General associative memory based on incremental neural network

Appendix

Table of Contents

[Chapter 1 2](#_Toc453669194)

[Chapter 2 19](#_Toc453669197)

[Chapter 3 19](#_Toc453669198)

[Chapter 4 19](#_Toc453669199)

# Chapter 1

#  -> **Memory Layer**

# --Memory Layer

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

use work.mem\_structure.all;

entity memory\_layer is

port(Reg\_x:in std\_logic\_vector (image\_vector\_len-1 downto 0);

 Reg\_cx:in std\_logic\_vector (7 downto 0);

 assoc\_learning\_done,learning\_finished,

 clk,reset:in std\_logic;

 assoc\_learning\_start:out std\_logic

 );

end memory\_layer;

architecture mem\_layer of memory\_layer is

--controller signlsn

signal comparator: std\_logic\_vector(1 downto 0) ;

signal ld\_counter,en\_counter,en\_node\_counter,

 en\_connections:std\_logic;

signal M1\_select,M2\_select,

 M3\_select,M4\_select,M5\_select,

 M6\_select,DM\_select:std\_logic\_vector(1 downto 0);

signal x,c,w,T,M,rd\_wr:std\_logic;

--datapath signals

--i/p to memory module

signal M2\_w\_i:std\_logic\_vector (image\_vector\_len-1 downto 0);

signal M1\_node\_i,M3\_Th\_i,M4\_M\_i:std\_logic\_vector(7 downto 0);

--i/p to node\_counter

 --signal Reg\_cx\_node\_counter:std\_logic\_vector(7 downto 0);

--i/p to update Ws1 Ws2 Ths1

 --from Reg\_ws1,Reg\_ws2,Reg\_ws3,controller

--i/p to upcounter

 --from controller, and "000...00"

--i/p to addition

signal M\_o\_adder: std\_logic\_vector(7 downto 0);

--i/p to ED calculator

signal DM\_ED\_calc:std\_logic\_vector (image\_vector\_len-1 downto 0);

--i/p to 2min finder

signal ED\_2min:std\_logic\_vector (image\_vector\_len-1 downto 0);

signal DM\_2min:std\_logic\_vector(7 downto 0);

--i/p to comparator

signal M5\_Comparator,M6\_comparator:std\_logic\_vector(7 downto 0);

--i/p to M1

signal node\_counter\_M1,upcounter\_M1: std\_logic\_vector(7 downto 0);

--i/p to M2

signal ws1\_M2,ws2\_M2:std\_logic\_vector (image\_vector\_len-1 downto 0);

--i/p to M3

signal Ths1\_M3: std\_logic\_vector(7 downto 0); --from Reg Ms1

--i/p to M4

 --from Reg Ms1

--i/p to M5

signal class\_o\_M5,upcounter\_M5: std\_logic\_vector(7 downto 0);

--i/p to M6

 --signal Reg\_cx\_comparator:std\_logic\_vector(7 downto 0);

--i/p to DM

signal w\_o\_DM:std\_logic\_vector (image\_vector\_len-1 downto 0);

-- signal ,Reg\_x\_ED\_calc:std\_logic\_vector (image\_vector\_len-1 downto o);

-- signal Reg\_cx\_class\_i:std\_logic\_vector(7 downto 0);

-- signal node\_counter\_M1, upcounter\_M1: std\_logic\_vector(7 downto 0);

-- signal ws1\_M2,ws2\_M2:std\_logic\_vector (image\_vector\_len-1 downto o);

-- signal Ths1\_M3,Reg\_Ms1\_M3: std\_logic\_vector(7 downto 0); --from Reg Ms1

-- signal Reg\_min1\_M4:std\_logic\_vector(7 downto 0);

-- signal class\_o\_M5,upcounter\_M5,Reg\_Ths1\_M5: std\_logic\_vector(7 downto 0);

-- signal Reg\_cx\_M6,Reg\_min1:std\_logic\_vector(7 downto 0);

-- signal DM\_ED\_calc,DM\_Reg\_ws1, DM\_Reg\_ws2,w\_o\_DM M2\_out\_W\_i:std\_logic\_vector (image\_vector\_len-1 downto o);

-- signal M1\_out\_node\_i,M3\_out\_Th\_i,M4\_out\_M\_i:std\_logic\_vector(7 downto 0);

--Registers

signal Reg\_node\_min1,Reg\_node\_min2,Reg\_ED\_min1,Reg\_ED\_min2: std\_logic\_vector(7 downto 0);

signal Reg\_node\_max,Reg\_Ths1,Reg\_Ms1: std\_logic\_vector(7 downto 0);

--signal Reg\_x,Reg\_ws1,Reg\_ws2:std\_logic\_vector (image\_vector\_len-1 downto o);

signal Reg\_ws1,Reg\_ws2:std\_logic\_vector (image\_vector\_len-1 downto 0);

signal NO\_OP:std\_logic\_vector(7 downto 0);

begin

--Reg\_x<=x;

--Reg\_cx<=c;

node\_counter: entity work.node\_counter

 port map(en=>en\_node\_counter ,class\_i=>Reg\_cx,node\_o=>node\_counter\_M1);

calc\_ws1\_ws2\_ths1:entity work.calculate\_ws1\_ws2\_ths1

 port map(x\_in=>Reg\_x,ws1\_in=>Reg\_ws1,

 ws2\_in=>Reg\_ws2,Ths1\_in=>Reg\_Ths1,

 Ms1\_in=>Reg\_Ms1,ED\_min1\_in=>Reg\_ED\_min1,

 ws1\_out=>ws1\_M2,ws2\_out=>ws2\_M2,

 Ths1\_out=>Ths1\_M3);

up\_counter:entity work.UpCounter

 GENERIC MAP (size => 6)

 port map(clock=>clk,load=>ld\_counter,

 enable=>en\_counter,din=>"00000000",

 dout=>upcounter\_M1);

ED\_calc:entity work.euclidean\_distance

 generic map(size=>image\_vector\_len)

 port map(input1=>Reg\_x,input2=>DM\_ED\_calc,ed=>ED\_2min);

min\_finder:entity work.min2

 generic map(n=>8)

 port map(vector1=>Reg\_ED\_min1,vector2=>Reg\_ED\_min2,

 min\_node1=>Reg\_node\_min1,min\_node2=>Reg\_node\_min2,

 in\_node=>upcounter\_M1,in\_vector=>ED\_2min);

comp: entity work.comparator

 generic map(size => 8)

 port map(din1=>M5\_Comparator, din2=>M6\_comparator ,comp\_out=>comparator);

add\_Ms1: entity work.adder

 generic map(size=>8)

 port map(M\_o\_adder,"00000001", Reg\_Ms1);

M1: entity work.mux4X1

 generic map(size => 8)

 port map(din0=>node\_counter\_M1,din1=>upcounter\_M1,

 din2=>Reg\_node\_min1,din3=>Reg\_node\_min2,

 select\_in=>M1\_select,dout=>M1\_node\_i);

M2: entity work.mux4X1

 generic map(size => 8)

 port map(din0=>Reg\_x,din1=>ws1\_M2,

 din2=>ws2\_M2,din3=>NO\_OP,

 select\_in=>M2\_select,dout=>M2\_w\_i);

M3: entity work.mux4X1

 generic map(size => 8)

 port map(din0=>NO\_OP,din1=>Reg\_ED\_min1,

 din2=>Ths1\_M3,din3=>NO\_OP,

 select\_in=>M3\_select,dout=>M3\_Th\_i);

M4: entity work.mux4X1

 generic map(size => 8)

 port map(din0=>"00000001",din1=>Reg\_Ms1,

 din2=>NO\_OP,din3=>NO\_OP,

 select\_in=>M4\_select,dout=>M4\_M\_i);

M5: entity work.mux4X1

 generic map(size => 8)

 port map(din0=>class\_o\_M5,din1=>upcounter\_M5,

 din2=>Reg\_Ths1,din3=>NO\_OP,

 select\_in=>M5\_select,dout=>M5\_Comparator);

M6: entity work.mux4X1

 generic map(size => 8)

 port map(din0=>Reg\_cx,din1=>Reg\_node\_max,

 din2=>Reg\_ED\_min1,din3=>NO\_OP,

 select\_in=>M6\_select,dout=>M6\_comparator);

DM: entity work.demux1X4

 generic map(size => 8)

 port map(din=>w\_o\_DM,select\_in=>DM\_select,

 dout0=>DM\_ED\_calc,dout1=>Reg\_ws1,

 dout2=>Reg\_ws2);

Controller: entity work.controller\_mem\_layer

 port map(clock=>clk,reset=>reset,learning\_finished=>learning\_finished,

 assoc\_learning\_done=>assoc\_learning\_done ,

 comparator1=>comparator,ld\_upcounter1=>ld\_counter,

 en\_upcounter1=>en\_counter,

 en\_node\_counter=>en\_node\_counter,

 en\_connection=>en\_connections,

 x=>x,c=>c,w=>w,T=>T,M=>M,rd\_wr=>rd\_wr,

 assoc\_learning\_start=>assoc\_learning\_start,

 select\_mux1=>M1\_select,select\_mux2=>M2\_select,

 select\_mux3=>M3\_select,

 select\_mux4=>M4\_select,select\_mux5=>M5\_select,

 select\_mux6=>M6\_select,select\_demux=> DM\_select

 );

mem: entity work.memory

 port map(x\_i=>Reg\_x, class\_i=>Reg\_cx, node\_i=>M1\_node\_i,

 w\_i=>M2\_w\_i, Th\_i=>M3\_Th\_i ,M\_i=>M4\_M\_i,

 x=>x,c=>c,w=>w,T=>T, M=>M ,

 class\_o=>class\_o\_M5,

 w\_o=>w\_o\_DM, Th\_o=>Reg\_Ths1,M\_o=>M\_o\_adder,

 rd\_wr=>rd\_wr

 );

connection\_mem:entity work.connection\_memory

 generic map (node\_count=>20)

 port map(node1\_i=>to\_integer(unsigned(Reg\_node\_min1)),

 node2\_i=>to\_integer(unsigned(Reg\_node\_min2)),

 class\_i=>to\_integer(unsigned(Reg\_cx)),

 en=>to\_integer(unsigned(en\_connections)),

 learning\_done=>to\_integer(unsigned(learning\_done))

 );

end mem\_layer;

* Calculate\_ws1\_ws2\_ths1

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

USE ieee.numeric\_std.ALL;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use work.mem\_structure.all;

entity calculate\_ws1\_ws2\_ths1 is

port(x\_in,ws1\_in,ws2\_in:in std\_logic\_vector (image\_vector\_len-1 downto 0);

 Ths1\_in,Ms1\_in,ED\_min1\_in:in std\_logic\_vector (7 downto 0);

 --calc\_ws1\_ws2\_ths1\_start:in std\_logic;

 --calc\_ws1\_ws2\_ths1\_done:out std\_logic;

 ws1\_out,ws2\_out:out std\_logic\_vector (image\_vector\_len-1 downto 0);

 Ths1\_out:out std\_logic\_vector(7 downto 0)

 );

end calculate\_ws1\_ws2\_ths1;

architecture Behavioral of calculate\_ws1\_ws2\_ths1 is

signal Ths1\_add\_div:std\_logic\_vector (7 downto 0);

signal Ms1\_100:std\_logic\_vector (7 downto 0);

begin

ms1\_mul: entity work.mul

 generic map(length\_1=>8,length\_2=>8,result\_width=>8)

 port map(Ms1\_in,"01100100",Ms1\_100);

Ws1\_gen: for i in 0 to image\_dimension generate

Ws1: entity work.calculate\_8bit

 port map(x\_in( (8\*(i+1))-1 downto (8\*i) ),ws1\_in( (8\*(i+1))-1 downto (8\*i) ),Ms1\_in,ws1\_out( (8\*(i+1))-1 downto (8\*i) ));

 end generate Ws1\_gen;

Ws2\_gen: for i in 0 to image\_dimension generate

Ws2:entity work.calculate\_8bit

 port map(x\_in( (8\*(i+1))-1 downto (8\*i) ),ws2\_in( (8\*(i+1))-1 downto (8\*i) ),

 Ms1\_100,ws2\_out( (8\*(i+1))-1 downto (8\*i) ) );

 end generate Ws2\_gen;

Ths1\_add: entity work.adder

 generic map(size=>8)

 port map(Ths1\_in,ED\_min1\_in,Ths1\_add\_div);

Ths1: entity work.division

 generic map(n=>8)

 port map(Ths1\_out,Ths1\_add\_div,2);

end Behavioral;

* **Controller**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity controller\_mem\_layer is

port (clock,reset,learning\_finished,assoc\_learning\_done:in std\_logic;

 comparator1:in std\_logic\_vector (1 downto 0);

 ld\_upcounter1,en\_upcounter1,en\_node\_counter,en\_connection: out std\_logic;

 x,c,w,T,M,rd\_wr,assoc\_learning\_start: out std\_logic;

 select\_mux1,select\_mux2,select\_mux3,select\_mux4, select\_mux5,

 select\_mux6, select\_demux: out std\_logic\_vector (1 downto 0)

);

end controller\_mem\_layer;

architecture Behavioral of controller\_mem\_layer is

constant rd:std\_logic :='0';

constant wr:std\_logic:='1';

type states\_T is (idle,waiting\_assoc,new\_input,no\_class,existing\_class,

 read\_MWT,update\_M\_compare\_Th\_ED,greater\_than\_Th,

 less\_than\_Th,update\_Ths1, write\_Ws1\_Ths1,write\_Ws2, Connections);

signal present\_state,next\_state: states\_T;

begin

process(clock)

begin

 if rising\_edge(clock) then

 if reset='1' then

 present\_state<=idle;

 else

 present\_state<=next\_state;

 end if;

 end if;

end process;

process (present\_state)

begin

 case(present\_state) is

 when idle=>

 x <='0' ;

 c <='0' ;

 w <='0' ;

 T <='0';

 M <='0' ;

 rd\_wr <=rd ;

 ld\_upcounter1 <= '1';

 en\_upcounter1 <='0' ;

 en\_node\_counter<= '0';

 select\_mux1 <= "11";

 select\_mux2 <= "11";

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<= '0' ;

 en\_connection<='0';

 when waiting\_assoc=>

 x <='0';

 c <= '0';

 w <='0' ;

 T <= '0';

 M <= '0';

 rd\_wr <= rd;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <='0';

 en\_node\_counter<= '0';

 select\_mux1 <="11" ;

 select\_mux2 <="11" ;

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <="11" ;

 select\_mux6 <= "11";

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<='1' ;

 en\_connection<='0';

 when new\_input=>

 x <= '0' ;

 c <= '1';

 w <= '0';

 T <= '0';

 M <= '0';

 rd\_wr <=rd ;

 ld\_upcounter1 <= '1' ;

 en\_upcounter1 <= '0';

 en\_node\_counter<='0' ;

 select\_mux1 <= "11";

 select\_mux2 <= "11";

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <= "00"; --rd from class\_o; to comparator1

 select\_mux6 <= "00"; --rd from i/p class reg

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<= '0' ;

 en\_connection<='0';

 when no\_class=>

 x <= '1';

 c <= '1';

 w <= '1';

 T <= '1';

 M <= '1';

 rd\_wr <= wr;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <='0' ;

 en\_node\_counter<= '1' ;

 select\_mux1 <= "00" ; --select node\_counter o/p

 select\_mux2 <= "00"; --selects i/p vector x(Reg X)

 select\_mux3 <= "00"; --select 0 vector

 select\_mux4 <= "00"; --select 1

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <="11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<= '0' ;

 en\_connection<='0';

 when existing\_class=>

 x <= '0';

 c <= '0';

 w <= '1'; --rd W

 T <= '0';

 M <= '0';

 rd\_wr <= rd;

 ld\_upcounter1 <='0' ;

 en\_upcounter1 <='1' ;

 en\_node\_counter<= '0' ;

 select\_mux1 <= "01"; --select o/p of UpCounter

 select\_mux2 <= "11";

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <= "01"; --select o/p of UpCounter

 select\_mux6 <= "01"; --select node\_max register

 select\_demux <= "00"; --o/p of demux to ED calculator

 --calc\_Ws1\_Ws2\_Ths1<='0';

 --SIGNAL T0 ACTIVATE AND DEACTIVATE ED BLOCK,2 MIN BLOCK

 assoc\_learning\_start<= '0';

 en\_connection<='0';

 when read\_MWT=>

 x <= '0';

 c <= '0';

 w <= '1';

 T <= '1';

 M <= '1';

 rd\_wr <=rd ;

 ld\_upcounter1 <= '1' ;

 en\_upcounter1 <= '0';

 en\_node\_counter<= '0' ;

 select\_mux1 <= "10"; -- Reg min\_node1

 select\_mux2 <= "11";

 select\_mux3 <="11" ;

 select\_mux4 <= "11";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <= "01"; --to Reg Ws1

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<='0' ;

 en\_connection<='0';

 when update\_M\_compare\_Th\_ED=>

 x <='0';

 c <='0';

 w <='0';

 T <='0';

 M <='1';

 rd\_wr <= wr;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <= '0';

 en\_node\_counter<= '0';

 select\_mux1 <= "01"; --reg min\_node1

 select\_mux2 <= "11";

 select\_mux3 <= "11";

 select\_mux4 <= "01"; --Reg Ms1(updated)

 select\_mux5 <= "10"; --Reg Ths1

 select\_mux6 <= "10"; --Reg ED\_min1

 select\_demux <="11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<='0' ;

 en\_connection<='0';

 when greater\_than\_Th=>

 x <='1';

 c <='1';

 w <='1';

 T <='1';

 M <='1';

 rd\_wr <=wr;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <='0' ;

 en\_node\_counter<='1' ;

 select\_mux1 <= "00"; --node\_counter

 select\_mux2 <= "00";

 select\_mux3 <= "01";

 select\_mux4 <= "00";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux<= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<= '0';

 en\_connection<='0';

 when less\_than\_Th=> --rd Ws2

 x <= '0';

 c <= '0';

 w <= '1';

 T <= '0';

 M <= '0';

 rd\_wr <= rd;

 ld\_upcounter1 <= '1';

 en\_upcounter1 <= '0';

 en\_node\_counter<='0';

 select\_mux1 <= "11"; --reg min\_node2

 select\_mux2 <= "11";

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <= "10"; --to Reg Ws2

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<= '0' ;

 en\_connection<='0';

 when update\_Ths1=> -- >Th state

 x <= '0';

 c <= '0';

 w <= '0';

 T <= '1';

 M <= '0';

 rd\_wr <=wr ;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <='0';

 en\_node\_counter<= '0';

 select\_mux1 <= "10"; --Reg min\_node1

 select\_mux2 <= "11";

 select\_mux3 <= "01"; --from Reg ED\_min1

 select\_mux4 <= "11";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<= '0';

 en\_connection<='0';

 -- when calculate\_Ws1\_Ws2\_Ths1=>

 -- x <='0' ;

 -- c <='0';

 -- w <='0' ;

 -- T <='0' ;

 -- M <='0' ;

 -- rd\_wr <= rd;

 -- ld\_upcounter1 <='1' ;

 -- en\_upcounter1 <='0' ;

 -- en\_node\_counter<='0' ;

 -- select\_mux1 <= "11";

 -- select\_mux2 <= "11";

 -- select\_mux3 <= "11";

 -- select\_mux4 <= "11";

 -- select\_mux5 <= "11";

 -- select\_mux6 <= "11";

 -- select\_demux <="11";

 -- calc\_Ws1\_Ws2\_Ths1<='1';

 -- assoc\_learning\_start<='0' ;

 -- en\_connection<='0';

 when write\_Ws1\_Ths1=>

 x <='0' ;

 c <='0' ;

 w <='1' ;

 T <='1' ;

 M <='0' ;

 rd\_wr <= wr;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <= '0';

 en\_node\_counter<= '0';

 select\_mux1 <= "10"; --Reg min\_node1

 select\_mux2 <= "01"; --o/p of calc\_Ws1\_ws2\_Ths1

 select\_mux3 <= "10";

 select\_mux4 <= "11";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<='0' ;

 en\_connection<='0';

 when write\_Ws2=>

 x <='0' ;

 c <='0' ;

 w <='1' ;

 T <='0' ;

 M <='0' ;

 rd\_wr <= wr;

 ld\_upcounter1 <='1' ;

 en\_upcounter1 <= '0';

 en\_node\_counter<= '0';

 select\_mux1 <= "11"; --Reg min\_node2

 select\_mux2 <= "10"; --o/p of calc\_Ws1\_ws2\_Ths1

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <= "11";

 select\_mux6 <= "11";

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<='0' ;

 en\_connection<='0';

 when Connections=>

 x <='0' ;

 c <='0';

 w <='0' ;

 T <='0';

 M <='0';

 rd\_wr <= rd ;

 ld\_upcounter1 <='1';

 en\_upcounter1 <='0' ;

 en\_node\_counter<='0' ;

 select\_mux1 <="11" ;

 select\_mux2 <="11" ;

 select\_mux3 <= "11";

 select\_mux4 <= "11";

 select\_mux5 <="11" ;

 select\_mux6 <= "11";

 select\_demux <= "11";

 --calc\_Ws1\_Ws2\_Ths1<='0';

 assoc\_learning\_start<='0' ;

 en\_connection<='1';

 end case;

end process;

process(present\_state,comparator1) --add al i/ps to controller

begin

 case present\_state is

 when idle=>

 if learning\_finished='1' then

 next\_state<=idle ;

 else

 next\_state<=new\_input;

 end if;

 when waiting\_assoc=>

 if assoc\_learning\_done='1' then

 next\_state<=idle;

 else

 next\_state<=waiting\_assoc;

 end if;

 when new\_input=>

 if comparator1="00" then --equal

 next\_state<=existing\_class;

 else

 next\_state<=no\_class;

 end if;

 when no\_class=>

 next\_state <= new\_input;

 when existing\_class=> --verify if ED and 2min calc for 1 i/p is done in 1 cycle

 if comparator1= "00" then

 next\_state<=read\_MWT ;

 else

 next\_state<=existing\_class;

 end if;

 when read\_MWT=>

 next\_state<=update\_M\_compare\_Th\_ED;

 when update\_M\_compare\_Th\_ED=>

 if comparator1= "10" then --if >Ths1

 next\_state<=greater\_than\_Th ;

 else

 next\_state<=less\_than\_Th;

 end if;

 when greater\_than\_Th=>

 next\_state<=update\_Ths1;

 when less\_than\_Th=>

 next\_state<=write\_Ws1\_Ths1;

 when update\_Ths1=>

 next\_state<=connections;

 -- when calculate\_Ws1\_Ws2\_Ths1=>

 -- next\_state<=write\_Ws1\_Ths1;

 when write\_Ws1\_Ths1=>

 next\_state<=write\_Ws2;

 when write\_Ws2=>

 next\_state<= connections;

 when Connections=>

 next\_state<=waiting\_assoc;

 end case;

end process;

end Behavioral;

* **Connection\_mem**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.Numeric\_Std.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use work.mem\_structure.all;

entity connection\_memory is

generic (node\_count:integer:=20) ;

port (node1\_i,

 node2\_i,

 class\_i:in integer;

 en,

 learning\_done:in std\_logic

 );

end connection\_memory;

architecture Behavioral of connection\_memory is

signal connection\_mem:connection\_mem\_T ;

begin

process(en,learning\_done) is

begin

 if rising\_edge(en) then

 if (node1\_i < node2\_i) then

 connection\_mem(class\_i).node(node1\_i).connected\_node(node2\_i).connection\_presence<='1';

 connection\_mem(class\_i).node(node1\_i).connected\_node(node2\_i).age<=0;

 else

 connection\_mem(class\_i).node(node2\_i).connected\_node(node1\_i).connection\_presence<='1';

 connection\_mem(class\_i).node(node2\_i).connected\_node(node1\_i).age<=0;

 end if;

 for i in 0 to node1\_i-1 loop

 if (connection\_mem(class\_i).node(i).connected\_node(node1\_i).connection\_presence='1') then

 connection\_mem(class\_i).node(i).connected\_node(node1\_i).age <= connection\_mem(class\_i).node(i).connected\_node(node1\_i).age+1;

 end if;

 end loop;

 for i in node1\_i+1 to node\_count loop

 if (connection\_mem(class\_i).node(node1\_i).connected\_node(i).connection\_presence='1') then

 connection\_mem(class\_i).node(node1\_i).connected\_node(node2\_i).age <= connection\_mem(class\_i).node(node1\_i).connected\_node(node2\_i).age+1;

 end if;

 end loop;

 end if;

end process;

end Behavioral;

* **Node counter**

entity node\_counter is

port(en:in std\_logic;

 class\_i:in std\_logic\_vector(7 downto 0) ; -- 4+4 classes

 node\_o:out std\_logic\_vector(7 downto 0)

 );

end node\_counter;

architecture Behavioral of node\_counter is

type node\_count\_T is array (7 downto 0) of std\_logic\_vector(7 downto 0);

signal node\_count:node\_count\_T;

begin

process(en)

begin

 if rising\_edge(en) then

 node\_o <= node\_count(to\_integer(unsigned(class\_i)));

 node\_count(to\_integer(unsigned(class\_i)))<= node\_count(to\_integer(unsigned(class\_i)))+ '1' ;

 end if;

end process;

end Behavioral;

* **2 min finder**

entity min2 is

generic (n: integer := 8);-- n for vectors

 -- (m: integer := 8) ); -- m for nodes

port( vector1 : out std\_logic\_vector(n-1 downto 0) ; --Min 1

 vector2 : out std\_logic\_vector(n-1 downto 0) ; --Min 2

 min\_node1 : out std\_logic\_vector(n-1 downto 0) ; -- node Min1

 min\_node2 : out std\_logic\_vector(n-1 downto 0) ; -- node Min2

 in\_node : in std\_logic\_vector(n-1 downto 0);

 in\_vector : in std\_logic\_vector(n-1 downto 0)

);

end min2;

--architecture of entity

architecture Behavioral of min2 is

signal temp1: integer :=2\*\*n;

signal temp2: integer :=2\*\*n;

signal temp3: integer :=2\*\*n;

begin

process(in\_vector)

begin

temp3 <= conv\_integer(in\_vector);

if (temp3 < temp1 ) then --checking whether num1 is greater than num2

vector1 <= in\_vector;

temp1 <= temp3;

min\_node1 <= in\_node;

elsif (temp3 < temp2) then --checking whether num1 is greater than num2

vector2 <= in\_vector;

temp2 <= temp3;

min\_node2 <= in\_node;

else

report "Error in Logic of Min2" severity warning;

end if;

end process; -- process ends with a 'end process' statement

end Behavioral;

* **Memory**

use work.mem\_structure.all;

entity memory is

 port(x\_i:in std\_logic\_vector(511 downto 0) ; -- 8\*8\*8 - vector

 class\_i:in std\_logic\_vector(7 downto 0) ; -- 4+4 classes

 node\_i:in std\_logic\_vector(7 downto 0) ; -- 20 nodes

 w\_i:in std\_logic\_vector(7 downto 0) ; -- ~X

 Th\_i:in std\_logic\_vector(7 downto 0) ; --

 M\_i:in std\_logic\_vector(7 downto 0) ;

 --E\_i:in std\_logic\_vector(7 downto 0) ;

 --I\_i:in std\_logic\_vector(7 downto 0) ;

 x:in std\_logic;

 c:in std\_logic;

 w:in std\_logic;

 T:in std\_logic;

 M:in std\_logic;

 --E:in std\_logic;

 --I:in std\_logic;

 x\_o:out std\_logic\_vector(512 downto 0) ;

 class\_o:out std\_logic\_vector(7 downto 0) ; --not required??

 node\_o:out std\_logic\_vector(7 downto 0) ;

 w\_o:out std\_logic\_vector(7 downto 0) ;

 Th\_o:out std\_logic\_vector(7 downto 0) ;

 M\_o:out std\_logic\_vector(7 downto 0) ;

 -- E\_o:out std\_logic\_vector(7 downto 0) ;

 --I\_o:out std\_logic\_vector(7 downto 0) ;

 rd\_wr:in std\_logic); --0-read, 1-write declare const, read and write

end memory;

architecture Behavioral of memory is

signal mem:memory\_T ; --mem/class

begin

process(rd\_wr,x,c,w,T,M) is begin

if rd\_wr='0' then --read='0'

 if(x='1') then

 x\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).x; --convert to int for addressing

 end if;

 if(c='1') then

 class\_o <= mem(to\_integer(unsigned(class\_i))).class\_name;

 end if;

 --node\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).;

 if(w='1') then

 w\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).w;

 end if;

 if(T='1') then

 Th\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).Th;

 end if;

 if(M='1') then

 M\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).M;

 end if;

 -- if(E='1') then

 -- E\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).E;

 -- end if;

 -- if(I='1') then

 -- I\_o <= mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).I;

 -- end if;

else --write

 -- if state==no class then

 -- mem(to\_integer(unsigned(class\_i))).class\_name <= class\_i;

 -- mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).node\_vector <=x\_i;

 -- mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).w <=w\_i;

 -- mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).Th <=Th\_i;

 -- mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).M <=M\_i;

 -- elsif state==existing\_class then

 -- mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).M <=M\_i;

 -- elsif state==greater\_than\_Th then

 if x='1' then

 mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).x<=x\_i;

 end if;

 if c='1' then

 mem(to\_integer(unsigned(class\_i))).class\_name<=class\_i;

 end if;

 if w='1' then

 mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).w<=w\_i;

 end if;

 if T='1' then

 mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).Th<=Th\_i;

 end if;

 if M='1' then

 mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).M<=M\_i;

 end if;

 -- if I='1' then

 -- mem(to\_integer(unsigned(class\_i))).node(to\_integer(unsigned(node\_i))).I<=I\_i;

 -- end if;

end if;

end process;

end Behavioral;

# Chapter 2

1. Associative\_memory module

------------------------------------------------------------------------------------------------------

-- Filename: associative\_memory.vhd

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an associative memory.

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use work.input\_vector\_type.all;

**entity** mem is port(

 clk: in std\_logic;

 reset: in std\_logic;

 phase: in std\_logic; --phase tells training phase = 0 or recall phase = 1

 rd\_wr: in std\_logic; -- 1 is write and 0 is read

 Cxy: in std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 class\_bit\_out: in std\_logic;

 wb: in vector\_input;

 mx: in std\_logic\_vector (NODES-1 downto 0);

 wij: in std\_logic\_vector (NODES-1 downto 0);

 cd: in std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 class\_bit\_m: out std\_logic;

 mx\_m: out std\_logic\_vector (NODES-1 downto 0);

 wij\_m: out std\_logic\_vector (NODES-1 downto 0);

 cd\_m: out std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 wd: out vector\_input

);

**end mem;**

**architecture** behavioral of mem is

type state\_type is (reset\_state, read\_learn\_state, read\_recall\_state, write\_state);

constant RD: std\_logic := '0';

constant WR: std\_logic := '1';

signal associative\_memory, associative\_memory\_next:classes;

signal state\_reg, state\_next: state\_type;

signal class\_bit\_m\_next, class\_bit\_out\_next: std\_logic;

signal mx\_m\_next: std\_logic\_vector (NODES-1 downto 0);

signal wij\_m\_next: std\_logic\_vector (NODES-1 downto 0);

signal cd\_m\_next: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

signal wd\_next, wb\_next: vector\_input := (others=> (others=>'0'));

signal mx\_next: std\_logic\_vector (NODES-1 downto 0);

signal wij\_next: std\_logic\_vector (NODES-1 downto 0);

signal cd\_next: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

begin

 **process**(clk, reset)

 begin

 -- control path: state register

 if(reset = '0' and phase = '1')then

 state\_reg <= reset\_state;

 elsif(clk'event and clk = '1')then

 state\_reg <= state\_next;

 end if;

 **end process;**

 -- control path: next-state/output logic

 **process**(rd\_wr, phase, state\_reg)

 begin

 if(rd\_wr = RD and phase = '1') then

 state\_next <= read\_learn\_state;

 elsif(rd\_wr = RD and phase = '0') then

 state\_next <= read\_recall\_state;

 elsif(rd\_wr = '1' and phase = '1') then

 state\_next <= write\_state;

 elsif(rd\_wr = '1' and phase = '1') then

 state\_next <= read\_recall\_state;

 end if;

 **end process;**

-- data path: data register

 **process**(clk, reset, phase)

 begin

 if(reset = '0' and phase = '1')then

 associative\_memory\_next <= zero\_memory;

 elsif(clk'event and clk = '1')then

 --outputs

 class\_bit\_m <= class\_bit\_m\_next;

 mx\_m <= mx\_m\_next;

 wij\_m <= wij\_m\_next;

 cd\_m <= cd\_m\_next;

 wd <= wd\_next;

 --inputs;

 class\_bit\_out\_next <= class\_bit\_out;

 wb\_next <= wb;

 mx\_next <= mx;

 wij\_next <= wij;

 cd\_next <= cd;

 end if;

 **end process;**

-- data path: routing multiplexer

 **process** (Cxy, cd, mx, class\_bit\_out\_next, wb\_next, mx\_next, wij\_next, cd\_next, rd\_wr, phase, reset, associative\_memory\_next)

 begin

 report "message";

 if(reset = '0' and phase = '1')then

 associative\_memory <= associative\_memory\_next;

 elsif(rd\_wr = RD and phase = '1')then

 class\_bit\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).class\_bit;--<= associative\_memory;

 mx\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).associative\_index;

 wij\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).arrow\_weight\_cd(to\_integer(unsigned(cd)));

 cd\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).response\_class(to\_integer(unsigned(mx)));

 wd\_next <= associative\_memory(to\_integer(unsigned(Cxy))).weight;

 elsif(rd\_wr = RD and phase = '0')then

 class\_bit\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).class\_bit;--<= associative\_memory;

 mx\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).associative\_index;

 wij\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).arrow\_weight(to\_integer(unsigned(mx)));

 cd\_m\_next <= associative\_memory(to\_integer(unsigned(Cxy))).response\_class(to\_integer(unsigned(mx)));

 wd\_next <= associative\_memory(to\_integer(unsigned(Cxy))).weight;

 elsif(rd\_wr = WR and phase = '1')then

 associative\_memory(to\_integer(unsigned(Cxy))).class\_bit <= class\_bit\_out\_next;

 associative\_memory(to\_integer(unsigned(Cxy))).weight <= wb\_next;

 associative\_memory(to\_integer(unsigned(Cxy))).associative\_index <= mx\_next;

 associative\_memory(to\_integer(unsigned(Cxy))).arrow\_weight\_cd(to\_integer(unsigned(cd))) <= wij\_next;

 associative\_memory(to\_integer(unsigned(Cxy))).arrow\_weight(to\_integer(unsigned(mx))) <= wij\_next;

 associative\_memory(to\_integer(unsigned(Cxy))).response\_class(to\_integer(unsigned(mx))) <= cd\_next;

 end if;

 **end process;**

 --associative\_memory;

**end behavioral;**

1. Testing

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: memory\_layer.vhd

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an algorithm, learning of the associative memory. This implments the

-- association between key class and response vectors through nodes in associative memory.

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

use work.input\_vector\_type.all; --package

entity memory\_layer is port(

clk : in std\_logic;

node\_index: in std\_logic\_vector (NODES-1 downto 0);

Mp\_m: out std\_logic\_vector (NODES-1 downto 0);

wb\_m: out vector\_input

 );

end memory\_layer;

architecture behavior of memory\_layer is

------------------------------------------------------------------------0

constant constant\_array\_0: vector\_input:= ();

-------------------------------------------------------------------------1

constant constant\_array\_1: vector\_input:= ();

-----------------------------------------------------------------------2

constant constant\_array\_2: vector\_input:= ();

.

.

.

.

-------------------------------------------------------------------------------------63

constant constant\_array\_63: vector\_input:= ();

--soon till 64 arrays

signal constant\_array: test\_mem\_vector;

signal Mp\_memory: Mp\_array;

begin

constant\_array(0) <= constant\_array\_0;

constant\_array(1) <= constant\_array\_1;

constant\_array(2) <= constant\_array\_2;

constant\_array(3) <= constant\_array\_3;

constant\_array(4) <= constant\_array\_4;

constant\_array(5) <= constant\_array\_5;

constant\_array(6) <= constant\_array\_6;

constant\_array(7) <= constant\_array\_7;

constant\_array(8) <= constant\_array\_8;

constant\_array(9) <= constant\_array\_9;

constant\_array(10) <= constant\_array\_10;

constant\_array(11) <= constant\_array\_11;

constant\_array(12) <= constant\_array\_12;

constant\_array(13) <= constant\_array\_13;

constant\_array(14) <= constant\_array\_14;

constant\_array(15) <= constant\_array\_15;

constant\_array(16) <= constant\_array\_16;

constant\_array(17) <= constant\_array\_17;

constant\_array(18) <= constant\_array\_18;

constant\_array(19) <= constant\_array\_19;

constant\_array(20) <= constant\_array\_20;

constant\_array(21) <= constant\_array\_21;

constant\_array(22) <= constant\_array\_22;

constant\_array(23) <= constant\_array\_23;

constant\_array(24) <= constant\_array\_24;

constant\_array(25) <= constant\_array\_25;

constant\_array(26) <= constant\_array\_26;

constant\_array(27) <= constant\_array\_27;

constant\_array(28) <= constant\_array\_28;

constant\_array(29) <= constant\_array\_29;

constant\_array(30) <= constant\_array\_30;

constant\_array(31) <= constant\_array\_31;

constant\_array(32) <= constant\_array\_32;

constant\_array(33) <= constant\_array\_33;

constant\_array(34) <= constant\_array\_34;

constant\_array(35) <= constant\_array\_35;

constant\_array(36) <= constant\_array\_36;

constant\_array(37) <= constant\_array\_37;

constant\_array(38) <= constant\_array\_38;

constant\_array(39) <= constant\_array\_39;

constant\_array(40) <= constant\_array\_40;

constant\_array(41) <= constant\_array\_41;

constant\_array(42) <= constant\_array\_42;

constant\_array(43) <= constant\_array\_43;

constant\_array(44) <= constant\_array\_44;

constant\_array(45) <= constant\_array\_45;

constant\_array(46) <= constant\_array\_46;

constant\_array(47) <= constant\_array\_47;

constant\_array(48) <= constant\_array\_48;

constant\_array(49) <= constant\_array\_49;

constant\_array(50) <= constant\_array\_50;

constant\_array(51) <= constant\_array\_51;

constant\_array(52) <= constant\_array\_52;

constant\_array(53) <= constant\_array\_53;

constant\_array(54) <= constant\_array\_54;

constant\_array(55) <= constant\_array\_55;

constant\_array(56) <= constant\_array\_56;

constant\_array(57) <= constant\_array\_57;

constant\_array(58) <= constant\_array\_58;

constant\_array(59) <= constant\_array\_59;

constant\_array(60) <= constant\_array\_60;

constant\_array(61) <= constant\_array\_61;

constant\_array(62) <= constant\_array\_62;

constant\_array(63) <= constant\_array\_63;

Mp\_memory <= (

"0000",

"0001",

"0001",

"0001",

"0100",

"0001",

"0001",

"0001",

"0010",

"0010",

"0010",

"0010",

"0001",

"0001",

"0010",

"0001",

"0100",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0001",

"0100",

"0011",

"0010",

"0010",

"0001",

"0010",

"0011",

"0010",

"0010",

"0001",

"0001",

"0001",

"0001",

"0001"

);

**process**(clk)

begin

wb\_m <= constant\_array(to\_integer(unsigned(node\_index)));

Mp\_m <= Mp\_memory(to\_integer(unsigned(node\_index)));

**end process;**

**end behavior**;

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: associative\_mem\_controller\_tb.vhd

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an algorithm, learning of the associative memory. This implments the

-- association between key class and response vectors through nodes in associative memory.

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use std.env.all;

use work.input\_vector\_type.all;

entity test\_bench is

end test\_bench;

**architecture** behavior of test\_bench is

signal clk: std\_logic;

signal reset: std\_logic;

signal phase: std\_logic; --phase tells training phase = 0 or recall phase = 1

signal start: std\_logic;

signal class\_in: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

signal class\_bit\_m: std\_logic;

signal input\_weight: vector\_input;

signal mx\_m: std\_logic\_vector (NODES-1 downto 0);

signal Mp\_m: std\_logic\_vector (NODES-1 downto 0);

signal wb\_m: vector\_input;

signal wij\_m: std\_logic\_vector (NODES-1 downto 0);

signal cd\_m: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

signal done: std\_logic;

signal class\_bit\_out: std\_logic;

signal Cxy: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

signal wb: vector\_input;

signal mx: std\_logic\_vector (NODES-1 downto 0);

signal wij: std\_logic\_vector (NODES-1 downto 0);

signal cd: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

signal node\_index: std\_logic\_vector (NODES-1 downto 0);

signal rd\_wr: std\_logic;

signal wd: vector\_input;

signal index: std\_logic\_vector(CLASS\_SIZE+NODES-1 downto 0);

begin

**DUT**: entity work.asm

port map(

 clk => clk,

 reset => reset,

 phase => phase, --phase tells training phase = 0 or recall phase = 1

 start => start,

 class\_in => class\_in,

 class\_bit\_m => class\_bit\_m,

 input\_weight => input\_weight,

 mx\_m => mx\_m,

 Mp\_m => Mp\_m,

 wb\_m => wb\_m,

 wij\_m => wij\_m,

 done => done,

 class\_bit\_out => class\_bit\_out,

 Cxy => Cxy,

 wb => wb,

 mx => mx,

 wij => wij,

 cd => cd,

 node\_index => node\_index,

 rd\_wr => rd\_wr

 );

**AS\_MEM**:entity work.mem --associate memory

port map(

 clk => clk,

 reset => reset,

 phase => phase, --phase tells training phase = 0 or recall phase = 1

 rd\_wr => rd\_wr,

 Cxy => Cxy,

 class\_bit\_out => class\_bit\_out,

 wb => wb,

 mx => mx,

 wij => wij,

 cd => cd,

 class\_bit\_m => class\_bit\_m,

 mx\_m => mx\_m,

 wij\_m => wij\_m,

 cd\_m => cd\_m,

 wd => wd

 );

**MEM**:entity work.memory\_layer --memory layer

port map(

clk => clk,

node\_index => node\_index,

Mp\_m => Mp\_m,

wb\_m => wb\_m

);

**clock**: process

begin

 clk <= '0';

 wait for CLOCK\_PERIOD/2 ;

 clk <= '1';

 wait for CLOCK\_PERIOD/2;

end process;

**stimuli**: process

begin

reset <= '0';

phase <= '1';

start <= '0';

class\_in <= "01";

input\_weight <=

(

"00000000",

"00000000",

"00000001",

"00000001",

"00000001",

"00000010",

"00001100",

"00001110",

"00010000",

"00010010",

"00010100",

"00010110",

"00011000",

"00011010",

"11111111",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000"

);

wait for 50 ns;

reset <= '1';

wait for 30 ns;

start <= '1';

wait for 20 ns;

start <= '0';

wait for 100 ns;

class\_in <= "10";

input\_weight <=

(

"00000000",

"00000000",

"00000001",

"00000001",

"00000001",

"00000010",

"00001010",

"00001100",

"00001110",

"00010000",

"00010010",

"00010100",

"00010110",

"00011000",

"11111111",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000"

);

wait for 20 ns;

start<= '1';

wait for 20 ns;

start <= '0';

wait for 100 ns;

--2nd input

class\_in <= "01";

input\_weight <=

(

"00000000",

"00000000",

"00000001",

"00000001",

"00000001",

"00000010",

"00001100",

"00001110",

"00010000",

"00010010",

"00010100",

"00010110",

"00011000",

"00011010",

"11111111",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000"

);

wait for 50 ns;

reset <= '1';

wait for 30 ns;

start <= '1';

wait for 20 ns;

start <= '0';

wait for 1000 ns;

class\_in <= "10";

input\_weight <=

(

"00000000",

"00000000",

"00000001",

"00000001",

"00000001",

"00000010",

"00001010",

"00001100",

"00001110",

"00010000",

"00010010",

"00010100",

"00010110",

"00011000",

"11111111",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000",

"10000000"

);

wait for 20 ns;

start<= '1';

wait for 20 ns;

start <= '0';

wait for 1000 ns;

stop(0);

**end process;**

**end behavior;**

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: associative\_mem\_controller.vhd

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an algorithm, learning of the associative memory. This implments the

-- association between key class and response vectors through nodes in associative memory.

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

package input\_vector\_type is

constant PIXEL\_RESOLUTION : integer := 8; --bits wide

constant IMAGE\_SIZE : integer := 64; --bits wide

constant CLASS\_SIZE : integer := 2; --bits wide

constant NODES : integer := 4;

constant ONE: std\_logic\_vector (NODES-1 downto 0) :="0001";

constant FIFTEEN: std\_logic\_vector (NODES-1 downto 0) := (others=>'1');

constant ZERO: std\_logic := '0';

constant CLOCK\_PERIOD : time := 10 ns;

type vector\_input is array(0 to IMAGE\_SIZE-1) of std\_logic\_vector (PIXEL\_RESOLUTION -1 downto 0);

--type class\_input std\_logic\_vector (CLASS\_SIZE-1 downto 0);

--type nodes std\_logic\_vector (NODES-1 downto 0);

--type arrow\_weight std\_logic\_vector ((CLASS\_SIZE\*NODES)-1 downto 0);

--associated memory related package items

type arrow is array(0 to ((2\*\*NODES)-1)) of std\_logic\_vector (NODES-1 downto 0);

type response is array(0 to ((2\*\*NODES)-1)) of std\_logic\_vector (CLASS\_SIZE-1 downto 0);

type arrow\_class is array(0 to ((2\*\*CLASS\_SIZE)-1)) of std\_logic\_vector (NODES-1 downto 0);

-- class structure

type class is record

 class\_bit: std\_logic;

 weight: vector\_input;

 associative\_index: std\_logic\_vector (NODES-1 downto 0); --mx

 arrow\_weight: arrow;

 response\_class: response;

 arrow\_weight\_cd: arrow\_class;--seperate

end record;

--class array

type classes is array(0 to ((2\*\*CLASS\_SIZE)-1)) of class;

constant const\_class: class:= (

 class\_bit => '0',

 weight => (others=> (others=>'0')),

 associative\_index => (others=>'0'),

 arrow\_weight => (others=> (others=>'0')),

 response\_class => (others=> (others=>'0')),

 arrow\_weight\_cd => (others=> (others=>'0')));

constant zero\_memory: classes:= (others => const\_class);

type test\_mem\_vector is array (0 to ((2\*\*NODES) \* (2\*\*CLASS\_SIZE))-1) of vector\_input;

type Mp\_array is array (0 to ((2\*\*NODES) \* (2\*\*CLASS\_SIZE))-1) of std\_logic\_vector (NODES-1 downto 0);

end package input\_vector\_type;

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: generic\_adder.vhd

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an generic adder

-- source : http://www.alvie.com/zpuino/vhdl4.html

------------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity generic\_adder is

 generic (

 bits: integer

 );

 port (

 A: in std\_logic\_vector(bits-1 downto 0);

 B: in std\_logic\_vector(bits-1 downto 0);

 CI: in std\_logic;

 O: out std\_logic\_vector(bits-1 downto 0);

 CO: out std\_logic

 );

end entity generic\_adder;

architecture behave of generic\_adder is

begin

process(A,B,CI)

 variable sum: integer;

 -- Note: we have one bit more to store carry out value.

 variable sum\_vector: std\_logic\_vector(bits downto 0);

begin

 -- Compute our integral sum, by converting all operands into integers.

 sum := conv\_integer(A) + conv\_integer(B) + conv\_integer(CI);

 -- Now, convert back the integral sum into a std\_logic\_vector, of size bits+1

 sum\_vector := conv\_std\_logic\_vector(sum, bits+1);

 -- Assign outputs

 O <= sum\_vector(bits-1 downto 0);

 CO <= sum\_vector(bits); -- Carry is the most significant bit

end process;

end behave;

-- Reference : http://esd.cs.ucr.edu/labs/adder/adder.html

LIBRARY IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- We declare the 1-bit adder with the inputs and outputs

-- shown inside the port().

-- This will add two bits together(x,y), with a carry in(cin) and

-- output the sum(sum) and a carry out(cout).

entity BIT\_ADDER is

 port( a, b, cin : in STD\_LOGIC;

 sum, cout : out STD\_LOGIC );

end BIT\_ADDER;

-- This describes the functionality of the 1-BIT adder.

architecture BHV of BIT\_ADDER is

begin

 -- Calculate the sum of the 1-BIT adder.

 sum <= (not a and not b and cin) or

 (not a and b and not cin) or

 (a and not b and not cin) or

 (a and b and cin);

 -- Calculates the carry out of the 1-BIT adder.

 cout <= (not a and b and cin) or

 (a and not b and cin) or

 (a and b and not cin) or

 (a and b and cin);

end BHV;

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: 2to1\_mux.vhdl

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an generic 2to1 multiplexer

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

use work.input\_vector\_type.all;

entity mux\_2to1\_Wb is

 Port ( SEL : in STD\_LOGIC;

 A : in vector\_input;

 B : in vector\_input;

 X : out vector\_input );

end mux\_2to1\_Wb;

architecture Behavioral of mux\_2to1\_Wb is

begin

 X <= A when (SEL = '1') else B;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

use work.input\_vector\_type.all;

entity mux\_2to1\_Mp is

 Port ( SEL : in std\_logic;

 A : in std\_logic\_vector (NODES-1 downto 0);

 B : in std\_logic\_vector (NODES-1 downto 0);

 X : out std\_logic\_vector (NODES-1 downto 0));

end mux\_2to1\_Mp;

architecture Behavioral of mux\_2to1\_Mp is

begin

 X <= A when (SEL = '1') else B;

end Behavioral;

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: associative\_mem\_controller.vhd

-- Created by: Bharath Reddy Godi, Surendra Maddula, Nikhil Marda

-- Date: May 29, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an algorithm, learning of the associative memory. This implments the

-- association between key class and response vectors through nodes in associative memory.

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use work.input\_vector\_type.all;

entity asm is port(

 clk: in std\_logic;

 reset: in std\_logic;

 phase: in std\_logic; --phase tells training phase = 0 or recall phase = 1

 start: in std\_logic;

 class\_in: in std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 class\_bit\_m: in std\_logic;

 input\_weight: in vector\_input;

 mx\_m: in std\_logic\_vector (NODES-1 downto 0);

 Mp\_m: in std\_logic\_vector (NODES-1 downto 0);

 wb\_m: in vector\_input;

 wij\_m: in std\_logic\_vector (NODES-1 downto 0);

 done: out std\_logic;

 class\_bit\_out: out std\_logic;

 Cxy: out std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 wb: out vector\_input;

 mx: out std\_logic\_vector (NODES-1 downto 0);

 wij: out std\_logic\_vector (NODES-1 downto 0);

 cd: out std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 node\_index: out std\_logic\_vector (NODES-1 downto 0);

 rd\_wr: out std\_logic

 );

end asm;

architecture behavioral of asm is

 type state\_type is (idle, detect, set\_values, sort\_values, cal\_values, response, read\_arrow, arrow, done\_state, phase\_state, reset\_state);

 signal state\_reg, state\_next: state\_type;

 signal class\_in\_next,class\_in\_reg: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 signal wb\_next, wb\_reg, wb\_mux: vector\_input;

 signal mx\_next,mx\_reg, mx\_addition: std\_logic\_vector (NODES-1 downto 0);

 signal Mp\_next,Mp\_reg,Mp\_mux: std\_logic\_vector (NODES-1 downto 0); --Mp value is assumed to be 16

 signal wij\_next, wij\_reg, wij\_addition: std\_logic\_vector (NODES-1 downto 0);

 signal cd\_next, cd\_reg: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 signal cb\_next, cb\_reg: std\_logic\_vector (CLASS\_SIZE-1 downto 0);

 signal bit\_next, bit\_reg: std\_logic;

 signal bit\_addition, CO: std\_logic;

 signal class\_bit\_next, class\_bit\_reg: std\_logic;

 signal Mp\_signal: std\_logic;

 signal node\_next,node\_reg,node\_addition: std\_logic\_vector (NODES-1 downto 0);

begin

 -- control path: state register

 process(clk, reset, phase)

 begin

 if (phase = '0')then

 state\_reg <= phase\_state;

 elsif (reset = '0') then

 state\_reg <= done\_state;

 elsif (clk'event and clk = '1') then

 state\_reg <= state\_next;

 end if;

 end process;

 -- control path: next-state/output logic

 process(state\_reg, start, class\_bit\_m, node\_addition, bit\_reg) --Sensitive list with parameters that needs to go in

 begin

 case state\_reg is

 when phase\_state =>

 state\_next <= done\_state;

 when reset\_state =>

 state\_next <= done\_state;

 when idle =>

 if(start = '1')then

 state\_next <= detect;

 else

 state\_next <= idle;

 end if;

 when detect =>

 if(class\_bit\_m = '1')then

 state\_next <= sort\_values;

 else

 state\_next <= set\_values;

 end if;

 when sort\_values =>

 if(node\_addition < FIFTEEN)then

 state\_next <= sort\_values;

 else

 state\_next <= cal\_values;

 end if;

 when set\_values =>

 if(bit\_next = '1')then

 state\_next <= done\_state;

 else

 state\_next <= read\_arrow;

 end if;

 when cal\_values =>

 if(bit\_next = '1')then

 state\_next <= done\_state;

 else

 state\_next <= read\_arrow;

 end if;

 when done\_state =>

 if(bit\_next = '1')then

 state\_next <= response;

 else

 state\_next <= idle;

 end if;

 when response =>

 if(start = '1')then

 state\_next <= detect;

 else

 state\_next <= response;

 end if;

 when read\_arrow =>

 state\_next <= arrow;

 when arrow =>

 state\_next <= done\_state;

 end case;

 end process;

-- control path: output logic

rd\_wr <= '1' when (state\_reg = set\_values) or (state\_reg = cal\_values) or (state\_reg = arrow) else '0'; -- 1 is write and 0 is read

done <= '1' when (state\_reg = done\_state) else '0';

-- data path: data register

 process(clk, reset, phase)

 begin

 if (phase = '0')then

 elsif (reset = '0') then

 class\_in\_reg <= (others=>'0');

 mx\_reg <= (others=>'0');

 wij\_reg <= (others=>'0');

 wb\_reg <= (others=> (others=>'0'));

 Mp\_reg <= (others=>'0');

 node\_reg <= (others=>'0');

 cd\_reg <= (others=>'0');

 cb\_reg <= (others=>'0');

 bit\_reg <= '0';

 class\_bit\_reg <= '0';

 elsif (clk'event and clk = '1') then

 class\_in\_reg <= class\_in\_next;

 mx\_reg <= mx\_next;

 wij\_reg <= wij\_next;

 wb\_reg <= wb\_next;

 Mp\_reg <= Mp\_next;

 node\_reg <= node\_next;

 cd\_reg <= cd\_next;

 cb\_reg <= cb\_next;

 bit\_reg <= bit\_next;

 class\_bit\_reg <= class\_bit\_next;

 end if;

 end process;

-- data path: routing multiplexer

 process(state\_reg,class\_in\_reg,mx\_reg,wij\_reg,wb\_reg,Mp\_reg,node\_reg,cd\_reg,cb\_reg,bit\_reg,class\_bit\_reg,class\_in,mx\_m,wij\_m,input\_weight,class\_bit\_m,wij\_addition,mx\_addition,wb\_m,bit\_addition, node\_addition)

 begin

 case state\_reg is

 when phase\_state =>--finished

--rd

 class\_in\_next <= class\_in\_reg;

 mx\_next <= mx\_reg;

 wij\_next <= wij\_reg;

 wb\_next <= wb\_reg;

 Mp\_next <= Mp\_reg;

 node\_next <= node\_reg;

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= class\_bit\_reg;

 when done\_state =>--finished

--rd

 class\_in\_next <= class\_in\_reg;

 mx\_next <= mx\_reg;

 wij\_next <= wij\_reg;

 wb\_next <= wb\_reg;

 Mp\_next <= Mp\_reg;

 node\_next <= node\_reg;

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= class\_bit\_reg;

 when reset\_state =>--finished

--rd

 class\_in\_next <= (others=>'0');

 mx\_next <= (others=>'0');

 wij\_next <= (others=>'0');

 wb\_next <= (others=> (others=>'0'));

 Mp\_next <= (others=>'0');

 node\_next <= (others=>'0');

 cd\_next <= (others=>'0');

 cb\_next <= (others=>'0');

 bit\_next <= '0';

 class\_bit\_next <= '0';

 when idle =>--finished

--rd

 class\_in\_next <= class\_in;

 mx\_next <= mx\_m;

 wij\_next <= wij\_m;

 wb\_next <= input\_weight;

 Mp\_next <= (others=>'0');

 node\_next <= (others=>'0');

 cd\_next <= (others=>'0');

 cb\_next <= class\_in;

 bit\_next <= '0';

 class\_bit\_next <= class\_bit\_m;

 when detect =>--finished

--rd

 class\_in\_next <= class\_in;

 mx\_next <= mx\_m;

 wij\_next <= wij\_m;

 wb\_next <= wb\_reg;

 Mp\_next <= Mp\_reg;

 node\_next <= (others=>'0');

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_addition;

 class\_bit\_next <= class\_bit\_m;

 when sort\_values =>--finished

--rd

 class\_in\_next <= class\_in\_reg;

 mx\_next <= mx\_reg;

 wij\_next <= wij\_reg;

 wb\_next <= wb\_mux;

 Mp\_next <= Mp\_mux;

 node\_next <= node\_addition;

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= '1';

 when set\_values =>--finished

--wr

 class\_in\_next <= class\_in\_reg;

 mx\_next <= (others=>'0');

 wij\_next <= wij\_reg;

 wb\_next <= wb\_reg;

 Mp\_next <= Mp\_reg;

 node\_next <= node\_reg;--doubt

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= '1';

 when cal\_values =>--finished

--wr

 class\_in\_next <= class\_in\_reg;

 mx\_next <= mx\_addition;

 wij\_next <= wij\_reg;

 wb\_next <= wb\_reg;

 Mp\_next <= Mp\_reg;

 node\_next <= node\_reg;--doubt

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= '1';

 when response =>--finished

--rd

 class\_in\_next <= class\_in;

 mx\_next <= mx\_m;

 wij\_next <= wij\_m;

 wb\_next <= input\_weight;

 Mp\_next <= (others=>'0');

 node\_next <= (others=>'0');

 cd\_next <= class\_in;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= class\_bit\_m;

 when read\_arrow => --finished

--rd

 class\_in\_next <= cb\_reg;--here

 mx\_next <= mx\_m;

 wij\_next <= wij\_m;

 wb\_next <= wb\_m;

 Mp\_next <= (others=>'0');

 node\_next <= (others=>'0');

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= class\_bit\_m;

 when arrow =>--finished

--wr

 class\_in\_next <= cb\_reg;

 mx\_next <= mx\_m;

 wij\_next <= wij\_addition;

 wb\_next <= wb\_m;--it can be also wb\_reg

 Mp\_next <= (others=>'0');

 node\_next <= (others=>'0');

 cd\_next <= cd\_reg;

 cb\_next <= cb\_reg;

 bit\_next <= bit\_reg;

 class\_bit\_next <= '1';

 end case;

 end process;

class\_bit\_out <= class\_bit\_next;

Cxy <= class\_in\_next;

wb <= wb\_next;

mx <= mx\_next;

wij <= wij\_next;

cd <= cd\_next;

node\_index <= node\_next;

 mx\_adder: entity work.generic\_adder

 generic map (

 bits => NODES

 )

 port map (

 A => mx\_reg,

 B => ONE,

 CI => ZERO,

 O => mx\_addition,

 CO => CO

 );

 wij\_adder: entity work.generic\_adder

 generic map (

 bits => NODES

 )

 port map (

 A => wij\_m,

 B => ONE,

 CI => ZERO,

 O => wij\_addition,

 CO => CO

 );

 node\_adder: entity work.generic\_adder

 generic map (

 bits => NODES

 )

 port map (

 A => node\_reg,

 B => ONE,

 CI => ZERO,

 O => node\_addition,

 CO => CO

 );

 bit\_adder: entity work.BIT\_ADDER

 port map( a => bit\_reg,

 b => '1',

 cin => '0',

 sum => bit\_addition,

 cout => CO);

 Mp\_signal <= '1' when Mp\_m > Mp\_reg else '0';

 mux\_mp:entity work.mux\_2to1\_Mp

 port map( SEL => Mp\_signal,

 A => Mp\_m,

 B => Mp\_reg,

 X => Mp\_mux);

 mux\_wb:entity work.mux\_2to1\_Wb

 port map( SEL => Mp\_signal,

 A => wb\_m,

 B => wb\_reg,

 X => wb\_mux );

end behavioral;

# Chapter 3

**Code for the Sub components**

**Subtraction Component:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all; -- this is the standard package where signed is defined

-- never use non-standard ieee.std\_logic\_arith and ieee.std\_logic\_unsigned

entity sub is

 generic(

 length\_1: integer := 8;

 result\_width: integer := 8

 );

port (

 X: in std\_logic\_vector(length\_1-1 downto 0);

 Y: in std\_logic\_vector(length\_1-1 downto 0);

 F: out std\_logic\_vector(length\_1-1 downto 0)

);

end entity;

architecture S of sub is

begin

process(X,Y) is

begin

 F <= std\_logic\_vector(abs(signed(X)-signed(Y)));

end process;

end architecture;

**Multiplication Component:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use ieee.STD\_LOGIC\_ARITH.all;

entity mul is

 generic(

 length\_1: integer;

 length\_2: integer;

 result\_width: integer

 );

 port(

 a :in std\_logic\_vector(length\_1-1 downto 0);

 b :in std\_logic\_vector(length\_2-1 downto 0);

 p :out std\_logic\_vector(result\_width-1 downto 0)

 );

end entity;

architecture A of mul is

signal s\_p :std\_logic\_vector(length\_1+length\_2-1 downto 0);

signal j : std\_logic\_vector(result\_width-1 downto 0);

begin

 s\_p <= unsigned(a) \* unsigned(b);

 j <= (others => '0');

 p <= j(result\_width-1 downto (length\_1+length\_2)) & s\_p;

end architecture;

**Generic Adder:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity generic\_adder is

 generic (

 bits: integer

 );

 port (

 A: in std\_logic\_vector(bits-1 downto 0);

 B: in std\_logic\_vector(bits-1 downto 0);

 CI: in std\_logic;

 O: out std\_logic\_vector(bits-1 downto 0);

 CO: out std\_logic

 );

end entity generic\_adder;

architecture behave of generic\_adder is

begin

process(A,B,CI)

 variable sum: integer;

 -- Note: we have one bit more to store carry out value.

 variable sum\_vector: std\_logic\_vector(bits downto 0);

begin

 -- Compute our integral sum, by converting all operands into integers.

 sum := conv\_integer(A) + conv\_integer(B) + conv\_integer(CI);

 -- Now, convert back the integral sum into a std\_logic\_vector, of size bits+1

 sum\_vector := conv\_std\_logic\_vector(sum, bits+1);

 -- Assign outputs

 O <= sum\_vector(bits-1 downto 0);

 CO <= sum\_vector(bits); -- Carry is the most significant bit

end process;

end behave;

-- Reference : http://esd.cs.ucr.edu/labs/adder/adder.html

LIBRARY IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- We declare the 1-bit adder with the inputs and outputs

-- shown inside the port().

-- This will add two bits together(x,y), with a carry in(cin) and

-- output the sum(sum) and a carry out(cout).

entity BIT\_ADDER is

 port( a, b, cin : in STD\_LOGIC;

 sum, cout : out STD\_LOGIC );

end BIT\_ADDER;

-- This describes the functionality of the 1-BIT adder.

architecture BHV of BIT\_ADDER is

begin

 -- Calculate the sum of the 1-BIT adder.

 sum <= (not a and not b and cin) or

 (not a and b and not cin) or

 (a and not b and not cin) or

 (a and b and cin);

 -- Calculates the carry out of the 1-BIT adder.

 cout <= (not a and b and cin) or

 (a and not b and cin) or

 (a and b and not cin) or

 (a and b and cin);

end BHV;

**Algorithm 4 main code**

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: algorith4.vhd

-- Created by: Surendra Maddula

-- Date: May 27, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an algorithm, learning of the associative memory. This implments the

-- association between key class and response vectors through nodes in associative memory.

------------------------------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use work.input\_vector\_type.all;

entity Algorithm4 is port(

clk,reset,phase, start: in std\_logic;

Wx : in vector\_input;

Wkm : in vector\_input;

Thm : in std\_logic\_vector(7 downto 0);

done : out std\_logic;

not\_present : out std\_logic;

rd\_wr : out std\_logic;

class\_index : out std\_logic\_vector(CLASS\_SIZE-1 downto 0);

node\_index : out std\_logic\_vector((NODES+CLASS\_SIZE)-1 downto 0));

end Algorithm4;

architecture behavioral of Algorithm4 is

-- FSM states

type ED\_ARRAY is array(0 to IMAGE\_SIZE-2) of std\_logic\_vector((6+(2\*PIXEL\_RESOLUTION))-1 downto 0);

type CO\_ARRAY is array(0 to IMAGE\_SIZE-2) of std\_logic;

type state\_type is (RST\_STATE,IDLE,ITERATE\_FOR\_SUM,SQUARE\_ROOT,DELTA,COMPARE,MAXIMUM,DONE\_ST,ERROR\_ST,PHASE\_STATE); -- newly added DELTA state.

 constant two\_delta : std\_logic\_vector ((6+(2\*PIXEL\_RESOLUTION))-1 downto 0) := (0 => '0', 1 => '1', others => '0');

 constant three\_delta: std\_logic\_vector ((6+(2\*PIXEL\_RESOLUTION))-1 downto 0) := (0 => '1', 1 => '1', others => '0');

 constant one\_delta: std\_logic\_vector ((6+(2\*PIXEL\_RESOLUTION))-1 downto 0) := (0 => '1', others => '0');

 constant one\_sub: std\_logic\_vector(((6+(2\*PIXEL\_RESOLUTION))/2)-1 downto 0):= (0 => '1', others => '0');

 constant one : std\_logic\_vector((NODES+CLASS\_SIZE)-1 downto 0) := (0 => '1', others => '0');

 constant full: std\_logic\_vector((NODES+CLASS\_SIZE)-1 downto 0) := (others => '1');

 constant WID : positive := 16;

 constant one\_for\_A : std\_logic\_vector(NODES-2 downto 0) := (0 => '1', others => '0');

 constant a\_const : std\_logic\_vector(NODES-2 downto 0) := (others => '0');

 signal a,b :std\_logic\_vector(7 downto 0);

 signal state\_current, state\_next: state\_type;

 signal C0: CO\_ARRAY;

 signal incr\_A, cout,cout\_A, cout\_sqrt, cout\_node\_index,done\_temp,not\_present\_temp: std\_logic;

 signal mu,sum : std\_logic\_vector((2\*PIXEL\_RESOLUTION)-1 downto 0);

 signal Wx\_reg, Wx\_next, WSUB : vector\_input;

 signal WMUL : mul\_vector\_input;

 signal WED\_reg, WED\_next: std\_logic\_vector((6+(2\*PIXEL\_RESOLUTION))-1 downto 0); --Worst case : (255-0)^2 + (255-0)^2+ ... = (255^2)\*64 = 4161600 which is 22 bit value.

 signal delta\_reg, delta\_next, delta\_sum: std\_logic\_vector((6+(2\*PIXEL\_RESOLUTION))-1 downto 0); -- newly added for delta state+

 signal sq\_reg, sq\_next, sq\_sum: std\_logic\_vector((6+(2\*PIXEL\_RESOLUTION))-1 downto 0); -- newly added for delta state

 signal sqrt\_next, sqrt\_reg, sqrt\_concatenate, sqrt\_val: std\_logic\_vector(((6+(2\*PIXEL\_RESOLUTION))/2)-1 downto 0); -- newly added for square root state

 signal sum\_out : ED\_ARRAY;

 signal Th\_reg,Th\_next : std\_logic\_vector(PIXEL\_RESOLUTION-1 downto 0);

 signal a\_reg,a\_next, x, A\_plus : std\_logic\_vector(NODES-2 downto 0);

 signal b\_reg,b\_next : std\_logic\_vector(NODES-2 downto 0);

 signal delta\_flag, sel: std\_logic; -- newly added for DELTA state.

 signal class\_index\_next,class\_index\_reg : std\_logic\_vector(CLASS\_SIZE-1 downto 0);

 signal node\_index\_reg, node\_index\_next, node\_index\_sum : std\_logic\_vector((NODES+CLASS\_SIZE)-1 downto 0);

begin

 -- control path: state register

 process(clk, reset, phase)

 begin

 if(phase ='1') then

 state\_current <= PHASE\_STATE;

 elsif (reset = '0') then

 state\_current <= RST\_STATE;

 elsif (clk'event and clk = '1') then

 state\_current <= state\_next;

 end if;

 end process;

 -- control path: next-state/output logic

process(state\_current, start, delta\_flag, node\_index\_next,A\_next)

 variable count : integer := 0;

 begin

 case state\_current is

 when RST\_STATE =>

 state\_next <= DONE\_ST;

 when PHASE\_STATE =>

 state\_next <= DONE\_ST;

 when IDLE =>

 if(start = '1')then

 state\_next <= ITERATE\_FOR\_SUM;

 else

 state\_next <= IDLE;

 end if;

 when ITERATE\_FOR\_SUM =>

 -- calculate Euclidean distance and move to compare state.

 state\_next <= DELTA;

 when DELTA => --newly added state

 if(delta\_flag = '1')then

 state\_next <= SQUARE\_ROOT;

 elsif(delta\_flag = '0')then

 state\_next <= DELTA;

 end if;

 when SQUARE\_ROOT=>

 state\_next <= COMPARE;

 when COMPARE =>

 -- Compare the euclidean distance with the Threshold.

 -- If It is less than Threshold Increment A\_next. and move to maximum state.

 state\_next <= MAXIMUM;

 when MAXIMUM =>

 -- Compare A and B which ever is greater, that one goes to B\_next.

 --Move to Iterate state next.

 if(node\_index\_next < full)then

 state\_next <= ITERATE\_FOR\_SUM;

 else

 if(A\_next = a\_const) then

 state\_next <= ERROR\_ST;

 else

 state\_next <= DONE\_ST;

 end if;

 end if;

 when DONE\_ST =>

 -- Assert Done flag and move to IDLE state.

 state\_next <= IDLE;

 when ERROR\_ST =>

 state\_next <= IDLE;

 end case;

end process;

 -- control path: output logic

done<= '1' when (state\_current = DONE\_ST) else '0';

not\_present <= '1' when (state\_current = ERROR\_ST) else '0';

 -- data path: data register

 process(clk, reset, phase)

 begin

 if(reset = '0') then

 report "entered if reset =0 case";

 Wx\_reg <= (others=>(others=>'0'));

 WED\_reg <= (others=>'0');

 Th\_reg <= (others=> '0');

 node\_index\_reg <= (others=> '0');

 sqrt\_reg <= (others => '0');

 A\_reg <= (others=>'0');

 B\_reg <= (others=>'0');

 class\_index\_reg <= (others=>'0');

 delta\_reg <= (others => '0'); -- newly added

 sq\_reg <= (others => '0'); -- newly added

 elsif (clk'event and clk= '1') then

 report "entered elsif reset =0 case";

 Wx\_reg <= Wx\_next;

 WED\_reg <= WED\_next;

 Th\_reg <= Th\_next;

 node\_index\_reg <= node\_index\_next;

 sqrt\_reg <= sqrt\_next;

 A\_reg <= A\_next;

 B\_reg <= B\_next;

 class\_index\_reg <= class\_index\_next;

 delta\_reg <= delta\_next; -- newly added

 sq\_reg <= sq\_next; -- newly added

 end if;

 end process;

 -- data path: routing multiplexer

 process(state\_current, Wx, sum\_out, Thm, delta\_sum, sq\_sum, sqrt\_val, sqrt\_reg, Th\_reg,sel,incr\_A)

 begin

report "state";

 case state\_current is

 when RST\_STATE =>

 Wx\_next <= (others=>(others=>'0'));

 WED\_next <= (others=>'0');

 Th\_next <= (others=> '0');

 node\_index\_next <= (others=> '0');

 A\_next <= (others=>'0');

 B\_next <= (others=>'0');

 class\_index\_next <= (others=>'0');

 sqrt\_next <= (others=>'0');

 delta\_next <= (others=>'0');

 sq\_next <= (others=>'0');

 when PHASE\_STATE =>

 when IDLE =>

 Wx\_next <= Wx;

 WED\_next <= (others=>'0');

 Th\_next <= (others=> '0');

 node\_index\_next <= (others=> '0');

 A\_next <= (others=>'0');

 B\_next <= (others=>'0');

 class\_index\_next <= (others=>'0');

 sqrt\_next <= (others=>'0');

 delta\_next <= three\_delta;

 sq\_next <= one\_delta;

 when ITERATE\_FOR\_SUM =>

 Wx\_next <= Wx\_reg;

 WED\_next <= sum\_out(IMAGE\_SIZE-2);

 Th\_next <= Thm;

 node\_index\_next <= node\_index\_reg;

 A\_next <= A\_reg;

 B\_next <= B\_reg;

 class\_index\_next <= class\_index\_reg;

 sqrt\_next <= (others=>'0');

 delta\_next <= delta\_reg;

 sq\_next <= one\_delta;

 when DELTA =>

 Wx\_next <= Wx\_reg;

 WED\_next <= WED\_reg;

 Th\_next <= Thm;

 node\_index\_next <= node\_index\_reg;

 A\_next <= A\_reg;

 B\_next <= B\_reg;

 class\_index\_next <= class\_index\_reg;

 sqrt\_next <= (others=>'0');

 delta\_next <= delta\_sum;

 sq\_next <= sq\_sum;

 when SQUARE\_ROOT =>

 Wx\_next <= Wx\_reg;

 WED\_next <= WED\_reg;

 Th\_next <= Th\_reg;

 node\_index\_next <= node\_index\_reg;

 A\_next <= A\_reg;

 B\_next <= B\_reg;

 class\_index\_next <= class\_index\_reg;

 sqrt\_next <= sqrt\_val;

 delta\_next <= delta\_reg;

 sq\_next <= sq\_reg;

 when COMPARE =>

 Wx\_next <= Wx\_reg;

 WED\_next <= WED\_reg;

 Th\_next <= Th\_reg;

 node\_index\_next <= node\_index\_reg;

 B\_next <= B\_reg;

 class\_index\_next <= class\_index\_reg;

 sqrt\_next <= sqrt\_reg;

 delta\_next <= three\_delta;

 sq\_next <= one\_delta;

 if(incr\_A ='1') then

 A\_next <= A\_plus;

 else

 A\_next <= A\_reg;

 end if;

 when MAXIMUM =>

 Wx\_next <= Wx\_reg;

 WED\_next <= WED\_reg;

 Th\_next <= Th\_reg;

 node\_index\_next <= node\_index\_reg;

 A\_next <= A\_reg;

 sqrt\_next <= sqrt\_val;

 node\_index\_next <= node\_index\_sum;

 delta\_next <= three\_delta;

 sq\_next <= one\_delta;

 B\_next <= x;

 if(sel='1') then

 class\_index\_next <= node\_index\_reg(CLASS\_SIZE+NODES-1 downto CLASS\_SIZE+NODES-2);

 else

 class\_index\_next <= class\_index\_reg;

 end if;

 when DONE\_ST =>

 Wx\_next <= Wx\_reg;

 WED\_next <= WED\_reg;

 Th\_next <= Th\_reg;

 node\_index\_next <= node\_index\_reg;

 A\_next <= A\_reg;

 B\_next <= B\_reg;

 class\_index\_next <= class\_index\_reg;

 sqrt\_next <= sqrt\_reg;

 delta\_next <= three\_delta;

 sq\_next <= one\_delta;

 when ERROR\_ST =>

 Wx\_next <= Wx\_reg;

 WED\_next <= WED\_reg;

 Th\_next <= Th\_reg;

 node\_index\_next <= node\_index\_reg;

 A\_next <= A\_reg;

 B\_next <= B\_reg;

 class\_index\_next <= class\_index\_reg;

 sqrt\_next <= sqrt\_reg;

 delta\_next <= three\_delta;

 sq\_next <= one\_delta;

 end case;

 end process;

ADD\_NODE\_INDEX:entity work.generic\_adder

 generic map(

 bits => (NODES+CLASS\_SIZE))

 port map(node\_index\_reg, one,'0',node\_index\_sum,cout\_node\_index);

SUB\_GEN: for i in 0 to IMAGE\_SIZE-1 generate

S1 : entity work.sub

 generic map(

 length\_1 => PIXEL\_RESOLUTION,

 result\_width => PIXEL\_RESOLUTION)

 port map(X => Wx(i),Y => wkm(i),F => WSUB(i));

 end generate SUB\_GEN;

MUL\_GEN: for i in 0 to IMAGE\_SIZE-1 generate

M1 : entity work.mul

 generic map(

 length\_1 => PIXEL\_RESOLUTION,

 length\_2 => PIXEL\_RESOLUTION,

 result\_width => 6+(2\*PIXEL\_RESOLUTION))

 port map(WSUB(i),WSUB(i),WMUL(i));

 end generate MUL\_GEN;

ADD\_GEN\_1:entity work.generic\_adder

 generic map(

 bits => (6+(2\*PIXEL\_RESOLUTION)))

 port map(WMUL(0), WMUL(1),'0',sum\_out(0),C0(0));

ADD\_GEN: for i in 1 to IMAGE\_SIZE-2 generate

A1 : entity work.generic\_adder

 generic map(

 bits => (6+(2\*PIXEL\_RESOLUTION)))

 port map(sum\_out(i-1), WMUL(i),C0(i-1),sum\_out(i),C0(i));

 end generate ADD\_GEN;

ADD\_DELTA:entity work.generic\_adder

 generic map(

 bits => (6+(2\*PIXEL\_RESOLUTION)))

 port map(delta\_reg, two\_delta,'0',delta\_sum,cout);

A\_INC: entity work.generic\_adder

 generic map(

 bits => NODES-1)

 port map(A\_reg,one\_for\_A,'0',A\_plus,cout\_A);

ADD\_SQRT:entity work.generic\_adder

 generic map(

 bits => (6+(2\*PIXEL\_RESOLUTION)))

 port map(delta\_reg, sq\_reg,'0', sq\_sum, cout\_sqrt);

delta\_flag <= '1' when sq\_sum > WED\_reg else '0';

sqrt\_concatenate <= '0' & delta\_reg(((6+(2\*PIXEL\_RESOLUTION))/2)-1 downto 1);

sel <='1' when A\_reg > B\_reg else '0';

incr\_A <= '1' when sqrt\_reg <= Th\_reg else '0';

S1 : entity work.sub

 generic map(

 length\_1 => ((6+(2\*PIXEL\_RESOLUTION))/2),

 result\_width => ((6+(2\*PIXEL\_RESOLUTION))/2))

 port map(X => sqrt\_concatenate,Y => one\_sub,F => sqrt\_val);

MUX1 : entity work.mux\_2to1\_Mp

 port map(SEL => sel, A => A\_reg,B => B\_reg,X => x );

class\_index <= class\_index\_reg;

node\_index <= node\_index\_next;

rd\_wr <= '0'; --rd\_wr = '0' read and '1' write

end behavioral;

**Testbench for Algorithm 4:**

The testbench consists of lengthy input sequeces which will make this document very big. So I am not putting the testbench logic here. All the code is available in the attachement.

------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------

-- Filename: algorith4\_tb.vhd

-- Created by: Surendra Maddula

-- Date: May 27, 2016

-- ECE 590: Digital systems design using hardware description language (VHDL).

-- Final Project: A General Associative memory based on self-organizing incremental neural network

-- This is an Implementation of an algorithm, learning of the associative memory. This implments the

-- association between key class and response vectors through nodes in associative memory.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

use std.env.all;

use work.input\_vector\_type.all; --package

ENTITY algorith4\_tb is

END algorith4\_tb;

ARCHITECTURE behavior OF algorith4\_tb IS

component Algorithm4 port(

clk,reset,phase, start: in std\_logic;

Wx : in vector\_input;

Wkm : in vector\_input;

Thm : in std\_logic\_vector(7 downto 0);

class\_index : out std\_logic\_vector(CLASS\_SIZE-1 downto 0);

node\_index : out std\_logic\_vector((NODES+CLASS\_SIZE)-1 downto 0);

rd\_wr : out std\_logic;

done: out std\_logic;

not\_present: out std\_logic

);

end component;

signal clk, start, done, not\_present, phase : std\_logic;

signal reset : std\_logic;

signal Wx : vector\_input;

signal Wkm : vector\_input;

signal Thm : std\_logic\_vector(7 downto 0);

signal rd\_wr: std\_logic;

signal class\_index : std\_logic\_vector(CLASS\_SIZE-1 downto 0);

signal node\_index : std\_logic\_vector((NODES+CLASS\_SIZE)-1 downto 0);

onstant\_array(0) <= constant\_array\_0;

constant\_array(1) <= constant\_array\_1;

constant\_array(2) <= constant\_array\_2;

constant\_array(3) <= constant\_array\_3;

constant\_array(4) <= constant\_array\_4;

constant\_array(5) <= constant\_array\_5;

constant\_array(6) <= constant\_array\_6;

constant\_array(7) <= constant\_array\_7;

constant\_array(8) <= constant\_array\_8;

constant\_array(9) <= constant\_array\_9;

constant\_array(10) <= constant\_array\_10;

constant\_array(11) <= constant\_array\_11;

constant\_array(12) <= constant\_array\_12;

constant\_array(13) <= constant\_array\_13;

constant\_array(14) <= constant\_array\_14;

constant\_array(15) <= constant\_array\_15;

constant\_array(16) <= constant\_array\_16;

constant\_array(17) <= constant\_array\_17;

constant\_array(18) <= constant\_array\_18;

constant\_array(19) <= constant\_array\_19;

constant\_array(20) <= constant\_array\_20;

constant\_array(21) <= constant\_array\_21;

constant\_array(22) <= constant\_array\_22;

constant\_array(23) <= constant\_array\_23;

constant\_array(24) <= constant\_array\_24;

constant\_array(25) <= constant\_array\_25;

constant\_array(26) <= constant\_array\_26;

constant\_array(27) <= constant\_array\_27;

constant\_array(28) <= constant\_array\_28;

constant\_array(29) <= constant\_array\_29;

constant\_array(30) <= constant\_array\_30;

constant\_array(31) <= constant\_array\_31;

constant\_array(32) <= constant\_array\_32;

constant\_array(33) <= constant\_array\_33;

constant\_array(34) <= constant\_array\_34;

constant\_array(35) <= constant\_array\_35;

constant\_array(36) <= constant\_array\_36;

constant\_array(37) <= constant\_array\_37;

constant\_array(38) <= constant\_array\_38;

constant\_array(39) <= constant\_array\_39;

constant\_array(40) <= constant\_array\_40;

constant\_array(41) <= constant\_array\_41;

constant\_array(42) <= constant\_array\_42;

constant\_array(43) <= constant\_array\_43;

constant\_array(44) <= constant\_array\_44;

constant\_array(45) <= constant\_array\_45;

constant\_array(46) <= constant\_array\_46;

constant\_array(47) <= constant\_array\_47;

constant\_array(48) <= constant\_array\_48;

constant\_array(49) <= constant\_array\_49;

constant\_array(50) <= constant\_array\_50;

constant\_array(51) <= constant\_array\_51;

constant\_array(52) <= constant\_array\_52;

constant\_array(53) <= constant\_array\_53;

constant\_array(54) <= constant\_array\_54;

constant\_array(55) <= constant\_array\_55;

constant\_array(56) <= constant\_array\_56;

constant\_array(57) <= constant\_array\_57;

constant\_array(58) <= constant\_array\_58;

constant\_array(59) <= constant\_array\_59;

constant\_array(60) <= constant\_array\_60;

constant\_array(61) <= constant\_array\_61;

constant\_array(62) <= constant\_array\_62;

constant\_array(63) <= constant\_array\_63;

A1 : process(node\_index)

begin

 Wkm <= constant\_array((to\_integer(unsigned(node\_index))));

end process;

clocked\_process: process

begin

 clk <= '0';

 wait for CLOCK\_PERIOD/2 ;

 clk <= '1';

 wait for CLOCK\_PERIOD/2;

end process;

Stimuli: process

begin

 reset <= '0'; -- reset state

 phase <= '0';

 start <= '0';

 wait for 20 ns;

 reset <= '1';

 wait for 20 ns;

 start <= '1';

 Thm <= "00000101";

 Wx <=("01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111",

"01111111");

wait for 1000 ns;

 start <= '0';

wait for 1000000 ns;

finish(0);

end process;

end behavior;

Chapter 4 CONTROLLER\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

entity controller\_tb is

end entity;

architecture behaviour of controller\_tb is

signal Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out:std\_logic;

signal Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : std\_logic;

signal Cx: std\_logic\_vector(1 downto 0);

constant clk\_period:time:= 10 ns;

constant temp\_period:time:= 450 ns;

signal algorithm\_out:Std\_logic;

component controller is

port( Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,algorithm\_out: in std\_logic;

 Cx : in std\_logic\_vector(1 downto 0);

 Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : out std\_logic);

end Component;

begin

DUT: controller port map (Clock =>Clock ,Reset => Reset,Done =>Done,Comparator\_Out => Comparator\_Out,Sort\_Done => Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out,Cx => Cx,

Reg\_Cx\_Load => Reg\_Cx\_Load,Counter\_Load => Counter\_Load,Counter\_Reset => Counter\_Reset,Counter\_Enable => Counter\_Enable,Mbx\_Load => Mbx\_Load,Mbx\_Reset => Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter,Wd\_Load =>Wd\_Load,Comparator\_Enable => Comparator\_Enable,Cy\_Load => Cy\_Load,

Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Rd\_Memory => Rd\_Memory,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg => Clear\_Temp\_Reg,Toggle => Toggle,algorithm\_out => algorithm\_out

);

clk\_process: process

begin

 Clock <= '0';

 wait for clk\_period/2;

 Clock <= '1';

 wait for Clk\_period/2;

end process;

stimuli\_process: process

begin

reset <= '1';

wait for clk\_period;

Reset <= '0';

temp\_reg\_out <= '1';

Comparator\_Out <= '1';

Done <= '1';

wait for temp\_period;

temp\_reg\_out <= '0';

reset <= '0';

Comparator\_Out <= '1';

Done <= '1';

wait for temp\_period;

end process;

end behaviour;

counter:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_Std.all;

entity counter is

port( Memory\_class\_mbx: in std\_logic\_vector(7 downto 0);

counter\_load,counter\_reset,reset,counter\_enable : in std\_logic;

counter\_out: out integer

);

end counter;

architecture behaviour of counter is

signal register\_counter :integer;

begin

 counter\_out <= register\_counter;

 process(counter\_load,counter\_enable,reset,counter\_reset)

 begin

 if (reset = '1' or counter\_reset = '1') then -- -- initial state or reset

 register\_counter <= 0;

 elsif (counter\_load = '1') then -- load the counter

 register\_counter <= to\_integer(unsigned(Memory\_class\_mbx));

 elsif (counter\_enable = '1') then

 register\_counter <= register\_counter - 1;

 end if;

 end process;

end behaviour;

DATAPATH:

library ieee;

use ieee.std\_logic\_1164.all;

entity datapath is

port(clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: out std\_logic;

data\_out:out std\_logic\_vector(7 downto 0);

-- memory input to the datapath

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0)

);

end datapath;

architecture behaviour of datapath is

--signal clock,reset,Cx,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : std\_logic;

--signal Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,data\_out: std\_logic;

signal Reg\_cy\_output: std\_logic\_vector(15 downto 0);

signal counter\_out:integer;

signal reg\_K\_out:integer;

begin

DUT\_0: entity work.reg\_k

port map(

reg\_k\_load => reg\_k\_load,

 Reg\_k\_out => Reg\_k\_out

);

DUT\_1: entity work.reg\_cx

port map (

 clk => clk,

 reg\_cx\_load => reg\_cx\_load,

 reg\_cx\_in => Cx ,

 Memory\_in => Memory\_cx\_in

);

DUT\_2: entity work.counter

port map(

Memory\_class\_mbx => Memory\_class\_mbx\_out,

counter\_load =>counter\_load,

counter\_reset => counter\_reset,

reset => reset,

counter\_enable => counter\_enable,

counter\_out => counter\_out

);

DUT\_3: entity work.reg\_mbx

port map(

 reset => reset,

 Mbx\_Reset => Mbx\_Reset,

 Mbx\_load => Mbx\_load,

 downcounter\_out => counter\_out,

 memory\_responseclass => memory\_class\_mbx\_in

);

DUT\_4: entity work.temp\_reg

port map(

load\_temp\_reg => load\_temp\_reg,

clear\_temp\_reg => clear\_temp\_reg,

temp\_reg\_out => temp\_reg\_out

);

DUT\_5: entity work.comparator

port map (

comparator\_enable => comparator\_enable,

 reg\_k\_out => reg\_k\_out,

 downcounter\_out => counter\_out,

 comparator\_out => comparator\_out);

DUT\_6:entity work.cy\_register

port map( clk => clk,

reset => reset,

cy\_load => cy\_load,

Memory\_cy => Memory\_cy\_out,

Memory\_wij => Memory\_wij\_out,

sorted => sort\_done,

output => Reg\_cy\_output

);

DUT\_7: entity work.PISO

port map (

clk => clk,

input=> Reg\_cy\_output , -- output from the register cy as input to PISO

PISO\_enable => shift\_enable,

output => Memory\_cy\_in -- output from PISO as input to Memory

);

DUT\_8: entity work.wd\_register

port map(

wd\_load => Wd\_Load,

memory\_wd\_in =>Memory\_Wd\_out,

output => data\_out,

done => Done

);

--DUT3: entity work.Memory

--port map(

--

--);

end behaviour;

DATAPATH\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

entity datapath\_tb is

end datapath\_tb;

architecture behaviour of datapath\_tb is

component datapath is

port(clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: out std\_logic;

data\_out:out std\_logic\_vector(7 downto 0);

-- memory input to the datapath

Memory\_Cx\_in,Memory\_cy\_out,Memory\_cy\_in: inout std\_logic\_vector(1 downto 0);

Memory\_wij\_out: inout std\_logic\_vector(5 downto 0);

Memory\_wd\_in : inout std\_logic\_vector(7 downto 0);

Memory\_class\_mbx\_out : inout std\_logic\_vector(7 downto 0);

Memory\_class\_mbx\_in: inout integer );

end component;

signal clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : std\_logic:='0';

signal Cx: std\_logic\_vector(1 downto 0);

signal Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: std\_logic:='0';

signal data\_out: std\_logic\_vector(7 downto 0):="00000000";

-- memory signals

signal Memory\_Cx\_in,Memory\_cy\_out,Memory\_cy\_in: std\_logic\_vector(1 downto 0);

signal Memory\_wij\_out: std\_logic\_vector(5 downto 0);

signal Memory\_wd\_in : std\_logic\_vector(7 downto 0);

signal Memory\_class\_mbx\_out : std\_logic\_vector(7 downto 0);

signal Memory\_class\_mbx\_in: integer;

constant clk\_period:time:= 10 ns;

begin

DUT: datapath port map (

clk => clk,reset => reset,Reg\_Cx\_Load =>Reg\_Cx\_Load ,Counter\_Load => Counter\_Load ,Counter\_Reset => Counter\_Reset ,Counter\_Enable =>Counter\_Enable ,Mbx\_Load => Mbx\_Load ,Mbx\_Reset=> Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter ,Wd\_Load =>Wd\_Load,

Comparator\_Enable =>Comparator\_Enable ,Cy\_Load =>Cy\_Load ,Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Rd\_Memory => Rd\_Memory,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg =>Clear\_Temp\_Reg ,PISO\_enable=> PISO\_enable,

Cx => Cx,

Done => Done,Comparator\_Out =>Comparator\_Out ,Sort\_Done =>Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out, data\_out =>data\_out,

Memory\_Cx\_in => Memory\_Cx\_in,

Memory\_cy\_out => Memory\_cy\_out,

Memory\_cy\_in => Memory\_cy\_in ,

Memory\_wij\_out => Memory\_wij\_out,

Memory\_wd\_in => Memory\_wd\_in,

Memory\_class\_mbx\_out => Memory\_class\_mbx\_out,

Memory\_class\_mbx\_in => Memory\_class\_mbx\_in

);

clk\_process: process

begin

clk <= '0';

 wait for clk\_period/2;

 clk <= '1';

 wait for clk\_period/2;

end process;

stimuli\_process: process

begin

reset <= '0';

reg\_cx\_load <= '1';

load\_temp\_reg <= '1';

clear\_temp\_reg <= '0';

reg\_k\_load <= '1';

counter\_reset <= '1';

Cx <= "01";

wait for clk\_period;

Memory\_class\_mbx\_out <= "00000011";

reg\_cx\_load <= '0';

counter\_reset <= '0';

wait for clk\_period;

counter\_load <= '1';

reg\_k\_load <= '1';

wait for clk\_period;

counter\_load <= '0';

counter\_enable <= '1';

load\_temp\_reg <= '0';

clear\_temp\_reg <= '1';

wait for clk\_period;

counter\_enable <='0';

mbx\_load <= '1';

reg\_k\_load <= '1';

wait for clk\_period;

memory\_cy\_out <= "01";

memory\_wij\_out <= "000010";

wait for clk\_period;

cy\_load <= '1';

mbx\_load <='0';

wait for clk\_period;

cy\_load <= '0';

comparator\_enable <= '1';

wait for clk\_period;

counter\_load <= '0';

counter\_enable <= '1';

load\_temp\_reg <= '0';

clear\_temp\_reg <= '1';

wait for clk\_period;

counter\_enable <='0';

mbx\_load <= '1';

reg\_k\_load <= '1';

wait for clk\_period;

memory\_cy\_out <= "10";

memory\_wij\_out <= "000011";

wait for clk\_period;

cy\_load <= '1';

mbx\_load <='0';

wait for clk\_period;

cy\_load <= '0';

comparator\_enable <= '1';

wait for clk\_period;

shift\_enable <= '1';

comparator\_enable <= '0';

end process;

end behaviour;

EVENSORT:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY even\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 even\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY even\_sort;

ARCHITECTURE even\_sorter OF even\_sort IS

-- use this module to perform data comparison

COMPONENT comp IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 num0\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num1\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num0\_out : OUT std\_logic\_vector (7 DOWNTO 0);

 num1\_out : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT comp;

BEGIN

-- sort data0 and data1

even\_comp1:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in0, even\_in1, even\_out0,

 even\_out1);

-- sort data2 and data3

even\_comp2:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in2, even\_in3, even\_out2,

 even\_out3);

-- sort data4 and data5

even\_comp3:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in4, even\_in5, even\_out4,

 even\_out5);

-- sort data6 and data7

even\_comp4:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in6, even\_in7, even\_out6,

 even\_out7);

END ARCHITECTURE even\_sorter;

LOAD\_TEMP\_REG:

library ieee;

use ieee.std\_logic\_1164.all;

entity temp\_reg is

port(load\_temp\_reg,clear\_temp\_reg : in std\_logic;

 temp\_reg\_out : out std\_logic

);

end temp\_reg;

architecture behaviour of temp\_reg is

begin

 process(load\_temp\_reg,clear\_temp\_reg)

 begin

 if (load\_temp\_reg /= clear\_temp\_reg) then

 if (load\_temp\_reg = '1' and clear\_temp\_reg = '0' ) then

 temp\_reg\_out <= '1';

 elsif (load\_temp\_reg <= '0' and clear\_temp\_reg = '1') then

 temp\_reg\_out <= '0';

 end if;

 else

 temp\_reg\_out <= 'X';

 end if;

 end process;

end behaviour;

NEW\_SORT:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY new\_sort IS

 PORT (

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 old\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 mux\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY new\_sort;

ARCHITECTURE mux OF new\_sort IS

BEGIN

 PROCESS (reset, new\_set, new\_in0, new\_in1, new\_in2, new\_in3, new\_in4,

 new\_in5, new\_in6, new\_in7, old\_in0, old\_in1, old\_in2,

 old\_in3, old\_in4, old\_in5, old\_in6, old\_in7)

 BEGIN

 -- reset everything to '0' when reset is asserted

 IF (reset = '1') THEN

 mux\_out0 <= (OTHERS => '0');

 mux\_out1 <= (OTHERS => '0');

 mux\_out2 <= (OTHERS => '0');

 mux\_out3 <= (OTHERS => '0');

 mux\_out4 <= (OTHERS => '0');

 mux\_out5 <= (OTHERS => '0');

 mux\_out6 <= (OTHERS => '0');

 mux\_out7 <= (OTHERS => '0');

 ELSE

 -- if new\_set is asserted, the mux output should contain the

 -- new set of data

 IF (new\_set = '1') THEN

 mux\_out0 <= new\_in0;

 mux\_out1 <= new\_in1;

 mux\_out2 <= new\_in2;

 mux\_out3 <= new\_in3;

 mux\_out4 <= new\_in4;

 mux\_out5 <= new\_in5;

 mux\_out6 <= new\_in6;

 mux\_out7 <= new\_in7;

 -- otherwise, let the old data coming out of odd sort through

 ELSE

 mux\_out0 <= old\_in0;

 mux\_out1 <= old\_in1;

 mux\_out2 <= old\_in2;

 mux\_out3 <= old\_in3;

 mux\_out4 <= old\_in4;

 mux\_out5 <= old\_in5;

 mux\_out6 <= old\_in6;

 mux\_out7 <= old\_in7;

 END IF;

 END IF;

 END PROCESS;

END ARCHITECTURE mux;

ODDSORT:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY odd\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 odd\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY odd\_sort;

ARCHITECTURE odd\_sorter OF odd\_sort IS

-- use this module to perform data comparison

COMPONENT comp IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 num0\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num1\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num0\_out : OUT std\_logic\_vector (7 DOWNTO 0);

 num1\_out : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT comp;

BEGIN

 PROCESS (reset, clk)

 BEGIN

 -- reset everything to '0' if reset is asserted

 IF (reset = '1') THEN

 odd\_out0 <= (OTHERS => '0');

 odd\_out7 <= (OTHERS => '0');

 -- send data0 and data7 straight through

 ELSIF (rising\_edge (clk)) THEN

 odd\_out0 <= odd\_in0;

 odd\_out7 <= odd\_in7;

 END IF;

 END PROCESS;

-- sort data1 and data2

odd\_comp1:comp

PORT MAP (clk, dir\_of\_sort, reset, odd\_in1, odd\_in2, odd\_out1,

 odd\_out2);

-- sort data3 and data4

odd\_comp2:comp

PORT MAP (clk, dir\_of\_sort, reset, odd\_in3, odd\_in4, odd\_out3,

 odd\_out4);

-- sort data5 and data6

odd\_comp3:comp

PORT MAP (clk, dir\_of\_sort, reset, odd\_in5, odd\_in6, odd\_out5,

 odd\_out6);

END ARCHITECTURE odd\_sorter;

PISO:

library ieee;

use ieee.std\_logic\_1164.all;

entity PISO is

port(clk: in std\_logic;

input: in std\_logic\_vector(0 to 15); -- output from the register cy

PISO\_enable: in std\_logic;

output: out std\_logic\_vector(0 to 1)

);

end PISO;

architecture behaviour of PISO is

signal i:integer:=0;

begin

process(clk)

begin

 if(rising\_Edge(clk)) then

 if(PISO\_enable = '1') then

 if( i <= 15) then

 output <= input(i to i+1);

 i <= i+1+1;

 end if;

 end if;

 end if;

end process;

end behaviour;

REG\_cX:

library ieee;

use ieee.std\_logic\_1164.all;

entity reg\_cx is

port ( clk: in std\_logic;

 reg\_cx\_load : in std\_logic;

 reg\_cx\_in : in std\_logic\_vector(1 downto 0);

 Memory\_in : out std\_logic\_vector(1 downto 0)

);

end reg\_cx;

architecture behaviour of reg\_cx is

begin

 process(clk)

 begin

 if (reg\_cx\_load = '1') then

 Memory\_in <= reg\_cx\_in;

 end if;

 end process;

end behaviour;

REG\_K:

library ieee;

use ieee.std\_logic\_1164.all;

entity reg\_k is

port (reg\_k\_load : in std\_logic;

 Reg\_k\_out : out integer

 );

end reg\_k;

architecture behaviour of reg\_k is

signal register\_k : integer;

begin

 process(reg\_k\_load)

 begin

 if (reg\_k\_load = '1' ) then

 register\_k <= 1;

 else

 register\_k <= 1;

 end if;

 Reg\_k\_out <= register\_k;

 end process;

end behaviour;

REG\_MBX:

library ieee;

use ieee.std\_logic\_1164.all;

entity reg\_mbx is

port ( reset: in std\_logic;

 Mbx\_Reset: in std\_logic;

 Mbx\_load : in std\_logic;

 downcounter\_out: in integer;

 memory\_responseclass: out integer

);

end reg\_mbx;

architecture behaviour of reg\_mbx is

signal Register\_Mbx : integer;

begin

 process(Mbx\_load,reset,Mbx\_Reset,downcounter\_out)

 begin

 if (reset = '1' or Mbx\_Reset = '1') then -- initial state or reset

 Register\_Mbx <= 0;

 elsif (Mbx\_load = '1') then

 Memory\_responseclass <= downcounter\_out;

 end if;

 end process;

end behaviour;

REGISTER:

library ieee;

use ieee.std\_logic\_1164.all;

entity cy\_register is

port(

clk : in std\_logic;

reset: in std\_logic;

cy\_load :in std\_logic;

Memory\_cy: in std\_logic\_vector(1 downto 0);

Memory\_wij: in std\_logic\_vector(5 downto 0);

sorted: out std\_logic;

output: out std\_logic\_vector(15 downto 0)

);

end cy\_register;

architecture behavioural of cy\_register is

type lutable\_1 is array(0 to 7) of std\_logic\_vector(1 downto 0);

type lutable\_2 is array(0 to 7) of std\_logic\_vector(5 downto 0);

signal sample\_array\_1: lutable\_1:= ((others =>(others => '0'))) ;

signal sample\_array\_2:lutable\_2:= ((others =>(others => '0'))) ;

signal i: integer:=0;

signal sort\_en: std\_logic :='0';

signal dir\_of\_sort: bit := '1';

signal new\_set: std\_logic := '0';

signal sorter\_in0: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in1:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in2:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in3: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in4: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in5: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in6: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in7: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out0:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out1:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out2: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out3: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out4: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out5: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out6: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out7: std\_logic\_vector(7 downto 0) := "00000000";

signal sorted\_signal: std\_logic :='0';

signal out0: std\_logic\_vector(1 downto 0) := "00";

signal out1: std\_logic\_vector(1 downto 0):= "00";

signal out2: std\_logic\_vector(1 downto 0) := "00";

signal out3: std\_logic\_vector(1 downto 0) := "00";

signal out4: std\_logic\_vector(1 downto 0) := "00";

signal out5: std\_logic\_vector(1 downto 0) := "00";

signal out6: std\_logic\_vector(1 downto 0) := "00";

signal out7: std\_logic\_vector(1 downto 0) := "00";

COMPONENT sorter is

PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 in0 : IN std\_logic\_vector (7 DOWNTO 0);

 in1 : IN std\_logic\_vector (7 DOWNTO 0);

 in2 : IN std\_logic\_vector (7 DOWNTO 0);

 in3 : IN std\_logic\_vector (7 DOWNTO 0);

 in4 : IN std\_logic\_vector (7 DOWNTO 0);

 in5 : IN std\_logic\_vector (7 DOWNTO 0);

 in6 : IN std\_logic\_vector (7 DOWNTO 0);

 in7 : IN std\_logic\_vector (7 DOWNTO 0);

 out0 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out1 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out2 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out3 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out4 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out5 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out6 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out7 : INOUT std\_logic\_vector (7 DOWNTO 0);

 sorted : OUT std\_logic

 );

END COMPONENT sorter;

begin

s:sorter

PORT MAP (clk,dir\_of\_sort,new\_set,reset,sort\_en,sorter\_in0,sorter\_in1,sorter\_in2,sorter\_in3,sorter\_in4,sorter\_in5,sorter\_in6,sorter\_in7,sorter\_out0,sorter\_out1,sorter\_out2,sorter\_out3,sorter\_out4,sorter\_out5,sorter\_out6,sorter\_out7,sorted\_signal);

sorted <= sorted\_signal;

 out0 <= sorter\_out0(1 downto 0);

 out1 <= sorter\_out1(1 downto 0);

 out2 <= sorter\_out2(1 downto 0);

 out3 <= sorter\_out3(1 downto 0);

 out4 <= sorter\_out4(1 downto 0);

 out5 <= sorter\_out5(1 downto 0);

 out6 <= sorter\_out6(1 downto 0);

 out7 <= sorter\_out7(1 downto 0);

output <= out0&out1&out2&out3&out4&out5&out6&out7;

process (cy\_load,clk)

begin

 if (rising\_Edge(cy\_load)) then

 if (i <= 7) then

 sample\_array\_1(i) <= memory\_cy;

 sample\_array\_2(i) <= memory\_wij;

 i <= i+1;

 end if;

 end if;

 if (rising\_Edge(clk)) then

 sorter\_in0 <= sample\_array\_2(0) & sample\_array\_1(0);

 sorter\_in1 <= sample\_array\_2(1) & sample\_array\_1(1);

 sorter\_in2 <= sample\_array\_2(2) & sample\_array\_1(2);

 sorter\_in3 <= sample\_array\_2(3) & sample\_array\_1(3);

 sorter\_in4 <= sample\_array\_2(4) & sample\_array\_1(4);

 sorter\_in5 <= sample\_array\_2(5) & sample\_array\_1(5);

 sorter\_in6 <= sample\_array\_2(6) & sample\_array\_1(6);

 sorter\_in7 <= sample\_array\_2(7) & sample\_array\_1(7);

 sort\_en <= '1';

 new\_set <= '1';

 end if;

end process;

end behavioural;

REGISTER\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

entity register\_tb is

end register\_tb;

architecture behaviour of register\_tb is

component cy\_register is

port(

clk : in std\_logic;

reset: in std\_logic;

cy\_load :in std\_logic;

Memory\_cy: in std\_logic\_vector(1 downto 0);

Memory\_wij: in std\_logic\_vector(5 downto 0);

sorted: out std\_logic;

output: out std\_logic\_vector(15 downto 0)

-- reg\_cy\_out : out std\_logic\_vector(n+2 downto 0);

);

end component;

signal clk,reset,cy\_load:std\_logic:='0';

signal Memory\_cy:std\_logic\_vector(1 downto 0):="00";

signal Memory\_wij:std\_logic\_vector(5 downto 0):="000000";

signal sorted:std\_logic:= '0';

signal output :std\_logic\_vector(15 downto 0) :="0000000000000000";

constant clk\_period:time:= 10 ns;

begin

dut: cy\_register port map (

clk => clk,reset => reset,cy\_load => cy\_load,Memory\_cy => Memory\_cy ,Memory\_wij =>Memory\_wij ,sorted => sorted,output =>output );

clk\_process: process

begin

 clk <= '0';

 wait for clk\_period/2;

 clk <= '1';

 wait for clk\_period/2;

end process;

stimuli: process

begin

reset <= '0';

cy\_load <= '0';

wait for clk\_period/2;

memory\_cy <= "01";

memory\_wij <= "001000";

cy\_load <= '1';

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "11";

memory\_wij <= "000100";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "10";

memory\_wij <= "000010";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "11";

memory\_wij <= "000001";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "00";

memory\_wij <= "000111";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "01";

memory\_wij <= "000111";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "01";

memory\_wij <= "000010";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "10";

memory\_wij <= "000101";

wait for clk\_period/2;

end process;

end behaviour;

SORTDONE:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY sort\_done IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 num0 : IN std\_logic\_vector (7 DOWNTO 0);

 num1 : IN std\_logic\_vector (7 DOWNTO 0);

 num2 : IN std\_logic\_vector (7 DOWNTO 0);

 num3 : IN std\_logic\_vector (7 DOWNTO 0);

 num4 : IN std\_logic\_vector (7 DOWNTO 0);

 num5 : IN std\_logic\_vector (7 DOWNTO 0);

 num6 : IN std\_logic\_vector (7 DOWNTO 0);

 num7 : IN std\_logic\_vector (7 DOWNTO 0);

 sorted : OUT std\_logic

 );

END ENTITY sort\_done;

ARCHITECTURE verify OF sort\_done IS

BEGIN

 PROCESS (reset, dir\_of\_sort, sort\_en, num0, num1, num2, num3, num4,

 num5, num6, num7,clk)

 BEGIN

 IF (reset = '1') THEN

 sorted <= '0';

 ELSE

 CASE dir\_of\_sort IS

 -- direction of sort is ascending (num0 = min, num7 = max),

 -- so check to see if all inputs are in desired order;

 -- assert sorted if sort\_en is asserted and all numbers are

 -- in order

 WHEN '1' =>

 IF (sort\_en = '1' AND num0(7 downto 2) <= num1(7 downto 2) AND num1(7 downto 2) <= num2(7 downto 2) AND

 num2(7 downto 2) <= num3(7 downto 2) AND num3(7 downto 2) <= num4(7 downto 2) AND num4(7 downto 2) <= num5(7 downto 2) AND

 num5(7 downto 2) <= num6(7 downto 2) AND num6(7 downto 2) <= num7(7 downto 2)) THEN

 sorted <= '1';

 ELSE

 sorted <= '0';

 END IF;

 -- direction of sort is descending (num0 = max, num7 = min),

 -- so check to see if all inputs are in desired order;

 -- assert sorted if sort\_en is asserted and all numbers are

 -- in order

 WHEN '0' =>

 IF (sort\_en = '1' AND num0(7 downto 2) >= num1(7 downto 2) AND num1(7 downto 2) >= num2(7 downto 2) AND

 num2(7 downto 2) >= num3(7 downto 2) AND num3(7 downto 2) >= num4(7 downto 2) AND num4(7 downto 2) >= num5(7 downto 2) AND

 num5(7 downto 2) >= num6(7 downto 2) AND num6(7 downto 2) >= num7(7 downto 2)) THEN

 sorted <= '1';

 ELSE

 sorted <= '0';

 END IF;

 END CASE;

 END IF;

 END PROCESS;

END ARCHITECTURE verify;

SORTER:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY sorter IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 in0 : IN std\_logic\_vector (7 DOWNTO 0):= "00000000";

 in1 : IN std\_logic\_vector (7 DOWNTO 0):= "00000000";

 in2 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in3 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in4 : IN std\_logic\_vector (7 DOWNTO 0):= "00000000";

 in5 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in6 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in7 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out0 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out1 : INOUT std\_logic\_vector (7 DOWNTO 0):= "00000000";

 out2 : INOUT std\_logic\_vector (7 DOWNTO 0):= "00000000";

 out3 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out4 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out5 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out6 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out7 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 sorted : OUT std\_logic

 );

END ENTITY sorter;

ARCHITECTURE sort OF sorter IS

type signals is array(0 to 7) of std\_logic\_vector(7 downto 0);

signal even\_odd\_0,even\_odd\_1,even\_odd\_2,even\_odd\_3,even\_odd\_4,even\_odd\_5,even\_odd\_6,even\_odd\_7 : signals;

signal sorted\_done:std\_logic;

signal I: integer;

COMPONENT even\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 even\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT even\_sort;

COMPONENT odd\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 odd\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT odd\_sort;

COMPONENT sort\_done IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 num0 : IN std\_logic\_vector (7 DOWNTO 0);

 num1 : IN std\_logic\_vector (7 DOWNTO 0);

 num2 : IN std\_logic\_vector (7 DOWNTO 0);

 num3 : IN std\_logic\_vector (7 DOWNTO 0);

 num4 : IN std\_logic\_vector (7 DOWNTO 0);

 num5 : IN std\_logic\_vector (7 DOWNTO 0);

 num6 : IN std\_logic\_vector (7 DOWNTO 0);

 num7 : IN std\_logic\_vector (7 DOWNTO 0);

 sorted : OUT std\_logic

 );

END COMPONENT sort\_done;

BEGIN

labels: for I in 0 to 6 GENERATE

even:if I mod 2 = 0 generate

es:even\_sort

PORT MAP (clk, dir\_of\_sort, reset, even\_odd\_0(i),even\_odd\_1(i),even\_odd\_2(i),even\_odd\_3(i),even\_odd\_4(i),even\_odd\_5(i),

 even\_odd\_6(i),even\_odd\_7(i),even\_odd\_0(i+1),even\_odd\_1(i+1),even\_odd\_2(i+1)

 ,even\_odd\_3(i+1),even\_odd\_4(i+1),

 even\_odd\_5(i+1),even\_odd\_6(i+1),even\_odd\_7(i+1));

end generate even;

odd: if I mod 2 /= 0 generate

os:odd\_sort

PORT MAP (clk, dir\_of\_sort, reset, even\_odd\_0(

i),even\_odd\_1(i),even\_odd\_2(i),even\_odd\_3(i),even\_odd\_4(i),even\_odd\_5(i),

 even\_odd\_6(i),even\_odd\_7(i),even\_odd\_0(i+1),even\_odd\_1(i+1),even\_odd\_2(i+1)

 ,even\_odd\_3(i+1),even\_odd\_4(i+1),

 even\_odd\_5(i+1),even\_odd\_6(i+1),even\_odd\_7(i+1));

end generate odd;

end GENERATE labels;

sd:sort\_done

PORT MAP (clk =>clk ,dir\_of\_sort => dir\_of\_sort, reset => reset, sort\_en => sort\_en, num0 => out0, num1 => out1, num2 => out2, num3 => out3, num4 => out4,

 num5 => out5, num6 => out6, num7 => out7, sorted => sorted\_done);

sorted <= sorted\_done;

process(sort\_en,clk)

begin

if (sort\_en = '1') then

even\_odd\_0(0) <= in0;

even\_odd\_1(0) <= in1;

even\_odd\_2(0) <= in2;

even\_odd\_3(0) <= in3;

even\_odd\_4(0) <= in4;

even\_odd\_5(0) <= in5;

even\_odd\_6(0) <= in6;

even\_odd\_7(0) <= in7;

out0 <= even\_odd\_0(7);

out1 <= even\_odd\_1(7);

out2 <= even\_odd\_2(7);

out3 <= even\_odd\_3(7);

out4 <= even\_odd\_4(7);

out5 <= even\_odd\_5(7);

out6 <= even\_odd\_6(7);

out7 <= even\_odd\_7(7);

else

out0 <= "00000000";

out1 <= "00000000";

out2 <= "00000000";

out3 <= "00000000";

out4 <= "00000000";

out5 <= "00000000";

out6 <= "00000000";

out7 <= "00000000";

end if;

end process;

END ARCHITECTURE sort;

SORTER\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

entity sorter\_tb is

end sorter\_tb;

architecture behaviour of sorter\_tb is

component sorter is

PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 in0 : IN std\_logic\_vector (7 DOWNTO 0);

 in1 : IN std\_logic\_vector (7 DOWNTO 0);

 in2 : IN std\_logic\_vector(7 DOWNTO 0);

 in3 : IN std\_logic\_vector(7 DOWNTO 0);

 in4 : IN std\_logic\_vector (7 DOWNTO 0);

 in5 : IN std\_logic\_vector(7 DOWNTO 0);

 in6 : IN std\_logic\_vector(7 DOWNTO 0);

 in7 : IN std\_logic\_vector(7 DOWNTO 0);

 out0 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out1 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out2 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out3 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out4 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out5 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out6 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out7 : INOUT std\_logic\_vector(7 DOWNTO 0);

 sorted : OUT std\_logic

 );

end component;

signal clk,new\_set,reset,sort\_en,sorted : std\_logic := '0';

signal dir\_of\_sort:bit := '0';

signal in0,in1,in2,in3,in4,in5,in6,in7,out0,out1,out2,out3,out4,out5,out6,out7 :Std\_logic\_vector(7 downto 0);

signal clk\_period:time:= 10 ns;

begin

DUT: sorter port map (clk,dir\_of\_sort,new\_set,reset,sort\_en,in0,in1,in2,in3,in4,in5,in6,in7,out0,out1,out2,out3,out4,out5,out6,out7);

clkprocess:process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

stimuliprocess: process

begin

wait for clk\_period/2;

dir\_of\_sort <= '1';

new\_set <= '1';

reset <= '0';

sort\_en <= '1';

in0 <= "00100001";

in1 <= "00010011";

in2 <= "00001010";

in3 <= "00000111";

in4 <= "00011100";

in5 <= "00011101";

in6 <= "00001001";

in7 <= "00010110";

end process;

end behaviour;

Testbench:

library ieee;

use ieee.std\_logic\_1164.all;

entity testbench is

end testbench;

architecture behaviour of testbench is

signal clk,reset: std\_logic;

signal Cx: std\_logic\_vector(1 downto 0);

signal data\_out: std\_logic\_vector(7 downto 0);

signal Rd\_Memory: std\_logic;

-- outputs and inputs from the memory

signal memory\_class\_mbx\_out: std\_logic\_vector(7 downto 0);

signal memory\_cy\_out: std\_logic\_vector(1 downto 0);

signal memory\_wij\_out: std\_logic\_vector(5 downto 0);

signal memory\_wd\_out: std\_logic\_vector(7 downto 0);

signal memory\_cx\_in: std\_logic\_vector(1 downto 0);

signal Memory\_class\_mbx\_in : integer;

signal memory\_cy\_in: std\_logic\_vector(1 downto 0);

signal algorithm\_out: std\_logic;

constant clk\_period:time:= 10 ns;

component controller\_datapath is

port(clk,reset: in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

data\_out: out std\_logic\_vector(7 downto 0);

Rd\_Memory: out std\_logic;

-- outputs and inputs from the memory

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

algorithm\_out: in std\_logic;

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0));

end component;

begin

DUT: controller\_datapath port map (clk => clk, reset => reset, Cx => Cx

,data\_out => data\_out,Rd\_memory => Rd\_memory,memory\_class\_mbx\_out =>memory\_class\_mbx\_out,

memory\_cy\_out => memory\_cy\_out,memory\_wij\_out => memory\_wij\_out,memory\_wd\_out => memory\_wd\_out,memory\_cx\_in =>memory\_cx\_in,

 Memory\_class\_mbx\_in => Memory\_class\_mbx\_in, memory\_cy\_in => memory\_cy\_in,algorithm\_out => algorithm\_out );

clk\_process:process

begin

clk <='0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

stimuli\_process:process

begin

Cx <= "01";

reset <= '0';

wait for clk\_period/2;

end process;

Memory\_process: process

begin

wait for clk\_period;

memory\_class\_mbx\_out <= "00000110";

memory\_cy\_out <= "01";

memory\_wij\_out <= "000010";

memory\_wd\_out <= "00000111";

end process;

end behaviour;

WD\_REGISTER:

-- Wd\_load,memory\_wd

library ieee;

use ieee.std\_logic\_1164.all;

entity wd\_Register is

port ( wd\_load : in std\_logic;

memory\_wd\_in: in std\_logic\_vector(7 downto 0);

output: out std\_logic\_vector(7 downto 0);

done: out std\_logic

);

end Wd\_Register;

architecture behaviour of Wd\_Register is

signal i:integer :=0;

begin

process (wd\_load)

begin

if (wd\_load ='1') then

 if (i <8) then

 output <= memory\_wd\_in;

 done <= '0';

 i <= i+1;

 else

 done<= '1';

 end if;

end if;

end process;

end behaviour;

COMP:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY comp IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 num0\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num1\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num0\_out : OUT std\_logic\_vector (7 DOWNTO 0);

 num1\_out : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY comp;

ARCHITECTURE compare OF comp IS

BEGIN

 PROCESS (reset, clk)

 BEGIN

 -- reset everything to '0' when reset is asserted

 IF (reset = '1') THEN

 num0\_out <= (OTHERS => '0');

 num1\_out <= (OTHERS => '0');

 -- the flip-flops are sensitive to the rising edge of clk

 ELSIF (rising\_edge (clk)) THEN

 CASE dir\_of\_sort IS

 -- direction of sort is ascending

 WHEN '1' =>

 -- num0\_in is greater than num1\_in, so switch them

 IF (num0\_in(7 downto 2) > num1\_in (7 downto 2)) THEN

 num0\_out <= num1\_in;

 num1\_out <= num0\_in;

 -- num0\_in and num1\_in are in order

 ELSE

 num0\_out <= num0\_in;

 num1\_out <= num1\_in;

 END IF;

 -- direction of sort is descending

 WHEN '0' =>

 -- num0\_in is less than num1\_in, so switch them

 IF (num0\_in(7 downto 2) < num1\_in(7 downto 2)) THEN

 num0\_out <= num1\_in;

 num1\_out <= num0\_in;

 -- num0\_in and num1\_in are in order

 ELSE

 num0\_out <= num0\_in;

 num1\_out <= num1\_in;

 END IF;

 END CASE;

 END IF;

 END PROCESS;

END ARCHITECTURE compare;

COMPARATOR:

library ieee;

use ieee.std\_logic\_1164.all;

entity comparator is

port(comparator\_enable : in std\_logic;

 reg\_k\_out : in integer;

 downcounter\_out : in integer;

 comparator\_out : out std\_logic

);

end comparator;

architecture behaviour of comparator is

begin

process (comparator\_enable)

begin

if (comparator\_enable = '1') then

 if (reg\_k\_out = downcounter\_out) then

 comparator\_out <= '1';

 else

 comparator\_out <= '0';

 end if;

end if;

end process;

end behaviour;

CONTROLLER:

library ieee;

use ieee.std\_logic\_1164.all;

entity Controller is

port( Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,algorithm\_out: in std\_logic;

 Cx : in std\_logic\_vector(1 downto 0);

 Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : out std\_logic);

end Controller;

architecture behaviour of Controller is

TYPE state\_type is (INITIAL,MEMORY\_READ,LOAD\_COUNTER\_REGISTERS,ENABLE\_COUNTER,REG\_Mbx\_LOAD,LOAD\_CY\_REGISTER,COMPARATOR\_ENABLE\_STATE,BIT\_SHIFTING,MEMORY\_READ\_RESPONSE,OUTPUT); -- controller states

SIGNAL Present\_State,Next\_State: State\_Type;

begin

-- added a new signal "counter\_reset", "Mbx\_reset"

-- to the intial state to reset the counter in this state.

process(Clock)

begin

 if rising\_edge(Clock) then

 if Reset = '1' or algorithm\_out = '1' then

 Present\_State <= INITIAL;

 ELSE

 Present\_State <= Next\_State;

 end if;

 end if;

end process;

process(Present\_State)

begin

 case Present\_State is

 when INITIAL =>

 Counter\_Reset <= '1';

 Mbx\_Reset <= '1';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Clear\_Temp\_Reg <= '0';

 Load\_Temp\_Reg <= '1';

 Reg\_Cx\_Load <= '1';

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Rd\_Memory <= '0';

 when MEMORY\_READ =>

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '0';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when LOAD\_COUNTER\_REGISTERS => -- load from memory

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '1';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when ENABLE\_COUNTER =>

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '1';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Clear\_Temp\_Reg <= '1';

 Load\_Temp\_Reg <= '0';

 Reg\_Cx\_Load <= '0';

 when REG\_Mbx\_LOAD => -- load from counter

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '1';

 Reg\_Cx\_Load <= '0';

 When LOAD\_CY\_REGISTER =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '1';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 When COMPARATOR\_ENABLE\_STATE =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <='0' ; -- read mbx from memory

 -- added now

 Counter\_Load <='0' ;

 Wd\_Load <= '0';

 Comparator\_Enable <='1' ;

 Cy\_Load <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when BIT\_SHIFTING =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '1';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '1';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when MEMORY\_READ\_RESPONSE =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 -- Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0' ;

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Load\_Temp\_Reg <= '0';

 Clear\_Temp\_Reg <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '0';

 Mbx\_Load <='0' ;

 Clear\_Temp\_Reg <= '0';

 Load\_Temp\_Reg <= '0';

 Reg\_Cx\_Load <= '0';

 when OUTPUT =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <='1';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <='1';

 Load\_Temp\_Reg <= '0';

 Clear\_Temp\_Reg <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '0';

 Mbx\_Load <= '0';

 Clear\_Temp\_Reg <= '0';

 Load\_Temp\_Reg <= '0';

 Reg\_Cx\_Load <= '0';

 end case;

end process;

process(present\_State,clock,reset,Toggle,comparator\_out,sort\_done,Temp\_reg\_out,Cx)

begin

 case Present\_State is

 when INITIAL =>

 Next\_State <= MEMORY\_READ;

 when MEMORY\_READ =>

 if (Temp\_Reg\_Out = '1') then

 Next\_State <= LOAD\_COUNTER\_REGISTERS;

 else

 Next\_State <= LOAD\_CY\_REGISTER;

 end if;

 when LOAD\_COUNTER\_REGISTERS =>

 Next\_State <= ENABLE\_COUNTER;

 when ENABLE\_COUNTER =>

 Next\_State <= REG\_Mbx\_LOAD;

 when REG\_Mbx\_LOAD =>

 Next\_State <= MEMORY\_READ;

 when LOAD\_CY\_REGISTER =>

 if (sort\_done = '1') then

 Next\_State <= COMPARATOR\_ENABLE\_STATE;

 else

 next\_State <= LOAD\_CY\_REGISTER;

 end if ;

 when COMPARATOR\_ENABLE\_STATE =>

 if(Comparator\_Out = '0') then

 Next\_State <= ENABLE\_COUNTER;

 else

 Next\_State <= BIT\_SHIFTING;

 end if;

 when BIT\_SHIFTING =>

 Next\_State <= MEMORY\_READ\_RESPONSE;

 when MEMORY\_READ\_RESPONSE =>

 Next\_State <= OUTPUT;

 when OUTPUT =>

 if (Done = '1') then

 Next\_State <= INITIAL;

 else

 Next\_State <= BIT\_SHIFTING;

 end if;

 end case;

end process;

end behaviour;

CONTROLLER\_DATAPATH:

library ieee;

use ieee.std\_logic\_1164.all;

entity controller\_datapath is

port(clk,reset: in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

data\_out: out std\_logic\_vector(7 downto 0);

Rd\_Memory: out std\_logic;

-- outputs and inputs from the memory

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

algorithm\_out: in std\_logic;

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0)

);

end controller\_datapath;

architecture behaviour of controller\_datapath is

signal Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Toggle,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable :std\_logic;

signal Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out:std\_logic;

component controller is

port(Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,algorithm\_out: in std\_logic;

 Cx : in std\_logic\_vector(1 downto 0);

 Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : out std\_logic);

end component;

component datapath is

port(clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: out std\_logic;

data\_out:out std\_logic\_vector(7 downto 0);

-- memory input to the datapath

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0));

end component;

begin

DUT\_1: controller port map (Clock =>Clk ,Reset => Reset,Done =>Done,Comparator\_Out => Comparator\_Out,Sort\_Done => Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out,Cx => Cx,

Reg\_Cx\_Load => Reg\_Cx\_Load,Counter\_Load => Counter\_Load,Counter\_Reset => Counter\_Reset,Counter\_Enable => Counter\_Enable,Mbx\_Load => Mbx\_Load,Mbx\_Reset => Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter,Wd\_Load =>Wd\_Load,Comparator\_Enable => Comparator\_Enable,Cy\_Load => Cy\_Load,

Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Rd\_Memory => Rd\_Memory,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg => Clear\_Temp\_Reg,Toggle => Toggle,algorithm\_out => algorithm\_out

);

DUT\_2: datapath port map (clk => clk,reset => reset,Reg\_Cx\_Load =>Reg\_Cx\_Load ,Counter\_Load => Counter\_Load ,Counter\_Reset => Counter\_Reset ,Counter\_Enable =>Counter\_Enable ,Mbx\_Load => Mbx\_Load ,Mbx\_Reset=> Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter ,Wd\_Load =>Wd\_Load,

Comparator\_Enable =>Comparator\_Enable ,Cy\_Load =>Cy\_Load ,Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg =>Clear\_Temp\_Reg ,PISO\_enable=> PISO\_enable,

Cx => Cx,

Done => Done,Comparator\_Out =>Comparator\_Out ,Sort\_Done =>Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out, data\_out =>data\_out,

Memory\_Cx\_in => Memory\_Cx\_in,

Memory\_cy\_out => Memory\_cy\_out,

Memory\_cy\_in => Memory\_cy\_in ,

Memory\_wij\_out => Memory\_wij\_out,

Memory\_wd\_out => Memory\_wd\_out,

Memory\_class\_mbx\_out => Memory\_class\_mbx\_out,

Memory\_class\_mbx\_in => Memory\_class\_mbx\_in

);

end behaviour;

# Chapter 4

**CONTROLLER\_TB:**

library ieee;

use ieee.std\_logic\_1164.all;

entity controller\_tb is

end entity;

architecture behaviour of controller\_tb is

signal Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out:std\_logic;

signal Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : std\_logic;

signal Cx: std\_logic\_vector(1 downto 0);

constant clk\_period:time:= 10 ns;

constant temp\_period:time:= 450 ns;

signal algorithm\_out:Std\_logic;

component controller is

port( Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,algorithm\_out: in std\_logic;

 Cx : in std\_logic\_vector(1 downto 0);

 Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : out std\_logic);

end Component;

begin

DUT: controller port map (Clock =>Clock ,Reset => Reset,Done =>Done,Comparator\_Out => Comparator\_Out,Sort\_Done => Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out,Cx => Cx,

Reg\_Cx\_Load => Reg\_Cx\_Load,Counter\_Load => Counter\_Load,Counter\_Reset => Counter\_Reset,Counter\_Enable => Counter\_Enable,Mbx\_Load => Mbx\_Load,Mbx\_Reset => Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter,Wd\_Load =>Wd\_Load,Comparator\_Enable => Comparator\_Enable,Cy\_Load => Cy\_Load,

Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Rd\_Memory => Rd\_Memory,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg => Clear\_Temp\_Reg,Toggle => Toggle,algorithm\_out => algorithm\_out

);

clk\_process: process

begin

 Clock <= '0';

 wait for clk\_period/2;

 Clock <= '1';

 wait for Clk\_period/2;

end process;

stimuli\_process: process

begin

reset <= '1';

wait for clk\_period;

Reset <= '0';

temp\_reg\_out <= '1';

Comparator\_Out <= '1';

Done <= '1';

wait for temp\_period;

temp\_reg\_out <= '0';

reset <= '0';

Comparator\_Out <= '1';

Done <= '1';

wait for temp\_period;

end process;

end behaviour;

counter:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_Std.all;

entity counter is

port( Memory\_class\_mbx: in std\_logic\_vector(7 downto 0);

counter\_load,counter\_reset,reset,counter\_enable : in std\_logic;

counter\_out: out integer

);

end counter;

architecture behaviour of counter is

signal register\_counter :integer;

begin

 counter\_out <= register\_counter;

 process(counter\_load,counter\_enable,reset,counter\_reset)

 begin

 if (reset = '1' or counter\_reset = '1') then -- -- initial state or reset

 register\_counter <= 0;

 elsif (counter\_load = '1') then -- load the counter

 register\_counter <= to\_integer(unsigned(Memory\_class\_mbx));

 elsif (counter\_enable = '1') then

 register\_counter <= register\_counter - 1;

 end if;

 end process;

end behaviour;

DATAPATH:

library ieee;

use ieee.std\_logic\_1164.all;

entity datapath is

port(clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: out std\_logic;

data\_out:out std\_logic\_vector(7 downto 0);

-- memory input to the datapath

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0)

);

end datapath;

architecture behaviour of datapath is

--signal clock,reset,Cx,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : std\_logic;

--signal Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,data\_out: std\_logic;

signal Reg\_cy\_output: std\_logic\_vector(15 downto 0);

signal counter\_out:integer;

signal reg\_K\_out:integer;

begin

DUT\_0: entity work.reg\_k

port map(

reg\_k\_load => reg\_k\_load,

 Reg\_k\_out => Reg\_k\_out

);

DUT\_1: entity work.reg\_cx

port map (

 clk => clk,

 reg\_cx\_load => reg\_cx\_load,

 reg\_cx\_in => Cx ,

 Memory\_in => Memory\_cx\_in

);

DUT\_2: entity work.counter

port map(

Memory\_class\_mbx => Memory\_class\_mbx\_out,

counter\_load =>counter\_load,

counter\_reset => counter\_reset,

reset => reset,

counter\_enable => counter\_enable,

counter\_out => counter\_out

);

DUT\_3: entity work.reg\_mbx

port map(

 reset => reset,

 Mbx\_Reset => Mbx\_Reset,

 Mbx\_load => Mbx\_load,

 downcounter\_out => counter\_out,

 memory\_responseclass => memory\_class\_mbx\_in

);

DUT\_4: entity work.temp\_reg

port map(

load\_temp\_reg => load\_temp\_reg,

clear\_temp\_reg => clear\_temp\_reg,

temp\_reg\_out => temp\_reg\_out

);

DUT\_5: entity work.comparator

port map (

comparator\_enable => comparator\_enable,

 reg\_k\_out => reg\_k\_out,

 downcounter\_out => counter\_out,

 comparator\_out => comparator\_out);

DUT\_6:entity work.cy\_register

port map( clk => clk,

reset => reset,

cy\_load => cy\_load,

Memory\_cy => Memory\_cy\_out,

Memory\_wij => Memory\_wij\_out,

sorted => sort\_done,

output => Reg\_cy\_output

);

DUT\_7: entity work.PISO

port map (

clk => clk,

input=> Reg\_cy\_output , -- output from the register cy as input to PISO

PISO\_enable => shift\_enable,

output => Memory\_cy\_in -- output from PISO as input to Memory

);

DUT\_8: entity work.wd\_register

port map(

wd\_load => Wd\_Load,

memory\_wd\_in =>Memory\_Wd\_out,

output => data\_out,

done => Done

);

--DUT3: entity work.Memory

--port map(

--

--);

end behaviour;

DATAPATH\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

entity datapath\_tb is

end datapath\_tb;

architecture behaviour of datapath\_tb is

component datapath is

port(clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: out std\_logic;

data\_out:out std\_logic\_vector(7 downto 0);

-- memory input to the datapath

Memory\_Cx\_in,Memory\_cy\_out,Memory\_cy\_in: inout std\_logic\_vector(1 downto 0);

Memory\_wij\_out: inout std\_logic\_vector(5 downto 0);

Memory\_wd\_in : inout std\_logic\_vector(7 downto 0);

Memory\_class\_mbx\_out : inout std\_logic\_vector(7 downto 0);

Memory\_class\_mbx\_in: inout integer );

end component;

signal clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : std\_logic:='0';

signal Cx: std\_logic\_vector(1 downto 0);

signal Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: std\_logic:='0';

signal data\_out: std\_logic\_vector(7 downto 0):="00000000";

-- memory signals

signal Memory\_Cx\_in,Memory\_cy\_out,Memory\_cy\_in: std\_logic\_vector(1 downto 0);

signal Memory\_wij\_out: std\_logic\_vector(5 downto 0);

signal Memory\_wd\_in : std\_logic\_vector(7 downto 0);

signal Memory\_class\_mbx\_out : std\_logic\_vector(7 downto 0);

signal Memory\_class\_mbx\_in: integer;

constant clk\_period:time:= 10 ns;

begin

DUT: datapath port map (

clk => clk,reset => reset,Reg\_Cx\_Load =>Reg\_Cx\_Load ,Counter\_Load => Counter\_Load ,Counter\_Reset => Counter\_Reset ,Counter\_Enable =>Counter\_Enable ,Mbx\_Load => Mbx\_Load ,Mbx\_Reset=> Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter ,Wd\_Load =>Wd\_Load,

Comparator\_Enable =>Comparator\_Enable ,Cy\_Load =>Cy\_Load ,Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Rd\_Memory => Rd\_Memory,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg =>Clear\_Temp\_Reg ,PISO\_enable=> PISO\_enable,

Cx => Cx,

Done => Done,Comparator\_Out =>Comparator\_Out ,Sort\_Done =>Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out, data\_out =>data\_out,

Memory\_Cx\_in => Memory\_Cx\_in,

Memory\_cy\_out => Memory\_cy\_out,

Memory\_cy\_in => Memory\_cy\_in ,

Memory\_wij\_out => Memory\_wij\_out,

Memory\_wd\_in => Memory\_wd\_in,

Memory\_class\_mbx\_out => Memory\_class\_mbx\_out,

Memory\_class\_mbx\_in => Memory\_class\_mbx\_in

);

clk\_process: process

begin

clk <= '0';

 wait for clk\_period/2;

 clk <= '1';

 wait for clk\_period/2;

end process;

stimuli\_process: process

begin

reset <= '0';

reg\_cx\_load <= '1';

load\_temp\_reg <= '1';

clear\_temp\_reg <= '0';

reg\_k\_load <= '1';

counter\_reset <= '1';

Cx <= "01";

wait for clk\_period;

Memory\_class\_mbx\_out <= "00000011";

reg\_cx\_load <= '0';

counter\_reset <= '0';

wait for clk\_period;

counter\_load <= '1';

reg\_k\_load <= '1';

wait for clk\_period;

counter\_load <= '0';

counter\_enable <= '1';

load\_temp\_reg <= '0';

clear\_temp\_reg <= '1';

wait for clk\_period;

counter\_enable <='0';

mbx\_load <= '1';

reg\_k\_load <= '1';

wait for clk\_period;

memory\_cy\_out <= "01";

memory\_wij\_out <= "000010";

wait for clk\_period;

cy\_load <= '1';

mbx\_load <='0';

wait for clk\_period;

cy\_load <= '0';

comparator\_enable <= '1';

wait for clk\_period;

counter\_load <= '0';

counter\_enable <= '1';

load\_temp\_reg <= '0';

clear\_temp\_reg <= '1';

wait for clk\_period;

counter\_enable <='0';

mbx\_load <= '1';

reg\_k\_load <= '1';

wait for clk\_period;

memory\_cy\_out <= "10";

memory\_wij\_out <= "000011";

wait for clk\_period;

cy\_load <= '1';

mbx\_load <='0';

wait for clk\_period;

cy\_load <= '0';

comparator\_enable <= '1';

wait for clk\_period;

shift\_enable <= '1';

comparator\_enable <= '0';

end process;

end behaviour;

EVENSORT:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY even\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 even\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY even\_sort;

ARCHITECTURE even\_sorter OF even\_sort IS

-- use this module to perform data comparison

COMPONENT comp IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 num0\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num1\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num0\_out : OUT std\_logic\_vector (7 DOWNTO 0);

 num1\_out : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT comp;

BEGIN

-- sort data0 and data1

even\_comp1:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in0, even\_in1, even\_out0,

 even\_out1);

-- sort data2 and data3

even\_comp2:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in2, even\_in3, even\_out2,

 even\_out3);

-- sort data4 and data5

even\_comp3:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in4, even\_in5, even\_out4,

 even\_out5);

-- sort data6 and data7

even\_comp4:comp

PORT MAP (clk, dir\_of\_sort, reset, even\_in6, even\_in7, even\_out6,

 even\_out7);

END ARCHITECTURE even\_sorter;

LOAD\_TEMP\_REG:

library ieee;

use ieee.std\_logic\_1164.all;

entity temp\_reg is

port(load\_temp\_reg,clear\_temp\_reg : in std\_logic;

 temp\_reg\_out : out std\_logic

);

end temp\_reg;

architecture behaviour of temp\_reg is

begin

 process(load\_temp\_reg,clear\_temp\_reg)

 begin

 if (load\_temp\_reg /= clear\_temp\_reg) then

 if (load\_temp\_reg = '1' and clear\_temp\_reg = '0' ) then

 temp\_reg\_out <= '1';

 elsif (load\_temp\_reg <= '0' and clear\_temp\_reg = '1') then

 temp\_reg\_out <= '0';

 end if;

 else

 temp\_reg\_out <= 'X';

 end if;

 end process;

end behaviour;

NEW\_SORT:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY new\_sort IS

 PORT (

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 old\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 old\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 new\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 mux\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 mux\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY new\_sort;

ARCHITECTURE mux OF new\_sort IS

BEGIN

 PROCESS (reset, new\_set, new\_in0, new\_in1, new\_in2, new\_in3, new\_in4,

 new\_in5, new\_in6, new\_in7, old\_in0, old\_in1, old\_in2,

 old\_in3, old\_in4, old\_in5, old\_in6, old\_in7)

 BEGIN

 -- reset everything to '0' when reset is asserted

 IF (reset = '1') THEN

 mux\_out0 <= (OTHERS => '0');

 mux\_out1 <= (OTHERS => '0');

 mux\_out2 <= (OTHERS => '0');

 mux\_out3 <= (OTHERS => '0');

 mux\_out4 <= (OTHERS => '0');

 mux\_out5 <= (OTHERS => '0');

 mux\_out6 <= (OTHERS => '0');

 mux\_out7 <= (OTHERS => '0');

 ELSE

 -- if new\_set is asserted, the mux output should contain the

 -- new set of data

 IF (new\_set = '1') THEN

 mux\_out0 <= new\_in0;

 mux\_out1 <= new\_in1;

 mux\_out2 <= new\_in2;

 mux\_out3 <= new\_in3;

 mux\_out4 <= new\_in4;

 mux\_out5 <= new\_in5;

 mux\_out6 <= new\_in6;

 mux\_out7 <= new\_in7;

 -- otherwise, let the old data coming out of odd sort through

 ELSE

 mux\_out0 <= old\_in0;

 mux\_out1 <= old\_in1;

 mux\_out2 <= old\_in2;

 mux\_out3 <= old\_in3;

 mux\_out4 <= old\_in4;

 mux\_out5 <= old\_in5;

 mux\_out6 <= old\_in6;

 mux\_out7 <= old\_in7;

 END IF;

 END IF;

 END PROCESS;

END ARCHITECTURE mux;

ODDSORT:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY odd\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 odd\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY odd\_sort;

ARCHITECTURE odd\_sorter OF odd\_sort IS

-- use this module to perform data comparison

COMPONENT comp IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 num0\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num1\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num0\_out : OUT std\_logic\_vector (7 DOWNTO 0);

 num1\_out : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT comp;

BEGIN

 PROCESS (reset, clk)

 BEGIN

 -- reset everything to '0' if reset is asserted

 IF (reset = '1') THEN

 odd\_out0 <= (OTHERS => '0');

 odd\_out7 <= (OTHERS => '0');

 -- send data0 and data7 straight through

 ELSIF (rising\_edge (clk)) THEN

 odd\_out0 <= odd\_in0;

 odd\_out7 <= odd\_in7;

 END IF;

 END PROCESS;

-- sort data1 and data2

odd\_comp1:comp

PORT MAP (clk, dir\_of\_sort, reset, odd\_in1, odd\_in2, odd\_out1,

 odd\_out2);

-- sort data3 and data4

odd\_comp2:comp

PORT MAP (clk, dir\_of\_sort, reset, odd\_in3, odd\_in4, odd\_out3,

 odd\_out4);

-- sort data5 and data6

odd\_comp3:comp

PORT MAP (clk, dir\_of\_sort, reset, odd\_in5, odd\_in6, odd\_out5,

 odd\_out6);

END ARCHITECTURE odd\_sorter;

PISO:

library ieee;

use ieee.std\_logic\_1164.all;

entity PISO is

port(clk: in std\_logic;

input: in std\_logic\_vector(0 to 15); -- output from the register cy

PISO\_enable: in std\_logic;

output: out std\_logic\_vector(0 to 1)

);

end PISO;

architecture behaviour of PISO is

signal i:integer:=0;

begin

process(clk)

begin

 if(rising\_Edge(clk)) then

 if(PISO\_enable = '1') then

 if( i <= 15) then

 output <= input(i to i+1);

 i <= i+1+1;

 end if;

 end if;

 end if;

end process;

end behaviour;

REG\_cX:

library ieee;

use ieee.std\_logic\_1164.all;

entity reg\_cx is

port ( clk: in std\_logic;

 reg\_cx\_load : in std\_logic;

 reg\_cx\_in : in std\_logic\_vector(1 downto 0);

 Memory\_in : out std\_logic\_vector(1 downto 0)

);

end reg\_cx;

architecture behaviour of reg\_cx is

begin

 process(clk)

 begin

 if (reg\_cx\_load = '1') then

 Memory\_in <= reg\_cx\_in;

 end if;

 end process;

end behaviour;

REG\_K:

library ieee;

use ieee.std\_logic\_1164.all;

entity reg\_k is

port (reg\_k\_load : in std\_logic;

 Reg\_k\_out : out integer

 );

end reg\_k;

architecture behaviour of reg\_k is

signal register\_k : integer;

begin

 process(reg\_k\_load)

 begin

 if (reg\_k\_load = '1' ) then

 register\_k <= 1;

 else

 register\_k <= 1;

 end if;

 Reg\_k\_out <= register\_k;

 end process;

end behaviour;

REG\_MBX:

library ieee;

use ieee.std\_logic\_1164.all;

entity reg\_mbx is

port ( reset: in std\_logic;

 Mbx\_Reset: in std\_logic;

 Mbx\_load : in std\_logic;

 downcounter\_out: in integer;

 memory\_responseclass: out integer

);

end reg\_mbx;

architecture behaviour of reg\_mbx is

signal Register\_Mbx : integer;

begin

 process(Mbx\_load,reset,Mbx\_Reset,downcounter\_out)

 begin

 if (reset = '1' or Mbx\_Reset = '1') then -- initial state or reset

 Register\_Mbx <= 0;

 elsif (Mbx\_load = '1') then

 Memory\_responseclass <= downcounter\_out;

 end if;

 end process;

end behaviour;

REGISTER:

library ieee;

use ieee.std\_logic\_1164.all;

entity cy\_register is

port(

clk : in std\_logic;

reset: in std\_logic;

cy\_load :in std\_logic;

Memory\_cy: in std\_logic\_vector(1 downto 0);

Memory\_wij: in std\_logic\_vector(5 downto 0);

sorted: out std\_logic;

output: out std\_logic\_vector(15 downto 0)

);

end cy\_register;

architecture behavioural of cy\_register is

type lutable\_1 is array(0 to 7) of std\_logic\_vector(1 downto 0);

type lutable\_2 is array(0 to 7) of std\_logic\_vector(5 downto 0);

signal sample\_array\_1: lutable\_1:= ((others =>(others => '0'))) ;

signal sample\_array\_2:lutable\_2:= ((others =>(others => '0'))) ;

signal i: integer:=0;

signal sort\_en: std\_logic :='0';

signal dir\_of\_sort: bit := '1';

signal new\_set: std\_logic := '0';

signal sorter\_in0: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in1:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in2:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in3: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in4: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in5: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in6: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_in7: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out0:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out1:std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out2: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out3: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out4: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out5: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out6: std\_logic\_vector(7 downto 0) := "00000000";

signal sorter\_out7: std\_logic\_vector(7 downto 0) := "00000000";

signal sorted\_signal: std\_logic :='0';

signal out0: std\_logic\_vector(1 downto 0) := "00";

signal out1: std\_logic\_vector(1 downto 0):= "00";

signal out2: std\_logic\_vector(1 downto 0) := "00";

signal out3: std\_logic\_vector(1 downto 0) := "00";

signal out4: std\_logic\_vector(1 downto 0) := "00";

signal out5: std\_logic\_vector(1 downto 0) := "00";

signal out6: std\_logic\_vector(1 downto 0) := "00";

signal out7: std\_logic\_vector(1 downto 0) := "00";

COMPONENT sorter is

PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 in0 : IN std\_logic\_vector (7 DOWNTO 0);

 in1 : IN std\_logic\_vector (7 DOWNTO 0);

 in2 : IN std\_logic\_vector (7 DOWNTO 0);

 in3 : IN std\_logic\_vector (7 DOWNTO 0);

 in4 : IN std\_logic\_vector (7 DOWNTO 0);

 in5 : IN std\_logic\_vector (7 DOWNTO 0);

 in6 : IN std\_logic\_vector (7 DOWNTO 0);

 in7 : IN std\_logic\_vector (7 DOWNTO 0);

 out0 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out1 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out2 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out3 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out4 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out5 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out6 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out7 : INOUT std\_logic\_vector (7 DOWNTO 0);

 sorted : OUT std\_logic

 );

END COMPONENT sorter;

begin

s:sorter

PORT MAP (clk,dir\_of\_sort,new\_set,reset,sort\_en,sorter\_in0,sorter\_in1,sorter\_in2,sorter\_in3,sorter\_in4,sorter\_in5,sorter\_in6,sorter\_in7,sorter\_out0,sorter\_out1,sorter\_out2,sorter\_out3,sorter\_out4,sorter\_out5,sorter\_out6,sorter\_out7,sorted\_signal);

sorted <= sorted\_signal;

 out0 <= sorter\_out0(1 downto 0);

 out1 <= sorter\_out1(1 downto 0);

 out2 <= sorter\_out2(1 downto 0);

 out3 <= sorter\_out3(1 downto 0);

 out4 <= sorter\_out4(1 downto 0);

 out5 <= sorter\_out5(1 downto 0);

 out6 <= sorter\_out6(1 downto 0);

 out7 <= sorter\_out7(1 downto 0);

output <= out0&out1&out2&out3&out4&out5&out6&out7;

process (cy\_load,clk)

begin

 if (rising\_Edge(cy\_load)) then

 if (i <= 7) then

 sample\_array\_1(i) <= memory\_cy;

 sample\_array\_2(i) <= memory\_wij;

 i <= i+1;

 end if;

 end if;

 if (rising\_Edge(clk)) then

 sorter\_in0 <= sample\_array\_2(0) & sample\_array\_1(0);

 sorter\_in1 <= sample\_array\_2(1) & sample\_array\_1(1);

 sorter\_in2 <= sample\_array\_2(2) & sample\_array\_1(2);

 sorter\_in3 <= sample\_array\_2(3) & sample\_array\_1(3);

 sorter\_in4 <= sample\_array\_2(4) & sample\_array\_1(4);

 sorter\_in5 <= sample\_array\_2(5) & sample\_array\_1(5);

 sorter\_in6 <= sample\_array\_2(6) & sample\_array\_1(6);

 sorter\_in7 <= sample\_array\_2(7) & sample\_array\_1(7);

 sort\_en <= '1';

 new\_set <= '1';

 end if;

end process;

end behavioural;

REGISTER\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

entity register\_tb is

end register\_tb;

architecture behaviour of register\_tb is

component cy\_register is

port(

clk : in std\_logic;

reset: in std\_logic;

cy\_load :in std\_logic;

Memory\_cy: in std\_logic\_vector(1 downto 0);

Memory\_wij: in std\_logic\_vector(5 downto 0);

sorted: out std\_logic;

output: out std\_logic\_vector(15 downto 0)

-- reg\_cy\_out : out std\_logic\_vector(n+2 downto 0);

);

end component;

signal clk,reset,cy\_load:std\_logic:='0';

signal Memory\_cy:std\_logic\_vector(1 downto 0):="00";

signal Memory\_wij:std\_logic\_vector(5 downto 0):="000000";

signal sorted:std\_logic:= '0';

signal output :std\_logic\_vector(15 downto 0) :="0000000000000000";

constant clk\_period:time:= 10 ns;

begin

dut: cy\_register port map (

clk => clk,reset => reset,cy\_load => cy\_load,Memory\_cy => Memory\_cy ,Memory\_wij =>Memory\_wij ,sorted => sorted,output =>output );

clk\_process: process

begin

 clk <= '0';

 wait for clk\_period/2;

 clk <= '1';

 wait for clk\_period/2;

end process;

stimuli: process

begin

reset <= '0';

cy\_load <= '0';

wait for clk\_period/2;

memory\_cy <= "01";

memory\_wij <= "001000";

cy\_load <= '1';

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "11";

memory\_wij <= "000100";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "10";

memory\_wij <= "000010";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "11";

memory\_wij <= "000001";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "00";

memory\_wij <= "000111";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "01";

memory\_wij <= "000111";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "01";

memory\_wij <= "000010";

wait for clk\_period/2;

cy\_load <= '0';

wait for clk\_period/2;

cy\_load <= '1';

memory\_cy <= "10";

memory\_wij <= "000101";

wait for clk\_period/2;

end process;

end behaviour;

SORTDONE:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY sort\_done IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 num0 : IN std\_logic\_vector (7 DOWNTO 0);

 num1 : IN std\_logic\_vector (7 DOWNTO 0);

 num2 : IN std\_logic\_vector (7 DOWNTO 0);

 num3 : IN std\_logic\_vector (7 DOWNTO 0);

 num4 : IN std\_logic\_vector (7 DOWNTO 0);

 num5 : IN std\_logic\_vector (7 DOWNTO 0);

 num6 : IN std\_logic\_vector (7 DOWNTO 0);

 num7 : IN std\_logic\_vector (7 DOWNTO 0);

 sorted : OUT std\_logic

 );

END ENTITY sort\_done;

ARCHITECTURE verify OF sort\_done IS

BEGIN

 PROCESS (reset, dir\_of\_sort, sort\_en, num0, num1, num2, num3, num4,

 num5, num6, num7,clk)

 BEGIN

 IF (reset = '1') THEN

 sorted <= '0';

 ELSE

 CASE dir\_of\_sort IS

 -- direction of sort is ascending (num0 = min, num7 = max),

 -- so check to see if all inputs are in desired order;

 -- assert sorted if sort\_en is asserted and all numbers are

 -- in order

 WHEN '1' =>

 IF (sort\_en = '1' AND num0(7 downto 2) <= num1(7 downto 2) AND num1(7 downto 2) <= num2(7 downto 2) AND

 num2(7 downto 2) <= num3(7 downto 2) AND num3(7 downto 2) <= num4(7 downto 2) AND num4(7 downto 2) <= num5(7 downto 2) AND

 num5(7 downto 2) <= num6(7 downto 2) AND num6(7 downto 2) <= num7(7 downto 2)) THEN

 sorted <= '1';

 ELSE

 sorted <= '0';

 END IF;

 -- direction of sort is descending (num0 = max, num7 = min),

 -- so check to see if all inputs are in desired order;

 -- assert sorted if sort\_en is asserted and all numbers are

 -- in order

 WHEN '0' =>

 IF (sort\_en = '1' AND num0(7 downto 2) >= num1(7 downto 2) AND num1(7 downto 2) >= num2(7 downto 2) AND

 num2(7 downto 2) >= num3(7 downto 2) AND num3(7 downto 2) >= num4(7 downto 2) AND num4(7 downto 2) >= num5(7 downto 2) AND

 num5(7 downto 2) >= num6(7 downto 2) AND num6(7 downto 2) >= num7(7 downto 2)) THEN

 sorted <= '1';

 ELSE

 sorted <= '0';

 END IF;

 END CASE;

 END IF;

 END PROCESS;

END ARCHITECTURE verify;

SORTER:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY sorter IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 in0 : IN std\_logic\_vector (7 DOWNTO 0):= "00000000";

 in1 : IN std\_logic\_vector (7 DOWNTO 0):= "00000000";

 in2 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in3 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in4 : IN std\_logic\_vector (7 DOWNTO 0):= "00000000";

 in5 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in6 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 in7 : IN std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out0 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out1 : INOUT std\_logic\_vector (7 DOWNTO 0):= "00000000";

 out2 : INOUT std\_logic\_vector (7 DOWNTO 0):= "00000000";

 out3 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out4 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out5 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out6 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 out7 : INOUT std\_logic\_vector(7 DOWNTO 0):= "00000000";

 sorted : OUT std\_logic

 );

END ENTITY sorter;

ARCHITECTURE sort OF sorter IS

type signals is array(0 to 7) of std\_logic\_vector(7 downto 0);

signal even\_odd\_0,even\_odd\_1,even\_odd\_2,even\_odd\_3,even\_odd\_4,even\_odd\_5,even\_odd\_6,even\_odd\_7 : signals;

signal sorted\_done:std\_logic;

signal I: integer;

COMPONENT even\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 even\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 even\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 even\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT even\_sort;

COMPONENT odd\_sort IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 odd\_in0 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in1 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in2 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in3 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in4 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in5 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in6 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_in7 : IN std\_logic\_vector (7 DOWNTO 0);

 odd\_out0 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out1 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out2 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out3 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out4 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out5 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out6 : OUT std\_logic\_vector (7 DOWNTO 0);

 odd\_out7 : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END COMPONENT odd\_sort;

COMPONENT sort\_done IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 num0 : IN std\_logic\_vector (7 DOWNTO 0);

 num1 : IN std\_logic\_vector (7 DOWNTO 0);

 num2 : IN std\_logic\_vector (7 DOWNTO 0);

 num3 : IN std\_logic\_vector (7 DOWNTO 0);

 num4 : IN std\_logic\_vector (7 DOWNTO 0);

 num5 : IN std\_logic\_vector (7 DOWNTO 0);

 num6 : IN std\_logic\_vector (7 DOWNTO 0);

 num7 : IN std\_logic\_vector (7 DOWNTO 0);

 sorted : OUT std\_logic

 );

END COMPONENT sort\_done;

BEGIN

labels: for I in 0 to 6 GENERATE

even:if I mod 2 = 0 generate

es:even\_sort

PORT MAP (clk, dir\_of\_sort, reset, even\_odd\_0(i),even\_odd\_1(i),even\_odd\_2(i),even\_odd\_3(i),even\_odd\_4(i),even\_odd\_5(i),

 even\_odd\_6(i),even\_odd\_7(i),even\_odd\_0(i+1),even\_odd\_1(i+1),even\_odd\_2(i+1)

 ,even\_odd\_3(i+1),even\_odd\_4(i+1),

 even\_odd\_5(i+1),even\_odd\_6(i+1),even\_odd\_7(i+1));

end generate even;

odd: if I mod 2 /= 0 generate

os:odd\_sort

PORT MAP (clk, dir\_of\_sort, reset, even\_odd\_0(

i),even\_odd\_1(i),even\_odd\_2(i),even\_odd\_3(i),even\_odd\_4(i),even\_odd\_5(i),

 even\_odd\_6(i),even\_odd\_7(i),even\_odd\_0(i+1),even\_odd\_1(i+1),even\_odd\_2(i+1)

 ,even\_odd\_3(i+1),even\_odd\_4(i+1),

 even\_odd\_5(i+1),even\_odd\_6(i+1),even\_odd\_7(i+1));

end generate odd;

end GENERATE labels;

sd:sort\_done

PORT MAP (clk =>clk ,dir\_of\_sort => dir\_of\_sort, reset => reset, sort\_en => sort\_en, num0 => out0, num1 => out1, num2 => out2, num3 => out3, num4 => out4,

 num5 => out5, num6 => out6, num7 => out7, sorted => sorted\_done);

sorted <= sorted\_done;

process(sort\_en,clk)

begin

if (sort\_en = '1') then

even\_odd\_0(0) <= in0;

even\_odd\_1(0) <= in1;

even\_odd\_2(0) <= in2;

even\_odd\_3(0) <= in3;

even\_odd\_4(0) <= in4;

even\_odd\_5(0) <= in5;

even\_odd\_6(0) <= in6;

even\_odd\_7(0) <= in7;

out0 <= even\_odd\_0(7);

out1 <= even\_odd\_1(7);

out2 <= even\_odd\_2(7);

out3 <= even\_odd\_3(7);

out4 <= even\_odd\_4(7);

out5 <= even\_odd\_5(7);

out6 <= even\_odd\_6(7);

out7 <= even\_odd\_7(7);

else

out0 <= "00000000";

out1 <= "00000000";

out2 <= "00000000";

out3 <= "00000000";

out4 <= "00000000";

out5 <= "00000000";

out6 <= "00000000";

out7 <= "00000000";

end if;

end process;

END ARCHITECTURE sort;

SORTER\_TB:

library ieee;

use ieee.std\_logic\_1164.all;

entity sorter\_tb is

end sorter\_tb;

architecture behaviour of sorter\_tb is

component sorter is

PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 new\_set : IN std\_logic;

 reset : IN std\_logic;

 sort\_en : IN std\_logic;

 in0 : IN std\_logic\_vector (7 DOWNTO 0);

 in1 : IN std\_logic\_vector (7 DOWNTO 0);

 in2 : IN std\_logic\_vector(7 DOWNTO 0);

 in3 : IN std\_logic\_vector(7 DOWNTO 0);

 in4 : IN std\_logic\_vector (7 DOWNTO 0);

 in5 : IN std\_logic\_vector(7 DOWNTO 0);

 in6 : IN std\_logic\_vector(7 DOWNTO 0);

 in7 : IN std\_logic\_vector(7 DOWNTO 0);

 out0 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out1 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out2 : INOUT std\_logic\_vector (7 DOWNTO 0);

 out3 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out4 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out5 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out6 : INOUT std\_logic\_vector(7 DOWNTO 0);

 out7 : INOUT std\_logic\_vector(7 DOWNTO 0);

 sorted : OUT std\_logic

 );

end component;

signal clk,new\_set,reset,sort\_en,sorted : std\_logic := '0';

signal dir\_of\_sort:bit := '0';

signal in0,in1,in2,in3,in4,in5,in6,in7,out0,out1,out2,out3,out4,out5,out6,out7 :Std\_logic\_vector(7 downto 0);

signal clk\_period:time:= 10 ns;

begin

DUT: sorter port map (clk,dir\_of\_sort,new\_set,reset,sort\_en,in0,in1,in2,in3,in4,in5,in6,in7,out0,out1,out2,out3,out4,out5,out6,out7);

clkprocess:process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

stimuliprocess: process

begin

wait for clk\_period/2;

dir\_of\_sort <= '1';

new\_set <= '1';

reset <= '0';

sort\_en <= '1';

in0 <= "00100001";

in1 <= "00010011";

in2 <= "00001010";

in3 <= "00000111";

in4 <= "00011100";

in5 <= "00011101";

in6 <= "00001001";

in7 <= "00010110";

end process;

end behaviour;

Testbench:

library ieee;

use ieee.std\_logic\_1164.all;

entity testbench is

end testbench;

architecture behaviour of testbench is

signal clk,reset: std\_logic;

signal Cx: std\_logic\_vector(1 downto 0);

signal data\_out: std\_logic\_vector(7 downto 0);

signal Rd\_Memory: std\_logic;

-- outputs and inputs from the memory

signal memory\_class\_mbx\_out: std\_logic\_vector(7 downto 0);

signal memory\_cy\_out: std\_logic\_vector(1 downto 0);

signal memory\_wij\_out: std\_logic\_vector(5 downto 0);

signal memory\_wd\_out: std\_logic\_vector(7 downto 0);

signal memory\_cx\_in: std\_logic\_vector(1 downto 0);

signal Memory\_class\_mbx\_in : integer;

signal memory\_cy\_in: std\_logic\_vector(1 downto 0);

signal algorithm\_out: std\_logic;

constant clk\_period:time:= 10 ns;

component controller\_datapath is

port(clk,reset: in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

data\_out: out std\_logic\_vector(7 downto 0);

Rd\_Memory: out std\_logic;

-- outputs and inputs from the memory

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

algorithm\_out: in std\_logic;

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0));

end component;

begin

DUT: controller\_datapath port map (clk => clk, reset => reset, Cx => Cx

,data\_out => data\_out,Rd\_memory => Rd\_memory,memory\_class\_mbx\_out =>memory\_class\_mbx\_out,

memory\_cy\_out => memory\_cy\_out,memory\_wij\_out => memory\_wij\_out,memory\_wd\_out => memory\_wd\_out,memory\_cx\_in =>memory\_cx\_in,

 Memory\_class\_mbx\_in => Memory\_class\_mbx\_in, memory\_cy\_in => memory\_cy\_in,algorithm\_out => algorithm\_out );

clk\_process:process

begin

clk <='0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

stimuli\_process:process

begin

Cx <= "01";

reset <= '0';

wait for clk\_period/2;

end process;

Memory\_process: process

begin

wait for clk\_period;

memory\_class\_mbx\_out <= "00000110";

memory\_cy\_out <= "01";

memory\_wij\_out <= "000010";

memory\_wd\_out <= "00000111";

end process;

end behaviour;

WD\_REGISTER:

-- Wd\_load,memory\_wd

library ieee;

use ieee.std\_logic\_1164.all;

entity wd\_Register is

port ( wd\_load : in std\_logic;

memory\_wd\_in: in std\_logic\_vector(7 downto 0);

output: out std\_logic\_vector(7 downto 0);

done: out std\_logic

);

end Wd\_Register;

architecture behaviour of Wd\_Register is

signal i:integer :=0;

begin

process (wd\_load)

begin

if (wd\_load ='1') then

 if (i <8) then

 output <= memory\_wd\_in;

 done <= '0';

 i <= i+1;

 else

 done<= '1';

 end if;

end if;

end process;

end behaviour;

COMP:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY comp IS

 PORT (

 clk : IN std\_logic;

 dir\_of\_sort : IN bit;

 reset : IN std\_logic;

 num0\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num1\_in : IN std\_logic\_vector (7 DOWNTO 0);

 num0\_out : OUT std\_logic\_vector (7 DOWNTO 0);

 num1\_out : OUT std\_logic\_vector (7 DOWNTO 0)

 );

END ENTITY comp;

ARCHITECTURE compare OF comp IS

BEGIN

 PROCESS (reset, clk)

 BEGIN

 -- reset everything to '0' when reset is asserted

 IF (reset = '1') THEN

 num0\_out <= (OTHERS => '0');

 num1\_out <= (OTHERS => '0');

 -- the flip-flops are sensitive to the rising edge of clk

 ELSIF (rising\_edge (clk)) THEN

 CASE dir\_of\_sort IS

 -- direction of sort is ascending

 WHEN '1' =>

 -- num0\_in is greater than num1\_in, so switch them

 IF (num0\_in(7 downto 2) > num1\_in (7 downto 2)) THEN

 num0\_out <= num1\_in;

 num1\_out <= num0\_in;

 -- num0\_in and num1\_in are in order

 ELSE

 num0\_out <= num0\_in;

 num1\_out <= num1\_in;

 END IF;

 -- direction of sort is descending

 WHEN '0' =>

 -- num0\_in is less than num1\_in, so switch them

 IF (num0\_in(7 downto 2) < num1\_in(7 downto 2)) THEN

 num0\_out <= num1\_in;

 num1\_out <= num0\_in;

 -- num0\_in and num1\_in are in order

 ELSE

 num0\_out <= num0\_in;

 num1\_out <= num1\_in;

 END IF;

 END CASE;

 END IF;

 END PROCESS;

END ARCHITECTURE compare;

COMPARATOR:

library ieee;

use ieee.std\_logic\_1164.all;

entity comparator is

port(comparator\_enable : in std\_logic;

 reg\_k\_out : in integer;

 downcounter\_out : in integer;

 comparator\_out : out std\_logic

);

end comparator;

architecture behaviour of comparator is

begin

process (comparator\_enable)

begin

if (comparator\_enable = '1') then

 if (reg\_k\_out = downcounter\_out) then

 comparator\_out <= '1';

 else

 comparator\_out <= '0';

 end if;

end if;

end process;

end behaviour;

CONTROLLER:

library ieee;

use ieee.std\_logic\_1164.all;

entity Controller is

port( Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,algorithm\_out: in std\_logic;

 Cx : in std\_logic\_vector(1 downto 0);

 Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : out std\_logic);

end Controller;

architecture behaviour of Controller is

TYPE state\_type is (INITIAL,MEMORY\_READ,LOAD\_COUNTER\_REGISTERS,ENABLE\_COUNTER,REG\_Mbx\_LOAD,LOAD\_CY\_REGISTER,COMPARATOR\_ENABLE\_STATE,BIT\_SHIFTING,MEMORY\_READ\_RESPONSE,OUTPUT); -- controller states

SIGNAL Present\_State,Next\_State: State\_Type;

begin

-- added a new signal "counter\_reset", "Mbx\_reset"

-- to the intial state to reset the counter in this state.

process(Clock)

begin

 if rising\_edge(Clock) then

 if Reset = '1' or algorithm\_out = '1' then

 Present\_State <= INITIAL;

 ELSE

 Present\_State <= Next\_State;

 end if;

 end if;

end process;

process(Present\_State)

begin

 case Present\_State is

 when INITIAL =>

 Counter\_Reset <= '1';

 Mbx\_Reset <= '1';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Clear\_Temp\_Reg <= '0';

 Load\_Temp\_Reg <= '1';

 Reg\_Cx\_Load <= '1';

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Rd\_Memory <= '0';

 when MEMORY\_READ =>

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '0';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when LOAD\_COUNTER\_REGISTERS => -- load from memory

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '1';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when ENABLE\_COUNTER =>

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '1';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Clear\_Temp\_Reg <= '1';

 Load\_Temp\_Reg <= '0';

 Reg\_Cx\_Load <= '0';

 when REG\_Mbx\_LOAD => -- load from counter

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '1';

 Reg\_Cx\_Load <= '0';

 When LOAD\_CY\_REGISTER =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '0'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Wd\_Load <= '0';

 Comparator\_Enable <= '0';

 Cy\_Load <= '1';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 When COMPARATOR\_ENABLE\_STATE =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <='0' ; -- read mbx from memory

 -- added now

 Counter\_Load <='0' ;

 Wd\_Load <= '0';

 Comparator\_Enable <='1' ;

 Cy\_Load <= '0';

 Shift\_Enable <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when BIT\_SHIFTING =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '1';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '1';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <= '0';

 Reg\_K\_Load <= '1';

 Mbx\_Load <= '0';

 Reg\_Cx\_Load <= '0';

 when MEMORY\_READ\_RESPONSE =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 -- Load\_Enable\_Sorter <= '0';

 Wd\_Load <= '0' ;

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <= '0';

 Load\_Temp\_Reg <= '0';

 Clear\_Temp\_Reg <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '0';

 Mbx\_Load <='0' ;

 Clear\_Temp\_Reg <= '0';

 Load\_Temp\_Reg <= '0';

 Reg\_Cx\_Load <= '0';

 when OUTPUT =>

 -- no load\_enable\_sorter and enable\_sorter

 Rd\_memory <= '1'; -- read mbx from memory

 -- added now

 Counter\_Load <= '0';

 Load\_Enable\_Sorter <= '0';

 Wd\_Load <='1';

 Comparator\_Enable <= '0';

 Cy\_Load <= '0';

 Sorter\_Enable <= '0';

 Shift\_Enable <='1';

 Load\_Temp\_Reg <= '0';

 Clear\_Temp\_Reg <= '0';

 Counter\_Reset <= '0';

 Mbx\_Reset <= '0';

 Counter\_Enable <='0';

 Reg\_K\_Load <= '0';

 Mbx\_Load <= '0';

 Clear\_Temp\_Reg <= '0';

 Load\_Temp\_Reg <= '0';

 Reg\_Cx\_Load <= '0';

 end case;

end process;

process(present\_State,clock,reset,Toggle,comparator\_out,sort\_done,Temp\_reg\_out,Cx)

begin

 case Present\_State is

 when INITIAL =>

 Next\_State <= MEMORY\_READ;

 when MEMORY\_READ =>

 if (Temp\_Reg\_Out = '1') then

 Next\_State <= LOAD\_COUNTER\_REGISTERS;

 else

 Next\_State <= LOAD\_CY\_REGISTER;

 end if;

 when LOAD\_COUNTER\_REGISTERS =>

 Next\_State <= ENABLE\_COUNTER;

 when ENABLE\_COUNTER =>

 Next\_State <= REG\_Mbx\_LOAD;

 when REG\_Mbx\_LOAD =>

 Next\_State <= MEMORY\_READ;

 when LOAD\_CY\_REGISTER =>

 if (sort\_done = '1') then

 Next\_State <= COMPARATOR\_ENABLE\_STATE;

 else

 next\_State <= LOAD\_CY\_REGISTER;

 end if ;

 when COMPARATOR\_ENABLE\_STATE =>

 if(Comparator\_Out = '0') then

 Next\_State <= ENABLE\_COUNTER;

 else

 Next\_State <= BIT\_SHIFTING;

 end if;

 when BIT\_SHIFTING =>

 Next\_State <= MEMORY\_READ\_RESPONSE;

 when MEMORY\_READ\_RESPONSE =>

 Next\_State <= OUTPUT;

 when OUTPUT =>

 if (Done = '1') then

 Next\_State <= INITIAL;

 else

 Next\_State <= BIT\_SHIFTING;

 end if;

 end case;

end process;

end behaviour;

CONTROLLER\_DATAPATH:

library ieee;

use ieee.std\_logic\_1164.all;

entity controller\_datapath is

port(clk,reset: in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

data\_out: out std\_logic\_vector(7 downto 0);

Rd\_Memory: out std\_logic;

-- outputs and inputs from the memory

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

algorithm\_out: in std\_logic;

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0)

);

end controller\_datapath;

architecture behaviour of controller\_datapath is

signal Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Toggle,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable :std\_logic;

signal Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out:std\_logic;

component controller is

port(Clock,Reset,Toggle,Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out,algorithm\_out: in std\_logic;

 Cx : in std\_logic\_vector(1 downto 0);

 Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Rd\_Memory,Load\_Temp\_Reg,Clear\_Temp\_Reg : out std\_logic);

end component;

component datapath is

port(clk,reset,Reg\_Cx\_Load,Counter\_Load,Counter\_Reset,Counter\_Enable,Mbx\_Load,Mbx\_Reset,Load\_Enable\_Sorter,Wd\_Load,

Comparator\_Enable,Cy\_Load,Reg\_K\_Load,Sorter\_Enable,Shift\_Enable,Load\_Temp\_Reg,Clear\_Temp\_Reg,PISO\_enable : in std\_logic;

Cx: in std\_logic\_vector(1 downto 0);

Done,Comparator\_Out,Sort\_Done,Temp\_Reg\_Out: out std\_logic;

data\_out:out std\_logic\_vector(7 downto 0);

-- memory input to the datapath

memory\_class\_mbx\_out: in std\_logic\_vector(7 downto 0);

memory\_cy\_out: in std\_logic\_vector(1 downto 0);

memory\_wij\_out: in std\_logic\_vector(5 downto 0);

memory\_wd\_out:in std\_logic\_vector(7 downto 0);

memory\_cx\_in: out std\_logic\_vector(1 downto 0);

Memory\_class\_mbx\_in : out integer;

memory\_cy\_in: out std\_logic\_vector(1 downto 0));

end component;

begin

DUT\_1: controller port map (Clock =>Clk ,Reset => Reset,Done =>Done,Comparator\_Out => Comparator\_Out,Sort\_Done => Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out,Cx => Cx,

Reg\_Cx\_Load => Reg\_Cx\_Load,Counter\_Load => Counter\_Load,Counter\_Reset => Counter\_Reset,Counter\_Enable => Counter\_Enable,Mbx\_Load => Mbx\_Load,Mbx\_Reset => Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter,Wd\_Load =>Wd\_Load,Comparator\_Enable => Comparator\_Enable,Cy\_Load => Cy\_Load,

Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Rd\_Memory => Rd\_Memory,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg => Clear\_Temp\_Reg,Toggle => Toggle,algorithm\_out => algorithm\_out

);

DUT\_2: datapath port map (clk => clk,reset => reset,Reg\_Cx\_Load =>Reg\_Cx\_Load ,Counter\_Load => Counter\_Load ,Counter\_Reset => Counter\_Reset ,Counter\_Enable =>Counter\_Enable ,Mbx\_Load => Mbx\_Load ,Mbx\_Reset=> Mbx\_Reset,Load\_Enable\_Sorter =>Load\_Enable\_Sorter ,Wd\_Load =>Wd\_Load,

Comparator\_Enable =>Comparator\_Enable ,Cy\_Load =>Cy\_Load ,Reg\_K\_Load => Reg\_K\_Load,Sorter\_Enable => Sorter\_Enable,Shift\_Enable => Shift\_Enable,Load\_Temp\_Reg => Load\_Temp\_Reg,Clear\_Temp\_Reg =>Clear\_Temp\_Reg ,PISO\_enable=> PISO\_enable,

Cx => Cx,

Done => Done,Comparator\_Out =>Comparator\_Out ,Sort\_Done =>Sort\_Done,Temp\_Reg\_Out => Temp\_Reg\_Out, data\_out =>data\_out,

Memory\_Cx\_in => Memory\_Cx\_in,

Memory\_cy\_out => Memory\_cy\_out,

Memory\_cy\_in => Memory\_cy\_in ,

Memory\_wij\_out => Memory\_wij\_out,

Memory\_wd\_out => Memory\_wd\_out,

Memory\_class\_mbx\_out => Memory\_class\_mbx\_out,

Memory\_class\_mbx\_in => Memory\_class\_mbx\_in

);

end behaviour;