**Generic Multiplier**

**The purpose of this experiment is to write a generic 2s complement multiplier**

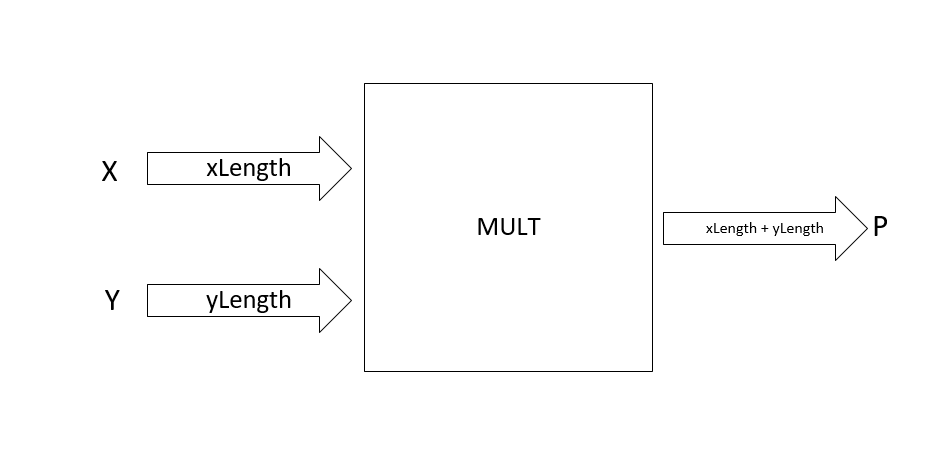
**Jacob Cox**

**1118901**

**Project Description:**

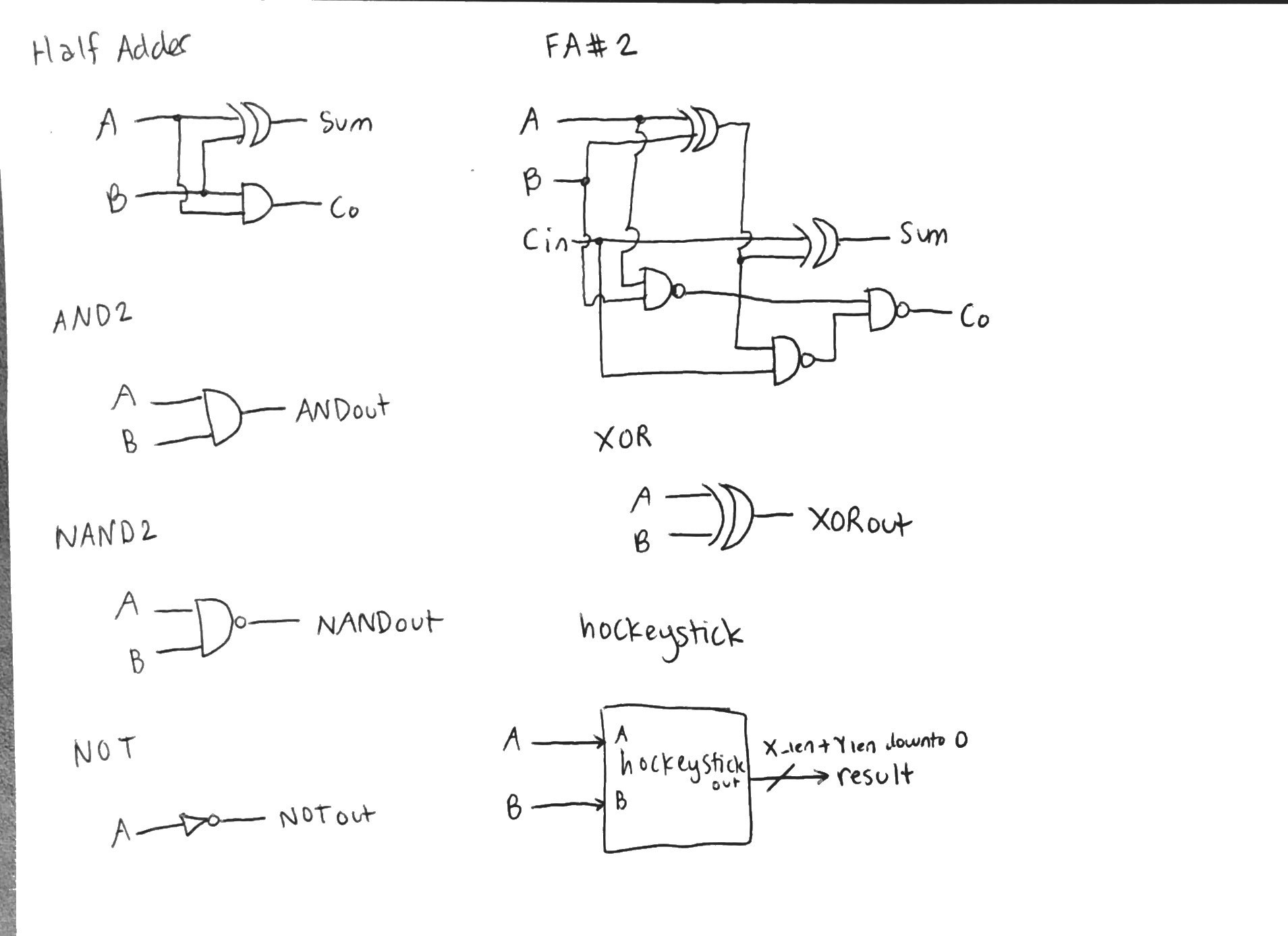
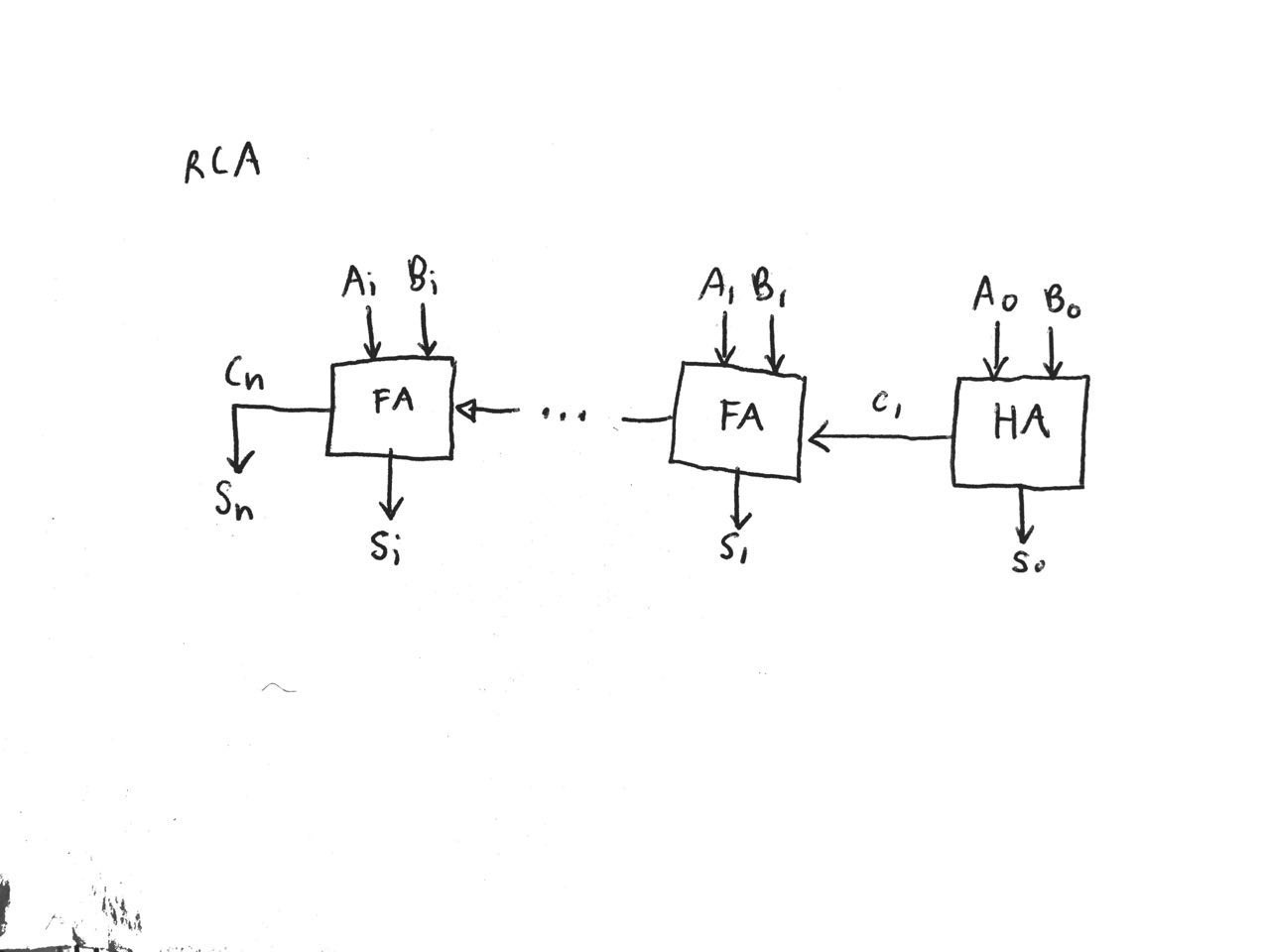
In VHDL we made an entity called MULT. MULT takes in X and Y and multiplies them using an array multiplier with the Baugh-Wooley algorithm. It outputs the result as a std\_logic\_vector with a length of Xlength + Ylength.

**Block Diagram of the multiply Chip:**

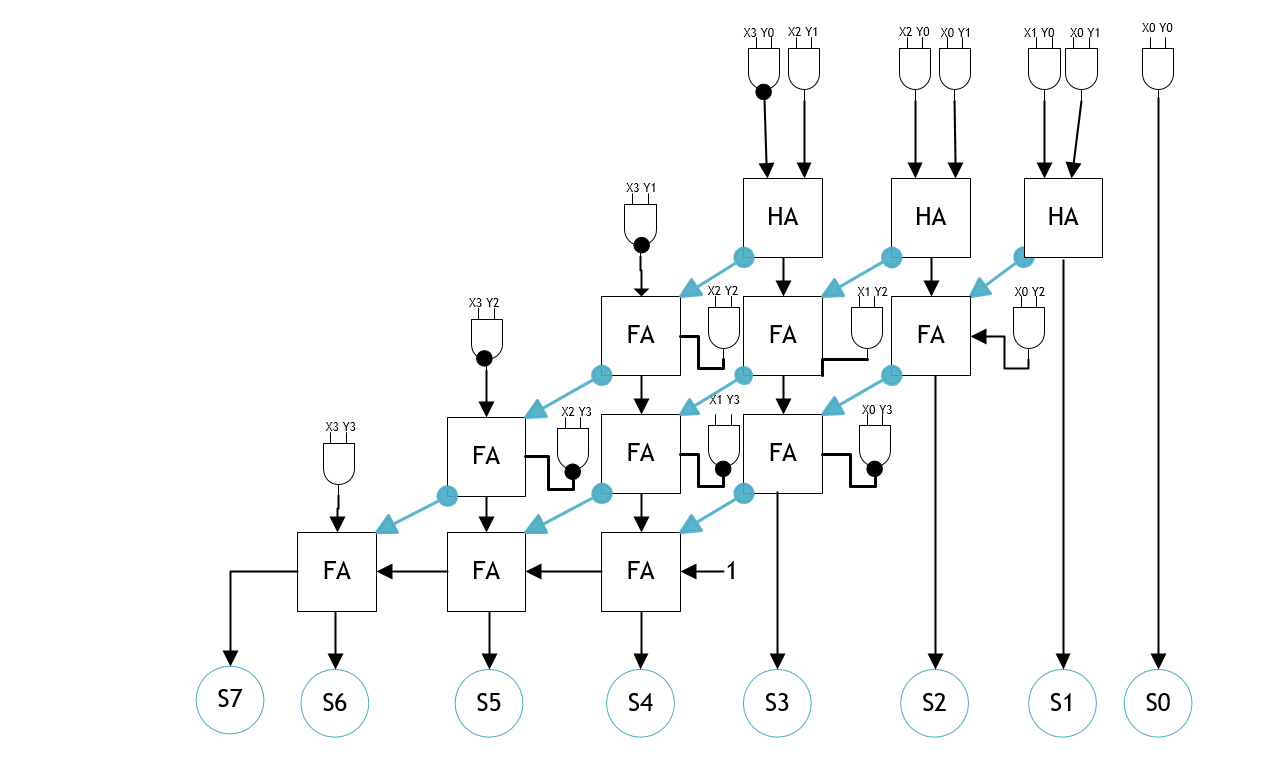


**Gate level design:**

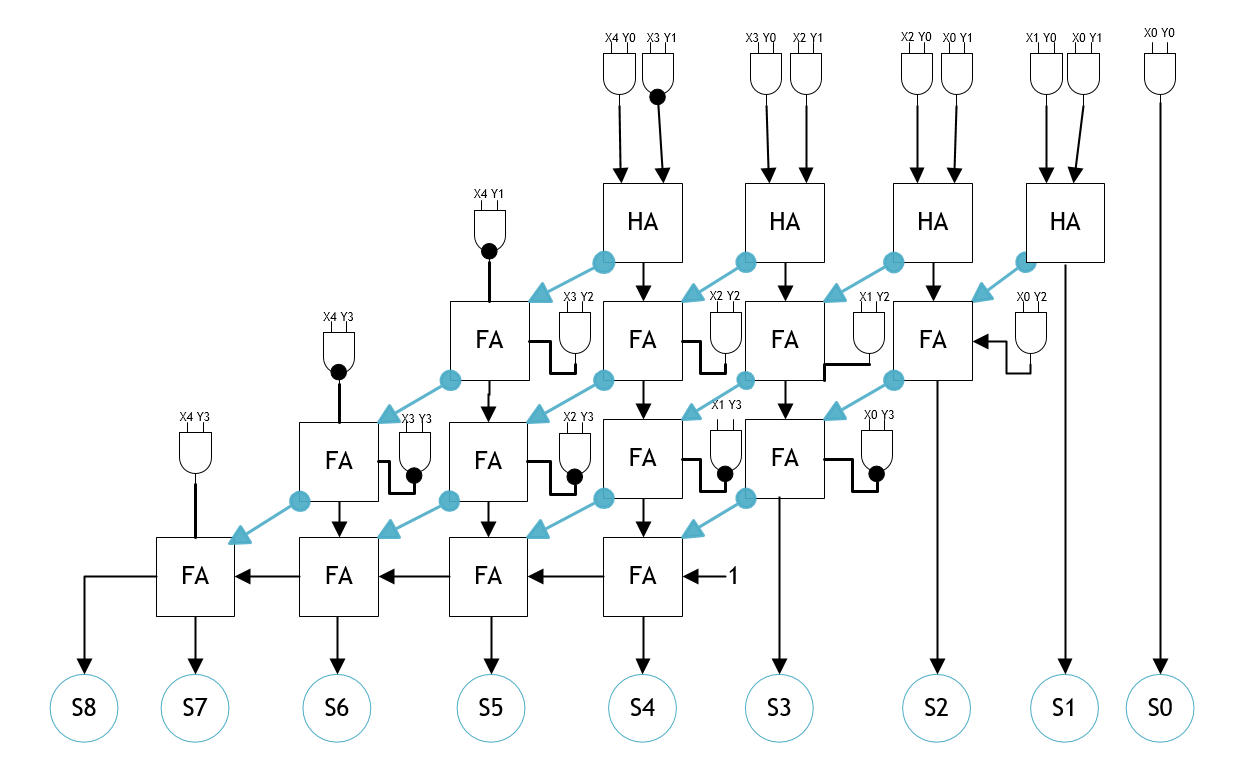




**Design of MULT (4X4)**



**Design of MULT (5X4)**



**MULT VHDL Code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

package arrayTypes is

type arrayType is array(natural range <>, natural range <>) of std\_logic;

end package arrayTypes;

------ end package ------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- AND ---------

entity myAND is

port(A, B: in std\_logic;

ANDout: out std\_logic);

end;

architecture behav of myAND is

begin

ANDout <= A and B;

end;

------- end AND ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- NAND ---------

entity myNAND is

port(A, B: in std\_logic;

NANDout: out std\_logic);

end;

architecture behav of myNAND is

begin

NANDout <= A nand B;

end;

------- end NAND ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- XOR ---------

entity myXOR is

port(A, B: in std\_logic;

XORout: out std\_logic);

end;

architecture behav of myXOR is

begin

XORout <= A xor B;

end;

------- end XOR ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- NOT ---------

entity myNOT is

port(A: in std\_logic;

NOTout: out std\_logic);

end;

architecture behav of myNOT is

begin

NOTout <= not A;

end;

------- end NOT ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- Half Adder ---------

entity HA is

port(A, B: in std\_logic;

Sum, Co: out std\_logic);

end;

architecture behav of HA is

begin

Sum <= A xor B;

Co <= B and A;

end;

------- end Half Adder ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- Full Adder ---------

entity FA is

port(A, B, Cin: in std\_logic;

Sum, Co: out std\_logic);

end;

architecture behav of FA is

signal x1 : std\_logic;

signal x2 : std\_logic;

signal n1 : std\_logic;

signal n2 : std\_logic;

begin

x1 <= A xor B;

Sum <= x1 xor Cin;

n1 <= not(A and B);

n2 <= not(x1 and Cin);

Co <= not(n1 and n2);

end;

------- end Full Adder ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

----------- RCA Adder ---------

-- arbitrarily sized RCA

-- only works if X and Y are same length

entity RCA is

generic (len: integer := 4) ;

port(X, Y : in std\_logic\_vector(len -1 downto 0);

Cin: in std\_logic;

S: out std\_logic\_vector(len-1 downto 0));

end;

architecture behav of RCA is

component HA

port(A, B: in std\_logic;

Sum, Co: out std\_logic);

end component;

component FA

port(A, B, Cin: in std\_logic;

Sum, Co: out std\_logic);

end component;

signal c: std\_logic\_vector(len downto 0) := (others => '0');

begin

c(0) <= Cin;

RCA\_gen: for i in 0 to len-1 generate

FA\_comp: FA port map (X(i), Y(i), c(i), S(i), c(i + 1));

end generate;

--S(len-1) <= c(len-1); --both were indexed to length, changed to 4

end;

------- end RCA Adder ---------

Library IEEE;

use IEEE.std\_logic\_1164.all;

library work;

use work.arrayTypes.all;

entity MULT is

-- resource: https://surf-vhdl.com/vhdl-syntax-web-course-surf-vhdl/vhdl-generics/

generic ( X\_Len, Y\_Len: integer := 4 ); --do we want to leave this set to 4?

port (x : in std\_logic\_vector(X\_Len-1 downto 0);

y : in std\_logic\_vector(Y\_Len-1 downto 0);

P : out std\_logic\_vector( X\_Len + Y\_Len-1 downto 0));

end MULT;

architecture struct of MULT is

component myAND

port(A, B: in std\_logic;

ANDout: out std\_logic);

end component;

component myNAND

port(A, B: in std\_logic;

NANDout: out std\_logic);

end component;

component myXOR

port(A, B: in std\_logic;

XORout: out std\_logic);

end component;

component myNOT

port(A: in std\_logic;

NOTout: out std\_logic);

end component;

component HA

port(A, B: in std\_logic;

Sum, Co: out std\_logic);

end component;

component FA

port(A, B, Cin: in std\_logic;

Sum, Co: out std\_logic);

end component;

component RCA

generic (len: integer := 4) ;

port(X, Y : in std\_logic\_vector(len -1 downto 0);

Cin: in std\_logic;

S: out std\_logic\_vector(len-1 downto 0));

end component;

constant xTop : integer := X'length -1;

constant yTop : integer := Y'length -1;

--initializing arrays

type PP is array(0 to yTop, 0 to xTop) of std\_logic;

signal inter\_product : PP; --Y rows, X columns

type Sum\_array is array(0 to Y\_Len, 0 to xTop) of std\_logic;

signal inter\_sum: Sum\_array;

type Carry\_array is array(0 to Y\_Len, 0 to xTop) of std\_logic;

signal inter\_carry: Carry\_array;

signal tempInterSum : arrayType(0 to Y\_Len, 0 to xTop);

signal tempInterCarry : arrayType(0 to Y\_Len, 0 to xTop);

signal tempBits : std\_logic\_vector(X\_Len+Y\_Len-1 downto 0) := (others => '0');

signal tempresult : std\_logic\_vector(X\_Len+Y\_Len-1 downto 0) := (others => '0');

signal result : std\_logic\_vector(X\_Len+Y\_Len-1 downto 0) := (others => '0');

signal carryo : std\_logic\_vector(0 to X\_Len+Y\_Len-1) := (others => '0');

begin

--generate 2d array of partial products

ppgen1: for i in 0 to yTop generate

ppgen2: for j in 0 to xTop generate

PP\_AND: if not(i /= yTop xor j /= xTop) generate

bob: myAND port map(Y(i),X(j),inter\_product(i,j));

end generate;

PP\_NAND: if (i /= yTop xor j /= xTop) generate

banana: myNAND port map(Y(i),X(j),inter\_product(i,j));

end generate;

end generate;

end generate;

inter\_sum(0,0) <= inter\_product(0,0);

--Half adder for loop

hagen1: for j in 1 to xTop generate

pineapple: HA port map(inter\_product(0, j), inter\_product(1, j-1), inter\_sum(1, j-1), inter\_carry(1, j));

end generate;

--Middle rows for loop

--inside FA loop

mgen1: for i in 2 to yTop generate

mgen2: for j in 0 to xTop-2 generate

mango: FA port map(inter\_product(i, j), inter\_sum(i-1, j+1), inter\_carry(i-1,j+1), inter\_sum(i,j), inter\_carry(i,j+1)) ;

end generate;

apple: FA port map(inter\_product(i, xTop-1), inter\_product(i-1, xTop), inter\_carry(i-1, xTop), inter\_sum(i, xTop-1), inter\_carry(i, xTop)) ;

end generate;

--last row for loop

-- NOT ANYMORE lonely full adder (half adder whose carry-in is always 1)

durian: FA port map (inter\_sum(yTop, 0), inter\_carry(yTop, 1), '0', inter\_sum(Y\_Len, 0), inter\_carry(Y\_Len, 1));

--inside FA loop

finalgen: for i in 1 to xTop-2 generate

grape: FA port map (inter\_sum(yTop, i), inter\_carry(yTop, i+1),inter\_carry(Y\_Len, i), inter\_sum(Y\_Len, i), inter\_carry(Y\_Len, i+1));

end generate;

-- lonely full adder (whose carry-out is special (bottom left most FA))

cherry: FA port map (inter\_product(yTop, xTop), inter\_carry(yTop, xTop), inter\_carry(Y\_Len, xTop-1) , inter\_sum(Y\_Len, xTop-1), inter\_sum(Y\_Len, xTop)); --the last term is the carry out which gets put into the sum array

-- tempInterSum <= arrayType(inter\_sum);

-- tempInterCarry <= arrayType(inter\_carry);

--get the result

zack: for i in 0 to Y\_Len generate --column 0

tempresult(i) <= inter\_sum(i,0); -- (Y row, X col)

end generate;

jake: for j in 1 to xTop generate

tempresult(j+y\_len) <= inter\_sum(Y\_Len,j);

end generate;

--make temp for adding 1's

addbits: if (xTop = yTop) generate

tempBits(Y\_Len) <= '1';

end generate;

addNotBits: if (xTop /= yTop) generate

tempBits(xTop) <= '1';

tempBits(yTop) <= '1';

tempBits(X\_Len+Y\_Len-1) <= '1';

end generate;

--add 1's

joe: RCA

generic map (result'length)

port map(tempresult, tempBits, '0', result);

P <= result;

end struct;

**VHDL Test Bench:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use std.textio.all;

use IEEE.std\_logic\_textio.all;

use IEEE.numeric\_std.all;

entity Mult\_tb is

end entity;

architecture TESTBENCH of Mult\_tb is

file output\_manager: TEXT open write\_mode is "SM\_TEST.txt";

constant top\_line: string := " X Y | Result";

constant middle\_line: string := "-----------------------------";

constant divider : string := "| ";

component MULT

generic ( X\_Len : integer := 4 ; Y\_Len: integer := 4 );

port ( X : in std\_logic\_vector(X\_Len-1 downto 0);

Y : in std\_logic\_vector(Y\_Len-1 Downto 0);

P : out std\_logic\_vector( X\_Len + Y\_Len -1 downto 0));

end component;

signal X\_thing: Std\_logic\_vector(4 downto 0); -- length of 5

signal Y\_thing: Std\_logic\_vector(4 downto 0); -- length of 4

signal z\_thing: Std\_logic\_vector(5 downto 0); -- length of 5

signal result1 : Std\_logic\_vector(X\_thing'length+ Y\_thing'length -1 downto 0);

signal result2 : Std\_logic\_vector(Z\_thing'length+ Y\_thing'length -1 downto 0);

begin

U1: MULT

generic map ( X\_thing'length , Y\_thing'length )

port map ( x\_thing,

Y\_thing,

result1);

U2: MULT

generic map ( z\_thing'length , y\_thing'length)

port map ( z\_thing,

Y\_thing,

result2);

DATA: process

variable L: line;

variable I, J: integer;

variable w : unsigned(x\_thing'high downto 0):= (others => '0');

variable v : unsigned(Y\_thing'high downto 0):= (others => '0');

variable u : unsigned(z\_thing'high downto 0):= (others => '0');

begin

-- write header

write(L, top\_line);

writeline(output\_manager,L);

write(L, middle\_line);

writeline(output\_manager,L);

for I in 0 to 15 loop

for J in 0 to 15 loop

X\_thing<= Std\_logic\_vector(w);

Y\_thing <= Std\_logic\_vector(v);

write(L, X\_thing, left, 10);

write(L, Y\_thing, left, 10 );

WRITE(L, divider, left, 3);

write(L, Result1, left, 15);

writeline(output\_manager,L);

v := v + "1";

wait for 10 ns;

end loop;

v:=(others => '0');

w := w + "1";

end loop;

write(L, middle\_line);

writeline(output\_manager,L);

for I in 0 to 32 loop

for J in 0 to 15 loop

Z\_thing <= Std\_logic\_vector(u);

Y\_thing <= Std\_logic\_vector(v);

write(L, Z\_thing, left, 10);

write(L, Y\_thing, left, 10);

WRITE(L, divider, left, 5);

write(L, result2, left, 10);

writeline(output\_manager,L);

v := v + "1";

wait for 10 ns;

end loop;

v:=(others => '0');

u := u + "1";

end loop;

end process;

end TESTBENCH;