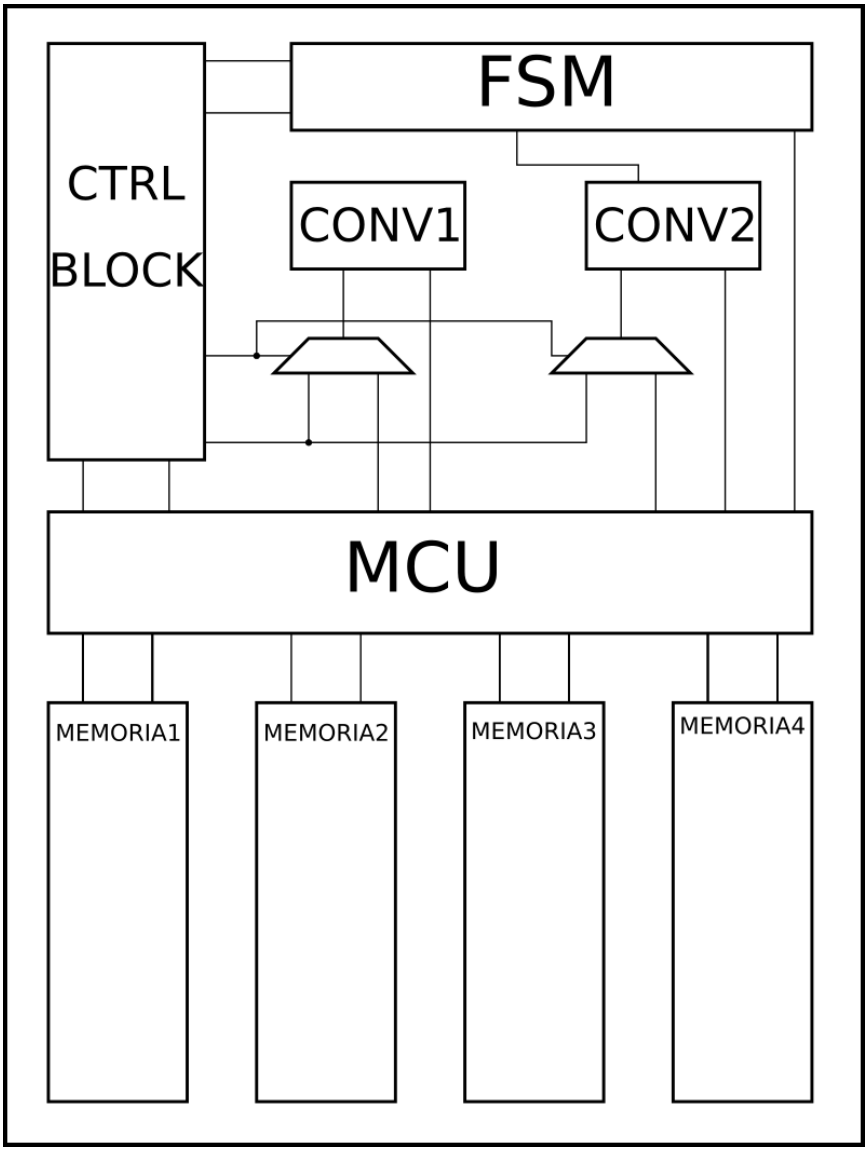
Contents

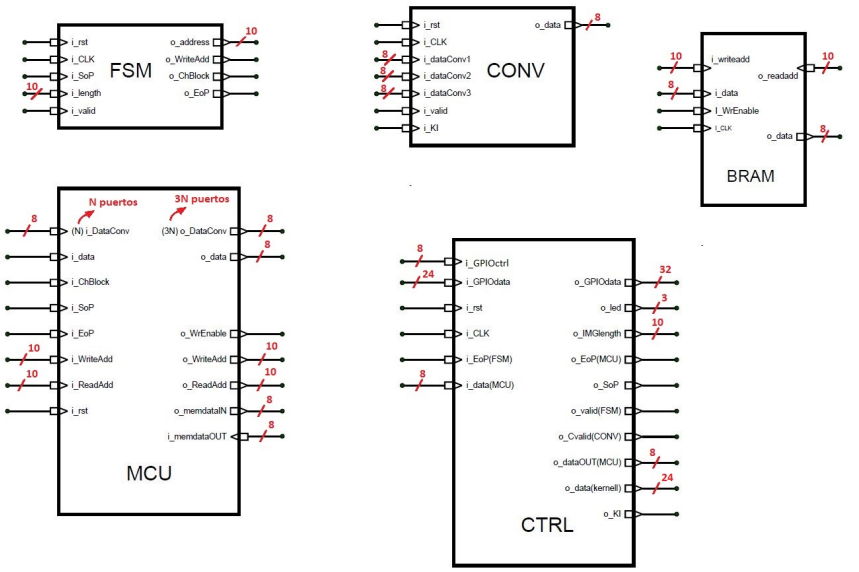
1 Estructura

En esquema simplificado de la estructura se muestra a continuacion



Estructura para un caso con 2 modulos de convolucion

El siguiente esquema muestra con mas detalles los puertos de cada uno de los bloques



A continuacion se hace una descripcion mas detallada de cada uno de los bloques

1.1 Modulo de convolucion

El modulo de convolucion se encarga de hacer un producto interno de Frobenius entre una matriz precargada llamada Kernel (filtro), y un sector de una imagen. El sector tiene la misma dimension que el kernel.

1.1.1 Estructura interna

El modulo tiene dos arreglos de registros, en uno esta cargado el kernel, el cual solo se ve modificado durante la configuracion. El otro arreglo tiene cargado la porcion de la imagen. Una vez entregado un resultado, el arreglo que contiene la imagen desplaza sus filas hacia arriba, perdiendo la fila superior y cargando como fila inferior los valores que tiene en los puertos de entrada. La salida de la operacion aumenta el rango de bits de trabajo, por lo que se debe reajustar el rango antes de ponerlo a la salida.

1. I/O **Input**
    - InputN: 3 puertos de $\$BITS_{IMAGEN}$ bits que leen las filas de la imagen cargada en memoria, o las filas del kernel, dependiendo de los valores que tome el puerto **K/I**

- K/I: Puerto de un bit que determina si los valores leidos en **InputN** se deben cargar en el kernel o en el registro de la imagen. Cualquier modificacion dependera del estado leido en **valid**.
  - Si se escribe un 1 en el puerto, el kernel sera modificado
  - Si se escribe un 0 en el puerto, la imagen sera modificada

  Durante un uso normal (fuera de la etapa de configuracion), deberia haber un 0 en el puerto
- Valid: Habilita la modificacion de los registros si hay un estado alto en el puerto
  - Si se escribe un 1, los registros son modificados con cada flanco de clock.
  - Si se escribe un 0, los registros permanecen intactos.
- Reset
- Clk

OutPut

- Out: Salida de $\$BITS_{IMAGEN}$ que representa el resultado de las operaciones

1.2 Modulo de control

1. I/O **Input**

   - $GPIO_{CTRLIn}$: Puerto de 8 bits que recibe las instrucciones de control.
   - $GPIO_{DataIn}$: Puerto de 24 bits que recibe los datos desde el exterior
   - Rst
   - Clk
   - EoP: Indica que se ha finalizado la etapa de procesamiento, se debe salir del estado de **RUN**
   - $Data_{In}$: Puerto de 3*[8] bits que recibe los datos almacenados en memoria

   OutPut

   - $GPIO_{Out}$: Puerto de 32 bits utilizado para enviar informacion al exterior

- Estado: Puerto de 3 bits que indica en que estado se esta
- SoP: Indica que se ha pasado al estado de **RUN** y se debe comenzar el procesamiento de la informacion
- EoP: Indica que se ha pasado al estado de **OUT**
- Valid: Se pone en estado alto durante un ciclo cuando detecta un flanco de subida del **bit Valid** de la instruccion de control presente en $\mathbf{GPIO_{CTRLIn}}$
- C_{Valid}: Se pone en estado alto durante un ciclo cuando detecta un flanco de subida del **bit Valid** de la instruccion de control presente en $\mathbf{GPIO_{CTRLIn}}$ *unicamente durante la etapa de configuracion del kernel en la estapa de LOAD*
- Length: Puerto de [10] bits que tiene a su salida la altura de la imagen con la que se esta trabajando
- $Data_{Out}$: Puerto de [24] bits que reflejan los datos recibidos desde $\mathbf{GPIO_{DataIn}}$ durante la etapa de carga de imagen en el estado de **LOAD**
- Kernel: Puerto de [24] bits que representan una fila del kernel (filtro) que se desea cargar, la configuracion del kernel debe ser habilitada mediante el puerto **K/I** y sincronizada utilizando el puerto $\mathbf{C_{Valid}}$
- K/I: Indica si los datos entregados desde la memoria deben ser cargados a los registros del Kernel o el sector de la imagen.

1.3 FSM

1. I/O **Input**

   - Clk
   - Rst
   - SoP: Start of process, indica a la FSM que se ha terminado con la etapa de carga de datos y se debe comenzar con el procesamiento, al dar esa orden, la **FSM** pasa a tomar el control y lo retorna una vez terminado el procesamiento, a travez del puerto **EoP**
     - Un 0 indica que no se esta en la etapa de procesamiento
     - Un 1 indica que se debe dar inicio a la etapa de procesamiento
   - ImgLength: Puerto de [10] bits, donde se indica la altura de la imagen (en pixels), la **FSM** procesara los datos en memoria hasta alcanzar dicho tamaño

- Valid: Indica cuando se deben incrementar las direcciones de memoria ($\mathbf{R_{Addr}}$ y $\mathbf{W_{Addr}}$) con cada ciclo de clock
  - Un 1 habilita el incremento
  - Un 0 mantiene fija la direccion

OutPut

- R_{Addr}: [10] bits, es la direccion a ser leida en el proximo flanco de clock
- W_{Addr}: [10] bits, es la direccion donde sera escrita la salida del modulo **CONV**
- EoP: Indica que se ha finalizado la etapa de procesamiento poniendo la salida en alto
- Ch_{block}: Usado en los estados de **LOAD** y **OUT**, indican al **MCU** que se ha finalizado la lectura de un bloque y se debe pasar al siguiente.
  - La salida esta en alto cuando se debe cambiar de bloque

1.4 Memoria

1. I/O **Input**

- W_{Addr}: Direccion donde se debe escribir el dato presente en $\mathbf{Data_{In}}$
- R_{Addr}: Direccion que indica el dato a ser leido a traves del puerto $\mathbf{Data_{Out}}$
- Clk
- WE: Write Enable, habilita la escritura de los datos presentes en $\mathbf{Data_{In}}$ en la memoria
  - Un 1 habilita la escritura
  - Un 0 no permite la escritura
- $Data_{In}$: Puerto de [8] bits, los datos presentes en este puerto seran escritos en la direccion inidicada por $\mathbf{W_{Addr}}$ si **WE** esta habilitado

Output

- $Data_{Out}$: Puerto de [8] bits, la salida representa los datos de la direccion inidicada por $\mathbf{R_{Addr}}$ en el ciclo anterior

1.5 MCU

Es el modulo encargado de la administracion de los bloques de memoria, hace de interfaz entre las memorias y el resto de los bloques. Se encarga del routeo de la informacion

1. I/O **Input _** CLK

   - Rst
   - $i_{DataConvN}$: Puerto de [8] bits donde se escribe la salida del N-esimo modulo **CONV**
   - $Data_{In}$: Puerto de 3*[8] bits donde se escriben los valores de la imagen en la etapa de **LOAD**
   - Ch_{Block}: Utilizado durante los estados de **LOAD** y **OUT**, indica que se ha terminado de leer un bloque y se debe pasar al siguiente
     - Un estado alto indica el paso al siguiente bloque
   - SoP: Start of process, indica que se ha entrado al estado de **RUN**, se entregan los datos apuntados por $\mathbf{R_{Addr}}$ en los puertos $\mathbf{O_{DataConvNM}}$ y se escriben los datos presentes en los puertos $\mathbf{I_{DataConvN}}$ en las direcciones indicadas por $\mathbf{W_{Addr}}$
     - Un estado alto indica el paso al estado **RUN**
   - R_{Addr}: [10]bits. Indica la direccion del dato a leer, si se esta en el estado de **RUN**, los datos leidos seran entregados por los puertos $\mathbf{O_{DataConvNM}}$, caso contrario seran entregados al puerto $\mathbf{Data_{Out}}$
   - W_{Addr}: [10]bits. Indica la direccion de memoria donde escribir los datos, si se esta en el estado de **RUN**, los datos escritos seran los que se encuentran en el puerto $\mathbf{i_{DataConvN}}$, caso contrario seran los del puerto $\mathbf{Data_{In}}$
   - EoP: Indica que se ha finalizado la etapa de procesamiento, es decir se sale del estado de **RUN** y se pasa al estado **OUT**, donde el **MCU** esta listo para entregar los datos de memoria a travez del puerto $\mathbf{Data_{Out}}$
     - Un 1 indica que se esta en el estado **OUT**
     - Un 0 indica que se esta en cualquier otro estado
       SoP y **EoP** nunca deberian estar en 1 al mismo tiempo
   - $MemData_{InN}$: Puerto de [8] bits que recibe los datos de la N-esima memoria que se encuentran en la direccion indicada por $\mathbf{R_{Addr}}$

Output

- $\underline{O_{DataConvNM}}$: Puerto de [8] bits que tiene en su salida el dato que para el **InputM** del N-esimo modulo **CONV**
- $\underline{WEN}$: Write Enable para la N-esima memoria
- $\underline{R_{AddrN}}$: Puerto de [10] bits que indica la direccion de la N-esima memoria que se quiere leer
- $\underline{W_{AddrN}}$: Puerto de [10] bits que indica la direccion de la N-esima memoria donde se quiere escribir el dato que se encuentra en $\mathbf{MemData_{Out}}$ si **WE** lo permite
- $\underline{Data_{Out}}$: Puerto de 3*[8] bits donde se tiene el valor ubicado en la direccion indicada por $\mathbf{R_{Addr}}$ de 3 memorias

1.5.1 Algoritmo de routeo de la informacion

(a) Introducción Este algoritmo se desarrolló con el fin de poder realizar una implementación eficiente en lo que refiere a economía de recursos, trabajando con un FPGA (Field Programmable Gate Array) Artix-7.

(b) ORGANIZACIÓN Se irá explicando paso a paso en cada etapa correspondiente al flujo de carga y/o trabajo, como procederá este algoritmo con cada módulo implementado. Se anexará la correspondiente arquitectura detallándola con imágenes y tablas.

(c) DESARROLLO **CARGA:** En la primera etapa(carga), se escriben todas consecutivamente. (col1 en mem1, col2 en mem2, y así sucesivamente.)

Para la implementación en hardware sirve la asignación:

0	0	...	1
M1	M2	...	Mn+2

Por cada ciclo quedan 2 memorias que te ahorras de escribir, ya que las mismas se necesitan, pero ya están cargadas. Si k es el tamaño de kernell, k-1 memorias quedarían para reutilizar. Concluyendo esta etapa quedaría como última memoria cargada la n+2. Una vez que se termina de procesar, se carga empezando por la memoria siguiente a la última memoria que se terminó de cargar. Y se cargan N memorias. Por ejemplo: si N= 3 convolucionadores, y escribiste por último la memoria número 4, en la siguiente etapa escribirías las 5, la 1 y la 2 (respetando orden).

ROUTING: Van a haber E= N/2 + 1 estados. si N es par (siendo N el número de convolucionadores); y E = N+2 si N es impar. El primer estado se conformará de la siguiente forma:

- M1: C11
- M2: C12; C21
- M3: C13; C22; C31
- M4: C23; C32; C41
- ...
- Mn: C(n-2)3; C(n-1)2; CN1
- Mn+1: C(n-1)3; Cn2
- Mn+2: Cn3

Donde:

- Memoria : convolucionador asignado
- Los : (dos puntos) denotan ASIGNACIÓN (tal memoria asignada a tal convolucionador).
- Cxy = x, convolucionador, y entrada del mismo.

Para el siguiente estado, se efectúa una rotación circular, de N lugares.

IMPLEMENTACIÓN EN HARDWARE Para generalizar, teniendo E estados, N convolucionadores, se tiene que se necesitarán 3 N multiplexores siendo N el número de convolucionadores, es decir, un multiplexor por cada input perteneciente al módulo del convolucionador. Cada multiplexor va a tener a su vez E entradas, y cada multiplexor corresponde a una de las entradas de un módulo convolucionador

ESCRITURA EN MEMORIA: En el estado RUN. En cuanto a la salida de los módulos convolucionadores, se tienen varias etapas en función a las tareas de los mismos. La secuencia de escritura de la salida de los convolucionadores es la misma que la secuencia de escritura que la imagen en memoria. *Por ejemplo*: si se tienen 4 convolucionadores, su salida se escribirá primero en 1 2 3 4, y en el siguiente ciclo en 5 6 1 2.

JUSTIFICACION Se optó por este algoritmo porque con esta implementación se hace uso de N + 2 memorias, siendo N el número de convolucionadores. Con otros algoritmos analizados, como por ejemplo que cada módulo se encargue de cierto sector, comenzando desde los extremos, no se tiene la misma eficiencia en

cuanto a la economía de recursos, ya que se necesitan 3N memorias, e inclusive existía una etapa de solapamiento donde más de una memoria tenía cargado en los mismos datos.