FPGA based real time low latency image convolution

Apurva Nandan, Nitish Vikas Deshpande,
Tushar Singhal, Anshul Patel, Japneet Singh

Electronics Club IIT Kanpur
Summer Project 2018

Contents

Chapter 1

Getting Started

1.1 Intro

1.1.1 What is FPGA?

FPGA stands for field programmable gate array. As the name suggests FPGA's are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. It is basically a device which you can use to configure Logic blocks by just writing a program in a language such as Verilog/VHDL or any other so-called Hardware description Languages(HDL). FPGA's can be configured to perform complex combinational functions, or merely simple logic gates like AND or XOR.

1.1.2 What is HDL?

HDL stands for Hardware description Language. It is a language used to describe the structure of Electronic circuit most commonly Digital electronic circuit. VHDL and Verilog are two most common HDL's. VHDL stands for Very high Speed Integrable Hardware description Language.

1.1.3 Why FPGA and not something like ASIC?

As you know An **Application-Specific Integrated Circuit (ASIC)** is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. An ASIC can no longer be altered after it gets out of the production line. The programmable nature of an FPGA allows the manufacturers to correct mistakes and to even send out patches or updates after the product has been bought.

Figure 1.1: Virtex-5 XUPV5-LX110T

1.1.4 Specifications of our FPGA Board and additional components

We will be using Virtex-5 XUPV5-LX110T FPGA board. It has 128KB of BRAM which is too small to store the data coming from a camera/pc through VGA. But, fortunately it has 256MB of DDR2 RAM which is quite sufficient for our purpose to store the input coming from Camera/pc. This Version of FPGA has VGA in and DVI out ports. We will be using a HDMI to VGA converter to input the data to our FPGA and to display the output of FPGA on a monitor we will be using DVI to VGA converter and display the output on monitor. To upload code on our FPGA board we will use JTAG cable which will directly upload the compiled code on FPGA on just a click.

We will need the following software's

- Xilinx ISE design Suite (preferably use Linux)
- Xilinx Platform Studio
- Xilinx Software Development Kit
- Chipscope

1.2 VGA

1.2.1 What is VGA

VGA stands for video graphics array. A VGA video signal contains 5 active signals. Two signals compatible with TTL logic levels, horizontal sync and

to vga.jpeg

Figure 1.2: HDMI to VGA converter

vertical sync, are used for synchronization of the video. Three analog signals with 0.7 to 1.0-Volt peak-to-peak levels are used to control the colour. The colour signals are Red, Green, and Blue. They are often collectively referred to as the RGB signals. By changing the analog levels of the three RGB signals all other colours are produced.

1.2.2 What we did?

We created a red jpeg file using paint by setting R value to 255 and G , B to 0. We then removed the VGA cable from the monitor and tested this image signals using an oscilloscope. See fig 1.3 for the pinout of VGA port.

1.2.3 Our observations

We observed the Hsync, Vsync and colour signals on the oscilloscope. In fig1.4, the pink graph is Hsync and yellow graph is red colour signal. Observe the small gap in the 2 graphs. This is known as porching. In fig 1.5, the yellow graph is Vsync and pink graph is red colour signal. the main purpose is to give the time to beam scanning for reverse direction (right to left) to start new line.

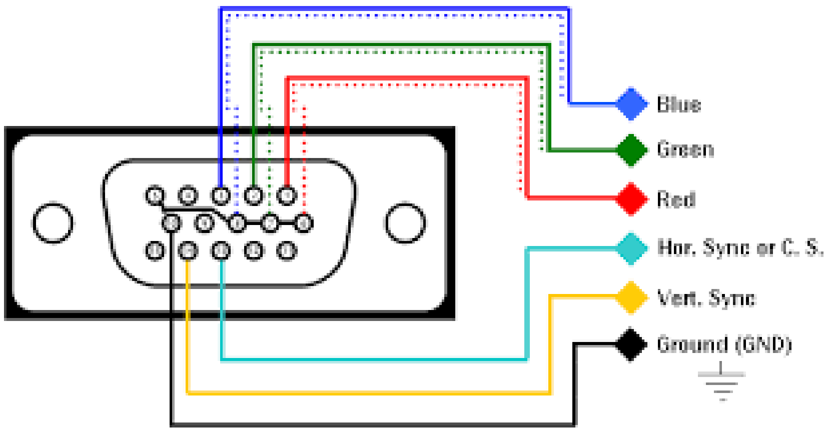


Figure 1.3: VGA port

vs hsync.jpg

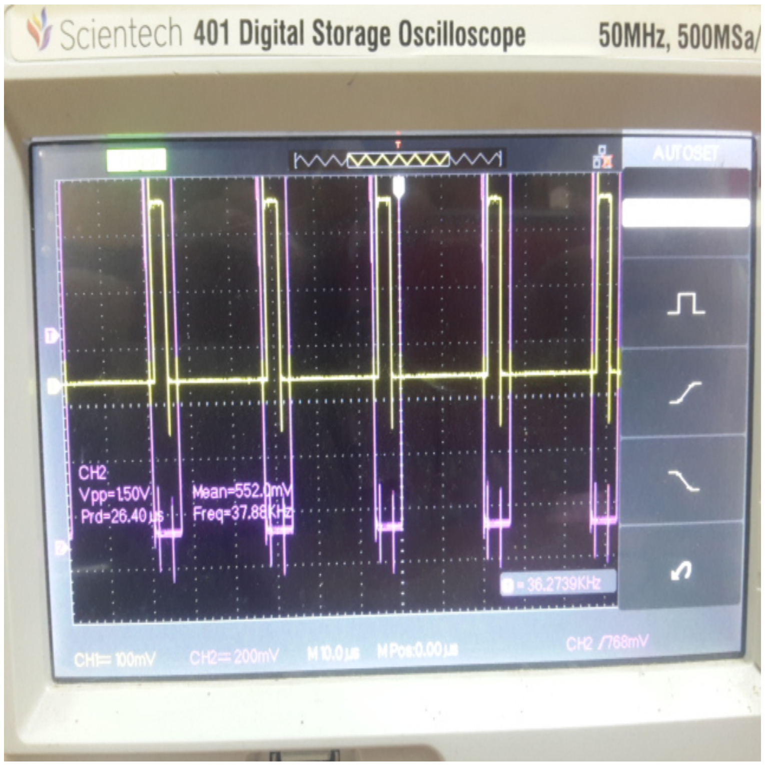


Figure 1.4: red vs hsync

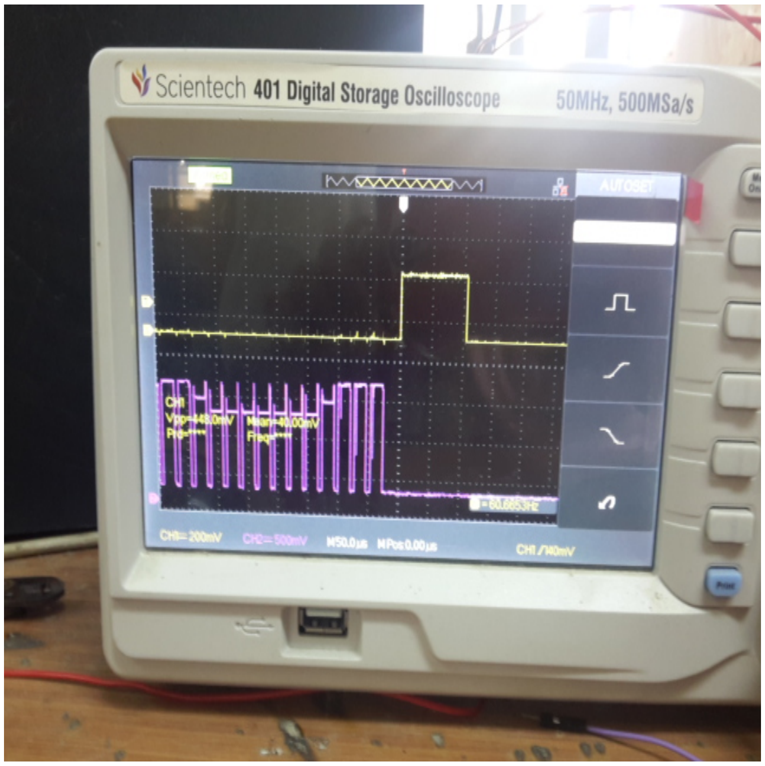
vs vsync.jpg 

Figure 1.5: red vs vsync

Chapter 2

Image convolution

Implementation of image convolution in **vhdl**

2.1 Introduction

2.1.1 Definition

Image convolution is a common algorithm used to filter images. It can be found in most graphics editors, such as Adobe Photoshop and GNU Image Manipulation Program. Typical filters are blur, sharpen and edge detection.

2.1.2 Maths behind it

The image convolution algorithm uses discrete convolution for two dimensions, the definition is shown in the equation below.

$$O(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(x-i, y-j) \cdot H(i,j)$$

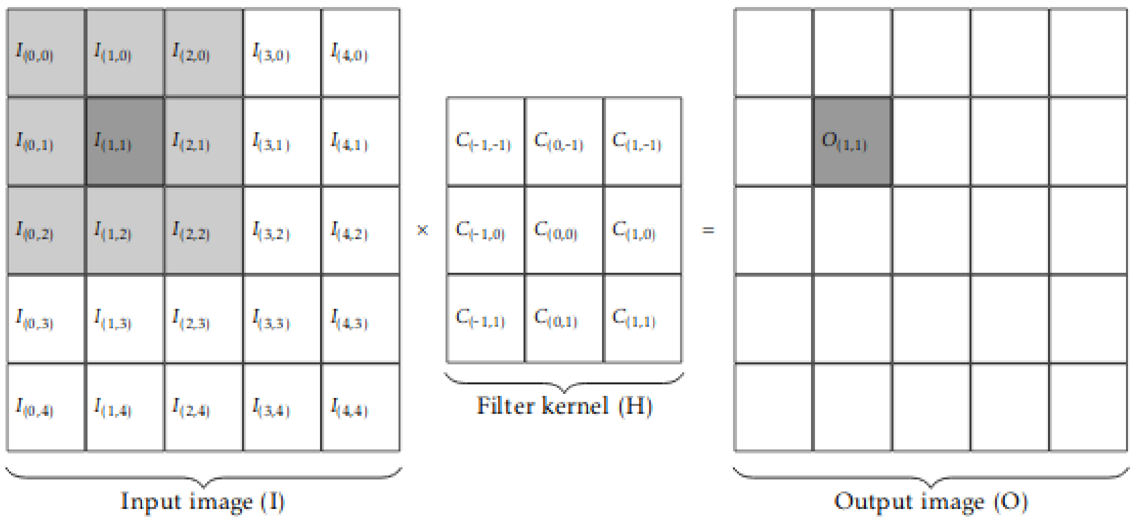
Where O (x, y) is the pixel at position (x, y) in the output image, I (x, y) is the corresponding pixel in the input image and H (i, j) is the filter kernel.

2.2 Implementation in vhdl

2.2.1 Single convolution block diagram

The image convolution consists of the following main modules.

- buffer module
- convolution module
- filter selection module



$I_{(0,0)}$ $I_{(1,0)}$ $I_{(2,0)}$ $I_{(3,0)}$ $I_{(4,0)}$
$I_{(0,1)}$ $I_{(1,1)}$ $I_{(2,1)}$ $I_{(3,1)}$ $I_{(4,1)}$
$I_{(0,2)}$ $I_{(1,2)}$ $I_{(2,2)}$ $I_{(3,2)}$ $I_{(4,2)}$
$I_{(0,3)}$ $I_{(1,3)}$ $I_{(2,3)}$ $I_{(3,3)}$ $I_{(4,3)}$
$I_{(0,4)}$ $I_{(1,4)}$ $I_{(2,4)}$ $I_{(3,4)}$ $I_{(4,4)}$
×
$C_{(-1,-1)}$ $C_{(0,-1)}$ $C_{(1,-1)}$
$C_{(-1,0)}$ $C_{(0,0)}$ $C_{(1,0)}$
$C_{(-1,1)}$ $C_{(0,1)}$ $C_{(1,1)}$
=
$O_{(1,1)}$
Input image (I)
Filter kernel (H)
Output image (O)

Figure 2.1: convolution

- Clock Divider Module

The input is in form of 1 d array with the 3 values of each pixel in a series. For example , consider the stream of pixels with colour as red(255,0,0) , green(0,255,0) , blue(0,0,255) then the data in stream will be 11111111 00000000 00000000 00000000 11111111 00000000 00000000 00000000 11111111 .

Each 8 bit vector value will be received by the top module with every rising edge of clock. This stream of data will pass through the buffer module and then for convolution the buffer module will extract the 9 pixel values as shown in the figure.(9 since we are using a 3x3 convolver).

2.2.2 Buffer module

We implemented circular buffer using fifo(first in first out). A circular buffer, circular queue, cyclic buffer or ring buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. This structure lends itself easily to buffering data streams. The useful property of a circular buffer is that it does not need to have its elements shuffled around when one is consumed. (If a non-circular buffer were used then it would be necessary to shift all elements when one is consumed.) In other words, the circular buffer is well-suited as a FIFO buffer while a standard, non-circular buffer is well suited as a LIFO buffer.

The following is a fifo of 9 depth. The pixel data is read at every rising edge. This fifo has 3 pipes . The data is read from these 3 pipes situated at an interval of 3. The read pointers are incremented after every clock cycle.

This is a prototype of our fifo module.

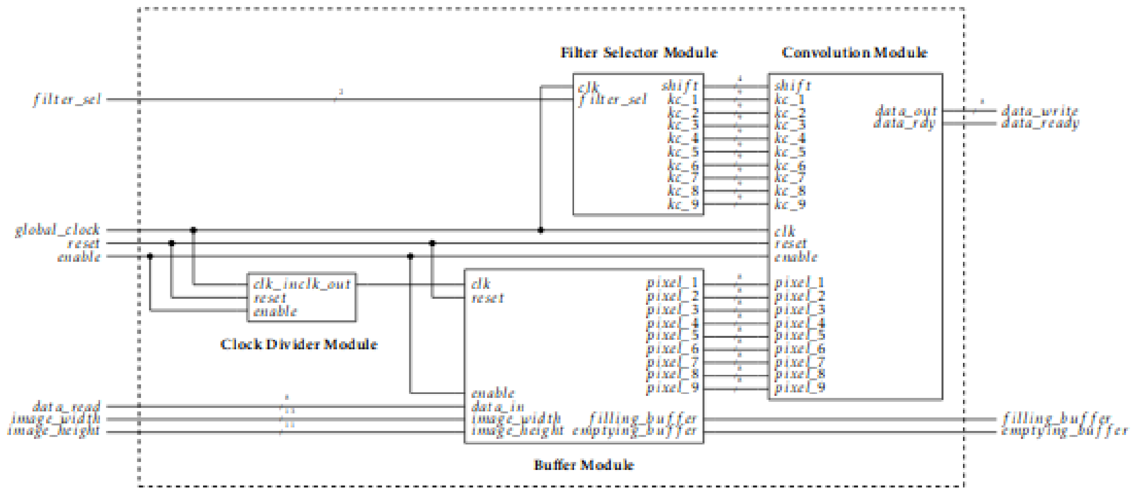


Figure 2.2: Single convolution

```
entity smallfifo is
generic(
fifodepth : integer := 9;
pixelvectorsize : integer := 8;
stride : integer := 1
);
port (

clk :in std_logic ;
datain :in std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout1 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout2 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout3 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
wr :in std_logic;
rd1 :in std_logic;
rd2 :in std_logic;
rd3 :in std_logic;
full :out std_logic;
reset : in std_logic
);






```

buffer.png

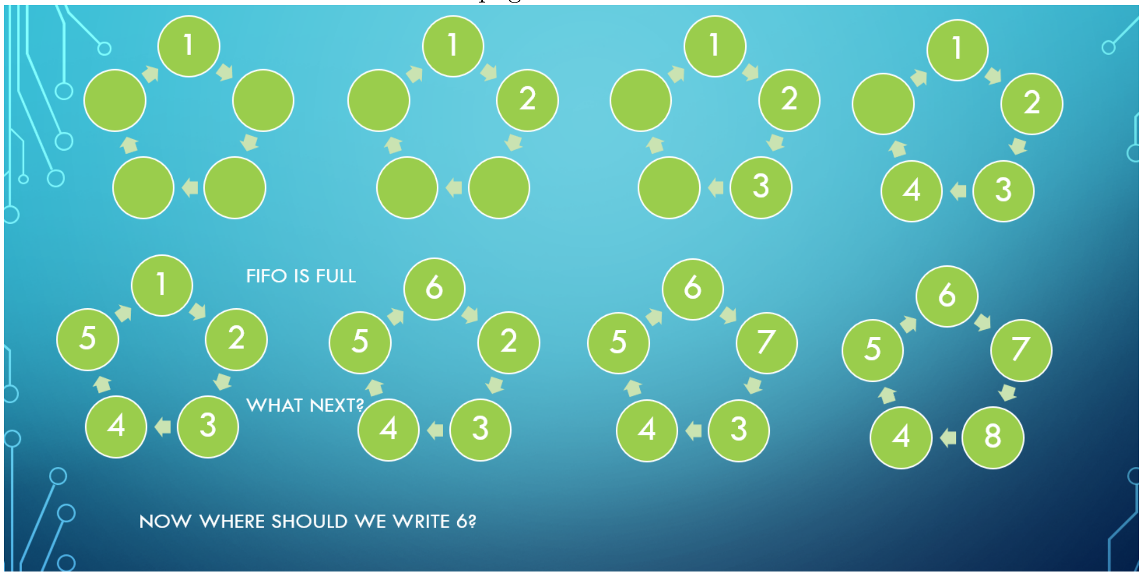
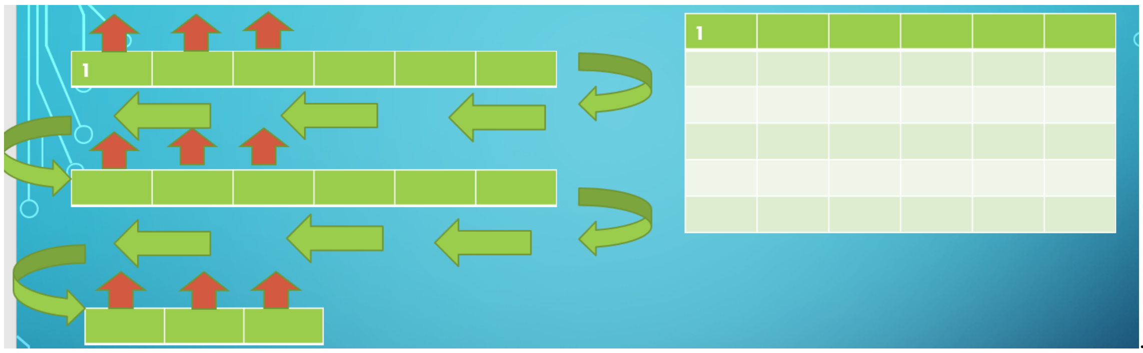
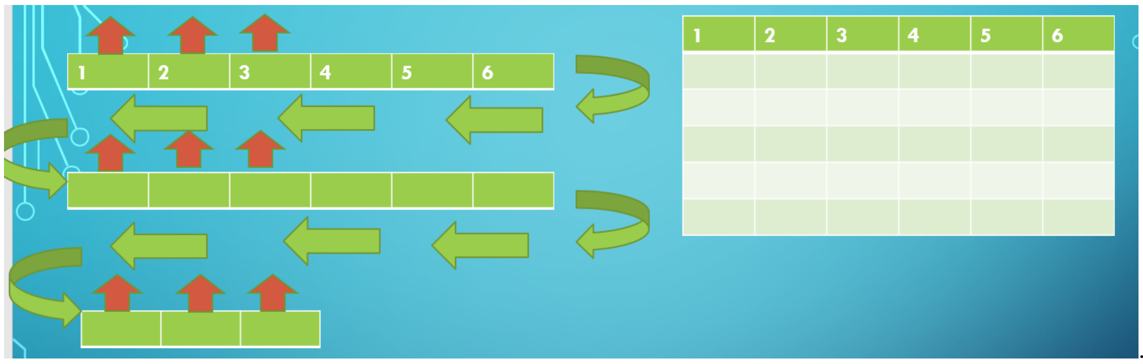


Figure 2.3: Circular buffer



.png

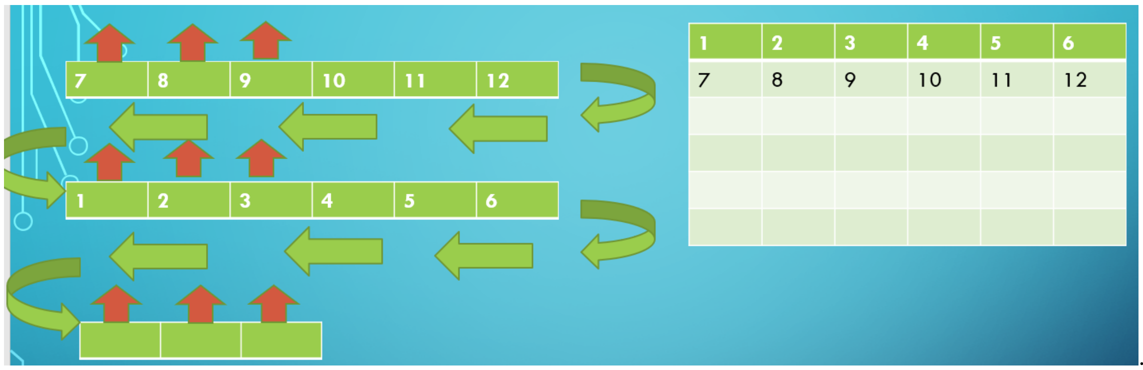
Figure 2.4: starts filling



1	2	3	4	5	6

.png

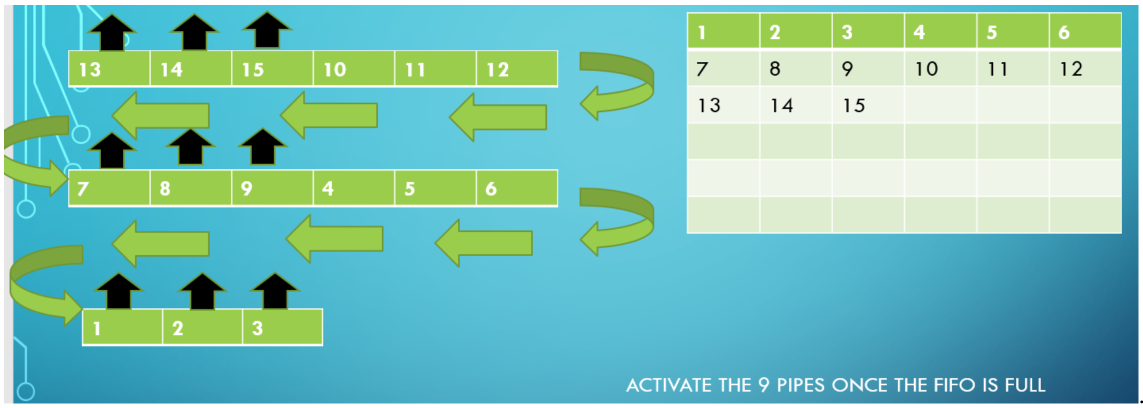
Figure 2.5: fifo 1 is full



1	2	3	4	5	6
7	8	9	10	11	12

.png

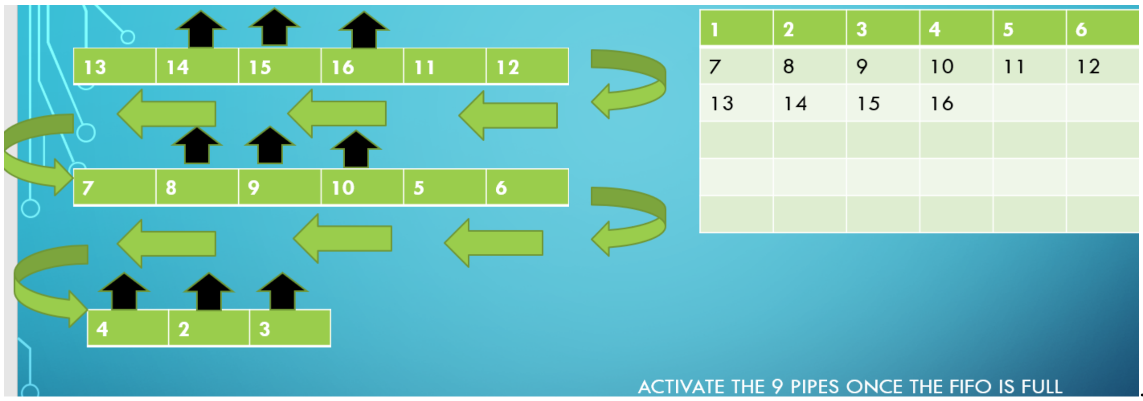
Figure 2.6: fifo 2 is full



1	2	3	4	5	6
7	8	9	10	11	12
13	14	15			

.png

Figure 2.7: fifo 3 is full and pipes are activated



1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16		

.png

Figure 2.8: pipes start shifting

```vhdl
end smallfifo;

architecture Behavioral of smallfifo is
type pixels_in_a_row is array(0 to fifodepth -1 ) of std_ulogic_vector(pixelvectorsize-1 downto 0);
signal pix_row : pixels_in_a_row :=(others => (others => '0'));
constant nbits : natural := integer(ceil(log2(real(fifodepth))));
begin
process(clk) --writing process

variable writeptrc: std_logic_vector( nbits-1 downto 0) :=( others => '0');
variable fullvar : std_logic:='0';
begin
if(clk'event and clk='1' and wr ='1') then
pix_row(conv_integer(writeptrc)) <= datain;
writeptrc := writeptrc+'1';
end if;
if(writeptrc=conv_std_logic_vector(fifodepth,nbits))then
writeptrc:=(others => '0');
fullvar:= '1';
end if;
if(reset = '1') then
writeptrc := (others => '0');
fullvar:= '0';
end if;
full<=fullvar;
end process;
process(clk) --reading process 1
variable r1ptrc: std_logic_vector( nbits-1 downto 0):=( others => '0');
begin
if(clk'event and clk ='0' and rd1='1') then
dataout1 <= pix_row(conv_integer(r1ptrc));
```

```
r1ptrc:= r1ptrc+'1';
end if;
if(r1ptrc=conv_std_logic_vector(fifodepth-6,nbits))then
r1ptrc:=(others => '0');
end if;
if(reset = '1') then
r1ptrc := (others => '0');
end if;
end process;

process(clk) --reading process 2
variable r2ptrc: std_logic_vector( nbits-1 downto 0):=conv_std_logic_vector(3,nbits);
begin
if(clk'event and clk ='0' and rd2='1') then
dataout2 <= pix_row(conv_integer(r2ptrc));
r2ptrc:= r2ptrc+'1';
end if;
if(r2ptrc=conv_std_logic_vector(fifodepth-3,nbits))then
r2ptrc:=conv_std_logic_vector(3,nbits);
end if;
if(reset = '1') then
r2ptrc := conv_std_logic_vector(3,nbits);
end if;
end process;

process(clk) --reading process 3
variable r3ptrc: std_logic_vector( nbits-1 downto 0):=conv_std_logic_vector(6,nbits);
begin
if(clk'event and clk ='0' and rd3='1') then
dataout3 <= pix_row(conv_integer(r3ptrc));
r3ptrc:= r3ptrc+'1';
end if;
if(r3ptrc=conv_std_logic_vector(fifodepth,nbits))then
r3ptrc:=conv_std_logic_vector(6,nbits);
end if;
if(reset = '1') then
r3ptrc := conv_std_logic_vector(6,nbits);
end if;
end process;


end Behavioral;
```

This is a fifo of depth 1920.

```
entity fifo is
generic(
fifodepth : integer := 150;
pixelvectorsize : integer := 8;
stride : integer := 1
);
port (

clk :in std_logic ;
datain :in std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout1 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout2 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout3 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout :out std_ulogic_vector(pixelvectorsize -1 downto 0);
wr :in std_logic;
rd1 :in std_logic;
rd2 :in std_logic;
rd3 :in std_logic;
rd :in std_logic;
full :out std_logic;
reset : in std_logic
);
end fifo;

architecture Behavioral of fifo is
type pixels_in_a_row is array(0 to fifodepth -1 ) of std_ulogic_vector(pixelvectorsize-1 downto 0);
signal pix_row : pixels_in_a_row :=(others => (others => '0'));
constant nbits : natural := integer(ceil(log2(real(fifodepth))));
begin
process(clk) --writing process

variable writeptrc: std_logic_vector( nbits-1 downto 0) :=( others => '0');
variable fullvar : std_logic:='0';
begin
if(clk'event and clk='1' and wr ='1') then
pix_row(conv_integer(writeptrc)) <= datain;
writeptrc := writeptrc+'1';
end if;
if(writeptrc=conv_std_logic_vector(fifodepth,nbits))then
writeptrc:=(others => '0');
end if;
if(writeptrc=conv_std_logic_vector(fifodepth-1 ,nbits))then
fullvar:= '1';
else
fullvar:= '0';
end if;
if(reset = '1') then
writeptrc := (others => '0');
fullvar :='0';
```

```
end if;
full<=fullvar;
end process;
process(clk) --reading process 1
variable r1ptrc: std_logic_vector( nbits-1 downto 0):=( others => '0');
begin
if(clk'event and clk ='0' and rd1='1') then
dataout1 <= pix_row(conv_integer(r1ptrc));
r1ptrc:= r1ptrc+'1';
end if;
if(r1ptrc=conv_std_logic_vector(fifodepth-6,nbits))then
r1ptrc:=(others => '0');
end if;
if(reset = '1') then
r1ptrc := (others => '0');
end if;
end process;

process(clk) --reading process 2
variable r2ptrc: std_logic_vector( nbits-1 downto 0):=conv_std_logic_vector(3,nbits);
begin
if(clk'event and clk ='0' and rd2='1') then
dataout2 <= pix_row(conv_integer(r2ptrc));
r2ptrc:= r2ptrc+'1';
end if;
if(r2ptrc=conv_std_logic_vector(fifodepth-3,nbits))then
r2ptrc:=conv_std_logic_vector(3,nbits);
end if;
if(reset = '1') then
r2ptrc := conv_std_logic_vector(3,nbits);
end if;
end process;

process(clk) --reading process 3
variable r3ptrc: std_logic_vector( nbits-1 downto 0):=conv_std_logic_vector(6,nbits);
begin
if(clk'event and clk ='0' and rd3='1') then
dataout3 <= pix_row(conv_integer(r3ptrc));
r3ptrc:= r3ptrc+'1';
end if;
if(r3ptrc=conv_std_logic_vector(fifodepth,nbits))then
r3ptrc:=conv_std_logic_vector(6,nbits);
end if;
if(reset = '1') then
r3ptrc := conv_std_logic_vector(6,nbits);
end if;
end process;
```

```
process(clk) --reading process
variable readptrc: std_logic_vector( nbits-1 downto 0) :=( others => '0');
begin
if(clk'event and clk ='0' and rd='1') then
dataout <= pix_row(conv_integer(readptrc));
readptrc := readptrc+'1';
end if;
if(readptrc=conv_std_logic_vector(fifodepth,nbits))then
readptrc:=(others => '0');
end if;
if(reset = '1') then
readptrc := (others => '0');
end if;
end process;

end Behavioral;
```

This is the top module. In this module data is sent and 9 pixel values are extracted from the 9 pipes.

```
entity bigfifo is
generic(
pixelvectorsize : integer := 8;
);



port (
datain :in std_ulogic_vector(pixelvectorsize -1 downto 0);
clock : in std_logic ;
pixel_1 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_2 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_3 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_4 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_5 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_6 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_7 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_8 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_9 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
datardy :out std_logic;
reset : in std_logic
);

end bigfifo;

architecture Behavioral of bigfifo is
```

```
component fifo is
port(
clk :in std_logic ;
datain :in std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout1 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout2 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout3 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout :out std_ulogic_vector(pixelvectorsize -1 downto 0);
wr :in std_logic;
rd1 :in std_logic;
rd2 :in std_logic;
rd3 :in std_logic;
rd :in std_logic;
full :out std_logic;
reset : in std_logic
);
end component;

component smallfifo is
port (

clk :in std_logic ;
datain :in std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout1 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout2 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
dataout3 :out std_ulogic_vector(pixelvectorsize -1 downto 0);
wr :in std_logic;
rd1 :in std_logic;
rd2 :in std_logic;
rd3 :in std_logic;
full :out std_logic;
reset : in std_logic
);
end component;

-- internal fifo connections
signal f1_to_f2 : std_ulogic_vector(pixelvectorsize -1 downto 0);
signal f2_to_f3 : std_ulogic_vector(pixelvectorsize -1 downto 0);
-- write enable in individual fifo
signal wr_enable1 : std_logic :='1';
signal wr_enable2 : std_logic :='0';
signal wr_enable3 : std_logic :='0';
-- read enable from 3 pipes in fifo1
signal fifo1_r1 : std_logic :='0';
signal fifo1_r2 : std_logic :='0';
signal fifo1_r3 : std_logic :='0';
-- read enable from 3 pipes in fifo2
```

```
signal fifo2_r1 : std_logic :='0';
signal fifo2_r2 : std_logic :='0';
signal fifo2_r3 : std_logic :='0';
-- read enable from 3 pipes in fifo3
signal fifo3_r1 : std_logic :='0';
signal fifo3_r2 : std_logic :='0';
signal fifo3_r3 : std_logic :='0';
-- read for popping out values from 1 fifo and send to next fifo
signal fifo1_r : std_logic :='0';
signal fifo2_r : std_logic :='0';

signal pixel_1_unsigned :std_ulogic_vector (pixelvectorsize -1 downto 0);
signal pixel_2_unsigned :std_ulogic_vector (pixelvectorsize -1 downto 0);
signal pixel_3_unsigned :std_ulogic_vector(pixelvectorsize -1 downto 0);
signal pixel_4_unsigned :std_ulogic_vector (pixelvectorsize -1 downto 0);
signal pixel_5_unsigned :std_ulogic_vector (pixelvectorsize -1 downto 0);
signal pixel_6_unsigned :std_ulogic_vector (pixelvectorsize -1 downto 0);
signal pixel_7_unsigned :std_ulogic_vector(pixelvectorsize -1 downto 0);
signal pixel_8_unsigned :std_ulogic_vector (pixelvectorsize -1 downto 0);
signal pixel_9_unsigned :std_ulogic_vector(pixelvectorsize -1 downto 0);

signal fifo1_full : std_logic ;
signal fifo2_full : std_logic ;
signal fifo3_full : std_logic ;
signal datardy_sig : std_logic :='0';
begin

FIFO1 : fifo port map (
clk => clock,
datain => datain,
dataout1 => pixel_7_unsigned,
dataout2 => pixel_8_unsigned,
dataout3 => pixel_9_unsigned,
dataout => f1_to_f2,
wr => wr_enable1,
rd1 => fifo1_r1,
rd2 => fifo1_r2,
rd3 => fifo1_r3,
rd => fifo1_r,
full => fifo1_full,
reset => reset
);
FIFO2 : fifo port map (
clk => clock,
datain => f1_to_f2,
dataout1 => pixel_4_unsigned,
dataout2 => pixel_5_unsigned,
```

```vhdl
dataout3 => pixel_6_unsigned,
dataout => f2_to_f3,
wr => wr_enable2,
rd1 => fifo2_r1,
rd2 => fifo2_r2,
rd3 => fifo2_r3,
rd => fifo2_r,
full => fifo2_full,
reset => reset
);
FIFO3 : smallfifo port map (
clk => clock,
datain => f2_to_f3,
dataout1 => pixel_1_unsigned,
dataout2 => pixel_2_unsigned,
dataout3 => pixel_3_unsigned,
wr => wr_enable3,
rd1 => fifo3_r1,
rd2 => fifo3_r2,
rd3 => fifo3_r3,
full => fifo3_full,
reset => reset
);


process(clock)
variable flag_fifo1 :std_logic := '0';
--used flags to ensure that the following blocks of code are executed only once

begin
if(clock'event and clock='1' and fifo1_full ='1' and flag_fifo1='0') then
fifo1_r <= '1';
wr_enable2 <= '1';
flag_fifo1 := '1';
end if;
if(reset ='1') then
fifo1_r <= '0';
wr_enable2 <= '0';
flag_fifo1 := '0';
end if;
end process;

process(clock)
variable flag_fifo2 :std_logic := '0';

begin
if(clock'event and clock='1' and fifo2_full ='1' and flag_fifo2 ='0') then
```

```
fifo2_r <= '1';
wr_enable3 <= '1';
flag_fifo2 := '1';
end if;
if(reset ='1') then
wr_enable3 <= '0';
fifo2_r <= '0';
flag_fifo2 := '0';
end if;
end process;

process(clock)
variable flag_fifo3 :std_logic := '0';
begin
if(clock'event and clock='1' and fifo3_full ='1' and flag_fifo3 ='0') then
fifo1_r1 <= '1';
fifo1_r2 <= '1';
fifo1_r3 <= '1';
fifo2_r1 <= '1';
fifo2_r2 <= '1';
fifo2_r3 <= '1';
fifo3_r1 <= '1';
fifo3_r2 <= '1';
fifo3_r3 <= '1';
datardy_sig<='1';
--activated all the 9 pixel pipes
flag_fifo3 := '1';
end if;
if(reset = '1') then
fifo1_r1 <= '0';
fifo1_r2 <= '0';
fifo1_r3 <= '0';
fifo2_r1 <= '0';
fifo2_r2 <= '0';
fifo2_r3 <= '0';
fifo3_r1 <= '0';
fifo3_r2 <= '0';
fifo3_r3 <= '0';
datardy_sig<='0';
flag_fifo3 := '0';
end if;
end process;

datardy<=datardy_sig ;
pixel_1 <= pixel_1_unsigned;
pixel_2 <= pixel_2_unsigned;
pixel_3 <= pixel_3_unsigned;
```

```
pixel_4 <= pixel_4_unsigned;
pixel_5 <= pixel_5_unsigned;
pixel_6 <= pixel_6_unsigned;
pixel_7 <= pixel_7_unsigned;
pixel_8 <= pixel_8_unsigned;
pixel_9 <= pixel_9_unsigned;

end Behavioral;
```

This is the code of convolution module.

```
entity matrix_multiply_top is
generic(pixelvectorsize : integer := 8);
port (
             clock: in STD_LOGIC ;
                                 nd_sig : in std_logic;
                                 rdy :out std_logic;
           pixel_1 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_2 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_3 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_4 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_5 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_6 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_7 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_8 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_9 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);

           data_out : out STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0)--output
              );
end matrix_multiply_top;
architecture Behavioral of matrix_multiply_top is

--signal clock: std_logic;
begin
process(clock, nd_sig)
variable output: STD_ULOGIC_VECTOR(7 downto 0);
variable kp_1: float32;
variable kp_2: float32;
variable kp_3: float32;
variable kp_4: float32;
variable kp_5: float32;
variable kp_6: float32;
variable kp_7: float32;
variable kp_8: float32;
variable kp_9: float32;
variable sum : float32;
variable saturated_sum: float32;
variable kc_1: float32 := "00111111100000000000000000000000";
variable kc_2: float32 := "00000000000000000000000000000000";
variable kc_3: float32 := "10111111100000000000000000000000";
variable kc_4: float32 := "01000000000000000000000000000000";
variable kc_5: float32 := "00000000000000000000000000000000";
variable kc_6: float32 := "11000000000000000000000000000000";
variable kc_7: float32 := "00111111100000000000000000000000";
variable kc_8: float32 := "00000000000000000000000000000000";
variable kc_9: float32 := "10111111100000000000000000000000";

variable done :STD_LOGIC :='0';
begin
done := '0';
if(  clock'event and clock='1' and nd_sig='1') then
kp_1:= kc_1*to_float(unsigned(pixel_1));
kp_2:= kc_2*to_float(unsigned(pixel_2));
kp_3:= kc_3*to_float(unsigned(pixel_3));
kp_4:= kc_4*to_float(unsigned(pixel_4));
kp_5:= kc_5*to_float(unsigned(pixel_5));
kp_6:= kc_6*to_float(unsigned(pixel_6));
kp_7:= kc_7*to_float(unsigned(pixel_7));
kp_8:= kc_8*to_float(unsigned(pixel_8));
kp_9:= kc_9*to_float(unsigned(pixel_9));
sum:=(((kp_1+kp_2)+(kp_3+kp_4))+((kp_5+kp_6)+(kp_7+kp_8)))+kp_9;
if(sum>255.0) then
saturated_sum:="01000011011111110000000000000000";
elsif(sum<0.0) then
saturated_sum := "00000000000000000000000000000000";
else
saturated_sum:= sum;
end if;
output:=std_ulogic_vector(to_unsigned(to_integer(saturated_sum), output'length));
done:='1';
end if;
data_out<= output;
rdy <= nd_sig and done;
end process;
end Behavioral;
```

This is the code for top module.

```
entity top is
generic(
fifodepth : integer := 150;
pixelvectorsize : integer := 8;
stride : integer := 1
);
```

```
port(datain: IN STD_uLOGIC_VECTOR(pixelvectorsize -1 downto 0);
 dataout: out STD_uLOGIC_VECTOR(pixelvectorsize -1 downto 0);
 clock: in STD_LOGIC;
 dat_rdy : out STD_LOGIC;
 reset : in STD_LOGIC
 );

end top;

architecture Behavioral of top is
component bigfifo is
port (
datain :in std_ulogic_vector(pixelvectorsize -1 downto 0);
clock : in std_logic ;
pixel_1 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_2 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_3 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_4 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_5 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_6 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_7 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_8 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
pixel_9 : out std_ulogic_vector(pixelvectorsize -1 downto 0);
datardy :out std_logic;
reset : in std_logic
);
end component;

component matrix_multiply_top is

port (
           clock: in STD_LOGIC ;
                          nd_sig : in std_logic;
                          rdy :out std_logic;
           pixel_1 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_2 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_3 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_4 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_5 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_6 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_7 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_8 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
           pixel_9 : in  STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);

            data_out : out STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0)--output
              );
end component;
```

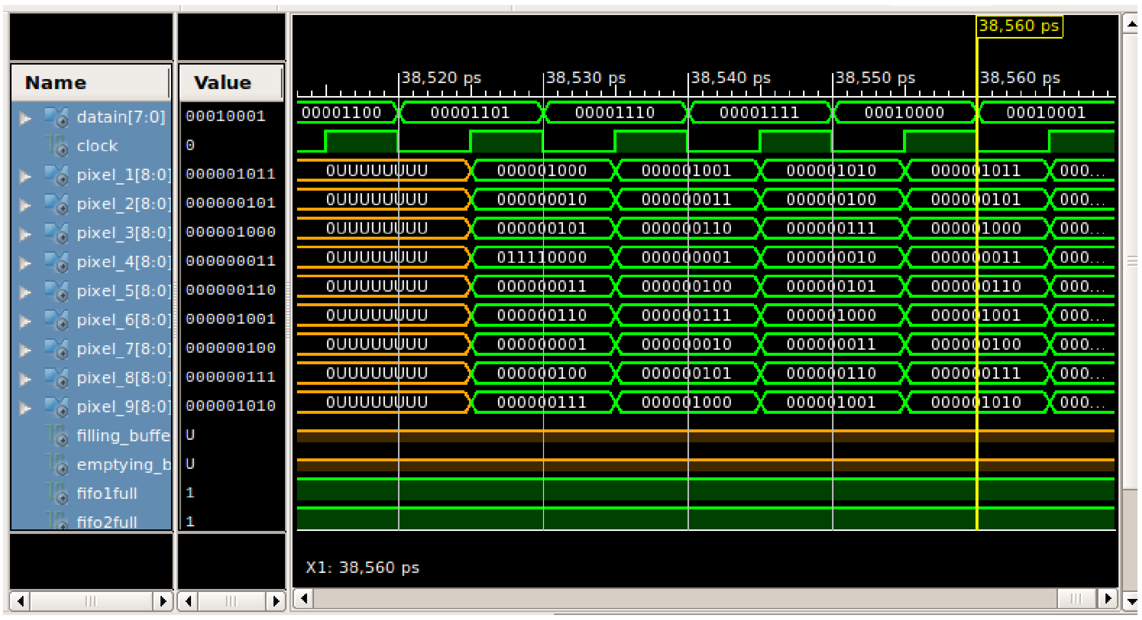
```
signal  pixel1 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel2 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel3 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel4 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel5 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel6 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel7 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel8 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal  pixel9 : STD_ULOGIC_VECTOR(pixelvectorsize -1 downto 0);
signal nd_sig :std_logic;
signal kc_1 : float32;
signal kc_2 : float32;
signal kc_3 : float32;
signal kc_4 : float32;
signal kc_5 : float32;
signal kc_6 : float32;
signal kc_7 : float32;
signal kc_8 : float32;
signal kc_9 : float32;


begin

fifo : bigfifo port map (
datain => datain,
pixel_1 =>pixel1,
pixel_2 =>pixel2,
pixel_3 =>pixel3,
pixel_4 =>pixel4,
pixel_5 =>pixel5,
pixel_6 =>pixel6,
pixel_7 =>pixel7,
pixel_8 =>pixel8,
pixel_9 =>pixel9,
clock => clock,
datardy => nd_sig,
reset=> reset);

conv: matrix_multiply_top port map(
clock => clock,
nd_sig => nd_sig,
rdy => dat_rdy,
pixel_1 =>pixel1,
pixel_2 =>pixel2,
pixel_3 =>pixel3,
pixel_4 =>pixel4,
pixel_5 =>pixel5,
pixel_6 =>pixel6,
pixel_7 =>pixel7,
pixel_8 =>pixel8,
pixel_9 =>pixel9,
data_out => dataout

);


end Behavioral;
```

The code for test bench is as follows.

```
ENTITY testbench IS
  END testbench;

        ARCHITECTURE behavior OF testbench IS

  -- Component Declaration
          COMPONENT top is
port(
 datain: IN STD_uLOGIC_VECTOR(7 downto 0);
 dataout: out STD_uLOGIC_VECTOR(7 downto 0);
 clock: in STD_LOGIC;
 dat_rdy : out STD_LOGIC;
 reset: in std_logic
 );
END COMPONENT;

          SIGNAL data_out :  std_ulogic_vector(7 downto 0) ;
          SIGNAL data_in :  std_ulogic_vector(7 downto 0);
                          signal clock : std_logic := '0';
                          signal data_rdy : std_logic;
                           constant clock_period : time := 1 ns;
                                signal eof_sig : std_logic :='0';
  ----------------------------------------------------------------------------
  -- Testbench Internal Signals
  ----------------------------------------------------------------------------
  file file_VECTORS : text;
  file file_RESULTS : text;

  constant c_WIDTH : natural := 8;

 signal temp: std_ulogic_vector(7 downto 0);



```

```
begin
  uut: top PORT MAP (
  datain =>data_in,
  dataout=>data_out,
  clock=>clock,
  dat_rdy=>data_rdy,
   reset => eof_sig);
  ---------------------------------------------------------------------------
  -- Instantiate and Map UUT
  ---------------------------------------------------------------------------


   clock_process :process
   begin
                clock <= '0';
                wait for clock_period/2;
                clock <= '1';
                wait for clock_period/2;
   end process;


  -------------------------------------------------------------------------
  -- This procedure reads the file input_vectors.txt which is located in the
  -- simulation project area.
  -- It will read the data in and send it to the ripple-adder component
  -- to perform the operations.  The result is written to the
  -- output_results.txt file, located in the same directory.
  -------------------------------------------------------------------------
 reading: process
    variable v_ILINE     : line;

    variable vector: std_ulogic_vector(c_WIDTH-1 downto 0);

    variable v_SPACE     : character;
     variable v1GoodRead      : boolean := true;
          variable v2GoodRead      : boolean := true;
     variable testing : std_logic :='1';
     variable i: integer :=0;
  begin

    file_open(file_VECTORS, "a.txt",  read_mode);
    while not endfile(file_vectors) loop
      readline(file_VECTORS, v_ILINE);
--                read(v_ILINE, vector, v1GoodRead);
--            read(v_ILINE, v_SPACE, v2GoodRead );
            while (i <150) loop--not (v_SPACE = CR ) and v1GoodRead and v2GoodRead) loop
                                read(v_ILINE, vector, v1GoodRead);
            read(v_ILINE, v_SPACE, v2GoodRead );
                                if (v1GoodRead ) then
                                data_in<= vector;
                                wait for clock_period ;
            end if;
                                i:=i+1;
                                end loop;
          i:=0;
   if(endfile(file_VECTORS)) then eof_sig <= '1' ;
        else eof_sig <='0';
        end if;
        end loop;
    file_close(file_VECTORS);
         wait;
           end process reading;
  writing: process
      variable v_OLINE     : line;
                variable vector: std_ulogic_vector(c_WIDTH-1 downto 0);
                variable v_SPACE     : character :=' ';
                variable new_line : character :=CR;
                variable pixels_c: integer:= 0;
      variable line_count: integer :=0;
                begin
 file_open(file_RESULTS, "c.txt", write_mode);
 while ( line_count<50) loop
  wait until data_rdy'event and data_rdy='1';
  vector := data_out;
  write(v_OLINE, vector);
  if(pixels_c>142) then
    pixels_c:=0;
    write(v_OLINE, new_line);
    line_count := line_count+1;
  else write(v_OLINE, v_SPACE);
    pixels_c:=pixels_c+1; end if;
end loop;
writeline(file_RESULTS, v_OLINE);
          file_close(file_RESULTS);
      wait;
    end process writing;

end;
```

Our results of test bench were as follows.



```
Name            Value
datain[7:0]     00010001
clock           0
pixel_1[8:0]    000001011
pixel_2[8:0]    000000101
pixel_3[8:0]    000001000
pixel_4[8:0]    000000011
pixel_5[8:0]    000000110
pixel_6[8:0]    000001001
pixel_7[8:0]    000000100
pixel_8[8:0]    000000111
pixel_9[8:0]    000001010
filling_buffe   U
emptying_b      U
fifo1full       1
fifo2full       1
```

```
38,560 ps
         38,520 ps   38,530 ps   38,540 ps   38,550 ps   38,560 ps
datain   00001100  00001101   00001110   00001111   00010000   00010001
pixel_1  0UUUUUUUU  000001000  000001001  000001010  000001011  000...
pixel_2  0UUUUUUUU  000000010  000000011  000000100  000000101  000...
pixel_3  0UUUUUUUU  000000101  000000110  000000111  000001000  000...
pixel_4  0UUUUUUUU  011110000  000000001  000000010  000000011  000...
pixel_5  0UUUUUUUU  000000011  000000100  000000101  000000110  000...
pixel_6  0UUUUUUUU  000000110  000000111  000001000  000001001  000...
pixel_7  0UUUUUUUU  000000001  000000010  000000011  000000100  000...
pixel_8  0UUUUUUUU  000000100  000000101  000000110  000000111  000...
pixel_9  0UUUUUUUU  000000111  000001000  000001001  000001010  000...
X1: 38,560 ps
```

Figure 2.9: Simulation of fifo

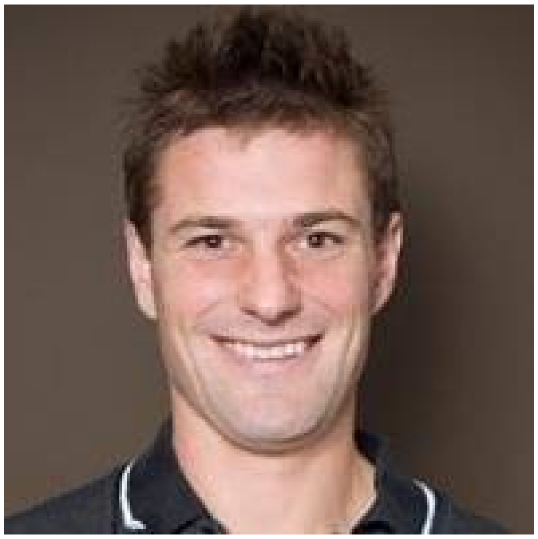
filter.jpg 

Figure 2.10: actual image

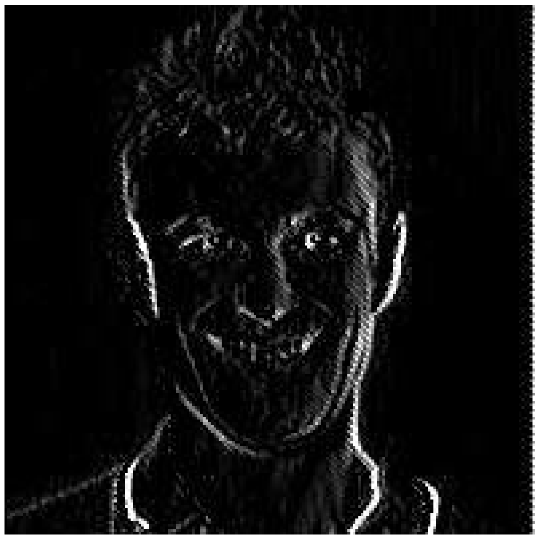
filter.jpg 

Figure 2.11: image after application of sobel filter

Chapter 3

XPS

3.1 Introduction to XPS

Xilinx Platform Studio (XPS) is a key component of the ISE Embedded Edition Design Suite, helping the hardware designer to easily build, connect and configure embedded processor-based systems; from simple state machines to full-blown 32-bit RISC microprocessor systems.

XPS employs graphical design views and sophisticated correct-by-design wizards to guide developers through the steps necessary to create custom processor systems within minutes.

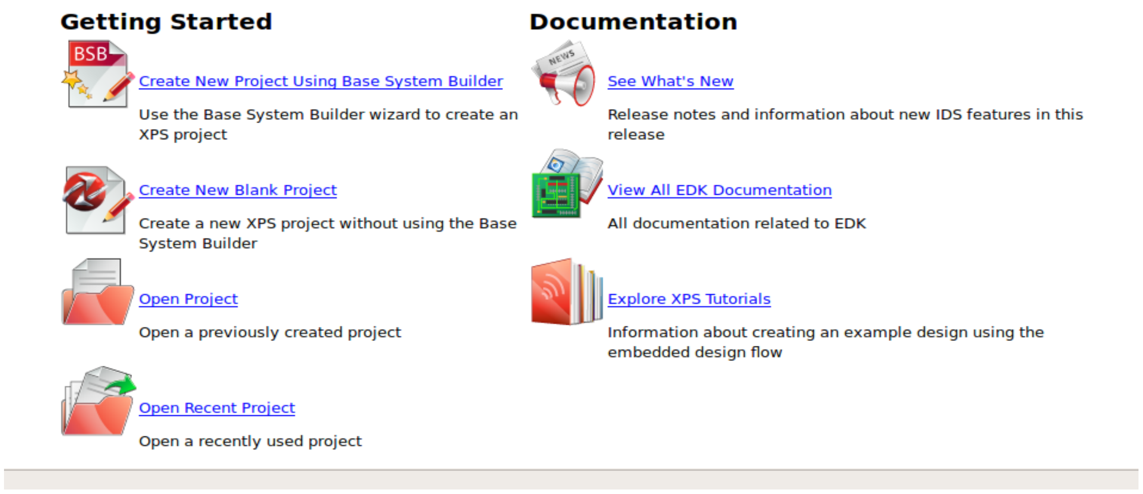
The true potential of XPS emerges with its ability to configure and integrate plug and play IP cores from the Xilinx Embedded IP catalog, with custom or 3rd party Verilog and VHDL designs.

Firmware and software developers benefit from XPS integration with Xilinx SDK which allows the automatic generation of critical system software such as boot loaders, bare metal BSP, and Linux BSPs. This capability ensures that OS porting and applications development can begin without delay caused by firmware development.

3.2 Why is it of use to us?

Our vhdl module for sobel filter written in ISE design suite and tested on Isim is just a standalone module . Although its possible to incorporate our module with dvi and vga in ise , we would not be able to make full use of ddr2ram. So we decided to shift to xps and make full use of ddr2ram using MPMC (Multi port Memory Controller).

We first studied how to make a coprocessor and link it to microblaze. We followed the tutorial in cs 150 .


Getting Started

Create New Project Using Base System Builder

Use the Base System Builder wizard to create an XPS project

Create New Blank Project

Create a new XPS project without using the Base System Builder

Open Project

Open a previously created project

Open Recent Project

Open a recently used project

Documentation

See What's New

Release notes and information about new IDS features in this release

View All EDK Documentation

All documentation related to EDK

Explore XPS Tutorials

Information about creating an example design using the embedded design flow

Figure 3.1: xps main window

3.3 From where did we start?

Instead of starting afresh with the vga and dvi cores , we picked up a template of a project from cs150 . His project was a basic skin mask filter. We edited his code by removing the skin mask . In its place we added our own sobelcop module which was succesfully simulated in Isim .

3.4 A step by step guide to create your own pcore

3.4.1 Start from template

- Clone the repository FPGASummerProject18 from

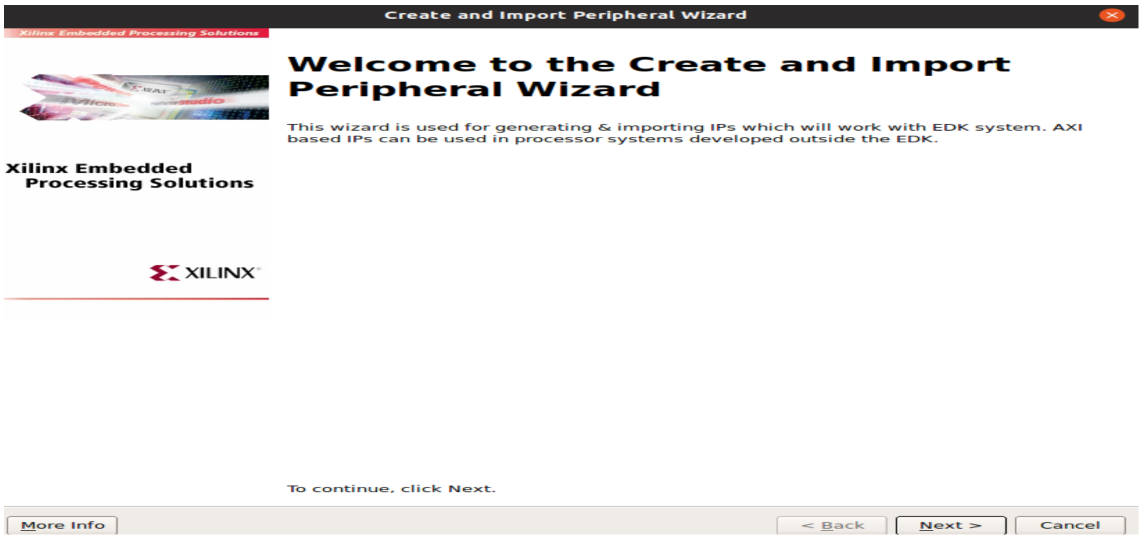
  https://github.com/apurvanandan1997?tab=repositories. This repository contains the edited code from cs 150.

- Open XPS.

  XPS can be opened from terminal by typing xps.

  The following window appears. Click on Open project . Go to the directory where you have extracted the files from the repository . In this location FPGASummerProject18/cs150-master-no-filter/dviproj/ You will find system.xmp. Open it. XMP stands for xilinx microprocessor project.

- Go to hardware -¿ Create or import peripheral. The following window will appear. Click next.


Create and Import Peripheral Wizard

Xilinx Embedded Processing Solutions

Welcome to the Create and Import Peripheral Wizard

This wizard is used for generating & importing IPs which will work with EDK system. AXI based IPs can be used in processor systems developed outside the EDK.

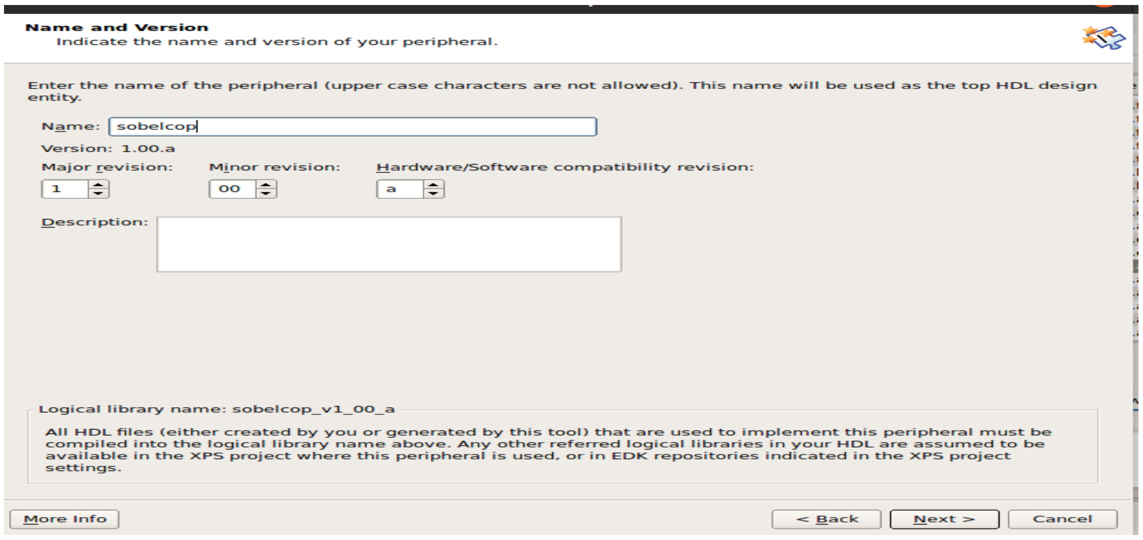
To continue, click Next.

Figure 3.2: welcome

- In this window peripheral name , type name as sobelcop.
- In the bus settings , choose fsl . FSL stands for fast simplex link.
- In the window of fsl settings , type number of input and output words as 1
- In peripheral implementation support , put a tick on the first 2 options.
- Click finish in the last window . Your peripheral is succesfully generated.

3.4.2 What's present in pcores ?

- Open the pcores directory . See fig.
- You will find 3 folders by default .
  - Data
  - devl
  - hdl
- We created an extra folder for our purpose Netlist which contain the ngc files for chipscope and fifo generator . These are compiled files and so you dont need to include the verilog files for the same.
- In the data directory , you will find 3 files



Name and Version
Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name: sobelcop

Version: 1.00.a

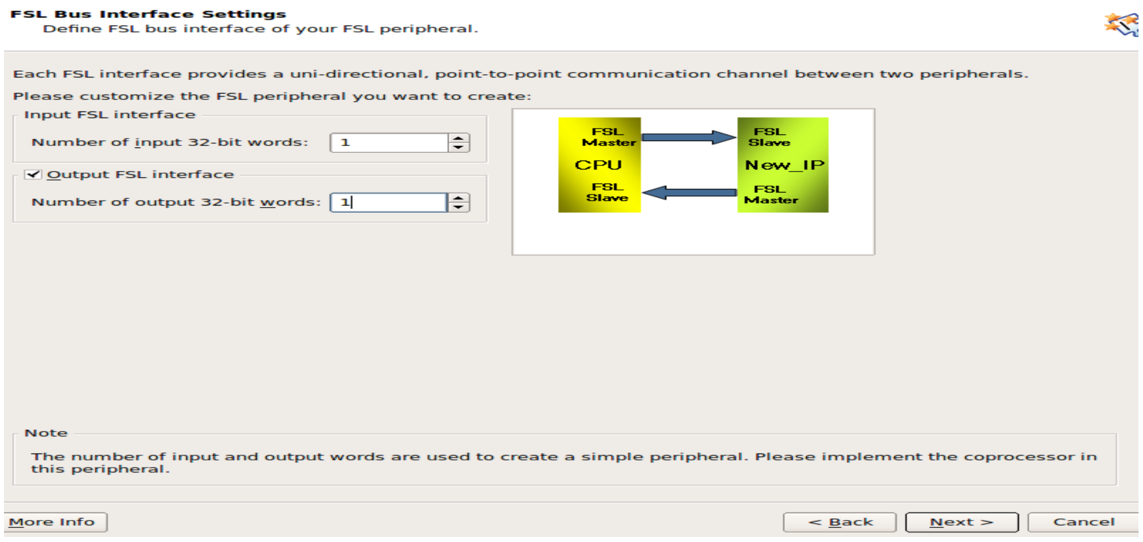
Major revision: 1   Minor revision: 00   Hardware/Software compatibility revision: a

Description:

Logical library name: sobelcop_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

Figure 3.3: peripheral name



FSL Bus Interface Settings
Define FSL bus interface of your FSL peripheral.

Each FSL interface provides a uni-directional, point-to-point communication channel between two peripherals.

Please customize the FSL peripheral you want to create:

Input FSL interface

Number of input 32-bit words: 1

[✓] Output FSL interface

Number of output 32-bit words: 1

Note

The number of input and output words are used to create a simple peripheral. Please implement the coprocessor in this peripheral.

Figure 3.4: fsl settings

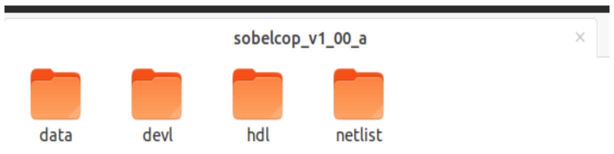


Figure 3.5: sobelcop directory

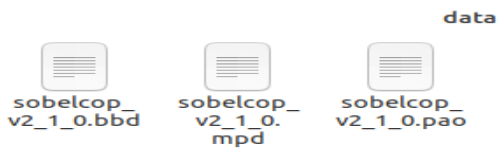


Figure 3.6: data

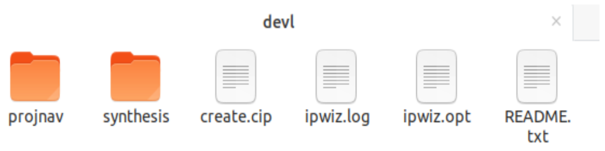


Figure 3.7: devl

- .bbd (Black Box Definition) . It include all the .ngc files in comma separated format
- .mpd(microprocessor peripheral definition).It include all the port definitions.
- .pao(peripheral analyze order).It includes information of all the libraries that you include in your code.

- The hdl directory , you will find the verilog and vhdl codes. The template for sobelcop will be present in the verilog folder as we had selected language as verilog . You need to edit this code in order to suite your purpose.
- Once you have made the necessary edits in the above mentioned files , you may proceed to the xpsgui.

3.4.3 Adding IP core to your design

- Open the xpsgui .
- In the IP catalog , you will find your user defined ip cores as shown in fig.

verilog
vhdl

Figure 3.8: hdl

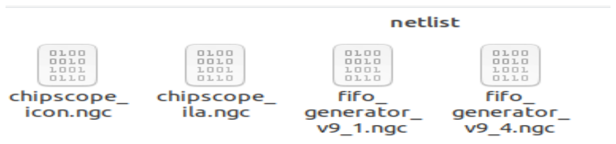
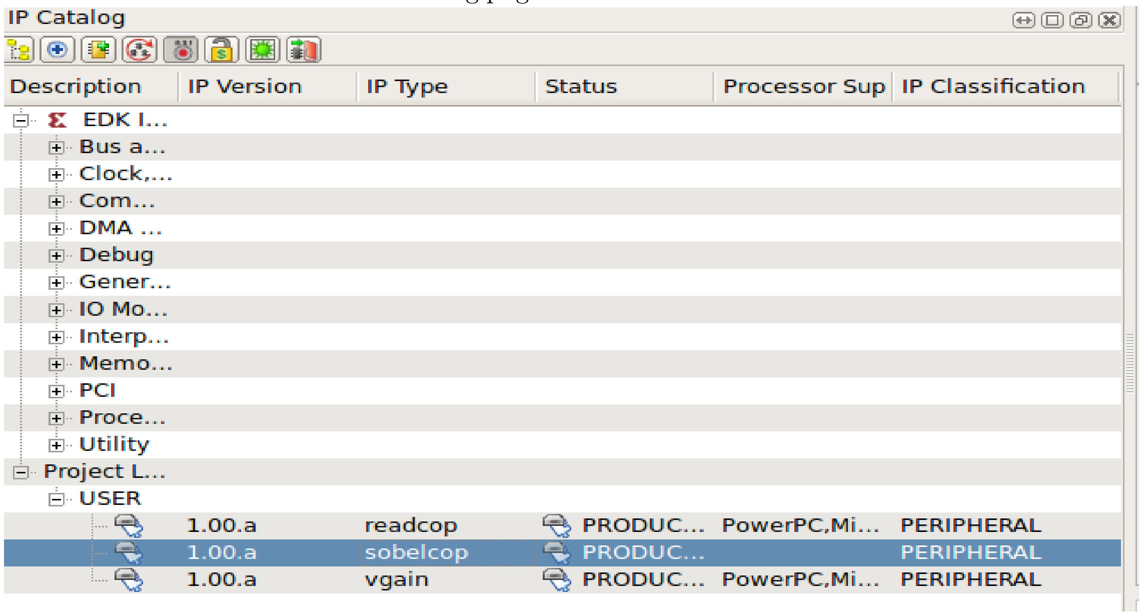


Figure 3.9: netlist

catalog.png



Description	IP Version	IP Type	Status	Processor Sup	IP Classification
EDK I...					
Bus a...					
Clock,...					
Com...					
DMA ...					
Debug					
Gener...					
IO Mo...					
Interp...					
Memo...					
PCI					
Proce...					
Utility					
Project L...					
USER					
	1.00.a	readcop	PRODUC...	PowerPC,Mi...	PERIPHERAL
	1.00.a	sobelcop	PRODUC...		PERIPHERAL
	1.00.a	vgain	PRODUC...	PowerPC,Mi...	PERIPHERAL

Figure 3.10: IP catalog

diagram of ip.png

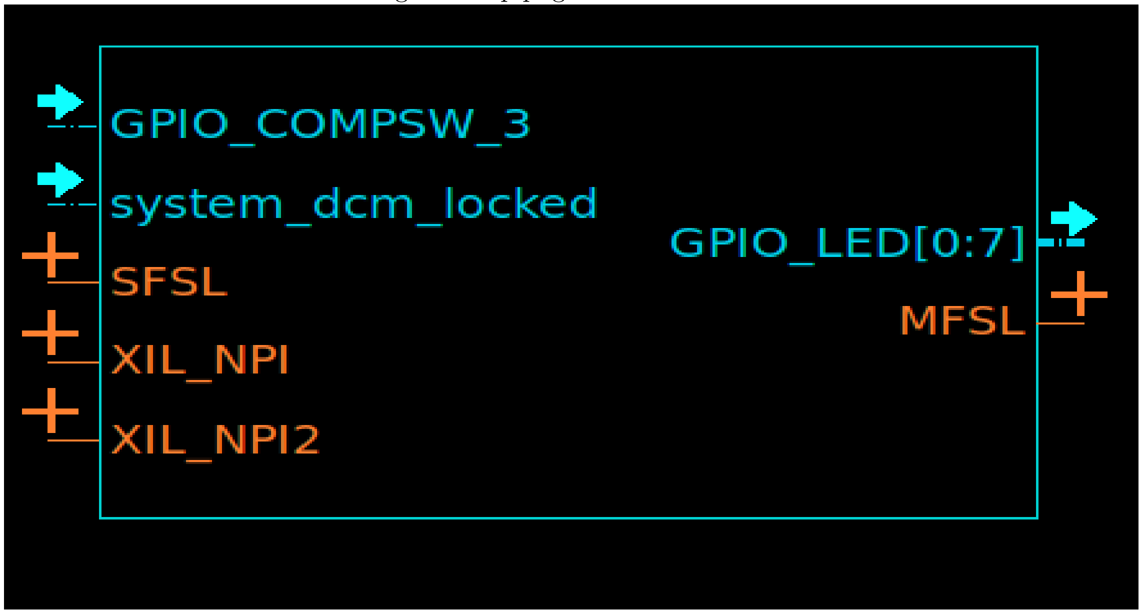


Figure 3.11: block diagram of IP

- Double click on sobelcop and add ip instance to your design. A window will open as shown in fig.
- Confirm that all the ports mentioned in your IP are present in the block diagram . If not go back to your code and make edits.

3.4.4 Connect your IP with other components of design

- In the xpsgui system assembly view , you will find 3 tabs
  - bus interfaces
  - ports
  - addresses
- Explore all these tabs before making connections.
- In the addresses tab observe the base address and high address of ddr2 sdram . According you need to put address in your code whenever you are reading and writing to dram.
- In the ports tab , you can make connections. Alternative option is to make changes in the .mhs file (microprocessor hardware specification) in the project tab .

- Once the connections are done , you can verify them by viewing the Graphical design view.

3.4.5 Your design is ready !!!

Click on Export design in the Navigators toolbar. In the window that appears next , click on Export and Launch SDK.

Dont worry if your design isnt compiled in the first attempt . It took us almost a day and a half to resolve all the errors . If you face any kind of errors, search on stack exchange or xilinx forums.

3.5 From XPS to SDK

Once your compilation process is completed , The sdk window will open. As we arent running any C/C++ application , we can directly proceed with the programming of fpga. Go to Xilinx tools - Program FPGA. Make sure you have connected the Jtag cable to fpga !!!

Enjoy! Your project has been uploaded to the board.