ASSIGNMENT-3

ANANNYA MATHUR
2019TT10953

DESIGN OVERVIEW:

An image filter in VHDL that uses a **3x3 sliding window** for QQVGA size images, with **8-bit pixel resolution**, has to be designed.

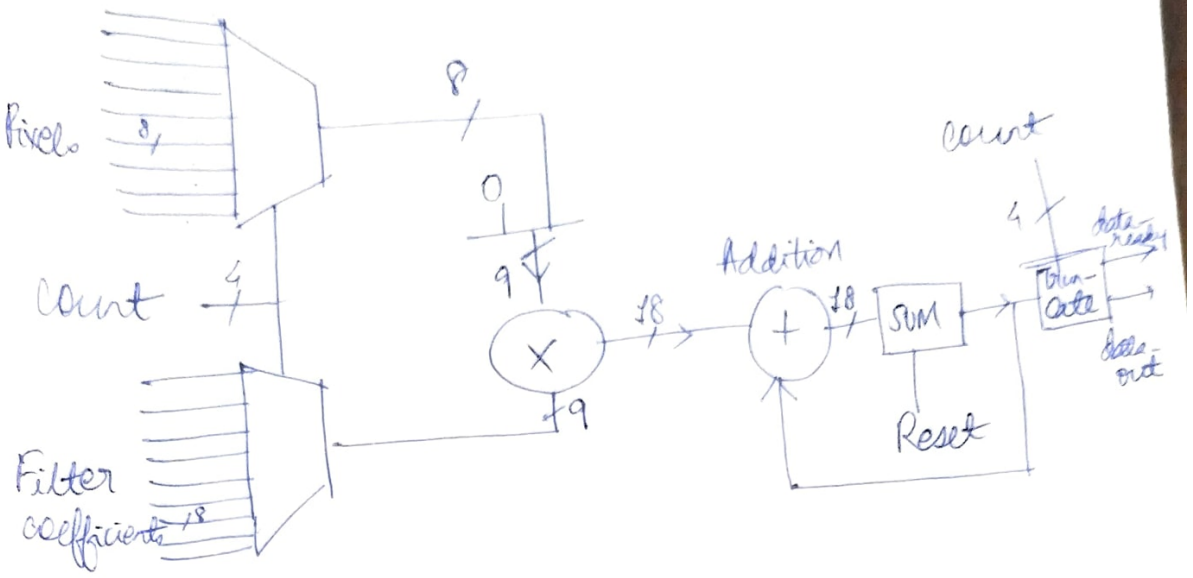
The design makes use of only one multiplier, instead of nine multipliers for multiplying every pixel value with its corresponding filter coefficient. Hence, one multiplier will be used nine times by making use of a multiplexer. The multiplexer will be controlled by a select input (a counter that counts from 0 to 8 and then resets to 0).

Before the multiplication is performed, the pixel values are converted from eight-bit unsigned values to nine bit two's complement values by concatenating a 0 as a sign bit.

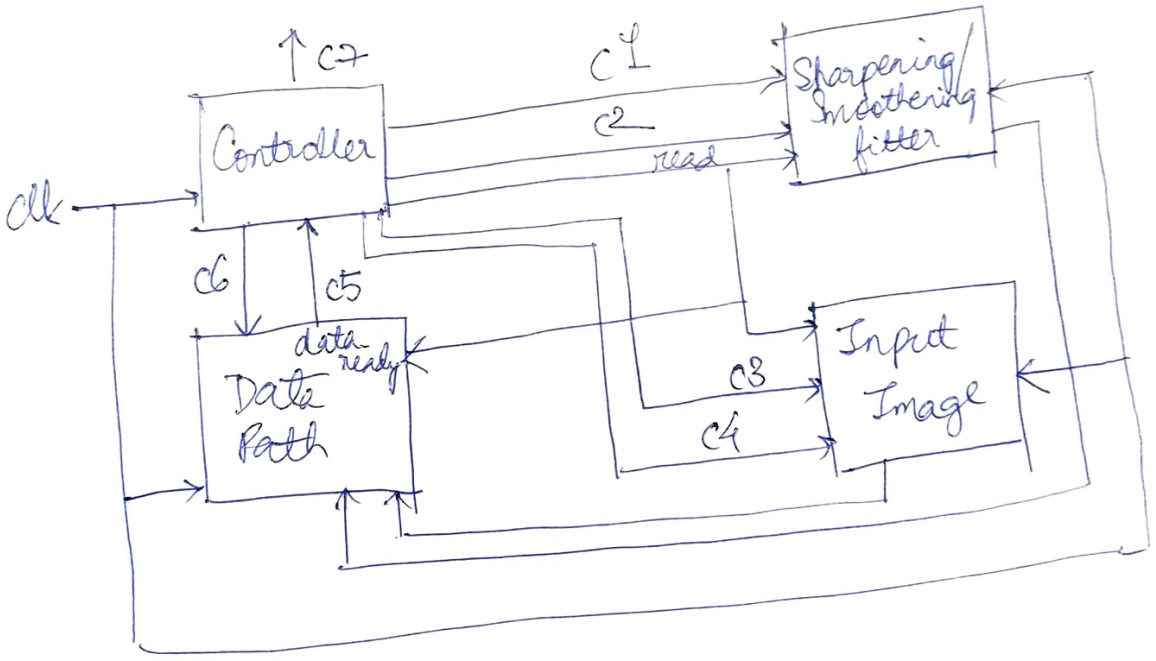
After the last multiplication is conducted, the final sum is truncated.

The select input for truncate is same as count input for the multiplexers.

DATA PATH



OVERALL DESIGN

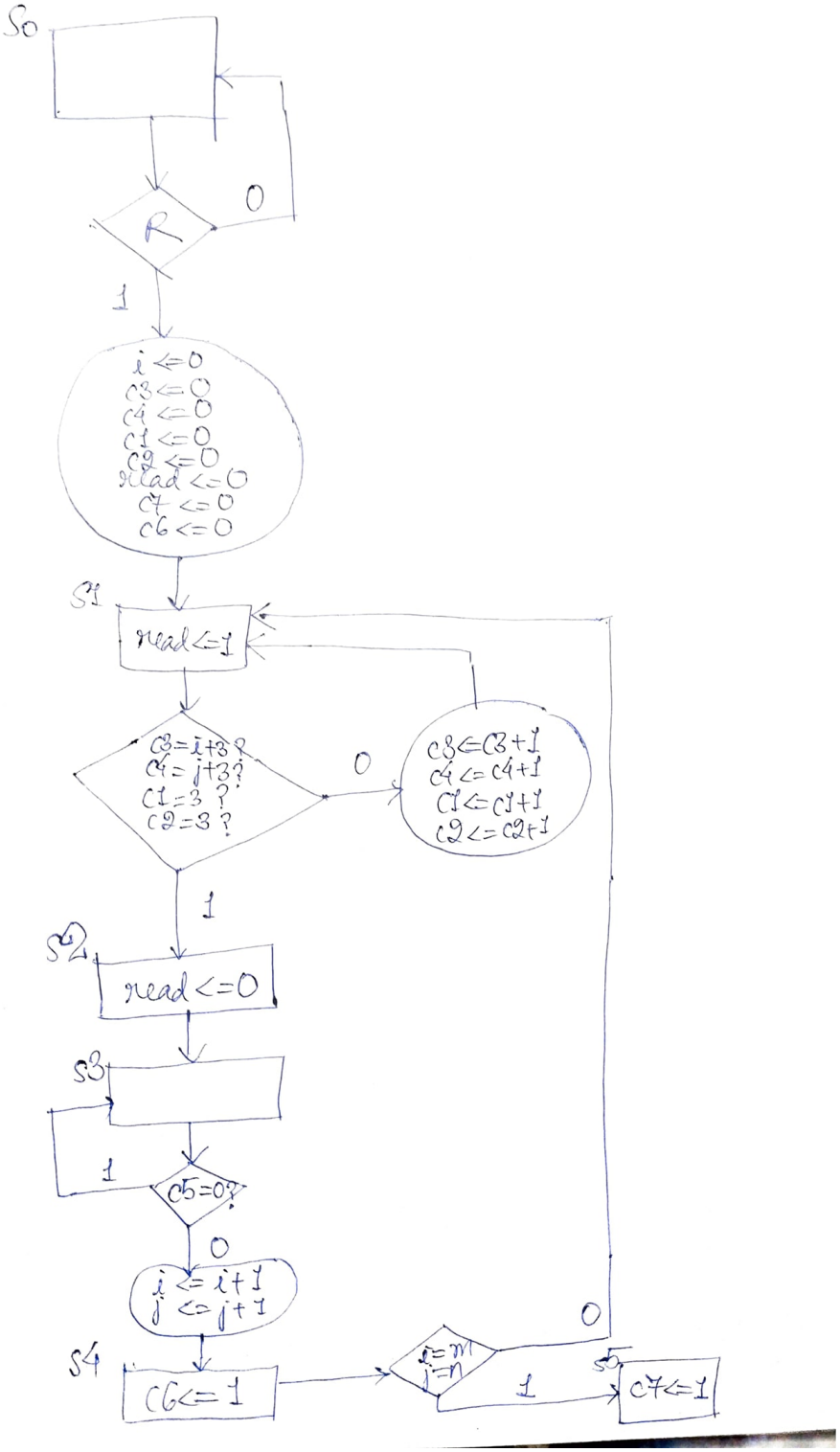


CONTROLLER DESIGN:

In state S0, parameters will be initialised.

In state S1, read will be assigned 1 to read the data stored in memory.

ASM for the Controller:



VHDL code:

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity RAM_64Kx8 is
 port (
 clock : in std_logic;
 read_enable, write_enable : in std_logic; -- signals that enable
read/write operation
 address : in std_logic_vector(15 downto 0); -- 2^16 = 64K
 data_in : in std_logic_vector(7 downto 0);
 data_out : out std_logic_vector(7 downto 0)
 );
end RAM_64Kx8;

entity ROM_32x9 is
 port (
 clock : in std_logic;
 read_enable : in std_logic; -- signal that enables read operation
 address : in std_logic_vector(4 downto 0); -- 2^5 = 32
 data_out : out std_logic_vector(7 downto 0)
 );
end ROM_32x9;

entity MAC is
 port (
 clock : in std_logic;
 control : in std_logic; -- ‘0’ for initializing the sum
 data_in1, data_in2 : in std_logic_vector(17 downto 0);
 data_out : out std_logic_vector(17 downto 0)
 );
end MAC;

architecture Artix of RAM_64Kx8 is
```

```
 type Memory_type is array (0 to 65535) of std_logic_vector (7 downto
0);
 signal Memory_array : Memory_type;
begin
 process (clock) begin
    if rising_edge (clock) then
        if (read_enable = '1') then -- the data read is available after the
clock edge
       data_out <= Memory_array (to_integer (unsigned (address)));
       end if;
       if (write_enable = '1') then -- the data is written on the clock edge
 Memory_array (to_integer (unsigned(address))) <= data_in;
       end if;
   end if;
 end process;
end Artix;
```

```
architecture Artix of ROM_32x9 is
 type Memory_type is array (0 to 31) of std_logic_vector (8 downto 0);
 signal Memory_array : Memory_type;
begin
 process (clock) begin
     if rising_edge (clock) then
         if (read_enable = '1') then -- the data read is available after the
clock edge
              data_out <= Memory_array (to_integer (unsigned (address)));
          end if;
    end if;
 end process;
end Artix;
```

```
architecture Artix of MAC is
 signal sum, product : signed (17 downto 0);
begin
 data_out <= std_logic_vector (sum);
 product <= signed (data_in1) * signed (data_in2)
 process (clock) begin
```

```
    if rising_edge (clock) then -- sum is available after clock edge
        if (control = '0') then -- initialize the sum with the first product
              sum <= std_logic_vector (product);
        else -- add product to the previous sum
              sum <= std_logic_vector (product + signed (sum));
         end if;
      end if;
 end process;
end Artix;
```

OVERALL DESIGN:

```
entity CONTROLLER is
 port (
 clock : in std_logic;
 reset : in std_logic; -- ‘1’ for reset
 c5 : in std_logic; -- ‘1’ if data ready to store the sum
 c7: out std_logic-- ‘1’ if done
 );
end CONTROLLER;

architecture Behavioral of COUNTER is
controller: process(reset, clock)
begin
   if reset= ‘1’ then state<=S0;
   else if (clock’event and clock= ‘1’) then
      case state is
          when S0 =>
              c7<=0;
              if start<=1 then
                  i<=0;
                  j<=0;
                  c3<=-1;
                  c4<=-1;
                  c1<=-1;
                  c2<=-1;
                  read<=0;
                  c6<=0;
```

```
        state<=S1;
      end if;
when S1 =>
        read<=1;
        if (c3=i+1 or c4=j+1 or c1=1 or c2=1) then
            i<=i+1;
            j<=j+1;
            c1<=-1;
            c2<=-1;
            c3<=i-1;
            c4<=j-1;
            state<=S2;
        else
             c3<=c3+1;
             c4<=c4+1;
             c1<=c1+1;
             c2<=c2+1;
       end if;
 when S2 =>
              state<=S3;
  when S3=>
               state<=S4;
  when S4=>
               read<=0;
               state<=S5;
  when S5=>
               if c5=1 then
                  state<=S6;
               end if;
   when S6=>
               c6<=1;
               if (i=m-2 or j=n-2) then
                    state<=S7;
               else
                     state<=S1;
                end if;
   when S7=>
```

```
                        c7<=1;
                        if start=1 then
                        state<=S0;
                        end if;
          end case;
        end if;
 end process;
end Behavioral;
```

```
entity FILTER is            --for applying sharpening/smoothening filters
 port (
 select: in std_logic; --for selecting smoothening/sharpening filter
 clock : in std_logic;
 read : in std_logic; -- '1' for reading the data
 address: out std_logic_vector(4 downto 0);
 read_enable: out std_logic;
 filter_out : out std_logic_vector(8 downto 0)
 );
end FILTER;

architecture Behavioural of FILTER is
Filter_coefficient:process(clock, select, read)

 begin
   if(clock'event and clock= '1') then
      if read=1 then
         read_enable<=1;
         if select=1 then --for smoothening filter
            address<= '00000';
         else
            address<= '10000'; --for sharpening filter
          end if;
          filter_out<=data_out1
      end if;
   end if;
 end process;
```

```
end Behavioural;

entity INPUT_IMAGE is
 port (
 clock : in std_logic;
 read : in std_logic; -- ‘1’ for reading/writing the data
 address: out std_logic_vector(15 downto 0);
 read_enable: out std_logic;
 write_enable: out std_logic;
 data : out std_logic_vector(7 downto 0)
 );
end INPUT_IMAGE;

architecture Behavioural of INPUT_IMAGE is
input_image:process(clock, read)

 begin
   address<= ‘000000000000000’;
   if(clock’event and clock= ‘1’) then
      if read=1 then
         read_enable<=1;
         write_enable<=1;
         data<=data_in;
         address<= address+1;
      else
         read_enable<=0;
         write_enable<=0;

  end process;
end Behavioural;

entity Data_Path is
 port (
 clock : in std_logic;
 write: in std_logic;
 c7: in std_logic; --whether to display
 c6: in std_logic; --to store the data
```

```
 start: in std_logic;
 address: in std_logic_vector(15 downto 0);
 read : in std_logic; -- ‘1’ for reading the data
 Pixel_in : in std_logic_vector(7 downto 0);
 FilterCoeff_in: in std_logic_vector(8 downto 0);
 Data_ready: out std_logic; -- ‘1’ when data is ready
 data_out : out std_logic_vector(8 downto 0)
 c6: out std_logic
 );
end Data_Path;

architecture Behavioural of Data_Path is
DataPath: process(clock, read)
signal count: bit_vector(3 downto 0);
signal A,B: bit_vector(8 downto 0);
signal dataOut: bit_vector(17 downto 0);
signal shift_register: bit_vector(6 downto 0):= others<= ‘0’;


 begin
  if(clock’event and clock= ‘1’) then
     if read=0 then
        count<= ‘0000’;
     else
        Pixel_in <= data;
        FilterCoeff_in<=filter_out;
        count<= count+1;
     end if;
     If count= ‘0000’ then
         A<= ‘0’ and Pixel_in(0);
         B<= ‘0’ and FilterCoeff_in(0);
     Else if count= ‘0001’ then
         A<= ‘0’ and Pixel_in(1);
         B<= ‘0’ and FilterCoeff_in(1);
     Else if count= ‘0010’ then
         A<= ‘0’ and Pixel_in(2);
         B<= ‘0’ and FilterCoeff_in(2);
```

```
      Else if count= ‘0011’ then
         A<= ‘0’ and Pixel_in(3);
         B<= ‘0’ and FilterCoeff_in(3);
      Else if count= ‘0100’ then
         A<= ‘0’ and Pixel_in(4);
         B<= ‘0’ and FilterCoeff_in(4);
      Else if count= ‘0101’ then
         A<= ‘0’ and Pixel_in(5);
         B<= ‘0’ and FilterCoeff_in(5);
      Else if count= ‘0110’ then
         A<= ‘0’ and Pixel_in(6);
         B<= ‘0’ and FilterCoeff_in(6);
      Else if count= ‘0111’ then
         A<= ‘0’ and Pixel_in(7);
         B<= ‘0’ and FilterCoeff_in(7);
      Else if count= ‘1000’ then
         A<= ‘0’ and Pixel_in(8);
         B<= ‘0’ and FilterCoeff_in(8);
       end if;
MAC: MAC port map(clock=>clock, control=>read, data_in1=>A,
data_in2=>B, data_out=>dataOut);
      If (count= ‘1000’) then
          data_ready<=1;
         If c6=1 then -- whether to store the data in table
          dataOut <= shift_register(5 downto 0) and dataOut; --for
truncating
      address<= ‘32768’;

         write<=1;

       else
          data_ready<=0;
           address<= ‘00000’
           write<=0;
            read<=1;
      end if;
      if c7=1 then
```

```
        if start=1 then
           data_out<=dataOut;

        end if;



  end if;
 end process;
end Behavioural;
```