estados: El estado *idle*, donde se espera hasta recibir el bit de inicio. El estado *Start* donde, una vez recibido el bit de inicio, se esperan 8 ticks antes de pasar al siguiente estado (cada bit se lee en 16 ticks, por lo que se espera 8 ticks para poder leer los datos a la mitad del bit). El estado *Data*, donde se leen los bits cada 16 ticks. Una vez leídos 8 datos, se pasa al siguiente estado. El estado *Stop* espera 16 ticks y activa la bandera de que ya se leyó el dato, y pasa al estado *Idle*.

Transmitter: Este módulo envía los bits que están guardados en paralelo, en serie. También tiene cuatro estados: *Idle*, donde espera a que se active la bandera para empezar a transmitir. El estado *Start*, que genera un "bit de inicio", mandando la señal tx en 0 y esperando 16 ticks antes de pasar al siguiente estado. Estado *Data*, que manda un bit de dato en tx cada 16 ticks hasta haber mandado los 8 bits. Una vez mandados los 8 bits, se pasa al estado *Stop*, que cambia la señal tx a 1 y espera 16 ticks antes de pasar al estado *Idle* nuevamente.

RBuffer: Es el buffer donde se guardan los datos del Receiver, el mismo recibe el dato a escribir y lo hace una vez que la bandera *wr* está en 1. Se lleva un contador de la cantidad de datos escritos, y cuando éste llega a 5 se escribe en el registro 5 un 1 indicando que la memoria está llena. Para leer se recibe el addres correspondiente y en el momento en que se habilita la señal de lectura se devuelven los datos y se cambia el registro 5 a 0.

TBuffer: Es el buffer donde se guarda la información a transmitir. Cuando se habilita una escritura, se recibe el address y se le resta 6 (ya que los primeros 6 lugares corresponden al RBuffer). Una vez que se hayan escrito los 4 lugares se levanta la bandera full, habilitando al Transmitter para comenzar a transmitir. Cada vez que el Transmitter lee un dato se aumenta un contador y, cuando el mismo llega a 4 se desactiva la señal de full, además de reiniciar los contadores.

Para garantizar la coherencia entre el sistema integrador y la UART, se incorporó un conversor de bits. Dada la diferencia en los tamaños de datos entre el sistema (32 bits) y la UART (8 bits), este conversor desempeña un papel crucial en la interfaz de comunicación.

- Conversión 32 a 8 bits (Receptor): Antes de llegar al receptor, el conversor descarta los bits adicionales del sistema de 32 bits, ajustando la información al formato de 8 bits requerido por la UART.
- Conversión 8 a 32 bits (Transmisor): Después de la transmisión, el conversor expande los datos de 8 bits a 32 bits, cumpliendo con los requisitos del sistema integrador.

El conversor descarta bits adicionales o rellena con ceros según sea necesario para mantener la consistencia entre ambos sistemas.

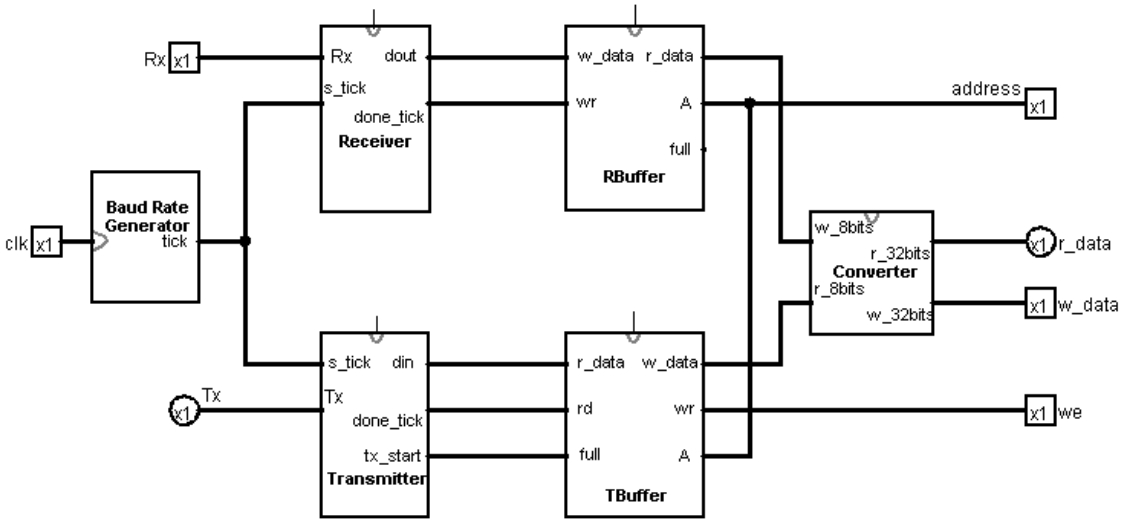

Fig. 3: Diagrama de bloques del módulo UART

El módulo de memoria y el procesador se conectan entre sí mediante el módulo TOP.

B. Módulo de Software

La rutina de entrada/salida y manipulación de los datos para realizar la multiplicación, que luego fue guardada dentro de la memoria de instrucciones se realizó en lenguaje ensamblador a través de RARS. La misma consiste en la carga de datos

dígito a dígito ingresando la dirección donde se encuentra la UART. Dado que los datos entran como números del 0 al 9 y codificados en ASCII, se les debe realizar un tratamiento antes de operar con ellos. Para transformarlos en los operandos que se necesitaban primero se le restó 48 a cada dígito (dado que en la codificación ASCII los números enteros se representan con los números del 48 al 57). Luego, al número que correspondería a la decena se lo multiplica por 10 y se le suma la unidad.
Luego, se procede a realizar la multiplicación. Dado que no se tiene la operación “mul”, el resultado se logra realizando un ciclo while, sumando el operando 1 tantas veces como indique el operando dos.
Por último se prepara el resultado para ser transmitido. Esto es, se divide el resultado para separarlo en 4 dígitos (unidad de mil, centena, decena y unidad). Después se le suma 48 para volver a la codificación ASCII. Por último se guardan los cuatro dígitos en los registros correspondientes para su transmisión.

IV. Pruebas

Se probó el correcto funcionamiento del sistema mediante la implementación de test bench. Con los mismos, se terminó de comprender el funcionamiento de los distintos módulos del Hardware.

Una de las pruebas realizadas fue la de la multiplicación “87*93”, cuyos resultados se muestran en el anexo.

La figura 4 muestra la prueba de la recepción. Allí se puede ver cómo se van guardando los datos en serie correspondientes a los códigos ASCII de los números 8,7,9 y 3 en los registros en paralelo, además de ver cómo cambia la bandera del registro 5.

De igual forma, la figura 5 muestra cómo se transmiten los datos.

V. Conclusión

En este informe, hemos detallado la implementación de un sistema integrador destinado a realizar operaciones de multiplicación de números de dos dígitos. El diseño se compone de un módulo de hardware (MHW) que incluye un procesador RV32I y una UART, junto con un módulo de software (MSW) desarrollado en lenguaje ensamblador.

Si bien se enfrentaron desafíos, especialmente en la implementación del módulo UART y la conexión de este con el procesador, estos obstáculos pudieron superarse proporcionando valiosas lecciones y oportunidades para mejorar nuestras habilidades de resolución de problemas.

La integración del MHW con el MSW permitió aplicar los conceptos teóricos adquiridos durante el curso a un escenario práctico. Las pruebas realizadas confirmaron el correcto rendimiento del sistema, cumpliendo con los requisitos establecidos en la consigna. En la práctica, este proyecto ha permitido comprender de manera más profunda la relación entre hardware y software.

Referencias

[1] Pong P. Chu, *FPGA prototyping by Verilog examples*, New Jersey: John Wiley & Sons, Inc, 2008.
[2] Material de cátedra

Anexo

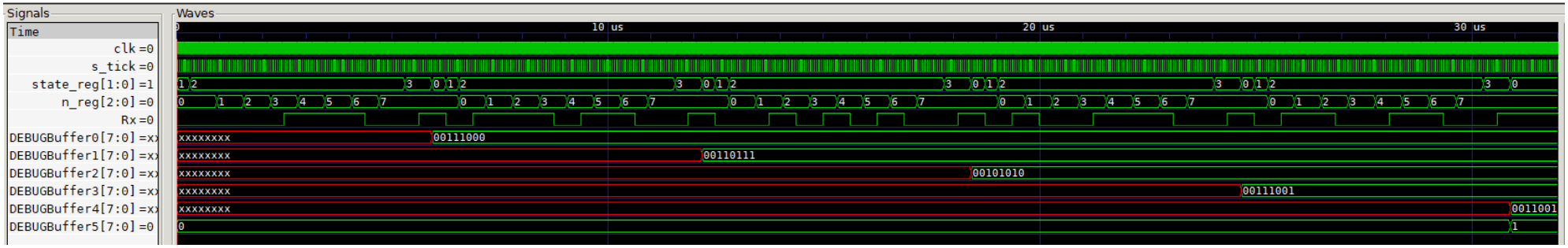

Fig. 4: Test bench datos recibidos

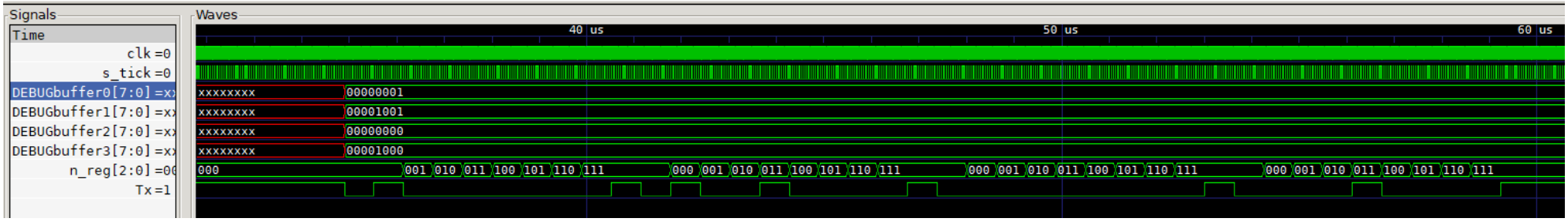

Fig. 5: Test bench datos enviados