CS 232 Lab 4 Q1

Arya Amin 200050014

8th February 2022

Adder(sel = 000)(`Adder.vhd`)

I have designed the carry-look-ahead adder using the 1-bit Ripple carry adder designed in the previous labs. The logic using in the adder is as follows:

p0 = a0 xor b0,
p1 = a1 xor b1,
p2 = a2 xor b2,
p3 = a3 xor b3,

c0 = cin = 0,
c1 = c0.p0 + a0.b0,
c2 = c1.p1 + a1.b1,
c3 = c2.p2 + a2.b2,

c0 is cin whereas c1, c2 and c3 are intermediate carry bits.

result(4) = c3.p3 + a3.b3

result(0) = FullAdderSum(a0, b0, cin=c0),
result(1) = FullAdderSum(a1, b1, cin=c1),
result(2) = FullAdderSum(a2, b2, cin=c2),
result(3) = FullAdderSum(a3, b3, cin=c3)

Here, result(4) is the 5th bit that is the carry and p0, p1, p2, p3, c0, c1, c2 and c3 are 1-bit signals.

Total,
XOR = 4,
FullAdder = 4

For implementing the XOR Gate I have designed xorgate entity in the project where I have used basic OR, NOT and AND Gates to design the XOR Gate.

Subtractor(sel = 001)(`Subtractor.vhd`)

I have designed the carry-look-ahead subtractor using the 1-bit Ripple carry adder designed in the previous labs. The logic using in the adder is as follows:

d = cin XOR b i.e.,

d0 = c0 XOR b0,
d1 = c0 XOR b1,
d2 = c0 XOR b2,
d3 = c0 XOR b3

Replace b with d inside the logical expression of the above adder,

k0 = a0 xor d0,
k1 = a1 xor d1,
k2 = a2 xor d2,
k3 = a3 xor d3,

c0 = cin = 0,
c1 = c0.k0 + a0.d0,
c2 = c1.k1 + a1.d1,
c3 = c2.k2 + a2.d2,

c0 is cin whereas c1, c2 and c3 are intermediate carry bits.

result(4) = c3.k3 + a3.d3

result(0) = FullAdderSum(a0, d0, cin=c0),
result(1) = FullAdderSum(a1, d1, cin=c1),
result(2) = FullAdderSum(a2, d2, cin=c2),
result(3) = FullAdderSum(a3, d3, cin=c3)

Here, result(4) is the 5th bit that is the carry and k0, k1, k2, k3, c0, c1, c2 and c3 are 1-bit signals, d is 4-bit signal

Total,
XOR = 8,
FullAdder = 4

For implementing the XOR Gate I have designed xorgate entity in the project where I have used basic OR, NOT and AND Gates to design the XOR Gate.

Multiplier(sel = 010)(`Multiplier.vhd`)

I have designed the array multiplier using OnebitFullAdder, OnebitHalfAdder designed in the previous labs, with the help of basic gates.

andij = ai and bj,

hasi = HalfAdderInstancei's Sum,
haci = HalfAdderInstancei's Carry Out,
fasi = FullAdderSum of ith instance,
faci = FulladderCarry of ith instance,

result0 = and00,
result1 = has1(and01, and11),
result2 = has3(and02, fas1(and,11, and20, hac1(and01, and11))),
result3 = has4,
result4 = fas6,
result5 = fas7,
result6 = fas8,
result7 = fac8

I am not specifying the arguments in result 3 to 7, its done inside the code.

Total,
Ands = 16,
Half Adder = 4,
Full Adder = 8

For implementing the And Gate I have designed andgate entity in the project using basic gates

Comparator(sel = 011)(`Comparator.vhd`)

I have designed the comaprator using Xnor gate designed in the main project using Basic Gates and basic gates. It is designed such that when:

a < b = result = xxxxx001
a > b = result = xxxxx100
a = b = result = xxxxx010

Means, when a is lesser then result0 is 1, when b is lesser result2 is 1, else result1 is 1.

s0 = a0 xnor b0,
s1 = a1 xnor b1,
s2 = a2 xnor b2,
s3 = a3 xnor b3

Here s is a 4-bit vector signal.

result1 = s0 and s1 and s2 and s3,
result2 = (a3 and b3) or (s3 and a2 and b2) or (s3 and s2 and a1 and b1) or (s3 and s2 and s1 and a0 and b0),
result0 = replace a with b in resutl2 formula.

Total,
Xnor = 4

Bitwise NAND(sel = 100)(`Bitwise_NAND.vhd`)

I have designed this using AND and Not entities in the main project.

and00 = a0 and b0,
and11 = a1 and b1,
and22 = a2 and b2,
and33 = a3 and b3

result0 = and00,
result1 = and11,
result2 = and22,
result3 = and33

AND Gates = 4,
NOT Gates = 4

Bitwise NOR(sel = 101)(`Bitwise_NOR.vhd`)

I have designed this using OR and NOT entities in the main project.

or00 = a0 OR b0,
or11 = a1 OR b1,
or22 = a2 OR b2,
or33 = a3 OR b3

result0 = or00,
result1 = or11,
result2 = or22,
result3 = or33

OR Gates = 4,
NOT Gates = 4

Bitwise XOR(sel = 110)(`Bitwise_XOR.vhd`)

I have designed this using XOR entities in the main project.

result0 = a0 xor b0,
result1 = a1 xor b1,
result2 = a2 xor b2,
result3 = a3 xor b3

Bitwise XNOR(sel = 111)(`Bitwise_XNOR.vhd`)

I have designed this using XNOR entities in the main project.

result0 = a0 xnor b0,
result1 = a1 xnor b1,
result2 = a2 xnor b2,
result3 = a3 xnor b3

MUX(`mux.vhd`)

ra when "000",
rb when "001",
rc when "010",
rd when "011",
re when "100",
rf when "101",
rg when "110",
rh when "111",
ra when others;

Here, ra, rb, rc, rd, re, rf, rg, rh are 7-bit vectors which gets as output based on the value of the 3-bit Selector.

FoutbitALU(`FourbitALU.vhd`)

Here I have the above designed MUX and,
I have stored result as,

Adder = ra,
Subtractor = rb,
Multiplier = rc,
Comparator = rd,
BitwiseNAND = re,
BitwiseNOR = rf,
BitwiseXOR = rg,
BitwiseXNOR = rh

The mux gives the correct result based on the selector.