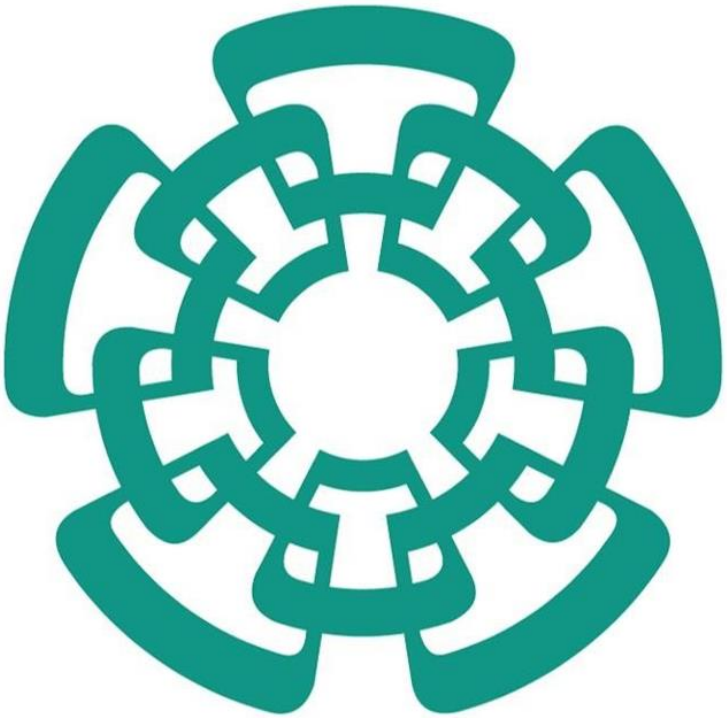


Cinvestav

Guadalajara

Proyect 1

Procesador de convolución (P1)

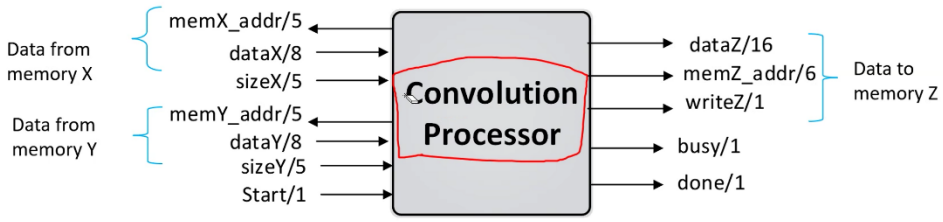
Fernando Leonel Gallo Hinojosa

Prof.: Vidkar Anibal Delgado Gallardo

Fecha: 12 de mayo del 2024.

6. Project – P1

- Implement a convolution coprocessor using the top-down approach.
- Inputs: Data stored in memory X and memory Y. Size of the signals stored in each memory. Start signal.
- Outputs: Done signal (One-shot). Busy signal. Store result in memory Z.



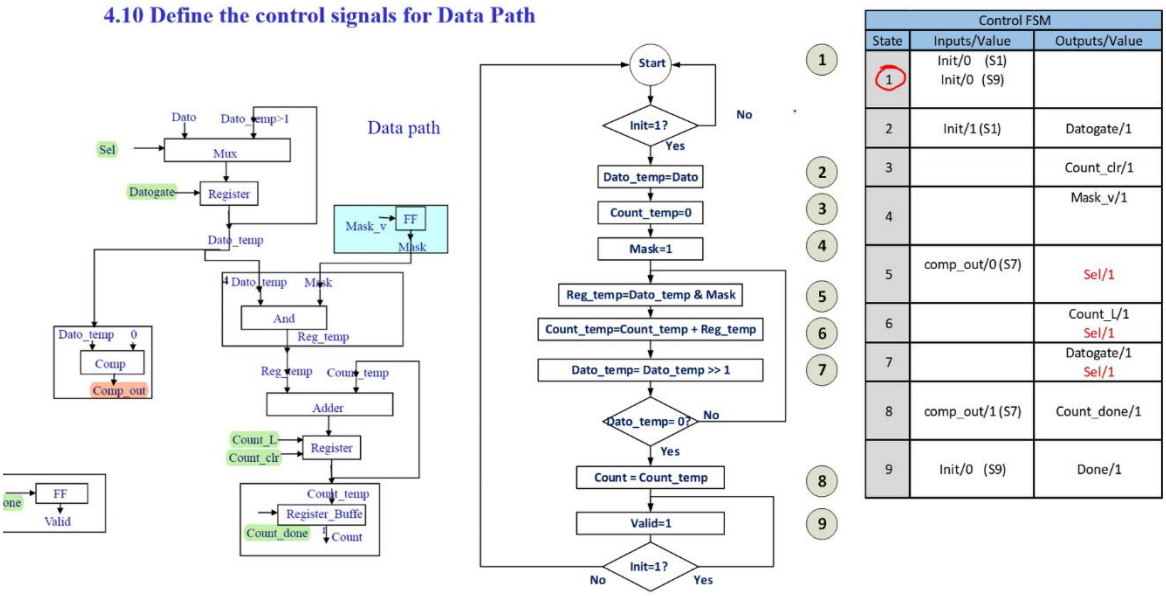
P1:

Implementar un coprocesador de convolución utilizando el enfoque descendente.

Entradas: Datos almacenados en la memoria X y en la memoria Y. Tamaño de las señales almacenadas en cada memoria. Señal de inicio.

Salidas: Señal de Hecho (One-shot). Señal de ocupado. Almacenar resultado en memoria Z.

1. Almacenamiento de resultados: Asegúrate de que los resultados parciales de la convolución se almacenen adecuadamente en la memoria Z.
2. Gestión de errores y excepciones: Implementa mecanismos para manejar errores como desbordamiento de memoria, errores de acceso, etc.



considero que este es el modelo en el que debo estructurar la idea pero aun falta mas análisis.

1. Comprensión del problema:

- El programa ConvolutionCoprocessor realiza una convolución digital entre dos señales de entrada (X e Y) y almacena el resultado en la memoria Z.
- El tamaño de las señales X e Y se define por los parámetros sizeX y sizeY, respectivamente.
- El programa se activa mediante la señal start y termina cuando se genera la señal done.

2. Especificación de señales y relaciones entrada-salida:

Entradas:

- clk: Reloj del sistema.
- reset: Señal de reinicio.
- start: Señal de inicio de la convolución.
- memX_addr: Dirección de memoria para la señal X.
- dataX: Datos de la señal X.
- sizeX: Tamaño de la señal X.
- memY_addr: Dirección de memoria para la señal Y.
- dataY: Datos de la señal Y.
- sizeY: Tamaño de la señal Y.

Salidas:

- done: Señal de finalización de la convolución (one-shot).
- busy: Señal de ocupado del coprocesador.
- memZ_addr: Dirección de memoria para el resultado de la convolución (Z).
- dataZ: Datos del resultado de la convolución (Z).
- writeZ: Señal de escritura en la memoria Z.

3. Definición del sistema de caja negra:

- El sistema de caja negra recibe como entrada las señales X, Y, sizeX, sizeY, memX_addr, memY_addr y start.
- Procesa las entradas realizando una convolución digital entre X e Y.
- Almacena el resultado de la convolución en la memoria Z.
- Genera las señales de salida done, busy, memZ_addr, dataZ y writeZ.

4. Definición del algoritmo y pseudocódigo:

El algoritmo del programa ConvolutionCoprocessor puede dividirse en las siguientes etapas:

1. **Inicialización:**

- Se inicializan las variables row, col, size, accumulator y convolving.
- Se establece el estado del FSM a INIT.

2. **Convolución:**

- Se realiza la multiplicación punto a punto entre los elementos de las señales X e Y.
- El resultado de la multiplicación se acumula en la variable accumulator.
- Se actualizan las variables row y col para recorrer las señales X e Y.
- El estado del FSM se actualiza a CONVOLVING o WRITE_Z según las condiciones.

3. **Escritura en memoria Z:**

- Se escribe el resultado de la convolución (accumulator) en la memoria Z.
- Se actualiza la dirección de memoria memZ_addr.
- Se genera la señal writeZ.
- El estado del FSM se actualiza a DONE.

4. **Hecho:**

- Se mantiene el estado del FSM en DONE hasta que se recibe una señal de reinicio (reset).

5. **Descripción del algoritmo en diagrama ASM:**

En base al código creado anteriormente obtenemos la siguiente máquina de estado de algoritmo, este cuenta con un total de 25 estados y 4 decisiones.

```
+---------------------+
| Bloque de entrada   |
+---------------------+
1. Inicialización de señales
   1.1 start_signal = start
   1.2 memoria_x_addr = primer elemento de memoria_x
   1.3 dataX = segundo elemento de memoria_x
   1.4 sizeX = tercer elemento de memoria_x
   1.5 memoria_y_addr = primer elemento de memoria_y
   1.6 dataY = segundo elemento de memoria_y
   1.7 sizeY = tercer elemento de memoria_y

2. Leer los datos de memoria X e Y
   2.1 llamar a leer_datos_memoria con memoria_x_addr y dataX
   2.2 llamar a leer_datos_memoria con memoria_y_addr y dataY

3. Realizar la convolución
   3.1 llamar a bloque_convolucion con dataX y dataY

+-----------------------+
| Bloque de convolución |
+-----------------------+
4. Realizar la convolución entre los datos de memoria X e Y
   4.1 llamar a convolucion con dataX y dataY

+--------------------+
| Bloque de salida |
+--------------------+
5. Inicialización de señales
   5.1 writeZ = 1
   5.2 dataZ = resultado_convolucion
   5.3 memZ_addr = 0
   5.4 sizeZ = sizeX + sizeY
   5.5 busy = 0
   5.6 done = 1

6. Escribir los resultados en memoria Z
   6.1 llamar a escribir_datos_memoria con writeZ, dataZ y memZ_addr

7. Marcar que la operación está terminada
```

7.1 llamar a marcar_operacion_terminada con done

+--------------------------------+
| Funciones auxiliares |
+--------------------------------+

8. Lógica para leer los datos de memoria
   8.1 Implementar la lógica para leer los datos de memoria

9. Lógica para realizar la convolución
   9.1 Implementar la lógica para realizar la convolución

10. Lógica para escribir los datos en memoria
    10.1 Implementar la lógica para escribir los datos en memoria

11. Lógica para marcar que la operación está terminada
    11.1 Implementar la lógica para marcar que la operación está terminada

Pseudocódigo:

```
/* Bloque de entrada */
// 1. Inicialización de señales
Inicio:
    // 1.1. Señal de inicio
    start_signal = start

    // 1.2. Dirección de memoria X (5 bits)
    memoria_x_addr = memoria_x[0]

    // 1.3. Datos de memoria X (8 bits)
    dataX = memoria_x[1]

    // 1.4. Tamaño total de memoria X (5 bits)
    sizeX = memoria_x[2]

    // 1.5. Dirección de memoria Y (5 bits)
    memoria_y_addr = memoria_y[0]

    // 1.6. Datos de memoria Y (8 bits)
    dataY = memoria_y[1]

    // 1.7. Tamaño total de memoria Y (5 bits)
    sizeY = memoria_y[2]

// 2. Leer los datos de memoria X e Y
```

```
Leer_datos_memoria(memoria_x_addr, dataX)
Leer_datos_memoria(memoria_y_addr, dataY)

// 3. Realizar la convolución
Bloque_convolucion(dataX, dataY)

/* Bloque de convolución */
// 4. Realizar la convolución entre los datos de memoria X e Y
Convolucion(dataX, dataY)

// Bloque de salida
/* Bloque de salida */
// 5. Inicialización de señales
// 5.1. Control de escritura en memoria Z (1 bit)
writeZ = 1

// 5.2. Datos a escribir en memoria Z (16 bits)
dataZ = resultado_convolucion

// 5.3. Dirección de memoria Z (6 bits)
memZ_addr = 0

// 5.4. Tamaño total de memoria Z (sizeX + sizeY bits)
sizeZ = sizeX + sizeY

// 5.5. Señal de ocupado (1 bit)
busy = 0

// 5.6. Señal de finalización (1 bit)
done = 1

// 6. Escribir los resultados en memoria Z
Escribir_datos_memoria(writeZ, dataZ, memZ_addr)

// 7. Marcar que la operación está terminada
Marcar_operacion_terminada(done)

/* Función para leer los datos de memoria */
// 8. Lógica para leer los datos de memoria
Leer_datos_memoria(memoria_addr, data)

/* Función para convolucionar los datos de X e Y */
// 9. Lógica para realizar la convolución
```

Convolucion(dataX, dataY)

/* Función para escribir los datos en memoria */
// 10. Lógica para escribir los datos en memoria
Escribir_datos_memoria(writeZ, dataZ, memZ_addr)

/* Función para marcar que la operación está terminada */
// 11. Lógica para marcar que la operación está terminada
Marcar_operacion_terminada(done)

Bloque de entrada:

- Inicialización de señales necesarias.

Asignación de direcciones y datos de entrada para memoria X y memoria Y.
Determinación del tamaño total de memorias X e Y.

- Leer los datos de memoria X e Y:

Invocación de la función Leer_datos_memoria para obtener los datos de memoria X e Y usando las direcciones correspondientes.

- Realizar la convolución:

Invocación de la función Bloque_convolucion para llevar a cabo la operación de convolución entre los datos de memoria X e Y.

- Bloque de convolución:

Ejecución de la lógica específica para realizar la convolución entre los datos de memoria X e Y.

- Bloque de salida:

Inicialización de señales para la salida de los resultados.
Asignación de datos resultantes de la convolución para escribir en memoria Z.
Determinación de la dirección y tamaño total de memoria Z.
Establecimiento de señales de estado para indicar ocupación y finalización de la operación.

- Escribir los resultados en memoria Z:

Invocación de la función Escribir_datos_memoria para escribir los resultados de la convolución en memoria Z.

- Marcar que la operación está terminada:

Invocación de la función Marcar_operacion_terminada para indicar la finalización de la operación.

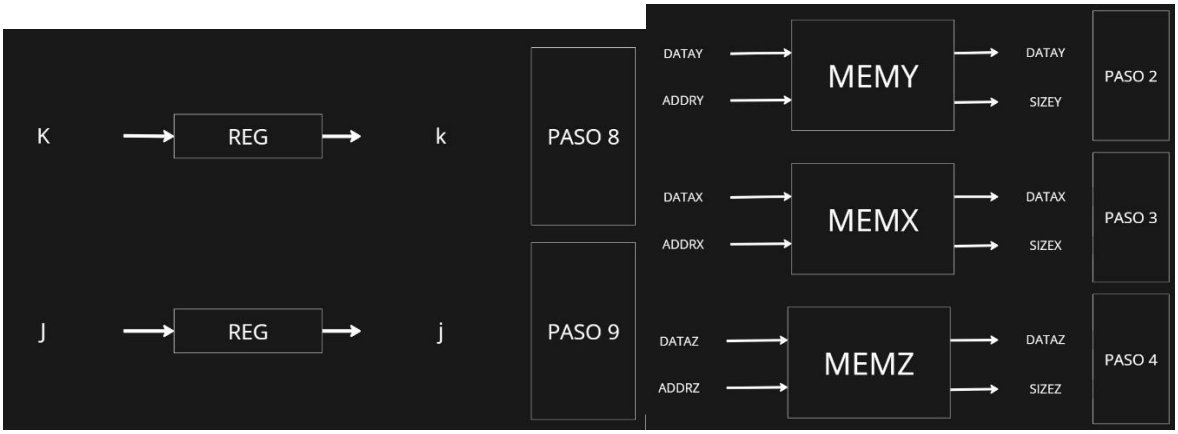
- Funciones auxiliares:

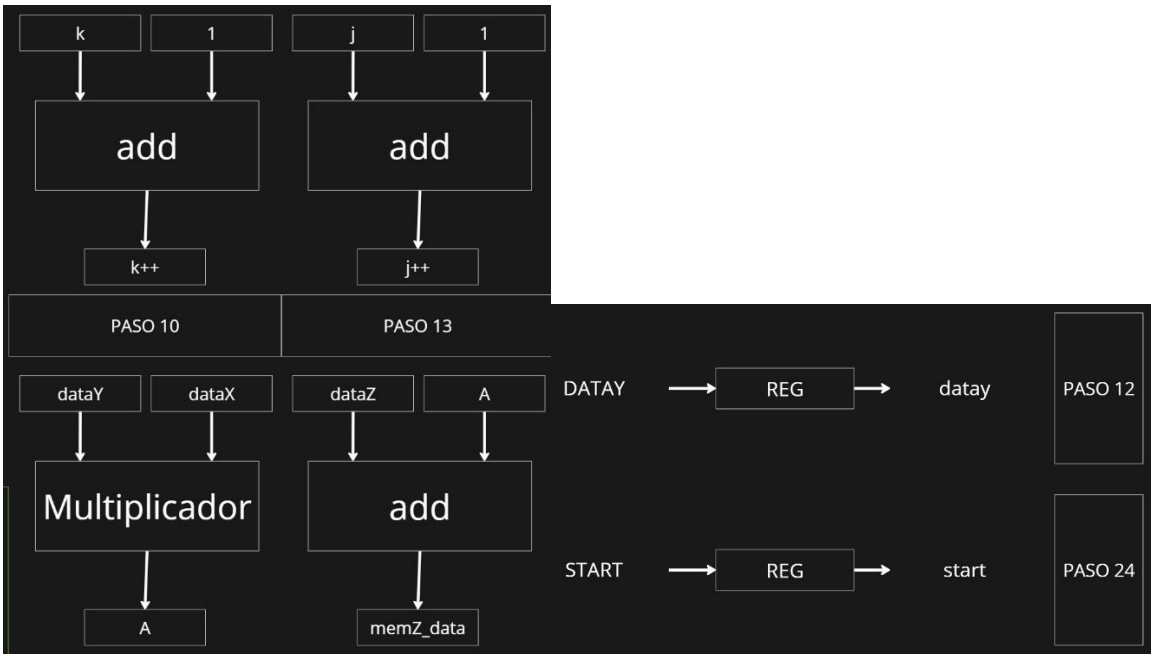
Definición de las funciones Leer_datos_memoria, Convolucion, Escribir_datos_memoria y Marcar_operacion_terminada para realizar tareas específicas como lectura de datos de memoria, convolución y escritura de resultados en memoria.

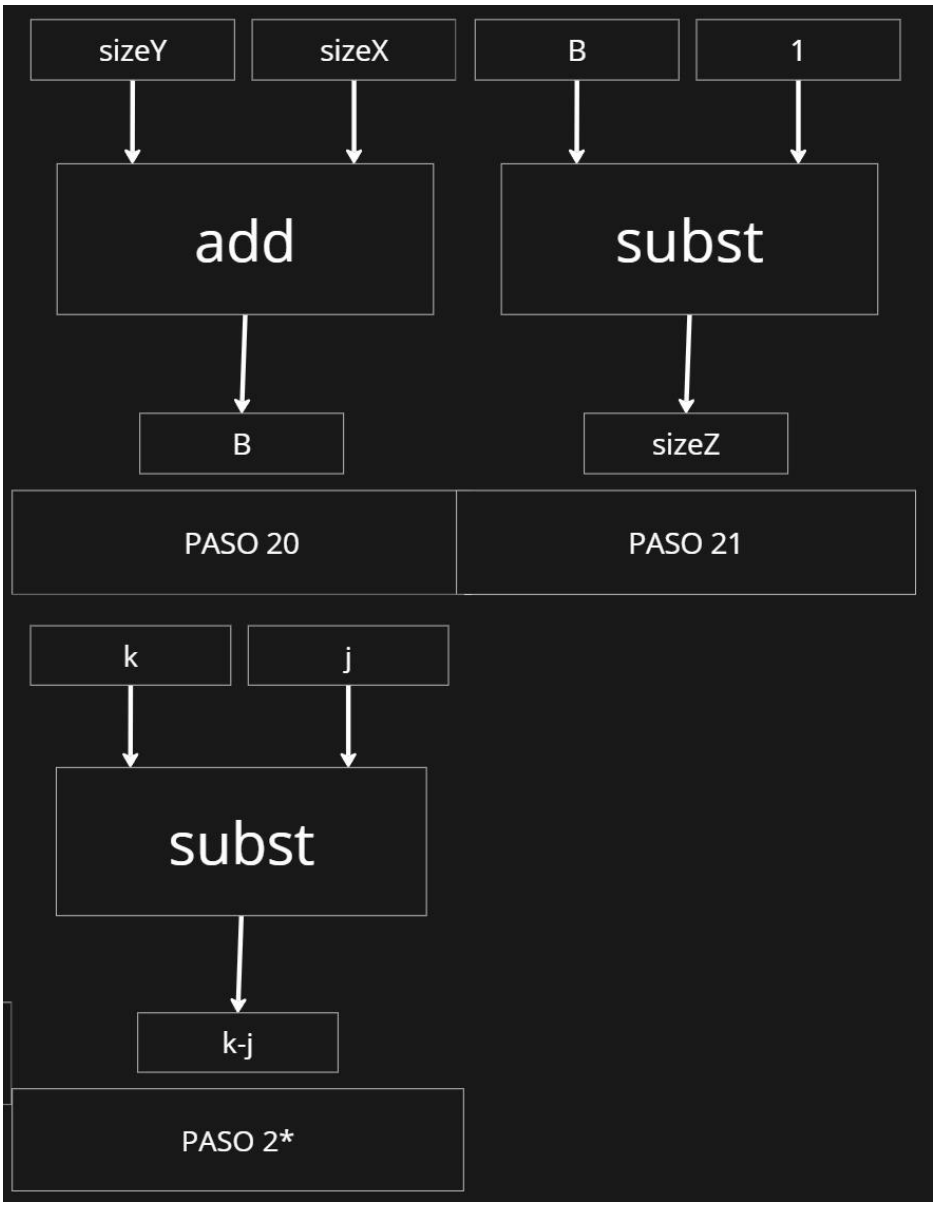
```
// Bibliotecas
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUMEROS_A_GENERAR 5

// Función para convolucionar los datos de X e Y
void convolucion(int *dataX, int *dataY, int *resultado_convolucion) {
  for (int i = 0; i < NUMEROS_A_GENERAR; i++) {
    resultado_convolucion[i] = 0; // Inicializa el resultado de la convolución en 0
    for (int j = 0; j <= i; j++) {
      // Multiplica los elementos de dataX y dataY y los suma al resultado de la convolución
      resultado_convolucion[i] += dataX[j] * dataY[i - j];
    }
  }
}

int main() {
  FILE *archivo;
  int dataX[NUMEROS_A_GENERAR];
  int dataY[NUMEROS_A_GENERAR];
  int resultado_convolucion[NUMEROS_A_GENERAR];

  // Abre el archivo para escribir los datos de entrada
  archivo = fopen("datos_de_entrada.txt", "w");
  if (archivo == NULL) {
    printf("Error al abrir el archivo para escribir datos de entrada.");
    return 1;
  }

  // Inicializa la semilla para la generación de números aleatorios
  srand(time(NULL));

  // Genera y escribe los números aleatorios en el archivo en formato hexadecimal
  for (int i = 0; i < NUMEROS_A_GENERAR; i++) {
    dataX[i] = rand() % 16; // Números aleatorios entre 0 y 15
    dataY[i] = rand() % 16;
    fprintf(archivo, "%X %X\n", dataX[i], dataY[i]); // Usa "%X" para formato hexadecimal
```

```
}

// Cierra el archivo de datos de entrada
fclose(archivo);

// Abre el archivo para escribir los resultados de la convolución
archivo = fopen("resultados_de_convolucion.txt", "w");
if (archivo == NULL) {
   printf("Error al abrir el archivo para escribir los resultados de convolución.");
   return 1;
}

// Realiza la convolución
convolucion(dataX, dataY, resultado_convolucion);

// Escribe los resultados de la convolución en el archivo
for (int i = 0; i < NUMEROS_A_GENERAR; i++) {
   fprintf(archivo, "%X\n", resultado_convolucion[i]);
}

// Cierra el archivo de resultados de convolución
fclose(archivo);

printf("Datos generados y convolucionados correctamente.\n");-
return 0;}
```

En base al diagrama ASM podemos encontrar los componentes principales que conformaran nuestro coprocesador, las cuales constan de memorias, Flip-Flops, registros, bloques de suma, resta y multiplicación.




k
1
j
1
add
add
k++
j++
PASO 10
PASO 13
dataY
dataX
dataZ
A
Multiplicador
add
A
memZ_data
DATAY
REG
datay
PASO 12
START
REG
start
PASO 24


sizeY
sizeX
B
1
add
subst
B
sizeZ
PASO 20
PASO 21
k
j
subst
k-j
PASO 2*