Multi Operation ALU

Progetto 2

Lorenzo Olearo, 856394
Alessandro Riva, 856156

Introduzione

Il Sistema Multi-Operation ALU (MOALU) è composto da una unità centrale (logico-aritmetica, ALU) che esegue le seguenti operazioni su due numeri A e B di Nb bits:

1. Moltiplicazione per 2 di entrambi i numeri
2. Divisione per 2 di entrambi i numeri
3. Comparazione (A>B)

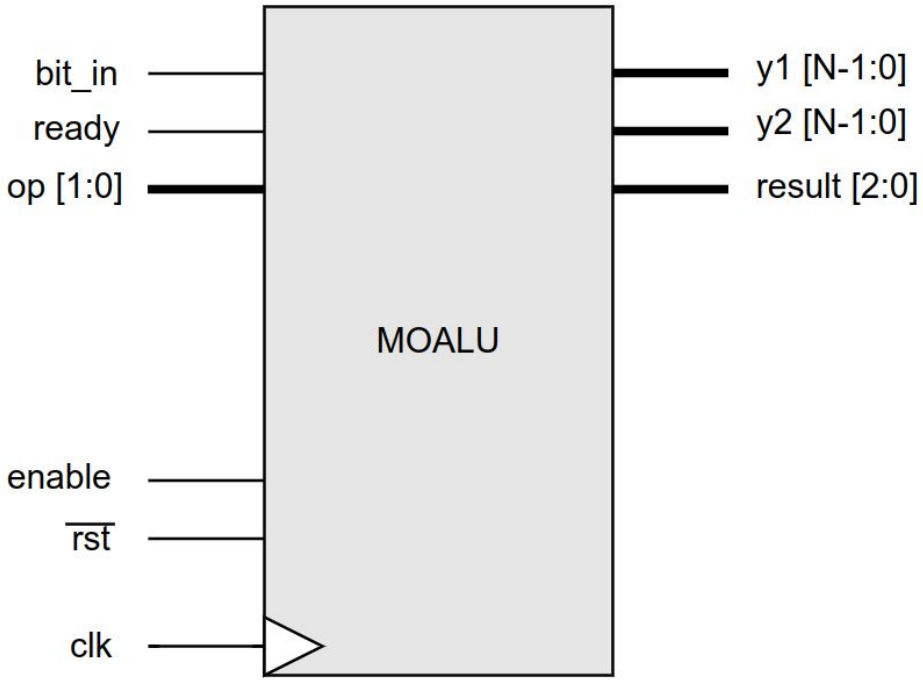
Componente MOALU

Input:

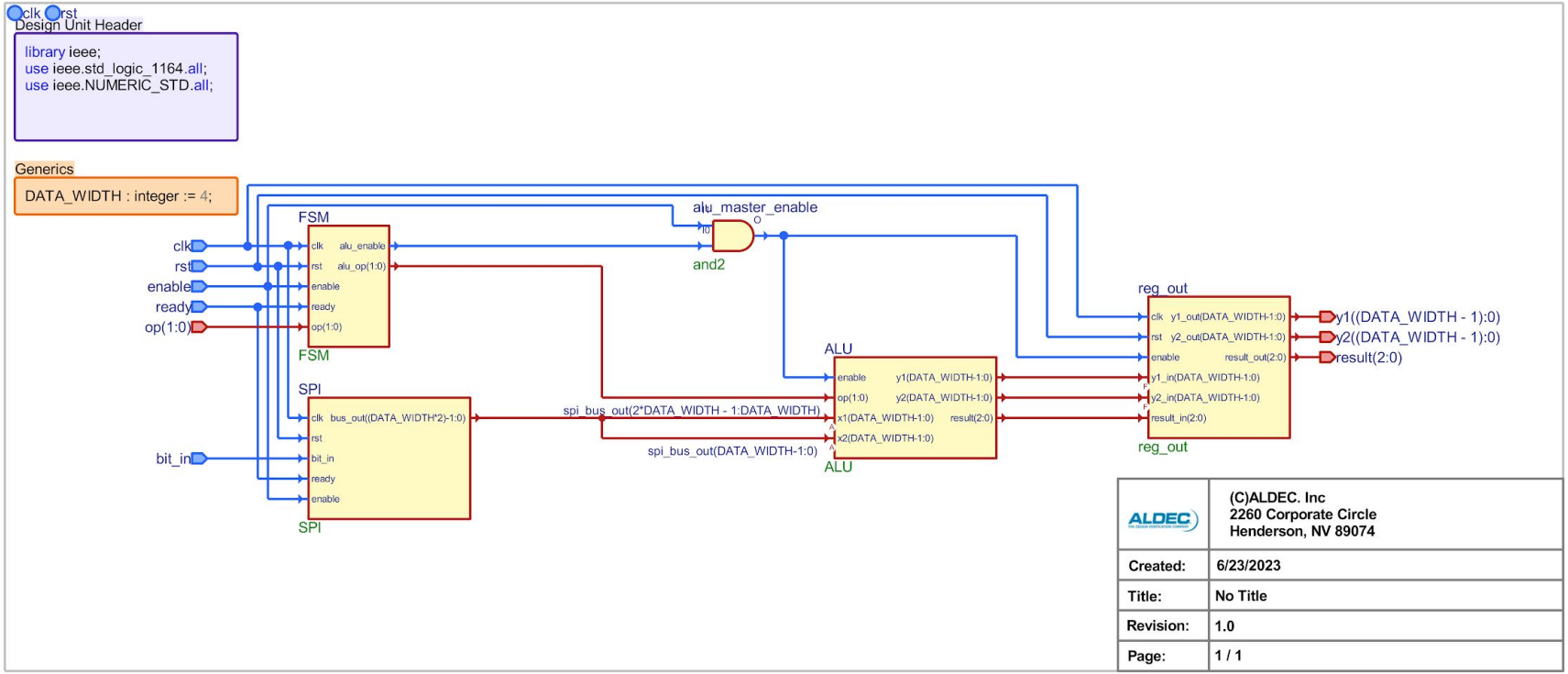
- bit_in: input seriale dei numeri
- ready: segnala il termine dell'inserimento seriale dei due numeri
- op [1:0]: seleziona l'operazione da effettuare
- enable: abilita il componente
- rst: reset asincrono attivo basso
- clk: clock

Output:

- y1: [N-1:0] risultato dell'operazione sul primo numero
- y2: [N-1:0] risultato dell'operazione sul secondo numero
- result: [2:0] risultato della comparazione o segnale di overflow sull'operazione eseguita



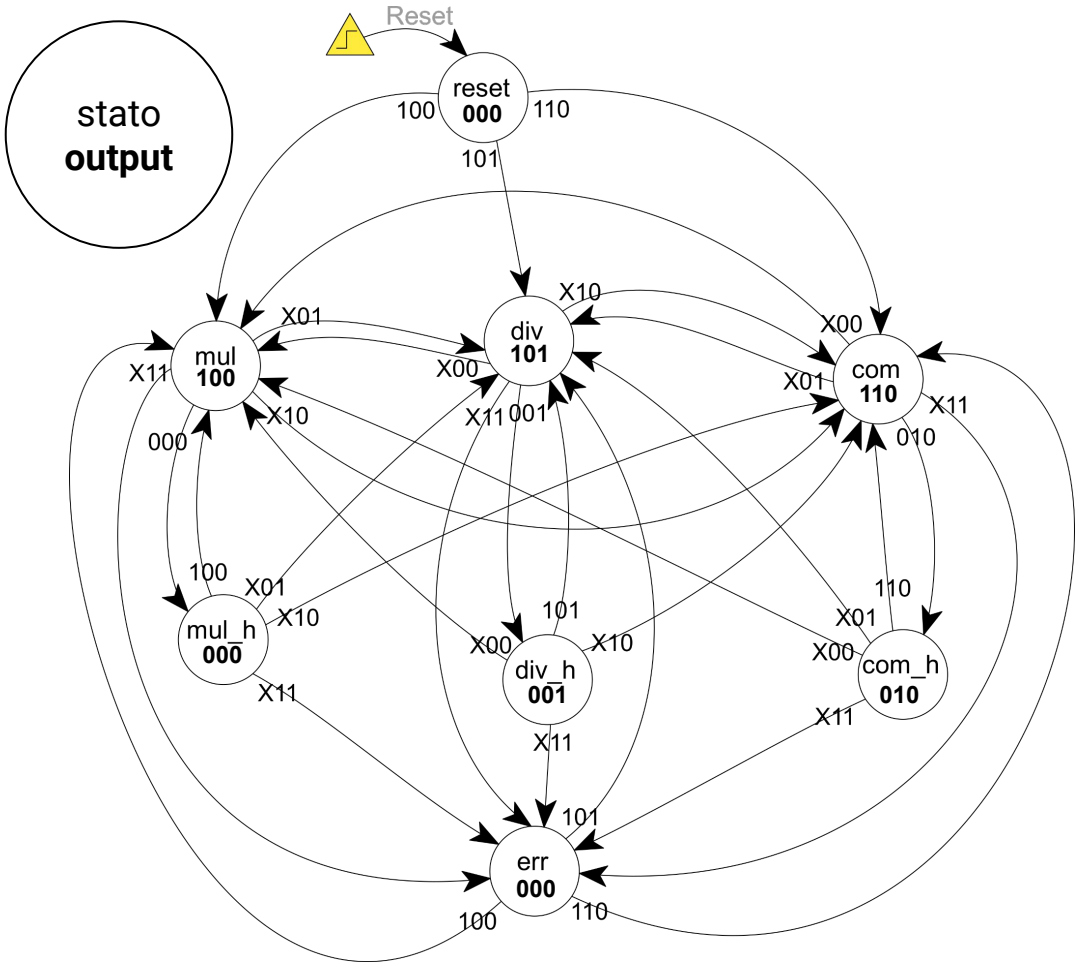
Schema a blocchi



Unità di controllo

	2	1	0
IN	ready	op	
OUT	alu_enable	alu_op	

All'occorrenza di un reset la macchina a stati torna allo stato di reset, da cui può uscire solo dopo un segnale di ready da parte dell'utente.



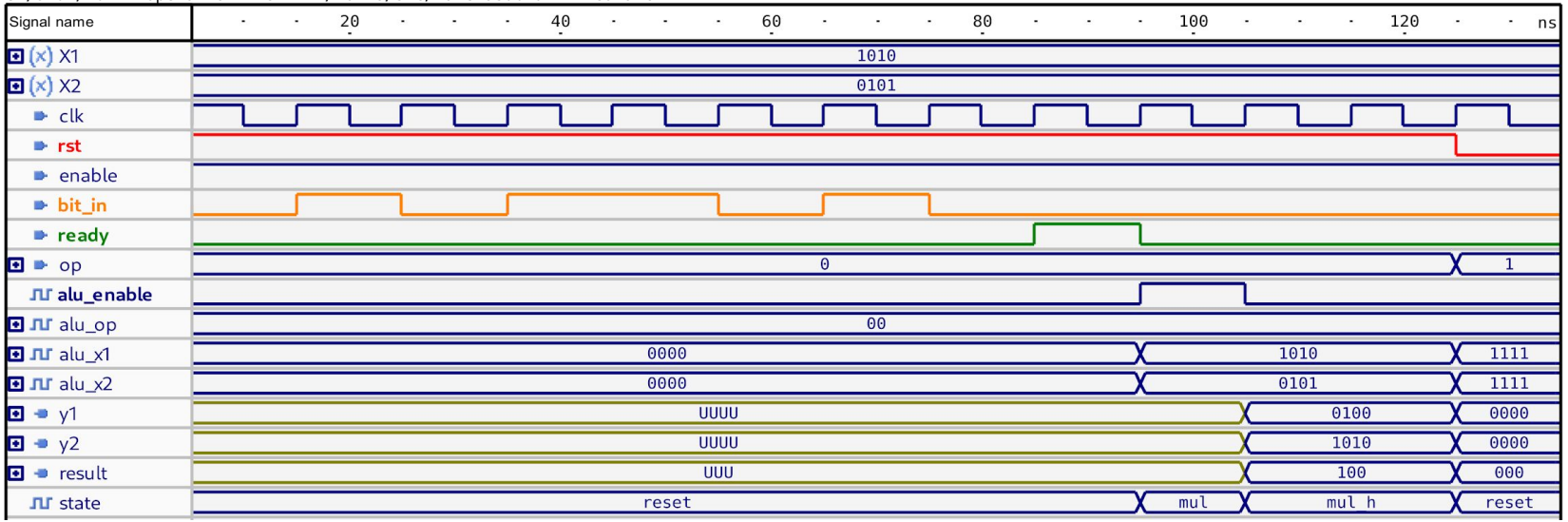
Unità di controllo - funzione di transizione

	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
reset	reset	reset	reset	reset	mul	div	com	error
mul_h	mul_h	div	com	error	mul	div	com	error
div_h	mul	div_h	com	error	mul	div	com	error
com_h	mul	div	com_h	error	mul	div	com	error
mul	mul_h	div	com	error	mul	div	com	error
div	mul	div_h	com	error	mul	div	com	error
com	mul	div	com_h	error	mul	div	com	error
error	error	error	error	error	mul	div	com	error

Testbench 1 - prodotto per 2

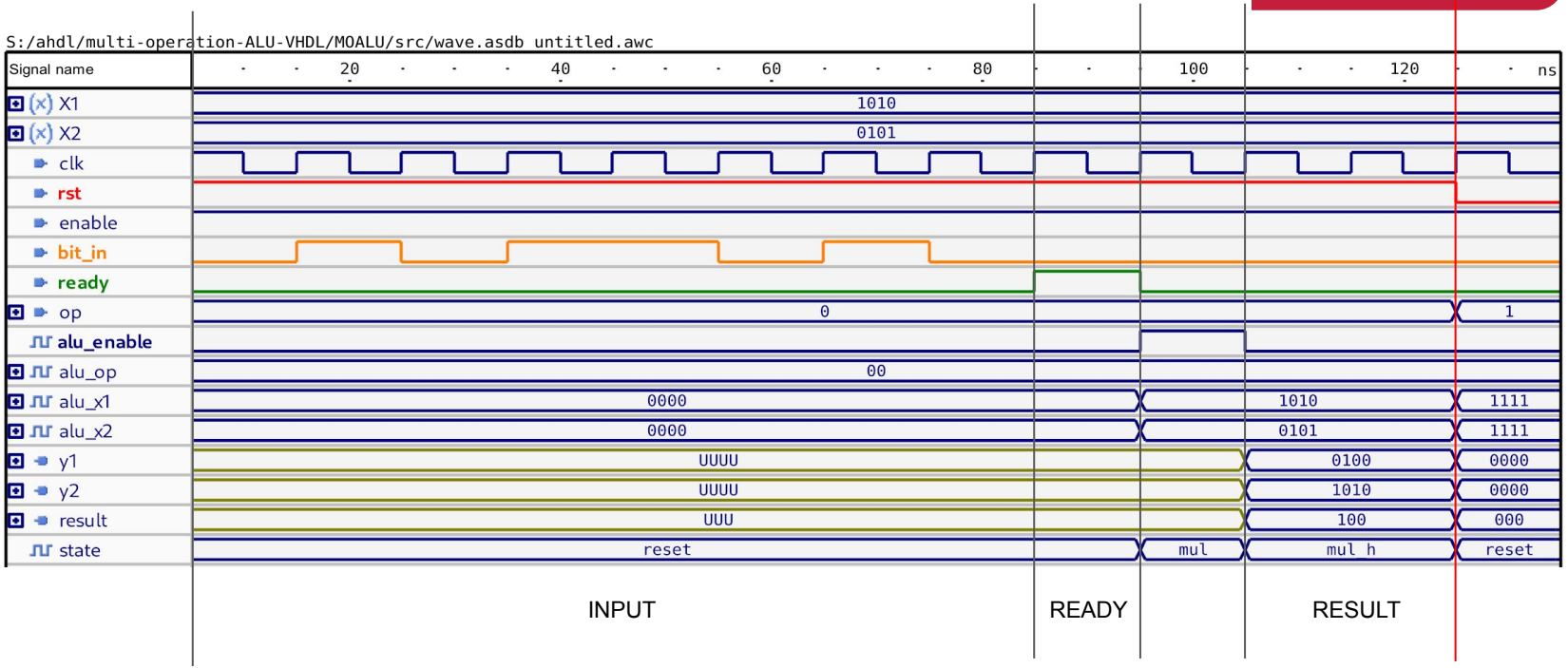
```
x1 [3:0]
x2 [3:0]
clk: 10 ns
```

S:/ahdl/multi-operation-ALU-VHDL/MOALU/src/wave.asdb untitled.awc



Testbench 1 - prodotto per 2

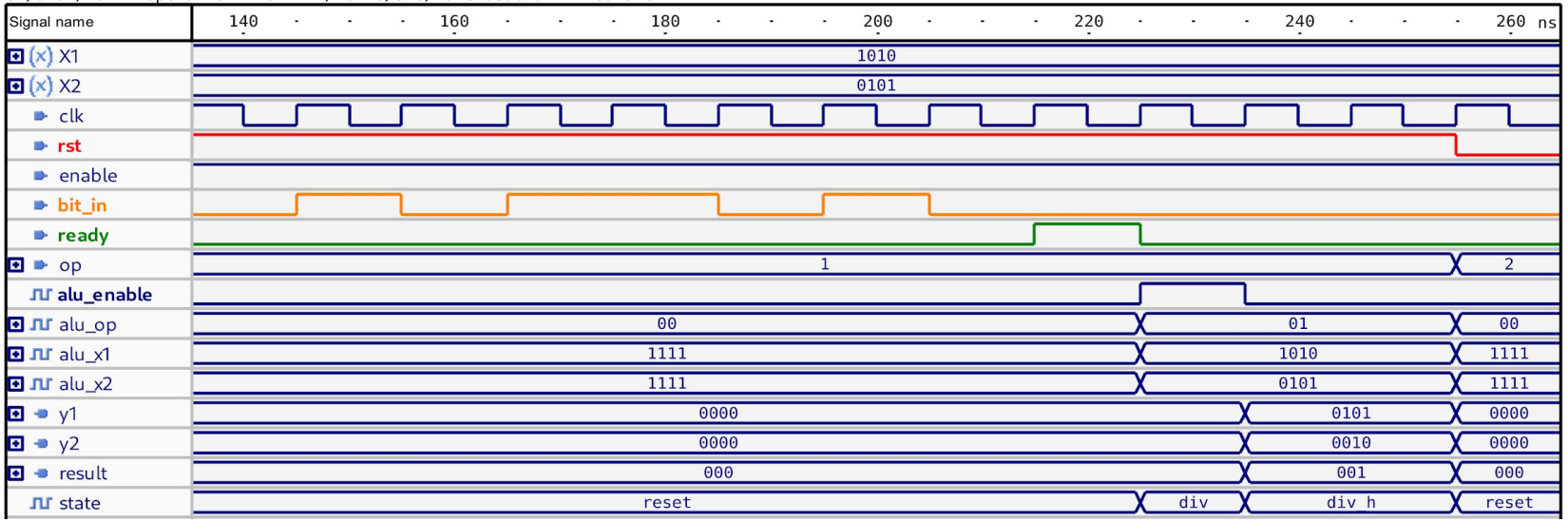
```
x1 [3:0]
x2 [3:0]
clk: 10 ns
```



Testbench 2 - divisione per 2

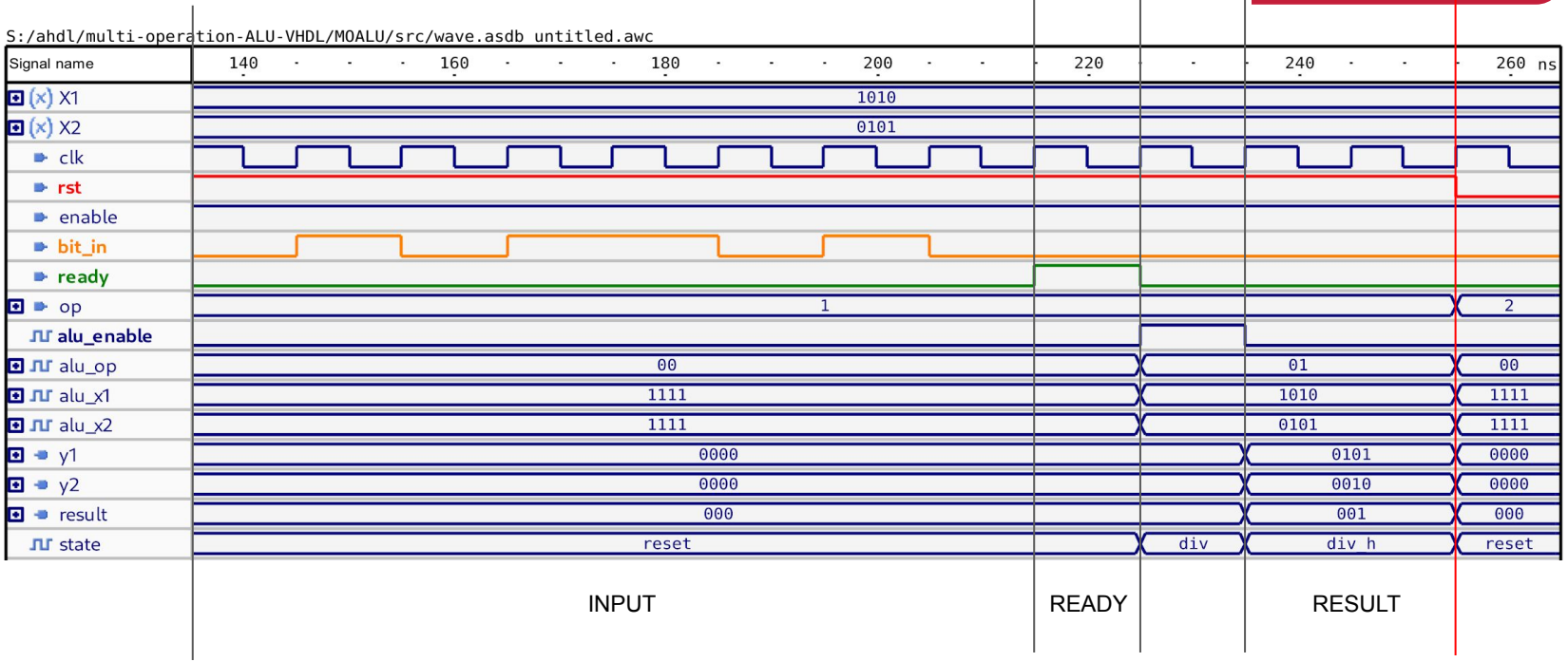
```
x1 [3:0]
x2 [3:0]
clk: 10 ns
```

S:/ahdl/multi-operation-ALU-VHDL/MOALU/src/wave.asdb untitled.awc



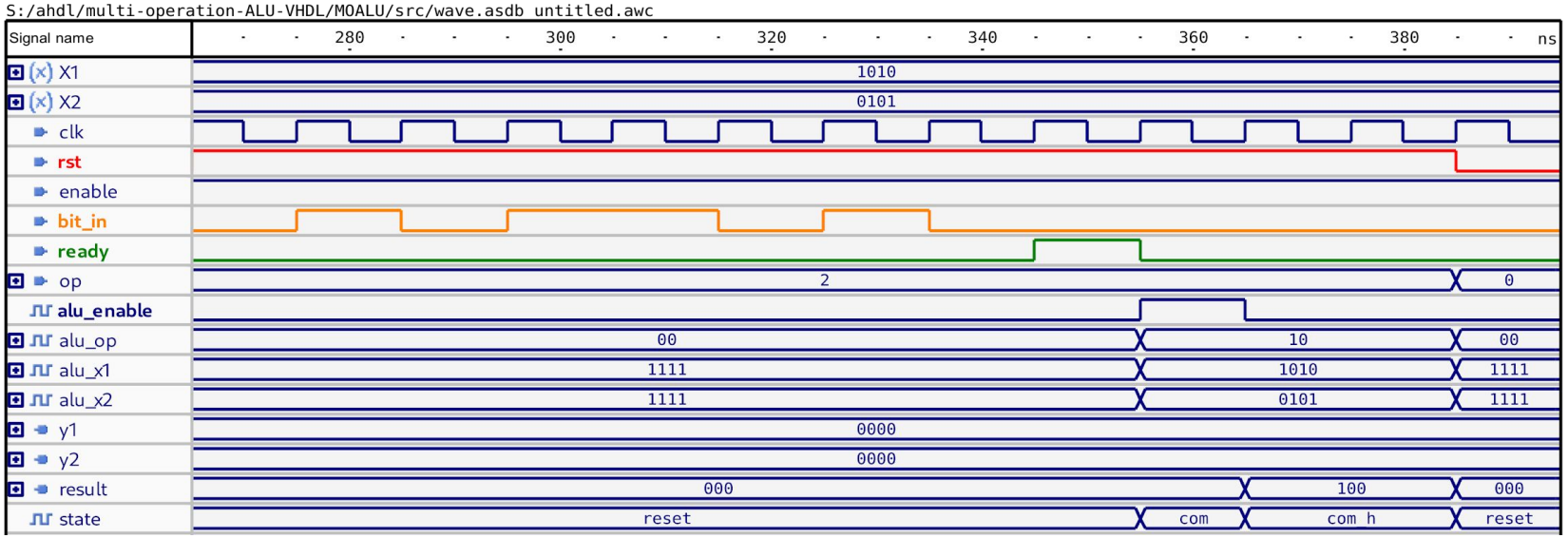
Testbench 2 - divisione per 2

```
x1 [3:0]
x2 [3:0]
clk: 10 ns
```



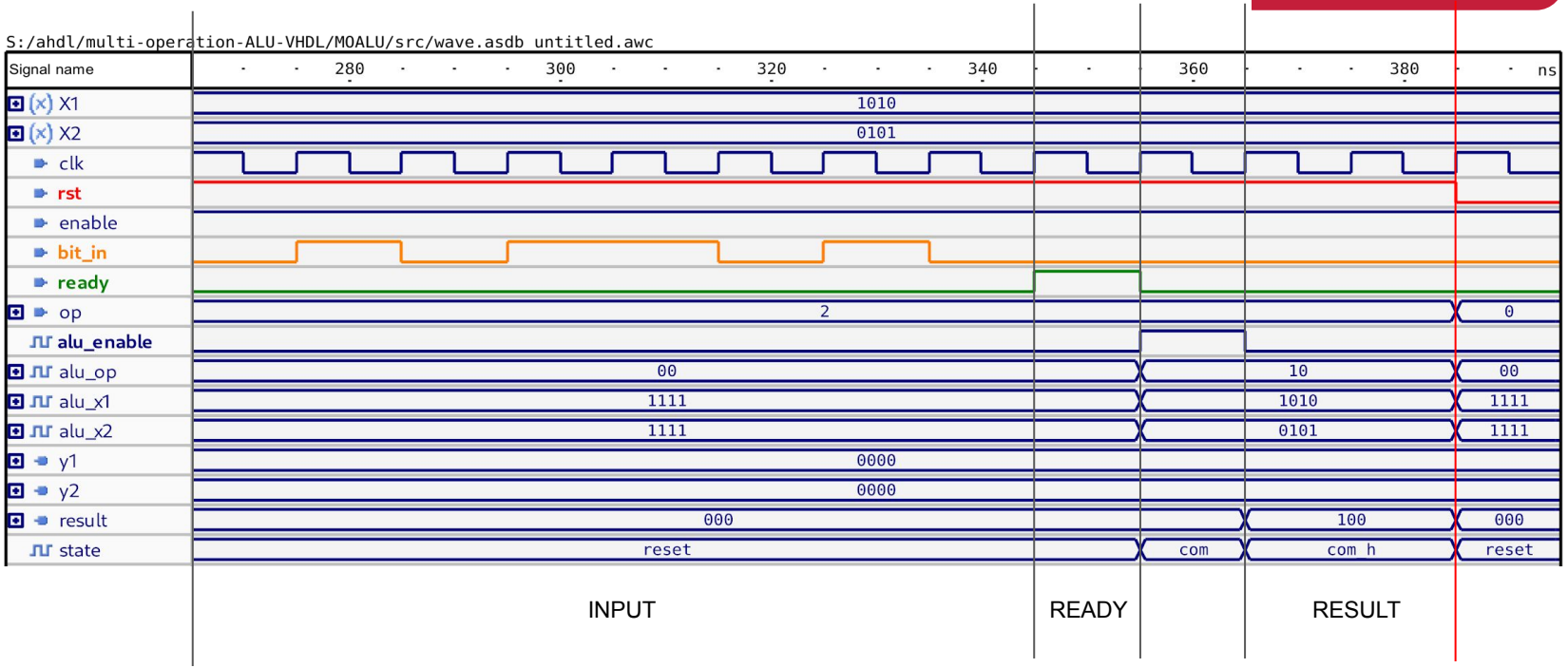
Testbench 3 - comparazione

```
x1 [3:0]
x2 [3:0]
clk: 10 ns
```



Testbench 3 - comparazione

```
x1 [3:0]
x2 [3:0]
clk: 10 ns
```



Implementazione VHDL - ENTITY

```
6  ENTITY MOALU IS
7    GENERIC(
8      DATA_WIDTH : integer := 4
9      );
10
11   PORT(
12     clk    : IN  std_logic;
13     rst    : IN  std_logic;
14     enable : IN  std_logic;
15     bit_in : IN  std_logic;
16     ready  : IN  std_logic;
17     op     : IN  std_logic_vector(1 DOWNTO 0);
18     y1     : OUT std_logic_vector((DATA_WIDTH - 1) DOWNTO 0);
19     y2     : OUT std_logic_vector((DATA_WIDTH - 1) DOWNTO 0);
20     result : OUT std_logic_vector(2 DOWNTO 0)
21     );
22 END ENTITY MOALU;
```

Implementazione VHDL - ARCHITECTURE

```
25 ARCHITECTURE MOALU_arch OF MOALU IS
26
27   SIGNAL spi_bus_out       : std_logic_vector((DATA_WIDTH*2)-1 DOWNTO 0) := (OTHERS => '0');
28   SIGNAL alu_enable        : std_logic;
29   SIGNAL alu_op            : std_logic_vector(1 DOWNTO 0);
30   SIGNAL alu_x1            : unsigned(DATA_WIDTH-1 DOWNTO 0)             := (OTHERS => '0');
31   SIGNAL alu_x2            : unsigned(DATA_WIDTH-1 DOWNTO 0)             := (OTHERS => '0');
32   SIGNAL alu_y1            : unsigned(DATA_WIDTH-1 DOWNTO 0);
33   SIGNAL alu_y2            : unsigned(DATA_WIDTH-1 DOWNTO 0);
34   SIGNAL alu_result        : std_logic_vector(2 DOWNTO 0);
35   SIGNAL alu_master_enable : std_logic;
36
37 BEGIN
38
39   alu_x1 <= unsigned(spi_bus_out(DATA_WIDTH-1 DOWNTO 0));
40   alu_x2 <= unsigned(spi_bus_out((DATA_WIDTH*2)-1 DOWNTO DATA_WIDTH));
41
42   alu_master_enable <= alu_enable AND enable;
```

Implementazione VHDL - MAPPING

```
SPI : ENTITY work.SPI
  GENERIC MAP (
    DATA_WIDTH => DATA_WIDTH
    )
  PORT MAP (
    clk     => clk,
    rst     => rst,
    bit_in  => bit_in,
    ready   => ready,
    enable  => enable,
    bus_out => spi_bus_out
    );
```

```
FSM : ENTITY work.FSM
  PORT MAP(
    clk         => clk,
    rst         => rst,
    enable      => enable,
    ready       => ready,
    op          => op,
    alu_enable  => alu_enable,
    alu_op      => alu_op
    );
```

```
ALU : ENTITY work.ALU
  GENERIC MAP (
    DATA_WIDTH => DATA_WIDTH
    )
  PORT MAP (
    enable => alu_master_enable,
    op     => alu_op,
    x1     => alu_x1,
    x2     => alu_x2,
    y1     => alu_y1,
    y2     => alu_y2,
    result => alu_result
    );
```

```
reg_out : ENTITY work.reg_out
  GENERIC MAP (
    DATA_WIDTH => DATA_WIDTH
    )
  PORT MAP (
    clk         => clk,
    rst         => rst,
    enable      => alu_master_enable,
    y1_in       => std_logic_vector(alu_y1),
    y2_in       => std_logic_vector(alu_y2),
    result_in   => alu_result,
    y1_out      => y1,
    y2_out      => y2,
    result_out => result
    );
```

Conclusioni

- Il sistema rispetta le specifiche definite nella traccia
- Tutti i testbench ottengono in output il risultato atteso