ENTREGA 1: MÓDULO RS-232

Miembros del grupo: Emil Diesen Soleim y Valentín Martínez López Fecha: 08/10/2023

ÍNDICE

1) Introducción.

Se trata de un proyecto de implementación de un módulo de comunicación de datos con el protocolo RS232 en la FPGA Nexys4DDR. La comunicación se puede realizar mediante el puerto PMOD y mediante el puerto serial USB-UART de la placa.
En el capítulo 2.1 se describe el sistema y todos los bloques que lo componen.
En el capítulo 2.2 se hablará de las pruebas que se han hecho y de la resolución.
En el capítulo 2.3 se mencionan brevemente los resultados de la síntesis.
En el capítulo 2.4 se habla de las mejoras realizadas en el módulo.

2) Módulo RS-232

Retos técnicos superados del RS-232	Conseguido (Sí/No)
Implementación del RX en VHDL	Sí
Implementación del TX en VHDL	Sí
Simulación Behavioral	Sí
Simulación con retardos	Sí
Sistema RS232 funcionando en placa	Sí

Mejoras del RS-232

Velocidad de transmisión ajustable con switches, con cuatro velocidades elegibles
Número de bits ajustable con switches, entre cinco y ocho bits
Sumador del número de carga en display

2.1) Descripción del sistema RS-232

El sistema está formado por cuatro módulos cómo se puede ver en la figura 1. Estos módulos son el RS232_TX que se usa para transmitir, RS232_RX para recibir, un shift register para almacenar cada bit que salga hasta formar un byte entero, y un FIFO para almacenar los bytes válidos que se ha recibido. Estos cuatro módulos van a ser descritos en los próximos subcapítulos. Para controlar la transmisión de datos y gestión del fifo hay señales de control. Las señales de control están explicadas en la tabla 1. El sistema tiene un reloj Clk y un reset asíncrono activo a nivel bajo, rst.

Tabla 1: Señales de control

Señal	**Sentido**	**Función**
Valid_D	Entrada	Confirmación de que los datos en Data_in son válidos.
ACK_in	Salida	Confirmación de que la transmisión ha empezado, activa a nivel bajo. Se desactiva cuando Valid_D se desactiva.
TX_RDY	Salida	Indicador de que el transmisor está listo para enviar. Se empieza a enviar cuando están activos Valid_D y TX_RDY a la vez.
Data_read	Entrada	Gestiona la lectura del FIFO. También carga los datos de Data_in en un registro

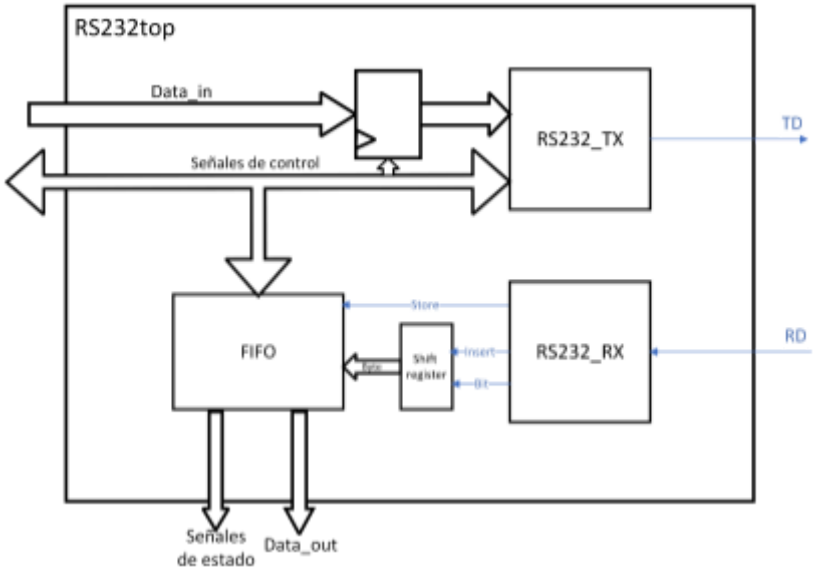

Figura 1: Diagrama de bloques del sistema RS232.
No todas las señales están incluidas, y puede que algunas tengan otro nombre en el código VHDL.

2.1.1) RS232_TX

El módulo RS232_TX es el módulo que se encarga de transmitir datos. Además de Clk y Reset, sus entradas son Start y Data, con salidas de EOT y TX. Start es la señal que empieza el envío de datos y Data son los datos que se va a enviar. EOT se activa para indicar que el módulo esté listo para enviar, y TX es dónde se ponen los bits de envío. Para realizar el envío de datos el módulo se ha realizado como una máquina de estados finitos de tipo mealy, con salidas registradas para evitar glitches.

La máquina de estados finitos tiene cuatro estados, Idle, StartBit, SendData y StopBit. El diagrama se puede ver en la figura 2. Para que se envíe un bit a la frecuencia de transmisión hay un contador llamado pulse_count que cuenta hasta un pulse_width antes de pasar al siguiente bit. También hay otro contador para saber el bit que se está enviando para seleccionar el bit correcto de Data. La salida EOT se activa solo en el estado de idle.

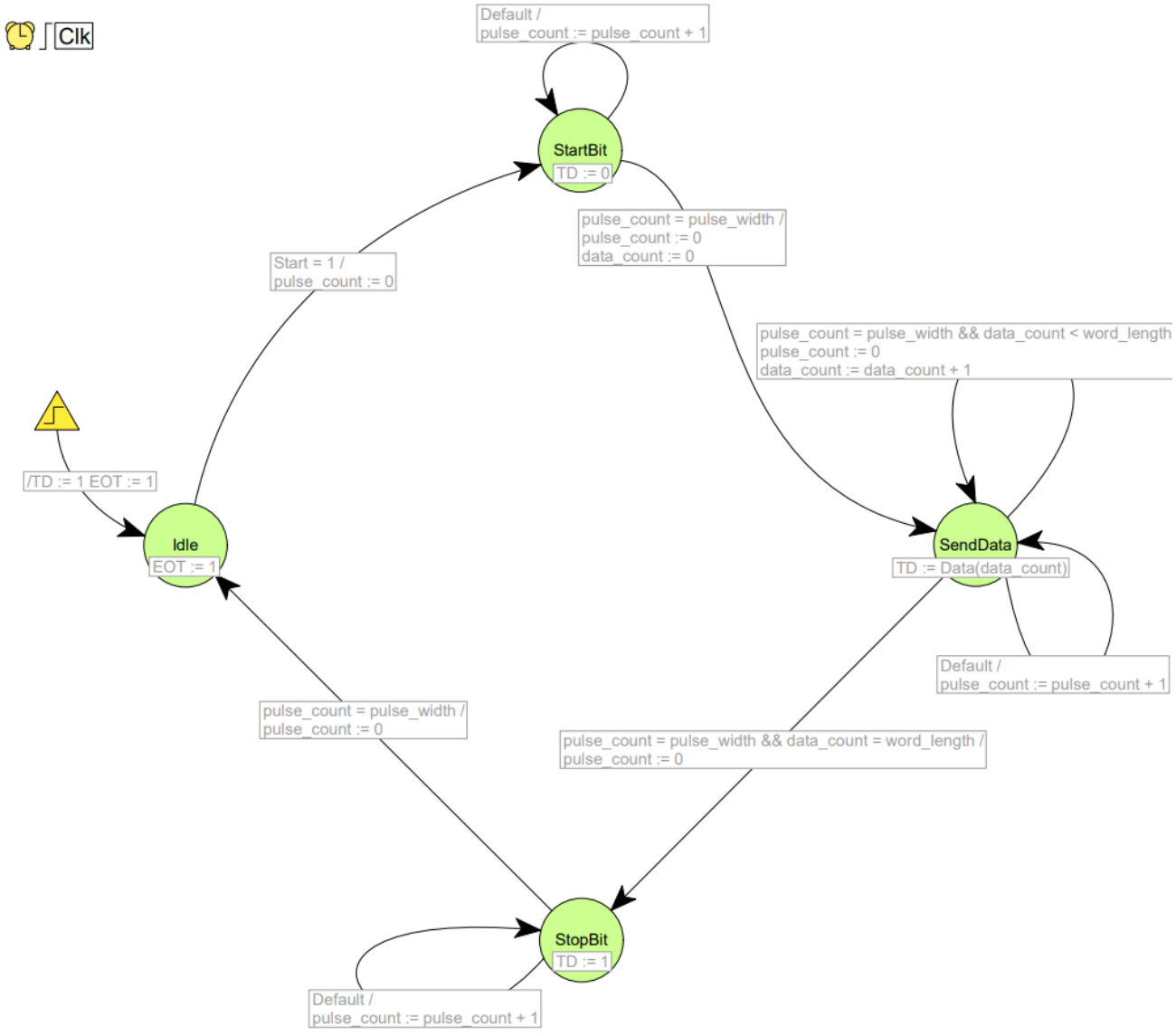

Figura 2: Máquina de estados finitos para transmisión. pulse_count y data_count son registros internos para contar y pulse_width y word_length son constantes.

2.1.2) RS232_RX

El módulo RS232_RX se encarga de recibir datos. Aparte de clk y reset, su entrada es LineRD_in, que en el top está conectada directamente a RD de fuera. Sus salidas son Valid_out para decir que un bit está listo para poner en el shift register, Code_out que es el bit leído, y Store_out que se usa para decir si el byte entero era válido. En este caso Store_out se activa solo si el stopbit leído es cero.

Como el RS232_TX este módulo también tiene cuatro estados, pero esta vez hay RcvData en vez de SendData. La máquina sale del estado de idle cuando hay un cero en LineRD_in, y espera medio ciclo para capturar en el centro de los pulses para minimizar la probabilidad de error. Después de ese medio ciclo se espera un ciclo por cada bit, muestreándolo al final del ciclo hasta el último bit donde pasamos al estado de Stopbit. Para saber si el byte es válido muestreamos el stopbit y lo pasamos directamente a Store_out, ya que Store_out se activa a nivel alto y si stopbit es alto, el byte es válido. Después de esto hay que esperar medio ciclo, y por eso ponemos un flag para decir que ahora sí queremos pasar al estado de idle, y después de medio ciclo se hará.

Tenga en cuenta que en Figura 1, la señal de Code_out es bit, Valid_out es Insert y Store_out es Store.

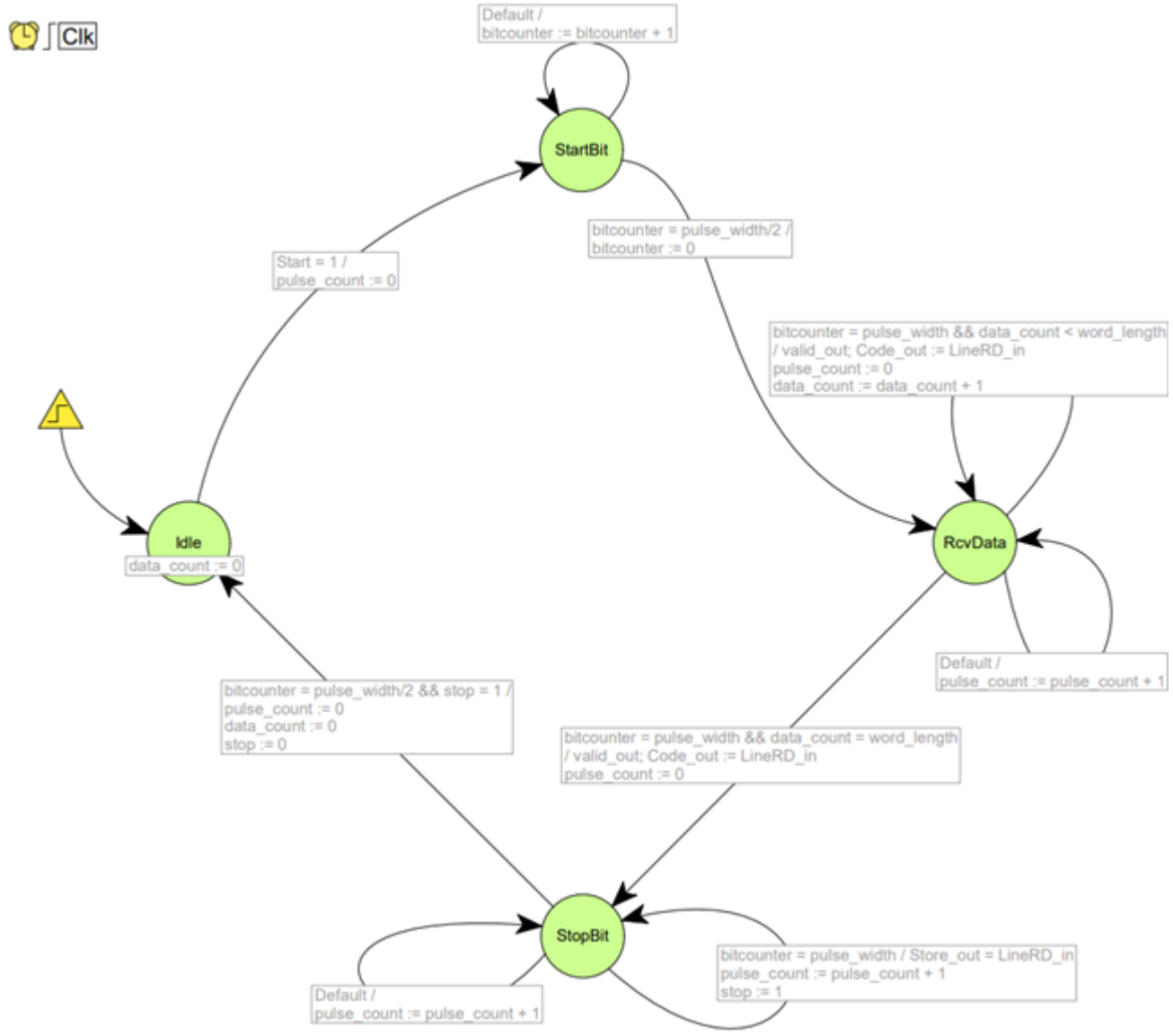

Figura 3: Máquina de estados finitos para recepción de datos. pulse_count y data_count son registros internos para contar y pulse_width y word_length son constantes. Stop es un registro interno para contar otro medio ciclo al final.

2.1.3) Shift Register

El Shift Register es el módulo que se encarga de juntar todos los bits de lectura antes de enviar el byte al FIFO. Se ha realizado con una concatenación dónde se quita el bit menos significativo y se meten los nuevos bits por la parte más significativa.

2.1.4) FIFO

El FIFO (first in first out) se encarga de almacenar los bytes recibidos hasta que el cliente los saque. En esta implementación solo contiene un byte y sirve como buffer, pero eso se puede cambiar muy fácilmente. El FIFO también tiene señales de estado que se envía al cliente que son Full y Empty para informar que el FIFO está lleno o vacío.

El FIFO se ha generado automáticamente usando un bloque IP de VIVADO.

2.2) Simulaciones y prueba en placa del RS-232

Para comprobar que cada entidad funciona correctamente se ha ido realizando un testbench para cada entidad, para el registro de desplazamiento, en especial para el módulo RX y el módulo TX, y para las entidades top para comprobar que la integración es correcta.
Para el testbench del RX se ha usado la entidad TX para enviarle los datos, con el fin de simplificar el proceso.
Para comprobar que la entidad Nexys_RS232 funciona correctamente se ha realizado una simulación con retrasos y su carga en la placa física. Para ello primero se han quitado las advertencias de síntesis, en el proyecto final se tienen tres avisos:

[Synth 8-3331] design nexys_RS232 has unconnected port SW[14].
[Synth 8-3917] design nexys_RS232 has port LED[12] driven by constant 0.
[Synth 8-6014] Unused sequential element sum_t_reg was removed.

Los dos primeros avisos se han obviado por simplicidad de código y para mantener el reset y los LEDs como estaban definidos.
El último aviso se quitó al añadir una señal registrada de sum_t, pero se usan 28 flipflops adicionales, se ha considerado que no es tan importante como para añadir registros.

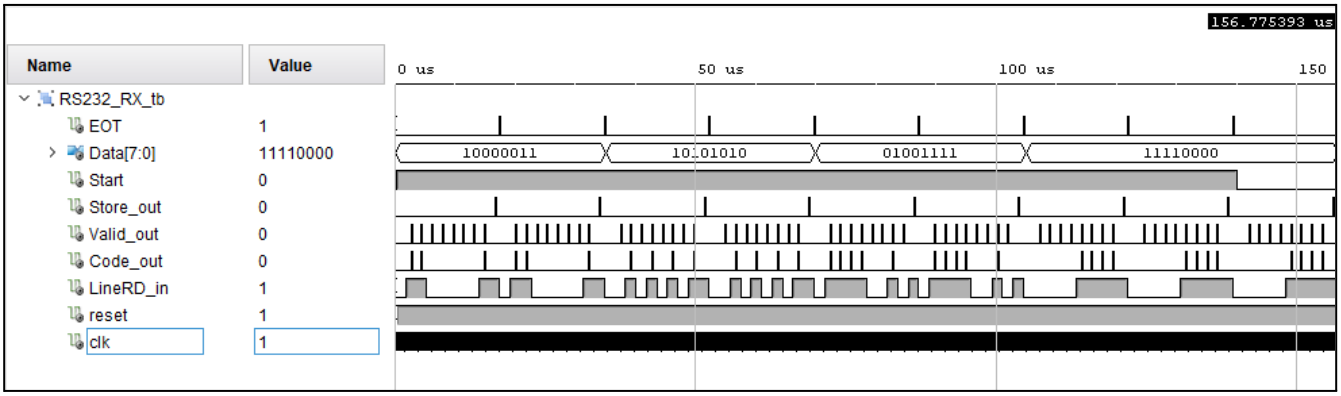

Figura 4: Simulación generada en el testbench usando RX a la salida de TX.

En la simulación del fichero Nexys_RS232 se usa el método Transmit del archivo RS232_test para asignar valores a RD, se ha modificado al introducir las mejoras para hacerlo compatible con la diferencia de tiempos.

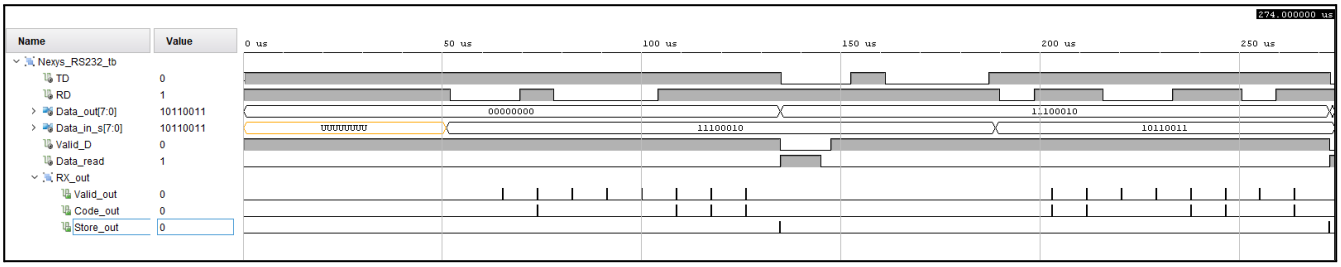

Figura 5: Simulación generada por el testbench Nexys_RS232_tb.

2.3) Resultados de la síntesis

	F_CLK	Slices	LUTs	FFs	DSP (BIOB)	BRAM
CLK_20MHz	100 MHz	-	-	-	-	-
RS232top	20 MHz	189	90	99	-	0.5
Display_ctrl	20 MHz	1190	1095	95	-	-
Leaf Cells	-	463	463	-	-	-
Total	20 MHz	1842	1648	194	49	-

Tabla 1: Reporte de uso de los recursos de la placa en síntesis

La entidad RS232top únicamente usa 189 slices.
El bloque IP del FIFO usa 24 LUTs y 36 flipflops.
El registro de desplazamiento únicamente emplea 8 flipflops, esta entidad usa tan pocos recursos porque su estructura es muy simple, su única operación es una concatenación.
Las entidades de recibo y transmisión de datos usan 53 y 49 unidades.

La entidad Display_ctrl es la que más recursos usa con diferencia, debido a que tiene que realizar sumas de números de 27 bits, aunque es desproporcionado el uso de registros porque en la entidad inferior SplitDig se consumen 270 slices con una mayoría de LUTs porque tiene que realizar operaciones complejas (resto, potencias y división) con dichos números.

2.4) Mejoras del RS-232

2.4.1) Número de bits variable

Se ajusta el número de bits con otros dos interruptores, se han definido cuatro modos de funcionamiento, desde 5 bits a 8 bits.
Se ha realizado la mejora dentro de los bloques TX y RX, de manera que cuando llega al tamaño de palabra seleccionado se pasa al siguiente estado, Stopbit, en ambas entidades.
Se ha realizado creando un tipo enumerado con cuatro posibles valores: fivebits, sixbits, sevenbits, eightbits. La conversión del std_logic_vector al tipo definido nbits_t se realiza en la entidad RS232top.
Se crea una señal del tamaño de palabra seleccionado que se introduce en la máquina de estados de ambas entidades (RX y TX) definiendo el límite de bits a enviar o recibir en el estado SendData o RcvData.
Se podría haber realizado dentro de la entidad top usando solo los bits que interesen en función del modo, pero esto es menos eficiente que la solución adoptada porque requeriría que se pasen en todos los casos 8 bits, cuando en el resto de modos no es necesario.
Ya que el shift register es fijo con 8 bits, es necesario rellenar con ceros al final de la palabra en el receptor si el número de bits es menor de 8. Esto se hace en el estado de StopBit antes de muestrear el stopbit ya que un pulse_width equivale a 174 ciclos de reloj (87 en el caso de velocidad doble) y por máximo habría que poner tres ceros.

En la figura 6 se observa que al introducir el número 11100010, al cambiar el modo cambia el dato de salida. 5=>”00000010”, 6=>”00100010”, 7=>”01100010”, 8=>”11100010”.

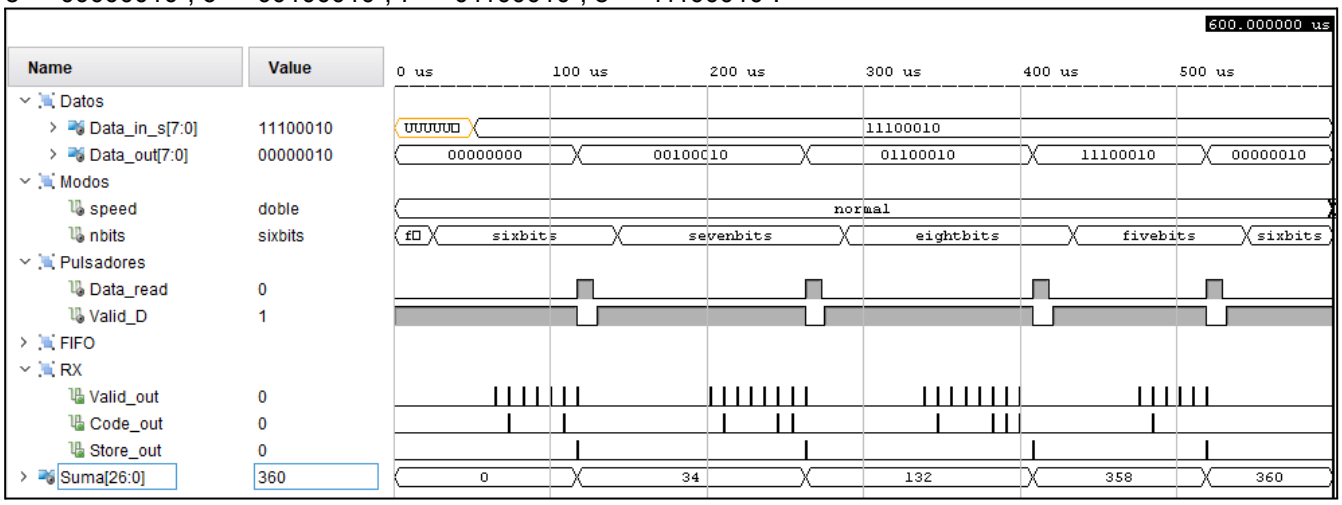

Figura 6: Simulación generada por Nexys_RS232_tb para probar distintos modos de números de bits.

2.4.2) Velocidad de transmisión variable

Se ajusta la velocidad de transmisión con dos interruptores, se han definido cuatro velocidades: 28800, 57600, 115200 y 230400.
Se ha realizado la mejora usando otro tipo enumerado definido speed_t {quarter, half, normal, doble}.
Este parámetro define el ancho de pulso en la máquina de estados. Se usa de manera análoga a la anterior mejora, salvo que ahora se usa en todos los estados para evaluar cada bit.
Para realizar el cambio de pulse_width se ha hecho un multiplexor con el tipo enumerado como señal de selección y constantes para las señales entre las que elegir.
En la figura inferior se observa el funcionamiento de la mejora.

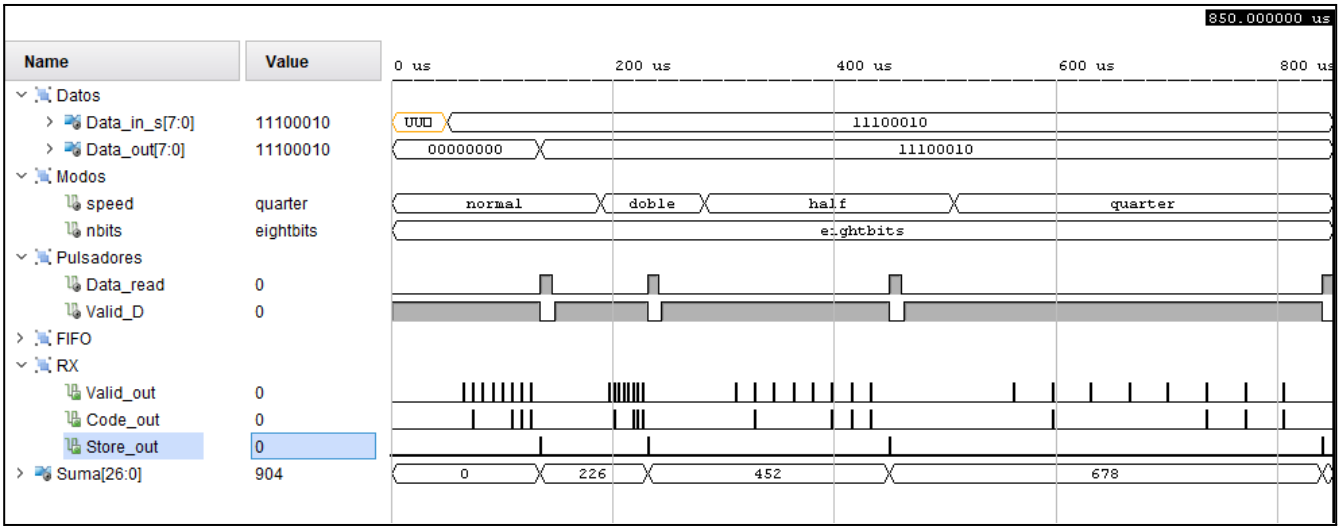

Figura 7: Simulación generada por Nexys_RS232_tb para probar distintos modos de velocidad.

2.4.3) Sumador de número de carga en display

Se trata de una mejora donde se incorporan los ocho dígitos de siete segmentos, se muestra la suma del número cargado a la suma anterior, partiendo de cero.

Figura 8: Señales de la entidad top de esta mejora.

Esta entidad tiene dos bloques inferiores, SplitDig y Decoder. A su vez, se gestiona la frecuencia de iluminación de los segmentos (60 hercios por dígito) con un contador y se ilumina un dígito cada vez mediante la señal DIGCTRL.
Se tiene una señal de suma que acumula la suma anterior y suma el dato cuando entra. Esta suma es introducida a SplitDig cuando se pulsa el botón de carga (sum_enable), solo se suma una vez por pulsación con una condición introducida. Del bloque SplitDig entra la señal suma de 27 bits (99.999.999 es el número máximo) y sale una señal de 32 bits, de cuatro bits por dígito.
Esta señal de salida se divide en 8 para ir cada una a un decoder, que transforma el vector binario en la señal de segmentos que representa ese número.

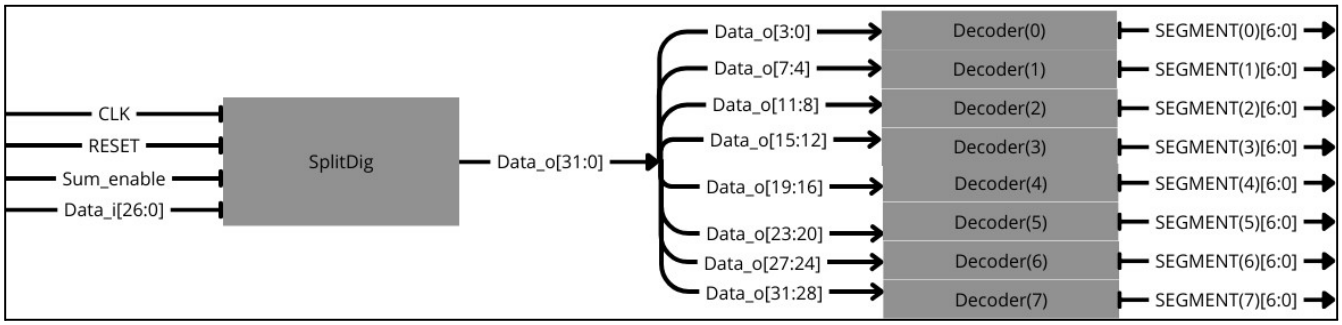

Figura 9: Conexión de bloques para generar la señal de los segmentos.

Para la prueba de esta mejora se ha decidido probarlo en la placa que es donde realmente se puede observar si se ilumina correctamente y si detecta correctamente la señal de sumar.