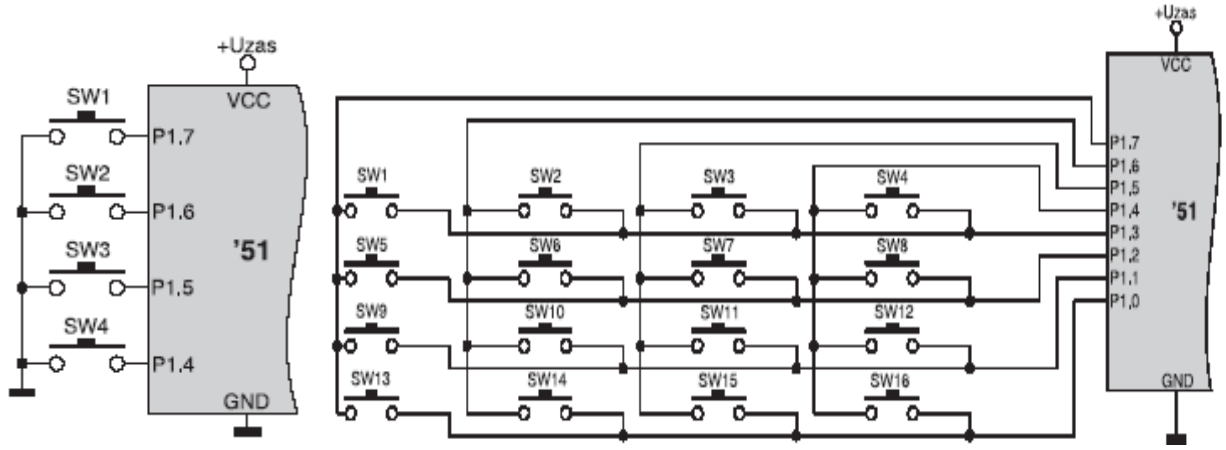
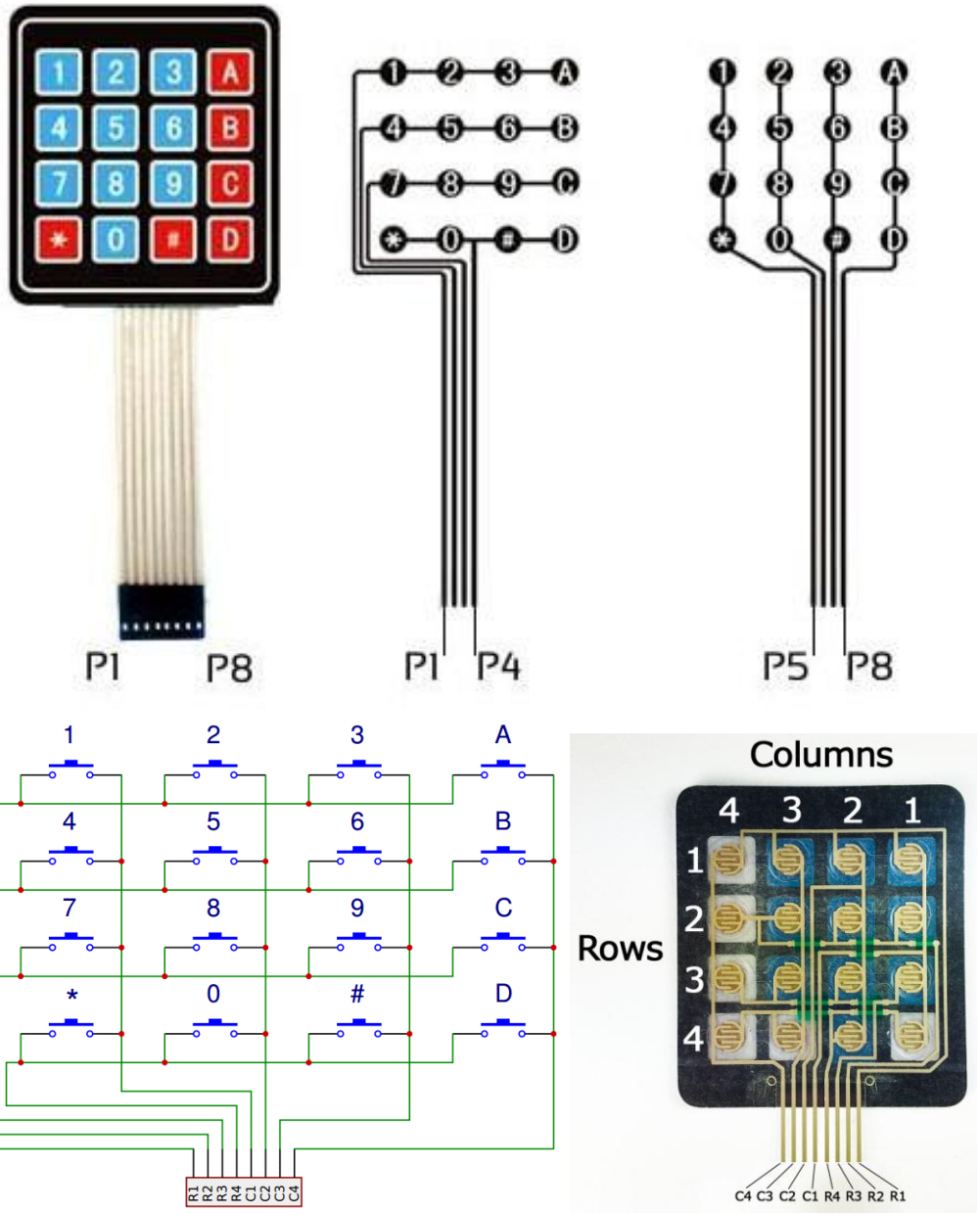
Klawiatura matrycowa

Budowa matrycy klawiatury.

Nieodzownym elementem każdego systemu mikroprocesorowego jest klawiatura. Umożliwia ona wpływ użytkownika na wykonywany przez niego program. Jednak teoretycznie zastosowanie w systemie klawiatury składającej się z 16-tu przycisków wymaga zastosowania takiej samej ilości linii portu wejścia-wyjścia.



Oczywiście nie można sobie na taką rozrzutność pozwolić. Zastosowanie matrycy klawiaturowej pokazanej na rysunku 1 pozwala na zredukowanie ilości linii portu do ośmiu.
Cała operacja wymaga jednak napisania odpowiedniego programu tzw. przeglądania klawiatury.
Jak widać mamy w tej klawiaturze cztery kolumny i cztery wiersze.
Wiersze są podłączone do linii D14 do D17, kolumny do linii D18-D21.
Mikrokontroler będzie wysyłał na kolejne kolumny tzw. krążące zero i czytał za każdym razem stan linii wierszy. Jeśli w danej kolumnie któryś z klawiszy jest wciśnięty to na danej temu klawiszowi linii wiersza pojawi się stan logiczny „0”. Następnie zero przesuwane jest na bit sterujący następną kolumną i znowu następuje odczyt stanu linii wierszy. Osobny problem stanowi eliminacja drgań zestyków, realizuje się ją programowo przez dwukrotne sprawdzanie czy po upływie około kilkunastu mS ten sam klawisz jest nadal wciśnięty, dopiero teraz program może potraktować informację o klawiaturze jako wiarygodną.



Aby móc używać klawiaturę matrycową z Arduino należy, korzystając z menadżera bibliotek zainstalować, bibliotekę **Keypad.h**

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

Uwagi na temat użycia biblioteki Keypad:

- Funkcje biblioteki nie blokują działania pozostałej części program – w przypadku naciśnięcia i przytrzymywania dowolnego klawisza dalszy ciąg kodu będzie wykonywany.
- Użycie funkcji np. delay(250) może sprawiać wrażenie, że program nie reaguje na wciskanie klawiszy.
- Funkcja getKey() zwraca kod klucza natychmiast po jego naciśnięciu lecz nie powoduje jego automatycznego powtarzania. Możliwe jest również wykrycie zdarzenia zwolnienia klawisza (RELEASED) jeśli użyjemy możliwości eventListener z biblioteki.
- Zaleca się nie używać pinu cyfrowego 0 i 1 przeznaczonych do komunikacji szeregowej.

Wersja 3.0 (19.07.2012) została zmodyfikowana umożliwiając domyślnie obsługę jednoczesnego wciśnięcia kilku klawiszy przy zachowaniu pełnej kompatybilności wstecznej.

- Nie są wymagane zewnętrzne rezystory bądź diody ze względu na wykorzystanie wewnętrznych rejestrów podciągających procesora.

Konstruktor tworzący obiekt klasy Keypad:

Keypad(**makeKeymap**(userKeymap), row[], col[], rows, cols)

i przykład jego użycia

```
const byte rows = 4;           //cztery wiersze
const byte cols = 3;           //trzy kolumny
char keys[rows][cols] = {
 {'1','2','3'},
 {'4','5','6'},
 {'7','8','9'},
 {'#','0','*'}
};
byte rowPins[rows] = {5, 4, 3, 2};   //oznaczenie pinów przyłączonych do wierszy klawiatury
byte colPins[cols] = {8, 7, 6};        // oznaczenie pinów przyłączonych do kolumn klawiatury
Keypad  keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );
```

Powyższy fragment kodu spowoduje utworzenie obiektu keypad, który korzysta z zadeklarowanych końcówek Arduino oraz podanej mapy znaków na klawiaturze.

Przykład 1:

```
#include  <Keypad.h>
const byte  ROWS = 4; // ustawia układ złożony z czterech wierszy
const byte  COLS = 4;  // ustawia układ złożony z czterech kolumn
char  keys[ROWS][COLS] =
 { {'1','2','3','A'},
   {'4','5','6','B'},
   {'7','8','9','C'},
   {'*','0','#','D'} };
byte  rowPins[ROWS] = {14, 15, 16, 17};
byte  colPins[COLS] = {18, 19, 20, 21};
Keypad  keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

#define ledpin 13
```

```
void setup()
{
  Serial.begin(9600);
  pinMode(ledpin,OUTPUT);
  digitalWrite(ledpin, HIGH);
}

void loop()
{
 char key = keypad.getKey();            // przeglądanie klawiatury i odczytanie znaku
 if (key != NO_KEY)
 {
  switch (key)
  {
    case '*':
      digitalWrite(ledpin, LOW);
      break;
    case '#':
      digitalWrite(ledpin, HIGH);
      break;
    default:
      Serial.println(key);
  }
  delay(100);
 }
}
```

Przykład 2:

```
#include <LiquidCrystal.h>
#include  <Keypad.h>
const  byte ROWS = 4; // ustawia układ złożony z czterech wierszy
const  byte COLS = 4; // ustawia układ złożony z czterech kolumn
char  keys[ROWS][COLS] =
 { {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'} };
byte rowPins[ROWS] = {14, 15, 16, 17};
byte colPins[COLS] = {18, 19, 20, 21};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);  //RS,EN, D0, D1, D2, D3
```

```
void setup()
{
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
}

void loop()
{
  char key;
 if (key = keypad.getKey())
  lcd.write(key);
  delay(100);
}
```

Zadania:

1. Napisać program sterowania zamkiem cyfrowym:
   - tajny kod czterocyfrowy powinien być podany przez port szeregowy w sekcji setup()
   - świecenie czerwonej diody LED sygnalizuje zamknięcie sejfu,
   - otwarcie sejfu (zaświecenie diody zielonej) następuje po podaniu z klawiatury właściwego kodu zakończonego znakiem #,
   - zamknięcie sejfu i zapalenie czerwonej diody następuje po podaniu znaku *
2. Napisać program symulujący działanie kalkulatora czterodziałaniowego umożliwiający wprowadzenie dwóch liczb całkowitych i wybór rodzaju operacji na nich.