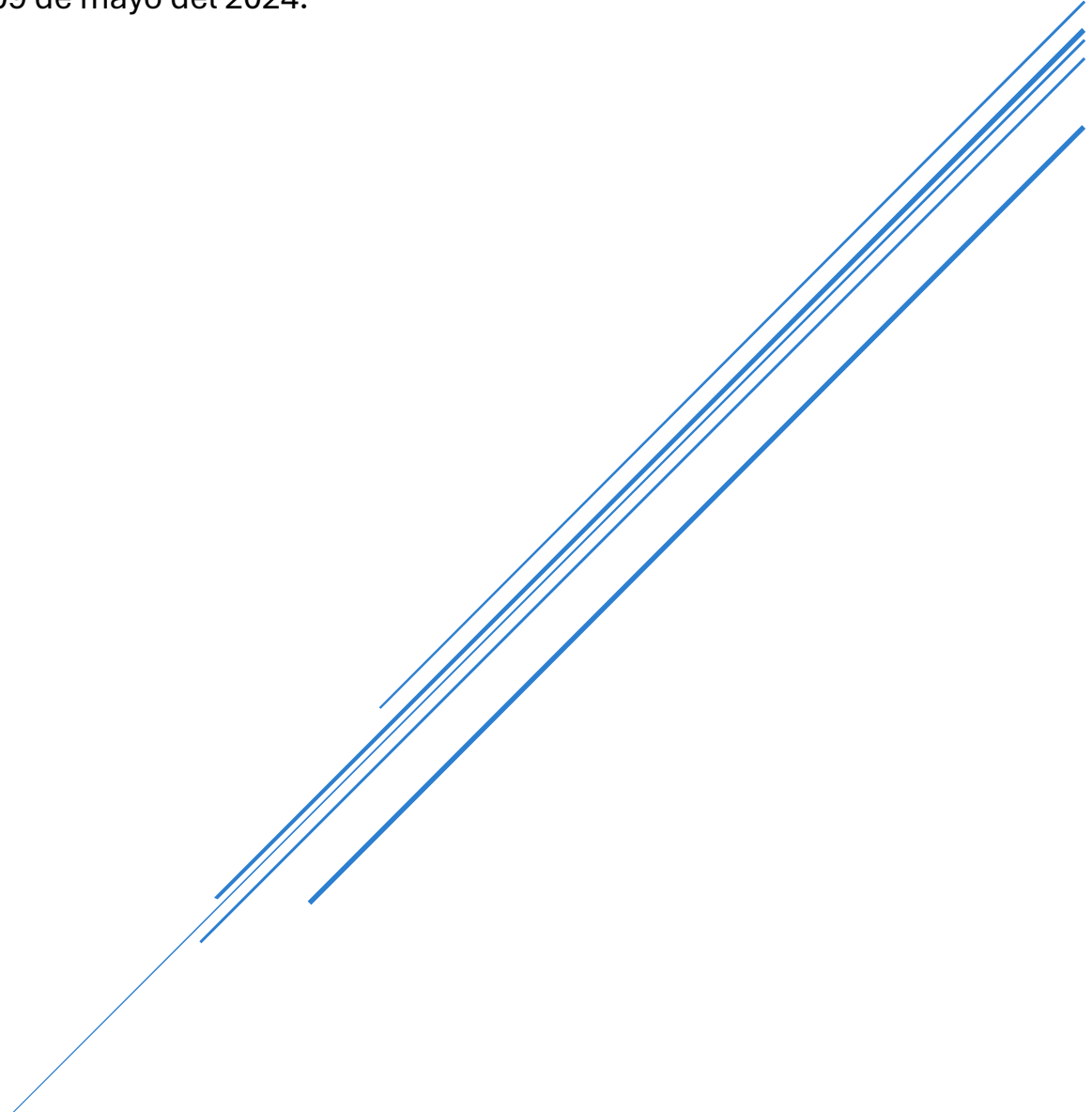


CONVOLUTION PROCESSOR

Reporte

Alumno: Anette Mora Plascencia.

Fecha: 09 de mayo del 2024.



TAE en Sistemas Embebidos
Metodología de diseño de System-On-Chip

Tabla de contenido

Descripción del problema.	2
Diagrama de caja negra y definición de señales de entrada y salida.	3
Pseudo code.	4
Validación de funcionamiento de pseudocódigo en C y Matlab.	5
Código C:.....	6
Diagrama ASM.	7
Data Path y Maquina de estados.	8
Data path.....	9
Máquina de estados.	10
Resultados de simulación y síntesis en FPGA.	11
Esquemático Top Level.	11
Programa generador de archivos para pruebas.	11
Wafeform de simulacion.	13
Área.	13
Máxima Frecuencia.	14
Ciclos de reloj necesarios para la realización de la convolución de 2 señales (5 y 10 muestras).	14

Descripción del problema.

Implementación de un coprocesador de convolución, donde los datos provenientes de la señal Y (señal desplazada) son de una Memoria de 32 direcciones externa y los datos de la señal H (señal fija) es proveniente de una memoria interna en el convolucionador.

Manualmente la resolución de la convolución utiliza la siguiente formula:

$$Z[n] = \sum_{k=0}^{M-1} H[k] \cdot Y[n - k]$$

Donde:

$Z[n]$ → Resultado de la convolución en la posición n.

$H[k]$ → Señal fija en la posición k.

$Y[n - k]$ → Es la señal de la entrada desplazada en k posiciones respecto a n.

La convolución en el coprocesador implica la multiplicación y suma de productos entre los valores de la señal fija y los valores desplazados de la señal de entrada utilizando la formula anterior. Este proceso es repetido en cada posición n de la señal de salida.

El resultado de la convolución tendrá $n + m - 1$ sumas de productos. Es decir, se obtendrá a la salida, un vector de este tamaño donde:

n → Número de datos en la señal fija H.

m → Número de datos en la señal de entrada Y.

El proceso se encarga de desplazar la señal de entrada, utilizando sus posiciones en el vector, y multiplicarla con la señal fija, de igual manera utilizando las posiciones, solo se realiza la multiplicación de los números en posición que se cruce, como se observa en la siguiente imagen, y se realiza la suma de productos correspondiente.

			H0	H1	H2	...	Hn-1						
Ym-1	...	Y1	Y0							Z0 =	H0*Y0		
		Ym-1	...	Y1	Y0					Z1 =	H0*Y1 + H1*Y0		
			Ym-1	...	Y1	Y0				Z2 =	H0*Y.. + H1*Y1 + H2 * Y0		
				Ym-1	...	Y1	Y0			Z3 =	H0*Ym-1 + H1*Y... + H2*Y1 + H...*Y0		
					Ym-1	...	Y1	Y0		Z4 =	H1*Ym-1 + H2*Y... + H.. *Y... + Hn-1 *Y1		
						Ym-1	...	Y1	Y0	Z5 =	H2 * Ym-1 + H...*Y... + Hn-1*Y...		
							Ym-1	...	Y1	Y0	Z... =	H... *Ym- + Hn-1*Y...	
								Ym-1	...	Y1	Y0	Zm+n-1 =	Hn-1*Ym-1

Ilustración 1. Proceso de convolución.

Diagrama de caja negra y definición de señales de entrada y salida.

El diagrama de caja negra del convolucionador, nos muestra explícitamente las señales de salida y entrada que este tendrá, así como el ancho de bits de cada una de estas. Se puede observar que de entrada tendremos 3 señales, la que nos indicará que inicie el proceso (Start), dataY de 8 bits, nos proporcionará el dato que haya en la dirección memY_addr de 5 bits que se enviará previamente a la memoria externa conectada al convolucionador. Además, tenemos una entrada sizeY de 5 bits, que nos indicará el numero de datos que hay en la memoria (desde 0 hasta 32), todas estas señales, nos servirán para la lectura de la memoria Y. Por otro lado, tenemos las salidas busy y done, ambas de 1 bit. Busy se mantendrá en un nivel alto mientras el convolucionador esté trabajando, mientras que donde, únicamente será activada en un ciclo de reloj, indicando que ha terminado el proceso. Las salidas a la memoria Z, son dataZ, de 16 bits, que nos indicará el valor a guardar en la dirección memZ_addr de 6 bits previamente mostrada (0-63 direcciones), y por último la señal writeZ de un bit, activada para que se pueda guardar el dataZ en la memoria.

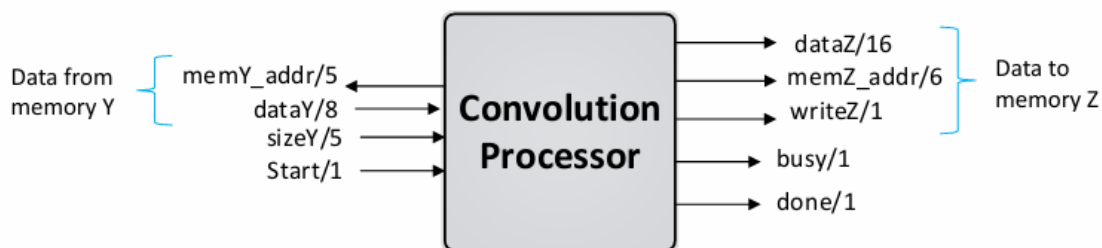


Ilustración 2. Diagrama de caja negra del convolution Processor.

Pseudo code.

Este algoritmo recorre todas las posiciones n del resultado de la convolución y realiza la suma de productos entre la señal de entrada desplazada por el kernel en cada posición n. Al envío de la señal de inicio, la señal de busy se pone en alto para indicar que se está realizando un proceso.

Dentro del segundo for anidado, se realiza el producto de los datos necesarios (esto condicionado por el if) y se acumulan para que una vez se realicen todos, salir del for y mostrar que se va a escribir en memoria, el dato y la dirección en la que este se colocará, esta última dada por el primer for. Una vez obtenido todos los datos de la convolución en cada una de las direcciones necesarias, se activa la señal de Done que nos indica que el proceso se ha completado.

```
1.- while Start = 0
2.- end while
3.-
4.- busy = 1;
5.- done = 0;
6.- writeZ = 0;
7.- sizeY_temp = sizeY;
8.-
9.- tam_convolucion = sizeH_int + sizeY_temp -1;
10.-
11.- for (i = 0; i < tam_convolucion; i++)
12.-
13.-     dataZ_temp = 0;
14.-     for (j = 0; j < sizeY_temp; j++)
15.-         if ((i - j) >= 0 && (i - j) < sizeX_int)
16.-             memH_addr = i - j;
17.-             memY_addr = j;
18.-             dataH = memory_H[memH_addr];
19.-             dataY_temp = dataY;
20.-             mult_temp = dataH * dataY_temp;
21.-             dataZ_temp = dataZ_temp + mult_temp;
22.-         end if
23.-     end for
24.-     memZ_addr = i;
25.-     dataZ = dataZ_temp;
26.-     writeZ = 1;
27.-     writeZ = 0;
28.- end for
29.- busy = 0;
30.- done = 1;
31.- done = 0;
32.- if Start=0
33.- goto (while Start = 0)
34.- else goto (done = 1)
```

Ilustración 3. Pseudo code del convolucionador.

Validación de funcionamiento de pseudocódigo en C y Matlab.

Para validar que el pseudocódigo seleccionado, se realiza la implementación de este en C, realizando varios ejemplos de convolución y comparado con el resultado obtenido en Matlab que nos realiza este proceso.

Resultado C	Resultado Matlab
<pre> Busy: 1 writeZ: 1, dataZ: 1, memZ_addr: 0 Busy: 1 writeZ: 1, dataZ: -2, memZ_addr: 1 Busy: 1 writeZ: 1, dataZ: -5, memZ_addr: 2 Busy: 1 writeZ: 1, dataZ: -4, memZ_addr: 3 Busy: 1 writeZ: 1, dataZ: 10, memZ_addr: 4 Busy: 1 writeZ: 1, dataZ: 19, memZ_addr: 5 Busy: 1 writeZ: 1, dataZ: 19, memZ_addr: 6 Busy: 1 writeZ: 1, dataZ: 6, memZ_addr: 7 done: 1 </pre>	<pre> >> H = [-1 2 3 5 2]; >> Y = [-1 0 2 3]; >> Z = conv(H,Y) Z = 1 -2 -5 -4 10 19 19 6 </pre>
<pre> Busy: 1 writeZ: 1, dataZ: -2, memZ_addr: 0 Busy: 1 writeZ: 1, dataZ: -1, memZ_addr: 1 Busy: 1 writeZ: 1, dataZ: 2, memZ_addr: 2 Busy: 1 writeZ: 1, dataZ: 2, memZ_addr: 3 Busy: 1 writeZ: 1, dataZ: 0, memZ_addr: 4 Busy: 1 writeZ: 1, dataZ: -1, memZ_addr: 5 done: 1 </pre>	<pre> >> H = [2 1 0 -1]; >> Y = [-1 0 1]; >> Z = conv(H,Y) Z = -2 -1 2 2 0 -1 </pre>
<pre> Busy: 1 writeZ: 1, dataZ: 5, memZ_addr: 0 Busy: 1 writeZ: 1, dataZ: 5, memZ_addr: 1 Busy: 1 writeZ: 1, dataZ: 21, memZ_addr: 2 Busy: 1 writeZ: 1, dataZ: 5, memZ_addr: 3 Busy: 1 writeZ: 1, dataZ: 77, memZ_addr: 4 Busy: 1 writeZ: 1, dataZ: 38, memZ_addr: 5 Busy: 1 writeZ: 1, dataZ: -31, memZ_addr: 6 Busy: 1 writeZ: 1, dataZ: -21, memZ_addr: 7 Busy: 1 writeZ: 1, dataZ: 0, memZ_addr: 8 Busy: 1 writeZ: 1, dataZ: 1, memZ_addr: 9 done: 1 </pre>	<pre> >> H = [5 -5 11 13 2 -1]; >> Y = [1 2 4 -2 -1]; >> Z = conv(H,Y) Z = 5 5 21 5 77 38 -31 -21 0 1 </pre>
<pre> Busy: 1 writeZ: 1, dataZ: -5, memZ_addr: 0 Busy: 1 writeZ: 1, dataZ: -3, memZ_addr: 1 Busy: 1 writeZ: 1, dataZ: 4, memZ_addr: 2 Busy: 1 writeZ: 1, dataZ: 3, memZ_addr: 3 Busy: 1 writeZ: 1, dataZ: -1, memZ_addr: 4 Busy: 1 writeZ: 1, dataZ: -6, memZ_addr: 5 Busy: 1 writeZ: 1, dataZ: 2, memZ_addr: 6 Busy: 1 writeZ: 1, dataZ: 6, memZ_addr: 7 done: 1 </pre>	<pre> >> H = [5 3 1 0 2 6]; >> Y = [-1 0 1]; >> Z = conv(H,Y) Z = -5 -3 4 3 -1 -6 2 6 </pre>

Código C:

```
1  #include <stdio.h>
2
3  int main () {
4      int sizeY_temp = 3;
5      int sizeH_temp = 6;
6      int dataZ_temp;
7      int memH_addr, memY_addr, memZ_addr;
8      int k;
9      int dataH, dataY, dataZ;
10     int memory_H[] = {5, 3, 1, 0, 2, 6};
11     int memory_Y[] = {-1, 0, 1};
12
13     int busy = 1;
14     int done = 0;
15     int writeZ = 0;
16     printf("Busy: %i\n", busy);
17     int tam_conv = sizeH_temp + sizeY_temp - 1;
18
19     for(int i = 0; i < tam_conv; i++){
20         dataZ_temp = 0;
21         for(int j = 0; j < sizeY_temp; j++){
22             if(((i - j) >= 0) && ((i - j) < sizeH_temp)){
23                 memH_addr = i - j;
24                 memY_addr = j;
25                 dataH = memory_H[memH_addr];
26                 dataY = memory_Y[memY_addr];
27                 int mult_temp = dataH*dataY;
28                 dataZ_temp = dataZ_temp + (mult_temp);
29             }
30         }
31         memZ_addr = i;
32         dataZ = dataZ_temp;
33         writeZ = 1;
34         printf("writeZ: %i, dataZ: %i, memZ_addr: %i \n", writeZ, dataZ, memZ_addr);
35         writeZ = 0;
36     }
37     busy = 0;
38     done = 1;
39     printf("Busy: %i\n", busy);
40     printf("done: %i\n", done);
41     done = 0;
42
43     return 0;
44 }
```

Ilustración 4. Código en C implementado para validación del pseudo code.

Diagrama ASM.

A partir del modelo de oro, el algoritmo previamente validado, se realiza el diagrama ASM, donde observamos como será el flujo de las señales, teniendo en cada bloque de asignación expresiones a nivel RTL. Además de realizar la asignación de estados a cada bloque.

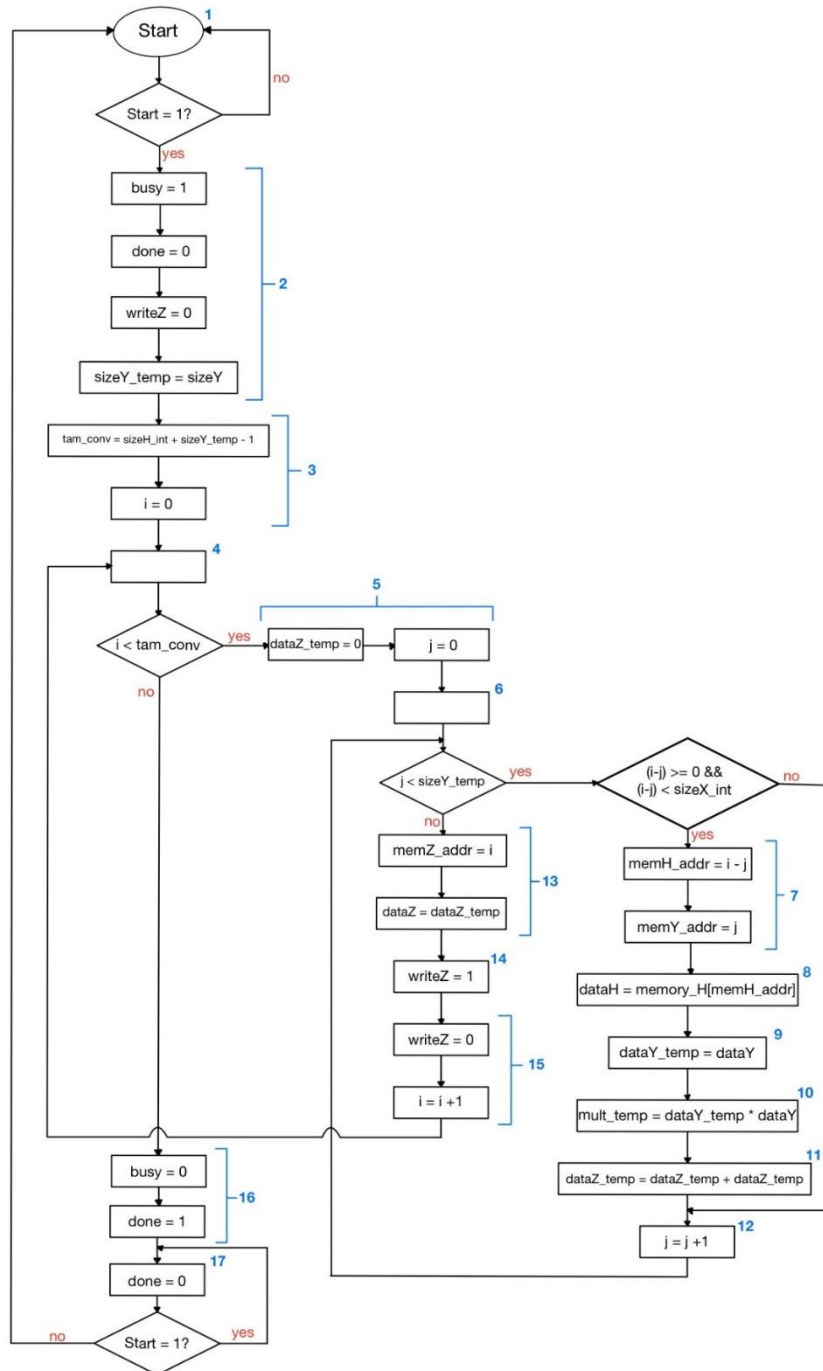


Ilustración 5. Diagrama ASM.

Data Path y Maquina de estados.

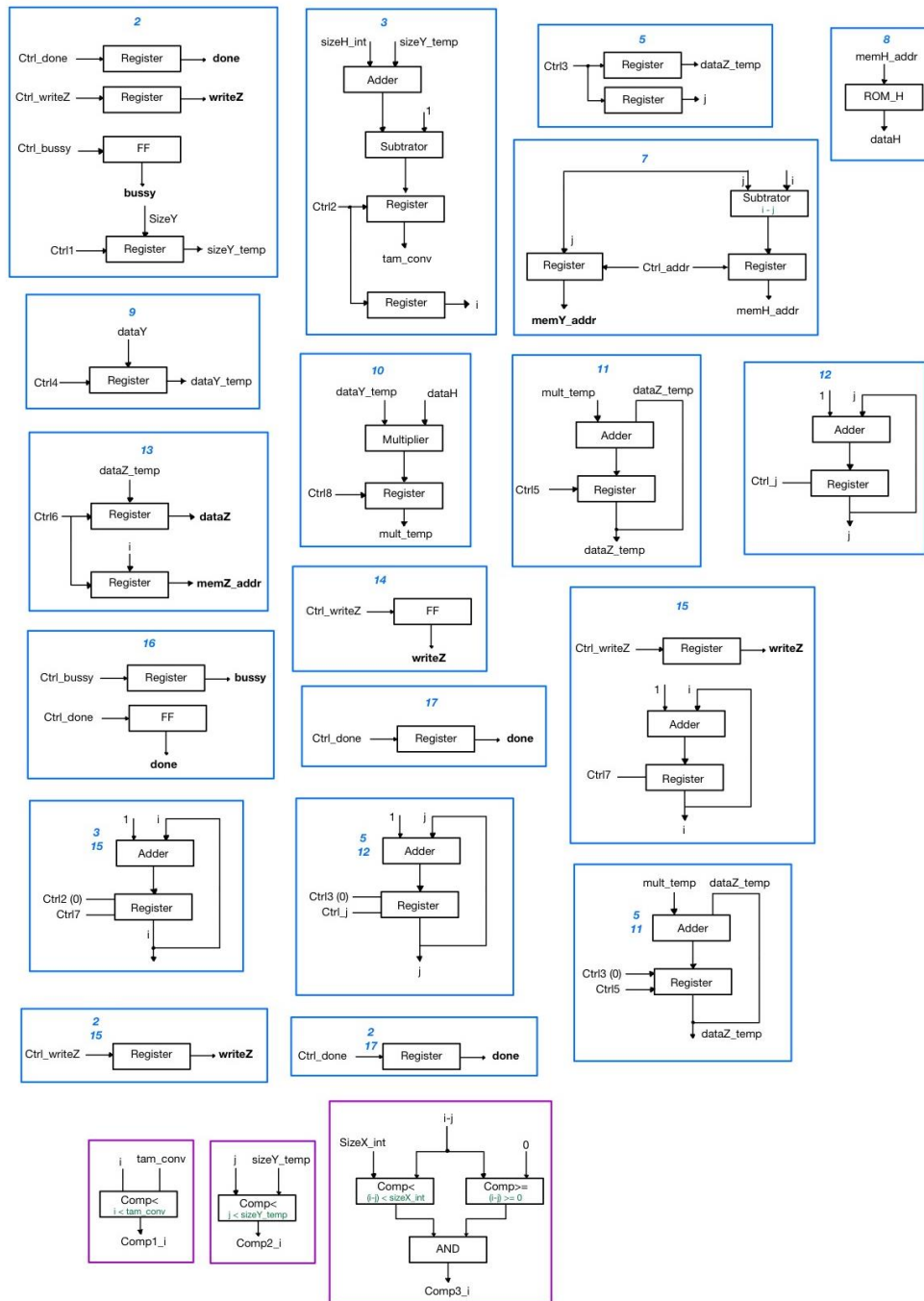


Ilustración 6. Bloques de construcción óptimos del Data Path.

Cada uno de los bloques indica el estado correspondiente de acuerdo con el diagrama ASM, así mismo, se pueden observar 5 bloques de optimización, donde indica cuales son los estados optimizados. Los 3 bloques (3-15, 5-12 y 5-11) corresponde a una optimización, donde es necesario un reset de las señales, estas se optimizan usando

un solo registro. En las otras 2 (2-15 y 2-17) únicamente, se comparte la señal de activación, ya que este necesita mandar a 0 en 2 estados diferentes una misma señal.

Una vez obtenidos los bloques y las optimizaciones que se tienen en cada estado, se procesa a unirlos, dando como resultado el siguiente Data-path.

Data path.

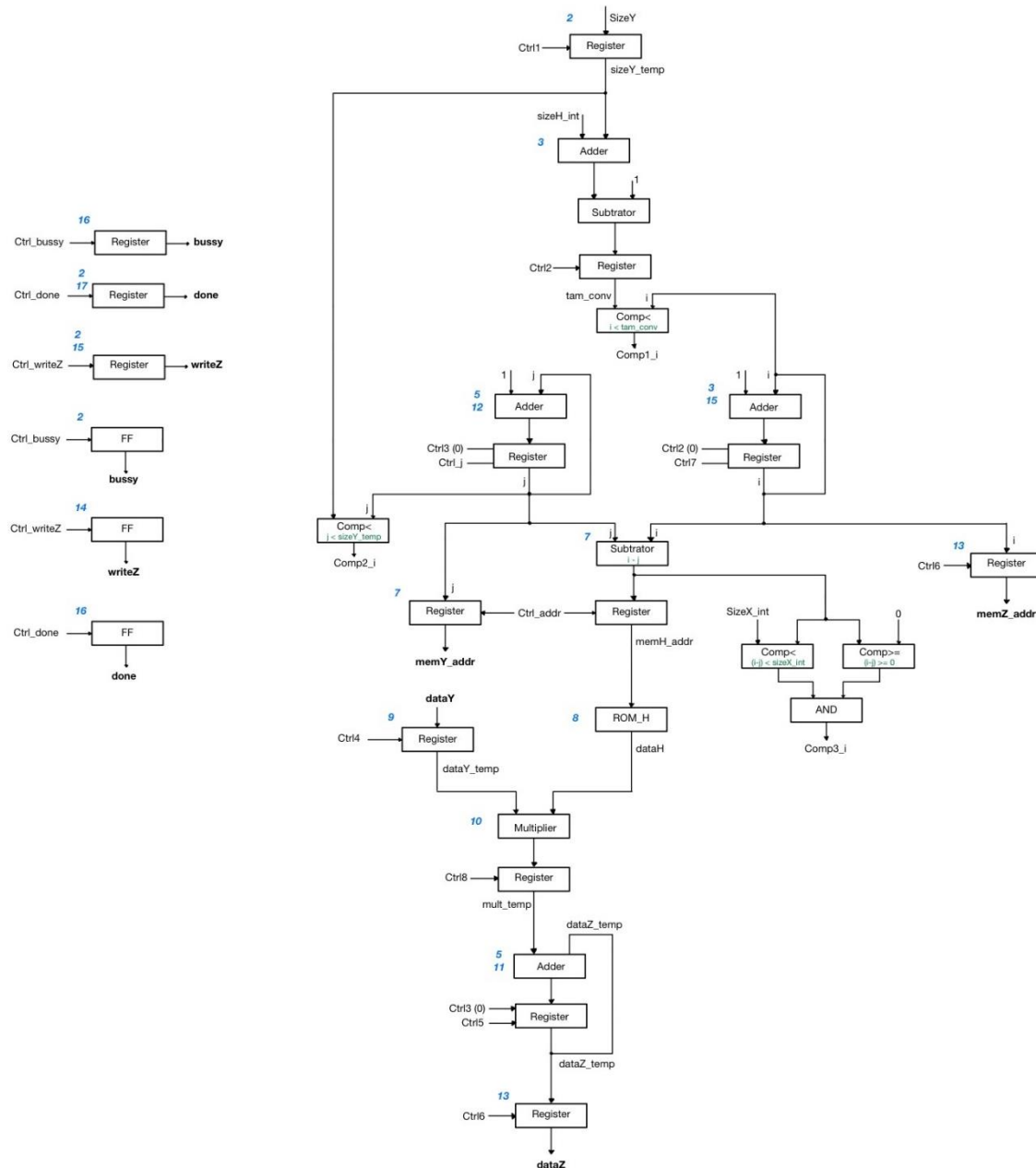
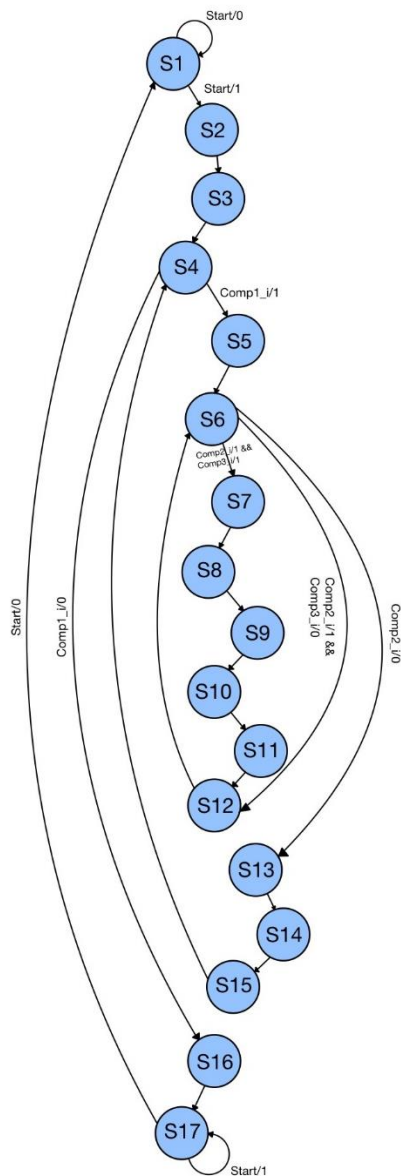


Ilustración 7. Data Path del convolucionador.

Así mismo, a partir del diagrama ASM y el Data-path, obtenemos la maquina de estados del sistema, donde, en la tabla se muestra los estados, las señales de entrada a estos estados y las señales de salida que se activan en cada estado. Además, en la columna de las entradas, se puede ver las transiciones de estados y las señales necesarias para que se realicen.

Máquina de estados.



State	Inputs/Value	Outputs/Value
1	Start/0 (S1) Start/0 (S17)	
2	Start/1 (S1)	Ctrl1/1 Ctrl_busy/1 Ctrl_done/0 Ctrl_writeZ/0
3	(S2)	Ctrl2/1 Ctrl_busy/1
4	(S3) (S15)	Ctrl_busy/1
5	Comp1_i/1 (S4)	Ctrl3/1 Ctrl_busy/1
6	(S5) (S12)	Ctrl_busy/1
7	Comp2_i/1 && Comp3_i/1(S6)	Ctrl_addr/1 Ctrl_busy/1
8	(S7)	Ctrl_busy/1
9	(S8)	Ctrl4/1 Ctrl_busy/1
10	(S9)	Ctrl_8/1 Ctrl_busy/1
11	(S10)	Ctrl5/1 Ctrl_busy/1
12	Comp2_i/1 && Comp3_i/0(S6) (S11)	Ctrl_j/1 Ctrl_busy/1
13	Comp2_i/0 (S6)	Ctrl6/1 Ctrl_busy/1
14	(S13)	Ctrl_writeZ/1 Ctrl_busy/1
15	(S14)	Ctrl7/1 Ctrl_writeZ/0 Ctrl_busy/1
16	Comp1_i/0 (S4)	Ctrl_busy/0 Ctrl_done/1
17	(S16) Start/1 (S17)	Ctrl_done/0

Tabla 1. Control FSM.

Ilustración 8. Máquina de estados del convolucionador.

Resultados de simulación y síntesis en FPGA.

Esquemático Top Level.

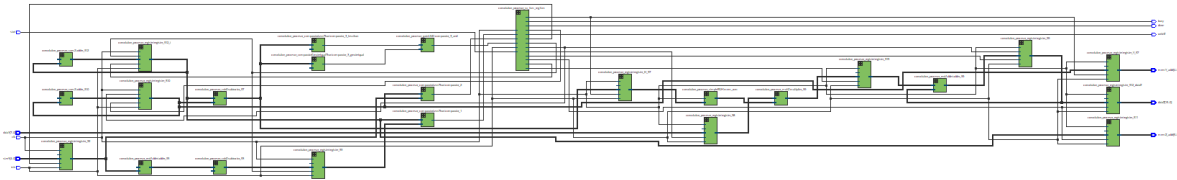


Ilustración 9. Esquemático RTL viewer Convolution Processor.

Programa generador de archivos para pruebas.

Para generar los archivos para pruebas se generó el siguiente programa, el cual guardará siempre el archivo con el mismo nombre y en la ubicación donde este este programa. Estos se guardan en hexadecimal y además, se muestra la solución de la convolución de estos números aleatorios creados (la cantidad de datos de cada archivo así como los datos en cada archivo).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define NOMBRE_ARCHIVO_H "MemH.txt"
5  #define NOMBRE_ARCHIVO_Y "MemY.txt"
6
7  int main() {
8      FILE* archivoH;
9      FILE* archivoY;
10     int sizeY_temp;
11     int sizeH_temp;
12
13     srand(time(0));
14
15     //CREACION DE ARCHIVO DE MEMORIA Y
16
17     archivoY = fopen(NOMBRE_ARCHIVO_Y, "w");
18
19     if (archivoY == NULL) {
20         printf("Error al abrir el archivo.");
21         return 1;
22     }
23
24     sizeY_temp = (rand() % 32) + 1; //TOTAL DE DATOS EN Y
25     int numerosY[sizeY_temp];
26     printf("sizeY_temp: %i\n", sizeY_temp);
27
28     printf("Y = [");
29     for (int i = 0; i < sizeY_temp - 1; i++){
30         int numero = rand() % (256 * 2) - 128;
31         printf(" %02X ", numero & 0xFF);
32         fprintf(archivoY, "%02X\n", numero & 0xFF);
33         numerosY[i] = numero;
34     }
35     int numero = rand() % (256 * 2) - 128;
36     fprintf(archivoY, "%02X", numero & 0xFF);
37     printf(" %02X ", numero & 0xFF);
38     numerosY[sizeY_temp-1] = numero;
39     printf("]\n\n");
40     fclose(archivoY);
41 }
```

```
42 //CREACION DE ARCHIVO DE MEMORIA H
43
44 archivoH = fopen(NOMBRE_ARCHIVO_H, "w");
45
46 if (archivoH == NULL) {
47     printf("Error al abrir el archivo.");
48     return 1;
49 }
50
51 sizeH_temp = (rand() % 32) + 1; //TOTAL DE DATOS EN H
52 int numerosH[sizeH_temp];
53 printf("sizeH_temp: %i\n", sizeH_temp);
54
55 printf("H = [");
56 for (int i = 0; i < sizeH_temp - 1; i++){
57     int numero = rand() % (256 * 2) - 128;
58     printf(" %02X ", numero & 0xFF);
59     fprintf(archivoH, "%02X\n", numero & 0xFF);
60     numerosH[i] = numero;
61 }
62 numero = rand() % (256 * 2) - 128;
63 fprintf(archivoH, "%02X", numero & 0xFF);
64 printf(" %02X ", numero & 0xFF);
65 numerosH[sizeH_temp-1] = numero;
66 printf("]\n\n");
67 fclose(archivoH);
68
69
70 //RESOLUCION DE CONVOLUCION CON LOS DATOS GENERADOS Y GUARDADOS EN ARCHIVO
71
72 int dataZ_temp;
73 int k, dataZ;
74 int tam_conv = sizeH_temp + sizeY_temp - 1;
75 int convolucion[tam_conv];
76
77 printf("tam_conv: %i\n", tam_conv);
78 printf("Z = [");
79 for(int i = 0; i < tam_conv; i++){
80     dataZ_temp = 0;
81     for(int j = 0; j < sizeY_temp; j++){
82         k = i - j;
83         if((k >= 0) && (k < sizeH_temp)){
84             dataZ_temp = dataZ_temp + (numerosH[k]*numerosY[j]);
85         }
86     }
87     dataZ = dataZ_temp;
88     printf(" %04X ", dataZ & 0xFFFF);
89 }
90 printf("]");
91
92 return 0;
93 }
94
```

Ilustración 10. Código generador de archivos .txt para muestras.

Waveform de simulacion.

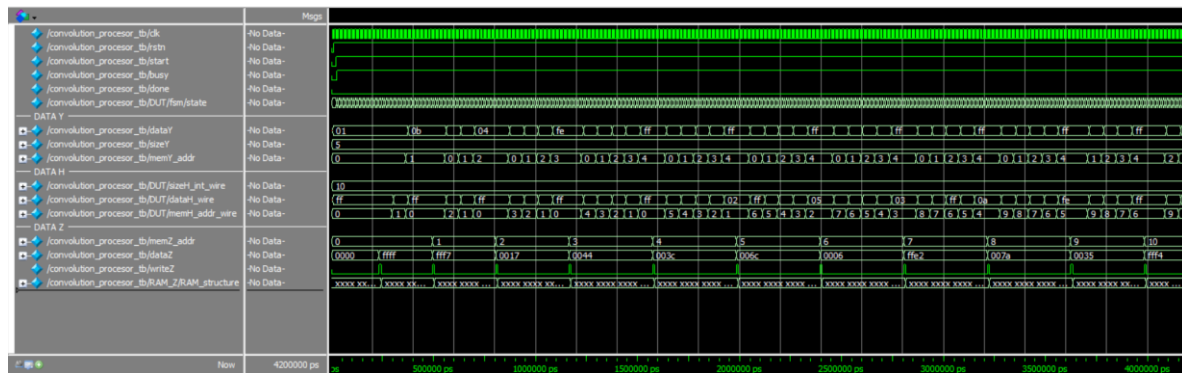


Ilustración 11. Waveforme de simulación (Radix hexa).

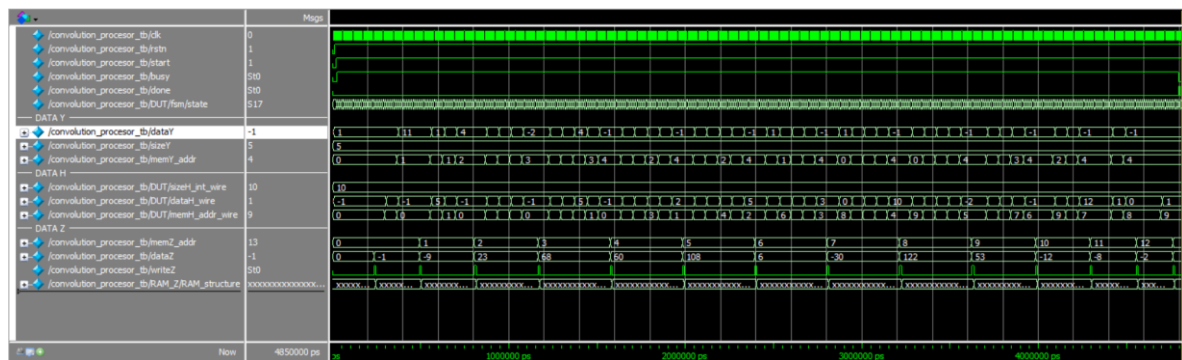


Ilustración 12. Waveforme de simulación (Radix decimal).

Área.

Flow Status	Successful - Thu May 9 21:29:19 2024
Quartus Prime Version	22.1std.1 Build 917 02/14/2023 SC Lite Edition
Revision Name	convolution_procesor
Top-level Entity Name	convolution_procesor
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	44 / 32,070 (< 1 %)
Total registers	112
Total pins	46 / 457 (10 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	1 / 87 (1 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Ilustración 13. Área de convolution Processor.

Máxima Frecuencia.


Slow 1100mV 85C Model Fmax Summary				
 <<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	298.6 MHz	298.6 MHz	clk	

Ilustración 14. Máxima frecuencia de convolution Processor.

Ciclos de reloj necesarios para la realización de la convolución de 2 señales (5 y 10 muestras).

Para la realización de la convolucion de las siguientes señales se ecesita 470 ciclos de reloj.

MemY = [01 0B 04 FE FF];

MemH = [FF 02 05 03 0A FE FF 0C 00 01];

MemZ = [FFFF FFF7 0017 0044 003C 006C 0006 FFE2 007A 0035 FFF4 FFF8 FFFE FFFF];