CSE331 - Computer Organization

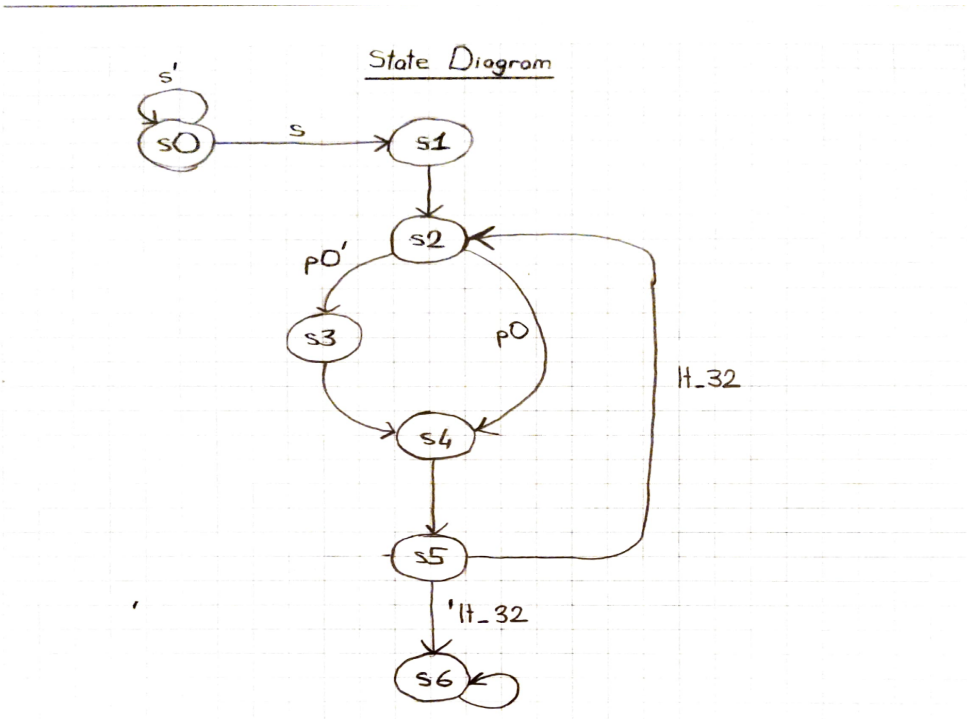
Homework 3 - Report

Ahmet Tuğkan Ayhan

1901042692

PART 1

State Diagram



inputs : s , p0, lt_32

7 state , 3 select bit

s0	000
s1	001
s2	010
s3	011
s4	101
s6	110

Truth Table & Boolean Expression

Truth Table

s_2	s_1	s_0	s	p0	lt_32	n_2	n_1	n_0
0	0	0	0	X	X	0	0	0
0	0	0	1	X	X	0	0	1
0	0	1	X	X	X	0	1	0
0	1	0	X	0	X	0	1	1
0	1	0	X	1	X	1	0	0
0	1	1	X	X	X	1	0	0
1	0	0	X	X	X	1	0	1
1	0	1	X	X	0	1	1	0
1	0	1	X	X	1	0	1	0
1	1	0	X	X	X	~	~	~
1	1	1	X	X	X	~	~	~

Boolean Expressions

$$n_2 = s_2's_1s_0'p0 + s_2's_1s_0 + s_2s_1's_0' + s_2s_1's_0\,lt\_32'$$

$$n_1 = s_2's_1's_0 + s_2's_1s_0'p0' + s_2s_1's_0(lt\_32 + lt\_32')$$

$$n_0 = s_2's_1's_0's + s_2's_1s_0'p0' + s_2s_1's_0'$$

PART 2

TESTBENCH RESULTS OF ALL OF MY MODULES

Module: half_adder_tb

```
# Top level modules:
#       half_adder_tb
ModelSim> vsim work.half_adder_tb
# vsim work.half_adder_tb
# Loading work.half_adder_tb
# Loading work.half_adder
VSIM 4> step -current
# Time:  0, value1: 0, value2: 0, sum: 0, carry_out: 0
# Time: 20, value1: 0, value2: 1, sum: 1, carry_out: 0
# Time: 40, value1: 1, value2: 0, sum: 1, carry_out: 0
# Time: 60, value1: 1, value2: 1, sum: 0, carry_out: 1
```

Module: full_adder_tb

```
# Loading work.full_adder
# Loading work.half_adder
VSIM 4> step -current
# Time:   0, value1: 0, value2: 0, carry_in: 0, sum: 0, carry_out: 0
# Time:  20, value1: 0, value2: 0, carry_in: 1, sum: 1, carry_out: 0
# Time:  40, value1: 0, value2: 1, carry_in: 0, sum: 1, carry_out: 0
# Time:  60, value1: 0, value2: 1, carry_in: 1, sum: 0, carry_out: 1
# Time:  80, value1: 1, value2: 0, carry_in: 0, sum: 1, carry_out: 0
# Time: 100, value1: 1, value2: 0, carry_in: 1, sum: 0, carry_out: 1
# Time: 120, value1: 1, value2: 1, carry_in: 0, sum: 0, carry_out: 1
# Time: 140, value1: 1, value2: 1, carry_in: 1, sum: 1, carry_out: 1
```

Module: adder_4bit_tb

!!! I checked overflow condition in 4 bit adder, because of that numbers here are signed values and as you can see at Time: 60, 6 + 7 + (carry_in) = 14 will give overflow because 14 (1110) is a negative number in two's complement system and we can't find a negative value from the sum of two positive numbers.

!!! In pdf there was no rule about using signed or unsigned values except for mult32 so I used signed values in 4 bit adders in order to detect overflow which later I used in adder_32bit, sub_32bit and slt_32bit

```
# Loading work.adder_4bit_tb
# Loading work.adder_4bit
# Loading work.full_adder
# Loading work.half_adder
VSIM 6> step -current
# Time:   0, value1: 0000( 0), value2: 0001( 1), carry_in: 0, sum: 0001( 1), carry_out: 0, overflow: 0
# Time:  20, value1: 0010( 2), value2: 0011( 3), carry_in: 1, sum: 0110( 6), carry_out: 0, overflow: 0
# Time:  40, value1: 0100( 4), value2: 0101( 5), carry_in: 0, sum: 1001(-7), carry_out: 0, overflow: 1
# Time:  60, value1: 0110( 6), value2: 0111( 7), carry_in: 1, sum: 1110(-2), carry_out: 0, overflow: 1
# Time:  80, value1: 1000(-8), value2: 1001(-7), carry_in: 0, sum: 0001( 1), carry_out: 1, overflow: 1
# Time: 100, value1: 1010(-6), value2: 1011(-5), carry_in: 1, sum: 0110( 6), carry_out: 1, overflow: 1
# Time: 120, value1: 1100(-4), value2: 1101(-3), carry_in: 0, sum: 1001(-7), carry_out: 1, overflow: 0
# Time: 140, value1: 1110(-2), value2: 1111(-1), carry_in: 1, sum: 1110(-2), carry_out: 1, overflow: 0
```

Module: adder_32bit_tb

!!! In first parentheses I showed hexadecimal representation of the 32 bit binary values

!!! In second parentheses I showed decimal representation of the 32 bit binary values

!!! In this module I also used signed values for the testbench

```
# Time  : 0
# value1: 00000000000000011000011010100000 (000186a0) (     100000)
# value2: 00000000000000110000110101000000 (00030d40) (     200000)
# sum   : 00000000000000100100100111100001 (000493e1) (     300001)
# carry_in : 1 carry_out: 0  overflow: 0
#
# Time  : 20
# value1: 10000000000000000000000000000000 (80000000) (-2147483648)
# value2: 01111111111111111111111111111111 (7fffffff) ( 2147483647)
# sum   : 00000000000000000000000000000000 (00000000) (          0)
# carry_in : 1 carry_out: 1  overflow: 0
#
# Time  : 40
# value1: 00010010000110100010101100111000 (12345678) (  305419896)
# value2: 10000111011000101010000011001000 (87654321) (-2023406815)
# sum   : 10011001100110011001100110011001 (99999999) (-1717986919)
# carry_in : 0 carry_out: 0  overflow: 0
#
# Time  : 60
# value1: 01111111111111111111111111111111 (7fffffff) ( 2147483647)
# value2: 00000000000000000000000000000001 (00000001) (          1)
# sum   : 10000000000000000000000000000000 (80000000) (-2147483648)
# carry_in : 0 carry_out: 0  overflow: 1
```

Module: sub_32bit_tb

!!! In second example, the result give overflow because the equation is A - B which also equals A + (-B) and A is a negative number as we used signed numbers, B is a positive value but -B is negative. When we tried to sum up those two negative numbers we found a positive number. Thus, this is a overflow.

```
#
# Time  : 0
# value1: 00000000000000110110010000001110 (0003640e) (     222222)
# value2: 00000000000000110000110101000000 (00030d40) (     200000)
# result: 00000000000000000101011011001110 (000056ce) (      22222) - overflow: 0
#
# Time  : 20
# value1: 10000000000000000000000000000000 (80000000) (-2147483648)
# value2: 01111111111111111111111111111111 (7fffffff) ( 2147483647)
# result: 00000000000000000000000000000001 (00000001) (          1) - overflow: 1
#
# Time  : 40
# value1: 00001011111010111100000111111111 (0bebc1ff) (  199999999)
# value2: 00000000000000000000000000000001 (00000001) (          1)
# result: 00001011111010111100000111111110 (0bebc1fe) (  199999998) - overflow: 0
#
# Time  : 60
# value1: 00000101010111010100101010000000 (055d4a80) (   90000000)
# value2: 00000000101010011000101011000111 (00a98ac7) (   11111111)
# result: 00000100101100111011111110111001 (04b3bfb9) (   78888889) - overflow: 0
#
# Time  : 80
# value1: 00000000000000000000000000000001 (00000001) (          1)
# value2: 00000000000000000000000000000010 (00000002) (          2)
# result: 11111111111111111111111111111111 (ffffffff) (         -1) - overflow: 0
#
```

Module: and_32bit_tb

```
VSIM 13> vsim work.and_32bit_tb
# vsim work.and_32bit_tb
# Loading work.and_32bit_tb
# Loading work.and_32bit
VSIM 14> step -current
#
# Time    : 0
# value1 : 11110000000000001111000000000000 (f000f000) (4026593280)
# value2 : 00001111000000000001000000000000 (0f001000) ( 251662336)
# result : 00000000000000000001000000000000 (00001000) (      4096)
#
# Time    : 20
# value1 : 11111111111111110000000000000000 (ffff0000) (4294901760)
# value2 : 00000000000000001111111111111111 (0000ffff) (     65535)
# result : 00000000000000000000000000000000 (00000000) (         0)
#
# Time    : 40
# value1 : 00001111000011110000111100001111 (0f0f0f0f) ( 252645135)
# value2 : 00001010000010100000101000001010 (0a0a0a0a) ( 168430090)
# result : 00001010000010100000101000001010 (0a0a0a0a) ( 168430090)
#
# Time    : 60
# value1 : 10101010101010101010101010101010 (aaaaaaaa) (2863311530)
# value2 : 11111111111111111111111111111111 (ffffffff) (4294967295)
# result : 10101010101010101010101010101010 (aaaaaaaa) (2863311530)
```

Module: or_32bit_tb

```
ModelSim> vsim work.or_32bit_tb
# vsim work.or_32bit_tb
# Loading work.or_32bit_tb
# Loading work.or_32bit
VSIM 4> step -current
#
# Time    : 0
# value1 : 11110000000000001111000000000000 (f000f000) (4026593280)
# value2 : 00001111000000000001000000000000 (0f001000) ( 251662336)
# result : 11111111000000001111000000000000 (ff00f000) (4278251520)
#
# Time    : 20
# value1 : 11111111111111110000000000000000 (ffff0000) (4294901760)
# value2 : 00000000000000001111111111111111 (0000ffff) (     65535)
# result : 11111111111111111111111111111111 (ffffffff) (4294967295)
#
# Time    : 40
# value1 : 00001111000011110000111100001111 (0f0f0f0f) ( 252645135)
# value2 : 00001010000010100000101000001010 (0a0a0a0a) ( 168430090)
# result : 00001111000011110000111100001111 (0f0f0f0f) ( 252645135)
#
# Time    : 60
# value1 : 10101010101010101010101010101010 (aaaaaaaa) (2863311530)
# value2 : 11111111111111111111111111111111 (ffffffff) (4294967295)
# result : 11111111111111111111111111111111 (ffffffff) (4294967295)
```

Module: xor_32bit_tb

```
ModelSim> vsim work.xor_32bit_tb
# vsim work.xor_32bit_tb
# Loading work.xor_32bit_tb
# Loading work.xor_32bit
VSIM 4> step -current
#
# Time    : 0
# value1 : 11110000000000001111000000000000 (f000f000) (4026593280)
# value2 : 00001111000000000001000000000000 (0f001000) ( 251662336)
# result : 11111111000000001110000000000000 (ff00e000) (4278247424)
#
# Time    : 20
# value1 : 11111111111111110000000000000000 (ffff0000) (4294901760)
# value2 : 00000000000000001111111111111111 (0000ffff) (     65535)
# result : 11111111111111111111111111111111 (ffffffff) (4294967295)
#
# Time    : 40
# value1 : 00001111000011110000111100001111 (0f0f0f0f) ( 252645135)
# value2 : 00001010000010100000101000001010 (0a0a0a0a) ( 168430090)
# result : 00000101000001010000010100000101 (05050505) (  84215045)
#
# Time    : 60
# value1 : 10101010101010101010101010101010 (aaaaaaaa) (2863311530)
# value2 : 11111111111111111111111111111111 (ffffffff) (4294967295)
# result : 01010101010101010101010101010101 (55555555) (1431655765)
```

Module: nor_32bit_tb

```
ModelSim> vsim work.nor_32bit_tb
# vsim work.nor_32bit_tb
# Loading work.nor_32bit_tb
# Loading work.nor_32bit
VSIM 4> step -current
#
# Time    : 0
# value1 : 11110000000000001111000000000000 (f000f000) (4026593280)
# value2 : 00001111000000000001000000000000 (0f001000) ( 251662336)
# result : 00000000111111111000011111111111 (00ff0fff) (  16715775)
#
# Time    : 20
# value1 : 11111111111111110000000000000000 (ffff0000) (4294901760)
# value2 : 00000000000000001111111111111111 (0000ffff) (     65535)
# result : 00000000000000000000000000000000 (00000000) (         0)
#
# Time    : 40
# value1 : 00001111000011110000111100001111 (0f0f0f0f) ( 252645135)
# value2 : 00001010000010100000101000001010 (0a0a0a0a) ( 168430090)
# result : 11110000111100001111000011110000 (f0f0f0f0) (4042322160)
#
# Time    : 60
# value1 : 10101010101010101010101010101010 (aaaaaaaa) (2863311530)
# value2 : 11111111111111111111111111111111 (ffffffff) (4294967295)
# result : 00000000000000000000000000000000 (00000000) (         0)
```

Module: not_32bit_tb

```
# Loading work.not_32bit_tb
# Loading work.not_32bit
VSIM 4> step -current
#
# Time    : 0
# value   : 11110000000000001111000000000000 (f000f000) (4026593280)
# result  : 00001111111111111000111111111111 (0fff0fff) ( 268374015)
#
# Time    : 20
# value   : 11111111111111110000000000000000 (ffff0000) (4294901760)
# result  : 00000000000000001111111111111111 (0000ffff) (     65535)
#
# Time    : 40
# value   : 00001111000011110000111100001111 (0f0f0f0f) ( 252645135)
# result  : 11110000111100001111000011110000 (f0f0f0f0) (4042322160)
#
# Time    : 60
# value   : 10101010101010101010101010101010 (aaaaaaaa) (2863311530)
# result  : 01010101010101010101010101010101 (55555555) (1431655765)
```

Module: slt_32bit_tb

```
# Loading work.slt_32bit_tb
# Loading work.slt_32bit
# Loading work.sub_32bit
# Loading work.not_32bit
# Loading work.adder_32bit
# Loading work.adder_4bit
# Loading work.full_adder
# Loading work.half_adder
# Loading work.mux2x1_32bit
# Loading work.mux2x1
VSIM 4> step -current
#
# Time    : 0
# value1 : 00000000000011111111111111110000 (000ffff0) (   1048560)
# value2 : 00000000000011111111111111111111 (000fffff) (   1048575)
# slt    : 11111111111111111111111111111111
#
# Time    : 20
# value1 : 01000000000000000000000000000000 (40000000) (1073741824)
# value2 : 00111111111111111111111111111111 (3fffffff) (1073741823)
# slt    : 00000000000000000000000000000000
#
# Time    : 40
# value1 : 00000000000000000000000000000010 (00000002) (         2)
# value2 : 00000000000000000000000000000001 (00000001) (         1)
# slt    : 00000000000000000000000000000000
#
# Time    : 60
# value1 : 00000000000000000000000000000001 (00000001) (         1)
# value2 : 00000000000000000000000000000010 (00000002) (         2)
# slt    : 11111111111111111111111111111111
#
# Time    : 80
# value1 : 11111111111100000000000000000000 (fff00000) (  -1048576)
# value2 : 00000000000000000000000000000000 (00000000) (         0)
# slt    : 11111111111111111111111111111111
```

!!! slt works as it should be but I just Want to mention that, when the most significant bit is 1 value becomes negative, thus even 0 is bigger than that value (Time:80) and because of that slt is equal to 1

Module: mux2x1_tb

```
# vsim work.mux2x1_tb
# Loading work.mux2x1_tb
# Loading work.mux2x1
VSIM 7> step -current
#
# Time  : 0
# input1: 1  -  input2: 0
# select: 0
# result: 1
#
# Time  : 20
# input1: 1  -  input2: 0
# select: 1
# result: 0
```

Module: mux8x1_tb

```
# vsim work.mux8x1_tb
# Loading work.mux8x1_tb
# Loading work.mux8x1
# Loading work.mux2x1
VSIM 15> step -current
#
# Time  : 0
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 000
# result: 1
#
# Time  : 20
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 001
# result: 1
#
# Time  : 40
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 010
# result: 1
#
# Time  : 60
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 011
# result: 0
#
# Time  : 80
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 100
# result: 0
#
# Time  : 100
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 101
# result: 1
#
# Time  : 120
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 110
# result: 0
#
# Time  : 140
# input0: 1 - input1: 1 - input2: 1 - input3: 0 - input4: 0 - input5: 1 - input6: 0 - input7: 1
# select: 111
# result: 1
```

Module: mux2x1_32bit_tb

```
# Loading work.mux2x1_32bit_tb
# Loading work.mux2x1_32bit
# Loading work.mux2x1
VSIM 5> step -current
# Inputs(32 bit -> hexadecimal)
# ------
# 0) aaaaaaaa
# 1) 11111111
#
#
# Time  : 5
# select: 0
# result: aaaaaaaa
#
# Time  : 20
# select: 1
# result: 11111111
```

Module: mux8x1_32bit_tb

```
# Inputs(32 bit -> hexadecimal)
# ------
# 0) ffffffff
# 1) eeeeeeee
# 2) dddddddd
# 3) cccccccc
# 4) bbbbbbbb
# 5) aaaaaaaa
# 6) 99999999
# 7) 88888888
#
#
# Time  : 5
# select: 000
# result: ffffffff
#
# Time  : 20
# select: 001
# result: eeeeeeee
#
# Time  : 40
# select: 010
# result: dddddddd
#
# Time  : 60
# select: 011
# result: cccccccc
#
# Time  : 80
# select: 100
# result: bbbbbbbb
#
# Time  : 100
# select: 101
# result: aaaaaaaa
#
# Time  : 120
# select: 110
# result: 99999999
#
# Time  : 140
# select: 111
# result: 88888888
```

ALU TESTBENCH RESULTS

First Test

```
VSIM 32> step -current
# Time  : 0 - Select: 000
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 10000110100001101000011010000110 (86868686)
#
# Time  : 20 - Select: 001
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 01111000011110000111100001111000 (78787878)
#
# Time  : 40 - Select: 010
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 01101000011010000110100001101000 (68686868)
#
# Time  : 60 - Select: 011
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx (xxxxxxxx)
#
# Time  : 80 - Select: 100
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 00000000000000000000000000000000 (00000000)
#
# Time  : 100 - Select: 101
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 10000000100000001000000010000000 (80808080)
#
# Time  : 120 - Select: 110
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 00000111000001110000011100000111 (07070707)
#
# Time  : 140 - Select: 111
# value1: 01110111011101110111011101110111 (77777777)
# value2: 00001111000011110000111100001111 (0f0f0f0f)
# result: 01111111011111110111111101111111 (7f7f7f7f)
#
```

hex

ADD

XOR

SUB

MULT

SLT

NOR

AND

OR

Second Test

```
#
# Time  : 160 - Select: 000
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 11011101110111011101110111011101 (dddddddd)
#
# Time  : 180 - Select: 001
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 10011001100110011001100110011001 (99999999)
#
# Time  : 200 - Select: 010
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 01110111011101110111011101110111 (77777777)
#
# Time  : 220 - Select: 011
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx (xxxxxxxx)
#
# Time  : 240 - Select: 100
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 11111111111111111111111111111111 (ffffffff)
#
# Time  : 260 - Select: 101
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 01000100010001000100010001000100 (44444444)
#
# Time  : 280 - Select: 110
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 00100010001000100010001000100010 (22222222)
#
# Time  : 300 - Select: 111
# value1: 10101010101010101010101010101010 (aaaaaaaa)
# value2: 00110011001100110011001100110011 (33333333)
# result: 10111011101110111011101110111011 (bbbbbbbb)
#
```

ADD

XOR

SUB

MULT

SLT

NOR

AND

OR

My Modules and Their Parameters

- half_adder (value1, value2, sum, carry_out)
- full_adder (value1, value2, sum, carry_out, carry_in)
- adder_4bit (value1, value2, sum, carry_out, carry_in, overflow)
- adder_32bit (value1, value2, sum, carry_out, carry_in, overflow)
- and_32bit (value1, value2, result)
- or_32bit (value1, value2, result)
- xor_32bit (value1, value2, result)
- nor_32bit (value1, value2, result)
- not_32bit (value, result)
- sub_32bit (value1, value2, result, overflow)
- slt_32bit (value1, value2, slt)
- mux2x1 (input1, input2, select, result)
- mux2x1_32bit (input1, input2, select, result)
- mux8x1 (input1, input2, ..., input8, select, result)
- mux8x1_32bit (input1, input2, ..., input8, select, result)
- alu_32bit (value1, value2, select, result)

!! Every module has their own testbench module so I didn't include them here.

!! There are total of 32 modules (16 module + 16 testbench module)