Εργασία:

Διασύνδεση πληκτρολογίου με την πλακέτα DE2-115 και υλοποίηση ενός πολύ απλού calculator με πρόσθεση και αφαίρεση

1. Βασική ιδέα του κυκλώματος

Στην εργασία αυτή μου ζητήθηκε να υλοποιήσω έναν απλό calculator ο οποίος θα παίρνει σαν είσοδο από το πληκτρολόγιο μονοψήφιους αριθμούς και θα εκτελεί δύο πράξεις πρόσθεση και αφαίρεση. Για την άσκηση επέλεξα οι αριθμοί να αναγνωρίζονται από το βασικό πληκτρολόγιο και από το numeric keypad. Το κύκλωμα αρχικά περιμένει σαν είσοδο τον πρώτο αριθμό, στη συνέχεια το πρόσημο της πράξης, έπειτα τον δεύτερο αριθμό και τέλος το κουμπί της ισότητας, ώστε να εμφανιστεί στην πλακέτα το αποτέλεσμα.

2. Αναλυτική περιγραφή και τρόπος υλοποίησης του κυκλώματος

2.1. Είσοδοι – έξοδοι κυκλώματος

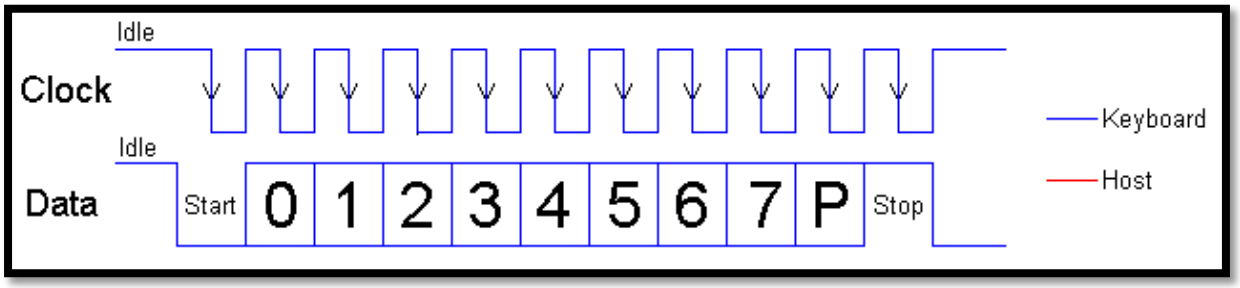
Για την υλοποίηση του κυκλώματος χρειαστήκαμε τρεις εισόδους (input=>sys_clk,reset,keyb_data), μία είσοδο-έξοδο (inout=>keyb_clk) και έξι εξόδους (output=>HEX7, HEX6,HEX5,HEX4,HEX3,HEX2). Το sys_clk είναι το ρολόι της πλακέτας, το reset είναι το SW[17] της πλακέτας, το keyb_data η γραμμή μεταφοράς των δεδομένων από το πληκτρολόγιο, το key_clk το ρολόι του πληκτρολογίου και

τα HEX7,…,HEX2 τα έξι πρώτα 7-segments για την αναπαράσταση της πράξης. Το παρακάτω πινακάκι μας δείχνει σε τι αντιστοιχεί το κάθε 7-segment.

7-SEGMENT	ΑΝΑΠΑΡΑΣΤΑΣΗ
HEX7	1ο τελούμενο πράξης
HEX6	τελεστής πράξης
HEX5	2ο τελούμενο πράξης
HEX4	σύμβολο ισότητας
HEX3	πρόσημο ή αριθμός αποτελέσματος
HEX2	αριθμός αποτελέσματος

2.2. Right shift registers, πολυπλέκτες 2 σε 1

Αρχικά μελέτησα διεξοδικά τα δύο site της εκφώνησης που ήταν στα αγγλικά, τα οποία εξηγούσαν τον τρόπο με τον οποίο το πληκτρολόγιο στέλνει πληροφορία στο host (στην περίπτωση μας είναι η πλακέτα). Από αυτή την μελέτη συμπέρανα ότι όταν πατήσουμε ένα κουμπί το πληκτρολόγιο στέλνει πολλαπλούς scan κωδικούς (τουλάχιστον 3, όπου ο καθένας έχει μήκος 8 bits), η δειγματοληψία του keyb_data πρέπει να γίνεται όταν ανιχνευθεί αρνητική ακμή του key_clk και ο προτελευταίος scan κωδικός είναι πάντα ο F0.



Χρησιμοποίησα δύο right shift registers τον keyb_clk_samples για την δειγματοληψία του ρολογιού του πληκτρολογίου και τον keyb_data_reg για την δειγματοληψία της γραμμής μεταφοράς των δεδομένων του πληκτρολογίου. Ο keyb_clk_samples έχει μέγεθος 6 bits, ώστε να εξαλείψω τυχόν θέματα θορύβου, ενώ ο keyb_data_reg έχει μέγεθος 11 bits, γιατί το πληκτρολόγιο εκτός από τον scan κωδικό μεγέθους 8 bits στέλνει ένα start bit που είναι πάντα 0, ένα parity bit και ένα stop bit, άρα για κάθε scan κωδικό στέλνει συνολικά 11 bits. Ο keyb_data_reg είναι απαραίτητο να αρχικοποιείται πάντα με 1, επειδή το πρώτο bit του scan κωδικού είναι 0, ώστε να ξέρω πότε έχει γεμίσει και πότε όχι. Οι δύο αυτοί καταχωρητές λειτουργούν στις θετικές ακμές του ρολογιού της πλακέτας ή στις θετικές ακμές του reset. Ο keyb_clk_samples σε κάθε θετική ακμή του ρολογιού της πλακέτας δειγματοληπτεί τη γραμμή μεταφοράς του ρολογιού του πληκτρολογίου κάνοντας shift κάθε φορά μια θέση προς τα δεξιά. Επομένως τα πιο πρόσφατα δείγματα βρίσκονται στις σημαντικότερες θέσεις του καταχωρητή. Άρα αν το keyb_clk_samples=000111, έχω εντοπίσει μια αρνητική ακμή του ρολογιού του πληκτρολογίου, οπότε ο keyb_data_reg δειγματοληπτεί τη γραμμή μεταφοράς keyb_data κάνοντας επίσης κάθε φορά shift μια θέση προς τα δεξιά. Αν ο καταχωρητής keyb_data_reg έχει γεμίσει, δηλαδή το λιγότερο σημαντικό ψηφίο του είναι 0 (αφού το πρώτο ψηφίο που στέλνει το πληκτρολόγιο είναι πάντα 0 και τον είχαμε αρχικοποιήσει όλο με 1), τότε κρατάμε τα 8 από τα 11 bit, δηλαδή τον

καθαρό scan κωδικό στον καταχωρητή keyb_data_8bit, ώστε να τα πάρει σαν είσοδο ο πρώτος αποκωδικοποιητής, να μας βγάλει σαν έξοδο την τιμή του κουμπιού που πατήθηκε στο δυαδικό και να την αποθηκεύσει στον καταχωρητή c μεγέθους 6 bits. Εδώ να σημειώσω πως ο πρώτος αποκωδικοποιητής μπορεί να αποκωδικοποιήσει μόνο αριθμούς. Σε οποιαδήποτε άλλη περίπτωση, δηλαδή αν πατηθεί οποιοδήποτε άλλο κουμπί που δεν αντιστοιχεί σε αριθμό η έξοδος του αποκωδικοποιητή θα είναι c=100000. Επειδή κανένας αριθμός δεν έχει σημαντικότερο ψηφίο στον καταχωρητή c μονάδα, ελέγχοντας μόνο το c[5], ξέρουμε αν έχει πατηθεί αριθμός ή όχι. Ο πρώτος αποκωδικοποιητής περιέχει όλους τους scan κωδικούς των αριθμών (0-9) από το βασικό πληκτρολόγιο και από το numeric keypad. Όταν κρατήσουμε τα 8 bits του scan κωδικού στον καταχωρητή keyb_data_8bit, αρχικοποιούμε ξανά τον καταχωρητή keyb_data_reg με 1, ώστε να κάνουμε λήψη του επόμενου scan κωδικού. Επίσης, όσο ο καταχωρητής keyb_data_reg είναι γεμάτος το keyb_clk=0, ώστε να μην μπορούμε να ανιχνεύσουμε αρνητική ακμή και άρα να μην μπορεί το πληκτρολόγιο να μας στείλει δεδομένα, ενώ όταν ο καταχωρητής αρχικοποιηθεί ξανά με 1 τότε το keyb_clk συνεχίζει να παίρνει τιμές από το πληκτρολόγιο. Σημαντικό ρόλο στο κύκλωμα παίζουν οι παρακάτω δύο πολυπλέκτες 2 σε 1.

```
assign mux=(next_calc_state ==WAIT_EQ)?result:c;
assign result=op?a+(~b)+1'b1:a+b;
```

Ο πρώτος πολυπλέκτης κανονίζει τι θα στείλω στον δεύτερο αποκωδικοποιητή σαν είσοδο. Έτσι αν η επόμενη κατάσταση του fsm είναι WAIT_EQ, δηλαδή έχω δώσει τα δύο τελούμενα και τον τελεστή στέλνω στον αποκωδικοποιητή το result, αλλιώς στέλνω το c. Ο δεύτερος πολυπλέκτης κανονίζει τι πράξη θα εκτελεστεί ανάλογα με την τιμή του καταχωρητή op. Έτσι αν op=1 εκτελείται αφαίρεση, αλλιώς αν op=0 εκτελείται πρόσθεση. Να σημειώσω πως η δουλειά του δεύτερου αποκωδικοποιητή είναι να αποκωδικοποιεί όλα τα δυνατά αποτελέσματα των δύο πράξεων. Άρα αποκωδικοποιεί όλους τους αριθμούς στο διάστημα [-9,18]. Η έξοδος του είναι δύο καταχωρητές disp1, disp2 των 7 bits όπου ο κάθε καταχωρητής ανάλογα την κατάσταση του fsm αντιστοιχίζεται σε ένα διαφορετικό 7-segment. Ο καταχωρητής disp1 χρειάζεται μόνο στην αναπαράσταση του αποτελέσματος. Άρα το αντίστοιχο 7-segment θα είναι είτε σβηστό αν το αποτέλεσμα είναι μονοψήφιο θετικό, είτε το πρόσημο μείον αν το αποτέλεσμα είναι μονοψήφιο αρνητικό, είτε αριθμός αν το αποτέλεσμα είναι διψήφιο. Ο καταχωρητής disp2 ανάλογα αν την κατάσταση του fsm αναπαριστά ή τελούμενο ή μέρος του αποτελέσματος.

Μέχρι στιγμής έχω αναλύσει τον καταχωρητή δειγματοληψίας του ρολογιού του πληκτρολογίου, τον καταχωρητή δειγματοληψίας της γραμμής μεταφοράς των δεδομένων του πληκτρολογίου, τον πρώτο αποκωδικοποιητή

που επιστρέφει την δυαδική τιμή των τελούμενων της πράξης στο διάστημα [0,9] και τον δεύτερο αποκωδικοποιητή ο οποίος επιστρέφει τις τιμές των 7-segments για τους αριθμούς στο διάστημα [-9,18]. Οτιδήποτε άλλο δώσουμε σαν είσοδο που δεν είναι μέσα στο διάστημα αυτό η έξοδος για τα disp1, disp2 θα είναι το e για το αντίστοιχο 7-segment.

2.3. Μηχανή πεπερασμένων καταστάσεων (fsm)

Το τελευταίο μέρος του κυκλώματος που συνδέει όλα τα παραπάνω είναι η μηχανή πεπερασμένων καταστάσεων (finite state machine - fsm). Σύμφωνα με την εκφώνηση και την λογική λειτουργίας του calculator έχουμε 4 κύριες και 2 δευτερεύουσες καταστάσεις λειτουργίας του fsm. Αυτές είναι WAIT_OPERAND1, WAIT_OPERATOR, WAIT_OPERAND2, WAIT_EQ, WAIT_F0, AFTER_F0. Για να περάσω από οποιαδήποτε κύρια κατάσταση σε μία άλλη πρέπει να περάσω πρώτα από τις καταστάσεις WAIT_F0 και AFTER_F0. Το γιατί πρέπει να γίνει αυτό θα το εξηγήσω παρακάτω. Το fsm εκτελείται σε κάθε θετική ακμή του ρολογιού της πλακέτας και όταν έχω λάβει ένα scan κωδικό. Στο reset του fsm αρχικοποιώ όλα τα 7-segment να είναι σβηστά, καθώς και την επόμενη κατάσταση του next_calc_state να είναι η WAIT_OPERAND1.

2.3.1. Κατάσταση WAIT_OPERAND1

Την επόμενη φορά που θα λάβω ένα scan κωδικό αφού η τιμή του next_calc_state είναι η WAIT_OPERAND1 κατάσταση, θα σβήσω όλα τα 7-segment της πλακέτας εκτός από το HEX7 επειδή σε αυτό το segment θέλω να δείξω τον αριθμό που πατήθηκε μόλις ή το e αν έχει πατηθεί οτιδήποτε άλλο

εκτός από αριθμό. Για να διαπιστώσουμε αν έχει πατηθεί αριθμός ή όχι αρκεί να δούμε αν το c[5] είναι διάφορο του 1 και αν ισχύει τότε έχει πατηθεί αριθμός. Αν έχει πατηθεί αριθμός κρατάω σε έναν καταχωρητή a την έξοδο του πρώτου αποκωδικοποιητή c και ορίζω σαν next_calc_state, δηλαδή επόμενη κατάσταση του fsm την WAIT_F0. Είτε έχει πατηθεί αριθμός, είτε όχι δείχνουμε στο HEX7 την έξοδο του αποκωδικοποιητή disp2, όπου στην περίπτωση που είναι αριθμός θα ανάψει ο αριθμός, αλλιώς θα ανάψει το e. Επίσης κρατάμε την κύρια κατάσταση του fsm στον καταχωρητή state, η οποία είναι η WAIT_OPERAND1 και θα μας χρειαστεί στην κατάσταση AFTER_F0. Η τιμή του καταχωρητή state θα μας ορίσει την τιμή του καταχωρητή next_calc_state. Συμπέρασμα πως αν πατηθεί οτιδήποτε εκτός από αριθμό θα ανάψει το e στο HEX7 και το fsm θα κολλήσει στην κατάσταση WAIT_OPERAND1, μέχρι να δώσουμε έναν αριθμό.

2.3.2. Κατάσταση WAIT_OPERATOR

Στην κατάσταση αυτή περιμένουμε από το πληκτρολόγιο τον τελεστή πρόσθεσης ή τον τελεστή αφαίρεσης. Αν πατηθεί κάποιος από τους δύο τελεστές δείχνουμε στο HEX6 της πλακέτας το αντίστοιχο σύμβολο πράξης, εκχωρούμε στον καταχωρητή op το 0 αν πρόκειται για πρόσθεση ή το 1 αν πρόκειται για αφαίρεση και στον καταχωρητή next_calc_state την κατάσταση WAIT_F0. Επίσης κρατάμε την κύρια κατάσταση του fsm στον καταχωρητή state οποία είναι η WAIT_OPERATOR, όπου θα μας χρειαστεί πάλι στην κατάσταση AFTER_F0. Αν πατηθεί οτιδήποτε άλλο

εκτός από (+) ή (–) το fsm δείχνει το e στο αντίστοιχο 7-segment της πλακέτας και κολλάει στην κατάσταση WAIT_OPERATOR, μέχρι να πατήσουμε ένα από τα δύο πλήκτρα. Οι scan κωδικοί πρόσθεσης και αφαίρεσης ελέγχονται μέσα στο fsm.

2.3.3. Κατάσταση WAIT_OPERAND2

Όπως και στην κατάσταση WAIT_OPERAND1 που περιμένουμε το πρώτο τελούμενο της πράξης, στην κατάσταση αυτή περιμένουμε το δεύτερο τελούμενο της πράξης. Οι ενέργειες είναι αντίστοιχες στις δύο καταστάσεις με μόνη διαφορά πως εδώ δεν σβήνουμε τα υπόλοιπα 7-segment της πλακέτας και την έξοδο του πρώτου αποκωδικοποιητή την αποθηκεύουμε σε έναν ξεχωριστό καταχωρητή b.

2.3.4. Κατάσταση WAIT_EQ

Η κατάσταση αυτή αντιστοιχεί στην περίπτωση που πατήσουμε το πλήκτρο (=), ώστε να εμφανιστεί το αποτέλεσμα στα αντίστοιχα 7-segment. Αν λοιπόν πατηθεί το (=), τότε στο HEX4 δείχνουμε το (=), στο HEX3 την τιμή του disp1 (σβηστό ή αριθμός ή το αρνητικό πρόσημο) και στο HEX2 την τιμή του disp2 (αριθμός). Η επόμενη κατάσταση του fsm θα είναι η WAIT_F0. Αν δεν πατηθεί το (=), ανάβει το e. Τέλος κρατάμε την κύρια κατάσταση του fsm, δηλαδή την WAIT_EQ.

2.3.5. Κατάσταση WAIT_F0

Όπως είπα παραπάνω πατώντας ένα πλήκτρο στέλνονται πολλαπλοί scan κωδικοί. Ο προτελευταίος είναι πάντα ο F0. Ο κωδικός αυτός στέλνεται όταν αφήσουμε το πλήκτρο ακολουθούμενος από έναν ακόμα κωδικό ο οποίος τις πιο πολλές φορές είναι ο scan κωδικός του κουμπιού. Αυτή η κατάσταση είναι πολύ σημαντική, γιατί αν εμείς πατάμε για πολύ ώρα το κουμπί δεν μπορούμε να αλλάξουμε κατάσταση αν δεν ξέρουμε πως το έχουμε αφήσει. Αν αλλάξουμε κατάσταση πριν αφήσουμε το κουμπί, τότε και στην επόμενη κατάσταση θα θεωρηθεί πως πατάμε το ίδιο με πριν κουμπί, με αποτέλεσμα να μην λειτουργεί σωστά ο calculator, εμφανίζοντας το e και στο επόμενο 7-segment. Επομένως η κατάσταση αυτή ελέγχει πότε ο scan κωδικός είναι ο F0 και μόνο τότε προχωράμε στην επόμενη κατάσταση που είναι η κατάσταση AFTER_F0.

2.3.6. Κατάσταση AFTER_F0

Είναι η πιο σημαντική κατάσταση στο fsm, επειδή σε αυτή αποφασίζεται ποια θα είναι η επόμενη κύρια κατάσταση. Έτσι ανάλογα με την τρέχουσα κύρια κατάσταση state, θα αποφασίσω ποια θα είναι η επόμενη κατάσταση next_calc_state του fsm. Η αλληλουχία των καταχωρητών state και next_calc_state φαίνεται στο παρακάτω πινακάκι.

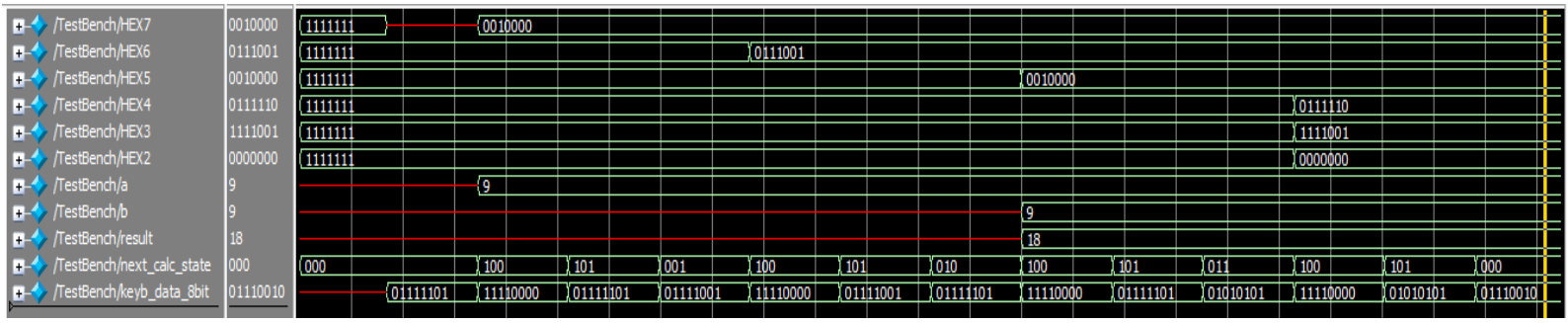
state	**next_calc_state**
WAIT_OPERAND1	WAIT_OPERATOR
WAIT_OPERATOR	WAIT_OPERAND2
WAIT_OPERAND2	WAIT_EQ
WAIT_EQ	WAIT_OPERAND1

Στην τελευταία περίπτωση όπου το state=WAIT_EQ, το next_calc_state=WAIT_OPERAND1, ώστε το fsm να είναι έτοιμο να εκτελέσει την επόμενη πράξη.

3. Εξομοίωση στο Modelsim

Εξομοίωσα στο modelsim το κύκλωμα εκτελώντας μια πρόσθεση και μία αφαίρεση στο ίδιο testbench. Αρχικά εκτελείται η πρόσθεση και στην συνέχεια η αφαίρεση. Η πρόσθεση που εκτελέστηκε είναι 9+9 και η αφαίρεση 2-9. Παρακάτω φαίνονται screenshots από την εξομοίωση των δύο πράξεων.

Πρόσθεση 9+9



Αφαίρεση 2-9