UNIVERSITÀ DEGLI STUDI DI TRIESTE

dipartimento di ingegneria e architettura

Pipelined MAC

A.Carini – Progettazione di sistemi elettronici

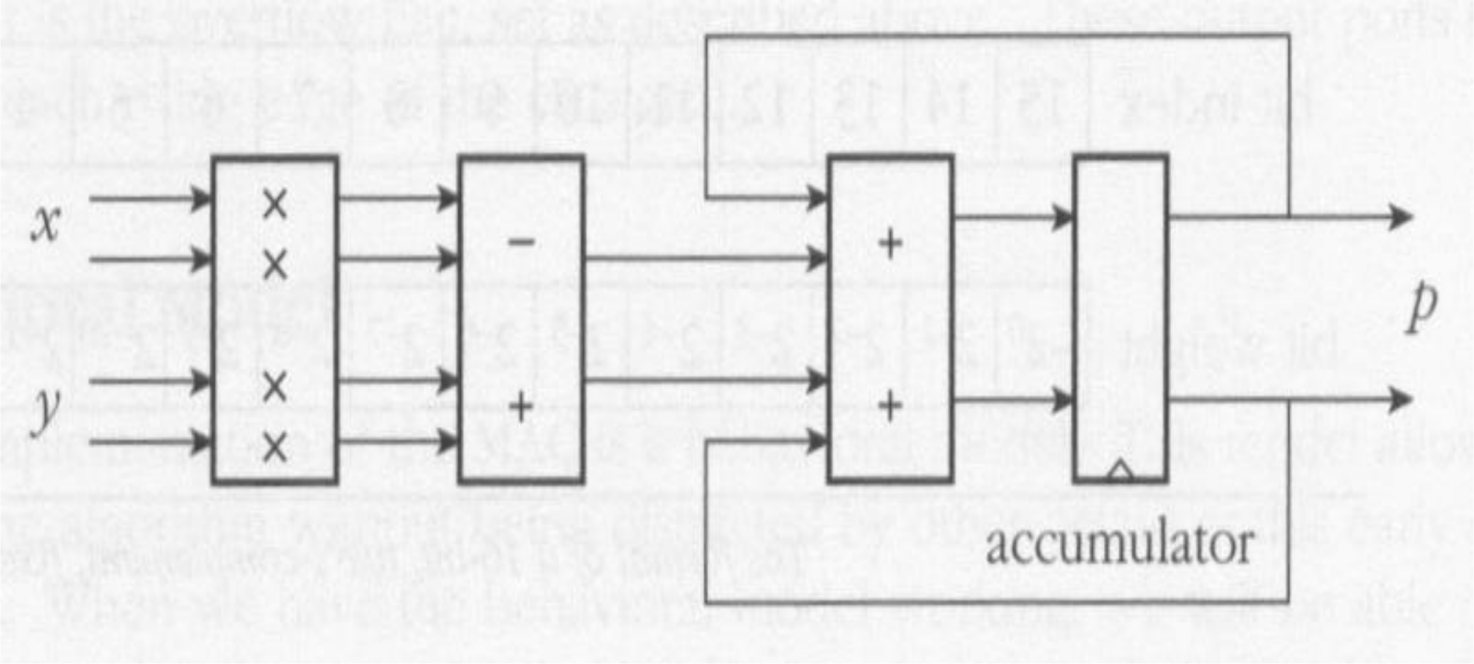
MAC between complex numbers

$$\sum_{i=1}^{N} x_i y_i$$

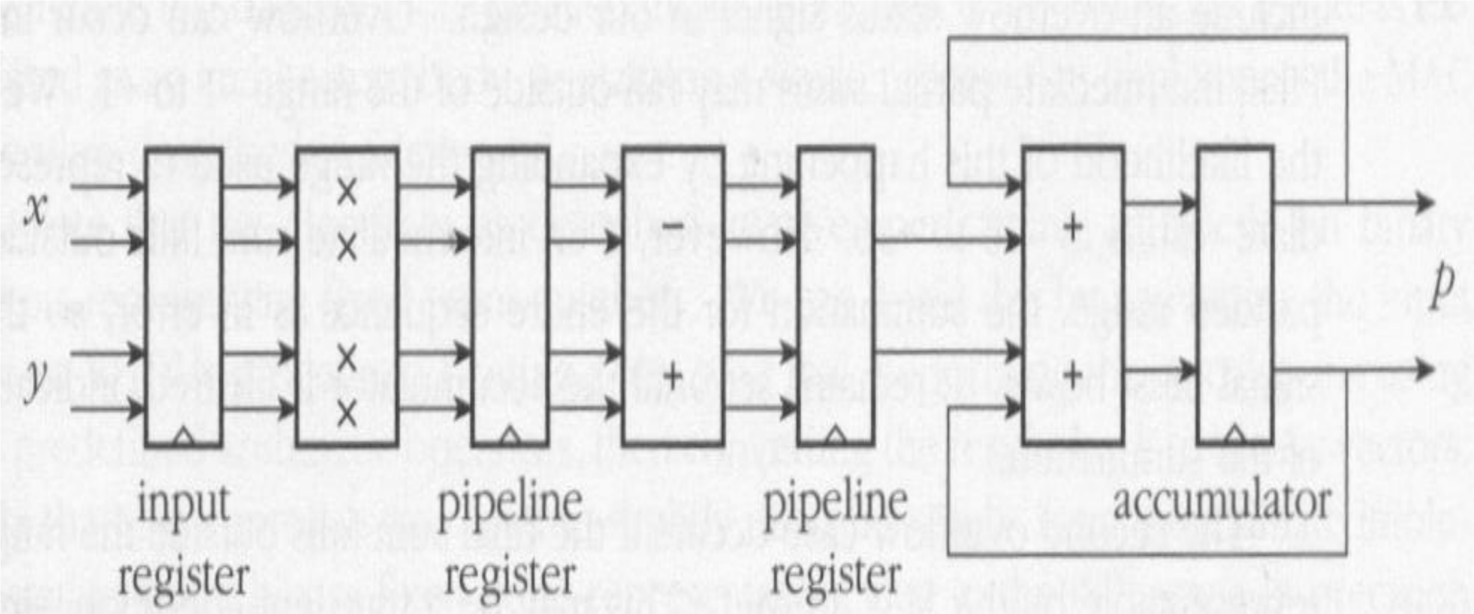
$$p_{real} = x_{real} \times y_{real} - x_{imag} \times y_{imag}$$
$$p_{imag} = x_{real} \times y_{imag} + x_{imag} \times y_{real}$$

$$s_{real} = x_{real} + y_{real}$$
$$s_{imag} = x_{imag} + y_{imag}$$

MAC between complex numbers



Pipelined MAC between complex numbers



Fixed point numeric format Q1.15

bit index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bit weight	-2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}

The format of a 16-bit, two's-complement, fixed-point binary number.

The entity declaration

```
library ieee;  use ieee.std_logic_1164.all;
entity mac is
    port ( clk, clr : in std_ulogic;
           x_real : in std_ulogic_vector(15 downto 0);
           x_imag : in std_ulogic_vector(15 downto 0);
           y_real : in std_ulogic_vector(15 downto 0);
           y_imag : in std_ulogic_vector(15 downto 0);
           s_real : out std_ulogic_vector(15 downto 0);
           s_imag : out std_ulogic_vector(15 downto 0);
           ovf : out std_ulogic );
end entity mac;
```

Behavioral model

- In the first implementation we focus on the algorithm.
- Instead of doing the numeric operations on binary data, in the behavioral model we can:
  - Transform the input data in values of type *real*,
  - Perform all operations with the predefined operators,
  - Transform the real result in the output binary format.

Binary/real conversion functions

- Study the attached modules in the following order:
  - to_fp.vhdl entity to_fp
  - to_fp-behavioral.vhdl architecture behavioral of to_fp
  - to_fp_test.vhdl entity to_fp_test
  - to_fp_test-bench.vhdl architecture bench of to_fp_test

  - to_vector.vhdl entity to_vector
  - to_vector-behavioral.vhdl architecture behavioral of to_vector
  - to_vector_test.vhdl entity to_vector_test
  - to_vector_test-bench.vhdl architecture bench of to_vector_test

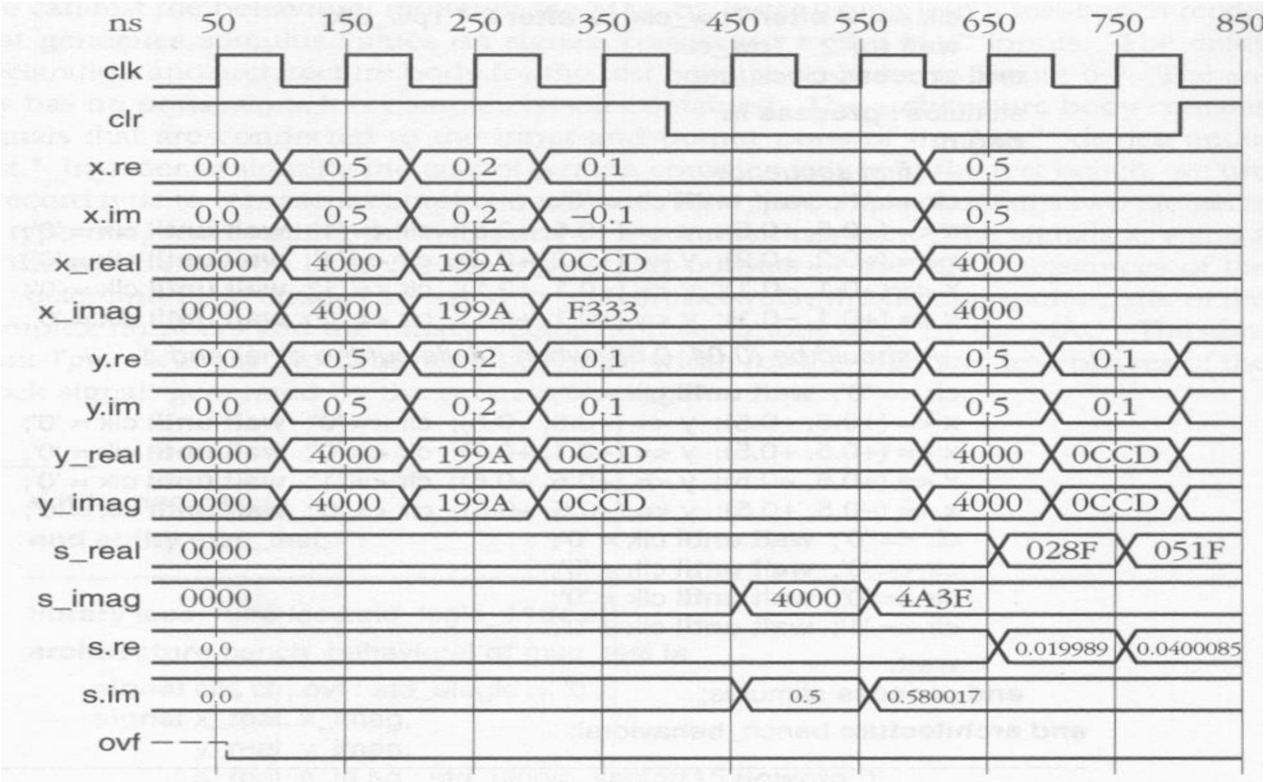
Behavioral model of MAC

- Study:
  - mac.vhdl entity mac
  - mac-behavioral.vhdl architecture behavioral of mac
- Note how all pipeline computations are performed from the output to the input in order to simulate the behavior of pipeline.
- N.B.: it is assumed to have 5 guard bits in the accumulator (even though the result has 16 bits), and to have a sufficient number of guard bits for summations and partial products.

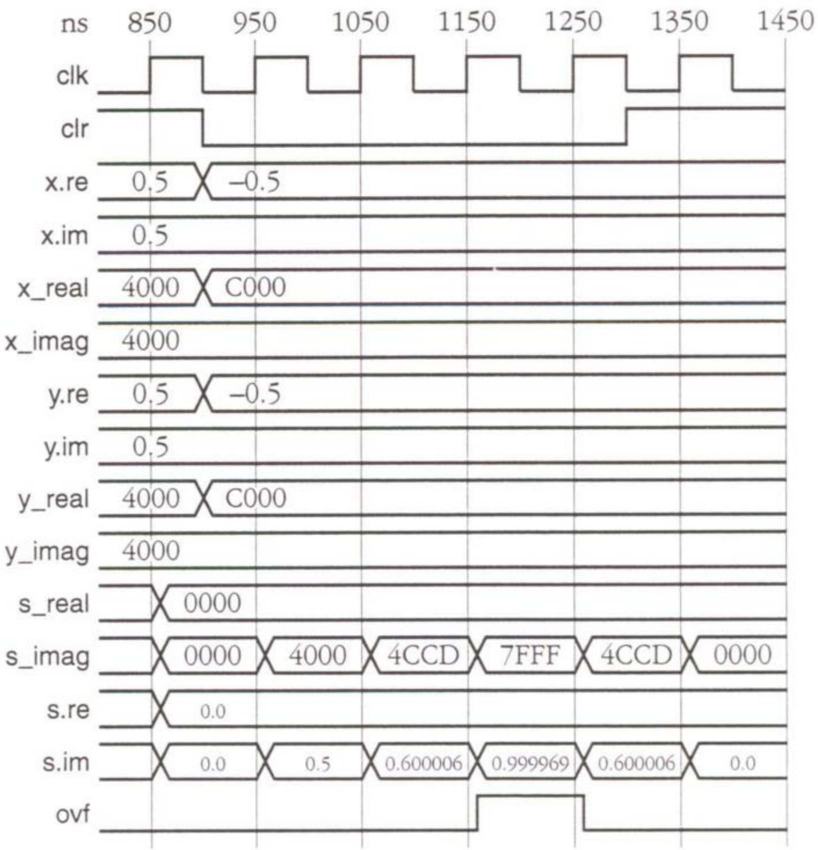
Behavioral model test

- Study:
  - mac_test.vhdl — entity mac_test
  - mac_test-bench_behavioral.vhdl — architecture bench_behavioral of mac_test
- The complex type has been introduced for simplifying the test bench implementation.
- The first sequence verifies the sum, the second sequence verifies the overflow detection.

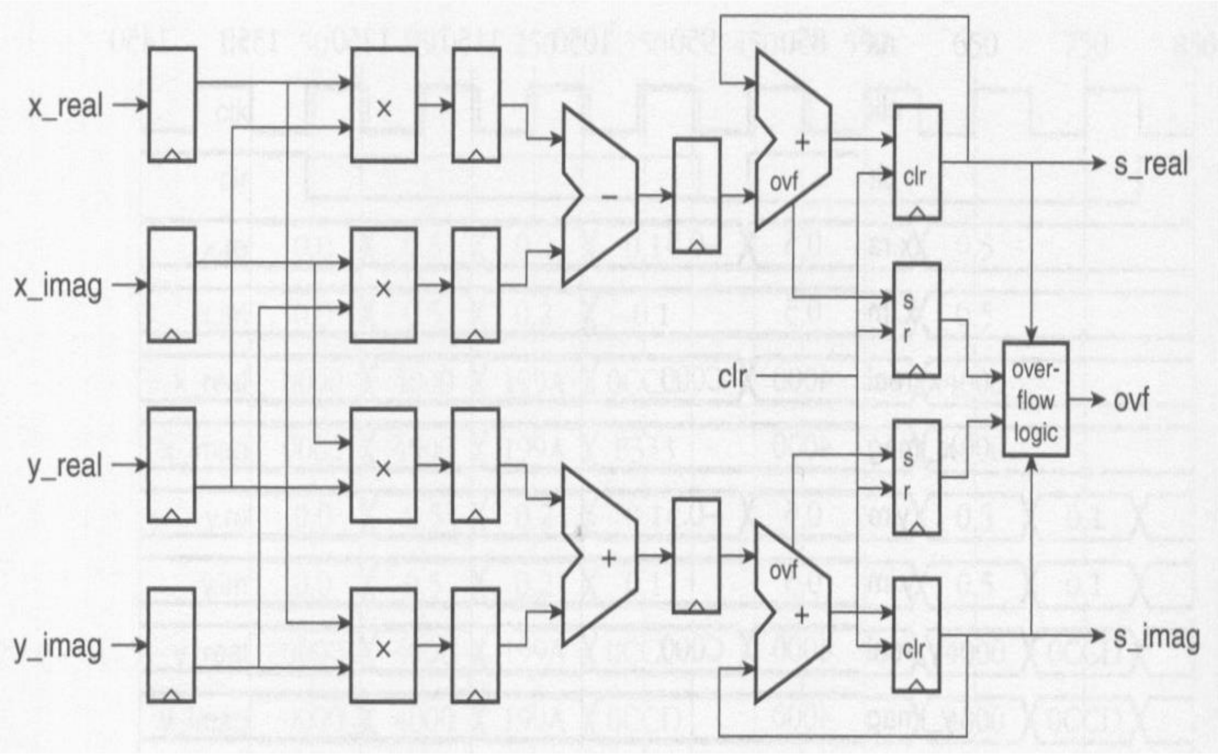
Result of test (1)



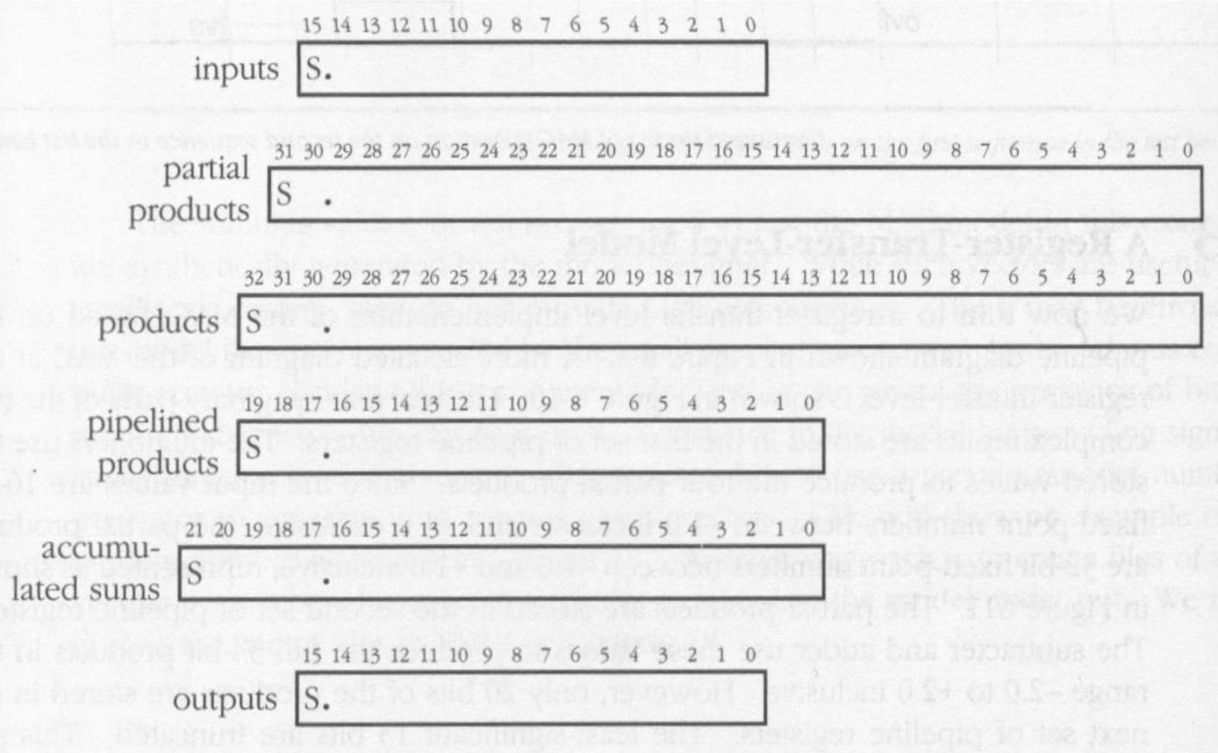
Result of test (2)



RTL model


x_real
x_imag
y_real
y_imag
clr
s_real
ovf
s_imag
over-
flow
logic
ovf
ovf
clr
clr
s
r
s
r

Data format



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
inputs
S.
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
partial products
S .
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
products
S .
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
pipelined products
S .
21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
accumu-lated sums
S .
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
outputs
S.

RTL model modules

- Study in the following order:
  - reg.vhdl — entity reg
  - reg-behavioral.vhdl — architecture behavioral of reg
  - multiplier.vhdl — entity multiplier
  - multiplier-behavioral.vhdl — architecture behavioral of multiplier
  - multiplier_test.vhdl — entity multiplier_test
  - multiplier_test-bench.vhdl — architecture bench of multiplier_test

RTL model modules

- product_adder_subtracter.vhdl
  entity product_adder_subtracter
- product_adder_subtracter-behavioral.vhdl
  architecture behavioral of behavioral
- accumulator_adder.vhdl
  entity accumulator_adder
- accumulator_adder-behavioral.vhdl
  architecture behavioral of accumulator_adder
- accumulator_reg.vhdl entity accumulator_reg
- accumulator_reg-behavioral.vhdl
  architecture behavioral of accumulator_reg

RTL model modules

– synch_sr_ff.vhdl	entity synch_sr_ff
– synch_sr_ff-behavioral.vhdl	architecture behavioral of synch_sr_ff
– overflow_logic.vhdl	entity overflow_logic
– overflow_logic-behavioral.vhdl	architecture behavioral of overflow_logic
– mac-rtl.vhdl	architecture rtl of mac (pay attention to the final scaling to obtain a Q1.15 data format)

RTL model modules

- The test can be performed with same test bench we used for the behavioral model or by comparing the RTL model with the behavioral model.
- Study:
  - mac_test-bench_rtl.vhdl
  - architecture bench_rtl of mac_test
  - mac_test-bench_verify.vhdl
  - architecture bench_verify of mac_test

See:

- Peter Ashenden, «The designers’ guide to VHDL» Morgan Kaufmann, ed 2002
  - Chapter 13