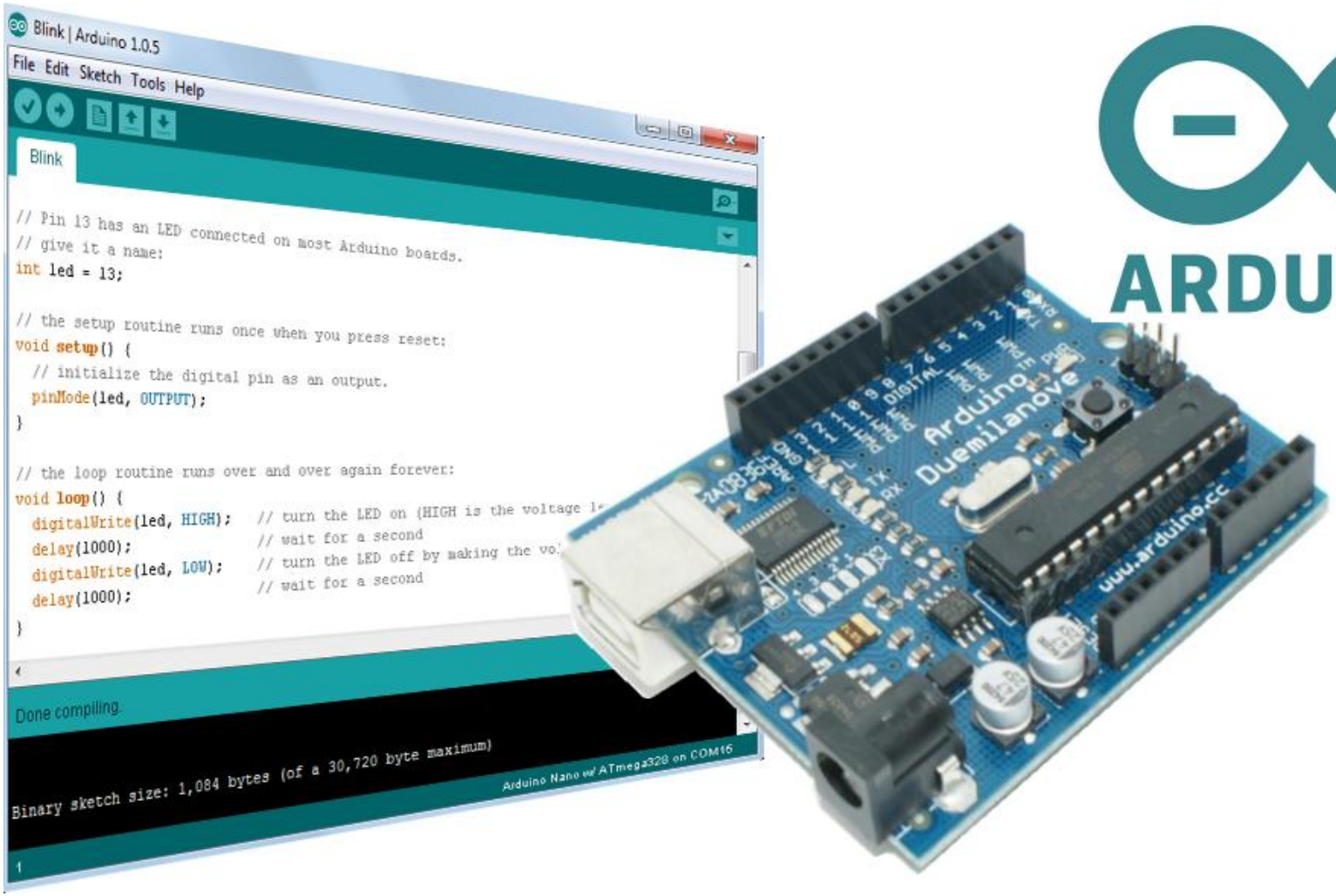


Bevezetés a mikrovezérlők programozásába:

Kapcsolók, kapcsolósorok és -mátrixok

Arduino projektek

- **dipswitch** – két kapcsolóval négy fokozatban szabályozzuk a LED fényerejét.
- **_4x4_keypad_baremetal** – 4x4 billentyűmátrix kezelése támogatói könyvtár nélkül
- **MultiKey** – 4x4 billentyűmátrix kezelése a **Keypad** programkönyvtár felhasználásával
- **EventKeypad** – a 4x4 billentyűmátrix kezelése a **Keypad** programkönyvtár felhasználásával, s az események felhasználása egy LED vezérlésére.
- **Libraries/Keypad** – Arduino programkönyvtár billentyűmátrix kezelésére

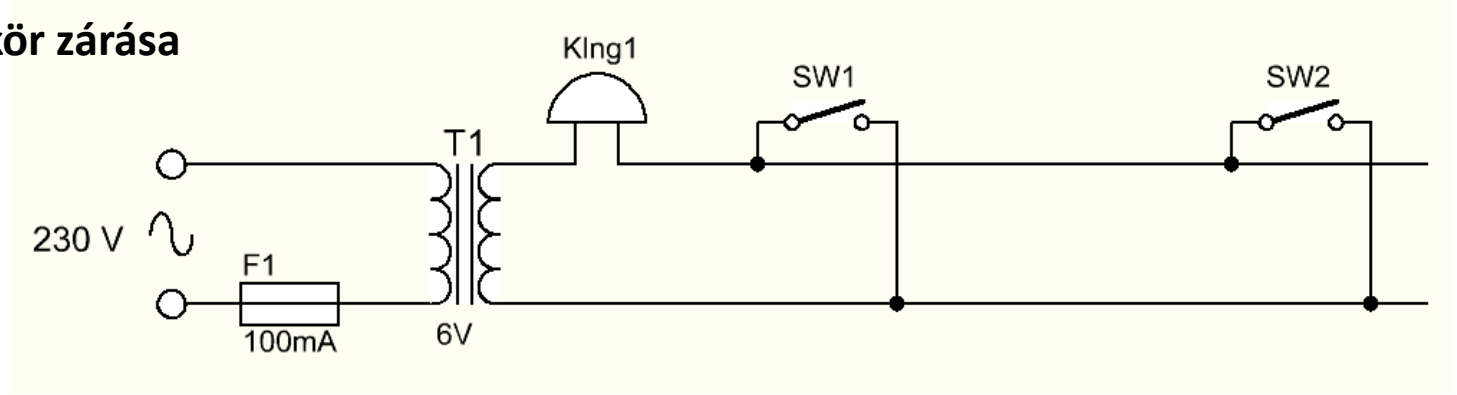
Megjegyzés: A honlapról letölthető **Arduino_Lab26.zip** állományt az alábbi mappába bontsuk ki: c:\Fehasználók\<usernév>\Dokumentumok\Arduino

Fontos, hogy a kibontott csomagban található *libraries* almappa tartalmát az alábbi mappába mozgassuk át: c:\Fehasználók\<usernév> \Dokumentumok\Arduino\libraries

Kapcsolók és nyomógombok

A kapcsolók és a nyomógombok a legegyszerűbb eszközei a felhasználói beavatkozásnak.

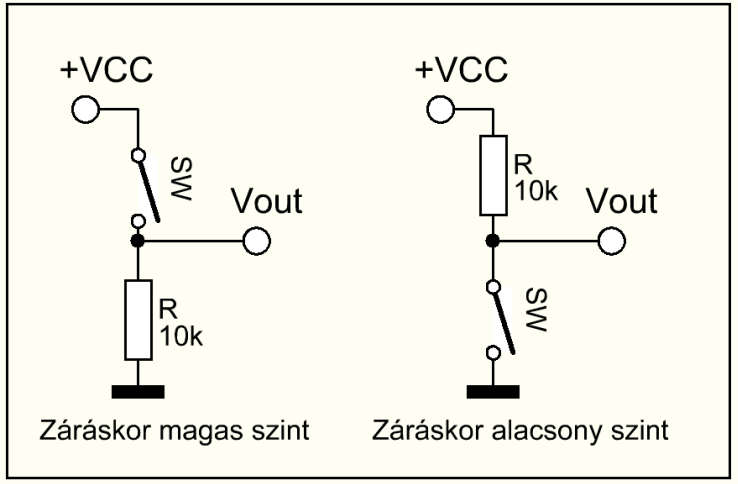
1. **Áramkör zárása**



- Villanycsengő, vagy lépcsőházi világítás: nyomógombok párhuzamosan kapcsolva
- Présgép vagy vágógép „kétkezes” indítógombja: nyomógombok **sorbakapcsolva!**

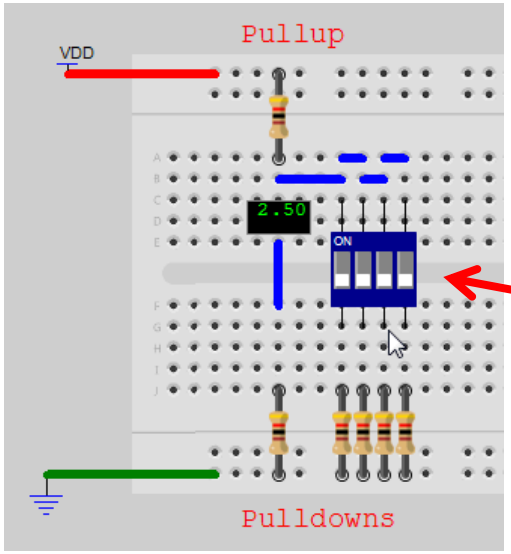
2. **Logikai szint beállítása**

- **Logikai bemenet aktiválása** (pl. mikrovezérlő alaphelyzetbe állítása, dallamcsengő indítása, stb)
- **Választójel** (pl. mikrovezérlő bootloader vagy programfuttatás mód)



Kapcsolósorok

- Adatbevitel: kapcsolóregiszter
- Konfigurálás: címkiválasztás, IRQ csatorna választás (ISA buszos kártyák)
- Paraméterek beállítása: osztási arány, erősítés, hangerő , stb.

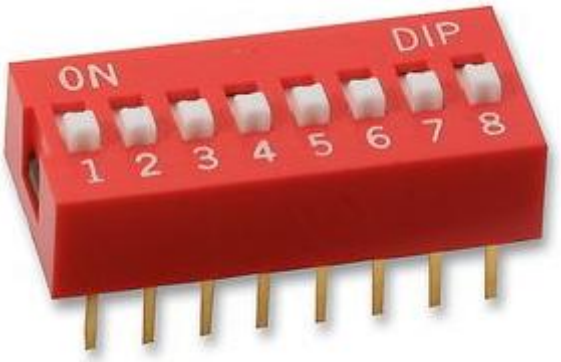


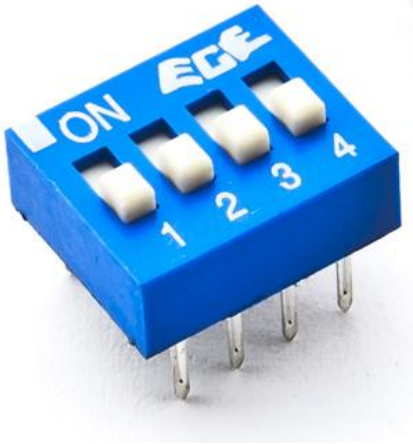
Címválasztó kapcsolósor

Feszültségosztó osztási arányának változtatása.

DIP kapcsolósorok

DIP = Dual in-line package, azaz két sorban helyezkednek el a kivezetések.
A DIP tokozású IC-khez hasonlóan a lábsorok távolsága itt is 0.3 inch (7.62 mm), a lábak távolsága pedig 0.1 inch (2.54 mm).
A szemben levő lábak páranként egy-egy kapcsoló kivezetései.
Szabadalmi bejelentés kelte: 1974, elfogadva 1976-ban.





Forgókapcsoló (Rotary DIP switch) és kódkapcsoló

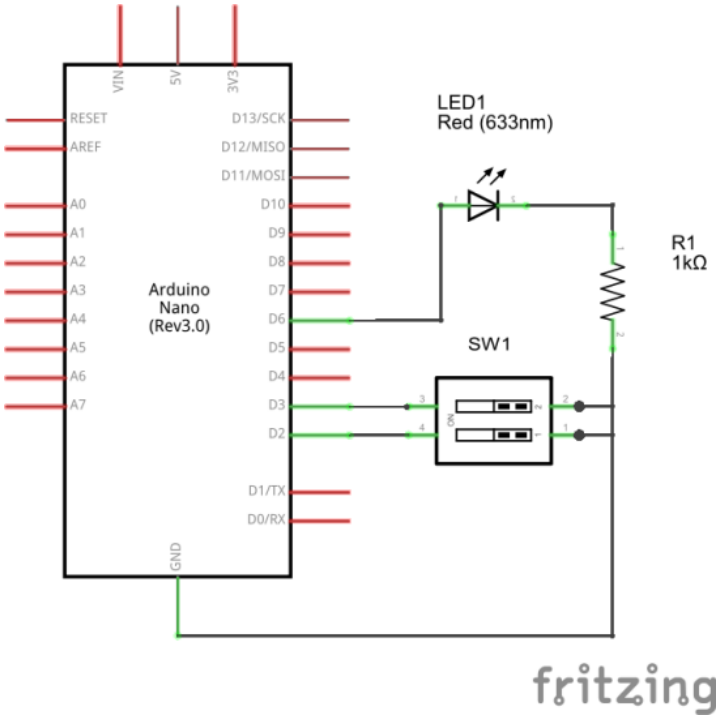
A forgókapcsoló egyetlen kontaktust kapcsol egy kiválasztható kivezetéshez. A képen látható kódkapcsoló pedig már négybites kód formájában adja meg a beállított számot.

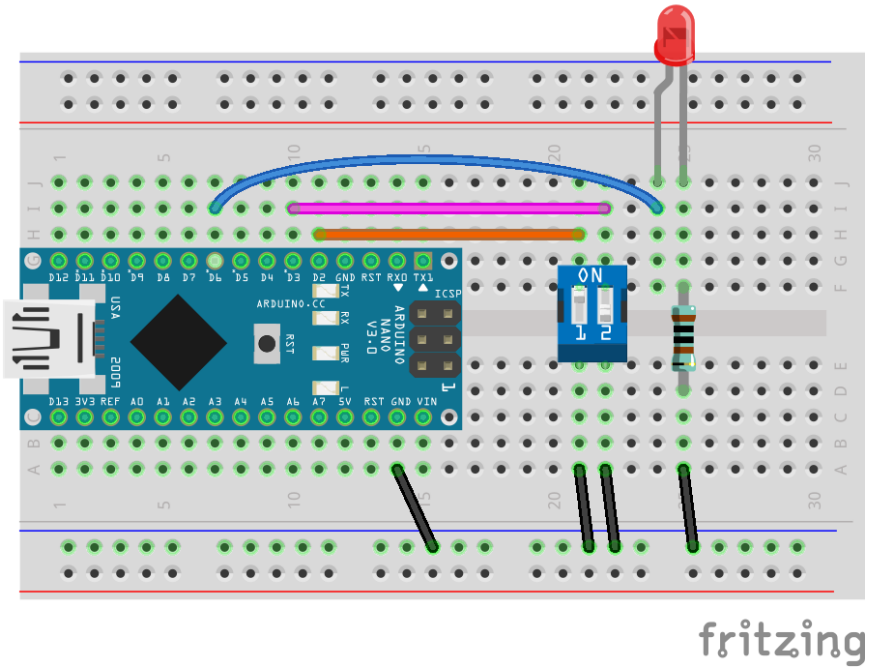
BCD

C	1	2	4	8	
●					0
●	●				1
●		●			2
●	●	●			3
●			●		4
●	●		●		5
●		●	●		6
●	●	●	●		7
●				●	8
●	●			●	9

Arduino vezérlése kapcsolókkal

Az alábbi egyszerű mintapéldában a belső felhúzásra állított D2 és D3 bemenetek egy-egy kapcsolóval lehúzhatók, s vele például a D6 kimenetre kötött LED fényereje vagy villogási frekvenciája négy fokozatban beállítható.





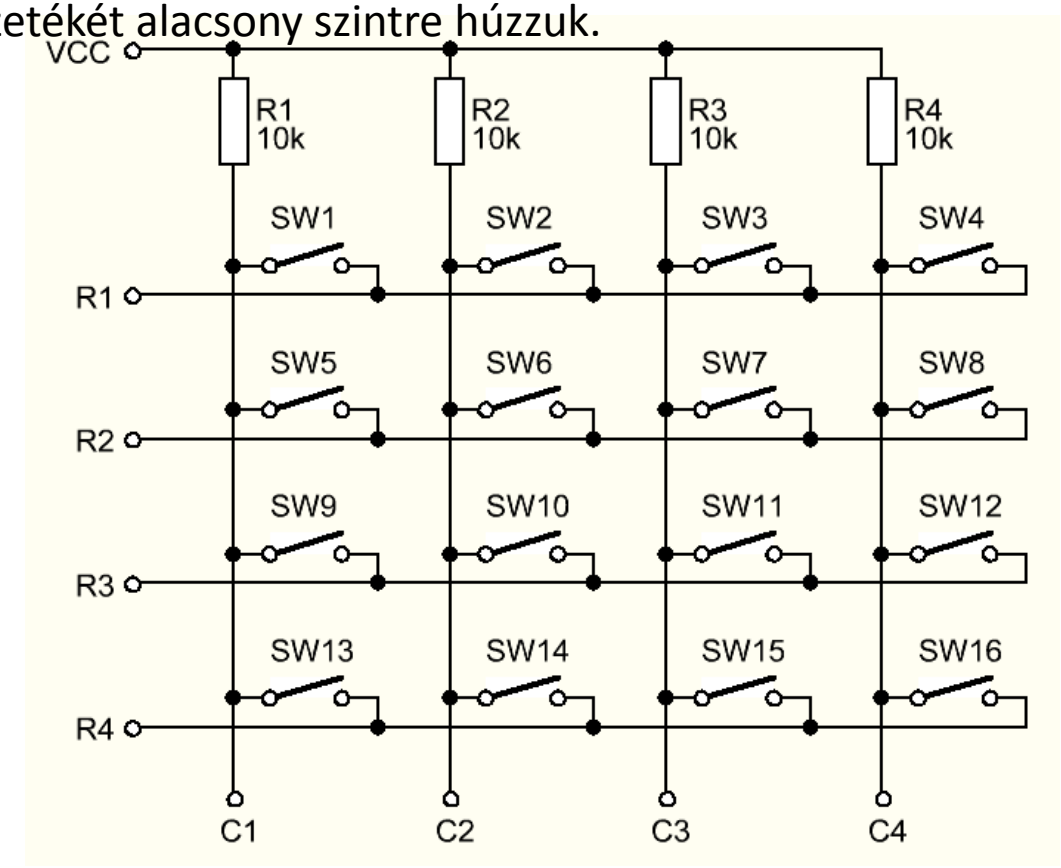
dipswitch.ino

A kapcsolók állása szerint négy fokozatban szabályozzuk a LED fényerejét. A LED fényerejét a D6 kimenet, mint PWM csatorna, kitöltési tényezőjének változtatásával szabályozzuk. A kitöltést nem lineárisan, hanem mértani haladvány szerint csökkentjük/növeljük, mivel a szemünk sem lineárisan érzékel (a kétszer nagyobb fényteljesítményt nem érezzük kétszer fényesebbnek). Számolgatás helyett az előre kiszámolt értékeket egy tömbben tároljuk („függvénytáblázat”).

```
uint8_t duty[] = {4,16,64,255 }; //exponenciálisan növekvő kitöltés

void setup() {
  pinMode(2, INPUT_PULLUP);       //digitális bemenet felhúzással
  pinMode(3, INPUT_PULLUP);       //digitális bemenet felhúzással
  analogWrite(6,duty[0]);         //Kezdeti fényerő
}

void loop() {
  int keys = digitalRead(2) +
             2*digitalRead(3);    //kapcsolók állásának beolvasása
  analogWrite(6,duty[keys]);      //LED fényerő beállítása
  delay(100);                     //Várunk egy kicsit...
}
```

Nyomógomb mátrix

Az oszlopvonalakat felhúzzuk egy-egy ellenálláson keresztül (a külső felhúzás helyett használhatunk belső felhúzást is). Ha a sorvonalakat egyesével alacsony szintre húzzuk, az oszlopvonalakon észlelhető a gomblenyomás: a lenyomott gombhoz tartozó oszlopvonalon alacsony szint jelenik meg, amikor a megfelelő sor vezetékét alacsony szintre húzzuk.



Bekötés az Arduinohoz

Bekötési vázlat:

D2 = row 0. (1,2,3,A)
D3 = row 1. (4,5,6,B)
D4 = row 2. (7,8,9,C)
D5 = row3. (*,0,#,D)

D6 = col 0. (1,4,7,*)
D7 = col 1. (2,5,8,0)
D8 = col 2. (3,6,9, #)
D9 = col 3. (A,B,C,D)

_4x4_keypad_baremetal.ino 3/1

```
const byte rnum=4;                          //sorok száma
const byte cnum=4;                          //oszlopok száma
const byte rows[rnum]={2,3,4,5};            //a sorvezetékekhez rendelt kimenetek
const byte cols[cnum]={6,7,8,9};            //az oszlopvezetékekhez rendelt bemenetek
const char keyTable[rnum][cnum] = {         //Nyomógombokhoz rendelt kódok
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}};
//--- Logikai kifejezés, igaz, ha egy gomb le van nyomva és a sora le van húzva
#define KEY_PRESSED() ((!digitalRead(cols[0])) || (!digitalRead(cols[1])) ||
                       (!digitalRead(cols[2])) || (!digitalRead(cols[3])))
//--- Minden sorvezeték alacsony szintre állítása -----
void allRowLow() {
  for(byte i=0; i<rnum; i++) {
    digitalWrite(rows[i],LOW);  //Minden sorvezeték legyen alacsony szintre húzva
  }
}
//--- Egy kiválasztott sorvezeték alacsony szintre állítása -----
void oneRowLow(byte n) {
  for(byte i=0; i<rnum; i++) {
    if(i == n) digitalWrite(rows[i],LOW);
    else digitalWrite(rows[i],HIGH);
  }
}
```

Folytatás a következő oldalon...

_4x4_keypad_baremetal.ino 3/2

```
//--- Gomblenyomás figyelése és feldolgozása -------
char keyscan() {
  byte xcol, xrow;                        //A lenyomott gomb koordinátái

  //-- A lenyomott gombhoz tartozó oszlop meghatározása
  if (!digitalRead(cols[0])) xcol = 0;
  else if (!digitalRead(cols[1])) xcol = 1;
  else if (!digitalRead(cols[2])) xcol = 2;
  else if (!digitalRead(cols[3])) xcol = 3;
  else return(0);                         //Kilépés: nem történt gomblenyomás

  //-- A lenyomott gombhoz tartozó sor meghatározása
  for (byte xrow=0; xrow < rnum; xrow++) {
    oneRowLow(xrow);                      //egy sorvezeték lehúzása
    if (KEY_PRESSED()) {
      allRowLow();                        //minden sor lehúzása
      return(keyTable[xrow][xcol]);       //Visszatérés az azonosított gomb kódjával
    }
  }
  allRowLow();                            //minden sor lehúzása
  return(0);                              //Kilépés: sikertelen gomblenyomás azonosítás
}
```

Folytatás a következő oldalon...

_4x4_keypad_baremetal.ino 3/3

```
void setup() {
  for(byte i=0; i<rnum; i++) {
    pinMode(rows[i],OUTPUT);          //Minden sorvezeték: digitális kimenet
    digitalWrite(rows[i],LOW);        //Minden sorvezeték alacsony szintre húzva
  }
  for(byte i=0; i<cnum; i++) {
    pinMode(cols[i],INPUT_PULLUP);    //Minden oszlopvezeték: bemenet felhúzással
  }
  Serial.begin(9600);                 //UART kimenet inicializálása
}

void loop() {
  delay(20);
  char key = keyscan();
  if(key) {
    if(key == '#') Serial.println(key);
    else Serial.print(key);
    delay(200);
  }
}
```

Megjegyzések:

- A program a többes gombnyomásokat nem jól kezeli, hibás eredményt ad!
- A gomb hosszantartó (t > 220 ms) lenyomásakor a kód ismétlődik.
- A # gomb lenyomásakor új sort kezdünk.

Keypad programkönyvtár

Mark Stanley, *Alexander Brevig*: ***Keypad library*** ***(***https://github.com/Chris--A/Keypad***)***

Leírás: http://playground.arduino.cc/Code/Keypad

A legfontosabb API függvények és publikus változók:

Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS) – A konstruktor függvény. Lehetővé teszi a méret, a lábkiosztás és a billentyűkódok megadását

bool **getKeys()** – Frissíti az aktív gombok listáját (max .10 db) tér vissza. TRUE értékkel tér vissza, ha bármelyik gong állapota megváltozott.

char **getKey** (void) – csak egy billentyűlenyomást ad vissza (többszörös lenyomásnál is)

bool **stateChanged** – a hozzá tartozó gomb állapotváltozását jelzi.

Keystate **kstate** – a hozzá tartozó gomb állapotát jelzi (IDLE, PRESSED, HOLD, RELEASED)

void **addEventListener**(void(*listener)(char)) – visszahívási függvény megadása a billentyűzet-események kezelésére

MultiKey.ino 2/1

```
#include <Keypad.h>

const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}};

byte rowPins[ROWS] = {2,3,4,5}; //connect to the row pinouts of the kpd
byte colPins[COLS] = {6,7,8,9}; //connect to the column pinouts of the kpd

Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

String msg;

void setup() {
  Serial.begin(9600);
  msg = "";
}
```

Mark Stanley, Alexander Brevig: ***Keypad library***
https://github.com/Chris--A/Keypad
Mintaprogram: **MultiKey**
Max. 10 gomb együttes lenyomását is kezeli!

A felhasznált Arduino kivezetések

Folytatás a következő oldalon...

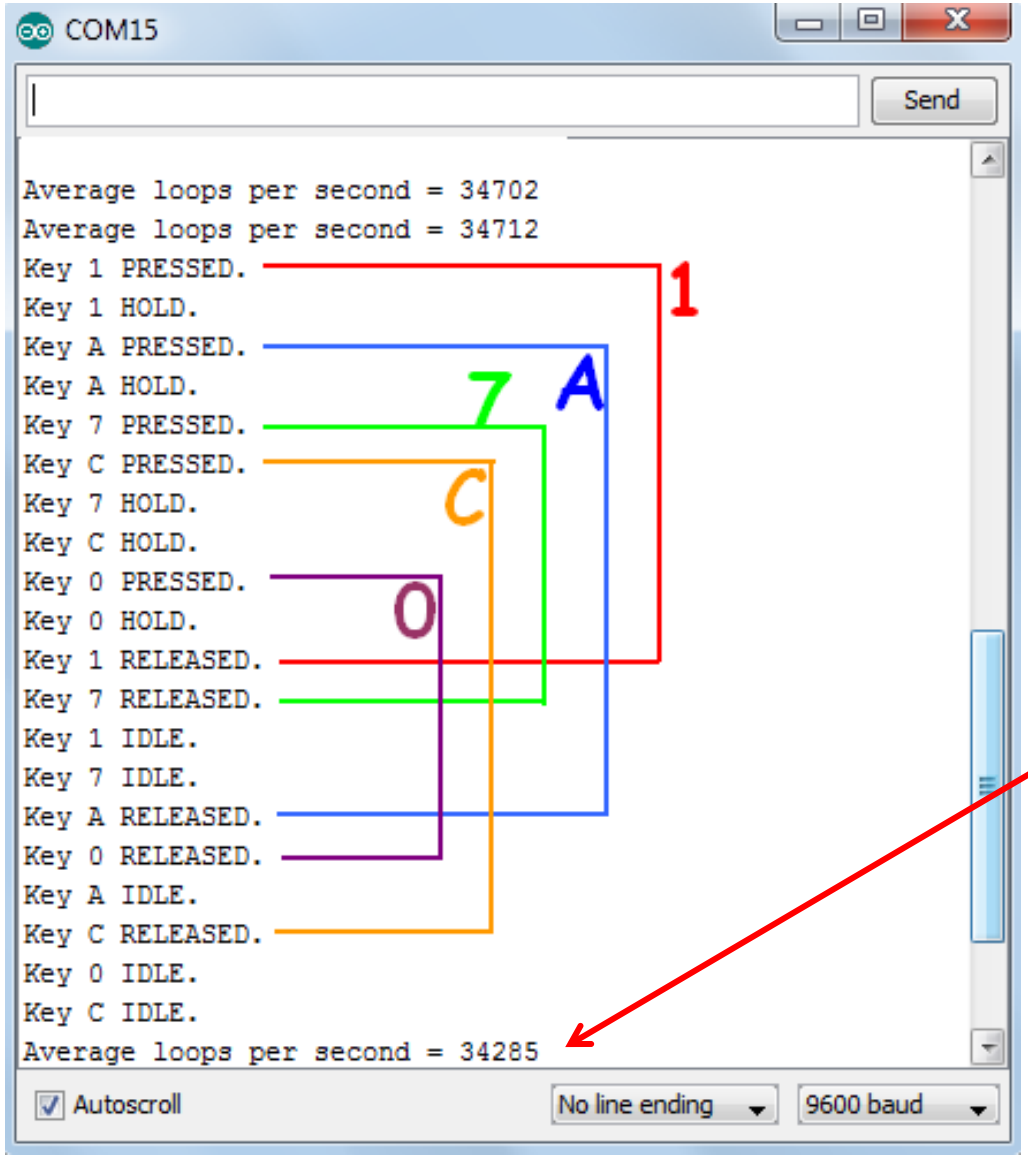
MultiKey.ino 2/2

```
void loop() {
  // Fills kpd.key[ ] array with up-to 10 active keys.
  if (kpd.getKeys()) {                    //true if there are ANY active keys.
    for (int i=0; i<LIST_MAX; i++) {      //Scan the whole key list.
      if ( kpd.key[i].stateChanged ) {    //find keys that have changed state.
        switch (kpd.key[i].kstate) {      //report active key states
          case PRESSED:
                msg = " PRESSED.";
                break;
          case HOLD:
                msg = " HOLD.";
                break;
          case RELEASED:
                msg = " RELEASED.";
                break;
          case IDLE:
                msg = " IDLE.";
        }
        Serial.print("Key ");
        Serial.print(kpd.key[i].kchar);
        Serial.println(msg);
      }
    }
  }
}
```

Az aktív gomboknak négy lehetséges állapota van: PRESSED, HOLD, RELEASED, IDLE. Ezeket az állapotváltozásokat követjük nyomon.

A program mellékesen a vizsgálati ciklusokat is számlálja és kiírja. Ennek részleteit a mellékelt listából kihagytuk.

Multikey.ino futási eredmény



Az ábrán egy ötszörös lenyomás eredménye látható: 1, A, 7, C, 0

A felengedés sorrendje nem egyezik meg a lenyomási sorrenddel!

A program mellékesen a másodpercenkénti billentyűzet-vizsgálatok számát is regisztrálja.

EventKeypad 2/1

Egy gyakorlatiasabb mintapélda: LED vezérlése nyomógomb események felhasználásával.

```
#include <Keypad.h>
const byte ROWS = 4;                          //nyomógomb sorok száma
const byte COLS = 4;                          //nyomógomb oszlopok száma
char keys[ROWS][COLS] = {{'1','2','3','A'}, {'4','5','6','B'},
                         {'7','8','9','C'}, {'*','0','#','D'}};
byte rowPins[ROWS] = {2,3,4,5};               //lábkiosztás a sorvezetékekhez
byte colPins[COLS] = {6,7,8,9};               //lábkiosztás az oszlopvezetékekhez
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
byte ledPin = 13;                             // D13-ra van kötve a LED
boolean blink = false;
boolean ledPin_state;

void setup(){
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);                    // ledpin legyen digitális kimenet
  digitalWrite(ledPin, HIGH);                 // bekapcsoljuk a LED-et
  ledPin_state = digitalRead(ledPin);         // eltároljuk a LED kiinduló állapotát
  keypad.addEventListener(keypadEvent); // eseménykiszolgáló hozzárendelése
}

void loop(){
  char key = keypad.getKey();                 // gomblenyomás figyelése
  if (key) Serial.println(key);               // kiíratjuk, ha volt lenyomott gomb
  if (blink){                                 // LED villogtatás (ha kell)
    digitalWrite(ledPin,!digitalRead(ledPin)); // LED állapotváltás
    delay(100);                               // 100 ms várakozás
  }
}
```

Billentyűzet definiálása

LED és LED állapot definiálása

EventKeypad 2/2

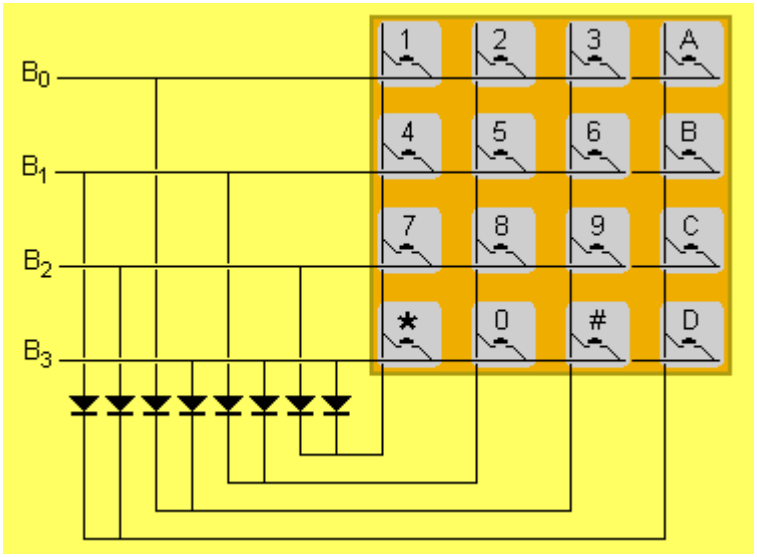
```
//--- Billentyűzet eseményeinek feldolgozása ---------
void keypadEvent(KeypadEvent key){
  switch (keypad.getState()){                    // az esemény vizsgálata
   case PRESSED:                                 // ha gomblenyomás történt
     if (key == '#') {                           // Ha ez a # gomb:
        digitalWrite(ledPin,!digitalRead(ledPin)); // LED ellenkező állapotba!
        ledPin_state = digitalRead(ledPin);        // megjegyzi a LED állapotát
     }
     break;

   case RELEASED:                                // ha felengedés történt,
     if (key == '*') {                           // és ez a * gomb volt:
       digitalWrite(ledPin,ledPin_state); // a LED eredeti állapotba!
       blink = false;                           // a villogtatást letiltjuk
     }
     break;

   case HOLD:                                    // ha lenyomva tartás történt,
     if (key == '*') {                           // és ez a * gomb, akkor:
       blink = true;                            // a villogtatást engedélyezzük
     }
     break;
  }
}
```

Nyomógomb-mátrix 4 vonalon

Forrás: Talking electronics – tippek és trükkök a bemenetekkel
Link: www.talkingelectronics.com/pay/PIC/PIC-Page21.html

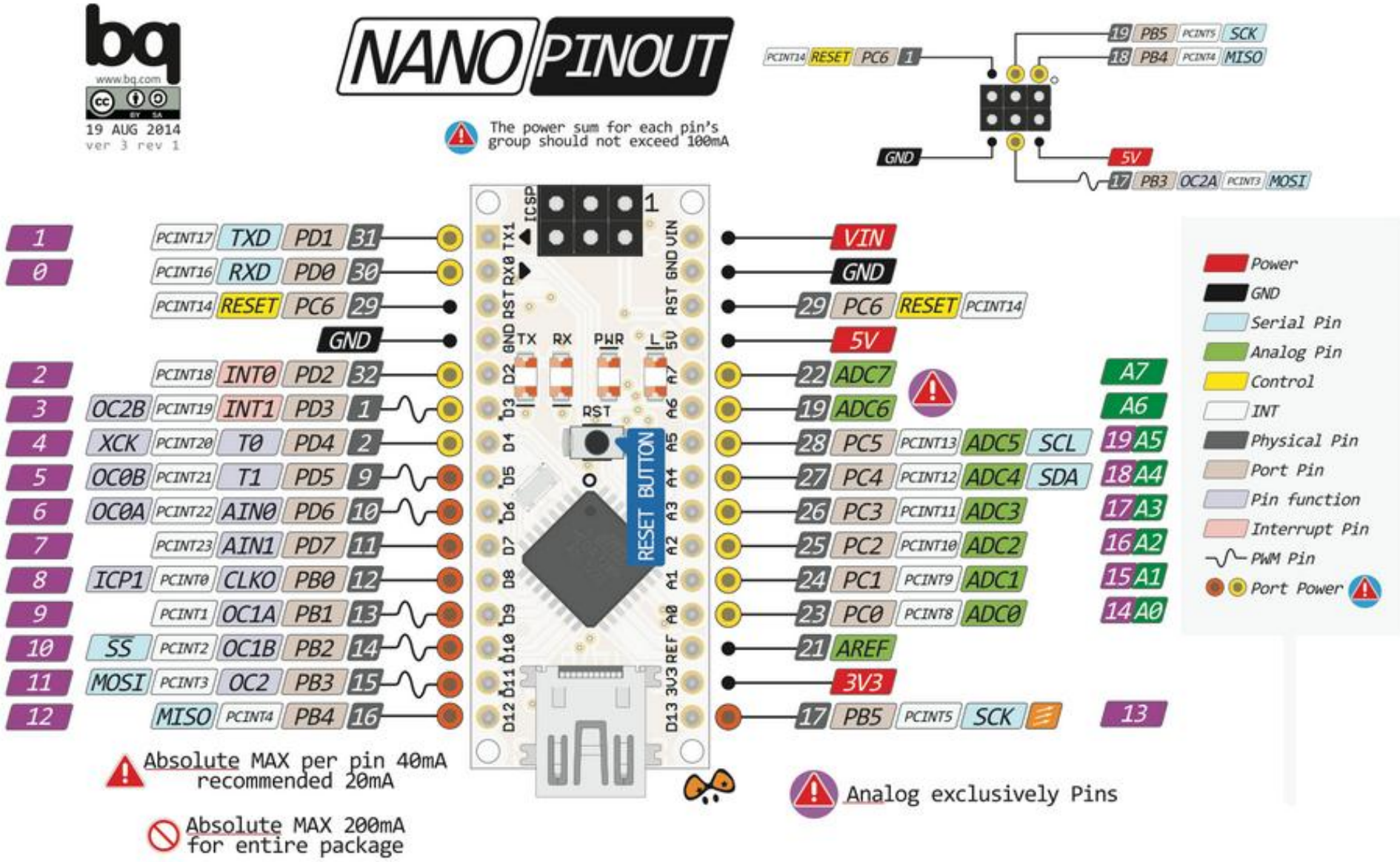
Csak érdekesség gyanánt: néhány dióda beépítése és némi bonyodalom árán kevesebb adatvonalon is megoldható a nyomógomb-mátrix kezelése:

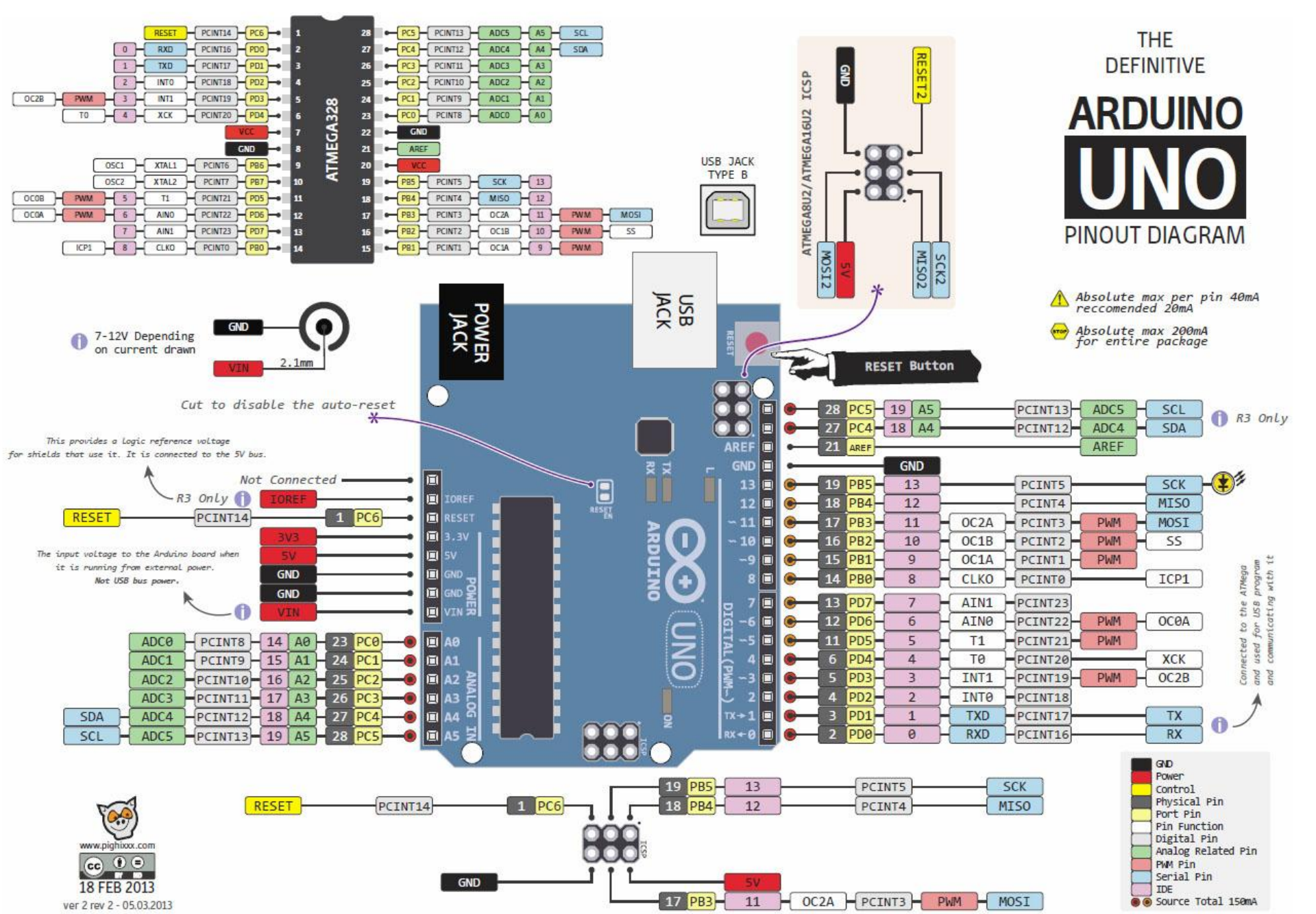
- A B0, B1, B2, B3 vonalakat egyenként alacsony szintű kimenetnek állítjuk, s közben megvizsgáljuk a többi három, gyenge belső felhúzású bemenetnek állított vonal állapotát. Ahogy az alábbi táblázatban láthatjuk, minden gomblenyomás egyedi, tehát azonosításra alkalmas kódot ad.



SETTING:				READING:				
B3	B2	B1	B0	B3	B2	B1	B0	KEY
INPUT (weak Pull-up)	INPUT (weak Pull-up)	INPUT (weak Pull-up)	OUTPUT LOW	0	0	1	0	1
				0	1	0	0	2
				0	1	1	0	3
				1	0	0	0	A
INPUT (weak Pull-up)	INPUT (weak Pull-up)	OUTPUT LOW	INPUT (weak Pull-up)	0	0	0	1	4
				0	1	0	1	5
				0	1	0	0	6
				1	0	0	1	B
INPUT (weak Pull-up)	OUTPUT LOW	INPUT (weak Pull-up)	INPUT (weak Pull-up)	0	0	1	1	7
				0	0	0	1	8
				0	0	1	0	9
				1	0	0	1	C
OUTPUT LOW	INPUT (weak Pull-up)	INPUT (weak Pull-up)	INPUT (weak Pull-up)	0	1	0	1	*
				0	0	1	1	0
				0	1	1	0	#
				0	0	0	1	D

Emlékeztető: Arduino nano v3.0




THE DEFINITIVE
ARDUINO UNO
PINOUT DIAGRAM
ATMEGA328
ATMEGA8U2/ATMEGA16U2 ICSP
USB JACK TYPE B
POWER JACK
USB JACK
RESET Button
7-12V Depending on current drawn
2.1mm
Cut to disable the auto-reset
This provides a Logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
The input voltage to the Arduino board when it is running from external power. Not USB bus power.
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
Connected to the ATMega and used for USB program and communicating with it
www.pighixxx.com
18 FEB 2013
ver 2 rev 2 - 05.03.2013
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA

Mikrovezérlő programozás

ARM® mbed™ környezetben

26. Kapcsolók, kapcsolósorok és -mátrixok

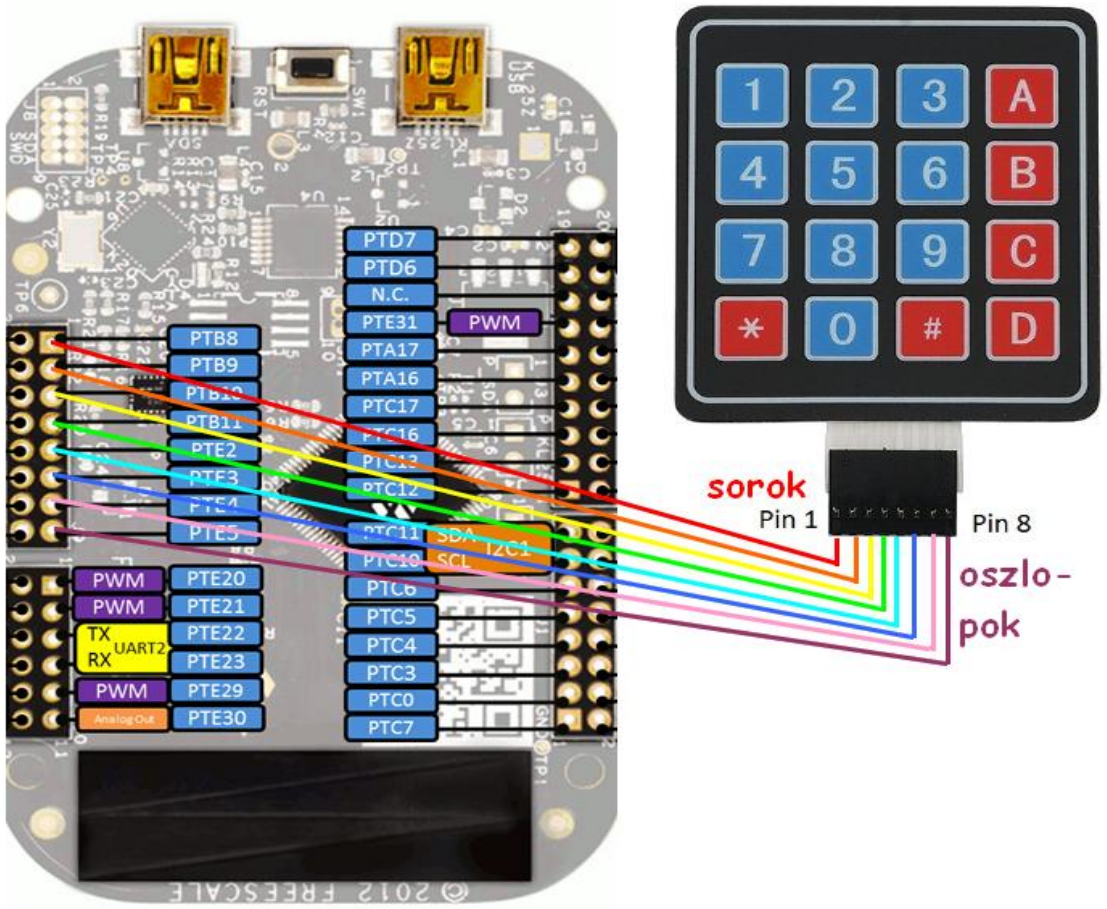
Billentyűmátrix mbed környezetben

4x4 billentyűmátrix kezeléséhez az mbed honlapján a **Components/other** szekcióban **Hotboards keypad** néven találunk programkönyvtárat és mintaprogramokat.
A programkönyvtár lényegében Alexander Brevig : **Keypad** nevű Arduino programkönyvtárának adaptációja, így az Arduino mintaprogramoknál elmondottak itt is felhasználhatók.

Bekötési vázlat:

PTB8 = row 0. (1,2,3,A)
PTB9 = row 1. (4,5,6,B)
PTB10 = row 2. (7,8,9,C)
PTB11 = row3. (*,0,#,D)

PTE2 = col 0. (1,4,7,*)
PTE3 = col 1. (2,5,8,0)
PTE4 = col 2. (3,6,9, #)
PTE5 = col 3. (A,B,C,D)



A Hotboards_keypad programkönyvtár

Keypad(char *userKeymap, DigitalInOut *row, DigitalInOut *col, uint8_t numRows, uint8_t numCols) – A konstruktor függvény. Lehetővé teszi a méret, a lábkiosztás és a billentyűkódok megadását

char **getKey** (void) – csak egy billentyűlenyomást ad vissza (többszörös lenyomásnál is)

bool **getKeys** (void) – az aktív gombok (max. 10) listájának frissítése

KeyState **getState** (void) – az aktív gombok listáján a legelső gomb státusza

void **begin**(char *userKeymap) – a billentyűkódok újradefiniálása (a konstruktornál megadott méretben!)

bool **isPressed** (char keyChar) – igaz értékkel tér vissza, ha a kiválasztott gomb volt lenyomva

void **setDebounceTime**(uint) – pergésmentesítési idő megadása (legalább 1 ms legyen!)

void **setHoldTime**(uint) – a Hold (lenyomva tartott) állapotba lépéshez szükséges idő megadása.

void **addEventListener**(void(*listener)(char)) – visszahívási függvény megadása a billentyűzet-események kezelésére

int **findInList** (char keyChar) – megadott karakter keresése az aktív gombok listájában

int **findInList** (int keyCode) – keresés az aktív gombok listájában billentyűkód alapján

char **waitForKey**(void) – gomblenyomásra vár (blokkoló várakozás!)

bool **keyStateChanged**(void) – megadja, hogy történt-e állapotváltozás, az aktív gombok listájának első eleménél.

uint8_t **numKeys** (void) – az aktív gombok listán szereplő gombok száma

Hotboards_MultiKey 2/1

```
#include "mbed.h"
#include "Hotboards_keypad.h"
#include <string>
using std::string;

char keys[ 4 ][ 4 ] = {
    { '1' , '2' , '3' , 'A' },
    { '4' , '5' , '6' , 'B' },
    { '7' , '8' , '9' , 'C' },
    { '*' , '0' , '#' , 'D' }
};

// Defines the pins connected to the rows
DigitalInOut rowPins[ 4 ] = { PTB8 , PTB9 , PTB10 , PTB11 };
// Defines the pins connected to the cols
DigitalInOut colPins[ 4 ] = { PTE2 , PTE3 , PTE4 , PTE5 };

// Creates a new keyboard with the values entered before
Keypad kpd( makeKeymap( keys ) , rowPins , colPins , 4 , 4 );

// Configures the serial port
Serial pc( USBTX , USBRX );

int i;
```

Hotboards_MultiKey 2/2

```
int main() {
  string msg;
  while(1) {
     // Fills kpd.key[ ] array with up-to 10 active keys.
     // Returns true if there are ANY active keys.
     if( kpd.getKeys( ) ) {
       for( i = 0 ; i < LIST_MAX ; i++ ) { // Scan the whole key list.
         if( kpd.key[ i ].stateChanged ) { // keys that have changed state.
            switch( kpd.key[ i ].kstate ){
              case PRESSED:  msg = " PRESSED. ";  break;
              case HOLD:     msg = " HOLD. ";     break;
              case RELEASED: msg = " RELEASED. "; break;
              case IDLE:     msg = " IDLE. ";     break;
              default: break;
            }
            // Print the current state of the key pressed
            pc.printf( "Key " );
            pc.printf( "%c" , kpd.key[ i ].kchar );
            pc.printf( "%s" , msg.c_str() );
            pc.printf( "\n\r" );
         }
       }
     }
  }
}
```

Hotboards_EventKeypad 3/1

```
#include "mbed.h"
#include "Hotboards_keypad.h"
#include <string>
using std::string;
char keys[ 4 ][ 4 ] = {
    { '1' , '2' , '3' , 'A' },
    { '4' , '5' , '6' , 'B' },
    { '7' , '8' , '9' , 'C' },
    { '*' , '0' , '#' , 'D' }
};

// Defines the pins connected to the rows
DigitalInOut rowPins[ 4 ] = { PTB8 , PTB9 , PTB10 , PTB11 };
// Defines the pins connected to the cols
DigitalInOut colPins[ 4 ] = { PTE2 , PTE3 , PTE4 , PTE5 };
// Creates a new keyboard with the values entered before
Keypad kpd( makeKeymap( keys ) , rowPins , colPins , 4 , 4 );

Serial pc( USBTX , USBRX );     // Configures the serial port

DigitalOut led1( LED1 );        // We will use LED1 (the RED_LED)
bool blink = false;             // we will blink LED1 wehen it becames true
bool ledPin_state;              // variable to preserve LED1 state
```

Hotboards_EventKeypad 3/2

```
// Taking care of some special events.
void kpdEvent( KeypadEvent key )
{
    switch( kpd.getState( ) ) {
        case PRESSED:
            if( key == '#' ) {
                led1 = !led1;
                ledPin_state = led1; // Remember LED state, lit or unlit.
            }
            break;
        case RELEASED:
            if( key == '*' ) {
                led1 = ledPin_state; // Restore LED state
                blink = false;
            }
            break;
        case HOLD:
            if( key == '*' ) {
                blink = true;        // Blink the LED when holding the * key.
            }
            break;
    }
}
```

Rövidített forma a **led1.write()** és a **led1.read()** helyett

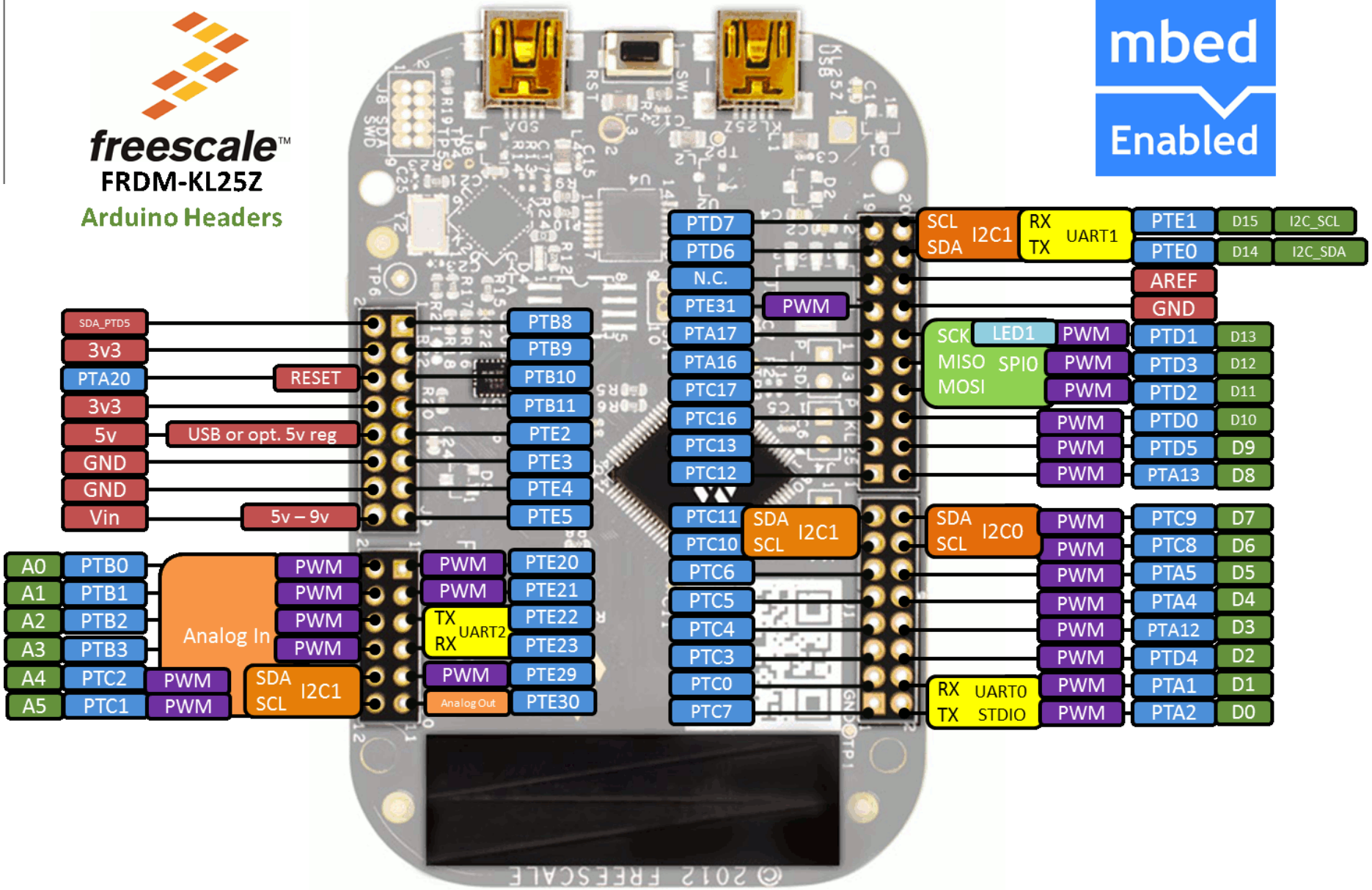
Hotboards_EventKeypad 3/3

```
int main() {
    led1 = 1;                                // Turn the LED on.
    ledPin_state = led1;                     // Store initial LED state.
    kpd.addEventListener( kpdEvent );  // Add an event listener for this keypad
    while(1) {
        char key = kpd.getKey( );
        if( key ) {
            pc.printf( "%c" , key );
        }
        if( blink ) {
            led1 = !led1;                    // Change the ledPin from Hi2Lo or Lo2Hi
            wait_ms( 100 );
        }
    }
}
```

Rövidített forma a **led1.write()** és a **led1.read()** helyett

freescale™
FRDM-KL25Z
Arduino Headers

mbed Enabled



freescale™

FRDM-KL25Z

Additional Peripherals