Καρατζάς Δημήτρης **icsd13072**
Κατσιώπης Νίκος **icsd13076**

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ ΓΙΑ ΤΟ ΜΑΘΗΜΑ ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΣΥΝΟΨΗ

Για την εργασία αυτή, είχαμε να κάνουμε το κύκλωμα ενός απλού calculator το οποίο να κάνει είτε τη πράξη της αφαίρεσης είτε τη πράξη της πρόσθεσης μεταξύ δύο αριθμών από 0 έως 9.

Έχουμε φτιάξει ένα test bench που δίνει διαδοχικά κάποιο bit του scan code ενός πλήκτρου (πχ keypad4) σε κάθε αρνητική ακμή του keyboard clock. Το scan code το δίνει από το λιγότερο σημαντικό ψηφίο πρώτα (όπως ένα αληθινό πληκτρολόγιο) οπότε έχουμε έναν data shift register (δηλωμένος ως **keyb_data_sreg, 11bit**) που λαμβάνει το 11bit πακέτο (bit-by-bit).

Όταν ο clock shift register (δηλωμένος ως **keyb_clock_sreg, 6bit**) είναι πλέον **000111** τότε ο data shift register θα κάνει ένα bit shift προς τα δεξιά, και θα μπει ως πιο σημαντικό ψηφίο αυτό που έλαβε από πληκτρολόγιο.

Άρα (όταν το δεξιότερο bit γίνει 0) θα έχει λάβει το scan code, οπότε τώρα με μια απλή αναζήτηση των σωστών τιμών για τα scan tables μπορεί να γίνει το **decoding σε δυαδική μορφή**. Μόλις σταλεί το scan code, το πληκτρολόγιο θα στείλει στη συνέχεια το scan code **F0** οπότε το ελέγχουμε και αυτό, καθώς επίσης και το ίδιο το scan code που έστειλε πριν (άρα πχ αν πατηθεί το keypad4 θα σταλεί από το keyboard το 11bit πακέτο που εμπεριέχει το scan code του keypad4 , το 11bit πακέτο που περιέχει το scan code του F0 και μετά πάλι το ίδιο 11bit που εμπεριέχει το scan code του keypad4).

Ο **δεύτερος decoder** θα δώσει στους registers που κρατάνε τα bits “on/off” για το άναμμα ή όχι του αντίστοιχου led. Οπότε αν πχ το decoded (από τον πρώτο decoder) value είναι το 4 (άρα 6’b000100) ο δεύτερος decoder θα δώσει κατάλληλη τιμή στους register “ανάμματος” τέτοια ώστε να ανάψει το “4” σε κάποιο από τα 7-segment display. Επίσης ο δεύτερος decoder δεν ελέγχει μόνο τη τιμή που δόθηκε από το πληκτρολόγιο αλλά και το αποτέλεσμα της πράξης, οπότε οδηγείται από έναν **πολυπλέκτη** ο οποίος ελέγχει αν είμαστε στην κατάσταση αναμονής του “**=**”, και αν ναι τότε θα δώσει ως έξοδο το αποτέλεσμα της πράξης, αλλιώς θα δώσει κανονικά τη τιμή του πρώτου decoder. Με αυτό τον τρόπο ο δεύτερος decoder μπορεί να δώσει κατάλληλες τιμές για οποιοδήποτε αριθμό στο [-9,18].

Για να ξέρουμε όμως ποιο display θα πάρει τον αριθμό χρειαζόμαστε καταστάσεις οπότε έχουμε φτιάξει και το **FSM** που χρειάζεται ώστε να κάνουμε τις απαραίτητες ενέργειες ανάλογα τη κατάσταση στην οποία βρισκόμαστε.
Κάθε κατάσταση του FSM κάνει κάποιο κομμάτι του calculator, δηλαδή η πρώτη κατάσταση περιμένει τον πρώτο αριθμό και ανάβει το πρώτο led, η δεύτερη περιμένει “+” ή “-” και ανάβει το 2ο led, η τρίτη περιμένει τον 2ο αριθμό και ανάβει το 3ο led, η τέταρτη περιμένει το “=” και ανάβει το 4ο led και υπολογίζει το αποτέλεσμα και το δείχνει στα δύο τελευταία leds.

ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ

Για την υλοποίηση του calculator, θα πρέπει πρώτα να υλοποιήσουμε το κύκλωμα το οποίο αναγνωρίζει ένα πάτημα κουμπιού.
Το κύκλωμα έχει ένα εσωτερικό clock το οποίο είναι πολύ πιο γρήγορο από του πληκτρολογίου. Για το λόγο αυτό, ο keyb_clk_sreg που διαβάζει τη τιμή του clock του πληκτρολογίου (σε κάθε κύκλο ρολογιού του κυκλώματος) για να ξέρουμε πως όντως έχει έρθει αρνητική ακμή ρολογιού από το keyboard, κρατάει τις 6 τελευταίες τιμές.
Άρα τη περισσότερη ώρα θα έχει τιμή “111111” και μόλις έρθει η πρώτη αρνητική ακμή θα πάρει τη τιμή “011111” και για να είμαστε σίγουροι πως αυτή η αρνητική ακμή δεν προέκυψε από θόρυβο, λαμβάνουμε και τη τιμή του clock του πληκτρολογίου άλλες 2 φορές, οπότε αν έρθει πάλι “0” ο shift register θα πάρει τη τιμή “000111”.

Τώρα που έχει αυτή τη συγκεκριμένη τιμή ο keyb_clk_sreg, θα γίνει shift στον keyb_data_sreg και θα δεχτεί ένα bit από το πληκτρολόγιο. Αυτός ο καταχωρητής είναι ένας 11bit και αρχικοποιείται στο “11111111111”. Το πληκτρολόγιο στέλνει πακέτα 11bits τα οποία περιέχουν και το 8bit scan code. Πάντα όμως στέλνει το 0 ως πρώτο bit (Start bit).
Το shifting γίνεται ως εξής:
keyb_data_sreg <= {keyb_data, keyb_data_sreg[10:1]} ,
όπου **keyb_data** το bit δεδομένων που δέχεται από το πληκτρολόγιο.

Οπότε ξέρουμε ότι έχουμε λάβει ένα πλήρες πακέτο από το πληκτρολόγιο όταν ο keyb_data_sreg έχει τιμή 0 στο τελευταίο bit του. Όταν γίνει αυτό τον κάνουμε επαναφορά πάλι στο “11111111111” ώστε να ξεκινήσει πάλι η διαδικασία του shifting με σκοπό να λάβει το επόμενο πακέτο. Ο register αυτός δίνει την έξοδο του σε ένα συνδυαστικό κύκλωμα αποκωδικοποίησης (**keyb_data_sreg_dec**) και αυτό το κύκλωμα ελέγχει το scan code (8bit) από όλο το πακέτο και βγάζει ως έξοδο την δυαδική του μορφή (6bit).

Όλη η διαδικασία αυτή έχει ως σκοπό να λαμβάνουμε ένα πακέτο από το πληκτρολόγιο. Όμως εφόσον λάβουμε ένα πακέτο και το αποκωδικοποιήσουμε πρέπει να το χρησιμοποιήσουμε. Εδώ χρειαζόμαστε το FSM.

Το FSM έχει τις καταστάσεις
- WAIT_OPERAND1
- WAIT_OPERATOR
- WAIT_OPERAND2
- WAIT_EQ
- WAIT_F0
- AFTER_F0

Καταρχήν, το FSM λειτουργεί **μόνο** όταν έχουμε λάβει ένα πλήρες πακέτο από το πληκτρολόγιο, δηλαδή όταν ο καταχωρητής δεδομένων (keyb_data_sreg) έχει ως τελευταίο bit το 0, αλλιώς δεν έχει νόημα το FSM (δε θα δουλεύει όπως θέλουμε).

Χρειαζόμαστε τους εξής καταχωρητές για το FSM:
- **op1, op2** καταχωρητές 6bit που κρατάνε την δυαδική τιμή των δύο αριθμών.
- **error** 1bit που ελέγχει αν έχει δοθεί μη επιτρεπτή τιμή (πχ αριθμός αντί για “+”).
- **sub** 1bit που δηλώνει αν έχουμε αφαίρεση (τιμή 1) ή πρόσθεση (τιμή 0).
- **state, current_state** για να κρατάνε τις καταστάσεις. Η κατάσταση που ελέγχεται κάθε φορά είναι στον state, ο current_state έχει άλλο σκοπό που θα εξηγηθεί στη συνέχεια (state → κατάσταση FSM, current_state → κατάσταση calculator).

Επιπλέον χρειαζόμαστε και δύο καλώδια:

- **result** 6bit το οποίο είναι είτε η πρόσθεση είτε η αφαίρεση των δύο τελουμένων. Άν το αποτέλεσμα είναι αρνητικό τότε το result θα αναπαριστά τον αρνητικό αριθμό ως συμπλήρωμα ως προς 2.
- **mux_out** 6bit, είναι η έξοδος ενός πολυπλέκτη. Η έξοδος του πολυπλέκτη αυτού είναι η είσοδος στον δεύτερο decoder. Συγκεκριμένα, ελέγχει αν είμαστε στην κατάσταση WAIT_EQ, αν είμαστε τότε θέλουμε το αποτέλεσμα της πράξης, αλλιώς θέλουμε την decoded τιμή από το πληκτρολόγιο (του πρώτου decoder).

Αρχικά βρισκόμαστε στην κατάσταση **WAIT_OPERAND1**.
Μόλις λάβουμε το πρώτο πακέτο από το πληκτρολόγιο ελέγχουμε την decoded τιμή.
Άν αυτή αντιστοιχεί σε αριθμό τότε θέτουμε στον καταχωρητή **op1** την τιμή αυτή και δίνουμε στο **hex7 (1ο led)** τη τιμή που μας δίνει ο 2ος decoder, αλλιώς θέτουμε **error** register ίσο με 1 και δίνουμε τιμή στο hex7 τέτοια ώστε να ανάψει το "e".
Τώρα πρέπει να λάβουμε το F0 καθώς και το scan code (το ίδιο πάλι, αλλά δεν ελέγχεται).
Θέτουμε **state <= WAIT_F0** και μόλις λάβουμε το scan code του F0 (άρα πάλι γίνεται shifting των shift registers και αναμονή μέχρι να ληφθεί όλο το πακέτο) τότε θέτουμε **state <= AFTER_F0.** Στη κατάσταση αυτή ελέγχουμε τώρα αν υπάρχει κάποιο σφάλμα, δηλαδή αν **error** είναι ίσο με 1.
Αν είναι τότε η κατάσταση του calculator θα πρέπει να είναι η ίδια με την προηγούμενη κατάσταση του, μια εκ των WAIT_OPERAND1, WAIT_OPERATOR, WAIT_OPERAND2 ή WAIT_EQ. Για αυτό το λόγο, επειδή χρειάζεται να ξέρουμε ποια ήταν η προηγούμενη κατάσταση, θέτουμε τον **current_state <= WAIT_OPERAND1** ώστε να μπορούμε να ξέρουμε που να "γυρίσουμε" αν υπάρξει σφάλμα **πριν** φύγουμε από τη κατάσταση αυτή.
Στη κατάσταση AFTER_F0 αν εντοπίσουμε σφάλμα επιστρέφουμε στη προηγούμενη κατάσταση → **state <= current_state.**
Άν δεν υπάρχει σφάλμα τότε ελέγχουμε την current_state και απλώς προχωράμε στην επόμενη κατάσταση, δηλαδή αν:

- current_state == WAIT_OPERAND1 → state <= WAIT_OPERATOR
- current_state == WAIT_OPERATOR → state <= WAIT_OPERAND2
- current_state == WAIT_OPERAND2 → state <= WAIT_EQ
- current_state == WAIT_EQ → state <= WAIT_OPERAND1

Με αυτόν τον τρόπο προχωράμε σε επόμενες καταστάσεις. Οι υπόλοιπες καταστάσεις κάνουν πάνω-κάτω τις ίδιες διαδικασίες, δηλαδή

WAIT_OPERATOR:
Έλεγχος decoded τιμής, αν είναι "-" ή "+" (ο decoder1 αναγνωρίζει και τα "+", "-" και "=" αν και θα μπορούσαμε απευθείας μέσω του keyb_data_sreg να ελέγξουμε) τότε θέτουμε στον **hex6** (2ο led) τη τιμή του δεύτερου decoder αλλιώς "e" και **error** ίσο με 1.
Επίσης αν είναι "-" τότε θέτουμε **sub** ίσο με 1, αλλιώς 0.
Θέτουμε **current_state <= WAIT_OPERATOR** και **state <= WAIT_F0.**
Στη συνέχεια γίνεται ακριβώς η ίδια διαδικασία με πριν, state διαβάζει το F0, πάει στο AFTER_F0 κτλ.

WAIT_OPERAND2:
Ακριβώς ίδια διαδικασία με WAIT_OPERAND1, απλώς δίνει τιμή στον **op2** αντί για τον op1, και ανάβει το **hex5 αντί** για το hex7. Θέτουμε **current_state <= WAIT_OPERAND2** και **state <= WAIT_F0.**

WAIT_EQ:
Ελέγχεται πάλι η decoded τιμή, αν βρεθεί ότι είναι το "=" τότε δίνονται στα **hex1** και **hex0** οι τιμές του δεύτερου αποκωδικοποιητή, αλλιώς **error** ίσο με 1 και ανάβει μόνο το **hex1** να δείξει το "e". Θέτουμε **current_state <= WAIT_EQ** και **state <= WAIT_F0.**
.
Τελικά, με αυτά τα κυκλώματα έχουμε καταφέρει να υλοποιήσουμε τον απλό calculator της άσκησης. Τα **scan codes** που στέλνει το πληκτρολόγιο και χρησιμοποιήσαμε είναι τα παρακάτω (χωρίς το "/", "*" και "**.**"):

6C/F0 6C	Keypad 7	70/F0 70	Keypad 0
6B/F0 6B	Keypad 4	7E/F0 7E	Keypad *
69/F0 69	Keypad 1	7D/F0 7D	Keypad 9
77/F0 77	Keypad /	74/F0 74	Keypad 6
75/F0 75	Keypad 8	7A/F0 7A	Keypad 3
73/F0 73	Keypad 5	71/F0 71	Keypad .
72/F0 72	Keypad 2	84/F0 84	Keypad -
55/F0 55	=	7C/F0 7C	Keypad +

Αυτό που ελέγχουμε στον πρώτο decoder, είναι τα 8bit (scan code) του πακέτου, δηλαδή το keyb_data_sreg[8:1] στην ουσία, εφόσον γίνεται shifting δεξιά κάθε φορά θα έχει πάντα 0 στη θέση 0 καθώς και τα δύο πρώτα (end bit → 1 και parity bit) δε μας απασχολούν.
Εφόσον το keyboard στέλνει το scan code από το πιο **ασήμαντο** ψηφίο πρώτα, δηλαδή πχ για το Keypad0 θα στείλει την binary μορφή του 70, την 01110000, όμως την στέλνει ανάποδα, θα στείλει εν τέλει το **{0, 00001110,p1}** οπότε ο keyb_data_sreg όταν λάβει όλο το πακέτο θα έχει την τιμή **{1,p,01110000,0}** , δηλαδή τα bits που μας νοιάζουν (scan code) μπορούν να ελεγθούν απευθείας χωρίς να χρειαστεί να τα αντιστρέψουμε.
Με αυτό τον τρόπο απλώς βάζουμε στον πρώτο decoder τα κατάλληλα bits για κάθε πλήκτρο που μας ενδιαφέρει να ελέγχεται.

Σχετικά με τον δεύτερο decoder: Ο δεύτερος decoder έχει ως σκοπό να δώσει κατάλληλες τιμές ώστε να ανάψουν τα λαμπάκια. Έχει υλοποιηθεί ως εξής:
Χρειάζεται δύο registers **led1** και **led2** των 7bit οι οποίοι κρατάνε τιμές τέτοιες ώστε να ανάψει κάποιες συγκεκριμένες γραμμές των leds στη πλακέτα.
Δέχεται ως **είσοδο** την έξοδο του πολυπλέκτη (την mux_out που περιγράψαμε πιο πάνω).
Ελέγχει την τιμή:

- Αν είναι αριθμός από 0 μέχρι 9 τότε δίνει **μόνο** στον **led1** τιμή τέτοια ώστε να ανάψει ο αριθμός αυτός στο led. Επίσης θέτει και τον led2 στο "1111111" → σβηστό.
- Άν είναι αριθμός από -9 μέχρι -1 τότε δίνει στον **led1** τιμή ώστε να ανάψει το "-" και στο **led2** την αντίστοιχη τιμή του αριθμού.
- Άν είναι διψήφιος, από 10 μέχρι 18 τότε δίνει στον **led1** τιμή τέτοια ώστε να ανάψει το "1" και στον led2 την αντίστοιχη τιμή του δευτέρου ψηφίου.

Έτσι, οι καταστάσεις του FSM δίνουν στα αντίστοιχα **hex7, hex6, hex5, hex4, hex1** και **hex0** με βάση τα led1, led2. Η χρήση του **led2** χρησιμοποιείται μόνο στην κατάσταση WAIT_EQ αφού μόνο εκεί μπορεί να υπάρξει διψήφιος αριθμός ή αρνητικός αριθμός.

TEST BENCH

Το test bench που έχουμε φτιάξει έχει ως σκοπό να εξομοιώσει τη λειτουργία του ps2 πληκτρολογίου. Έχουμε βάλει αρχικά την εξής ακολουθία πλήκτρων για να δείξουμε τη λειτουργία του bench (αρχείο **main_bench.v**):
Keypad4 → Keypad Μείων → Keypad8 → Equal
Οπότε αυτό που κάνουμε είναι:

- Εξομοίωση των clocks (συγκεκριμένα βάλαμε το clock του keyboard να είναι 10 φορές πιο αργό από του κυκλώματος).
- Αρχικοποίηση του κυκλώματος με reset στο 1 και επαναφορά στο 0 έπειτα.
- Στέλνουμε συνέχεια δεδομένα στο κύκλωμα.

Τα δεδομένα που στέλνουμε στο κύκλωμα δεν είναι τίποτα άλλο παρά scan codes. Άρα η σειρά με την οποία στέλνουμε τα δεδομένα στο κύκλωμα είναι αυτή:

KeyPad4 → F0 → KeyPad4 → Keypad Μείων → F0 → Keypad Μείων → Keypad8 → F0 → Keypad8 → Equal → F0 → Equal

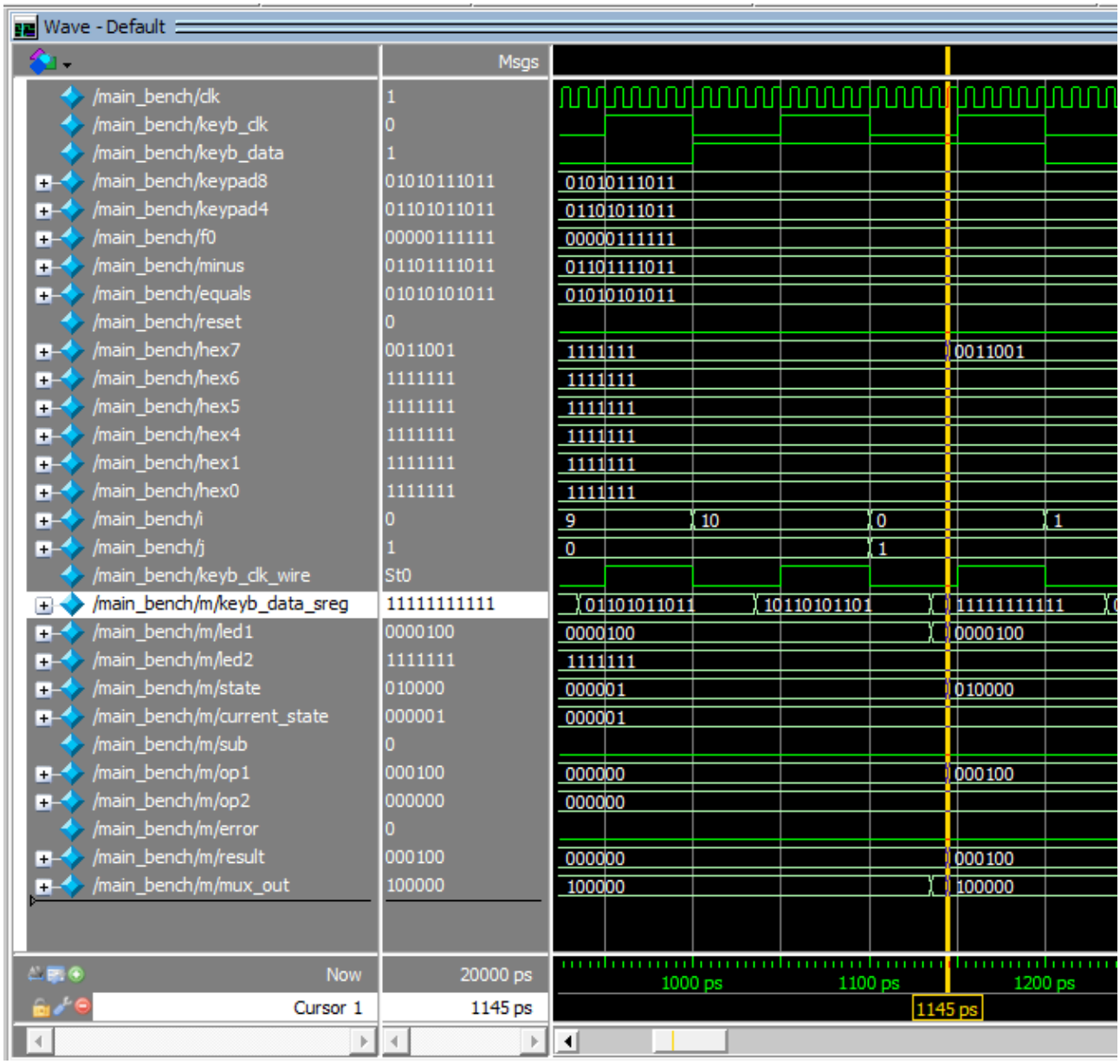
Ο λόγος που στέλνουμε με αυτή τη σειρά είναι για να εξομοιώσουμε τη λειτουργία του πληκτρολογίου.
Σε κάθε κύκλο ρολογιού του keyboard (σε κάθε αρνητική ακμή) θα σταλεί ένα bit από κάποιον τέτοιο κωδικό. Οπότε θα χρειαστούν 11 κύκλοι ρολογιού για να σταλεί ένα scan code, ενώ για να σταλεί ολόκληρη η ακολουθία scan codes θα χρειαστούν 11*12 κύκλοι ρολογιού. (Με το που σταλεί ένα scan code αμέσως στέλνουμε κανονικά τα bits του επόμενου scan code).
Στο bench, αυτή τη λειτουργία τη πετυχαίνουμε με τις μεταβλητές i και j, το i βοηθάει για να σταλεί ένα scan code bit-by-bit ενώ το j για να σταλθεί το επόμενο στη σειρά scan code. Οπότε το i πάει από 0 μέχρι 10 και μηδενίζεται όταν σταλεί κάποιο 11bit πακέτο, ενώ το j πάει από 0 μέχρι 11, δηλαδή όσα πακέτα στείλουμε (12 εδώ).
Άρα κάθε φορά που έρθει αρνητική ακμή του εξομοιωμένου clock του πληκτρολογίου στέλνουμε ένα bit των παραπάνω πακέτων, στον επόμενο κύκλο θα σταλθεί το επόμενο, μέχρις ότου σταλθεί όλο το πακέτο, οπότε και τότε θα αλλάξει το j → j=j+1 για να σταλθεί το επόμενο πακέτο.

Στη συνέχεια θα στείλουμε μια λάθος ακολουθία, καθώς και επαναφορά από αυτή (**main_bench2.v**), δηλαδή:

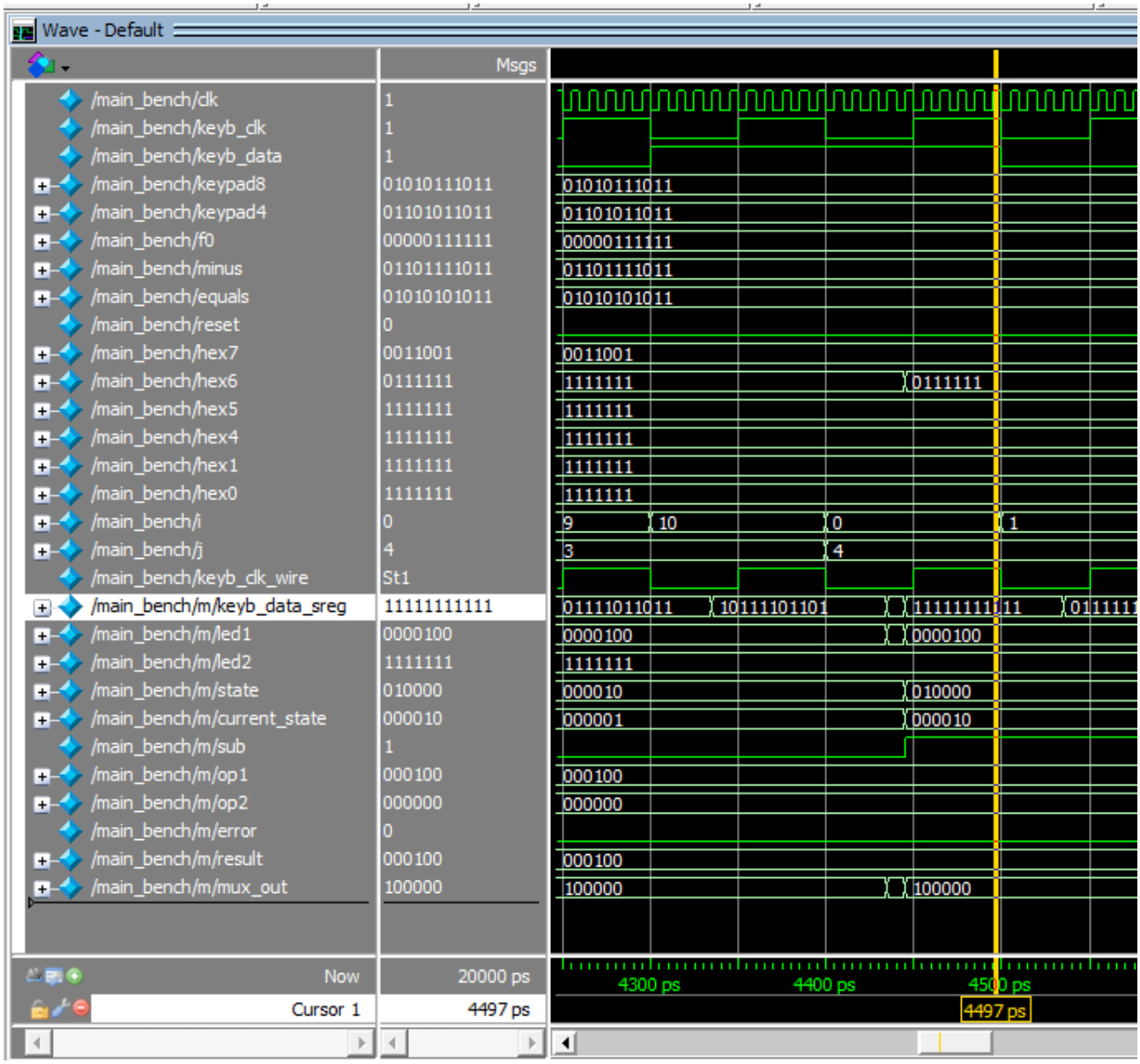
Keypad9 → F0 → Keypad9 → Keypad9 → F0 → Keypad9 → Keypad+ → F0 → Keypad+ → Keypad0 → F0 → Keypad0 → Equal → F0 → Equal

Το κύκλωμα θα πρέπει να εντοπίζει το λάθος και να παραμείνει στην κατάσταση **WAIT_OPERATOR** (αφού εκεί δίνεται λάθος τιμή) και μόλις του δοθεί “+” ή “-” να προχωρήσει κανονικά και να δουλέψει.

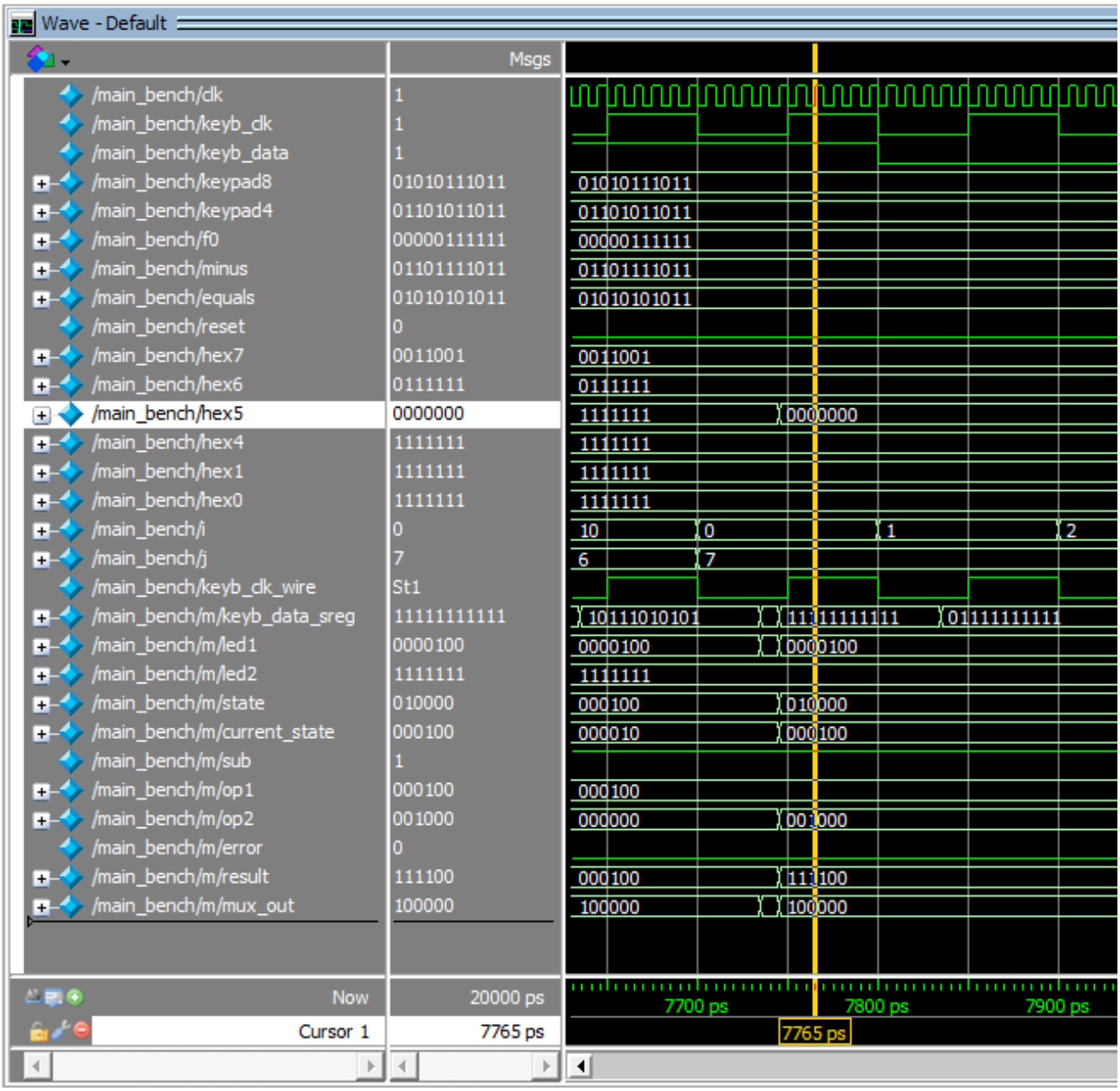
ΠΕΡΙΠΤΩΣΗ 1: ΑΠΛΗ ΑΚΟΛΟΥΘΙΑ



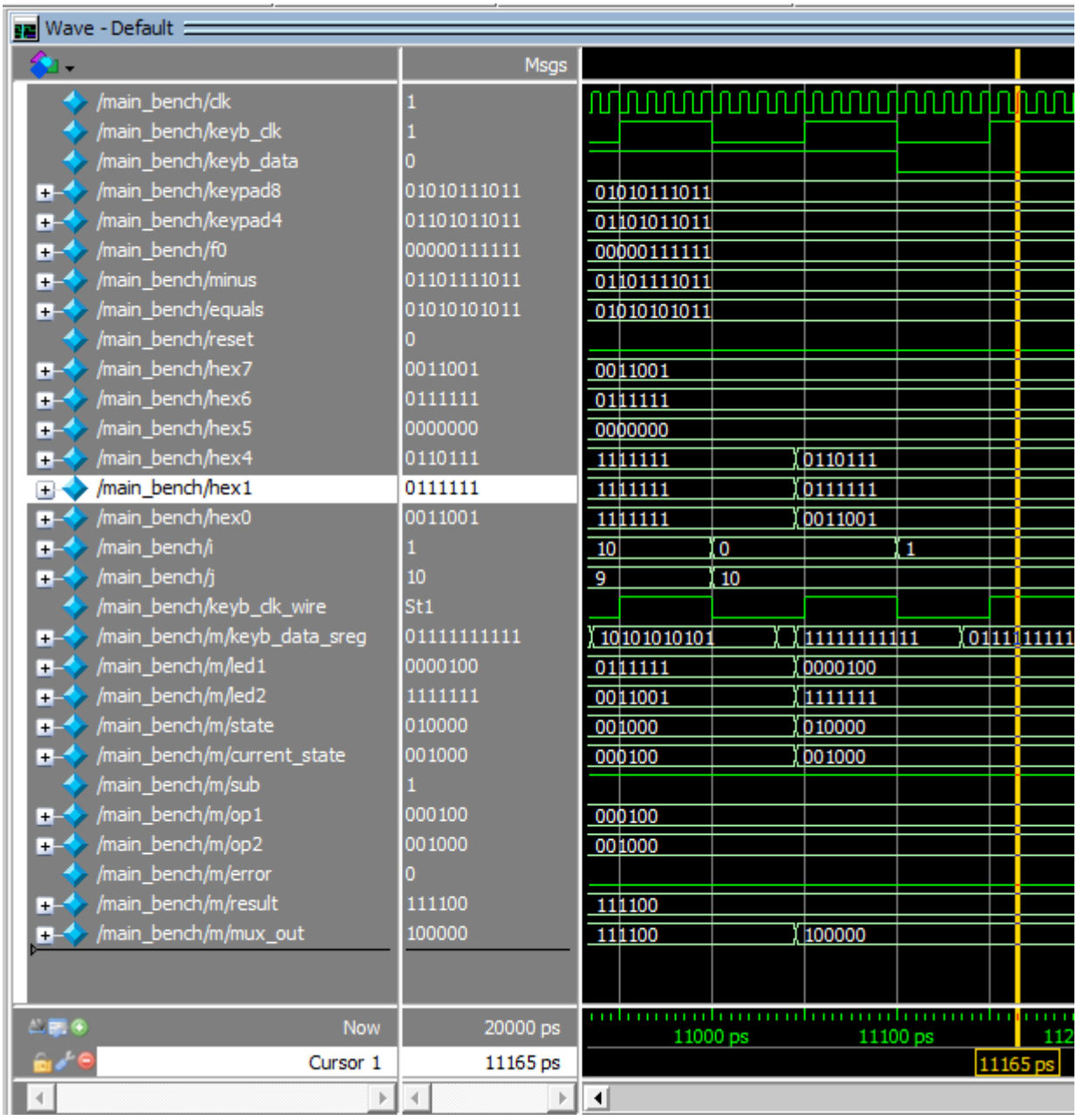
Εδώ βλέπουμε τη χρονική στιγμή στην οποία έχει περάσει η κατάσταση “WAIT_OPERAND1” και έχει ανάψει το πρώτο λαμπάκι (hex7, έχει ανάψει το 4). Από εκεί και μετά περιμένουμε το F0 καθώς και το ίδιο το scan code (δηλαδή ο register **state** είναι στην WAIT_F0 και ο **current_state** είναι στην WAIT_OPERATOR1).



Τη χρονική στιγμή αυτή, εφόσον έχει έρθει το F0 και το scan code (του προηγούμενου αριθμού) είμαστε στη κατάσταση “WAIT_OPERATOR” και μας ήρθε το “-” οπότε ανάβουμε το hex6 ώστε να δείχνει “-” και πάλι προχωράμε το state ώστε να περιμένει F0 και scan code.

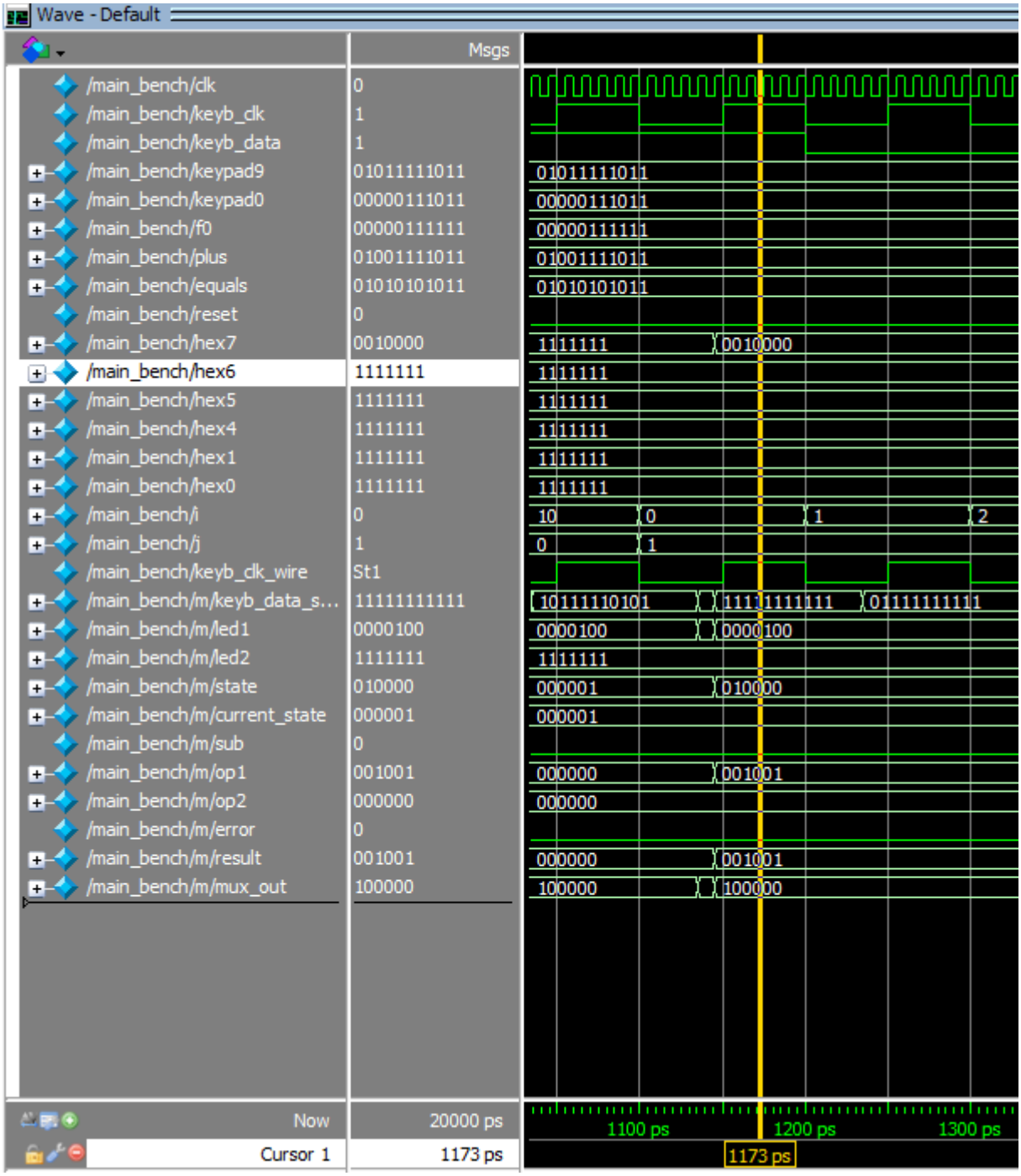


Εδώ λαμβάνουμε τον 2ο αριθμό, είμαστε στην WAIT_OPERAND2 και όπως και πριν έτσι και εδώ ανάβουμε το αντίστοιχο φωτάκι (hex5) και αναμένουμε F0 και scan code. Συγκεκριμένα έχει έρθει το πακέτο που στέλνει το πληκτρολόγιο για το Keypad8 οπότε το hex5 ανάβει όλες τις γραμμές (για αυτό είναι 0000000).

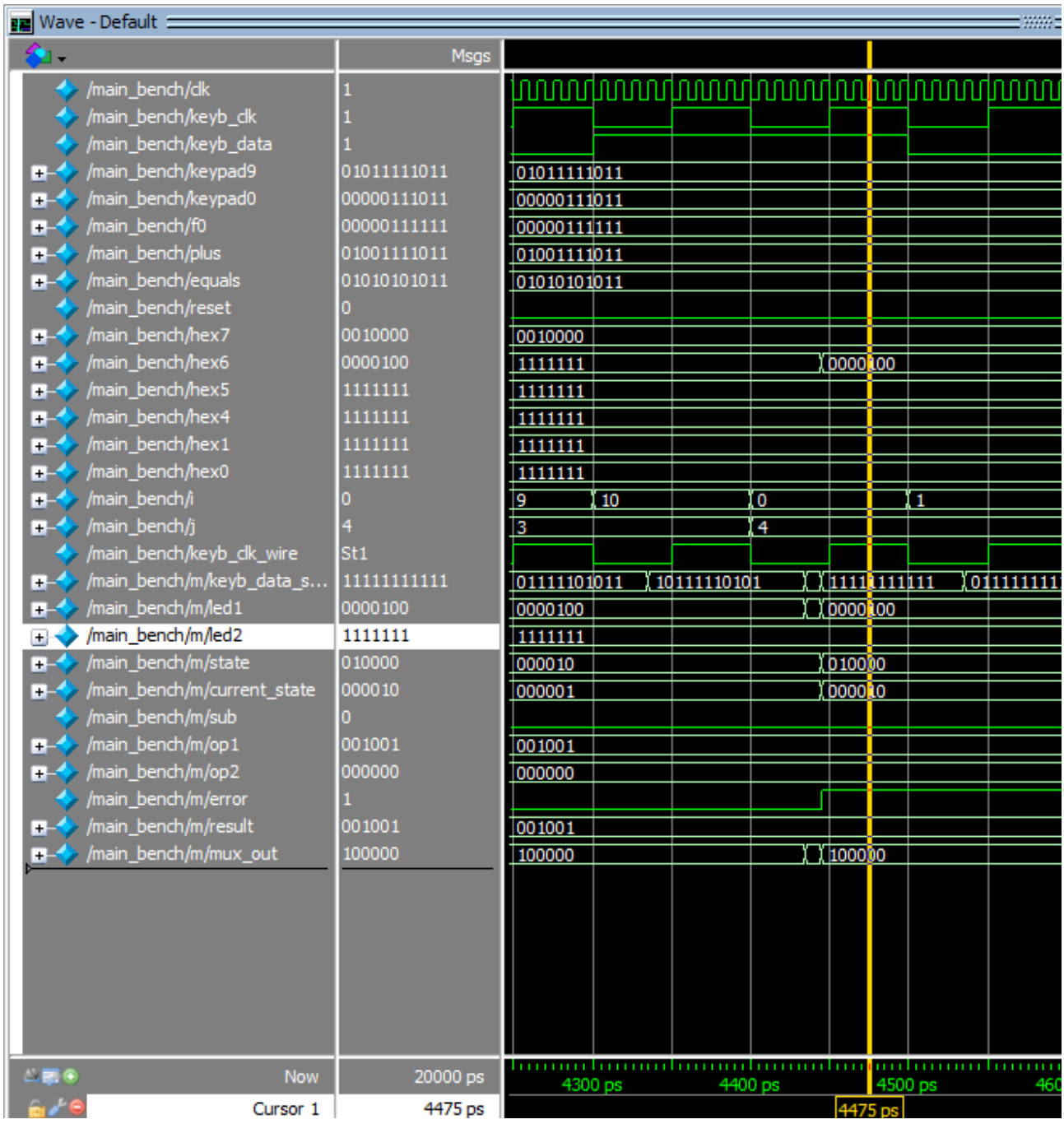


Τέλος, εφόσον έχουμε λάβει τώρα και το “=”, υπολογίζεται το αποτέλεσμα, δηλαδή είτε η αφαίρεση είτε η πρόσθεση και αποθηκεύεται στην result ως συμπλήρωμα ως προς 2 (αν είναι αρνητικός). Στη συγκεκριμένη περίπτωση result = -4 και άρα οι καταχωρητές led1 και led2 θα δώσουν κατάλληλες τιμές ώστε ο led1 → να ανάψει το “-” και led2→ να ανάψει το 4.
Πλέον έχει ολοκληρωθεί το test bench, τώρα απλώς περιμένουμε F0 και scan code του “=” και στη συνέχεια το κύκλωμα επιστρέφει πάλι στην κατάσταση WAIT_OPERATOR1.
Τα led1 και led2 στη συνέχεια έχουν αλλάξει, αφού είναι συνδυαστικό κύκλωμα, σε κάθε shifting θα δείχνουν διαφορετικές τιμές, αλλά αυτό δεν μας απασχολεί διότι εμείς παίρνουμε τις τιμές τους τη κατάλληλη στιγμή (όταν έχει ληφθεί όλο το πακέτο).

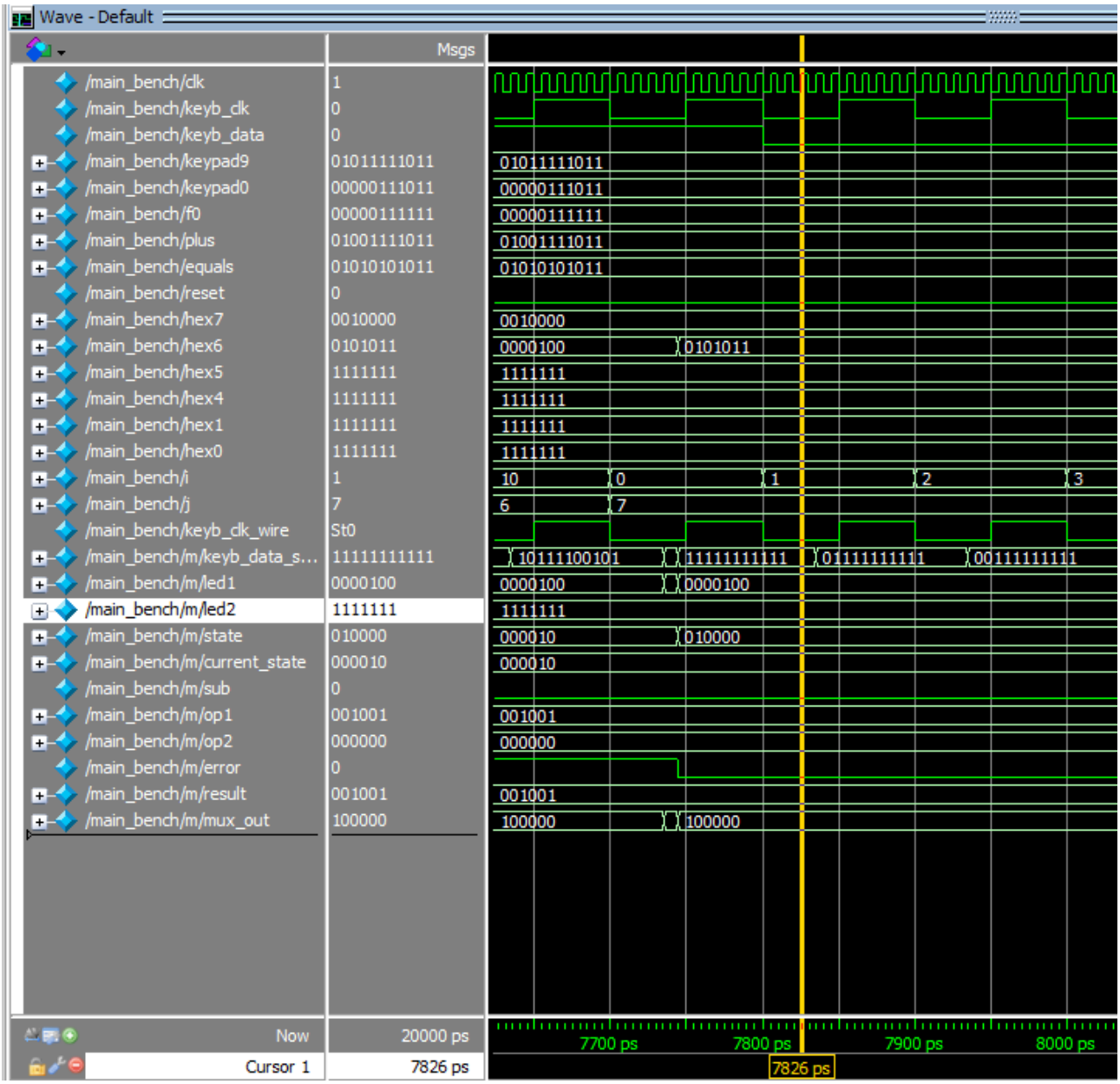
ΠΕΡΙΠΤΩΣΗ 2Η: ΕΠΑΝΑΦΟΡΑ ΑΠΟ ΣΦΑΛΜΑ



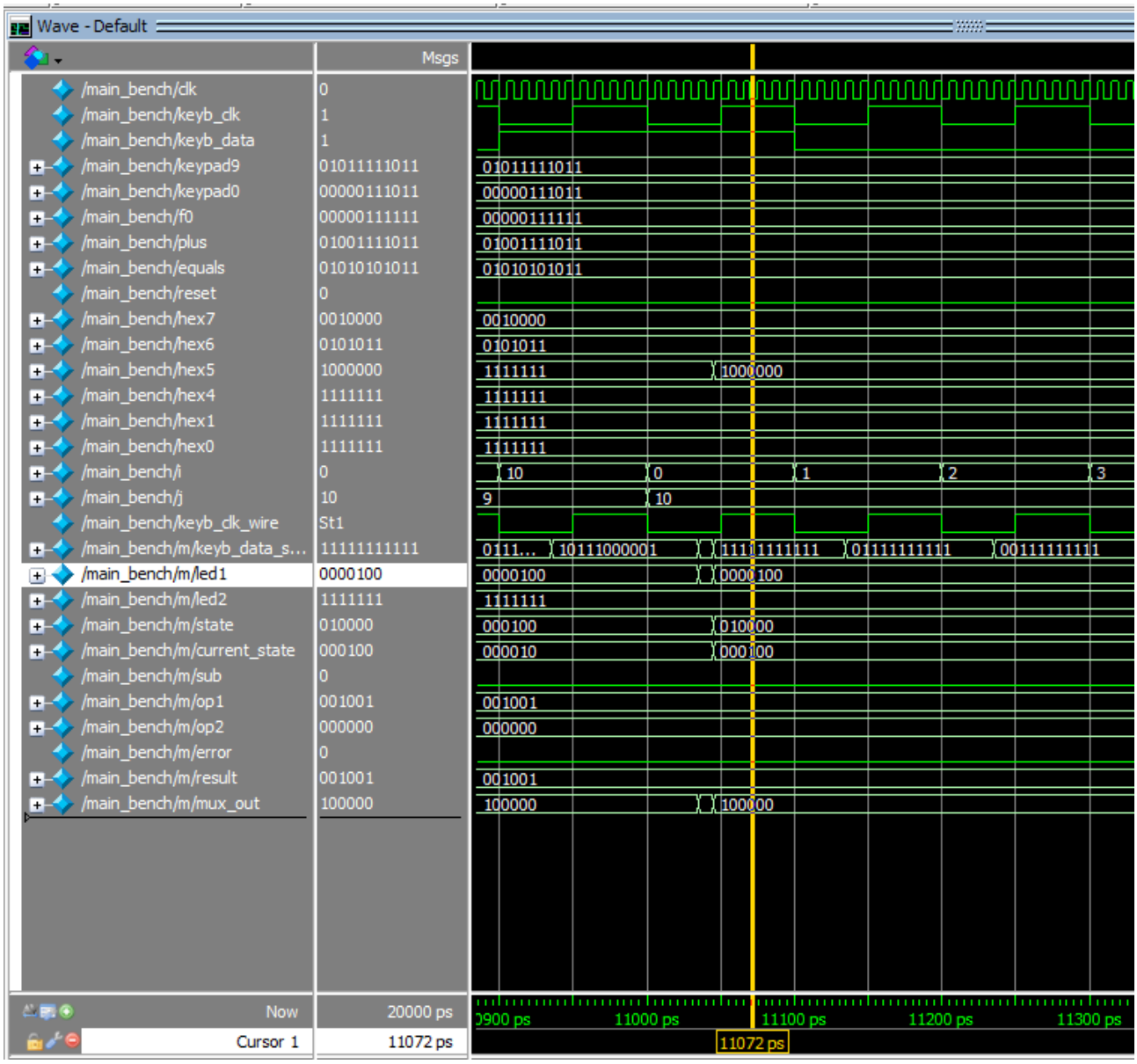
Αρχικά αναγνωρίζεται το πακέτο που περιλαμβάνει το keypad9 και ανάβει το πρώτο led.



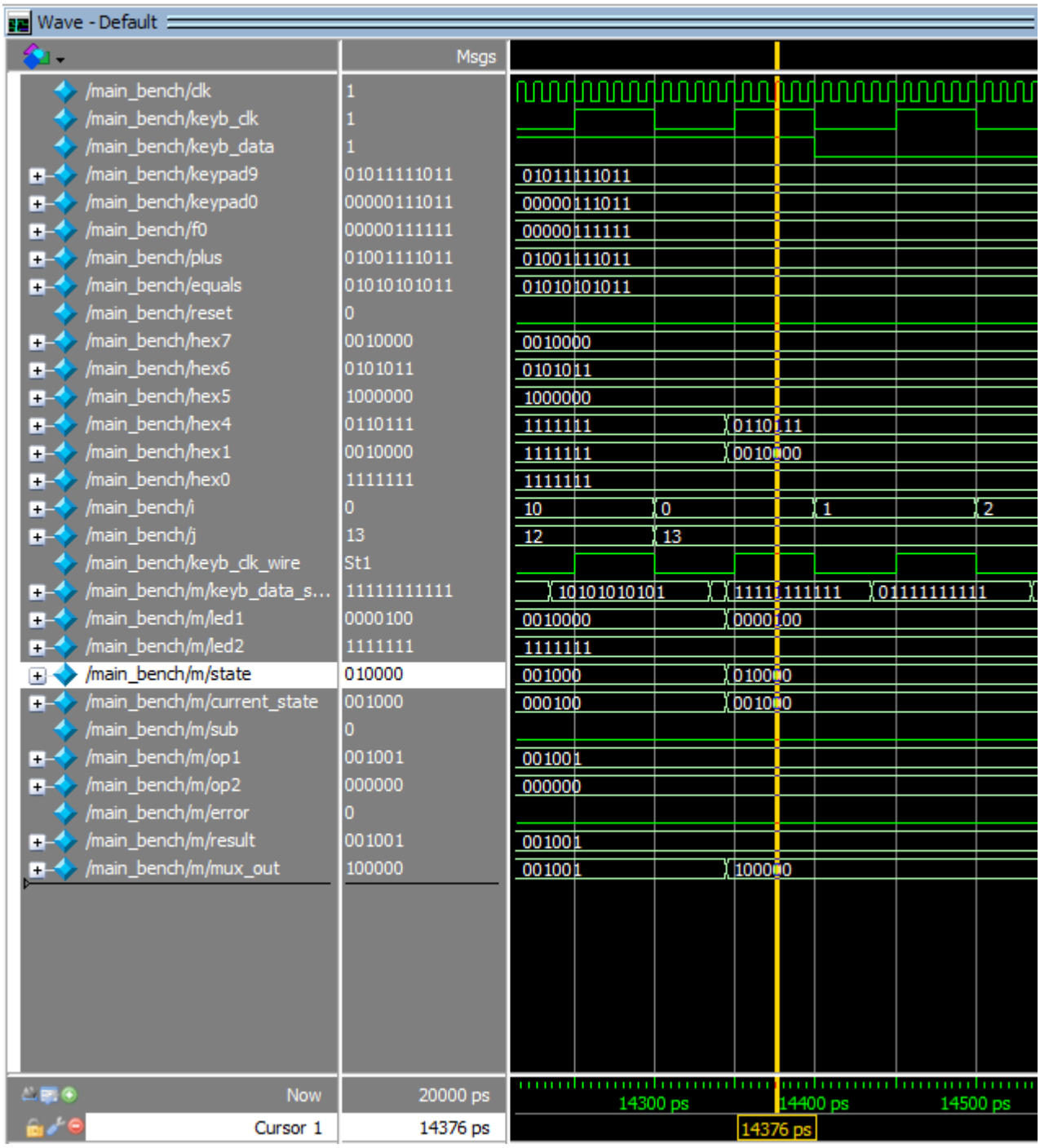
Αφού διαβαστεί το F0 και το πακέτο του keypad9, τώρα θα διαβάζεται πάλι το keypad9. Λόγω ότι είμαστε στη κατάσταση WAIT_OPERATOR, το FSM θέτει τη τιμή του **error** register ίση με 1. Τώρα θα δεχτεί το F0 και το scan code και θα ξανα **επιστρέψει** στην WAIT_OPERATOR, χάρης στον **current_state** καταχωρητή. Το δεύτερο λαμπάκι τώρα δείχνει το “e” (hex6).



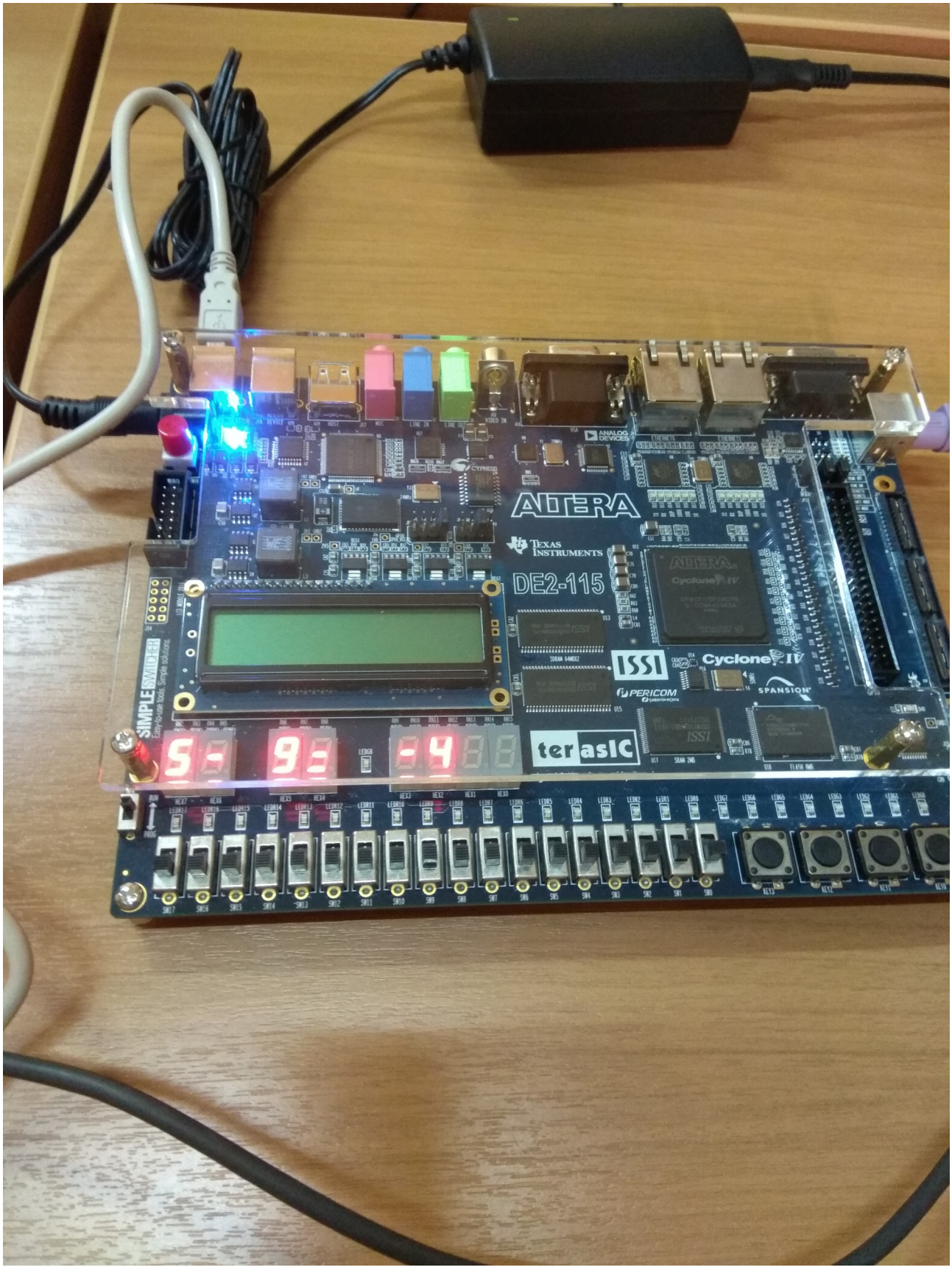
Τώρα διαβάζεται το “+” και ανάβει κανονικά το 2ο led να δείχνει το “+” καθώς και το error register τίθεται ίσο με 0, οπότε και πάμε κανονικά στην WAIT_F0, AFTER_F0 και μετά στην WAIT_OPERATOR2

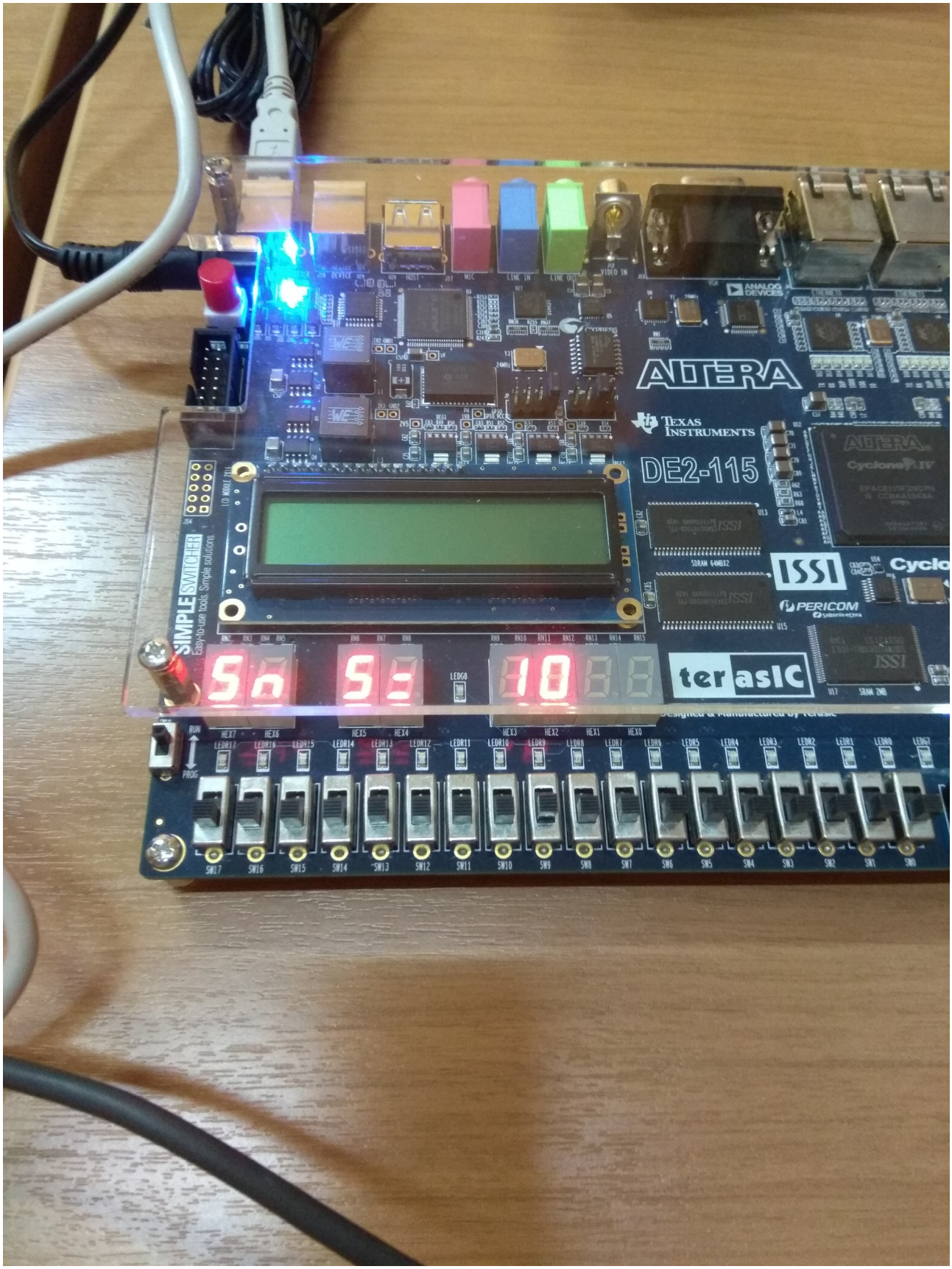


Εδώ διαβάζεται κανονικά το Keypad0. Πάλι ανάβει το 3ο led να δείχνει στο 0 και προχωράμε κανονικά στην WAIT_EQ (αφού πρώτα πάλι περάσουμε από τις WAIT/AFTER_F0).



Αφού διαβαστεί και το “=” υπολογίζεται κανονικά το αποτέλεσμα και εμφανίζεται στην οθόνη, μαζί με το “=”. Συγκεκριμένα 9+0 = 9, οπότε **led1 = 9** και **led2 = σβηστό**.


ALTERA
DE2-115
Cyclone IV
ISSI
PERICOM
SPANSION
terasIC


ALTERA
TEXAS INSTRUMENTS
DE2-115
ISSI
PERICOM
terasIC
SIMPLE SWITCHER
Easy-to-use tools. Simple solutions.
RUN
PROG