EAMTA 2021: Diseño Digital
Trabajo Final

Gonzalo Gabriel Fernández

Resumen

El objetivo es integrar bloques básicos y aplicar el análisis de complejidad y timing sobre el diseño. La aplicación radica en una memoria de logueo con filtros FIR y operaciones aritméticas.

I. Descripción

El trabajo consiste en la implementación de dos memorias de logueo de 16 bits de entrada, dos filtros FIR de 8 bits de salida y un bloque que ejecuta operaciones de suma y producto de las muestras de entrada. En la figura 1 se observa un diagrama de bloques del proyecto realizado. Se incluye además, como en el resto de los laboratorios, los módulos VIO e ILA con el objetivo de facilitar el control del diseño.

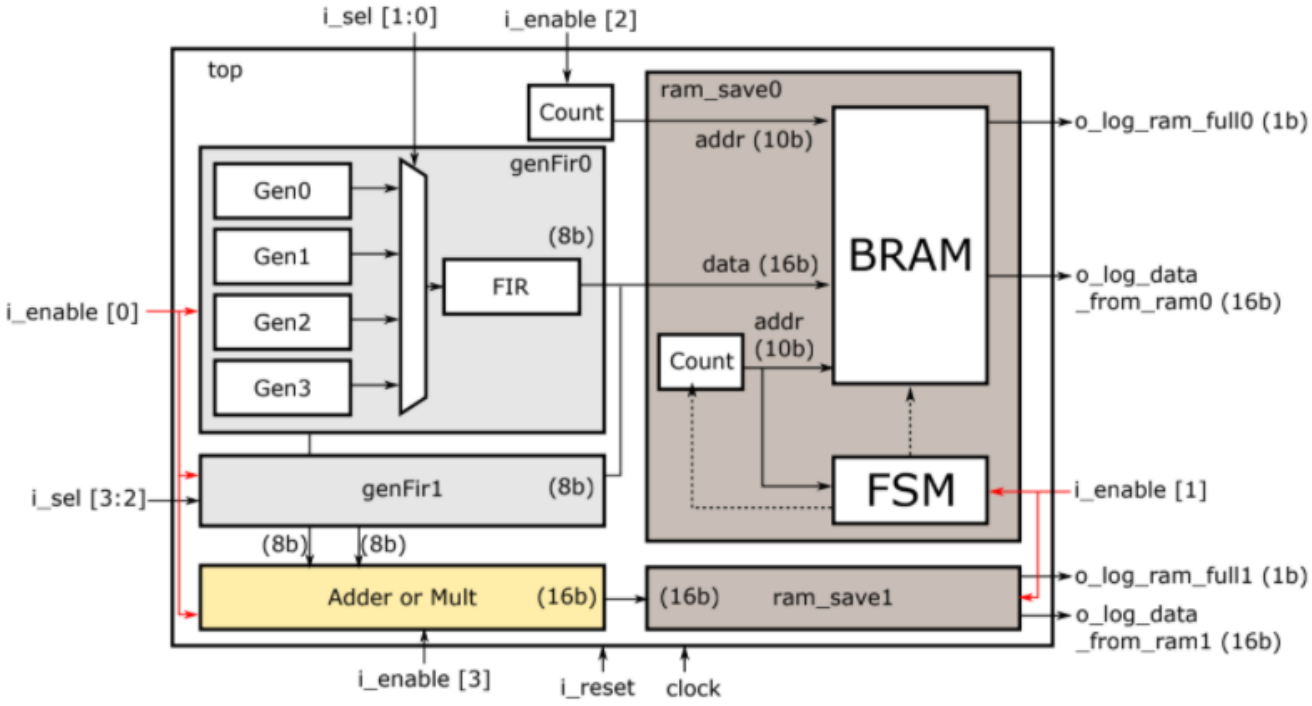


Figura 1. Diagrama de bloques del proyecto.

Los módulos *gen_fir* y *ram_save* se implementaron igual que en el laboratorio 3.

La entrada de cada filtro FIR se obtiene del multiplexado de señales generadas a partir de archivos ".hex"[1]. En el primer bloque BRAM con FSM se loguea la concatenación de señales filtradas: dato1[7:0], dato0[7:0] (16 bits). En el segundo se memoriza el resultado de la operación aritmética.

i_sel[3:0] consiste en entradas de pulsadores, por lo que se considera su vuelta a reposo (modificación de estado con flanco ascendente). Dicha lógica se encuentra en el módulo *button_logic*. *i_sel[1:0]* permite la selección de señal generada por el primer *gen_fir0*, e *i_sel[3:2]* permite la selección de la señal generada por el módulo *gen_fir1*.

i_enable[3:0] consiste en entrada de switch.

- *i_enable[0]* permite la habilitación y deshabilitación de los módulos *gen_fir* y el módulo de operaciones aritemétícas *adder_mult*.
- *i_enable[1]* permite comenzar la escritura en ambos bloques *ram_save* (no se controlan de forma independiente).

[1]Se consideró que los archivos hex se encuentran en formato punto fijo S(8, 6).

- *i_enable[2]* habilita o deshabilita el contador de dirección de lectura en ambas BRAM, con lo que permite la lectura de las memorias.
- *i_enable[3]* permite la selección de operación aritmética a realizar, ya sea adición (0) o producto (1).

Como en el laboratorio 3, se tienen las mismas salidas asociadas a las memorias: *o_log_ram_full*, flag que indica la escritura de la memoria completa, y *o_log_data_from_ram* que es el dato leído de la memoria.

II. Diseño

A continuación se hace una breve descripción de los detalles más importantes de cada módulo diseñado.

II-A. Módulo button_logic

Lógica de cambio de estado por flanco de subida de señal (pulsador).

Ante cada flanco de subida de un pulsador, su estado asociado se conmuta. Ante un alto en la señal de reset, el estado de los pulsadores se coloca en bajo.

II-B. Módulo adder_mult

Operación aritmética (multiplicación o adición) entre dos datos de entrada. Si la señal de enable se encuentra en 0 los datos se suman, si se encuentra en 1 se multiplican.

El módulo se plantea de tal forma de poder recibir datos de entrada con igual o distinto formato de punto fijo (siempre signado). Por lo tanto, la mayor dificultad en su implementación es la alineación genérica de la coma, con la correspondiente expansión de signo y completado con 0, necesario para efectuar correctamente la suma.

Es importante tener en cuenta el formato de punto fijo del resultado, ya que dependerá de los datos de entrada. En general, el resultado del producto tendrá una cantidad de bits fraccionales igual a la suma de los bits fraccionales de entrada, y el resultado de la suma tendrá una cantidad de bits fraccionales igual a la del dato de entrada con mayor cantidad de bits fraccionales.

También se debe tener en cuenta que al pasar como parámetro la cantidad de bits de salida, estos sean por lo menos suficientes para contener el resultado de la multiplicación, es decir, la suma de los bits de entrada. Luego el resultado de la suma, con una cantidad de bits igual (o un bit más) a la del dato de entrada con mayor cantidad de bits, se alineará con los bits menos significativos del puerto de salida.

II-C. Módulos ram_save y gen_fir

El módulo *ram_save* es exactamente igual al utilizado en el laboratorio 3. El módulo *gen_fir* básicamente consiste en la interconexión del módulo de generación de señal con multiplexado y el módulo que implementa el filtro FIR.

III. Simulación

Dado que el proyecto consiste en varios módulos, se decidió dividir la simulación para verificar el correcto funcionamiento del top en diferentes testbench.

III-A. Testbench tb_button_logic

Se implementó un testbench para evaluar el correcto funcionamiento de la lógica de pulsador (*tb_button_logic*). En la figura 2 se muestran los resultados de la simulación.

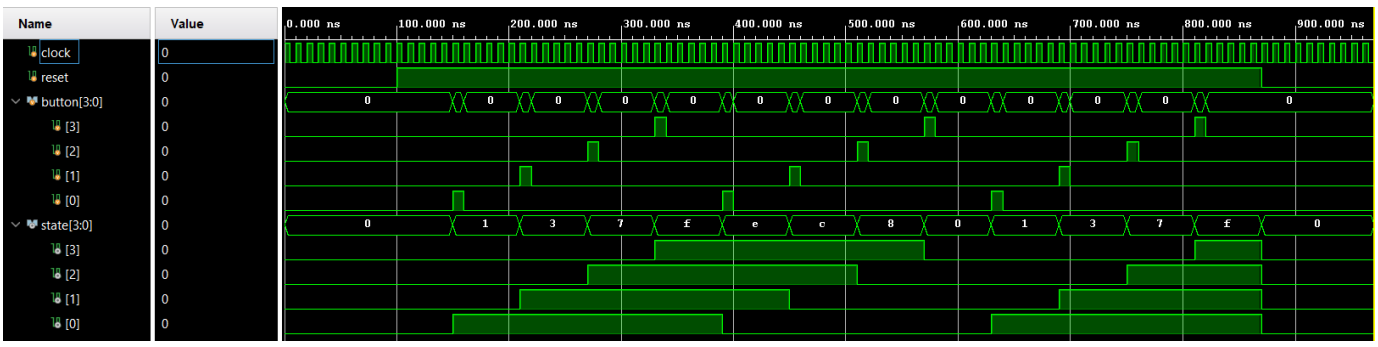
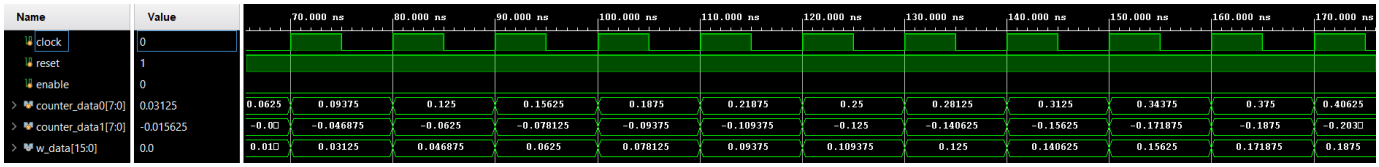
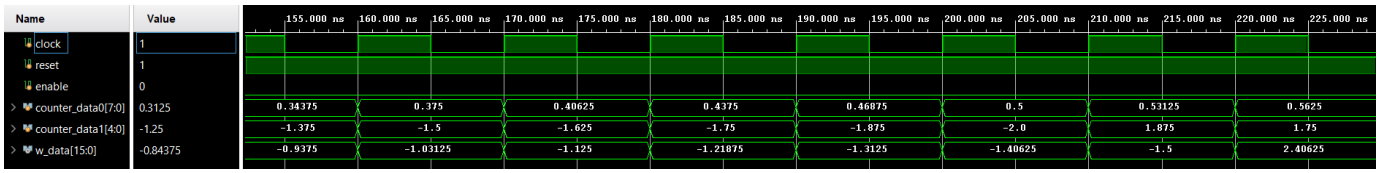


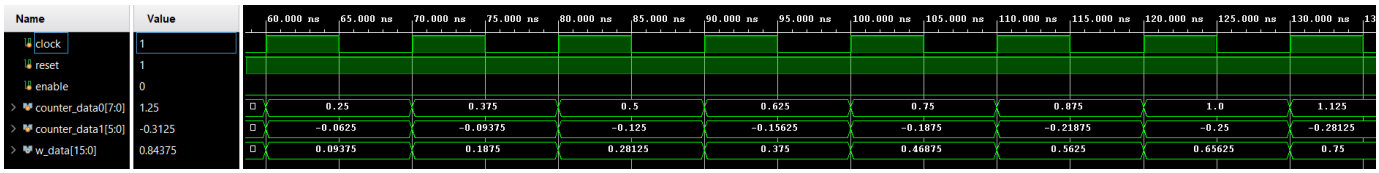
Figura 2. Resultado de simulación. Testbench: *tb_button_logic*.



(a) Adición S(8, 6) con S(8, 6).

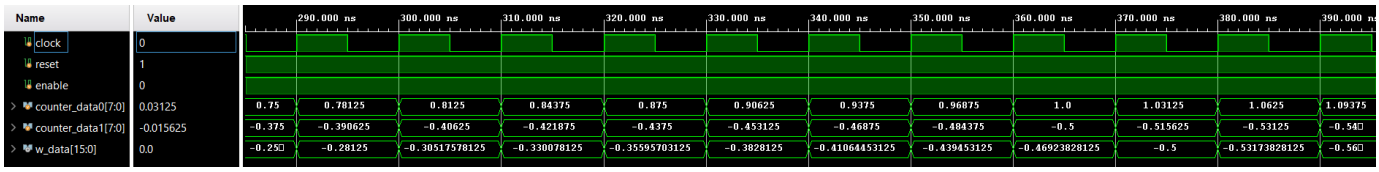


(b) Adición S(8, 6) con S(5, 3).

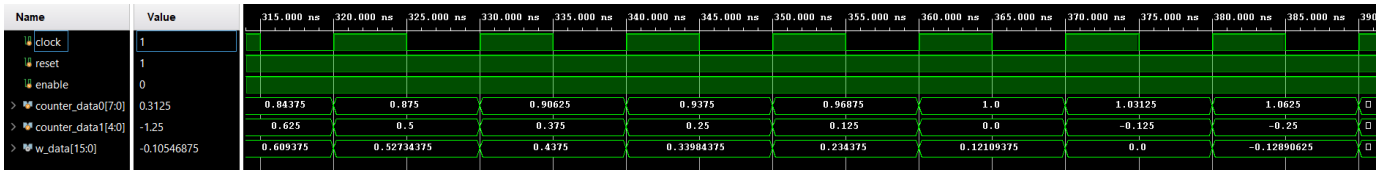


(c) Adición S(8, 4) con S(6, 5).

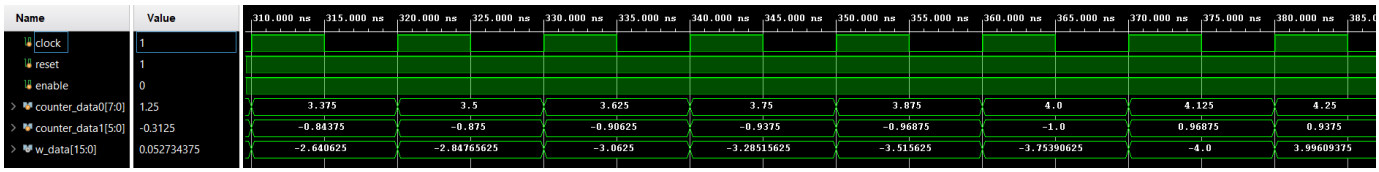
Figura 3. Resultado de simulación. Testbench: *tb_adder_mult*, adición.



(a) Producto S(8, 6) con S(8, 6).



(b) Producto S(8, 6) con S(5, 3).



(c) Producto S(8, 4) con S(6, 5).

Figura 4. Resultado de simulación. Testbench: *tb_adder_mult*, producto.

III-B. Testbench tb_adder_mult

Se escribió un testbench para el correcto funcionamiento del módulo de operación aritmética. Los resultados fueron los de las figuras 3 y 4. Recordar que el resultado se encuentra desfasado un pulso de clock respecto de los datos de entrada.

III-C. Testbench tb_ram_save y tb_gen_fir

En el laboratorio 3, se implementó un testbench para el módulo *ram_save*, *tb_ram_save*, como en el trabajo no se modificó se lo considera funcionando correctamente.

Sucede algo similar para el módulo *gen_fir*, se modificó el testbench del laboratorio anterior donde se testeaba generación y filtro como instancias separadas. En este caso se instanció directamente el módulo *gen_fir*, *tb_gen_fir*, los resultados de la simulación se resumen en la figura 5 (iguales al laboratorio 3).

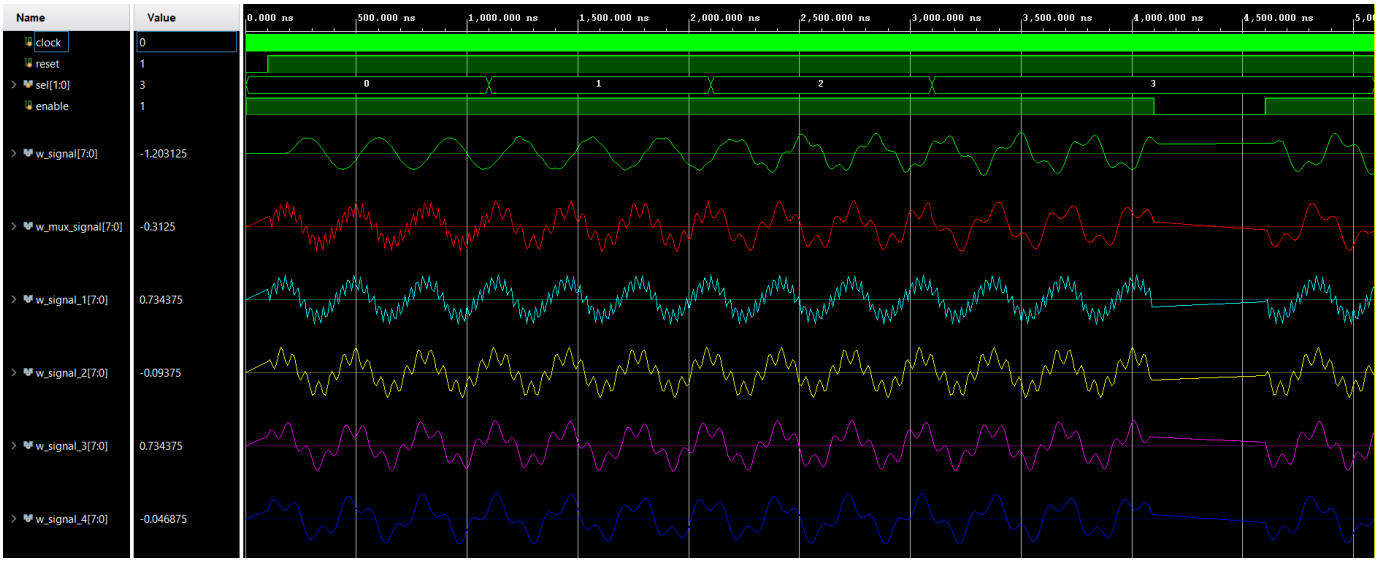


Figura 5. Resultados de simulación. Testbench *tb_gen_fir*.

III-D. Testbench tb_top

Por último, se escribió un testbench para la simulación del módulo top, que integra el resto de los módulos.

Las acciones del testbench son, luego de inicializar las entradas, quitar el estado de reset y habilitar la generación, filtrado y operación con señales. Luego, un bucle independiente va cambiando la señal seleccionada en cada módulo *gen_fir* cada determinado intervalo de tiempo, lo mismo (pero en un intervalo de tiempo mayor) con la entrada de selección entre suma o producto. Por último, otro bucle independiente habilita la escritura de las memorias, espera el flag de memoria llena y procede a realizar una lectura de ambas por un determinado intervalo de tiempo.

Para exponer los datos de la simulación de forma clara, fue necesario dividir tanto los datos de entrada a la primer memoria como los de salida en los 8 bits menos significativos y los 8 más significativos (recordar que la información se encuentra concatenada). En las figuras 6 y 7 se puede observar (señales analógicas) *w_data_ram0* señal de entrada a memoria, *log_data_from_ram0* datos leídos, en rojo los 8 bits menos significativos, en verde los 8 bits más significativos. Además, en amarillo se puede observar el resultado de la operación aritmética entre las señales filtradas, y en magenta dicho resultado pero leído de la segunda memoria (*log_data_from_ram1*).

Todas las señales analógicas se encuentran escaladas entre -2 y 2, por lo que puede verificarse visualmente cómo la adición (figura 6) es del doble de amplitud de la señal original, y el resultado de la multiplicación (figura 7) tiene una amplitud menor (valor de señales originales menor que 1). Otra observación para verificar visualmente la multiplicación es cómo los resultados son solo positivos, ya que las señales de entrada coinciden en hemiciclos positivos y hemiciclos negativos.

Otro punto importante a resaltar, es la necesidad de cambiar la configuración de punto fijo para visualizar correctamente la adición (6 bits fraccionales) y el producto (12 bits fraccionales).

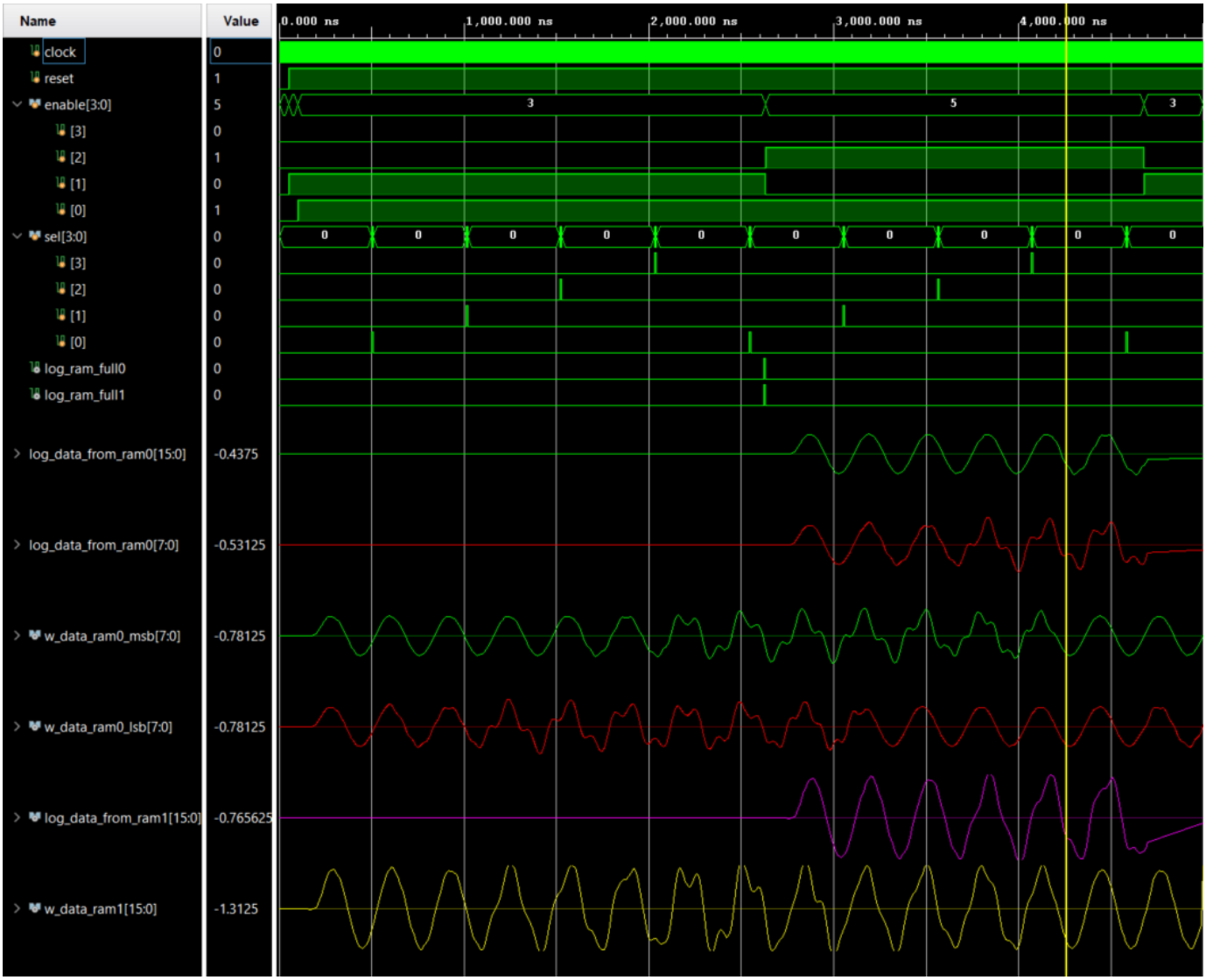


Figura 6. Resultados de simulación. Testbench *tb_top*, adición.

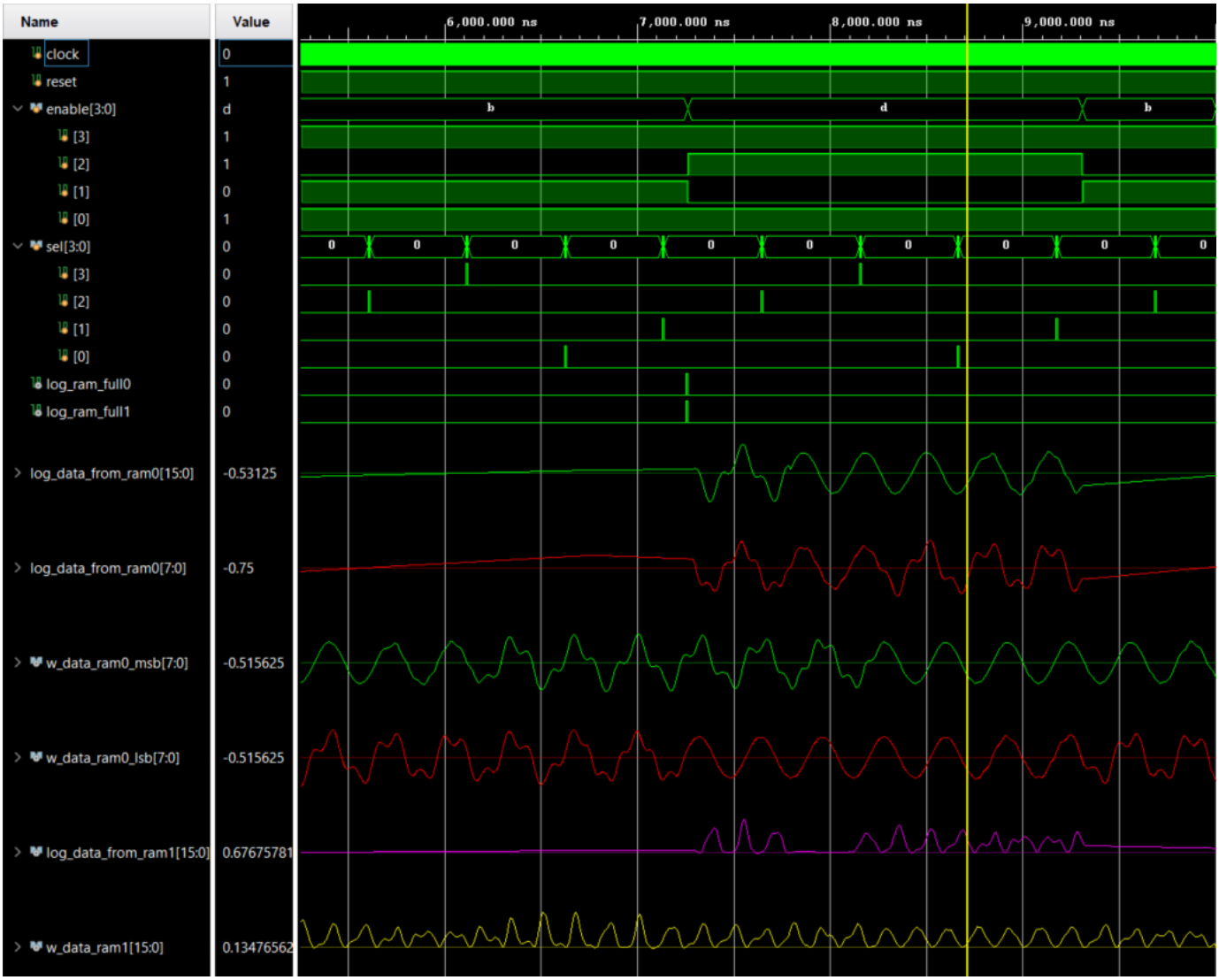


Figura 7. Resultados de simulación. Testbench *tb_top*, producto.

IV. Síntesis

En la carpeta **synthesis** pueden encontrarse los reportes de síntesis y de utilización obtenidos al sintetizar el proyecto. El detalle de celdas se encuentra en el reporte de síntesis *top_vio_ila.vds*. En el cuadro III se encuentra el detalle de celdas por jerarquía, en el cuadro I el detalle del mapeo de Block RAM y en el cuadro II el reporte de shift register estáticos. No se coloca el detalle del mapeo preliminar de DSP en el informe por su extensión, pero se encuentra en el reporte de síntesis.

Module Name	RTL Object	PORT A (Depth x Width)	W	R	PORT B (Depth x Width)	W	R	Ports driving FF	RAMB18	RAMB36
u_top/\ram_save0/u_bram	ram_reg	1 K x 16 (READ_FIRST)	W		1 K x 16 (WRITE_FIRST)		R	Port A and B	1	0
u_top/\ram_save1/u_bram	ram_reg	1 K x 16 (READ_FIRST)	W		1 K x 16 (WRITE_FIRST)		R	Port A and B	1	0

Cuadro I
Reporte de síntesis: Mapeo final de Block RAM.

Module Name	RTL Name	Length	Width	Reset Signal	Pull out first Reg	Pull out last Reg	SRL16E	SRLC32E
top	gen_fir1/u_fir_filter/register_reg[5][7]	4	16	YES	NO	YES	16	0
top	gen_fir0/u_fir_filter/register_reg[5][7]	4	16	YES	NO	YES	16	0

Cuadro II
Reporte de síntesis: Shift Register estáticos.

Instance	Module	Cells	
1	top		2329
2	u_vio	vio	9
3	u_top	top	2318
4	gen_fir0	gen_fir	1059
5	u_fir_filter	fir_filter_4	397
6	u_signal_mux	signal_mux_5	662
7	u_signal_gen_1	signal_generator_6	198
8	u_signal_gen_2	signal_generator__parameterized0_7	129
9	u_signal_gen_3	signal_generator__parameterized1_8	183
10	u_signal_gen_4	signal_generator__parameterized2_9	127
11	gen_fir1	gen_fir_0	1072
12	u_fir_filter	fir_filter	440
13	u_signal_mux	signal_mux	632
14	u_signal_gen_1	signal_generator	168
15	u_signal_gen_2	signal_generator__parameterized0	129
16	u_signal_gen_3	signal_generator__parameterized1	183
17	u_signal_gen_4	signal_generator__parameterized2	127
18	ram_save0	ram_save	55
19	u_bram	bram_2	1
20	u_ram_fsm	ram_fsm_3	5
21	ram_save1	ram_save_1	45
22	u_bram	bram	1
23	u_ram_fsm	ram_fsm	5
24	u_adder_mult	adder_mult	54
25	u_sel_logic	button_logic	12

Cuadro III
REPORTE DE SÍNTESIS: DETALLE DE CELDAS POR JERARQUÍAS.

V. IMPLEMENTACIÓN

En la carpeta **implementation** pueden encontrarse los reportes de timing e histogramas obtenidos al implementar el proyecto para 100 y 200 MHz. En el cuadro IV se observa la comparación de slack entre implementación en las dos frecuencias y en las figuras 8 y 9 los histogramas de timing correspondientes. Puede observarse que para un clock con frecuencia de 200MHz el timing falla, los requerimientos no se cumplen. Por lo tanto, la implementación en el hardware fue para 100MHz.

f_{clk}	Worst Negative Slack (WNS)	Total Negative Slack (TNS)	Worst Hold Slack (WHS)	Worst Pulse Width Slack (WPWS)
100MHz	0.544 ns	0.000 ns	0.031 ns	3.750 ns
200MHz	**-2.650 ns**	**-99.087 ns**	0.036 ns	1.116 ns

Cuadro IV
COMPARACIÓN DE SLACK (REPORTE DE TIMING).

Figura 8. Reporte de timing: Histograma para 100 MHz.

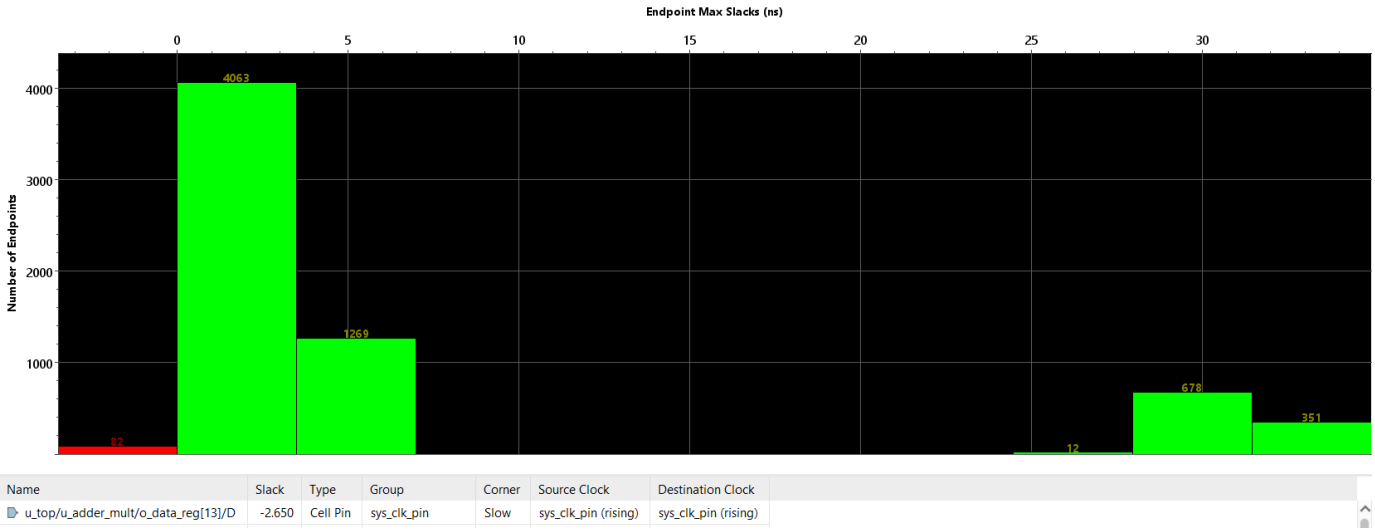
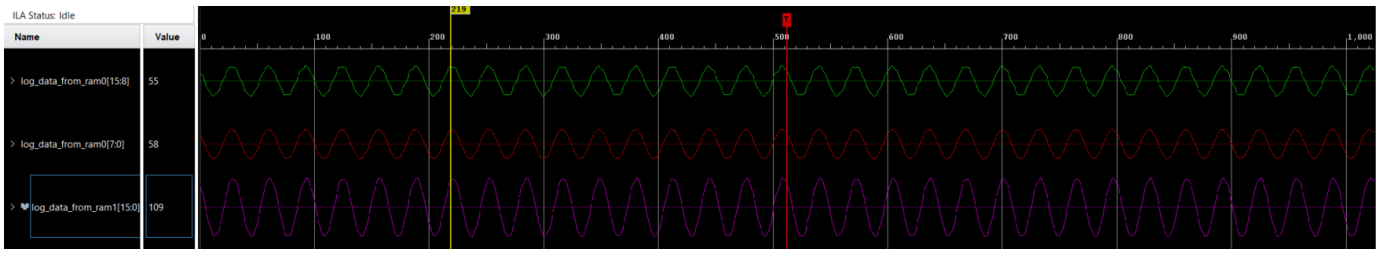


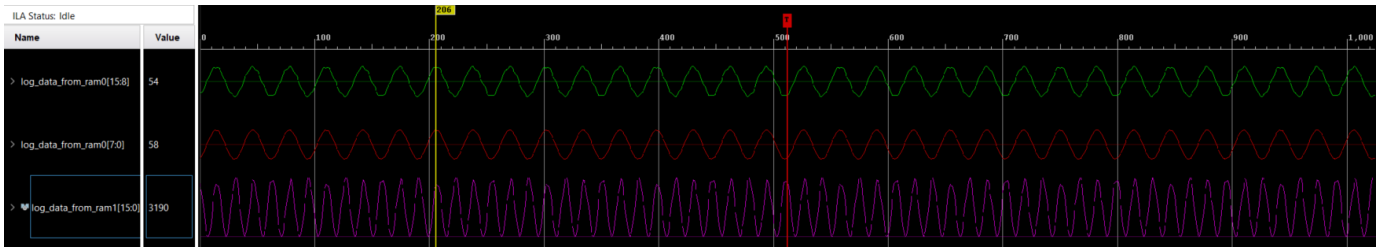
Figura 9. Reporte de timing: Histograma para 200 MHz.

VI. HARDWARE

Se programó el proyecto en FPGA remota y se interactuó con la placa a través del programa Vivado con las herramientas VIO e ILA. Algunos de los resultados obtenidos fueron los de la figura 10.



(a) Salida ILA: Adición de señales de filtradas.



(b) Salida ILA: Multiplicación de señales de filtradas.

Figura 10. Implementación en hardware: Información de ILA