# CSE331 – COMPUTER ORGANIZATION

HOMEWORK 3 REPORT

## PART 1 – MULT32.V MODULE

First I have made an FSM model:

*OUTPUTS*

AP: add multiplicand to the left half of the product

SR: Shift right the product register by 1 bit

WRITE: Writes the result

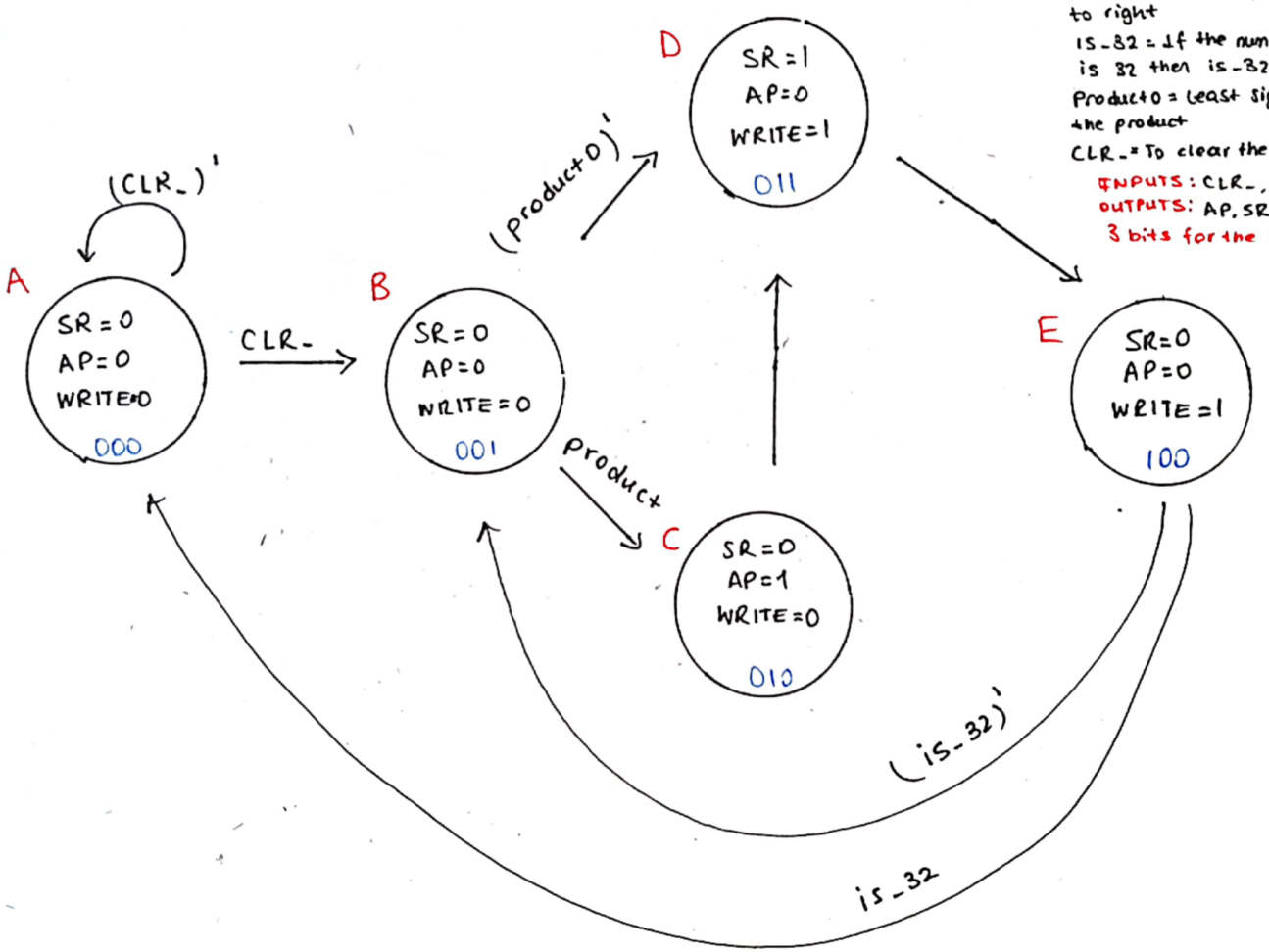
*INPUTS*

Product0: The least significant bit of the product.

İs_32: If the number of repetitions is 32 it inputs 1

CLR_: For resetting the process.

# FINAL STATE MACHINE




AP = Add multiplicand to the left half of the product.

WRITE = writes the result.

SR = Shift the product register by 1 bit to right

IS_32 = If the number of repetitions is 32 then is_32 is 1, otherwise 0.

Product0 = Least significant bit of the product

CLR_ = To clear the FSM.

INPUTS: CLR_, is_32, Product0

OUTPUTS: AP, SR, WRITE.

3 bits for the states (5 states)

FSM - 2

present_states | inputs | next state) | outputs.

| $S_2$ | $S_1$ | $S_0$ | is_32 | product 0 | start | $n_2$ | $n_1$ | $n_0$ | AP | SR | write |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | – | – | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | – | – | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | | 0 | | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | | 1 | | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | | | | 0 | 1 | 1 | 1 | | |
| 0 | 1 | 1 | | | | 1 | 0 | 0 | | 1 | |
| 1 | 0 | 0 | 0 | | | 0 | 0 | 1 | | | 1 |
| 1 | 0 | 0 | 1 | | | 0 | 0 | 0 | | | 1 |

write the equations:

$n_2 = S_2'S_1S_0$

$n_1 = S_2'(S_1 \oplus S_0)$

$n_0 = S_2'S_1'(S_0'start + S_0 product\,0') + S_0'(S_2'S_1' + S_2S_1'(is\_32)')$

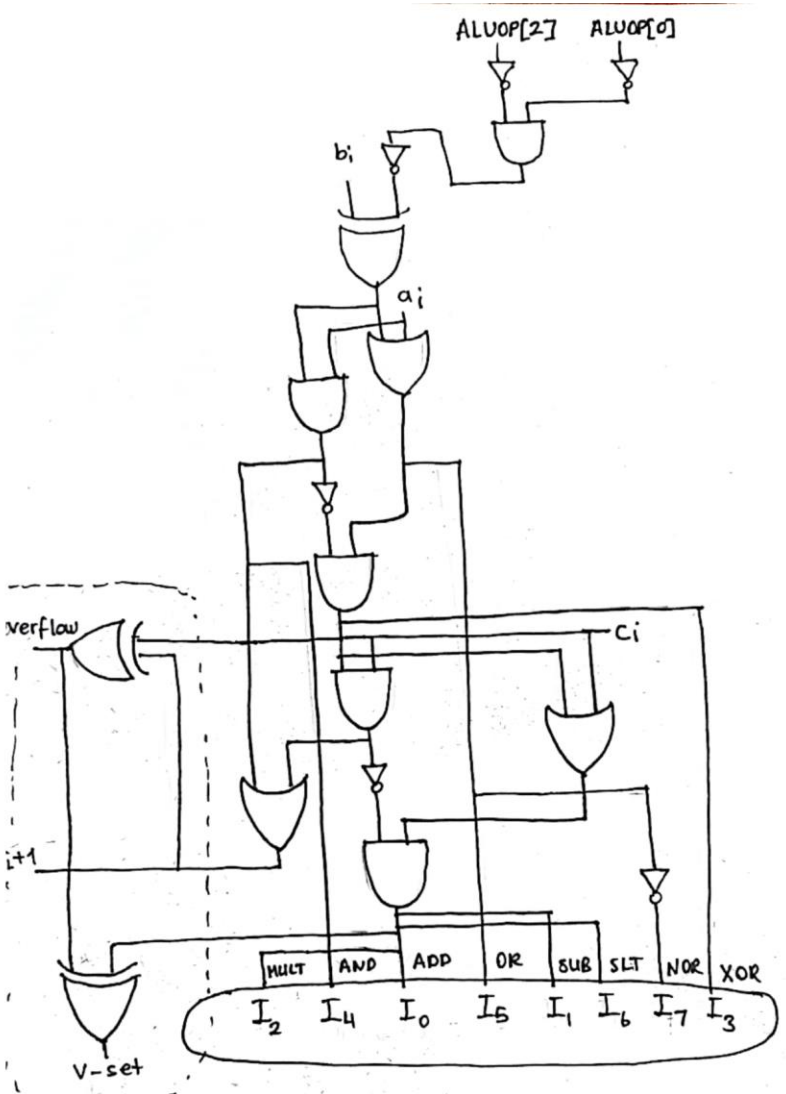
$AP = S_2'S_1S_0'$

$SR = S_2'S_1S_0$

$write = S_2S_1'S_0'(is\_32 \cdot (S-32)^0 \Rightarrow S_2S_1'S_0'$

## PART 2 – ALU DESIGN

Here is the layout of my 1-bit ALU design:




this part detects the overflow (if there is one) and v-set sets the LSB of the result of subtraction for Set less than operation.

This ALU design is only used for the most significant bit. For the rest of the bits the overflow detection part is not used because there is no need.

And in order to decide whether b or b' is going to be given as input to the adder, I have used a basic algorithm.

Addition and multiplicaion have the opcodes 000 and 010 respectively. ALUOP[2] and ALUOP[0] must be 0 for the operation to require an addition (which takes the original b value) but for subtraction and set less than operations b' is needed to calculate the correct result. Using this logic, if the NOT of ALUOP[2] and ALUOP[0] is both zero, two's complement of b must be given to the ALU. This is why I have XOR'ed the NOT of the AND of these two opcodes.

**TESTBENCHES**

```
# Loading work.mux2x1
add wave -position insertpoint  \
sim:/alu_32bit_testbench/a \
sim:/alu_32bit_testbench/alu_op \
sim:/alu_32bit_testbench/b \
sim:/alu_32bit_testbench/final_result
VSIM 5> step -current
# time =  0, a = 00000000000000000000000000001101, b = 00000000000000000000000000001100, alu_op2 = 0 ,alu_op1 = 0 ,alu_op0 = 0 ,result = 00000000000000000000000000011001 ADD = 000
# time = 20, a = 10000000000000000000000000001101, b = 10000000000000000000000000000000, alu_op2 = 0 ,alu_op1 = 0 ,alu_op0 = 0 ,result = 00000000000000000000000000001101
# time = 40, a = 00000000000000000000000000001101, b = 00000000000000000000000000001100, alu_op2 = 0 ,alu_op1 = 0 ,alu_op0 = 1 ,result = 00000000000000000000000000000001 SUB = 001
# time = 60, a = 00000000000000000000000000000000, b = 00000000000000000000000000001100, alu_op2 = 0 ,alu_op1 = 0 ,alu_op0 = 1 ,result = 11111111111111111111111111110100
# time = 80, a = 11110000000000000000000000000000, b = 11110000000000000000000000001100, alu_op2 = 0 ,alu_op1 = 0 ,alu_op0 = 1 ,result = 11111111111111111111111111110100
# time = 100, a = 00000000000000000000000000001101, b = 00000000000000000000000000000111, alu_op2 = 0 ,alu_op1 = 1 ,alu_op0 = 0 ,result = 00000000000000000000000000010100 MULT = 010
# time = 120, a = 10000000000000000000000000001101, b = 10000000000000000000000000001111, alu_op2 = 0 ,alu_op1 = 1 ,alu_op0 = 0 ,result = 00000000000000000000000000011100
# time = 140, a = 00000000000000000000000000001101, b = 00000000000000000000000000001100, alu_op2 = 0 ,alu_op1 = 1 ,alu_op0 = 1 ,result = 00000000000000000000000000000001 XOR = 011
# time = 160, a = 00000010000000000000000000001101, b = 00000010000000000000000000001100, alu_op2 = 1 ,alu_op1 = 0 ,alu_op0 = 0 ,result = 00000010000000000000000000001100 AND = 100
# time = 180, a = 00000010000000000000000000001101, b = 00100010000000000000000000001100, alu_op2 = 1 ,alu_op1 = 0 ,alu_op0 = 0 ,result = 00000010000000000000000000001100
# time = 200, a = 00000010000000000000000000001101, b = 00000010000000000000000000001100, alu_op2 = 1 ,alu_op1 = 0 ,alu_op0 = 1 ,result = 00000010000000000000000000001101 OR = 101
# time = 220, a = 00000000000000000000000000001101, b = 00000000000000000000000000001100, alu_op2 = 1 ,alu_op1 = 1 ,alu_op0 = 0 ,result = 00000000000000000000000000000000
# time = 240, a = 00000000000000000000000000000000, b = 00000000000000000000000000001100, alu_op2 = 1 ,alu_op1 = 1 ,alu_op0 = 0 ,result = 00000000000000000000000000000001 SLT = 110
# time = 260, a = 11110000000000000000000000000000, b = 11110000000000000000000000001111, alu_op2 = 1 ,alu_op1 = 1 ,alu_op0 = 0 ,result = 00000000000000000000000000000001
# time = 280, a = 00000010000000000000000000001101, b = 00000010000000000000000000001100, alu_op2 = 1 ,alu_op1 = 1 ,alu_op0 = 1 ,result = 11111101111111111111111111110010 NOR = 111

VSIM 6>
```