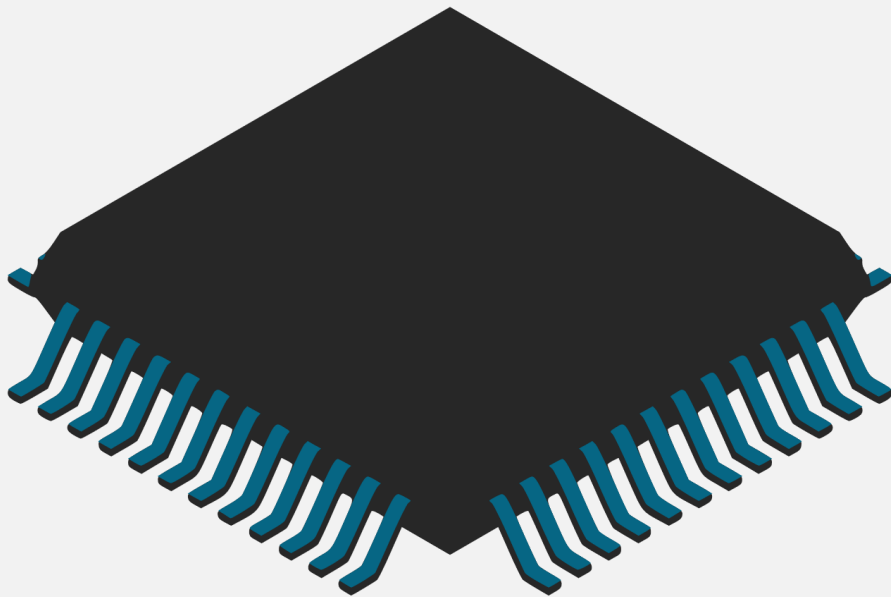


# ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά 1<sup>ης</sup> Εργαστηριακής Άσκησης

---

«ΣΧΕΔΙΑΣΗ ΚΑΙ ΠΡΟΣΩΜΟΙΩΣΗ ΜΟΝΑΔΑΣ ΑΡΙΘΜΗΤΙΚΩΝ  
ΚΑΙ ΛΟΓΙΚΩΝ ΠΡΑΞΕΩΝ ΚΑΙ ΑΡΧΕΙΟΥ ΚΑΤΑΧΩΡΗΤΩΝ»






















 Ζαχαριουδάκης Νικόλας 2016030073

 Γαλάνης Μιχάλης 2016030036

# ΠΕΡΙΕΧΟΜΕΝΑ

*Οι σύνδεσμοι για τις παρακάτω ενότητες είναι διαδραστικοί.  
Πιέστε πάνω στο επιθυμητό τμήμα για τη μετάβαση σε αυτό.*

	<b>ΕΙΣΑΓΩΓΗ.....</b>	<b>1</b>
	Σκοπός Εργαστηρίου .....	1
	Προαπαιτούμενα.....	1
	<b>A - ΣΧΕΔΙΑΣΗ ALU .....</b>	<b>1</b>
	A. Επισκόπηση .....	1
	A. Σχεδίαση & Υλοποίηση .....	2
	A. Κώδικας VHDL (Signals) .....	3
	A. Κώδικας VHDL (Main) .....	3
	A. Επαλήθευση Συστήματος (Δημιουργία Testbench) .....	5
	A. Προσομοίωση Συστήματος (Κυματομορφές) .....	6
	<b>B - ΚΑΤΑΣΚΕΥΗ REGISTER FILE.....</b>	<b>7</b>
	B.1. Παραγωγή Register - Επισκόπηση .....	7
	B.1. Σχεδίαση & Υλοποίηση .....	7
	B.1. Κώδικας VHDL.....	8
	B.1. Επαλήθευση Συστήματος (Δημιουργία Testbench) .....	8
	B.1. Προσομοίωση Συστήματος (Κυματομορφές) .....	8
	B.2. Register File - Επισκόπηση .....	9
	B.2.1 Σχεδίαση & Υλοποίηση (Components).....	10
	B.2.1 Κώδικας VHDL (Decoder 5-To-32).....	10

	B.2.1 Κώδικας VHDL (Multiplexer 1024-To-32) .....	10
	B.2.2 Σχεδίαση & Υλοποίηση (TOP MODULE).....	11
	B.2.2 Κώδικας VHDL (TOP MODULE Signals) .....	11
	B.2.2 Κώδικας VHDL (TOP MODULE Main) .....	11
	B.2. Επαλήθευση Συστήματος (Δημιουργία Testbench) .....	12
	B.2. Προσομοίωση Συστήματος (Κυματομορφές) .....	13

# ΕΙΣΑΓΩΓΗ

## Σκοπός Εργαστηρίου

Σκοπός της πρώτης εργαστηριακής άσκησης της οργάνωσης υπολογιστών είναι η επανάληψη της γλώσσας VHDL που υλοποιείται μέσω της σχεδίασης 2 συστημάτων:

- (Μέρος Α) : Μονάδα αριθμητικών και λογικών πράξεων
- (Μέρος Β) : Αρχείο Καταχωρητών MIPS

## Προαπαιτούμενα

Η μόνη προαπαιτούμενη ύλη είναι αυτή της Προχωρημένης Λογικής Σχεδίασης, δηλαδή η καλή κατανόηση της VHDL σε behavioral αλλά και structural μορφή καθώς και η εξοικείωση με εργαλεία σχεδιασμού της Xilinx.

## Α - ΣΧΕΔΙΑΣΗ ALU

### Α. Επισκόπηση

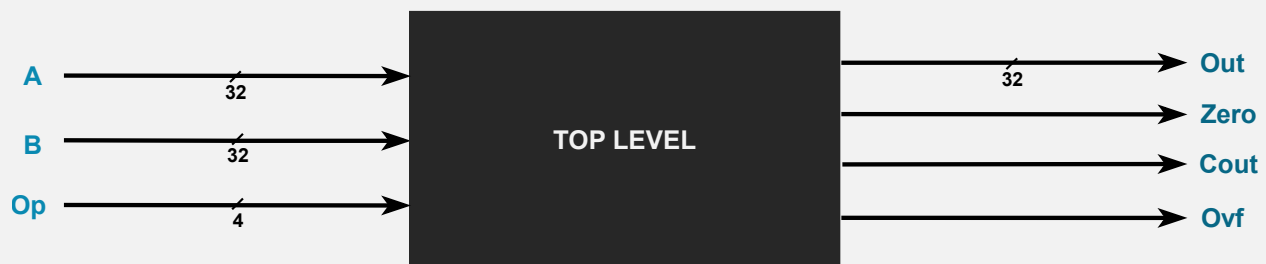
Ξεκινάμε από τις απαιτήσεις αυτού του συστήματος. Θεωρούμε ότι η μονάδα έχει τις εξής εισόδους και εξόδους:

Όνομα	Είδος	Πλάτος	Λειτουργία
<b>A</b>	<b>Είσοδος</b>	<b>32 bits</b>	Πρώτος τελεστέος σε συμπλήρωμα ως προς 2
<b>B</b>	<b>Είσοδος</b>	<b>32 bits</b>	Δεύτερος τελεστέος σε συμπλήρωμα ως προς 2
<b>Op</b>	<b>Είσοδος</b>	<b>4 bits</b>	Κωδικός πράξης
<b>Out</b>	<b>Έξοδος</b>	<b>32 bits</b>	Αποτέλεσμα σε συμπλήρωμα ως προς 2
<b>Zero</b>	<b>Έξοδος</b>	<b>1 bit</b>	Ενεργοποιημένη αν το αποτέλεσμα είναι μηδέν
<b>Cout</b>	<b>Έξοδος</b>	<b>1 bit</b>	Έλεγχος ύπαρξης κρατούμενου εξόδου
<b>Ovf</b>	<b>Έξοδος</b>	<b>1 bit</b>	Ενεργοποιημένη αν υπήρξε υπερχείλιση

Η είσοδος Op δηλώνει τον κωδικό της επιθυμητής πράξης. Παρακάτω φαίνονται οι επιθυμητές πράξεις που πρέπει να γίνουν ανάλογα με την τιμή της:

Κωδικός Op	Πράξη
<b>0000</b> (00)	<b>Πρόσθεση</b> ( $A + B$ )
<b>0001</b> (01)	<b>Αφαίρεση</b> ( $A - B$ )
<b>0010</b> (02)	<b>Λογικό ΚΑΙ</b> ( $A \& B$ )
<b>0011</b> (03)	<b>Λογικό Η</b> ( $A   B$ )
<b>0100</b> (04)	<b>Αντιστροφή</b> ( $\neg A$ )
<b>1000</b> (08)	<b>Αριθμητική Ολίσθηση Δεξιά</b>
<b>1001</b> (09)	<b>Λογική Ολίσθηση Δεξιά</b>
<b>1010</b> (10)	<b>Λογική Ολίσθηση Αριστερά</b>
<b>1100</b> (12)	<b>Κυλική Ολίσθηση Αριστερά</b>
<b>1101</b> (13)	<b>Κυλική Ολίσθηση Δεξιά</b>

Σε επόμενο σχήμα φαίνεται η σχηματική αναπαράσταση της αριθμητικής & λογικής μονάδας με βάση τις παραπάνω εισόδους & εξόδους.



## A. Σχεδίαση & Υλοποίηση

Όσον αφορά τη VHDL, το TOP LEVEL είναι υλοποιημένο εξ'ολοκλήρου σε behavioral δομή, καθώς μας δίνει τα απλούστερα αποτελέσματα, οπότε δεν ασχολούμαστε με υποσυστήματα αθροιστών/αφαιρετών κτλ.

Το κύκλωμα είναι απλό, συνδυαστικό δεν απαιτεί μηχανή πεπερασμένων καταστάσεων. Χρησιμοποιήσαμε τα ακόλουθα 3 βασικά πακέτα της IEEE:

**IEEE.STD\_LOGIC\_1164.ALL**, **IEEE.STD\_LOGIC\_unsigned.ALL** και **IEEE.numeric\_STD.ALL**.

Η εκφώνηση απαιτούσε επίσης καθυστέρηση των εξόδων κατά 10ns. Για αυτό, χρησιμοποιήσαμε εσωτερικά **temp signals**, στα οποία γίνονταν οι πράξεις και οι

τελικές έξοδοι δεν ήταν τίποτα παραπάνω από αυτά signals στα οποία εφαρμόζονταν η καθυστέρηση.

**⚠ Παρατήρηση 1:** Χρησιμοποιήσαμε ένα process για κάθε έξοδο με κατάλληλο κάθε φορά sensitivity list, κυρίως για λόγους οργάνωσης.

## A. Κώδικας VHDL (Signals)

Ενδιάμεσα σήματα που χρησιμοποιήθηκαν:

```
--signals
signal temp_out: STD_LOGIC_VECTOR (31 downto 0);
signal temp_A: STD_LOGIC_VECTOR (32 downto 0);
signal temp_B: STD_LOGIC_VECTOR (32 downto 0);
signal temp_carry: STD_LOGIC_VECTOR (32 downto 0);
signal temp_ovf: STD_LOGIC;
signal temp_zero: STD_LOGIC;
```

## A. Κώδικας VHDL (Main)

Κύριο μέρος του κώδικα για την υλοποίηση του συστήματος:

```
begin

-- Output
outpt: process (A,B,Op)
begin
    -- temp for output
    if Op="0000" then
        temp_out <= A + B;
    elsif Op="0001" then
        temp_out <= A - B;
    elsif Op="0010" then
        temp_out <= A and B;
    elsif Op="0011" then
        temp_out <= A or B;
    elsif Op="0100" then
        temp_out <= NOT A;
    elsif Op="1000" then
        temp_out <= STD_LOGIC_VECTOR(shift_right(signed(A),1));
    elsif Op="1001" then
        temp_out <= STD_LOGIC_VECTOR(shift_right(unsigned(A),1));
    elsif Op="1010" then
        temp_out <= STD_LOGIC_VECTOR(shift_left(unsigned(A),1));
    elsif Op="1100" then
        temp_out <= STD_LOGIC_VECTOR(rotate_left(signed(A),1));
    elsif Op="1101" then
        temp_out <= STD_LOGIC_VECTOR(rotate_right(signed(A),1));
    else
        temp_out <= "00000000000000000000000000000000";
    end if;
end process;
--Overflow
```

```

overflow: process(A,B,Op,temp_out)
begin
    if Op="0000" then
        --Overflow
        if A(31)='0' and B(31)='0' and temp_out(31) = '1' then
            temp_ovf <='1';
        elsif A(31)='1' and B(31)='1' and temp_out(31) = '0' then
            temp_ovf <='1';
        else
            temp_ovf <='0';
        end if;
    elsif Op="0001" then
        --Overflow
        if A(31)='1' and B(31)='0' and temp_out(31) = '0' then
            temp_ovf <='1';
        elsif A(31)='0' and B(31)='1' and temp_out(31) = '1' then
            temp_ovf <='1';
        else
            temp_ovf <='0';
        end if;
    else
        temp_ovf <='0';
    end if;
end process;

-- carry out
carr: process(A,B,Op)
begin
    temp_A <= '0' & A;
    temp_B <= '0' & B;
    if Op="0000" then
        temp_carry <= temp_A + temp_B;
    elsif Op="0001" then
        temp_carry <= temp_A - temp_B;
    else
        temp_carry <= "00000000000000000000000000000000";
    end if;
end process;

-- zero
Zr: process(temp_out)
begin
    if temp_out="00000000000000000000000000000000" then
        temp_zero <= '1';
    else
        temp_zero <= '0';
    end if;
end process;

Ovf <= temp_ovf after 10ns;
Output <= temp_out after 10ns;
zero <= temp_zero after 10ns;
Cout <= temp_carry(32) after 10ns;

end Behavioral;

```

**⚠ Παρατήρηση 2:** Δε παραθέτουμε ολόκληρο τον κώδικα του VHD αρχείου παρά μόνο ένα τμήμα που μας ενδιαφέρει (συνήθως ανάμεσα στις εντολές **begin ... end** του architecture).

## A. Επαλήθευση Συστήματος (Δημιουργία Testbench)

Δημιουργήσαμε ένα testbench για το TOP MODULE για να επαληθεύσουμε την ορθή λειτουργία του. Όπως μας συμβουλεύει και η εκφώνηση είναι αδύνατο να δοκιμάσουμε όλες τις περιπτώσεις ειδικά όταν έχουμε 32 bit τελεστές. Για αυτό επιλέγουμε διακριτές «ενδιαφέρουσες» περιπτώσεις (corner cases) που έχουν μεγαλύτερη πιθανότητα να εμφανιστεί κάποιο λάθος. Για να δοκιμαστούν αρκετά οι λειτουργίες overflow & carry out, επιλέγουμε τελεστές με «δραστηριότητα» στα τελευταία most significant bits.

```
begin
    wait for 50 ns;
    -----
    --    ADD SUB
    A <= "01111000000000000000000000000000";
    B <= "00111000000000000000000000000000";
    Op <= "0000";
    wait for 50 ns;
    Op <= Op +1;
    wait for 50 ns;

    --    ADD SUB
    A <= "11110000000000000000000000000000";
    B <= "11100000000000000000000000000000";
    Op <= "0000";
    wait for 50 ns;
    Op <= Op +1;
    wait for 50 ns;

    --    ADD SUB AND OR NOT
    A <= "10000000000000000000000000000000";
    B <= "11111111111111111111111111111111";
    Op <= "0000";
    wait for 50 ns;
    for i in 1 to 3 loop
        Op <= Op +1;
        wait for 50 ns;
    end loop;

    -- SHIFT_RIGHT SHIFT_LEFT
    A <= "10000000000000000000000000000001";
    Op <= "1000";
    wait for 50 ns;
    for i in 1 to 2 loop
        Op <= Op +1;
        wait for 50 ns;
    end loop;
```



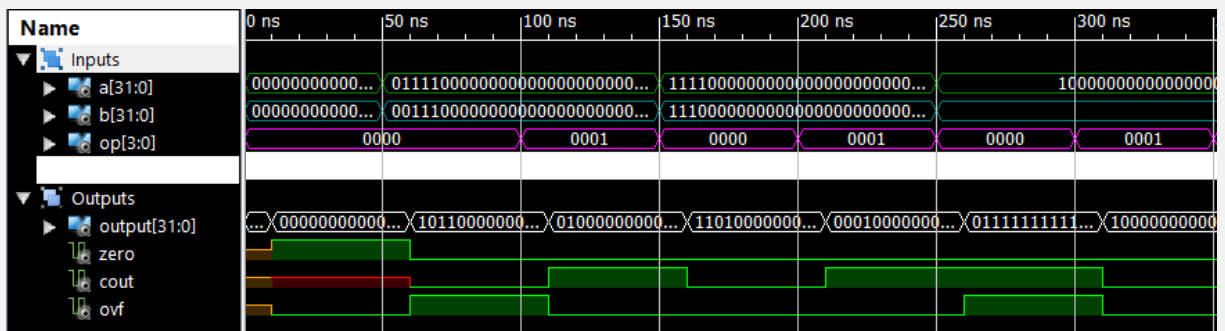
```

-- ROTATE
A <= "00011010101010101010101010100011";
Op <= "1100";
wait for 50 ns;
for i in 1 to 2 loop
    Op <= Op +1;
    wait for 50 ns;
end loop;
wait;
end process;

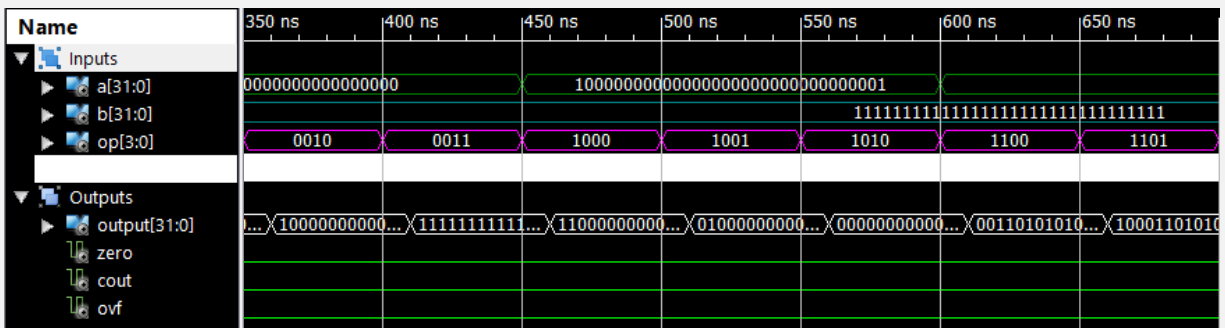
```

## 🕒 A. Προσομοίωση Συστήματος (Κυματομορφές)

0 ns – 350 ns:



350 ns – 700 ns:



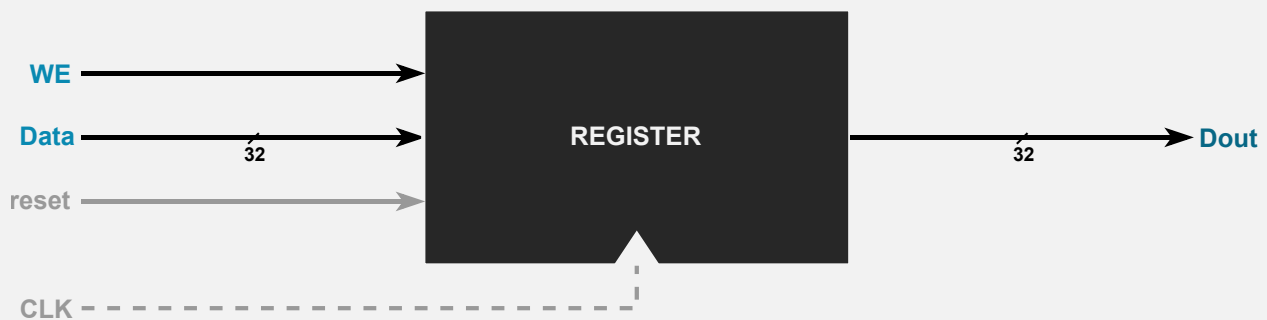
## B - ΚΑΤΑΣΚΕΥΗ REGISTER FILE

Για το δεύτερο μέρος του εργαστηρίου, κληθήκαμε να σχεδιάσουμε ένα καταχωρητή των 32 bit σε behavioral δομή για να χρησιμοποιηθεί ύστερα για τη κατασκευή ενός αρχείου καταχωρητών που συνδυάζει behavioral & structural δομή.

### B.1. Παραγωγή Register - Επισκόπηση

Ο καταχωρητής αποτελεί σύγχρονο κύκλωμα και χρησιμοποιούμε την έκδοσή του με Write Enable αλλά και Reset. Παρακάτω παραθέτουμε τις εισόδους/εξόδους του καταχωρητή καθώς και τη σχηματική του αναπαράσταση:

Όνομα	Είδος	Πλάτος	Λειτουργία
CLK	Είσοδος	1 bit	Ρολόι
reset	Είσοδος	1 bit	Reset για αρχικοποίηση
WE	Είσοδος	1 bit	Υπεύθυνο για ενεργοποίηση της εγγραφής
Data	Είσοδος	32 bits	Δεδομένα για εγγραφή
Dout	Έξοδος	32 bits	Αποτέλεσμα της εγγραφής



### B.1. Σχεδίαση & Υλοποίηση

Για τον καταχωρητή, χρησιμοποιήσαμε και πάλι behavioral σχεδίαση. Αυτή τη φορά, σε ένα process, όταν έχουμε θετική ακμή ρολογιού ελέγχουμε αν το write enable είναι ενεργοποιημένο, όπου στη περίπτωση αυτή θα επαναγράψουμε τα δεδομένα με τη νέα τιμή της εισόδου Data. Διαφορετικά, θα κρατήσουμε την παλιά.

Η καθυστέρηση υλοποιείται όπως και προηγουμένως με την εντολή **after**.

## B.1. Κώδικας VHDL

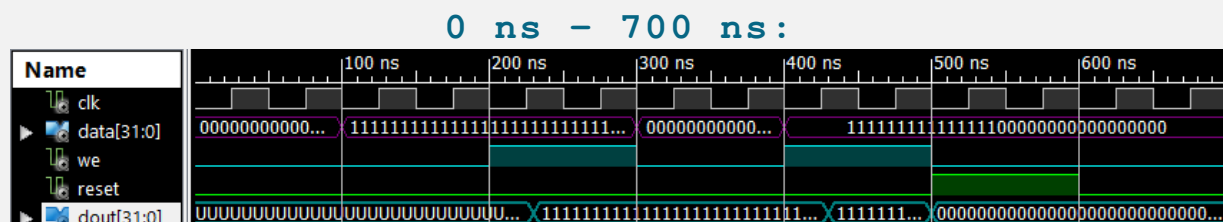
```
begin
  output: process(clk, reset)
  begin
    if reset='1' then
      Dout <= (others => '0');
    elsif rising_edge(clk) then
      if WE = '1' then
        Dout <= Data after 5ns;
      end if;
    end if;
  end process;
end Behavioral;
```

## B.1. Επαλήθευση Συστήματος (Δημιουργία Testbench)

```
begin
  -- hold reset state for 100 ns.
  wait for 100 ns;
  WE <= '0';
  Data <= "11111111111111111111111111111111";
  wait for clk_period*2;
  WE <= '1';
  Data <= "11111111111111111111111111111111";
  wait for clk_period*2;
  WE <= '0';
  Data <= "00000000000000000111111111111111";
  wait for clk_period*2;
  WE <= '1';
  Data <= "11111111111111111000000000000000";
  wait for clk_period*2;
  WE <= '0';
  Reset <= '1';
  wait for clk_period*2;
  reset <= '0';
  wait;
end process;
```

## B.1. Προσομοίωση Συστήματος (Κυματομορφές)

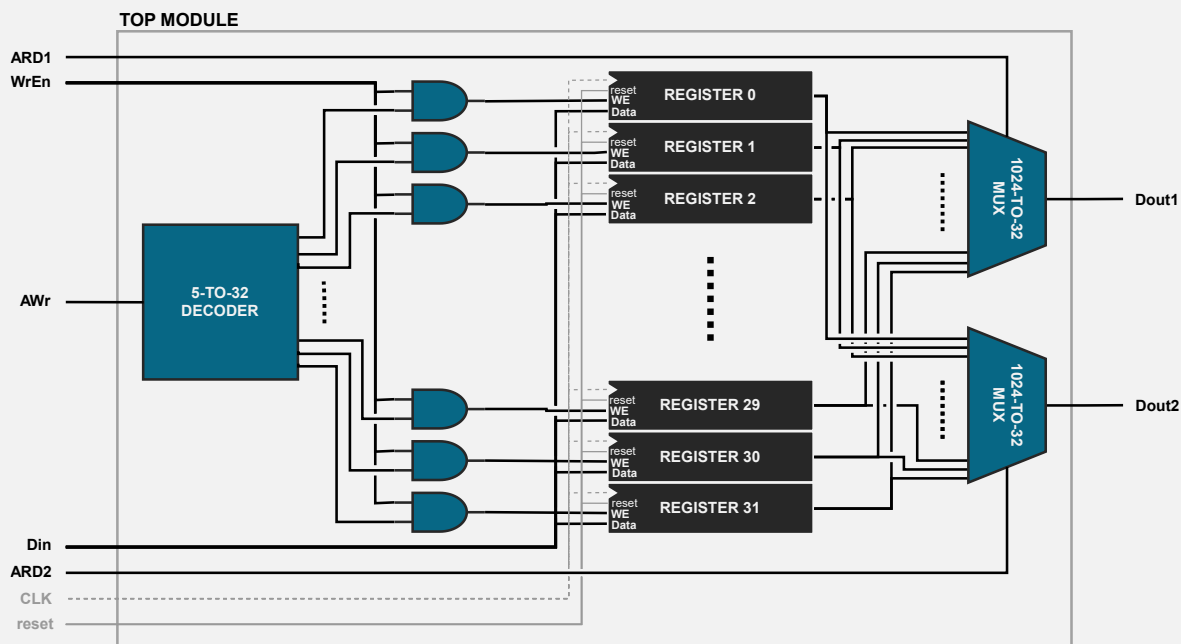
Σύμφωνα με το παραπάνω testbench, παράγεται η ακόλουθη κυματομορφή:



## 👁 B.2. Register File - Επισκόπηση

Το δεύτερο τμήμα αυτού του ερωτήματος περιλαμβάνει τη σχεδίαση ενός αρχείου καταχωρητών. Αυτό περιλαμβάνει 32 καταχωρητές των 32 bits που χρησιμοποιήσαμε προηγουμένως, έναν αποκωδικοποιητή, πύλες AND και 2 πολυπλέκτες. Παρακάτω παρουσιάζεται ο πίνακας εισόδων/εξόδων του TOP MODULE (ολικού συστήματος) και η σχηματική του αναπαράσταση. Υπενθυμίζουμε ότι καθένα από τα αναφερθείσα στοιχεία υλοποιούνται σε behavioral δομή και όλα μαζί συνδέονται τελικά για να σχηματιστεί το TOP MODULE (structural δομή).

Όνομα	Είδος	Πλάτος	Λειτουργία
CLK	Είσοδος	1 bit	Ρολόι
reset	Είσοδος	1 bit	Reset για αρχικοποίηση
Ard1	Είσοδος	5 bits	Διεύθυνση πρώτου καταχωρητή για ανάγνωση
Ard2	Είσοδος	5 bits	Διεύθυνση δεύτερου καταχωρητή για ανάγνωση
AWr	Είσοδος	5 bits	Διεύθυνση καταχωρητή για εγγραφή
WrEn	Είσοδος	1 bit	Ενεργοποίηση εγγραφής καταχωρητή
Din	Είσοδος	32 bits	Δεδομένα για εγγραφή
Dout1	Έξοδος	32 bits	Δεδομένα πρώτου καταχωρητή
Dout2	Έξοδος	32 bits	Δεδομένα δεύτερου καταχωρητή



### B.2.1 Σχεδίαση & Υλοποίηση (Components)

Εφόσον ο καταχωρητής αναλύθηκε σε προηγούμενο σημείο, θα επικεντρωθούμε στον αποκωδικοποιητή, στον πολυπλέκτη και στις πύλες. Και τα 3 components αποτελούν απλά συνδυαστικά κυκλώματα άρα δεν ασχολούμαστε με ρολόι.

Ο αποκωδικοποιητής διαθέτει ένα process στο οποίο μετατρέπουμε τη διεύθυνση εγγραφής σε integer αριθμό για να παράγουμε '1' στο bit εκείνο.

Οι πολυπλέκτες διαθέτουν επίσης ένα process στο οποίο μετατρέπουμε τη διεύθυνση ανάγνωσης σε integer αριθμό έτσι ώστε να περάσουμε την κατάλληλη 32άδα τιμών register στην έξοδο. Σημειώνουμε ότι η υλοποίηση των 1024 συνολικά bit εισόδων καταχωρητών υλοποιήθηκε με array package για να αποφύγουμε επανάληψη του κώδικά μας. Για το παραπάνω απαιτήθηκε η βιβλιοθήκη **work.dt\_array\_pkg.all**;

Οι πύλες AND λόγω της απλής υλοποίησης τους εφαρμόζονται κατευθείαν στη structural δομή όταν συνδέσουμε τα components.

### B.2.1 Κώδικας VHDL (Decoder 5-To-32)

```
begin
    process (Awr)
    begin
        -- we set the n bit of the output '1' and the rest of them '0'.
        -- n is the decimal format of the 5bit Awr.
        decAdr<=(to_integer(unsigned(Awr))=>'1',others =>'0') after 5ns;
    end process;
end Behavioral;
```

### B.2.1 Κώδικας VHDL (Multiplexer 1024-To-32)

```
package dt_array_pkg is
    type dt_array is array(31 downto 0) of std_logic_vector(31 downto 0);
end package;

.
.
.

begin

    process (muxArd)
    begin
        -- we set as dout the n register from the input array.
        -- n is the decimal format of the 5bit muxArd(0-31__00000-11111).
        Dout <= Reg(to_integer(unsigned(muxArd))) after 5 ns;
    end process;
end Behavioral;
```

## B.2.2 Σχεδίαση & Υλοποίηση (TOP MODULE)

Για να συναρμολογηθεί το TOP MODULE έπρεπε να δηλώσουμε όλα τα συμμετέχοντα components και με port mapping να συνδέσουμε τις εσωτερικές εισόδους και εξόδους δημιουργώντας τα απαραίτητα signals. Χρησιμοποιούμε for loop generate οπουδήποτε έχουμε επανάληψη πολλών ίδιων component για την απλούστευση του κώδικά μας.

## B.2.2 Κώδικας VHDL (TOP MODULE Signals)

```
signal register_dout: dt_array ;
signal dec_enbl : STD_LOGIC_VECTOR (31 downto 0);
signal reg_enbl : STD_LOGIC_VECTOR (31 downto 0);
```

## B.2.2 Κώδικας VHDL (TOP MODULE Main)

```
begin

-- The decoder for selecting the register we will write the din (input)
Decoder: decoder5To32
    Port map(
        Awr      => Awr,
        decAdr => dec_enbl
    );

-- Here we generate the "and" gate for each register's we signal with 2 ns
delay
WRENABLES: for i in 0 to 31 generate
    reg_enbl(i) <=    WrEn and dec_enbl(i) after 2 ns;
end generate ;

-- Generate 32 registers
RR: for i in 0 to 31 generate
    Registers: regi
        PORT MAP (   clk    => Clk,
                    Data    => Din,
                    Dout    => register_dout(i),
                    WE      => reg_enbl(i),
                    reset   => reset
                );
end generate ;

-- The multiplexer for the first output of register file
MUX_1: Mux1024To32
    PORT MAP (
        Reg      => register_dout,
        muxAdr => Ard1,
        Dout    => Dout1
    );

-- The multiplexer for the second output of register file
MUX_2: Mux1024To32
```

```

PORT MAP (
    Reg      => register_dout,
    muxAdr   => Ard2,
    Dout     => Dout2
);

end Structural;

```

## B.2. Επαλήθευση Συστήματος (Δημιουργία Testbench)

Για τα εσωτερικά components δημιουργήθηκαν testbenches που περιέχουν 1 ή 2 περιπτώσεις απλά για να επιβεβαιωθεί η λειτουργία τους και δε θα συμπεριληφθούν στην αναφορά αυτή για λόγους απλότητας. Όσον αφορά τον κώδικα του testbench για το TOP MODULE, αυτός παρουσιάζεται παρακάτω:

```

begin
    reset<='1';
    wait for clk_period*2;
    reset<='0';
    wait for 200ns;

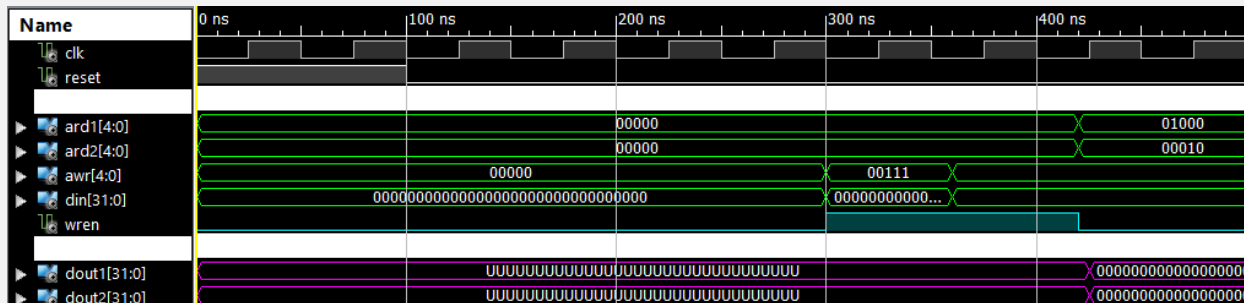
    WrEn <= '1';
    Awr <= "00111";
    Din <= "00000000000000001111111111111111" ;
    wait for 60 ns;
    Awr <= "11100";
    Din <= "11111111111111110000000000000000" ;
    wait for 60 ns;
    WrEn <= '0';
    Ard1 <= "01000";
    Ard2 <= "00010";
    wait for 100 ns;
    Ard1 <= "11100";
    Ard2 <= "00111";
    wait for 100 ns;
    Ard1 <= "00000";
    Ard2 <= "11111";
    wait for 100 ns;
    Ard1 <= "01000";
    Ard2 <= "00010";
    wait for 100 ns;
    Ard1 <= "00000";
    Ard2 <= "11111";
    wait for 100 ns;
    Ard1 <= "11100";
    Ard2 <= "00111";
    wait for 100 ns;
    Ard1 <= "01000";
    Ard2 <= "00010";
    wait for 100 ns;
    Ard1 <= "11100";
    Ard2 <= "00111";
    wait for 100 ns;
    Ard1 <= "00000";
    Ard2 <= "11111";

```

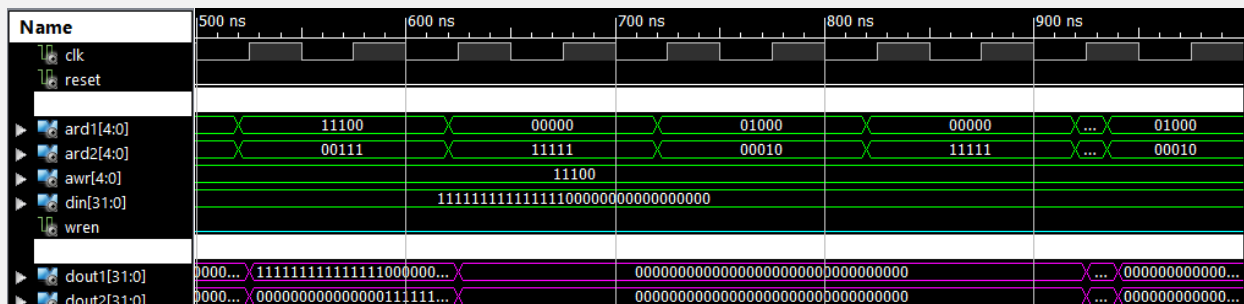
```
wait;
end process;
```

## 🕒 B.2. Προσομοίωση Συστήματος (Κυματομορφές)

0 ns – 500 ns:



500 ns – 900 ns:



Παρατηρούμε στις παραπάνω κυματομορφές ότι στη χρονική διάρκεια 300ns – 400ns γράφουμε σε δύο καταχωρητές του RF, αρχικά στον r7 και στη συνέχεια στον r28 δύο τιμές. Αργότερα, δίνοντας ως address οποιεσδήποτε άλλες τιμές εκτός από τις παραπάνω, οι έξοδοι Dout1, Dout2 μηδενίζονται. Αντίθετα οι άλλοι δίνουν ως έξοδο την τιμή που εκχωρήσαμε επιβεβαιώνοντας την ορθή λειτουργία του κυκλώματός μας.