MC542 - 2S 2011

Laboratório 02

Cristiano J. Miranda RA 083382

Descrição

Implementação de uma ALU em vhdl, utilizando a seguinte entity:

```
-- Definicao da entidade alu (arithimethic logic unit)
--  alucontrol definitions
--  000 A AND B
--  001 A OR B
--  010 A + B
--  011 not used
--  100 A AND not B
--  101 A OR not B
--  110 A - B
--  111 SLT
entity alu is
        generic(w : natural := 32; cw: natural := 3);
        port(srca : in std_logic_vector(w-1 downto 0);
                srcb : in std_logic_vector(w-1 downto 0);
                alucontrol : in std_logic_vector(cw-1 downto 0);
                aluresult : out std_logic_vector(w-1 downto 0);
                zero : out std_logic;
                overflow : out std_logic;
                carryout : out std_logic);
end alu;
```

A ALU deve receber 2 parametros (srca e srcb) com os valores que a unidade deve operar, e um parametro (alucontrol) que informa qual operação deve ser realizada, de tal forma que quando alucontrol tiver o valor ‘000’, a ALU tera como resultado o AND entra a entrada a (srca) e a entrada b (srcb), quando alucontrol tiver o valor ‘001’ o resultado será srca or srcb, quando alucontrol tiver o valor ‘010’ o resultado sera a soma entre srca e srcb, quando tiver o valor ‘100’ será retornado a operação srca or not srcb, quando tiver o valor ‘110’ será retorando a diferença entre srca e srcb e quando em alucontrol apresentar o valor ‘111’ será retornado um bus completo com o sinal ‘1’ caso srca seja menor que srcb, caso contrario será retornado um bus completo com o sinal ‘0’.
A operação de controle representada pelo valor ‘011’ será desconsiderada, ou seja, a ALU deve se manter no mesmo estado como se nenhuma operação tivesse ocorrido.

Também será retornado um flag de carry indicando que a operação que foi realizada apresenta carry(operações de soma(‘010’) e subtração(‘110’)).
A ALU tambem terá um sinal indicando overflow de operação, ou seja, toda vez que houver carry, tambem haverá overflow (operações de soma e subtração).
Haverá um outro sinal adicional para identificar resultados iguais a zero, ou seja, toda a vez que o resultado de um operação, independente de qual seja, retornar um bus com todos os sinais igual a ‘0’, esse ‘flag’ aprestará o valor ‘1’, caso contrario ‘0’.

Implementação

Criei um arquivo alu.vhd, representado a arquitetura behavior da minha ALU, implementando dessa forma todas as operação descritas anteriormente. Para implementar a operação AND (alucontrol = ‘000’) utilizei o vhdl and, assim como as operações or, soma, subtração, a or not b, a and not b.

As operações de soma e subtração poderiam ser implementadas utilizando um componente adder e subtract que conteria a implentação logica das portas para tais operações no caso da soma um xor bit a bit entre srca e srcb, sendo o carry calculado por srca.srcb (bit a bit). E a subtração poderia utilizar o mesmo componente adder, mas com entrada not srcb.

Para computar o flag zero implementei um if comparando o result no final do meu process caso result fosse todo ‘0’, setaria zero como ‘1’.
Para computar a carry eu resolvi criar uma variavel temporaria bloqueante com tamanho do srca mais um bit, de tal forma que se srca + srcb obtivesse carry o bit mais significativo do resultado seria o carry. O overflow foi computado utilizando a mesma abordagem da carry, uma vez ocorrido carry entre srca e srcb, certamente ocorreu overflow.

Para impedir que ALU alterace os valores de saida quando fosse informado um novo valor na lista de sencibilidade para a operação ‘011’, foi implementado um if que restringe a saida, utilizo variaveis temporarias para todas as operações, mantendo dessa forma os valores anteriores.

Test Bench

Implementei tb_alu.vhd, com varios testes (24 testes no total) explorando cada uma das operações da ALU. Os testes foram implementados utilizando a seguinte tecnica: setando valores em srca, srcb, alucontrol, esperando os sinais estabilizarem e computando o resultado em aluresult, zero, overflow e carry. Todos os cenarios foram impressos no prompt para facilitar a conferencia, de tal forma que se algum teste falhar todos os testes na sequência não não executados.
Segue uma breve descrição dos cenarios de testes providos:

- **Teste 1**: Testando a operação AND, seto o valor ‘00000000000000000000000000010101’ em srca e ‘00000000000000000000000000011100’ em srcb o teste verifica se a operação resulta em ‘00000000000000000000000000010100’.
- **Teste 2**: Igual ao cenario anterior porem com os valores ‘00000000000000000000000000010101’ AND ‘00000000000000000000000000000000’, é esperado o resultado ‘00000000000000000000000000000000’.
- **Teste 3**: Testando a operação AND entre os valores ‘11111111111111111111111111111111’ e seu complemento. É esperado o resultado todo zero.
- **Teste 4**: Testando operação AND entre 2 vetores ‘11111111111111111111111111111111’, é esperado um vetor semelhando como resultado.
- **Teste 5**: Testando operação or entre os vetores ‘00000000000000000000000000010101’ e ‘00000000000000000000000000011100’, como resultado é esperado ‘00000000000000000000000000011101’.
- **Teste 6**: Semelhante ao teste anterior porem utilizando os vetores A

=‘00000000000000000000000000010101’ e B=‘00000000000000000000000000000000’, é esperado o resultado igual a o vetor A.

- **Teste 7**: Testando operação or entre o vetor ‘11111111111111111111111111111111’ e seu complemento é esperado o resultado ‘11111111111111111111111111111111’.
- **Teste 8**: Testando operação OR entre 2 vetores ‘11111111111111111111111111111111’, é esperado um resultado semalhante aos vetores de entrada.
- **Teste 9**: Teste que valida a operação de soma da ALU. Como entrada em srca o valor 320 e em srcb o valor 80, como resultado é esperado o valor 400, que em binario é representado por ‘00000000000000000000000110010000’.
- **Teste 10**: Teste de overflow da operação de soma, entre os vetores ‘11111111111111111111111111111111’ e ‘00000000000000000000000000000000’, onde como resultado é esperado um vetor ‘00000000000000000000000000000000’, carry igual a ‘1’, e overflow igual a ‘1’ indicando overflow da operação.
- **Teste 11**: Teste de overflow da operação de soma entre ‘11111111111111111111111111111111’ e ‘11110000000000000000000000000001’. Como resultado é esperado carry = 1, overflow = 1 e result = ‘11110000000000000000000000000000’
- **Teste 12**: Igual ao teste anterior porem com os valores ‘11111111111111111111111110000000’, ‘00000000000000000000011111111111’, é esperado carry, overflow e resultado igual a ‘00000000000000000000011101111111’.
- **Teste 13**: Testa a operação ‘011’, ou seja, verifica se setando algum valor diferente em srca e srcb referente ao teste anterior os valores de carry, overflow e result se mantem iguais ao teste 12, ou seja, tal operação deve ser desconsiderada.
- **Teste 14**: Semelhando ao teste anterior porem com valores diferentes para srca e srcb.
- Teste 15: Testa operação A and not B, entre os valores ‘00000000000000000000000000000001’ e ‘11111111111111111111111111111110’, como resultado é esperado ‘00000000000000000000000000000001’
- **Teste 16**: Semelhante ao teste anterior porem com os valores ‘11111111111111111111111111111110’ e ‘00000000000000000000000000000001’, como resultado espera-se ‘11111111111111111111111111111110’.
- **Teste 17**: Teste operação a and not b, com 2 vetores ‘00000000000000000000000000000001’, como resultado espera-se um vetor zero, com zero.
- **Teste 18**: Testando operação a or not b com dois vetores ‘‘00000000000000000000000000000001’, como resultado é esperado um vetor ‘11111111111111111111111111111111’.
- **Teste 19**: Semelhante ao teste anterior com os valores ‘00000000000000000000000000000000’ e ‘11111111111111111111111111111111’, como resultado é esperado ‘00000000000000000000000000000000’ e zero = ‘1’.
- **Teste 20**: Testa a subtração de um vetor ‘11111111111111111111111111111111’ dele mesmo, como resultado é esperado um vetor todo ‘0’ e o vertor zero = ‘1’.
- **Teste 21**: Testa a subtração de um vetor contendo o valor 10 e um outro contendo o valor 2, como resultado é esperado um vetor contendo o valor 8, em binario ‘00000000000000000000000000001000’.
- **Teste 22**: O inverso do exemplo anterior, sendo que é esperado como resultado ‘11111111111111111111111111111000’, e ocorrendo o carry.
- **Teste 23:** Testando a operação SLT para os valores A=2 e B=10, como A < B é esperado como resultado ‘11111111111111111111111111111111’.
- **Teste 24**: Testando a operação SLT para os valores A=10 e B=2, como A > B é esperado como resultado ‘00000000000000000000000000000000’.