ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2024-2025

ΑΘΗΝΑ 8 Νοεμβρίου 2024

6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

ΟΜΑΔΑ 23

Συνεργάτες

Νικόλαος Αναγνώστου
03121818

Νικόλαος Λάππας
03121098

Ζήτημα 6.1: Στην συγκεκριμένη άσκηση καλούμαστε να ελέγξουμε την ενεργοποίηση τεσσάρων πλήκτρων από τα δεκαέξι συνολικά πλήκτρα του πληκτρολογίου της πλακέτας. Κάθε φορά που ανιχνεύεται ένα από τα ζητούμενα πλήκτρα, ανάβουμε το αντίστοιχο Led της πύλης PORTB. Αυτό που κάνουμε είναι να χρησιμοποιήσουμε, αρχικά, τις δοσμένες συναρτήσεις που ελέγχουν την εγγραφή και την ανάγνωση ενός από τους καταχωρητές ελέγχου του ολοκληρωμένου PCA9555 και στην συνέχεια ορίζουμε τις τέσσερις συναρτήσεις που ζητάει η εκφώνηση. Θα αναλύσουμε τις ζητούμενες συναρτήσεις με την σειρά που καλούνται στο πρόγραμμά μας. Συγκεκριμένα,

- Η συνάρτηση $keypad\_to\_ascii()$, παίρνει την 16-bit τιμή που επιστρέφει η $scan\_keypad\_rising\_edge()$ και την χωρίζει σε τέσσερις τετράδες. Κάθε τετράδα την αποθηκεύει στα τέσσερα LSB μιας νέας μεταβλητής η οποία δείχνει σε ποια γραμμή βρισκόμαστε. Σε αυτά τα τέσσερα bits που μας ενδιαφέρουν, είναι αποθηκευμένη η πληροφορία των στηλών που έχουν ενεργοποιηθεί στην συγκεκριμένη γραμμή. Επομένως, περνάμε σε έναν έλεγχο μετά αναλόγως σε ποια γραμμή βρισκόμαστε, και εξετάζουμε ποια στήλη έχει ενεργοποιηθεί, και ανάλογα στέλνουμε το κατάλληλο input (αντιστοίχιση που κάνουμε με τα πλήκτρα). Για παράδειγμα, το πλήκτρο ‘6’ έχει κωδική αντιστοίχιση με τον δυαδικό αριθμό ‘1011 1011’, με τα πρώτα τέσσερα MSB να αφορούν την στήλη και τα τέσσερα LSB να αφορούν την γραμμή. Έτσι στην main συνάρτηση κάνουμε τους απαραίτητους ελέγχους προκειμένου να αγνοούμε κάθε άλλο πλήκτρο εκτός των τεσσάρων που ζητούνται, και από αυτά τα τέσσερα κάθε φορά να αντιλαμβανόμαστε ποιο έχει πατηθεί.

- Η συνάρτηση $scan\_keypad\_rising\_edge()$, είναι υπεύθυνη να αντιμετωπίζει το πρόβλημα του σπινθηρισμού. Συγκεκριμένα, παίρνουμε μέσω της $scan\_keypad()$ το πρώτο κουμπί που θεωρητικά πατιέται, και μετά από 10 msec ξανακαλούμε την ίδια συνάρτηση και παίρνουμε ακόμα μια είσοδο. Από την δεύτερη τιμή αγνοούμε τα bit που διαφέρουν σε σχέση με την πρώτη τιμή εισόδου, και θα συγκρίνουμε το τελικό αποτέλεσμα με την προηγούμενη τιμή που έχουμε αποθηκευμένη σε κάθε νέα κλήση της εν λόγω συνάρτησης. Επιστρέφουμε τελικά την ελεγμένη τιμή πλήκτρου που πατήθηκε.

- Η συνάρτηση $scan\_keypad()$, καλεί τέσσερις φορές την συνάρτηση $scan\_row()$, μια φορά για κάθε γραμμή, και την 8-bit τιμή που επιστρέφεται σε αυτή, την κάνει shift αριστερά τέσσερις θέσεις (τα 4 LSB της 8-bit τιμής μας είναι χρήσιμα), προκειμένου να αποθηκευτεί όλη η χρήσιμη πληροφορία από αυτές τις τέσσερις κλήσεις, σε μια μόνο 16-bit μεταβλητή. Αυτή η μεταβλητή περιέχει τέσσερα συνεχόμενα bit για κάθε σειρά που δείχνουν ποια στήλη κάθε σειρά είναι ενεργοποιημένη (έχει τιμή μηδέν). Για παράδειγμα, αν παίρναμε ‘0111 1011 1101 1110’ θα ξέραμε ότι οι στήλες 4 της γραμμής 1, 3 της γραμμής 2, 2 της γραμμής 1 και 1 της γραμμής 4 έχουν ενεργοποιηθεί.

- Η συνάρτηση $scan\_row()$, ανάλογα με το ποια γραμμή ελέγχουμε κάθε φορά, μπαίνει στο αντίστοιχο case i, και ενεργοποιεί μια μάσκα, την μάσκα για την αντίστοιχη γραμμή i. Αυτή την μάσκα την χρησιμοποιούμε μετά προκειμένου να ορίσουμε ποιας γραμμής στήλες θέλουμε να εξετάσουμε. Συγκεκριμένα, κάνουμε $PCA9555\_0\_write(REG\_OUTPUT\_1, row\_mask)$, το οποίο ενεργοποιεί μέσω του ολοκληρωμένου την εκάστοτε γραμμή σαν έξοδο (τιμή 0) και όλες τις άλλες τις αγνοεί. και μετά $PCA9555\_0\_read(REG\_INPUT\_1)$ με το οποίο παίρνουμε το 8-bit δεδομένο εισόδου που αφορά τις ενεργοποιημένες στήλες τις γραμμής. Κάνουμε λογικό και απομονώνουμε τα τέσσερα MSB που αφορούν την είσοδο μας (στήλες) και το κάνουμε shift δεξιά τέσσερις θέσεις για να το έχουμε στα LSB για καλύτερη διαχείριση.

Ακολουθεί ο κώδικας με τις παραπάνω συναρτήσεις και την main loop μας, χωρίς τον κώδικα για το ολοκληρωμένο, ο οποίος έχει δοθεί.

```
#define ROW_1_MASK 0xFE // Row 1 low to check only that
#define ROW_2_MASK 0xFD // Row 2 low to check only that
#define ROW_3_MASK 0xFB // Row 3 low to check only that
#define ROW_4_MASK 0xF7 // Row 4 low to check only that

uint8_t scan_row(uint8_t row_to_check)
{
  uint8_t row_mask;
  switch (row_to_check)
  {
    case 1: row_mask = ROW_1_MASK; break;
    case 2: row_mask = ROW_2_MASK; break;
    case 3: row_mask = ROW_3_MASK; break;
    case 4: row_mask = ROW_4_MASK; break;
    default: return 0; // If invalid row
  }
  PCA9555_0_write(REG_OUTPUT_1, row_mask);

  uint8_t input = PCA9555_0_read(REG_INPUT_1);   // Read the input
  // Mask to take the column of keypad from input pressed
  // Return, the shifted to the right, result representing the active columns of the specific row
  return (input & 0xF0) >> 4;
}

uint16_t scan_keypad()
{
  uint16_t bottoms_pressed = 0;              // None pressed at the beginning
  for (uint8_t row = 1; row <= 4; row++)
  {
```

```
    uint8_t columns_selected = scan_row(row);   // Get the columns activated from each row
    bottoms_pressed |= columns_selected;
    // For each row, save the columns pressed into the 16-bit bottoms_pressed
    // For row 1, the 4-bit answer will be in the 4 MSB bits of bottoms_pressed
    // For row 2, the 4-bit answer will be in the 4 LSB of the upper half (8-bit) and so on ...
    if (row != 4) {bottoms_pressed = bottoms_pressed << 4;}
    _delay_ms(1);
  }
  return bottoms_pressed; /* e.g if bottoms_pressed = 0111 1011 1101 1110 means that
                * from row 1, column 4 is activated
                * from row 2, column 3 is activated
                * from row 3, column 2 is activated
                * from row 4, column 1 is activated
                */
}

uint16_t pressed_keys_tempo = 0xFFFF;           // First key pressed is 'none', save previous pressed key
uint16_t scan_keypad_rising_edge()
{
  uint16_t pressed_keys = scan_keypad();
  _delay_ms(10);
  uint16_t recheck_pressed_keys = scan_keypad();  // Check again (de-bouncing)

  recheck_pressed_keys |= pressed_keys;         // Get rid of the 'bits' that weren't previously pressed
  recheck_pressed_keys |= ~pressed_keys_tempo;   // Compare with the previous pressed keys
  pressed_keys_tempo = pressed_keys;          // Save the current pressed keys into tempo for future use
  return recheck_pressed_keys;                // Return verified key
}

uint16_t key;
uint8_t keypad_to_ascii()
{
  key = scan_keypad_rising_edge();            // 16-bit

  if (key == 0xFFFF) {return 0x00;}           // Nothing is pressed

  uint8_t row1 = (key >> 12) & 0x0F;          // Extract bits 12-15 (Row 1)
  uint8_t row2 = (key >> 8) & 0x0F;           // Extract bits 8-11 (Row 2)
  uint8_t row3 = (key >> 4) & 0x0F;           // Extract bits 4-7 (Row 3)
  uint8_t row4 = key & 0x0F;                  // Extract bits 0-3 (Row 4)

  for (uint8_t row = 1; row <= 4; row++)
  {
    switch (row)
    {
      case 1:
        if (row1 == 0b00001110) {return 0b11101110; break;}    // "*"
        else if (row1 == 0b00001101) {return 0b11011110; break;}// "0"
        else if (row1 == 0b00001011) {return 0b10111110; break;}// "#"
```

```
        else if (row1 == 0b00000111) {return 0b01111110; break;}// "D"

      case 2:
        if (row2 == 0b00001110) {return 0b11101101; break;}    // "7"
        else if (row2 == 0b00001101) {return 0b11011101; break;}// "8"
        else if (row2 == 0b00001011) {return 0b10111101; break;}// "9"
        else if (row2 == 0b00000111) {return 0b01111101; break;}// "C"

      case 3:
        if (row3 == 0b00001110) {return 0b11101011; break;}    // "4"
        else if (row3 == 0b00001101) {return 0b11011011; break;}// "5"
        else if (row3 == 0b00001011) {return 0b10111011; break;}// "6"
        else if (row3 == 0b00000111) {return 0b01111011; break;}// "B"

      case 4:
        if (row4 == 0b00001110) {return 0b11100111; break;}    // "*"
        else if (row4 == 0b00001101) {return 0b11010111; break;}// "0"
        else if (row4 == 0b00001011) {return 0b10110111; break;}// "#"
        else if (row4 == 0b00000111) {return 0b01110111; break;}// "D"
    }
  }
}

int main(void)
{
  twi_init();

  DDRB = 0xFF;                          // Initialize PORTB as output
  PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);    // Set IO1[0:3] as output and IO1[4:7] as input (set)

  while(1)
  {
    uint8_t input = keypad_to_ascii();        // Read input

    if (input == 0b01110111) PORTB = 0x01;      // Pressed 'A'
    else if (input == 0b10111011) PORTB = 0X04;  // Pressed '6'
    else if (input == 0b11011101) PORTB = 0x02;  // Pressed '8'
    else if (input == 0b11101110) PORTB = 0x08;  // Pressed '*'
    else PORTB = 0x00; // None of the desired keys is pressed
  }
}
```

Ζήτημα 6.2: Η συγκεκριμένη άσκηση αποτελεί μια παραλλαγή της προηγούμενης, και συγκεκριμένα, αντί να ανάβουμε προκαθορισμένα Led της θύρας εξόδου PORTB, χρησιμοποιούμε την LCD οθόνη και εκτυπώνουμε εκεί το αποτέλεσμα μας. Πέρα από τις συναρτήσεις που χρειαστήκαμε στην άσκηση 6.1, σε αυτή εισαγάγαμε και τις συναρτήσεις για την οθόνη από την 4η σειρά, όπου χρησιμοποιούσε την θύρα D και όχι το ολοκληρωμένο κύκλωμα για την οπτικοποίηση των δεδομένων. Αυτό που κάνουμε είναι να προσθέσουμε έξτρα το $PCA9555\_0\_write(REG\_CONFIGURATION\_0, 0x00)$, το οποίο ενδεχομένως είναι περιττό και μετά ανάλογα με την τιμή που θα επιστρέψει η συνάρτηση $keypad\_to\_ascii()$, θα μπούμε σε έναν από τους 16 συνολικούς ελέγχους και θα τυπώσουμε στην οθόνη τον αντίστοιχο χαρακτήρα.

Ακολουθεί η main συνάρτηση μας.

```
int main(){
  DDRD = 0xFF;
  twi_init();

  PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
  PCA9555_0_write(REG_CONFIGURATION_0, 0x00);

  lcd_init();

  while(1)
  {
    //lcd_clear_display();
    lcd_command(0x80);

    uint8_t input = keypad_to_ascii();        // Read input

    if (input == 0b01110111) lcd_data('A');     // Pressed 'A'
    else if (input == 0b10110111) lcd_data('3'); // Pressed '3'
    else if (input == 0b11010111) lcd_data('2'); // Pressed '2'
    else if (input == 0b11100111) lcd_data('1'); // Pressed '1'
    else if (input == 0b01111011) lcd_data('B'); // Pressed 'B'
    else if (input == 0b10111011) lcd_data('6'); // Pressed '6'
    else if (input == 0b11011011) lcd_data('5'); // Pressed '5'
    else if (input == 0b11101011) lcd_data('4'); // Pressed '4'
    else if (input == 0b01111101) lcd_data('C'); // Pressed 'C'
    else if (input == 0b10111101) lcd_data('9'); // Pressed '9'
    else if (input == 0b11011101) lcd_data('8'); // Pressed '8'
    else if (input == 0b11101101) lcd_data('7'); // Pressed '7'
    else if (input == 0b01111110) lcd_data('D'); // Pressed 'D'
    else if (input == 0b10111110) lcd_data('#'); // Pressed '#'
    else if (input == 0b11011110) lcd_data('0'); // Pressed '0'
    else if (input == 0b11101110) lcd_data('*'); // Pressed '*'
    //else lcd_clear_display(); // None of the desired keys is pressed
    //_delay_ms(2000);
```

```
  }
}
```

Ζήτημα 6.3:

Σε αυτή την άσκηση έπρεπε να φτιάξουμε σε c ένα πρόγραμμα κλειδαριάς όπου ο χρήστης πρέπει και καλείται να πατήσει δύο διαδοχικά ψηφία και μόνο τα οποία ψηφία έπρεπε να σχηματίζουν τον αριθμό της ομάδας μας , δηλαδή το 23 . Οποιοσδήποτε άλλος αριθμός έπρεπε να θεωρείται λάθος.

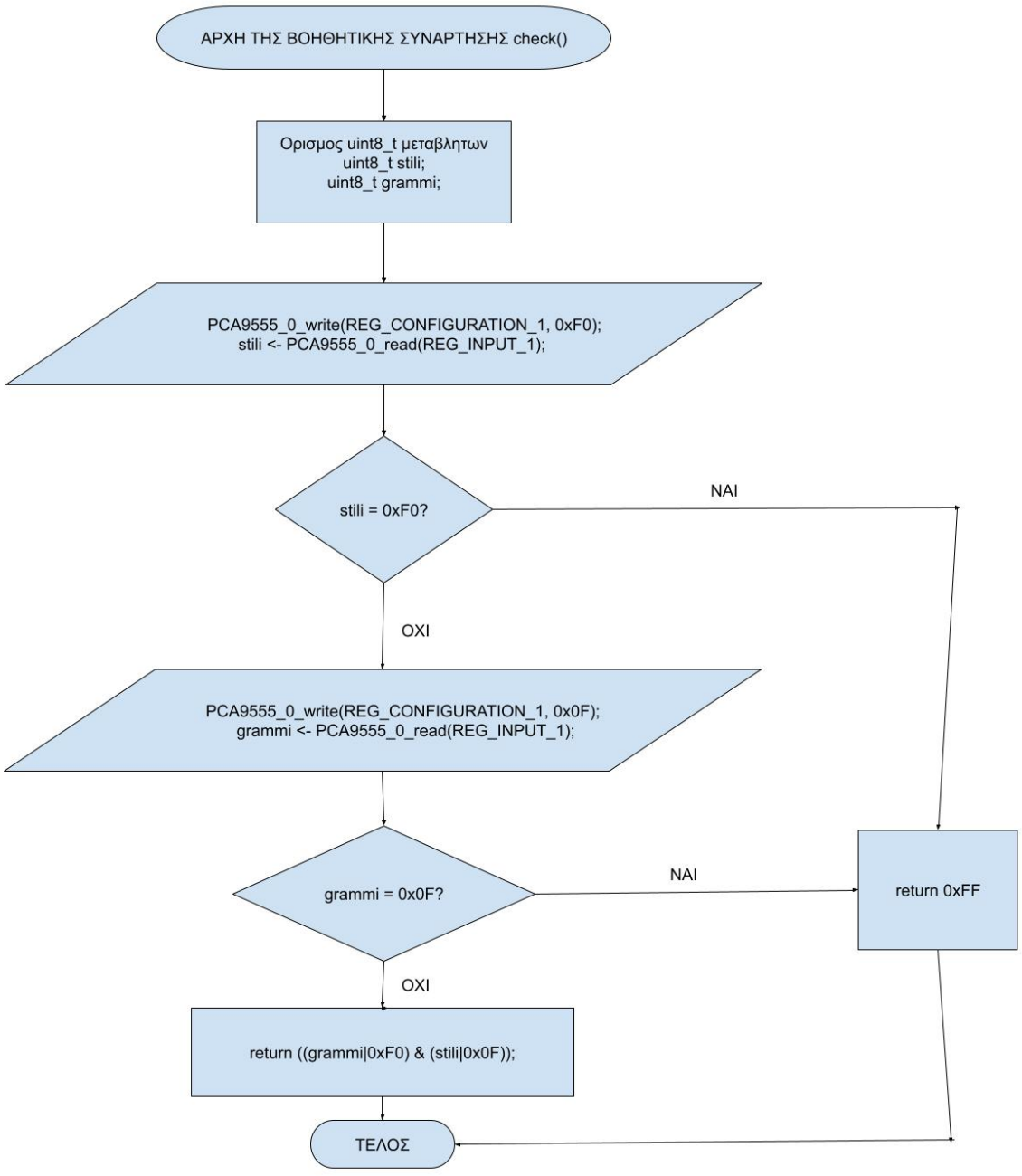
Καταρχήν ζητήθηκε όταν ο χρήστης κρατά πατημένο ένα πλήκτρο, το πρόγραμμα μας να το καταλαβαίνει ως ένα και μόνο πάτημα. Για αυτό το λόγο γίνεται ένας έλεγχος μετά από κάθε πάτημα. Αυτό σημαίνει ότι ο αριθμός-πλήκτρο που πατιέται κάθε φορά «γίνεται αντιληπτός από το πρόγραμμα» και η λογική της κλειδαριάς συνεχίζεται μονάχα όταν ο χρήστης αφήσει το εκάστοτε πλήκτρο είτε αυτό είναι το πρώτο ψηφίο είτε το δεύτερο. Όσο το κρατά πατημένο το πρόγραμμα κρατάει στη μνήμη του το πλήκτρο που πατήθηκε και κολλάει σε μια loop αναμένοντας πότε ο χρήστης θα το αφήσει προκειμένου να συνεχίσει την λειτουργία του και ουσιαστικά να σκεφτεί αν πατήθηκε το σωστό πλήκτρο.

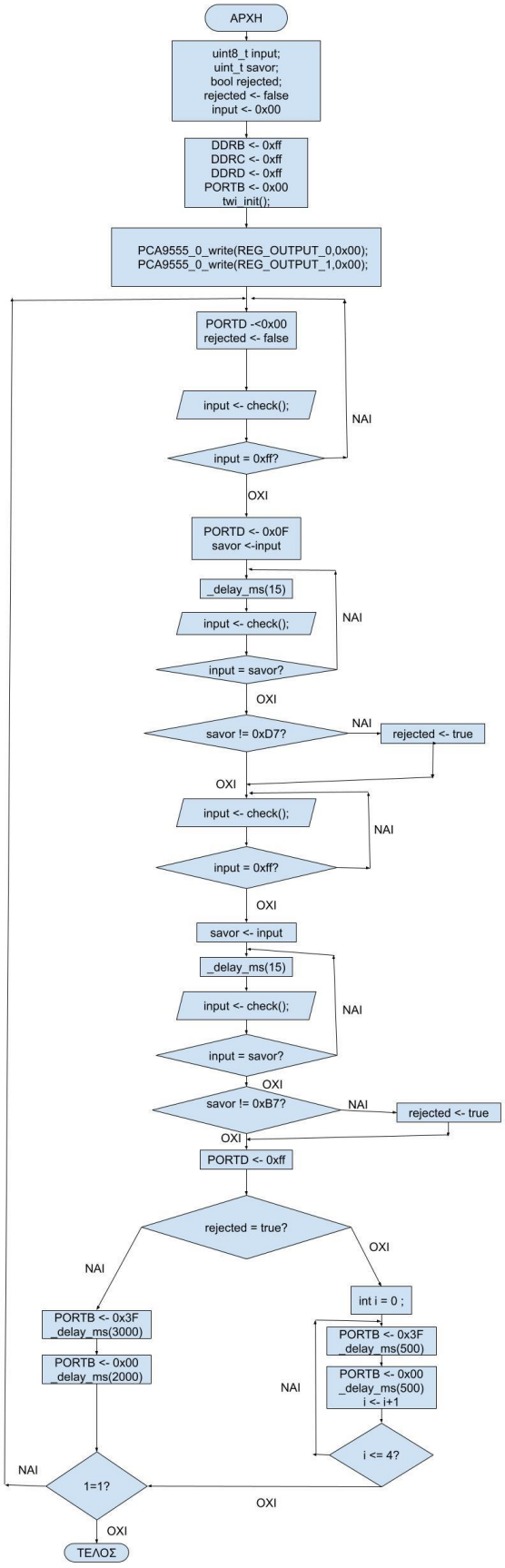
Πέρα από την απλή επίδειξη του κώδικα και των διαγραμμάτων ροής οφείλουμε να εξηγήσουμε την συνάρτηση check() η οποία κατασκευάστηκε με τρόπο διαφορετικό από αυτό που ζητούσαν ενδεχομένως η 6.2 και η 6.1 .

Η check() ουσιαστικά αντί να εκτελεί άλλες συναρτήσεις που κάνουν μέσα loops για να βρούνε στήλη και γραμμή επιλέχθηκε ένας πιο έξυπνος τρόπος .Συνοπτικά μέσα σε αυτήν σε πρώτο στάδιο γράφουμε στο configuration 1 πρώτα το 0xF0 προκειμένου να βρούμε έξυπνα σε ποια στήλη βρίσκεται το πλήκτρο που ο χρήστης έχει πατήσει εκείνη την στιγμή βλέποντας ποια από τα MSBS θα γίνει 0. Κατόπιν γράφουμε στο configuration 1 το 0x0F προκειμένου να βρούμε σε ποια γραμμή βρίσκεται το πλήκτρο που πατιέται κάθε φορά... έτσι στο τέλος επιστρέφοντας το return ((grammi|0xF0) & (stili|0x0F)) έχουμε έξυπνα τον κωδικοποιημένο αριθμό που βγάζει τη λογική που υποδεικνύει η εκφώνηση της άσκησης. **Ο κίνδυνος αυτού του έξυπνου τρόπου είναι η περίπτωση που είναι αρκετά συνήθης σε ανθρώπινους χρόνους είναι ότι ουσιαστικά το πάτημα του πλήκτρου μπορεί να αλλάξει σε μία ενδιάμεση κατάσταση με αποτέλεσμα να πάρουμε τιμή που υποδεικνυεί ότι πατιέται πλήκτρο σε μία γραμμή αλλά σε καμία στήλη...και το αντίστροφο. Αυτό ακούγεται καταστροφικό αλλά αντιμετωπίστηκε έξυπνα τσεκάροντας κάθε φορά αν δεν βρέθηκε καμία στήλη ή καμία γραμμή...αν αυτό συνέβη τότε η συνάρτηση check() σταματά βίαια και επιστρέφει το 0xff που σημαίνει πως κανένα πλήκτρο δεν είναι πατημένο .Με αυτό το πρόγραμμα τρέχει σωστά κάθε φορά.**

Όλα τα υπόλοιπα εξηγούνται από το κώδικα και τα διαγράμματα ροής και κρίνουμε ότι δεν χρήζουν επεξηγήσεως στην παρούσα αναφορά και θεωρούμε ότι θα τα εξηγήσουμε στην εξέταση. Παρακάτω δίνεται το διάγραμμα ροής για το check(),πιο κάτω για το πρόγραμμα

μας με την έτοιμη πλέον check() και αμέσως κάτω ο συνολικός κώδικας: (Προφανώς έγιναν χρήση των έτοιμων συναρτήσεων της εκφώνησης αυτής ή ενδεχομένως προηγούμενων οι οποίες προκειμένου να είναι απλά τα διαγράμματα ροής ,παραλείφθηκαν από αυτά και θεωρήθηκαν έτοιμα.)




ΑΡΧΗ
uint8_t input;
uint_t savor;
bool rejected;
rejected <- false
input <- 0x00
DDRB <- 0xff
DDRC <- 0xff
DDRD <- 0xff
PORTB <- 0x00
twi_init();
PCA9555_0_write(REG_OUTPUT_0,0x00);
PCA9555_0_write(REG_OUTPUT_1,0x00);
PORTD -<0x00
rejected <- false
input <- check();
NAI
input = 0xff?
OXI
PORTD <- 0x0F
savor <-input
_delay_ms(15)
input <- check();
NAI
input = savor?
OXI
savor != 0xD7?
NAI
rejected <- true
OXI
input <- check();
NAI
input = 0xff?
OXI
savor <- input
_delay_ms(15)
input <- check();
NAI
input = savor?
OXI
savor != 0xB7?
NAI
rejected <- true
OXI
PORTD <- 0xff
rejected = true?
NAI
OXI
PORTB <- 0x3F
_delay_ms(3000)
PORTB <- 0x00
_delay_ms(2000)
int i = 0 ;
PORTB <- 0x3F
_delay_ms(500)
PORTB <- 0x00
_delay_ms(500)
i <- i+1
NAI
i <= 4?
NAI
1=1?
OXI
OXI
ΤΕΛΟΣ

```
#define F_CPU 16000000UL  // 16 MHz

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdbool.h>

#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
#define TWI_READ 1 // reading from twi device
#define TWI_WRITE 0 // writing to twi device
#define SCL_CLOCK 100000L // twi clock in Hz
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2
// PCA9555 REGISTERS
typedef enum {
 REG_INPUT_0 = 0,
 REG_INPUT_1 = 1,
 REG_OUTPUT_0 = 2,
 REG_OUTPUT_1 = 3,
 REG_POLARITY_INV_0 = 4,
 REG_POLARITY_INV_1 = 5,
 REG_CONFIGURATION_0 = 6,
 REG_CONFIGURATION_1 = 7
} PCA9555_REGISTERS;
//----------- Master Transmitter/Receiver --------------------
#define TW_START 0x08
#define TW_REP_START 0x10
//---------------- Master Transmitter ----------------------
#define TW_MT_SLA_ACK 0x18
#define TW_MT_SLA_NACK 0x20
#define TW_MT_DATA_ACK 0x28
//---------------- Master Receiver ----------------
#define TW_MR_SLA_ACK 0x40
#define TW_MR_SLA_NACK 0x48
#define TW_MR_DATA_NACK 0x58

#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)


//initialize TWI clock
void twi_init(void)
{
 TWSR0 = 0; // PRESCALER_VALUE=1
 TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}

// Read one byte from the twi device (request more data from device)
```

```
unsigned char twi_readAck(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
while(!(TWCR0 & (1<<TWINT)));
 return TWDR0;
}
//Read one byte from the twi device, read is followed by a stop condition
unsigned char twi_readNak(void)
{
TWCR0 = (1<<TWINT) | (1<<TWEN);
while(!(TWCR0 & (1<<TWINT)));
 return TWDR0;
}

// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address)
{
 uint8_t twi_status;
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));
// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
// send device address
TWDR0 = address;
TWCR0 = (1<<TWINT) | (1<<TWEN);
// wail until transmission completed and ACK/NACK has been received
while(!(TWCR0 & (1<<TWINT)));

// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
 {
 return 1;
 }
return 0;
}

// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
 uint8_t twi_status;
 while ( 1 )
 {
// send START condition
```

```
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

 // wait until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;

 // send device address
 TWDR0 = address;
 TWCR0 = (1<<TWINT) | (1<<TWEN);

 // wail until transmission completed
 while(!(TWCR0 & (1<<TWINT)));

 // check value of TWI Status Register.
 twi_status = TW_STATUS & 0xF8;
 if ( (twi_status == TW_MT_SLA_NACK )||(twi_status ==TW_MR_DATA_NACK) )
 {
 /* device busy, send stop condition to terminate write operation */
 TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

 // wait until stop condition is executed and bus released
 while(TWCR0 & (1<<TWSTO));

 continue;
 }
break;
 }
}

// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write( unsigned char data )
{
// send data to the previously addressed device
TWDR0 = data;
TWCR0 = (1<<TWINT) | (1<<TWEN);
// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));
if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
return 0;
}
// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
// 1 failed to access device
unsigned char twi_rep_start(unsigned char address)
{
 return twi_start( address );
```

```
}

// Terminates the data transfer and releases the twi bus
void twi_stop(void)
{
 // send stop condition
TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
// wait until stop condition is executed and bus released
while(TWCR0 & (1<<TWSTO));
}

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_write(value);
 twi_stop();
}

uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
{
uint8_t ret_val;

 twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
 twi_write(reg);
 twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
 ret_val = twi_readNak();
 twi_stop();


 return ret_val;
}

uint8_t check(){

   uint8_t stili,grammi;
   //check which stili is pressed

   PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
   stili = PCA9555_0_read(REG_INPUT_1);
   if(stili == 0xF0) return 0xFF;


   PCA9555_0_write(REG_CONFIGURATION_1, 0x0F);

   grammi = PCA9555_0_read(REG_INPUT_1);
   if (grammi == 0x0F) return 0xFF;

   return ((grammi|0xF0) & (stili|0x0F));
```

```
}

int main(){

  uint8_t input = 0x00;
  uint8_t savor;
  bool rejected = false;

  DDRB = 0xff;
  DDRC = 0xff;
  DDRD = 0xff;

  PORTB = 0x00;

  twi_init();

  PCA9555_0_write(REG_OUTPUT_0,0x00);
  PCA9555_0_write(REG_OUTPUT_1,0x00);

  while(1){
    PORTD = 0x00;
    rejected = false;
    input = check();

    if (input == 0xFF) continue;

    //first digit
    else{

      PORTD = 0x0F;//recognised first
      savor = input;
      while(1){        //????????????
        _delay_ms(15);
        input = check();
        //PORTD = input;
        if (input==savor) continue;
        else  break;
      }
      //PORTD = savor;
      if(savor != 0xD7) rejected = true;  //if first digit is not 2 it is surely wrong passcode

      //second - we have to wait even if first digit is wrong
      while(1){
        input = check() ;
        if (input == 0xFF) continue;
        else{
          savor = input;
          while(1){        //????????????
            _delay_ms(15);
```

```
            //PORTD = input;
            input = check();
            if (input==savor) continue;
            else  break;
               }


          //PORTD = savor;
          if (savor!= 0xB7) rejected = true; //checks if second digit is different from correct 3

          break;

        }
      }


    }
    PORTD = 0xFF;//2 digits pressed
    if(rejected == true) {
      for(int i = 0 ; i<=4; i++){
        PORTB = 0x3F;
        _delay_ms(500);
        PORTB = 0x00;
        _delay_ms(500);

      }

      continue;
    }

    else{
      PORTB = 0x3F;
      _delay_ms(3000);
      PORTB = 0x00;
      _delay_ms(2000);
      continue;
    }


  }

}
```