Σχεδίαση Ψηφιακών Συστημάτων

Ακαδ. Έτος 2022-2023, Εαρινό Εξάμηνο

Εργασία 2 (20%)

→ Ημερομηνία παράδοσης**: Παρασκευή 9 Ιουνίου 2023, 11.55μμ** ←

Ανάλυση Εργασίας

Στόχος της 2ης Εργασίας είναι ο *ιεραρχικός* σχεδιασμός κυκλώματος που θα υλοποιεί μια Αριθμητική και Λογική Μονάδα (**Arithmetic Logic Unit – ALU**) στο Quartus.

Η ALU εκτελεί **αριθμητικές πράξεις** όπως πρόσθεση και αφαίρεση (σε σύστημα συμπληρώματος ως προς 2) και **λογικές πράξεις** όπως AND και OR. Στα πλαίσια της εργασίας θα πρέπει να υλοποιηθεί μια ALU η οποία θα μπορεί να εκτελεί τις ακόλουθες πράξεις σε σήματα των 16-bits:

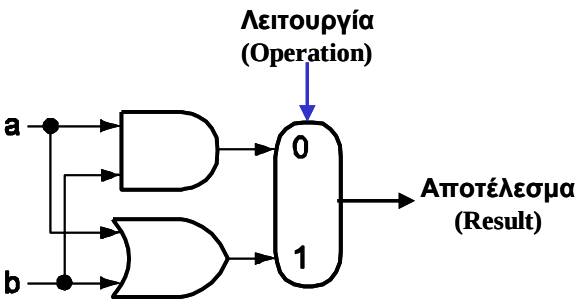
- Αριθμητικές πράξεις (με έλεγχο υπερχείλισης):
  - Πρόσθεση (ADD)
  - Αφαίρεση (SUB)
- Λογικές πράξεις:
  - Σύζευξη (AND)
  - Διάζευξη (OR)
  - Αποκλειστική διάζευξη (XOR)
  - Αντίθετο της διάζευξης (NOR)
  - Αντίθετο της σύζευξης (NAND)

Η σχεδίαση θα πρέπει να γίνει **ιεραρχικά**, με χρήση υποκυκλωμάτων (components). Στο *1ο μέρος* της εργασίας θα υλοποιηθεί μία **ALU για σήματα του 1-bit**. Στο *2ο μέρος* της εργασίας, η ALU του 1-bit θα χρησιμοποιηθεί ως component για την υλοποίηση της **ALU των 16-bits** που αποτελεί το τελικό ζητούμενο της εργασίας. Στη συνέχεια εξηγούνται αναλυτικά τα **βήματα** για τη σχεδίαση και υλοποίηση της ALU.

Μέρος 1ο – Σχεδίαση και υλοποίηση της 1-bit ALU

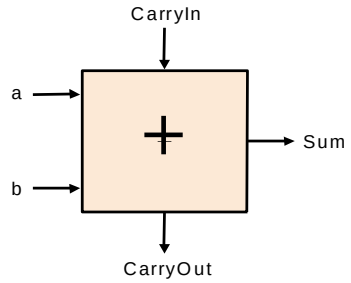
1. Υλοποίηση ενός κομματιού (slice) της 1-bit ALU για τις πράξεις AND, OR και πρόσθεση (ADD)

Λογικές πράξεις **AND** και **OR:** Η λογική μονάδα του 1-bit για το AND και το OR μοιάζει με αυτή της *εικόνας 1*. Ο πολυπλέκτης στα δεξιά επιλέγει το (a AND b) αν το bit ελέγχου Operation=0 ή το (a OR b) αν Operation=1, όπου a, b σήματα του 1 bit.



Εικόνα 1: ***Βασική λειτουργία AND και OR με πολυπλέκτη***

Αριθμητική πράξη **πρόσθεσης:** Ένας αθροιστής πρέπει να έχει 2 εισόδους 1-bit για τους τελεστέους και μια έξοδο 1-bit για το άθροισμα. Πρέπει να υπάρχει μια δεύτερη έξοδος για να μεταβιβάζεται το κρατούμενο, που ονομάζεται CarryOut (Έξοδος κρατουμένου). Προκειμένου ο αθροιστής να μπορεί να χρησιμοποιηθεί ως component για την άθροιση αριθμών περισσότερων bits, χρειάζεται να μπορεί να διαχειρίζεται και κρατούμενο ως είσοδο, οπότε πρέπει να συμπεριληφθεί και μια 3η είσοδος CarryIn (Είσοδος κρατουμένου). Η *εικόνα 2* δείχνει τον αθροιστή που περιγράψαμε.

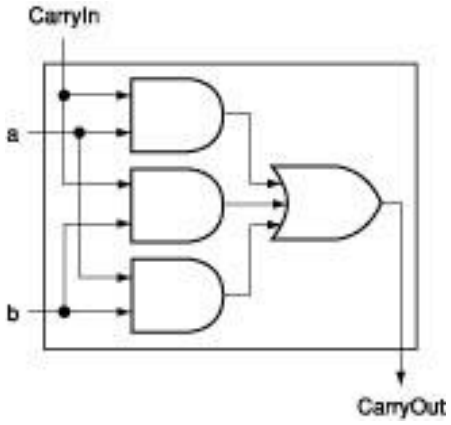


Αυτός ο αθροιστής ονομάζεται **Πλήρης Αθροιστής** (full adder). Ονομάζεται επίσης αθροιστής (3,2) επειδή έχει 3 εισόδους και 2 εξόδους.

Εικόνα 2: ***Πλήρης αθροιστής***

Οι συναρτήσεις εξόδου (για τα Sum και CarryOut) μπορούν να εκφραστούν ως λογικές εξισώσεις άρα και με λογικές πύλες. Όπως γνωρίζουμε, κατά την πράξη της άθροισης, προκύπτει κρατούμενο (**CarryOut**) όταν ισχύει η ακόλουθη λογική εξίσωση:

CarryOut = (b • CarryIn) + (a • CarryIn) + (a • b), την οποία μπορούμε να αναπαραστήσουμε ως εξής:

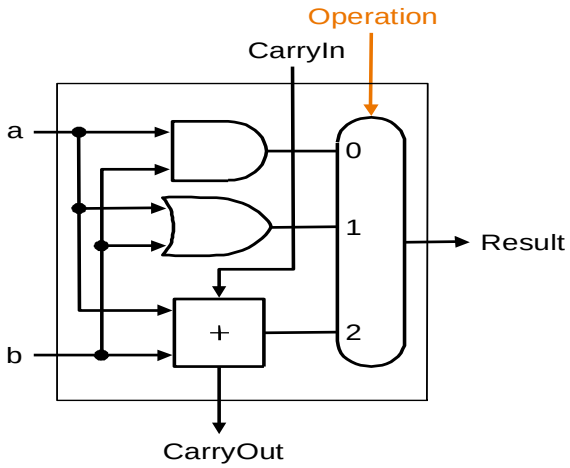


Εικόνα 3: ***Τμήμα του αθροιστή για την υλοποίηση του σήματος CarryOut.***

Η έξοδος **Sum** παίρνει την τιμή 1, όταν ακριβώς μία είσοδος έχει τιμή 1 ή όταν και οι τρεις είσοδοι έχουν τιμή 1. Δηλαδή:

$$Sum = \left(a \bullet \bar{b} \bullet \overline{\text{CarryIn}}\right) + \left(\bar{a} \bullet b \bullet \overline{\text{CarryIn}}\right) + \left(\bar{a} \bullet \bar{b} \bullet \text{CarryIn}\right) + \left(a \bullet b \bullet \text{CarryIn}\right)$$

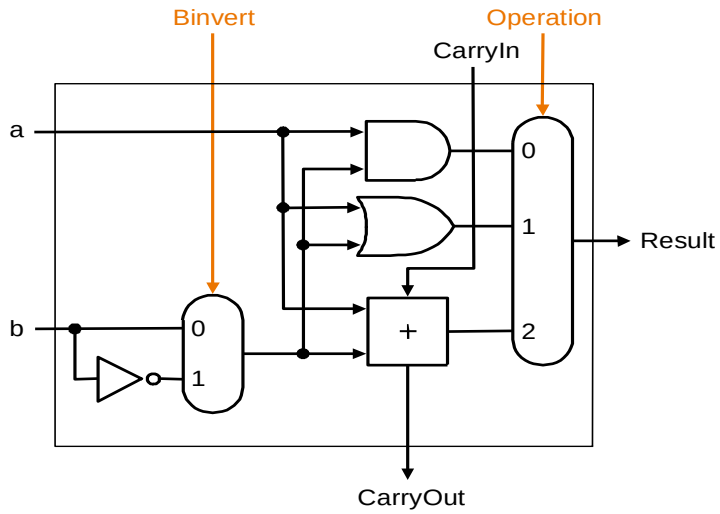
Το κομμάτι (slice) της 1-bit ALU που έχει δημιουργηθεί από το συνδυασμό των λειτουργιών του αθροιστή και των λογικών AND και OR φαίνεται στην *εικόνα 4*.



Εικόνα 4: ***Μια ALU του 1-bit που εκτελεί τις πράξεις AND (operation=0), OR (operation=1) και ADD (operation=2)***

2. Προσθέτοντας στην ALU μας την αφαίρεση (SUB) και τις λογικές πράξεις NOR και NAND

Αντί για **αφαίρεση**, προσθέτουμε το **συμπλήρωμα ως προς 2** του αφαιρετέου! Για να βρούμε το συμπλήρωμα ως προς 2 ενός αριθμού, αντιστρέφουμε κάθε bit (*συμπλήρωμα ως προς 1*) και μετά προσθέτουμε 1. Για να αντιστρέψουμε κάθε bit, προσθέτουμε ένα πολυπλέκτη 2-σε-1 που επιλέγει ανάμεσα σε *b* και $\bar{b}$ όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 5: ***Μια ALU του 1-bit που εκτελεί τις πράξεις AND, OR και πρόσθεση των a και b ή των a και*** $\bar{b}$ ***(αφαίρεση).***

Θέτοντας Binvert = 1 (δηλ. επιλέγοντας το $\bar{b}$) και CarryIn = 1, πραγματοποιείται **αφαίρεση**, προσθέτοντας το συμπλήρωμα ως προς 2 του b στο a, αλλιώς πραγματοποιείται πρόσθεση του b στο a.

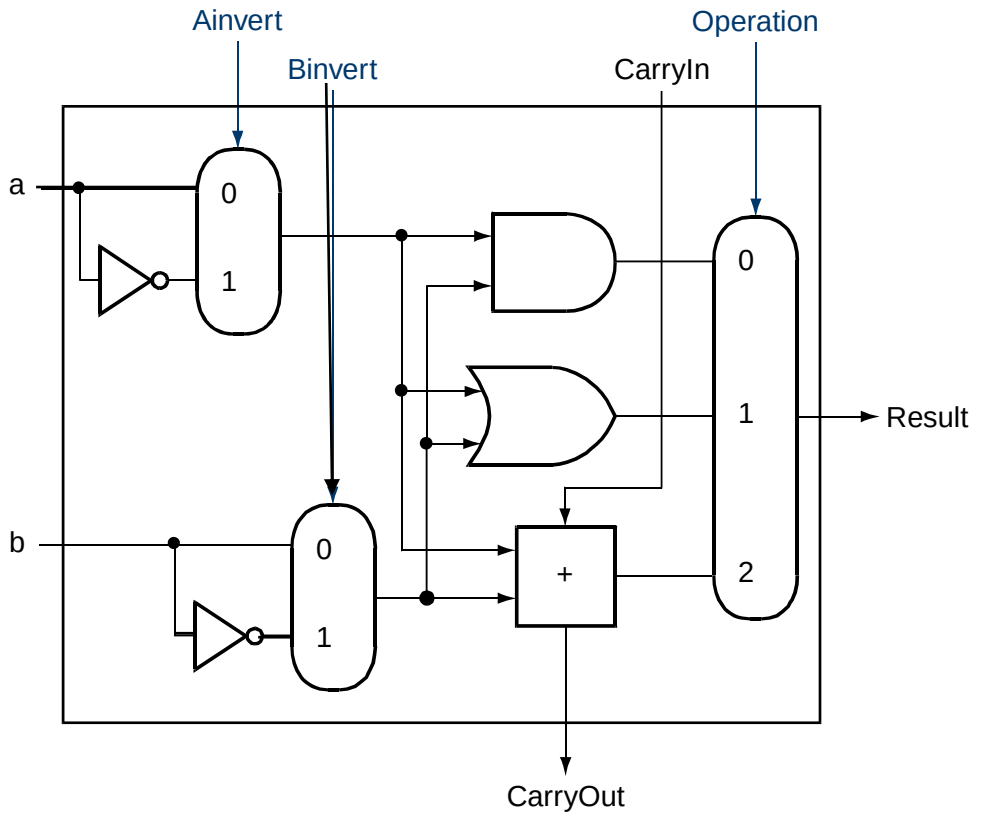
Λογική πράξη NOR: Σύμφωνα με το θεώρημα De Morgan $\overline{x+y} = \bar{x} \cdot \bar{y}$

Άρα αφού έχουμε το AND και το NOT b χρειάζεται να προσθέσουμε αντίστοιχα και το NOT a.

Επιλέγοντας **το** $\bar{a}$ **(Ainvert= 1) και το** $\bar{b}$ **(Binvert = 1)** ως εισόδους στην πύλη **AND**, παίρνουμε το **a NOR b.**

Για την υλοποίηση της πράξης NAND: Σύμφωνα με το θεώρημα De Morgan $\overline{x \cdot y} = \bar{x} + \bar{y}$

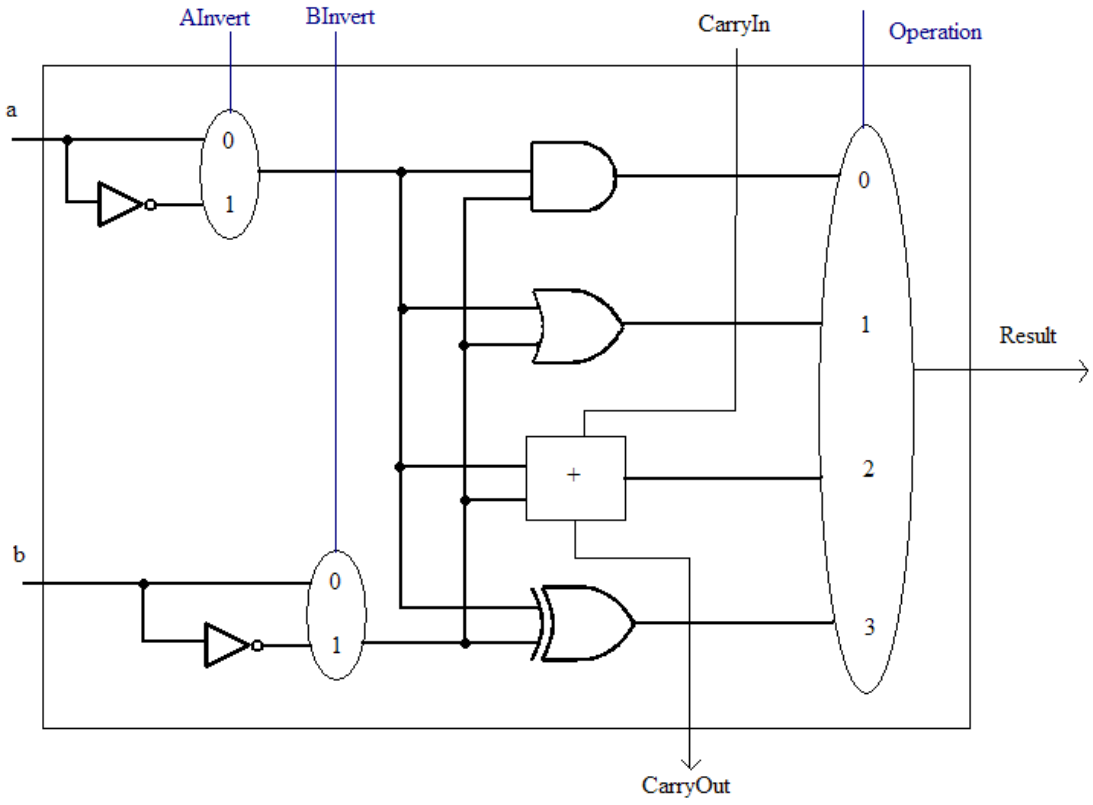
Επιλέγοντας το $\bar{a}$ (**Ainvert= 1**) και το $\bar{b}$ (**Binvert = 1**) ως εισόδους στην πύλη OR**,** παίρνουμε το **a NAND b**.



Εικόνα 6: ***Μια 1-bit ALU που εκτελεί τις πράξεις AND, OR, NAND, NOR και πρόσθεση/αφαίρεση.***

3. Προσθέτοντας στην ALU μας τη λογική πράξη XOR

Τέλος προσθέτουμε μια επιπλέον πύλη XOR στην 1-bit ALU μας και επεκτείνουμε τον πολυπλέκτη που παράγει το τελικό αποτέλεσμα για την υλοποίηση και της **πράξη XOR** (*Εικόνα 7*).



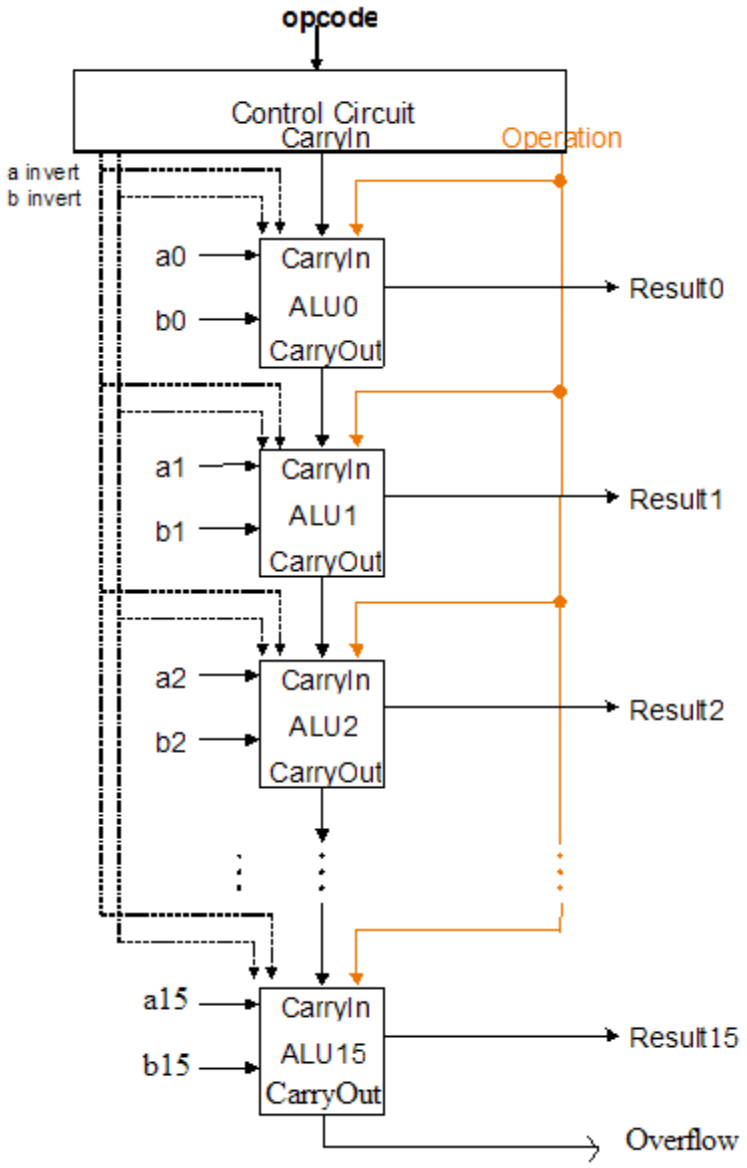
Εικόνα 7. ***Η τελική μορφή της 1-bit ALU που υλοποιεί όλες τις απαιτούμενες πράξεις.***

Μέρος 2ο - Υλοποίηση ιεραρχικής ALU των 16-bits με slices της 1-bit ALU

Στην **Εικόνα 8** φαίνεται μια ALU των 16-bits βασιζόμενη στη διαδοχική σύνδεση 16 slices της 1-bit ALU. Στην 16-bits ALU το CarryOut του κάθε slice γίνεται CarryIn για το επόμενο slice (π.χ. το CarryOut της ALU0 γίνεται CarryIn για την ALU1, το CarryOut της ALU1 γίνεται CarryIn για την ALU2, κ.ο.κ.), δηλαδή για την πράξη της άθροισης/αφαίρεσης υλοποιείται *αθροιστής διάδοσης κρατουμένου (ripple carry adder*).

Το αποτέλεσμα της εκτελούμενης κάθε φορά πράξης εξάγεται στο **Result0..Result15**. Πρέπει όμως κάπως η ALU να ειδοποιήσει το σύστημα για πιθανή **υπερχείλιση**. Υπερχείλιση συμβαίνει όταν το αποτέλεσμα της πράξης που εκτελεί η ALU είναι μεγαλύτερο από τους τελεστέους (στην περίπτωσή μας μεγαλύτερο από 16 bits). Το αν προκύπτει ή όχι υπερχείλιση μπορεί να ελεγχθεί βάση του τύπου: $\text{Overflow} = c_{n-1} \oplus c_n$

Προσοχή: ο έλεγχος υπερχείλισης έχει νόημα μόνο για τις αριθμητικές (όχι τις λογικές) πράξεις που εκτελεί η ALU. Δεν θα πρέπει να εμφανίζεται υπερχείλιση κατά την προσομοίωση εκτέλεσης των λογικών πράξεων στις κυματομορφές.



Εικόνα 8: ***Μία 16-bits ALU σε ιεραρχική σύνδεση με ripple carry που υποστηρίζει έλεγχο υπερχείλισης***

Προκειμένου να μπορεί να χρησιμοποιηθεί η 16-bits ALU, χρειαζόμαστε ένα κύκλωμα ελέγχου με το οποίο θα ορίζεται κάθε φορά ποια πράξη εκτελεί η ALU στα 16-bits σήματα εισόδου της.
Το **κύκλωμα ελέγχου (Control Circuit)** ορίζει τις πράξεις που θα γίνουν από την ALU με βάση τους *κωδικούς λειτουργίας (**opcode**)* που παρουσιάζονται στον *Πίνακα 1*. Ανάλογα με την πράξη που θέλουμε να εκτελεστεί

από την ALU (opcode), το Control Circuit δίνει τις τιμές στα Ainvert (1-bit), Binvert (1-bit), CarryIn (1-bit) και Operation (2-bits), τα οποία δίνονται ως είσοδοι στην ALU όπως φαίνεται στο σχήμα της εικόνας 8.

ΠΡΑΞΗ	OPCODE	Operation	Ainvert	Binvert	CarryIn
AND	**000**	**00**	**0**	**0**	**0**
OR	**001**	**01**	**0**	**0**	**0**
ADD	**010**	**10**	**0**	**0**	**0**
SUB	**011**	**10**	**0**	**1**	**1**
NOR	**100**	**00**	**1**	**1**	**0**
NAND	**101**	**01**	**1**	**1**	**0**
XOR	**110**	**11**	**0**	**0**	**0**
	111	***Δεν χρησιμοποιείται το opcode=111***			

Πίνακας 1: ***Η λειτουργία του Control Circuit***

Παρατήρηση: Τα Ainvert, Binvert, CarryIn και Operation αντιστοιχούν στα σήματα που φαίνονται στην **Εικόνα 7** για την 1-bit ALU.

Ζητούμενα

1. **Μέρος 1ο**
   i. Χρησιμοποιήστε το λογισμικό Quartus, για να γράψετε πηγαίο κώδικα (πρόγραμμα) στη γλώσσα VHDL, ο οποίος να υλοποιεί το κύκλωμα της **Εικόνας 7** (1 slice της 1-bit ALU) με **structural τρόπο χρησιμοποιώντας components**, με βάση την αναλυτική περιγραφή των συστατικών της ALU που σας δίνεται.
   Προσοχή: *σήματα εισόδου για την 1-bit ALU αποτελούν οι δύο 1-bit αριθμοί, τα Ainvert, Binvert, CarryIn και Operation, ενώ σήματα εξόδου το αποτέλεσμα της πράξης και το CarryOut.*
   ii. Χρησιμοποιήστε την προσομοίωση λειτουργίας (waveforms) του λογισμικού Quartus για να αποδείξετε την ορθότητα της υλοποίησής σας για όλες τις πιθανές τιμές των a, b (σήματα του 1-bit) και τις πράξεις που υποστηρίζει. Φροντίστε ώστε το αποτέλεσμα της λειτουργικής προσομοίωσης να είναι εμφανές για όλες τις πράξεις. Σας συστήνουμε να τρέξετε τη λειτουργική προσομοίωση περισσότερες φορές, ώστε να φαίνονται καθαρά οι κυματομορφές, μπορείτε για παράδειγμα να τρέξετε μία προσομοίωση για τις πράξεις AND και OR, άλλη για τις πράξεις ADD και SUB κι άλλη για τις πράξεις NAND, NOR και XOR.

2. **Μέρος 2ο**
   i. Χρησιμοποιήστε το λογισμικό Quartus, για να γράψετε πηγαίο κώδικα (πρόγραμμα) στη γλώσσα VHDL, ο οποίος να υλοποιεί το κύκλωμα της **Εικόνας 8** (16-bit ALU) με **structural** τρόπο, χρησιμοποιώντας το κύκλωμα που υλοποιήσατε για το ζητούμενο 1, δηλαδή την 1-bit ALU, ως component.
   Προσοχή: *σήματα εισόδου για την 16-bit ALU αποτελούν μόνο οι δύο 16-bit αριθμοί και το opcode, ενώ σήματα εξόδου το αποτέλεσμα της πράξης και το overflow.*
   ii. Χρησιμοποιήστε την προσομοίωση λειτουργίας (waveforms) του λογισμικού Quartus II για να αποδείξετε την ορθότητα της υλοποίησής σας για τις ακόλουθες περιπτώσεις (**χωριστή κυματομορφή ανά περίπτωση**):
      i. Πρόσθεση (ADD) και αφαίρεση (SUB) των a, b και των c, d (*οι αριθμοί δίνονται σε μορφή συμπληρώματος ως προς 2*) όπου:
      a = 0110110011001101 και b = 0010011100011100,
      c = 1001001100110011 και d = 0010011100011100
      ii. Λογικές πράξεις AND και OR των a, b όπου:
      a = 1001001100001010
      b = 0101010101010001
      iii. Λογικές πράξεις NAND, NOR και XOR των a, b όπου:
      a = 1011001100101101
      b = 1111010101010001

Οδηγίες

Θα παραδώσετε μέσω του eClass ένα αρχείο zip/rar, που θα ονομάσετε project2_AM1_AM2_AM3.rar (όπου AM: Αριθμοί Μητρώου όλων των μελών της ομάδας, π.χ. project2_3220400_3220401_3220402.rar), το οποίο θα περιλαμβάνει:

- για το Μέρος 1ο
  - **project** του Quartus για την 1-bit ALU
  - κυματομορφές (waveforms) που προκύπτουν από τη λειτουργική προσομοίωση του κυκλώματος που υλοποιήσατε για όλες τις πράξεις που εκτελεί η 1-bit ALU (ως screenshots στο PDF)
  - RTL διάγραμμα (RTL viewer) του κυκλώματος της 1-bit ALU (στο PDF)
- για το Μέρος 2ο
  - **project** του Quartus για την 16-bits ALU
  - κυματομορφές (waveforms) που προκύπτουν από τη λειτουργική προσομοίωση του κυκλώματος που υλοποιήσατε για όλες τις πράξεις που εκτελεί η 16-bits ALU (ως screenshots στο PDF)
  - RTL διάγραμμα (RTL viewer) του κυκλώματος της 16-bits ALU (στο PDF)
  - ένα **pdf αρχείο** το οποίο θα περιέχει τα στοιχεία (ονοματεπώνυμο, αριθμό μητρώου και email) των μελών της ομάδας, παρατηρήσεις και ό,τι άλλο θεωρείτε απαραίτητο σχετικά με την υλοποίηση των προβλημάτων. Το PDF θα πρέπει **κατ' ελάχιστον** να περιέχει τις κυματομορφές που αναφέρονται παραπάνω για τα δύο μέρη της εργασίας καθώς και τα αντίστοιχα RTL διαγράμματα των κυκλωμάτων.

Προσοχή! Ο κώδικας VHDL θα πρέπει να περιλαμβάνει και επαρκή επεξηγηματικά **σχόλια**.