Cinvestav

Reporte de coprocesador de convolución

Corona Aldana Mariana

MATERIA: METODOLOGIA DE DISEÑO SYSTEM ON CHIPS

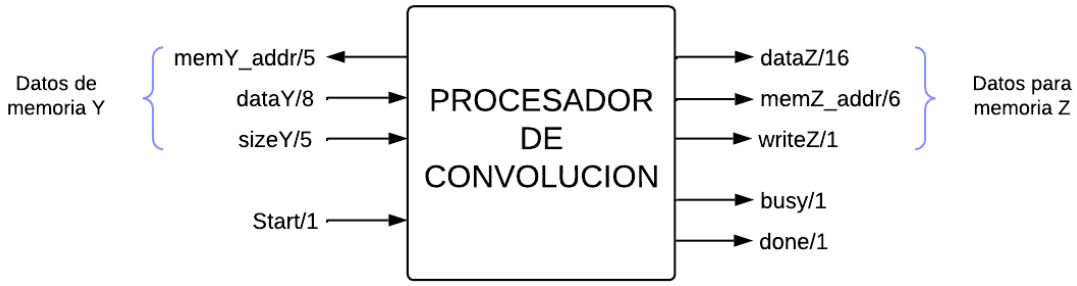
PROFESOR: VIDKAR ANIBAL DELGADO GALLARDO

DIPLOMADO TAE EN SISTEMAS EMBEBIDOS

1) Descripción del problema

Diseñar un sistema con la metodología top-down que combina software y hardware para implementar un coprocesador de convolución que calcule una convolución matricial entre una señal guardada en una memoria externa y otra guardada dentro del sistema, enviando los resultados a otra memoria de almacenamiento externa.

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



Señales de entrada

- Una entrada de control (Start), con 1 bit de longitud, que señala el inicio del proceso.
- Una entrada de datos (dataY), con 8 bits de longitud, que entrega un dato guardado en la memoria Y.
- Una entrada de datos (sizeY), con 5 bits de longitud, que describe el tamaño de la señal guardada en la memoria Y.

Señales de salida

- Una salida de control (busy), con 1 bit de longitud, que señala cuando el proceso sigue en curso.
- Una salida de control (done), con 1 bit de longitud, que señala cuando ha terminado el proceso.
- Una salida de datos (memY_addr), con 5 bits de longitud, que especifica la dirección de memoria de Y a la cual se quiere acceder.
- Una salida de datos (dataZ), con 16 bits de longitud, que entrega el valor de un dato en la memoria Z.
- Una salida de datos (memZ_addr), con 6 bits de longitud, que especifica la dirección de memoria de Z en la cual se quiere escribir.
- Una salida de control (writeZ), con 1 bit de longitud, que especifica cuando se quiere escribir en la memoria de Z.

3) Pseudocódigo y breve explicación.

ALGORITMO

1. Esperar a que se reciba la señal Start
2. Inicializar salidas de control (busySignal, doneSignal y writeZ) y variables de datos (sizeZ y romX_addr)
3. Si el valor de romX_addr es menor al de sizeZ, entrar al primer ciclo
    4. Reiniciar valores dataZ_temp y memY_addr
    5. Si el valor de memY_addr es menor al de sizeY, entrar al segundo ciclo
        6. Calcular el valor de aux como romX_addr-memY_addr, que muestra el valor de la dirección del dato deseado en ese momento de la memoria X
        7. Si el valor de aux no es negativo, entrar a la primera condición
            8. Obtener los valores de las memorias X y Y en las direcciones aux y memY_addr respectivamente
            9. Sumar el valor de dataZ_temp con la multiplicación de dataX y dataY
        10. Aumentar el valor de memY_addr
    11. Asignar valor de romX_addr a memZ_addr
    12. Escribir valor de dataZ_temp en la memoria Z con la dirección memZ_addr
    13. Aumentar el valor de romX_addr
14. Actualizar salidas de control (busySignal y doneSignal) para indicar la finalización del proceso
15. Esperar 'apagado' de la señal Start

PSEUDOCODIGO

```
1. While Start=0
2. End While
3. busySignal=1;
4. doneSignal=0;
5. writeZ=0;
6. romX_addr=0;
7. While romX_addr<sizeZ
8.    dataZ_temp=0;
9.    memY_addr=0;
10.         While memY_addr<sizeY
11.            aux= romX_addr - memY_addr;
12.            if(aux>=0 && aux<sizeX)
13.               dataZ_temp=dataZ_temp+ dataY*dataX;
14.            memY_addr=memY_addr+1;
15.         End while
16.         memZ_addr=romX_addr;
17.         writeZ=1;
18.         dataZ= dataZ_temp;
```

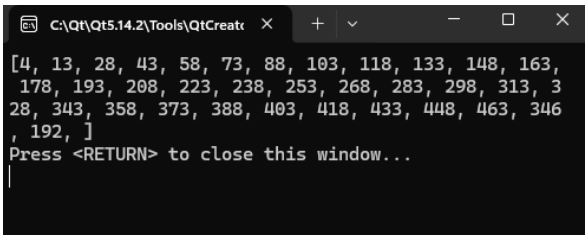
```
19.         writeZ=0;
20.         romX_addr=romX_addr+1;
21.      End while
22.      busySignal=0;
23.      doneSignal=1;
24.      doneSignal=0;
25.      if Start=0
26.         goto 1
27.      else goto 25
```

APLICACIÓN DE ALGORITMO EN C++

A continuación, se muestra una implementación de un convolucionador de dos señales guardadas en dos arreglos (en vez de memorias). Como se puede ver, la lógica y los pasos expresados en el algoritmo son los mismos que los que están en el programa.

```cpp
#include <iostream>
using namespace std;
int main()
{
    const int sizeX = 32;
    const int sizeY = 3;
    const int sizeZ = sizeX+sizeY-1;

    int memY_addr=0, memZ_addr=0, romX_addr=0, dataZ_temp, aux = 0;
    int x [sizeX] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
                     18,19,20,21,22,23,24,25,26,27,28,29,30,31,32};
    int y [sizeY] = {4,5,6};
    int z [sizeZ];

    while (romX_addr<sizeZ){
        dataZ_temp=0;
        memY_addr=0;
        while (memY_addr<sizeY){
            aux = romX_addr - memY_addr;
            if (aux>=0 && aux<sizeX)
                dataZ_temp+=y[memY_addr]*x[aux];
            memY_addr=memY_addr+1;
        }
        memZ_addr=romX_addr;
        z[memZ_addr]=dataZ_temp;
        romX_addr=romX_addr+1;
    }
    cout << "[";
    for(int i = 0; i<sizeZ; i++)
        cout << z[i] << ", ";
    cout << "]" << endl;
    return 0;
}
```

La salida de este programa es la siguiente:

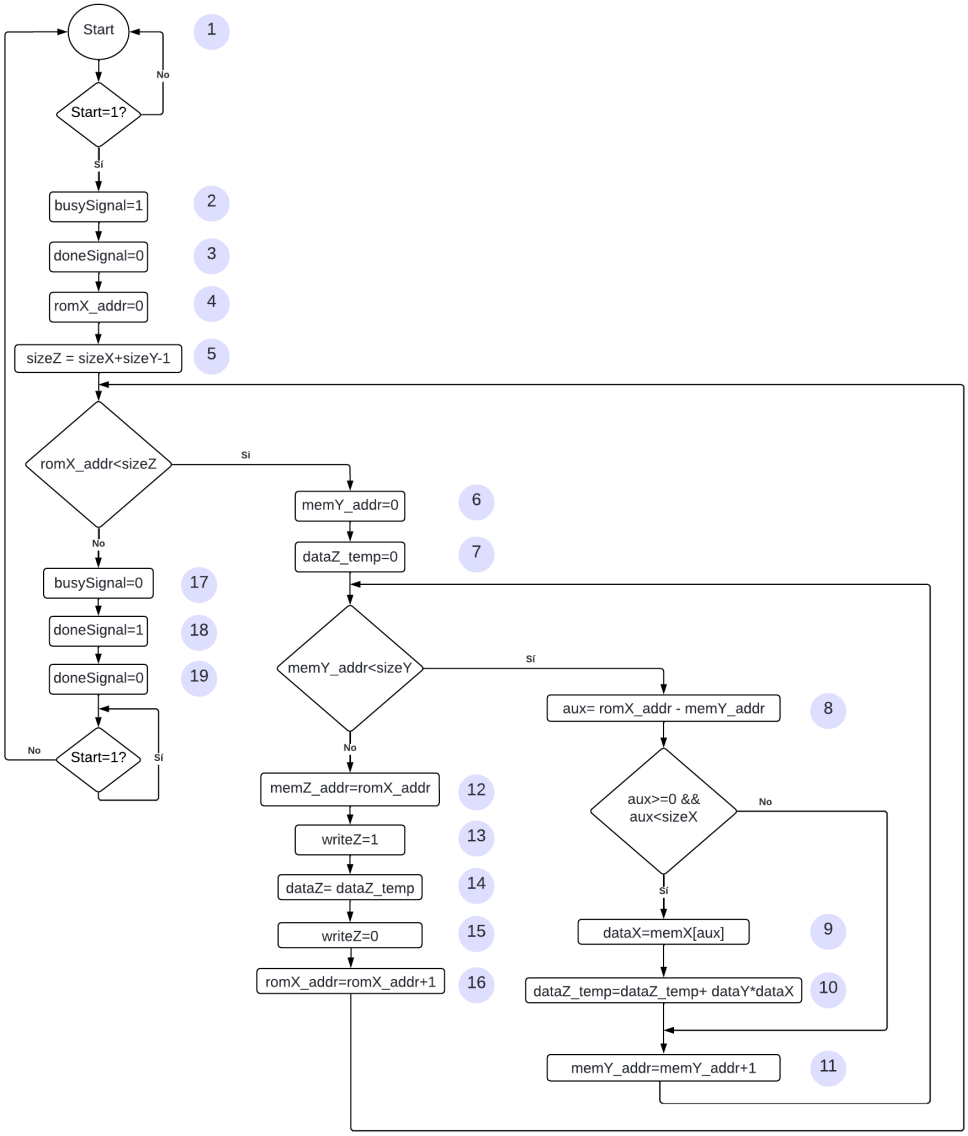


Y se comparó con la convolución de las mismas dos señales en Matlab:

```
>> u = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32];
>> v = [4 5 6];
>> w = conv(u,v)

w =

  Columns 1 through 32

     4    13    28    43    58    73    88   103   118   133   148   163   178   193   208   223   238   253   268   283   298   313   328   343   358   373   388   403   418   433   448   463

  Columns 33 through 34

   346   192
```

Como se puede ver, ambas respuestas coinciden. Demostrando el buen funcionamiento del algoritmo.

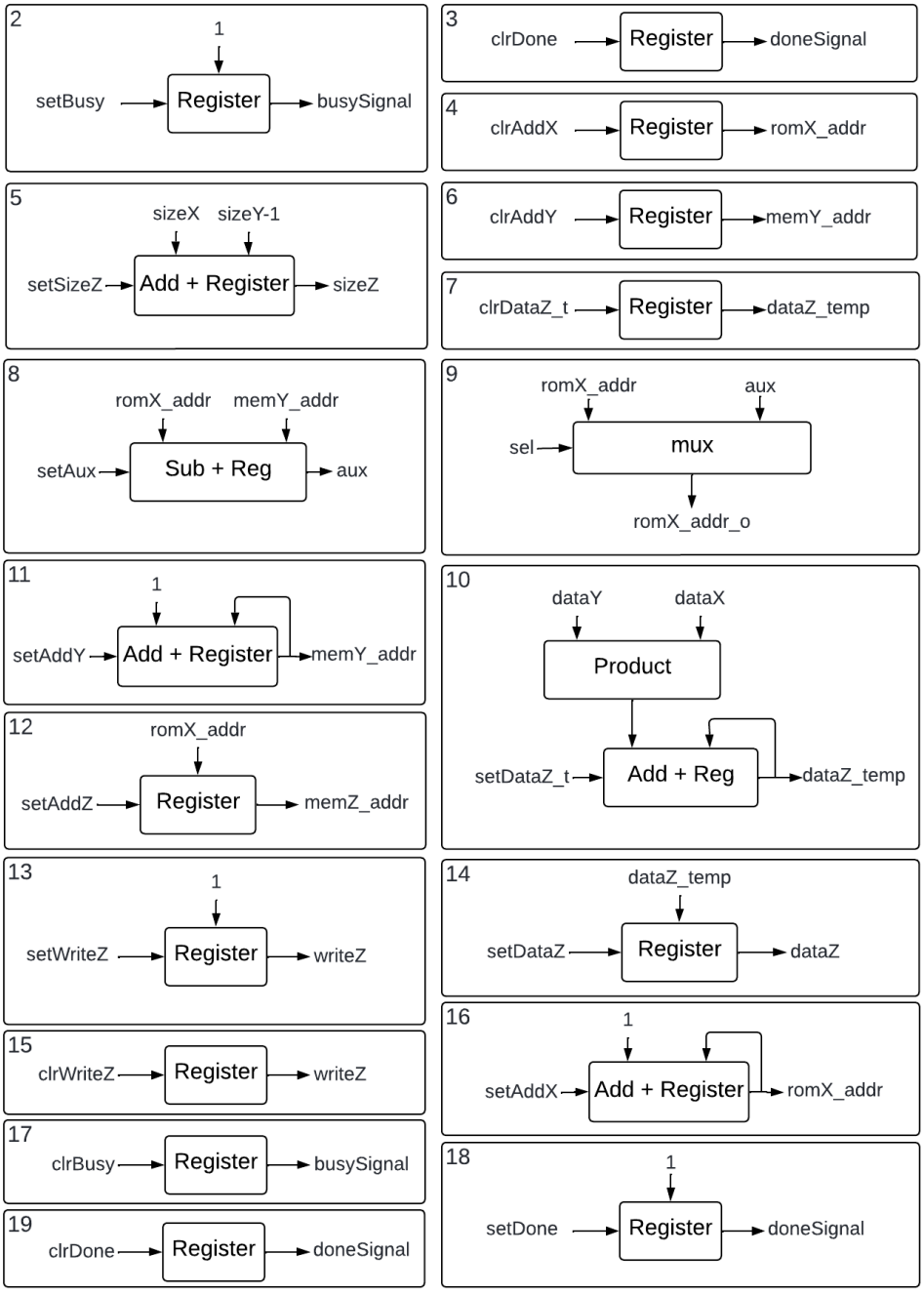
4) Diagrama ASM.



5) Diagrama de datapath y máquina de estados. (Explicar si combinaron estados, que bloques óptimos decidieron agregar, etc.

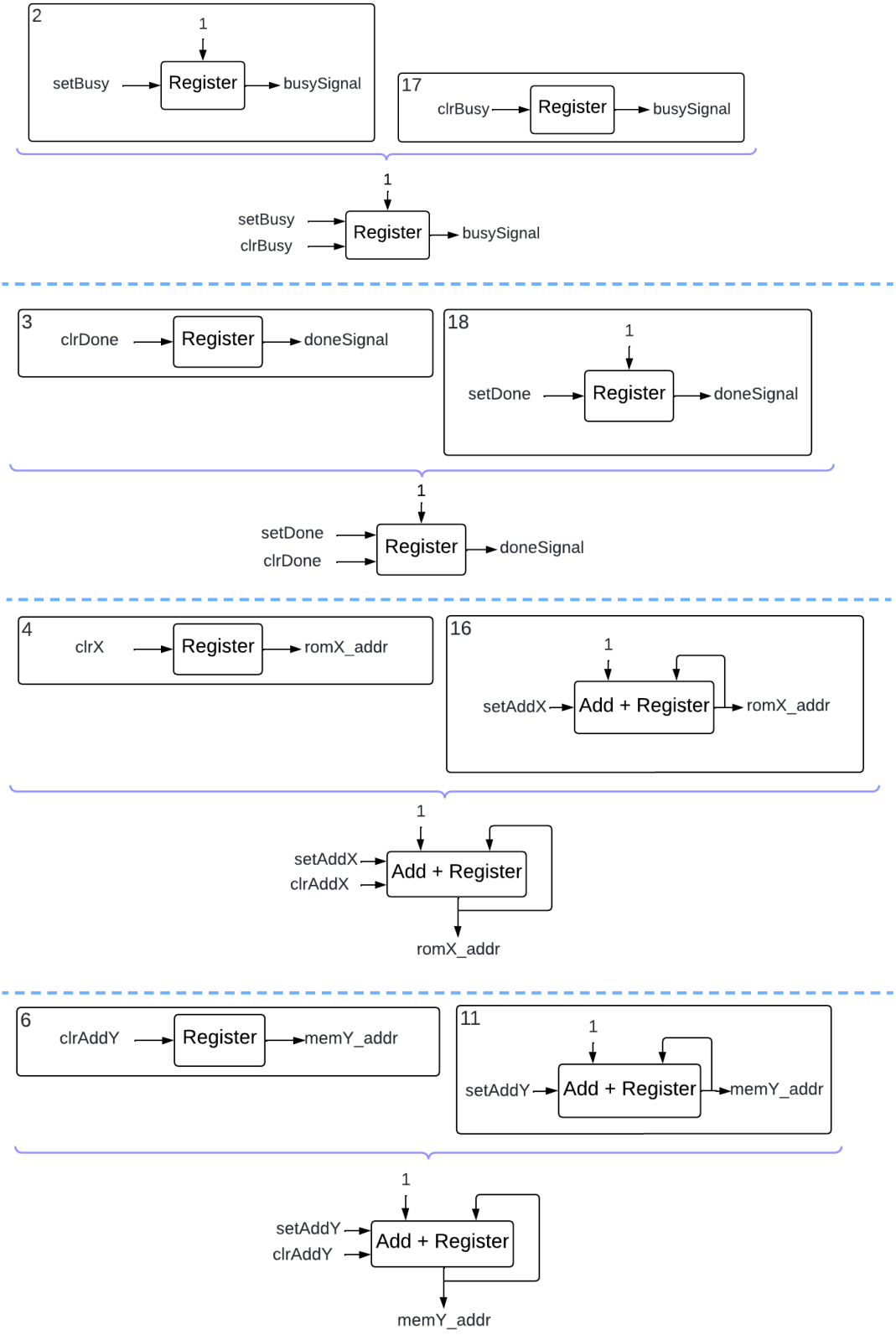
BLOQUES OPTIMOS

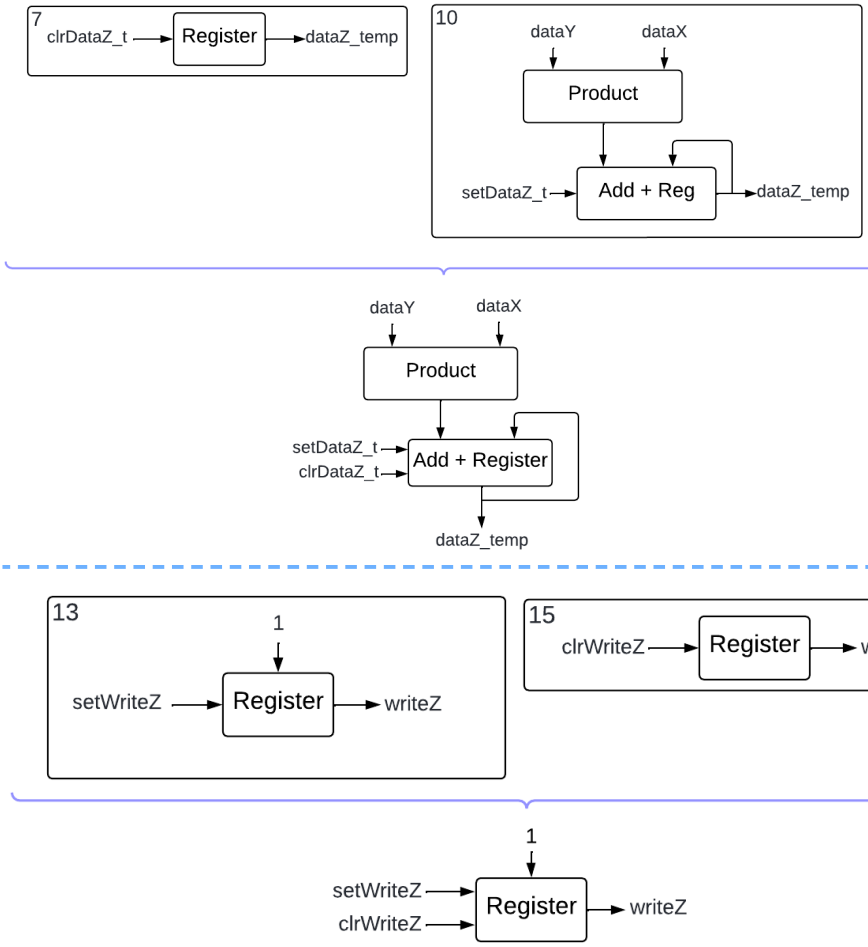
Se decidió crear los bloques optimo con módulos tales como registros, registros sumadores, registros restadores, multiplexores y multiplicadores



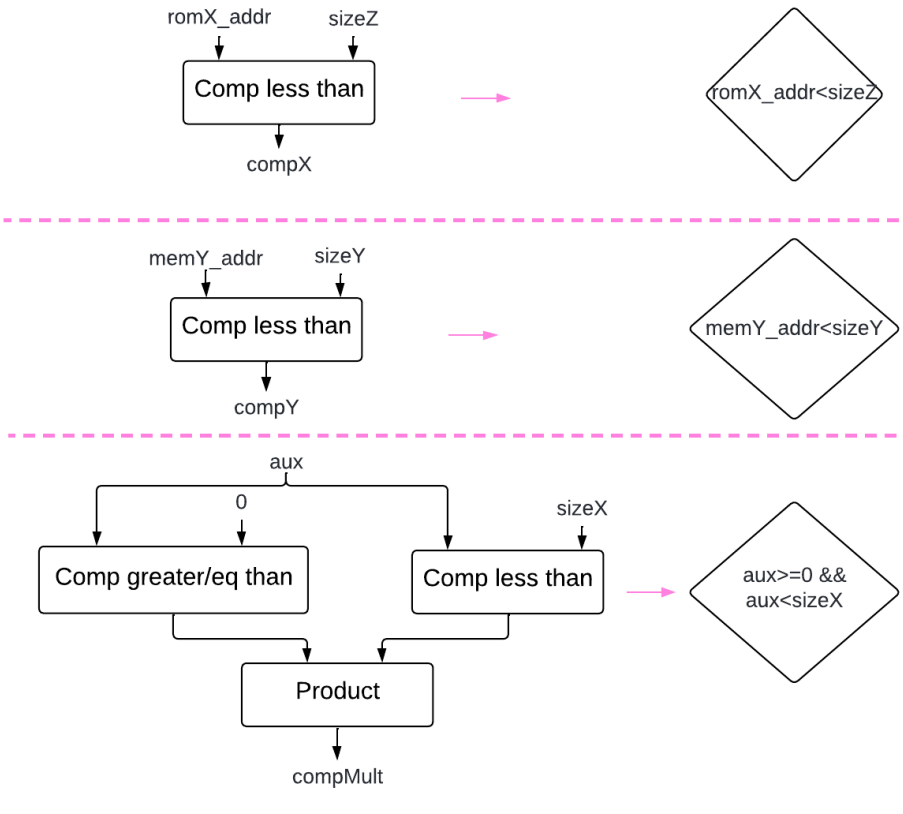
OPTIMIZACION DE BLOQUES OPTIMOS

Se encontraron y optimizaron aquellos bloques que tenían la misma señal de salida



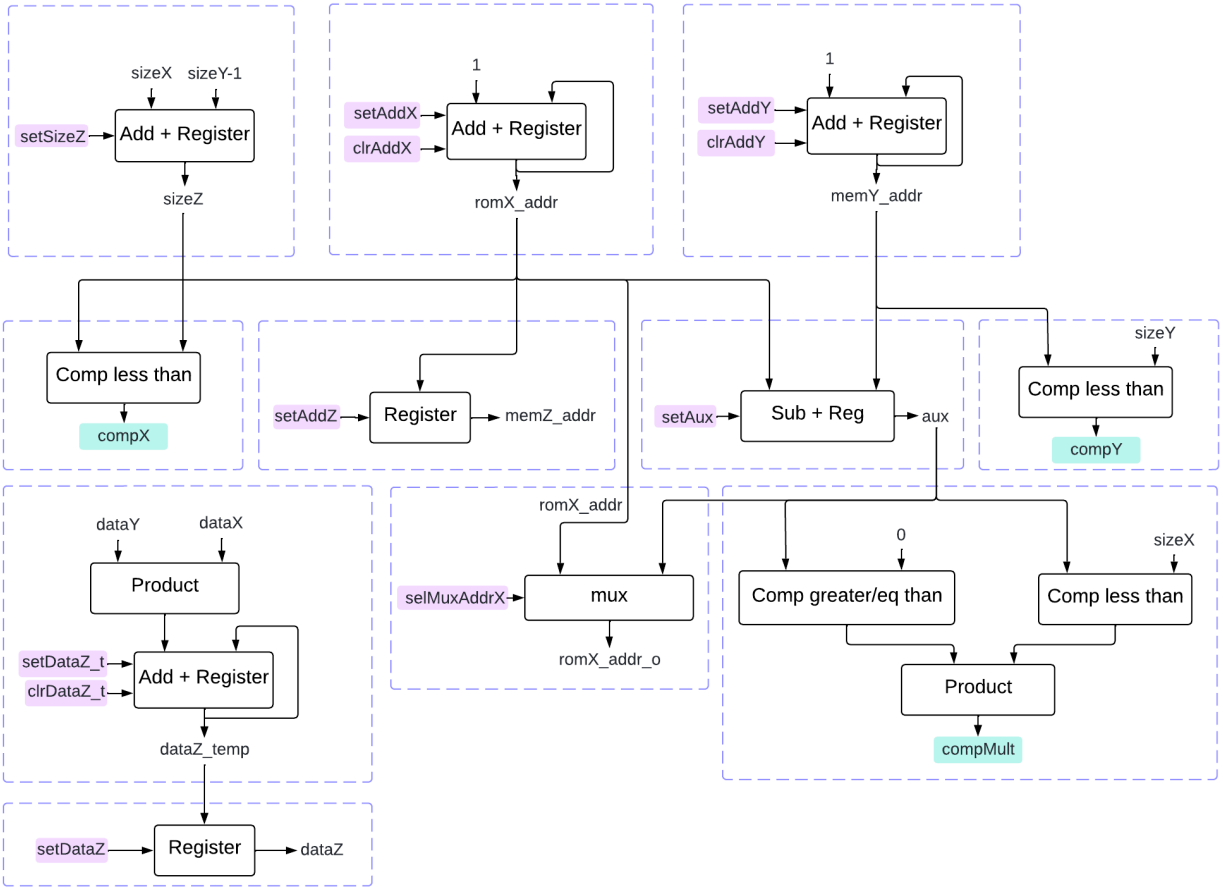


CONDICIONES EN BLOQUES OPTIMOS

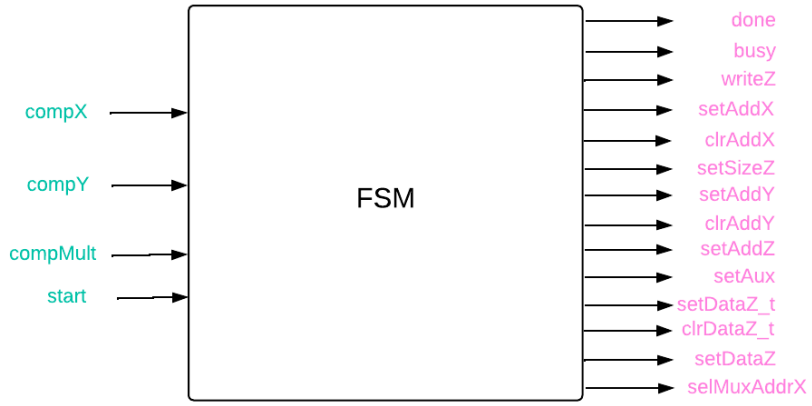


DATAPATH

Se combinaron los bloques óptimos y se creó el datapath que da una idea bastante clara de sus interacciones y dependencias



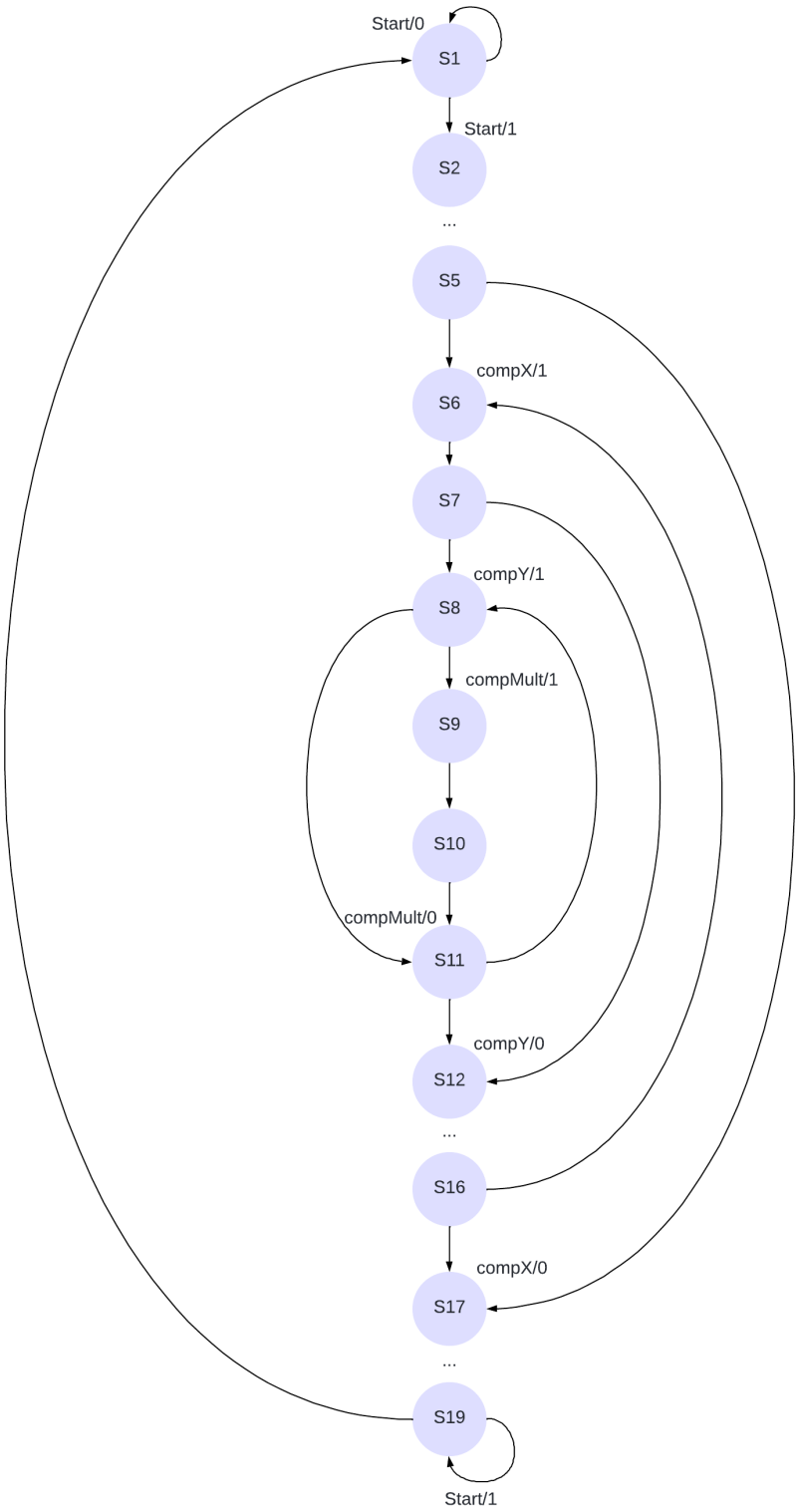
MÁQUINA DE ESTADOS (FSM)



CONTROL DE ESTADOS DE LA FSM

Control FSM		
State	Inputs/Value	Outputs/Value
1	Start/0 (S1) Start/0 (S19)	
2	Start/1 (S1)	setBusy/1
3		clrDone/1
4		clrAddX/1
5		setSizeZ/1
6	compX/1 (S5) compX/1 (S16)	clrAddY/1
7		clrDataZ_t/1
8	compY/1 (S7) compY/1 (S11)	setAux/1
9	compMult/1 (S8)	selMuxAddrX_o/1
10		setDataZ_t/1
11	(S10) compMult/0 (S8)	setAddY/1
12	compY/0 (S7) compY/0 (S11)	setAddZ/1
13		setWriteZ/1
14		setDataZ/1
15		clrWriteZ/1
16		setAddX/1
17	compX/0 (S5) compX/0 (S16)	clrBusy/1
18		setDone/1
19		clrDone/1

DIAGRAMA DE SECUENCIA DE ESTADOS DE LA FSM

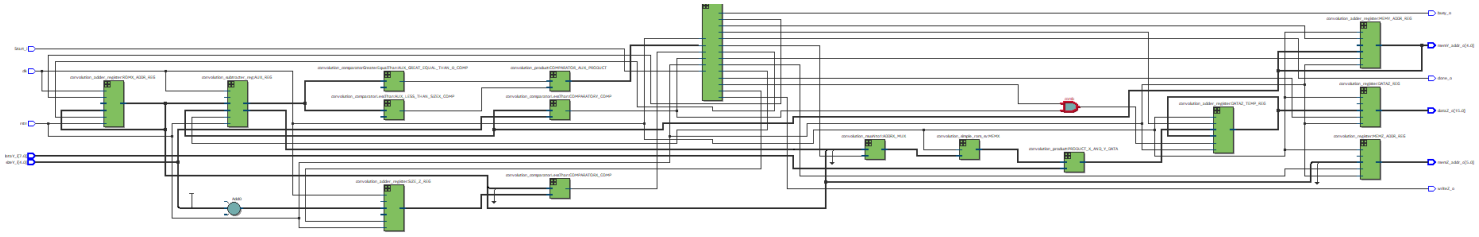


6) (Opcional) Alguna mejora al diseño.
No hay.

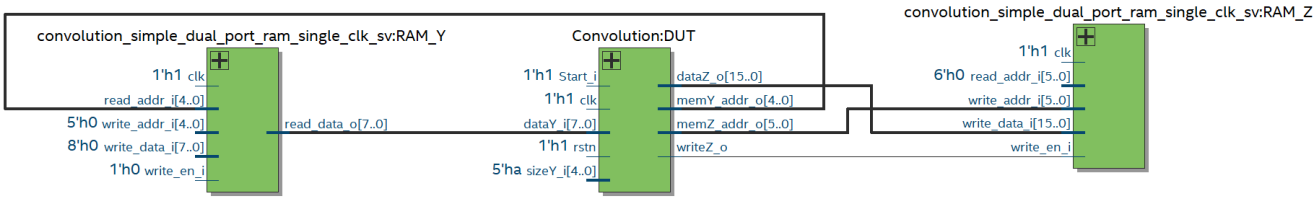
7) Resultados de simulación y síntesis en FPGA.

a) Esquemático Top Level generado por la herramienta.

Esquemático de Convolucionador



Esquemático de Test Bench del Convolucionador

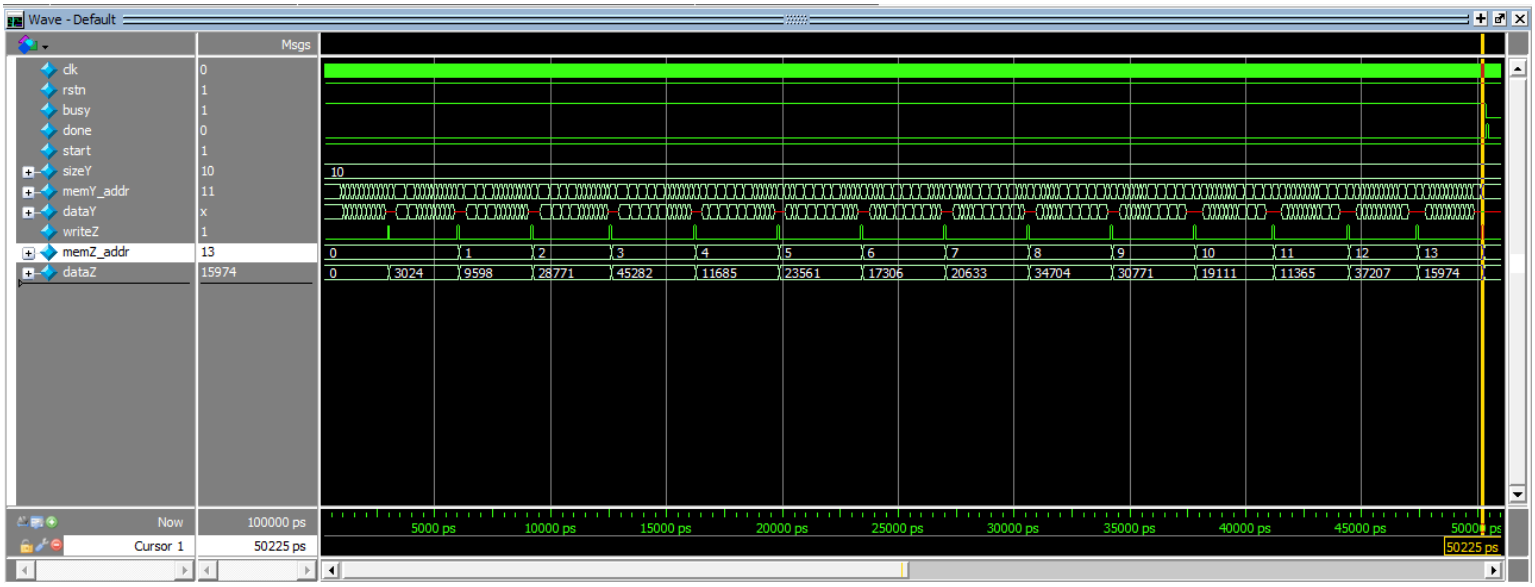


b) Escribir un programa que genere archivos para que puedan ser cargados en las memorias de entrada de su cama de prueba.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

/*CONSTANTES DEL ARCHIVO*/
#define TAM_NOMBRE_ARCHIVO 100
/// cambiar el nombre de archivo
#define NOMBRE_ARCHIVO "memX.txt"
/// cambiar ubicacion de archivo
#define UBICACION_ARCHIVO "../../Memorias/"

/*CONSTANTES DE LA MEMORIA*/
/// cambiar cantidad de registros
#define CANTIDAD_REG 5
/// cambiar el numero de bits que tendra la salida
#define CANTIDAD_BITS 8

int guarda_num(int numLigas [], FILE* archivo);
int* conv_a_bin(int n);
void conv_a_bin(int n, int num []);
void reiniciar(int num []);

int main()
{

    FILE* archivo = NULL;
    char nombreArchivo[TAM_NOMBRE_ARCHIVO] = {};
```

```
    int num [CANTIDAD_BITS] = {0};
    sprintf(nombreArchivo, "%s%s", UBICACION_ARCHIVO,
   NOMBRE_ARCHIVO);

    srand(time(NULL));
    archivo=fopen(nombreArchivo, "wb");

    printf("[");
    for(int i=0;i<CANTIDAD_REG; i++){
        conv_a_bin( (int) rand()%((int) pow(2,CANTIDAD_BITS)), num);
        guarda_num(num, archivo);
        reiniciar(num);
    }
    printf("]\n");

    fclose(archivo);
    return 0;
}

int guarda_num(int num [], FILE* archivo){
    int valido = 1;
    char car;
    char espacio='\n';
    if(archivo==NULL){
        perror("El archivo no esta abierto para guardar");
        return 0;
    }
    for(int j = CANTIDAD_BITS-1; j>=0; j-- ){      //imprime todos
   los elementos almacenados en el vector
        car = (num[j]==0)? '0' : '1';
        if(!fwrite(&(car), sizeof(char), 1, archivo))
           printf("\nNo se pudo guardar el dato\n");
    }
    fwrite(&(espacio), sizeof(char), 1, archivo);
    return valido;
}

void conv_a_bin(int n, int num []){
    int R;
    printf("%i ", n);
    for(int i=0; n!=0; i++ )
    {
        R  = n%2;
        n /= 2;
        num[i] = R;
    }
}

void reiniciar(int num []){
    for(int j = CANTIDAD_BITS-1; j>=0; j-- ){      //imprime todos
   los elementos almacenados en el vector
        num[j]=0;
    }
}
```

c) Waveform de la simulación del punto (f).



d) Área.

Top-level Entity Name	Convolution
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	40 / 32,070 (< 1 %)
Total registers	79
Total pins	46 / 457 (10 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	2 / 87 (2 %)
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 %)

e) Máxima Frecuencia.

	Fmax	Restricted Fmax	Clock Name	Note
1	152.28 MHz	152.28 MHz	clk	

f) Número de ciclos de reloj para hacer convolución a partir de que “start“ es puesto en nivel alto hasta que la señal de “Done“ es puesta en nivel alto. Para dos señales de entrada guardadas en memoria (aquí se deben cargar los archivos generados en el punto b) X e Y, una de 5 muestras y la otra de 10 muestras.

Teniendo en cuenta que todo el proceso duró un total de 50250 ps y que un ciclo de reloj toma 100 ps, se puede concluir que tomo 502 ciclos de reloj.