Vzdelávacia sada 26 v 1 pre Arduino UNO R4

Úvod

Vitajte v príručke k vývojovej doske Arduino Uno R4.

Začnime objavovať Arduino Uno R4 a naučme sa, ako používať jeho širokú škálu elektronických modulov. Táto vývojová doska obsahuje 26 ľahko zrozumiteľných, zaujímavých a inšpiratívnych lekcií, ktoré vás krok za krokom prevedú celým procesom. Prostredníctvom týchto lekcií sa zoznámite s rôznymi elektronickými modulmi, rozviniete schopnosti logického myslenia, podnietite kreativitu a naučíte sa programovať rôzne moduly.

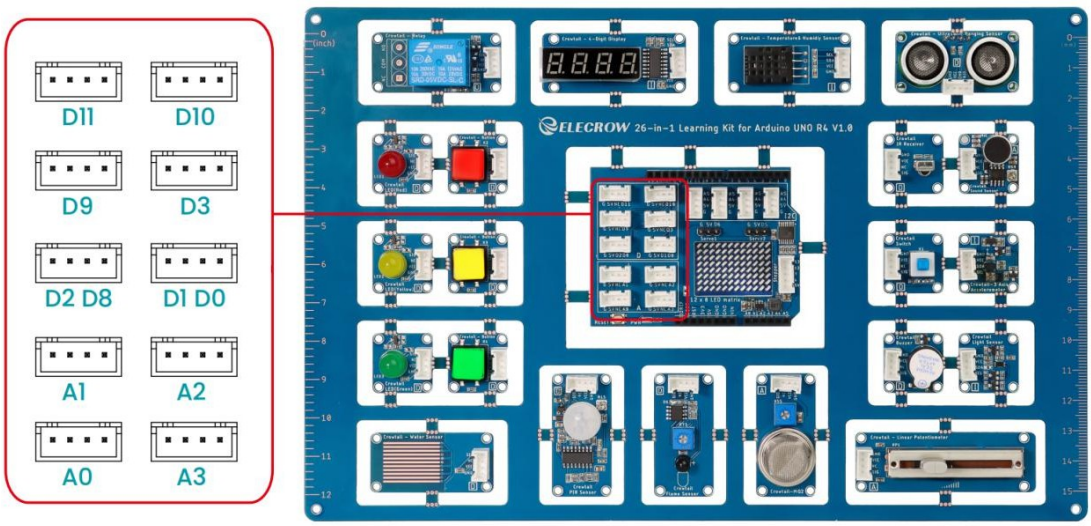
Lekcie začínajú základmi, ako je inštalácia softvéru Arduino. Následne sa zoznámite s doskou Arduino Uno R4 a jej senzormi, potom sa naučíte programovať tieto moduly a osvojíte si príslušné programovacie koncepty a jazyky. Nakoniec sa naučíte, ako tieto senzory využiť v praktických projektoch. Každý krok je jasne vysvetlený, vďaka čomu si programovanie Arduina rýchlo osvoja aj začiatočníci.

Doska Arduino Uno R4 obsahuje 21 elektronických modulov, z ktorých každý má jedinečné funkcie a vlastnosti, čo z nej robí ideálnu voľbu pre začiatočníkov. Pri prechádzaní lekciami nielenže pochopíte základné pojmy, ako sú senzory, digitálne a analógové signály,

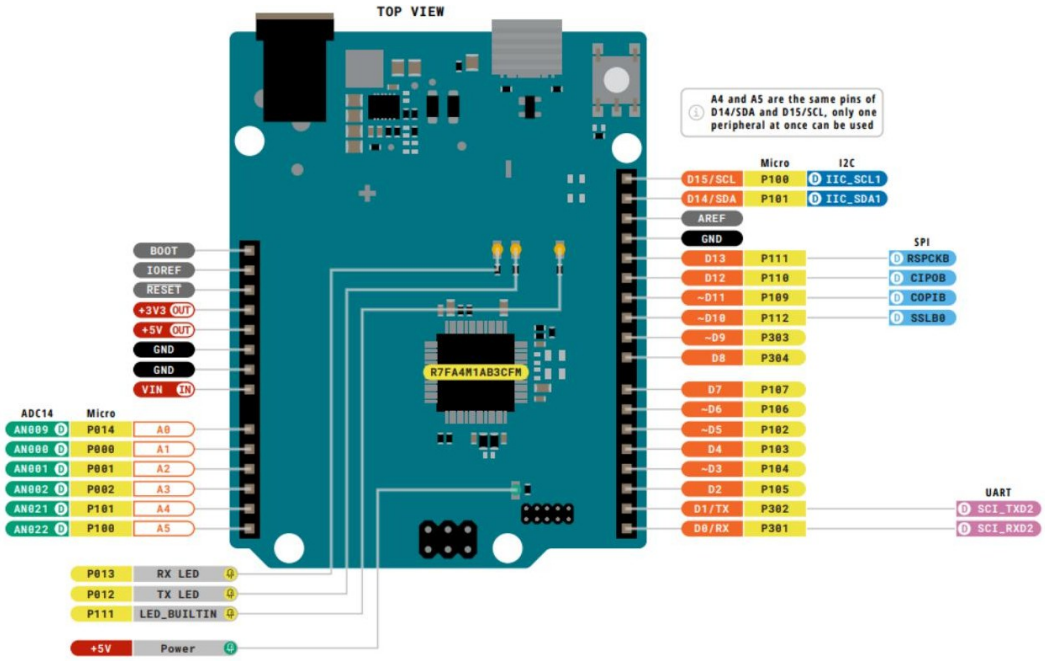
analogovo-digitálny prevod a programovaciu logiku, ale získate aj skúsenosti s používaním pokročilejších modulov. A čo je najdôležitejšie, oficiálne začnete svoju cestu do sveta programovania Arduina a posilníte svoje schopnosti logického myslenia.

Na programovanie budeme používať softvér Arduino (Arduino IDE). Platforma Arduino patrí medzi najpopulárnejšie open-source platformy na svete. Je jednoduchá na používanie a ponúka bohaté hardvérové zdroje (rôzne dosky Arduino) aj softvérové zdroje (Arduino IDE), vďaka čomu je jednou z najlepších volieb na štúdium programovania.

Úvod do vývojovej dosky

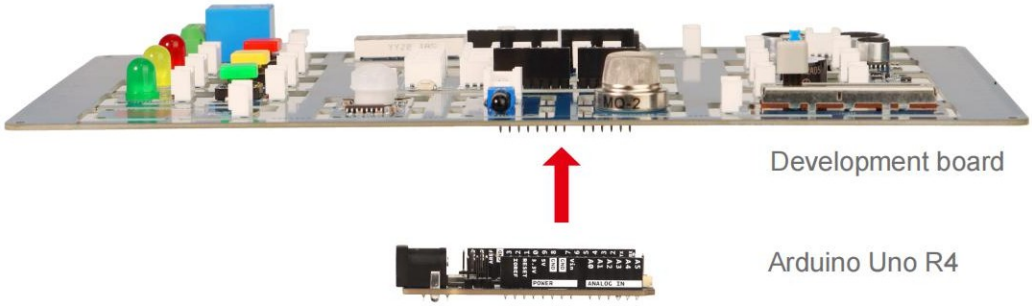


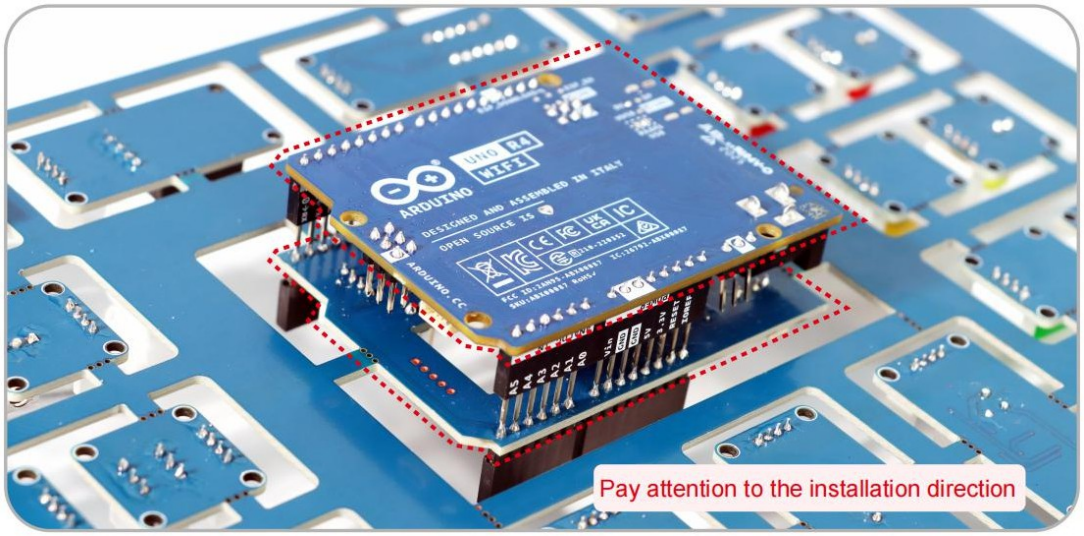
Pozrime sa bližšie na vývojovú dosku Arduino Uno R4! Mnohí začiatočníci možno nepoznajú čísla a označenia vytlačené na doske. V nasledujúcich častiach prejdeme jedno po druhom všetky označenia, aby ste pochopili funkciu každého pinu a rozhrania a mohli tak dosku využívať efektívnejšie.

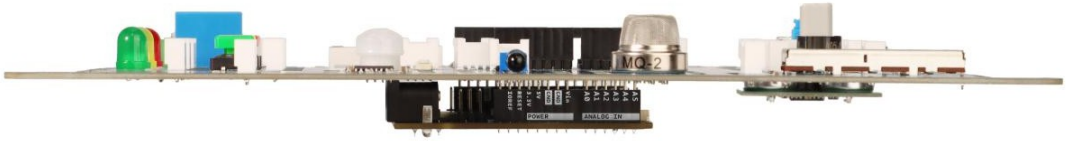


Inštalácia hlavnej dosky

Najskôr vložte hlavnú riadiacu dosku Arduino Uno R4 do vývojovej dosky zo zadnej strany. Pred vložením sa uistite, že sú všetky piny správne zarovnané. Vývojová doska má jasne označené polohy jednotlivých pinov zodpovedajúce doske Arduino Uno R4, čo uľahčuje správne pripojenie všetkých komponentov.







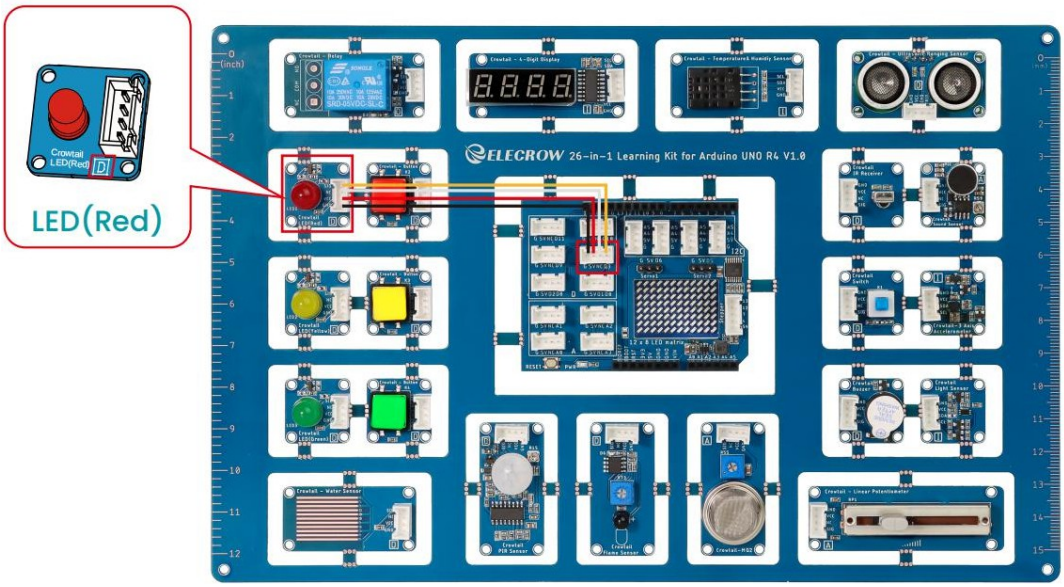
Vysvetlenie označenia pinov a pripojenia modulov

Na vývojovej doske si všimnete, že každý senzorový modul je označený písmenom „A“ alebo „D“. Tieto písmená označujú typ pinu, ku ktorému by mal byť senzor pripojený.

„A“ znamená analógový, ktorý sa zvyčajne používa na vstup a slúži hlavne na čítanie analógových hodnôt zo senzorov.

„D“ znamená digitálny, ktorý sa môže používať ako na vstup, tak aj na výstup, napríklad na čítanie stavov senzorov alebo na odosielanie signálov high/low do výstupných modulov.

Napríklad modul LED je označený písmenom „D“, čo znamená, že by mal byť pripojený k jednému z digitálnych pinov.



Zamerajme sa na dosku Arduino Uno R4 zobrazenú vyššie. Možno si všimnete, že niektoré digitálne („D“) piny majú vedľa seba symbol „~“. Tento symbol je veľmi dôležitý – označuje, že pin podporuje výstup PWM (pulzná šírková modulácia).

To znamená, že nie sme obmedzení len na zapínanie a vypínanie LED. Namiesto toho môžeme vysielať signály PWM na rôznych úrovniach, čo umožňuje LED svietiť s rôznou intenzitou.

Napríklad:

„D4“ označuje digitálny pin. Zvyčajne sa používa na:

Prijímať signály s vysokou alebo nízkou úrovňou zo vstupných modulov (napríklad stlačenie alebo uvoľnenie tlačidla) Ovládať stav zapnutia/vypnutia výstupných modulov (napríklad zapnutie alebo vypnutie LED) Čítať signály zo senzorov, ktoré majú len dva stavy (zapnuté/vypnuté)

„~D3“ označuje digitálny pin s podporou PWM. Tilda („~“) označuje, že pin podporuje PWM. Takéto piny môžu:

Vysielať štandardné digitálne signály s vysokou/nízkou úrovňou

Vysielať analógové signály PWM na ovládanie jasu, rýchlosti alebo výšky tónu Čítať digitálne signály zo senzorov

„A2“ označuje analógový vstupný pin. Používa sa na čítanie spojitých analógových signálov zo senzorov, napríklad meniacich sa úrovní napätia. Je to bežné u senzorov s viacerými stavmi, ako je napríklad posuvný potenciometer. Hoci analógové piny môžu čítať aj jednoduché dvojstavové signály, na tento účel sa zvyčajne uprednostňujú digitálne piny.

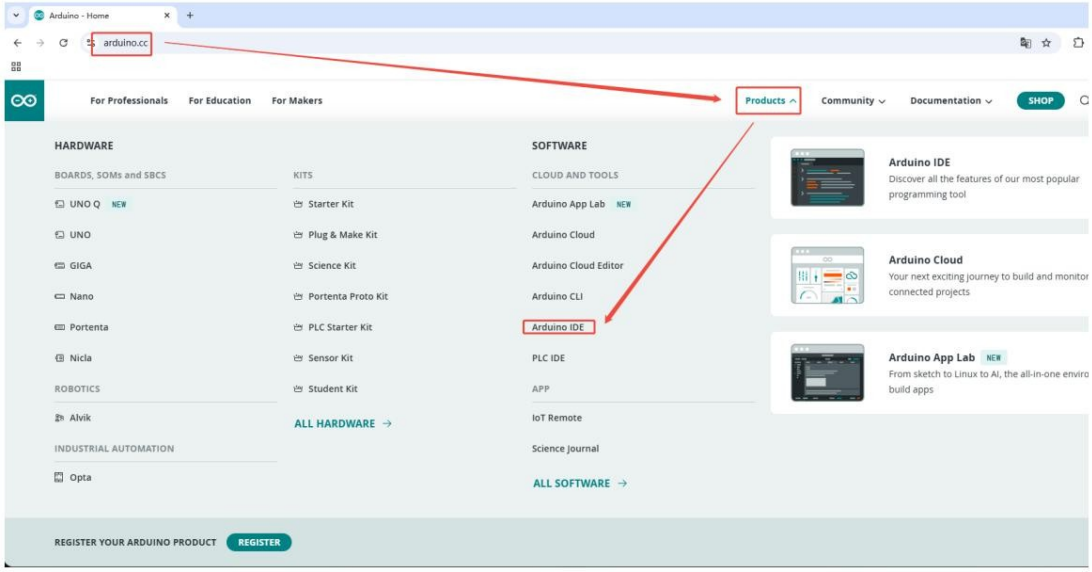
Inštalácia Arduino IDE

Stiahnite si Arduino do systému Windows

Krok 1:

Prihláste sa na oficiálnu webovú stránku Arduino a stiahnite si

Arduino Oficiálna webová stránka Arduino:



https://www.arduino.cc/

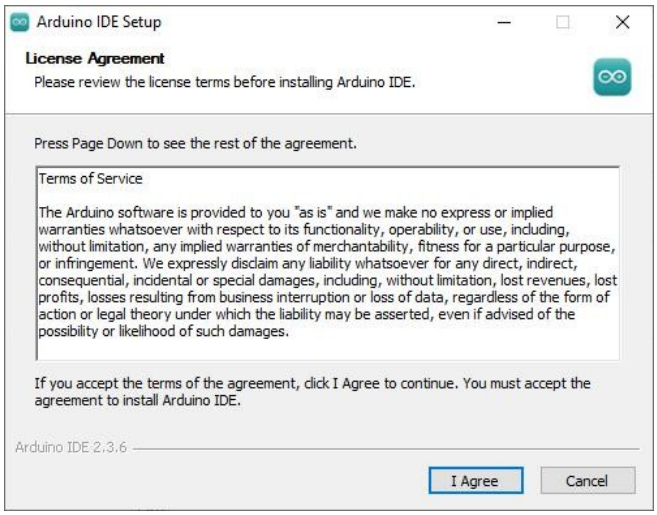
Krok 2:

Vyberte operačný systém vášho počítača, napríklad Windows. Poznámka: V tomto návode sa používa verzia 2.3.6. Môžete vyskúšať aj iné verzie, ale ak sa počas flashovania vyskytnú akékoľvek problémy, prejdite späť na verziu 2.3.6 a skúste to znovu.

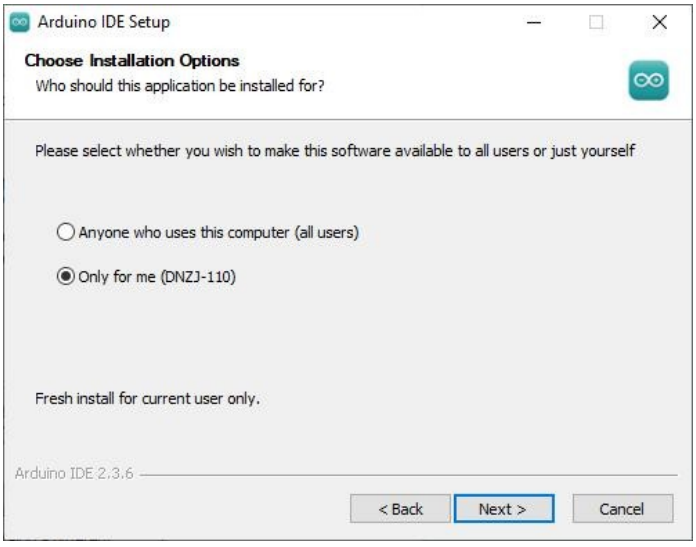
Krok 3:

1. Pri inštalácii Arduina vyhľadajte spustiteľný súbor s príponou .exe v priečinku, do ktorého ste ho predtým uložili, čo je inštalačný balík Arduina.

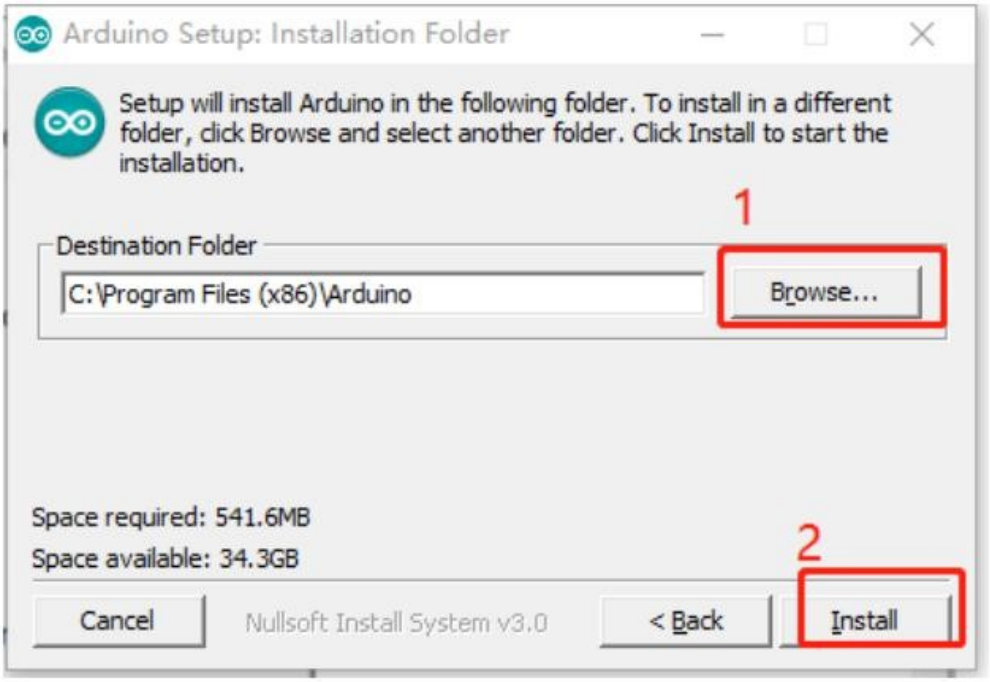
2. Po dvojitom kliknutí na inštalačný balík sa zobrazí táto stránka. Kliknite na „Súhlasím“.



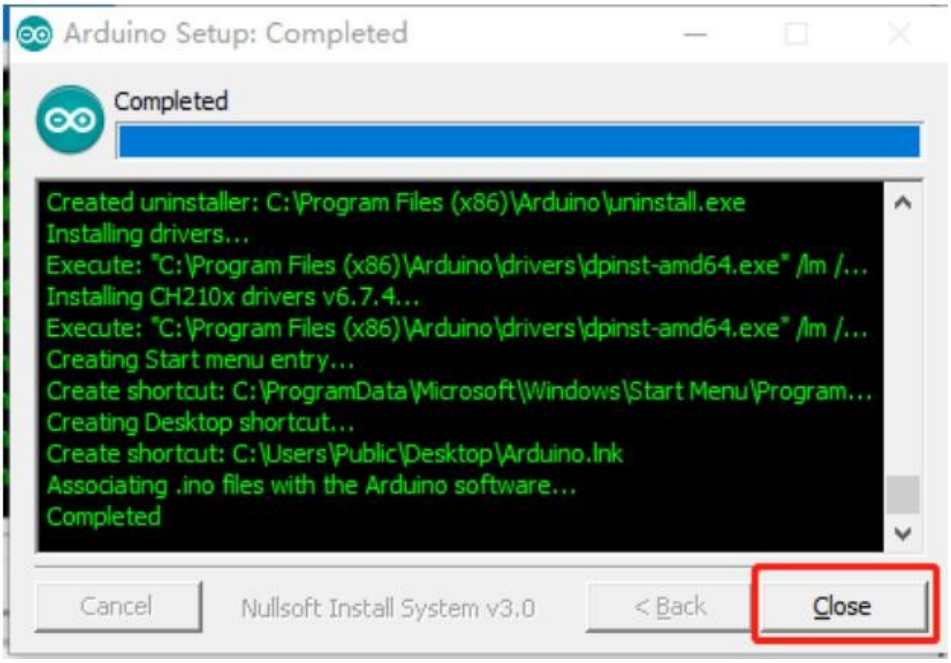
3. Zaškrtnite všetky predvolené možnosti a kliknite na „Ďalej“.



4. Kliknite na tlačidlo „Prehľadávať“ a vyberte miesto inštalácie; odporúča sa nainštalovať program na akýkoľvek iný disk ako disk C:. Potom kliknite na tlačidlo „Inštalovať“.



5. Inštalácia bola dokončená



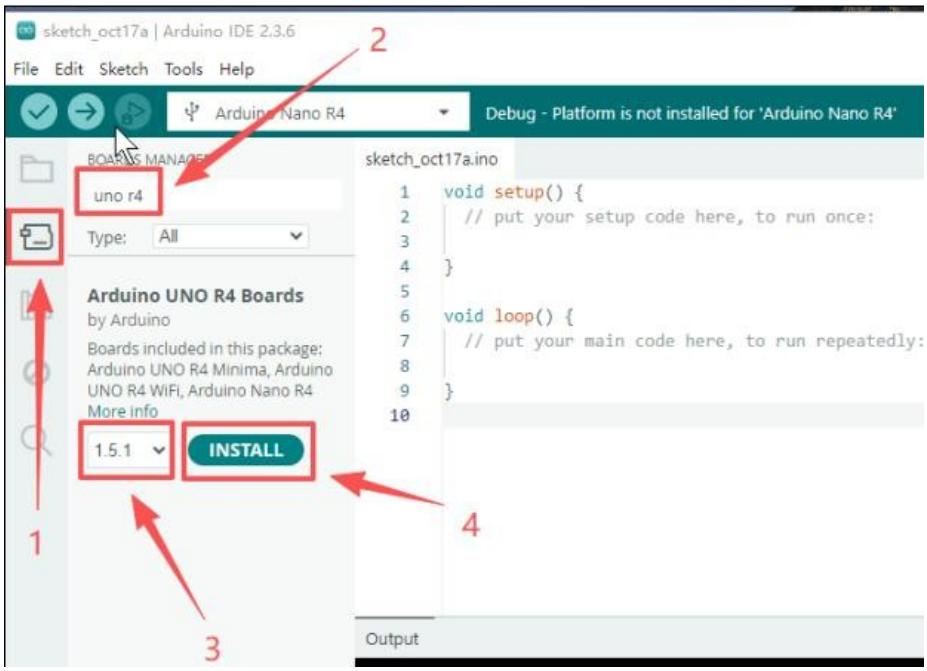
Inštalácia vývojovej dosky Uno R4

① Kliknite na ikonu vývojovej dosky vľavo

② Do vyhľadávacieho poľa zadajte „uno r4“

③ Vyberte verziu 1.5.1. Môžete vyskúšať aj iné verzie, ale ak sa nahratie kódu nezdaří, prejdite späť na verziu 1.5.1. Kód v tomto tutoriáli bol vyvinutý pomocou verzie 1.5.1.

④ Kliknite na „INSTALL“ (Inštalovať)

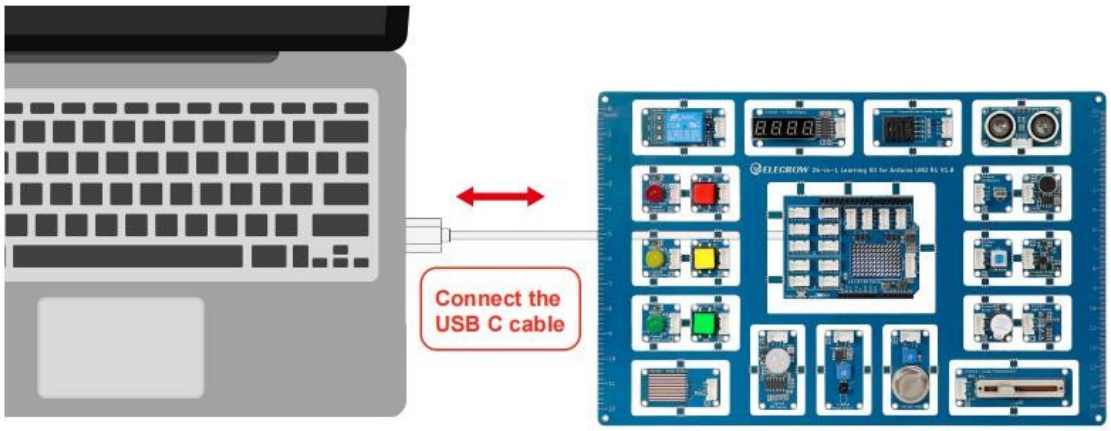


Počas inštalácie sa zobrazí okno – stačí kliknúť na „Yes“.

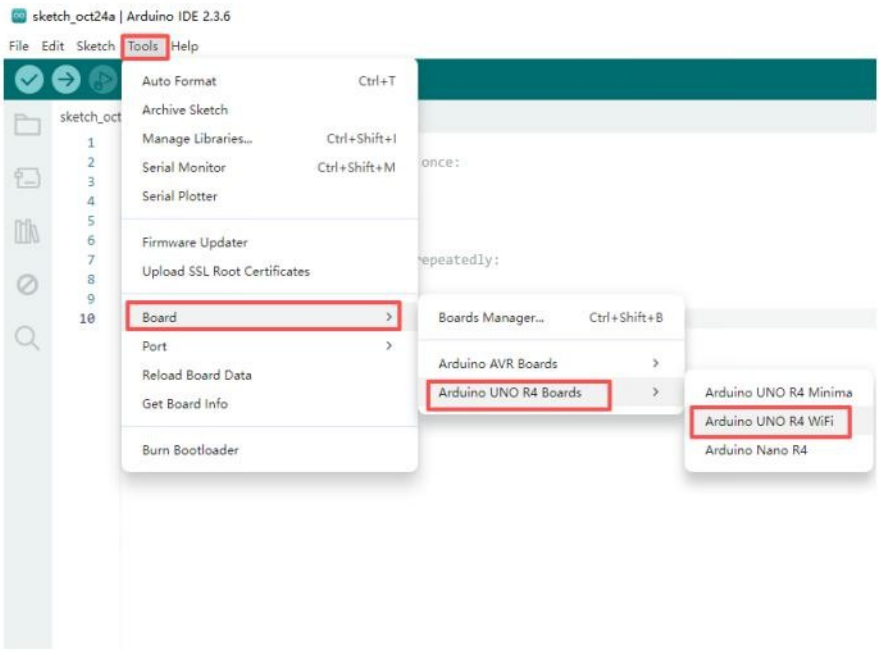
Kroky na nahratie (dôležité)

Vo všetkých nasledujúcich lekciách sú kroky na nahrávanie rovnaké. Prečítajte si ich pozorne a ak niečo zabudnete, môžete sa sem vrátiť a postupovať podľa týchto krokov:

1. Pripojte port na nahrávanie typu C k počítaču.

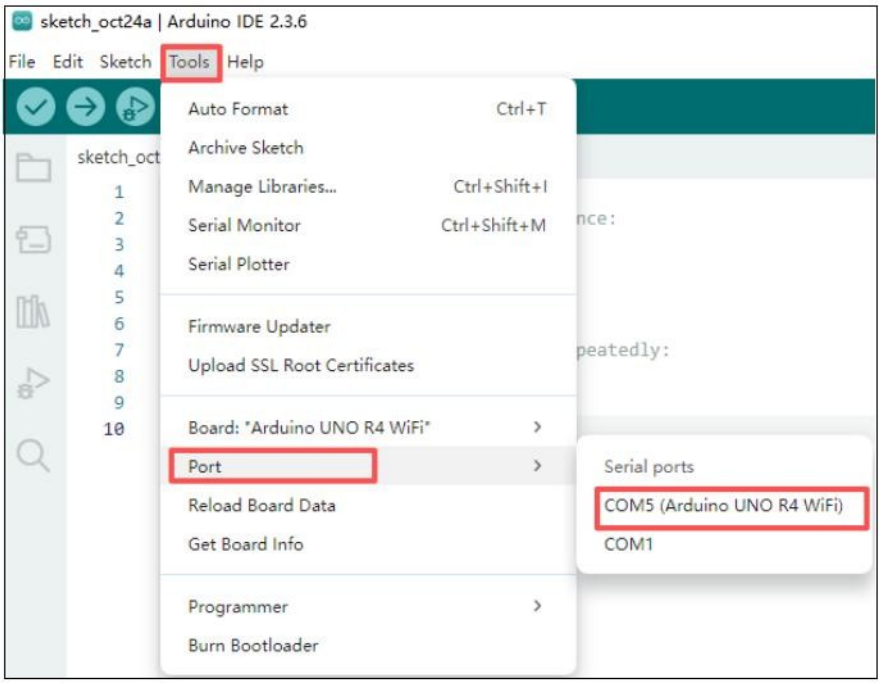


2. Po napísaní kódu vyberte dosku, na ktorú chcete nahrať: „Nástroje“ -> „Doska“ -> „Dosky Arduino UNO R4“ -> „Arduino Uno R4 WiFi“



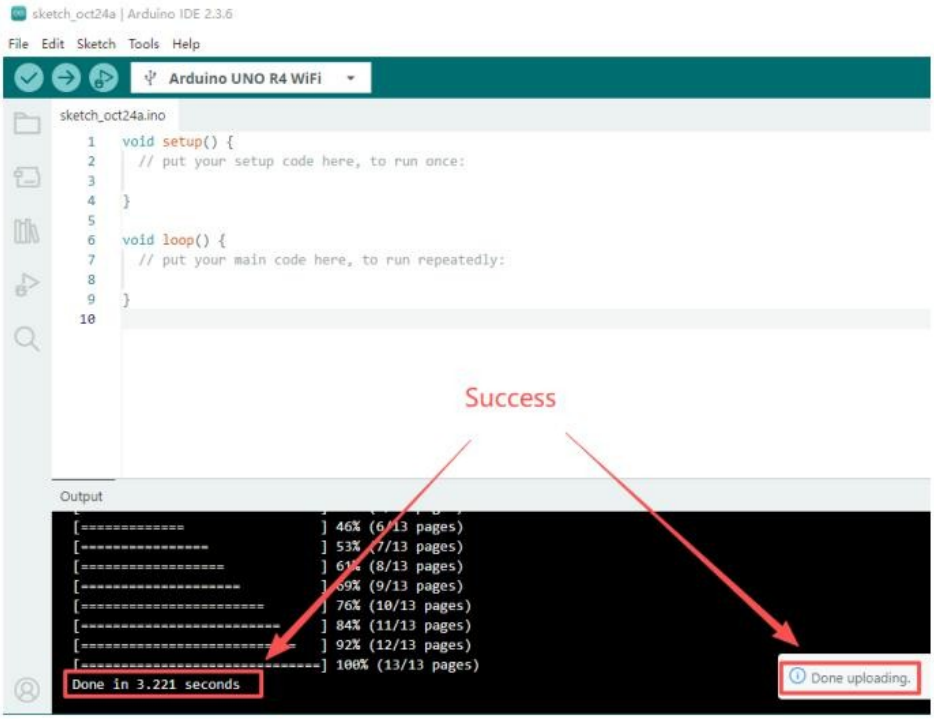
3. Vyberte sériový port pre nahrávanie (poznámka: v tutoriáli je zobrazený COM5, ale váš môže byť iný – uistite sa, že zodpovedá vašej doske Arduino Uno R4 WiFi).

„Tools“ -> „Port“ -> „COM5 (Arduino Uno R4 WiFi)“



4. Kliknite na tlačidlo „Upload“ (Nahrať) na nahratie kódu do vývojovej dosky.

5. Potvrďte, že nahratie prebehlo úspešne.



Pridať knižnicu

Knižnice Arduino slúžia na zjednodušenie ovládania hardvéru a procesu programovania. Knižnice obsahujú základný kód potrebný na prevádzku rôznych senzorov a modulov, čo umožňuje vývojárom vykonávať zložité komunikačné a riadiace úlohy pomocou jednoduchých volaní funkcií bez nutnosti rozumieť detailom na nízkej úrovni, ako je konfigurácia registrov alebo časovanie komunikácie. To výrazne skracuje dobu potrebnú na osvojenie si systému.

Vďaka používaniu knižníc môžu vývojári zvýšiť efektivitu vývoja a stabilitu programu.

Osvedčené knižnice sú zvyčajne dôkladne otestované a kompatibilné s viacerými doskami a modulmi Arduino, čo znižuje pravdepodobnosť chýb. To pomáha začiatočníkom rýchlejšie sa zoznámiť s prácou a uľahčuje rozširovanie a údržbu projektov v budúcnosti.

Existujú dva spôsoby pridávania knižníc:

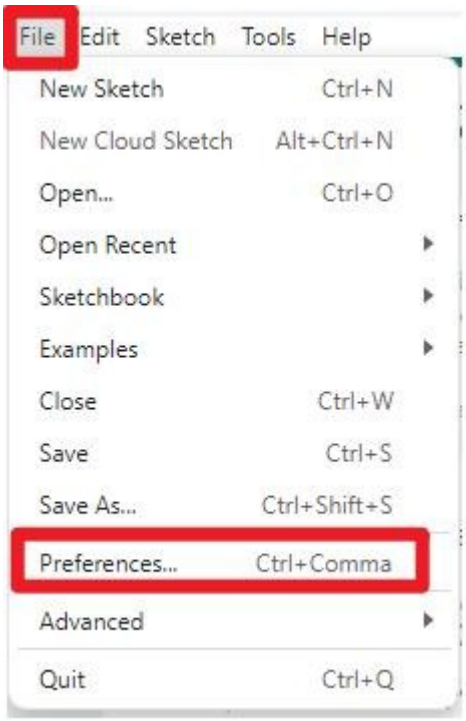
Metóda jedna:

Stiahnite si ich priamo z oficiálneho repozitára Elecrow na GitHub. Adresa GitHub:

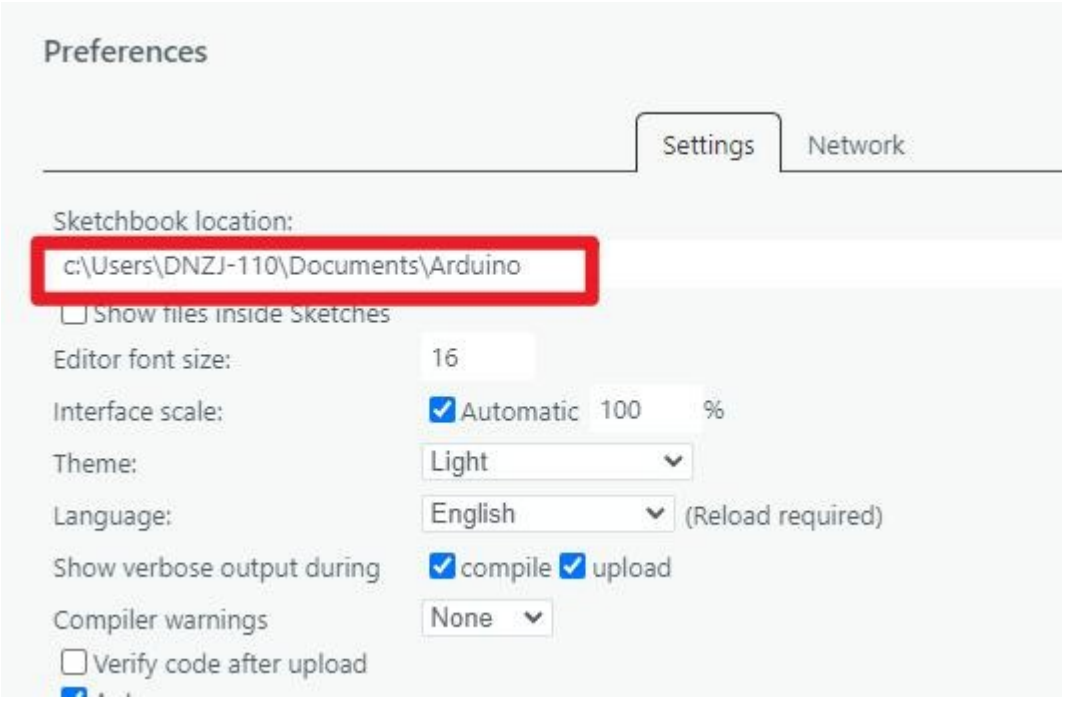
https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/libraries

Uložte stiahnutú knižnicu do adresára knižníc Arduina:

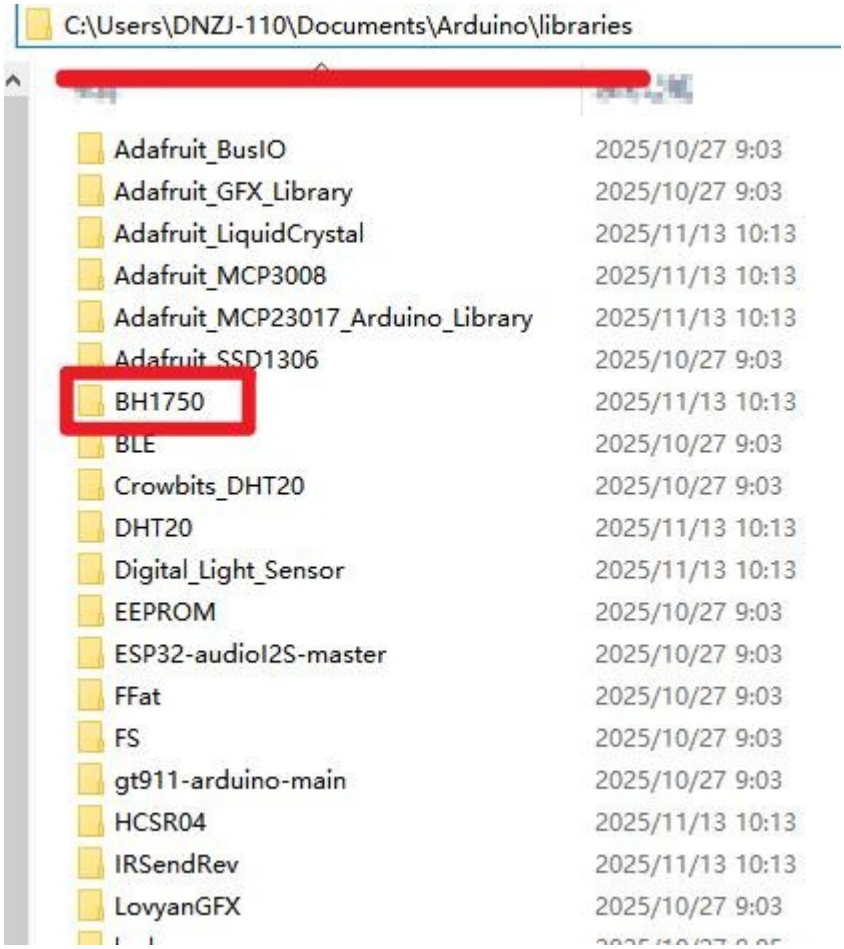
① Ak chcete nájsť adresár knižníc, kliknite na Súbor → Nastavenia.



② Prejdite do priečinka Sketchbook;



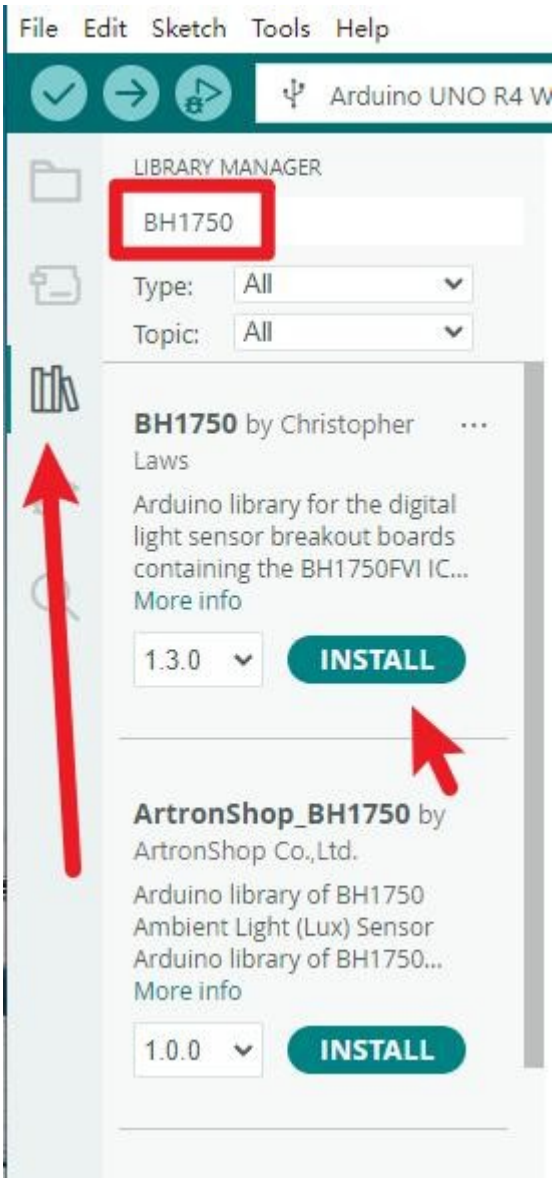
③ Umiestnite stiahnuté súbory knižnice do zložky libraries v tejto ceste;



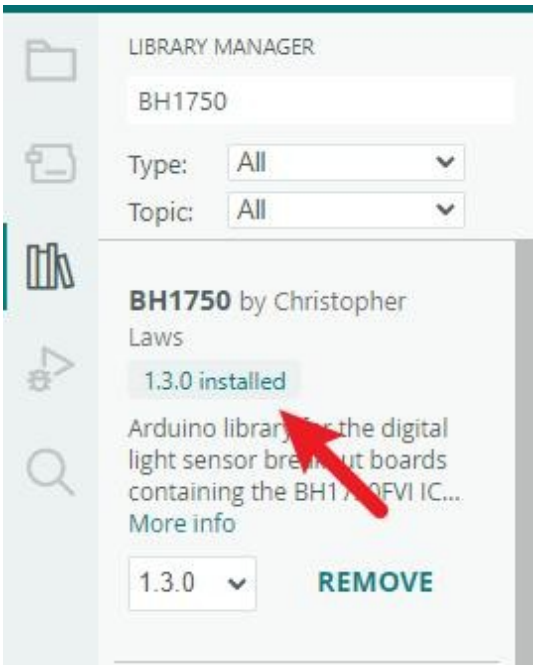
Metóda č. 2:

Stiahnite a nainštalujte priamo v softvéri Arduino.

Kliknite na tretiu možnosť vľavo s názvom „Správca knižníc“. Zadajte názov súboru knižnice, ktorý potrebujete – v tomto prípade použijeme ako príklad BH1750. Akonáhle sa zobrazí požadovaná knižnica, vyberte príslušnú verziu (v tomto prípade používame verziu 1.3.0) a potom kliknite na „Inštalovať“.



Inštalácia bola úspešná. Pripravené na použitie.


LIBRARY MANAGER
BH1750
Type:
All
Topic:
All
BH1750 by Christopher Laws
1.3.0 installed
Arduino library
the digital light sensor
boards containing the BH1
FVI IC...
More info
1.3.0
REMOVE

Lekcia 01 – Ovládanie LED

Úvod

v kombinácii s algoritmom výpočtu polohy (Madgwickovo filtrovanie) a bzučiakom, aby sme vytvorili inteligentný systém alarmu pri naklonení.
Na konci tejto lekcie budete schopní pochopiť, ako sú v skutočnosti implementované systémy, ako napríklad alarm proti pádu telefónu, alarm proti prevráteniu zariadenia a detekcia polohy robota.

Ciele výučby

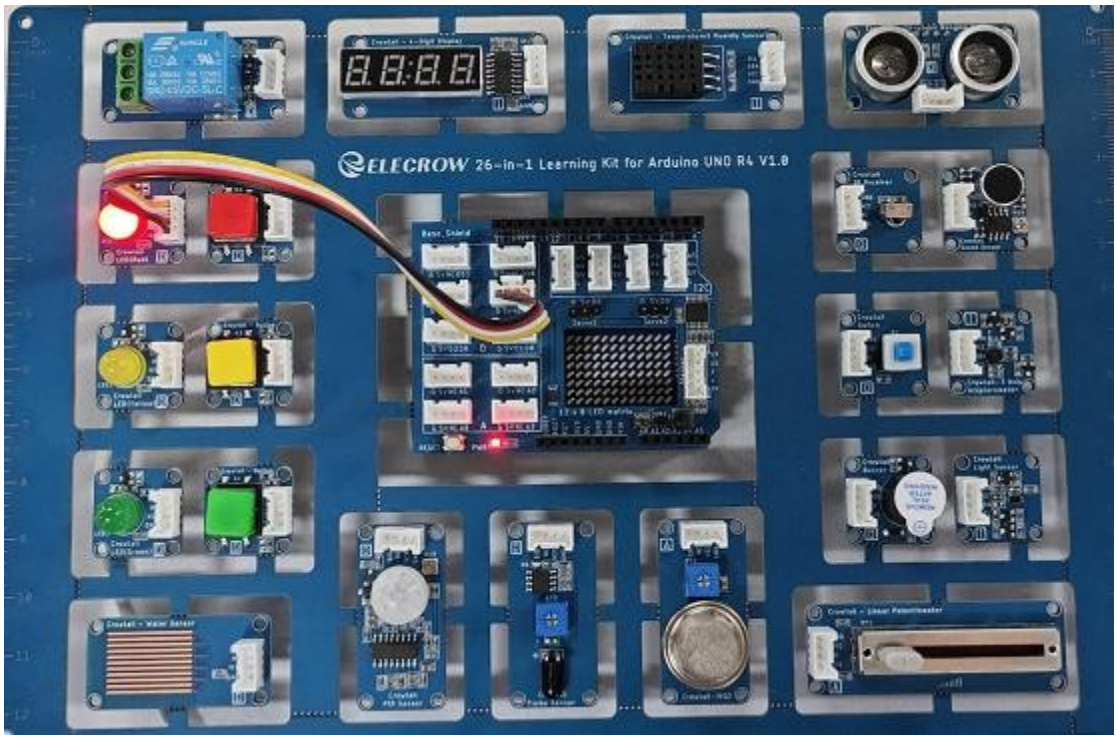
1. Naučiť sa ovládať digitálne výstupy. 2.

Pochopiť funkcie pinMode() a digitalWrite().

3. Vytvorte program na blikanie LED.

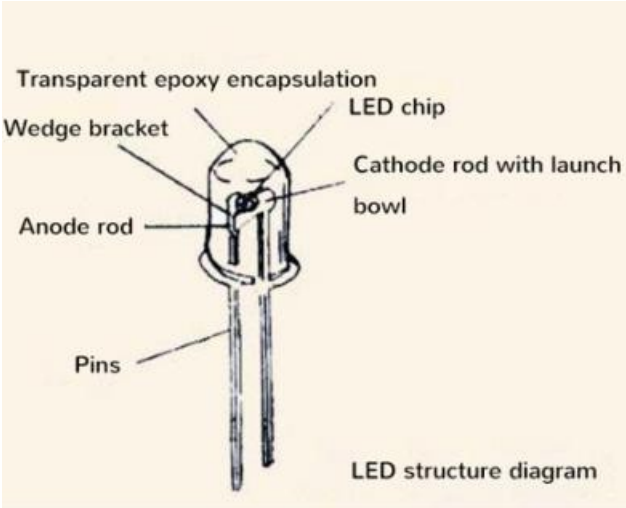
Náhľad výsledku

Červená LED bliká.

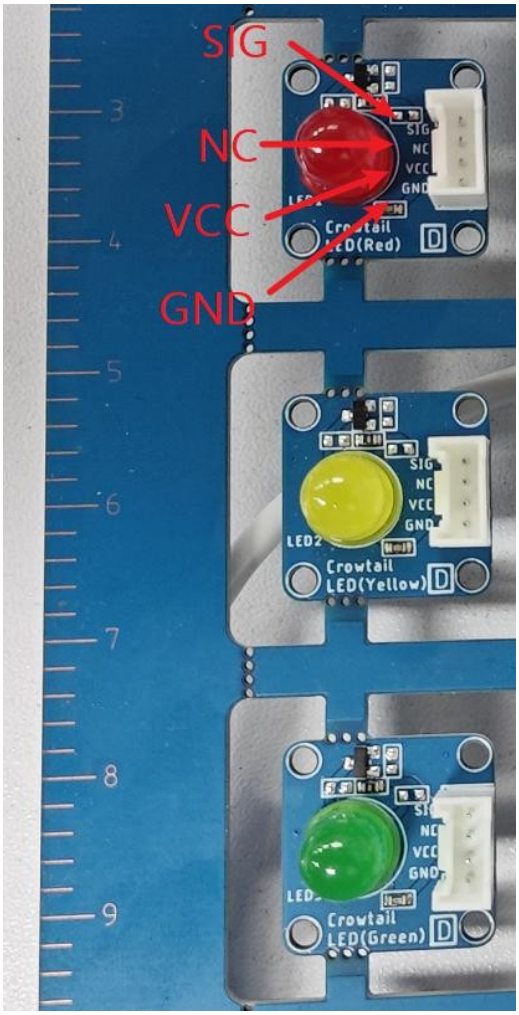


1. Vysvetlenie princípu

LED, alebo svetelná dióda, je polovodičové zariadenie, ktoré priamo premieňa elektrickú energiu na viditeľné svetlo. Jej základom je polovodičový čip: jedna strana čipu je upevnená na vývodovom ráme a pripojená k zápornému pólu napájacieho zdroja, zatiaľ čo druhá strana je pripojená k kladnému pólu. Vo väčšine prípadov je dlhší vývod kladný (anóda) a kratší vývod záporný (katóda).



Na našej vývojovej doske je tento LED modul už integrovaný. Pre uľahčenie zapojenia a používania má štyri vývody: SIG, NC, VCC a GND.



SIG: Signálny pin, ktorý sa používa na prenos dát alebo riadiacich signálov. NC: Nepripojený; nezúčastňuje sa na obvode.

VCC: Kábel kladného napájania, ktorý slúži na napájanie modulu.

GND: Zem, záporný pól napájacieho zdroja a spoločný referenčný bod pre všetky

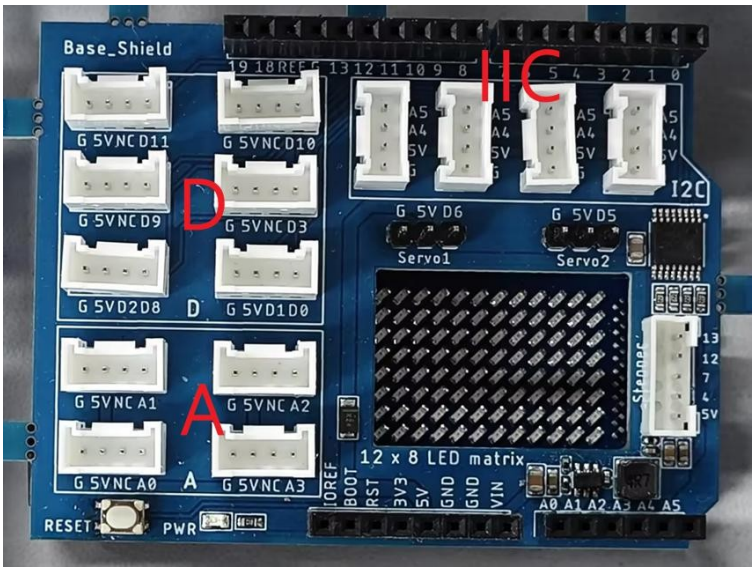
zariadení.

Arduino UNO R4 ponúka niekoľko typov pinov, vrátane digitálnych, analógových a I2C pinov.

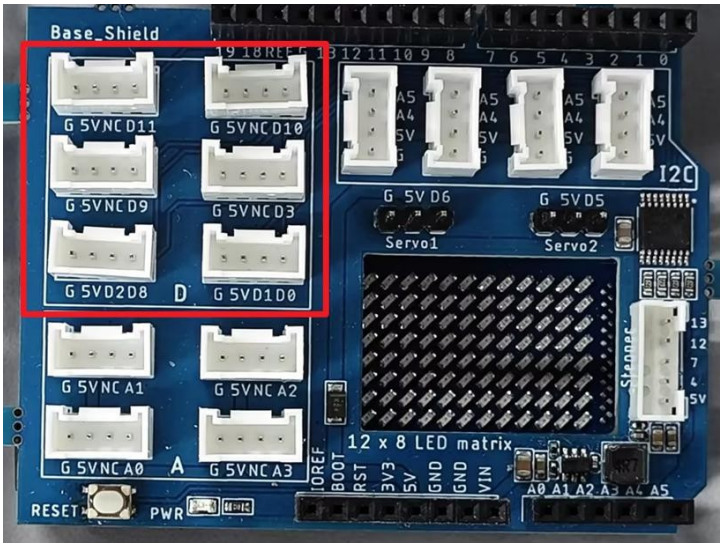
Digitálne piny: Nachádzajú sa v oblasti konektora D a slúžia na čítanie alebo výstup digitálnych signálov s iba dvoma stavmi – HIGH alebo LOW (1 alebo 0). Môžu sa používať na ovládanie zariadení, ako sú LED diódy, tlačidlá a relé. Piny 3, 5, 6, 9, 10 a 11 podporujú PWM výstup.

Analógové piny: Nachádzajú sa v oblasti konektora A a slúžia na snímanie plynule sa meniacich analógových signálov, napríklad z fotorezistorov alebo potenciometrov. ADC prevádza tieto signály na hodnoty v rozmedzí od 0 do 1023.

Piny I2C: Tieto piny sa nachádzajú v oblasti konektora IIC a sú určené na komunikáciu I2C: SDA (dáta) a SCL (hodiny). Používajú sa na pripojenie viacerých senzorov alebo modulov, napríklad senzorov teploty a vlhkosti, rozširujúcich dosiek a LCD displejov.



LED je digitálny modul. Jeho signálny pin je pripojený k jednému z digitálnych pinov Arduino UNO R4, umiestnených v oblasti konektora D.



2. Požadované moduly

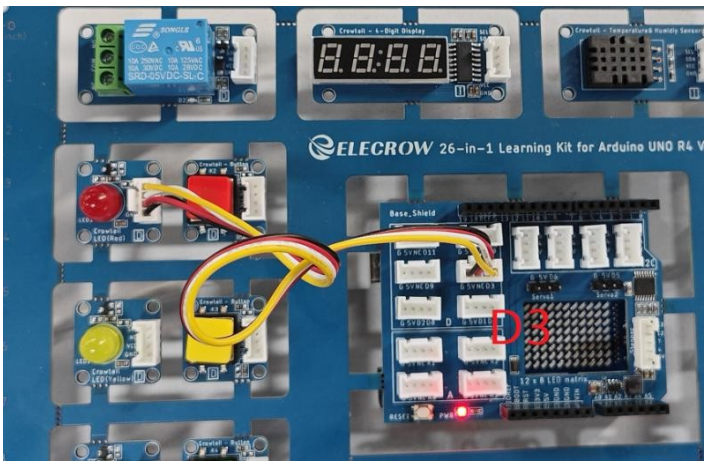
Crowtail – LED × 1

3. Spôsob zapojenia

Crowtail -- LED → port DIGITAL D3

Špecifikácia štvorportového rozhrania
Crowtail:

- GND (čierny) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3



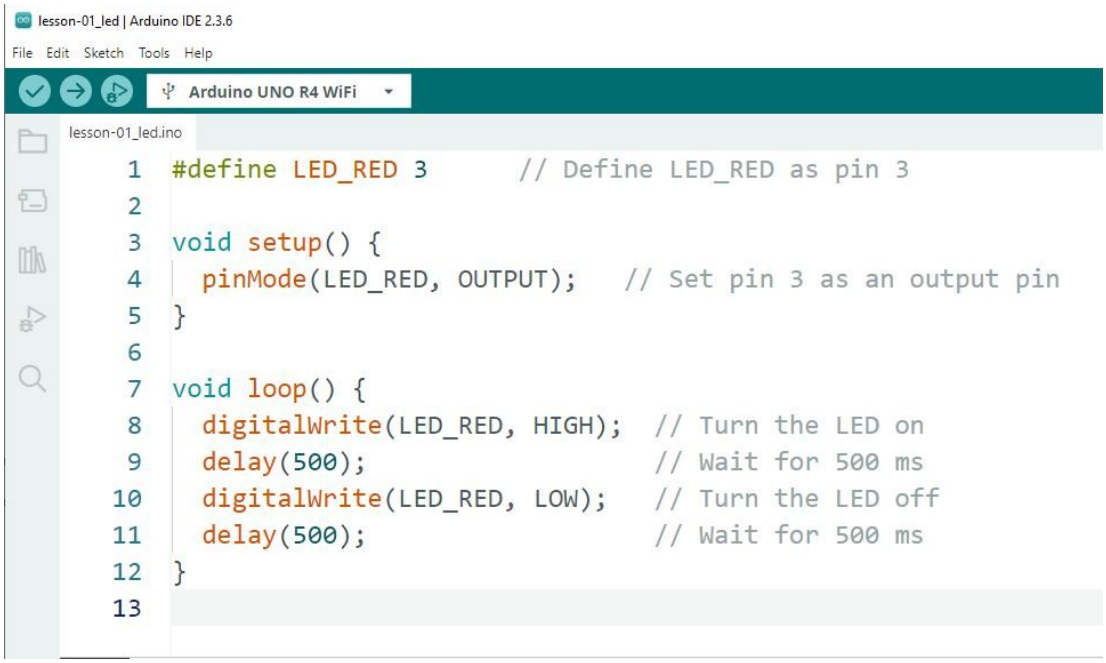
4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad

kódu Odkaz na GitHub：

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_code/lesson-01_led

Otvorte program Lekcia 1 pomocou Arduino IDE.



Vysvetlenie kľúčových kódov

(1) #define sa používa na vytvorenie mena pre číslo alebo hodnotu. Pred kompiláciou vykonáva nahradenie textu, čím sa program stáva prehľadnejším a ľahšie sa udržiava.

(2) setup() je inicializačná funkcia programu Arduino.

Po zapnutí alebo reštartovaní Arduina sa funkcia `setup()` spustí len raz.

Bežne sa používa na nastavenie režimov pinov, inicializáciu sériového portu a inicializáciu senzorov.

(3) pinMode(pin, mode) je funkcia používaná na nastavenie prevádzkového režimu pinu.

pin: Číslo pinu, založené na používanom hardvérovom pine, napríklad 2, 3 alebo 4.

mode: Režim pinu, ktorý môže byť 'INPUT', 'INPUT_PULLUP' alebo 'OUTPUT'.

Prečo musíme nastaviť režim?

Každý pin GPIO na doske Arduino je možné nakonfigurovať na jeden z nasledujúcich režimov:

INPUT: Pin sa používa ako vstup a vyžaduje externý pull-up alebo pull-down rezistor. Ak je signál nestabilný, pin môže kolísať. Tento režim sa používa na čítanie externých signálov, ako sú tlačidlá alebo výstupy senzorov.

INPUT_PULLUP: Vstupný režim s povoleným interným pull-up rezistorom, ktorý automaticky nastaví

pin do stavu HIGH. To je veľmi výhodné pre čítanie tlačidiel (stlačené = LOW).

VÝSTUP: Pin sa používa ako výstup a aktívne vysiela signály s úrovňou HIGH alebo LOW na ovládanie externých zariadení, ako sú LED diódy, ovládače motorov alebo relé.

(4) pinMode(3, OUTPUT) znamená konfiguráciu pinu 3 (D3) Arduina ako výstupu. LED dióda potrebuje na riadenie výstupné napätie, preto musí byť pin nastavený na OUTPUT.

(5) loop() je hlavná slučková funkcia a beží opakovane od začiatku do konca.

Kým Arduino beží, **loop()** sa nikdy nezastaví.

Toto je základná štruktúra vykonávania programu Arduina.

(6) **Funkcia digitalWrite(pin, value) slúži na nastavenie stavu digitálneho pinu.**

pin: Číslo pinu, napríklad pin 3. value:

HIGH alebo LOW.

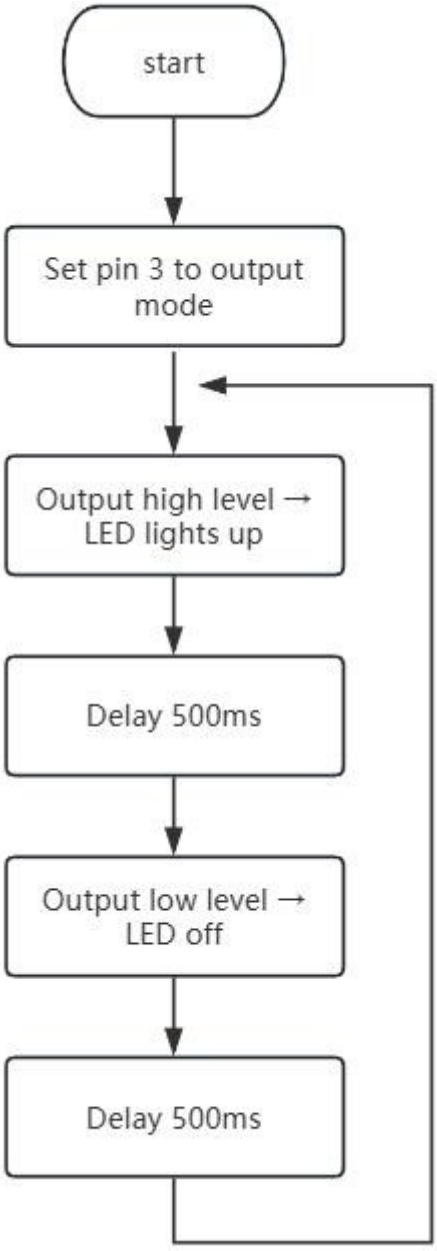
V tomto prípade nastavenie D3 na HIGH zapne LED; nastavenie na LOW LED vypne.

(7) delay(ms) je funkcia na oneskorenie.

Parameter je v milisekundách (ms). Napríklad delay(1000) znamená 1 sekundu. Počas delay() sa program pozastaví.

delay(500) pozastaví program na 500 milisekúnd (0,5 sekundy).

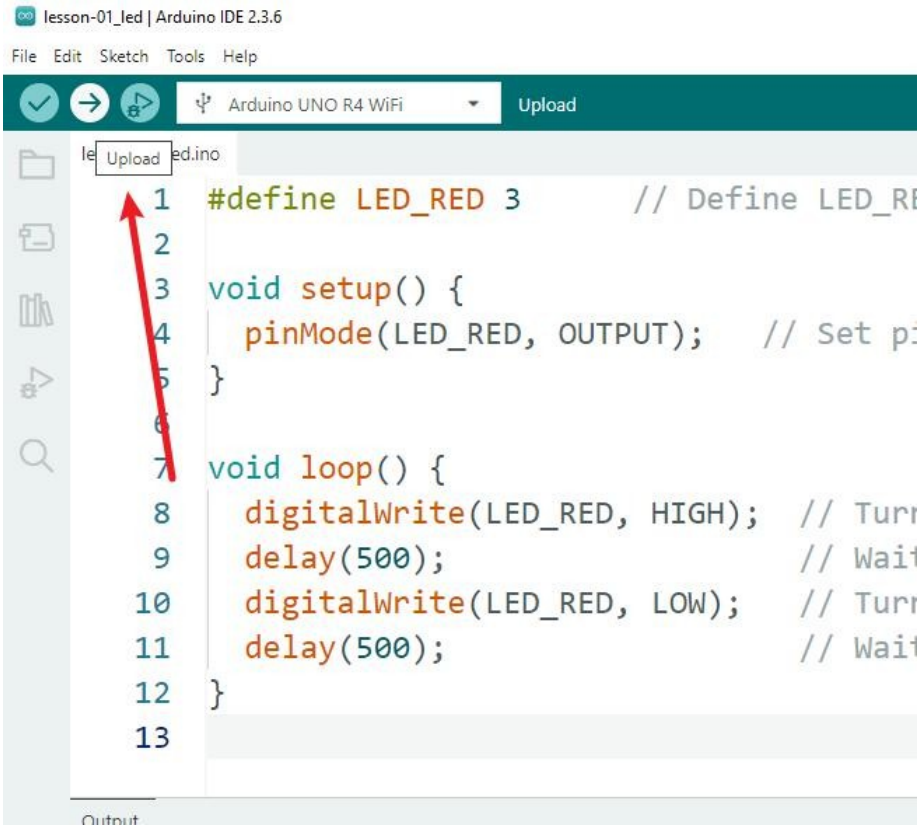
(8) Celkový diagram logického toku kódu



5. Spustite program a sledujte výsledky

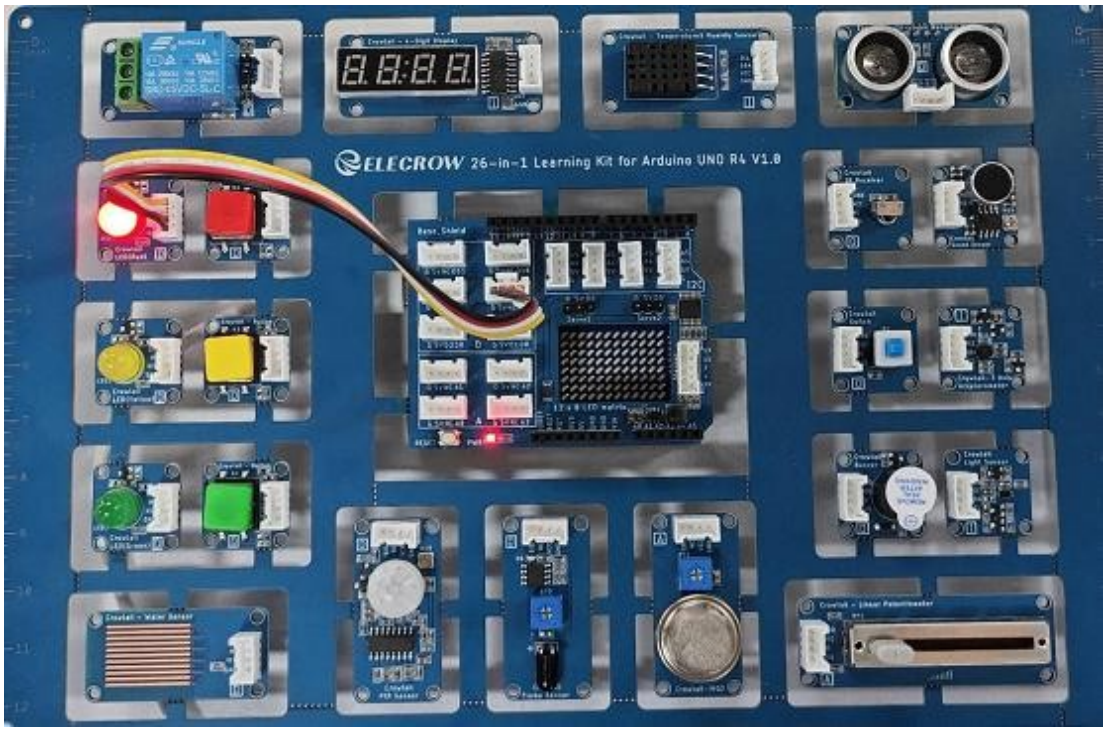
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Stiahnuť“:



(2) Uvidíte:

LED dióda sa rozsvieti raz za 0,5 sekundy.



Lekcia 02 – Tlačidlo

Úvod

V tejto lekcii budeme pokračovať v skúmaní toho, ako používať digitálny vstup čítaním signálov z tlačidla a ich využitím na ovládanie zapínania a vypínania LED diódy. V reálnych aplikáciách tlačidlá často spôsobujú jav nazývaný „odskok“, preto sa táto lekcia venuje aj princípom a implementácii techník potlačenia odskoku. Na konci tejto lekcie pochopíte

celý pracovný postup od „čítania tlačidla → logického rozhodovania → ovládania výstup."

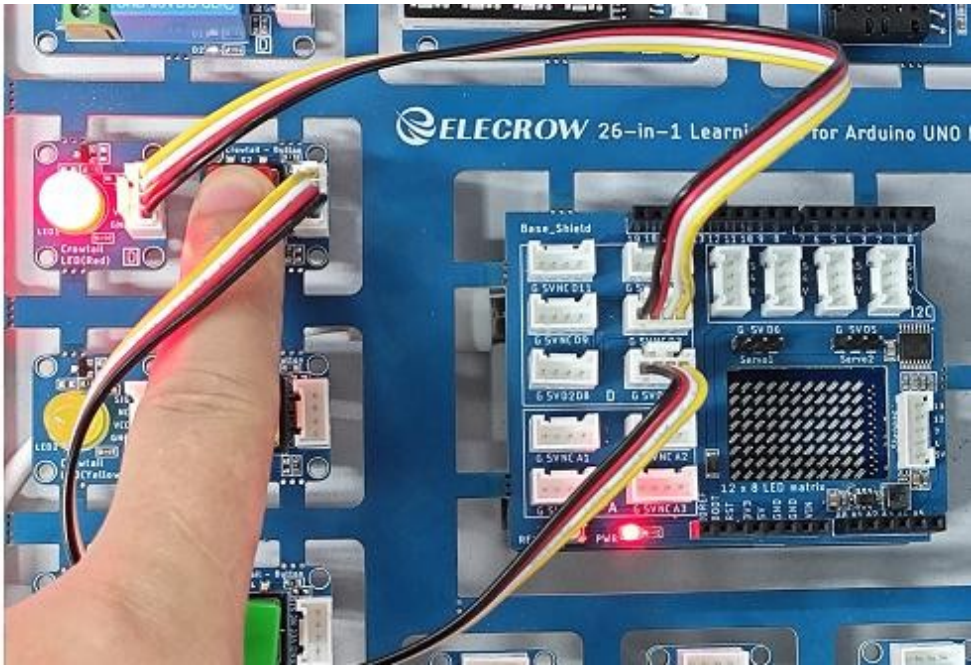
Konečný výsledok bude: LED sa rozsvieti po stlačení tlačidla a zhasne po jeho uvoľnení.

Ciele výučby

1. Naučiť sa čítať digitálne vstupné signály.
2. Porozumieť základným princípom odstraňovania odskokov tlačidiel. 3. Implementovať logiku LED ovládanú tlačidlom: vstup → rozhodnutie → výstup.

Náhľad výsledku

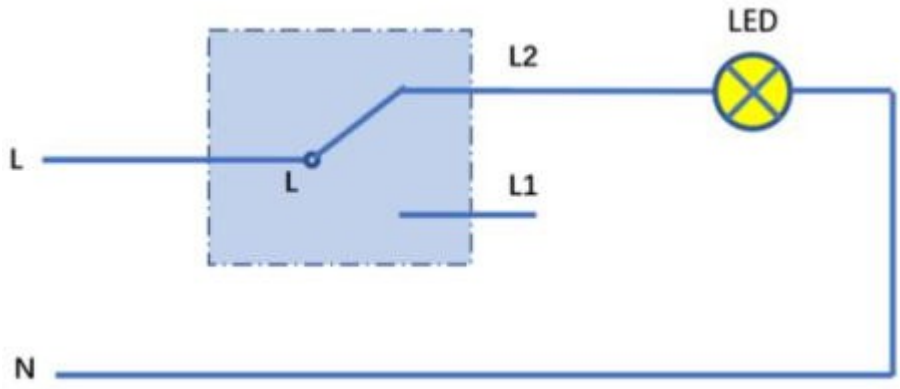
Po stlačení tlačidla sa LED rozsvieti.



1. Vysvetlenie princípu

①Ako funguje tlačidlo.

Tlačidlový modul funguje ako **jednopólový prepínač s dvoma polohami (SPDT)**. Ako je znázornené na schéme: pri stlačení tlačidla sa prepínač pripojí k svorke **L2**, čím sa uzavrie obvod; pri uvoľnení tlačidla sa prepínač pripojí k svorke **L1**, čím sa obvod otvorí.



②Konštrukcia tlačidlového modulu.

Tlačidlový modul Crowtail sa zvyčajne skladá z nasledujúcich častí:

SIG (Signal): Vysiela nízku úroveň (LOW), keď je tlačidlo stlačené, a vysokú úroveň (HIGH), keď je uvoľnené.

NC: Nepripojené; nezúčastňuje sa na obvode. VCC: Kladné

napájanie používané na napájanie modulu.

GND: Zem, záporný pól napájacieho zdroja a spoločný referenčný bod pre všetky zariadenia.

Tlačidlo je v podstate mechanický spínač. Pri jeho stlačení alebo uvoľnení môžu nastať krátke elektrické oscilácie – známe ako odskok tlačidla (Bounce) – ktoré spôsobia rýchle prepínanie signálu medzi stavmi HIGH a LOW.

Aby sa zabránilo nesprávnym hodnotám v programe, je potrebné použiť odskok (Debounce), aby bol stav tlačidla čítaný systémom stabilný a spoľahlivý.

③Prehľad vstupov/výstupov Arduino UNO R4.

Digitálne piny (D2, D3, D4 atď.) je možné nakonfigurovať ako vstupy alebo výstupy.

Ak je pin nastavený na INPUT_PULLUP, aktivuje sa interný pull-up rezistor. V tomto režime je pin štandardne nastavený na stav HIGH a pri stlačení tlačidla sa signál stiahne na GND (LOW).

④Postup čítania tlačidla.

A :**pinMode(pin, INPUT_PULLUP)**

Aktivuje interný pull-up rezistor, čím sa vstup stáva stabilnejším a menej citlivým na rušenie. B:

digitalRead(pin)

Načíta stav tlačidla. Úroveň HIGH znamená, že tlačidlo nie je stlačené, zatiaľ čo úroveň LOW znamená, že

tlačidlo je stlačené.

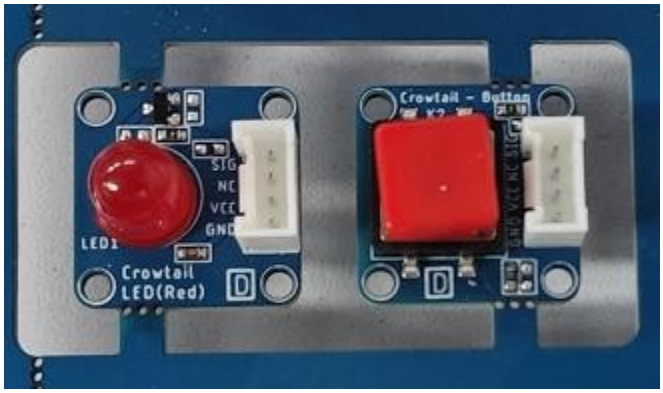
C: **Ovládanie LED na základe načítanej hodnoty.**

Na základe zistenej hodnoty nastavte LED na stav HIGH alebo LOW, čím LED zodpovedajúcim spôsobom zapnete alebo vypnete.

2. Potrebné moduly

Crowtail – LED × 1

Crowtail – tlačidlo × 1



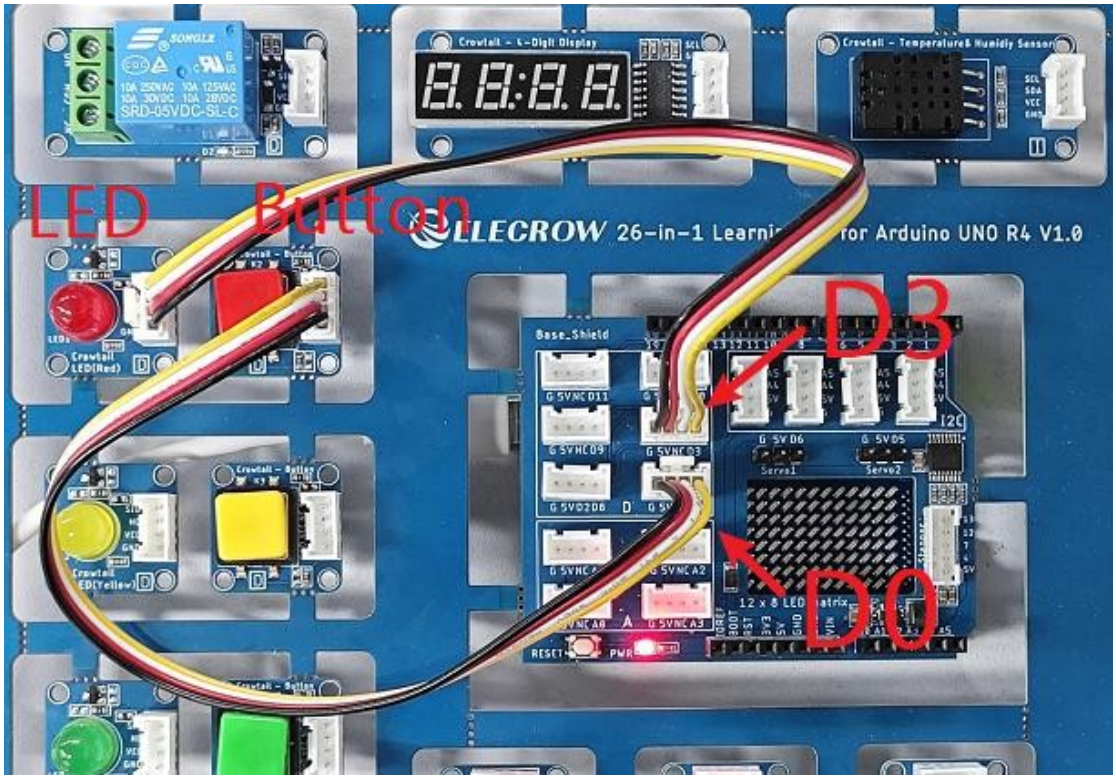
3. Spôsob zapojenia

LED Crowtail → port DIGITAL D3 Tlačidlo

Crowtail → port DIGITAL D0 Špecifikácia

štvorportového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červená) → 5V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3/D0



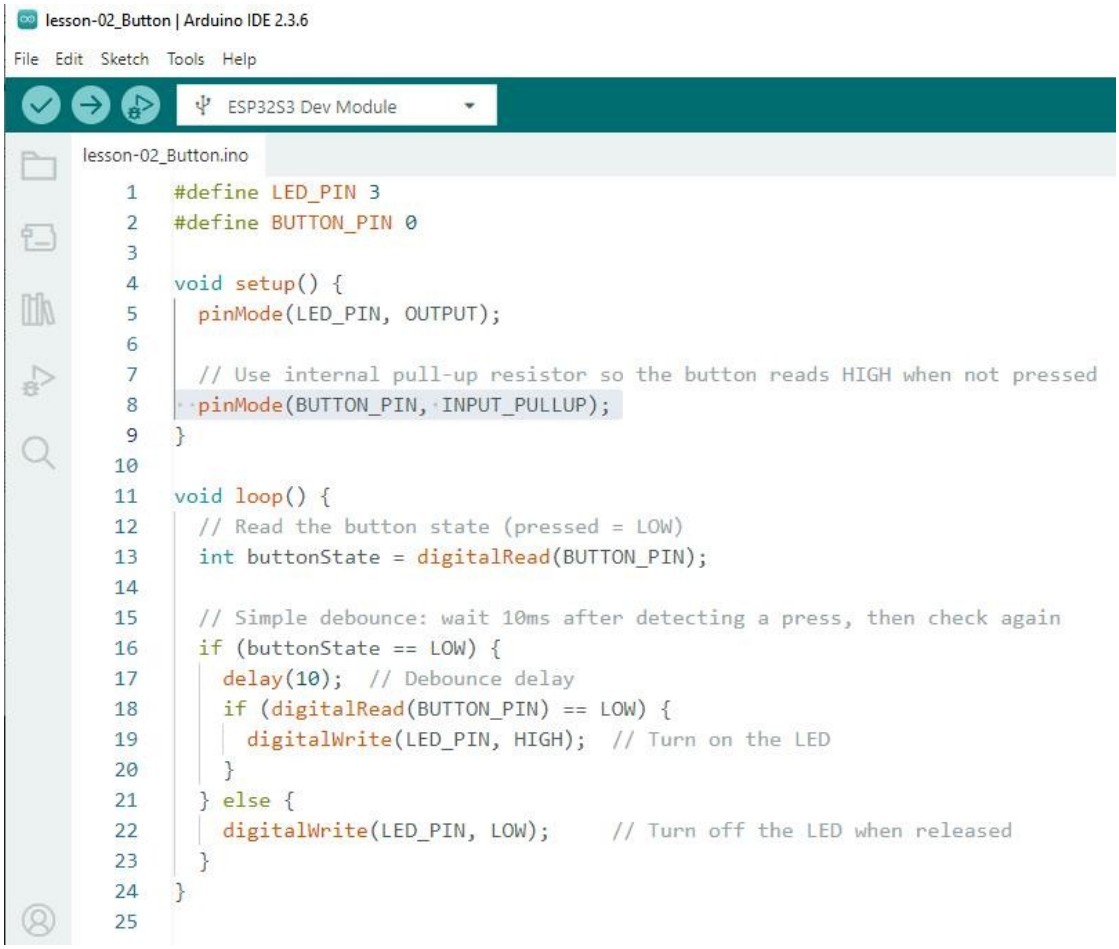
4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad kódu.

Odkaz na GitHub：

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-02_Button

Spustite program Lekcia 2 pomocou prostredia Arduino IDE.



Vysvetlenie kľúčového kódu

(1) Najprv definujte digitálne piny, aby bol kód čitateľnejší.

```
#define LED_PIN 3
#define BUTTON_PIN 0
```

Tu je LED priradená k digitálnemu pinu 3 a tlačidlo k digitálnemu pinu 0.

(2) Nastavte pin tlačidla tak, aby používal interný pull-up rezistor (INPUT_PULLUP).

```
pinMode(BUTTON_PIN, INPUT_PULLUP);
```

Vďaka tejto konfigurácii má tlačidlo v predvolenom stave hodnotu HIGH a po stlačení sa pripojí k GND a má hodnotu LOW.

Výhody:

A: Nie je potrebný žiadny externý rezistor.

B: Lepšia odolnosť voči elektrickému šumu a rušeniu. C:

Jednoduchšia a intuitívnejšia logika tlačidla.

(3) Digitálne čítanie pomocou digitalRead()

Používa sa na čítanie stavu pinu:

```
int buttonState = digitalRead(BUTTON_PIN);
```

načíta logickú úroveň BUTTON_PIN a uloží hodnotu do premennej buttonState. HIGH znamená,

že tlačidlo nie je stlačené.

LOW znamená, že tlačidlo je stlačené.

(4) Odstránenie odskoku tlačidla (Debounce)

Pri stlačení tlačidla môže dôjsť k krátkemu odskoku, čo môže spôsobiť, že program omylom zaznamená viacero stlačení.

Riešenie:

```
delay(10);
if (digitalRead(BUTTON_PIN) == LOW)
```

Inými slovami:

①Zistiť stlačenie tlačidla;
②Počkajte 10 ms;
③Znovu prečítajte stav tlačidla na potvrdenie.

(5) Logika riadenia stavu.

Na kontrolu stavu tlačidla použite príkaz if. Ak je stav LOW, zapnite LED; ak je stav HIGH, vypnite LED.

```
if (buttonState == LOW) {
      delay(10);      // Oneskorenie na odstránenie odskoku
      if (digitalRead(BUTTON_PIN) == LOW) {
          digitalWrite(LED_PIN, HIGH);        // Zapni LED
      }
  } else {

      digitalWrite(LED_PIN, LOW);              // Vypnúť LED po uvoľnení
  }
```

Základná štruktúra príkazu if:

```
if (podmienka) {
      // Podmienka je pravdivá — tento kód sa spustí
}
```

Podmienka musí mať hodnotu „true“ alebo „false“.

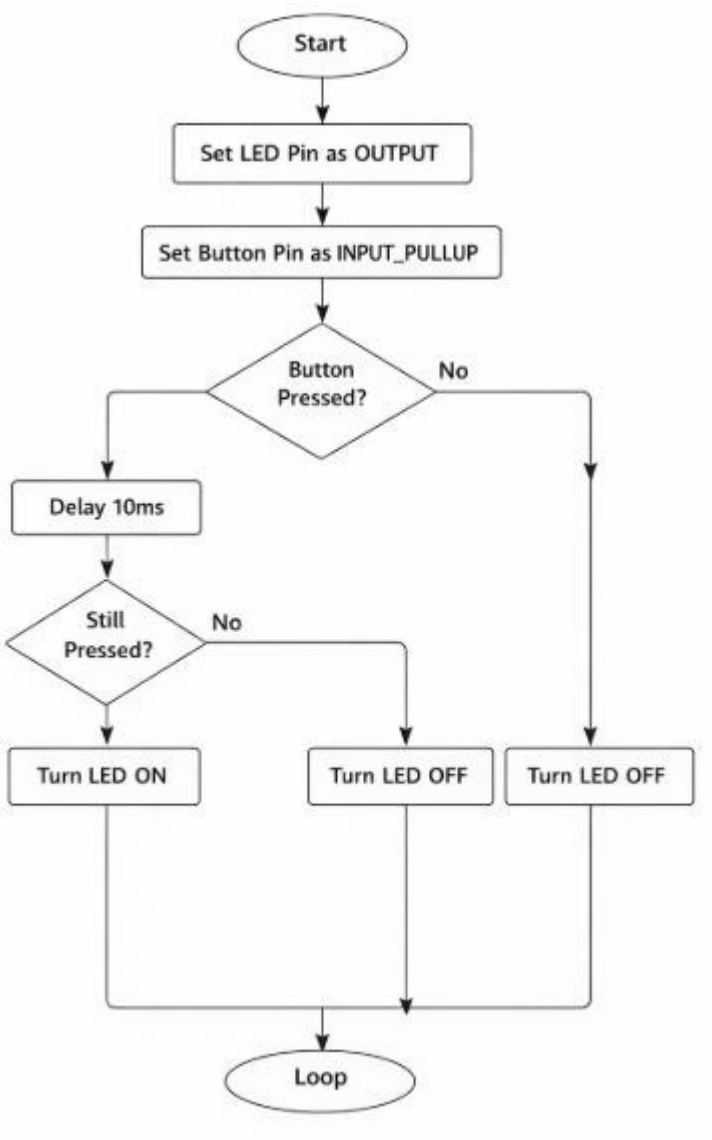
Kód vnútri { } sa spustí len vtedy, ak je podmienka pravdivá.

② Štruktúra príkazu if...else

```
if (podmienka) {
    // Podmienka je pravdivá — tento kód sa spustí
} else {
    // Podmienka je nepravdivá — namiesto toho sa spustí tento kód
}
```

„else“ znamená „inak“ a používa sa na vykonanie alternatívneho bloku kódu, ak nie je splnená podmienka.

(6) **Celkový diagram logiky kódu**

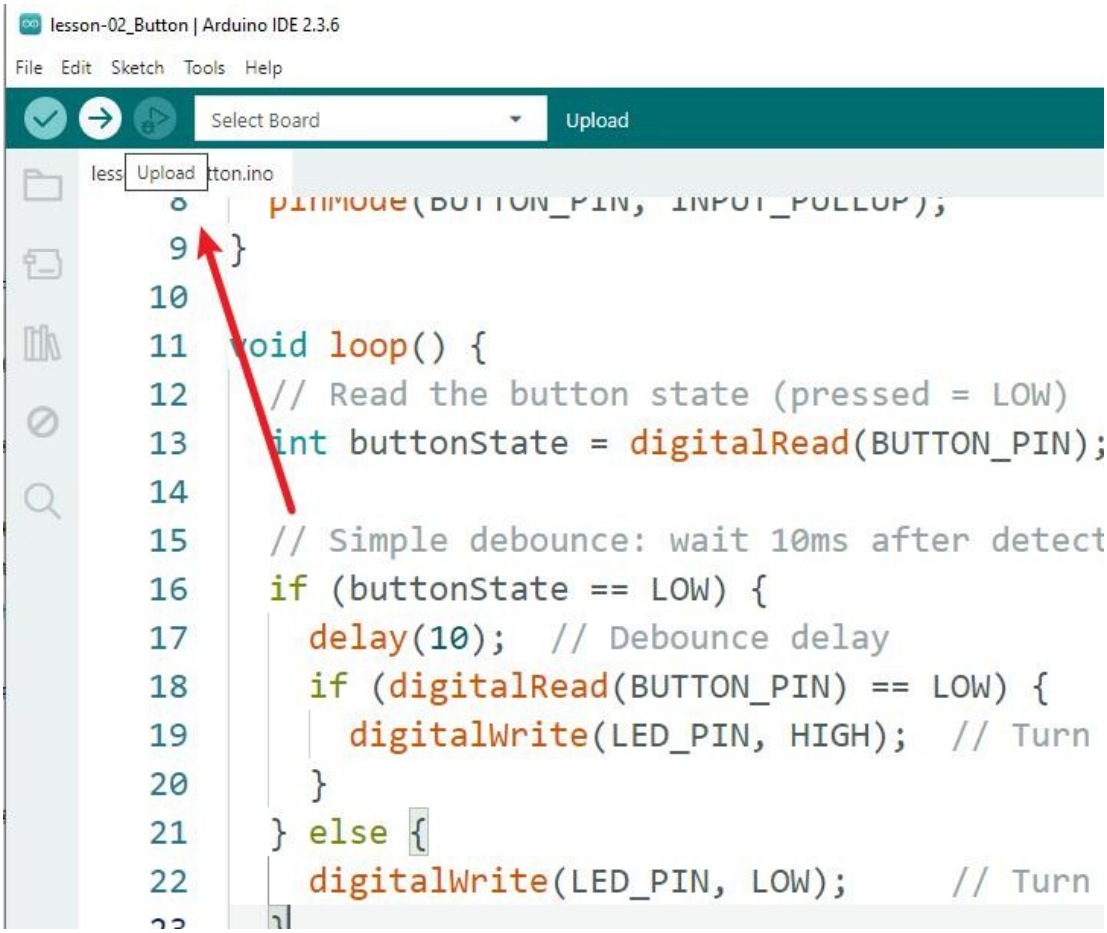


5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač

a dokončite základnú konfiguráciu nahrávania.

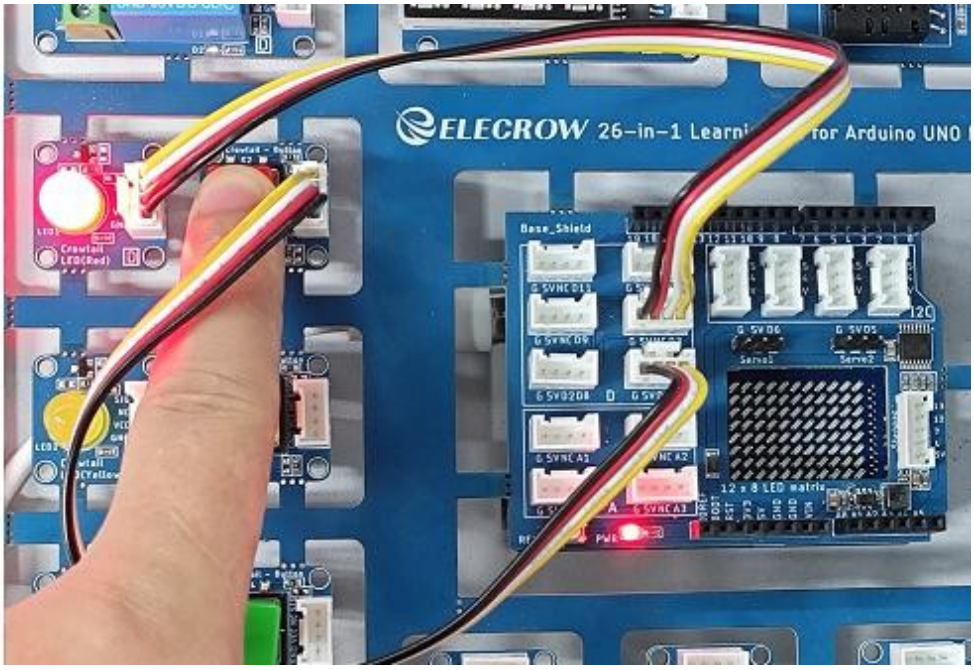
Kliknite na „Download“:



(2) Uvidíte:

Keď je tlačidlo stlačené, LED sa rozsvieti; keď je tlačidlo uvoľnené, LED zhasne

—žiadne falošné spustenia, žiadne blikanie a stabilné, spoľahlivé ovládanie.


ELECROW 26-in-1 Learni
for Arduino UNO
Base_Shield
12 x 8 LED matrix

Lekcia 03 – Bzučiak

Úvod

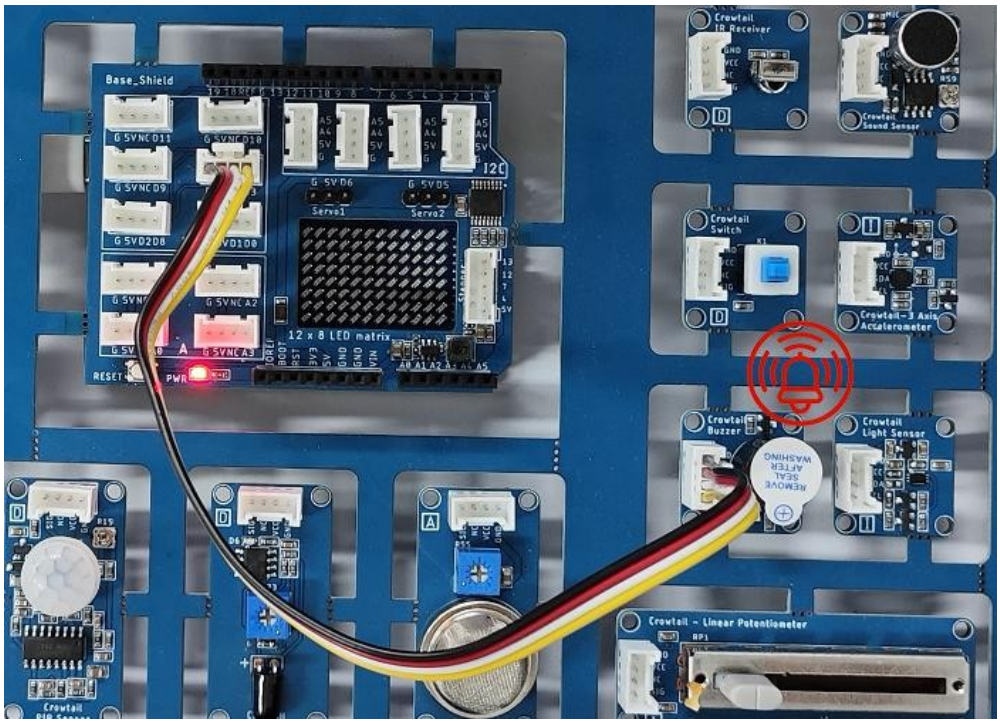
V tejto lekcii sa naučíme, ako pomocou Arduina ovládať aktívny bzučiak, aby vydával rôzne výstražné zvuky a rytmy. Aktívny bzučiak má vstavaný oscilátor, takže automaticky generuje zvuk, keď Arduino vysiela signál HIGH, a zastaví sa, keď je výstup LOW. Vďaka tomu je ideálny pre projekty pre začiatočníkov, ako sú oznamovacie zvuky, alarmy a odpočítavacie výstrahy.

Ciele

1. Porozumieť tomu, ako funguje aktívny bzučiak: výstup HIGH ho spustí.
2. Naučiť sa spúšťať a zastavovať bzučiak pomocou digitalWrite().
3. Použiť funkciu delay() na vytvorenie rôznych zvukových vzorov a rytmov.
4. Navrhujte jednoduché vzory upozornení, ako sú krátke pípnutia alebo zvukové signály.

Náhľad výsledku

Bzučiak vydáva rytmické, vzorované zvuky.



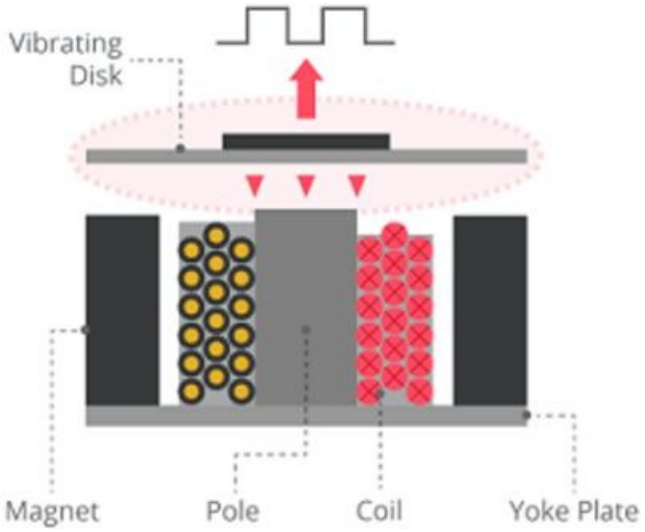
1. Vysvetlenie princípu

①Ako funguje aktívny bzučiak

Aktívny bzučiak má zabudovaný oscilátorový obvod. Pokiaľ je napájaný (napríklad 5 V), vnútorný oscilátor automaticky generuje signál s pevnou frekvenciou, vďaka čomu bzučiak nepretržite vydáva zvuk.

Z tohto dôvodu aktívny bzučiak začne znieť hneď po zapnutí. Na generovanie tónu nepotrebuje Arduino a nedokáže prehrávať melódie – vydáva len jediný, konštantný výstražný zvuk „pípnutia“.

Arduino jednoducho ovláda zapnutie alebo vypnutie bzučiaka vyslaním signálu HIGH alebo LOW.



2. Potrebné moduly

Crowtail – bzučiak × 1

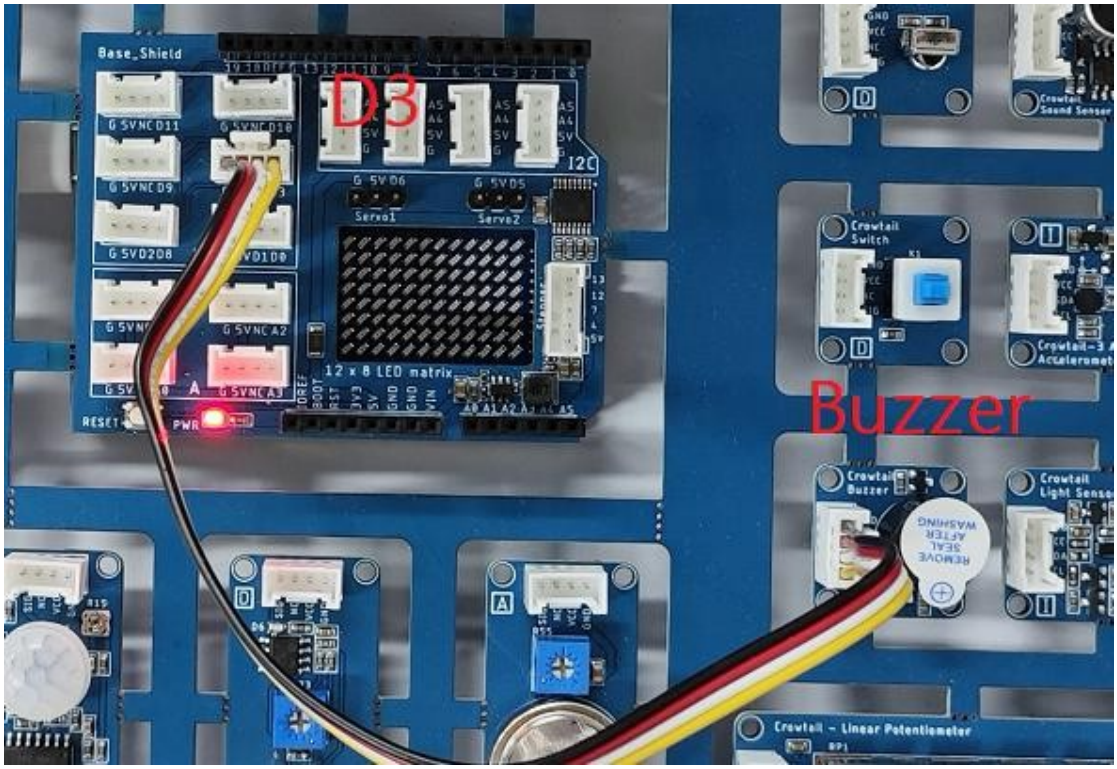
3. Spôsob zapojenia

Crowtail -- Bzučiak → port DIGITAL D3

Crowtail -- Tlačidlo → port DIGITAL D0

Špecifikácia štvorportového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červený) → 5V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3



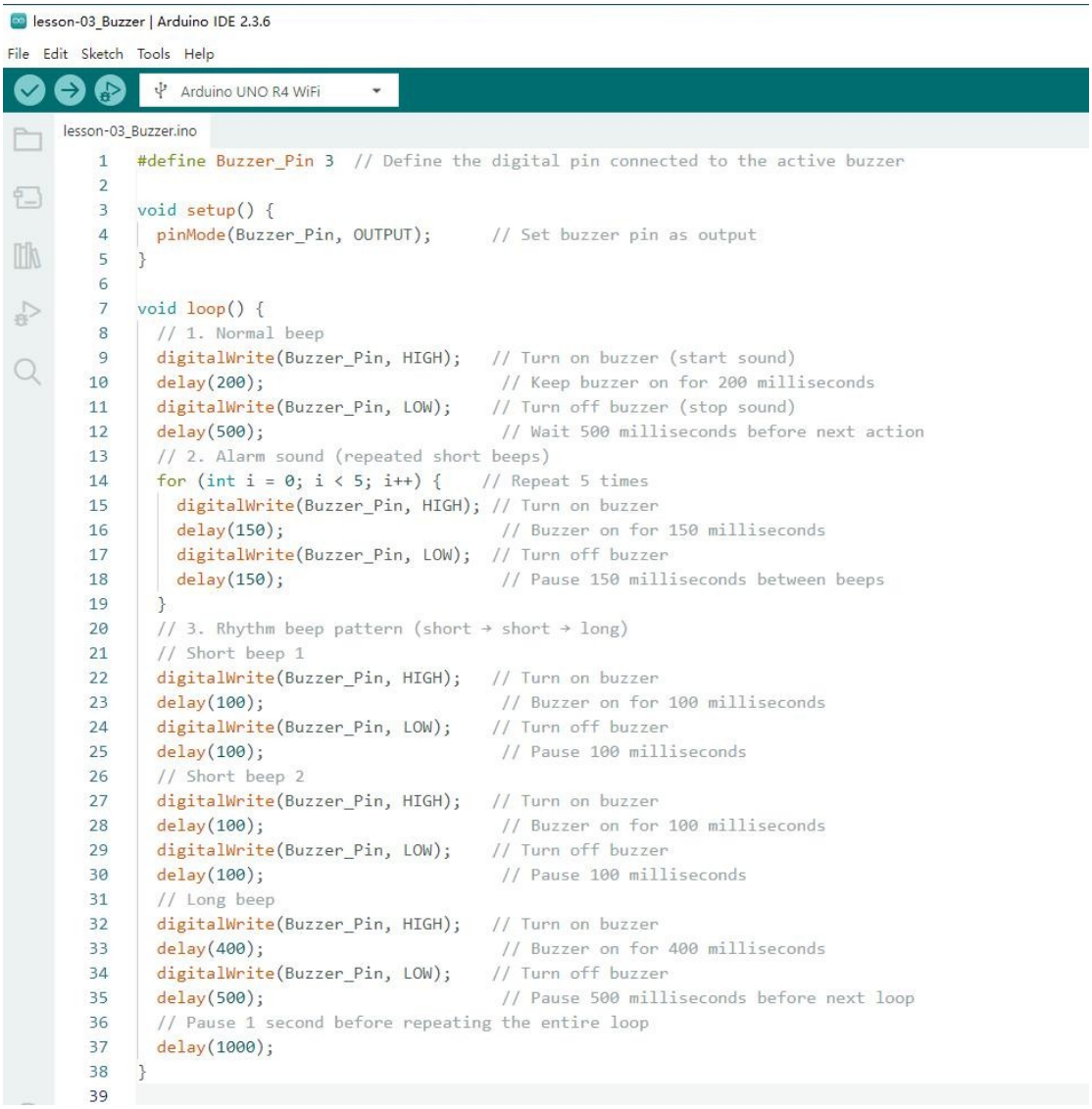
4. Vysvetlenie príkladu

Kliknite na nižšie uvedený odkaz a stiahnite si oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-03_Buzzer

Otvorte program Lekcia 3 pomocou Arduino IDE.




```ino
#define Buzzer_Pin 3  // Define the digital pin connected to the active buzzer

void setup() {
  pinMode(Buzzer_Pin, OUTPUT);      // Set buzzer pin as output
}

void loop() {
  // 1. Normal beep
  digitalWrite(Buzzer_Pin, HIGH);   // Turn on buzzer (start sound)
  delay(200);                       // Keep buzzer on for 200 milliseconds
  digitalWrite(Buzzer_Pin, LOW);    // Turn off buzzer (stop sound)
  delay(500);                       // Wait 500 milliseconds before next action
  // 2. Alarm sound (repeated short beeps)
  for (int i = 0; i < 5; i++) {    // Repeat 5 times
    digitalWrite(Buzzer_Pin, HIGH); // Turn on buzzer
    delay(150);                     // Buzzer on for 150 milliseconds
    digitalWrite(Buzzer_Pin, LOW);  // Turn off buzzer
    delay(150);                     // Pause 150 milliseconds between beeps
  }
  // 3. Rhythm beep pattern (short → short → long)
  // Short beep 1
  digitalWrite(Buzzer_Pin, HIGH);   // Turn on buzzer
  delay(100);                       // Buzzer on for 100 milliseconds
  digitalWrite(Buzzer_Pin, LOW);    // Turn off buzzer
  delay(100);                       // Pause 100 milliseconds
  // Short beep 2
  digitalWrite(Buzzer_Pin, HIGH);   // Turn on buzzer
  delay(100);                       // Buzzer on for 100 milliseconds
  digitalWrite(Buzzer_Pin, LOW);    // Turn off buzzer
  delay(100);                       // Pause 100 milliseconds
  // Long beep
  digitalWrite(Buzzer_Pin, HIGH);   // Turn on buzzer
  delay(400);                       // Buzzer on for 400 milliseconds
  digitalWrite(Buzzer_Pin, LOW);    // Turn off buzzer
  delay(500);                       // Pause 500 milliseconds before next loop
  // Pause 1 second before repeating the entire loop
  delay(1000);
}
```

Vysvetlenie kódov tlačidiel

(1) Najprv definujte digitálne piny, aby bol kód čitateľnejší.

```
#define Buzzer_Pin 3
```

Účelom #define Buzzer_Pin 3 je dať číslu pinu zmysluplný „názov“. Buzzer_Pin predstavuje digitálny pin 3, čo znamená, že signálna linka bzučiaka je pripojená k D3. Výhody tohto prístupu:

Ak neskôr presuniete bzučiak na iný pin, stačí zmeniť len číslo pinu. Žiadne ďalšie časti kódu nie je potrebné upravovať.

(2) Konfigurácia režimu pinu.

```
pinMode(Buzzer_Pin, OUTPUT);                // Nastaviť pin bzučiaka ako výstup
```

Nastavte výstup bzučiaka do režimu OUTPUT, aby Arduino mohlo ovládať zapnutie a vypnutie bzučiaka.

(3) Bežný pípací tón (štandardný výstražný zvuk).

```
digitalWrite(Buzzer_Pin, HIGH);          // Zapnúť bzučiak (spustiť zvuk)

delay(200);                                    // Nechajte bzučiak zapnutý po dobu 200 milisekúnd
digitalWrite(Buzzer_Pin, LOW);           // Vypnúť bzučiak (zastaviť zvuk)

delay(500);                                    // Počkať 500 milisekúnd pred ďalšou akciou
```

digitalWrite(HIGH) zapne bzučiak a delay(200) ho nechá znieť 0,2 sekundy. digitalWrite(LOW) potom bzučiak vypne, nasleduje delay(500) na pauzu 0,5 sekundy. Táto sekvencia vytvára jednoduchý vzor upozornenia „jedno pípnutie, jedna pauza“.

(4) Zvuk alarmu (krátke, rýchle pípanie)

```
for (int i = 0; i < 5; i++) {          // Opakovať 5-krát

    digitalWrite(Buzzer_Pin, HIGH);

    delay(150);

    digitalWrite(Buzzer_Pin, LOW);

    delay(150);

}
```

Inými slovami:

V tejto časti sa pomocou cyklu „for“ automaticky päťkrát opakuje akcia „vydaj pípnutie na 150 ms, pauza na 150 ms“, čím vznikne rýchly, nepretržitý zvukový signál.

Účelom cyklu je opakovať tú istú akciu bez toho, aby bolo potrebné písať ten istý kód znova a znova.

(5) Rytmický vzor pípania (krátke → krátke → dlhé)

```
// Krátke pípnutie 1

digitalWrite(Buzzer_Pin, HIGH);          // Zapnutie bzučiaka
delay(100);                                    // Zapnutie bzučiaka na 100 milisekúnd
digitalWrite(Buzzer_Pin, LOW);           // Vypnúť bzučiak

delay(100);                                    // Pauza 100 milisekúnd

// Krátke pípnutie 2

digitalWrite(Buzzer_Pin, HIGH);          // Zapnutie bzučiaka
delay(100);                                    // Zapnutie bzučiaka na 100 milisekúnd
digitalWrite(Buzzer_Pin, LOW);           // Vypnúť bzučiak

delay(100);                                    // Pauza 100 milisekúnd
```

```
// Dlhé pípnutie
digitalWrite(Buzzer_Pin, HIGH);          // Zapnúť bzučiak
delay(400);                                    // Bzučiak zapnutý na 400
                                               milisekúnd
digitalWrite(Buzzer_Pin, LOW);           // Vypnúť bzučiak
delay(500);                                    // Pauza 500 milisekúnd pred ďalšou slučkou
```

Táto časť kombinuje tri rôzne dĺžky zapnutia/vypnutia, čím vytvára rytmus: dva krátke zvuky „píp-píp“, po ktorých nasleduje jeden dlhší zvuk „—píp—“.

Úpravou dĺžky oneskorenia môžete dosiahnuť, aby bzučiak vydával rôzne rytmické zvuky efekty.

(6) Na záver urobte pauzu na 1 sekundu.

```
delay(1000);
```

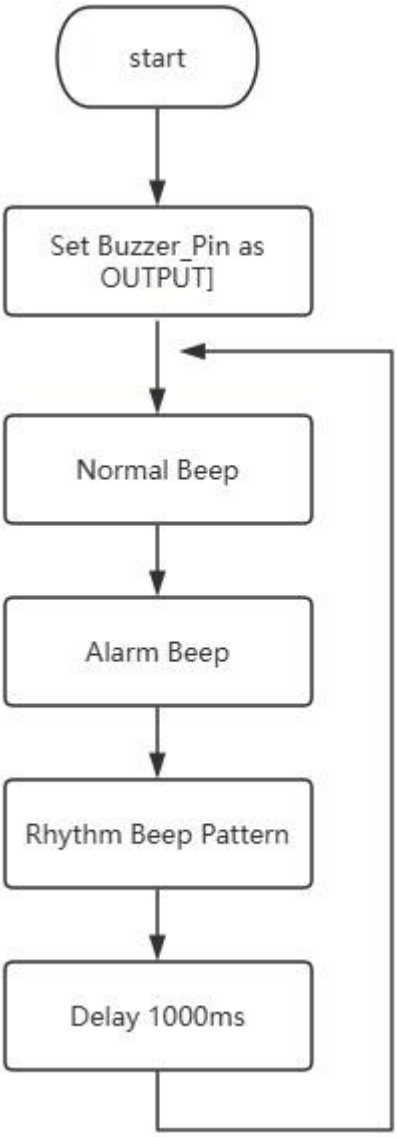
Táto pauza vytvára jasné oddelenie medzi rôznymi rytmami.

Program nepretržite prehráva tri rôzne vzory bzučiaka. Najprv vydá normálne upozornenie: bzučiak znie 0,2 sekundy, potom je 0,5 sekundy ticho. Potom vstúpi do cyklu for, čím bzučiak päťkrát zopakuje rýchly vzor „zapnutý na 0,15 sekundy, vypnutý na 0,15 sekundy“, čím vytvorí rýchly alarmový zvuk „píp-píp-píp-píp-píp“.

Nakoniec prehráva rytmus krátky → krátky → dlhý: prvé dva krátke pípnutia trvajú po 0,1 sekundy, s 0,1-sekundovou pauzou medzi nimi, po ktorej nasleduje dlhý pípací tón trvajúci 0,4 sekundy a 0,5-sekundová pauza.

Po dokončení všetkých troch zvukových vzorov sa program na 1 sekundu pozastaví a potom začne od začiatku, pričom nepretržite opakuje celú sekvenciu v pevnom, opakujúcom sa vzore.

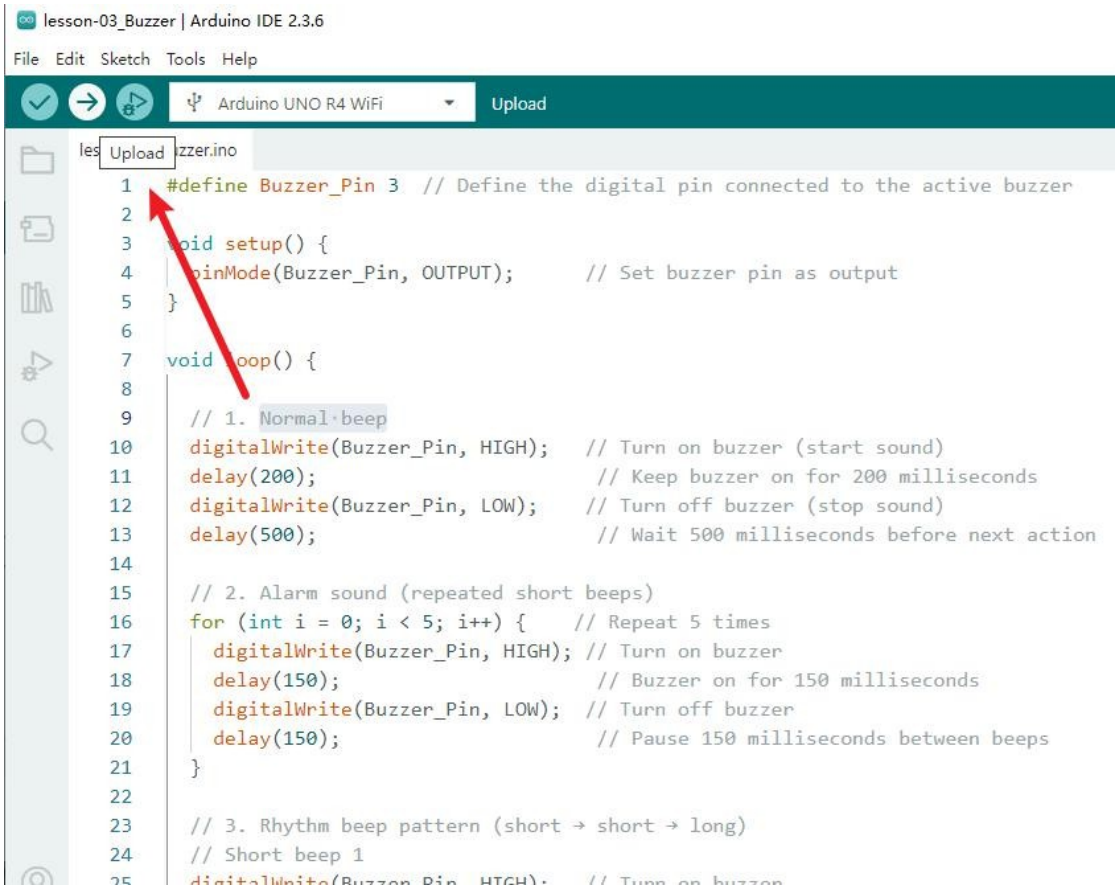
(7) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

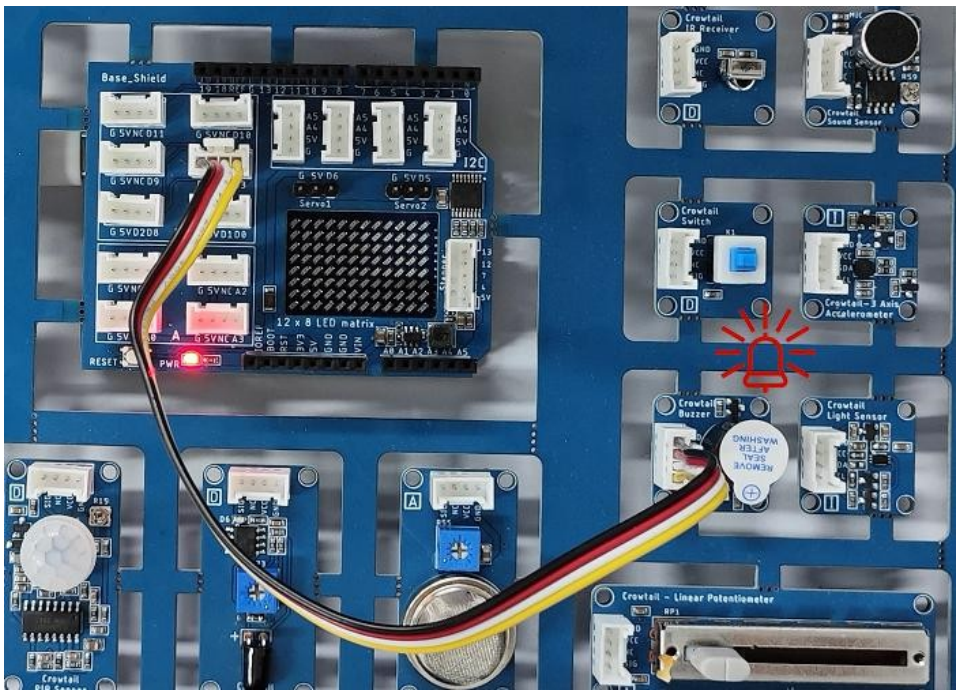
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Stiahnuť“:

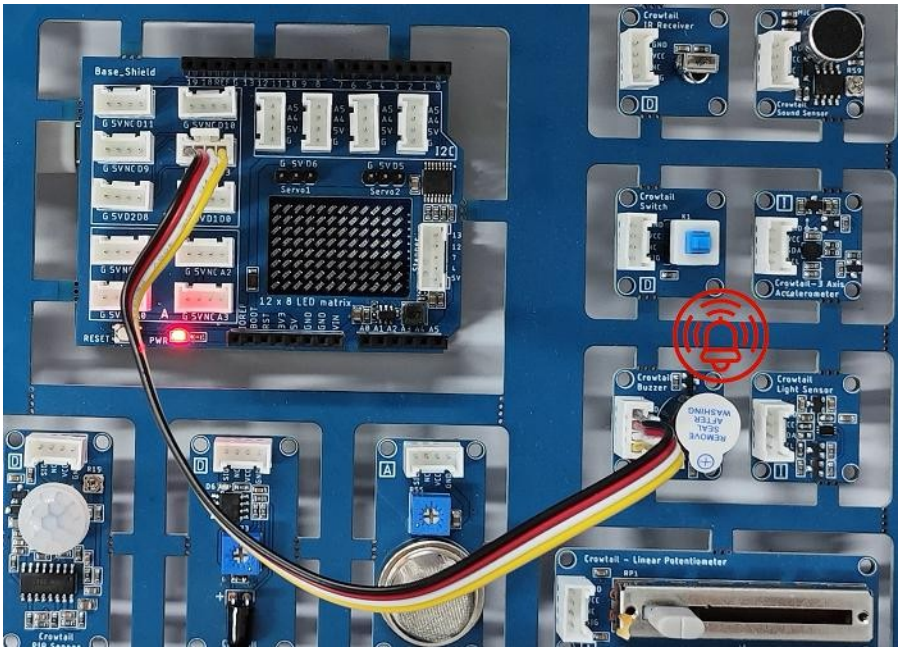


(2) Uvidíte:

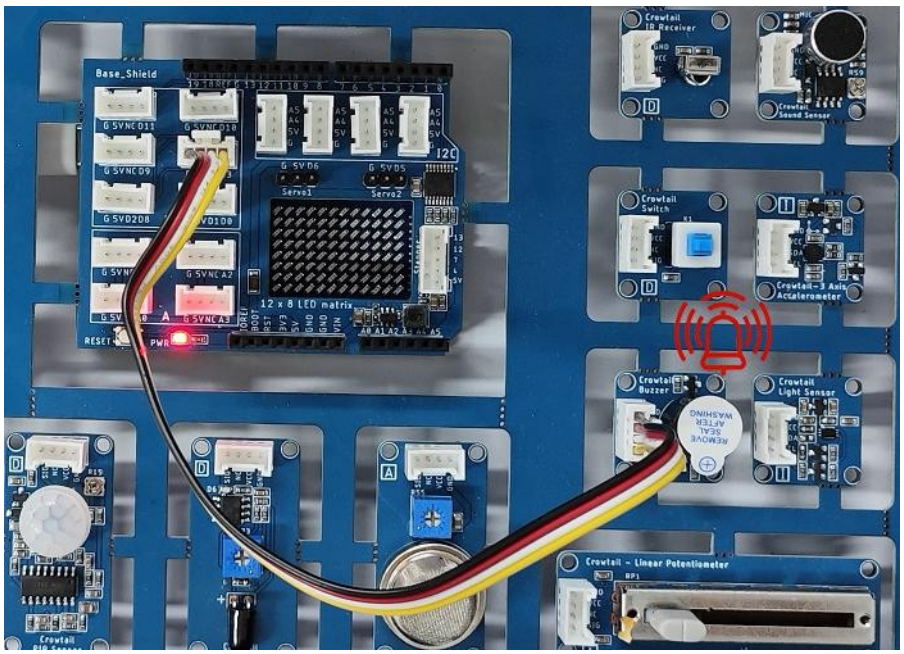
Keď sa program spustí, bzučiak vydáva zvuky s rôznymi rytmami. Najskôr zaznie bežný pípací tón.



Potom vydá rýchly alarmový zvuk.



Nakoniec vydáva rytmus s meniacim sa načasovaním a vzorom.



Lekcia 04 – Spínač

Úvod

V tejto lekcii sa budeme venovať ďalšiemu bežne používanému digitálnemu vstupnému zariadeniu – prepínaču.

Na prvý pohľad sa prepínače a tlačidlá podobajú, ale v reálnom použití sa správajú veľmi odlišne:

Tlačidlo: Stlačenie → spustí akciu; uvoľnenie → automaticky sa vráti do pôvodného stavu.

Prepínač: Prepnutie → stav sa zachová a automaticky sa neobnoví.

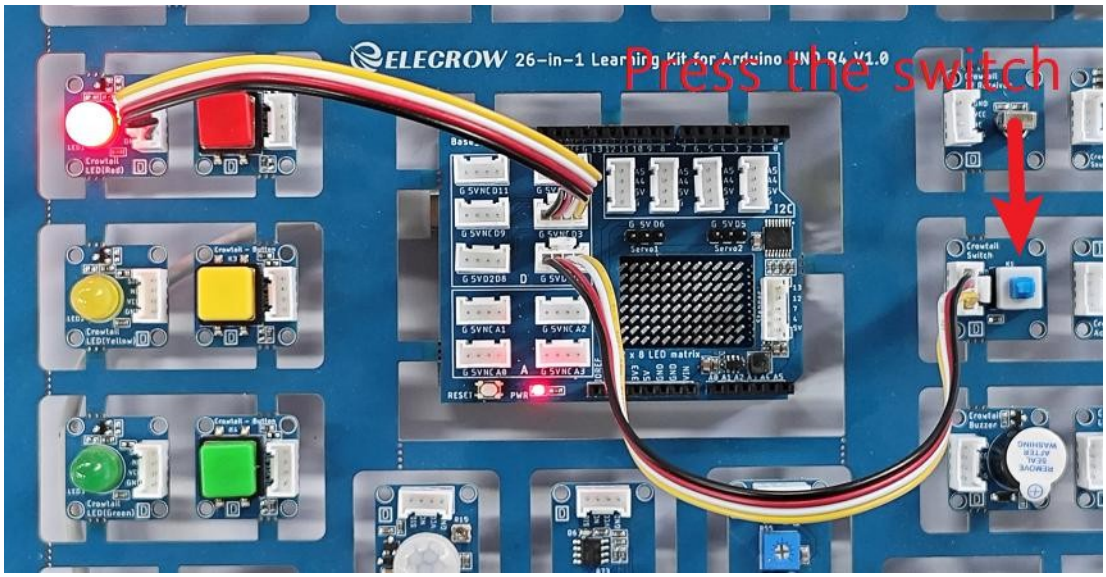
V tejto lekcii sa naučíte, ako čítať signál digitálneho prepínača a používať ho na ovládanie LED diódy, čím implementujete jednoduchú, ale veľmi bežnú ovládaciu funkciu.

Ciele výučby

1. Porozumieť štruktúre a princípom fungovania normálne otvorených a normálne uzavretých prepínačov.
2. Naučiť sa čítať digitálne vstupné signály prepínača.
3. Na implementáciu základnej riadiacej logiky použite vstup spínača.
4. Jasne rozlišujte medzi základným rozdielom medzi tlačidlom a prepínačom.

Náhľad výsledku

Keď je prepínač zapnutý, LED dióda svieti.

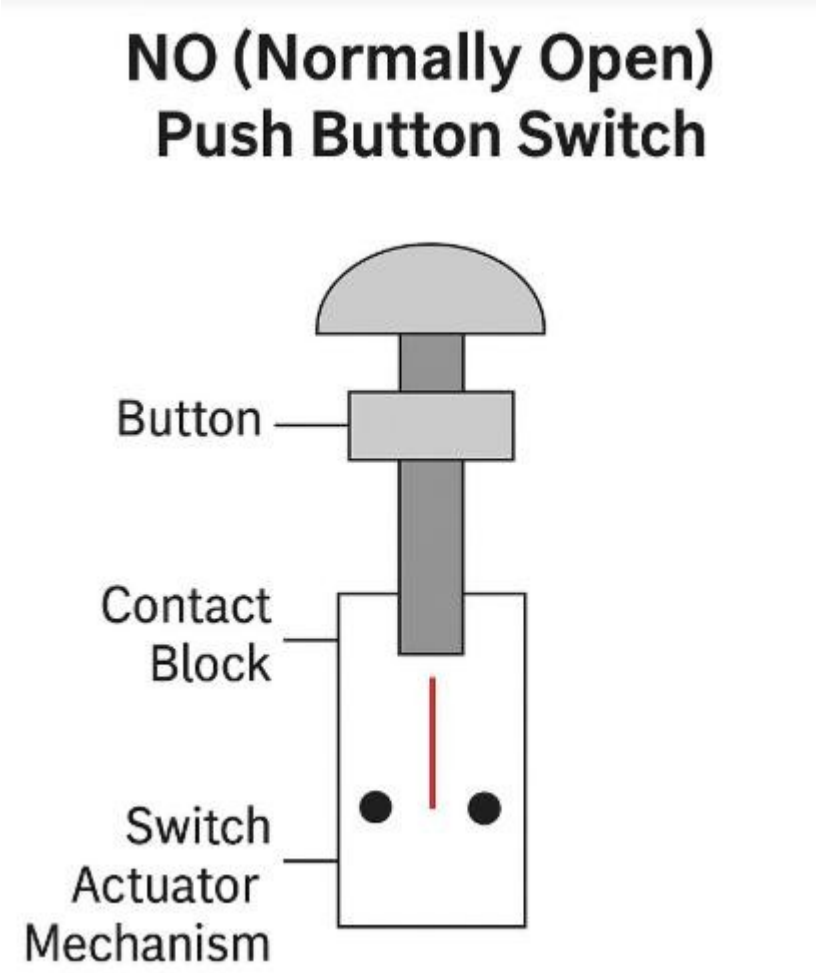


1. Vysvetlenie princípu

①Modul spínača

Spínač je mechanické zariadenie, ktoré si zachováva svoj stav. Po prepnutí zostane v danej polohe a automaticky sa nevráti späť.

Spínač použitý v tomto prípade má konfiguráciu normálne otvorenú. Keď nie je stlačený, obvod zostáva otvorený; keď je stlačený, obvod sa uzavrie.



Rozdiely medzi tlačidlom a prepínačom

Porovnanie	**Tlačidlo**	**Prepínač**
Správanie v stave	Vodí len počas stlačenia	Udrží svoj stav po prepnutí
Automatický návrat	Automaticky sa vráti do pôvodnej polohy	Nevráti sa automaticky
Typické použitie	Spúšťanie jednorazových akcií	Dlhodobé zapínanie/vypínanie zariadení
Odstránenie odskoku	Požadované	Zriedkavo potrebné (mechanicky stabilnejšie)

2. Požadované moduly

Crowtail - Spínač × 1

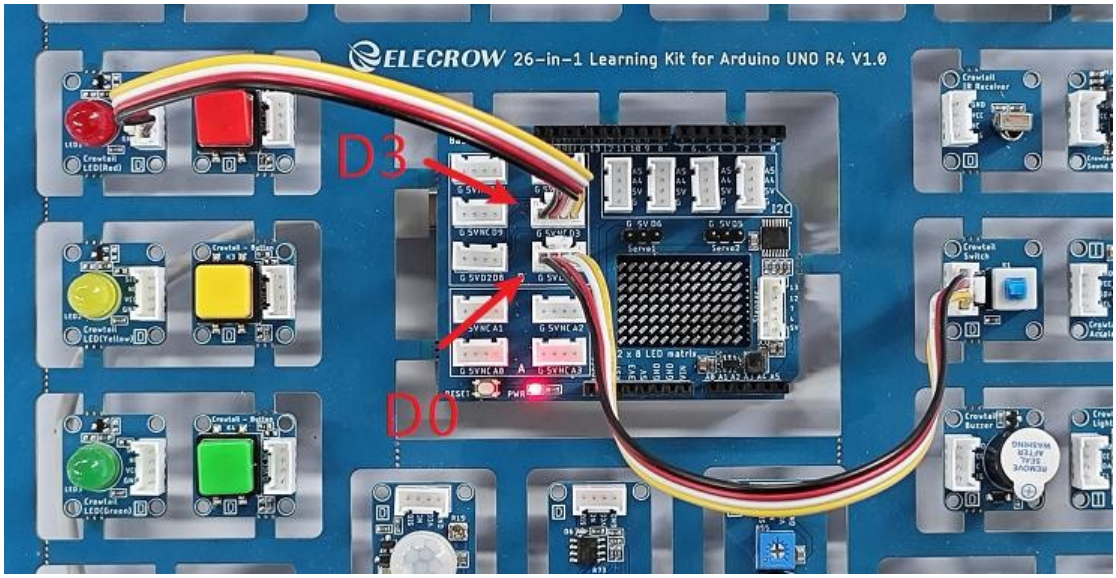
3. Spôsob zapojenia

Crowtail -- Spínač → Port DIGITAL D0 Crowtail

-- LED → Port DIGITAL D3 Špecifikácia

štvorportového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3/D0



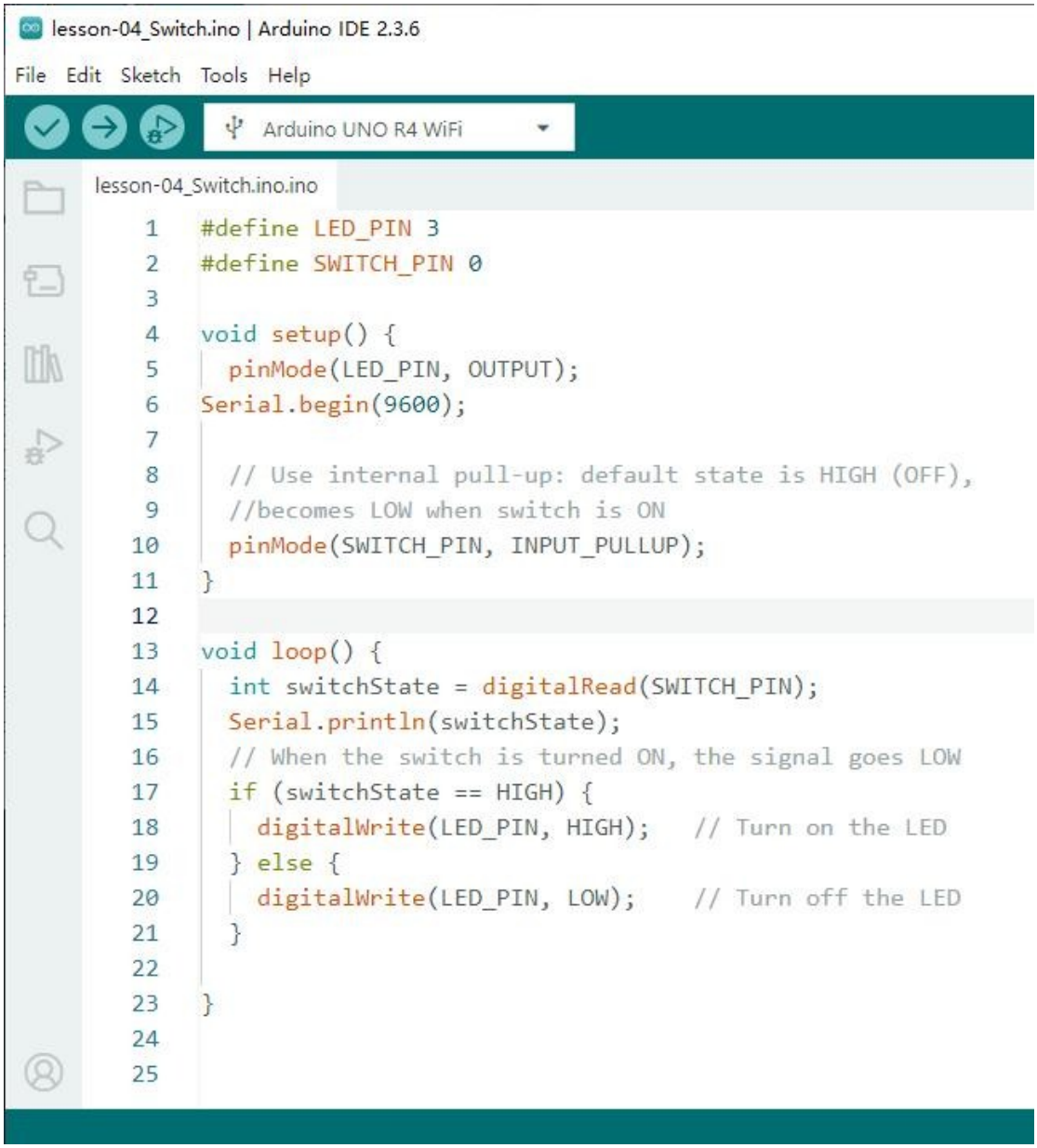
4. Vysvetlenie príkladu

Kliknite na odkaz nižšie a stiahnite si oficiálny príklad kódu. Odkaz

na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-04_Switch.ino

Otvorte program Lekcia 4 pomocou Arduino IDE.



Vysvetlenie kľúčového kódu

(1) Najskôr definujte digitálne piny, aby bol kód čitateľnejší.

```
#define LED_PIN 3

#define SWITCH_PIN 0
```

#define LED_PIN 3 a #define SWITCH_PIN 0 sa používajú na pridelenie výstižných „mien“ pinom. LED_PIN predstavuje digitálny pin 3, čo znamená, že signálny vodič LED je pripojený k D3.

SWITCH_PIN predstavuje digitálny pin 0, čo znamená, že signálny vodič prepínača je pripojený k D0.

Výhoda tohto postupu je jednoduchá: ak sa neskôr rozhodnete presunúť LED alebo prepínač na iné piny, stačí zmeniť len tieto dva riadky. Zvyšok kódu zostane nezmenený, vďaka čomu je program prehľadnejší a oveľa ľahšie sa udržiava.

(2) Konfigurácia režimu pinov.

```
pinMode(LED_Pin, OUTPUT);                  // Nastavte pin bzučiaka ako
                                           výstup
pinMode(SWITCH_PIN, INPUT_PULLUP); //
```

Nastavte pin LED do výstupného režimu (OUTPUT), aby Arduino mohlo LED zapínať a vypínať.

Nastavte pin prepínača do režimu interného pull-up (INPUT_PULLUP), aby Arduino mohlo spoľahlivo čítať stav prepínača.

(3) Funkcia loop() beží nepretržite, opakovane číta stav prepínača a ovláda LED.

```
int switchState = digitalRead(SWITCH_PIN);
```

Účel: Načíta aktuálnu úroveň napätia na SWITCH_PIN a uloží výsledok do premennej switchState (buď HIGH (1) alebo LOW (0)).

Typ premennej: int môže obsahovať hodnoty HIGH/LOW. Začiatočníci si to môžu predstaviť ako „poznámkový blok“, ktorý zaznamenáva aktuálny stav spínača.

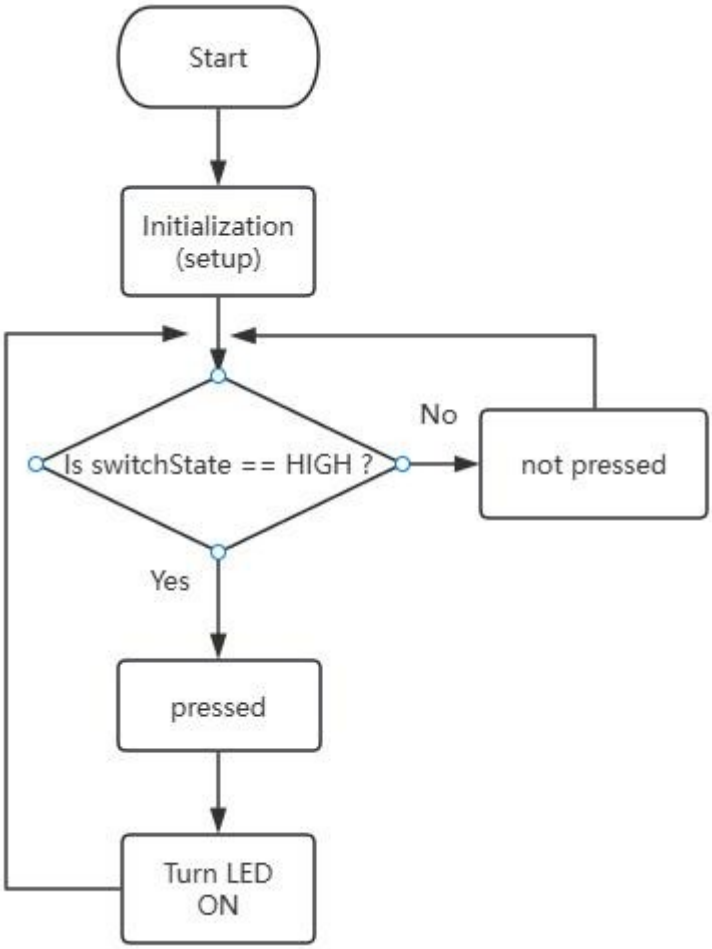
```
if (switchState == HIGH) { ... } else { ... }
```

Logika: Skontroluje, či je prepínač stlačený (v tomto nastavení HIGH znamená ON).

```
if (switchState == HIGH) {
    digitalWrite(LED_PIN, HIGH);          // Zapni LED
  } inak {
    digitalWrite(LED_PIN, LOW);           // Vypnúť LED
  }
```

Keď je stav spínača HIGH (zapnutý): digitalWrite(LED_PIN, HIGH); zapne LED. V opačnom prípade, keď je stav spínača LOW: digitalWrite(LED_PIN, LOW); vypne LED.

(4) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Download“:

lesson-04_Switch.ino | Arduino IDE 2.3.6

File Edit Sketch Tools Help

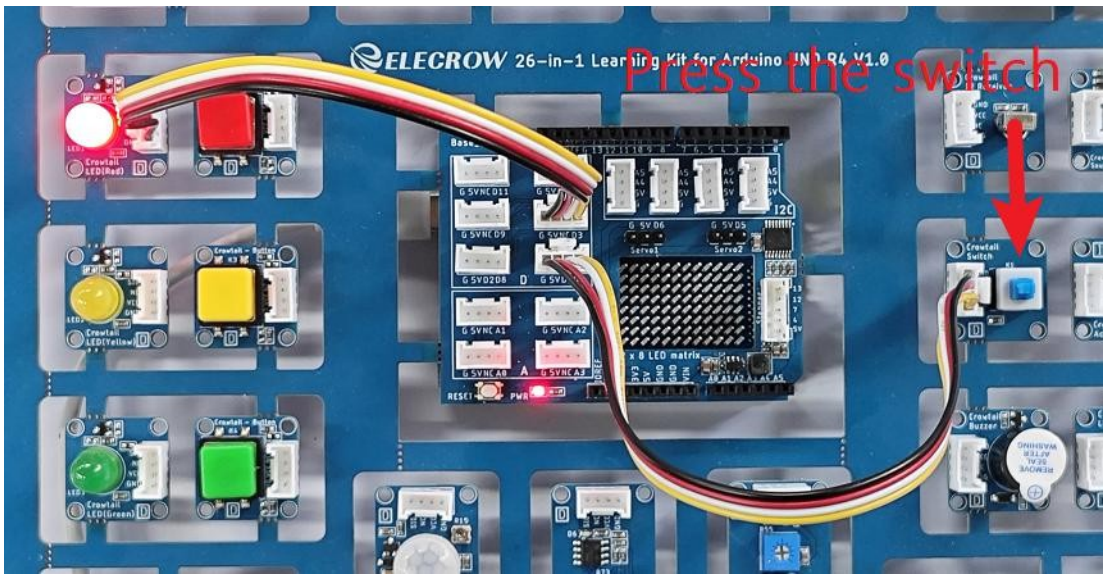
Arduino UNO R4 WiFi Upload

Upload witch.ino.ino

```
#define LED_PIN 3
#define SWITCH_PIN 0

void setup() {
  pinMode(LED_PIN, OUTPUT);
Serial.begin(9600);

  // Use internal pull-up: default state is HIGH (OFF),
  //becomes LOW when switch is ON
  pinMode(SWITCH_PIN, INPUT_PULLUP);
}

void loop() {
  int switchState = digitalRead(SWITCH_PIN);
  Serial.println(switchState);
  // When the switch is turned ON, the signal goes LOW
  if (switchState == HIGH) {
    digitalWrite(LED_PIN, HIGH);   // Turn on the LED
  } else {
    digitalWrite(LED_PIN, LOW);    // Turn off the LED
  }

}
```

(2) Uvidíte:

Po spustení programu sa stlačením prepínača rozsvieti LED.



Lekcia 05 – Senzor PIR

Úvod

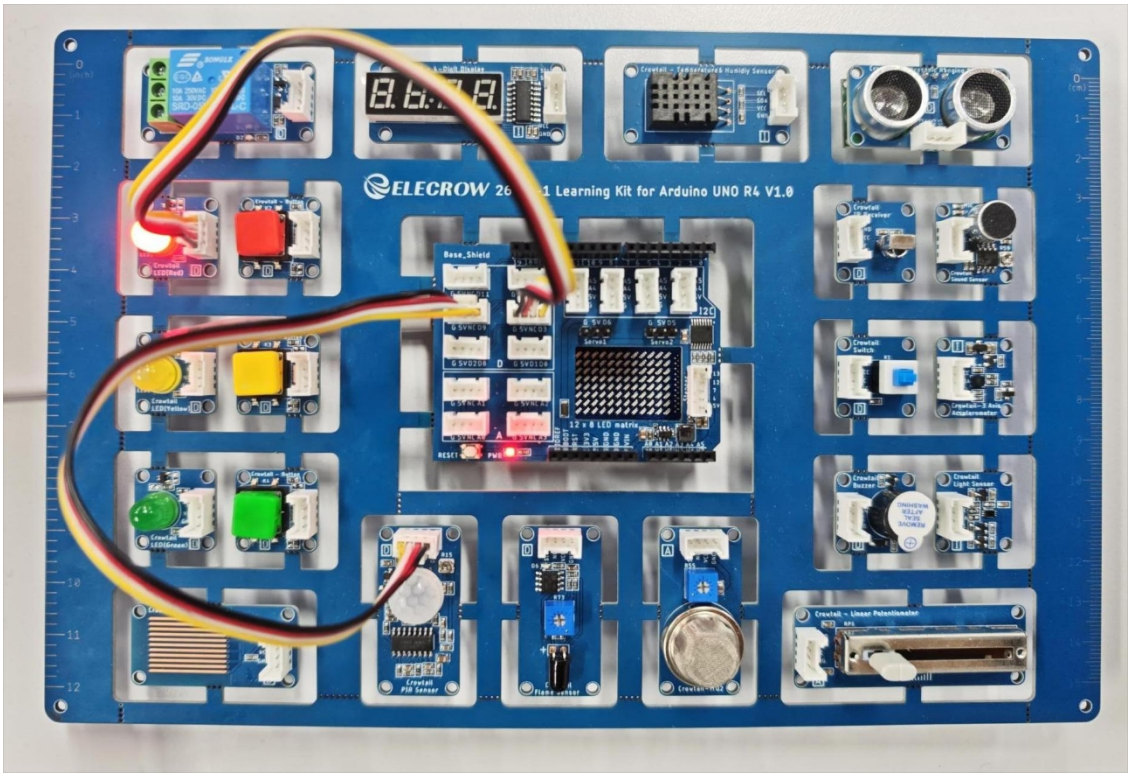
V tejto lekcii sa naučíme, ako čítať digitálny signál zo senzora PIR (pasívneho infračerveného) pomocou digitálneho pinu a ovládať LED na základe detekcie ľudského pohybu. Modul PIR vydáva vysokú úroveň (HIGH), keď zistí zmeny v infračervenom žiarení človeka, a v opačnom prípade nízku úroveň (LOW).

Ciele

1. Naučiť sa používať funkciu digitalRead() na čítanie stavu HIGH/LOW senzora z digitálneho pinu.
2. Porozumieť spúšťaciemu a podržovaciemu správaniu modulu PIR, ako aj bežným charakteristikám predohrevu (warm-up) a oneskorenia.
3. Osvojte si základné použitie a koordináciu funkcií pinMode(), digitalWrite() a sériového výstupu Serial.print().
4. Naučte sa používať signál senzora na podmienené vetvenie (if/else) s cieľom dosiahnuť ovládanie osvetlenia spúšťané človekom.

Náhľad výsledku

Keď je zistená prítomnosť človeka, sériový port vytlačí „Detected human body“ (Zistené ľudské telo) a zapne LED; keď nie je zistená prítomnosť človeka, vytlačí „No human presence was detected“ (Nebola zistená prítomnosť človeka) a vypne LED.



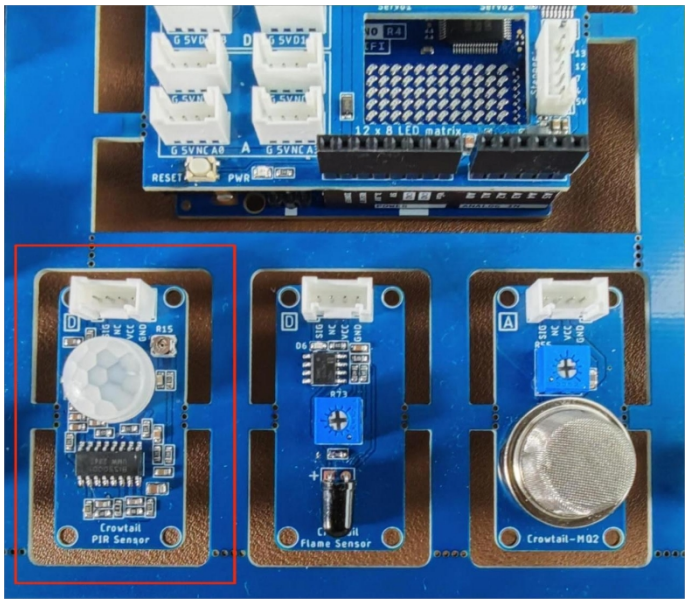
1. Vysvetlenie princípu

Senzor PIR (Passive InfraRed) sa používa na detekciu rýchlych zmien v infračervenom žiarení v prostredí (napríklad zmien v infračervenom žiarení spôsobených pohybom človeka) a jeho fungovanie je v skratke nasledovné:

Vstavaný pyroelektrický senzor zachytáva infračervené žiarenie z okolia a premieňa ho na slabý elektrický signál;

Na module sa nachádza zosilňovací obvod a filter, ktorý signál zosilňuje a formuje digitálny spínací signál;

Keď sa zistí významná zmena v infračervenom žiarení (napríklad vstup alebo pohyb osoby), modul vysiela digitálny signál s vysokou úrovňou (HIGH); ak nedochádza k žiadnej významnej zmene, vysiela signál s nízkou úrovňou (LOW).



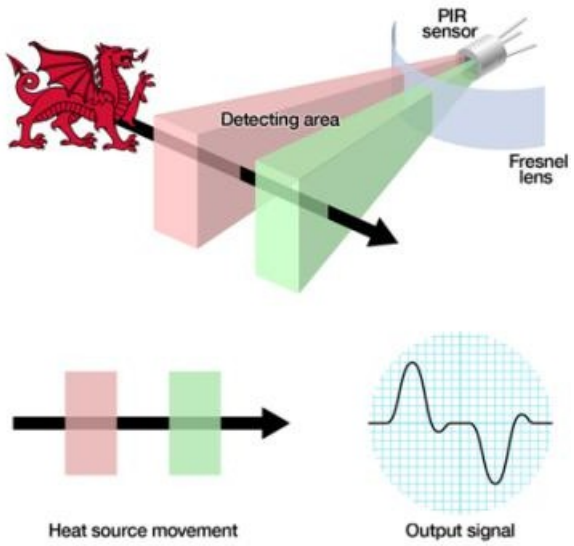
Poznámka: Väčšina PIR modulov vyžaduje zahrievanie po zapnutí a moduly sú zvyčajne vybavené nastaviteľnými potenciometrami: jeden na nastavenie citlivosti (detekcia vzdialenosti).
Otočte gombíkom doľava, aby ste zvýšili citlivosť; otočte ho doprava, aby ste citlivosť znížili.

Základný princíp:

Pohybový senzor PiR dokáže detekovať zmenu množstva infračerveného žiarenia odrážajúceho sa na ňom, čo závisí od teploty a povrchových vlastností objektu pred senzorom. Keď sa objekt, napríklad osoba, pohybuje v detekčnom rozsahu senzora PiR,

teplota v tomto bode v zornom poli senzora stúpne z izbovej teploty na telesnú teplotu a táto zmena sa vráti k senzoru a bude detekovaná.

Senzory PiR sa tiež nazývajú pohybové senzory, ktoré sa zvyčajne používajú na detekciu toho, či sa ľudia pohybujú v dosahu senzora alebo mimo neho. Senzory PiR sú malé, lacné, majú nízku spotrebu energie, sú ľahko použiteľné a neopotrebujú sa, preto sa senzory PiR zvyčajne používajú v elektrických spotrebičoch.



Konštrukcia senzora PIR a popis vývodov

Senzor PIR má zvyčajne 3 vývody:

- **VCC (napájanie):** zvyčajne 5 V
- **GND (zem):** Pripojené k spoločnému referenčnému bodu Arduina
- **SIG (výstup signálu):** Vysiela signál HIGH alebo LOW do Arduina
- Pin NC na rozhraní Crowtail sa nepoužíva

Charakteristiky signálu:

Vysiela signál HIGH (1), keď je detekovaná prítomnosť

človeka Vysiela signál LOW (0), keď nie je detekovaná

prítomnosť človeka **Digitálny vstupný pin**

V tejto lekcii používame: pinMode(PIR_PIN, INPUT);

Nastavte pin 9 do režimu vstupu, aby ste mohli zistiť digitálny stav výstupu

PIR. Digitálne vstupné piny môžu zistiť len dva stavy:

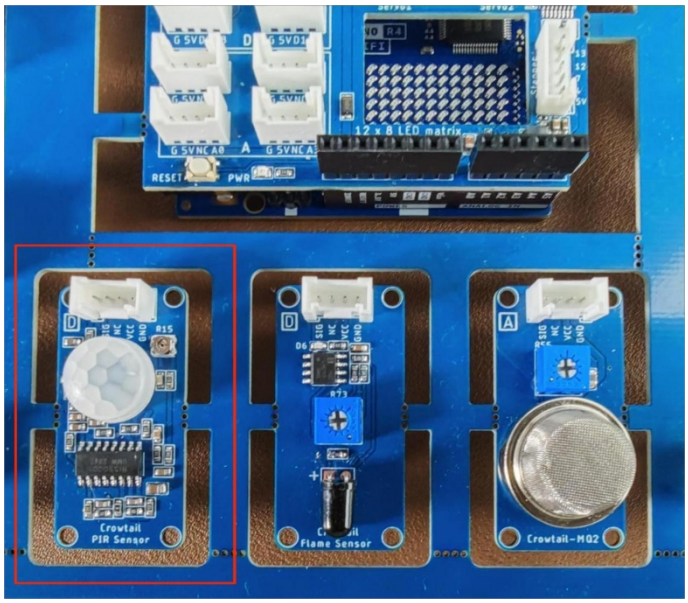
HIGH (vysoké napätie)

LOW (nízke napätie)

Vďaka tomu je veľmi vhodný na detekciu spínačov, tlačidiel, PIR senzorov, vibračných spínačov atď.

2. Potrebné moduly

Senzor Crowtail PIR (1 x infračervený senzor na detekciu ľudí) LED Crowtail (ľubovoľná farba x1)



3. Spôsob pripojenia

➢ PIR senzor → digitálny port D9 **Popis trojvodičového rozhrania Crowtail:** VCC (červený) → 5 V

GND (čierna) → GND

SIG (žltá) → D9

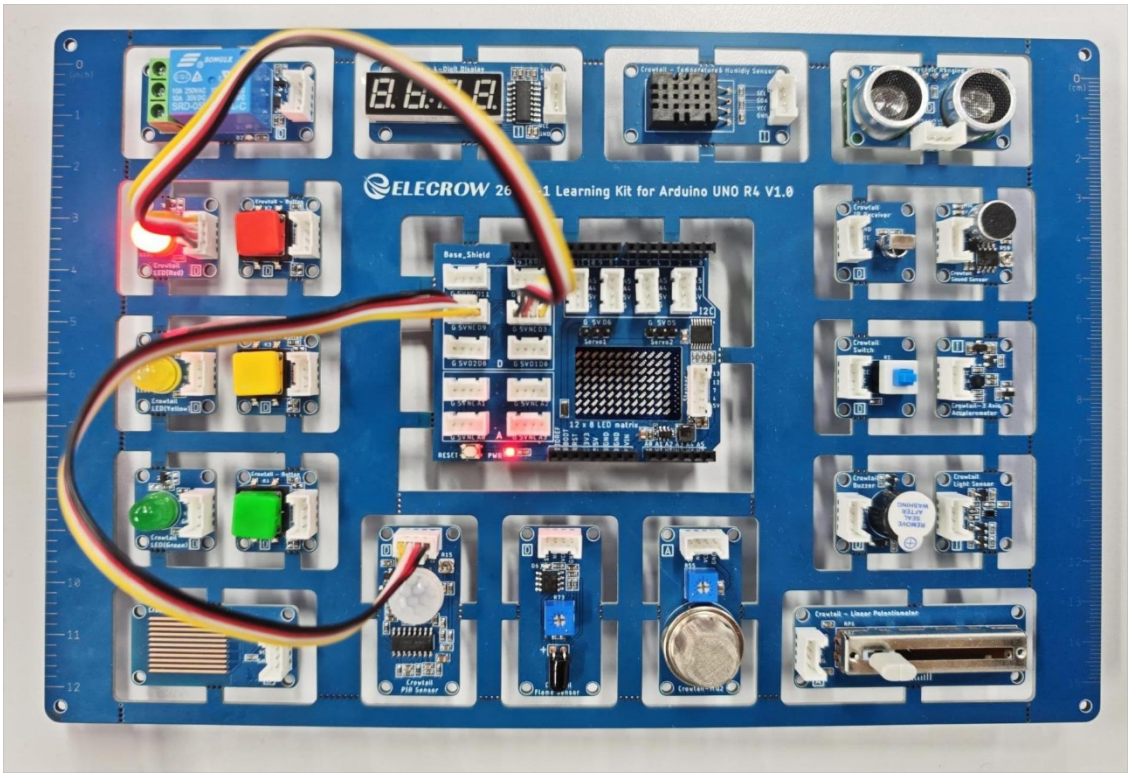
- LED Crowtail → port DIGITAL D3

Popis trojvodičového rozhrania Crowtail:

VCC (červená) → 5 V

GND (čierna) → GND

SIG (žltá) → D3



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite vzorový kód poskytnutý oficiálnym zdrojom:

Odkaz na Github:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-05_PIR_Sensor/5_PIR_Sensor

Otvorte program tejto triedy v priečinku **„5_PIR_Sensor“** pomocou Arduino IDE.

```
#define PIR_PIN    9        // PIR human body infrared sensor signal terminal
#define LED_PIN    3        // LED light (Digital Port D3)

int PIRState = 0;           // Used to store the status of the sensor

void setup() {
  Serial.begin(115200);

  pinMode(PIR_PIN, INPUT);      // The PIR is set as input.
  pinMode(LED_PIN, OUTPUT);     // The LED lights are set to output.
  digitalWrite(LED_PIN, LOW);   // Turn off the lights initially
}

void loop() {
  PIRState = digitalRead(PIR_PIN);  // Read the PIR status (HIGH or LOW)

  if (PIRState == HIGH) {
    Serial.println("Detected human body");
    digitalWrite(LED_PIN, HIGH);    // Detecting human presence → Light turns on
  }
  else {
    Serial.println("No human presence was detected.");
    digitalWrite(LED_PIN, LOW);     // No detection → Light off
  }
}
```

Vysvetlenie kľúčových kódov:

(1) #define definuje konštanty pinov

```
#define PIR_PIN        9

#define LED_PIN        3
```

- Nastavte makro **PIR_PIN** na hodnotu 9, čím určíte, že signál modulu PIR je pripojený k digitálnemu pinu D9. Použitie makier uľahčuje neskoršie úpravy (stačí zmeniť len jedno miesto).
- Definujte digitálny **pin D3** pre LED.

(2) Definujte premenné

```
int PIRState = 0;
```

Deklarujte celočíselnú premennú PIRState na uloženie aktuálneho stavu PIR (HIGH alebo LOW). Inicializujte ju na 0 (predstavuje LOW).

(3) Inicializačná sekcia setup()

```
void setup() {
   Serial.begin(115200);
   pinMode(PIR_PIN, INPUT);          // PIR je nastavený ako
                                     vstup.
   pinMode(LED_PIN, OUTPUT);           // LED diódy sú nastavené na výstup.
```

```
    digitalWrite(LED_PIN, LOW);          // Najskôr zhasnúť svetlá
}
```

- Funkcia **setup()** sa spustí raz pri zapnutí alebo resete Arduina.
- Serial.begin(**115200**); Otvorí sériový port a vypíše informácie o ladení pri prenosovej rýchlosti 115200 (sériový monitor musí byť nastavený na rovnakú prenosovú rýchlosť).
- Nastavte **PIR_PIN (D9)** do vstupného režimu. Arduino bude snímať úroveň tohto pinu. Poznámka: Väčšina PIR modulov má na výstupnom termináli (OUT) integrované pull-up/pull-down alebo obvod na spracovanie signálu, takže nie je

  potrebný žiadny dodatočný pull-up. Ak však modul nemá túto funkciu zabudovanú, môžete použiť

  INPUT_PULLUP a zmeniť smer (v závislosti od modulu). Zvyčajne stačí INPUT.
- Nastavte pin LED na výstup, aby ste riadili LED (LED svieti, keď je priložená vysoká úroveň,

  a zhasne, keď je priložená nízka úroveň).
- Najskôr vypnite LED, aby sa pri zapnutí náhodou nerozsvietila.

(4) Detekcia slučky

Prečítajte digitálny pin:

```
void loop() {
    PIRState = digitalRead(PIR_PIN);          // Prečítajte stav PIR (HIGH alebo LOW)
```

Funkcia **digitalRead()** vráti len dve hodnoty:

- HIGH → Označuje prítomnosť ľudského tela
- LOW → Označuje neprítomnosť ľudského tela

Hlavná slučka zakaždým zčíta digitálnu hodnotu z D9, vráti hodnotu HIGH (1) alebo LOW (0) a priradí ju premennej PIRState. To je priamy základ pre určenie, či modul zaznamenal prítomnosť ľudského tela.

(5) Určiť, či sa objavilo ľudské telo

```
    if (PIRState == HIGH) {
        Serial.println("Zistené ľudské telo");
        digitalWrite(LED_PIN, HIGH);          // Zistenie prítomnosti človeka → Rozsvieti sa
                                              svetlo
    }
    else {
        Serial.println("Nebola zistená prítomnosť človeka.");
        digitalWrite(LED_PIN, LOW);           // Žiadna detekcia → Svetlo
                                              zhasne
    }
```

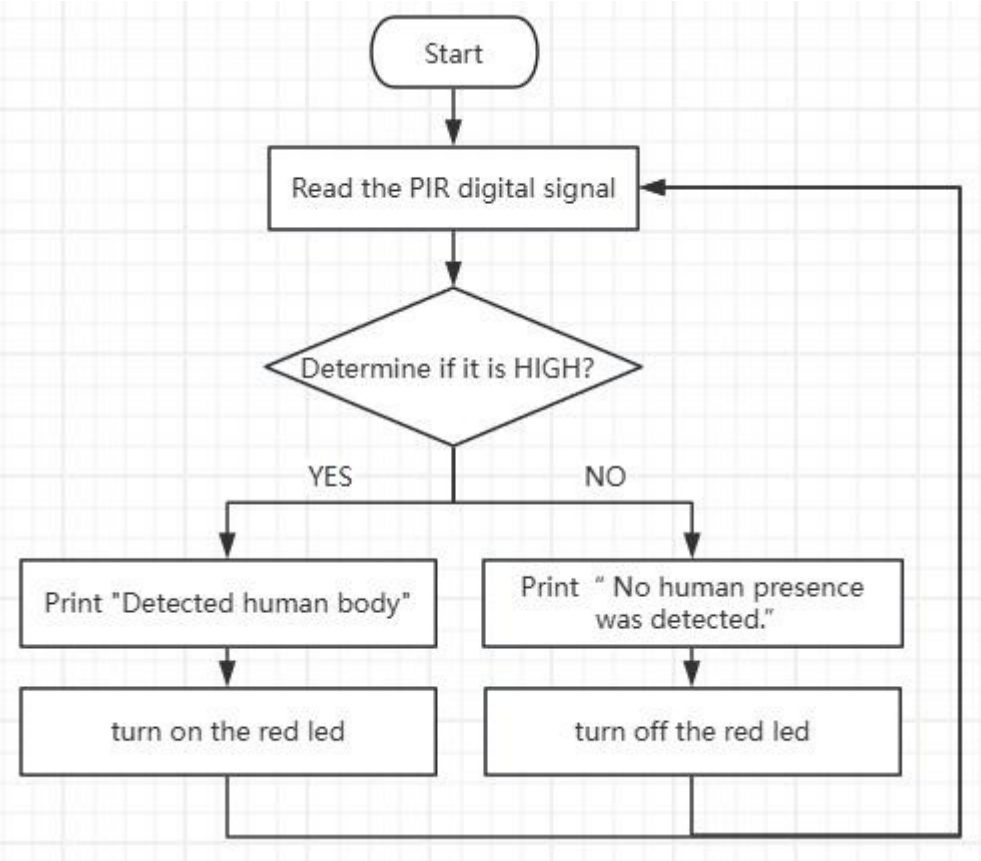
```
}
```

Podmienené rozhodnutie: Ak je detekovaná hodnota HIGH, vykoná sa vetva **„Zistená prítomnosť človeka“**; inak sa vykoná vetva **„Žiadna prítomnosť človeka“**.

Vo vetve **„Zistená prítomnosť človeka“**: Serial.println („Zistené ľudské telo“); vypíše správu a digitalWrite(LED_PIN, HIGH); zapne LED.

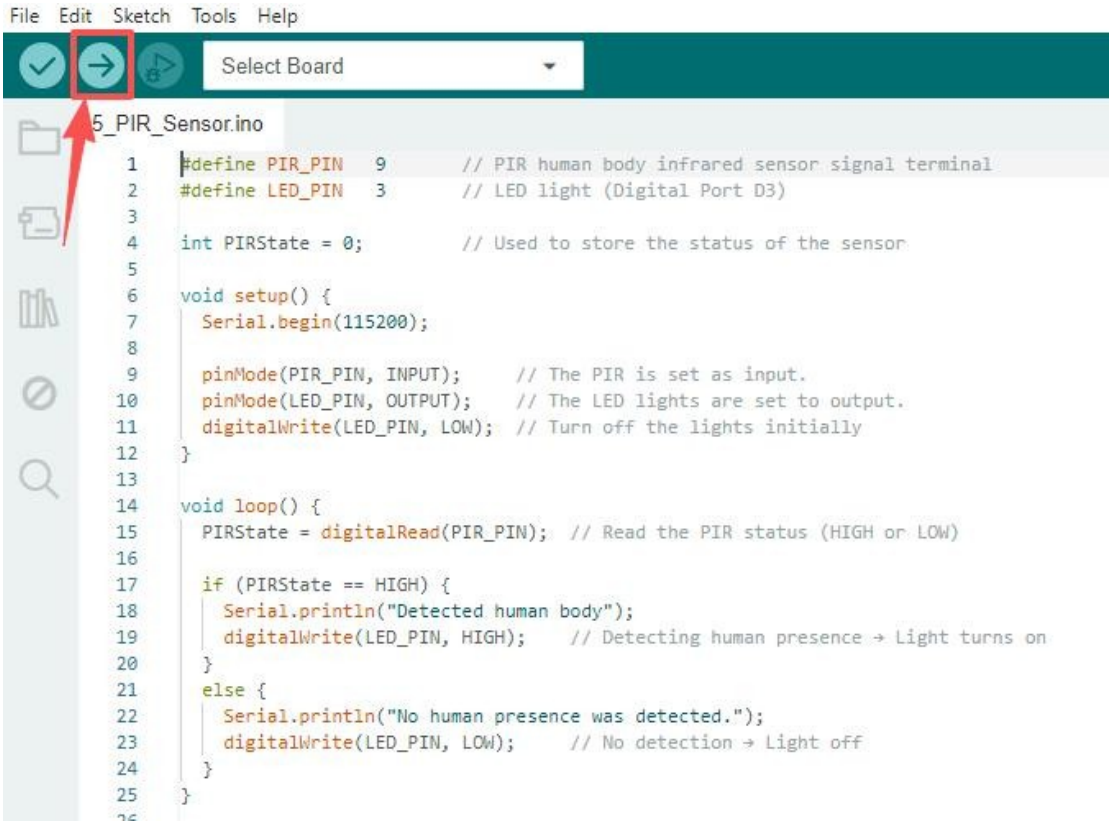
V vetve **„No Human“**: vypíše hlásenie „Nebola zistená prítomnosť človeka.“ a zhasne LED diódu.

(6) Logický tok kódu



5. Spustite program a sledujte výsledky

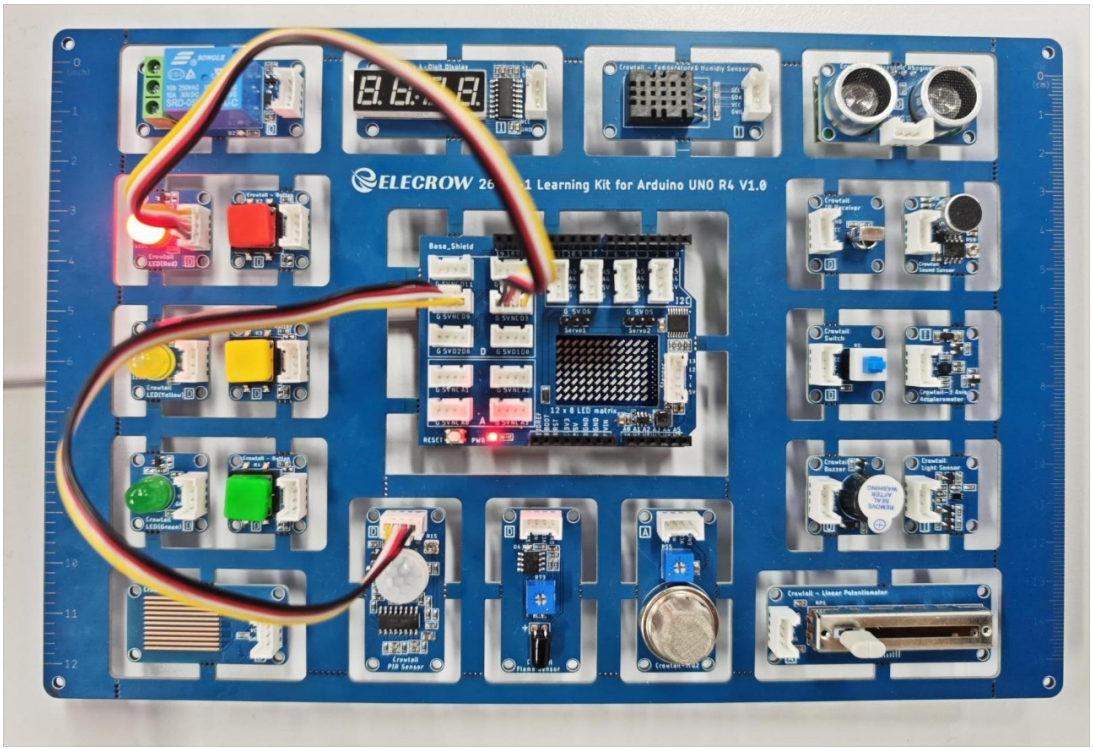
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.



(2) Po úspešnom stiahnutí skontrolujte výsledok:

Otvorte sériový monitor zabudovaný v prostredí Arduino IDE a nastavte prenosovú rýchlosť podľa špecifikácie v kóde. Týmto spôsobom budete môcť vidieť informácie vypísané cez sériový port.

(Ak sa teraz priblížite k senzoru PIR, môžete sledovať zmeny v údajoch zobrazených na sériovom monitore a vidieť, ako sa LED rozsvieti alebo zhasne.)


ELECROW
Learning Kit for Arduino UNO R4 V1.0
Base_Shield
12 x 8 LED matrix

Lekcia 06 – Senzor vlhkosti pôdy

Úvod

V tejto lekcii sa zoznámime s bežne používaným analógovým vstupným zariadením – senzorom vlhkosti pôdy.

Senzor vlhkosti pôdy detekuje zmeny vlhkosti pôdy a vysiela zodpovedajúci napäťový signál. Arduino môže toto napätie odčítať pomocou funkcie analogRead() a na základe výslednej hodnoty určiť, aká je vlhkosť pôdy.

Na rozdiel od spínača poskytuje senzor vlhkosti pôdy analógový vstup, čo znamená, že jeho hodnoty sa menia plynule, namiesto toho, aby boli obmedzené na jednoduché stavy HIGH alebo LOW.

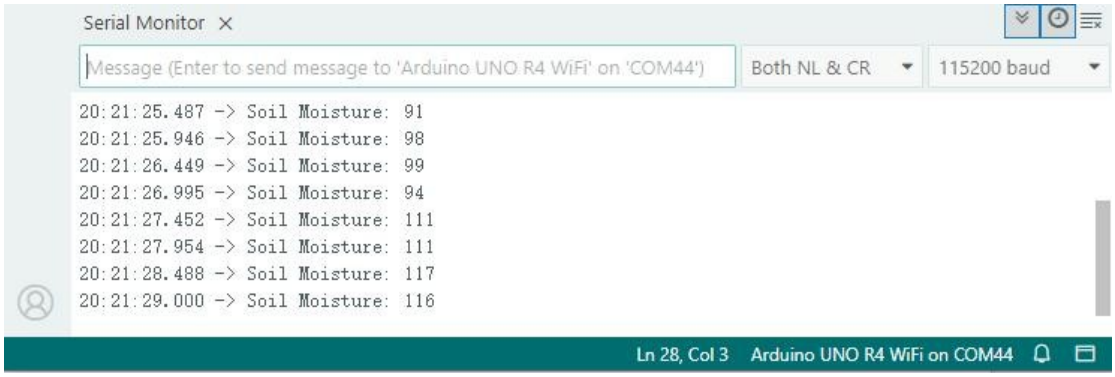
Ciele vzdelávania

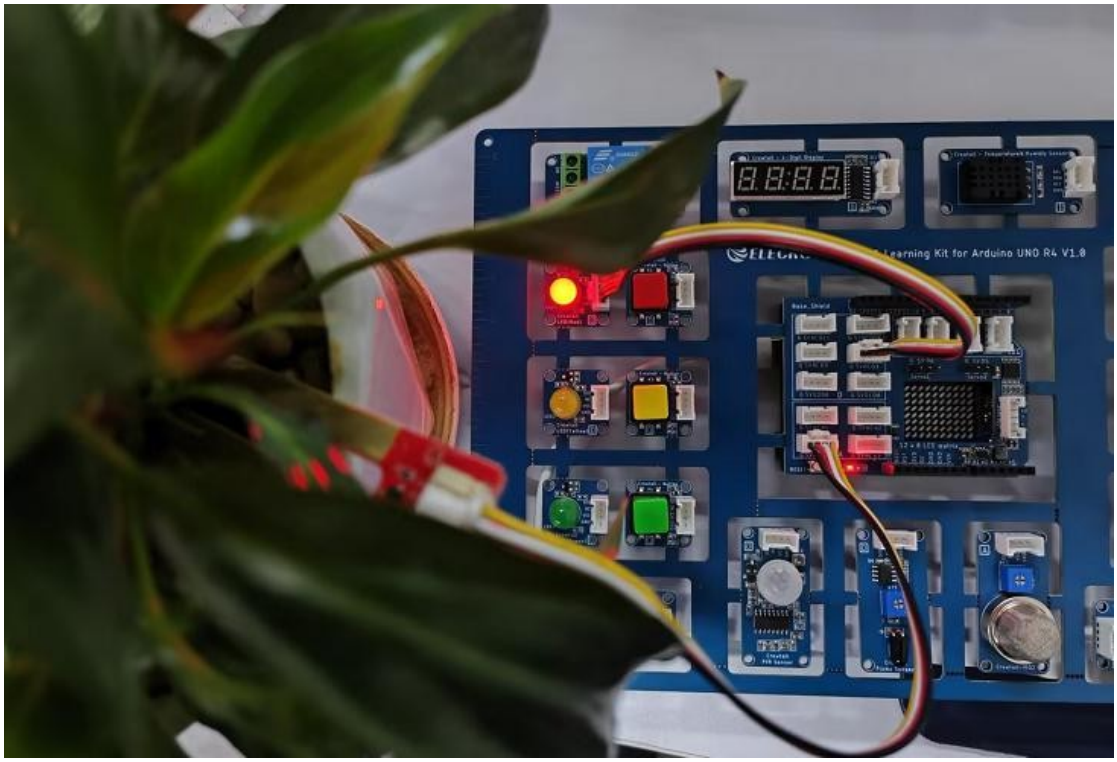
1. Porozumieť štruktúre a princípu fungovania senzora vlhkosti pôdy
2. Naučiť sa používať funkciu analogRead() na čítanie údajov zo senzora
3. Zobraziť hodnoty vlhkosti v sériovom monitore 4. Pochopiť rozdiel medzi analógovými a digitálnymi signálmi

Náhľad výsledku

Vložte senzor do pôdy.

Arduino prečíta hodnotu vlhkosti a vypíše ju do sériového monitora. Keď je hodnota vlhkosti nižšia ako 200, LED dióda sa rozsvieti; keď je hodnota 200 alebo vyššia, LED dióda zhasne.

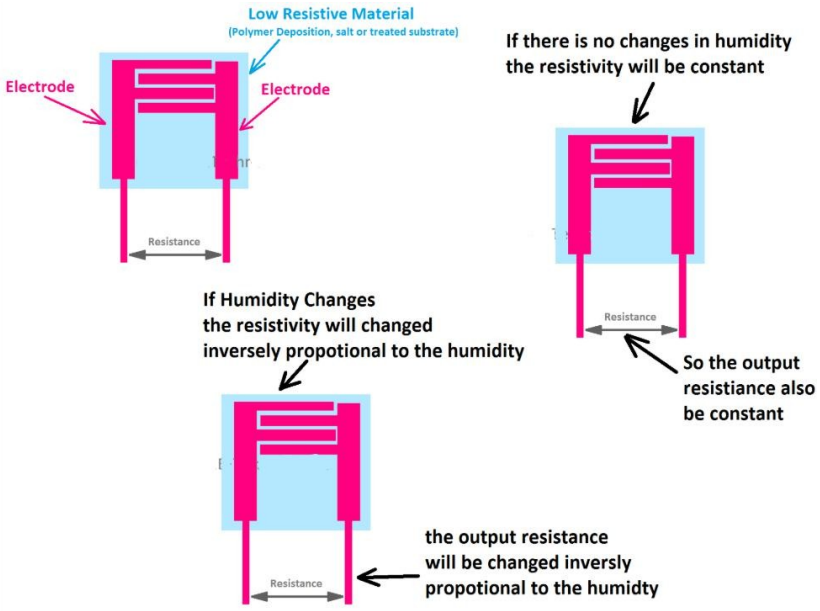




1. Vysvetlenie princípu

①Odporový senzor vlhkosti pôdy

Odporový senzor vlhkosti pôdy funguje na princípe zmeny svojho odporu v závislosti od zmeny úrovne vlhkosti. Je vyrobený z materiálov s veľmi nízkym odporom, ako sú pevné polymérne elektrolyty, nanesené polyméry, soli alebo upravené substráty. Dve elektródy sú usporiadané tak, aby medzi nimi vznikol maximálny kontakt.



Ako vidíte, obe elektródy sú usporiadané tak, aby mali veľkú kontaktnú plochu. Keď úroveň vlhkosti zostáva konštantná, výstupný odpor sa nemení. Keď sa však úroveň vlhkosti mení, odpor sa mení zodpovedajúcim spôsobom. Vzhľadom na to, že sa používajú materiály s nízkym odporom, je odpor medzi výstupnými svorkami zvyčajne veľmi nízky. Elektródy sú zvyčajne vyrobené zo zlata, striebra alebo platiny.
Merný odpor sa mení nepriamo úmerne s vlhkosťou. To znamená, že s rastúcou vlhkosťou klesá merný odpor a s klesajúcou vlhkosťou merný odpor stúpa. Meraním výstupného odporu môžeme určiť úroveň vlhkosti. Senzor sa dá použiť aj v obvodoch na meranie prúdu alebo poklesu napätia a zapojiť do série. Meraním napätia alebo prúdu sa dá vlhkosť merať nepriamo.
Hlavnými výhodami odporových senzorov vlhkosti pôdy sú ich nízka cena, jednoduchá konštrukcia, malé rozmery, možnosť umiestnenia ďaleko od hlavného obvodu a jednoduchá inštalácia.

Typické hodnoty snímača za rôznych podmienok:

Položka	Min	Typ hodnota	Max	Jednotka
Napätie	3,3	/	5	V
Prúd	0	~	35	mA
Senzor v suchej pôde	0	~	300	/
Senzor vo vlhkej pôde	300	~	500	/
Senzor vo vode	500	~	750	/

Analógové piny

Analógové piny sú vyhradené vstupné piny na vývojových doskách, ako sú Arduino a Pico, ktoré slúžia na čítanie analógových signálov. Dokážu čítať hodnoty nepretržitého napätia v rozsahu od 0 až po maximálne referenčné napätie.
Napríklad:
Arduino UNO: 0 ~ 5 V
Tieto úrovne spojitého napätia sa prevádzajú na digitálne hodnoty prostredníctvom ADC (analogovo-digitálny prevodník). Na Arduine prevádza ADC vstupné napätie na hodnotu v rozmedzí 0 až 1023.
Analógové piny sa bežne používajú na čítanie senzorov s analógovým výstupom, ako sú senzory vlhkosti pôdy, fotorezistory (LDR), potenciometre (gombíky), senzory plynu (séria MQ) a teplotné senzory (NTC termistory). Všetky tieto senzory vydávajú nepretržite sa meniace napätia.
Stručne povedané, analógové piny sú navrhnuté na snímanie nepretržitých zmien napätia a ich prevod na

digitálne hodnoty, vďaka ktorým zistíte napríklad „aká je intenzita osvetlenia“, „aká je vlhkosť pôdy“ alebo „o koľko je otočený gombík“.

2. Potrebné moduly

Crowtail – Senzor vlhkosti 2,0 × 1

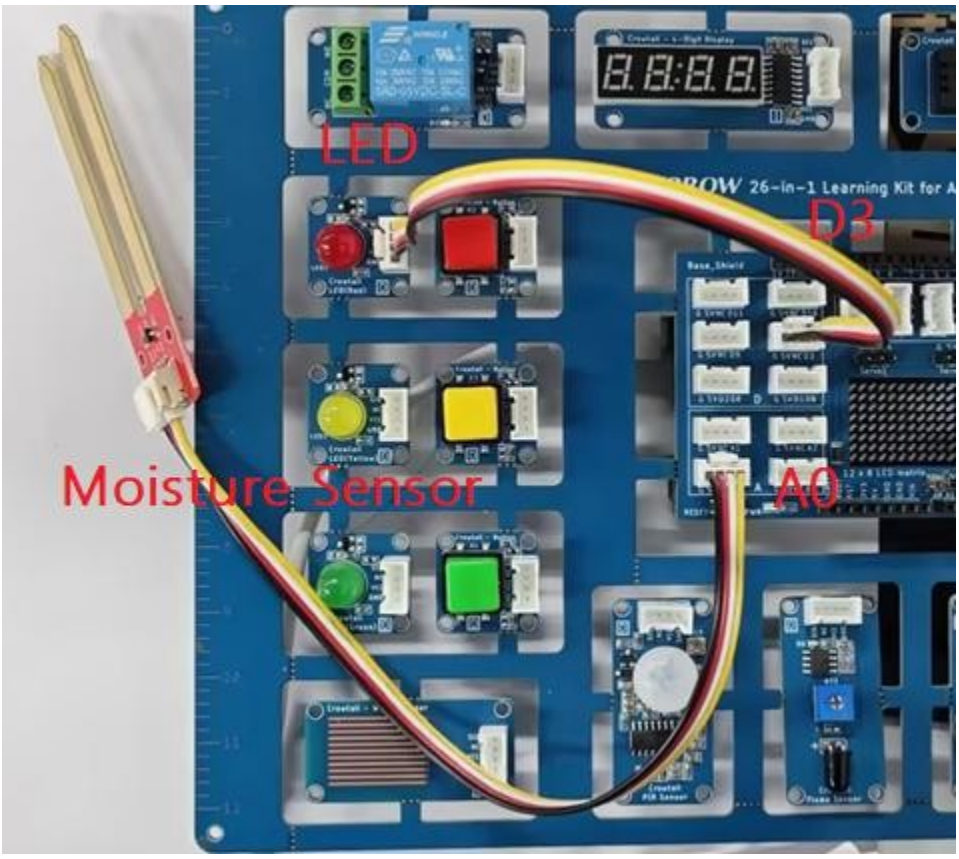
3. Spôsob zapojenia

Crowtail -- Senzor vlhkosti 2.0 → analógový port A0

Crowtail -- LED → digitálny port D3

Špecifikácia štvorportového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltý) → D3/A0



4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-06_Moisture_Sensor.ino

Spustite program pomocou prostredia Arduino IDE.

Vysvetlenie kódu

(1) Najskôr definujte digitálne piny, aby bol kód ľahšie čitateľný.

```
// Definujte pin LED pripojený k digitálnemu pinu 3
#define LED_PIN 3
// Definujte analógový vstupný pin senzora vlhkosti pôdy
#define MOISTURE_PIN A0 // Analógový vstupný pin pre senzor vlhkosti
```

#define LED_PIN 3 a #define MOISTURE_PIN A0 sa používajú na pridelenie výstižných „mien“ pinom. LED_PIN predstavuje digitálny pin 3, čo znamená, že vodič signálu LED je pripojený k D3. MOISTURE_PIN predstavuje analógový pin 0, čo znamená, že signálny vodič senzora je pripojený k A0. Výhodou písania kódu týmto spôsobom je, že ak neskôr budete chcieť presunúť LED alebo senzor na iný pin, stačí zmeniť len tieto dva riadky. Zvyšok kódu zostane rovnaký, vďaka čomu je kód prehľadnejší a ľahšie sa udržiava.

(2) Konfigurácia režimu pinov.

```
pinMode(LED_PIN, OUTPUT);
```

Pin LED je nastavený do režimu OUTPUT, aby bolo možné ho zapínať a vypínať. Pre analógové vstupné piny nie je potrebné nastavovať pinMode().

(3) Sériový monitor.

Sériový monitor sa používa na tlač textu z Arduina do vášho počítača. Čo je sériový monitor?

Je to nástroj v prostredí Arduino IDE, ktorý zobrazuje údaje zo senzorov a ladiace správy, čím uľahčuje kontrolu stavu programu počas behu. Môžete si ho predstaviť ako „okno chatu“ medzi Arduinom a vaším počítačom.

```
Serial.begin(115200);
Serial.println("Spustenie ukážky senzora vlhkosti pôdy");
```

Tento kód spustí sériovú komunikáciu medzi Arduinom a počítačom. 115200 je prenosová rýchlosť, čo znamená, že za sekundu sa prenesie 115 200 bitov. Medzi bežné prenosové rýchlosti patria 9600, 57600 a 115200.

115200 je veľmi rýchla a poskytuje rýchlu spätnú väzbu pri tlačení údajov, preto sa bežne používa na ladenie.

Tento riadok musí byť umiestnený v setup(), pretože sériová komunikácia musí byť inicializovaná pri spustení programu; inak nemôžu byť vytlačené žiadne informácie o ladení.

Serial.println("Soil Moisture Sensor Demo Start"); vytlačí spúšťaciu správu do počítača, čo pomáha pri ladení.

(4) Čítanie analógovej hodnoty

```
int moistureValue = analogRead(MOISTURE_PIN);
```

Funkcia: načíta aktuálnu analógovú hodnotu z MOISTURE_PIN a uloží ju do moistureValue. Funkcia analogRead() vráti celé číslo v rozmedzí od 0 do 1023. Vyššia hodnota znamená vlhkejšiu pôdu, zatiaľ čo nižšia

hodnota znamená suchšiu pôdu.

analogRead() je funkcia Arduina používaná na čítanie napätia na analógovom pine. Načíta napätie z analógového vstupného pinu (A0, A1, A2 atď.) a prevádza ho na digitálnu hodnotu.

```
// Vypíše hodnotu vlhkosti na sériový monitor
Serial.print("Vlhkosť pôdy: ");
Serial.println(moistureValue);
```

Týmto sa na sériový monitor vypíše hodnota vlhkosti pôdy.

Rozdiel medzi print a println:

Serial.print() vytlačí dáta bez presunu na nový riadok.

Serial.println() vytlačí údaje a potom automaticky prejde na ďalší riadok.

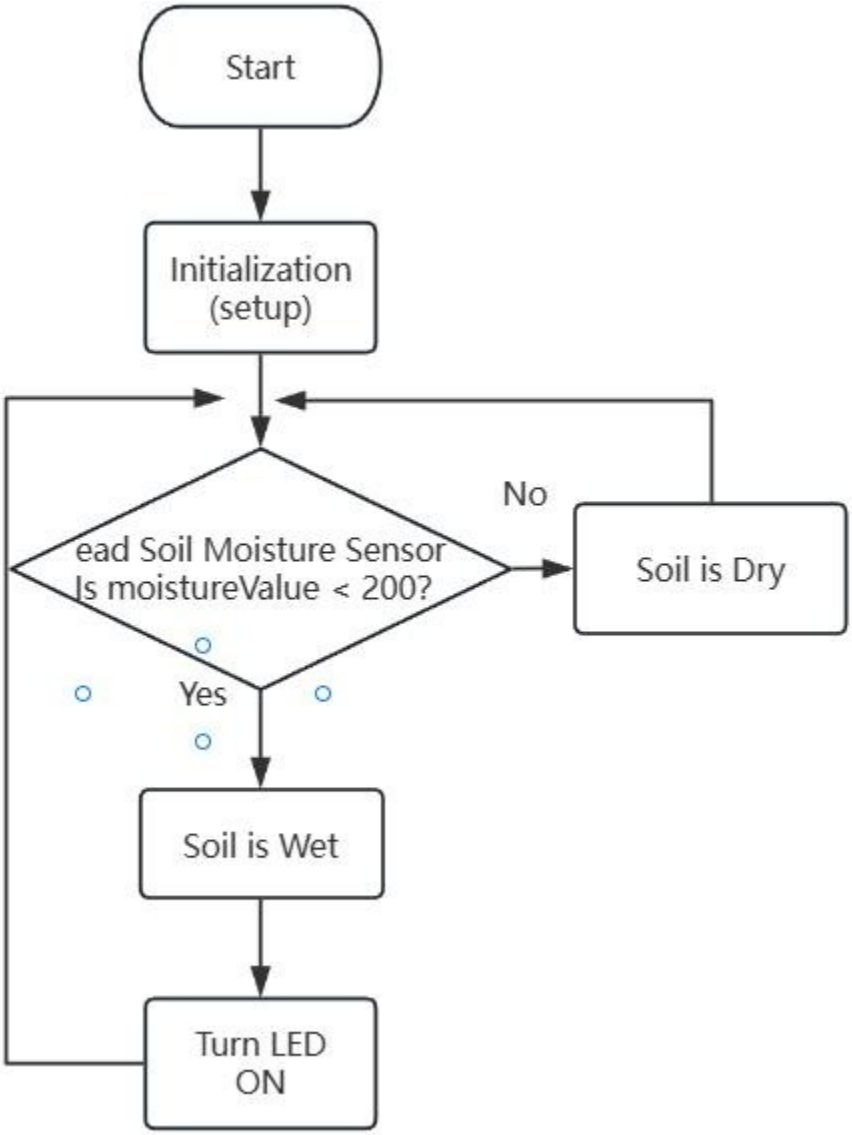
(5) Ovládanie LED

Logika: zapnúť LED ako varovanie, keď je pôda suchá, a vypnúť ju, keď je pôda vlhká.

```
if (moistureValue < 200) {
    // Keď je hodnota < 200 → pôda je suchá → LED
    zapnutá digitalWrite(LED_PIN, HIGH);
} else {
    // Ak je hodnota ≥ 200 → pôda je vlhká → LED
    vypnutá digitalWrite(LED_PIN, LOW);
}
```

Keď je hodnota vlhkosti menšia ako 200, digitalWrite(LED_PIN, HIGH) zapne LED. Keď je hodnota vlhkosti 200 alebo vyššia, digitalWrite(LED_PIN, LOW) vypne LED.

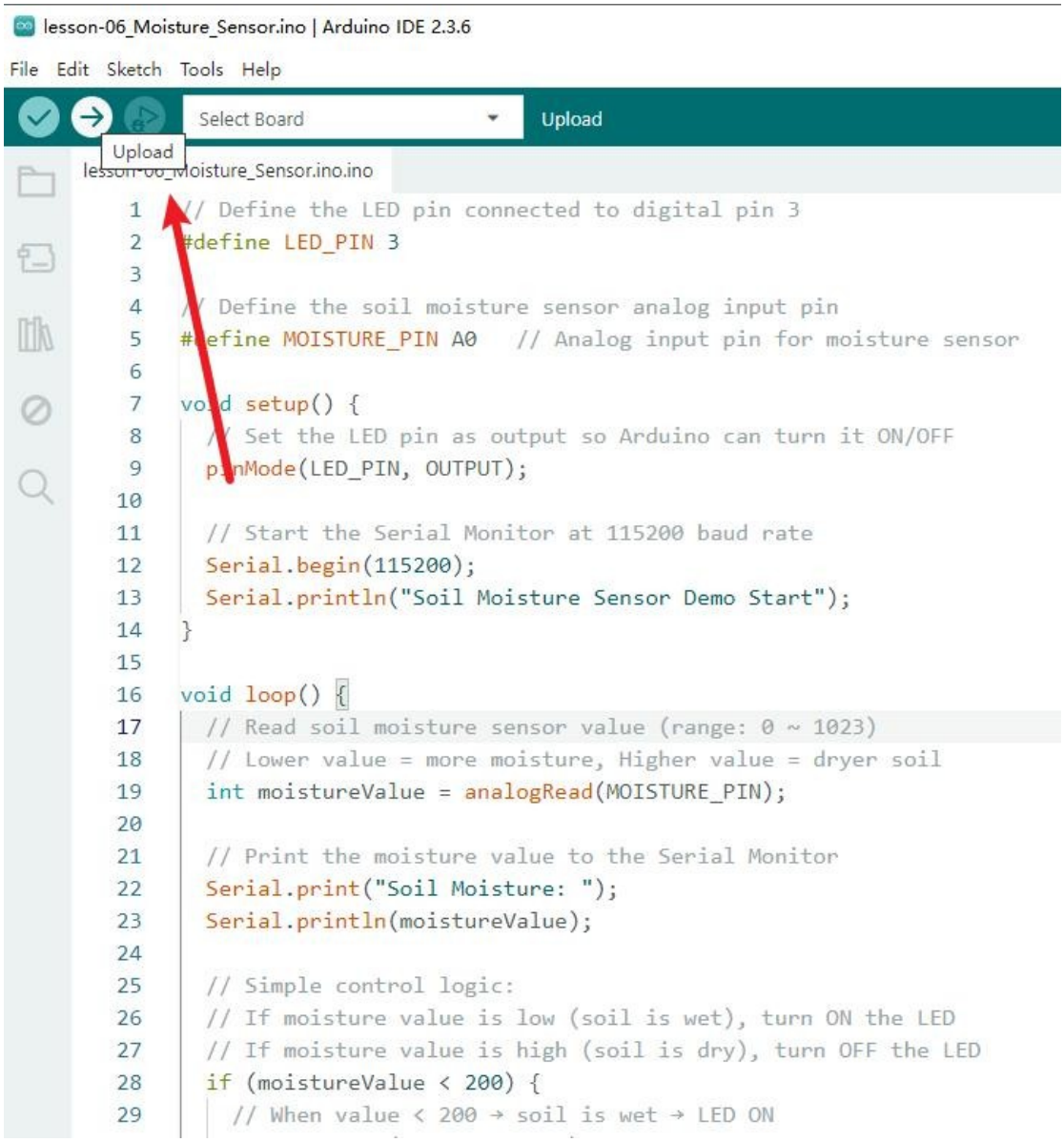
(6) Celkový diagram logického toku kódu



5. Spustite program a sledujte výsledky

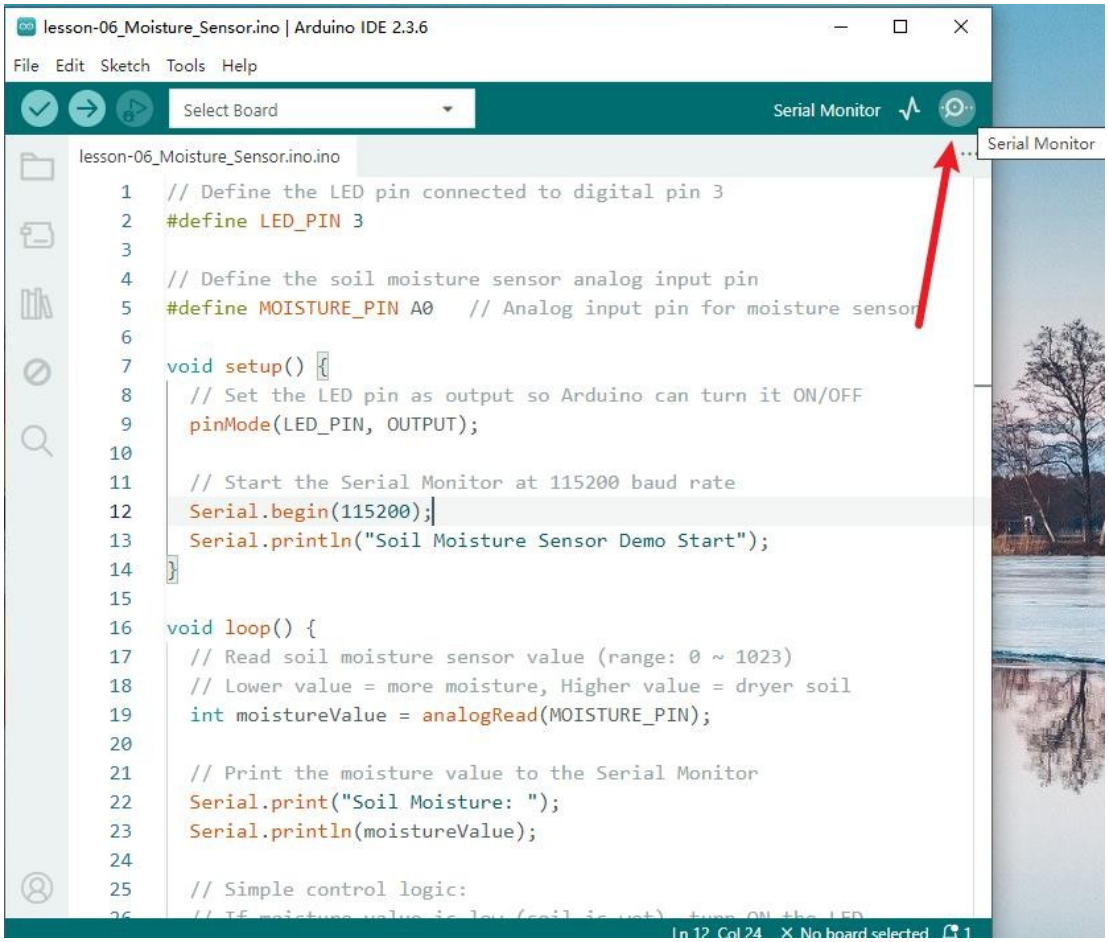
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Stiahnuť“:

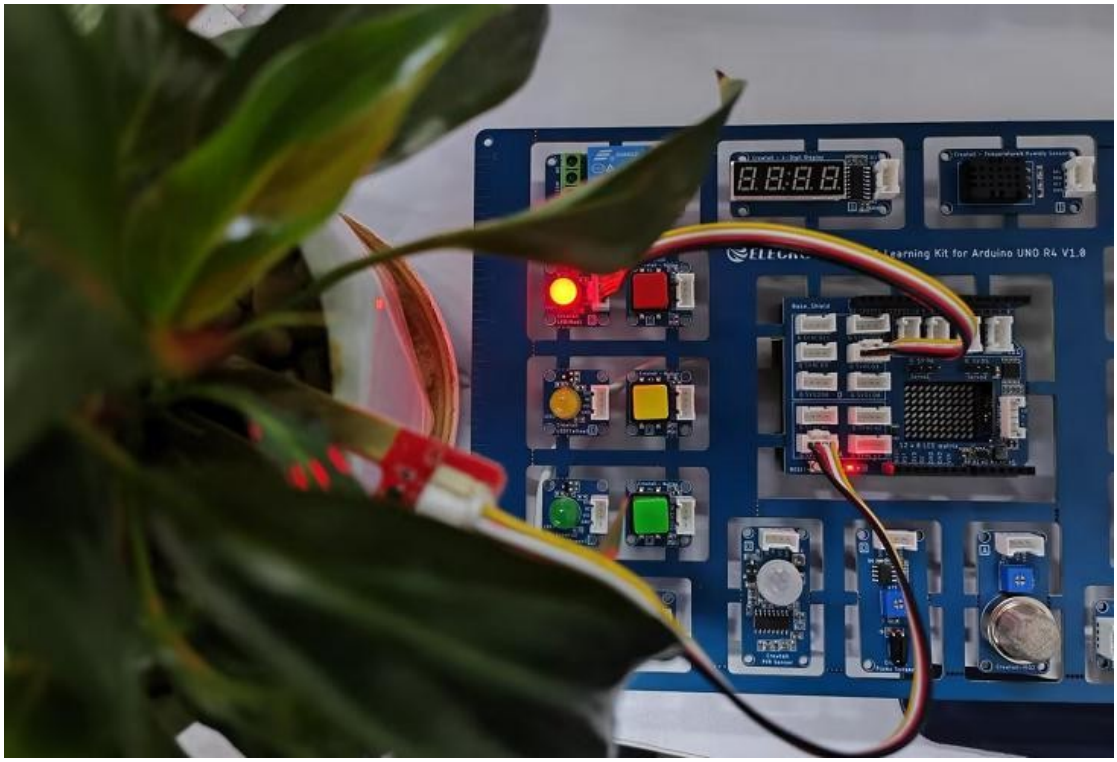


(2) Program sa spustí.

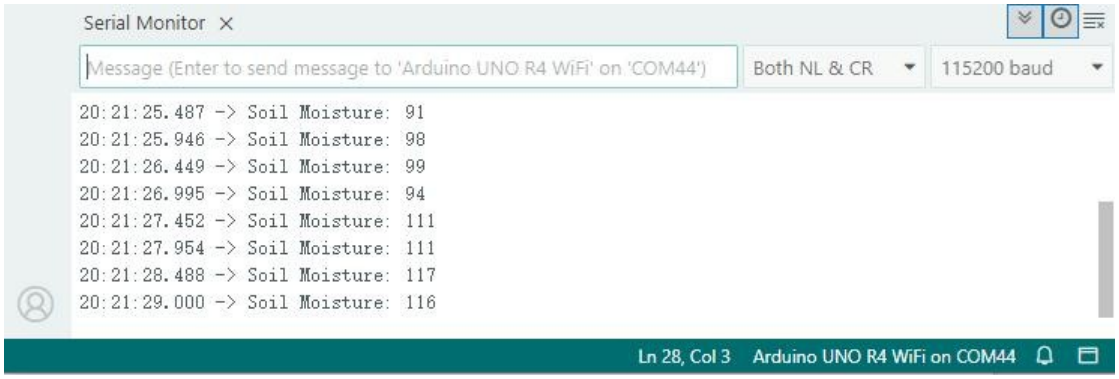
Kliknite na ikonu v pravom hornom rohu, aby ste otvorili sériový monitor.



Pred polievaním: pôda je suchá, hodnota vlhkosti je nižšia ako 200 a LED dióda svieti.

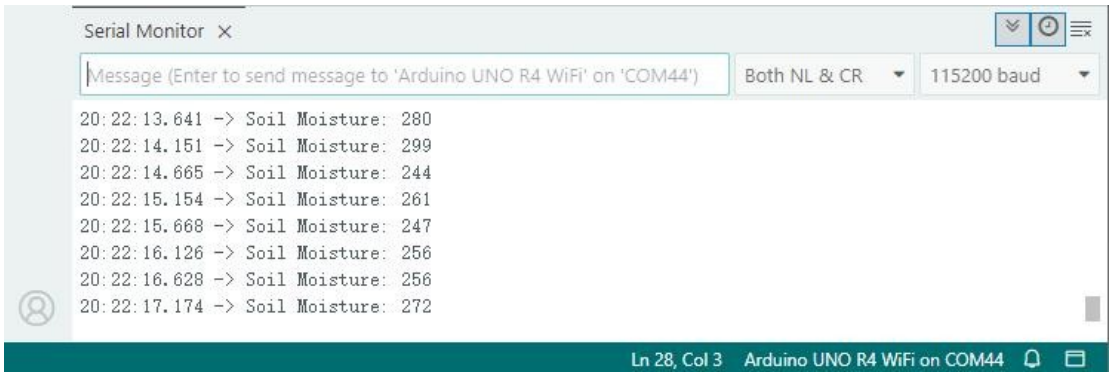


Hodnotu vlhkosti pôdy môžete vidieť v okne sériového monitora.



Po zalievaní: pôda je vlhká, hodnota vlhkosti je nad 200 a LED dióda zhasne.

Hodnotu vlhkosti pôdy je možné vidieť v okne sériového monitora.



Lekcia 07 — Lineárny potenciometer

Úvod

V tejto lekcii sa naučíme, ako používať lineárny potenciometer ako analógové vstupné zariadenie a ovládať jas LED diódy prostredníctvom kódu.

V tejto lekcii sa zameriate na dve veľmi dôležité funkcie:

analogRead() — slúži na čítanie hodnôt analógového vstupu

analogWrite() — slúži na ovládanie jasu výstupu pomocou PWM

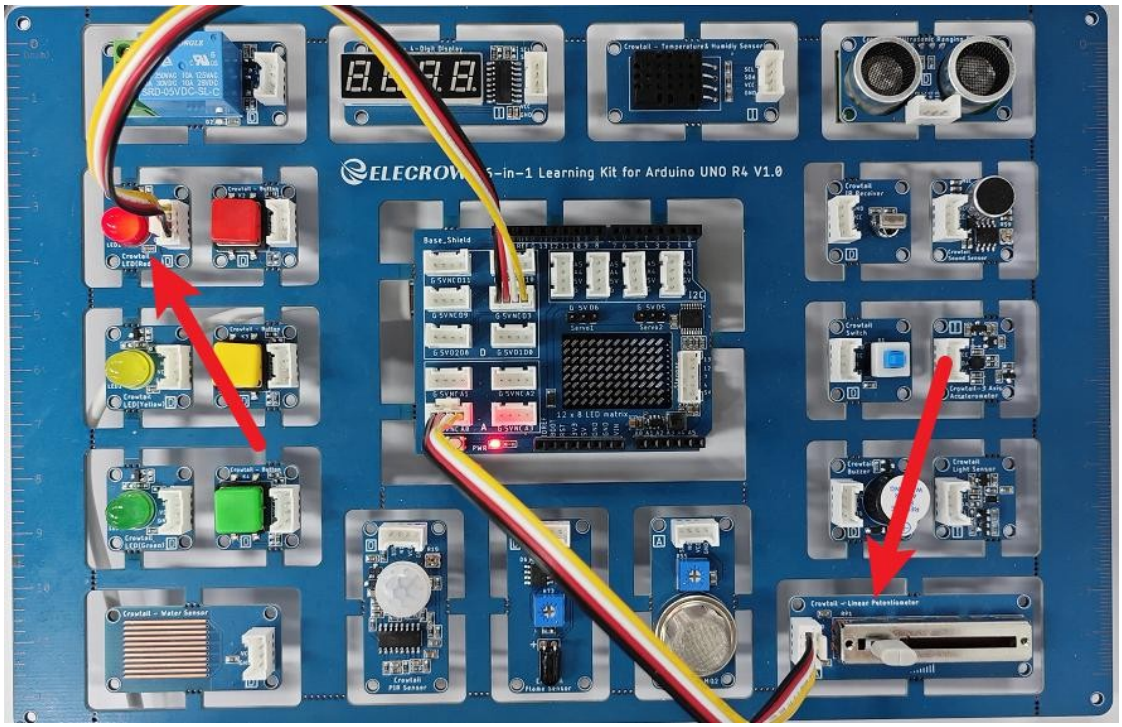
Táto lekcia vám pomôže vytvoriť kompletný pracovný postup „analógový vstup → spracovanie hodnoty → analógový výstup“. Ide o základný koncept pre prácu s rôznymi senzormi a ovládačov.

Ciele výučby

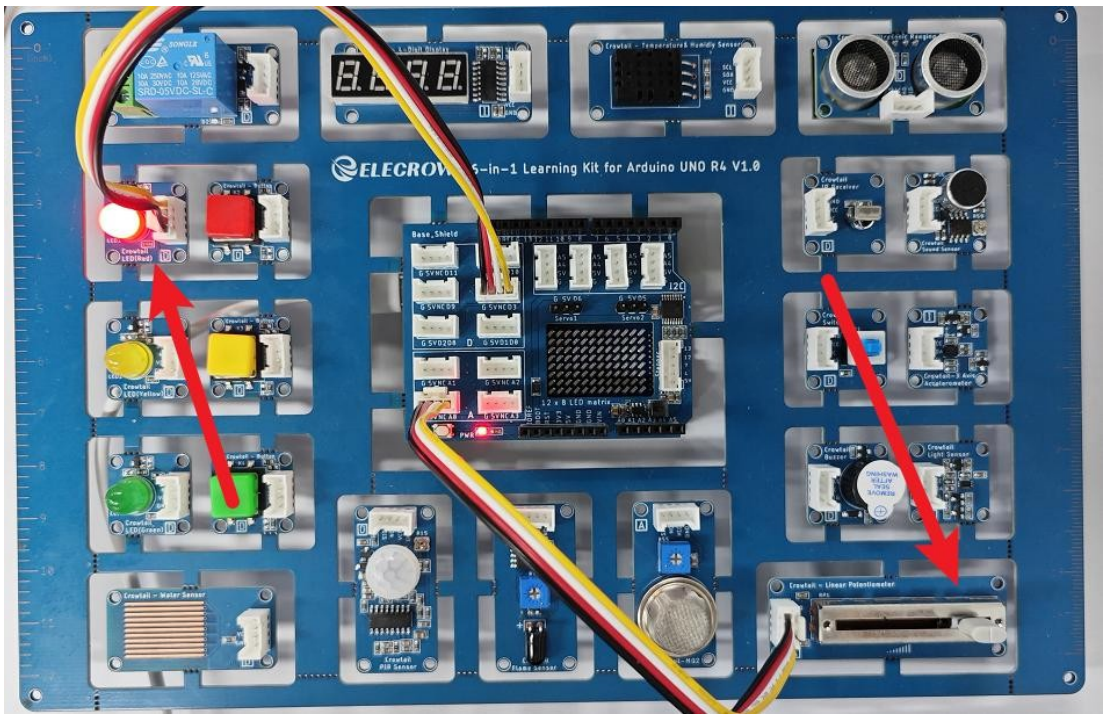
1. Porozumieť štruktúre a princípu fungovania potenciometra
2. Naučte sa používať funkciu map() na mapovanie hodnôt
3. Naučte sa používať PWM (analogWrite()) na ovládanie jasu LED

Náhľad výsledku

Posuňte potenciometer → Arduino prečíta hodnotu → Jas LED sa zodpovedajúcim spôsobom zmení
Posuňte doľava: LED sa stlmí



Posuňte doprava: LED svieti jasnejšie.



1. Vysvetlenie princípu

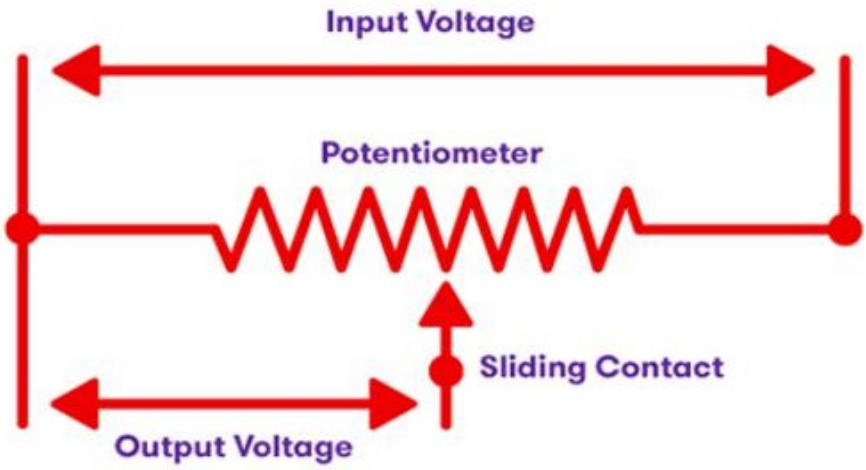
①Princíp fungovania potenciometra

Lineárny potenciometer si môžeme predstaviť ako nastaviteľný „posuvný rezistor“.

Vo vnútri sa nachádza odporová dráha a posuvný kontakt (stierač), ktorý sa pohybuje po dráhe a mení

výstupné napätie.

Stierač vľavo → bližšie k GND → výstupné napätie je nízke Stierač vpravo → bližšie k 5 V → výstupné napätie je vysoké



Vyššie uvedený diagram ilustruje úlohu potenciometra v obvode.

Potenciometer funguje ako nastaviteľný delič napätia: posúvaním kontaktného jazýčka po odporovom materiáli sa mení výstupné napätie. Vstupné napätie je pripojené na celú dĺžku rezistora. Výstupné napätie predstavuje pokles napätia medzi pevným vývodom a kontaktným jazýčkom.

Analógové piny Arduina (napr. A0) prevádzajú toto napätie na digitálnu hodnotu v rozmedzí od 0 do 1023.

Poloha kurzora	Napätie (0 — 5 V)	Hodnota analogRead (0 – 1023)
Najviac vľavo	0 V	0
Stred	≈2,5 V	≈512
Najviac vpravo	5 V	1023

②PWM (modulácia šírky impulzu)

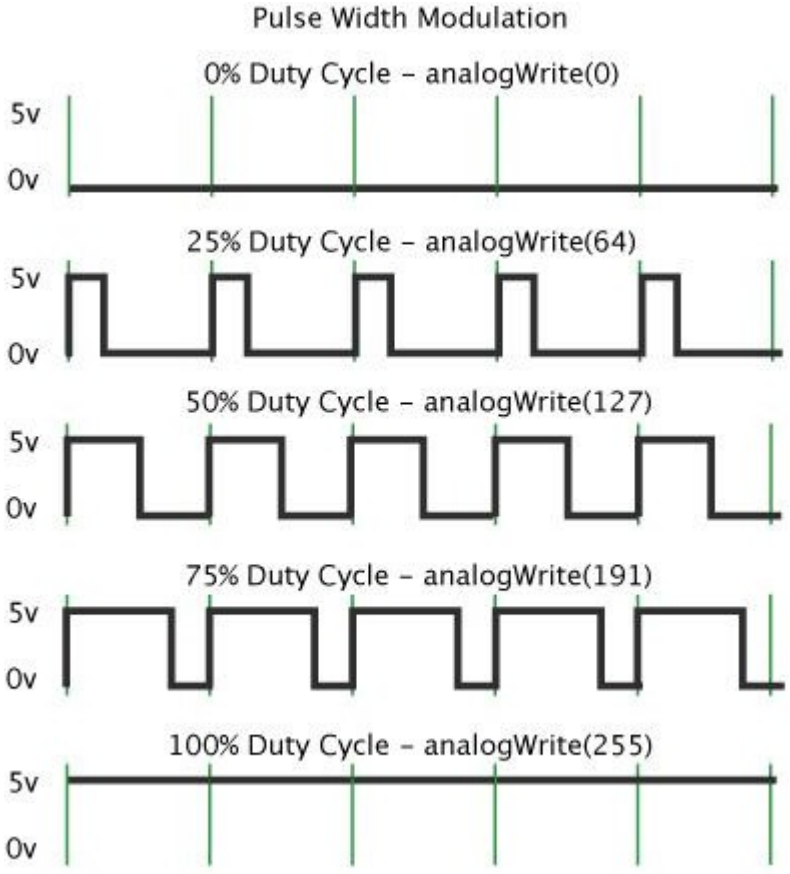
Skratka PWM znamená moduláciu šírky impulzu. Simuluje analógové napätie tým, že mení pomer času, počas ktorého je digitálny signál v stave HIGH a LOW, čo umožňuje nastaviteľné riadenie jasu, rýchlosti a ďalších výstupov.

PWM v skutočnosti nemení úroveň napätia. Namiesto toho spôsobuje, že sa pin rýchlo zapína a vypína. Zmenou pomeru času, počas ktorého zostáva zapnutý (pracovný cyklus), riadime efektívny výstup.

Napríklad pomer času, počas ktorého je LED zapnutá za jednu sekundu:

Pracovný cyklus	Účinok	Význam
0	LED vypnutá	Vždy LOW
50	LED svieti na polovicu	Polovicu času svieti, polovicu času nesvieti času
100	LED plne zapnuté	Vždy zapnuté

Ilustrácia rôznych pracovných cyklov:



Použitie funkcie analogWrite(pin, value):

pin označuje hardvérový pin, ktorý musí podporovať PWM (napr. D3, D5, D6, D9, D10, D11). Hodnota sa pohybuje v rozmedzí od 0 do 255.

Čím vyššia je hodnota, tým jasnejšia je LED.

2. Požadované moduly

Crowtail -- Liner potenciometer × 1

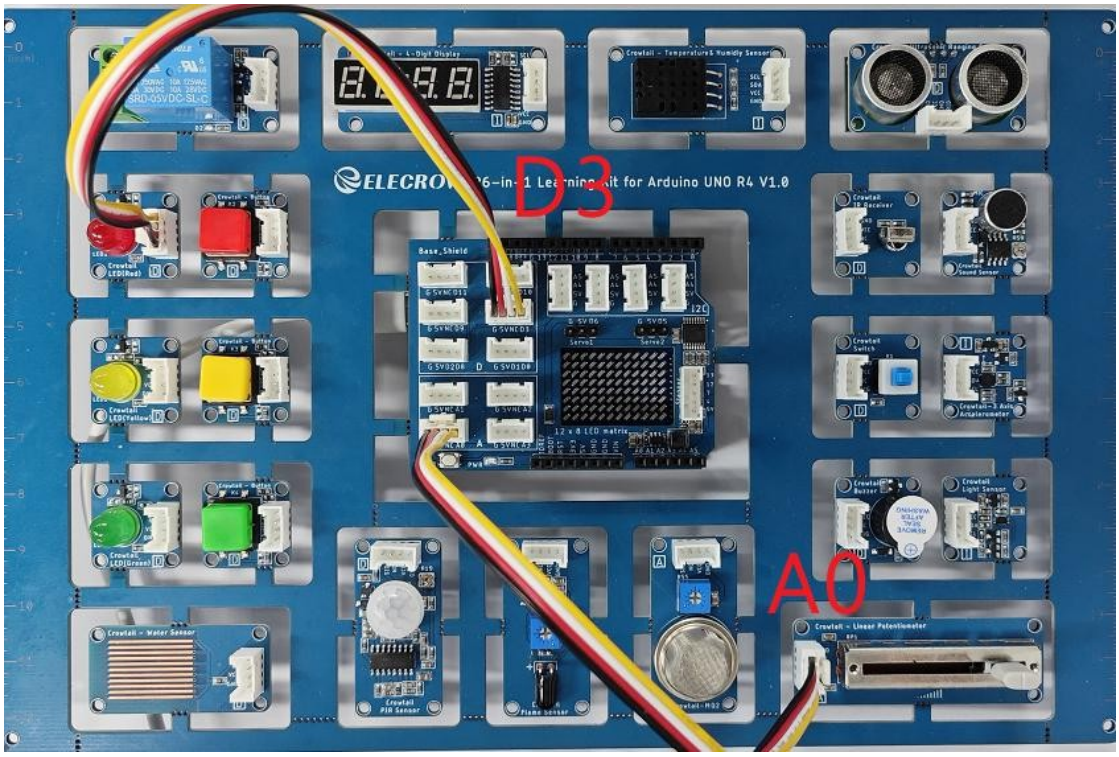
3. Spôsob zapojenia

Crowtail -- Liner potenciometer → Analógový port

A0 Crowtail -- LED → Port DIGITAL D3

Špecifikácia štvorportového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3/A0



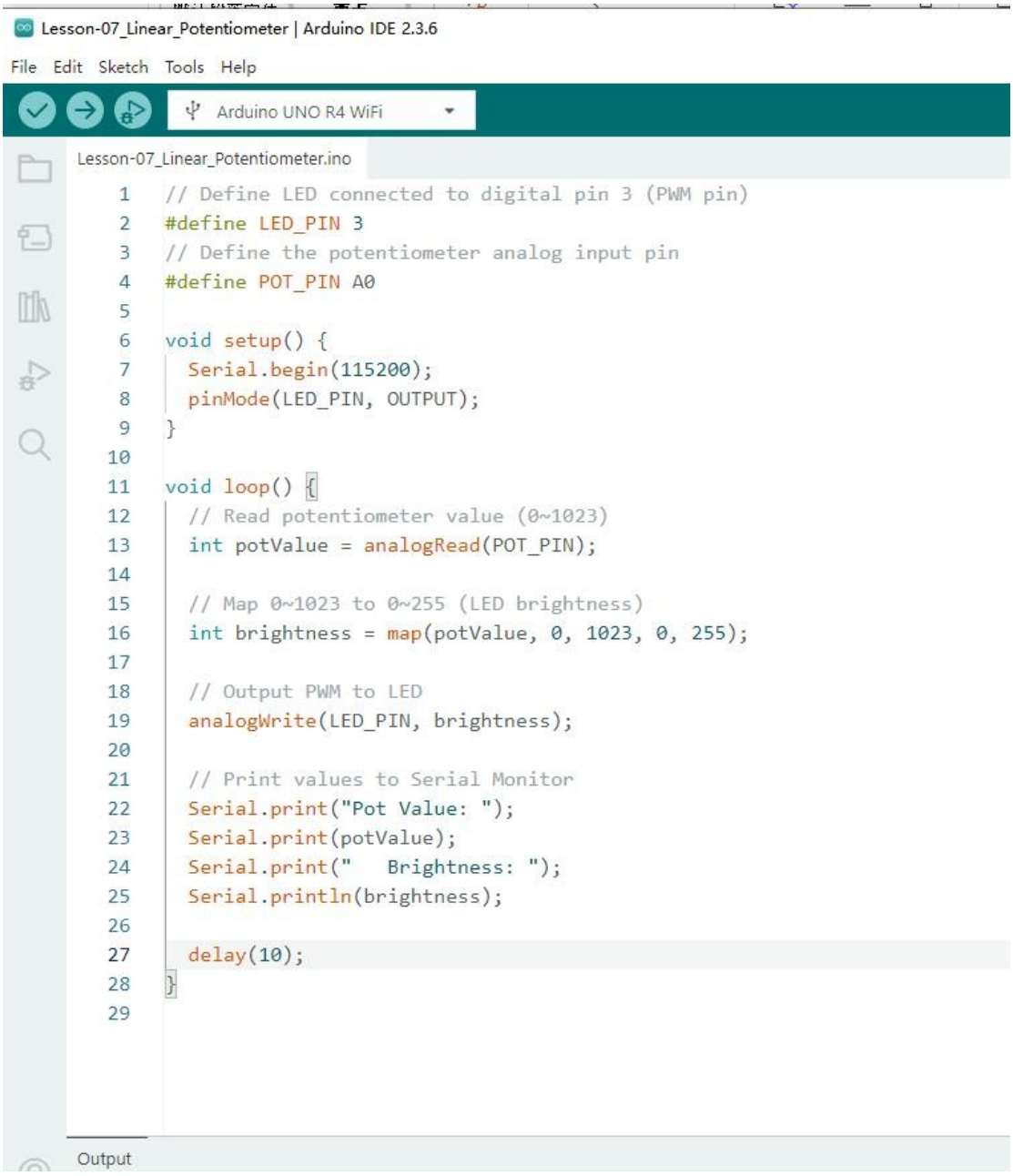
4. Príklad vysvetlenia

Kliknite na nižšie uvedený odkaz a stiahnite si oficiálny ukážkový

kód. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-07_Linear_Potentiometer

Otvorte program pomocou Arduino IDE.



Vysvetlenie kľúčového kódu

(1) Najskôr definujte digitálne piny, aby bol kód ľahšie čitateľný.

```
// Definujte LED pripojenú k digitálnemu pinu 3 (PWM pin)
#define LED_PIN 3
```

```
// Definujte analógový vstupný pin potenciometra
#define POT_PIN A0
```

#define LED_PIN 3 a #define POT_PIN A0 sa používajú na pridelenie výstižných „mien“ pinom. LED_PIN predstavuje digitálny pin 3, čo znamená, že vodič signálu LED je pripojený k D3.

POT_PIN predstavuje analógový pin 0, čo znamená, že signálny vodič potenciometra je pripojený k A0.

Výhodou písania kódu týmto spôsobom je, že ak neskôr budete chcieť presunúť LED alebo senzor na iný pin, stačí zmeniť len tieto dva riadky. Zvyšok kódu zostane nezmenený, čím je program prehľadnejší a ľahšie sa udržiava.

(2) Konfigurácia režimu pinov.

```
pinMode(LED_PIN, OUTPUT);
```

Pin LED je nastavený do režimu OUTPUT, aby sa dal zapínať a vypínať.

Analógové vstupné piny nie je potrebné konfigurovať pomocou pinMode().

(3) Inicializácia sériového portu.

```
Serial.begin(115200);
```

Tento riadok spúšťa sériovú komunikáciu medzi Arduinom a počítačom. 115200 je prenosová rýchlosť, čo znamená, že za sekundu sa prenesie 115 200 bitov. Medzi bežné prenosové rýchlosti patria 9600, 57600 a 115200.

115200 je rýchla a poskytuje rýchly výstup údajov, preto sa bežne používa na ladenie.

Tento riadok musí byť umiestnený v setup(), pretože sériová komunikácia musí byť inicializovaná pri spustení programu. Inak nie je možné vytlačiť žiadne informácie o ladení.

(4) Čítanie analógovej hodnoty

```
// Čítanie hodnoty potenciometra (0–1023)
int potValue = analogRead(POT_PIN);
```

Táto funkcia načíta aktuálnu analógovú hodnotu z POT_PIN a uloží ju do premennej potValue.

analogRead() vráti celé číslo v rozmedzí od 0 do 1023. Vyššia hodnota znamená, že potenciometer je otočený viac doprava, zatiaľ čo nižšia hodnota znamená, že je otočený viac doľava.

analogRead() je funkcia Arduina používaná na čítanie napätia na analógovom pine. Načíta napätie z analógového vstupného pinu (A0, A1, A2 atď.) a prevádza ho na digitálnu hodnotu.

(5) Mapovanie hodnôt — funkcia map()

```
int brightness = map(potValue, 0, 1023, 0, 255);
```

Toto priraďuje hodnotu potenciometra k premennej jasu.

① **Prečo potrebujeme „priradenie“?**

Hodnota načítaná z potenciometra prostredníctvom ADC sa pohybuje v rozmedzí:

0 — 1023 (10-bitové rozlíšenie)

Funkcia analogWrite() však ovláda jas PWM v rozsahu:

0 — 255 (8-bitové rozlíšenie)

Keďže tieto rozsahy sú odlišné, hodnotu ADC nemožno použiť priamo.

Musíme previesť hodnotu ADC v rozsahu 0 – 1023 na hodnotu jasu PWM v rozsahu 0 – 255. Tento proces prevodu sa nazýva **mapovanie**.

② **Čo robí funkcia map()?**

Funkcia map() prevádza hodnotu z jedného rozsahu do druhého, a to proporcionálne. Formát funkcie:

map(input, in_min, in_max, out_min, out_max) V tomto prípade:

map(potValue, 0, 1023, 0, 255)

Význam:

Vezmi potValue (0 — 1023)

Preveď ju proporcionálne na hodnotu medzi 0 a 255

③ **Jednoduchý, intuitívny príklad**

Hodnota ADC potenciometra	Mapovaná jas	Stav LED
0	0	Úplne vypnuté
512 (stred)	≈128	Polovičný jas
1023	255	Plný jas Keď je

potenciometer otočený do polovice, LED svieti tiež polovičným jasom. Vďaka tomu sa zmena javí ľudskému oku plynulá a prirodzená.

Naučiť sa map() znamená naučiť sa, ako premeniť zmeny v reálnom svete na správanie riadené programom.

(6) Ovládanie LED

Logika: použite namapovanú hodnotu na ovládanie jasu LED.

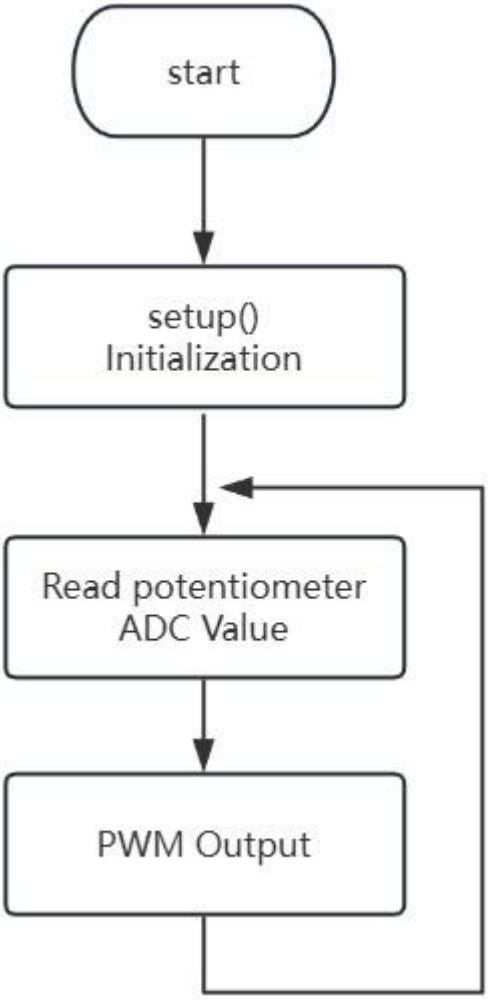
```
// Výstup PWM na LED
analogWrite(LED_PIN, brightness);
```

Týmto sa na výstupe LED_PIN generuje signál PWM, pričom jas je určený hodnotou brightness.

PWM v skutočnosti nemení úroveň napätia. Namiesto toho rýchlo zapína a vypína pin,

a využíva efekt zotrvačnosti videnia, aby LED dióda pôsobila jasnejšie alebo tmavšie.

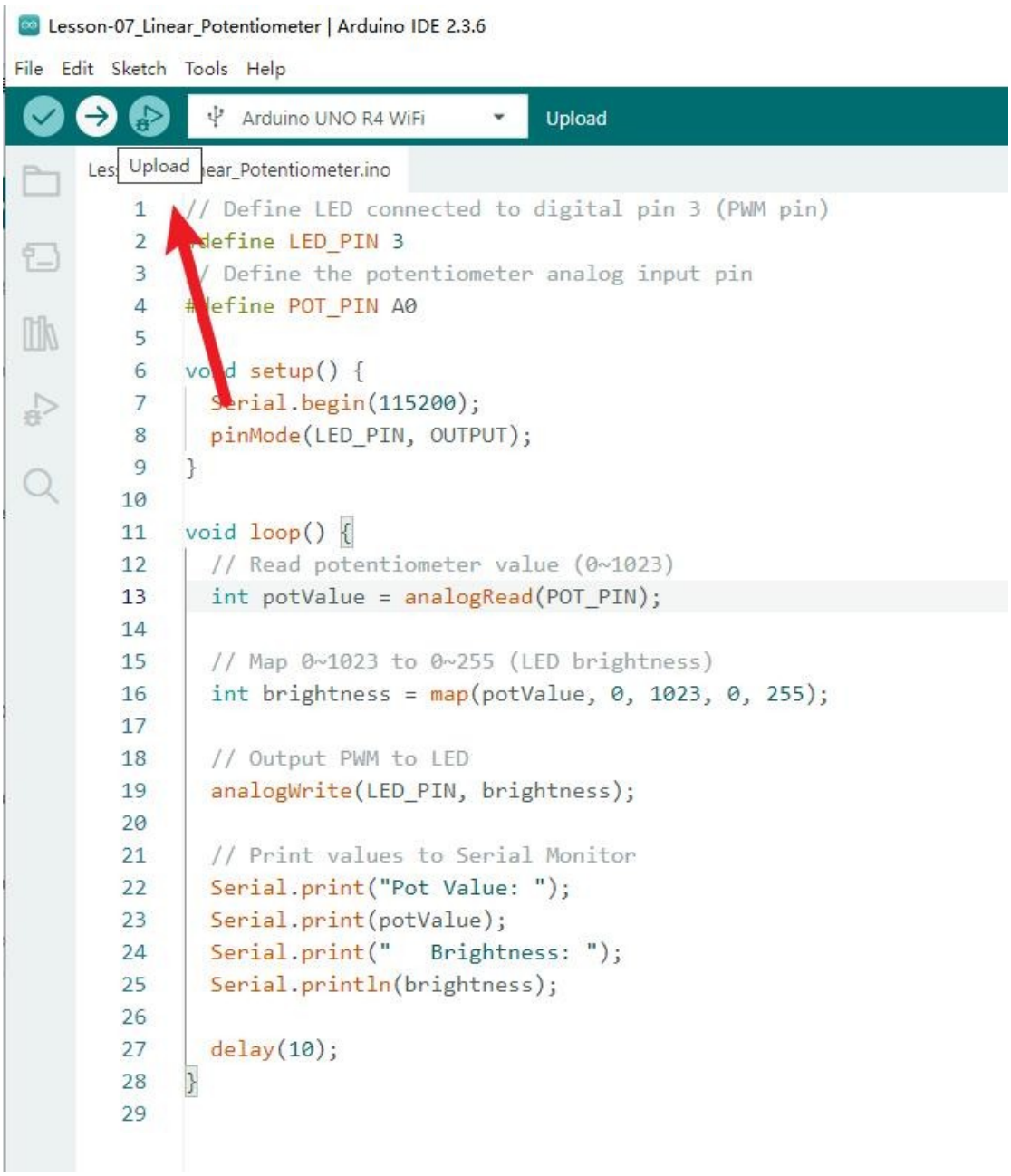
(7) Celkový diagram logického toku kódu



5. Spustite program a sledujte výsledky

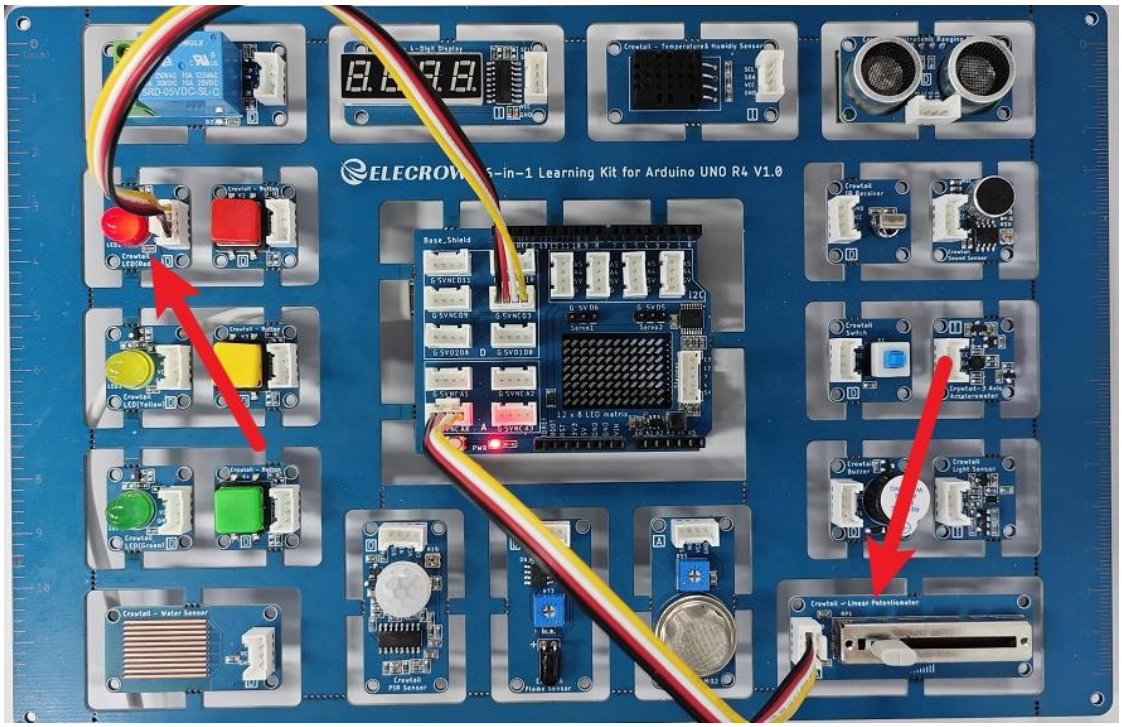
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Stiahnuť“:

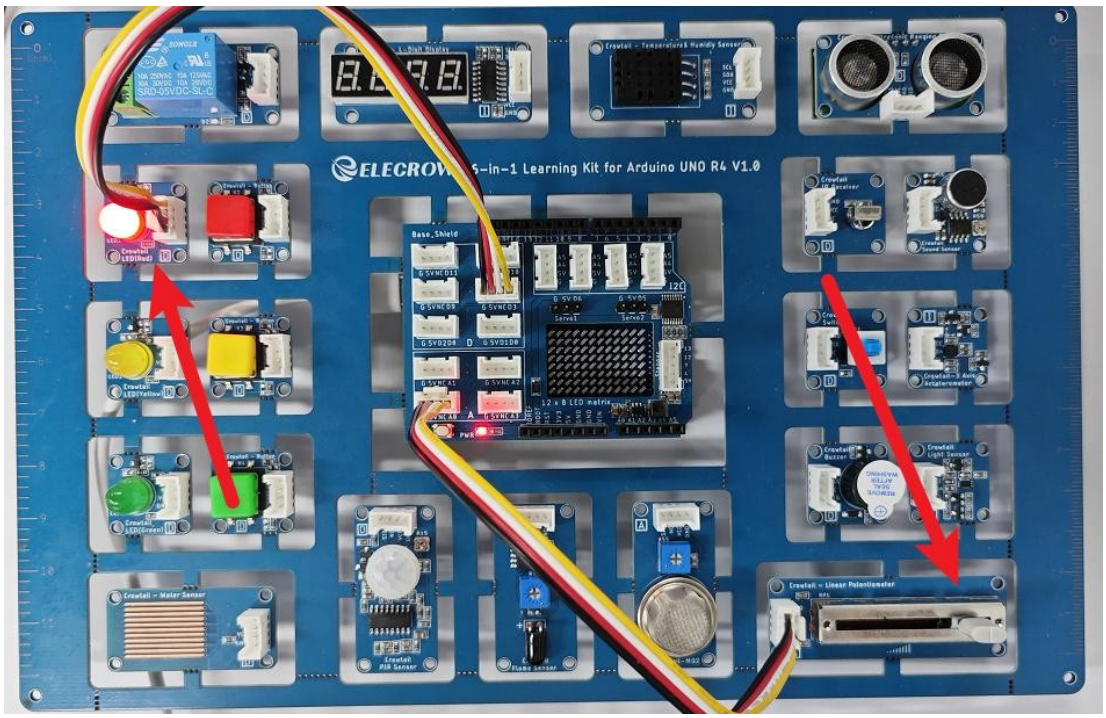


(2) Program sa spustí.

Keď je lineárny potenciometer posunutý úplne doľava, LED svieti veľmi slabo.



Keď je lineárny potenciometer posunutý úplne doprava, LED svieti jasne.



Lekcia 08 — Relé

Úvod

V tejto lekcii sa zoznámime s veľmi bežným modulom riadenia výstupu používaným v automatizácii a systémoch inteligentných domov – relé.

Relé priamo nevytvára svetlo ani zvuk. Funguje skôr ako vypínač napájania ovládaný Arduino, čo umožňuje mikrokontroléru bezpečne ovládať zariadenia s vyšším napätím alebo vyšším výkonom pri prevádzke na nízkom napätí.

V tejto lekcii sa naučíte, ako pomocou Arduina premeniť bežné zariadenie na programovateľný inteligentný vypínač.

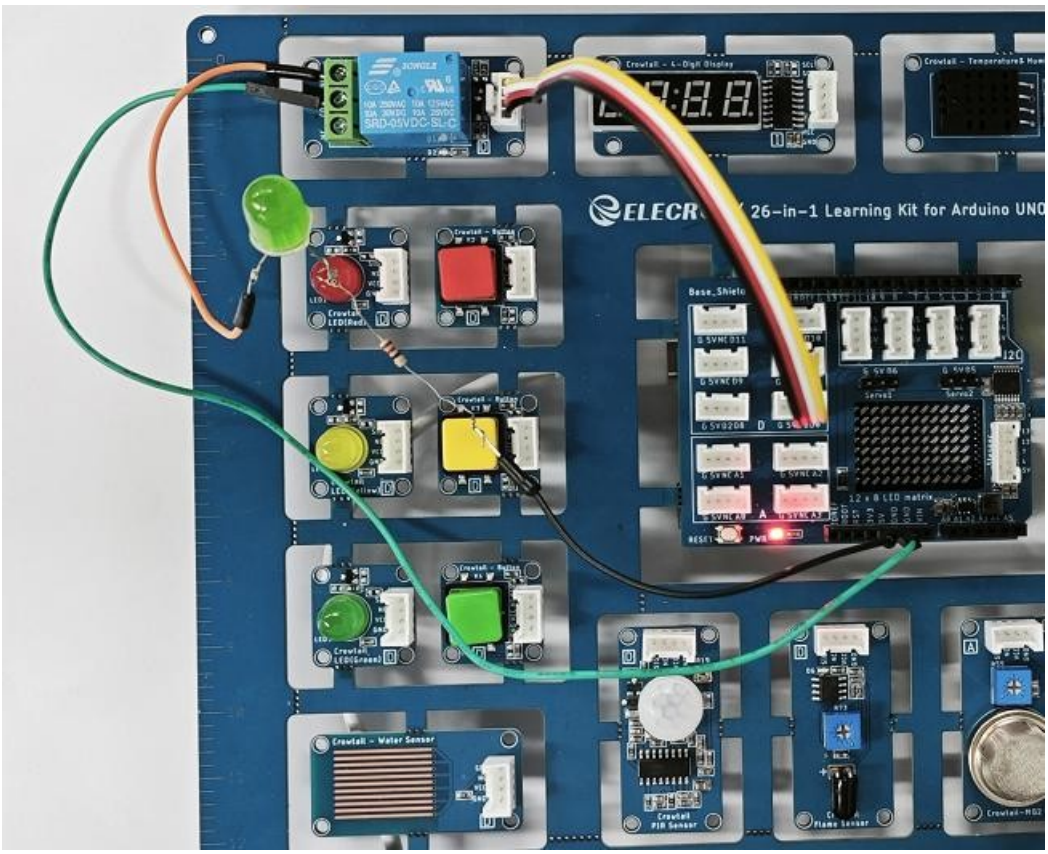
Tieto pojmy tvoria dôležitý základ pre neskoršie témy, ako sú inteligentné zásuvky, automatizované ovládanie a systémy inteligentnej domácnosti.

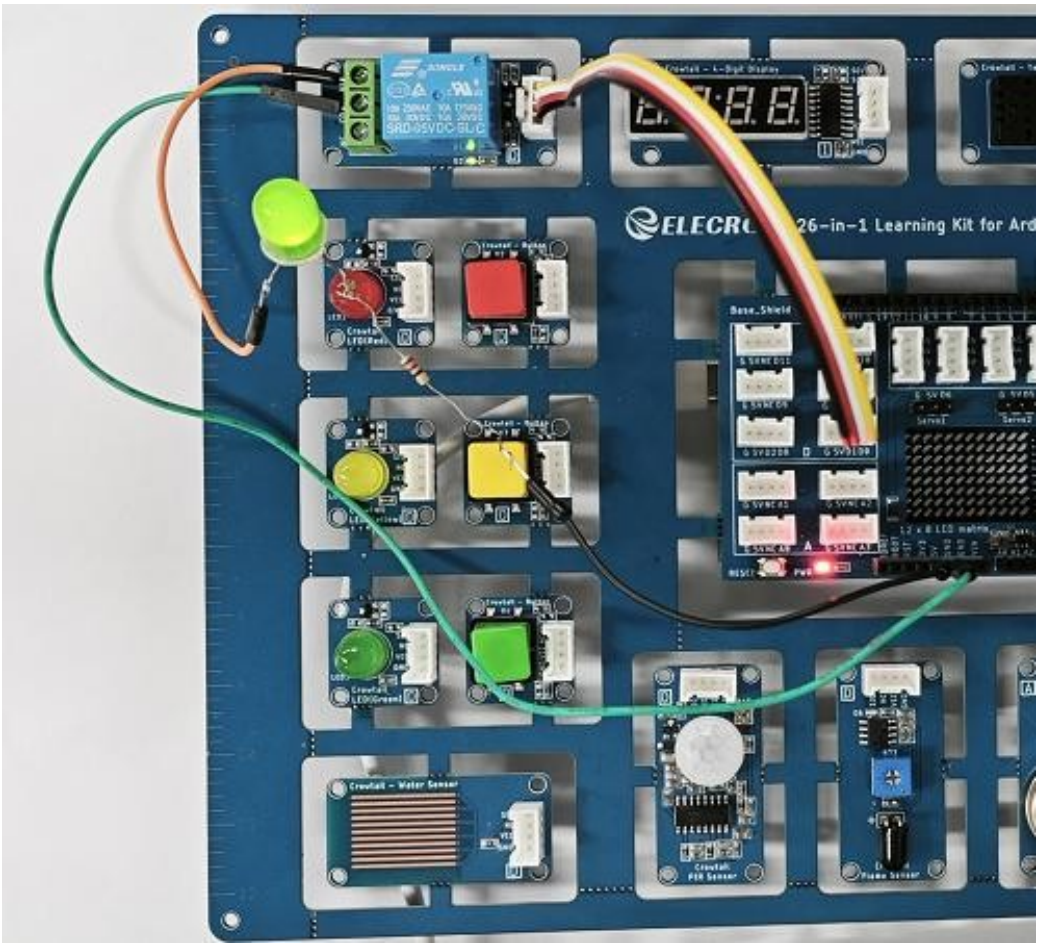
Ciele výučby

1. Porozumieť účelu relé
2. Naučte sa používať reléový modul
3. Porozumejte tomu, ako relé funguje ako „elektronický spínač“, vrátane jeho riadiacej logiky a zapojenia
4. Postavte si kompletnú ukážku spínania relé

Náhľad výsledku

Po zapnutí Arduina je relé štandardne vypnuté a LED dióda sa pravidelne zapína a vypína.

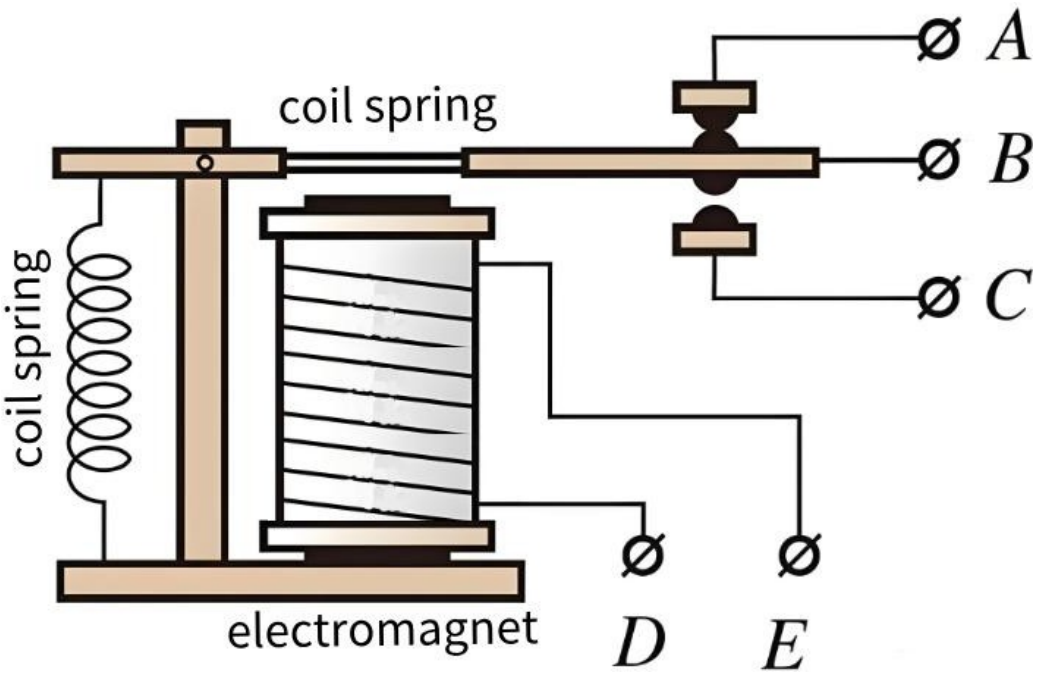

SRD-05VDC-SL-C
26-in-1 Learning Kit for Arduino UNO


SRD-05VDC-SL-C

1. Vysvetlenie princípu

①Ako funguje relé

Relé je v podstate elektronický spínač, ktorý pomocou malého riadiaceho prúdu spína oveľa väčší prúd. Napríklad signál s nízkym prúdom (napríklad výstup z nabíjačky telefónu) možno použiť na zapnutie zariadenia, ktoré vyžaduje oveľa vyšší prúd, ako je žiarovka alebo motor. To nielen chráni riadiace obvody s nízkym prúdom pred poškodením, ale umožňuje aj bezpečné diaľkové ovládanie.



B-COM (spoločný vývod):

Ide o pohyblivý kontakt v strede relé, podobný otočnému bodu hojdačky. Prepína spojenia v závislosti od toho, či je elektromagnet pod napätím.

A-NO (normálne otvorený kontakt):

V základnom stave (keď elektromagnet nie je pod napätím) je odpojený od COM. Keď je relé pod napätím, pripojí sa k COM.

C-NC (normálne uzavretý kontakt):

V predvolenom stave je pripojený k COM. Keď je relé pod napätím, odpojí sa od COM.

Relé bez napätia (normálny stav):

Pružina ťahá pohyblivú kotvu nahor, čím pripája COM k NC (normálne uzavretý kontakt uzavretý), pričom COM zostáva izolovaný od NO (normálne otvorený kontakt otvorený).

Relé pod napätím (aktívny stav):

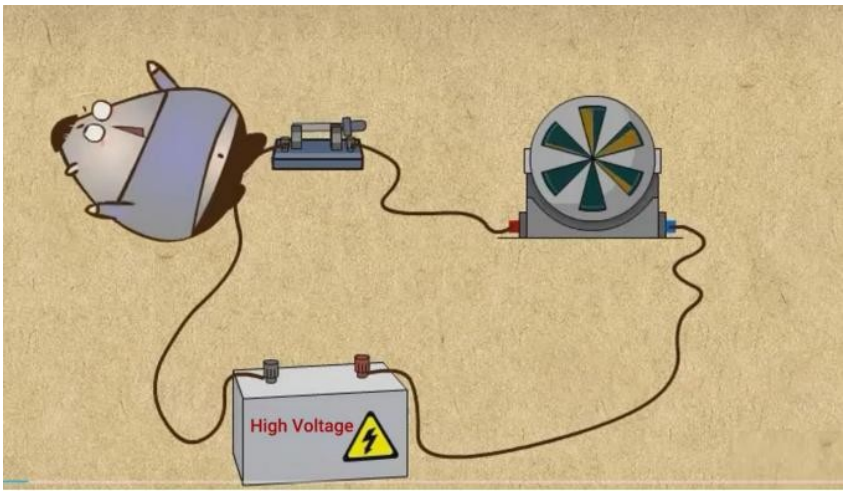
Elektromagnet ťahá kotvu nadol, čím pripája COM k NO (normálne otvorený kontakt uzavretý) a odpojí COM od NC (normálne uzavretý kontakt otvorený).

Na reléovom module sú svorky **NO**, **NC** a **COM** vyvedené prostredníctvom skrutkových svoriek, čo umožňuje ovládať rôzne externé zariadenia. Cievka relé je riadená komponentmi, ako sú tranzistor alebo MOSFET.

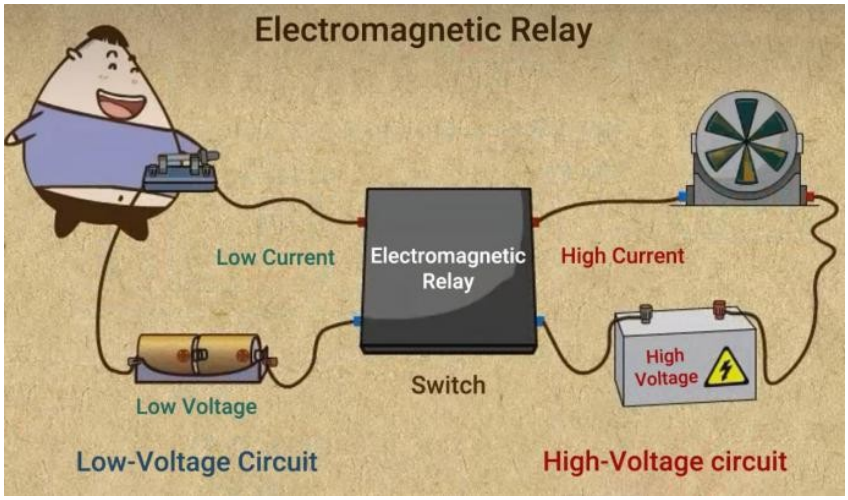
Keď I/O pin Arduina pošle aktivačný signál, výstup HIGH zapne tranzistor alebo MOSFET, čím dodá napájanie cievke relé. Tým sa zmení stav kontaktu COM vo vzťahu k NO a NC, čo umožní relé ovládať externé obvody.

Výhody použitia relé

Bez relé je priame ovládanie vysokonapäťových zariadení veľmi nebezpečné.



Keď vysokonapäťovú stranu spravuje relé, nízkonapäťový riadiaci obvod – a používateľ – zostávajú v bezpečí.



2. Potrebné moduly

Crowtail – Relé × 1

Nasledujúce položky nie sú súčasťou sady a je potrebné ich zabezpečiť samostatne: LED ×1
Prepojovacie vodiče (káble Dupont), niekoľko
Štandardný rezistor ×1

3. Spôsob zapojenia

Crowtail -- Relé → Analógový port D30

Špecifikácia štvorportového rozhrania
Crowtail:

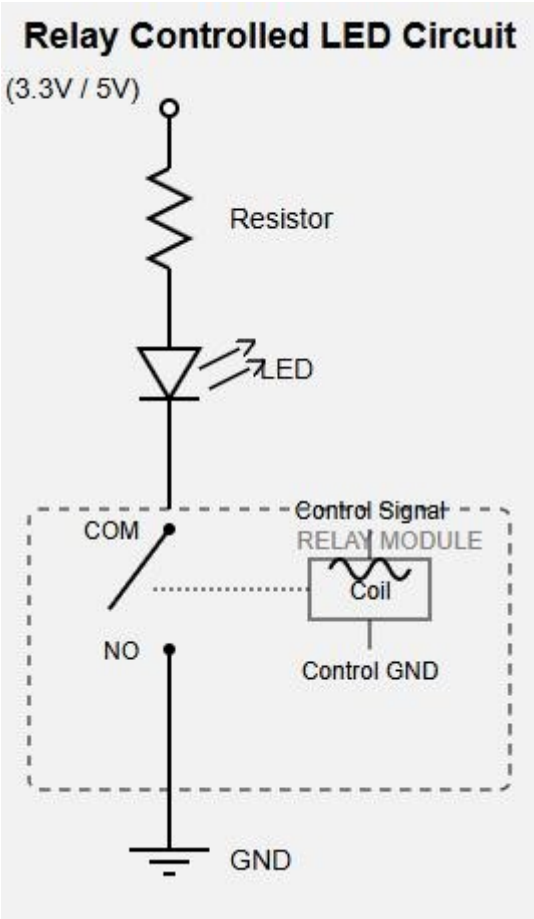
- GND (čierny) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3

Pravidlá zapojenia LED

Najprv musí LED vytvoriť uzavretý obvod. Do tohto obvodu sa potom zapojí relé ako „spínač“.

V tomto prípade používame normálne otvorený (NO) kontakt relé, aby sa správanie LED synchronizovalo so stavom relé. Ak by sa namiesto toho použil normálne uzavretý (NC) kontakt, správanie by bolo opačné.

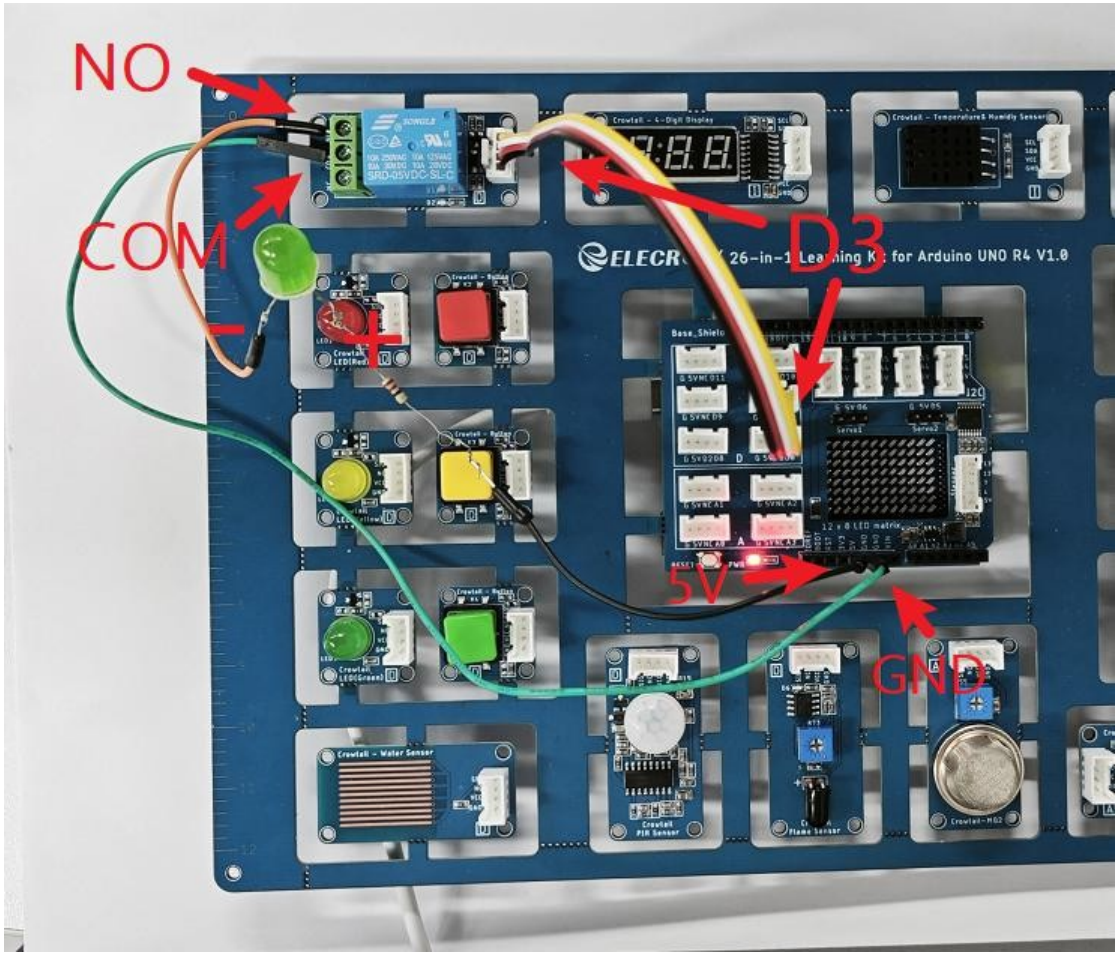
Zostavte kompletný obvod s LED. V tomto príklade sa používa normálne otvorený kontakt, ako je znázornené na schéme nižšie.



Pripojte kladný pól (anódu) LED k 5 V napájaciemu zdroju.

Pripojte záporný pól (katódu) LED k svorku NO (normálne otvorenému) relé. Pripojte svorku COM relé k GND, čím uzavriete obvod.

Tým sa vytvorí uzavretá slučka, v ktorej relé funguje ako spínač, čo vedie k vytvoreniu kompletného obvodu LED riadeného relé.



4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-08_Relay.ino

Otvorte program pomocou Arduino IDE.

Vysvetlenie kľúčového kódu

(1) Najskôr definujte digitálny pin, aby bol kód ľahšie čitateľný.

```
// Definujte digitálny pin používaný na ovládanie reléového modulu
#define RELAY_PIN 3 // Ovládací signál relé pripojený na digitálny pin D3
```

RELAY_PIN predstavuje digitálny pin D3, ktorý sa používa na ovládanie reléového modulu.

Výhodou tohto spôsobu písania kódu je, že ak neskôr budete chcieť presunúť tlačidlo alebo reléový modul na iný pin, stačí tu zmeniť len číslo pinu. Zvyšok kódu zostane nezmenený, vďaka čomu je program prehľadnejší a ľahšie sa udržiava.

(2) Konfigurácia režimu pinov.

```
// Konfigurácia pinu relé ako výstupu
// To umožňuje Arduinu posielať signály HIGH / LOW na ovládanie relé
pinMode(RELAY_PIN, OUTPUT);
```

RELAY_PIN (pin na ovládanie relé) je nastavený do režimu OUTPUT, čo umožňuje Arduinu ovládať reléový modul pomocou signálov HIGH a LOW.

(3) Inicializácia sériovej komunikácie.

```
// Inicializácia sériovej komunikácie medzi Arduinom a počítačom
// 115200 je prenosová rýchlosť (rýchlosť komunikácie)
Serial.begin(115200);
Serial.println("Spustenie ukážky automatického ovládania relé");
```

Tento kód spustí sériovú komunikáciu medzi Arduinom a počítačom. 115200 je prenosová rýchlosť, čo znamená, že za sekundu sa prenesie 115 200 bitov.

Začiatočníkom dôrazne odporúčame, aby si pri písaní kódu zvykli vždy zapínať sériový monitor, pretože to výrazne uľahčuje sledovanie správania programu a odstraňovanie chýb.

```
digitalWrite(RELAY_PIN, LOW);
```

Tým sa zabezpečí, že relé bude pri zapnutí vypnuté.

Účel:

Zabraňuje náhodnému spusteniu relé pri prvom zapnutí Arduina. Výslovne definuje počiatočný stav relé ako VYPNUTÉ.

(4) Zapnutie relé signálom HIGH

```
// Zapnutie relé
// HIGH napája cievku relé a uzatvára NO kontakt digitalWrite(RELAY_PIN,
HIGH);
Serial.println("Relé zapnuté (LED svieti)");
```

Signál HIGH napája cievku relé a uzatvára normálne otvorený (NO) kontakt volaním digitalWrite(RELAY_PIN, HIGH). Sériový monitor vypíše „Relé zapnuté (LED svieti)“.

(5) Vypnutie relé signálom LOW.

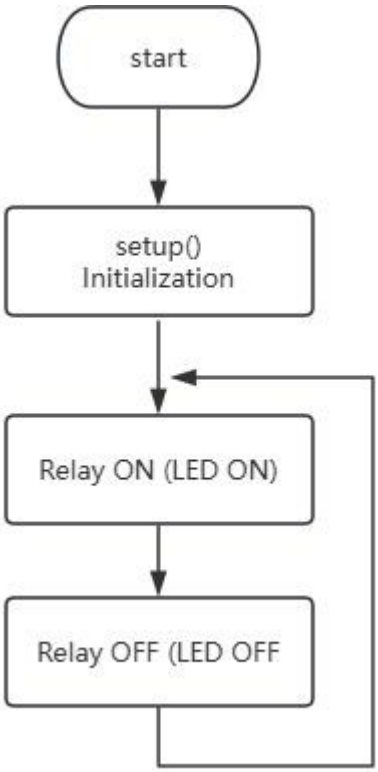
```
// Vypnutie relé
// Nízka úroveň odpojí cievku relé a otvorí kontakt
```

```
digitalWrite(RELAY_PIN, LOW);
Serial.println("Relé vypnuté (LED vypnuté)");
```

Signál LOW odpojí napájanie cievky relé a otvorí kontakt nastavením RELAY_PIN na LOW pomocou digitalWrite().

Sériový monitor vypíše „Relé VYPNUTÉ (LED VYPNUTÁ)“.

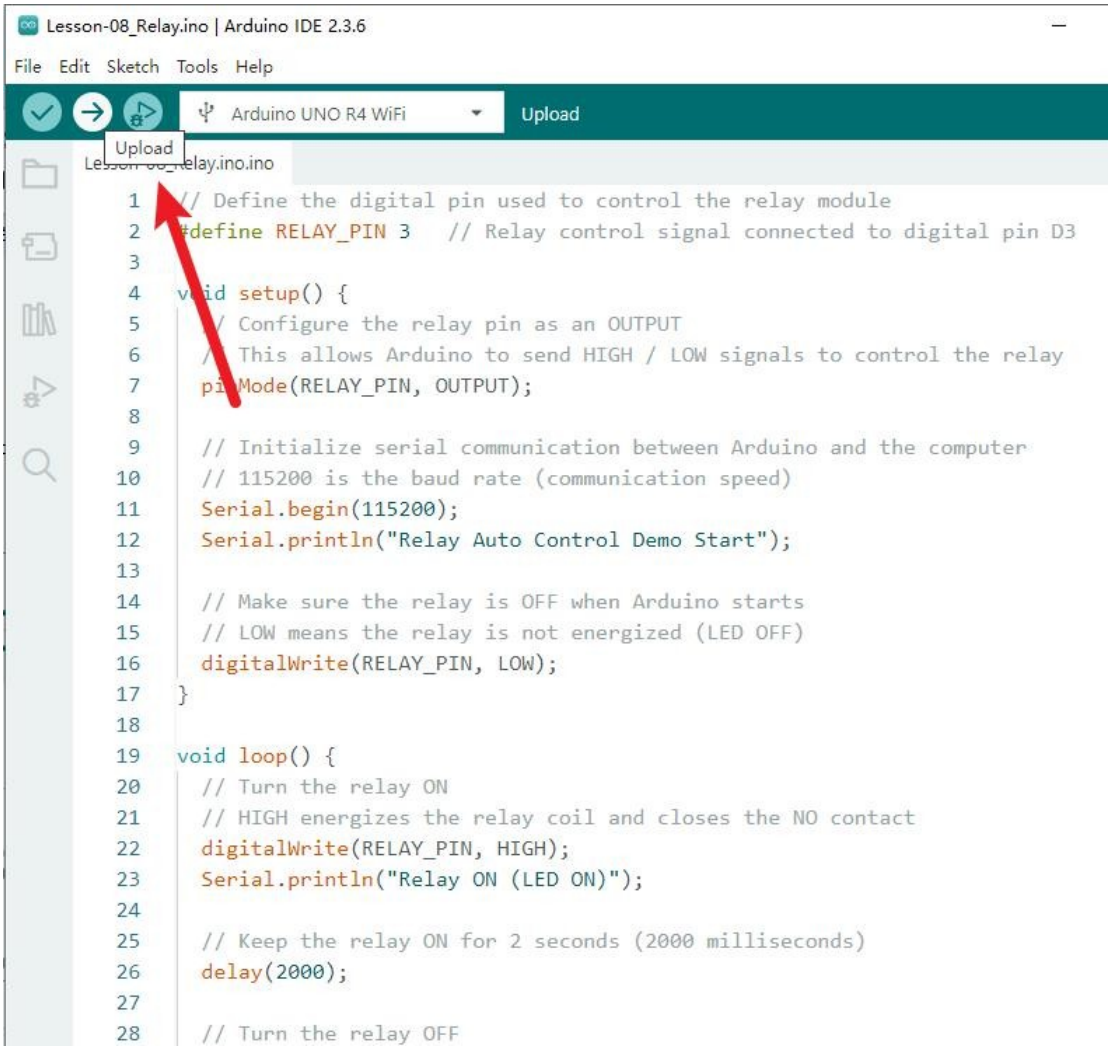
(6) Celkový diagram toku logiky kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

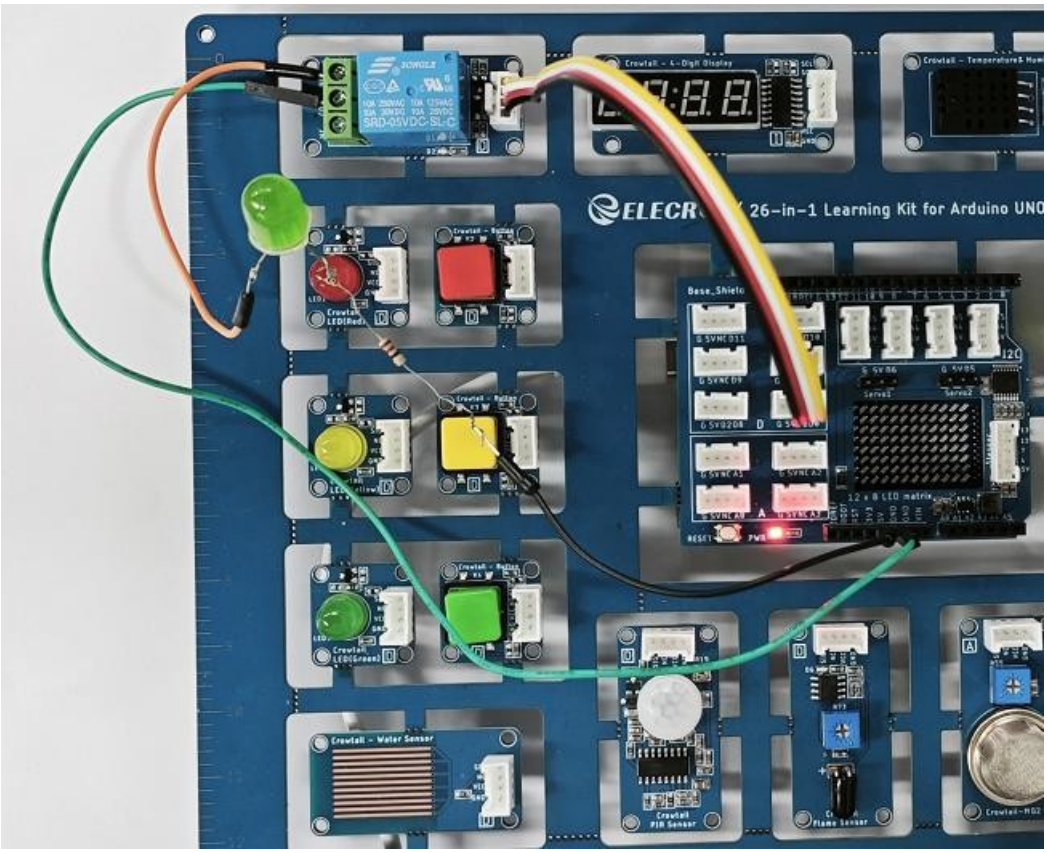
Kliknite na „Stiahnuť“:

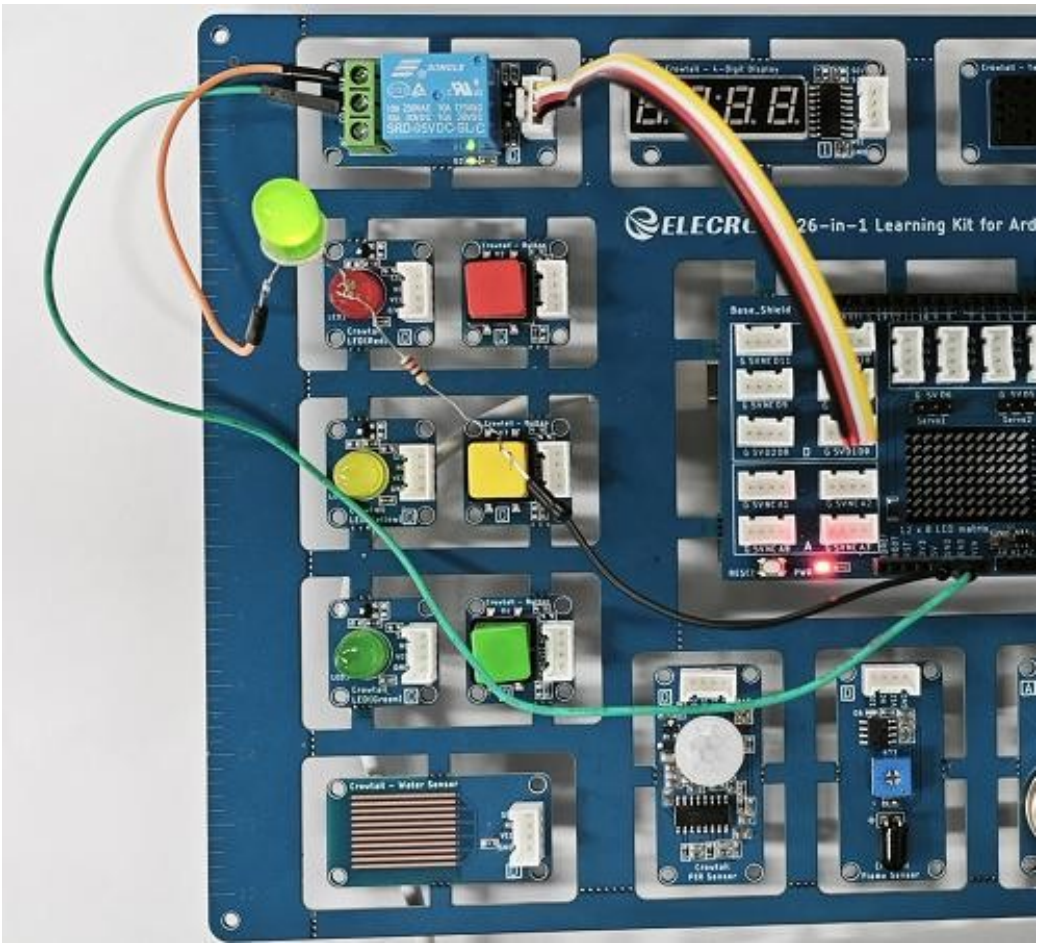


(2) Program sa spustí.

Správanie programu:

Počujete, ako relé vydáva zreteľné „kliknutie“, a LED dióda sa pravidelne zapína a vypína.


SRD-05VDC-SL-C
26-in-1 Learning Kit for Arduino UNO
Base_Shield
Crowtail - Water Sensor
Crowtail PIR Sensor
Crowtail Flame Sensor


SRD-05VDC-SL-C
26-in-1 Learning Kit for Ard
Base_Shield
Crowtail - Water Sensor
Crowtail PIR Sensor
Crowtail Flame Sensor

Lekcia 09 – Senzor vody

Úvod

V tejto lekcii si zostavíme jednoduché „zariadenie na varovanie pred dažďom“ s využitím senzora dažďových kvapiek a LED diódy. Prostredníctvom tohto malého experimentu pochopíte základný princíp fungovania senzora dažďových kvapiek a ďalej sa zoznámite s operáciami čítania a zápisu na digitálnych pinov, sériovým výstupom a používaním podmienených príkazov typu „if“. Tým si vytvoríte pevný základ pre zložitejšie projekty v budúcnosti.

Ciele výučby

1. Porozumieť princípu fungovania senzora dažďových kvapiek.
2. Zopakovať a upevniť gramatiku „if else“, aby ste ju mohli používať efektívnejšie.
3. Dokončiť simuláciu prípadu dažďového alarmu.

Náhľad výsledku

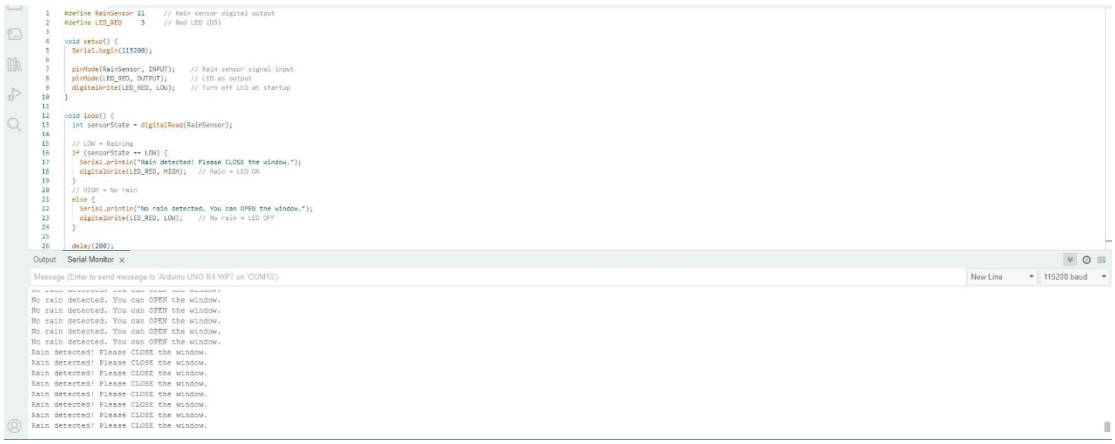
Po spustení programu bude systém disponovať nasledujúcimi funkciami:

Keď senzor zaznamená príchod dažďa: Červená LED dióda sa automaticky rozsvieti ako varovanie Keď nie je zaznamenaný žiadny dážď: Červená LED dióda sa automaticky zhasne

Vždy, keď sériový port zistí prítomnosť dažďa, zobrazí nasledujúci text:

- Zistená voda → Zobrazenie „Zistený dážď! Prosím, ZATVORTE okno.“

- Žiaden dážď → Zobraziť „Žiaden dážď nezistený. Okno môžete OTVORIŤ.“



1. Vysvetlenie princípu

Vodný senzor (Water Sensor) je typ senzora, ktorý detekuje vlhkosť na základe vodivosti. Keď kvapka vody príde do kontaktu s povrchom senzora, vlhkosť vytvorí vodivú dráhu, čo spôsobí zmenu výstupného stavu senzora.

Tento modul sa skladá zo **4 základných komponentov**:

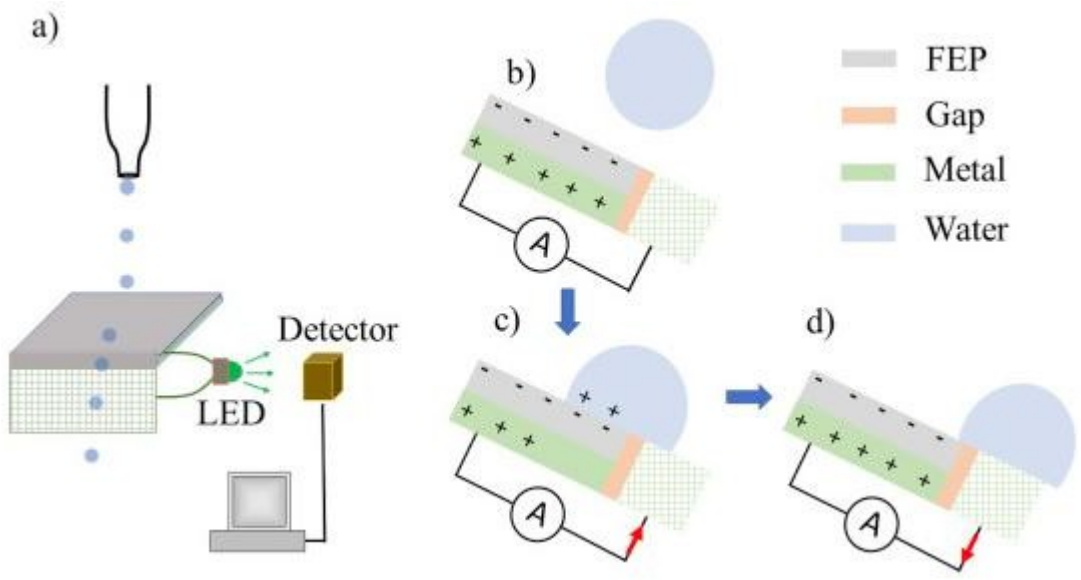
- Doska citlivá na vlhkosť (dlhá obdĺžniková kontaktná plocha vľavo na obrázku): Ide o rad paralelných medených kontaktných bodov zobrazených na obrázku. Keď na ňu dopadne kvapka vody, zmení sa vodivosť medzi kontaktmi;
- Porovnávací čip (malý čierny čip vpravo na obrázku): Skrytý na doske s obvodmi, tento malý čierny čip je zodpovedný za prevod „zmeny vodivosti kontaktov“ na digitálny signál;
- Digitálny výstupný konektor (biele rozhranie vpravo na obrázku): Ide o biely konektor a kolík označený **ako „SIG“** sa používa na pripojenie k digitálnym vstupným kolíkom Arduina/ESP32;
- Logická úroveň je tiež v súlade so štandardným modulom:
  Keď kontaktná plocha zistí kvapku vody (vlhkosť), pin SIG vydáva signál LOW (nízka úroveň);
  Keď je kontakt suchý, pin SIG vysiela HIGH (vysoká úroveň).

Preto môžeme na zistenie aktuálneho stavu použiť funkciu digitalRead() v Arduine.

Základný princíp fungovania

Tento senzor kvapiek vody je založený na detekcii kapacity ako svojom jadre. Kombinuje konštrukčný dizajn a konverziu signálu na dosiahnutie detekcie kvapiek vody: Jeho základná štruktúra je kapacitná jednotka typu „kovová elektróda + izolačná vrstva FEP“. V počiatočnom stave tvorí kovová elektróda kapacitu s vonkajším vzduchom a FEP slúži ako izolačné médium. V tomto momente je hodnota kapacity na referenčnej úrovni. Keď kvapka vody (voda je vodivá) spadne a priblíži sa k senzoru, nahradí vzduch ako „druhá elektródová doska“ kapacity. Vzhľadom na oveľa vyššiu dielektrickú konštantu vody v porovnaní so vzduchom a zníženie vzdialenosti medzi kvapkou vody a senzorom táto zmena spôsobuje, že hodnota kapacity začne stúpať. Keď kvapka vody dosadne na povrch senzora a pokryje ho, kontaktná plocha medzi kvapkou vody a kovovou elektródou sa ďalej rozširuje a efektívna plocha kapacitných dosiek sa zvyšuje, čo má za následok ďalšie výrazné zvýšenie hodnoty kapacity. Zároveň LED a detektor, ktoré sú súčasťou senzora, spustia detekčný obvod, aby začal pracovať počas procesu dopadu kvapky vody.

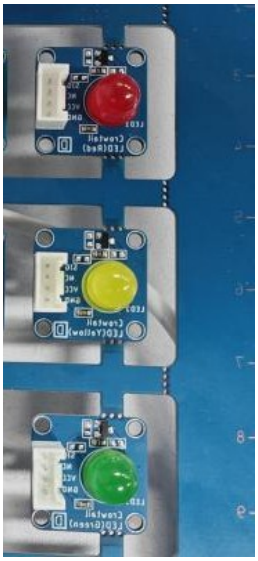
Nakoniec senzor prevádza nepretržité zmeny kapacity na dynamické zmeny aktuálnych hodnôt, ktoré zodpovedajú rôznym stavom „približovania sa – kontaktu – pokrytia“ kvapky vody, čím presne detekuje prítomnosť, polohu a stupeň pokrytia kvapky vody.



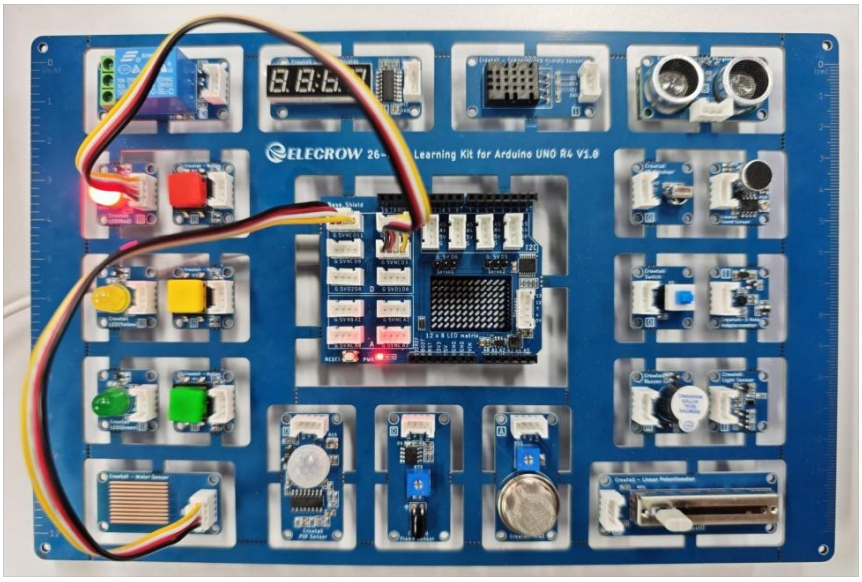
2. Požadované moduly

Vodný senzor Crowtail (1 ks) LED dióda

Crowtail (ľubovoľná farba x1)



3. Spôsob zapojenia



Senzor vody → port DIGITAL D11 LED dioda

Crowtail → port DIGITAL D3 Špecifikácia
štvorpinového rozhrania Crowtail:

SIG (žltá) / NC (biela) / VCC (červená) / GND (čierna)

4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-09_WaterSensor/9_WaterSensor

Otvorte program pre túto lekciu v priečinku **„9_WaterSensor“** pomocou Arduino IDE:

```
#define RainSensor 11     // Rain sensor digital output
#define LED_RED     3     // Red LED (D3)

void setup() {
  Serial.begin(115200);

  pinMode(RainSensor, INPUT);    // Rain sensor signal input
  pinMode(LED_RED, OUTPUT);      // LED as output
  digitalWrite(LED_RED, LOW);    // Turn off LED at startup
}

void loop() {
  int sensorState = digitalRead(RainSensor);

  // LOW → Raining
  if (sensorState == LOW) {
    Serial.println("Rain detected! Please CLOSE the window.");
    digitalWrite(LED_RED, HIGH);   // Rain → LED ON
  }
  // HIGH → No rain
  else {
    Serial.println("No rain detected. You can OPEN the window.");
    digitalWrite(LED_RED, LOW);    // No rain → LED OFF
  }

  delay(200);
}
```

Vysvetlenie kľúčového kódu:

(1) #define definuje konštanty pinov

```
#define RainSensor 11          // Digitálny výstup dažďového senzora
#define LED_RED 3              // Červená LED (D3)
```

- Pomenujte 11. pin **„RainSensor“**
- Pomenujte 3. pin **„LED_RED“**

Dôvod pre toto:

✓ Keď v budúcnosti uvidíte **„RainSensor“**, budete vedieť, že toto rozhranie je pripojené k senzoru dažďa

✓ Kód sa stane čitateľnejším

✓ Pri úprave pinov stačí zmeniť len tento riadok namiesto úpravy celého kódu

(2) Inicializačná časť „setup()“

```
void setup() {
  Serial.begin(115200);
```

```
    pinMode(RainSensor, INPUT);          // Vstup signálu senzora vody
    pinMode(LED_RED, OUTPUT);                // LED nastavená ako
                                             výstup
    digitalWrite(LED_RED, LOW);          // Vypnutie LED pri
                                             spustení
}
```

- Aktivujte funkciu sériovej komunikácie Arduina, aby sme mohli vidieť textový výstup na počítači.

  115200 je rýchlosť komunikácie (musí byť rovnaká ako v sériovom monitore).

- **pinMode(RainSensor, INPUT):** Oznámi Arduinu: Tento pin je v režime vstupu a slúži na čítanie signálu HIGH alebo LOW zo senzora.
- pin**Mode(LED_RED, OUTPUT):** Ak chcete ovládať zapnutie/vypnutie LED diódy, musíte najprv nastaviť režim pinov LED na výstupný režim.
- **digitalWrite(LED_RED, LOW):** Pri spustení je LED dioda štandardne vypnutá, aby sa zabránilo náhodnému rozsvieteniu pri zapnutí napájania.

(3) Hlavná slučka loop() číta stav senzora

```
void loop() {
    int sensorState = digitalRead(RainSensor);
```

Po prvé, RainSensor je alias, ktorý sme priradili k **GPIO 11**. Počas vykonávania programu umožňuje tento názov presne identifikovať príslušný pin.

Ďalej, funkcia **digitalRead()** slúži na odoslanie príkazu internému GPIO radiču čipu: „prečítaj úroveň tohto pinu“. Po vykonaní príkazu GPIO radič okamžite zistí stav napätia na pine 11. Rozpoznáva dve kľúčové úrovne:

- Vysoká úroveň: 3,3 V (alebo 5 V pre Arduino), zodpovedajúca číslu 1
- Nízka úroveň: 0 V, zodpovedajúca číslu 0

Pin SIG senzora dažďových kvapiek používa tieto dve úrovne na reprezentáciu stavov „sucho“ a „dážď“.

Nakoniec príkaz **int sensorState = digitalRead(RainSensor);** uloží hodnotu 1 alebo 0 (predstavujúcu stav napätia), ktorú zistil ovládač GPIO, vo forme celého čísla do premennej sensorState.

(4) Určenie, či bola zistená voda

Ak je zistená voda (LOW)

```
    if (sensorState == LOW) {
        Serial.println("Zistený dážď! Prosím, ZATVORTE okno.");
        digitalWrite(LED_RED, HIGH);
```

```
    }
```

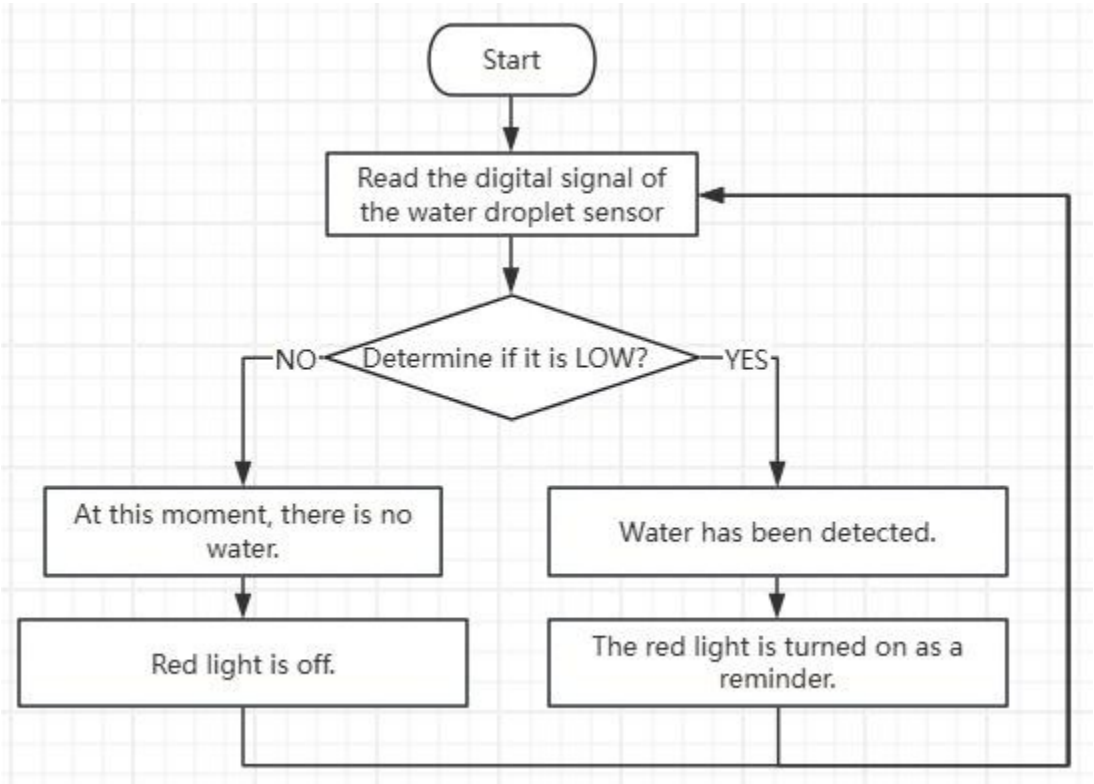
- If **(sensorState == LOW)** znamená, že ak je stav snímača LOW, čo znamená, že prší, potom sa vykoná obsah v zátvorkách. **Serial.println("Prší. Zatvorte okno!");** Tento riadok slúži na vytlačenie textu na počítači, vďaka čomu môžete vidieť
  správa v monitore sériového portu, ktorá vás informuje, že prší a mali by ste zavrieť okná.
- **digitalWrite(LED_RED, HIGH)** označuje, že svetlo by malo byť zapnuté. Tu môže stav zapnutia/vypnutia svietenia môže slúžiť ako indikátor dažďa. Napríklad zapnuté svetlo znamená, že práve prší.

Ak je zistená suchosť (HIGH)

```
    else {
        Serial.println("Nebol zistený dážď. Môžete OTVORIŤ okno.");
        digitalWrite(LED_RED, LOW);
    }
    delay(200);
}
```

- Ak sensorState nie je LOW, musí byť HIGH, čo znamená, že neprší. Kód tak prejde do sekcie else. **Serial.println("Zistený žiadny dážď. Môžete otvoriť okno.");**
  Táto veta jednoducho hovorí: „Ak neprší, môžete otvoriť okno.“ Kód **„digitalWrite(LED_RED, LOW)“** vypne svetlo, čo znamená, že aktuálne počasie je normálne.
- Nakoniec **„delay(200)“** pozastaví program na 200 milisekúnd, čo je 0,2 sekundy. To je urobené s cieľom zabrániť príliš častému snímaniu, zabrániť príliš rýchlemu blikaniu vytlačeného textu a znížiť kolísanie snímača.

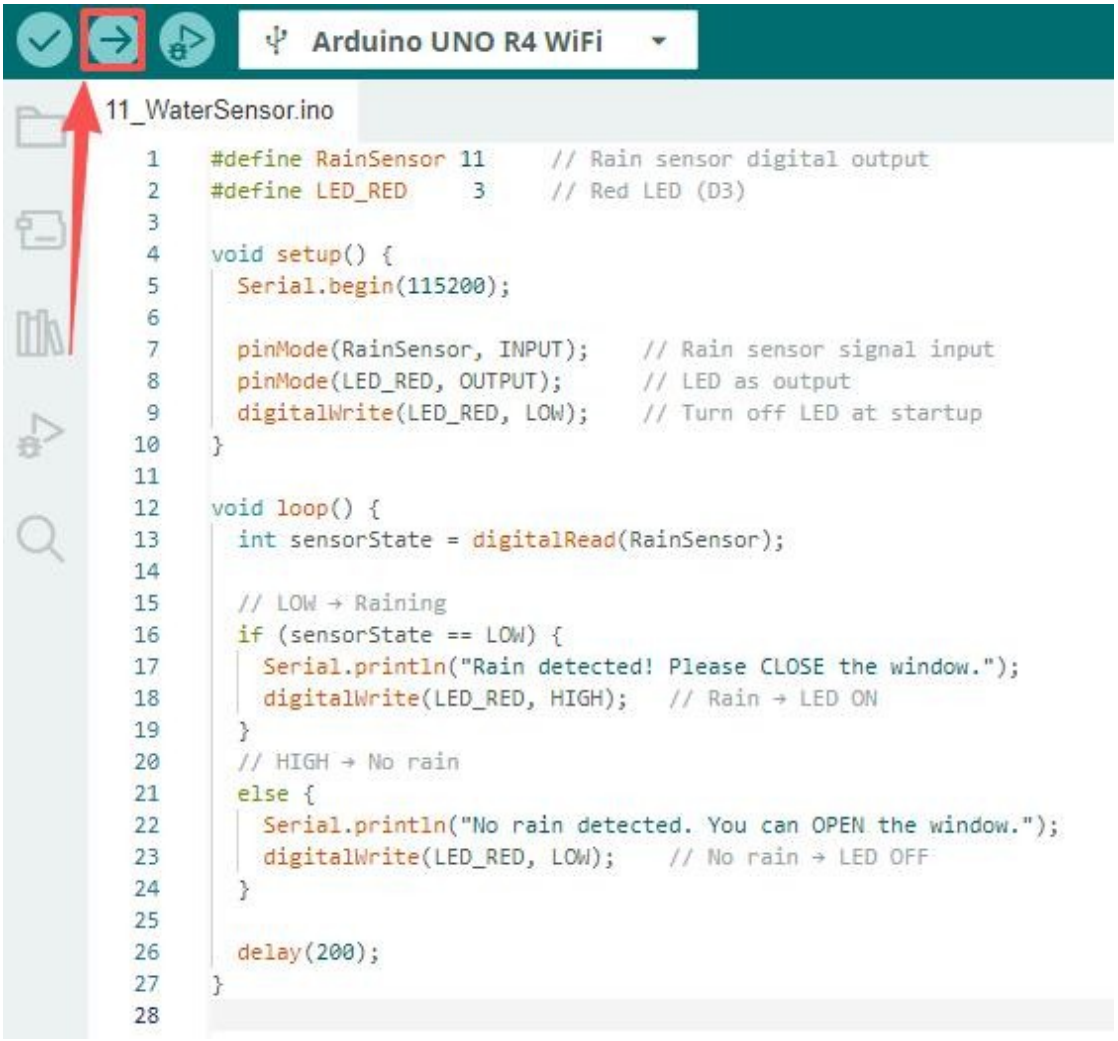
(5) Logický tok kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“**:

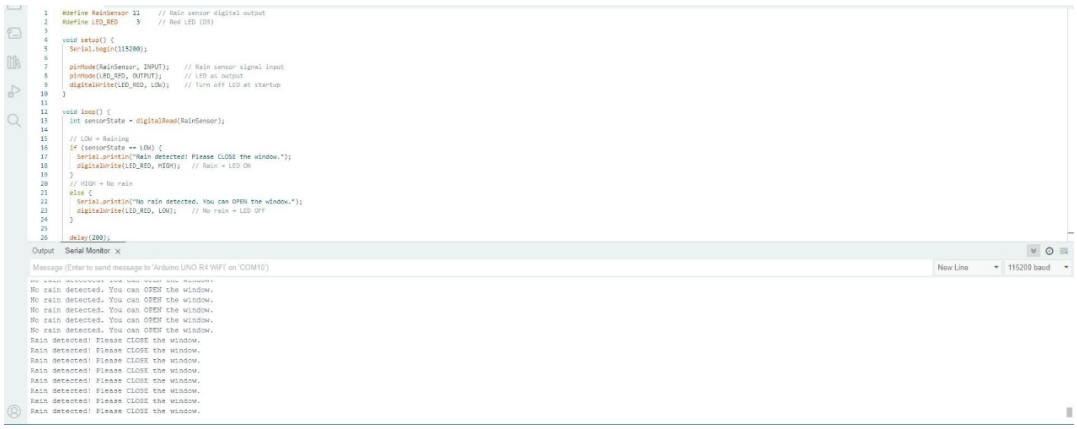



```ino
#define RainSensor 11     // Rain sensor digital output
#define LED_RED     3     // Red LED (D3)

void setup() {
  Serial.begin(115200);

  pinMode(RainSensor, INPUT);     // Rain sensor signal input
  pinMode(LED_RED, OUTPUT);       // LED as output
  digitalWrite(LED_RED, LOW);     // Turn off LED at startup
}

void loop() {
  int sensorState = digitalRead(RainSensor);

  // LOW → Raining
  if (sensorState == LOW) {
    Serial.println("Rain detected! Please CLOSE the window.");
    digitalWrite(LED_RED, HIGH);   // Rain → LED ON
  }
  // HIGH → No rain
  else {
    Serial.println("No rain detected. You can OPEN the window.");
    digitalWrite(LED_RED, LOW);    // No rain → LED OFF
  }

  delay(200);
}

```

(2) Po úspešnom stiahnutí skontrolujte výsledok:

Otvorte monitor sériového portu

(Ak v tomto momente kvapnete vodu na senzor dažďa, na monitore sériového portu sa zobrazia príslušné informácie a rozsvieti sa červené svetlo, ktoré vás upozorní, aby ste zavreli okno; ak tam nie je voda, červené svetlo zhasne)



```
#define RainSensor 11     // Rain sensor digital output
#define LED_RED    3      // Red LED (D3)

void setup() {
  Serial.begin(115200);

  pinMode(RainSensor, INPUT);    // Rain sensor signal input
  pinMode(LED_RED, OUTPUT);      // LED as output
  digitalWrite(LED_RED, LOW);    // Turn off LED at startup
}

void loop() {
  int sensorState = digitalRead(RainSensor);

  // LOW = Raining
  if (sensorState == LOW) {
    Serial.println("Rain detected! Please CLOSE the window.");
    digitalWrite(LED_RED, HIGH);   // Rain → LED ON
  }
  // HIGH = No rain
  else {
    Serial.println("No rain detected. You can OPEN the window.");
    digitalWrite(LED_RED, LOW);    // No rain → LED OFF
  }

  delay(200);
}
```

```
Output  Serial Monitor ×
Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM10')   New Line   115200 baud
No rain detected. You can OPEN the window.
No rain detected. You can OPEN the window.
No rain detected. You can OPEN the window.
No rain detected. You can OPEN the window.
No rain detected. You can OPEN the window.
Rain detected! Please CLOSE the window.
Rain detected! Please CLOSE the window.
Rain detected! Please CLOSE the window.
Rain detected! Please CLOSE the window.
Rain detected! Please CLOSE the window.
Rain detected! Please CLOSE the window.
Rain detected! Please CLOSE the window.
```

Lekcia 10 – Zvukový senzor

Úvod

V tejto lekcii sa naučíme, ako čítať analógové signály na Arduine, a získame základné vedomosti o fungovaní analógových senzorov.

Zvukový senzor dokáže previesť hlasitosť zvuku v prostredí na zmeny napätia a hodnoty v rozmedzí od 0 do 1023 sa načítajú cez analógové piny Arduina.

Ciele výučby

1. Naučte sa používať funkciu analogRead na čítanie analógových signálov zo zvukového senzora. 2. Pochopte pojem prahové hodnotenie (Threshold) a naučte sa vykonávať podmienené rozhodnutia na základe hodnôt zo senzora.

3. Ovládnite funkciu digitalWrite na ovládanie LED diódy, aby sa automaticky zapínala a vypínala podľa zmien v zvuku.

4. Naučte sa používať sériový monitor na výstup údajov v reálnom čase na pozorovanie zmien hodnôt zvukového senzora.

Náhľad výsledku

Po spustení programu:

Sériový port vysiela aktuálnu intenzitu zvuku (v rozmedzí od 0 do 1023) každých 50 milisekúnd.

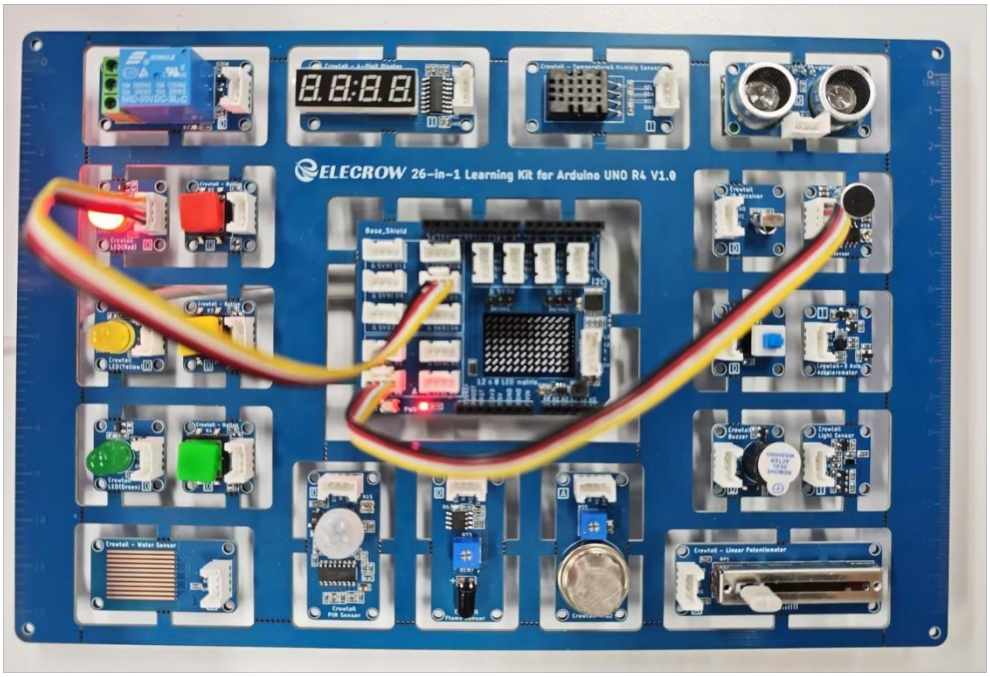
Ak je hlasitosť väčšia ako nastavená prahová hodnota (Threshold = 300):

Správa sériového portu: „>>> Zistený zvuk!“ Červená LED svieti

Červená LED svieti.

Ak je zvuk veľmi tichý a pod prahovou hodnotou: LED sa

automaticky vypne



1. Vysvetlenie princípu

Zvukový senzor je modul, ktorý prevádza hlasitosť okolitého zvuku (amplitúdu zvukových vĺn) na elektrický signál.

Základný princíp:

Základný princíp fungovania zvukového senzora Crowtail spočíva v premene zvukových vĺn z okolia na elektrické signály, ktoré dokáže rozpoznať mikroprocesor, čím umožňuje detekciu a spätnú väzbu o intenzite zvuku. Konkrétny proces prebieha takto:

- Najskôr vstavaný mikrofón zachytí vibrácie zvukových vĺn a premení ich na slabý a rušivý pôvodný elektrický signál;
- Potom vstavaný pásmový filter modulu odfiltruje irelevantné rušenie a zachová efektívne zvukové frekvenčné pásmo, a následne operačný zosilňovač LMV358 zosilní amplitúdu signálu do rozsahu, ktorý mikroprocesor dokáže rozpoznať;
- Zároveň potenciometer zabudovaný v module umožňuje používateľom nastaviť rozsah výstupného napätia, prispôsobiť sa rôznym požiadavkám zariadení a nastaviť citlivosť snímania;
- Výstupom je nakoniec analógový signál, pričom hodnota jeho napätia priamo súvisí s intenzitou zvuku. Mikroprocesor vyhodnocuje tieto údaje o napätí, aby určil intenzitu okolitého zvuku a tým umožnil rôzne ovládacie funkcie spúšťané zvukom.

Medzi hlavné komponenty zvukového senzora Crowtail patria:

Popis pinov modulu zvukového senzora Zvukový

senzor Crowtail má zvyčajne 4 piny:

- **GND (zem):** Záporný pól napájacieho zdroja
- **VCC (kladný pól napájania):** Poskytuje pracovné napätie (5 V) pre modul
- **NC (Not Connected):** Nezúčastňuje sa na obvode
- **SIG (signálny pin):** Vysiela analógový signál (hlasitosť zvuku → zmena napätia)

Hlavným výstupom zvukového senzora je analógový signál. Čím je zvuk hlasnejší, tým je napätie vyššie a hodnota nameraná Arduinom sa zvyšuje.

Nastavte doľava. Čím nižšia je citlivosť rozpoznávania zvukov, tým menšia je pravdepodobnosť, že sa zvuky zachytia;

Nastavte doprava. Čím vyššia je citlivosť rozpoznávania zvukov, tým je možné detekovať aj ten najmenší zvuk.

Analógové piny Arduina (Analog Pins)

Arduino UNO R4 ponúka viacero analógových vstupných rozhraní (A0 – A3).

Simulácia vstupných charakteristík:

- Na načítanie celého čísla v rozsahu od 0 do 1023 použite **analogRead(pin)**
- 0 predstavuje 0 V, 1023 predstavuje referenčné napätie (zvyčajne 5 V)
- Môže čítať signály, ktoré sa neustále menia, ako napríklad zvuk, intenzita svetla, teplota, napätie atď.

V tejto lekcii bude signálny pin zvukového senzora pripojený k **A0**, aby sa mohla čítať intenzita zvuku v reálnom čase.

2. Potrebné moduly

Zvukový senzor Crowtail (1 ks) LED

dióda Crowtail (ľubovoľná farba, 1 ks)

3. Spôsob pripojenia

Zvukový senzor Crowtail → port ANALOG A0

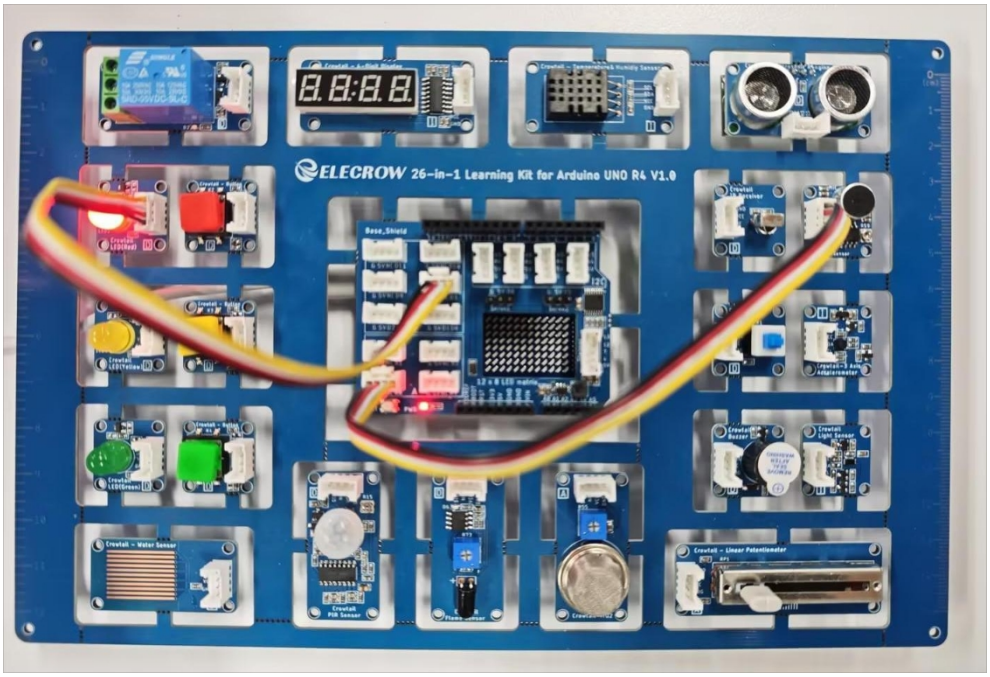
Popis štvorvodičového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červená) → 5V
- NC (biela) → Bez pripojenia
- SIG (žltý) → A0

LED Crowtail → port DIGITAL D3

Popis štvorvodičového rozhrania Crowtail:

SIG (žltá) / NC (biela) / VCC (červená) / GND (čierna)



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-10_SoundSensor/10_SoundSensor

Otvorte program pre túto lekciu pomocou Arduino IDE:

Otvorte program tejto lekcie v priečinku **„10_SoundSensor“** pomocou Arduino IDE.

```ino
#define Sound_Pin  A0
#define Threshold 300   // The sound threshold can be adjusted according to the actual environment.

#define LED_RED 3

void setup() {
  Serial.begin(115200);

  pinMode(LED_RED, OUTPUT);
  // Initialization, turn off the lights
  digitalWrite(LED_RED, LOW);
}

void loop() {
  int val = analogRead(Sound_Pin);  // 0~1023

  Serial.print("Sound Value = ");
  Serial.println(val);

  // Determine whether it exceeds the threshold
  if (val > Threshold) {
    Serial.println(">>> Sound detected! Red LED ON");
    digitalWrite(LED_RED, HIGH);    // Loud sound → Red light on
  } else {
    digitalWrite(LED_RED, LOW);     // Voice becomes quiet → Red light goes out
  }

  delay(50);
}

```

Vysvetlenie kódov klávesov

(1) Definujte piny a prahové hodnoty

```ino
#define Sound_Pin      A0

#define Threshold 300

#define LED_RED 3
```

- Zvukový senzor pripojený k **A0** (analógový vstup)
- Prahová hodnota je nastavená na **300** (možno ju prispôsobiť podľa prostredia)
- Červená LED pripojená k **digitálnemu portu 3**

(2) Inicializačná funkcia setup()

```ino
void setup() { Serial.begin(115200);

   pinMode(LED_RED, OUTPUT);

   // Inicializácia, zhasnutie svetiel

   digitalWrite(LED_RED, LOW);

}
```

- Otvorte sériový port na výstup hodnoty zvuku.
- Nastavte režim **výstupu** LED
- LED je pri spustení programu **vypnutá**.

(3) Hlavná slučka číta hodnoty zvuku.

```
void loop() {
   int val = analogRead(Sound_Pin);          // 0~1023
```

Signál zvukového senzora je analógový signál (analógová veličina), ktorý sa mení v závislosti od hlasitosti zvuku.

Funkcia **analogRead()** má za úlohu:

Previesť úroveň hlasitosti na číselnú hodnotu v rozmedzí **od 0 do 1023**.

- Keď je ticho → Hodnota je malá (v rozmedzí od niekoľkých desiatok do sto)
- Keď je hlasitosť vysoká → Hodnota sa zvýši (o niekoľko stoviek alebo aj viac)

(4) Vytlačiť hodnotu zvuku

```
   Serial.print("Hodnota zvuku = ");
   Serial.println(val);
```

Je to vhodné na sledovanie zmien zvuku v reálnom čase.

(5) Určite, či zvuk prekračuje prahovú hodnotu

```
   // Určite, či prekračuje prah
   if (val > Threshold) {
      Serial.println(">>> Zistený zvuk! Červená LED svieti");
      digitalWrite(LED_RED, HIGH);          // Hlasný zvuk → Červené svetlo
                                            svieti
   } else {
      digitalWrite(LED_RED, LOW);           // Hlas stíchne → Červené svetlo zhasne
   }


   delay(50);
}
```

Hlasný zvuk → LED svieti Tichý zvuk → LED zhasne

Príkaz „if“ v tomto kóde je najzákladnejším „nástrojom na posudzovanie podmienok“. Jeho hlavným účelom je „urobiť jednu vec, ak je podmienka splnená, a urobiť inú vec, ak nie je splnená“:

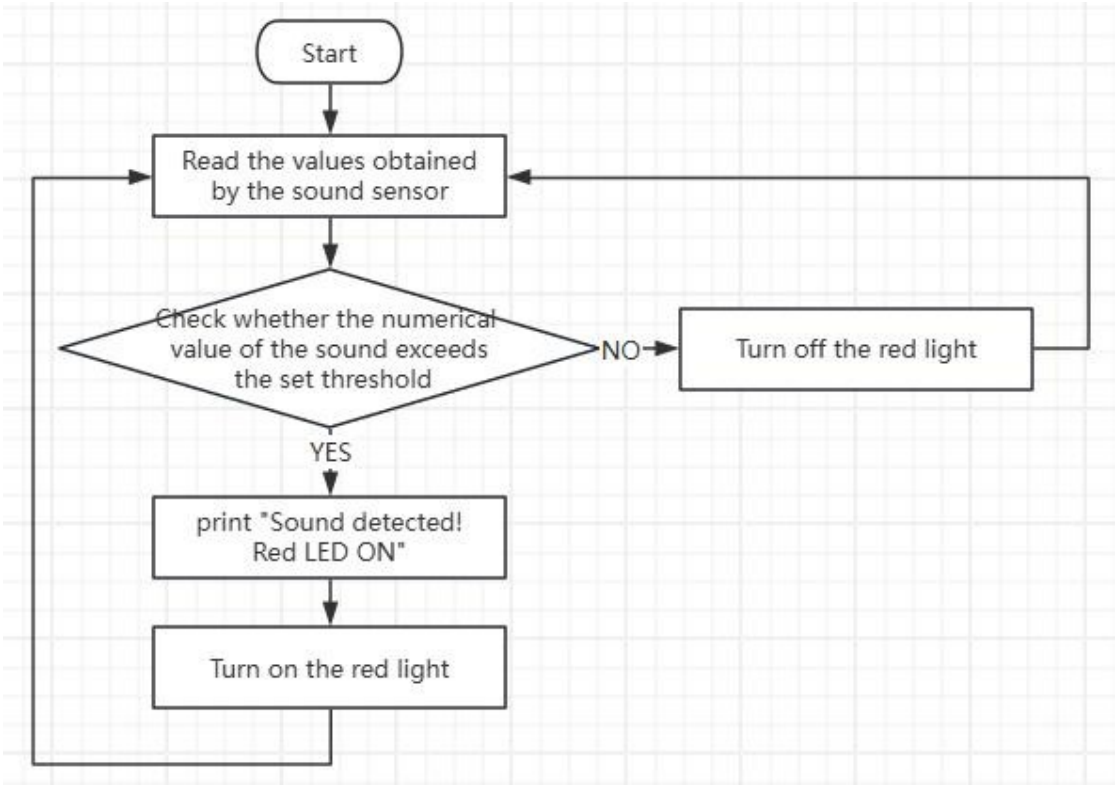
- Najprv prečítajte hodnotu zvukového senzora pomocou **analogRead(Sound_Pin)** (rozsah je 0 až 1023) a potom pomocou **podmienky if (val > Threshold)** zistiť, či je „hodnota zvuku väčšia ako nastavená prahová hodnota 300“ – ak áno (napríklad pri tlieskaní rukami, hlasnom hovorení alebo pri dostatočne

hlasný zvuk), vykonajte kód v zátvorkách: print **"Zistený zvuk! Červené svetlo svieti"**

a zapnite červenú LED **(digitalWrite(LED_RED, HIGH))**;

- ak nie (prostredie je tiché a hodnota zvuku je menšia alebo rovná 300), vykonajte kód za else: vypnite červenú LED **(digitalWrite(LED_RED, LOW)).**
- Zjednodušene povedané, príkaz „if“ funguje ako „rozhodovací spínač“, ktorý na základe splnenia alebo nesplnenia podmienky umožňuje programu vybrať si rôzne cesty.

(6) Logický tok kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov obsluhy Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Stiahnuť“:

Arduino UNO R4 WiFi

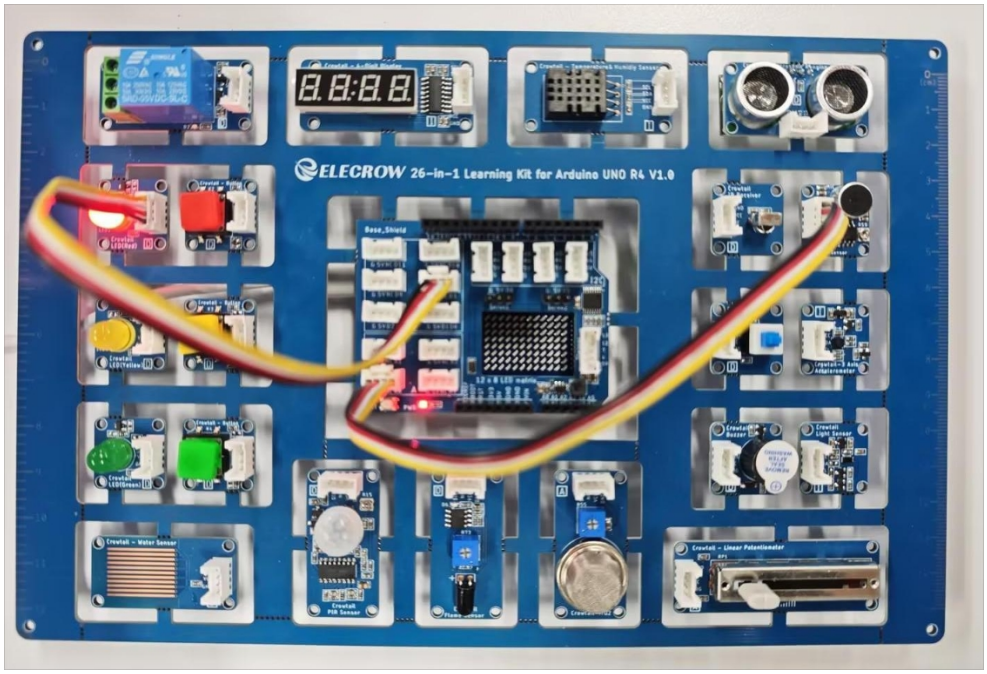
10_SoundSensor.ino

```
#define Sound_Pin  A0
#define Threshold 300   // The sound threshold can be adjusted according to the actual environment.

#define LED_RED 3

void setup() {
  Serial.begin(115200);

  pinMode(LED_RED, OUTPUT);
  // Initialization, turn off the lights
  digitalWrite(LED_RED, LOW);
}

void loop() {
  int val = analogRead(Sound_Pin);  // 0~1023

  Serial.print("Sound Value = ");
  Serial.println(val);

  // Determine whether it exceeds the threshold
  if (val > Threshold) {
    Serial.println(">>> Sound detected! Red LED ON");
    digitalWrite(LED_RED, HIGH);    // Loud sound → Red light on
  } else {
    digitalWrite(LED_RED, LOW);     // Voice becomes quiet → Red light goes out
  }

```

(2) Po úspešnom stiahnutí skontrolujte výsledok spustenia:

Otvorte sériový monitor zabudovaný v prostredí Arduino IDE a nastavte prenosovú rýchlosť podľa špecifikácie v kóde. Týmto spôsobom môžete vidieť informácie vypísané cez sériový port.

(V tomto momente môžete vydávať zvuk v blízkosti zvukového senzora a uvidíte, ako sa rozsvieti červené svetlo.)



Lekcia 11 – Senzor MQ2

Úvod

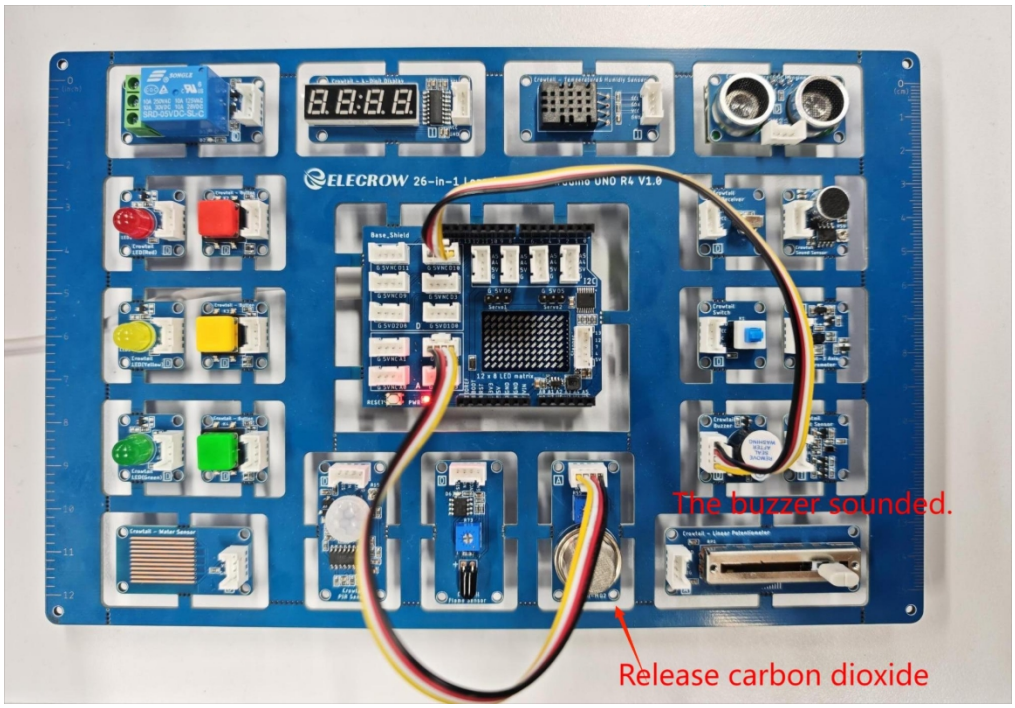
V tejto lekcii použijeme dymový senzor MQ2 a bzučiak na vytvorenie jednoduchého „dymového alarmu“. Zúročíte si zručnosti, ktoré ste sa naučili predtým: čítanie analógových hodnôt senzora, určenie prítomnosti dymu na základe prahovej hodnoty a ovládanie bzučiaka na spustenie alarmu. Zároveň budeme sledovať zmeny údajov v reálnom čase na sériovom porte, aby ste intuitívne pochopili fungovanie systému.

Ciele výučby

1. Porozumieť princípu fungovania detektora dymu.
2. Ďalej si upevnite pojem analogRead a naučte sa čítať analógové hodnoty. To je kľúč k získavaniu analógových veličín a je to aj účelom nášho opakovania.
3. Dokončite prípad „Dymový alarm“

Náhľad výsledku

Keď do detektora dymu vpustíte plyn (napríklad pri horení papiera, odparovaní alkoholu, prítomnosti zemného plynu alebo skvapalneného plynu), ozve sa bzučiak.



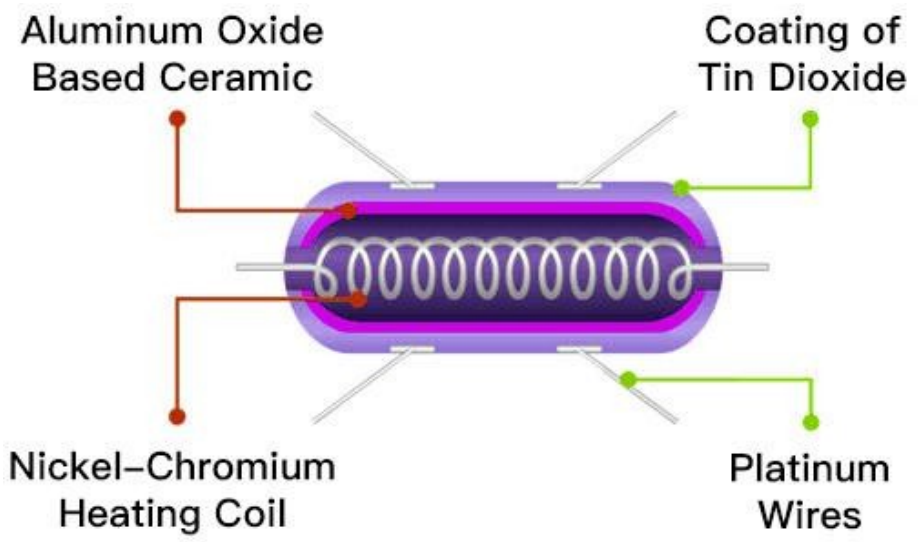
1. Vysvetlenie princípu

Dymový senzor MQ2 je zariadenie, ktoré detekuje dym na základe polovodičového efektu detekcie plynu. Jeho fungovanie možno opísať takto: Jadro senzora tvorí „keramický nosič na báze oxidu hlinitého + povlak z oxidu cínatého“. Do nosiča je vložený

vnútri sa nachádza niklovo-chrómová ohrievacia špirála a signál je vyvedený von cez platinové elektródy na vonkajšej strane. Počas prevádzky je niklovo-chrómová špirála pod napätím, čím sa zahrieva a udržiava

povlak z oxidu cínu na stabilnej teplote 200 až 400 °C – pri tejto teplote povrch

dioxidu cínu adsorbuje molekuly kyslíka zo vzduchu, čím vytvára „adsorpčnú vrstvu záporných iónov kyslíka“, čím sa udržuje merný odpor oxidu cínu na relatívne vysokej základnej úrovni. Keď dym (napríklad horľavé plyny, prchavé častice) príde do kontaktu s povlakom z oxidu cínu, redukčné molekuly v dyme chemicky reagujú s povrchovými zápornými iónmi kyslíka a spotrebujú adsorbované ióny kyslíka. Tento proces zmení koncentráciu nosičov v oxidu cínatom, čím sa zvýši jeho polovodičová vodivosť a výrazne zníži jeho merný odpor. Platínové drôtové elektródy okamžite zachytia túto zmenu odporu a premenia ju na signál, ktorý môže rozpoznať externý obvod (napríklad pokles hodnoty odporu alebo kolísanie napäťového signálu), čo v konečnom dôsledku zodpovedá úrovni koncentrácie dymu prostredníctvom amplitúdy zmeny signálu, čím sa dokončí detekcia dymu.



Arduino určuje úroveň koncentrácie na základe odčítania výstupného napätia **(0~5 V → 0~1023).**

Poznámka: Tu sa nachádza otočný gombík. Jeho otáčaním môžete nastaviť citlivosť detektora dymu pri detekcii dymu.

Ak ho otočíte doľava, citlivosť sa zníži a môžete ho nastaviť do vhodnej polohy, aby sa zabezpečilo, že prah detekcie bude v primeranom rozsahu;

Ak ho otočíte doprava, citlivosť bude vyššia. Niekedy môže prah prekročiť limit aj bez uvoľnenia plynu.

2. Požadované moduly

Senzor Crowtail MQ2 (senzor dymu x1)

Bzučiak Crowtail (bzučiak x1)

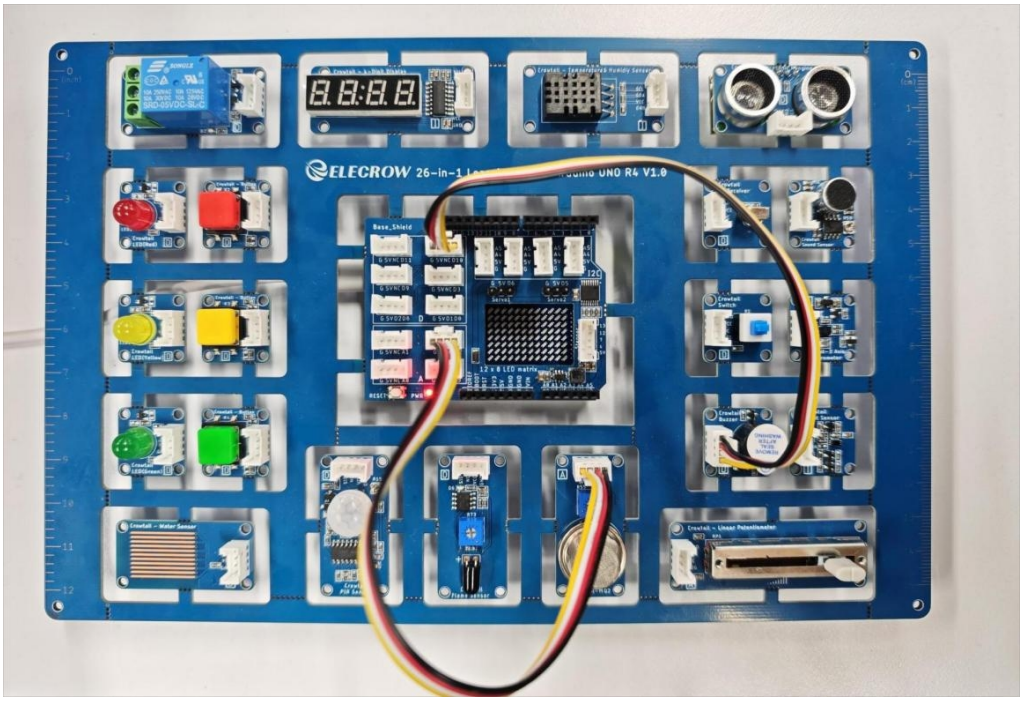
3. Spôsob zapojenia

Senzor dymu MQ2 → A2 Bzučiak

→ D10

Popis štvorvodičového rozhrania Crowtail:

SIG (žltá) / NC (biela) / VCC (červená) / GND (čierna)



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-11_MQ2/11_MQ2

Otvorte program pre túto lekciu v priečinku **„11_MQ2“** pomocou Arduino IDE:

```
// Define hardware pins
#define MQ2Pin A2    // MQ2 smoke sensor connected to analog pin A2
#define BUZ 10       // Buzzer connected to digital pin 10

// Define smoke alarm threshold (adjust according to actual testing)
// Note: MQ2 value is about tens~200 without smoke; the higher the smoke concentration, the high
const int smokeThreshold = 300;

int MQ2Value = 0;  // Store the value read from MQ2 sensor

void setup() {
  // Initialize serial port (for debugging to view sensor values)
  Serial.begin(115200);
  // Initialize pin modes
  pinMode(MQ2Pin, INPUT);   // MQ2 sensor set as input mode
  pinMode(BUZ, OUTPUT);     // Buzzer set as output mode
  // Turn off the buzzer initially
  digitalWrite(BUZ, LOW);
}

void loop() {
  // Read analog value from MQ2 sensor
  MQ2Value = analogRead(MQ2Pin);

  // Print current sensor value to serial port (for debugging and threshold calibration)
  Serial.print("Smoke sensor value: ");
  Serial.println(MQ2Value);

  // Smoke detection logic: trigger alarm if value exceeds threshold, otherwise turn off buzzer
  if (MQ2Value > smokeThreshold) {
    digitalWrite(BUZ, HIGH);  // Turn on buzzer for alarm
    Serial.println("⚠️ Smoke detected! Buzzer alarming...");
  } else {
    digitalWrite(BUZ, LOW);   // Turn off buzzer
    Serial.println("Environment normal, no smoke detected");
  }

  delay(200);  // Detection interval (adjustable as needed, 100-500ms recommended)
}
```

Vysvetlenie kľúčových kódov

(1) #define definuje konštanty pinov

```
#define MQ2Pin      A2

#define BUZ 10
```

- Tieto dva riadky sú definície makier (**#define**), ktoré priraďujú mená fyzickým pinom hardvérových pripojení.
- **MQ2Pin** reprezentuje analógový vstupný **port A2** (pripojený k analógovému výstupu MQ2). Použitie mena je lepšie ako písať A2 priamo do kódu, pretože v budúcnosti bude jednoduchšie zmeniť

  pin v budúcnosti jednoduchou úpravou tejto časti.
- **BUZ** označuje **digitálny port 10** (pripojený k bzučiaku). Rovnaký princíp platí aj pre jednoduchšiu údržbu.

(2) Definujte prahové hodnoty

```
const int smokeThreshold = 300;
```

- Toto je definícia konštantnej prahovej hodnoty. Účelom použitia **„const“** je zabezpečiť, aby táto prahová hodnota zostala počas vykonávania programu nemenná, čím sa zabráni jej náhodne zmenený iným kódom. Tým sa tiež celý program stáva prehľadnejším a čitateľnejším. Samozrejme, ak odstránite **„const“**, kód neoznámi chybu.
- Význam nastavenia **„smokeThreshold = 300“** je nasledovný: Ak je nameraná hodnota senzora MQ2 vyššia ako 300, považujeme situáciu za „prítomnosť dymu“ a je potrebné spustiť alarm.

Poznámka: Táto hodnota 300 by sa mala počas skutočného používania prispôsobiť na základe vášho senzora, prostredia, vzdialenosti a podmienok vetrania prostredníctvom experimentov. Hodnota MQ2 v čistom vzduchu môže byť medzi 40 a 200 (rôzne senzory sa veľmi líšia) a keď je prítomný dym, stúpne na niekoľko stoviek alebo dokonca niekoľko stoviek až niekoľko tisíc (v závislosti od referenčného napätia MCU a pripojenia).

(3) Definujte premenné

```
int MQ2Value = 0;
```

Tu je deklarovaná celočíselná premenná MQ2Value, ktorá sa používa na ukladanie hodnoty načítanej každou funkciou **analogRead()**. Je pôvodne nastavená na 0 a nemá žiadny konkrétny význam; ide iba o osvedčený postup.

(4) Sekcia nastavenia

```
void setup() {
  Serial.begin(115200);
  pinMode(MQ2Pin, INPUT);
  pinMode(BUZ, OUTPUT);
  digitalWrite(BUZ, LOW);
}
```

Toto je funkcia **setup()**, ktorá sa spustí len raz pri zapnutí alebo resete Arduina, aby vykonala rôzne inicializácie.

- Ser**ial.begin(115200);:** Aktivuje sériovú komunikáciu s prenosovou rýchlosťou **115200**. To vám umožňuje zobraziť výstup programu na sériovom monitore vo vašom počítači (na účely ladenia a kalibrácie hodnôt). Rýchlosť sériovej komunikácie musí byť zhodná s rýchlosťou sériového monitora; inak budú údaje skreslené.
- **pinMode(MQ2Pin, INPUT);:** Nastaví pin pripojený k MQ2 ako vstup. Pre analógový vstup, Prísne vzaté, **funkcia analogRead(A2)** nevyžaduje použitie pinMode, ale jej zaradenie zvyšuje sémantickú zrozumiteľnosť kódu (naznačuje, že tento pin slúži na čítanie signálov).

- **pinMode(BUZ, OUTPUT);:** Nastavte pin bzučiaka ako výstup, aby ste mohli bzučiak ovládať pomocou **digitalWrite()** a zapínať a vypínať ho.
- **digitalWrite(BUZ, LOW);:** Program vypne bzučiak hneď na začiatku (na nízku úroveň), aby sa zabránilo náhodnému pípaniu pri zapnutí. Ide o bezpečnostné opatrenie a zohľadnenie užívateľského zážitku

.

(5) Slučka

① **Funkcia loop() je** hlavná slučka, ktorá sa opakuje. Prvým krokom je **analógová funkcia analogRead(MQ2Pin),** ktorá načíta celé číslo v rozsahu **od 0 do 1023** z **A2** (10-bitový ADC Arduina).

```
void loop() {
  // Načítanie analógovej hodnoty zo senzora MQ2
  MQ2Value = analogRead(MQ2Pin);
```

Priraďte výsledok čítania k **MQ2Value**. Toto číslo predstavuje napätie na výstupe modulu MQ2 v pomere k referenčnému napätiu MCU. Čím väčšia je hodnota, tým vyššia je koncentrácia zisteného horľavého plynu/dymu (konkrétna krivka je určená senzorom a obvodom modulu).

② **Vytlačte získanú hodnotu**

```
  Serial.print("Hodnota snímača dymu: ");
  Serial.println(MQ2Value);
```

Tieto dva riadky vytlačia aktuálne namerané hodnoty na monitor sériového portu. **Funkcia Serial.print()** vypíše text bez nového riadka, zatiaľ čo **funkcia Serial.println()** vypíše číslo a pridá nový riadok.
Účelom je pomôcť vám sledovať hodnoty v reálnom čase, čím sa ľahšie určí, aké sú hodnoty v podmienkach „normálneho vzduchu“ a „s dymom“, aby sa **mohla kalibrovať** prahová hodnota smokeThreshold. To je veľmi dôležité počas fáz vývoja a **ladenia**; po tom, čo systém beží stabilne, môžete zvážiť zníženie frekvencie výstupu, aby ste ušetrili výkon.

③ **Logika posudzovania**

```
  if (MQ2Value > smokeThreshold) {
    digitalWrite(BUZ, HIGH);        // Zapnúť bzučiak pre alarm
    Serial.println(" Zistený dym! Bzučiak vydáva alarm...");
  } else {
    digitalWrite(BUZ, LOW);          // Vypnúť bzučiak
    Serial.println("Normálne prostredie, nedetekovaný dym");
  }
  delay(200);       // Interval detekcie (nastaviteľný podľa potreby, odporúča sa 100–500 ms)
```

```
}
```

Toto je základná logika pre vyhodnotenie havárie a riadenie akcií.

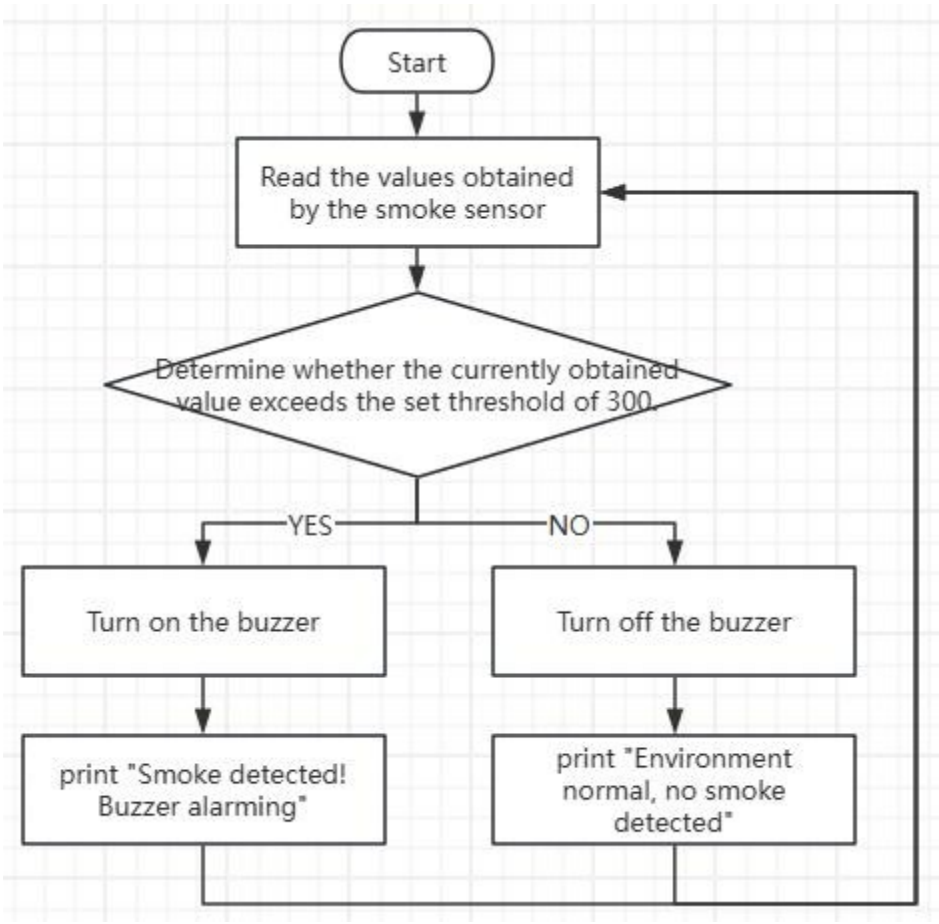
If (MQ2Value > smokeThreshold) kontroluje, či aktuálna hodnota prekračuje prednastavenú prahovú hodnotu.

- Ak áno (hodnota je vyššia ako 300), vykonaj kód v zátvorkách { ... }: **digitalWrite(BUZ, HIGH);** nastavte bzučiak na HIGH (zapnutý), čím vydá zvuk. Potom vytlačte varovnú správu na sériový port.
- Ak nie (hodnota je menšia alebo rovná prahovej hodnote), prejdite do vetvy else: vypnite **bzučiak (LOW)** a vytlačte **„Environment normal**".

Logika je tu jednoduchá: varovanie pri prekročení prahovej hodnoty sa spustí, keď je prahová hodnota prekročená, a keď klesne pod túto hodnotu, varovanie sa vypne.

- **delay(200);** spôsobí, že MCU sa na 200 milisekúnd pozastaví pred vstupom do ďalšej slučky. Toto má **dve funkcie**: **jednou** je zníženie rýchlosti tlače sériového portu, aby sa zabránilo blikaniu informácií; **druhou** je regulácia frekvencie čítania, aby sa zabránilo príliš častému čítaniu a spustenie. **200 ms** je bežný kompromis (rozsah 100 – 500 ms); ak je vysoká rýchlosť odozvy V prípade potreby je možné znížiť oneskorenie, čo však zvýši zaťaženie procesora a sériového portu; ak je spotreba energie z batérie dôležitým faktorom, je možné oneskorenie zvýšiť.

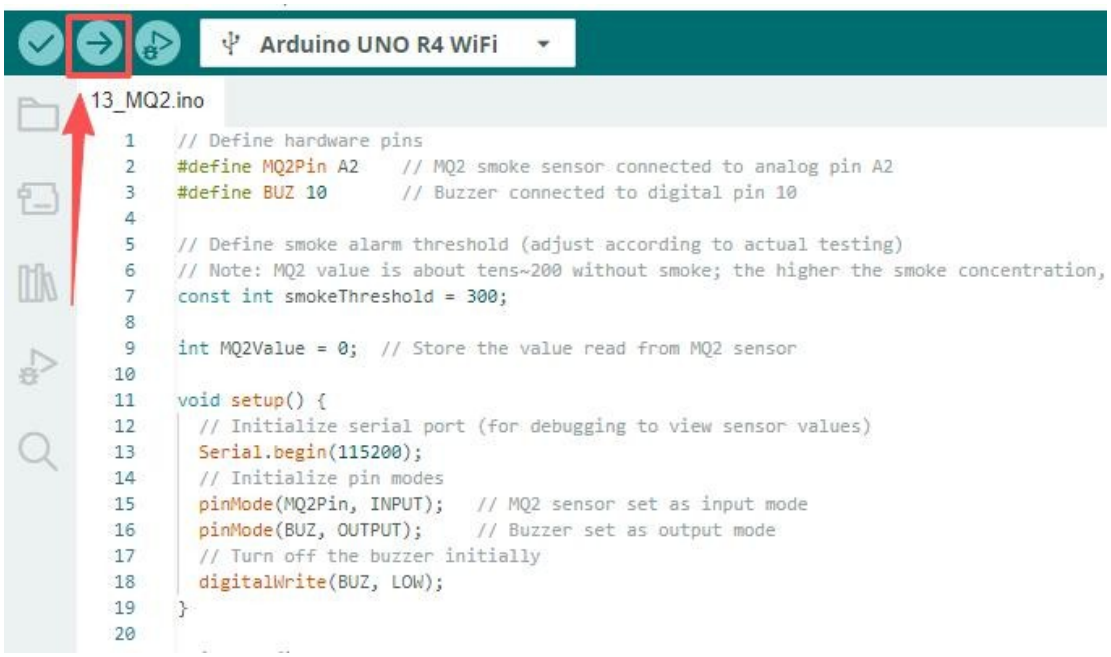
(6) Logický tok kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov obsluhy Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“**:



(2) Po úspešnom stiahnutí skontrolujte výsledok:

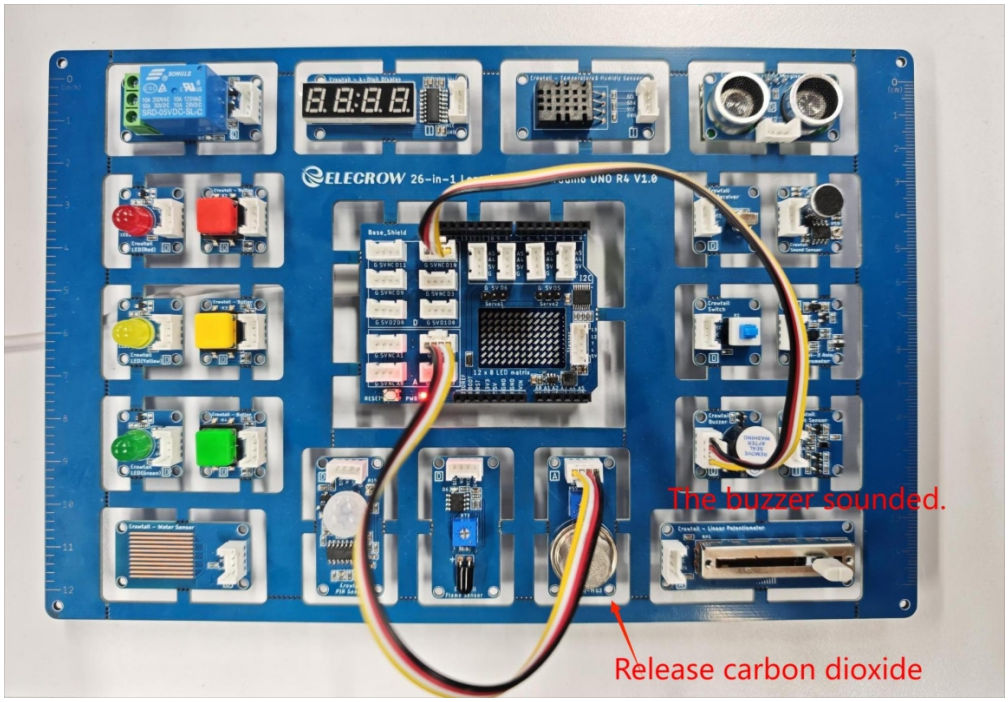
Otvorte sériový monitor zabudovaný v prostredí Arduino IDE

(3) Priložte zapaľovač, cigaretu alebo iný zdroj dymu k senzoru a uvidíte:

- Hodnota rýchlo stúpa (prekročí 300)
- Sériový port vytlačí výstražnú správu
- Zaznie bzučiak

Keď sa dym rozptýli a hodnota klesne:

- Bzučiak prestane vydávať výstražný signál
- Sériový port zobrazí hlásenie „Normálne prostredie“



Lekcia 12 – Snímač plameňa

Úvod

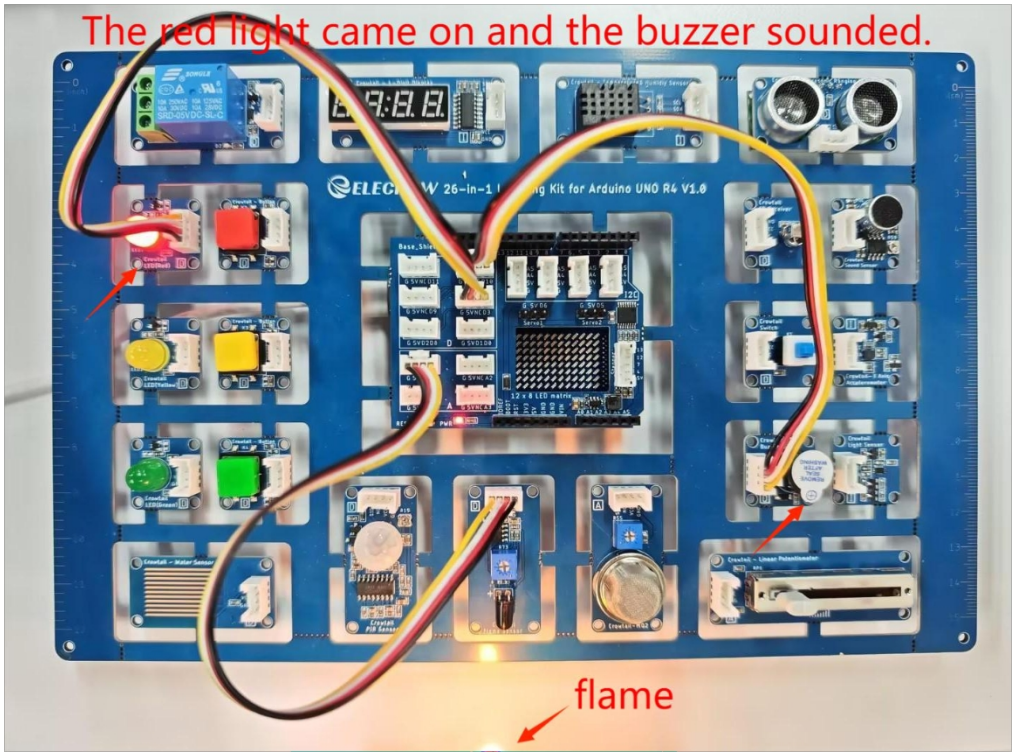
V tejto lekcii použijeme senzor plameňa, LED diódu a bzučiak na zostavenie jednoduchého „detektora plameňa“. Zjednotíte si základné postupy, ktoré ste sa naučili v predchádzajúcich lekciách, a to načítaním analógovej hodnoty zo senzora plameňa a nastavením podmienok na vyhodnotenie, čím dosiahnete detekciu plameňa a spustenie výstražného signálu.

Ciele výučby

1. Porozumieť princípu fungovania senzora plameňa.
2. Zopakovať si použitie funkcie analogRead na čítanie analógových veličín a upevniť si použitie funkcie digitalWrite na výstup vysokého a nízkeho napätia.
3. Dokončiť projekt „Plamenový alarm“

Náhľad výsledku

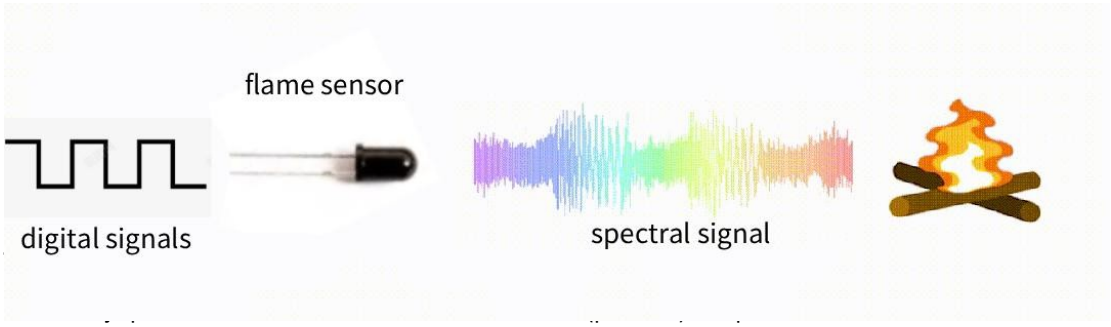
Keď sa plameň priblíži, LED dióda sa rozsvieti a ozve sa pípnutie; v opačnom prípade LED dióda zhasne a pípnutie sa vypne.



1. Vysvetlenie princípu

Najprv sonda (čierna súčasť na obrázku) zachytáva blízke infračervené svetlo a iné spektrálne signály uvoľňované počas horenia plameňa (odpovedajúce plameňu a

krivky farebného spektra na pravej strane obrázku). Následne vnútorný obvod senzora prevádza tento svetelný signál na elektrický signál vo forme napätia, ktorý sa po zosilnení a spracovaní odosiela (a nakoniec sa prevádza na krivku digitálneho signálu na ľavej strane obrázku); odpovedajúca logika medzi signálom a stavom plameňa je nasledovná: keď je plameň blízko, spektrálny signál prijatý sondou je silnejší a hodnota analógového napätia na výstupe senzora klesne (zvyčajne <100); naopak, keď nie je detekovaný žiadny plameň, spektrálny signál je slabý a hodnota analógového napätia na výstupe bude vyššia (zvyčajne > 100). Regulátor môže na základe tejto numerickej zmeny dosiahnuť detekciu plameňa a spustenie alarmu.



- Ak nie je zaznamenaný plameň → Simulovaná hodnota je vysoká (nad 100)

Analógové piny Arduina (Analog Pins)

Arduino UNO R4 ponúka viacero analógových vstupných rozhraní (A0 – A3).

V tejto lekcii budú signálne piny snímača plameňa pripojené k A1 pre čítanie intenzity okolitého plameňa v reálnom čase.

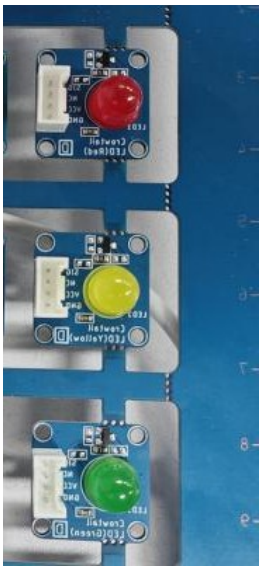
Vlastnosti analógového vstupu:

- Použite analogRead(pin) na čítanie celého čísla v rozsahu od 0 do 1023
- 0 predstavuje 0 V, 1023 predstavuje referenčné napätie (zvyčajne 5 V)
- Dokáže čítať signály, ktoré sa neustále menia, ako napríklad zvuk, intenzita svetla, teplota, napätie atď.

2. Potrebné moduly

Senzor plameňa Crowtail (×1)

LED dioda Crowtail (ľubovoľná farba ×1, v tomto kurze sa používa červená)



Bzučiak Crowtail (×1)

3. Spôsob zapojenia

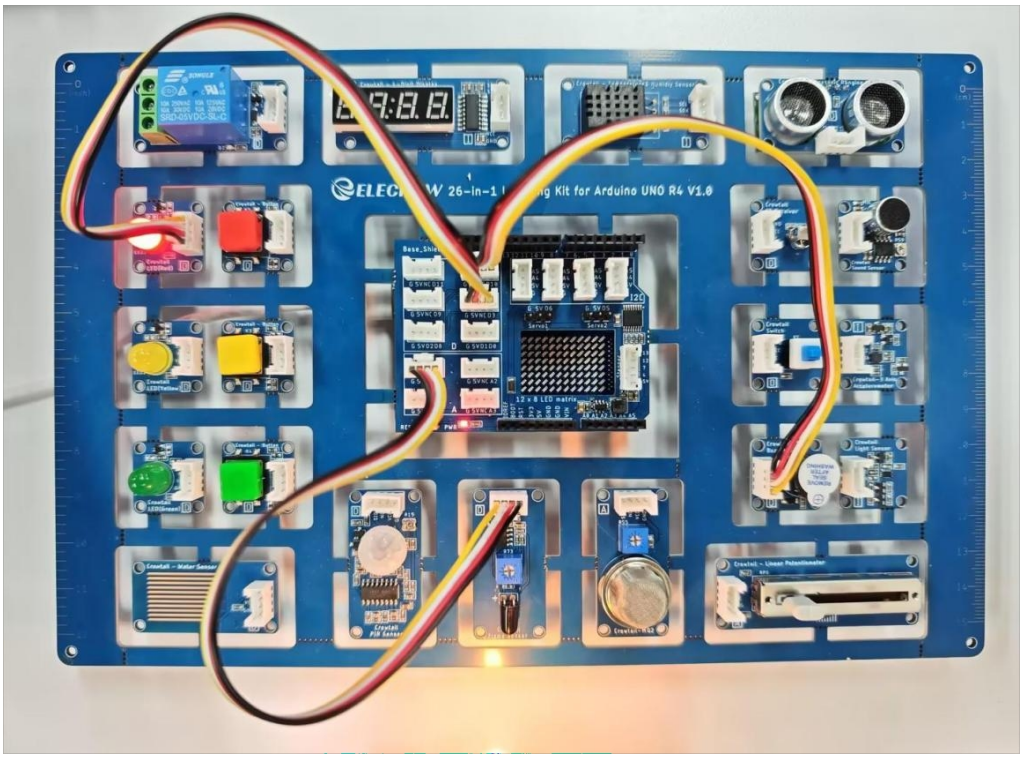
Snímač plameňa → analógový port A1

LED dióda Krautler → digitálny port D3

Bzučiak Krautler → digitálny port D10

Popis štvorvodičového rozhrania Krautler:

SIG (žltá) / Bez pripojenia (biela) / VCC (červená) / GND (čierna)



4. Vysvetlenie príkladu

Kliknite na odkaz a stiahnite si oficiálny príklad kódu:

Odkaz na GitHub：

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-12_FlameSensor/12_FlameSensor

Otvorte program pre túto lekciu v priečinku **„12_FlameSensor“** pomocou Arduino IDE:

```
#define Flame A1      // Flame sensor analog pin
#define LED_RED 3     // LED pin
#define BUZ 10        // Buzzer pin

int sensorValue = 0;

void setup() {
  Serial.begin(115200);

  pinMode(Flame, INPUT);   // Flame sensor input
  pinMode(LED_RED, OUTPUT);
  pinMode(BUZ, OUTPUT);

  digitalWrite(LED_RED, LOW);  // LED off at start
  digitalWrite(BUZ, LOW);      // Buzzer off at start
}

void loop() {
  sensorValue = analogRead(Flame);  // Read flame sensor

  // Flame detected (value < 100)
  if (sensorValue < 100) {
    Serial.println("--- Flame detected! Warning!");

    digitalWrite(LED_RED, HIGH); // LED ON
    digitalWrite(BUZ, HIGH);     // Buzzer ON
  }
  else {
    Serial.println("Safe. No flame detected.");

    digitalWrite(LED_RED, LOW); // LED OFF
    digitalWrite(BUZ, LOW);     // Buzzer OFF
  }

  delay(100);
}
```

Vysvetlenie kľúčového kódu

(1) #define

#define Flame A1 #define LED_RED 3 #define BUZ 10

Pomocou **#define** pomenujte piny podľa názvov modulov. Keď potom uvidíte toto pomenovanie, budete vedieť, že rozhranie **A1** je pripojené k senzoru plameňa, čo nám uľahčí neskoršie písanie a úpravy kódu.

- **Flame** predstavuje **A1** (analógový vstupný pin používaný na pripojenie senzora plameňa)
- **LED_RED** predstavuje **pin 3** (pripojený k červenej LED)
- **BUZ** predstavuje **pin 10** (pripojený k bzučiaku)

(2) Definície premenných

```
int sensorValue = 0;
```

- **Premenná sensorValue** slúži na ukladanie hodnoty snímača plameňa, ktorú zakaždým zčíta funkcia analogRead. Je typu int (celé číslo). Snímač plameňa vydáva analógovú hodnotu **(0 až 1023)** a táto hodnota sa tu ukladá, aby sa neskôr mohla použiť na určenie, „či je prítomný plameň“.
- Je inicializovaná na 0, čo je len počiatočná hodnota. Po spustení programu bude prepísaná skutočnou nameranou hodnotou.

(3) Inicializácia setup()

```
void setup() {
  Serial.begin(115200);
```

- Spustite sériovú komunikáciu s prenosovou rýchlosťou **115200**. Týmto spôsobom môže sériový monitor v počítači zobrazovať text odoslaný Arduinom.
- **115200** je bežne používaná prenosová rýchlosť, ktorá zabezpečuje rýchly prenos a je kompatibilná s väčšinou nástrojov pre sériové porty.

(4) Nastavte pracovné režimy každého pinu

```
  pinMode(Flame, INPUT);          // Vstup senzora
  plameňa pinMode(LED_RED, OUTPUT);
  pinMode(BUZ, OUTPUT);
```

- Nastavte **Flame (A1)** do režimu vstupu. V Arduine môžete čítať analógový vstup jednoduchým volaním **analogRead(A1)** a zvyčajne nie je potrebné explicitne nastavovať pinMode. Pridanie tohto riadku však nespôsobí žiadne chyby. Jasne vyjadruje zámer „Toto je vstupný pin“ a pomáha zlepšiť čitateľnosť kódu a demonštračné účely pri výučbe. Ak modul sám o sebe nemá interný pull-up alebo pull-down rezistor, môžete použiť aj **INPUT_PULLUP** a urobiť príslušné logické inverzné posúdenie. Pre väčšinu modulov snímača plameňa však stačí použiť **INPUT**.
- **pinMode(LED_RED, OUTPUT);** a **pinMode(BUZ, OUTPUT);:** Nastavte piny pripojené k Prepnite LED a bzučiak do režimu výstupu, pretože Arduino potrebuje z týchto pinov vysielať signály s vysokou alebo nízkou úrovňou na ovládanie zariadení.

(5) Vypnite LED a bzučiak pri spustení

```
digitalWrite(LED_RED, LOW);     // LED vypnutá pri štarte
digitalWrite(BUZ, LOW);         // Vypnutie bzučiaka pri spustení
```

Na začiatku programu sú LED a bzučiak vypnuté. LOW označuje výstup nízkeho napätia alebo 0 V na pin, čo je ekvivalentné vypnutiu.

Týmto spôsobom sa dá zabrániť tomu, aby bzučiak nepretržite pípal alebo LED blikala v dôsledku nesprávneho spustenia pri zapnutí, čím sa predíde falošným poplachom alebo vystrašeniu ľudí.

(6) loop() —— Neustále opakuje vykonávanie

① **Načítanie analógovej hodnoty z senzora plameňa (0 – 1023)**

```
void loop() {
  sensorValue = analogRead(Flame);        // Načítanie snímača plameňa
```

- Kód vo vnútri funkcie loop() sa bude neustále opakovať. Prvým krokom je **analogRead(Flame):** prečíta analógové napätie na pine A1 a výsledok uloží do premennej sensorValue.
- Vrátená hodnota ADC je celé číslo v rozmedzí **od 0 do 1023**, ktoré reprezentuje polohu vstupného napätia medzi 0 V a referenčným napätím (zvyčajne **5 V alebo 3,3 V,** v závislosti od dosky).
- V prípade modulu snímača plameňa, keď sa v blízkosti nachádza plameň, napätie na výstupe hodnota snímača sa zvyčajne znižuje, takže vrátená hodnota je tiež nižšia; ak nie je prítomný plameň, výstupné napätie je vyššie a hodnota je väčšia.

② **Zistiť, či horí**

```
  if (hodnotaSenzora < 100) {
```

if (hodnotaSenzora < 100): Ak je nameraná hodnota menšia ako 100, považuje sa to za prítomnosť plameňa. 100 je empirická hodnota (prahová hodnota) a nie je to pevný štandard. Rôzne senzory, rôzne polohy, rôzne svetelné a environmentálne podmienky môžu spôsobiť, že pôvodná hodnota bude odlišná, preto je potrebné otestovať vaše vlastné zariadenie, aby ste určili vhodnú prahovú hodnotu.

Zvyčajne najprv zistíme základnú hodnotu **sensorValue**, keď nie je prítomný plameň (môže sa pohybovať **v rozmedzí 600 až 1000**), potom zistíme hodnotu, keď sa priblíži malý plameň (môže klesnúť na niekoľko desiatok alebo ešte nižšie), a následne zvolíme prahovú hodnotu, ktorá dokáže spoľahlivo rozlíšiť tieto dva stavy, napríklad 100.

③ **Ak je prítomný plameň, vykonajte nasledujúci kód**

```
    Serial.println("--- Zistený plameň! Varovanie!");
```

```
    digitalWrite(LED_RED, HIGH); // LED
    zapnutá
    digitalWrite(BUZ, HIGH);          // Zapnutý
                                      bzučiak
}
```

- **digitalWrite(LED_RED, HIGH);:** Nastaví pin LED na vysokú úroveň (zapne LED). Poznámka: V skutočnom obvode je zvyčajne potrebné LED diodu zapojiť do série s odporom obmedzujúcim prúd (napr. $220\ \Omega$), aby sa zabránilo jej spáleniu. Ak používate modul s integrovaným odporom (napr. Crowtail, ktorý sa zvyčajne dodáva s odporom), môžete ho jednoducho pripojiť ho priamo; ak ide o holú LED, je potrebné pridať odpor.
- **digitalWrite(BUZ, HIGH);:** Poskytne bzučiaku vysokú úroveň, čím ho rozoznie. Tu je potrebné rozlíšiť typ bzučiaka: aktívny bzučiak (active) bude znieť, pokiaľ je mu poskytovaná vysoká úroveň; pasívny bzučiak (passive) vyžaduje frekvenčné riadenie (pomocou PWM alebo tone()) na vytvorenie špecifickej výšky tónu.
- **Rozšírenie:** Ak pripájate pasívny bzučiak, priame použitie **digitalWrite(HIGH)** nemusí vytvoriť skutočný **„alarmový zvuk**“. Navyše, ak má bzučiak vysoký odber prúdu, jeho priame pripojenie k vývodu Arduina môže prekročiť kapacitu daného vývodu. V takomto prípade by sa na riadenie mal použiť tranzistor alebo MOSFET a na ochranu dióda.

④ **Ak nie je prítomný plameň**

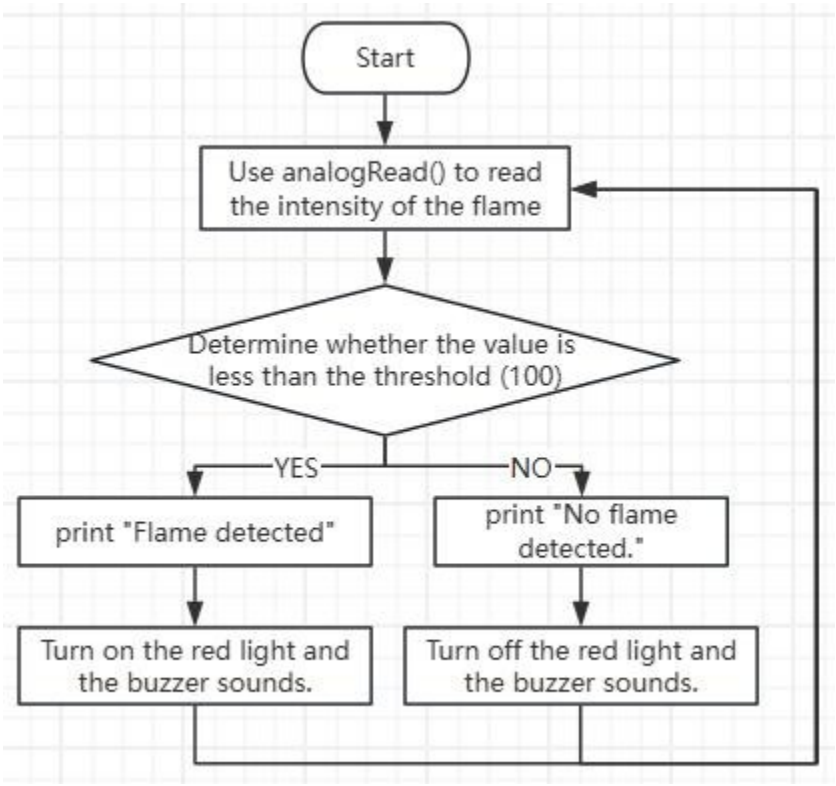
```
  else {
    Serial.println("Bezpečné. Nebol zistený
    plameň."); digitalWrite(LED_RED, LOW); //
    LED vypnutá
    digitalWrite(BUZ, LOW);           // Bzučiak
                                      vypnutý
  }
  delay(100);
}
```

Poznámka:

- Sériový port vypíše „Safe“
- LED je vypnutá
- Pípacie zariadenie je vypnuté

vetva „else“: Ak sa nezistí žiadny plameň, vypíše sa **hlásenie „Bezpečné. Plameň nezistený.“** a vypnú sa LED aj bzučiak.

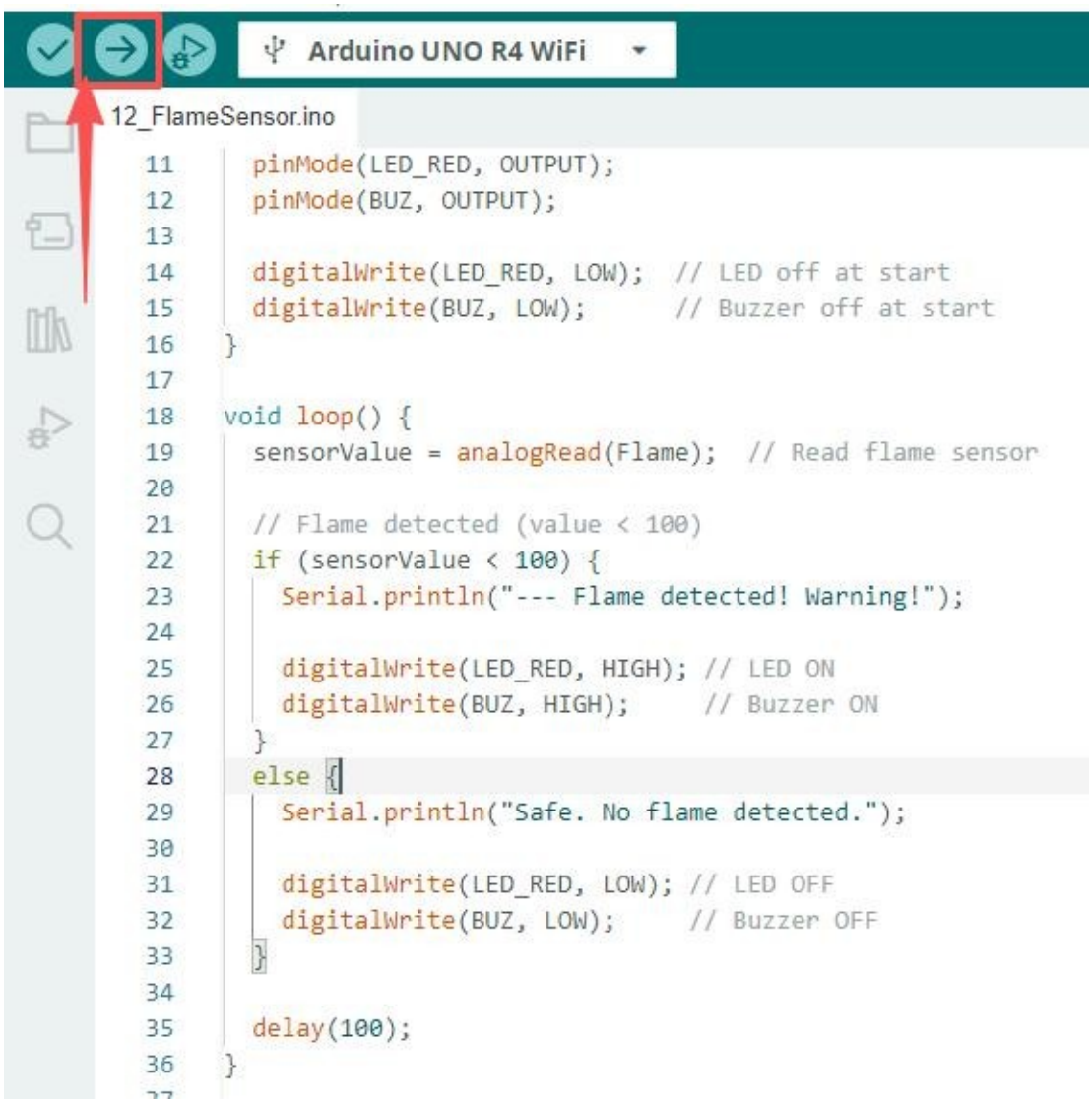
(7) Logický tok kódu



5. Spustite program a sledujte výsledky

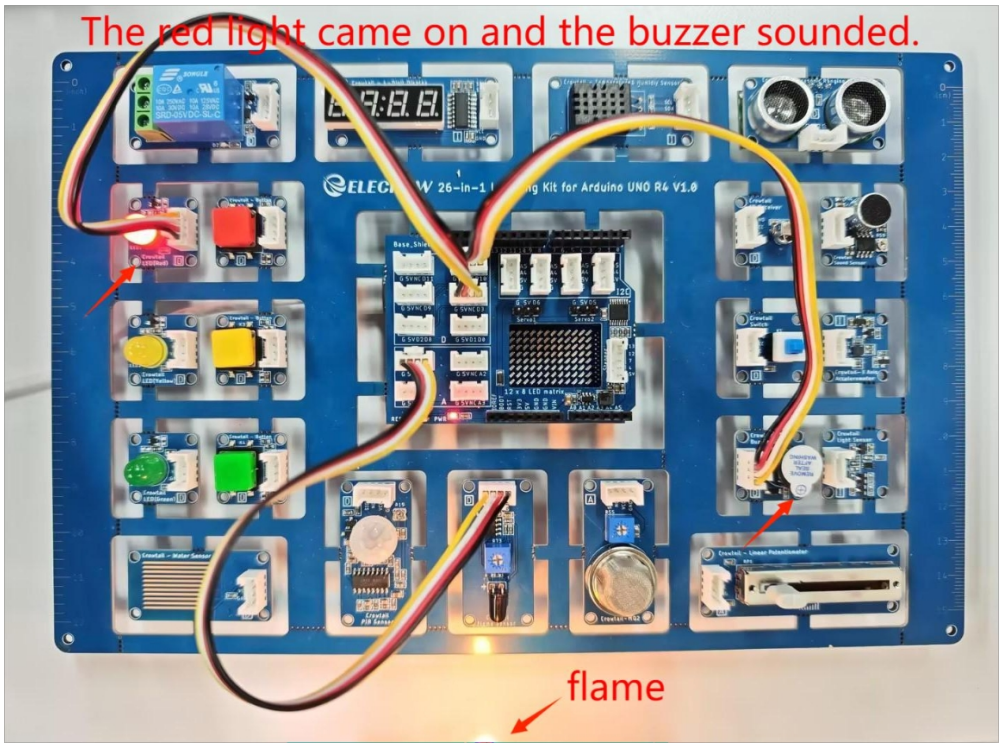
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“:**



(2) Po úspešnom stiahnutí skontrolujte funkčnosť:

Keď sa plameň priblíži, LED dióda sa rozsvieti a bzučiak vydá alarm; v opačnom prípade LED dióda zhasne a bzučiak sa vypne.


The red light came on and the buzzer sounded.
flame

Lekcia 13 — Svetelný senzor

Úvod

V tejto lekcii sa zoznámime s veľmi bežným vstupným modulom na snímanie prostredia — svetelným senzorom. Svetelný senzor detekuje zmeny v úrovni okolitého osvetlenia a prevádza tieto informácie na dáta, ktoré Arduino dokáže prečítať.

V tejto lekcii sa naučíte, ako nechať Arduino snímať jas okolia a automaticky ovládať výstupné zariadenia (napríklad LED diódy alebo relé) na základe intenzity svetla. To vám umožní

vytvoriť klasickú aplikáciu inteligentného ovládania – automatické osvetlenie.
Táto lekcia tvorí dôležitý základ pre neskoršie témy, ako sú inteligentné pouličné osvetlenie, energeticky úsporné osvetlenie a systémy inteligentnej domácej automatizácie.

Ciele výučby

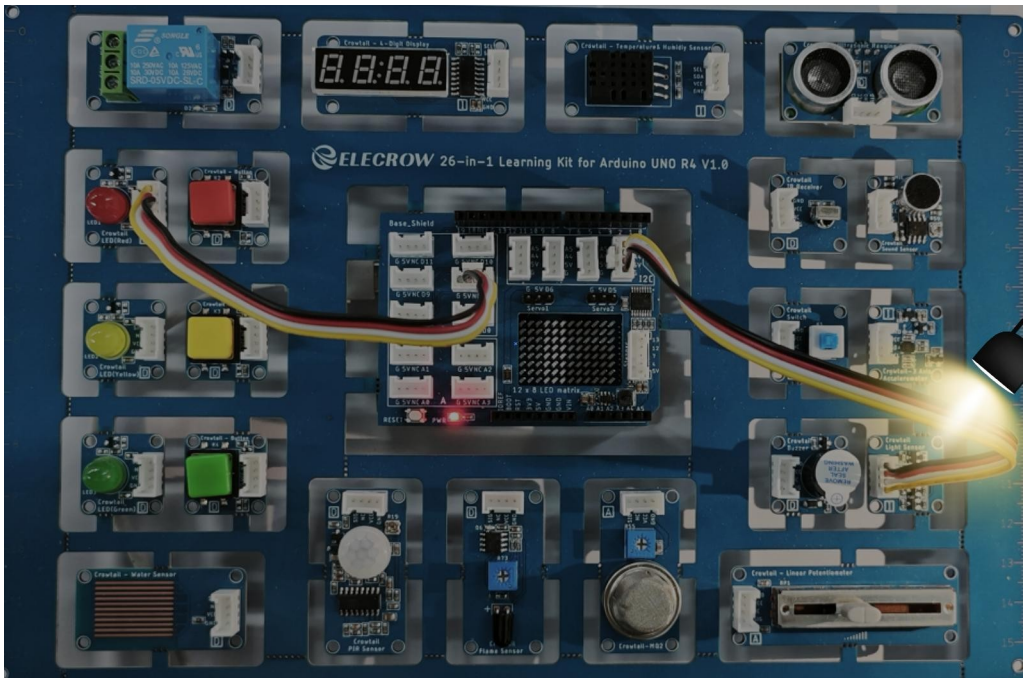
1. Porozumieť funkcii a metódam výstupu svetelného senzora
2. Naučiť sa pridávať a používať súbory knižníc
3. Naučiť sa čítať údaje o intenzite osvetlenia (v luxoch) 4.

Pochopiť koncept a použitie komunikácie I2C

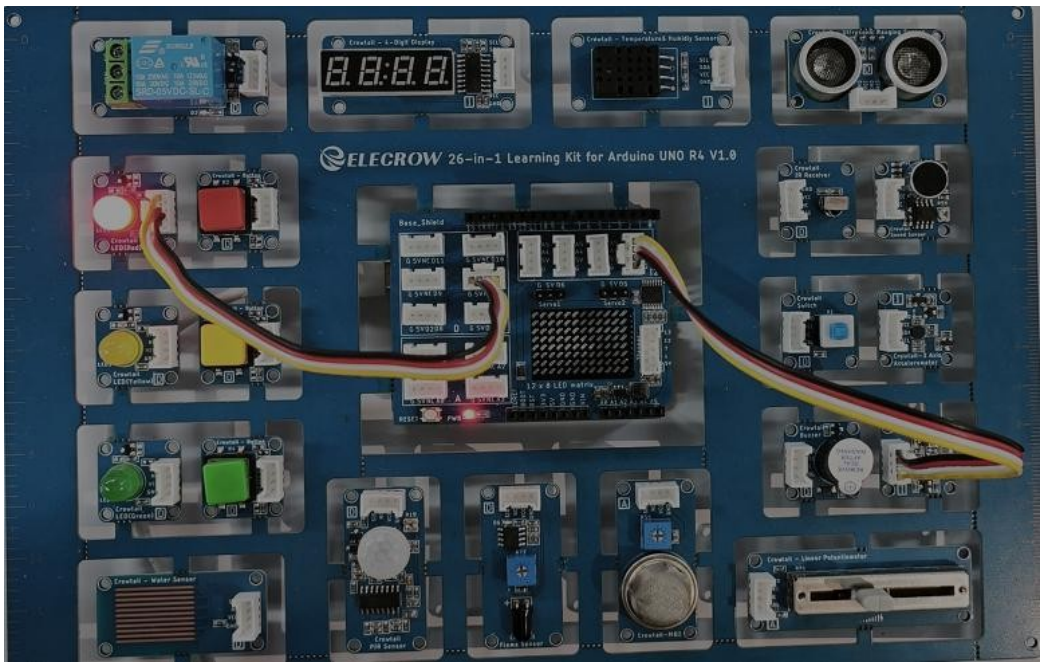
5. Vytvoriť kompletný demonštračný projekt automatického ovládania LED osvetlenia

Náhľad výsledku

Keď je úroveň okolitého osvetlenia dostatočná (hodnota jasu je vyššia ako 300 luxov), LED sa vypne.



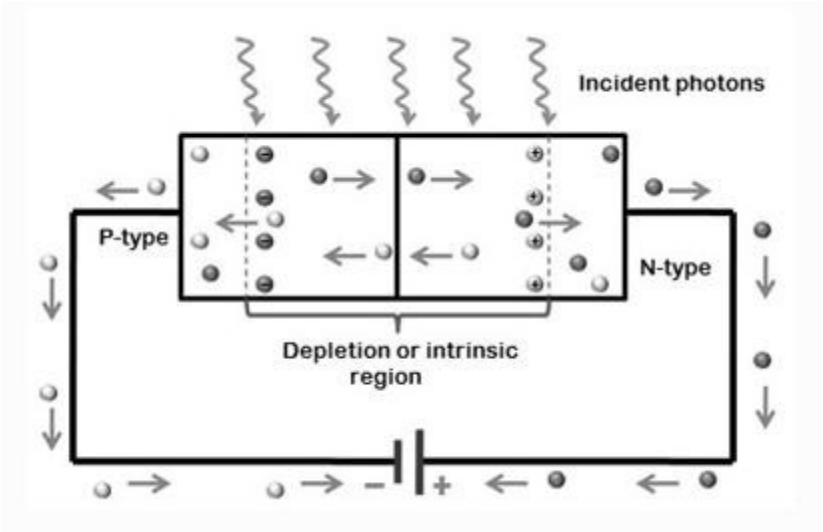
Keď je úroveň okolitého osvetlenia nízka (hodnota jasu je nižšia ako 300 luxov), LED dióda sa rozsvieti.



1. Vysvetlenie princípu

①Ako funguje digitálny svetelný senzor

Digitálny svetelný senzor (BH1750) používa fotodiódu na meranie okolitého osvetlenia. Fotodióda absorbuje svetelnú energiu a generuje prúd, ktorý je úmerný intenzite svetla. Namerať osvetlenie sa vyjadruje v luxoch (LUX).



Princíp fotodiódy

Fotodióda je dióda s PN prechodom. Na rozhraní medzi materiálmi typu P a typu N sa nachádza oblasť nazývaná zóna vyčerpania.

Svetlo prenáša energiu v malých balíčkoch nazývaných fotóny. Keď fotóny s dostatočnou energiou narazia na zónu vyprázdnenia, prerušia kovalentné väzby a vytvoria pár elektrón-diera – elektrón (čierna bodka na diagramoch) a diera (biela bodka na diagramoch).

Elektróny prúdia vodičom smerom k zdroju napätia a materiálu typu P.

Diery sa nemôžu pohybovať cez vodič, takže sa rekombinujú s prichádzajúcimi elektrónmi v materiáli typu P.

Tento proces generuje fotoprúd, ktorý sa prevádza na napätie, následne digitalizuje a nakoniec sa vyhodnocuje ako osvetlenie v luxoch.

Úvod do I2C

Digitálny svetelný senzor BH1750 využíva komunikáciu I2C.

Čo je I2C?

I2C (Inter-Integrated Circuit) je bežne používaný protokol sériovej komunikácie pre mikrokontroléry (ako Arduino), senzory, displeje, pamäťové čipy a rozširujúce moduly. Jeho kľúčovou vlastnosťou je, že viacero zariadení môže zdieľať len dve signálne linky.

Signálne linky I2C

Signál	Názov	Funkcia
SDA	Sériové dáta	Dátová linka pre prenos
SCL	Sériový takt	Hodiny na synchronizáciu komunikácie

Okrem toho každý modul vyžaduje VCC (napájanie) a GND (uzemnenie), čím sa typické zariadenie I2C skladá zo štyroch vodičov. Každé zariadenie na zbernici I2C má jedinečnú adresu.

Napríklad Arduino funguje ako master a BH1750 ako slave. Arduino komunikuje

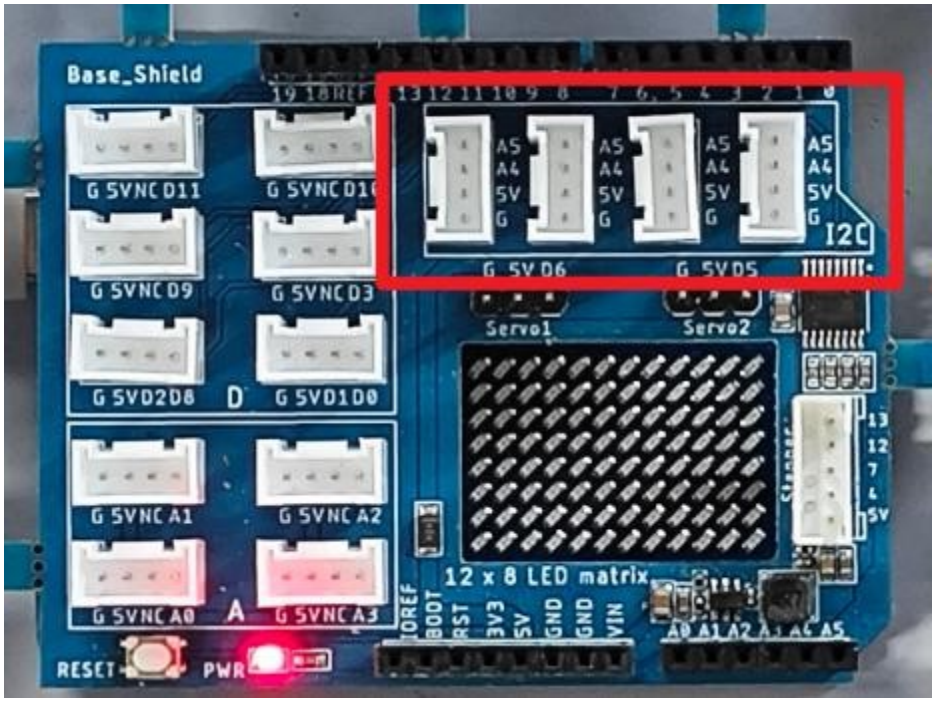
s konkrétnym zariadením prostredníctvom jeho adresy. V kóde: BH1750 lightMeter(0x5C);

0x5C je adresa I2C senzora BH1750. Arduino túto adresu používa na identifikáciu senzora a načítanie jeho údajov.

V Arduine komunikáciu I2C zabezpečuje knižnica Wire. Po inicializácii môžu všetky zariadenia I2C fungovať normálne:

```
#include <Wire.h>
void setup() {
    Wire.begin();        // Inicializácia zbernice I2C
}
```

Porty I2C sa zvyčajne nachádzajú tak, ako je znázornené na doske. K dispozícii sú štyri rozhrania I2C, čo umožňuje pripojenie viacerých zariadení súčasne.

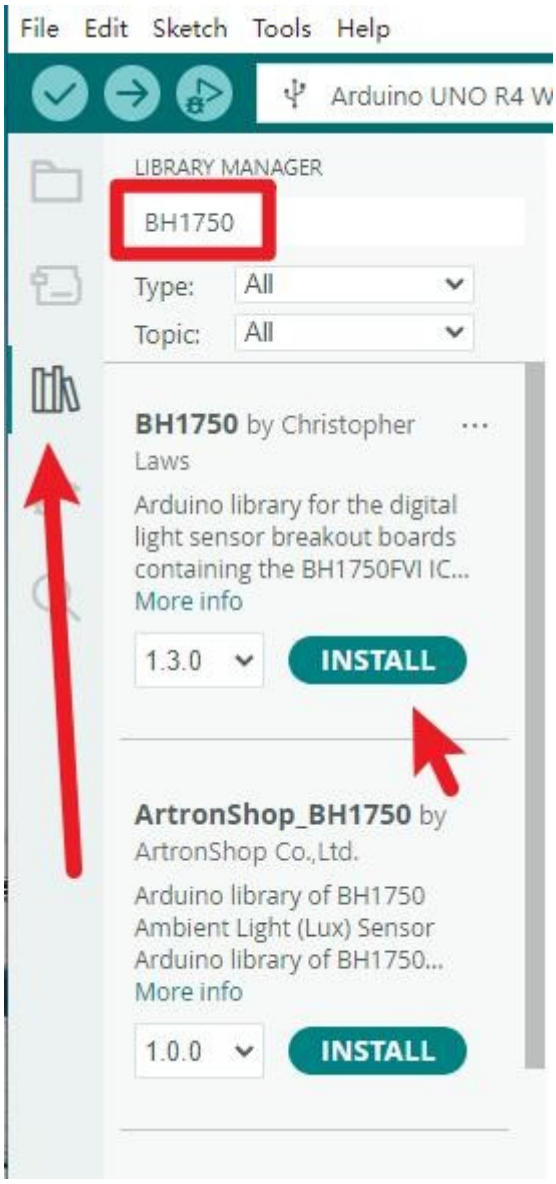


Nastavenie knižnice

Knižnica **BH1750** použitá v tomto návode je vo verzii 1.3.0. Inštalácia:

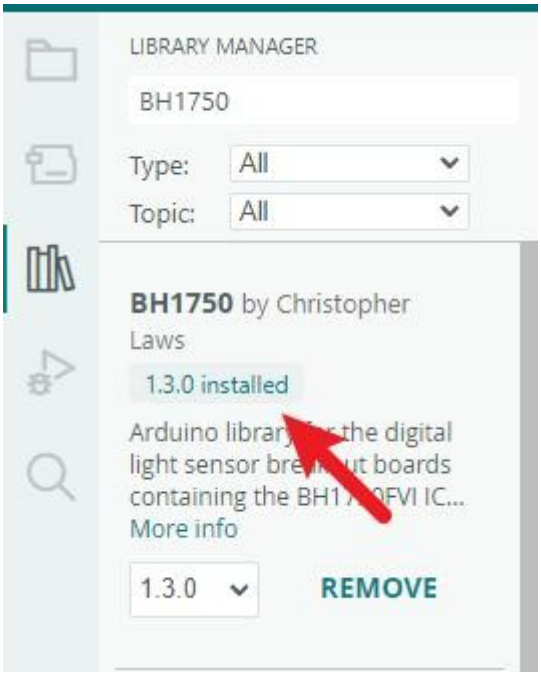
Otvorte Správcu knižníc v prostredí Arduino IDE (tretia ikona zľava).

Vyhľadajte názov knižnice, ktorú potrebujete.



Vyberte správnu verziu a kliknite na Inštalovať.

Knižnica Wire je integrovaná do systému a nevyžaduje inštaláciu. Po inštalácii je možné knižnicu BH1750 používať priamo.



2. Požadované moduly

Crowtail -- Svetelný senzor × 1

Crowtail – LED × 1

3. Spôsob zapojenia

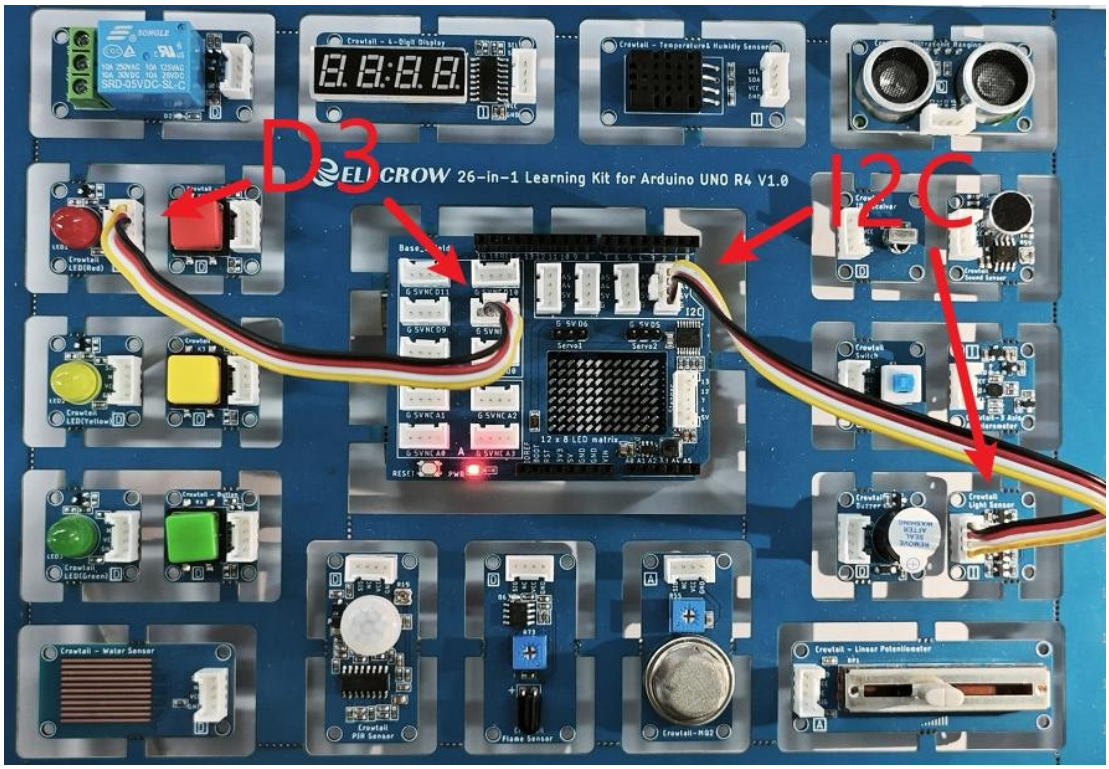
Crowtail -- Relé → Analógový port D3

Crowtail -- Svetelný senzor → Port I2C

Crowtail Špecifikácia štvorportového rozhrania:

- GND (čierna) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D3/I2C k

napájaniu 5 V.



4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-13_Light_Sensor

Otvorte program pomocou Arduino IDE.


```
#include <BH1750.h>      // Include BH1750 light sensor library
#include <Wire.h>        // Include I2C communication library

// Create a BH1750 sensor object with I2C address 0x5C
BH1750 lightMeter(0x5c);

// Variable to store the measured light intensity (lux)
float lux;

/************* LED Settings ************/
// Define the digital pin connected to the LED
#define LED_PIN 3

// Define the light intensity threshold (in lux)
// When light level is higher than this value, the LED will turn OFF
#define LUX_THRESHOLD 300

void setup() {
  // Initialize serial communication for debugging and monitoring
  Serial.begin(115200);

  // Initialize I2C communication (required for BH1750)
  Wire.begin();

  // Initialize the BH1750 sensor in continuous high resolution mode
  // Resolution: 1 lux
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
    Serial.println(F("BH1750 Advanced begin"));   // Initialization successful
  } else {
    Serial.println(F("Error initialising BH1750")); // Initialization failed
  }

  // Configure the LED pin as output
  pinMode(LED_PIN, OUTPUT);

  // Turn ON the LED at startup (default state)
  digitalWrite(LED_PIN, HIGH);
}

void loop() {
```

Vysvetlenie kľúčového kódu

(1) Vloženie knižnice a vytvorenie objektu.

```
#include <BH1750.h>       // Vloženie knižnice svetelného senzora BH1750
#include <Wire.h>         // Vloženie vstavanej knižnice I2C Arduina
```

BH1750.h — ovláda svetelný senzor BH1750.

Wire.h — vstavaná komunikačná knižnica Arduino I2C.

Keďže BH1750 pracuje cez I2C, musia byť zahrnuté obe knižnice.

```
BH1750 lightMeter(0x5C);
```

Vytvorí objekt senzora BH1750.

0x5C je adresa I2C senzora BH1750, ktorá umožňuje Arduinu lokalizovať ho na zbernici I2C.

```
float lux;
```

Ukladá aktuálnu intenzitu osvetlenia v luxoch.

Vyššie hodnoty znamenajú jasnejšie prostredie.

(2) Definícia LED a prahovej hodnoty.

```
#define LED_PIN 3

#define LUX_THRESHOLD 300
```

LED_PIN — LED je pripojená k digitálnemu pinu D3 na Arduine.

LUX_THRESHOLD — definuje prahovú hodnotu intenzity
osvetlenia.

Zmena pinu alebo prahu vyžaduje len úpravu týchto definícií.

(3) Inicializácia.

```
void setup() {
    Serial.begin(115200);
    Wire.begin();
```

Inicializuje sériovú komunikáciu pri 115200 baudov, čo vám umožňuje sledovať hodnoty osvetlenia a stav programu v sériovom monitore.

Inicializuje zbernicu I2C, ktorá je potrebná pre všetky zariadenia I2C.

```
    if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5C, &Wire)) {
        Serial.println(F("BH1750 Advanced begin"));
    } else {
        Serial.println(F("Chyba pri inicializácii BH1750"));
    }
```

Inicializuje senzor BH1750 a skontroluje, či bola inicializácia úspešná.

begin() nastaví komunikáciu I2C a režim senzora. Vráti true, ak je inicializácia úspešná, inak false.

CONTINUOUS_HIGH_RES_MODE — Režim nepretržitého merania s vysokým rozlíšením (presnosť 1 lux), najbežnejšie a najstabilnejšie nastavenie.

```
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, HIGH);
}
```

Nastaví pin LED ako výstup.

Pri spustení sa LED dioda štandardne zapne, aby boli viditeľné následné zmeny úrovne osvetlenia.

(4) Hlavná slučka.

```
if (lightMeter.measurementReady(true)) { lux
    = lightMeter.readLightLevel();
    Serial.print("[-] Svetlo: ["); Serial.print(lux);
    Serial.println("] lx");
}
```

Skontroluje, či BH1750 dokončil meranie (v prípade true počká na platné údaje). Načíta aktuálnu úroveň osvetlenia v luxoch.

Vytlačí hodnotu na sériový monitor na účely ladenia a pozorovania.

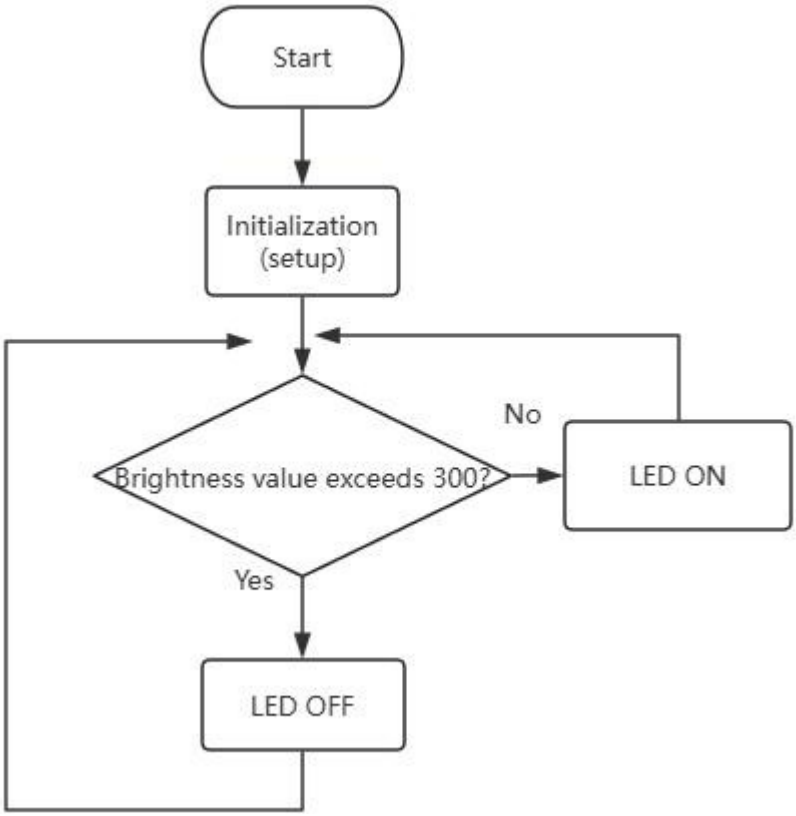
(5) Prahová logika a ovládanie LED.

```
if (lux > LUX_THRESHOLD) {
    digitalWrite(LED_PIN, LOW);
    Serial.println("LED vypnutá (príliš
    jasné)");
} else {
    digitalWrite(LED_PIN, HIGH); Serial.println("LED ON
    (Dark)");
}
```

Ak úroveň osvetlenia prekročí prahovú hodnotu, prostredie sa považuje za jasné a LED sa vypne.

Ak je úroveň osvetlenia nižšia ako prahová hodnota, prostredie sa považuje za tmavé a LED dióda sa rozsvieti.

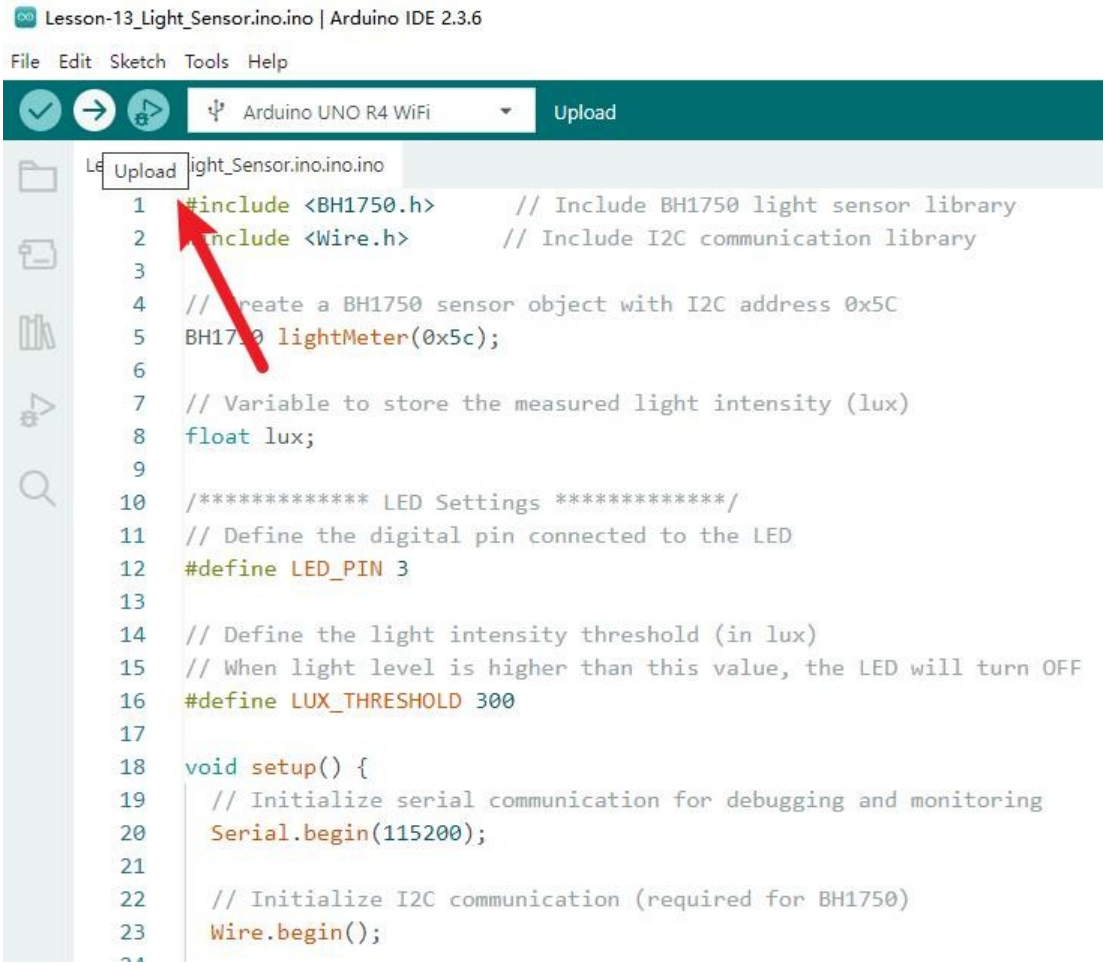
(6) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

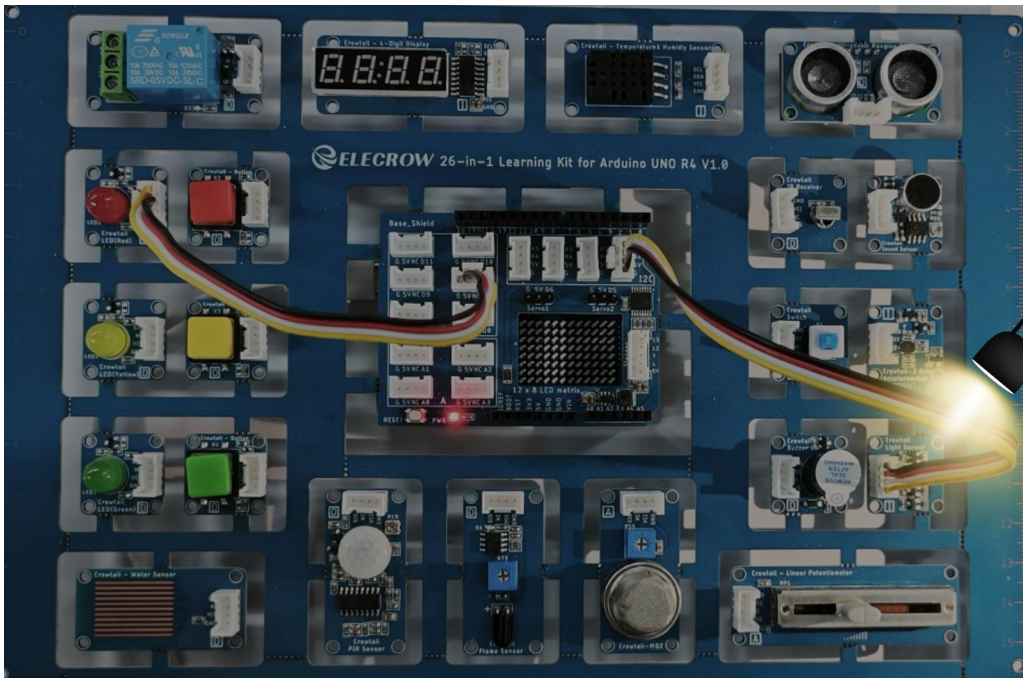
Kliknite na „Stiahnuť“:



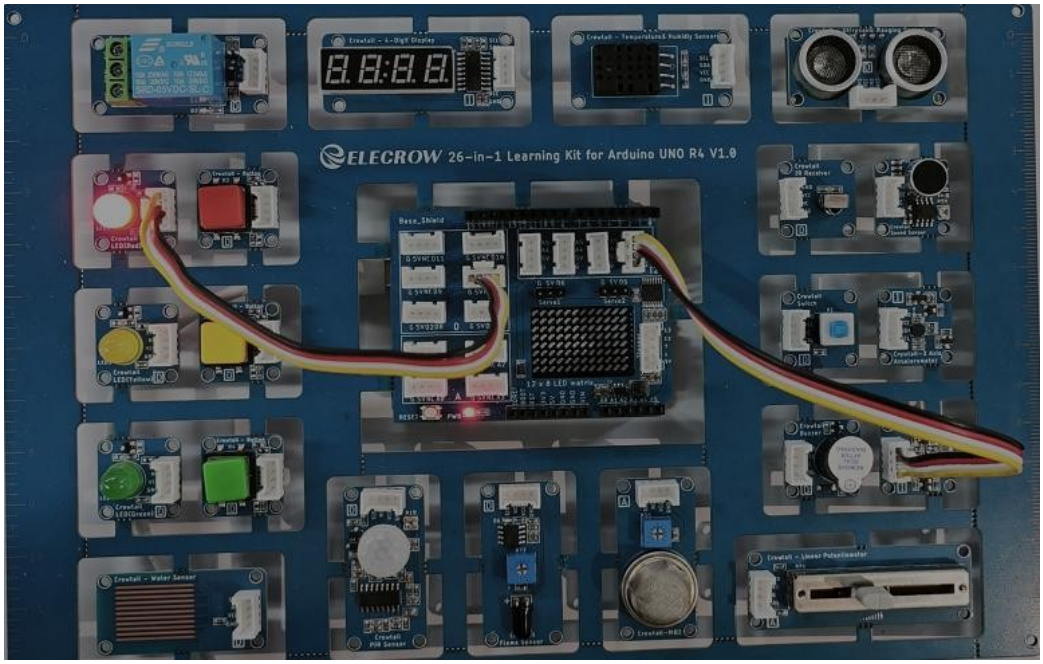
(2) Program sa spustí.

Správanie programu:

Keď je okolité osvetlenie dostatočné (jas nad 300 luxov), LED dióda zhasne.



Keď je okolité osvetlenie slabé (jas nižší ako 300 luxov), LED dióda sa rozsvieti.



Lekcia 14 – Digitálny displej

Úvod

V tejto lekcii sa zoznámime so štvormiestnym digitálnym displejom TM1650 a použijeme ho na vytvorenie 30-sekundového odpočítavacieho časovača. Štvormiestny digitálny displej môže zobrazovať štyri číslice a dá sa použiť na zobrazenie času, počítania hodnôt, napätia, prúdu a ďalších informácií. Na konci tejto lekcie budete mať hlboké pochopenie štruktúry digitálnych displejov a toho, ako ich ovládať pomocou kódu.

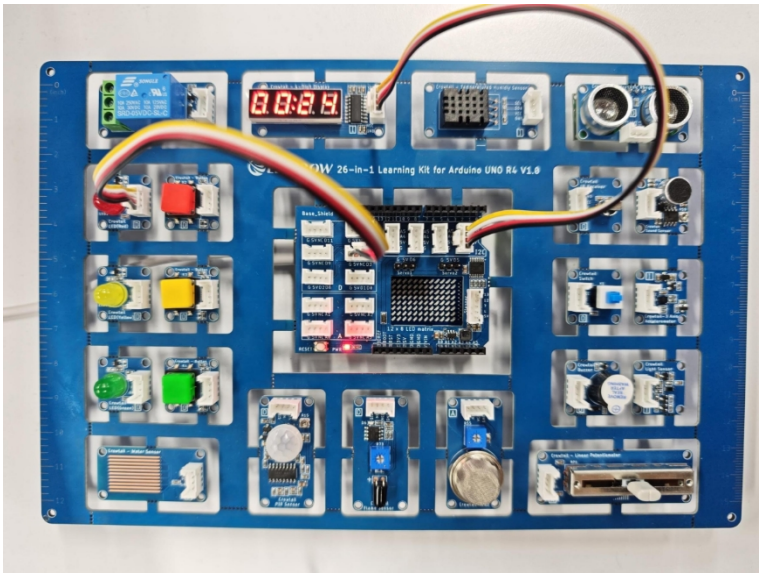
Ciele výučby

1. Zoznámte sa so základnými princípmi a spôsobmi zobrazenia 4-miestneho digitálneho displeja TM1650.
2. Naučte sa používať display.displayString() na zobrazenie číselných reťazcov.
3. Naučte sa používať funkciu display.setDot() na ovládanie desatinnej čiarky (používa sa na simuláciu blikania dvojbodky).
4. Ovládnite najzákladnejšiu metódu delenia čísel, delenie celých čísel na tisíce, stovky, desiatky a jednotky.
5. Naučte sa gramatické použitie while.
6. Naučte sa funkciu modifikátora void pre funkcie.
7. Zopakujte si a upevnite si gramatické použitie cyklov „for“.
8. Byť schopný implementovať kompletný program odpočítavania.

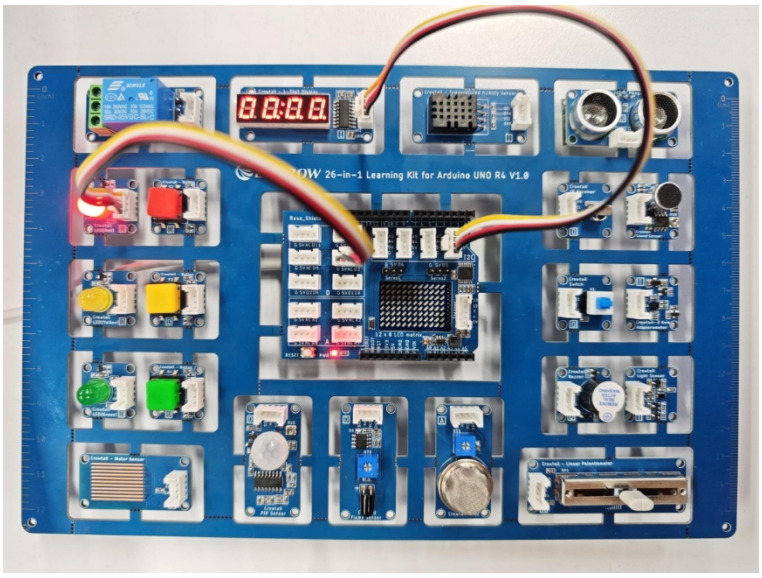
Náhľad výsledku

Štvormiestny displej zobrazuje zostávajúci čas (napríklad „0030“ predstavuje 30 sekúnd). Počas odpočítavania sa čas každú sekundu znižuje o jednu sekundu a bliká dvojbodka; keď sa odpočítavanie skončí, zobrazí sa „0000“ a čísla, desatinná čiarka, dvojbodka a červená LED dióda blikajú synchronizovane, aby na to upozornili.

Odpočítavanie začína: Digitálny displej sa dynamicky mení



Odpočítavanie skončilo: Digitálny displej bliká a bliká aj červené svetlo.



1. Vysvetlenie princípu

Sedemsegmentový displej je zobrazovací modul zložený z viacerých nezávislých LED segmentov. Skladá sa zo siedmich segmentov (a–g) na zobrazovanie čísiel a desatinnej čiarky (DP). Individuálnym rozsvietením týchto segmentov je možné zobraziť čísla od 0 do 9, ako aj niektoré písmená. Každý segment je LED diódou, takže môže svietiť len v stave „zapnuté“ alebo „vypnuté“.

Existujú dve základné schémy zapojenia digitálnych trubíc:

- Spoločná katóda: Záporné póly všetkých LED diód sú spojené dohromady. Pripojením kladného napätia na určitý segment sa tento rozsvieti.
- Spoločná anóda: Kladné póly všetkých LED diód sú spojené dohromady. Pripojením nízkeho napätia na určitý segment sa tento rozsvieti.

Tieto štyri digitálne trubice sú v skutočnosti štyri sedemsegmentové displeje integrované do jedného celku. Aby sa ušetrili piny, tieto štyri budú zdieľať riadiace linky segmentov (a~g a DP), ale každá číslica bude mať vlastné ovládanie „bit select“. Hlavný riadiaci čip rozsvieti každú číslicu vo veľmi rýchlej sekvencii (označované ako „dynamické skenovanie“), vďaka vizuálnej perzistencii ľudského oka sa bude zdať, že všetky štyri svietia súčasne.

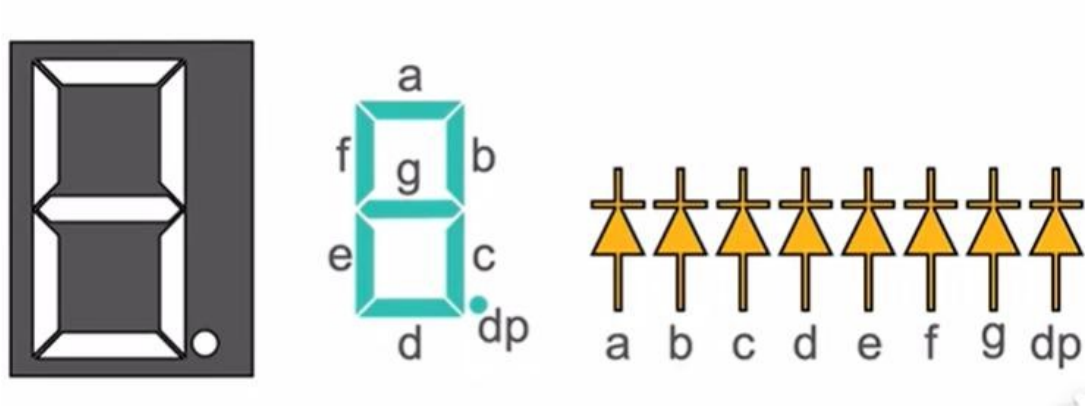
Na zjednodušenie ovládania sú na mnohých vývojových doskách štyri digitálne trubice často spárované s ovládačovým čipom, napríklad:

HT16K33, TM1650 atď. Ovládače digitálnych trubíc I2C

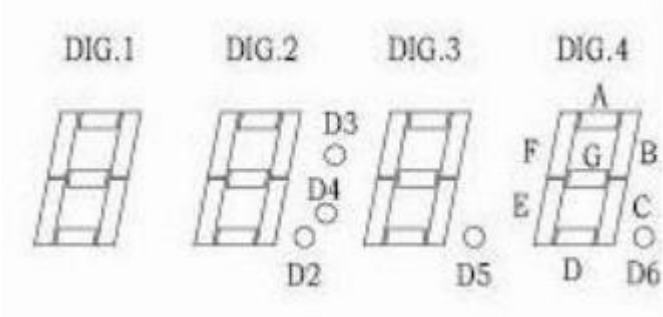
Tieto ovládače prijímajú pokyny prostredníctvom komunikácie I2C a interne vykonávajú:

- Ovládanie osvetlenia jednotlivých segmentov (a–g, DP)
- Štvormiestne dynamické skenovanie
- Nastavenie jasu
- Zobrazenie desatinného miesta a dvojbodky

Týmto spôsobom stačí, aby hlavná riadiaca doska odoslala znaky, ktoré sa majú zobraziť, napríklad „12:34“, „0000“, a riadiaci čip automaticky zobrazí čísla.



V tomto produkte nie je prvá desatinná čiarka vľavo osvetlená, pretože to vyžaduje hardvér.



Ako sme spomenuli vyššie, táto digitálna trubica komunikuje aj pomocou protokolu I2C. Podrobné informácie o I2C nájdete v predchádzajúcej lekcii (Lekcia 13 – Svetelný senzor).

2. Potrebné moduly

Digitálny displej Crowtail (sedemsegmentový displej x1)

LED dióda Crowtail (ľubovoľná farba x1)

3. spôsob zapojenia

Piny digitálneho trubicového modulu TM1650:

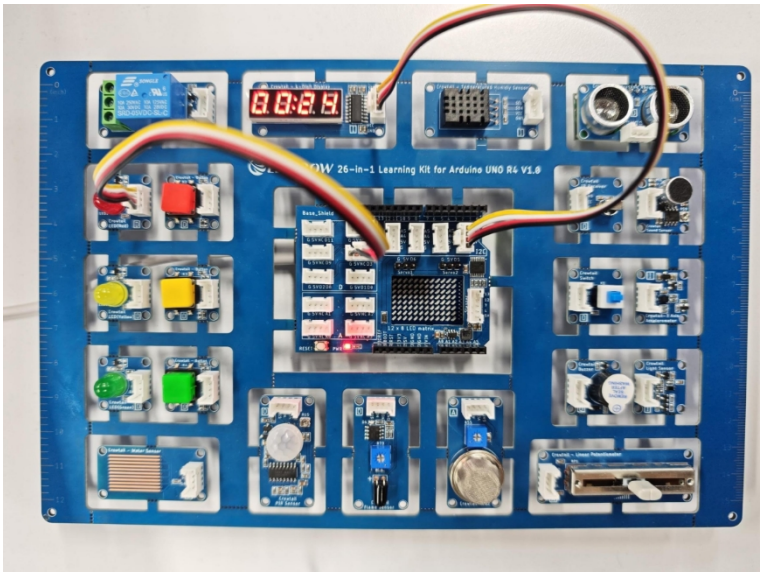
- **VCC** → Napájanie (pripojte k Arduino 5V)
- **GND** → Zem (pripojte k Arduino GND)
- **SDA** → Dátový vodič (pin Arduino A4)
- **SCL** → Hodinový vodič (pin A5 Arduina)

Piny modulu červenej LED:

LED Crowtail → port DIGITAL D3

Špecifikácia štvorpinového rozhrania Crowtail:

SIG (žltý) / NC (biely) / VCC (červený) / GND (čierny)



4. Príklad vysvetlenia

Kliknite na odkaz a stiahnite si ukážkový kód poskytnutý oficiálnym zdrojom:

Odkaz na GitHub：

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-14_DigitalDisplay/14_DigitalDisplay

Otvorte program pre túto lekciu zo zložky **„14_DigitalDisplay“** pomocou Arduino IDE.

14_DigitalDisplay.ino

```
#include <Wire.h>
#include <TM1650.h>

TM1650 display;

#define LED_RED 3

int countdownValue = 30;  // Countdown time (seconds)

// Display a four-digit number.
void showNumber(int num) {
  char buf[5];  //Four digits + one end character

  // Calculate each digit.
  int thousands = num / 1000;            // thousandth place
  int hundreds  = (num / 100) % 10;      // hundreds place
  int tens      = (num / 10) % 10;       // ten's place
  int ones      = num % 10;              // units digit

  // Convert to characters
  buf[0] = '0' + thousands;
  buf[1] = '0' + hundreds;
  buf[2] = '0' + tens;
  buf[3] = '0' + ones;
  buf[4] = '\0'; // String terminator

  display.displayString(buf);
}


void setup() {
  Wire.begin();
  Serial.begin(115200);

  pinMode(LED_RED, OUTPUT);
  digitalWrite(LED_RED, LOW);

  display.init();
  display.displayOn();
  display.setBrightness(3);
}

void loop() {

  bool colonState = false;   // Used for countdown flashing
```

Vysvetlenie kľúčového kódu

(1) Prvé dva riadky: Vloženie knižníc (ako požičanie nástrojov)

```
#include <Wire.h>

#include <TM1650.h>
```

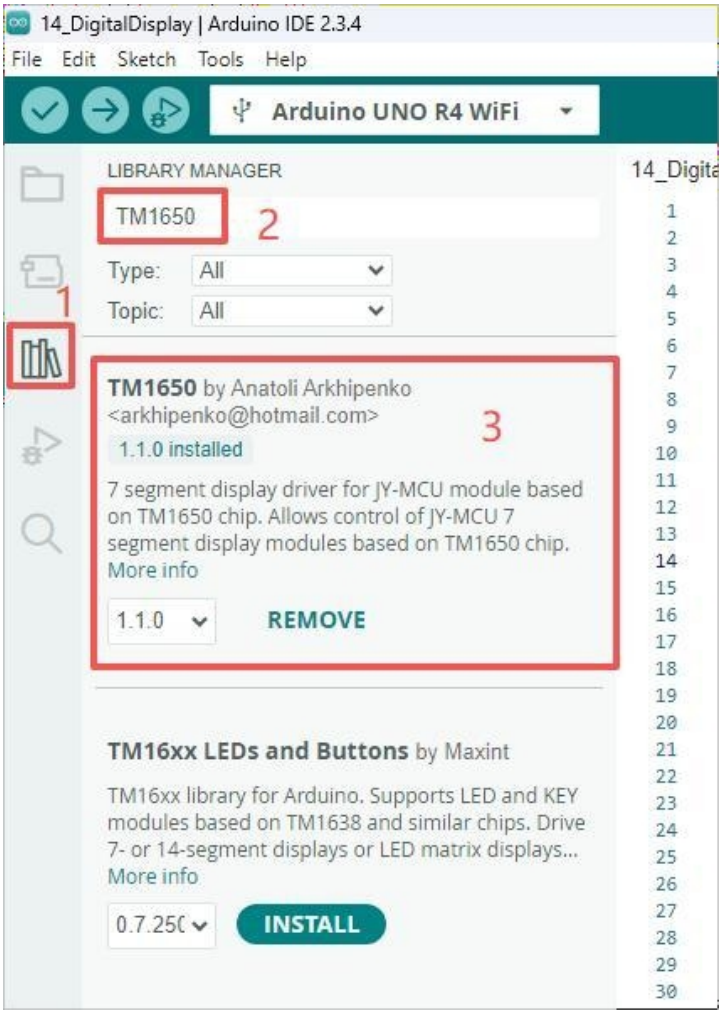
- #include = „použi túto sadu nástrojov“ v kóde.

- **Wire.h:** Vstavaná sada nástrojov pre komunikáciu **I2C** (umožňuje Arduinu komunikovať s modulom TM1650).
- **TM1650.h:** Špeciálna sada nástrojov na ovládanie digitálneho modulu **TM1650** (napísal to niekto iný, takže nám to ušetrí veľa práce).

➢ Stiahnutie v prostredí Arduino IDE

Kliknite na možnosť **„Library Manager“** na ľavej strane Arduino IDE. Tu vyhľadajte **„TM1650“** a stiahnite si ju.

(Inštalovaná verzia knižnice TM1650 je 1.1.0.)



(2) Vytvorte „objekt displeja“ (pomenujte našu digitálnu trubicu)

Po zapnutí alebo reštartovaní Arduina sa funkcia setup() vykoná len raz a slúži ako inicializačný proces pre prostredie.

```
Displej TM1650;
```

Tento riadok vytvorí „menovku“ s názvom display pre našu digitálnu trubicu. Teraz, kedykoľvek napíšeme **display.**something(), dáme digitálnej trubici pokyn, čo má robiť (napr. display.init() = „inicializovať digitálnu trubicu“).

(3) Definujte pin LED (priraďte LED skratku)

```
#define LED_RED 3
```

- #define = „priradiť prezývku k číslu“. Tu hovoríme: „odteraz **LED_RED** znamená pin 3“. Vďaka tomu je kód čitateľnejší – namiesto toho, aby sme si pamätali**, že „pin 3 je LED“**, jednoducho napíšeme **LED_RED**.
- Ak chcete neskôr presunúť LED na **pin 4**, stačí zmeniť tento riadok na **#define LED_RED 4** (ostatné riadky meniť netreba!).

(4) Nastavte počiatočnú hodnotu odpočítavania

```
int countdownValue = 30;
```

- **int** = „táto premenná ukladá celé číslo“ (napr. 30, 29, 0).
- **countdownValue** = názov nášho odpočítavacieho čísla (začína na 30 sekundách).
- **Číslo 30** môžete zmeniť na ľubovoľné číslo (napríklad 60 pre 1 minútu, 10 pre 10 sekúnd).

(5) Funkcia showNumber (zobrazenie 4-miestnych čísel)

```
void showNumber(int num) {
  char buf[5];      //Štyri číslice + jeden koncový znak


  // Vypočítaj každú číslicu.
  int thousands = num / 1000;                    // tisícina
  int stovky        = (num / 100) % 10;          // stovky
  int               = (num / 10) % 10;           // desiatky
desiatky
int                 = num % 10;                  // jednotky
jednotky


  // Premeniť na znaky
  buf[0] = '0' + tisíce; buf[1] = '0' +
  stovky; buf[2] = '0' + desiatky;
  buf[3] = '0' + jednotky;
  buf[4] = '\0'; // Ukončovač reťazca
```

```
    display.displayString(buf);
}
```

① **void showNumber(int num) {**

- **void**：Označuje, že táto funkcia nevracia žiadnu hodnotu (vykonáva len funkcie v kóde a nevracia žiadny výsledok).
- show**Number:** Toto je názov funkcie, čo znamená „zobraziť číslo“.
- **(int num)**：Funkcia má parameter s názvom num, ktorý je typu int (celé číslo). Keď zavoláte showNumber(30);, num bude rovné 30.

② **char buf[5];**

- Tento riadok vytvorí pole s názvom **„buf“**, s typom **„char“** a dĺžkou 5.
- V jazykoch **C/C++** sú reťazce zvyčajne reprezentované poľom znakov spolu s ukončovacím znakom **„\0“** (písaným aj ako 0). Preto môže **buf[5]** obsahovať **4** viditeľné znaky + **1** koncový znak.
- V tomto prípade plánujeme zobraziť 4-miestne čísla (napríklad „0030“, „0123“, „9999“), takže potrebujeme 4 pozície pre znaky plus jeden ukončovací znak '\0', čo je spolu 5.

Kľúčový pojem: reťazec v štýle C

Napríklad „1234“ v pamäti je {'1','2','3','4','\0'}. '\0' označuje funkcii, že reťazec tu končí.

③ **Výpočet digitálnych bitov**

```
// Calculate each digit.
int thousands = num / 1000;          // thousandth place
int hundreds  = (num / 100) % 10;    // hundreds place
int tens      = (num / 10) % 10;     // ten's place
int ones      = num % 10;            // units digit
```

Čo sú celočíselné delenie a zvyšok (/ a %)?

- **num / 1000:** Vydelíme číslo num číslom 1000 a vezmeme celú časť (desatinná časť sa zahodí). Napríklad: 1234 / 1000 = 1, 30 / 1000 = 0.
- **num % 10:** Vezmite zvyšok po vydelení num číslom 10, čo je posledná číslica (jednotková číslica). Napríklad 1234 % 10 = 4.
- **((num / 100) % 10):** Najprv vydelíme num číslom 100 (odstránime posledné dve číslice), potom vezmeme zvyšok po delení číslom 10, čím získame stovkovú číslicu. Príklad: 1234 / 100 = 12, 12 % 10 = 2.

Príklad: Ak je num = 305:

- **tisíce** = 305 / 1000 = 0
- **stovky** = (305 / 100) % 10 = 3 % 10 = 3

- **desiatky** = (305 / 10) % 10 = 30 % 10 = 0
- **jednotky** = 305 % 10 = 5

Výsledok je 0, 3, 0, 5 —— čo zodpovedá štyrom čísliciam, zobrazí sa „0305“ (všimnite si, že vedúce nuly zostanú zachované).

④ **Preveďte čísla na znaky**

```
// Convert to characters
buf[0] = '0' + thousands;
buf[1] = '0' + hundreds;
buf[2] = '0' + tens;
buf[3] = '0' + ones;
buf[4] = '\0'; // String terminator
```

- Vypočítané tisíce, stovky, desiatky a jednotky sú všetky celé čísla v rozmedzí od 0 do 9 (za predpokladu, že vstupné údaje sú správne).
- Premenná **buf** je však pole znakov a my do nej potrebujeme vložiť znaky od „0“ po „9“, takže musíme čísla previesť na príslušné znaky.
- V kódovaní ASCII má číselný kód znaku „0“ pevný vzťah k číslu 0: „0“ + 0 je znak „0“, „0“ + 1 je „1“ atď.
- Napríklad, ak ones = 5, potom '0' + ones sa stane znakom '5'.
- Toto je najbežnejšia a najrýchlejšia metóda prevodu čísel na znaky.

⑤ **buf[4] = '\0'**

```
// Convert to characters
buf[0] = '0' + thousands;
buf[1] = '0' + hundreds;
buf[2] = '0' + tens;
buf[3] = '0' + ones;
buf[4] = '\0'; // String terminator
```

- Tento riadok je mimoriadne dôležitý: Nastavte koncový znak na '\0' (nulový znak), ktorý označuje koniec reťazca.
- Ak sa tento koniec nepridá, funkcie ako display.displayString(buf) budú pokračovať v čítaní ďalší bajt v pamäti, kým nenájdu '\0', čo môže viesť k nesprávnemu výstupu alebo skresleným znakom (a môže tiež spôsobiť zrútenie programu).

⑥ **display.displayString(buf);**

- Tento riadok odovzdáva zostavený reťazec na zobrazenie (objekt ovládača TM1650).
- Funkcia displayString pravdepodobne očakáva reťazec typu C zakončený znakom '\0', preto je veľmi dôležité správne nastaviť formát premennej buf.

⑦ **Napríklad, aby sme lepšie pochopili**

Predpokladajme, že volanie je **showNumber**(30); ：

- **tisíce** = 30 / 1000 = 0

- **stovky** = (30 / 100) % 10 = 0 % 10 = 0
- **desiatky** = (30 / 10) % 10 = 3 % 10 = 3
- **jednotky** = 30 % 10 = 0
- buf -> {'0','0','3','0','\0'} → reťazec „0030“ → digitálny displej zobrazí 0030 (štyri číslice).

Predpokladajme showNumber(7); → „0007“

Predpokladajme showNumber(1234); →

„1234“ Predpokladajme showNumber(9999);

→ „9999“

(6) Funkcia setup() (spustí sa JEDENKRÁT pri zapnutí Arduina)

```
void setup() {
    Wire.begin();
    Serial.begin(115200);


    pinMode(LED_RED, OUTPUT);
    digitalWrite(LED_RED, LOW);


    display.init();
    display.displayOn();
    display.setBrightness(3);
}
```

Funkcia setup() je ako „príprava“ – Arduino vykoná všetky tieto kroky raz, keď ho zapojíte alebo stlačíte tlačidlo reset. Poďme na to riadok po riadku:

①**Wire.begin();:** Spustí komunikáciu I2C (umožňuje Arduinu komunikovať s modulom TM1650). Bez tohto kódu digitálna trubica nebude fungovať!

②**Serial.begin(115200);:** Zapne sériový port (umožňuje Arduinu posielať správy do sériového monitora vášho počítača, aby ste mohli vidieť čísla odpočítavania). 115200 je „rýchlosť“ (baudová rýchlosť) – neskôr sa uistite, že je sériový monitor nastavený na toto číslo.

③**pinMode(LED_RED, OUTPUT);:** Oznamuje Arduinu, že „pin 3 (LED_RED) je výstupný pin“ – výstupné piny používame na zapínanie a vypínanie LED diód (vstupné piny slúžia na čítanie senzorov).

④**digitalWrite(LED_RED, LOW);:** Vypne červenú LED pri prvom spustení Arduina (LOW = vypnuté, HIGH = zapnuté pre väčšinu LED). Nechceme, aby sa LED náhodou zapla!

⑤**display.init();:** Inicializuje digitálnu trubicu (resetuje ju, nastaví komunikáciu s TM1650). Predstavte si to ako „prvé zapnutie digitálnej trubice“.

⑥**display.displayOn();:** Uistí sa, že obrazovka digitálnej trubice je zapnutá (niektoré moduly sa štartujú automaticky).

⑦**display.setBrightness(3);:** Nastaví jas digitálnej trubice. Hodnota môže byť od 0 (najtmavšia) do 7 (najjasnejšia) – 3 je stredná (dobrá viditeľnosť bez toho, aby bola príliš jasná).

(7) Funkcia loop() (beží stále dookola)

Funkcia loop() je miestom, kde sa deje mágia – Arduino tu nepretržite opakuje všetko, kým ho neodpojíte.

Rozdelili sme ju na dve časti: odpočítavanie a blikajúce upozornenie, keď odpočítavanie skončí.

```
void loop() {
  bool colonState = false;          // Používa sa na blikanie odpočítavania
  // ----------------------------
  // Odpočítavanie s blikajúcim dvojbodkou
  // ----------------------------
  while (countdownValue >= 0) {
    showNumber(countdownValue);

    // Blikanie dvojbodky (s použitím prvej desatinnej čiarky ako dvojbodky)
    colonState = !colonState;
    display.setDot(1, colonState);          // Blikanie bodky bitu 1 → Blikanie dvojbodky

    // Udržujte desatinnú čiarku vypnutú.
    display.setDot(0, false);
    display.setDot(2, false);
    display.setDot(3, false);
    Serial.println(countdownValue);
    delay(1000);
    countdownValue--;
  }

  //
  // Režim blikania, keď odpočet dosiahne 0 (režim budíka)
  ------------------------------------------------------------
  // Čísla, dvojbodka, desatinná čiarka a LED diódy blikajú súčasne
  //
  while (true) {
    ------------------------------------------------------------
```

```
    // Zapnúť — zobraziť 0000 + všetky bodky + LED
    showNumber(0);

    for (int i = 0; i < 4; i++) {
      display.setDot(i, true);        // Všetky bodky (vrátane dvojbodiek) blikajú
    }
    digitalWrite(LED_RED, HIGH); delay(300);
    // Vypnúť — zavrieť obrazovku + vypnúť
    LED display.displayOff();
    digitalWrite(LED_RED, LOW);
    delay(300);
    display.displayOn();        // Znovu zapnúť pre ďalší cyklus
  }
}
```

Pozrime sa bližšie na to, čo presne sa v slučke popisuje.

① **Definujte premennú, ktorá riadi, či je dvojbodka rozsvietená.**

```
void loop() {

  bool colonState = false;    // Used for countdown flashing
```

- **„bool“** predstavuje booleovskú hodnotu, ktorá môže mať iba hodnotu true (**pravda**) alebo false (**nepravda**).
- Používame to na určenie, či je dvojbodka zapnutá alebo vypnutá.

„Blikanie“ dvojbodky je neustále striedanie medzi **hodnotami true** a **false**.

② **Začína fáza odpočítavania**

```
50    while (countdownValue >= 0) {
```

Toto je cyklus while. Pokiaľ je číslo odpočítavania väčšie alebo rovné 0, bude sa kód v ňom nepretržite vykonávať.

③ **Zobrazenie čísla**

```
52    showNumber(countdownValue);
```

Toto ste sa už naučili v predchádzajúcej časti. Táto funkcia rozloží číslo na štvormiestne čísla a zobrazí ich na digitálnom displeji.

④ **Dvojbodka bliká, keď je príkaz v hlavnej časti**

```
55    colonState = !colonState;
56    display.setDot(1, colonState);  // Bit 1 dot flash → Colon flash
```

colonState = !colonState;

- Ak je colonState **nepravdivé**

→ !false je pravda → stane sa pravdou

- Ak bola hodnota colonState pôvodne **true**

jeho invertovanie bude mať za následok hodnotu false

To znamená, že dvojbodka sa bude striedať zakaždým → Dvojbodka sa rozsvieti a zhasne každú sekundu = Bliká.

Potom tu, **display.setDot(1, colonState);,** toto je operácia na zapnutie a vypnutie dvojbodky v digitálnej trubici (t. j. blikajúci efekt)

⑤ **Vypnite ostatné DOT**

```
display.setDot(0, false);
display.setDot(2, false);
display.setDot(3, false);
```

Efekt uzamknutia:

Bude blikať len bit 1, zatiaľ čo ostatné desatinné miesta nebudú nikdy svietiť.

⑥ **Znížte číslo odpočítavania o 1**

```
66      countdownValue--;
67    }
```

Napríklad:

30 → 29 → 28 → ⋯ → 1 → 0 → -1

Keď sa stane -1:

podmienka **while (countdownValue >= 0)** neplatí

→ Fáza odpočítavania končí

→ Prechod do fázy alarmu

⑦ **Prechod do alarmovej slučky (nekonečné blikanie)**

```
72    // --------------
73    while (true) {
74
```

While(true) znamená „vždy pravda“ a program sa nikdy neukončí. Tomu sa hovorí „neustále bliká“.

⑧ **Zobraziť 0000 a rozsvietiť všetky desatinné miesta + Zapnúť LED**

```
75    // Turn ON — show 0000 + all dots + LED
76    showNumber(0);
77
78    for (int i = 0; i < 4; i++) {
79      display.setDot(i, true);  // All DOTS (including colons) are flashing
80    }
81
82    digitalWrite(LED_RED, HIGH);
83    delay(300);
```

Efekt:

- Zobraziť „0000“
- Všetky desatinné čiarky svietia
- LED svieti

Trvá 0,3 sekundy

⑨ **Vypnúť obrazovku + Vypnúť LED**

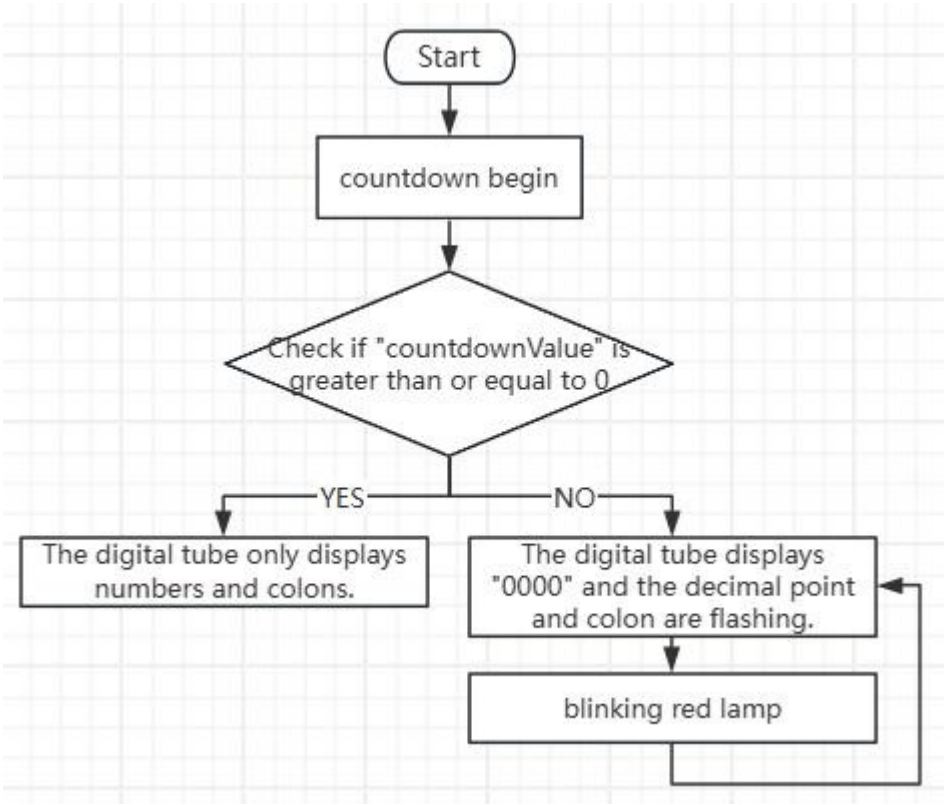
```
85      // Turn OFF – close screen + LED off
86      display.displayOff();
87      digitalWrite(LED_RED, LOW);
88      delay(300);
89
90      display.displayOn();  // Re-enable for next cycle
91    }
92  }
```

Efekt na javisku:

- Obrazovka zhasne
- LED zhasne
- Po 0,3 sekundách sa digitálna trubica opäť rozsvieti

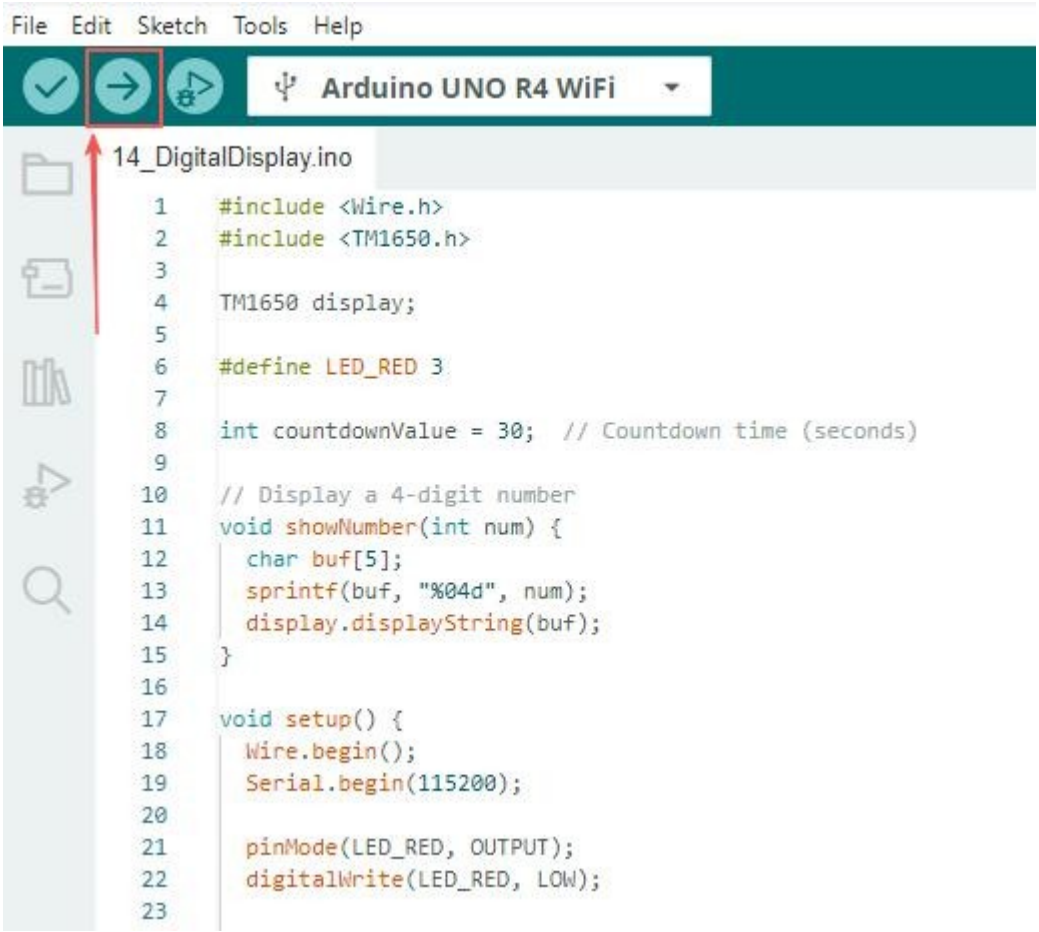
Výsledkom je blikajúci vzor zapnuté-vypnuté-zapnuté-vypnuté.

(8) Logický tok kódu



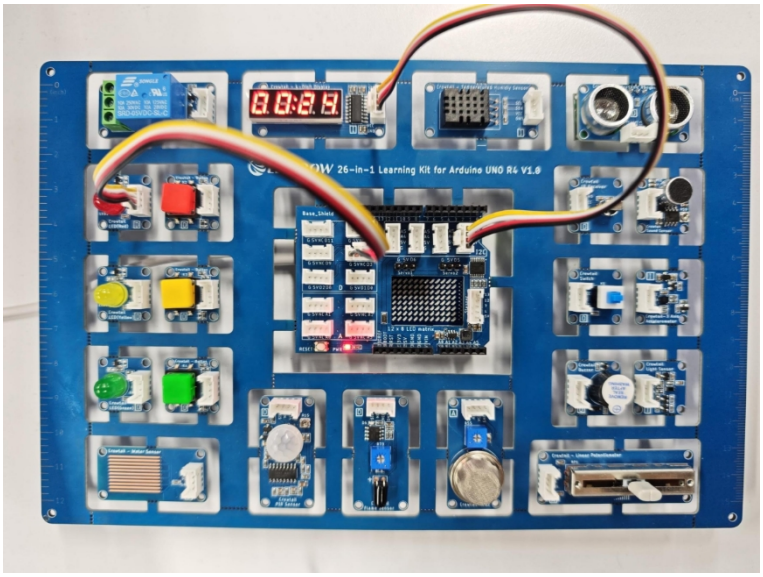
5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania. Kliknite **na „Stiahnuť“**:

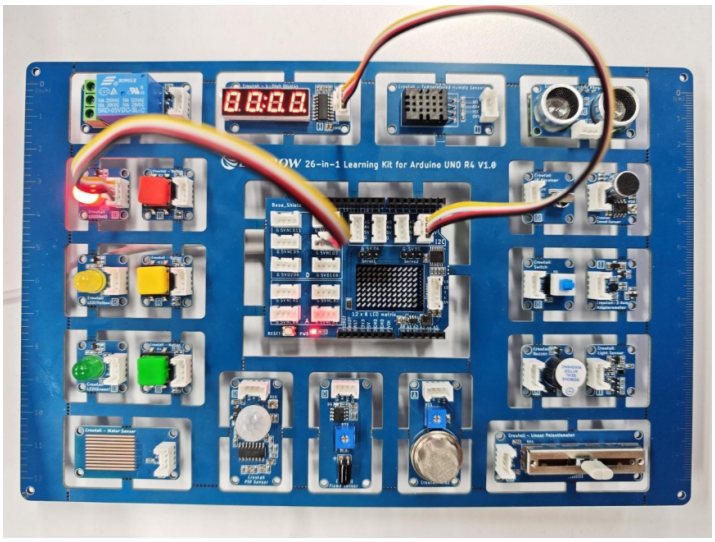


(2) Po dokončení sťahovania skontrolujte efekt behu:

Začína odpočítavanie: Digitálna trubica sa dynamicky mení



Odpočítavanie skončilo: Digitálna trubica bliká a bliká aj červené svetlo.



Lekcia 15 --- LCD1602

Úvod

V tejto lekcii sa naučíme používať Arduino na ovládanie displeja LCD1602, na ktorom sa budú v reálnom čase zobrazovať údaje o intenzite osvetlenia zo senzora BH1750. Senzor osvetlenia pripojíme k LCD displeju prostredníctvom zbernice I2C, čím dosiahneme dynamické aktualizovanie údajov o intenzite okolitého osvetlenia na displeji.

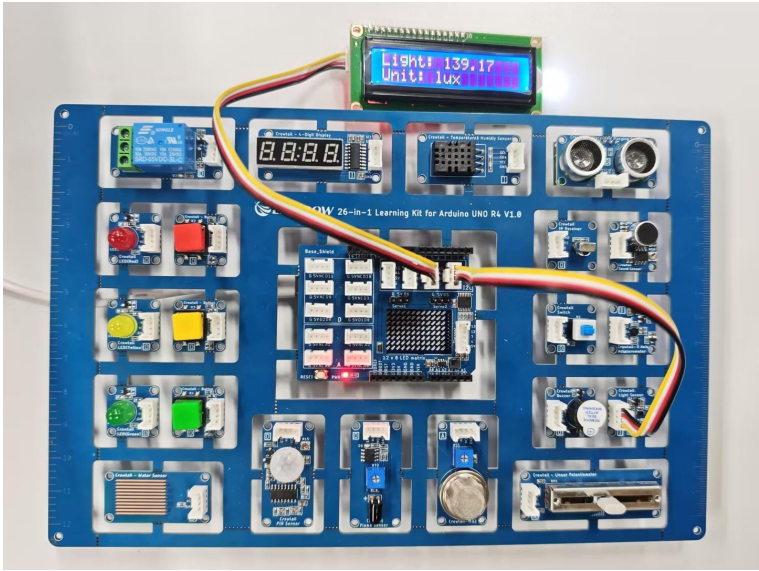
Po absolvovaní tejto lekcie budete mať jasnú predstavu o komunikácii viacerých zariadení cez I2C, vďaka čomu sa v budúcnosti vyhnete narušeniu programu pri súčasnom používaní viacerých senzorov I2C.

Ciele výučby

1. Porozumieť princípu fungovania displeja LCD1602.
2. Naučiť sa používať knižnice tretích strán (Adafruit_LiquidCrystal, BH1750) na rýchle ovládanie hardvéru.
3. Naučiť sa používať lcd.setBacklight(int num) na nastavenie jasu podsvietenia.
4. Naučte sa používať funkciu lcd.setCursor na nastavenie polohy kurzora, aby sa tu zobrazili údaje.
5. Dokončite spoločné zobrazenie údajov medzi svetelným senzorom a LCD obrazovkou.

Náhľad výsledku

LCD1602 zobrazuje v reálnom čase v prvom riadku „Svetlo: + konkrétna hodnota intenzity svetla“ a v druhom riadku sa stále zobrazuje „Jednotka: lux“.

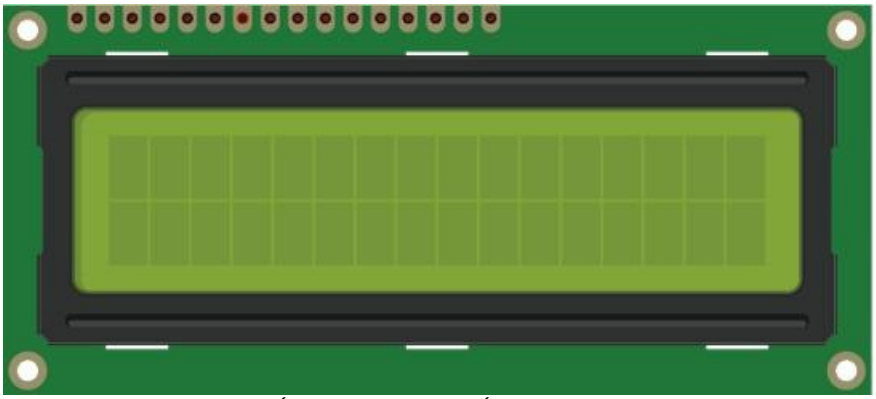


Zároveň monitor sériového portu bude tiež súčasne vysielať hodnotu intenzity osvetlenia a táto hodnota sa bude aktualizovať v reálnom čase podľa zmien okolitého osvetlenia.

1. Vysvetlenie princípu

LCD1602 je široko používaný znakový modul s tekutými kryštálmi, ktorý slúži na zobrazovanie písmen, čísel a symbolov. Skladá sa zo znakového displeja s tekutými kryštálmi (LCD), hlavného riadiaceho obvodu HD44780 a jeho rozšíreného riadiaceho obvodu HD44100, ako aj z malého počtu rezistorov, kondenzátorov a konštrukčných dielov namontovaných na doske s plošnými spojmi.

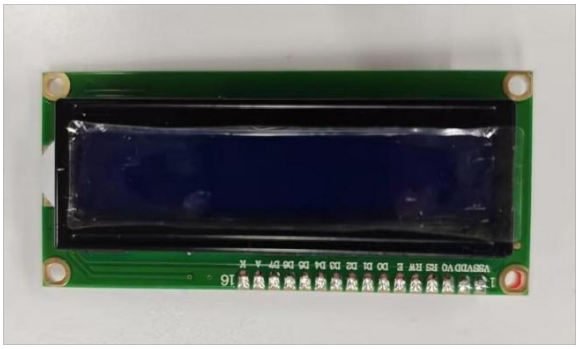
Tekutý kryštál s bodovou maticou sa skladá z M x N zobrazovacích jednotiek. Za predpokladu, že LCD displej



má 64 riadkov, každý riadok má 128 stĺpcov a každých 8 stĺpcov zodpovedá 1 bajtu s 8 bitmi, to znamená, že každý riadok má 16 bajtov, celkom 16x8 = 128 bodov. Zobrazovacie jednotky 64x16 na obrazovke zodpovedajú 1024 bajtom oblasti RAM displeja a obsah každého bajtu zodpovedá jasu príslušnej pozície na obrazovke.

2. Potrebné moduly

Modul Crowtail LCD1602 × 1

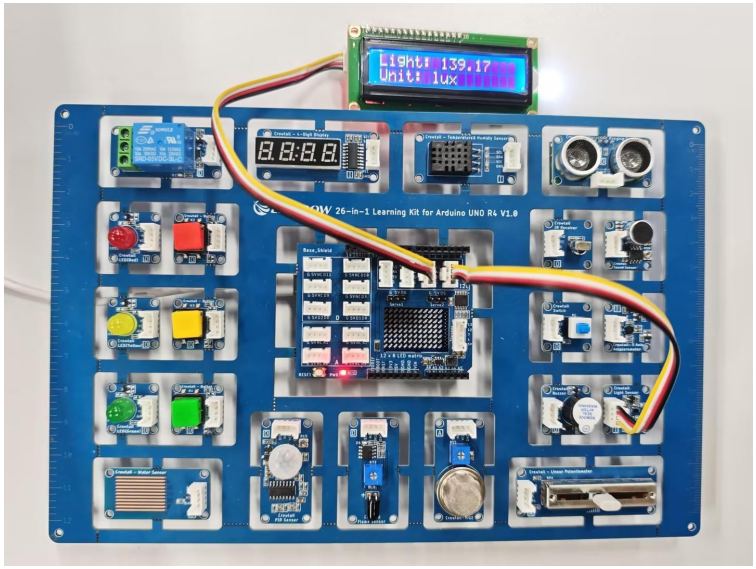


Modul Crowtail Light Sensor × 1

3. Spôsob zapojenia

Crowtail LCD1602 → ľubovoľný port I2C

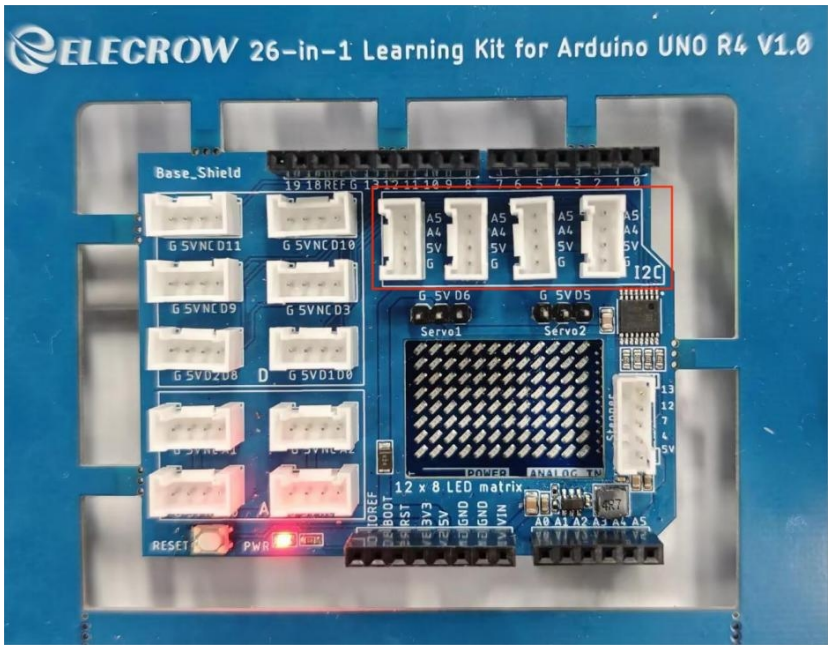
Senzor osvetlenia Crowtail → ľubovoľný



port I2C

Špecifikácia štvorportového rozhrania Crowtail:

- GND (čierna) → GND
- VCC (červená) → 5V
- SDA (biela) → A4
- SCL (žltá) → A5



4. Vysvetlenie príkladu

Kliknite na odkaz a stiahnite si oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-15_LCD1602/15_LCD1602

Otvorte program kurzu v priečinku **„15_LCD1602“** pomocou Arduino IDE:

15_LCD1602.ino

```
/****************** LCD1602 + BH1750 ******************/
#include "Adafruit_LiquidCrystal.h"  // LCD I2C library
#include <BH1750.h>                  // Light sensor library
#include <Wire.h>                    // I2C communication

Adafruit_LiquidCrystal lcd(0);       // LCD (I2C address handled by library)
BH1750 lightMeter(0x5c);             // BH1750 light sensor address 0x5C

float lux = 0;                       // Light intensity value

void setup() {
  Serial.begin(115200);   // Serial for debugging
  Wire.begin();           // Start I2C bus

  /************* LCD Initialization *************/
  while (!lcd.begin(16, 2)) {        // Initialize LCD (16x2)
    Serial.println("LCD init failed! Check wiring.");
    delay(50);
  }
  Serial.println("LCD initialized.");
  lcd.setBacklight(1);               // Turn on LCD backlight
  lcd.clear();

  /************* BH1750 Initialization *************/
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
    Serial.println("BH1750 init success!");
  } else {
    Serial.println("Error initializing BH1750.");
    lcd.setCursor(0, 0);
    lcd.print("BH1750 ERROR!");
    while (1); // Stop program if sensor fails
  }

  lcd.setCursor(0, 0);
  lcd.print("Light Sensor OK");
  delay(1000);
  lcd.clear();
}

void loop() {
  if (lightMeter.measurementReady(true)) {
    lux = lightMeter.readLightLevel();   // Read light intensity

```

Vysvetlenie kľúčových kódov

(1) Použité súbory knižnice

```
#include "Adafruit_LiquidCrystal.h" #include
<BH1750.h>
#include <Wire.h>
```

- **#include** sa používa na načítanie externých knižníc (vopred napísaných funkcií a typov) do programu, podobne ako pri pridávaní doplnkov.
- **Adafruit_LiquidCrystal.h** – Metódy a rozhrania na ovládanie verzie I2C

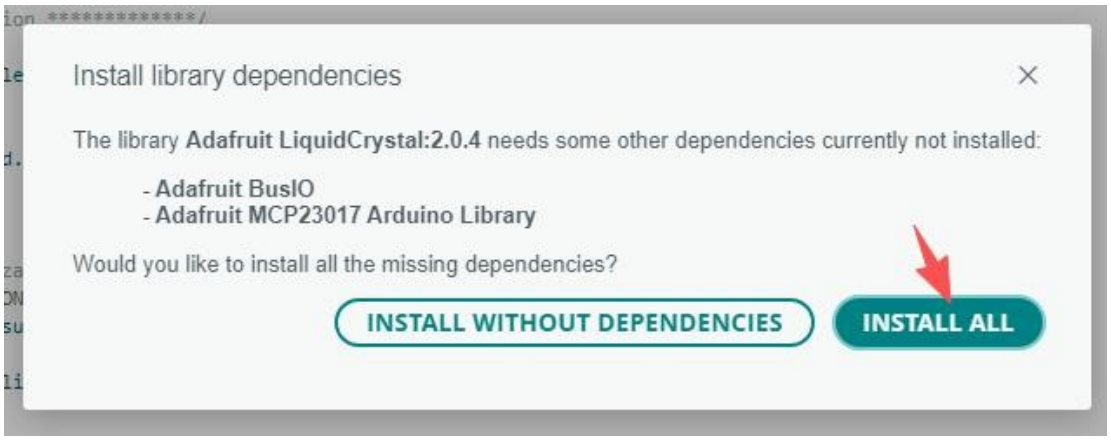
LCD1602 (znakový LCD).

- **BH1750.h** – Funkcie na interakciu so svetelným senzorom BH1750 (zahrňujúce príkazy senzora a namerané hodnoty).
- **Wire.h** – Vstavaná komunikačná knižnica I2C (dvojvodičové rozhranie) pre Arduino, implementujúca základné operácie čítania/zapisovania SDA/SCL.

➢ Stiahnutie v prostredí Arduino IDE

Verzia knižnice Adafruit LiquidCrystal, ktorú používame, je 2.0.4.

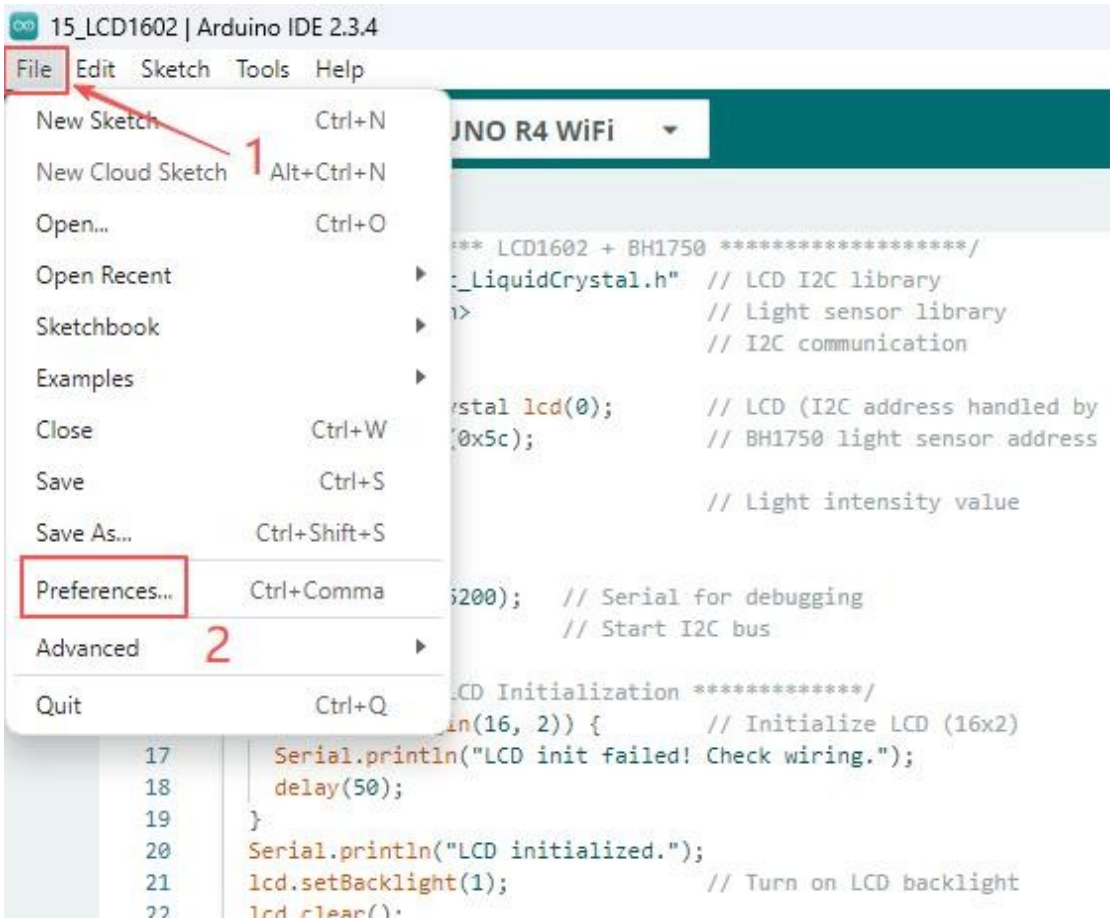
Inštaláciou **balíka Adafruit_LiquidCrystal** sa nainštalujú aj ďalšie potrebné závislosti. Na dokončenie inštalácie stačí kliknúť **na „Inštalovať všetko**“.



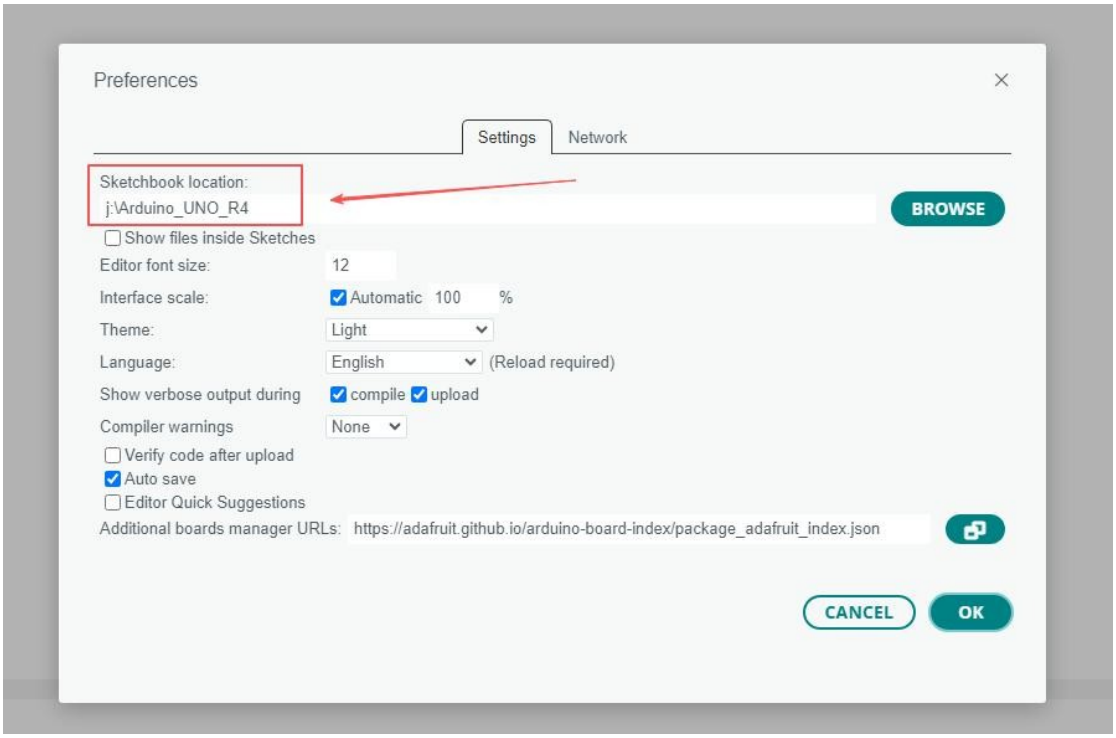
Po úspešnej inštalácii sa tieto súbory knižnice zobrazia v ceste ku knižnici, ktorú ste nastavili v prostredí Arduino IDE:

Ako skontrolovať umiestnenie vašich vlastných knižničných súborov:

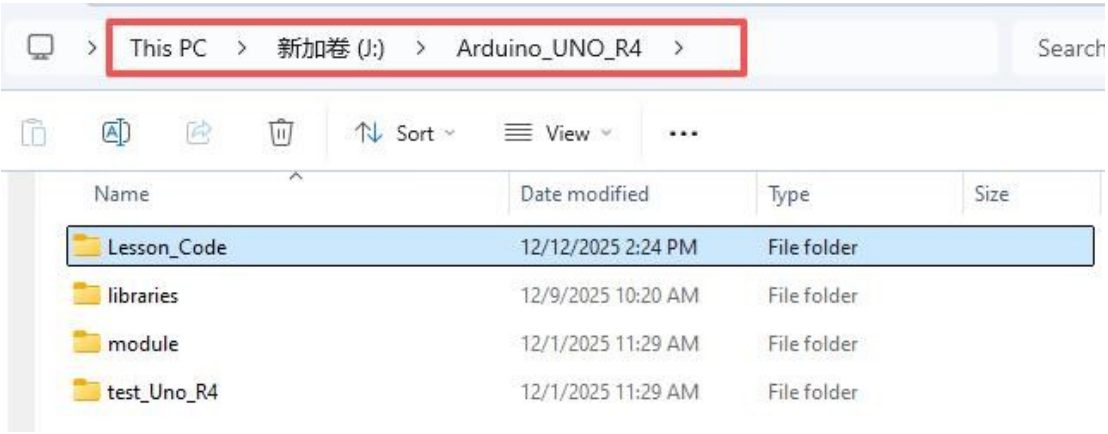
Kliknite na **File (Súbor)** v ľavom hornom rohu a potom vyberte **Preferences (Nastavenia)**



Nasledujúca cesta je **umiestnenie,** kde sú uložené vaše súbory knižníc. Nezabudnite si uchovať **svoju vlastnú verziu.**

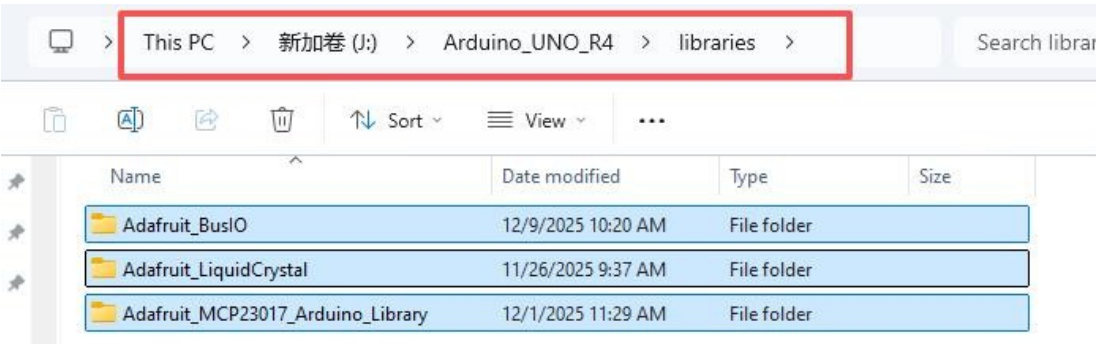


Potom prejdite **v súborovom systéme** na **túto cestu.**

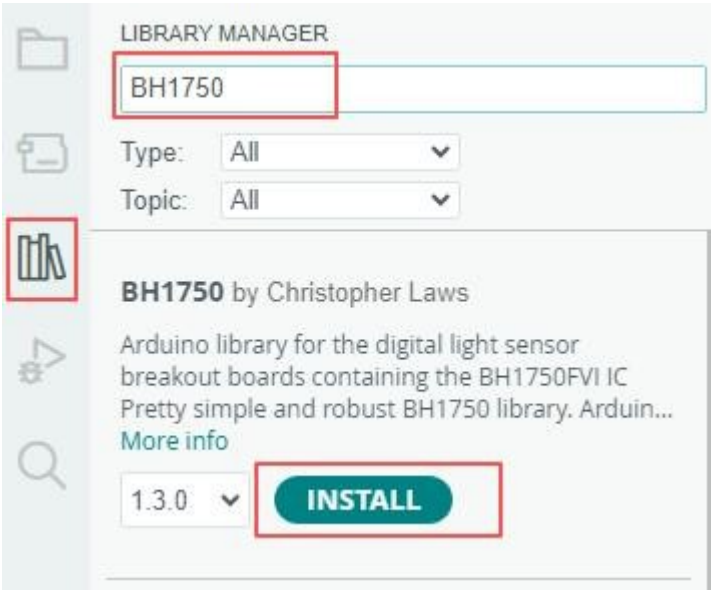


Vidíte, že moje súbory knižnice **„libraries“** sú umiestnené tu (môžete ich umiestniť kamkoľvek, kde to považujete za vhodné).

Po otvorení uvidíte, že súbory knižnice, ktoré sme práve stiahli, sú práve **tu**.



Podobne aj potrebný ovládač pre inštaláciu fotosenzora (verzia, ktorú sme použili, je 1.3.0 pre BH1750)



Prečo ich potrebujeme?

LCD aj **BH1750** komunikujú s Arduinom prostredníctvom protokolu I2C; knižnica Wire poskytuje zbernicu a ostatné dve knižnice ju využívajú na komunikáciu s konkrétnymi zariadeniami.

(2) Vytvorte objekty zariadení (inštanciujte a pomenujte periférie)

```
Adafruit_LiquidCrystal lcd(0);          // LCD (I2C adresa spravovaná knižnicou)
BH1750 lightMeter(0x5c);                // Adresa svetelného senzora BH1750 0x5C
```

Tieto dva riadky kódu v skutočnosti vytvárajú „softvérové agenty“ pre vaše hardvérové zariadenia.

Adafruit_LiquidCrystal lcd(0);

Táto veta vytvára objekt s názvom „lcd“, ktorý pochádza z triedy na ovládanie LCD od Adafruit a slúži konkrétne na riadenie prevádzky vášho displeja LCD1602. Napríklad na tlač textu: **lcd.print,** presun kurzora: **lcd.setCursor,** zapnutie/vypnutie podsvietenia: **lcd.setBacklight** atď.

Číslo **0** v zátvorkách znamená „nech knižnica automaticky vyhľadá a nájde adresu I2C LCD“, takže nemusíte ručne písať **0x27** alebo **0x3F**. Inými slovami, tento riadok kódu je ekvivalentný pokynu pre Arduino: „Prosím, priprav mi ovládač pre LCD obrazovku a automaticky mi pomôž ho nájsť.“

Druhý riadok, **„BH1750 lightMeter(0x5c);“,** vytvára objekt s názvom „lightMeter“ pre svetelný senzor BH1750. Tento objekt je špeciálne navrhnutý na zabezpečenie komunikácie so senzorom, čítanie úrovne osvetlenia a spracovanie technických detailov, ako sú režimy merania a prevodné vzorce.

Tu je **(0x5c)** skutočná adresa **BH1750** na zbernici I2C, ktorú musíte špecifikovať vy. Knižnica ju automaticky nevyhľadáva. Po vykonaní tohto riadku kódu bude mať Arduino **„digitálneho asistenta“**, ktorý dokáže komunikovať so senzorom. Odteraz stačí volať inštrukcie ako **„lightMeter.readLightLevel()“**, aby automaticky dokončil základnú hardvérovú komunikáciu.

Zhrnuté, tieto dva riadky kódu využívajú objektový mechanizmus C++ na vytvorenie „softvérových avatarov“ pre váš LCD displej a svetelný senzor. Všetky budúce operácie na hardvéri – či už ide o tlač textu alebo čítanie svetla – sú príkazy vydané týmto objektom a objekty automaticky za vás vybavia všetku zložitú základnú komunikáciu.

Z hľadiska hardvéru (čo je veľmi dôležité):

Na zbernici I2C má každé zariadenie svoju adresu (podobne ako číslo bytu). Ak majú dve zariadenia rovnakú adresu, dôjde ku konfliktu; ak je adresa nesprávna, program nebude schopný zariadenie nájsť.

(3) Premenné: Ukladá namerané hodnoty

```
float lux = 0;
```

float sa používa na ukladanie desatinných čísel, zatiaľ čo lux ukladá intenzitu osvetlenia (jednotka: lux, lx).

Prečo float?

BH1750 zvyčajne vráti hodnoty s desatinnými miestami (napríklad 123,45 lx), preto je potrebný typ float.

Poznámka: Operácie s plávajúcou desatinnou čiarkou v MCU (mikrokontroléri) sú pomalšie ako operácie s celými číslami a spotrebúvajú viac pamäte, ale na zobrazenie hodnôt zo senzora sú prijateľné.

(4) setup(): Zariadenie je potrebné inicializovať len raz

```
void setup() {

  Serial.begin(115200);          // Sériové rozhranie na
  ladenie
  Wire.begin();                  // Spustenie
                                 zbernice I2C


  /************* Inicializácia LCD *************/

  while (!lcd.begin(16, 2)) {                  // Inicializácia LCD
                                               (16x2)
    Serial.println("Inicializácia LCD zlyhala!
    Skontrolujte zapojenie.");

    delay(50);
  }
```

```
  Serial.println("LCD inicializované.");
  lcd.setBacklight(1);                    // Zapnúť podsvietenie
                                          LCD
  lcd.clear();


  /************* Inicializácia BH1750 *************/
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
    Serial.println("Inicializácia BH1750 úspešná!");
  } else {
    Serial.println("Chyba pri inicializácii BH1750.");
    lcd.setCursor(0, 0);
    lcd.print("CHYBA BH1750!");
    while (1); // Zastaviť program, ak senzor zlyhá
  }



  lcd.setCursor(0, 0);
  lcd.print("Svetelný senzor
  OK"); delay(1000);
  lcd.clear();
}
```

Funkcia **„setup()“** sa vykoná raz pri zapnutí napájania alebo resete a slúži na konfiguráciu sériového portu, zbernice I2C, obrazovky a senzorov.

① **Inicializácia sériového portu a zbernice I2C**

```
11  void setup() {
12    Serial.begin(115200);   // Serial for debugging
13    Wire.begin();           // Start I2C bus
```

- Funkcia: Inicializácia sériového portu (pre komunikáciu s počítačom). Informácie o ladení sa budú cez tento kanál odosielať do monitora sériového portu.

Pre**čo 115200? :** Rýchla a bežne používaná rýchlosť. Uistite sa, že monitor sériového portu je nastavený na rovnakú rýchlosť.

- **Funkcia**: Inicializuje zbernicu I2C pomocou dvoch vedení SDA a SCL. Pre väčšinu dosiek Arduino (Uno, Nano) možno použiť predvolené piny; pri niektorých doskách (ESP32) môže byť potrebný príkaz Wire.begin(SDA, SCL) na špecifikovanie pinov.

② **Inicializácia LCD**

```
15  /************* LCD Initialization *************/
16  while (!lcd.begin(16, 2)) {        // Initialize LCD (16x2)
17    Serial.println("LCD init failed! Check wiring.");
18    delay(50);
19  }
```

V tomto kóde sa program **v slučke while (!lcd.begin(16, 2))** neustále pokúša inicializovať LCD displej. Funkcia **lcd.begin(16, 2)** dáva knižnici LCD pokyn: „Toto je LCD displej s 16 stĺpcami a 2 riadkami. Prosím, začni pracovať.“ Za normálnych okolností táto funkcia vráti hodnotu true, čo znamená úspešnú inicializáciu; ak však nie je zapojenie správne, adresa I2C je nesprávna alebo displej nie je zapnutý, vráti hodnotu false. A symbol ! predstavuje „negáciu“, čo znamená „ak inicializácia zlyhá (vráti hodnotu false)“. Preto príkaz **„while (!lcd.begin(16, 2))“** znamená: „Kým sa LCD úspešne nespustí, zostanem tu a nebudem pokračovať ďalej.“

V rámci tejto slučky sa v monitore sériového portu bude nepretržite zobrazovať chybová správa **Serial.println("Inicializácia LCD zlyhala!** Skontrolujte zapojenie."), ktorá vás upozorní, aby ste skontrolovali zapojenie. Funkcia **delay(50)** spôsobuje 50-milisekundovú pauzu medzi každým opakovaným pokusom, čím zabraňuje nekonečnému rýchlemu obnovovaniu obrazovky.

Celkovo to možno chápať takto: Program čaká, kým LCD začne správne fungovať. Ak sa to nepodarí, bude vás na to upozorňovať, kým správne nepripojíte vedenie a modul sa nevráti do normálneho stavu; v tom momente bude pokračovať vo vykonávaní nasledujúceho kódu.

③ Zapnite podsvietenie, vymažte obrazovku a pripravte sa na zobrazenie

```
21  lcd.setBacklight(1);              // Turn on LCD backlight
22  lcd.clear();
```

V riadku **„lcd.setBacklight(1)“** program posiela LCD displeju pokyn: zapnúť podsvietenie.

Väčšina LCD modulov založených na I2C má ovládateľné podsvietenie, ktoré môže zlepšiť čitateľnosť obsahu obrazovky pri nedostatočnom osvetlení.

Číslo 1 v **„setBacklight(1)“** znamená „zapnuté“, zatiaľ čo 0 by znamenalo „vypnuté“.

Jej funkcia je teda celkom jednoduchá: rozsvietiť obrazovku, aby ste neskôr ľahšie videli zobrazený obsah.

Nasledujúca funkcia **„lcd.clear()“** vykonáva operáciu „vymazania obrazovky“. Keď je LCD zapnuté alebo reinicializované, môže si zachovať niektoré náhodné znaky alebo predchádzajúci obsah. Preto je vymazanie obrazovky veľmi bežnou operáciou pred oficiálnym zobrazením čohokoľvek. Funkcia **„lcd.clear()“** vymaže všetky dva riadky a 16 stĺpcov LCD a presunie kurzor späť do ľavého horného rohu (pozícia 0,0) obrazovky, čím zabezpečí, že text, ktorý sa chystáte

chystáte napísať, začína z pôvodnej pozície.

④ Inicializácia BH1750

```
24  /************* BH1750 Initialization *************/
25  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
26    Serial.println("BH1750 init success!");
27  } else {
28    Serial.println("Error initializing BH1750.");
29    lcd.setCursor(0, 0);
30    lcd.print("BH1750 ERROR!");
31    while (1); // Stop program if sensor fails
32  }
```

Hlavnou úlohou tohto kódu je na začiatku programu overiť, či bol svetelný senzor BH1750 úspešne aktivovaný a či funguje správne. **lightMeter.begin(...)** Ide o inicializačnú funkciu poskytovanú knižnicou **BH1750**. Pokúša sa nadviazať komunikáciu I2C so senzorom, informovať ho o prevádzkovom režime (v tomto prípade o režime nepretržitého

režim **s vysokým rozlíšením CONTINUOUS_HIGH_RES_MODE),** jeho adresu (**0x5C**) a akú zbernicu I2C má použiť (&Wire).

Ak senzor funguje správne, táto funkcia vráti hodnotu true a program vykoná **Serial.println("Inicializácia BH1750 úspešná!")** a bude pokračovať ďalej; ak však vráti hodnotu false, znamená to, že Arduino sa s senzorom úspešne nekomunikovalo, čo môže byť spôsobené nesprávnym zapojením, vypnutým modulom, nesprávnou adresou I2C alebo poškodením samotného zariadenia.

V tomto prípade program okamžite vypíše na LCD displej **„BH1750 ERROR!“** a vstúpi do „mŕtvej slučky“ **(while (1);)**. Účelom mŕtvej slučky nie je zdržiavať vás, ale zabrániť ďalšiemu behu celého programu, keďže pokračovanie vo vykonávaní by bolo bezvýznamné, ak senzor nebol úspešne inicializovaný, a mohlo by to viesť k následným chybám pri čítaní alebo dokonca ovplyvniť vaše ladenie.

⑤ Inicializácia fototranzistorového senzora bola úspešná

```
34    lcd.setCursor(0, 0);
35    lcd.print("Light Sensor OK");
36    delay(1000);
37    lcd.clear();
38  }
```

Po úspešnej inicializácii umiestnite kurzor na začiatok obrazovky LCD1602, potom vypíšte **text „Light Sensor OK“**, počkajte jednu sekundu, vymažte obrazovku a počkajte, kým sa zobrazia nasledujúce údaje zo senzora osvetlenia.

(5) loop

```
void loop() {
  if (lightMeter.measurementReady(true)) {
    lux = lightMeter.readLightLevel();          // Prečítať intenzitu
    osvetlenia

    /************* Výstup sériového portu *************/
    Serial.print("Svetlo: ");
    Serial.print(lux); Serial.println("
    lx");

    /************* Výstup na LCD *************/
    lcd.setCursor(0, 0);
    lcd.print("Svetlo:                ");      // Vymazať staré znaky
    lcd.setCursor(7, 0);
    lcd.print(lux);

    lcd.setCursor(0, 1);
    lcd.print("Jednotka:           ");
    lux

    delay(200); // Zabránenie blikaniu
    obrazovky
  }
}
```

Funkcia tejto slučky () je umožniť Arduinu nepretržite monitorovať intenzitu osvetlenia v slučke a súčasne zobrazovať namerané hodnoty na sériovom monitore aj na LCD1602.

①lightMeter.measurementReady(true)

```
40    void loop() {
41      if (lightMeter.measurementReady(true)) {
```

Program najprv pomocou **funkcie lightMeter.measurementReady(true)** zistí, či senzor **BH1750** dokončil nový súbor výsledkov merania (môžete si to predstaviť tak, že „senzor oznamuje: Zmeral som intenzitu osvetlenia, teraz si môžeš údaje prečítať“). Len ak je výsledok pravdivý, vykoná sa nasledujúci kód, aby sa zabránilo čítaniu neúplných údajov.

②lux = lightMeter.readLightLevel();

```
42        lux = lightMeter.readLightLevel();   // Read light intensity
43
```

Skutočne prečíta hodnotu osvetlenia vrátenú **BH1750** (v jednotkách lx) a uloží ju do premennej „lux“ (ako hodnotu s plávajúcou desatinnou čiarkou).

Knižnica pošle zariadeniu príkaz na čítanie, získa hodnotu z registra **cez I2C** do MCU, vykoná potrebnú konverziu jednotiek (niekedy konverziu a násobenie koeficientom) a nakoniec vráti hodnotu s plávajúcou desatinnou čiarkou.

③ **Tlač cez sériový port**

```
44      /************* Serial port output *************/
45      Serial.print("Light: ");
46      Serial.print(lux);
47      Serial.println(" lx");
```

Tlač cez sériový port je najčastejšie používanou metódou ladenia. Umožňuje zobraziť hodnoty zo senzorov na počítači a posúdiť, či sú primerané (napríklad v noci by nemala byť hodnota 5000 lx).

④ **LCD displej: polohovanie, vymazanie, zápis**

```
49      /************* LCD output *************/
50      lcd.setCursor(0, 0);
51      lcd.print("Light:        ");   // Clear old characters
52      lcd.setCursor(7, 0);
53      lcd.print(lux);
54
55      lcd.setCursor(0, 1);
56      lcd.print("Unit: lux     ");
```

- **lcd.setCursor(col, row):** Presunie kurzor na zadaný stĺpec a riadok (stĺpce začínajú od 0, riadky od 0 alebo 1).
- **lcd.print("Light:"):** Najprv vytlačí „Light:“, potom vyplní medzery, aby prepisal akýkoľvek predchádzajúci dlhší reťazec (znaková obrazovka automaticky nevymaže zostávajúce znaky; je potrebné ručné prepísanie).
- LCD je obrazovka založená na znakoch (nie na pixelovom plátne). Keď napíšete krátky reťazec a ten prepíše koniec dlhšieho reťazca, staré znaky zostanú zachované. Preto je bežnou technikou prekrytie medzerami.
- **lcd.setCursor(7, 0);** lcd.print(lux); Vytlačte hodnoty počnúc 7. stĺpcom (slovo „Light:“ zaberá 6 znakov, index stĺpca začína od 0 a 7. stĺpec bezprostredne nasleduje po „Light:“).
- **lcd.setCursor(0,1);** lcd.print("Unit: lux "); Druhý riadok zobrazuje jednotku a vyplní prázdne riadky na prepísanie predchádzajúceho obsahu.

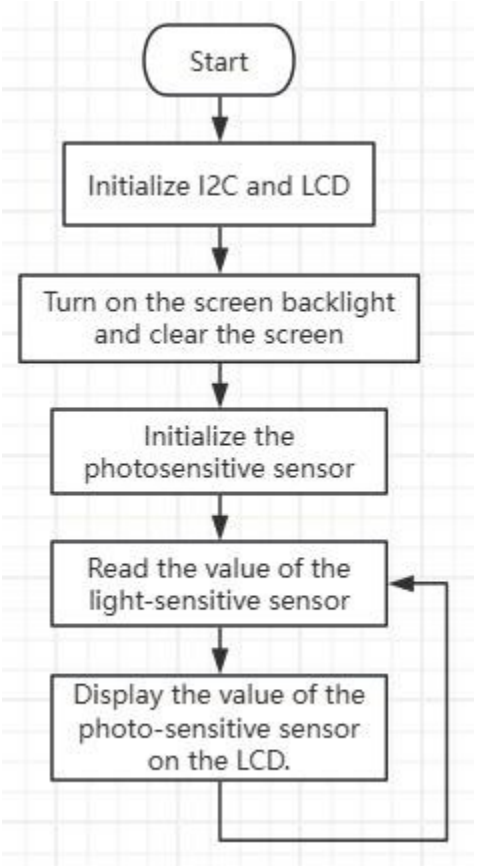
(6) delay(200)

```
58      delay(200); // Prevent screen flickering
59    }
60  }
61
```

Pozastaviť na **200** milisekúnd s cieľom:

- Zabrániť príliš rýchlemu blikaniu obrazovky
- Znížiť frekvenciu prístupu I2C a poskytnúť BH1750 čas na vzorkovanie

(7) Logický tok kódu



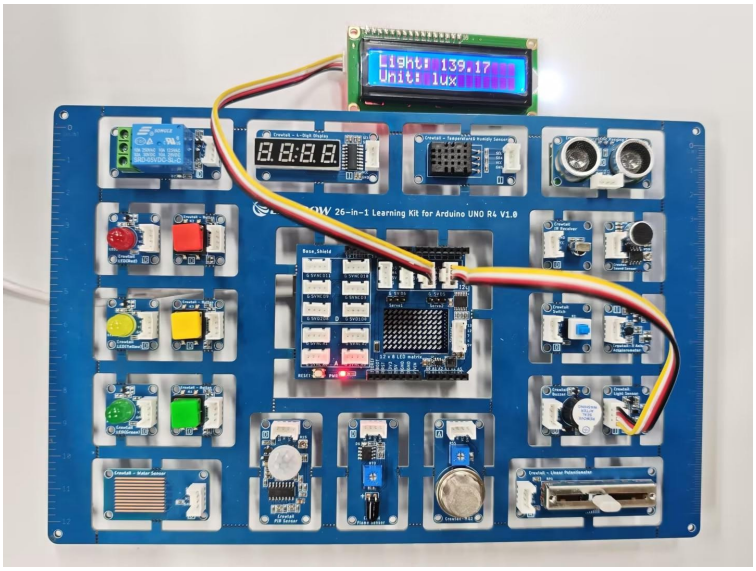
5. Stiahnite si program a sledujte výsledok

(1) Postupujte podľa krokov obsluhy Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“:**

Displej LCD1602 zobrazuje v prvom riadku v reálnom čase text **„Osvetlenie: + konkrétna hodnota intenzity osvetlenia**“ a

„Jednotka: lux“ sa zobrazuje stále v druhom riadku.



Lekcia 16 – Ultrazvukový snímač vzdialenosti

Úvod

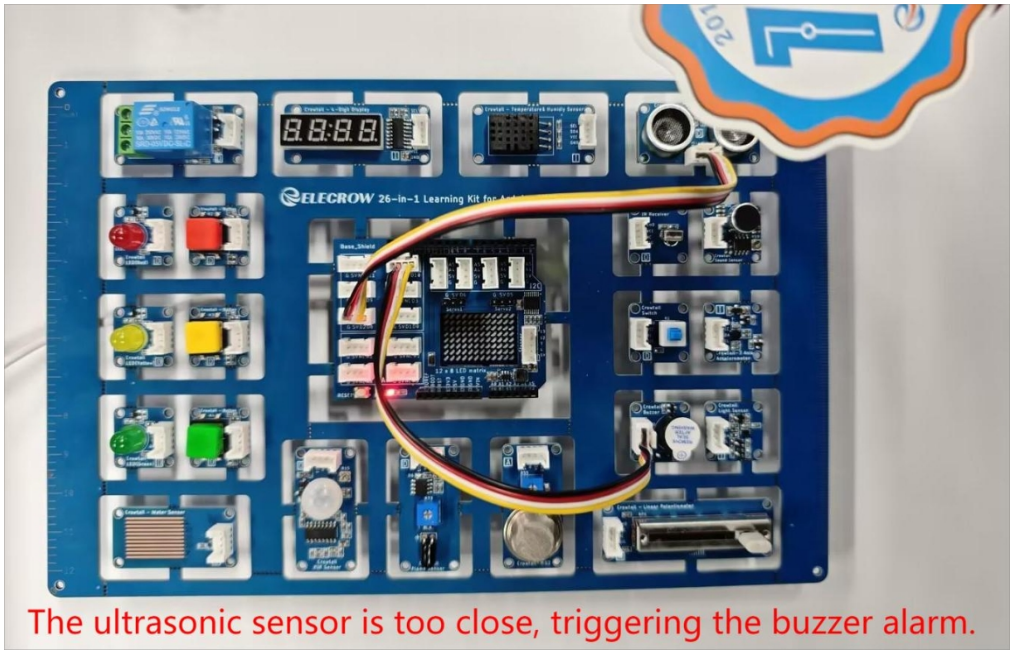
V tejto lekcii sa naučíme, ako používať ultrazvukový senzor vzdialenosti na meranie vzdialenosti medzi objektom a senzorom a ako využiť výsledky merania na ovládanie bzučiaka, aby sme dosiahli funkciu „alarmu pri príliš malej vzdialenosti“. Prostredníctvom malého príkladu „alarmu priblíženia“ v tejto lekcii sa naučíte čítať hodnoty ultrazvukového senzora v Arduine, určiť rozsah vzdialenosti a ovládať bzučiak podľa tohto rozsahu.

Ciele výučby

1. Porozumejte princípu fungovania modulu na meranie vzdialenosti pomocou ultrazvuku.
2. Ovládnite používanie knižnice HCSR04 a naučte sa čítať hodnoty vzdialenosti s presnosťou na centimetre.
3. Upevnite si používanie podmienených príkazov if.
4. Dokončite prípad „Systém blízkeho poplachu“

Náhľad výsledku

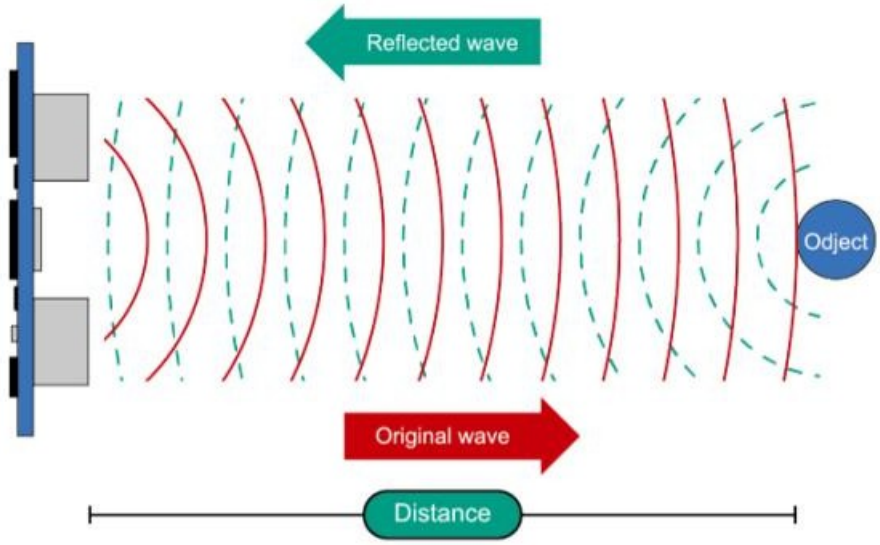
Keď je vzdialenosť od ultrazvukového senzora menšia ako 15 cm, ozve sa bzučiak.



1. Vysvetlenie princípu

Ultrazvukový senzor má dva vývody: jeden je **TRIG**, čo je vysielacia časť ultrazvukového senzora, a druhý je **ECHO**, čo je prijímacia časť ultrazvukového senzora. Vysiela

Ultrazvukový impulzný signál sa vysiela cez **TRIG**, **ECHO** prijíma ultrazvukový impulzný signál a potom sa ultrazvuková vlna prevádza na elektrický signál prostredníctvom prevodníka.



Ultrazvukový senzor vysiela ultrazvukový impulzný signál z vysielacej strany, ktorý sa odráža od objektu a potom sa vracia na prijímaciu stranu. Meria sa čas potrebný na prechod ultrazvukového impulzu od vyslania po prijatie. Podľa rýchlosti zvuku v prostredí je možné získať vzdialenosť od ultrazvukového senzora k objektu. Výpočtový vzorec je nasledovný:

Unified unit:

340 m/s=34000 cm/s

Algorithm is derived:

S=V*T=34000*T/2=17000*T

(**Note**: T is the time required for pulse return and return, which should be divided by 2 in calculation)

Zhrnutie základných princípov

- **Spúšť:** Spúšť
- **Odzvuk:** Príjem
- Vzdialenosť sa vypočíta na základe doby šírenia zvuku
- Čím je vzdialenosť menšia, tým je ozvena rýchlejšia a čas ozveny kratší
- Čím je vzdialenosť väčšia, tým je čas dlhší

Konštrukcia a popis vývodov ultrazvukového senzora

Typický modul HC-SR04:

- VCC: napájanie 5 V
- GND: uzemnenie
- TRIG: Spúšťač (Arduino vysiela impulz)
- ECHO: Echo (modul vysiela signál s vysokou úrovňou)

2. Potrebné moduly

Crowtail HC-SR04 (ultrazvukový senzor na meranie vzdialenosti ×1)

Bzučiak Crowtail (1 ks)

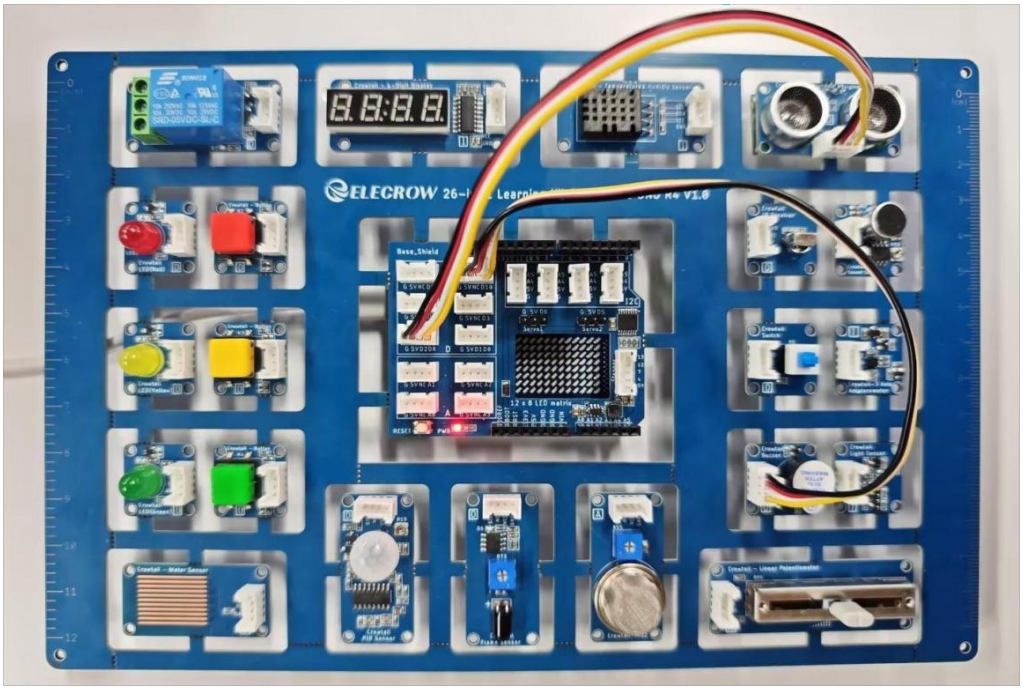
3. Spôsob zapojenia

Ultrazvukový snímač vzdialenosti → Digitálne porty D2 a D8

- VCC → 5 V
- GND → GND
- TRIG → **D8**

- ECHO → **D2**

Bzučiak Crowtail → Digitálny port D10



4. Vysvetlenie príkladu

Kliknite na odkaz a stiahnite si oficiálny vzorový kód:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-16_UltrasonicRanging/16_UltrasonicRanging

Otvorte program pre túto lekciu v priečinku **„16_UltrasonicRanging“** pomocou Arduino IDE:

```ino
#include <HCSR04.h>

// Ultrasonic sensor pins
const byte triggerPin = 8;
const byte echoPin = 2;

// Buzzer pin
#define BUZZER 10

UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);

// Alarm threshold (unit: cm)
const int dangerDistance = 15;

void setup()
{
  Serial.begin(115200);
  pinMode(BUZZER, OUTPUT);
  digitalWrite(BUZZER, LOW);
}

void loop()
{
  float distance = distanceSensor.measureDistanceCm();

  Serial.print((int)distance);
  Serial.println(" cm");

  // If the distance is too close → alarm
  if (distance > 0 && distance < dangerDistance) {
    digitalWrite(BUZZER, HIGH);   // Turn buzzer ON
  } else {
    digitalWrite(BUZZER, LOW);    // Turn buzzer OFF
  }

  delay(500);
}
```

Vysvetlenie kľúčových kódov

(1) Importovanie knižníc

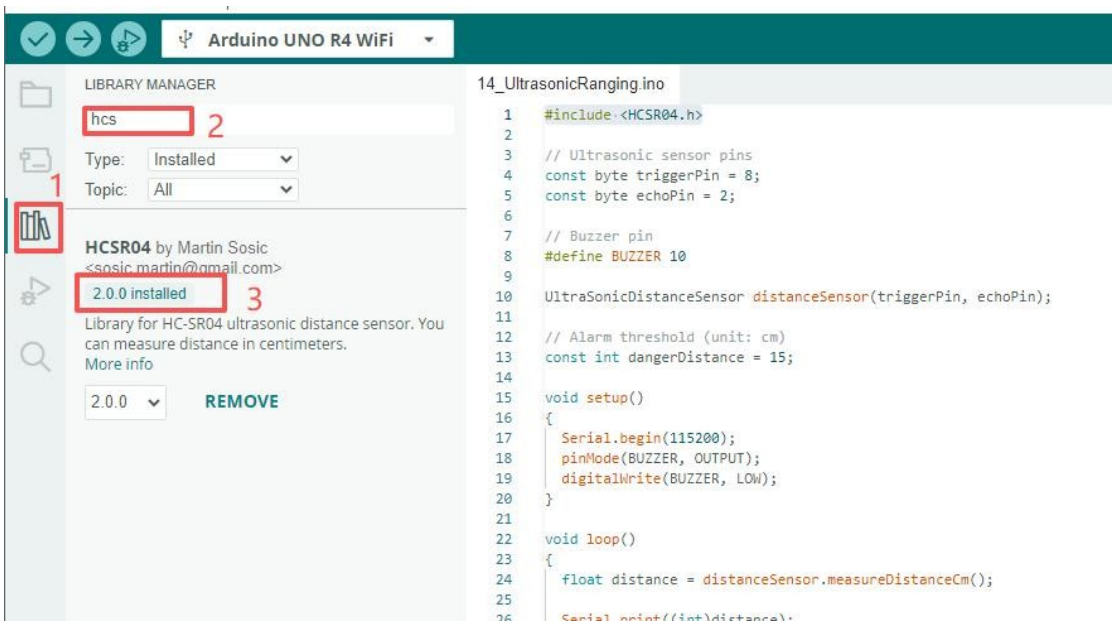
```
#include <HCSR04.h>
```

Na začiatku programu sa nachádza riadok: **#include <HCSR04.h>**.

- Tento riadok znamená „načítať“ existujúce funkcie a objekty z knižnice, aby ich program mohol priamo používať.
- Základný proces ultrazvukového merania vzdialenosti zahŕňa vyslanie krátkeho impulzu, meranie času, počas ktorého zostáva echo na pinu **ECHO** na úrovni HIGH, a jeho prevod na vzdialenosť na základe rýchlosti zvuku;

➢ Stiahnite si ho v prostredí Arduino IDE

Verzia HCSR04, ktorú tu používame, je 2.0.0



(2) Definície pinov (triggerPin, echoPin) a výber typu

```
const byte triggerPin = 8; const
byte echoPin = 2;
// Pin bzučiaka
#define BUZZER 10
```

- V programe sú spúšťací pin a echo pin pre ultrazvukové vlny definované ako: **const byte triggerPin = 8;** a **const byte echoPin = 2;** Tu sa „const“ používa na označenie „konštanty“ a hodnota sa počas behu programu nemení. Toto bolo podrobne vysvetlené v predchádzajúcich kurzoch. „byte“ je typ celého čísla, ktorý zaberá **1 bajt** a šetrí pamäť, čo znamená, že tieto dve premenné predstavujú čísla pinov.
- Význam a účel týchto dvoch riadkov je veľmi jednoduchý: oznámiť programu , že **Trig** je pripojený k **D8** a **Echo** je pripojený k **D2.** Výhodou zapísania pinov na začiatku je, že ak budete v budúcnosti potrebovať zmeniť zapojenie, stačí upraviť len tieto dva riadky. Ostatné časti môžu priamo odkazovať na tieto dva názvy, čím sa kód stáva čitateľnejším a ľahšie udržiavateľným.
- Použitím **#define BUZZER 10** sa pin bzučiaka definuje ako **D10**. Na zmenu konfigurácie pinov možno použiť ako definície makier, tak aj const
môžu byť použité na zmenu konfigurovateľnosti pinov. Makrá sú nahradené konkrétnymi číslami počas fázy predspracovania. Voľba použitia makier je tu hlavne kvôli zvyku a efekt je podobný ako pri const. Ak budete chcieť v budúcnosti nahradiť bzučiak D9, stačí len upraviť tento riadok.

(3) Vytvorte objekt senzora vzdialenosti

```
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);
```

UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin); Tento riadok vytvorí „objekt senzora“ na základe knižnice. Senzor si predstavte ako objekt s názvom „distanceSensor“, ktorý má funkciu „merania vzdialenosti“. Po zadaní pinov spúšťača a odozvy knižnica

interné nastavenie režimov týchto pinov a pripraví sa na vykonanie merania. Zapuzdrenie objektu robí následné volania intuitívnejšími: stačí zavolať **distanceSensor.measureDistanceCm(),** aby ste získali hodnotu vzdialenosti. To je výhoda priameho používania knižnice **HCSR04**.

(4) Definujte prahovú hodnotu alarmu dangerDistance

```
const int dangerDistance = 15;
```

Nastavte „nebezpečnú vzdialenosť“ na **15 cm**. Táto premenná označuje, že alarm by sa mal spustiť, keď je nameraná vzdialenosť menšia ako táto hodnota, čo sa považuje za **„príliš blízko**“.

Výhodou toho, že je táto hodnota navrhnutá ako konštanta a definovaná na začiatku, je to, že je pohodlné prispôsobovať citlivosť počas samotného merania: ak chcete, aby bola citlivejšia, znížte túto hodnotu; ak chcete, aby bola „tolerantnejšia“, zvýšte túto hodnotu.

(5) setup(): Inicializácia sériového portu a bzučiaka

```
void setup()
{
   Serial.begin(115200);
   pinMode(BUZZER, OUTPUT);
   digitalWrite(BUZZER, LOW);
}
```

Funkcia **„setup**()“ sa vykoná len raz pri zapnutí alebo resete programu Arduino.

- **„Serial.begin(115200);“:** Otvorí sériový monitor a nastaví prenosovú rýchlosť na 115200. Sériový port sa používa na odosielanie nameranej vzdialenosti alebo informácií o ladení do počítača, čo uľahčuje sledovanie prevádzkového stavu programu a hodnôt snímačov. Poznámka: Prenosová rýchlosť sériového monitora musí byť zhodná s touto hodnotou; inak bude výstup skreslený alebo informácie nebudú viditeľné.
- **„pinMode(BUZZER, OUTPUT);“:** Nastaví pin bzučiaka do výstupného režimu, čo znamená, že bzučiak budeme ovládať v kóde pomocou funkcie „digitalWrite“, ktorou na tento pin vysielame signál.

- **"digitalWrite(BUZZER, LOW);":** V predvolenom nastavení je bzučiak pri zapnutí vypnutý, aby sa zabránilo falošným poplachom spôsobeným momentálnym zapnutím, čo by viedlo k nepríjemným zážitkom.

(6) "loop()": Jadro merania a logika posudzovania

"loop()" je hlavná slučka programu a bude sa vykonávať opakovane na neurčito. Hlavnú logiku tohto programu možno rozdeliť do štyroch krokov: meranie vzdialenosti → tlač → posúdenie → ovládanie bzučiaka. Nasledujúci text podrobne vysvetľuje každý krok.

① **Meranie vzdialenosti (measureDistanceCm)**

```
void loop()
{
    float vzdialenosť = distanceSensor.measureDistanceCm();
```

Tento riadok kódu volá funkciu knižnice a vráti číslo **s pohyblivou desatinnou čiarkou** (jednotka: centimetre). Výhodou použitia typu float je, že zachováva desatinnú presnosť (napríklad 12,4 cm), čo uľahčuje presnejšie zobrazenie a podrobnejšie posúdenie.

Je potrebné, aby všetci pochopili rozdiel medzi číslami **s pohyblivou desatinnou čiarkou** a celými číslami: Čísla s pohyblivou desatinnou čiarkou zaberajú viac pamäte, ale môžu reprezentovať desatinné čísla a zvyčajne sa používajú pre údaje súvisiace s meraním.

Zároveň je potrebné zdôrazniť, že rôzne knižnice majú rôzne návratové hodnoty v prípade zlyhania merania (napríklad žiadna odozva alebo prekročenie rozsahu), niektoré vracajú 0, iné -1 a ďalšie maximálnu hodnotu. Preto pri písaní podmienkových príkazov je potrebné vykonať filtrovanie (v tomto programe sa používa distance > 0 na prvotné odfiltrovanie 0 alebo abnormálnych hodnôt).

② **Sériové tlačenie (Serial.print / println)**

```
    Serial.print((int)distance);
    Serial.println(" cm");
```

Najprv prevádzajte vzdialenosť na celé číslo a vytlačte ju, a potom na koniec pripojte jednotku „cm“.

Dôvod nútenej **konverzie „distance“ na (int)**: na jednej strane to zjednodušuje výstup sériového portu (bez tlače príliš veľa desatinných miest); na druhej strane to môže všetkým pripomenúť, že „keď presnosť nie je kľúčová, čísla s pohyblivou desatinnou čiarkou sa často prevádzajú na celé čísla pre účely čítania a ukladania“. Zároveň vysvetlite riziká núteného prevodu: napríklad 14,9 sa prevedie na **14** (**skrátenie**), čo môže ovplyvniť posúdenie kritického prahu. Preto pri skutočnom posudzovaní tu nepoužívame (int), ale priamo používame premennú typu float „distance“ na logické porovnanie (toto je metóda použitá v programe).

③ **Posúdenie a ovládanie bzučiaka (if / else)**

```
    if (distance > 0 && distance < dangerDistance) {
        digitalWrite(BUZZER, HIGH);            // Zapnúť bzučiak
    } else {
        digitalWrite(BUZZER, LOW);             // Vypnúť bzučiak
    }
    delay(500);
}
```

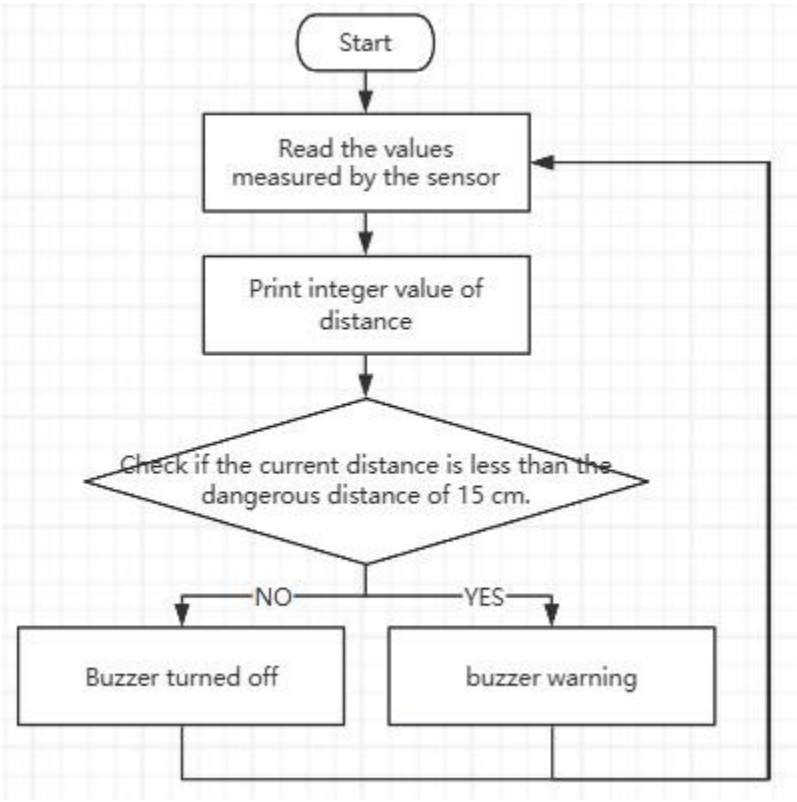
Toto je základná logika programu, vysvetlená bod po bode:

- **distance > 0:** Ide o kontrolu ochranných podmienok, ktorá slúži na vylúčenie neplatných hodnôt alebo chýb merania. V niektorých prípadoch môže funkcia merania vzdialenosti vrátiť hodnotu 0 (označujúcu, že nebolo zaznamenané žiadne echo) alebo zápornú hodnotu (označujúcu chybu), pričom priame porovnanie s prahovou hodnotou by viedlo k falošnému spusteniu. Tým, že sa najskôr overí, či je vzdialenosť kladná, sa zabráni tomu, aby sa situácia bez echa považovala za situáciu „veľmi blízko“.
- **vzdialenosť < dangerDistance:** Ak je nameraná vzdialenosť menšia ako nastavená prahová hodnota nebezpečenstva (v tejto lekcii je to štandardne 15 cm), považuje sa to za vstup do nebezpečného rozsahu.
- Ak sú podmienky splnené, použite **digitalWrite(BUZZER, HIGH)** na nastavenie bzučiaka na vysokú úroveň, t. j. na spustenie alarmu; v opačnom prípade použite **digitalWrite(BUZZER, LOW)** na vypnutie bzučiak.

Dôležité doplňujúce vysvetlenie:

- Keďže funkcia loop() sa vykonáva každých 500 ms s **delay(500)**, táto logika bude kontrolovať vzdialenosť každých **0,5 sekundy**. Zvuk bude „nepretržitý“ alebo „prerušovaný“ v závislosti od samotného bzučiaka a frekvencie posudzovania.
- Vysvetlenie skratovej charakteristiky operátora && (logické AND): Ak podmienka na ľavej strane, **distance > 0**, nie je splnená, pravá strana, **distance < dangerDistance**, nebude vyhodnotená. Týmto sa dá zabrániť nešpecifikovanému správaniu alebo poplachom, keď má distance abnormálnu hodnotu.

(7) Logický tok kódu



5. Spustite program a sledujte výsledky

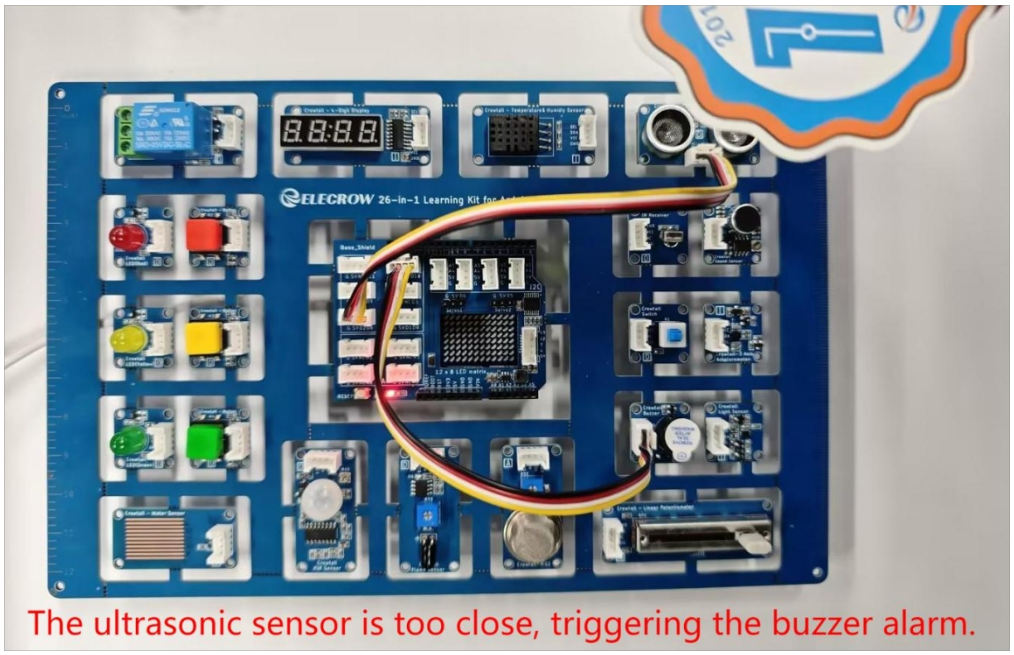
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“:**

(2) Po úspešnom stiahnutí skontrolujte výsledok:

Otvorte sériový monitor integrovaný v prostredí Arduino IDE

(V tomto momente sa pohybujte hore a dole nad ultrazvukovým senzorom. Keď sa priblížite k ultrazvukovému senzoru a vzdialenosť bude menšia ako **15 cm**, bzučiak spustí alarm)



Lekcia 17 – DHT20

Úvod

V tejto lekcii sa naučíme, ako skombinovať senzor teploty a vlhkosti (DHT20) a bzučiak, aby sme vytvorili inteligentný systém monitorovania prostredia s alarmom. Vďaka tejto lekcii získate odborné znalosti, napríklad ako zbierať údaje o teplote a vlhkosti a ako spolupracuje viacero modulov, a pochopíte, aké sú princípy alarmov mnohých systémov regulácie teploty na trhu.

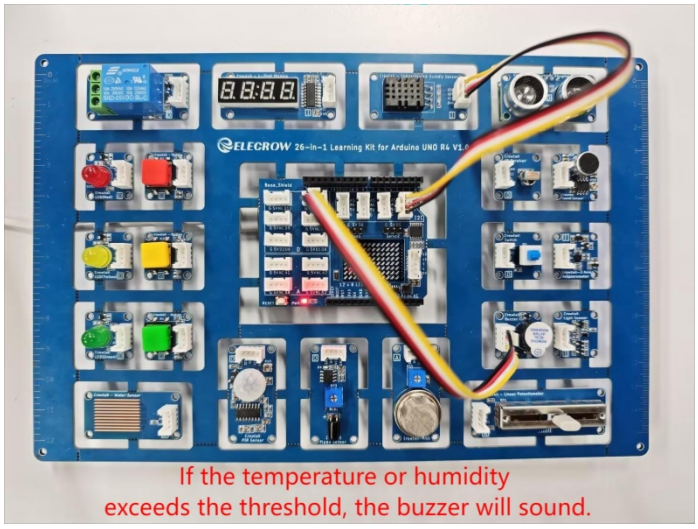
Ciele

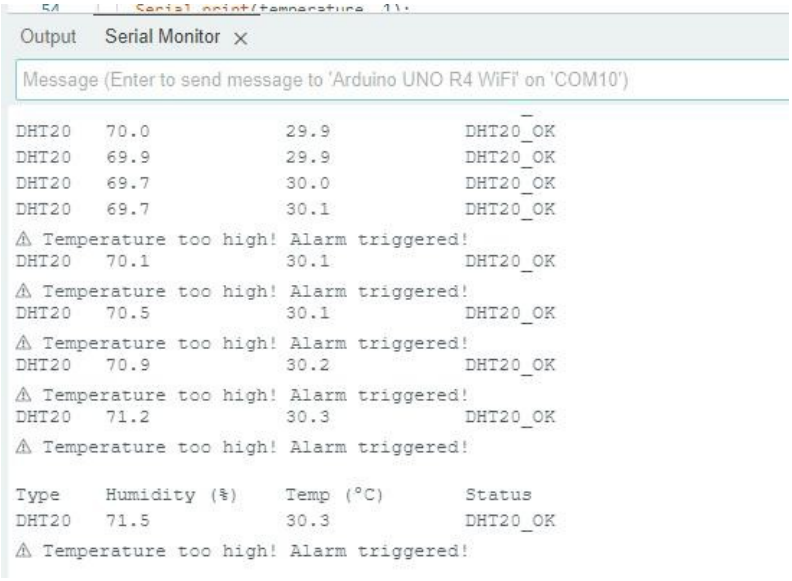
1. Porozumieť princípu fungovania senzora teploty a vlhkosti.
2. Naučte sa používať rozhrania funkcií v knižnici tretej strany (DHT20) na inicializáciu snímača teploty a vlhkosti a na čítanie príslušných informácií.
3. Zoznámte sa s tým, ako použiť výsledok priradenia booleovskej premennej „alarm = true“ ako základ pre ovládanie modulu bzučiaka.
4. Dokončite tento jednoduchý prípad „Inteligentný systém monitorovania prostredia s alarmom“

Náhľad výsledku

Systém nepretržite monitoruje teplotu a vlhkosť prostredia. Keď teplota prekročí 30 °C alebo vlhkosť klesne pod 40 %, bzučiak vydáva nepretržitý alarmový zvuk.

Zároveň sa informácie o poplachu zobrazia na počítači.



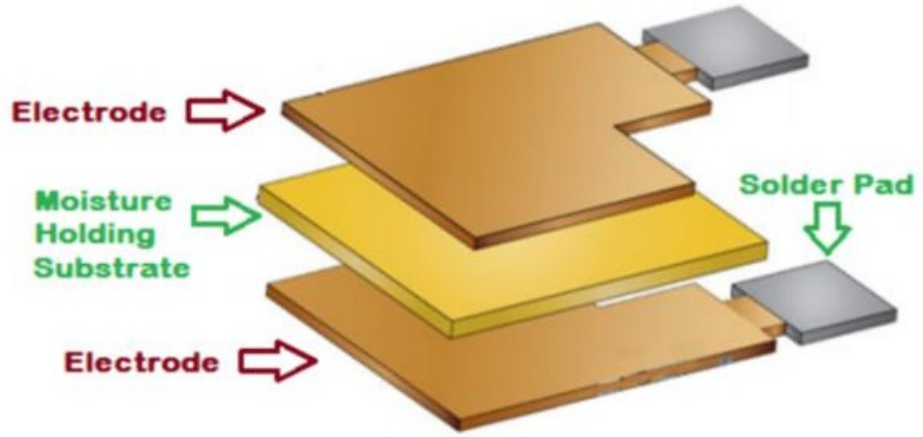


1. Vysvetlenie princípu

Základný princíp merania vlhkosti senzora vlhkosti a teploty DHT20 je založený na kapacitnej štruktúre „elektróda + substrát na adsorpciu vlhkosti“ (ako je znázornené na obrázku, ktorý ste poskytli: horná a dolná elektróda sú vložené medzi porézny substrát, ktorý môže adsorbovať a uvoľňovať vodnú paru) – keď sa zmení vlhkosť okolia, zmení sa množstvo vodnej pary adsorbovanej/uvoľnenej substrátom, čo spôsobí zmenu dielektrickej konštanty dielektrickej vrstvy, a tým dôjde k zodpovedajúcej zmene hodnoty kapacity medzi elektródami; zároveň DHT20 integruje aj nezávislý teplotný snímač (napr. termistor), ktorý dokáže snímať kolísanie fyzikálnej veličiny (odpor/napätie) spôsobené zmenami teploty. DHT20 nie je základný senzor; má zabudovaný špecializovaný čip ASIC: najprv prevádza zmeny fyzikálnych veličín „kapacitance (odpovedajúcej vlhkosti), termistorového prvku (odpovedajúceho teplote)“ na napäťové signály, potom prostredníctvom analógovo-digitálneho prevodu získava digitálne

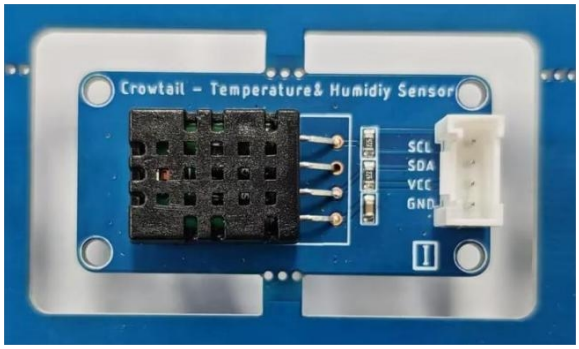
signály a nakoniec vypočíta presnú relatívnu vlhkosť (%) RH a teplotu (℃) na základe algoritmus továrenského kalibrovania a nakoniec prenáša digitálne údaje o teplote a vlhkosti na hlavnú riadiacu dosku prostredníctvom zbernice I2C, čím sa dosiahne automatický prevod „fyzikálnej

veličina → elektrický signál → digitálne údaje“ a vyznačuje sa nízkou spotrebou spotreby a vysokej presnosti (vlhkosť ±2 % RH, teplota ±0,5 ℃).



2. Potrebné moduly

Modul snímača teploty a vlhkosti Crowtail × 1

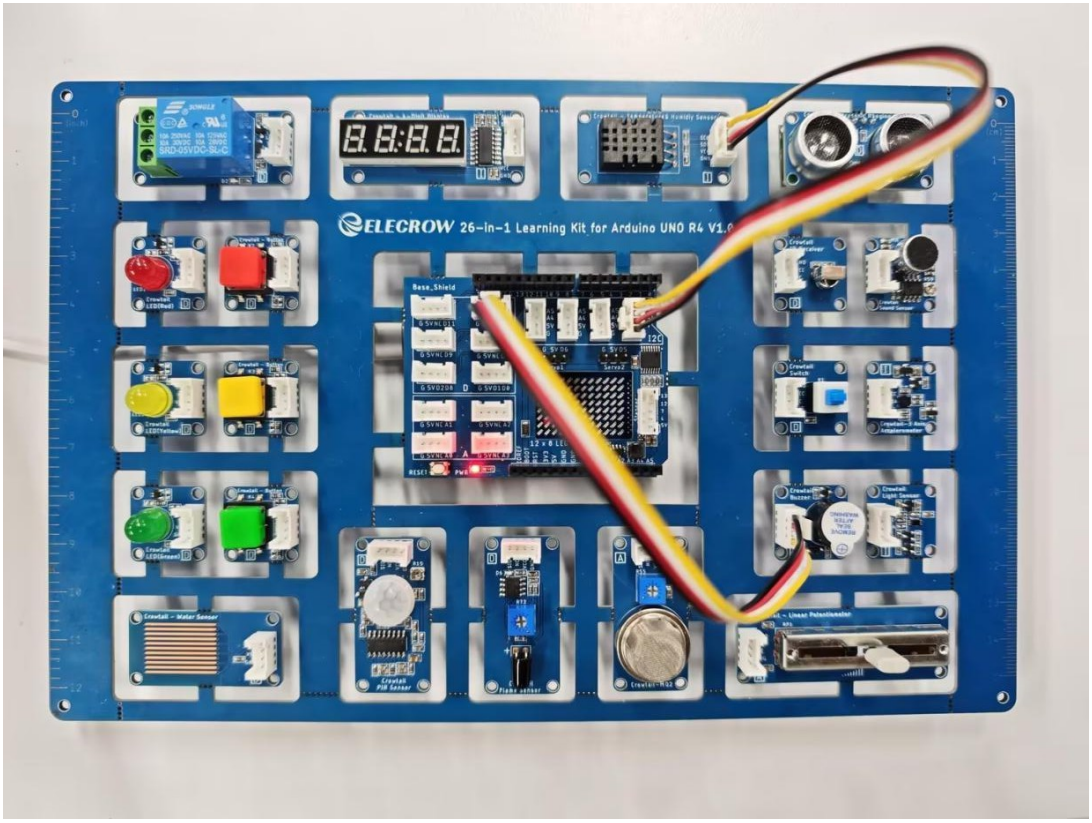


Modul bzučiaka Crowtail × 1

3. Spôsob zapojenia

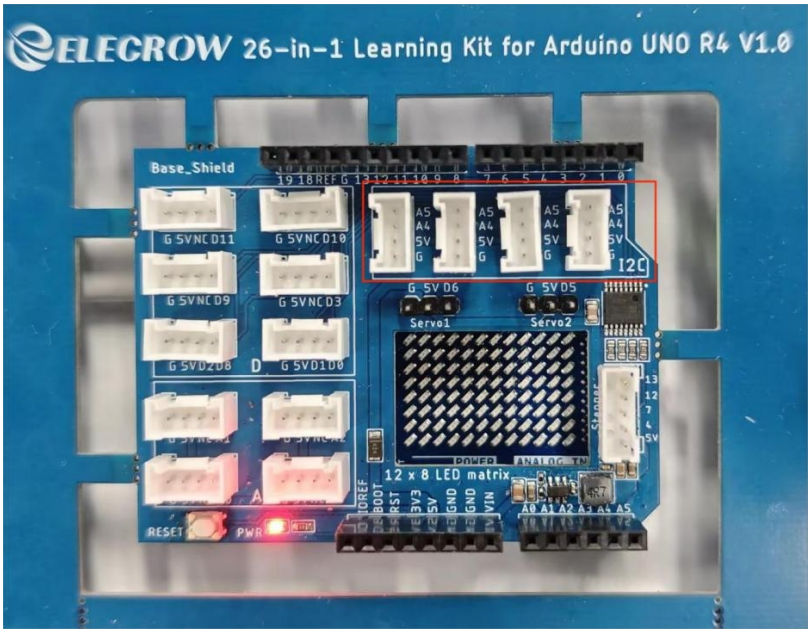
Senzor teploty a vlhkosti Crowtail → ľubovoľný port I2C Bzučiak

Crowtail → digitálny port D10



Špecifikácia štvorportového rozhrania Crowtail:

- GND (čierny) → GND
- VCC (červený) → 5V
- SDA (biela) → A4
- SCL (žltý) → A5



4. Vysvetlenie príkladu

Kliknite na odkaz a stiahnite si oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-17_DHT20/17_DHT20

Otvorte program kurzu v priečinku **„17_DHT20“** pomocou Arduino IDE:

17_DHT20.ino

```
#include "DHT20.h"

DHT20 DHT;

#define BUZ 10   // Beeper is on pin D10

// Set the alarm threshold
float tempThreshold = 30.0;  // Alarm triggered when temperature exceeds 30°C
float humiThreshold = 40.0;  // Alarm when humidity is below 40%

uint8_t count = 0;

void setup()
{
  Serial.begin(115200);
  Serial.println("DHT20 + BUZZER Alarm");

  Wire.begin();
  DHT.begin();

  pinMode(BUZ, OUTPUT);
  digitalWrite(BUZ, LOW);   // Default setting: Disable the buzzer

  delay(1000);
}

void loop()
{
  // Read every 1 second
  if (millis() - DHT.lastRead() >= 1000)
  {
    int status = DHT.read();   // Read the sensor
    float humidity = DHT.getHumidity();
    float temperature = DHT.getTemperature();

    // Print the title every 10 items.
    if ((count % 10) == 0)
    {
      count = 0;
      Serial.println();
      Serial.println("Type\tHumidity (%)\tTemp (°C)\tStatus");
    }
    count++;
```

Vysvetlenie kľúčového kódu

(1) Použité súbory knižnice

```
#include "DHT20.h"
```

Účelom tejto vety je:

Oznámiť Arduinu: Budem používať knižnicu pre senzor **teploty a vlhkosti DHT20**.

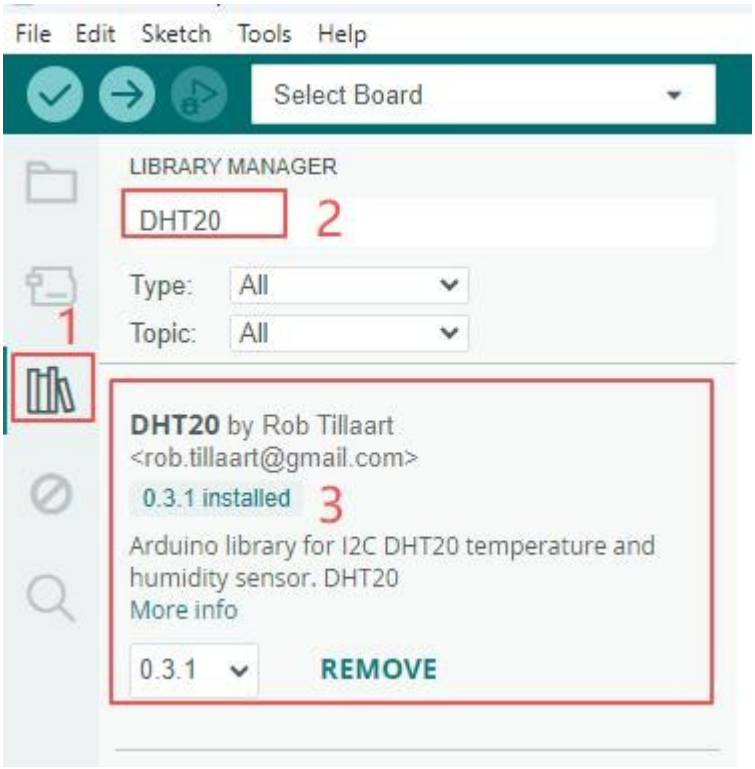
Táto knižnica už obsahuje metódy na čítanie teploty a vlhkosti a my ich len potrebujeme vyvolať.

Nemusíme sami písať základný kód pre komunikáciu I2C.

Na použitie tejto knižnice je možné zvoliť nasledujúce metódy:

- Stiahnuť v Arduino IDE

Verzia knižnice DHT20, ktorú používame, je 0.3.1.



(2) DHT20 DHT;

```
DHT20 DHT;
```

Tento riadok je mimoriadne dôležitý. Môžete si to vysvetliť takto:

Tento riadok kódu vytvorí „senzorový objekt DHT“ typu DHT20. Všetky budúce merania teploty a vlhkosti sa budú opierať oň.

Je to, ako keby ste vytvorili **„asistenta pre teplotu a vlhkosť“** a v budúcnosti ho môžete spustiť takto:

- DHT.read()
- DHT.getTemperature()

- DHT.getHumidity()

aby mohol vykonávať úlohy a vrátiť vám teplotu a vlhkosť.

(3) Definovanie pinov

```
#define BUZ 10
```

Týmto sa definuje konštanta s názvom BUZ, ktorej hodnotou je číslo **10**. Použitie príkazu „Define“ bolo už viackrát vysvetlené v predchádzajúcich lekciách, preto sa tomu tu nebudem podrobne venovať.

To znamená:

Bzučiak je pripojený k **pinu D10** Arduina.

(4) Nastavte prah alarmu

```
float tempThreshold = 30.0;       // Alarm sa spustí, keď teplota prekročí 30 °C float
humiThreshold = 40.0;
```

Toto je vaše pravidlo pre alarm, známe aj ako prahová

hodnota. Systému zadáte:

Ak teplota prekročí **30 °C** → Neštandardný stav → Alarm
Ak je vlhkosť nižšia ako **40 %** → Neštandardný stav → Alarm
Na základe týchto hodnôt systém určí, či spustí alarm.

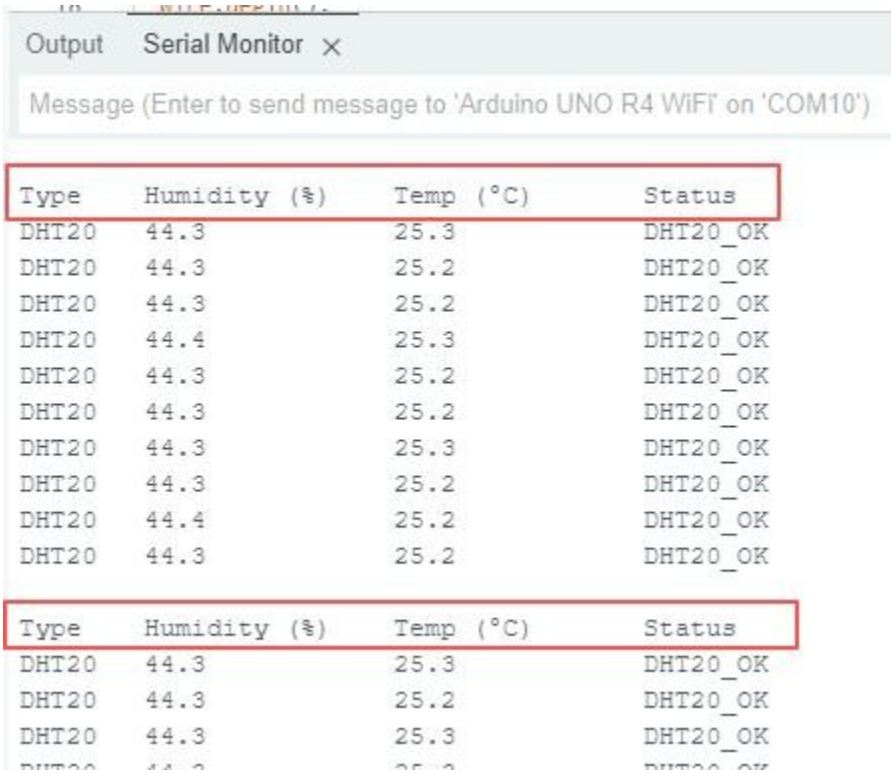
(5) Definujte počítadlo. Tu sa premenná používa na počítanie počtu tlačí.

```
uint8_t count = 0;
```

Úlohou tohto riadku kódu je vytvoriť „mini počítadlo“, ktoré zaznamenáva, koľko údajov o teplote a vlhkosti program vytlačil.

„uint8_t“ je veľmi malý typ celočíselného typu bez znamienka, ktorý zaberá len 1 bajt. Jeho rozsah hodnôt je **od 0 do 255,** čo je práve vhodné pre túto jednoduchú úlohu počítania, ktorá potrebuje počítať len **od 0 do 10** a potom začať znova od 0.

Premenná sa nazýva „**count**“, čo znamená „**počítanie**“; je jej priradená hodnota 0, čo znamená „začať počítať od 0“. V ďalšej časti programu sa bude zvyšovať o 1 zakaždým, keď sa vytlačia údaje zo senzora. Keď dosiahne hodnotu 10, opäť sa vytlačí hlavička (názvy stĺpcov), aby bol obsah okna sériového portu prehľadný a čitateľný, potom sa „count“ vynuluje a začne sa od začiatku. Nezúčastňuje sa na posudzovaní alarmov, neovplyvňuje senzor a je len malým pomocníkom pri formátovaní výstupu, vďaka čomu je vaše rozhranie na ladenie ľahšie zrozumiteľné.



Rovnako ako pri tlačení informácií zo sériového portu, vytlačte obsah **10-krát**, spočítajte 10-krát a potom znovu vytlačte hlavičku. **(Toto je účinok nasledujúcej operácie na premennú „count“.)**

17_DHT20.ino

```
23
24    delay(1000);
25  }
26
27  void loop()
28  {
29    // Read every 1 second
30    if (millis() - DHT.lastRead() >= 1000)
31    {
32      int status = DHT.read();   // Read the sensor
33      float humidity = DHT.getHumidity();
34      float temperature = DHT.getTemperature();
35
36      // Print the title every 10 items.
37      if ((count % 10) == 0)
38      {
39        count = 0;
40        Serial.println();
41        Serial.println("Type\tHumidity (%)\tTemp (°C)\tStatus");
42      }
43      count++;
```

(6) setup() — Program sa spustí len raz pri jeho spustení.

Funkcia **setup()** slúži ako „inicializačný proces pri spustení“, ktorý sa vykoná iba raz pri zapnutí Arduina. Je to podobné tomu, ako váš počítačový systém načíta ovládače a pripraví zariadenia pri spustení.

```
void setup()
{
   Serial.begin(115200); Serial.println("DHT20 +
   BUZZER Alarm"); Wire.begin();
   DHT.begin(); pinMode(BUZ,
   OUTPUT);
   digitalWrite(BUZ, LOW);          // Predvolené nastavenie: Vypnúť
   bzučiak delay(1000);
}
```

① **Inicializácia sériového portu**

```
Serial.begin(115200);
Serial.println("DHT20 + BUZZER Alarm");
```

Na účely ladenia vám to umožňuje zobraziť teplotu a vlhkosť spolu s informáciami o alarme na počítači. Najskôr spustite **príkaz „Serial.begin(115200)“**, aby ste otvorili sériovú komunikáciu a nastavili rýchlosť na **115200**. Týmto spôsobom môžete vidieť výstupné informácie v sériovom monitore vášho počítača; potom spustite **príkaz „Serial.println(„DHT20 + BUZZER Alarm“)“**, aby sa zobrazila uvítacie správa informujúca vás o úspešnom spustení programu.

② **Spustite I2C a senzor**

```
18    Wire.begin();
19    DHT.begin();
```

Wire.begin() iniciuje zbernicu I2C, čo umožňuje správnu komunikáciu medzi Arduinom a senzormi I2C, ako je DHT20;

D**HT.begin()** konkrétne inicializuje senzor teploty a vlhkosti DHT20, čím v podstate oznamuje senzoru: „Je čas začať pracovať“.

Po inicializácii zariadení I2C a senzora teploty a vlhkosti DHT20 môžeme normálne pokračovať v čítaní príslušných hodnôt teploty a vlhkosti.

③ **Nastavte režim pinu pre bzučiak**

```
21    pinMode(BUZ, OUTPUT);
22    digitalWrite(BUZ, LOW);
```

pinMode(BUZ, OUTPUT) nastaví pin D10, na ktorom sa nachádza bzučiak, do výstupného režimu, čo znamená,

že Arduino môže ovládať napätie tohto pinu;

digitalWrite(BUZ, LOW) udržuje bzučiak v uzavretom stave (nízke napätie), čím sa zabráni jeho náhlemu spusteniu pri zapnutí zariadenia.

Tieto dve tvrdenia sme podrobne rozoberali v predchádzajúcich lekciách, čo stačí na to, aby sme poukázali na ich dôležitosť a všeobecnú platnosť.

④ Pred vstupom do slučky počkajte jednu sekundu

Posledné **delay(1000)** sa používa na pozastavenie celého systému na **1 sekundu**, čím sa senzorom poskytne čas na dokončenie ich vnútorného zapnutia a dosiahnutie stabilného stavu, aby sa predišlo abnormálnym hodnotám pri prvých niekoľkých pokusoch v dôsledku príliš rýchlej inicializácie.

(7) Vstup do slučky () – logika, ktorá sa opakuje a vykonáva nepretržite Táto slučka

() je ako „mechanizmus srdcového tepu“ programu, neustále sa opakuje a vykonáva.

```
void loop()
{
   // Čítať každú 1 sekundu
   ak (millis() – DHT.lastRead() ≥ 1000)
   {
      int status = DHT.read();         // Načítanie senzora
      float humidity = DHT.getHumidity();
      float teplota = DHT.getTemperature();

      // Vytlač titulok každých 10
      položiek. if ((count % 10) == 0)
      {
         count = 0;
         Serial.println();
         Serial.println("Typ\tVlhkosť (%)\tTeplota (°C)\tStav");
      }
      count++;

      // vytlačiť údaje
```

```
Serial.print("DHT20\t");
Serial.print(vlhkosť, 1);
Serial.print("\t\t"); Serial.print(teplota, 1);
Serial.print("\t\t");

// zobraziť stav
if (status == DHT20_OK)
    Serial.println("DHT20_OK");
inak
    Serial.println("READ ERROR");

//================================
//                Logika posúdenia alarmu
//================================
bool alarm = false;          // Musíme zavolať políciu?

if (temperature > tempThreshold)
{
    Serial.println("⚠ Teplota je príliš vysoká! Spustil sa alarm!"); alarm =
    true;
}

if (vlhkosť < prahVlhkosti)
{
    Serial.println("⚠ Nízka vlhkosť! Spustil sa alarm!"); alarm
    = true;
}

// Ovládanie bzučiaka
if (alarm)
{
    digitalWrite(BUZ, HIGH);          // Bzučiak zvoní.
}
```

```
    inak
    {
        digitalWrite(BUZ, LOW);          // Vypni bzučiak, ak je všetko v poriadku.
    }
  }
}
```

①Ovládanie intervalu čítania (čítanie raz za 1 sekundu)

```
27    void loop()
28    {
29      // Read every 1 second
30      if (millis() - DHT.lastRead() >= 1000)
31      {
```

Tento riadok**, if (millis() - DHT.lastRead() >= 1000)**, je „jadro časového riadenia“ celého programu. Používa funkciu millis() Arduina (počet milisekúnd od zapnutia systému) na určenie, či uplynula viac ako 1 sekunda od posledného úspešného odčítania senzora. **DHT.lastRead()** je „časová pečiatka posledného úspešného odčítania“ zaznamenaná knižnicou DHT20.

Takže **výraz millis() - DHT.lastRead()** vyjadruje, „koľko času uplynulo od posledného zmerania“. Ak je tento časový rozdiel väčší alebo rovný **1000 milisekúnd** (teda 1 sekunde), povolíme programu načítať nové údaje o teplote a vlhkosti.

Tento prístup je veľmi profesionálny, pretože sa vyhýba používaniu delay() na nútené pozastavenie programu, čo umožňuje, aby celý cyklus bežal voľne a nezablokoval sa s inými úlohami. Preto tento riadok v skutočnosti implementuje „neblokujúci časovač“, ktorý zabezpečuje, že senzor číta údaje raz za sekundu, pričom zachováva plynulosť a efektívnosť programu. Je to veľmi typický a elegantný spôsob písania časového riadenia vo vstavanom programovaní.

② Čítanie teploty a vlhkosti

```
32        int status = DHT.read();    // Read the sensor
33        float humidity = DHT.getHumidity();
34        float temperature = DHT.getTemperature();
```

Tieto tri riadky kódu dokončujú celý proces „čítania údajov zo senzora“:

Prvý riadok **„int status = DHT.read();“** v skutočnosti odošle príkaz na odčítanie údajov do senzora DHT20, čím sa senzor aktualizuje svoje interné namerané hodnoty a vráti stavový kód (napríklad úspešné odčítanie je **DHT20_OK** a v prípade chyby sa vráti číslo chyby). Program následne na základe tohto stavového kódu určí, či bolo odčítanie úspešné;

Druhý riadok, **„float humidity = DHT.getHumidity();“,** nasleduje po úspešnom načítaní a extrahuje

najnovšie údaje o „vlhkosti“ z knižnice DHT20 a ukladá ich do premennej „humidity“;

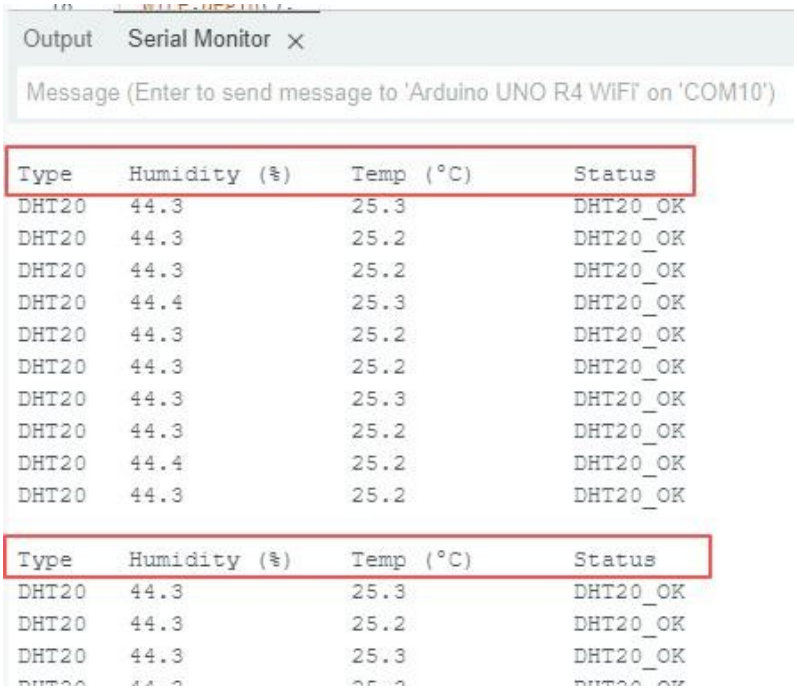
Tretí riadok, **„float temperature = DHT.getTemperature();“**, používa rovnakú metódu na získanie údajov o „teplote“.

Inými slovami, prvý riadok má za úlohu „spustiť senzor a oznámiť, či sa to podarilo alebo nie“, zatiaľ čo ďalšie dva riadky slúžia na „získanie najnovších hodnôt vlhkosti a teploty“ a tieto tri riadky spoločne tvoria kompletný proces „zistenia teploty a vlhkosti pomocou DHT20“, ktorý predstavuje kľúčový zdroj údajov pre následné vyhodnotenie poplachu v rámci celého programu.

③ Zobraziť hlavičku každých 10 položiek

```
36    // Print the title every 10 items.
37    if ((count % 10) == 0)
38    {
39      count = 0;
40      Serial.println();
41      Serial.println("Type\tHumidity (%)\tTemp (°C)\tStatus");
42    }
43    count++;
```

Funkcia tohto kódu je vytlačiť nadpis na sériovom monitore zakaždým, keď sa vygeneruje 10 sád údajov zo senzora, aby sa uľahčilo čítanie a rozlišovanie údajov:



Najskôr sa pomocou podmienky **„if ((count % 10) == 0)“** zistí, či je aktuálny počet násobkom čísla 10. To znamená, že kód vnútri sa vykoná každých 10 údajov. Príkaz **„count = 0;“** slúži na vynulovanie počítadla, aby sa zabezpečilo, že sa spustí raz za každých 10 údajov. Ďalej sa pomocou **„Serial.println();“** vypíše prázdny riadok, aby sa zabezpečilo vizuálne oddelenie. Potom **„Serial.println("Typ\tVlhkosť (%)\tTeplota (°C)\tStav");“** vytlačí hlavičku tabuľky, pričom sa pomocou **„\t“**

oddelenie obsahu jednotlivých stĺpcov, čím sa zabezpečí zarovnanie stĺpcov s údajmi a ľudskému oku sa uľahčí rýchle prehliadanie informácií.

Nakoniec **„count++;“** zvyšuje hodnotu count pri každom spustení slučky, čo znamená, že bol vygenerovaný jeden údaj. Takto sa pri každom prečítaní 10 údajov programom automaticky aktualizuje nadpis, čo vám pomôže jasne rozlíšiť a pochopiť každú skupinu údajov.

④ Vytlačte teplotu, vlhkosť a stav

```
45      // print data
46      Serial.print("DHT20\t");
47      Serial.print(humidity, 1);
48      Serial.print("\t\t");
49      Serial.print(temperature, 1);
50      Serial.print("\t\t");
51
52      // display status
53      if (status == DHT20_OK)
54        Serial.println("DHT20_OK");
55      else
56        Serial.println("READ ERROR");
```

Hodnota 1 znamená zachovať 1 desatinné miesto.

Následne na základe hodnoty **„status“** vypíšte:

- DHT20_OK
- READ ERROR

Je to vhodné na kontrolu, či senzor funguje správne.

```
Type    Humidity (%)    Temp (°C)       Status
DHT20   46.2            25.5            DHT20_OK
DHT20   46.2            25.5            DHT20_OK
DHT20   46.2            25.5            DHT20_OK
DHT20   46.2            25.5            DHT20_OK
DHT20   46.2            25.5            DHT20_OK
DHT20   46.2            25.5            DHT20_OK
DHT20   46.3            25.5            DHT20_OK
DHT20   46.2            25.5            DHT20_OK
```

⑤Posúdenie

```
61    bool alarm = false;   // Do we need to call the police?
62
63    if (temperature > tempThreshold)
64    {
65      Serial.println("⚠ Temperature too high! Alarm triggered!");
66      alarm = true;
67    }
68
69    if (humidity < humiThreshold)
70    {
71      Serial.println("⚠ Low humidity! Alarm triggered!");
72      alarm = true;
73    }
```

Funkcia tohto kódu je určiť, či aktuálna teplota a vlhkosť prekračujú nastavenú prahovú hodnotu alarmu: Najskôr sa vytvorí booleovská premenná **„alarm“** a inicializuje sa na hodnotu false (čo znamená, že alarm je vypnutý). Potom program skontroluje, či je teplota vyššia ako tempThreshold (napríklad **30 °C**). Ak túto hranicu prekročí, vypíše sa varovná správa a alarm sa nastaví na hodnotu true;

Ďalej sa skontroluje, či je vlhkosť nižšia ako humiThreshold (napríklad **40 %**). Ak nespĺňa normu, vypíše sa upozornenie o nízkej vlhkosti a alarm sa nastaví na hodnotu true; Preto hneď, ako je teplota príliš vysoká alebo vlhkosť príliš nízka, premenná alarm sa zapne a informuje nasledujúci kód, že „je potrebné spustiť alarm“. Ide o veľmi typickú a prehľadnú štruktúru posudzovania alarmu s viacerými podmienkami.

⑥ Spustenie alarmu na základe príznaku „alarm“

```
75    // Control the buzzer
76    if (alarm)
77    {
78      digitalWrite(BUZ, HIGH);   // The buzzer is ringing.
79    }
80    else
81    {
82      digitalWrite(BUZ, LOW);    // Turn off the buzzer when it is normal.
83    }
84  }
85  }
```

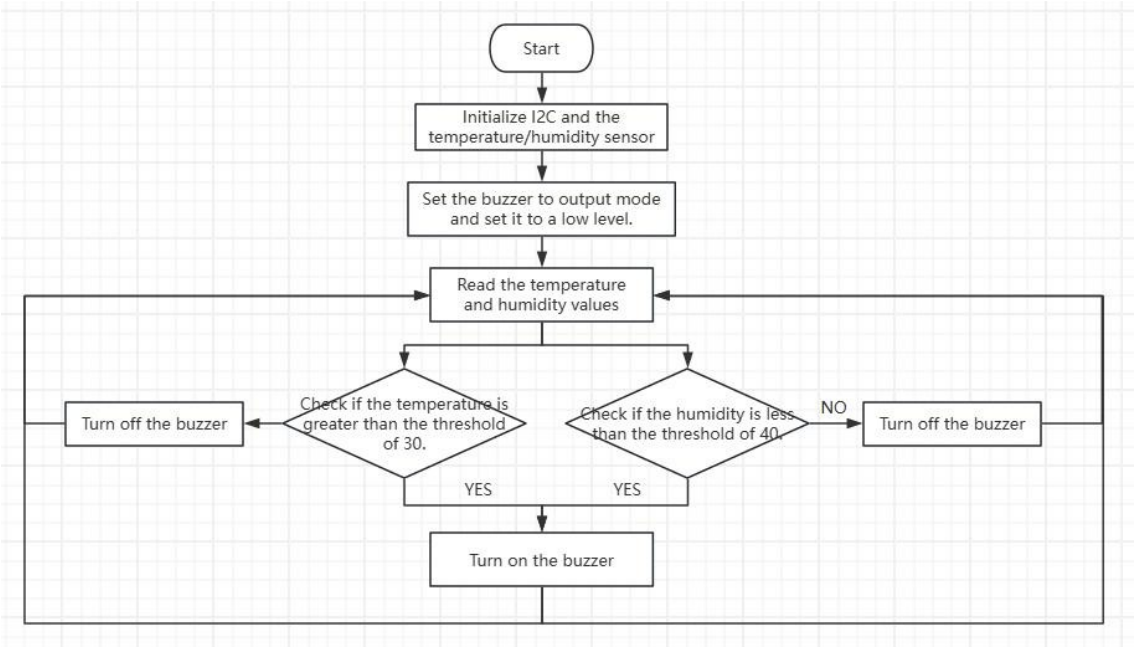
Tento kúsok kódu ovláda bzučiak na základe stavu alarmu vypočítaného skôr:

Ak je premenná alarm nastavená na hodnotu true, znamená to, že teplota alebo vlhkosť prekročili bezpečný rozsah. V takom prípade príkaz **digitalWrite(BUZ, HIGH)** zapne bzučiak pripojený k **pinu D10** a spustí jeho zvuk; Naopak, ak je premenná alarm nastavená na hodnotu false, znamená to, že podmienky v prostredí sú normálne. Program následne vykoná **príkaz digitalWrite(BUZ, LOW)**, čím bzučiak vypne a ten zostane tichý.

Tento kód teda v skutočnosti implementuje všetky predchádzajúce výsledky posúdenia prostredníctvom jednoduchej štruktúry if/else, čím nakoniec premení konečnú fyzickú akciu „či bzučiak zaznie alebo nie“ na skutočné zvukové upozornenie celého poplachového systému. Je to jadro vykonávania

alarmového systému, zodpovedného za premenu rozhodnutia programu na skutočnú zvukovú výzvu.

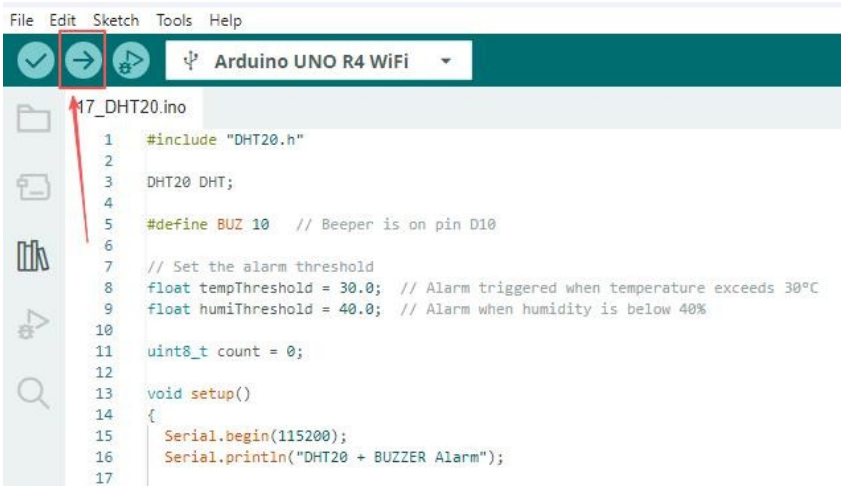
(8) Celkový diagram logiky kódu



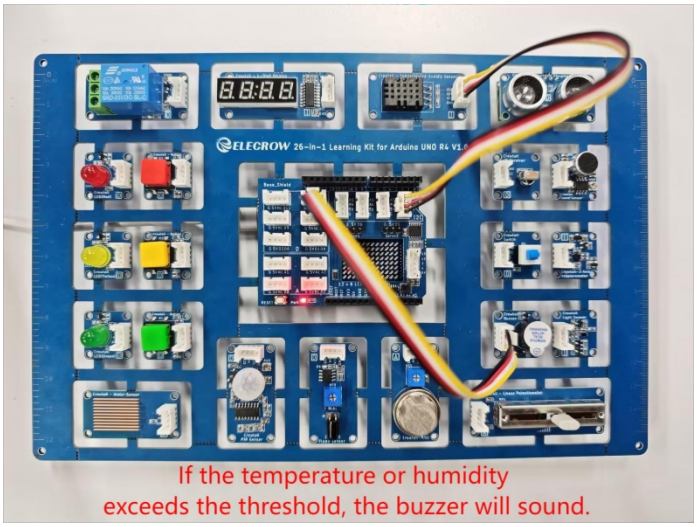
5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu prostredia Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základné nastavenie nahrávania.

Kliknite na **„Stiahnuť“:**



(2) Uvidíte: Systém nepretržite monitoruje teplotu a vlhkosť okolia. Keď teplota prekročí 30 °C alebo vlhkosť klesne pod 40 %, bzučiak vydá nepretržitý alarmový zvuk.



Zároveň sa informácie o poplachu zobrazia na počítači.

```
Output   Serial Monitor ×
Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM10')

DHT20    70.0            29.9            DHT20_OK
DHT20    69.9            29.9            DHT20_OK
DHT20    69.7            30.0            DHT20_OK
DHT20    69.7            30.1            DHT20_OK
⚠ Temperature too high! Alarm triggered!
DHT20    70.1            30.1            DHT20_OK
⚠ Temperature too high! Alarm triggered!
DHT20    70.5            30.1            DHT20_OK
⚠ Temperature too high! Alarm triggered!
DHT20    70.9            30.2            DHT20_OK
⚠ Temperature too high! Alarm triggered!
DHT20    71.2            30.3            DHT20_OK
⚠ Temperature too high! Alarm triggered!

Type     Humidity (%)    Temp (°C)       Status
DHT20    71.5            30.3            DHT20_OK
⚠ Temperature too high! Alarm triggered!
```

Lekcia 18 – IR ovládač

Úvod

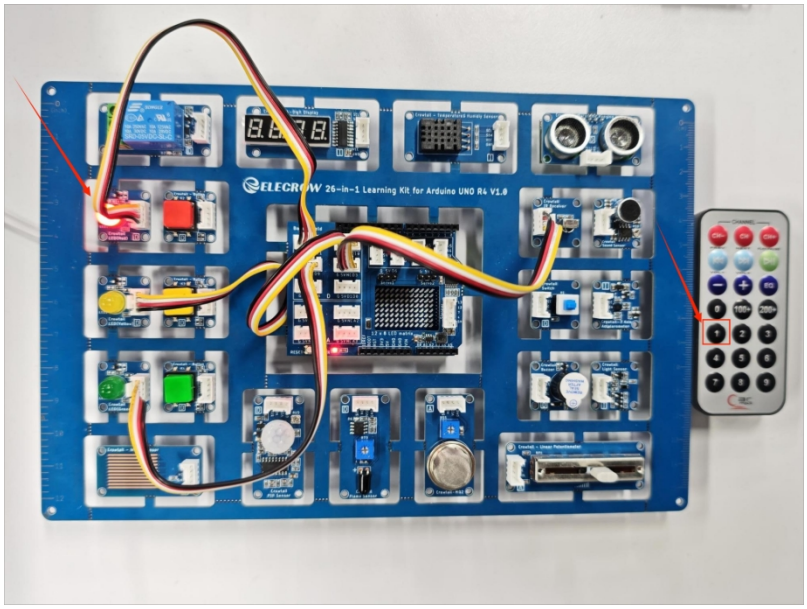
V tejto lekcii budeme používať infračervené diaľkové ovládanie na odosielanie infračervených príkazov do infračerveného prijímača, čím budeme ovládať svetelné efekty červeného, žltého a zeleného svetla. Po absolvovaní tejto lekcie budete mať hlboké znalosti o infračervenom module a budete tak môcť na základe našich materiálov uskutočňovať rozmanitejšie experimenty.

Ciele výučby

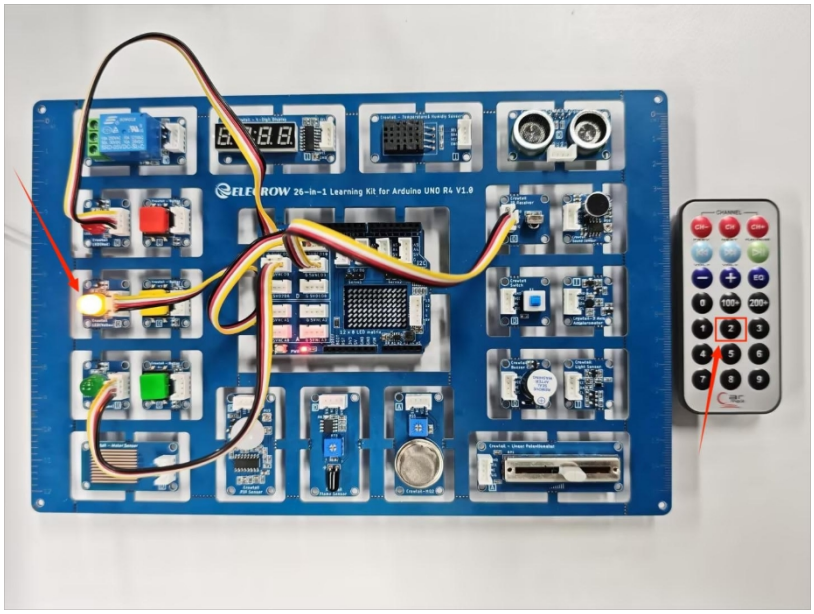
1. Porozumieť princípu fungovania infračerveného prijímacieho modulu.
2. Naučiť sa, ako získať informácie o infračervenom vysielaní a príjme pomocou oficiálnej knižnice <DIYables_IRcontroller.h>.
3. Porozumieť použitiu príslušných funkčných rozhraní v oficiálnej knižnici <DIYables_IRcontroller.h> (napríklad irController.begin(), irController.getKey()).
4. Naučte sa používať príkaz switch.
5. Dokončite efekt prípadu, keď infračervené diaľkové ovládanie zapne svetlo.

Náhľad výsledku

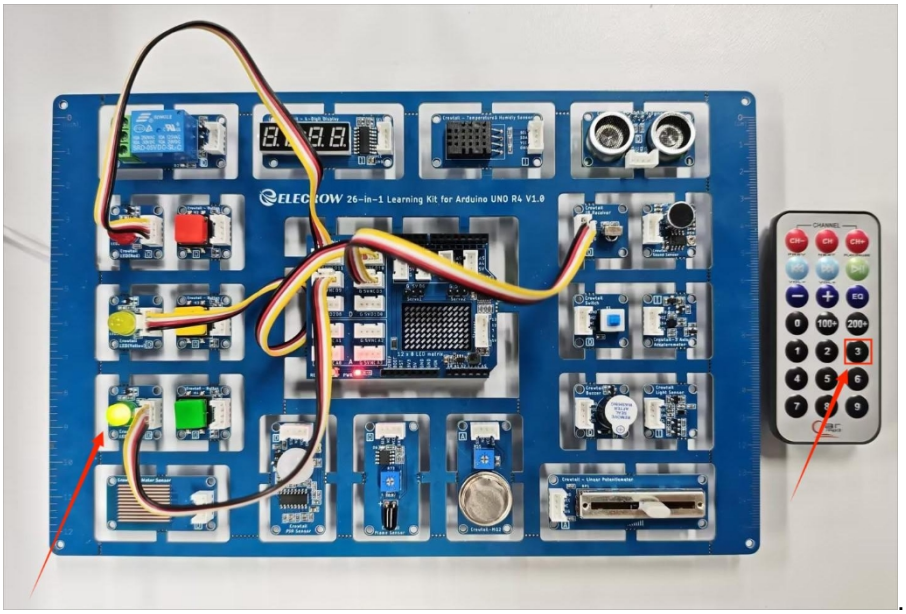
1、Stlačte tlačidlo **„1“** na infračervenom diaľkovom ovládači a rozsvieti sa červené svetlo.



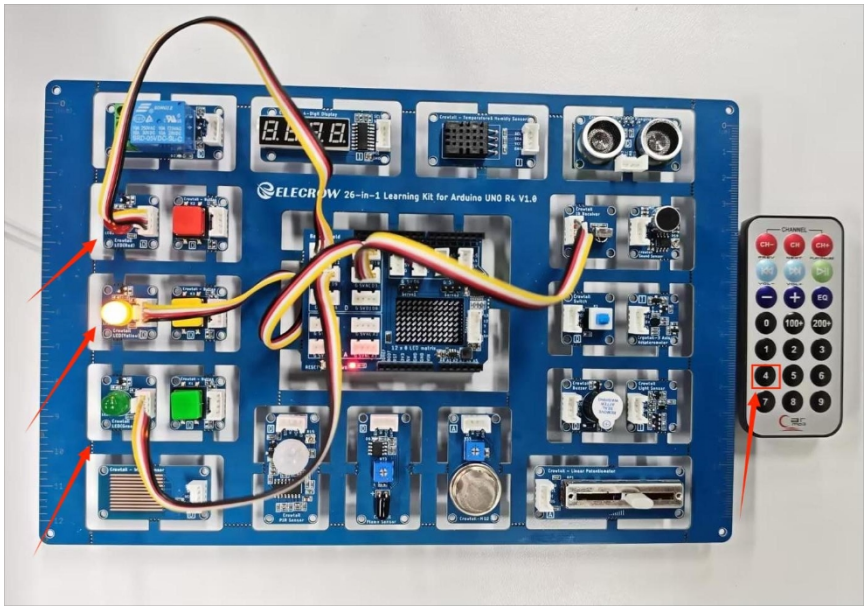
2、Stlačte tlačidlo **„2“** na infračervenom diaľkovom ovládači a rozsvieti sa žlté svetlo.



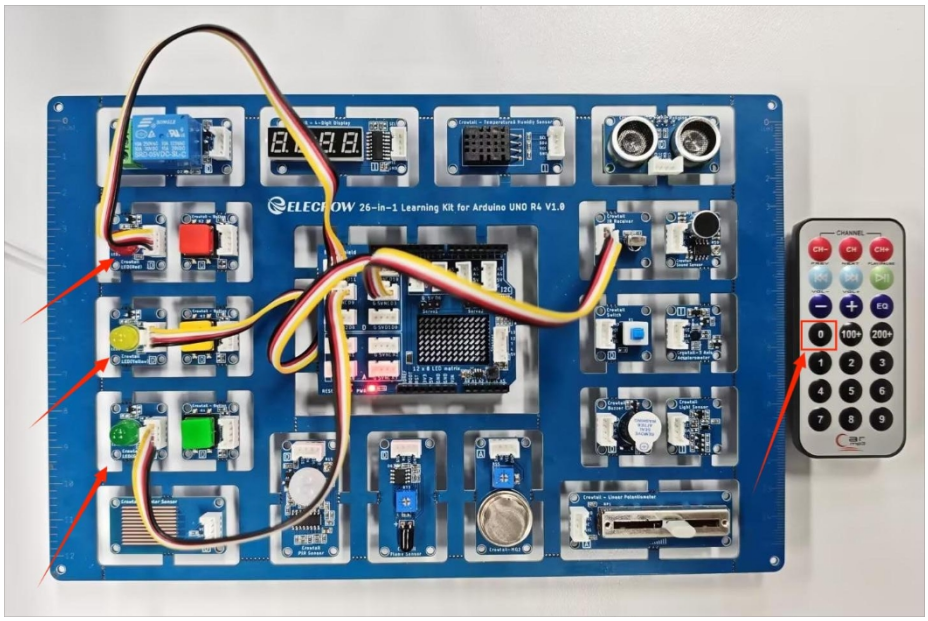
3. Stlačte tlačidlo **„3“** na infračervenom diaľkovom ovládači a rozsvieti sa zelené svetlo.



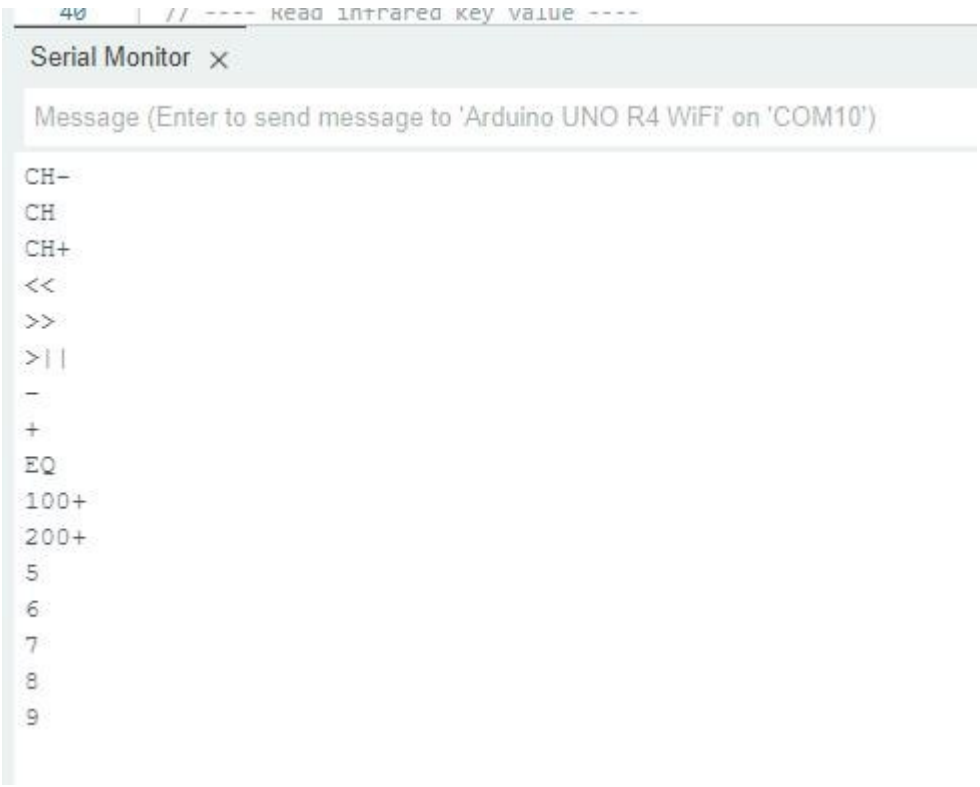
4、Stlačte tlačidlo **„4“** na infračervenom diaľkovom ovládači a tri kontrolky sa postupne rozsvietia v plynulom slede.



5、Stlačte tlačidlo **„0“** na infračervenom diaľkovom ovládači a všetky tri kontrolky zhasnú.

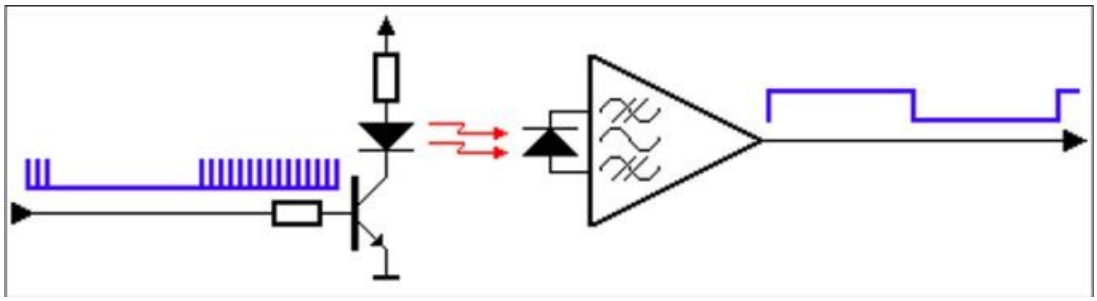


6、Stlačením iných tlačidiel na diaľkovom ovládači môžete vidieť hodnotu stlačeného tlačidla na monitore sériového portu.



1. Vysvetlenie princípu

Princíp fungovania tohto infračerveného prijímacieho senzora spočíva v systéme konverzie a prenosu signálu **typu „elektrický – optický – elektrický**“: Najskôr je kódovaný elektrický signál (zvyčajne impulzný signál modulovaný nosnou vlnou s frekvenciou 38 kHz) vstupujúci na prednej strane zosilnený tranzistorom, ktorý riadi činnosť infračervenej vysielacej trubice – v tomto momente vysielacia trubica prevádza elektrický signál na infračervené svetlo (neviditeľné ľudským okom) v rozsahu 850–940 nm, ktoré sa šíri priestorom vo forme svetla; keď infračervené svetlo dosiahne prijímaciu stranu, hlavná súčasť prijímacieho senzora (infračervená prijímacia trubica, fotodióda citlivá na infračervené svetlo) zaznamená zmenu intenzity svetla a opäť premení infračervený svetelný signál na slabý elektrický signál (čím je intenzita svetla silnejšia, tým je väčší prúd elektrického signálu); tento elektrický signál je však nielen slabý, ale aj zmiešaný s okolným svetlom a inými rušivými vplyvmi, preto je potrebné ho zosilniť operačným zosilňovačom a spracovať vnútornými integrovanými filtračnými a demodulačnými obvodmi, ktoré nakoniec odfiltrujú rušivé vplyvy a obnovia pôvodný kódovaný elektrický signál z vysielacej strany a vygenerujú signál s úrovňou, ktorú môžu rozpoznať nasledujúce obvody (napríklad mikrokontrolér), čím sa dokončí celý proces prijímania a interpretácie infračerveného signálu.



2. Potrebné moduly

Infračervené diaľkové ovládanie × 1

IR prijímač Crowtail × 1

LED Crowtail ×3

3. Spôsob zapojenia

IR prijímač Crowtail → port DIGITAL D3 LED

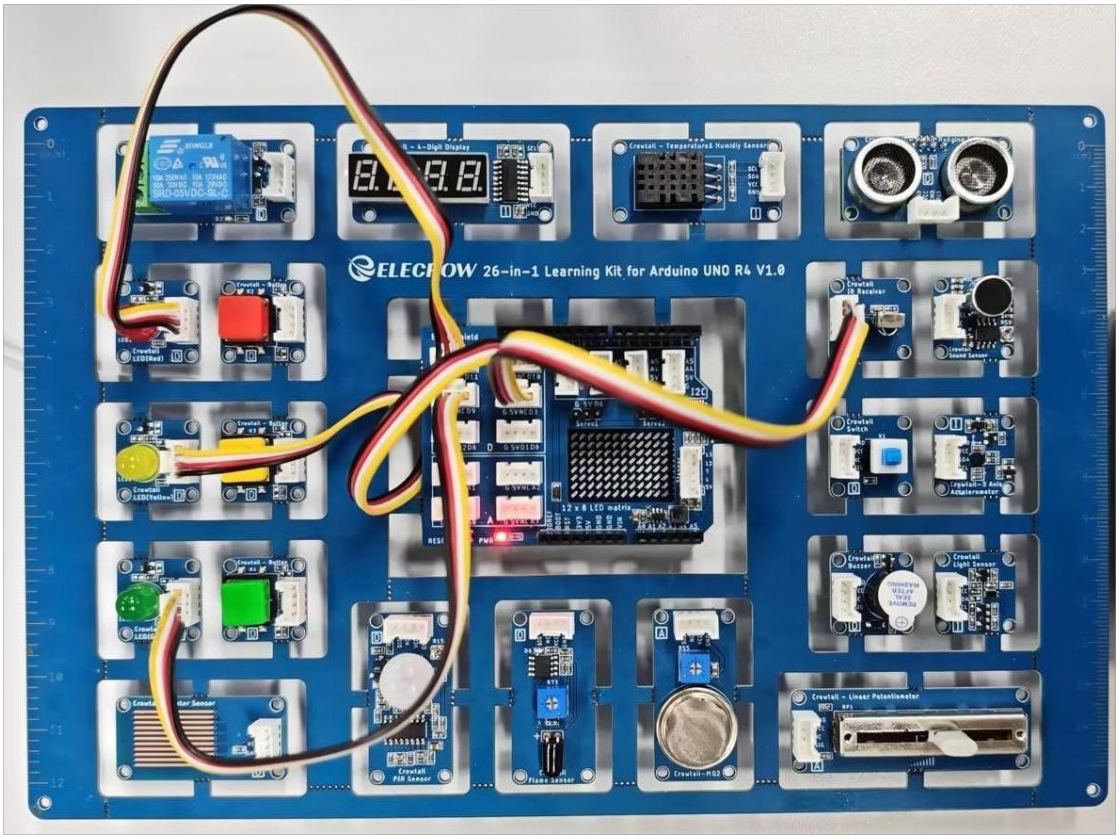
Crowtail (ČERVENÁ) → port DIGITAL D11 LED

Crowtail (ŽLTÁ) → port DIGITAL D10 LED Crowtail

(ZELENÁ) → port DIGITAL D9

Špecifikácia štvorportového rozhrania Crowtail:

SIG (žltá) / NC (biela) / VCC (červená) / GND (čierna)



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-18_IRcontroller/18_IRcontroller

Otvorte program pre túto lekciu v priečinku **„18_IRcontroller“** pomocou Arduino IDE:

18_IRcontroller.ino

```
/*
  IR Remote + LED Control
  Button 1: Red light on
  Button 2: Yellow light on
  Button 3: Green light on
  Button 4: Three lights in sequence (cyclic)
*/

#include <DIYables_IRcontroller.h>

#define IR_RECEIVER_PIN 3

DIYables_IRcontroller_21 irController(IR_RECEIVER_PIN, 200);

// LED pin
#define LED_RED 11
#define LED_YELLOW 10
#define LED_GREEN 9

int mode = 0;  // Current working mode: 1 = Red light / 2 = Yellow light / 3 = Green light / 4 = Continuous flow

void setup() {
  Serial.begin(115200);
  irController.begin();

  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
}

// Encapsulate a function: Clear all lights
void allOff() {
  digitalWrite(LED_RED, LOW);
  digitalWrite(LED_YELLOW, LOW);
  digitalWrite(LED_GREEN, LOW);
}

void loop() {

  // ---- Read infrared key value ----
  Key21 command = irController.getKey();
  if (command != Key21::NONE) {
    switch (command) {
```

Vysvetlenie kódov

(1) Použité súbory knižnice

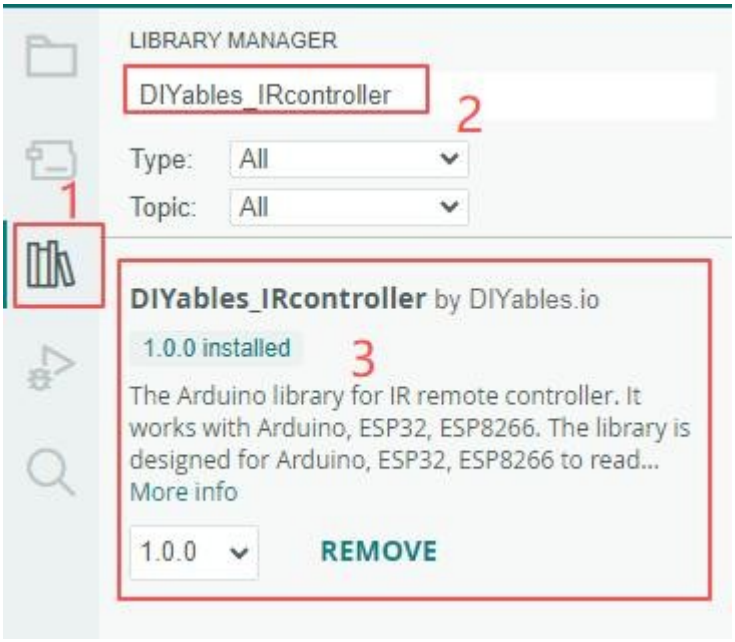
```
#include <DIYables_IRcontroller.h>
```

Načítajte knižnicu na dekódovanie infračerveného diaľkového ovládania od DIYables do Arduina, aby mohlo „rozumieť tlačidlám diaľkového ovládania“. Inak by hlavná doska nemala tušenie, čo prijatý infračervený signál znamená.

Na použitie tejto knižnice máte na výber nasledujúce metódy:

➢ Stiahnite si ju v prostredí Arduino IDE

Verzia knižnice DIYables_IRcontroller, ktorú tu používame, je 1.0.0.



(2) Definujte piny modulu na príjem infračerveného signálu

```
#define IR_RECEIVER_PIN 3
```

Oznámte programu, že infračervený prijímač je pripojený k 3. digitálnemu pinu Arduina, aby knižnica mohla čítať údaje o infračervených impulzoch zo správneho pinu, keď je v prevádzke.

(3) Vytvorte objekt

```
DIYables_IRcontroller_21 irController(IR_RECEIVER_PIN, 200);
```

Použite **knižnicu <DIYables_IRcontroller.h>** na vytvorenie objektu dekodéra infračerveného diaľkového ovládača s názvom irController a odovzdajte mu parametre prijímacieho pinu a citlivosti. Nechajte ho nepretržite počúvať infračervené signály z **IR_RECEIVER_PIN (**čo je **pin 3**) a nastavte „minimálny interval rozpoznania pre to isté tlačidlo“ na 200 milisekúnd. To zabezpečí rýchle rozpoznanie tlačidiel a zabráni nesprávnej interpretácii jedného stlačenia tlačidla ako po sebe idúcich spustení, čo umožní Arduinu presne a stabilne rozpoznať každú operáciu stlačenia tlačidla na diaľkovom ovládači.

(4) Definujte čísla vývodov pre tri LED diódy

```
#define LED_RED 11
#define LED_YELLOW 10
#define LED_GREEN 9
```

Funkcia týchto troch riadkov kódu je: priradiť trom pinom Arduina pripojeným k červenému, žltému a zelenému svetlu v našom obvode ľahko zapamätateľné mená – **LED_RED** zodpovedá **pinu 11**, **LED_YELLOW** zodpovedá **pinu 10** a **LED_GREEN** zodpovedá **pinu 9**.

Týmto spôsobom si pri neskoršom písaní programu nemusíte pamätať tie nudné čísla. Stačí napísať názvy svetiel a môžete ich ovládať.

(5) Definujte premenné na uloženie režimu funkcie

```
int režim = 0;
```

Vytvorte celočíselnú premennú s názvom **„mode“** a nastavte jej počiatočnú hodnotu na 0. Táto premenná sa používa na zaznamenanie aktuálneho pracovného režimu systému – je to ako príprava „slotu pamäte stavu“ pre program. Keď neskôr stlačíte diaľkové ovládanie alebo tlačidlá, zmeníte hodnotu „mode“, čím umožníte zariadeniu prepnúť na rôzne funkcie (napríklad režim **červeného** svetla, režim **žltého** svetla, režim **tečúceho** svetla atď.).

(6) sekcia setup

```
void setup() {
    Serial.begin(115200);
    irController.begin();


    pinMode(LED_RED, OUTPUT);
    pinMode(LED_YELLOW, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
}
```

Funkcia **setup()** slúži na nasledujúci účel: Keď sa program spustí, najprv vykoná „celkovú prípravu zariadenia na spustenie“. Medzi nimi **Serial.begin(115200)** otvorí sériový port a nastaví prenosovú rýchlosť na **115200**, čo umožňuje Arduinu odosielať informácie o ladení do počítača;

Kód **irController.begin()** je rozhranie funkcie v knižnici **<DIYables_IRcontroller.h>**. Toto rozhranie voláme priamo na inicializáciu infračerveného senzora, čím mu umožníme začať prijímať signály z diaľkového ovládača;

Ďalšie tri riadky **pinMode(..., OUTPUT)** nastavujú piny zodpovedajúce červenej, žltej a zelenej LED dióde do výstupného režimu, čo je ekvivalentné povedať Arduinu: „Tieto tri piny budem používať na rozsvietenie svetiel“, aby sa zabezpečilo, že celý systém prijímania infračerveného signálu + ovládania LED diód je plne pripravený pred vstupom do cyklu loop().

(7) Definovaná funkcia alloff

```
void allOff() {
```

```
    digitalWrite(LED_RED, LOW);
    digitalWrite(LED_YELLOW, LOW);
    digitalWrite(LED_GREEN, LOW);
}
```

Funkcia **allOff()** slúži na: nastavenie všetkých pinov **červeného, žltého a zeleného** svetla na LOW, čo znamená ich úplné vypnutie, čím v podstate funguje ako „univerzálny vypínač na vypnutie svetiel“; ak v budúcnosti budete chcieť vypnúť všetky svetlá naraz, nebudete musieť písať **digitalWrite()** trikrát pre každé svetlo jednotlivo. Namiesto toho stačí raz zavolať **allOff()**, aby sa súčasne vypli všetky LED diódy, čím sa hlavný program stane prehľadnejším a ľahšie čitateľným.

(8) Cyklus slučky

Pozrite si kompletný kód uvedený v úryvku kódu.

```
void loop() {
    // ---- Načítanie hodnoty infračerveného tlačidla ----
    Key21 command = irController.getKey(); if
    (command != Key21::NONE) {
        switch (command) {


            case Key21::KEY_CH_MINUS:
                Serial.println("CH-");
                // TODO: VAŠE OVLÁDANIE
                break;


            prípad Key21::KEY_CH:
                Serial.println("CH");
                // TODO: VAŠE OVLÁDANIE
                break;
......
            prípad Key21::KEY_1:    //Tlačidlo 1
                Serial.println("Režim 1: Svieti červené svetlo");
                mode = 1;
                break;
```

```
        prípad Key21::KEY_2:    // Tlačidlo 2
            Serial.println("Režim 2: Svieti žlté svetlo"); mode =
            2;
            break;

        case Key21::KEY_3:      // Tlačidlo 3
            Serial.println("Režim 3: Svieti zelené svetlo");
            mode = 3;
            break;
.....
        prípad Key21::KEY_8:
            Serial.println("8");
            // TODO: VAŠE OVLÁDANIE
            break;

        prípad Key21::KEY_9:
            Serial.println("9");
            // TODO: VAŠE OVLÁDANIE
            break;

        default:
            Serial.println("VAROVANIE: neznámy príkaz:"); break;
    }
}
```

Prvá polovica tejto sekcie **„loop**()“ je zodpovedná za nepretržité čítanie klávesov z infračerveného prijímača a ich prevod na vykonateľné inštrukcie: **„Key21 command = irController.getKey();“** na načítanie najnovšieho klávesu (vráti „NONE“, ak nie je stlačený žiadny kláves).

To sa dosiahne volaním funkcie **irController.getKey();** v súbore infračervenej knižnice na analýzu prijatého infračerveného príkazu.

Potom **podmienka „if (command != Key21::NONE)“** zabezpečí, že spracovanie pokračuje len v prípade, ak je klávesa skutočne stlačená. Následne sa prostredníctvom **„switch** (command)“ rozvetvujú rôzne klávesy – každý prípad „case“ vypíše prehľadné informácie na ladenie (napríklad **„Režim 1: Svieti červené svetlo“**) a upraví globálny stav „mode“ (napríklad **0** znamená **úplné vypnutie**, 1/2/3 znamenajú

režimy **červeného/žltého/zeleného** svetla, 4 predstavuje **režim toku**).

Neimplementované klávesy sú ponechané ako „// TODO“ pre budúce rozšírenie a vetva „**default**“ vydá varovanie, keď sa vyskytne neznámy alebo nespracovaný kláves; konštrukcia tohto procesu zabezpečuje, že každé stlačenie diaľkového ovládača je spoľahlivo rozpoznané, čitateľné (výstup sériového portu je vhodný na ladenie) a prevedené na interný signál „mode“, ktorý je jednotne spracovaný následnou časťou vykonávania režimu, čím sa jasne oddelí vstup diaľkového ovládača a logika riadenia osvetlenia, čo uľahčuje údržbu a rozšírenie.

V tomto kóde sa najprv stretávame so syntaxou „switch“. Ďalej vysvetlíme použitie syntaxe „switch“:

V kóde je „**switch (príkaz)**“ ekvivalentom „selektora kľúča“: berie príkaz (hodnota kľúča z diaľkového ovládača) ako index a priamo preskočí na zodpovedajúcu vetvu **„case Key21::...:“** na vykonanie tohto kódu (napríklad stlačenie **KEY_1** vstúpi do case **Key21::KEY_1**, vytlačí výzvu a nastaví **mode = 1**).

„**break**;“ na konci každej vetvy oznamuje programu „vykonaj túto vetvu a potom okamžite ukonči **switch**, nepokračuj vo vykonávaní nasledujúcich vetiev“, čím sa zabráni zmiešaniu spracovania viacerých klávesov;

Vetva **„default**“ je záložná spracovávacia vetva pre prípad, keď sa vyskytne neznáma alebo nespracovaná klávesa, čím sa zabezpečí, že program nezostane ticho alebo nevytvorí chybu;

V porovnaní s reťazcom **„if... else if...“** je štruktúra switch prehľadnejšia a čitateľnejšia pri práci s veľkým množstvom pevných konštánt (napríklad s hodnotami tlačidiel na diaľkovom ovládači) a je tiež jednoduchšie ju rozširovať a udržiavať.

(9) Režim prepínania

```
   switch (režim) {
       case 0:
          allOff();
          break;

       case 1: // Svieti červené svetlo
          allOff(); digitalWrite(LED_RED,
          HIGH); break;
.....
       case 4: // vodopádové
          svetlo allOff();
```

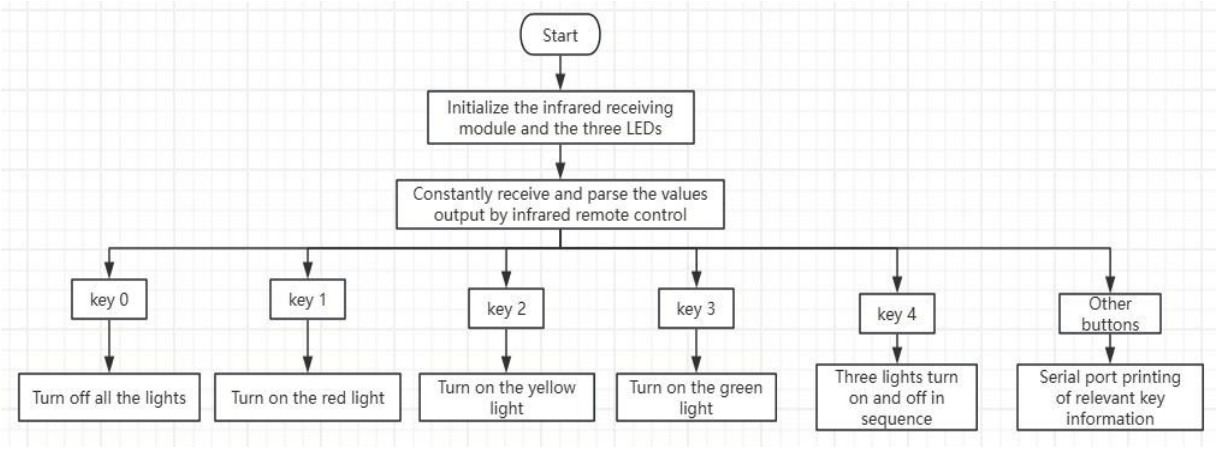
```
            digitalWrite(LED_RED, HIGH);
            delay(300);

            allOff(); digitalWrite(LED_YELLOW,
            HIGH); delay(300);

            allOff(); digitalWrite(LED_GREEN,
            HIGH); delay(300);
            break;

        predvolené:
            // Počiatočný stav:
            Všetko vypnuté allOff();
            break;
    }
}
```

Táto časť kódu „**switch (mode)**“ slúži ako „riadiace centrum osvetlenia“ celého programu. Nastavuje „režim“ na hodnoty v rozmedzí **od 0 do 4** na základe stlačených tlačidiel na diaľkovom ovládači a následne jednotne určuje, ako by mali svietiť svetlá:

- Keď je „**mode =** 0“, vykonajte „**case 0“** a vypnite všetky svetlá;
- Keď je **„mode =** 1“, prejdite na „**case** 1“, najskôr **zhasnite** svetlá a potom rozsvieťte **červené** svetlo;
- Podobne **„case 2“** rozsvieti **žlté** svetlo, **„case 3“** rozsvieti **zelené** svetlo;
- Najosobitejším prípadom je **„prípad 4“**, ktorý realizuje efekt **„tečúceho svetla**“: najskôr sa zhasnú svetlá, potom sa s oneskorením rozsvieti **červené** svetlo a následne sa prejde na **žlté** a **zelené** , pričom každý segment má interval 300 ms, vďaka čomu sa zdá, že svetlá tečú zľava doprava;

Každý **„prípad“** končí príkazom **„break“**, ktorý informuje program, aby po dokončení špecifikovaného režimu nepokračoval vo vykonávaní kódu iných režimov, inak by logika osvetlenia bola chaotická; A „default“ je záložné riešenie. V prípade, že režim nadobudne neočakávanú hodnotu, bezpečne vypne všetky svetlá. Funkcia celého tohto kódu je umožniť hlavnej slučke automaticky prehrávať príslušný svetelný efekt na základe hodnoty režimu, čím sa ovládanie diaľkovým ovládačom stáva štruktúrovanejším a usporiadanejším.

(10) Logický tok kódu



5. Spustite program a sledujte efekt

(1) Postupujte podľa krokov na obsluhu prostredia Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základné nastavenie nahrávania.

Kliknite na **„Stiahnuť“:**

File Edit Sketch Tools Help

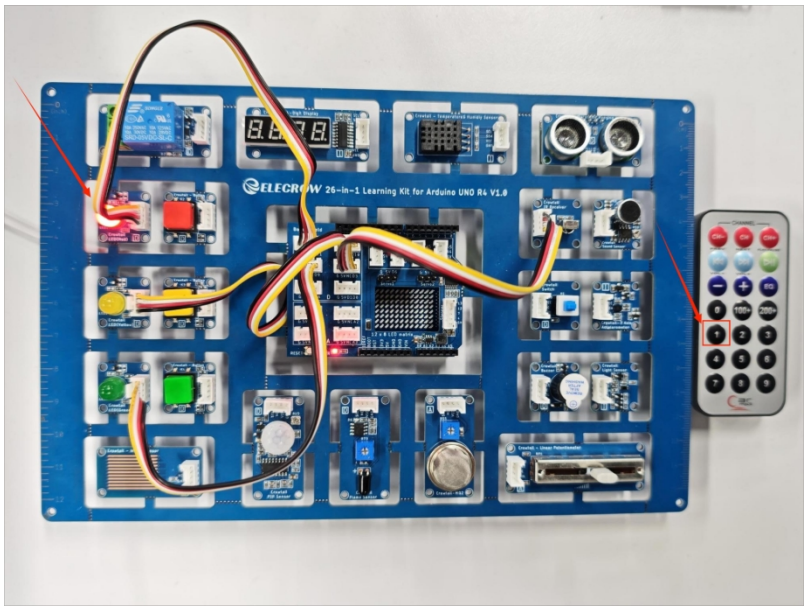
Arduino UNO R4 WiFi

18_IRcontroller.ino

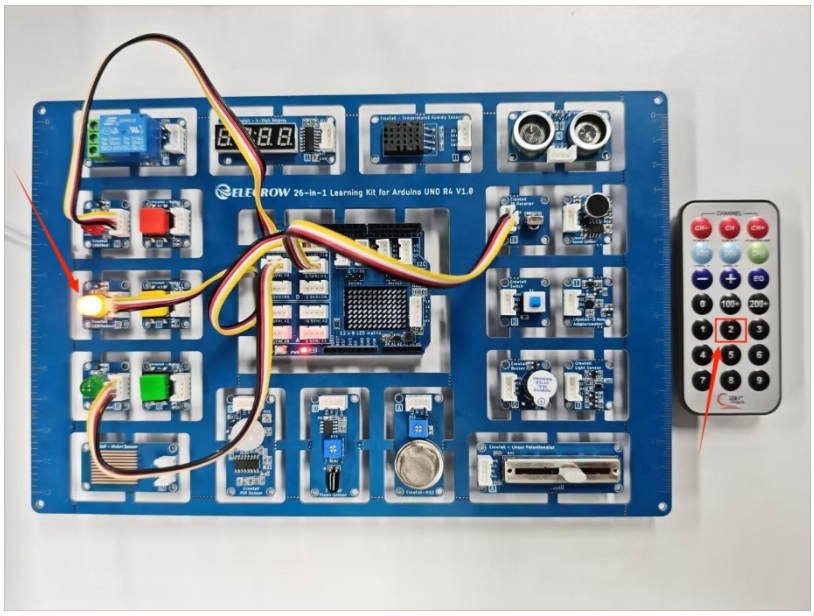
```
/*
  IR Remote + LED Control
  Button 1: Red light on
  Button 2: Yellow light on
  Button 3: Green light on
  Button 4: Three lights in sequence (cyclic)
*/

#include <DIYables_IRcontroller.h>

#define IR_RECEIVER_PIN 3

DIYables_IRcontroller_21 irController(IR_RECEIVER_PIN, 200);

// LED pin
#define LED_RED 11
#define LED_YELLOW 10
#define LED_GREEN 9

int mode = 0;  // Current working mode: 1 = Red light / 2 = Yellow li

void setup() {
  Serial.begin(115200);
  irController.begin();

  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
}
```

(2) Po dokončení sťahovania skontrolujte, či zariadenie funguje:

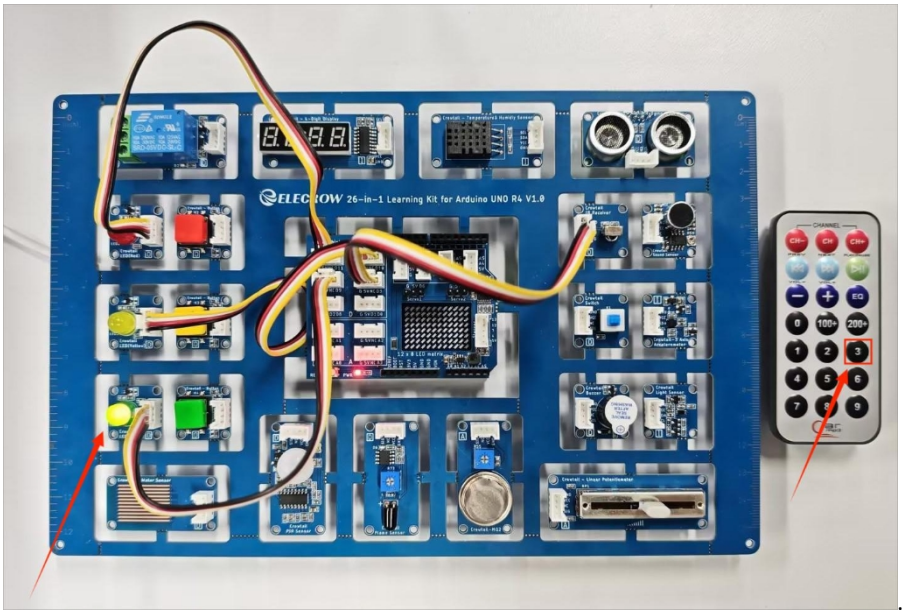
1、Stlačte tlačidlo „**1**“ na infračervenom diaľkovom ovládači a rozsvieti sa červené svetlo.



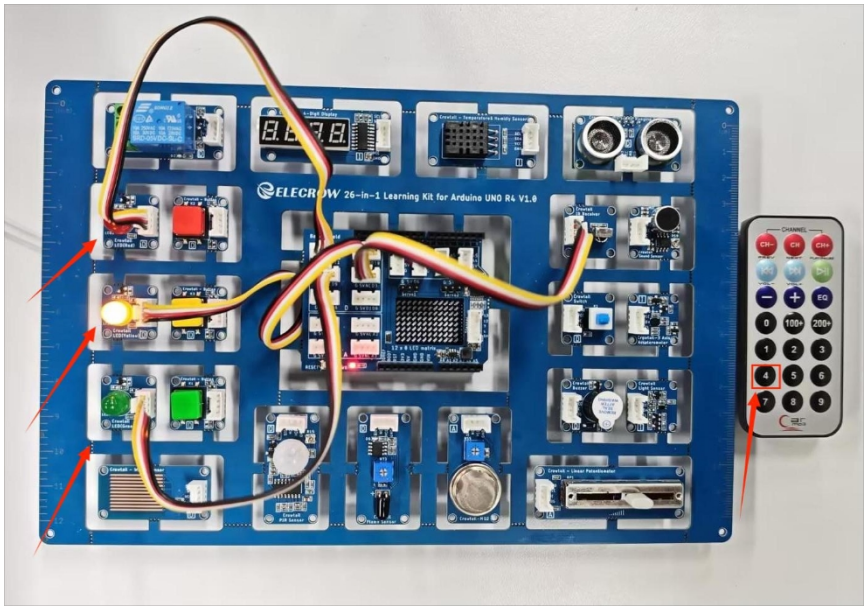
2、Stlačte tlačidlo „2“ na infračervenom diaľkovom ovládači a rozsvieti sa žlté svetlo.



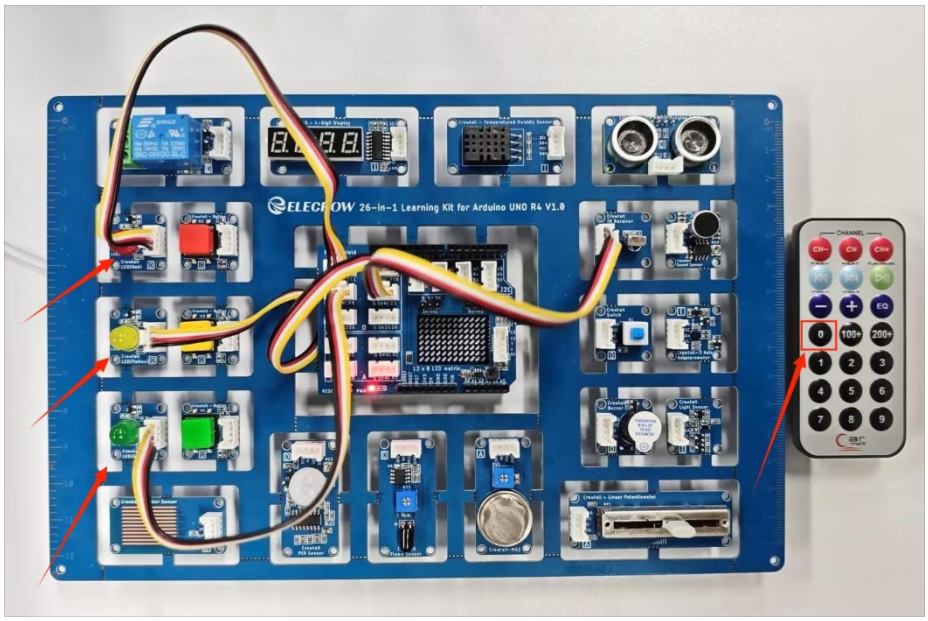
3、Stlačte tlačidlo **„3“** na infračervenom diaľkovom ovládači a rozsvieti sa zelené svetlo.



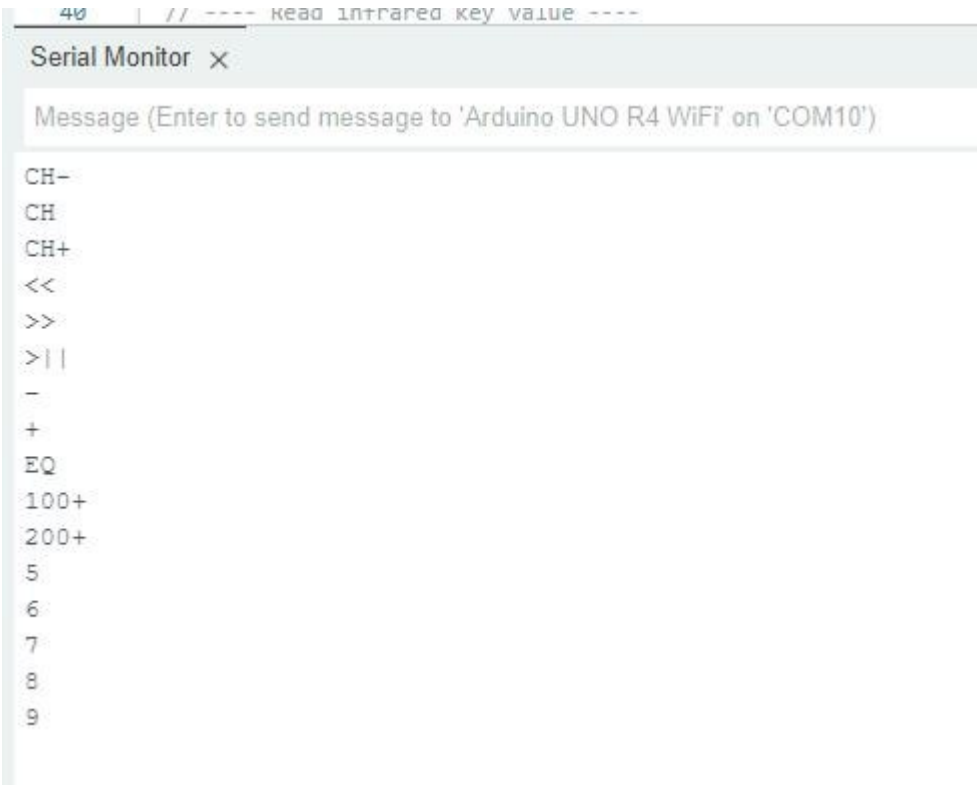
4. Stlačte tlačidlo „4“ na infračervenom diaľkovom ovládači a tri kontrolky sa postupne rozsvietia v plynulom slede.



5、Stlačte tlačidlo „**0**“ na infračervenom diaľkovom ovládači a všetky tri kontrolky zhasnú.



6、Stlačením iných tlačidiel na diaľkovom ovládači môžete vidieť hodnotu stlačeného tlačidla na monitore sériového portu.


40 // ---- Read infrared key value ----
Serial Monitor
Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM10')
CH-
CH
CH+
<<
>>
>||
-
+
EQ
100+
200+
5
6
7
8
9

Lekcia 19 – LSM6DS3

Úvod

V tejto lekcii sa naučíme, ako používať šesťosovú inerciálnu meracú jednotku (IMU) LSM6DS3 v kombinácii s algoritmom výpočtu polohy (Madgwickovo filtrovanie) a bzučiakom na vytvorenie inteligentného systému alarmu naklonenia.

Po skončení tejto hodiny budete vedieť, ako sú v praxi implementované systémy, ako napríklad alarm proti pádu telefónu, alarm proti prevráteniu zariadenia a detekcia polohy robota.

Ciele výučby

1. Porozumieť základnému princípu fungovania šesťosovej IMU LSM6DS3.
2. Naučiť sa inicializovať a čítať údaje o zrýchlení a gyroskope pomocou oficiálnej knižnice Arduino_LSM6DS3.
3. Pochopiť, prečo nie je možné priamo vypočítať uhol z akcelerácie.
4. Naučiť sa zaviesť a používať algoritmus výpočtu polohy MadgwickAHRS.
5. Naučte sa zistiť, či sa zariadenie prevrátilo, pomocou uhlov náklonu v osi Roll a Pitch.
6. Implementujte kompletný systém alarmu naklonenia pomocou logiky posudzovania stavu a bzučiaka.

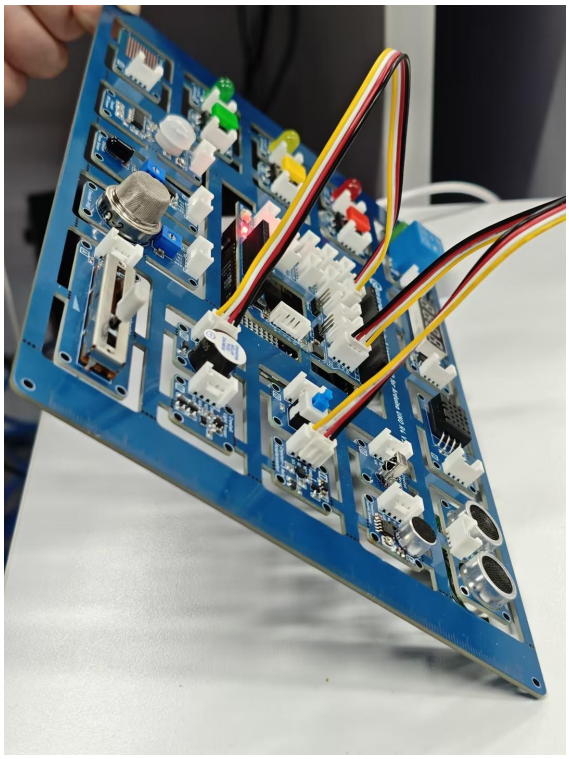
Náhľad výsledku

Systém bude nepretržite monitorovať polohu zariadenia s frekvenciou 10 Hz (10-krát za sekundu):

Keď uhol náklonu alebo sklonu zariadenia presiahne 30°:

(Tu môžete dočasne považovať uhol náklonu za otáčanie pozdĺž dlhej strany a uhol sklonu za otáčanie pozdĺž krátkej strany)

- Siréna vydáva nepretržitý výstražný tón (v prípade, že uhol náklonu presiahne 30°)



➢ Výstup sériového portu: !!! TILT ALARM !!!

```
37    void checkTilt() {
```

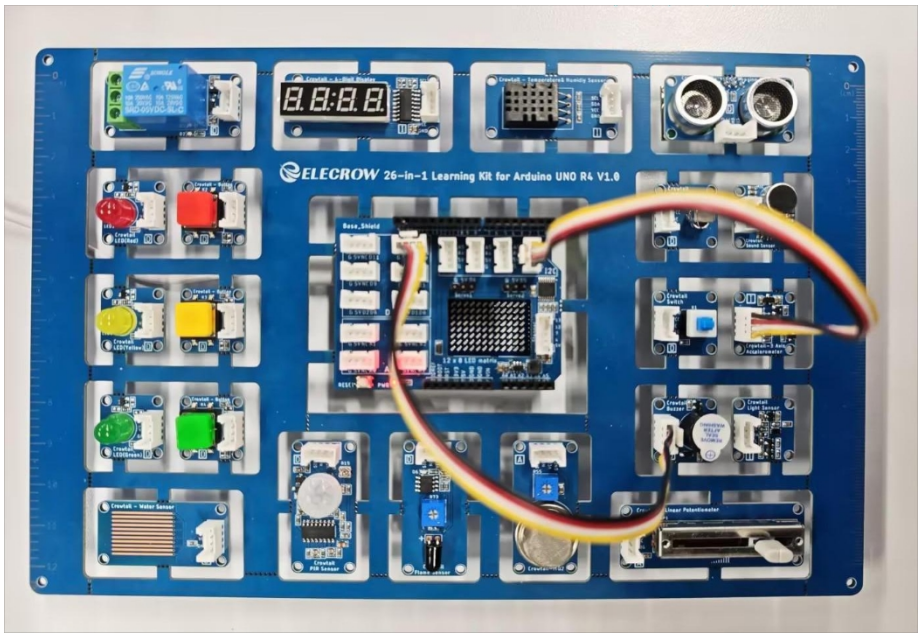
Output Serial Monitor ×

Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM10')

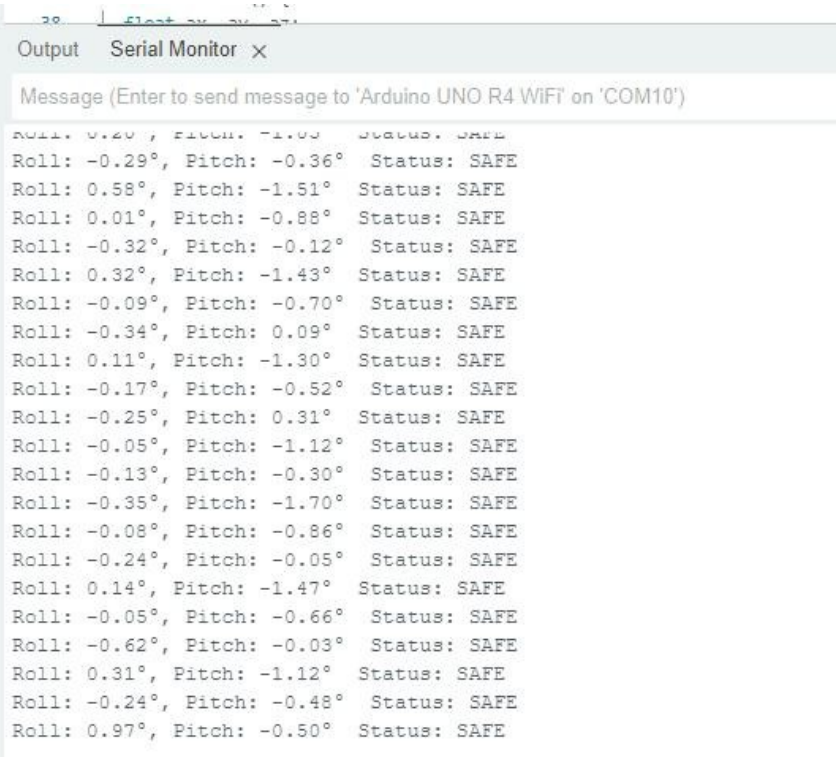
```
Roll: 0.61°, Pitch: -1.39°  Status: SAFE
Roll: -0.07°, Pitch: -0.86°  Status: SAFE
Roll: -0.47°, Pitch: -0.13°  Status: SAFE
Roll: 0.19°, Pitch: -1.44°  Status: SAFE
Roll: -0.22°, Pitch: -0.71°  Status: SAFE
Roll: -0.40°, Pitch: 0.09°  Status: SAFE
Roll: 0.41°, Pitch: -1.06°  Status: SAFE
Roll: -0.51°, Pitch: -1.01°  Status: SAFE
Roll: 2.10°, Pitch: 1.72°  Status: SAFE
Roll: 10.40°, Pitch: 2.56°  Status: SAFE
Roll: 15.53°, Pitch: 3.16°  Status: SAFE
Roll: 22.82°, Pitch: 2.97°  Status: SAFE
Roll: 27.34°, Pitch: 2.14°  Status: SAFE
Roll: 31.02°, Pitch: 0.48°  !!! TILT ALARM !!!
Roll: 33.90°, Pitch: -0.62°  !!! TILT ALARM !!!
Roll: 36.29°, Pitch: -0.26°  !!! TILT ALARM !!!
Roll: 39.68°, Pitch: -0.21°  !!! TILT ALARM !!!
Roll: 40.18°, Pitch: 0.62°  !!! TILT ALARM !!!
Roll: 41.44°, Pitch: 0.23°  !!! TILT ALARM !!!
Roll: 40.32°, Pitch: -0.07°  !!! TILT ALARM !!!
Roll: 39.32°, Pitch: 0.29°  !!! TILT ALARM !!!
```

Keď sa zariadenie nachádza v bezpečnom prevádzkovom rozsahu:

➢ Bzučiak mlčí



➢ Sériový port zobrazuje aktuálny uhol sklonu a výstupy: Stav: SAFE



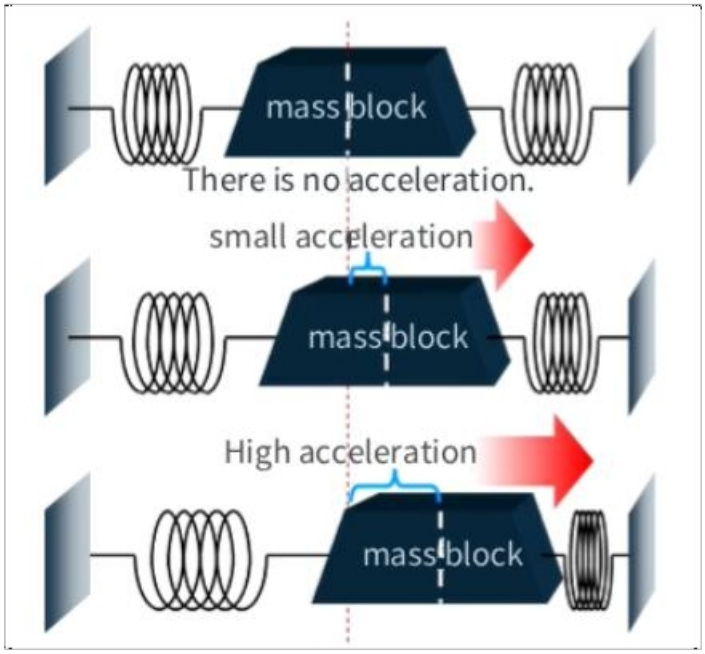
1. Vysvetlenie princípu

Konštrukcia „pružina-hmota-blok-plášť“ znázornená na nasledujúcom obrázku je základnou pracovnou jednotkou jednoosového akcelerometra: Keď je celá jednotka v pokoji alebo sa pohybuje konštantnou rýchlosťou vo vzťahu k meranému objektu, hmotný blok je udržiavaný v strednej rovnovážnej polohe pod napätím dvoch pružín bez akéhokoľvek relatívneho posunu; zatiaľ čo keď objekt generuje zrýchlenie v smere pružiny, hmotný blok sa posunie

vzhľadom na plášť v dôsledku zotrvačnosti – čím menšie zrýchlenie, tým menšie posunutie (ako je znázornené v stave zodpovedajúcom „malému zrýchleniu“ na obrázku), a čím väčšie zrýchlenie, tým väčšie posunutie (ako je znázornené v stave zodpovedajúcom „vysokému zrýchleniu“ na obrázku). Senzor premení tento posun na elektrický signál (napríklad zmenu kapacity, zmenu odporu) a následne vypočíta hodnotu zrýchlenia v danom smere. Toto je detekčná logika jednoosového zrýchlenia.

„Časť na detekciu zrýchlenia“ šesťosového senzora usporadúva tri z týchto jednoosových jednotiek v priestorovo ortogonálnej konfigurácii osí X, Y a Z. Keď sa objekt pohybuje v akomkoľvek smere, jeho zrýchlenie sa rozloží na zložky pozdĺž týchto troch osí. Každá zložka zodpovedá posunu hmotnostného bloku v jednoosovej jednotke, čím sa generujú údaje o lineárnom zrýchlení pozdĺž týchto troch osí. Okrem toho šesťosový senzor integruje tri ortogonálne gyroskopické jednotky. Každá gyroskopická jednotka detekuje rotačnú uhlovú rýchlosť objektu okolo osí X, Y a Z na základe princípu, že vibrujúci hmotnostný blok je posúvaný Coriolisovou silou. Kombinácia údajov o zrýchlení v troch osiach a údajov o uhlovej rýchlosti v troch osiach napokon umožňuje v reálnom čase vnímať stav pohybu objektu v šiestich stupňoch voľnosti (tri lineárne smery + tri rotačné smery) v priestore.

Nasledujúci obrázok znázorňuje šesťosovú IMU (inertiálnu meracú jednotku). Pojem „šesťosová“ označuje



3-osové akcelerometre + 3-osové gyroskopy, ktoré konkrétne snímajú pohyb v súradnicových osiach X, Y a Z. Podrobnosti sú nasledujúce:

①Zloženie šesťosovej jednotky: 3-osové akcelerometre + 3-osové gyroskopy

Jadrom šesťosovej IMU je integrácia dvoch senzorov:

3-osové akcelerometre: Detekujú lineárne zrýchlenie v smeroch X, Y a Z (zvyčajne v jednotkách g, 1 g ≈ 9,8 $\mathrm{m/s^2}$);

3-osové gyroskopy: Merajú uhlovú rýchlosť v smeroch X, Y a Z (zvyčajne v jednotkách °/s).

②**Definície jednotlivých osí** (v kombinácii s fyzickými smermi senzorov na obrázku) Osy X, Y a Z červeného senzora na doske plošných spojov na obrázku tvoria „súradnicový systém nosiča“ definovaný v čase výroby hardvéru, konkrétne zodpovedajú:

Osi X (červená): Paralelná so smerom predĺženia vývodov senzora (na obrázku vodorovne vľavo);

Os Y (zelená): kolmá na os X, v rovine dosky s plošnými spojmi senzora (na obrázku smeruje šikmo doprava hore);

Os Z (modrá): kolmá na rovinu dosky s plošnými spojmi senzora (na obrázku smeruje vertikálne nahor).

③**Obsah detekcie 3-osových akcelerometrov**

Akcelerometre detekujú „lineárne zrýchlenie pozdĺž osí X, Y a Z“, vrátane:

Zrýchlenie v osi X: Zrýchľovací/spomaľovací pohyb senzora v smere osi X (napríklad posunutie senzora v smere osi X doľava/doprava);

Zrýchlenie v osi Y: Pohyb senzora spojený so zrýchlením alebo spomalením v smere osi Y (napríklad pohyb senzora po diagonále v smere osi Y);

Zrýchlenie v osi Z: Zrýchlenie/spomalenie pohybu senzora v smere osi Z (napríklad zdvihnutie senzora vertikálne nahor/nadol).

Zároveň môžu akcelerometre nepriamo snímať statickú polohu senzora (napríklad uhol naklonenia) prostredníctvom „zložky gravitačného zrýchlenia v každej osi“ – obrázok však tiež ukazuje: Akcelerometre nemôžu detekovať odklon (rotáciu okolo osi Z), pretože odklon je horizontálny obrat a nedochádza k žiadnej zmene zložky gravitácie pozdĺž osi Z.

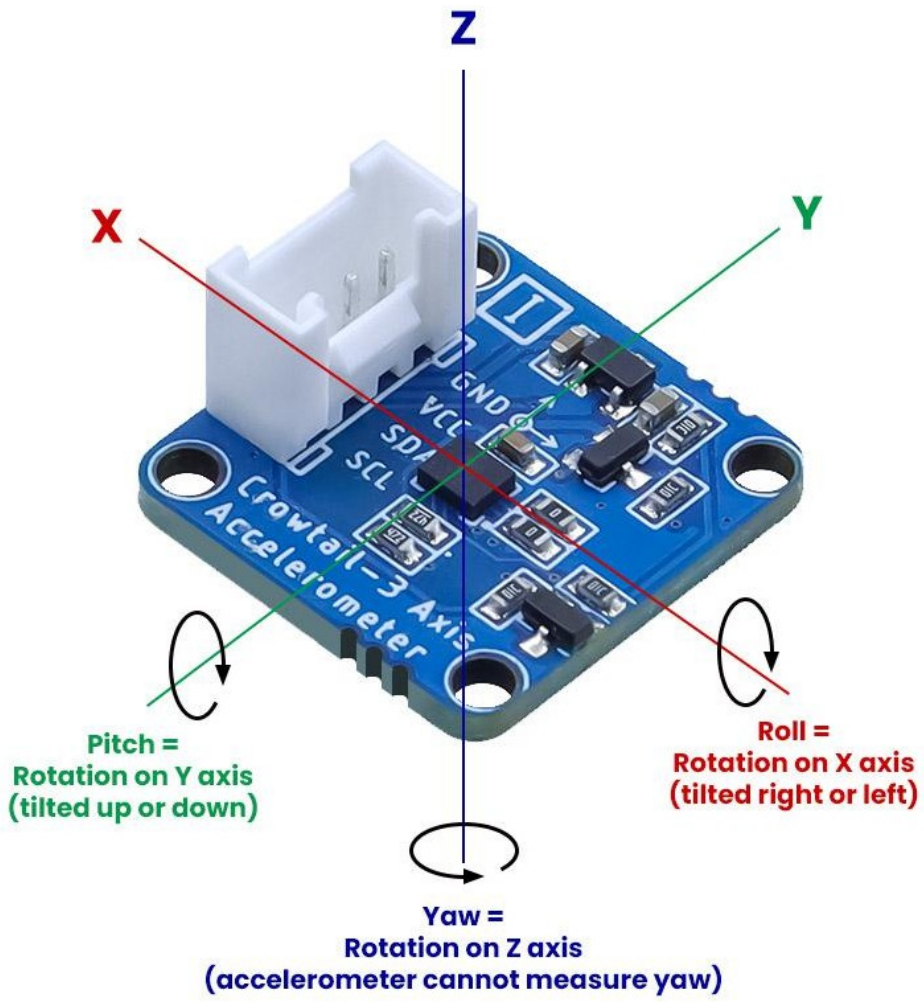
④**Obsah detekcie 3-osových gyroskopov**

Gyroskopy detekujú „rotačnú uhlovú rýchlosť okolo osí X, Y a Z“, čo zodpovedá trom uhlom polohy na obrázku:

Otáčanie okolo osi X (naklonenie, odklon): Gyroskop meria uhlovú rýchlosť osi X, čo zodpovedá rýchlosti otáčania senzora pri „naklonení doľava/doprava“ (smer otáčania červenej šípky na obrázku);

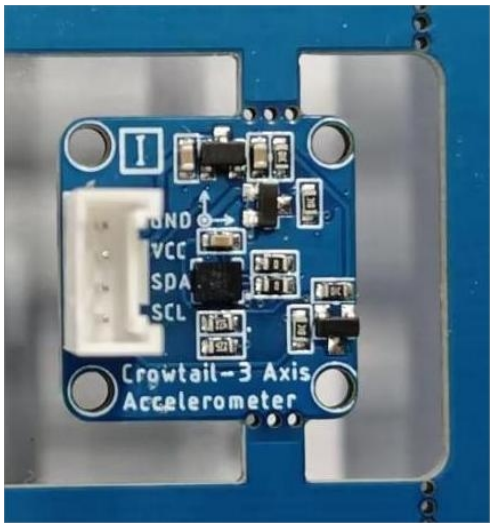
Otáčanie okolo osi Y (Pitch, sklon): Gyroskop detekuje uhlovú rýchlosť osi Y, čo zodpovedá rýchlosti otáčania senzora „naklonenie dopredu/dozadu“ (smer otáčania zelenou šípkou na obrázku);

Otáčanie okolo osi Z (Yaw, yaw): Gyroskop detekuje uhlovú rýchlosť osi Z, čo zodpovedá rýchlosti otáčania senzora „horizontálneho otočenia“ (smer otáčania modrej šípky na obrázku).



2. Požadované moduly

Modul 3-osového akcelerometra Crowtail × 1

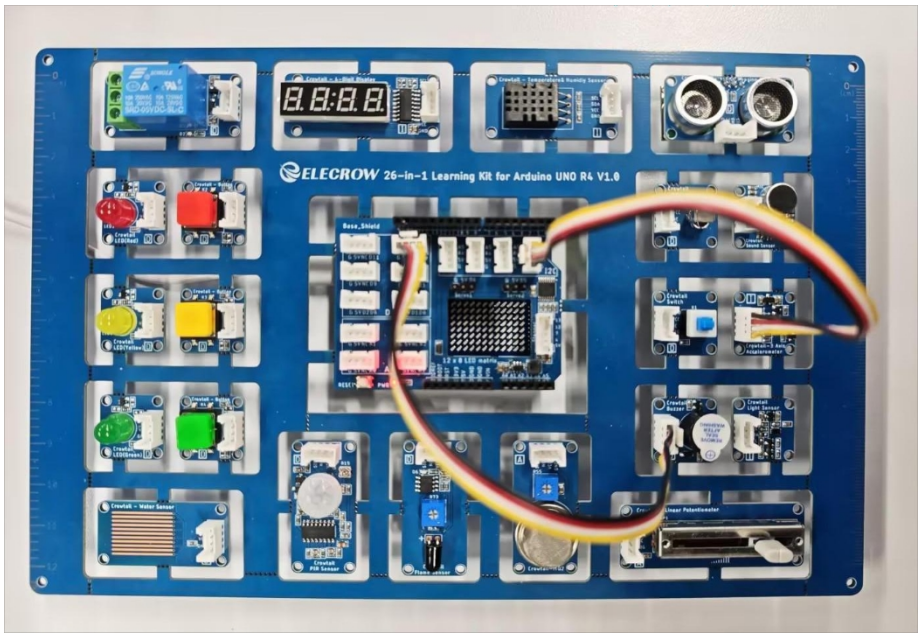


Modul bzučiaka Crowtail × 1

3. Spôsob zapojenia

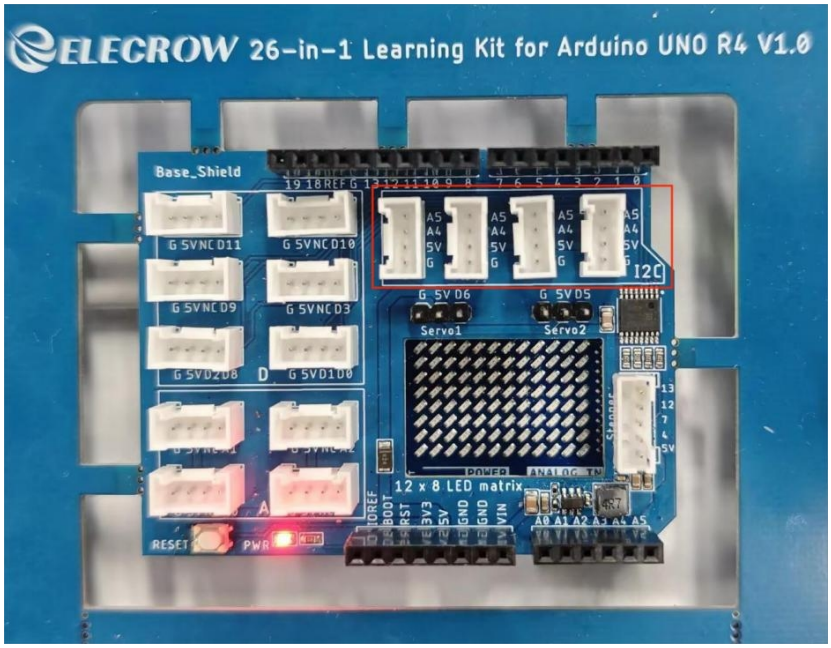
Senzor 3-osového akcelerometra Crowtail → ľubovoľný port I2C

Bzučiak Crowtail → port DIGITAL D10



Špecifikácia 4-portového rozhrania Crowtail:

- GND (čierny) → GND
- VCC (červený) → 5 V
- SDA (biela) → A4
- SCL (žltá) → A5



4. Príklad vysvetlenia

Pripravili sme pre vás dve sady kódov, ktoré vám pomôžu lepšie porozumieť a naučiť sa.

Prvý kód: „19_LSM6DS3_1“ – Základný kód, ktorý vysvetľuje, ako získať údaje zo šiestich osí. **Druhý kód:** „19_LSM6DS3_2“ – Kód hlavnej funkcie tejto lekcie, ktorý popisuje, ako implementovať funkciu alarmu proti spätnému chodu.

Kliknutím na odkaz si stiahnite oficiálny vzorový kód:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-19_LSM6DS3_1/19_LSM6DS3_1

„19_LSM6DS3_1“

Najskôr otvorte program kurzu nachádzajúci sa v priečinku „19_LSM6DS3_1“ pomocou Arduino IDE.

19_LSM6DS3_1.ino

```
#include <Arduino_LSM6DS3.h>
#include <MadgwickAHRS.h>

// Defines
#define SAMPLE_RATE 10  // in Hz

// Constructors
Madgwick filter;  // Madgwick algorithm for roll, pitch, and yaw calculations

void setup() {
  Serial.begin(115200);  // initialize serial bus (Serial Monitor)
  while (!Serial);  // wait for serial initialization
  Serial.print("LSM6DS3 IMU initialization ");
  if (IMU.begin()) {  // initialize IMU
    Serial.println("completed successfully.");
  } else {
    Serial.println("FAILED.");
    IMU.end();
    while (1);
  }
  Serial.println();
  filter.begin(SAMPLE_RATE);  // initialize Madgwick filter
}

void loop() {
  static unsigned long previousTime = millis();
  unsigned long currentTime = millis();
  if (currentTime - previousTime >= 1000/SAMPLE_RATE) {
    printValues();
    // printRotationAngles();
    previousTime = millis();
  }
}

// Prints IMU values.
void printValues() {
  float ax, ay, az;
  float gx, gy, gz;

  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
      && IMU.readAcceleration(ax, ay, az)
      && IMU.readGyroscope(gx, gy, gz)) {

```

Vysvetlenie kódov

(1) Použité súbory knižnice

```
#include <Arduino_LSM6DS3.h>
#include <MadgwickAHRS.h>
```

Úlohou týchto dvoch riadkov, **#include <Arduino_LSM6DS3.h>** a **#include <MadgwickAHRS.h>**, je zaviesť „externé funkcie“ na začiatku programu: Prvý riadok importuje oficiálnu knižnicu senzorov Arduino **LSM6DS3**, ktorá zapuzdruje komunikáciu medzi integrovaným IMU (akcelerometer + gyroskop) cez I2C/SPI a čítanie zrýchlenia a uhlovej rýchlosti a ďalších detailov na nízkej úrovni, čo nám umožňuje priamo používať funkcie ako **IMU.begin()**, **IMU.readAcceleration()** a **IMU.readGyroscope()** na získanie údajov;

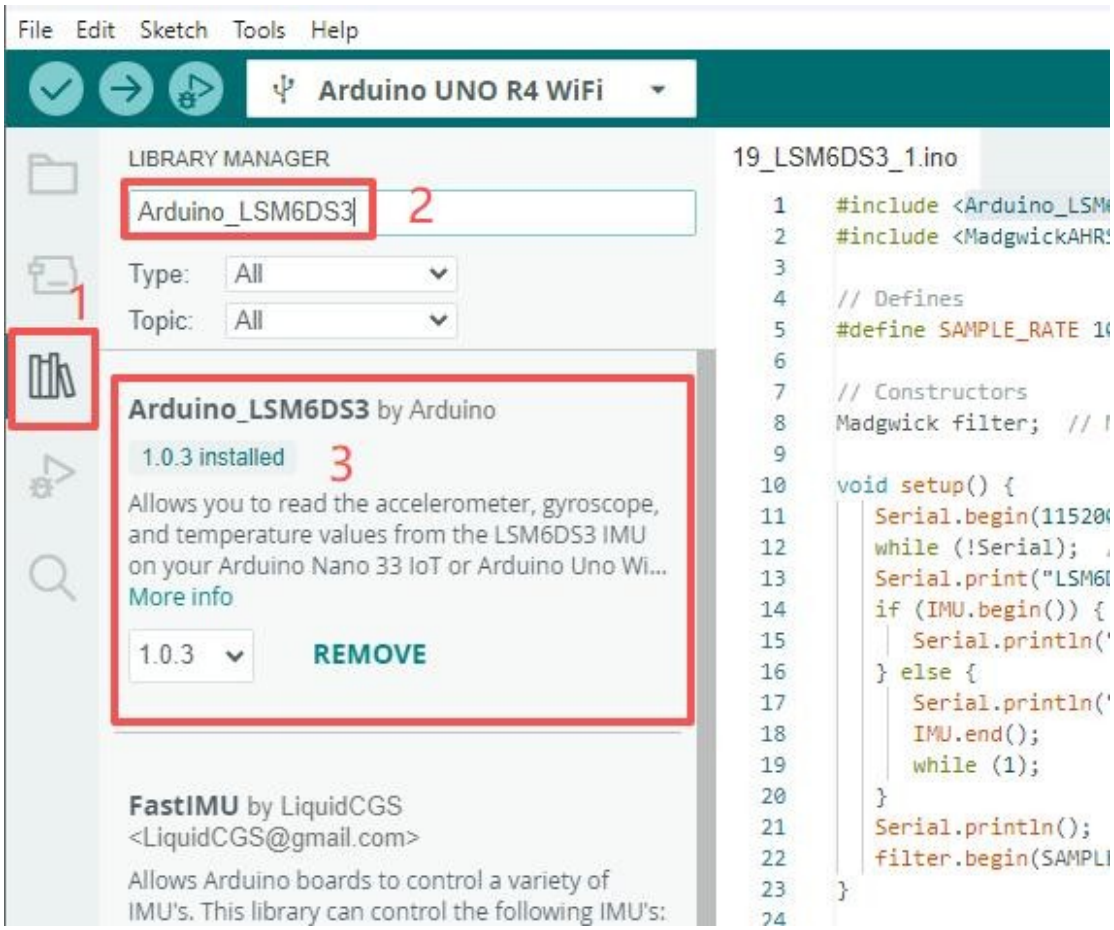
Druhý riadok importuje knižnicu algoritmov MadgwickAHRS, ktorá nezabezpečuje čítanie senzorov, ale slúži špecificky na fúziu údajov z akcelerometra a gyroskopu s cieľom získať stabilnejší a realistickejší uhol náklonu (Roll), uhol sklonu (Pitch) a uhol smeru (Yaw).

Tieto dva riadky #include teda v skutočnosti zodpovedajú za „zber surových údajov o pohybe“ a za „premenu surových údajov na zmysluplné uhly polohy“, čo je základom fungovania celého programu na výpočet polohy.

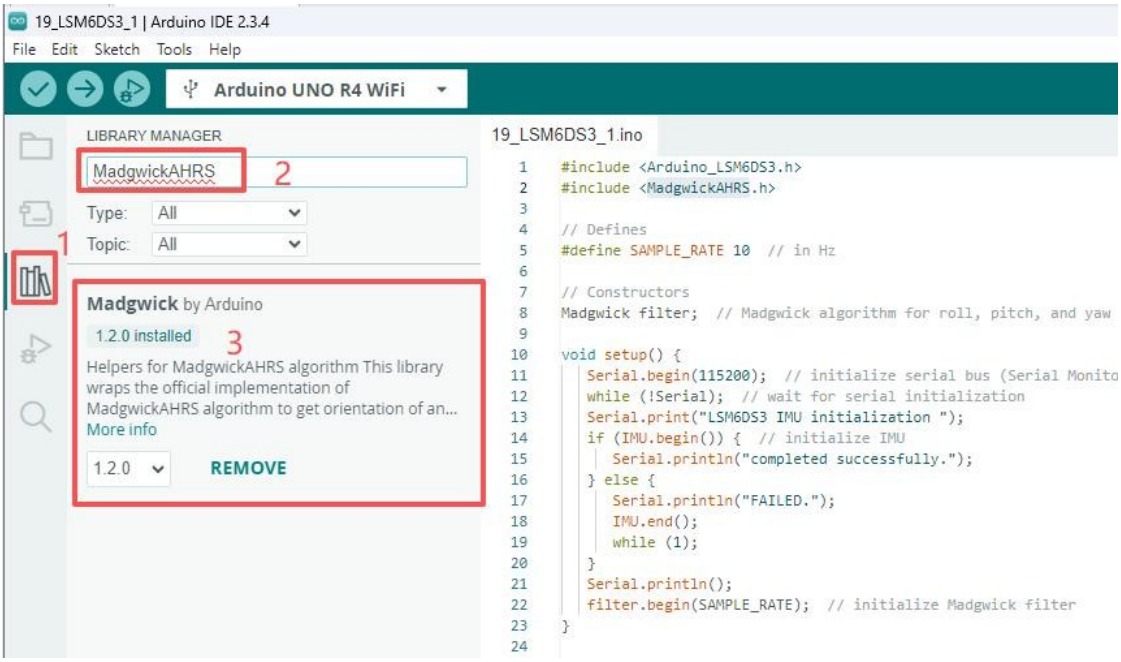
Na použitie týchto dvoch knižníc je možné zvoliť nasledujúce postupy:

➢ Stiahnutie v prostredí Arduino IDE

Verzia knižnice Arduino_LSM6DS3, ktorú používame, je 1.0.3.



Verzia knižnice MadgwickAHRS, ktorú používame, je 1.2.0.



(2) define

#define SAMPLE_RATE 10

Táto definícia makra oznamuje programu: „Chceme spracovávať údaje z IMU s frekvenciou 10 Hz (10-krát za sekundu)“. Neskôr, v cykle loop(), sa použije 1000 / SAMPLE_RATE, čo prevádza túto hodnotu „koľkokrát za sekundu“ na „koľko milisekúnd trvá jedno vykonanie“. Výhodou tohto prístupu je, že zmenou tejto jedinej hodnoty je možné ovládať celkovú rýchlosť vzorkovania a výpočtu, čo uľahčuje pochopenie a údržbu.

(3) Vytvorenie objektu

```
Madgwickov filter;
```

Madgwickov filter; Bol vytvorený objekt na odhad polohy Madgwick s názvom „filter“. Môžete si ho predstaviť ako „matematickú sadu nástrojov špeciálne navrhnutú na výpočet polohy“. Interné ukladá aktuálne stavy naklonenia, sklonu a odklonu a poskytuje funkcie ako begin(), updateIMU() a getRoll(). Zakaždým, keď nasledujúci program prečíta nové údaje o zrýchlení a gyroskopu, odovzdá ich tomuto filtru na výpočet fúzie. Tento riadok teda „nevypočítava výsledok“, ale skôr vopred pripravuje nepretržite fungujúce jadro pre výpočet polohy.

(4) Časť „setup“

```
void setup() {
    Serial.begin(115200);        // inicializácia sériovej zbernice (sériový
    monitor) while (!Serial);    // čakanie na inicializáciu sériového
    rozhrania Serial.print("Inicializácia IMU LSM6DS3 ");
    if (IMU.begin()) {       // inicializácia IMU
        Serial.println("úspešne dokončené.");
    } else {
        Serial.println("NEÚSPECH.");
        IMU.end();
        while (1);
    }
    Serial.println();
    filter.begin(SAMPLE_RATE);        // inicializácia Madgwickovho filtra
}
```

Funkciu tohto kódu setup() možno chápať takto: „Pred spustením programu sa naraz pripravia všetky potrebné hardvérové komponenty a algoritmy.“

①Informácie o tlači cez sériový port

```
    Serial.begin(115200);        // inicializácia sériovej zbernice (sériový monitor)
```

```
    while (!Serial);      // čakanie na inicializáciu sériovej komunikácie
    Serial.print("Inicializácia LSM6DS3 IMU");
```

Najskôr sa pomocou príkazu **Serial.begin(115200)** otvorí sériová komunikácia a nastaví prenosová rýchlosť na **115 200**, aby sme mohli vidieť ladiace informácie v sériovom monitore; následne sa vykoná nasledujúci príkaz „**while** (!Serial)“. Ide o čakací mechanizmus, ktorý zabezpečuje, že sériový port je skutočne pripravený, než program pokračuje ďalej, čím sa zabráni situácii, keď sa informácie nemôžu zobraziť ihneď po zapnutí.

②Inicializácia akcelerometra

```
    if (IMU.begin()) {      // inicializácia IMU
        Serial.println("úspešne dokončené.");
    }
```

Tento úsek kódu je typickým príkladom inicializácie hardvéru vo vývoji vstavaných systémov. Jeho podstatou je vykonanie navrhnutých inicializačných kontrol a spracovanie spätnej väzby pre senzor IMU (LSM6DS3): Funkcia **IMU.begin()** slúži ako inicializačná funkcia špecifická pre daný hardvér, nie ako bežné logické volanie. Vykoná kľúčové operácie, ako je detekcia fyzickej prítomnosti senzora, overenie komunikačného spojenia I2C/SPI, prebudenie čipu a základná konfigurácia; Vzhľadom na nekontrolovateľné faktory, ako sú chyby v zapojení, poškodenie čipu a abnormality v napájaní v hardvérovom prostredí, je táto funkcia navrhnutá tak, aby vracala booleovskú hodnotu (true označuje úspešnú inicializáciu, false označuje zlyhanie). Preto sa v kóde výsledok posudzuje pomocou if **(IMU.begin())**, čím sa vyhýba idealizovanému predpokladu, že „predvolená inicializácia musí byť úspešná“;
Vetva „if“ vykonáva iba operáciu vytlačenia správy „Inicializácia úspešná“ na sériový port; hoci to nie je funkčná požiadavka programu, ide o konštrukčný prvok slúžiaci na spätnú väzbu pri ladení pre vývojárov – umožňuje priamo overiť základný stav hardvéru senzora, komunikačného spojenia a knižnice ovládača, čím pomáha rýchlo odstraňovať základné chyby.

```
else {
        Serial.println("FAILED.");
        IMU.end();
        while (1);
    }
```

Vetva else však skutočne stelesňuje „profesionálne myslenie“ v oblasti vývoja vstavaných systémov: ak inicializácia zlyhá, program nebude pokračovať a „predstierať, že sa nič nestalo“. Vytlačením „FAILED.“ na začiatku sa môžete okamžite dozvedieť, že problém spočíva vo fáze inicializácie, namiesto toho, aby ste neskôr pri čítaní údajov narazili na nevysvetliteľnú chybu; následné volanie **IMU.end()** je čistiaca akcia, ktorá naznačuje „keďže inicializácia nebola úspešná, je jasné, že

uvolniť/zastaviť tento hardvérový objekt“, aby sa zabránilo tomu, že systém zostane v nebezpečnom **„čiastočne inicializovanom“**

;

Záverečná slučka **while (1);** je zámerne napísaná mŕtva slučka. Jej úlohou nie je spôsobiť zamrznutie programu, ale jasne oznámiť systému: tento program už nespĺňa predpoklady na ďalšiu prevádzku, pretože celý nasledujúci kód závisí od údajov z IMU. Ak inicializácia IMU zlyhá, ďalšie spúšťanie programu bude viesť len k generovaniu chybných údajov alebo k zavádzajúcim výsledkom pri ladení.

(5) Inicializácia algoritmu Madgwick

```
filter.begin(SAMPLE_RATE);        // inicializovať Madgwickov filter
```

Podstatou tohto riadku kódu je nastavenie referenčného času pre algoritmus odhadu polohy Madgwick: **Madgwick** nie je senzor, ale matematický algoritmus, ktorý prevádza surové údaje z gyroskopov a akcelerometrov na fyzikálne uhly. Na výpočet uhlov sa spolieha na integráciu uhlovej rýchlosti (gyroskop poskytuje výstup „uhol otáčania za sekundu“, ktorý je potrebné kombinovať s časovým intervalom na výpočet skutočného uhla), takže časový krok je základnou podmienkou pre presné fungovanie algoritmu. V **„filter.begin(SAMPLE_RATE)“** je **SAMPLE_RATE** plánovaná frekvencia aktualizácie údajov IMU (napríklad 10 Hz znamená aktualizáciu raz za 100 ms); volanie tohto rozhrania slúži na informovanie algoritmu, že „dáta budú vkladané s touto pevnou frekvenciou a všetky vnútorné výpočty časových krokov sú na tom založené“; ak sa vynechá alebo nastaví nesprávne, aj keď sú údaje z IMU presné, odhadované uhly budú vykazovať odchýlky, kolísanie a iné skryté chyby. Okrem toho inicializácia hardvéru IMU (**„IMU.begin()“**) rieši otázku „či je možné získať údaje“, zatiaľ čo inicializácia tohto algoritmu rieši otázku „či je možné údaje správne analyzovať“; obe musia byť dokončené v „setup()“ a sú základnými predpokladmi pre spustenie systému.

(6) Vstup do funkcie loop() – logika, ktorá sa neustále opakuje

Táto funkcia loop() je ako „srdcový tep“ programu, neustále sa opakuje a vykonáva.

```
void loop() {
    static unsigned long previousTime = millis();
    unsigned long currentTime = millis();
    if (currentTime - previousTime >= 1000/SAMPLE_RATE) { printValues();
        // printRotationAngles();
        previousTime = millis();
    }
```

```
}
```

①časová pečiatka

```
25  void loop() {
26      static unsigned long previousTime = millis();
27      unsigned long currentTime = millis();
28      if (currentTime - previousTime >= 1000/SAMPLE_RATE) {
29          printValues();
30          // printRotationAngles();
31          previousTime = millis();
32      }
33  }
```

Tento riadok kódu stelesňuje základnú myšlienku „využívania časových značiek na zaznamenanie minulosti a aktuálneho času na porovnanie“: výraz **„currentTime = millis()“** slúži na zistenie, „koľko milisekúnd uplynulo od spustenia programu až do súčasného okamihu“, čím reprezentuje aktuálny časový bod; zatiaľ **čo „static unsigned long previousTime = millis()“** používa kľúčové slovo **„static**“, aby sa táto premenná inicializovala len raz pri prvom vstupe do funkcie **„loop()“**, a potom aj v prípade, že sa funkcia „loop()“ vykonáva opakovane, vždy si zachová „časovú pečiatku, kedy bola vykonaná predchádzajúca úloha“, a nebude sa opakovane resetovať; práve prostredníctvom najzákladnejšieho výpočtu **„aktuálny čas – predchádzajúci čas“** môže program určiť, „či nastal čas“, čím sa dosiahne stabilná, kontrolovateľná a neblokujúca logika časového vykonávania. Tento spôsob zápisu je východiskovým bodom a základom všetkých časových mechanizmov založených na **„millis()**“.

② Prečítajte hodnoty

```
25  void loop() {
26      static unsigned long previousTime = millis();
27      unsigned long currentTime = millis();
28      if (currentTime - previousTime >= 1000/SAMPLE_RATE) {
29          printValues();
30          // printRotationAngles();
31          previousTime = millis();
32      }
33  }
```

Najprv sa pozrime na samotnú podmienku:

Krok **„currentTime – previousTime“** vypočítava, „koľko milisekúnd uplynulo od posledného vykonania hlavnej úlohy“. Tu sa používa millis(), čo je systémové hodiny poskytované Arduinom, ktoré udávajú, koľko milisekúnd uplynulo od zapnutia, a nie funkcia oneskorenia. Tento prístup „aktuálny čas – zaznamenaný čas“ je najklasickejší a najspoľahlivejší spôsob posudzovania času v embedded systémoch.

Potom sa pozrite na hodnotu 1000 / SAMPLE_RATE na pravej strane:

Jednotkou **SAMPLE_RATE** je Hz (koľkokrát za sekundu), zatiaľ čo millis() má jednotku milisekúnd. Kód teda musí najprv vykonať „prepočet jednotiek“ – 1 sekunda sa rovná 1000 milisekúnd, takže **1000 / SAMPLE_RATE** dáva počet milisekúnd, ktoré by mali uplynúť medzi dvoma vzorkami. Napríklad, ak **je SAMPLE_RATE = 10**, výsledok je 100 milisekúnd, čo znamená „vykonaj logiku vzorkovania každých 100 ms“.

Keď je podmienka splnená, program vstúpi do tela príkazu if:

printValues(); Ide o skutočný „funkčný kód“, čo znamená čítanie a výstup údajov z IMU v správnom časovom bode.

Posledný riadok, „previousTime = millis();“, je mimoriadne dôležitý. Jeho funkciou je uložiť „časový bod, kedy bola táto úloha vykonaná“ a použiť ho ako východiskový bod pre ďalšie kolo posudzovania času. Bez tohto riadku bude podmienka vždy pravdivá a kód sa bude zúfalo vykonávať v každom cykle loop(), čo spôsobí, že vzorkovacia frekvencia sa úplne vymkne spod kontroly.

(7) printValues()

```
// Vytlačí hodnoty IMU.
void printValues() {
  float ax, ay, az;
  float gx, gy, gz;
```

Funkcia najskôr definuje šesť premenných typu float: **ax, ay, az, gx, gy a gz.** Dôvodom je to, že údaje o zrýchlení a gyroskopu z IMU sú spojité fyzikálne veličiny a musia sa ukladať pomocou čísel s pohyblivou desatinnou čiarkou. Okrem toho je zaužívané označovať zrýchlenie písmenom „a“, gyroskop písmenom „g“ a x/y/z zodpovedajú trom osiam priestorového smeru.

```
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable() &&
      IMU.readAcceleration(ax, ay, az)
      && IMU.readGyroscope(gx, gy, gz)) {
```

V prvom rade je existencia funkcií **IMU.accelerationAvailable()** a **IMU.gyroscopeAvailable()** „vynútená hardvérom“. IMU typu LSM6DS3 negeneruje nové údaje ihneď po vyvolaní; namiesto toho priebežne aktualizuje registre pri svojej internej vzorkovacej frekvencii. Kód Arduina beží veľmi rýchlo. Ak sa najprv neopýtate „Existujú teraz nejaké nové údaje?“, pravdepodobne prečítate neaktuálne údaje z predchádzajúceho rámca alebo dokonca nedokončené údaje, ktoré neboli úplne aktualizované. Preto je podstatou týchto dvoch funkcií: „Je čip IMU teraz pripravený bezpečne prečítať sadu nových údajov o zrýchlení/gyroskope?“. Ide o typickú detekciu pripravenosti senzora (kontrola pripravenosti údajov), čo je štandardná prax v priemyselnom a robotickom kóde a nie je to voliteľná funkcia.

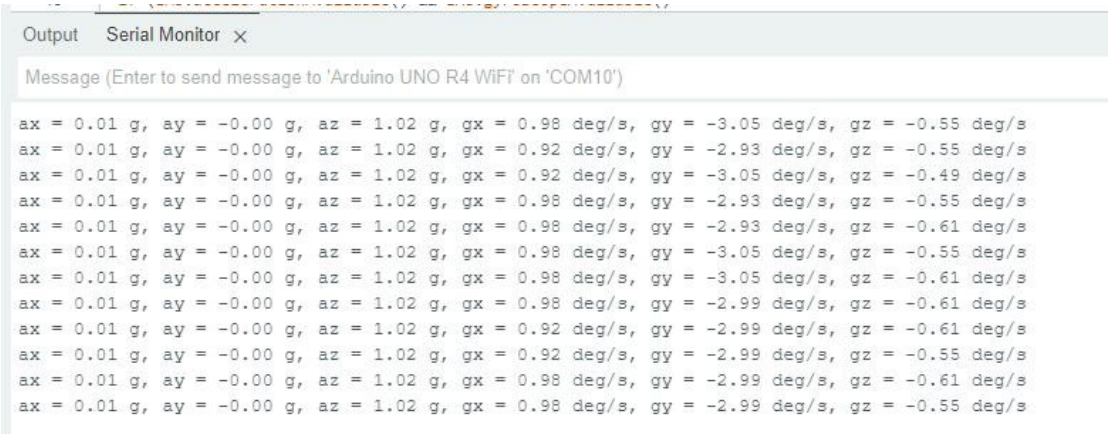
Ďalej nasleduje **IMU.readAcceleration(ax, ay, az)**. Tento riadok neslúži na „výpočet údajov“, ale na načítanie hodnoty zrýchlenia v troch osiach uložené vo vnútorných registroch čipu IMU prostredníctvom zbernice I2C/SPI a zapisuje ich do premenných ax/ay/az, ktoré ste zadali. Tu musíte vopred pripraviť tri premenné typu float a „odovzdať“ ich tejto funkcii. Vo vnútri funkcie sa podľa rozsahu senzora, citlivosti a pravidiel prevodu jednotiek pôvodné hodnoty registrov prevedú na čísla s plávajúcou desatinnou čiarkou už v jednotke g (násobok gravitačného zrýchlenia), preto nemusíte výpočet mierky vykonávať sami. Podobne funguje aj **IMU.readGyroscope(gx, gy, gz)**, s tým rozdielom, že číta register gyroskopu a prevádza ho na deg/s (koľko stupňov za sekundu), čo je formát vstupných údajov, ktorý najviac potrebuje algoritmus na výpočet polohy (napríklad Madgwick).

Spojte tieto štyri funkcie a vložte ich do podmienky „if“. Týmto sa v skutočnosti uplatňuje veľmi dôležitý technický princíp: „Ak zlyhá akýkoľvek krok, tento rámec údajov sa zahodí.“ Napríklad: ak údaje ešte nie sú pripravené, ak nastane chyba v komunikácii I2C, ak zlyhá určitá operácia čítania, celá podmienka „if“ sa nevykoná. Týmto spôsobom nevytlačíte chybné údaje a nepošlete neplatné údaje do algoritmu polohy. Stabilita systému sa výrazne zlepší. Preto vidíte veľmi dlhú podmienku **„if (...) && (...) && (...) && (...)“** namiesto toho, aby ste ju písali samostatne.

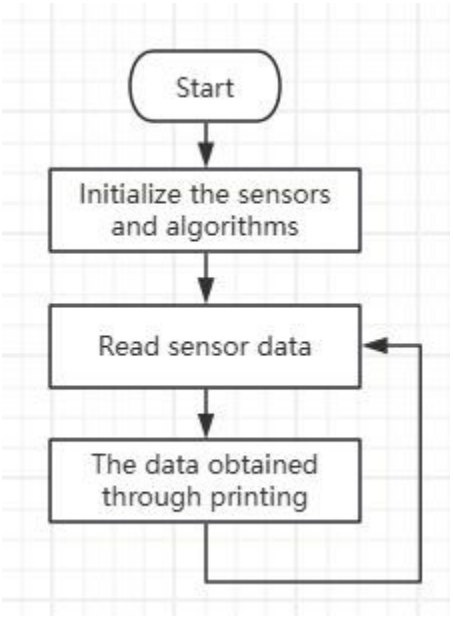
```
        Serial.print("ax = ");
        Serial.print(ax);
        Serial.print(" g, ");

        Serial.print("ay = ");
        Serial.print(ay);
        Serial.print(" g, ");

        Serial.print("az = ");
        Serial.print(az);
        Serial.print(" g, ");

        Serial.print("gx = ");
        Serial.print(gx);
        Serial.print(" deg/s, ");

        Serial.print("gy = ");
```

```
        Serial.print(gy);
        Serial.print(" deg/s, ");


        Serial.print("gz = ");
        Serial.print(gz);
        Serial.println(" deg/s");
    }
}
```

Po vstupe do podmienky if nie sú všetky príkazy Serial.print len náhodnými výstupmi. Namiesto toho sa zámerne pridávajú fyzikálne jednotky: za zrýchlenie sa pridáva „g“ a za uhlovú rýchlosť „deg/s“. Vďaka tomu sa pri zobrazení v sériovom monitore nejedná len o „holé čísla“, ale o namerané hodnoty senzorov s jasným fyzikálnym významom. To je kľúčové pre ladenie a pochopenie zmien v polohe. Nakoniec sa používa **Serial.println()** na ukončenie celého riadku výstupu, čím sa zabezpečí, že každý súbor údajov IMU zaberá samostatný riadok, čo uľahčuje následné pozorovanie a analýzu.



(8) Celkový diagram logického toku kódu



„19_LSM6DS3_2“

Teraz, keď sme pochopili získavanie údajov zo šiestich osí vrátane zrýchlenia a uhlovej rýchlosti, použijeme vedomosti, o ktorých sme práve hovorili, na implementáciu kľúčovej funkcie tejto lekcie: alarm proti prevráteniu.

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-19_LSM6DS3_2/19_LSM6DS3_2

Otvorte program kurzu v priečinku **„19_LSM6DS3_2“** pomocou prostredia Arduino IDE:

19_LSM6DS3_2.ino

```
#include <Arduino_LSM6DS3.h>
#include <MadgwickAHRS.h>

#define BUZ 10                  // Buzzer pin
#define SAMPLE_RATE 10          // IMU sampling rate (Hz)
#define TILT_THRESHOLD 30.0     // Inclination alarm threshold (degree)

Madgwick filter;

void setup() {
  Serial.begin(115200);
  while (!Serial);

  pinMode(BUZ, OUTPUT);
  digitalWrite(BUZ, LOW);

  Serial.print("Initializing LSM6DS3 IMU... ");
  if (!IMU.begin()) {
    Serial.println("FAILED");
    while (1);
  }
  Serial.println("OK");

  filter.begin(SAMPLE_RATE);
}

void loop() {
  static unsigned long lastTime = 0;
  unsigned long now = millis();

  if (now - lastTime >= 1000 / SAMPLE_RATE) {
    checkTilt();
    lastTime = now;
  }
}

void checkTilt() {
  float ax, ay, az;
  float gx, gy, gz;

  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
      && IMU.readAcceleration(ax, ay, az)
      && IMU.readGyroscope(gx, gy, gz)) {

    // Update attitude calculation
    filter.updateIMU(gx, gy, gz, ax, ay, az);
```

Väčšinu kódu sme už vysvetlili. Teraz budeme implementovať túto funkciu s využitím vedomostí, o ktorých sme práve hovorili.

Naše vysvetlenie sa preto bude týkať len kľúčových bodov.

(1) Nastavte prah naklonenia

19_LSM6DS3_2.ino

```
#include <Arduino_LSM6DS3.h>
#include <MadgwickAHRS.h>

#define BUZ 10                  // Buzzer pin
#define SAMPLE_RATE 10          // IMU sampling rate (Hz)
#define TILT_THRESHOLD 30.0     // Inclination alarm threshold (degree)
```

Tu je nastavená prahová hodnota náklonu 30 na následnú detekciu toho, či došlo k pádu. Tento riadok **#define TILT_THRESHOLD 30.0** v podstate vytvára jasnú fyzickú hranicu medzi „nebezpečenstvom“ a „bezpečnosťou“ tým, že umelo definuje jasnú fyzickú prahovú hodnotu pre „mieru náklonu považovanú za abnormálnu“ (30 stupňov), a je definovaná ako globálna konštanta pomocou preprocesora

makro, ktoré poskytuje jednotný referenčný štandard pre celý program pri rozhodovaní o tom, či spustiť alarm. Z pohľadu zdrojového kódu nebol tento riadok napísaný náhodne, ale vychádza z troch kľúčových bodov: Po prvé, hodnoty náklonu a sklonu vypočítané algoritmom Madgwicka sú už „hodnotami uhlov“ s jednotkami stupňov (°), takže prahová hodnota musí byť tiež definovaná v stupňoch; Po druhé, v skutočnosti ľudia a zariadenia zvyčajne považujú naklonenie do 30° za prijateľné a prekročenie 30° často znamená pád, prevrátenie alebo nebezpečnú polohu, čo vychádza skôr z technických skúseností ako z gramatických požiadaviek; Po tretie, použitie #define namiesto priameho pevného zakódovania čísla v if (abs(roll) > 30.0) je veľmi profesionálny spôsob písania, pretože oddeľuje „samotné pravidlo“ od „logiky posudzovania“ – odteraz stačí upraviť len tento riadok, aby sa celý systém premenil z „citlivého alarmu“ na „pomalý alarm“ bez nutnosti upravovať akýkoľvek kód posudzovania.

(2) filter.updateIMU(gx, gy, gz, ax, ay, az);

19_LSM6DS3_2.ino

```
  Serial.println("OK");

  filter.begin(SAMPLE_RATE);
}

void loop() {
  static unsigned long lastTime = 0;
  unsigned long now = millis();

  if (now - lastTime >= 1000 / SAMPLE_RATE) {
    checkTilt();
    lastTime = now;
  }
}

void checkTilt() {
  float ax, ay, az;
  float gx, gy, gz;

  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
      && IMU.readAcceleration(ax, ay, az)
      && IMU.readGyroscope(gx, gy, gz)) {

    // Update attitude calculation
    filter.updateIMU(gx, gy, gz, ax, ay, az);

    float roll  = filter.getRoll();
    float pitch = filter.getPitch();

    Serial.print("Roll: ");a
    Serial.print(roll);
    Serial.print("°, Pitch: ");
    Serial.print(pitch);
    Serial.print("°  ");

    // Tilt detection
    if (abs(roll) > TILT_THRESHOLD || abs(pitch) > TILT_THRESHOLD) {
      digitalWrite(BUZ, HIGH);
      Serial.println("!!! TILT ALARM !!!");
    } else {
      digitalWrite(BUZ, LOW);
      Serial.println("Status: SAFE");
    }
  }
}
```

filter.updateIMU(gx, gy, gz, ax, ay, az); Toto je základný kód systému detekcie naklonenia. Jeho

Podstatou je dosiahnutie fúzie senzorov a výpočtu polohy pomocou algoritmu **Madgwick AHRS**: Tento algoritmus sa opiera o technológiu výpočtu polohy a fúzie senzorov, aby kompenzoval nedostatky jediného senzora – akcelerometer (ax/ay/az) dokáže snímať iba smer gravitácie a je presný v kľude, ale náchylný na rušenie pohybom; gyroskop (gx/gy/gz) dokáže merať rýchlosť otáčania a je presný počas krátkeho obdobia, ale náchylný na časový posun.

V tomto riadku kódu algoritmus používa ako referenčnú hodnotu vzorkovaciu frekvenciu nastavenú pomocou **filter.begin(SAMPLE_RATE)**. Predvolene je časový interval medzi susednými údajmi pevne stanovený. Zadávajú sa surové údaje o uhlovej rýchlosti a zrýchlení a aktualizuje sa vnútorný stav polohy (kvaternion) namiesto priameho výstupu uhlov **náklonu a sklonu** (ktoré je potrebné neskôr prečítať pomocou **filter.getRoll()/filter.getPitch())**.

Hlavný význam tohto kódu spočíva v premenení IMU, ktorá výstupuje iba surové údaje, na „senzor polohy“, ktorý môže výstupovať stabilné polohy; a táto funkcia musí byť volaná nepretržite s pevnou frekvenciou. Ak chýba volanie alebo je frekvencia nestabilná, vypočítané uhly budú vykazovať kolísanie, posun alebo dokonca úplné chyby.

(3) Získanie hodnoty po spracovaní algoritmom

```
37  void checkTilt() {
38    float ax, ay, az;
39    float gx, gy, gz;
40
41    if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
42        && IMU.readAcceleration(ax, ay, az)
43        && IMU.readGyroscope(gx, gy, gz)) {
44
45      // Update attitude calculation
46      filter.updateIMU(gx, gy, gz, ax, ay, az);
47
48      float roll  = filter.getRoll();
49      float pitch = filter.getPitch();
50
51      Serial.print("Roll: ");a
52      Serial.print(roll);
53      Serial.print("°, Pitch: ");
54      Serial.print(pitch);
55      Serial.print("°  ");
56
57      // Tilt detection
58      if (abs(roll) > TILT_THRESHOLD || abs(pitch) > TILT_THRESHOLD) {
59        digitalWrite(BUZ, HIGH);
60        Serial.println("!!! TILT ALARM !!!");
61      } else {
62        digitalWrite(BUZ, LOW);
63        Serial.println("Status: SAFE");
64      }
65    }
66  }
```

Tieto dva riadky kódu, „float roll = filter.getRoll()“ a „float pitch = filter.getPitch()“, v podstate vykonávajú veľmi dôležitú, ale aj veľmi „efektívnu“ úlohu: prevádzajú „matematický stav polohy“, ktorý vnútorne udržiava Madgwickov algoritmus, na „hodnoty uhlov, ktoré

ľudia môžu priamo pochopiť a používať“. Z hľadiska toho, ako je kód „navrhnutý“, funkcia **„filter.updateIMU(...)“**, ktorú ste volali skôr, okamžite nevypočíta uhly a neposkytne vám ich; namiesto toho interne nepretržite aktualizuje kvaternionový model polohy (toto je najstabilnejšia a najprofesionálnejšia metóda reprezentácie pre výpočet polohy).

A tieto dva riadky, **„getRoll()“** a **„getPitch()“,** sú „rozhranie funkcie“, ktoré autor knižnice už pre vás zapuzdril, zodpovedné za skrytie všetkých zložitých matematických operácií s kvaternionmi (rotačné matice, inverzné trigonometrické funkcie atď.) a nakoniec vracajúce iba číslo s plávajúcou desatinnou čiarkou s jednotkou **„stupňov (°)**“.

Z fyzikálneho hľadiska sa „roll“ vzťahuje na uhol naklonenia zariadenia vľavo-vpravo okolo osi X (napríklad naklonenie doľava alebo doprava) a „pitch“ sa vzťahuje na uhol naklonenia zariadenia dopredu-dozadu okolo osi Y (napríklad sklonenie hlavy dopredu alebo zdvihnutie dozadu). To sú presne dva smery, ktoré vás najviac zaujímajú v „alarmu naklonenia“.

(4) Na určenie stavu pádu

```
37  void checkTilt() {
38    float ax, ay, az;
39    float gx, gy, gz;
40
41    if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
42        && IMU.readAcceleration(ax, ay, az)
43        && IMU.readGyroscope(gx, gy, gz)) {
44
45      // Update attitude calculation
46      filter.updateIMU(gx, gy, gz, ax, ay, az);
47
48      float roll  = filter.getRoll();
49      float pitch = filter.getPitch();
50
51      Serial.print("Roll: ");a
52      Serial.print(roll);
53      Serial.print("°, Pitch: ");
54      Serial.print(pitch);
55      Serial.print("°  ");
56
57      // Tilt detection
58      if (abs(roll) > TILT_THRESHOLD || abs(pitch) > TILT_THRESHOLD) {
59        digitalWrite(BUZ, HIGH);
60        Serial.println("!!! TILT ALARM !!!");
61      } else {
62        digitalWrite(BUZ, LOW);
63        Serial.println("Status: SAFE");
64      }
65    }
66  }
67
```

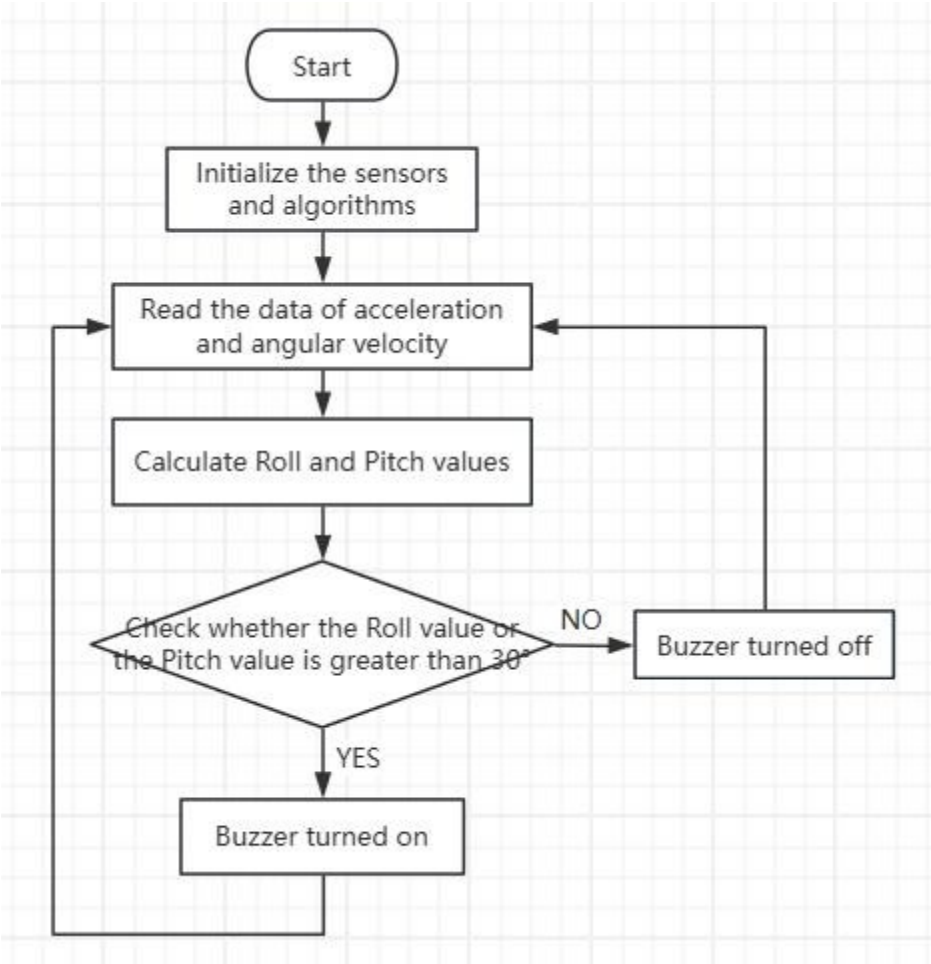
(abs(roll)) a **(abs(pitch))** používajú funkciu absolútnej hodnoty. Dôvodom je to, že bez ohľadu na to, či sa zariadenie nakláňa doľava alebo doprava, dopredu alebo dozadu, pokiaľ

„veľkosť“ uhla naklonenia prekročí prahovú hodnotu, považuje sa to za nebezpečné; samotný smer nie je dôležitý. **TILT_THRESHOLD** je „bezpečná hranica“, ktorú ste vopred nastavili a ktorá predstavuje maximálny uhol naklonenia, ktorý systém povoľuje.

Vý**znam logického operátora „||“ (alebo) v tomto kontexte je:** Ak sa vyskytne akýkoľvek problém v ktoromkoľvek smere, celý systém sa považuje za nebezpečný. To je v plnom súlade s konštrukčným princípom skutočných poplachových systémov, ktorý znie: „lepšie je hlásiť viac, ako nehlásiť vôbec“.

Keď je podmienka splnená, program prejde do vetvy if. Príkaz „digitalWrite(BUZ, HIGH)“ priamo aktivuje výstupný pin bzučiaka, čím sa zariadenie okamžite ozve. Ide o najpriamejšiu a najspoľahlivejšiu metódu riadenia výstupu v reálnom čase; zároveň sa cez sériový port vypíše hlásenie „!!! TILT ALARM !!!“. Ide o ladiacu funkciu a indikáciu stavu určenú pre „ľudí“. Naopak, vo vetve else, keď poloha zariadenia zostáva v bezpečnom rozsahu, bzučiak sa násilne vypne a vypíše „Status: SAFE“, čím jasne informuje systém a používateľov, že sa momentálne nachádzajú v normálnom stave.

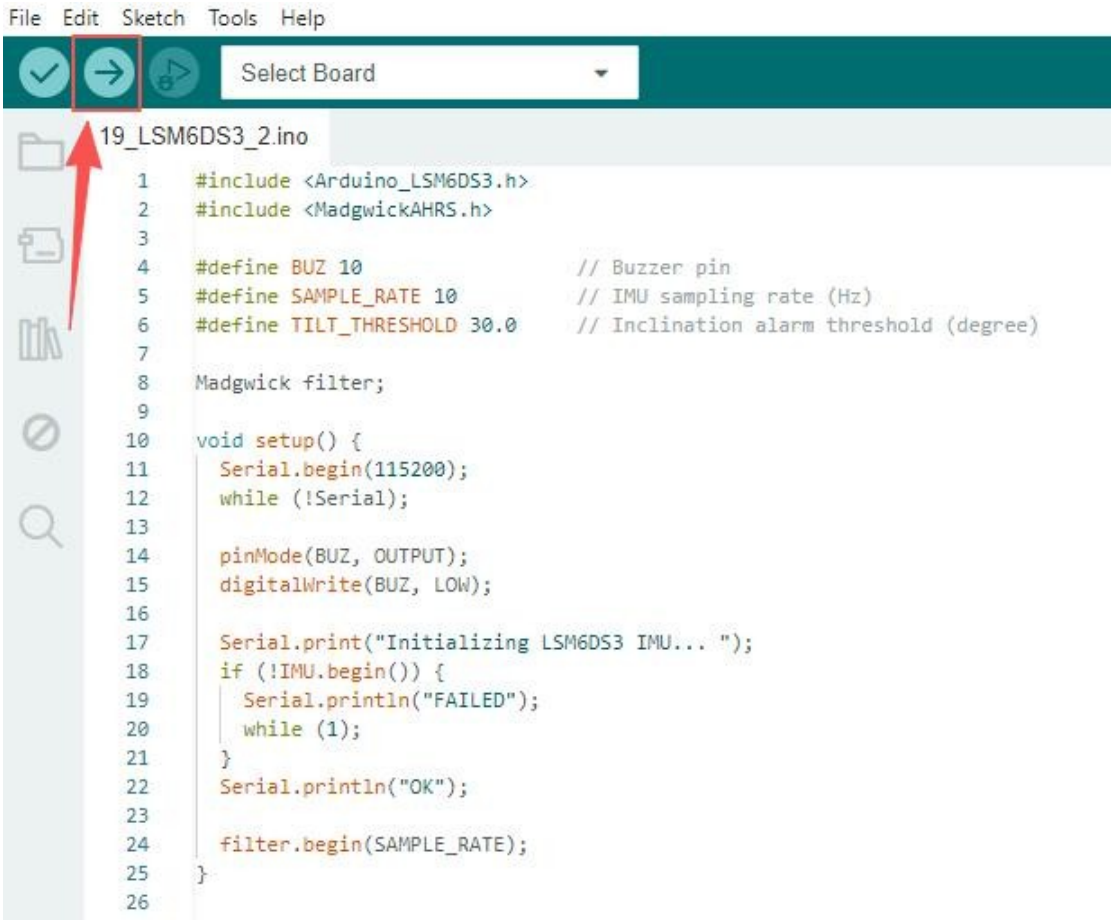
(5) Logický tok kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov obsluhy Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahratia.

Kliknite na „Download“:



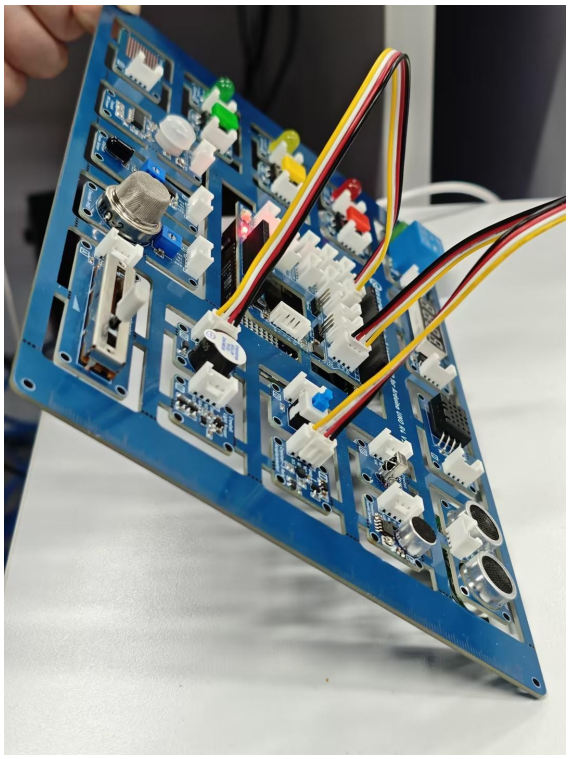
(2) Uvidíte:

Systém bude nepretržite monitorovať orientáciu zariadenia s frekvenciou 10 Hz (10-krát za sekundu):

Keď uhol náklonu alebo sklonu zariadenia presiahne 30°:

(Tu môžete dočasne považovať uhol náklonu za otáčanie pozdĺž dlhej strany a uhol sklonu za otáčanie pozdĺž krátkej strany)

➢ Bzučiak vydáva nepretržitý výstražný zvuk (to je vtedy, keď uhol náklonu presiahne 30°)



➢ Výstup sériového portu: !!! TILT ALARM !!!

```
37    void checkTilt() {
```

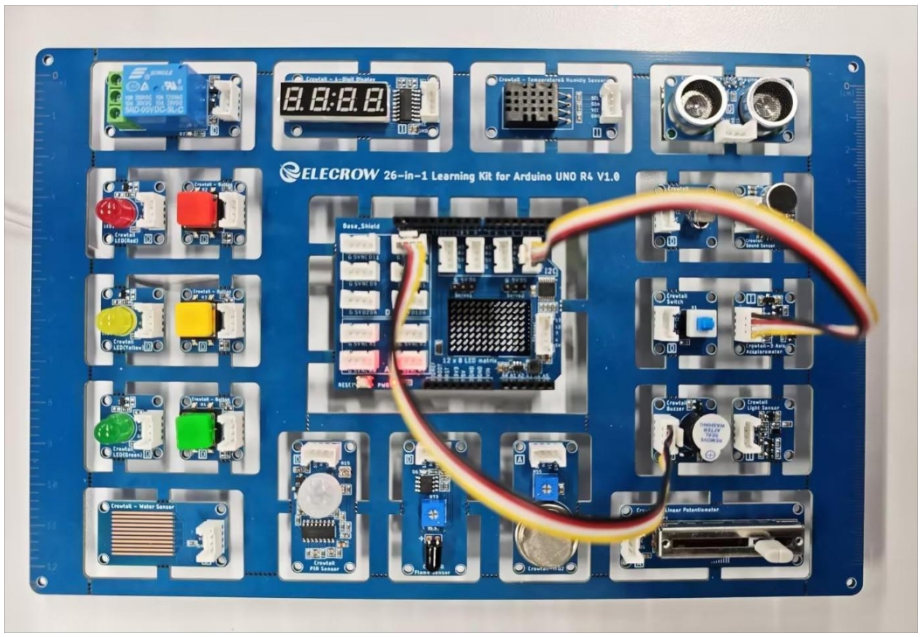
Output Serial Monitor ×

Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM10')

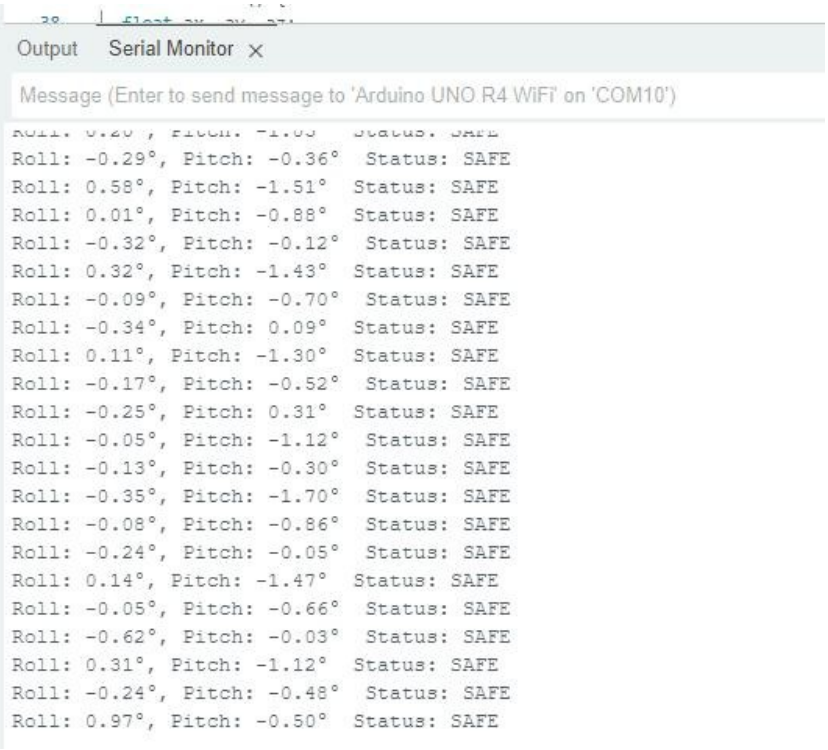
```
Roll: 0.61°, Pitch: -1.39°  Status: SAFE
Roll: -0.07°, Pitch: -0.86°  Status: SAFE
Roll: -0.47°, Pitch: -0.13°  Status: SAFE
Roll: 0.19°, Pitch: -1.44°  Status: SAFE
Roll: -0.22°, Pitch: -0.71°  Status: SAFE
Roll: -0.40°, Pitch: 0.09°  Status: SAFE
Roll: 0.41°, Pitch: -1.06°  Status: SAFE
Roll: -0.51°, Pitch: -1.01°  Status: SAFE
Roll: 2.10°, Pitch: 1.72°  Status: SAFE
Roll: 10.40°, Pitch: 2.56°  Status: SAFE
Roll: 15.53°, Pitch: 3.16°  Status: SAFE
Roll: 22.82°, Pitch: 2.97°  Status: SAFE
Roll: 27.34°, Pitch: 2.14°  Status: SAFE
Roll: 31.02°, Pitch: 0.48°  !!! TILT ALARM !!!
Roll: 33.90°, Pitch: -0.62°  !!! TILT ALARM !!!
Roll: 36.29°, Pitch: -0.26°  !!! TILT ALARM !!!
Roll: 39.68°, Pitch: -0.21°  !!! TILT ALARM !!!
Roll: 40.18°, Pitch: 0.62°  !!! TILT ALARM !!!
Roll: 41.44°, Pitch: 0.23°  !!! TILT ALARM !!!
Roll: 40.32°, Pitch: -0.07°  !!! TILT ALARM !!!
Roll: 39.32°, Pitch: 0.29°  !!! TILT ALARM !!!
```

Keď je zariadenie v bezpečnom prevádzkovom rozsahu:

➢ Zvukový signál nefunguje



➢ Sériový port zobrazuje aktuálny uhol náklonu a výstupy: Stav: SAFE



Lekcia 20 – Servo

Úvod

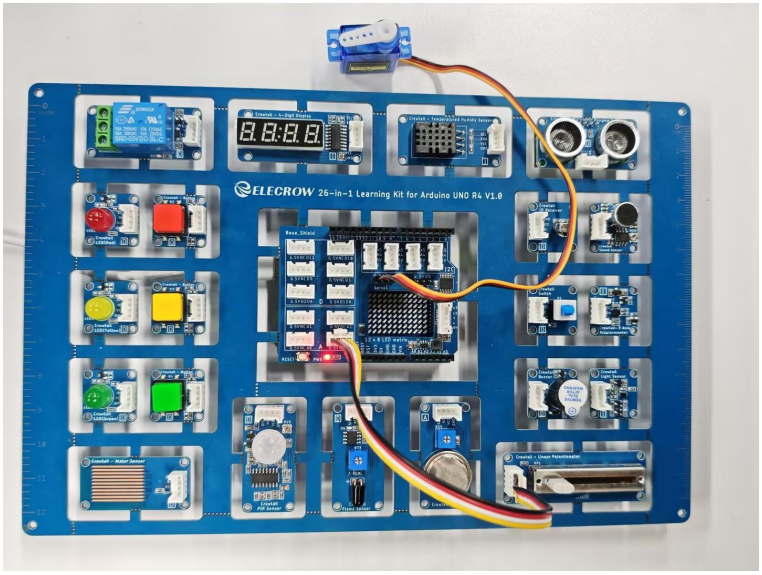
V tejto lekcii sa naučíme, ako pomocou posuvného reostatu ovládať uhol otáčania servomotora v reálnom čase. Na konci tejto lekcie už nebudete len „nechať servomotor otáčať sa sám od seba“, ale skutočne dosiahnete manuálne ovládanie uhla serva.

Ciele

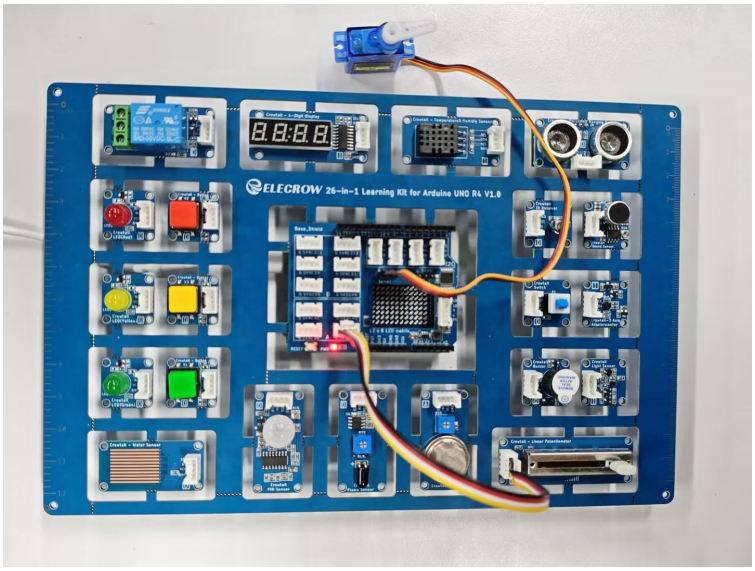
1. Porozumieť základnému princípu fungovania servomotora.
2. Naučte sa inicializovať a ovládať servomotor pomocou knižnice Servo.
3. Porozumejte vzťahu medzi funkciou analogRead() a rozsahom uhla servomotora.
4. Upevnite si používanie funkcie map() na numerické mapovanie.
5. Dokončite úlohu tejto lekcie: ovládanie uhla servomotora pomocou posuvného reostatu.

Náhľad výsledku

Systém bude nepretržite čítať zmeny polohy posuvného reostatu: Keď je posuvný reostat vľavo, aktuálny uhol servomotora je 0



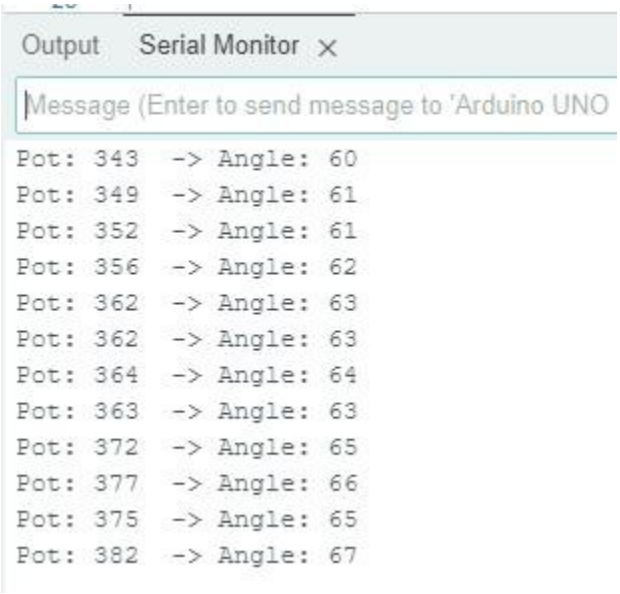
Keď je posuvný rezistor v pravej polohe, uhol servomotora je v tomto okamihu 180 stupňov.



Monitor sériového portu zobrazí v reálnom čase:

Aktuálnu hodnotu odporu (0 – 1023)

Mapovaný uhol serva (0° – 180°)

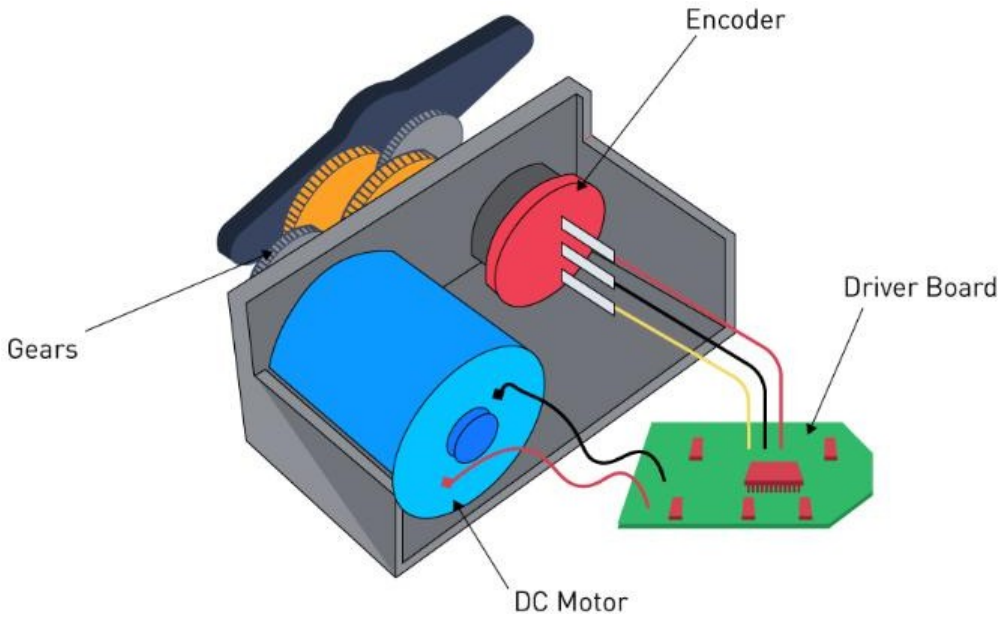


1. Vysvetlenie princípu

Funkcia servomotora je presný proces riadenia v uzavretej slučke: Najskôr externý ovládač pošle inštrukciu o cieľovom uhle na riadiacu dosku servomotora. Riadiaca doska potom aktivuje vstavaný jednosmerný motor a vysokorýchlostný výkon s nízkym krútiacim momentom, ktorý motor produkuje, sa prenáša na sústavu ozubených kolies. Zubovým prevodom viacerých ozubených kolies na spomalenie sa výkon premieňa na formu s nízkou rýchlosťou a vysokým krútiacim momentom, čím sa poháňa výstupný hriadeľ servomotora, aby sa začal otáčať.

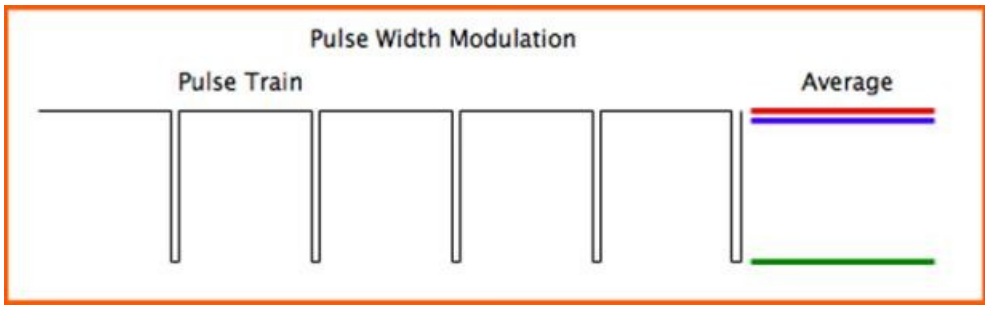
Súčasne snímač pripojený k výstupnému hriadeľu (alebo sústave ozubených kolies) nepretržite a v

zberá informácie o aktuálnom uhle výstupného hriadeľa a tieto údaje o polohe spätne odosiela na riadiacu dosku vo forme elektrického signálu. Riadiaca doska v reálnom čase porovná prijatý skutočný uhol s vopred nastaveným cieľovým uhlom a vypočíta rozdiel. Ak skutočný uhol nedosiahne cieľový uhol, riadiaca doska na základe smeru (dopredu
/ dozadu) a veľkosti (úprava rýchlosti) tohto rozdielu nepretržite vysiela riadiace signály do motora, čím umožňuje motoru poháňať výstupný hriadeľ, aby sa naďalej otáčal. Keď je skutočný uhol spätne prenesený enkodérom presne rovnaký ako cieľový uhol, riadiaca doska okamžite preruší napájanie motora, čím sa motor zastaví, a výstupný hriadeľ sa presne zastaví v polohe cieľového uhla, čím sa dokončí polohovacia akcia.



Tento obrázok znázorňuje signál s moduláciou šírky impulzu (PWM), ktorý predstavuje základný spôsob komunikácie, prostredníctvom ktorého servomotor prijíma príkaz na nastavenie cieľového uhla: Ovládacie doska servomotora rozpoznáva šírku každého jednotlivého impulzu v sekvencii PWM impulzov zasielanej zvonku (napríklad v bežnom rozsahu 1 ms – 2 ms) a

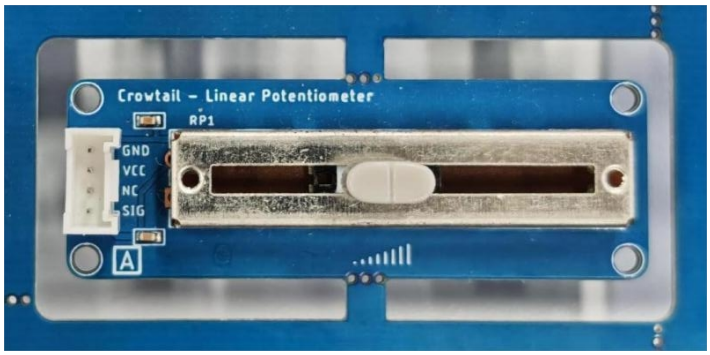
podľa toho interpretuje rôzne cieľové uhly (napríklad 1 ms zodpovedá 0° , 2 ms
odpovedá 180°); „Pulse Train“ na obrázku predstavuje nepretržitú sekvenciu PWM impulzov, zatiaľ čo „Average“ odráža skutočnosť, že zmena šírky impulzu zmení ekvivalentnú priemernú úroveň
signálu. Doska servopohonu určuje konkrétny cieľový uhol detekciou rozdielu v priemernej úrovni zodpovedajúcej tejto šírke impulzu, čím iniciuje následný
proces riadenia v uzavretej slučke (pohon motora, spätná väzba polohy a úprava polohovania), takže tento PWM signál je kľúčovým nosičom vstupu pre servomotor na prijímanie riadiacich inštrukcií a dosiahnutie riadenia uhla.



2. Požadované moduly

Trojvodičový mikroservo 9g servomodul × 1

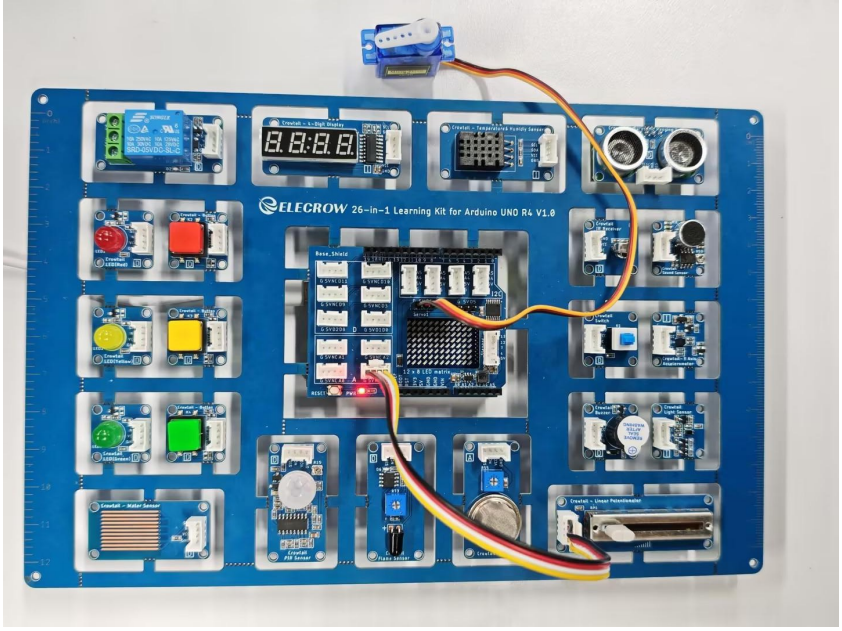
Lineárny potenciometerový modul Crowtail × 1



3. Spôsob zapojenia

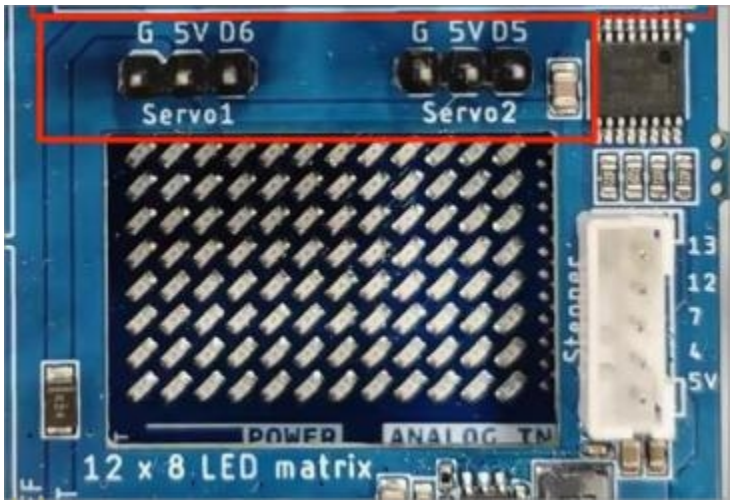
Trojvodičový mikroservo 9g modul riadiaceho mechanizmu → Servo1

Lineárny potenciometer Crowtail → Analógový port A3



Popis trojvodičového servo rozhrania:

- GND (čierny) → GND
- VCC (červená) → 5V
- SIG (žltá) → D6



4. Vysvetlenie príkladu

Kliknite na odkaz a stiahnite si oficiálny príklad kódu:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-20_Servo/20_Servo

Otvorte program kurzu v priečinku **„20_Servo“** pomocou prostredia Arduino IDE:

20_Servo.ino

```
#include <Servo.h>

// ====== Pin definition ======
#define Servo1_PIN 6
#define LinearPotentiometer_Pin A3

// ====== Create a servo object ======
Servo myservo1;

void setup() {
  Serial.begin(115200);

  // Bind the servo pin (with adjustable pulse width range)
  myservo1.attach(Servo1_PIN, 600, 2520);

  Serial.println("Potentiometer -> Servo Control Ready");
}

void loop() {
  // Read the sliding rheostat
  int potValue = analogRead(LinearPotentiometer_Pin);  // 0 ~ 1023

  // Value mapping: Potentiometer -> Rudder angle
  int angle = map(potValue, 0, 1023, 0, 180);

  // Control the rotation of the servo motor
  myservo1.write(angle);

  // Serial port debugging output
  Serial.print("Pot: ");
  Serial.print(potValue);
  Serial.print("  -> Angle: ");
  Serial.println(angle);

  delay(15);  // Give the steering gear some reaction time
}

```

Vysvetlenie kódov klávesov

(1) Použité súbory knižnice

```
#include <Servo.h>
```

Účelom tohto riadku **„#include <Servo.h>“** je zaviesť do aktuálneho programu knižnicu na ovládanie serva, ktorú poskytuje oficiálna webová stránka Arduina, čím vám umožní priamo používať triedu **„Servo“** a rôzne ovládacie funkcie, ktoré poskytuje na riadenie serva; keďže servo vyžaduje presné **PWM impulzné** signály (s periódou približne 20 ms a šírkou impulzu určujúcou uhol), spoliehanie sa výlučne na **„digitalWrite()“** to nedokáže dosiahnuť a knižnica **Servo** už pre vás zapracovala základné detaily, ako je konfigurácia časovača,

ovládanie impulzov **na úrovni mik**ro**sekúnd** a nepretržitý stabilný výstup. Preto potrebujete len

napísať **„attach**()“ a **„write**()“ ako **„vysokoúrovňové** príkazy“, aby sa servo otočilo do požadovaného uhla. Tento riadok je ekvivalentný tomu, ako keby ste kompilátoru povedali: Nasledujúci program bude používať „hotové a spoľahlivé schéma riadenia serva“, namiesto toho, aby implementoval zložité hardvérové časovanie od začiatku.

Na použitie tejto knižnice je možné zvoliť nasledujúce metódy:

➢ Stiahnutie v prostredí Arduino IDE

Verzia knižnice Servo, ktorú používame, je 1.2.2.

(2) define

```
#define Servo1_PIN 6

#define LinearPotentiometer_Pin A3
```

Tieto dva riadky „**#define Servo1_PIN** 6“ a „**#define LinearPotentiometer_Pin** A3“ využívajú koncept makier. Ich účelom je dať jasný a výstižný názov „číslam hardvérových pinov“: „**Servo1_PIN**“ označuje, že signálna linka serva je pripojená k digitálnemu **pinu 6**, a **„LinearPotentiometer_Pin“** označuje, že posuvný potenciometer je pripojený k analógovému **pinu A3;** takýto spôsob zápisu nie je povinnou požiadavkou Arduina, ale skôr veľmi dôležitým programátorským zvykom – na jednej strane uľahčuje sémantické pochopenie kódu (podľa názvu viete, k čomu je tento pin pripojený), a na druhej strane,

keď neskôr zmeníte zapojenie (napríklad nahradením serva na **pin 9**), stačí zmeniť len toto číslo a všetky nasledujúce časti kódu používajúce tento pin sa automaticky prispôsobia.

(3) Vytváranie objektov

```
Servo myservo1;
```

Tento riadok **„Servo myservo1;“** využíva znalosti o **„triedach a objektoch**“. Jeho význam je: Na základe triedy „**Servo**“, ktorá je už napísaná v knižnici Servo, vytvorte objekt s názvom „**myservo1**“ pre servomotor.

Triedu „Servo“ si môžete predstaviť ako „všeobecný návod na obsluhu servomotorov“, ktorý už obsahuje základné detaily, ako napríklad „ako generovať riadiace signály PWM a ako nastaviť servomotor do určitého uhla“, zatiaľ čo „myservo1“ je konkrétny servomotor, ktorý budete používať vo svojom kóde. Prostredníctvom neho môžete volať členské funkcie, ako napríklad **„attach**()“ a **„write**()“, na ovládanie hardvéru.

Dôvodom je to, že objektovo orientované myslenie Arduina kladie dôraz na princíp „najprv vytvor objekt, potom ovládať zariadenie“. Ak v budúcnosti pripojíte druhý servomotor, stačí len deklarovať ďalší **„Servo myservo2;**“ a oba servomotory bude možné ovládať nezávisle, čo je základný spôsob použitia, ktorý by ste si mali osvojiť.

(4) Časť „set up“

```
void setup() {
  Serial.begin(115200);


  // Pripojte servo pin (s nastaviteľným rozsahom šírky impulzu)
  myservo1.attach(Servo1_PIN, 600, 2520);


  Serial.println("Potenciometer -> Ovládanie serva pripravené");
}
```

Serial.begin(115200); Tento krok slúži na inicializáciu sériovej komunikácie a nastavenie prenosovej rýchlosti na 115 200. Účelom tohto kroku nie je ovládať servo, ale umožniť programu odosielať počas prevádzky ladiace informácie do počítača, aby sme mohli „vidieť“, čo program robí.

Ďalšia veta, **„myservo1.attach(Servo1_PIN, 600, 2520);“**, je najdôležitejším krokom pri inicializácii ovládania serva. Priraďuje predtým vytvorený objekt serva myservo1 k skutočnému hardvérovému pinu **Servo1_PIN** a špecifikuje minimálnu a maximálnu šírku **PWM** impulzov (v

mikrosekundách), ktoré môže servo mať (na základe princípu, že servo je riadené šírkou impulzu). Myšlienka za napísaním tohto riadku kódu je: najprv povedať programu, „na ktorý pin je toto servo pripojené“, a potom jasne „bezpečný rozsah otáčania tohto serva“.

Posledný riadok **„Serial.println(„Potentiometer -> Servo Control Ready“);“** slúži len na zobrazenie stavu a informuje používateľa, že inicializácia je dokončená a logika riadenia z potenciometra na servo v nasledujúcej **slučke** () môže fungovať normálne.

(5) loop

```
void loop() {
    // Načítanie posuvného reostatu
    int potValue = analogRead(LinearPotentiometer_Pin);              // 0 ~ 1023

    // Mapovanie hodnôt: Potenciometer -> Uhol
    smerovky int angle = map(potValue, 0, 1023, 0, 180);

    // Ovládanie otáčania servomotora
    myservo1.write(angle);

    // Výstup ladenia sériového portu
    Serial.print("Pot: ");
    Serial.print(potValue); Serial.print("
                        -> Uhol: ");
    Serial.println(angle);

    delay(15);      // Poskytnutie reakčného času riadiacej prevodovke
}
```

Najskôr sa pomocou **funkcie analogRead(LinearPotentiometer_Pin)** prečíta hodnota napätia posuvného potenciometra. Pomocou ADC v Arduine sa analógové napätie **0–5 V** prevádza na celé číslo v rozsahu **od 0 do 1023**. Tento krok využíva základné znalosti o vzorkovaní analógových veličín a analógovo-digitálnom prevode;

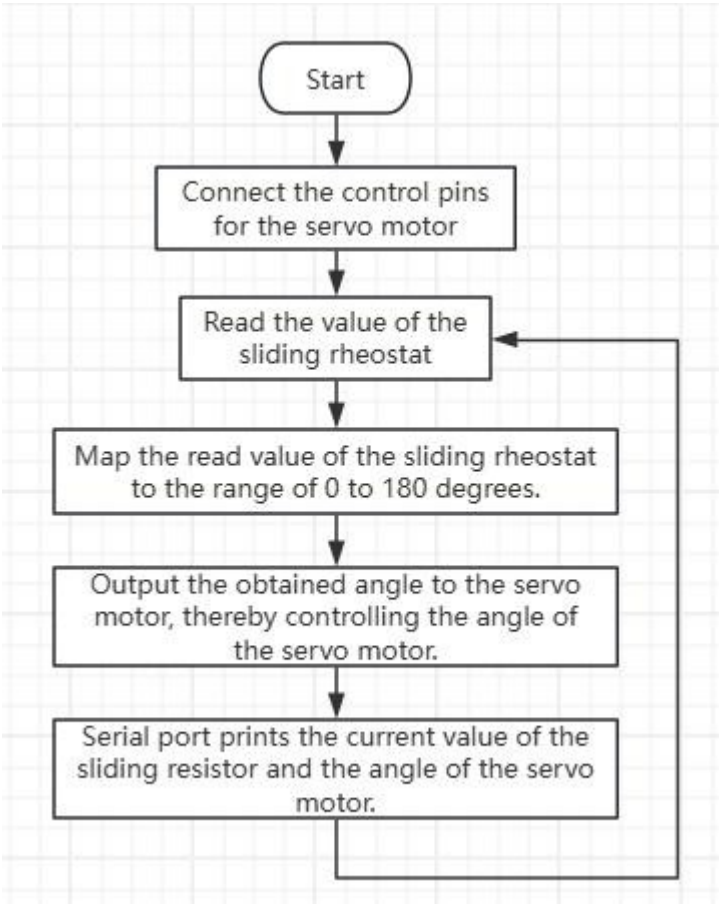
Potom sa pomocou **map(potValue, 0, 1023, 0, 180)** rozsah hodnoty potenciometra lineárne premietne do rozsahu uhlov, ktorý servo dokáže spracovať. To odráža princíp proporcionálneho premietania, ktorého cieľom je dosiahnuť, aby „o koľko sa posunie potenciometer, o toľko sa otočí servo“;

Následne **myservo1.write(angle)** prenáša vypočítaný uhol do knižnice **serva**. Knižnica interne generuje zodpovedajúci **PWM** impulz, ktorý poháňa servo k otáčaniu. Tento krok zakrýva základné časové detaily, čím sa ovládanie stáva intuitívnym;

Nasledujúce príkazy **Serial.print** sa na ovládaní nepodieľajú. Namiesto toho v reálnom čase vypíšu pôvodnú hodnotu potenciometra a konečný uhol, čo nám pomáha pochopiť a overiť, či program funguje podľa očakávaní;

Nakoniec, **delay(15)** poskytuje servu mechanickú dobu odozvy, čím sa zabráni chveniu serva alebo nestabilným reakciám v dôsledku rýchlych aktualizácií inštrukcií.

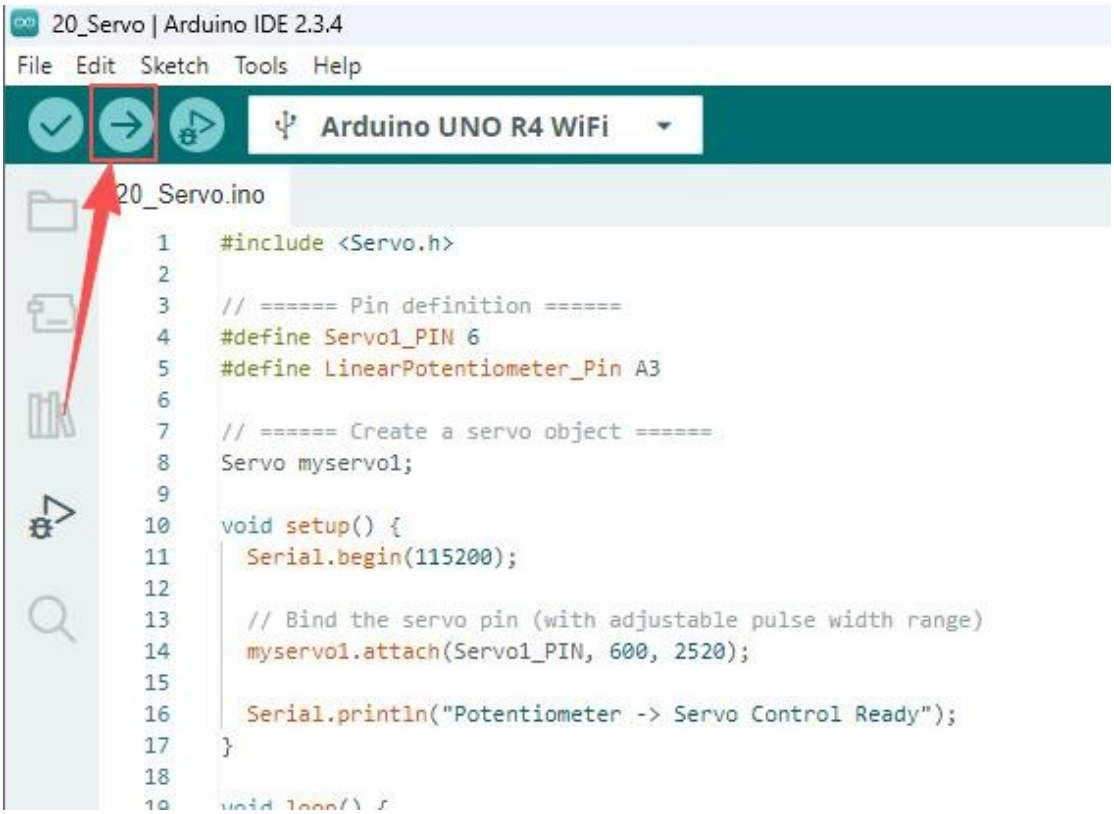
(6) Celkový diagram logického toku kódu



5. Spustite program a sledujte výsledky

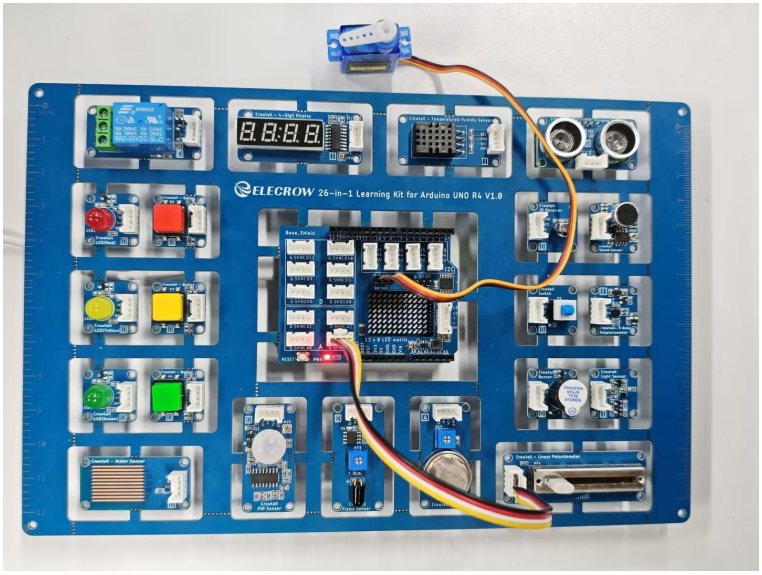
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“:**

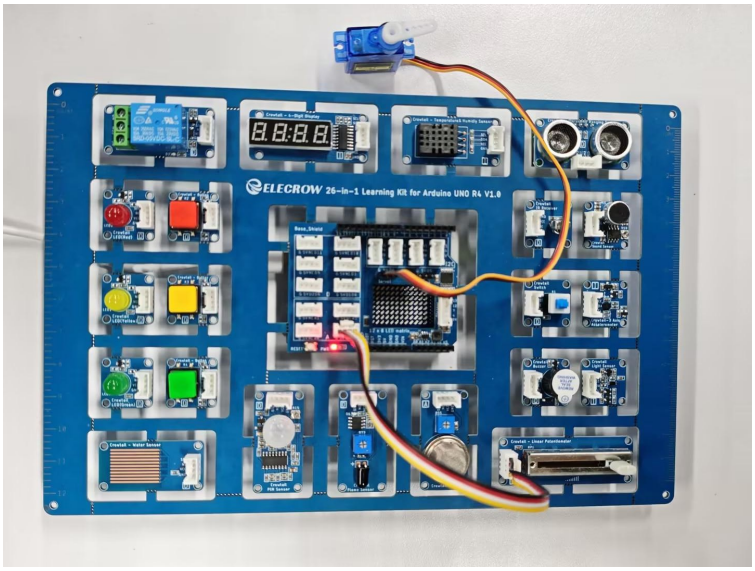


(2) Uvidíte:

Systém bude nepretržite čítať zmeny polohy posuvného reostatu: Keď je posuvný reostat vľavo, uhol servomotora je v tomto momente 0



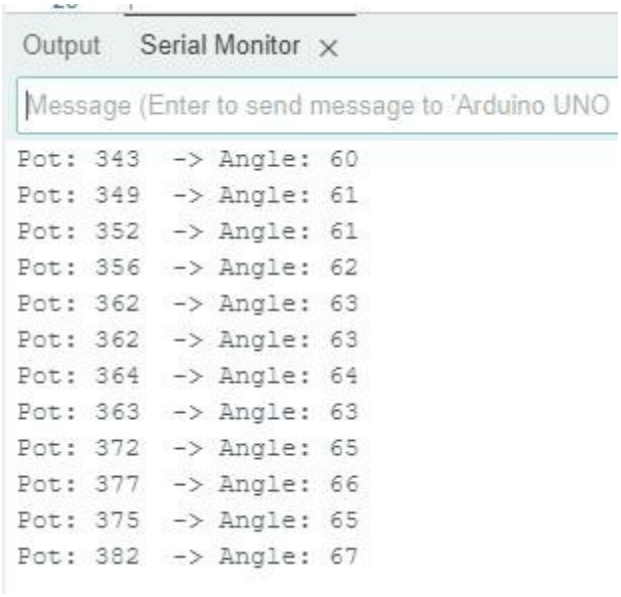
Keď je posuvný reostat vpravo, uhol servomotora je v tomto momente 180 stupňov.



Monitor sériového portu zobrazuje v reálnom čase:

aktuálnu hodnotu odporu (0 – 1023)

Mapovaný uhol serva (0° – 180°)



Lekcia 21 – Inteligentný systém automatických dverí

Úvod

V tejto lekcii sa naučíme, ako pomocou ultrazvukového snímača vzdialenosti zistiť, či sa v blízkosti nachádzajú ľudia, a ako skombinovať servomotor a bzučiak, aby sme vytvorili kompletný inteligentný systém automatických dverí. V tomto projekte je ultrazvukový snímač zodpovedný za „videnie“ zmien vo vzdialenosti, servomotor je zodpovedný za vykonávanie akcií otvárania a zatvárania a bzučiak sa používa na bezpečnostné upozornenia.

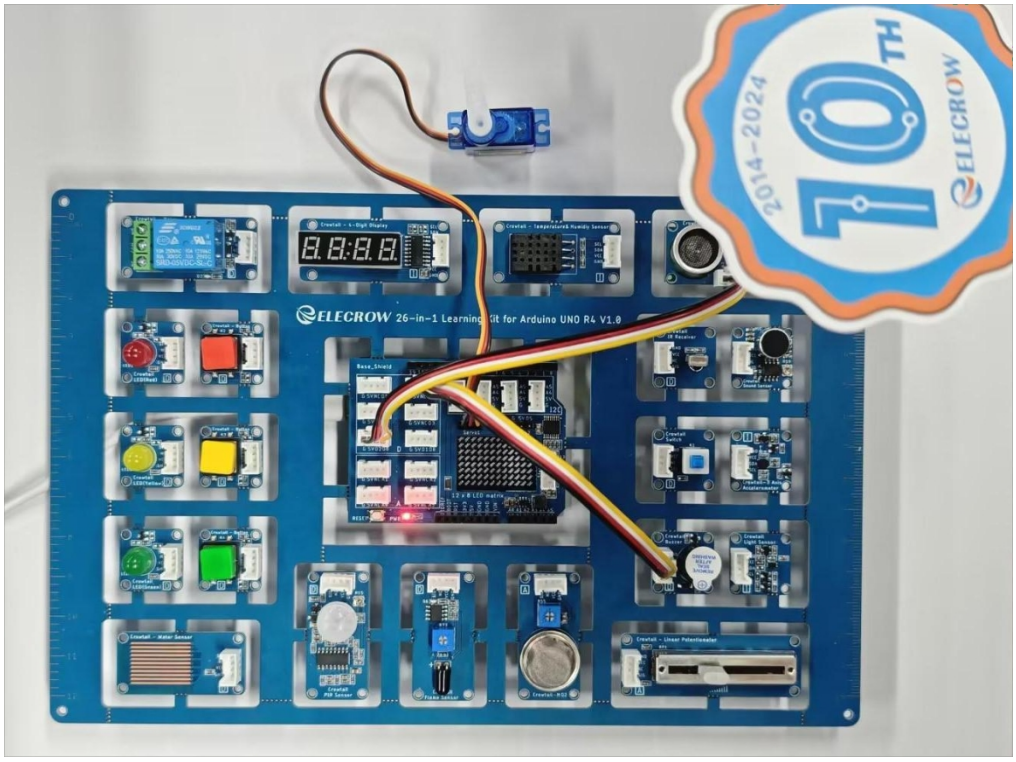
V rámci tohto kurzu sa už nebudete venovať len ovládaniu servomotora alebo čítaniu údajov zo senzorov, ale naučíte sa kombinovať viaceré moduly, robiť rozhodnutia na základe skutočnej vzdialenosti a nechať systém „myslieť ako človek“: kedy otvoriť dvere, kedy ich zavrieť a kedy vydávať varovanie.

Ciele výučby

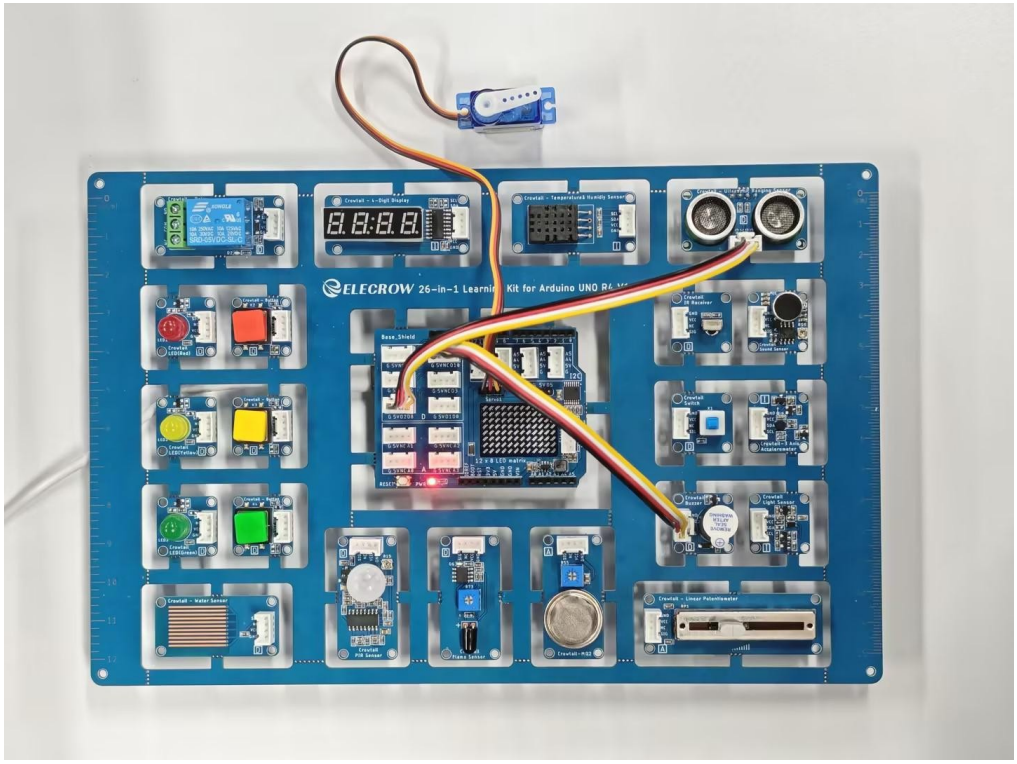
1. Zopakovať a upevniť si základné ovládanie uhla servomotora pomocou knižnice Servo.
2. Porozumieť úlohe podmieneného rozhodovania (if) v logike „otvorenia/zatvorenia“ automatických dverí.
3. Naučte sa používať funkciu millis(), aby ste dosiahli „odložené zatvorenie“ bez blokovania behu programu.
4. Naučte sa zapuzdriť funkcie, aby bola logika kódu jasná a zrozumiteľná.
5. Dokončite prípad z tejto lekcie: Inteligentný systém automatických dverí založený na ultrazvukovej detekcii.

Náhľad výsledku

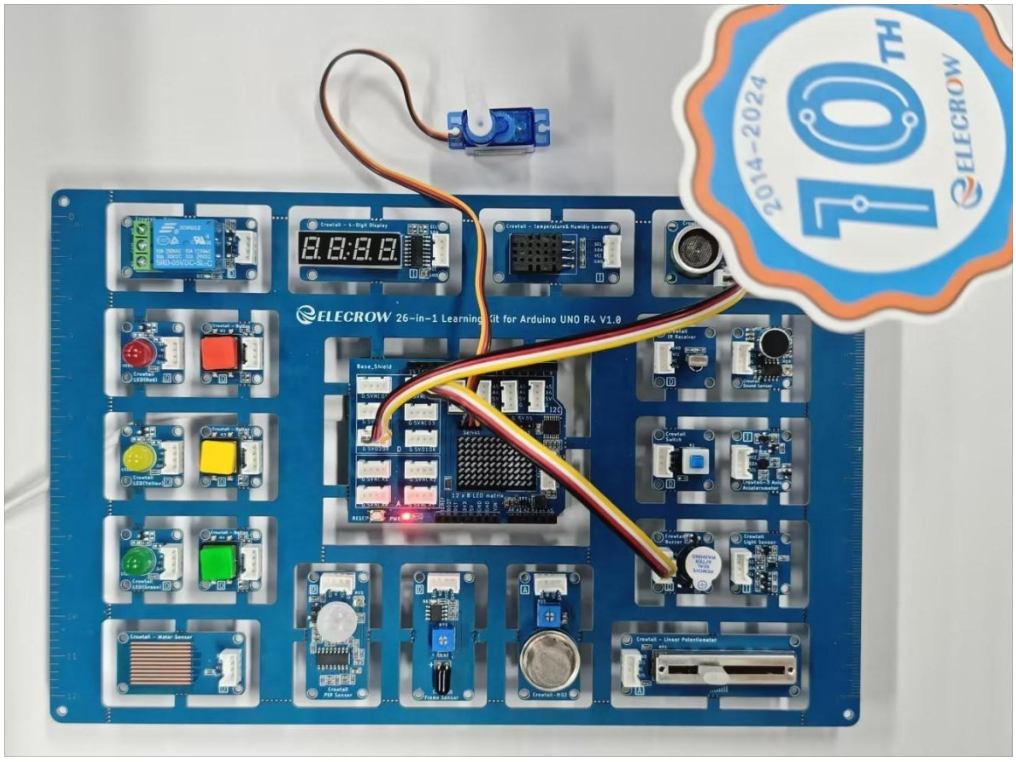
Keď ultrazvukový senzor zistí, že vzdialenosť od osoby je menšia ako 50 cm, servomotor simuluje otvorenie dverí. V tomto momente je servomotor v uhle 90 stupňov.



Keď ultrazvukový senzor zistí, že v danom okamihu nikto nie je prítomný a do 5 sekúnd sa nikto nepriblíži, servomotor simuluje zatváranie dverí. V tomto momente je servomotor v polohe 0°.



Ak je vzdialenosť príliš malá, menej ako 10 cm, bzučiak spustí alarm, aby upozornil na bezpečnostné opatrenie proti zachyteniu.



1. Vysvetlenie princípu

Princípy fungovania jednotlivých modulov boli podrobne vysvetlené v predchádzajúcich kurzoch. Nasledujúce kurzy budú zahŕňať komplexné praktické cvičenia založené na predchádzajúcich moduloch a vykonanie niektorých zaujímavých experimentov. Samozrejme, po dokončení štúdia budete schopní aplikovať viac modulov aj na praktické účely.

2. Požadované moduly

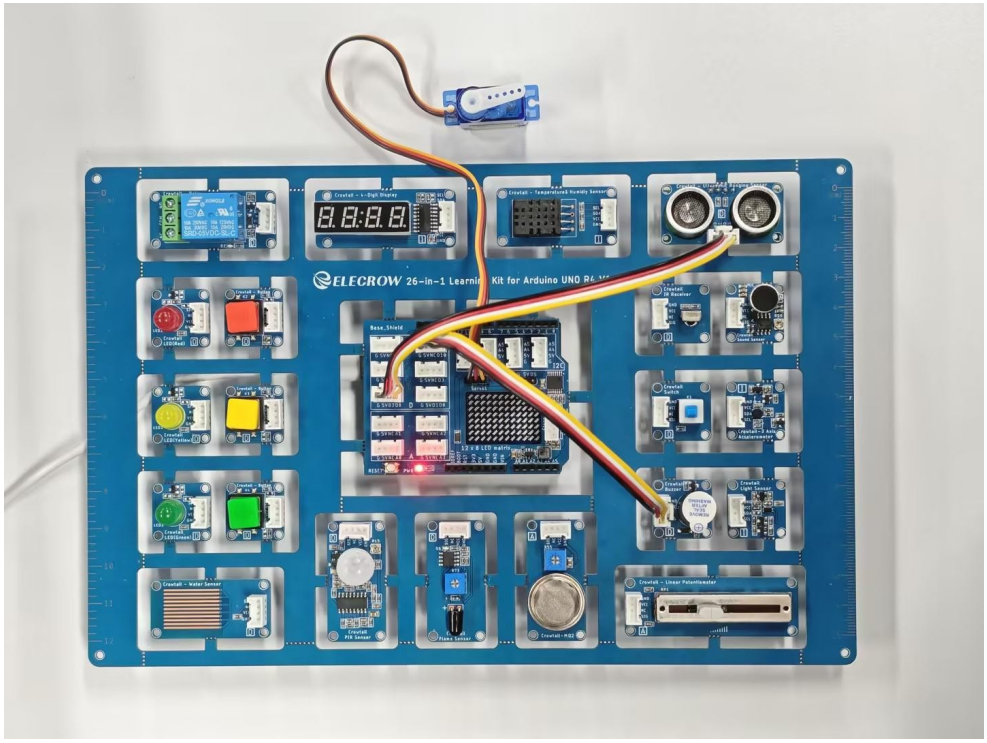
Trojvodičový mikroservo 9g servomodul × 1

Crowtail HC-SR04 (ultrazvukový senzor na meranie vzdialenosti ×1)

Crowtail bzučiak (1 ks)

3. Spôsob zapojenia

Trojvodičový mikroservo 9g Servo modul → Servo1 Ultrazvukový senzor vzdialenosti → Digitálne porty D2 a D8 Crowtail bzučiak → Digitálny port D10



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite vzorový kód poskytnutý oficiálnym zdrojom:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-21_Intelligent_Automatic_Door/21_Intelligent_Automatic_Door

Otvorte program kurzu v priečinku **„21_Intelligent_Automatic_Door“** pomocou Arduino IDE:

21_Intelligent_Automatic_Door.ino

```
#include <HCSR04.h>
#include <Servo.h>

/* ---------- Ultrasonic ---------- */
const byte triggerPin = 8;
const byte echoPin = 2;
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);

/* ---------- Servo ---------- */
#define SERVO1_PIN 6

Servo myservo1;

int doorOpenAngle = 90;    // Door open angle (adjustable)
int doorCloseAngle = 0;    // Door close angle

/* ---------- Buzzer ---------- */
#define BUZ 10

/* ---------- State variables ---------- */
bool doorIsOpen = false;          // Current door state
unsigned long lastDetectTime = 0; // Last time a person was detected
const unsigned long closeDelay = 5000; // 5 seconds

void setup() {
  Serial.begin(115200);

  myservo1.attach(SERVO1_PIN, 600, 2520);

  pinMode(BUZ, OUTPUT);

  // Initial state: door closed
  myservo1.write(doorCloseAngle);
}

void loop() {

  /* ---------- 1. Read distance ---------- */
  float distance = distanceSensor.measureDistanceCm();
  if (distance < 0)
    return;

  Serial.print("Distance: ");
```

Vysvetlenie kódov

(1) Použité súbory knižnice

```
#include <HCSR04.h>

#include <Servo.h>
```

HCSR04.h poskytuje kompletnú podporu pre modul na meranie vzdialenosti pomocou ultrazvuku HC-SR04. Interné riešenie sa už postaralo o základné časovacie a výpočtové úlohy, ako je odosielanie impulzov na **spúšťací** pin, časovanie na pinu **pre odozvu** a prevod rýchlosti zvuku na vzdialenosť. Preto

v nasledujúcom kóde stačí len volať rozhrania vyššej úrovne, ako je **measureDistanceCm**(), aby sme priamo získali „výsledky vzdialenosti na úrovni centimetrov“, bez toho, aby sme museli sami písať kód na meranie šírky impulzu.

Servo.h je oficiálna knižnica na ovládanie servomotorov pre Arduino. Skryje celý proces generovania **PWM** impulzy, ktoré musia byť presné na mikrosekundy. Na riadenie otáčania serva stačí použiť pojem „uhol“, ktorý je pre človeka ľahko zrozumiteľný (napríklad 0° , 90°). Práve to je základom pre napísanie logiky automatického „otvárania/zatvárania“ dverí

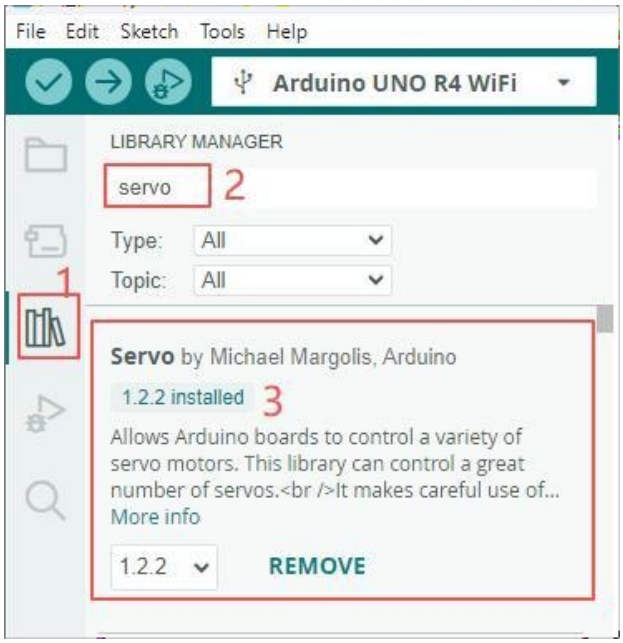
veľmi intuitívne.

Tieto dve knižnice možno použiť nasledujúcimi spôsobmi:

➢ Stiahnutie v prostredí Arduino IDE

Verzia HCSR04, ktorú tu používame, je 2.0.0

Verzia knižnice Servo, ktorú používame, je 1.2.2.



(2) Definície parametrov pre hardvér na meranie vzdialenosti pomocou ultrazvuku + Objektovo orientovaná zapuzdrenie

```
const byte triggerPin = 8; const
byte echoPin = 2;
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);
```

Najprv sa riadky **„const byte triggerPin = 8;“** a **„const byte echoPin = 2;**“ používajú na jasné informovanie programu: ktorý pin Arduina je zodpovedný za odosielanie ultrazvukového spúšťacieho signálu (**Trig**) a ktorý pin je zodpovedný za prijímanie odrazového signálu (**Echo**). Použitie „const“ tu znamená, že tieto piny zostávajú počas vykonávania programu nemenné, čo je bezpečné a zároveň v súlade s konvenciami hardvérového dizajnu.

Potom tento riadok **„UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);**“ nie je bežná premenná. Namiesto toho vytvára „objekt ultrazvukového senzora vzdialenosti“. Je to ekvivalentné zabaleniu „pin Trig, pin Echo, algoritmu merania vzdialenosti a riadenia časovania“ do inštancie s názvom **„distanceSensor**“. Od tohto momentu už nemusíte ručne ovládať úrovne pinov ani počítať časový rozdiel. Stačí zavolať funkciu merania vzdialenosti prostredníctvom tohto objektu a získate hodnotu vzdialenosti.

(3) Hardvérové mapovanie riadiaceho mechanizmu + riadiaci objekt riadiaceho mechanizmu + parametre mechanického pôsobenia dverí

```
#define SERVO1_PIN 6
Servo myservo1;
```

```
int uholOtvoreniaDverí =          // Uhol otvorenia dverí
90;                               (nastaviteľný)
int uholZatvoreniaDverí =        // Uhol zatvorenia
0;                                dverí
```

Najprv **#define SERVO1_PIN 6** používa definíciu makra na fixovanie pripojenia signálnej linky serva k 6. pinu Arduina. Výhodou tohto riešenia je, že kód je veľmi čitateľný a v prípade, že bude v budúcnosti potrebné zmeniť zapojenie, stačí upraviť len jeden riadok.

Potom **Servo myservo1;** neslúži na ovládanie otáčania serva, ale na vytvorenie objektu na ovládanie serva. Je to ekvivalent „diaľkového ovládača serva“. Všetky akcie otvárania a zatvárania dverí v nasledujúcej časti sú vydávané príkazmi prostredníctvom tohto objektu.

Nakoniec, **int doorOpenAngle = 90;** a **int doorCloseAngle = 0;** abstrahujú dva mechanické stavy „**otvorenie** dverí“ a **„zatvorenie** dverí“ do uhlových parametrov. To umožňuje programu opísať správanie dverí pomocou „uhla“ namiesto „činnosti motora“ a vy môžete navrhnúť uhly podľa vlastných predstáv.

(4) Hardvérové rozhranie alarmu + Pamäť stavu prevádzky dverí + Referencia času oneskoreného zatvárania

```
#define BUZ 10


/* ---------- Premenné stavu----------*/
bool dvereSúOtvorené = false;              // Aktuálny stav dverí
unsigned long lastDetectTime = 0; // Čas poslednej detekcie osoby const
unsigned long closeDelay = 5000; // 5 sekúnd
```

Najprv **#define BUZ 10** používa definíciu makra na špecifikovanie, že bzučiak je pripojený k pinu 10, čo umožňuje programu ovládať zvuk bzučiaka jednoducho použitím mena „BUZ“ neskôr, čím sa zabráni zmätku spôsobenému priamym zápisom čísla digitálneho pinu;

Potom je **bool doorIsOpen = false;** stavová premenná používaná na „zapamätanie“ si, či sú dvere otvorené alebo zatvorené. Keďže Arduino vykonáva funkciu loop() zakaždým od začiatku, bez takejto premennej by program nebol schopný určiť, či boli dvere otvorené;

Ďalej sa pomocou **unsigned long lastDetectTime = 0;** zaznamenáva časový okamih, kedy bol zaznamenaný príchod osoby. Používa sa v spojení s funkciou millis() a je kľúčom k implementácii „oneskoreného zatvárania“ – neblokujúcej logiky;

Nakoniec, **const unsigned long closeDelay = 5000;** definuje pevnú časovú hranicu (5 sekúnd), ktorá udáva najdlhší čas, počas ktorého zostanú dvere otvorené po tom, čo niekto odíde. Použitie „const“ tu zdôrazňuje, že ide o systémový parameter, ktorý sa v programe nebude meniť.

(5) Sekcia setup

```
void setup() {
    Serial.begin(115200);


    myservo1.attach(SERVO1_PIN, 600, 2520);


    pinMode(BUZ, OUTPUT);


    // Počiatočný stav: dvere zatvorené
    myservo1.write(doorCloseAngle);
}
```

Najprv sa pomocou **Serial.begin(115200);** otvorí sériová komunikácia, čo je pre nás výhodné na sledovanie stavu dverí, údajov o vzdialenosti a informácií o poplachu vo fáze ladenia;

Potom **príkaz myservo1.attach(SERVO1_PIN, 600, 2520);** priradí objekt serva k konkrétnemu PWM pinu a jasne určí knižnici na ovládanie serva „rozsah minimálnej a maximálnej šírky impulzu“. Cieľom je zabezpečiť, aby sa servo otáčalo stabilne bez chvenia, a zároveň sa tak predíde problému s nekonzistentnými zdvihmi u rôznych typov serv;

Potom **pinMode(BUZ, OUTPUT);** nakonfiguruje pin bzučiaka do výstupného režimu, čo znamená, že program ho bude aktívne ovládať, aby následne vydal zvuk;

Nakoniec **príkaz myservo1.write(doorCloseAngle);** prinúti dvierka, aby sa na začiatku programu otočili do „uzavretej polohy“. Tento krok je veľmi dôležitý, pretože ak sa nevykoná hneď na začiatku, servo sa môže zastaviť v náhodnej polohe, v akej bolo pri poslednom vypnutí, čo spôsobí, že dvierka zostanú otvorené hneď po obnovení napájania.

(6) Slučka

```
void loop() {


    /* ---------- 1. Prečítať vzdialenosť --------*/
    float distance = distanceSensor.measureDistanceCm(); if
    (distance < 0)
        vráť;


    Serial.print("Vzdialenosť: ");
```

```
    Serial.print((int)distance);
    Serial.println(" cm");


    /* ---------- 2. Alarm proti zachyteniu --------*/
    if (vzdialenosť > 0 && vzdialenosť <= 10) {
        buzzerWarning();
    }


    /* ---------- 3. Logika otvárania dverí ------------*/
    if (vzdialenosť > 0 && vzdialenosť <= 50) {
        otvorDvere();
        lastDetectTime = millis(); // Zaznamenaj čas, kedy bola zistená prítomnosť osoby
    }


    /* ---------- 4. Logika zatvárania dverí ---------*/
    if (dvereSúOtvorené) {
        if (millis() - lastDetectTime >= closeDelay) {
            closeDoor();
        }
    }
    delay(200);
}
```

Najskôr sa pomocou **funkcie distanceSensor.measureDistanceCm()** prečíta vzdialenosť nameraná ultrazvukovým senzorom**.** Ak je vrátená hodnota menšia ako 0, znamená to, že meranie vzdialenosti je neplatné, a program sa okamžite vráti, aby predčasne ukončil aktuálnu slučku a zabránil tak ovplyvneniu rozhodnutia systému nesprávnymi údajmi.

Následne sa aktuálna vzdialenosť vypíše cez sériový port, aby sme mohli ľahšie sledovať svet, ktorý systém „vidí“.

Následne sa spustí logika proti zablokovaniu: ak je zistená vzdialenosť cieľa v rozmedzí 10 cm, považuje sa to za prílišnú blízkosť osoby alebo prekážky a okamžite sa volá funkcia **buzzerWarning()**, aby vydala varovanie. Ide o typický návrh s prioritou bezpečnosti. (Podrobné vysvetlenie kódu je uvedené v nasledujúcom bode (9))

Potom nasleduje logika otvárania dverí: keď je vzdialenosť v rozmedzí 50 cm, znamená to, že sa niekto blíži k dverám. Systém volá **funkciu openDoor**(), aby otvoril dvere, a zároveň

Funkcia millis() slúži na zaznamenanie času, „keď bola osoba naposledy zaznamenaná“, čím sa pripraví následné automatické zatvorenie dverí. (Podrobné vysvetlenie kódu je uvedené v nasledujúcom bode **(7)**)
Ďalej nasleduje logika zatvárania dverí: len vtedy, keď sú dvere už otvorené (**doorIsOpen** == true), sa skontroluje rozdiel medzi aktuálnym časom a časom predchádzajúceho detekovania. Ak tento rozdiel prekročí nastavenú hodnotu closeDelay (5 sekúnd), vykoná sa funkcia **closeDoor**(). Tým sa zabráni situácii, keď sa dvere zatvoria skôr, ako sa osoba vzdiali. (Podrobné vysvetlenie kódu je uvedené v nasledujúcom bode **(8)**)
Nakoniec, **delay**(200) zabezpečuje, že hlavná slučka beží v rytme približne 200 ms, čo nielen zaručuje rýchlosť odozvy, ale aj zabraňuje príliš častému spúšťaniu ultrazvukového senzora a servomotora.

(7) openDoor

```
void openDoor() {
    if (!doorIsOpen) {
        myservo1.write(doorOpenAngle);
        doorIsOpen = true; Serial.println("Dvere
        otvorené");
    }
}
```

Funkcia **openDoor**() stelesňuje koncept „bezpečného otvárania dverí s posúdením stavu“: Najprv pomocou príkazu **if (!doorIsOpen)** skontroluje, či sú dvere momentálne v uzavretom stave. Až keď sú dvere skutočne „zatvorené“, bude povolené vykonanie akcie otvárania, čím sa zabráni opakovanému odosielaniu rovnakého riadiaceho príkazu na servomotor, keď sú dvere už otvorené; keď je podmienka splnená, **myservo1.write(doorOpenAngle)** odošle signál cieľového uhla na servomotor, čím sa servomotor otočí do vopred nastaveného „uhla otvorenia“, aby sa dosiahla fyzická akcia otvorenia. Potom sa premenná doorIsOpen nastaví na hodnotu true a tento stav sa jasne zaznamená v programe, aby sa označilo, že „dvere sa otvorili“. Táto stavová premenná sa použije v logike zatvárania v cykle loop(). Nakoniec sa cez sériový port odošle výstup „Door OPEN“, čo je vhodné na ladenie a zároveň umožňuje priamo overiť, či sa logika programu vykonáva podľa očakávania.

(8) closeDoor

```
void closeDoor() {
```

```
    myservo1.write(doorCloseAngle);
    doorIsOpen = false; Serial.println("Door
    CLOSE");
}
```

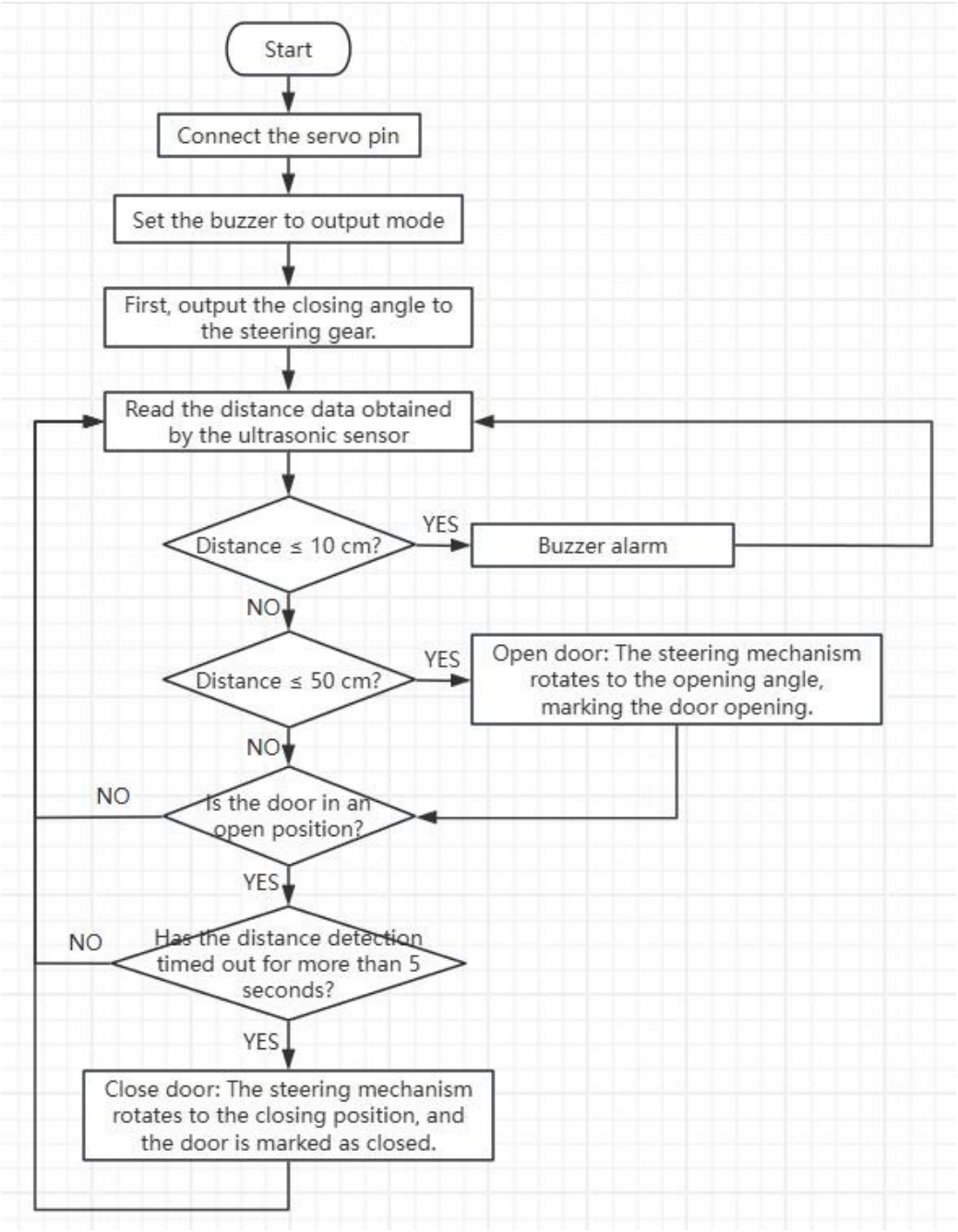
Funkcia **closeDoor**() realizuje „jednotný proces zatvárania“: Keď program zistí, že je potrebné dvere zatvoriť, jednoducho volá túto funkciu, aby dokončil celý súbor operácií. **myservo1.write(doorCloseAngle)** pošle signál cieľového uhla servu, čím sa servo otočí späť do vopred definovaného „uhla zatvárania“ a fyzicky dokončí akciu zatvárania; potom nastaví doorIsOpen na hodnotu false, čím synchronizovane aktualizuje aktuálny stav dverí na programovej úrovni a oznámi systému, že „dvere sa zatvorili“, čo je kľúčové pre následné posúdenie otvorenia a logiku načasovania; nakoniec sa prostredníctvom **Serial.println("Door CLOSE")** vypíše výstražná správa, ktorá slúži na ladenie sériového monitora a sledovanie, či sa správanie systému zhoduje s očakávaniami.

(9) buzzerWarning

```
void buzzerWarning() { for
    (int i = 0; i < 2; i++) {
        digitalWrite(BUZ, HIGH);
        delay(150);
        digitalWrite(BUZ, LOW);
        delay(150);
    }
}
```

Táto funkcia **„buzzerWarning()“** predstavuje vopred definované správanie zvukového alarmu: V rámci funkcie sa používa cyklus **„for**“, ktorý jasne vyjadruje zámer „opakovať dvakrát“; v každom cykle **„digitalWrite(BUZ, HIGH)“** aktivuje pin bzučiaka, čím sa spustí zvuk bzučiaka, a **„delay(150)“** sa používa na ovládanie trvania alarmu; potom **„digitalWrite(BUZ, LOW)“** vypne bzučiak a ďalšie „delay(150)“ sa používa na vytvorenie intervalu medzi dvoma alarmami. Celkový efekt je teda dvojtónový alarmový signál „ding – pauza – ding – pauza“. Táto metóda písania integruje „rytmus alarmu“ a „ovládanie hardvéru“ do jednej funkcie, čo je nielen ľahko zrozumiteľné, ale aj praktické pre budúce úpravy počtu alarmov alebo rytmu bez ovplyvnenia hlavného logického kódu.

(10) Celkový diagram logického toku kódu

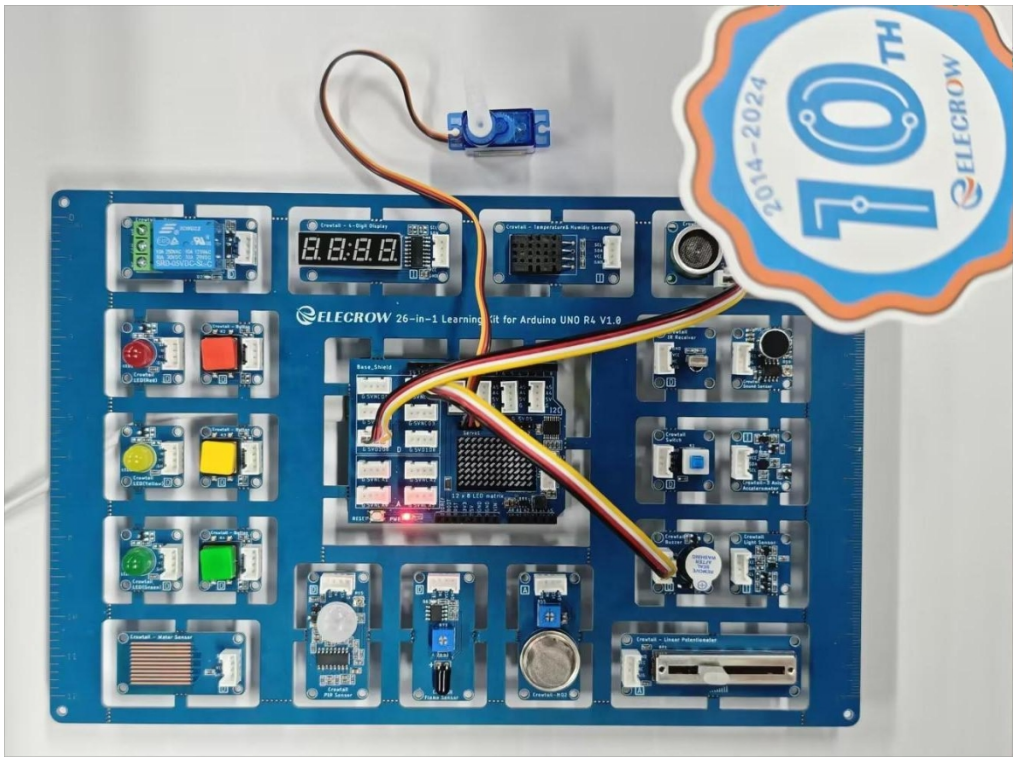


5. Spustite program a sledujte výsledky

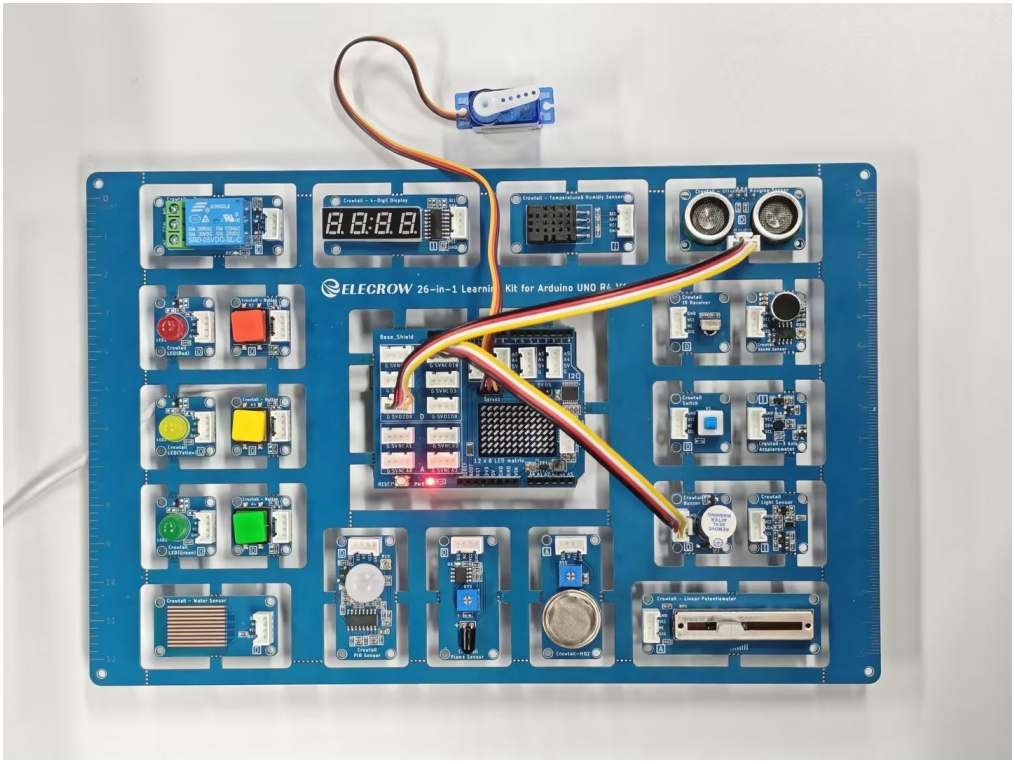
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahratia.

Kliknite na **„Stiahnuť“:**

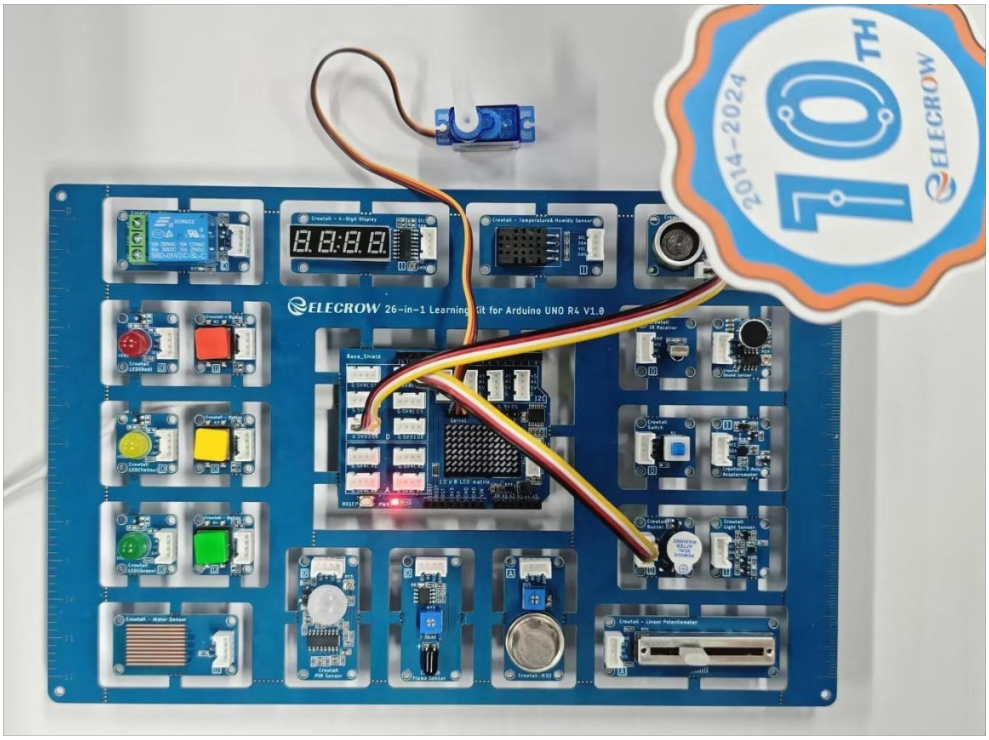
(2) Uvidíte nasledovné: Keď ultrazvukový senzor zistí, že vzdialenosť od osoby je menšia ako **50 cm**, servomotor simuluje otvorenie dverí. V tomto momente je servomotor v polohe **90** stupňov.



Keď ultrazvukový senzor zistí, že v danom momente nikto nie je prítomný a nikto sa nepriblíži do 5 sekúnd, servomotor simuluje zatvorenie dverí. V tomto momente je servomotor v polohe **0**°.



Keď je vzdialenosť príliš malá, menej ako **10 cm**, bzučiak spustí alarm, aby upozornil na bezpečnostné opatrenie proti zachyteniu.



Lekcia 22 – Inteligentný systém varovania pred teplotou

Úvod

V tejto lekcii sa naučíme, ako pomocou snímača teploty a vlhkosti DHT20 sledovať zmeny teploty okolia v reálnom čase, a ako pomocou trojfarebného LED indikátora a bzučiaka zostaviť kompletný inteligentný systém upozorňovania na teplotu.

V tomto projekte:

- Senzor DHT20 je zodpovedný za „vnímajú“ teplotu a vlhkosť okolia
- LED dióda je zodpovedná za vizuálne indikovanie aktuálneho stavu teploty pomocou farieb
- Bzučiak je zodpovedný za vydávanie zvukového upozornenia, keď je teplota abnormálna

Vďaka tomuto kurzu už nebudete len „čítať údaje zo senzora“ alebo „jednoducho rozsvietiť LED“, ale skutočne sa naučíte, ako nechať údaje zo senzora riadiť správanie systému a ako nastaviť systém tak, aby reagoval odlišne v závislosti od rôznych teplotných rozsahov, čím zariadenie bude schopné „posudzovať chladné a teplé podmienky ako človek“.

Ciele výučby

1. Preveriť a upevniť si základné metódy používania zariadení I2C (DHT20)
2. Porozumieť kľúčovej úlohe „posudzovania teplotného rozsahu“ v inteligentnom systéme podnetov
3. Naučiť sa organizovať rôzne správania hardvéru vo forme stavových funkcií
4. Pochopiť dôležitosť riadenia rytmu čítania senzora (millis + lastRead)
5. Kompletné riešenie: Inteligentný systém signalizačných svetiel založený na zmenách teploty

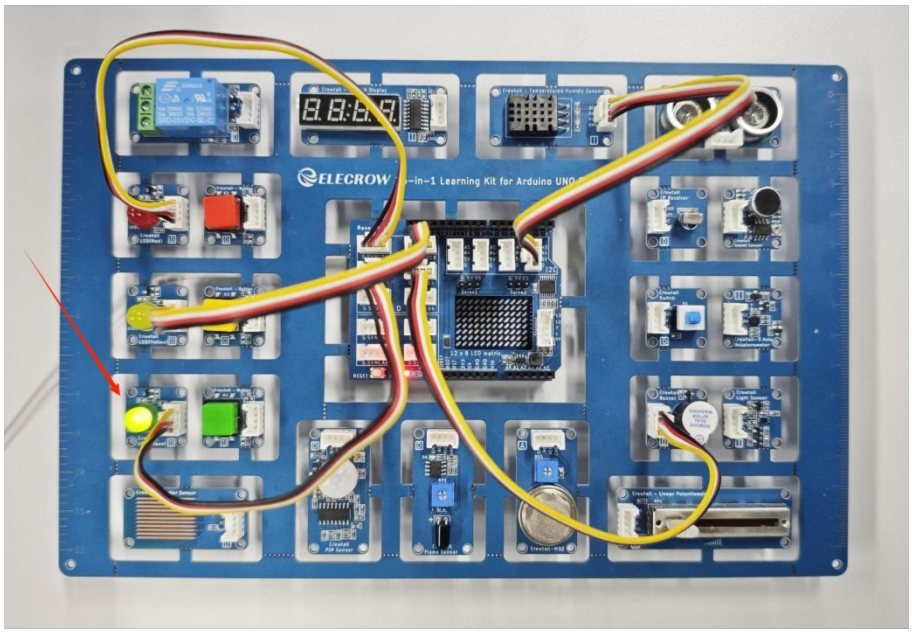
Náhľad výsledku

Keď je systém v prevádzke, budete pozorovať nasledujúce efekty:

- Normálna teplota (menej ako $28^\circ C$)

→ Zelené svetlo svieti nepretržite

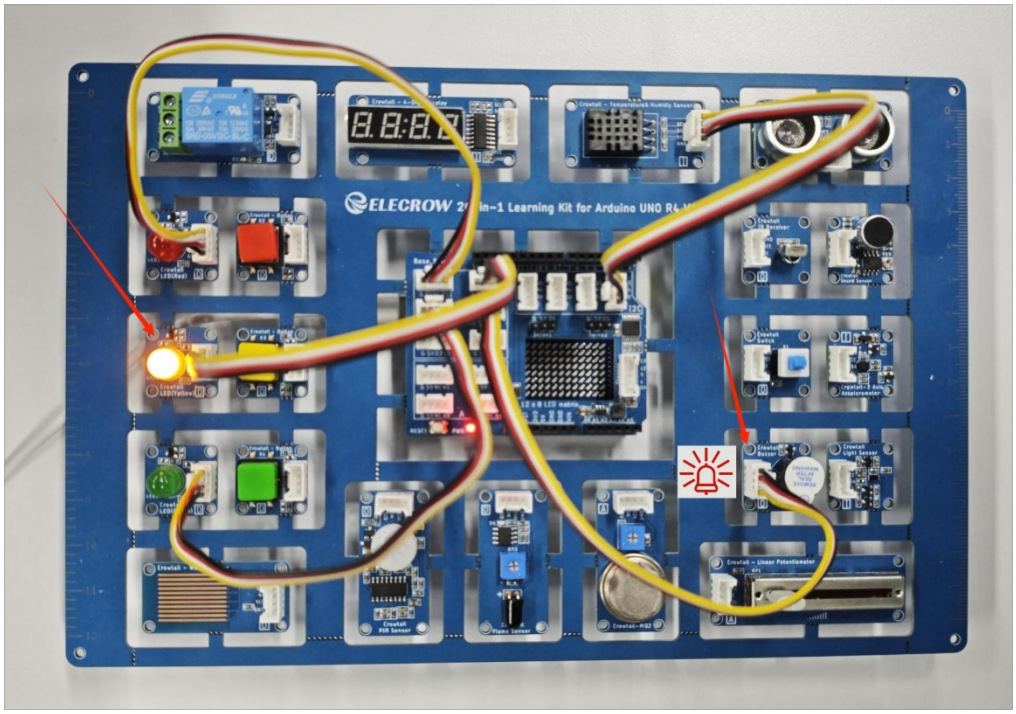
→ Zvukový signál je vypnutý



- Teplota je príliš vysoká (28 ° C ~ 32 ° C)

→ Žlté svetlo svieti nepretržite

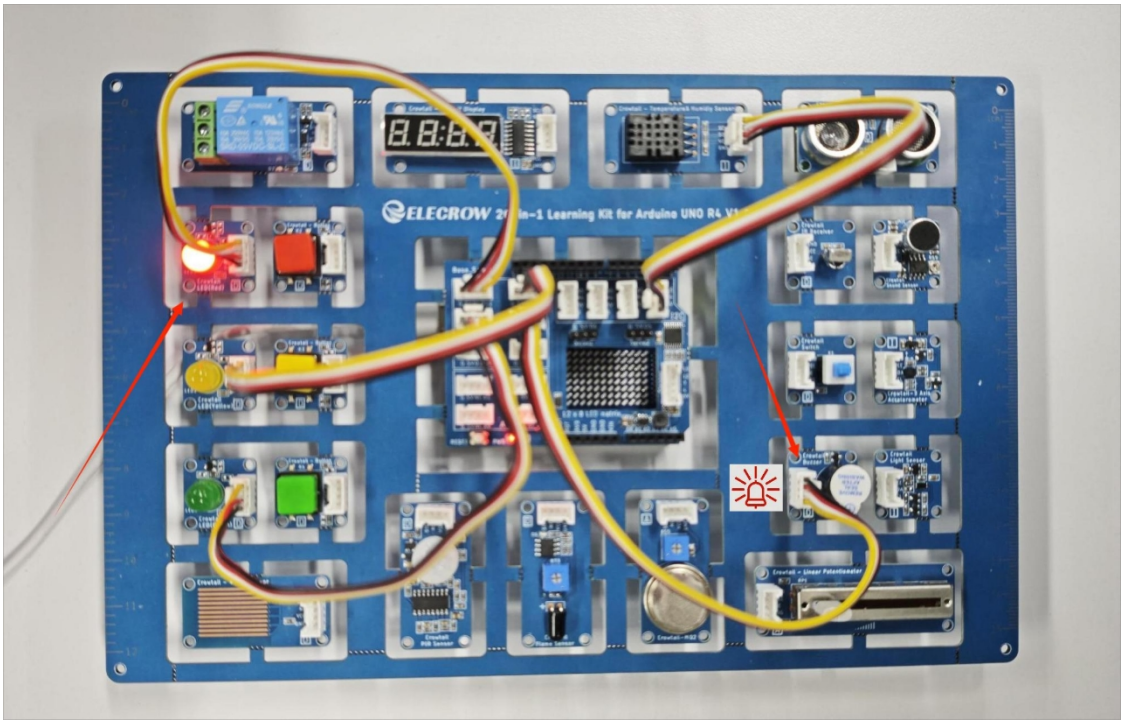
→ Zvukový signál vydá krátke varovné pípnutie



- Príliš vysoká teplota (≥ 32 ° C)

→ Bliká červené svetlo

→ Pípanie vydáva núdzový alarm



1. Vysvetlenie princípu

Princípy fungovania jednotlivých modulov boli podrobne vysvetlené v predchádzajúcich kurzoch. Nasledujúce kurzy budú zahŕňať komplexné praktické cvičenia k predchádzajúcim modulom a niekoľko zaujímavých experimentov. Samozrejme, po dokončení štúdia budete schopní používať aj ďalšie moduly na praktické aplikácie.

2. Požadované moduly

Modul snímača teploty a vlhkosti Crowtail × 1

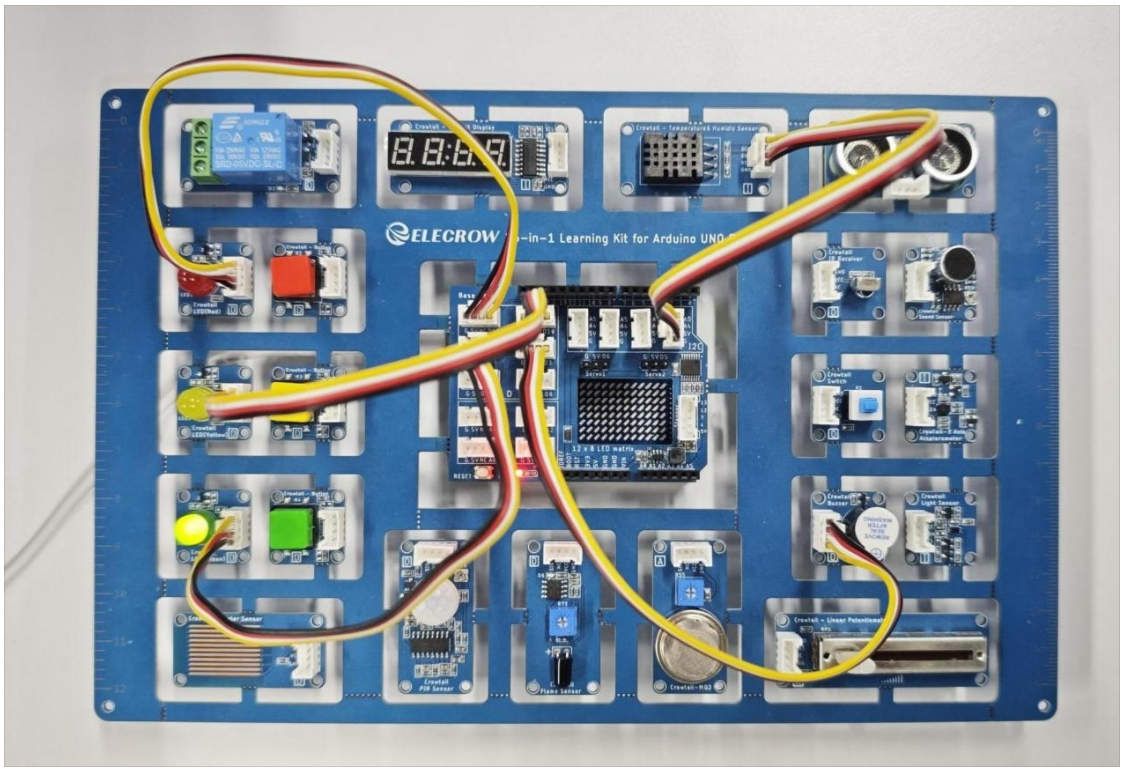
LED diódy Crowtail × 3

Bzučiak Crowtail (1 ks)

3. Spôsob zapojenia

Senzor teploty a vlhkosti Crowtail → ľubovoľný port **I2C** LED Crowtail (ČERVENÁ) → digitálny port **D11**

LED Crowtail (žltá) → port DIGITAL **D10** LED Crowtail (zelená) → port DIGITAL **D9** Bzučiak Crowtail → port Digital **D3**



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite vzorový kód poskytnutý oficiálnym zdrojom:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-22_Intelligent_Temperature_Warning1/22_Intelligent_Temperature_Warning1

Otvorte program kurzu v priečinku **„22_Intelligent_Temperature_Warning1“** pomocou Arduino IDE:

22_Intelligent_Temperature_Warning1.ino

```
#include <Wire.h>
#include "DHT20.h"

#define LED_RED     11
#define LED_YELLOW  10
#define LED_GREEN   9
#define BUZ         3

DHT20 DHT;

// Temperature threshold (can be modified as needed)
#define TEMP_NORMAL_MAX   28.0
#define TEMP_HIGH_MAX     32.0

void setup() {
  Serial.begin(115200);
  Wire.begin();
  DHT.begin();

  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(BUZ, OUTPUT);

  Serial.println("=== Intelligent Temperature Warning System ===");
}

void loop() {
  if (millis() - DHT.lastRead() >= 1000) {   // ⭐Key point: 1 time per second
    int status = DHT.read();

    if (status != DHT20_OK) {
      return;
    }

    float temp = DHT.getTemperature();
    float humi = DHT.getHumidity();

    Serial.print("Temp: ");
    Serial.print(temp);
    Serial.print(" °C  |  Humidity: ");
    Serial.print(humi);
    Serial.println(" %");

    // ====== state judgement ======
    if (temp < TEMP_NORMAL_MAX) {
```

Vysvetlenie kódov klávesov

Vysvetlenia v kóde tejto lekcie sme podrobne rozoberali v našich predchádzajúcich kurzoch.

Preto sa tu nebudeme podrobne zaoberať. Hlavným cieľom je pomôcť vám zopakovať a upevniť si vedomosti.

(1) Použité súbory knižníc

```
#include <Wire.h>

#include "DHT20.h"
```

Tieto dva riadky, **„#include <Wire.h>“** a **„#include „DHT20.h**““, tvoria základ celého systému merania teploty a vlhkosti na „čítanie reálnych údajov“:

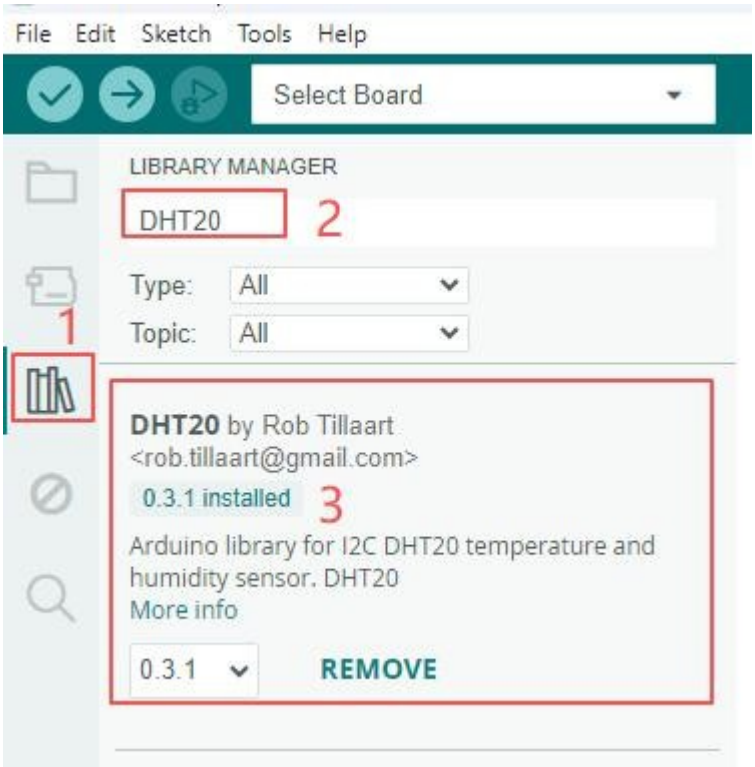
„Wire.h“ predstavuje komunikačnú knižnicu I2C pre Arduino, ktorá zabezpečuje samotný prenos dát. Je to ako keby sme mikrokontrolér vybavili „spoločným jazykom, ktorý dokáže komunikovať so zariadeniami I2C“;

Zatiaľ čo „DHT20.h“ je funkčné zapuzdrenie senzora **teploty** a **vlhkosti** DHT20 na báze I2C. Skryje zložité načasovanie komunikácie, čítanie a zapisovanie registrov atď. a priamo vám poskytuje **„begin()“, „read()“, „getTemperature()“ a „getHumidity()“** – tieto aplikačne orientované funkcie rozhrania. Podstatou týchto dvoch riadkov kódu je teda: najprv aktivovať komunikačnú schopnosť I2C a potom načítať „ovládačový modul“ špecificky určený na prevádzku senzora DHT20.

Na používanie tejto knižnice môžete zvoliť nasledujúce metódy:

➢ Stiahnutie v prostredí Arduino IDE

Verzia knižnice DHT20, ktorú používame, je 0.3.1.



(2) Definícia pinov + Objektovo orientované programovanie + Nastavenie teplotného prahu

```
#define LED_RED          11
#define LED_YELLOW       10
#define LED_GREEN        9
#define BUZ              3

DHT20 DHT;
```

```
// Teplotný prah (možno podľa potreby upraviť)
#define TEMP_NORMAL_MAX          28,0
#define TEMP_HIGH_MAX            32,0
```

Účelom tohto kódu je stanoviť jasné a prehľadné kritériá pre mapovanie hardvéru a rozhodovanie v rámci celého „Inteligentného systému varovania pred teplotou“:

Prvé štyri riadky používajú „#define“ na priradenie **červeného**, **žltého** a **zeleného** svetla a **bzučiaka** ku konkrétnym **IO pinom (11, 10, 9, 3)**. Týmto spôsobom v nasledujúcom kóde jednoduché napísanie **„LED_RED“** alebo „**BUZ**“ môže priamo označiť „ktorú súčasť ovládať“, čím sa vyhneme problémom s čitateľnosťou a údržbou spôsobeným priamym zapisovaním čísel. **„DHT20 DHT;**“ Potom vytvorte inštanciu objektu senzora DHT20, ktorý predstavuje „senzor teploty a vlhkosti fyzicky pripojený k zbernici I2C“. Všetky nasledujúce odčítania teploty a vlhkosti sa vykonávajú prostredníctvom tohto objektu.

Posledné dve definície makier pre teplotné prahové hodnoty, **„TEMP_NORMAL_MAX“** a **„TEMP_HIGH_MAX“**, sú v podstate umelo vytvorené „hraničné čiary“, ktoré delia spojité hodnoty teploty na tri logické intervaly: „normálny“, „varovanie“ a „alarm“. To umožňuje programu používať jasné podmienené rozhodnutia na spustenie rôznych správaní svetiel a bzučiakov.

(3) Sekcia nastavenia

```
void setup() {
    Serial.begin(115200);
    Wire.begin();
    DHT.begin();

    pinMode(LED_RED, OUTPUT);
    pinMode(LED_YELLOW, OUTPUT);
    pinMode(LED_GREEN, OUTPUT); pinMode(BUZ,
    OUTPUT);

    Serial.println("=== Inteligentný systém varovania pred teplotou ===");
}
```

Hlavnou funkciou tohto kódu setup() je dokončiť „počiatočné nastavenie“ celého inteligentného systému varovania pred teplotou po zapnutí:

Serial.begin(115200) otvorí sériový komunikačný kanál, ktorý sa používa na následné

ladenie a sledovanie teploty, vlhkosti a stavu systému v reálnom čase;

Funkcia `Wire.begin()` inicializuje zbernicu I2C, ktorá je nevyhnutným základným rozhraním pre komunikáciu digitálneho senzora **teploty** a **vlhkosti** DHT20 s hlavnou riadiacou jednotkou; ` ` a **`DHT.begin()`** slúžia na inicializáciu samotného senzora DHT20 v momente, keď je zbernica I2C už pripravená, čo je ekvivalentné signálu pre senzor: „systém je pripravený, môžeš začať pracovať“;

Potom skupina **pinMode(..., OUTPUT)** jasne špecifikuje, že piny **červenej**, **žltej** a **zelenej** LED a **bzučiaka** sú v režime výstupu, čo umožňuje mikrokontroléru aktívne ovládať kontrolky a výstražné zariadenie;

Nakoniec sa prostredníctvom **Serial.println(...)** vypíše informácia o spustení systému, čo nielen umožňuje používateľom potvrdiť, že program bol úspešne spustený, ale slúži aj na „vizualizáciu stavu systému“ pri výučbe a ladení.

(4) loop

```
void loop() {
  if (millis() - DHT.lastRead() >= 1000) {          // ☆Kľúčový bod: 1-krát za sekundu
    int status = DHT.read();

    if (status != DHT20_OK) {
      return;
    }

    float temp = DHT.getTemperature();
    float humi = DHT.getHumidity();

    Serial.print("Teplota: ");
    Serial.print(temp);
    Serial.print(" °C      |   Vlhkosť: ");
    Serial.print(humi);
    Serial.println(" %");

    // ====== posúdenie stavu ====== if
    (temp < TEMP_NORMAL_MAX) {
      normalState();
    }
```

```
      inak ak (temp < TEMP_HIGH_MAX) {
          warningState();
      }
      inak {
          alarmState();
      }
  }
}
```

Tento kód loop() stelesňuje kompletný koncept „získavania údajov zo senzorov na základe časového rytmu + posúdenia stavového automatu“.

① **Periodické čítanie**

```
28  void loop() {
29    if (millis() - DHT.lastRead() >= 1000) {  // ⭐Key point: 1 time per second
30      int status = DHT.read();
31
```

Tu sa používa systémové hodiny Arduina, **millis**(). Ukazujú, „koľko milisekúnd uplynulo od zapnutia mikrokontroléra“. **DHT.lastRead()** je „časová pečiatka posledného úspešného čítania senzora“ zaznamenaná interne knižnicou **DHT20**. V podstate ide tiež o hodnotu **v milisekundách**. Odpočítaním týchto dvoch hodnôt získame „koľko času uplynulo od posledného skutočného

odčítania senzora“. Keď je tento čas ≥ 1000 ms, znamená to, že „teraz je v poriadku odčítať senzor „again“, čo je typický spôsob zápisu softvérového časovača a v integrovaných systémoch sa bežne používa namiesto **funkcie delay**(). Znamená to, že:

- Program nebude blokovaný a hlavná slučka môže naďalej reagovať na iné úlohy kedykoľvek ;
- Senzor nebude čítaný príliš často (samotný DHT20 nepodporuje vysokofrekvenčné vzorkovanie);
- Frekvencia vzorkovania je stabilná a kontrolovateľná (tu je to explicitne 1 Hz).

② **Čítanie hodnôt snímača teploty a vlhkosti**

```
28  void loop() {
29    if (millis() - DHT.lastRead() >= 1000) {  // ⭐Key point: 1 time per second
30      int status = DHT.read();
31
```

Tento riadok kódu nie je taký jednoduchý, ako len „čítanie hodnoty“. Robí nasledovné:

- Odosiela príkaz „začať meranie“ cez zbernicu I2C do DHT20;
- Počká, kým senzor dokončí vnútorné meranie (teplota + vlhkosť);
- Potom načíta výsledky merania späť z registra senzora do MCU;
- Zároveň sa vráti stavový kód (status), ktorý informuje o tom, či bola táto komunikácia

bola úspešná.

③ Spätná väzba o stave

Keďže vždy existuje možnosť zlyhania hardvérovej komunikácie (rušenie I2C, nestabilné napájanie, senzor obsadený atď.), návrhári knižnice neumožnili „priamo používať údaje“. Namiesto toho vám najprv poskytli stav a vy musíte posúdiť:

```
28  void loop() {
29    if (millis() - DHT.lastRead() >= 1000) {   // ⭐Key point: 1 time per second
30      int status = DHT.read();
31
32      if (status != DHT20_OK) {
33        return;
34      }
35
```

④ Získanie teploty a vlhkosti

Následne sa „pôvodné I2C údaje z najnižšej vrstvy“ prevádzajú na fyzikálne veličiny, ktoré ľudia dokážu priamo pochopiť (stupne Celzia, percentuálna vlhkosť), prostredníctvom funkcií **getTemperature**() a **getHumidity**(), a vypíšu sa cez sériový port na účely monitorovania v reálnom čase;

```
36      float temp = DHT.getTemperature();
37      float humi = DHT.getHumidity();
38
39      Serial.print("Temp: ");
40      Serial.print(temp);
41      Serial.print(" °C   |   Humidity: ");
42      Serial.print(humi);
43      Serial.println(" %");
44
```

⑤ Posúdenie stavu

Táto posledná skupina podmienok if/else if/else predstavuje typickú logiku posudzovania stavu **(state judgment)**. Program sa nezaujíma o to, „ako rozsvietiť svetlá alebo ako vygenerovať zvuky“, ale na základe teplotného rozsahu len určí aktuálny stav systému (či ide o „normálny / varovný / poplach“) a následne deleguje konkrétne detaily vykonania na funkcie, ako **sú normalState**(), **warningState**() a **alarmState**(), aby ich dokončili.

```
45      // ====== state judgement ======
46      if (temp < TEMP_NORMAL_MAX) {
47        normalState();
48      }
49      else if (temp < TEMP_HIGH_MAX) {
50        warningState();
51      }
52      else {
53        alarmState();
54      }
55    }
56  }
```

(5) Funkcia normalState

```
void normalState() { digitalWrite(LED_GREEN,
    HIGH); digitalWrite(LED_YELLOW, LOW);
    digitalWrite(LED_RED, LOW);
    digitalWrite(BUZ, LOW);
}
```

V tomto prípade sa súčasne ovládajú tri kontrolky a bzučiak. Prostredníctvom funkcie **digitalWrite**() je jasne stanovené, že v normálnom stave by mala svietiť iba **zelená** LED (**LED_GREEN** = **HIGH**), zatiaľ čo žltá a červená kontrolka musia byť nútene vypnuté a bzučiak zostáva v neaktívnom stave. Kľúčový význam tohto prístupu spočíva v jedinečnosti a istote stavu – akonáhle sa vstúpi do normálneho stavu (normalState()), nebudú existovať žiadne zvyškové výstupné signály z predchádzajúcich stavov (napríklad alarmových alebo varovných stavov).

(6) Funkcia warningState

```
void warningState() {
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_YELLOW, HIGH);
    digitalWrite(LED_RED, LOW);

    digitalWrite(BUZ, HIGH);
    delay(100); digitalWrite(BUZ,
    LOW);
}
```

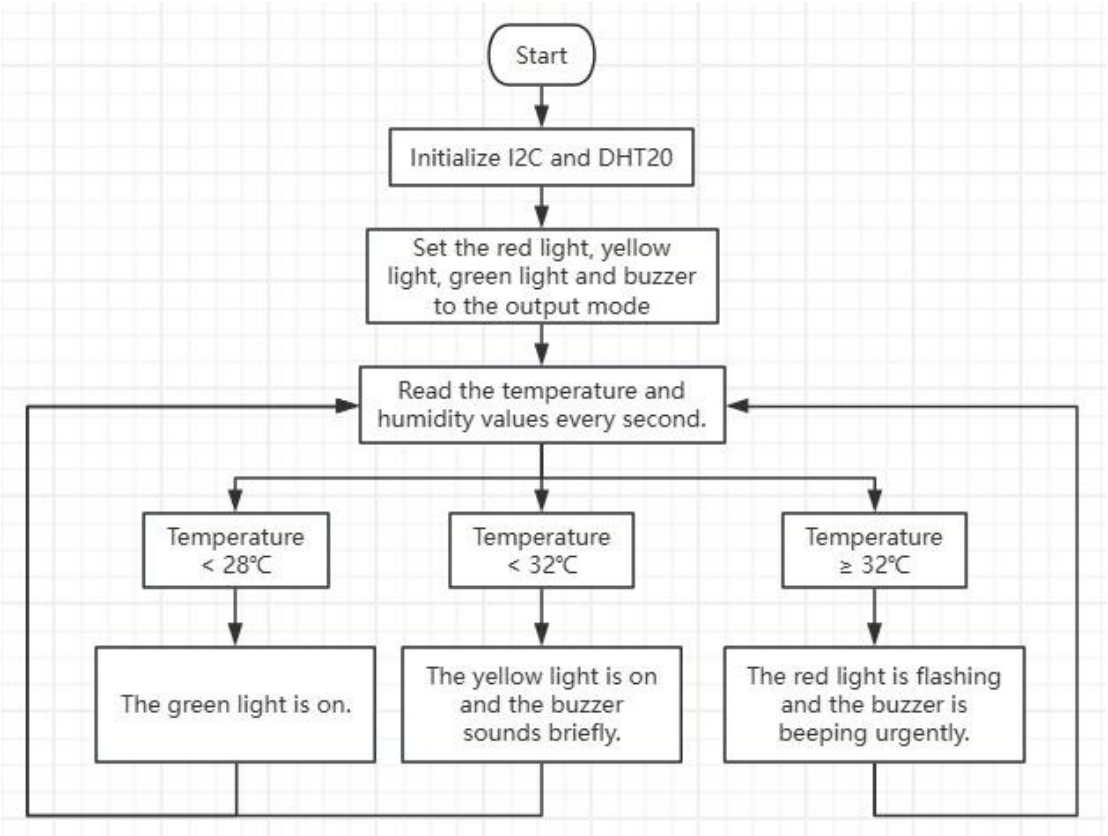
Z hľadiska významu kódu sú najprv prostredníctvom troch volaní **digitalWrite**() jasne definované pravidlá osvetlenia: **žltá** LED svieti, zatiaľ čo **zelená** a **červená** LED sú vypnuté. Týmto spôsobom sa pomocou farieb intuitívne odovzdáva používateľovi správa „pozor, ale nie je to naliehavé“;
V tomto prípade nie je ovládanie **bzučiaka** nastavené na nepretržitý zvuk, ale na jednorazové krátke pípnutie (100 ms). Ide o veľmi typický prístup v technickom dizajne – použitie „prerušovaných, krátkych“ zvukov na rozlíšenie medzi „varovaním“ a „alarmom“. **Oneskorenie** (100) tu neslúži na zablokovanie samotného programu, ale na umelé vytvorenie počuteľnej dĺžky zvuku bzučiaka, aby bzučiak skutočne „vydal zvuk“, namiesto toho, aby sa náhle zvýšil a znížil, čím by bol zvuk nepočuteľný.

(7) Funkcia alarmState

```
void alarmState() { digitalWrite(LED_GREEN,
    LOW); digitalWrite(LED_YELLOW, LOW);


    digitalWrite(LED_RED, HIGH);
    digitalWrite(BUZ, HIGH);
    delay(300); digitalWrite(LED_RED,
    LOW); digitalWrite(BUZ, LOW);
    delay(300);
}
```

Z hľadiska významu kódu funkcia okamžite nastaví **zelenú** a **žltú** LED diódu na **stav LOW**, čím jasne vypne všetky nealarmové kontrolky, aby sa zabránilo rušeniu informácií; následne súčasne zapne červenú LED diódu a bzučiak a pomocou **delay**(**300**) ich udrží zapnuté po dobu **300** milisekúnd, čo je krok k „vytvoreniu jasne vnímateľného alarmového momentu“; následne vypne červené svetlo a bzučiak a potom počká ďalších **300** milisekúnd. Toto striedavé zapínanie a vypínanie červeného svetla a prerušované bzučanie bzučiaka vytvára rytmus alarmu „blikajúce červené svetlo + prerušované bzučanie bzučiaka“. Účel použitia delay() v tomto návrhu je veľmi jasný: neslúži na programovú logiku, ale na umelé vytvorenie rytmu a vizuálnych/zvukových efektov alarmu, čo umožňuje ľuďom okamžite rozlíšiť „stav alarmu“ od predchádzajúceho „varovného stavu“.

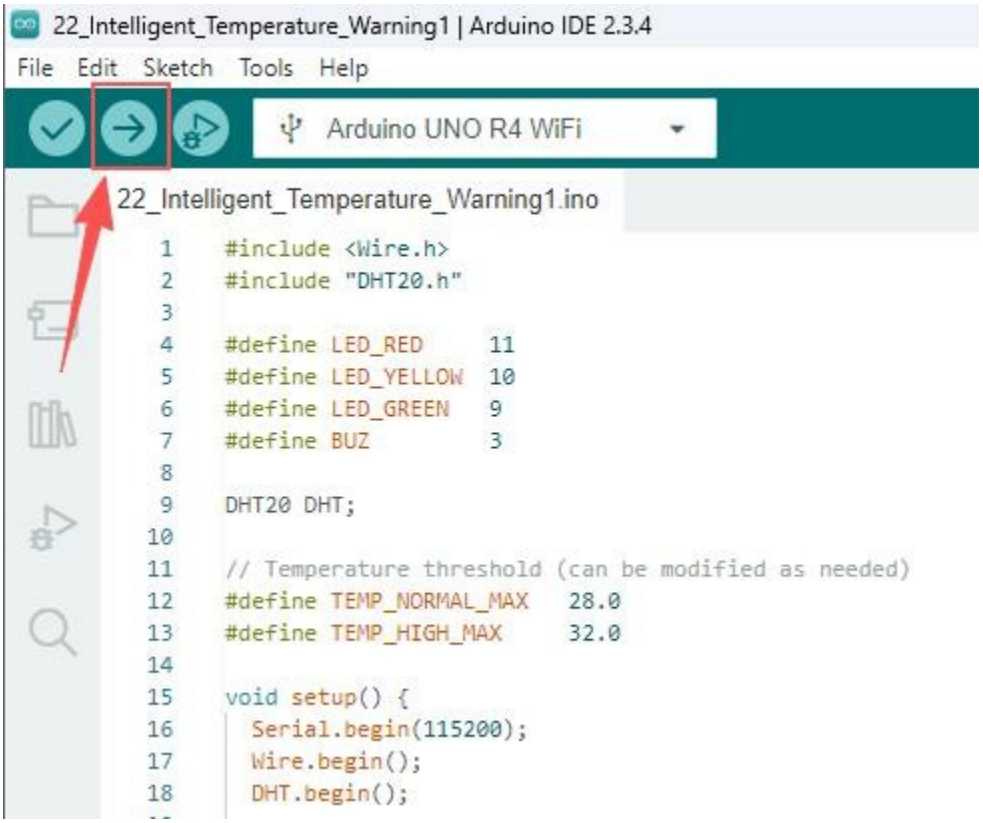
(8) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov obsluhy Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahratia.

Kliknite na **„Stiahnuť“:**

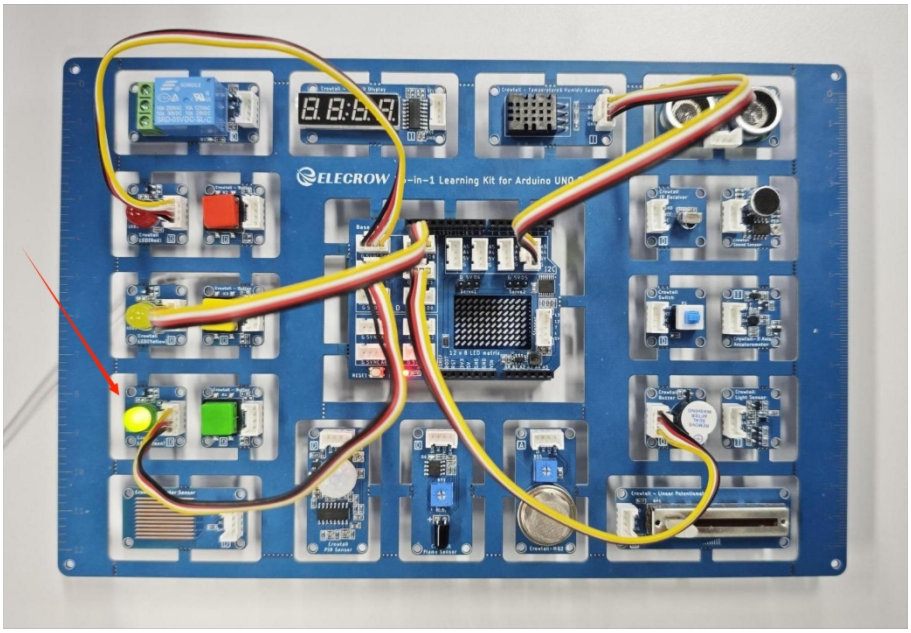


(2) Akonáhle systém začne fungovať, všimnete si nasledujúce účinky:

- Normálna teplota (menej ako 28 ° C)

→ Zelené svetlo svieti nepretržite

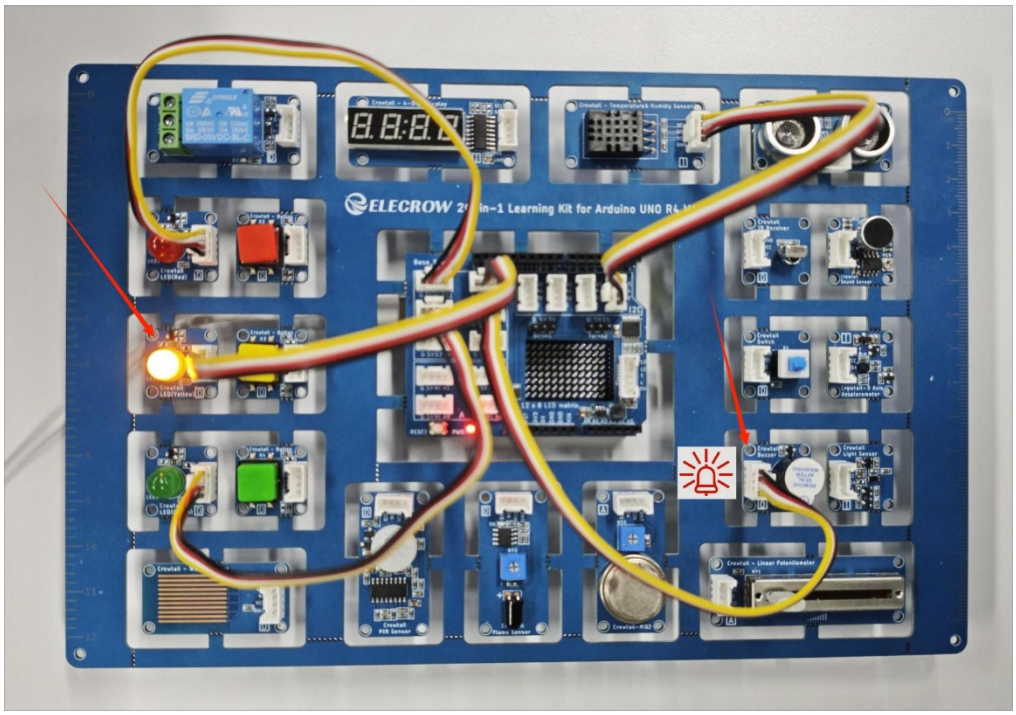
→ Zvukový signál nefunguje



- Teplota je príliš vysoká (28 ° C ~ 32 ° C)

→ Žlté svetlo svieti nepretržite

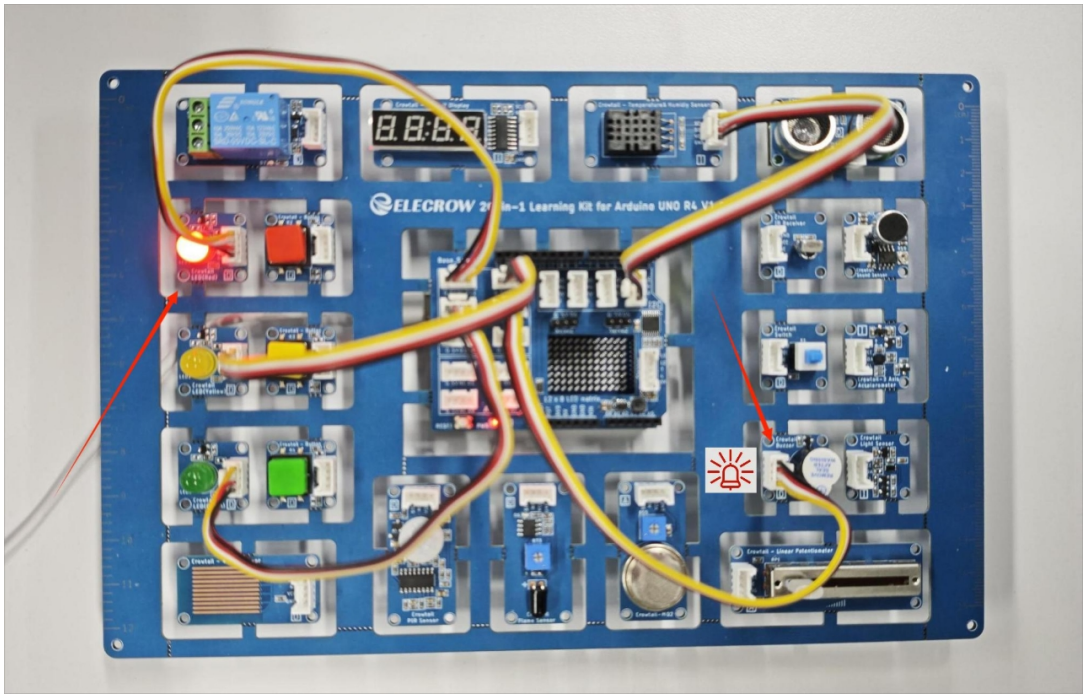
→ Zvukový signál vydá krátke varovné pípnutie



- Nadmerná teplota (≥ 32 ° C)

→ Červené svetlo bliká

→ Zvukový signál vydá naliehavý alarm



Lekcia 23 – Monitorovanie domáceho prostredia

Úvod

V predchádzajúcich kurzoch sme sa postupne naučili základné spôsoby použitia snímačov teploty a vlhkosti, svetelných snímačov, LCD displejov a bzučiakov. V tejto lekcii spojíme tieto zdanlivo nezávislé moduly a zostavíme kompletnú domáce monitorovaciu stanicu.

V tomto projekte:

- DHT20 zodpovedá za snímanie teploty a vlhkosti prostredia
- BH1750 je zodpovedný za snímanie intenzity osvetlenia prostredia
- LCD1602 je zodpovedný za zobrazovanie informácií o prostredí v reálnom čase
- Bzučiak je zodpovedný za vydávanie zvukových upozornení, keď je prostredie v abnormálnom stave

Vďaka tejto lekcii prekročíte rámec základnej prevádzky „jednoduchého čítania hodnôt senzorov“ a dôkladne zvládnuť kompletný súbor logiky inteligentnej interakcie: umožniť systému nepretržite vnímať stav okolia, automaticky identifikovať nezvyčajné situácie a aktívne spúšťať okamžitú spätnú väzbu.

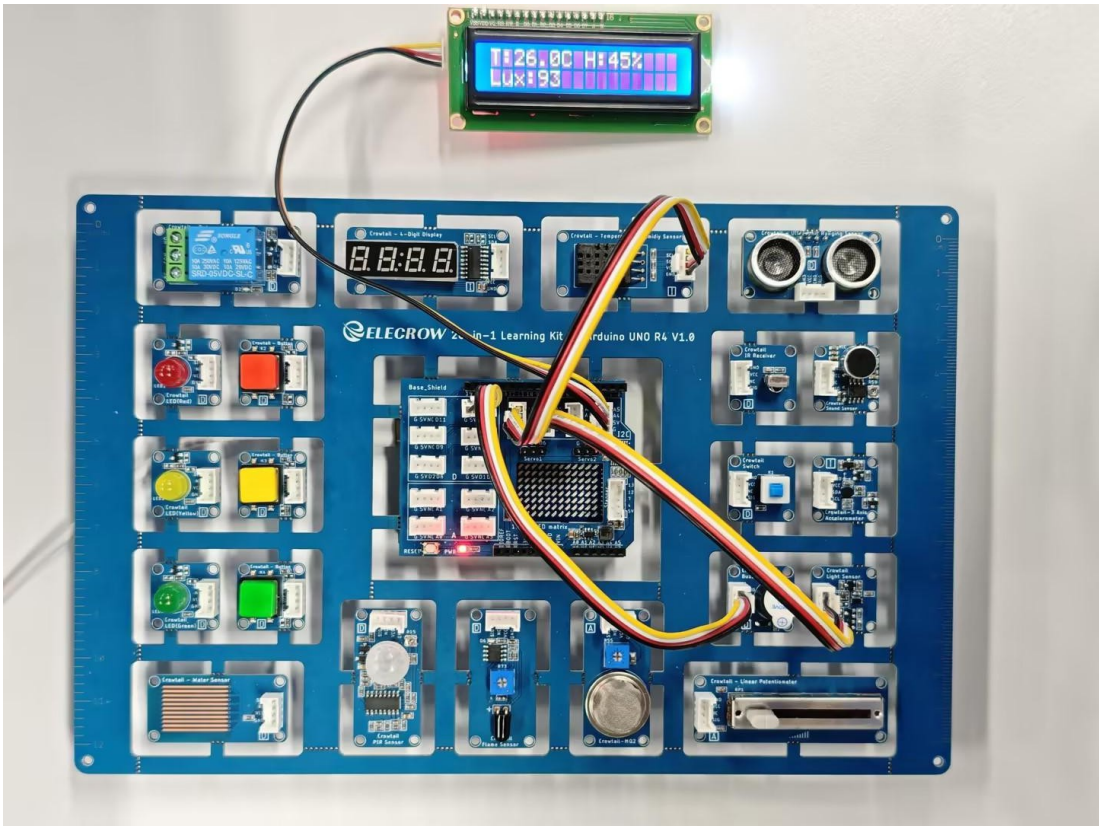
Ciele vzdelávania

1. Preverenie a upevnenie spôsobu použitia zbernice I2C v systémoch s viacerými senzormi. 2. Porozumenie tomu, ako súčasne čítať údaje z viacerých senzorov a spravovať ich jednotne.
3. Zopakovať a upevniť si základnú metódu dynamického obnovovania obsahu displeja LCD1602.
4. Zopakovať a upevniť si úlohu podmieneného posúdenia (if) v systémoch monitorovania prostredia a poplachových systémoch.
5. Realizujte kompletný projekt domácej stanice na monitorovanie životného prostredia.

Náhľad výsledku

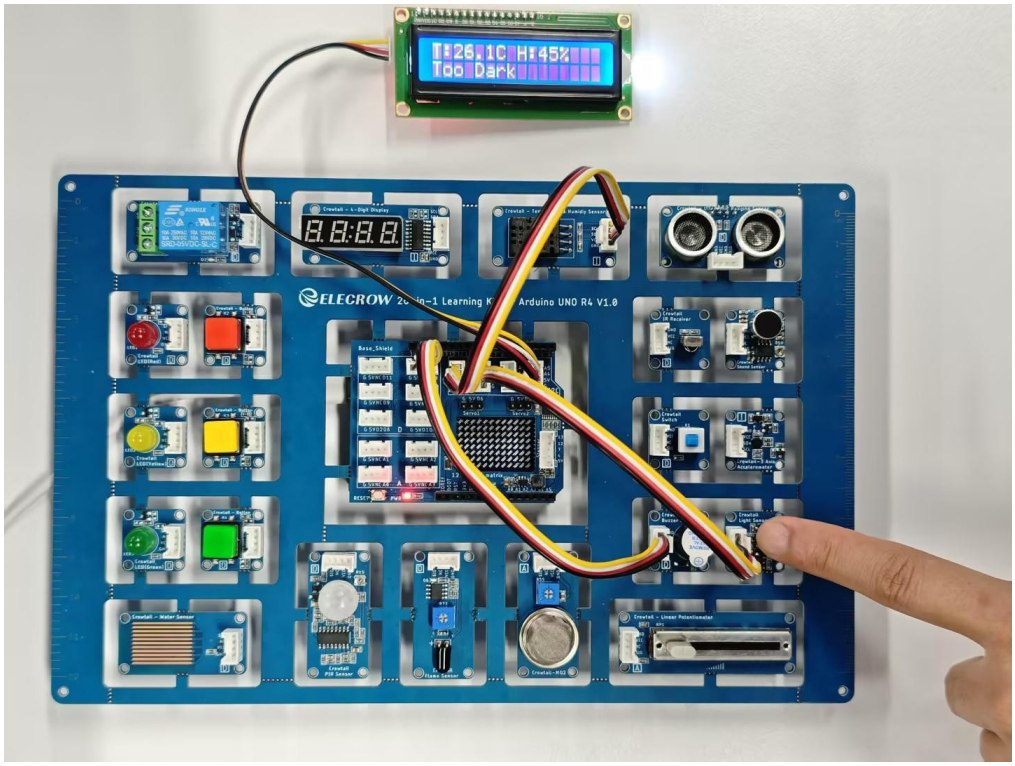
Po stiahnutí a spustení programu uvidíte:

Prvý riadok LCD displeja zobrazuje v reálnom čase: aktuálnu teplotu (°C) a aktuálnu vlhkosť (%).



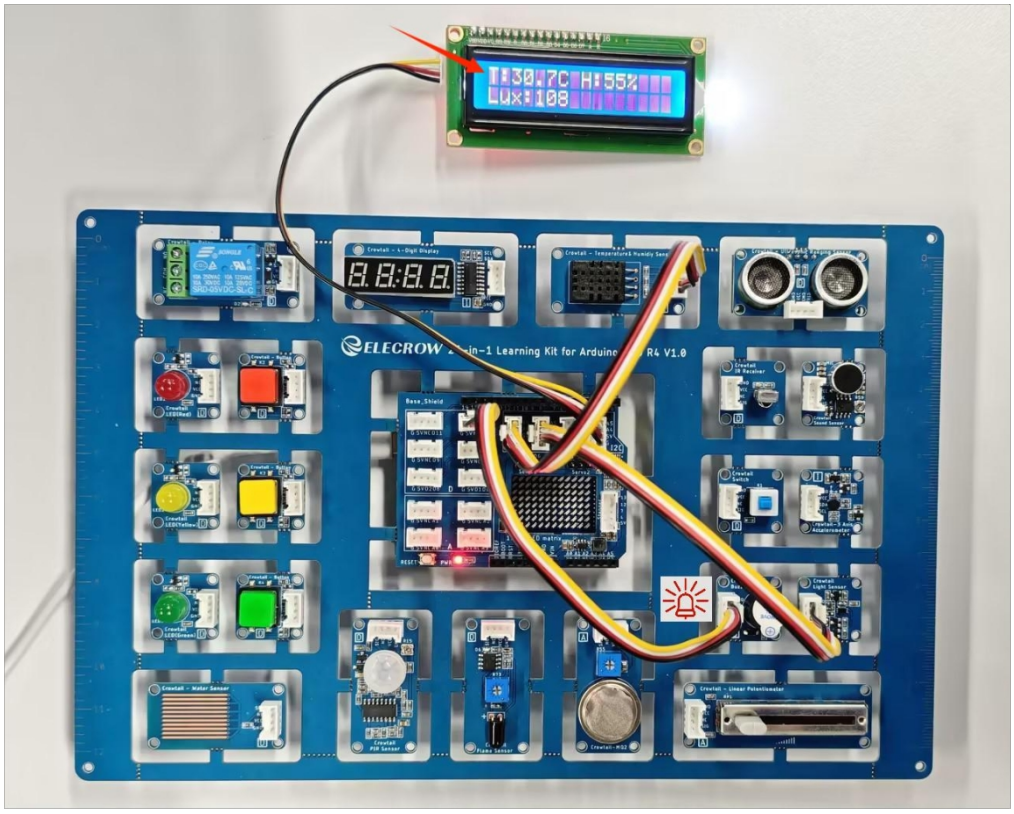
Druhý riadok LCD displeja sa dynamicky mení v závislosti od svetelných podmienok: Normálne osvetlenie → Zobrazuje aktuálnu hodnotu v luxoch

Príliš málo svetla → Zobrazuje „Too Dark“



Ak je teplota vyššia ako $30^\circ C$:

Bzučiak vydá krátky varovný signál, aby upozornil na prehriatie.

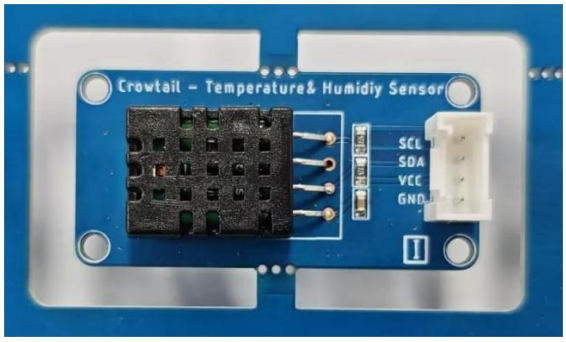


1. Vysvetlenie princípu

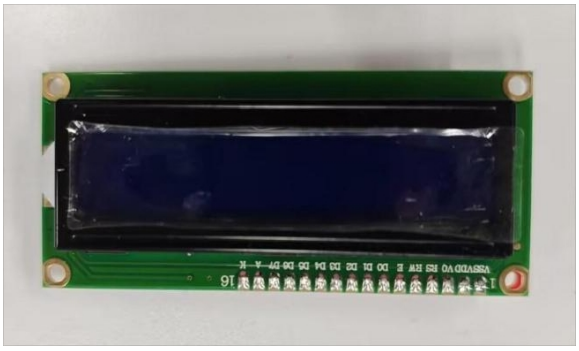
Princípy fungovania jednotlivých modulov boli podrobne vysvetlené v predchádzajúcich kurzoch. Nasledujúce kurzy budú zahŕňať komplexné praktické cvičenia založené na predchádzajúcich moduloch a vykonanie niekoľkých zaujímavých experimentov. Samozrejme, po dokončení štúdia budete schopní použiť viac modulov aj na praktické účely.

2. Potrebné moduly

Modul snímača teploty a vlhkosti Crowtail × 1



Modul Crowtail LCD1602 × 1



Modul svetelného senzora Crowtail × 1

Bzučiak Crowtail (1 ks)

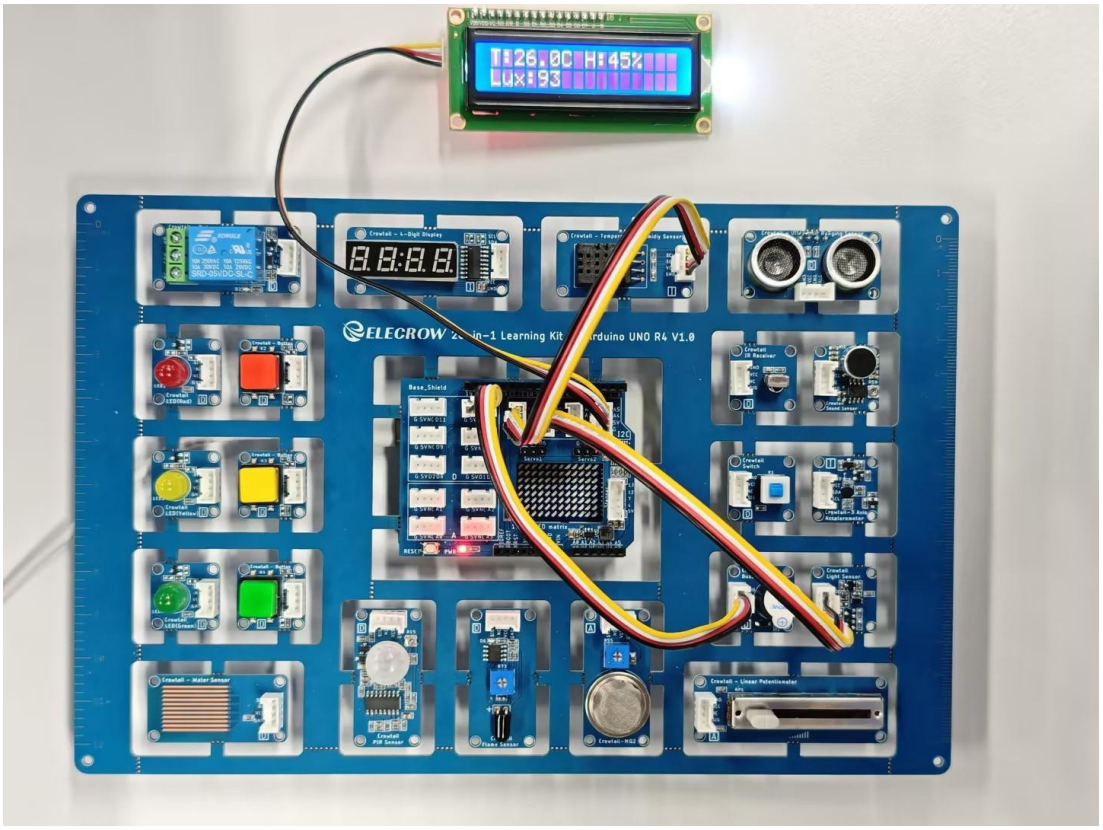
3. Spôsob zapojenia

Senzor teploty a vlhkosti Crowtail → ľubovoľný port I2C LCD1602

Crowtail → ľubovoľný port I2C

Snímač osvetlenia Crowtail → ľubovoľný port

I2C Bzučiak Crowtail → digitálny port D10



4. Vysvetlenie príkladu

Kliknutím na odkaz si stiahnite oficiálny vzorový kód:

Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-23_Home_Environment_Monitoring/23_Home_Environment_Monitoring

Otvorte program kurzu v priečinku **„23_Home_Environment_Monitoring“** pomocou Arduino IDE:

23_Home_Environment_Monitoring.ino

```
#include <Wire.h>
#include "DHT20.h"
#include <BH1750.h>
#include "Adafruit_LiquidCrystal.h"

// ================== object definition ==================
DHT20 DHT;
BH1750 lightMeter(0x5c);
Adafruit_LiquidCrystal lcd(0);

// ================== macro definition ==================
#define BUZ 10
#define TEMP_ALARM 30.0
#define DARK_LUX 50.0

// ================== variable ==================
float temperature = 0;
float humidity = 0;
float lux = 0;

// ================== initialize ==================
void setup() {
  Serial.begin(115200);
  Wire.begin();

  // ---- DHT20 ----
  DHT.begin();
  delay(1000);

  // ---- BH1750 ----
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
    Serial.println("BH1750 init OK");
  } else {
    Serial.println("BH1750 init FAIL");
  }

  // ---- LCD1602 ----
  while (!lcd.begin(16, 2)) {
    Serial.println("LCD init failed");
    delay(50);
  }
```

Vysvetlenie kľúčového kódu

Mnohé z kódov v tomto úryvku kódu sme podrobne vysvetlili v našich predchádzajúcich kurzoch. Tu ich pre vás len stručne zopakujeme a zhrnieme, bez toho, aby sme sa venovali ich podrobnému vysvetľovaniu.

(1) Použité súbory knižníc

```
#include <Wire.h>
#include "DHT20.h"
#include <BH1750.h>
#include "Adafruit_LiquidCrystal.h"
```

<Wire.h> poskytuje najzákladnejšie a najdôležitejšie funkcie komunikácie zbernice I2C v Arduine. Keďže nasledujúce moduly **DHT20**, **BH1750** a **LCD1602** komunikujú s hlavným riadiacim čipom prostredníctvom **I2C**, bez neho by celý systém „nevedel komunikovať“;

„DHT20.h“ zavádza knižnicu ovládačov pre senzor teploty a vlhkosti. Zahrňuje komplexné načasovanie I2C, výpočet kontrolného súčtu atď. do intuitívnych rozhraní, ako sú **„read**()“ a

„getTemperature()“, čo nám umožňuje písať kód s prístupom „čítať teplotu, čítať vlhkosť“;

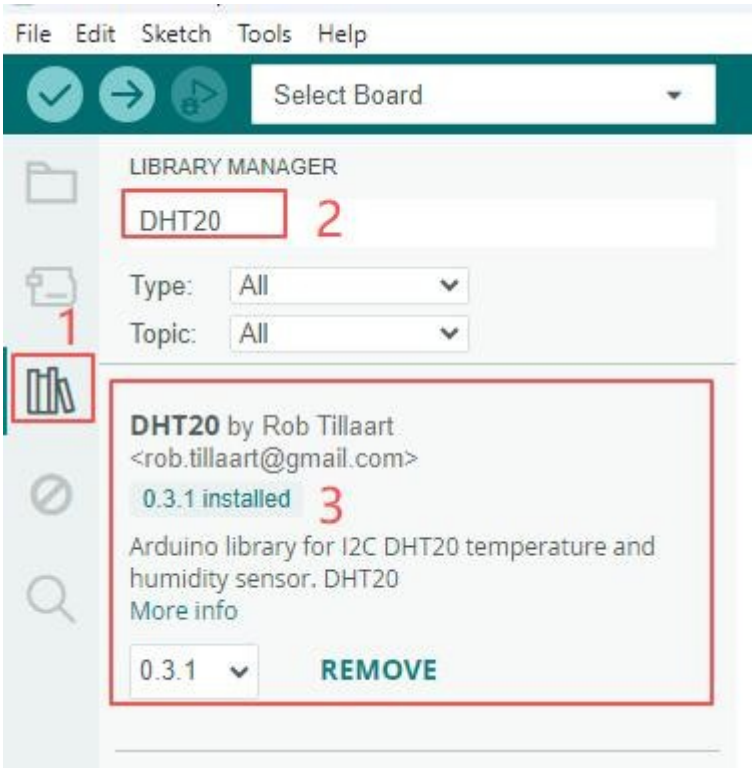
<BH1750.h> je knižnica pre senzor intenzity osvetlenia, ktorá prevádza surové údaje o intenzite osvetlenia z **BH1750** na štandardné jednotky lux a poskytuje profesionálne rozhrania, ako napríklad „či je meranie dokončené“ a „načítanie hodnoty osvetlenia“;

Nakoniec, **„Adafruit_LiquidCrystal.h“** je knižnica na ovládanie displeja pre **LCD1602**. Zabezpečuje komunikáciu medzi rozširujúcim čipom I2C a znakovým LCD na najnižšej úrovni, čo nám umožňuje sústrediť sa len na to, „kde sa nachádza kurzor a aký text sa má zobraziť“.

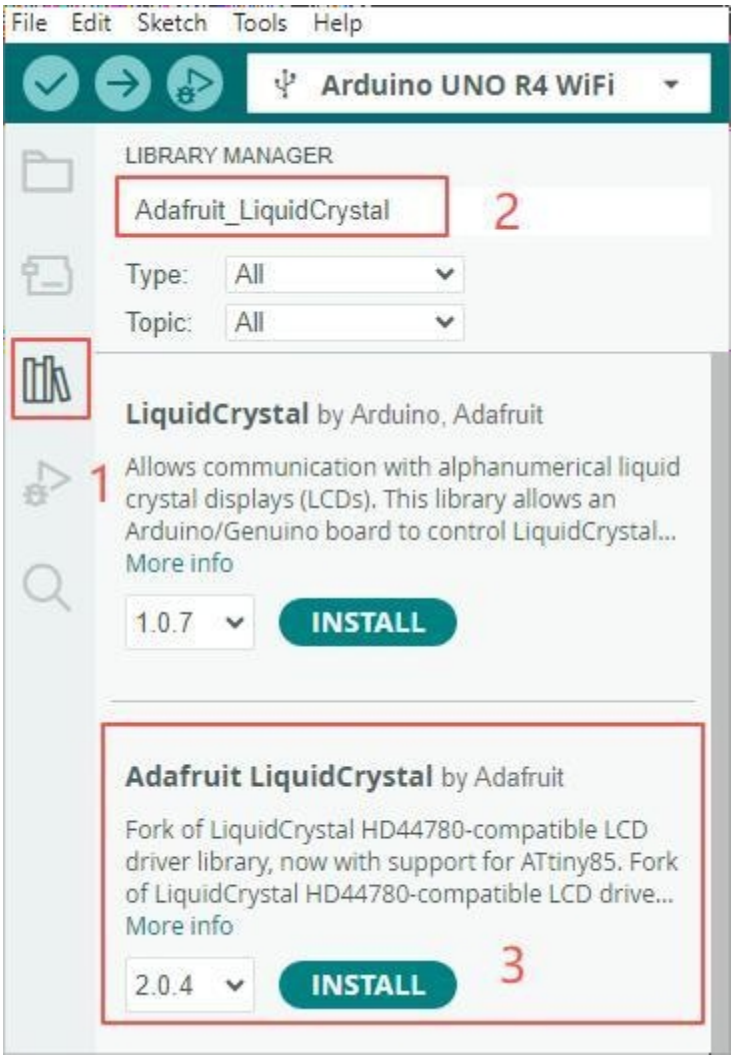
Tieto knižnice možno použiť nasledujúcimi spôsobmi:

➢ Stiahnutie v prostredí Arduino IDE

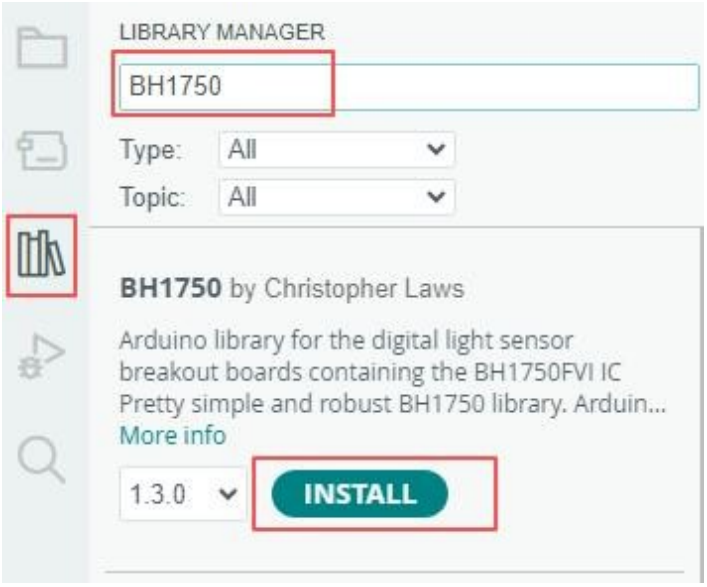
Verzia knižnice DHT20, ktorú používame, je 0.3.1.



Verzia knižnice Adafruit LiquidCrystal, ktorú používame, je 2.0.4.



Podobne aj potrebný ovládač na inštaláciu fotosenzora (my sme použili verziu 1.3.0 pre BH1750)



(2) Inštanciujte objekt

```
DHT20 DHT;

BH1750 lightMeter(0x5c);

Adafruit_LiquidCrystal lcd(0);
```

Tieto tri riadky kódu slúžia na „skutočné vytvorenie inštancií“ troch hardvérových knižníc, ktoré boli predtým naimportované, a to vo forme použiteľných objektov. Dá sa to tiež chápať tak, že v programe sa vytvára „softvérová replika“ pre senzor teploty a vlhkosti, senzor osvetlenia a LCD displej. **„DHT20 DHT;“** Vytvorí objekt s názvom „**DHT**“ pre senzor DHT20. Obsahuje už zapuzdrenú logiku potrebnú na čítanie a zapisovanie registrov, overovanie a konverziu dát pre komunikáciu so senzorom DHT20 cez I2C. Následne stačí na pokračovanie volať rozhrania ako **„DHT.read()“** a **„DHT.getTemperature()**“. **„BH1750 lightMeter(0x5c);“** Pri vytváraní objektu svetelného senzora sa potom programu výslovne oznamuje, že adresa zariadenia I2C pre **BH1750** je **0x5C**. Ide o konvenciu adries na hardvérovej úrovni, ktorá odráža základnú vlastnosť zbernice I2C, a to „viacero zariadení zdieľajúcich tú istú linku s rozlíšením adries“;
A **„Adafruit_LiquidCrystal lcd(0)**;“ vytvára objekt LCD displeja. Tu parameter **„0“** zvyčajne zodpovedá predvolenej adrese rozširujúceho čipu I2C alebo konfigurácii zbernice, čo umožňuje následné použitie pokročilých rozhraní, ako sú **„lcd.print()“** a **„lcd.setCursor()“**, na priamy výstup obsahu na obrazovku. Celkovo tieto tri riadky kódu realizujú veľmi dôležitý, ale často prehliadaný krok: premenu „knižnice“ na „objekt“, čím sa abstraktná schopnosť riadenia viaže na konkrétnu hardvérovú inštanciu. Ide o typické uplatnenie objektovo orientovaného myslenia v embedded systémoch a zároveň o základ pre stabilnú a kooperatívnu spoluprácu viacerých zariadení I2C v rámci jedného programu.

(3) Definovanie pinov a prahových hodnôt

```
#define BUZ 10

#define TEMP_ALARM 30.0

#define DARK_LUX 50.0
```

#define BUZ 10 Jasne oznamuje programu, že bzučiak je pripojený k **digitálnemu pinu 10**. Nahraďte „konkrétne číslo pinu“ výstižným názvom, aby ste pri pohľade **na digitalWrite(BUZ, HIGH)** na prvý pohľad pochopili hardvérový význam kódu;
#define TEMP_ALARM 30.0 Definuje prah teplotného alarmu, pričom jednotkou sú stupne Celzia. Počas vykonávania programu stačí, aby program určil, či aktuálna

teplota prekročí túto hodnotu, aby sa rozhodlo, či sa spustí alarm. Základnou riadiacou myšlienkou použitou v tomto riadku kódu je „porovnanie prahovej hodnoty + posúdenie čísla s pohyblivou desatinnou čiarkou“; zatiaľ čo **#define DARK_LUX 50.0** definuje kritérium pre príliš slabé okolité osvetlenie. Keď BH1750 zmeria intenzitu osvetlenia nižšiu ako 50 luxov, systém považuje aktuálne prostredie za príliš tmavé a na LCD displeji zobrazí „Too Dark“ (Príliš tmavé) alebo vykoná príslušnú logiku.

(4) Vytvorenie „globálnych stavových premenných“

```
float temperature = 0;
float humidity = 0;
float lux = 0;
```

Tieto tri riadky kódu slúžia na vytvorenie „globálnych stavových premenných“ pre tri najdôležitejšie typy environmentálnych údajov v systéme. Tieto premenné slúžia ako kontajnery na dlhodobé ukladanie a opakované použitie výsledkov merania senzorov. **„float temperature = 0;“** Slúži na ukladanie aktuálnej okolitej teploty nameranej senzorom DHT20. Keďže teplota má zvyčajne

ak má hodnota desatinnú časť (napríklad 26,3 °C), je potrebné použiť typ s pohyblivou desatinnou čiarkou „float“. **„float**
humidity = 0;“ Podobne sa „float humidity = 0;“ používa na ukladanie percentuálnej hodnoty vlhkosti prostredia, čo je tiež typická spojitá fyzikálna veličina;
zatiaľ **čo „float lux = 0;“** sa špecificky používa na ukladanie hodnoty intenzity osvetlenia meranej BH1750 (v luxoch), čo sú tiež údaje s vysokými požiadavkami na presnosť.

(5) Sekcia nastavenia

```
void setup() {
    Serial.begin(115200);
    Wire.begin();

    // ---- DHT20 ----
    DHT.begin();
    delay(1000);

    // ---- BH1750 ----
    if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
        Serial.println("BH1750 init OK");
    } else {
```

```
        Serial.println("Inicializácia BH1750 zlyhala");
    }

    // ---- LCD1602 ----
    while (!lcd.begin(16, 2)) {
        Serial.println("Inicializácia LCD
        zlyhala"); delay(50);
    }
    lcd.setBackLight(1); lcd.clear();

    // ---- bzučiak ----
    pinMode(BUZ, OUTPUT);
    digitalWrite(BUZ, LOW);

    Serial.println("Mini Environment Station Ready!");
}
```

Celá táto funkcia **„setup**()“ v podstate vykonáva „jednotnú kontrolu pripojenia a inicializáciu“ všetkých hardvérových modulov po zapnutí systému, čím zabezpečuje, že každé periférne zariadenie je v použiteľnom stave, keď sa spustí nasledujúci program.

Najskôr sa pomocou príkazu „**Serial.begin(115200)**“ otvorí sériový port na účely ladenia a výstupu stavu, zatiaľ čo príkaz „Wire.begin()“ inicializuje zbernicu I2C, keďže senzory DHT20, BH1750 a LCD využívajú komunikáciu cez I2C;

Potom sa volá **„DHT.begin()**“ na spustenie senzora teploty a vlhkosti a zámerne sa používa **„delay(1000)“**, aby mal senzor dostatočný čas na stabilizáciu po zapnutí, čo je veľmi typická a dôležitá technická skúsenosť pre hardvér typu senzorov.

Ďalej sa vykoná inicializácia svetelného senzora **BH1750**, kde sa použije **„lightMeter.begin(...)“** a vrátená hodnota sa skontroluje prostredníctvom „if“, čo demonštruje profesionálnu prax, že „inicializácia periférie musí skontrolovať úspešnosť“, a úspech alebo zlyhanie bude indikované cez sériový port.

Následne prebehne inicializácia modulu **LCD1602** pomocou **príkazu „while (!lcd.begin(16, 2))“**, ktorý čaká, kým LCD normálne zareaguje, čím sa zabezpečí, že zobrazovacie zariadenie je skutočne k dispozícii, a až potom sa pokračuje v spúšťaní programu. Následne sa zapne podsvietenie a vymaže obrazovka, aby bolo zabezpečené čisté zobrazovacie rozhranie.

Nakoniec sa pomocou „pinMode“ vykoná konfigurácia pinu bzučiaka a nastaví sa na predvolenú hodnotu low, aby sa zabránilo falošnému pípaniu pri zapnutí.

(6) Slučka

```
// ================== hlavný cyklus ==================
void loop() {


  // ------------- Čítanie           -------------
                   DHT20
  if (millis() - DHT.lastRead() >= 1000) {
      int status = DHT.read();
      if (status == DHT20_OK) {
          teplota = DHT.getTemperature(); vlhkosť =
          DHT.getHumidity();
      }
  }



  // ----------- Načítanie intenzity osvetlenia. ---------
  -
  if (lightMeter.measurementReady(true)) { lux =
      lightMeter.readLightLevel();
  }


  // ------------- LCD displej    -------------
  lcd.setCursor(0, 0);
  lcd.print("T:");
  lcd.print(temperature, 1);
  lcd.print("C ");
  lcd.print("H:");
  lcd.print(vlhkosť, 0);
  lcd.print("% ");



  lcd.setCursor(0, 1);
  if (lux < DARK_LUX) {
      lcd.print("Príliš                    ");
  } inak{vo
```

```
        lcd.print("Lux:");
        lcd.print(lux, 0);
        lcd.print("                    ");
    }

    // ---------- Teplotný alarm ----------
    if (temperature > TEMP_ALARM) {
        digitalWrite(BUZ, HIGH); delay(100);
        digitalWrite(BUZ, LOW);
    }



    // ---------- Výstup ladenia sériového portu ----------
    Serial.print("Temp: ");
    Serial.print(temperature);
    Serial.print(" C | Hum: ");
    Serial.print(humidity); Serial.print(" %
    | Lux: "); Serial.println(lux);



    delay(500);
}
```

Celá táto funkcia **loop**() môže byť považovaná za hlavnú logiku slučky, keď systém prejde do „pracovného stavu“. Po zapnutí Arduina a vykonaní funkcie **setup**() sa tento kód bude nepretržite a opakovane vykonávať od začiatku do konca.

① Časť na zisťovanie teploty a vlhkosti DHT20:

```
52  // ================== major cycle ==================
53  void loop() {
54
55    // ----------- Read DHT20 -----------
56    if (millis() - DHT.lastRead() >= 1000) {
57      int status = DHT.read();
58      if (status == DHT20_OK) {
59        temperature = DHT.getTemperature();
60        humidity = DHT.getHumidity();
61      }
62    }
63
```

Tu sa namiesto priameho delay() používa neblokujúce časové riadenie založené na millis(). **DHT.lastRead()** zaznamenáva čas posledného úspešného odčítania senzora. **millis**() je časovač v milisekundách od zapnutia systému. Odpočítaním týchto dvoch hodnôt je možné určiť, či uplynula aspoň 1 sekunda. Výhody tohto prístupu sú:

- Vyhnite sa častému odčítavaniu údajov zo senzora (senzor DHT20 má obmedzenie frekvencie odčítavania)
- Nezmrazuje celý program a je to veľmi štandardný a profesionálny spôsob písania v embedded systémoch

Keď je splnená časová podmienka, volá sa nasledujúci kód:

```
52  // ================== major cycle ==================
53  void loop() {
54
55    // ----------- Read DHT20 -----------
56    if (millis() - DHT.lastRead() >= 1000) {
57      int status = DHT.read();
58      if (status == DHT20_OK) {
59        temperature = DHT.getTemperature();
60        humidity = DHT.getHumidity();
61      }
62    }
63
```

Tento krok slúži na skutočné odoslanie príkazu na získanie údajov do DHT20. Vrátená hodnota „status“ sa používa na určenie, či bola komunikácia úspešná. Iba ak:

```
52  // ================== major cycle ==================
53  void loop() {
54
55    // ----------- Read DHT20 -----------
56    if (millis() - DHT.lastRead() >= 1000) {
57      int status = DHT.read();
58      if (status == DHT20_OK) {
59        temperature = DHT.getTemperature();
60        humidity = DHT.getHumidity();
61      }
62    }
63
```

V čase inicializácie boli načítané hodnoty teploty a vlhkosti.

② Zaznamenávanie intenzity osvetlenia BH1750

```
// ----------- Read the light intensity. -----------
if (lightMeter.measurementReady(true)) {
  lux = lightMeter.readLightLevel();
}
```

Toto ilustruje pracovný mechanizmus režimu nepretržitého merania **BH1750.** Funkcia **measurementReady**(true) sa používa na určenie, či sú v danom okamihu k dispozícii nové údaje o osvetlení na načítanie. Ak áno, vyvolá sa funkcia **readLightLevel**(). Tým sa zabráni načítaniu starých alebo neplatných údajov a zároveň sa zníži zaťaženie zbernice I2C. Ide o efektívnu, energeticky úspornú a rozumnú metódu čítania senzora.

③ Logika LCD displeja

```
// ----------- LCD display -----------
lcd.setCursor(0, 0);
lcd.print("T:");
lcd.print(temperature, 1);
lcd.print("C ");
lcd.print("H:");
lcd.print(humidity, 0);
lcd.print("% ");
```

Umiestnite kurzor do prvého riadku a prvého stĺpca a potom postupne vypíšte: "T:" +

Teplota (zaokrúhlená na 1 desatinné miesto) + "C"

„H:“ + vlhkosť (zaokrúhlená na celé číslo) + „%“

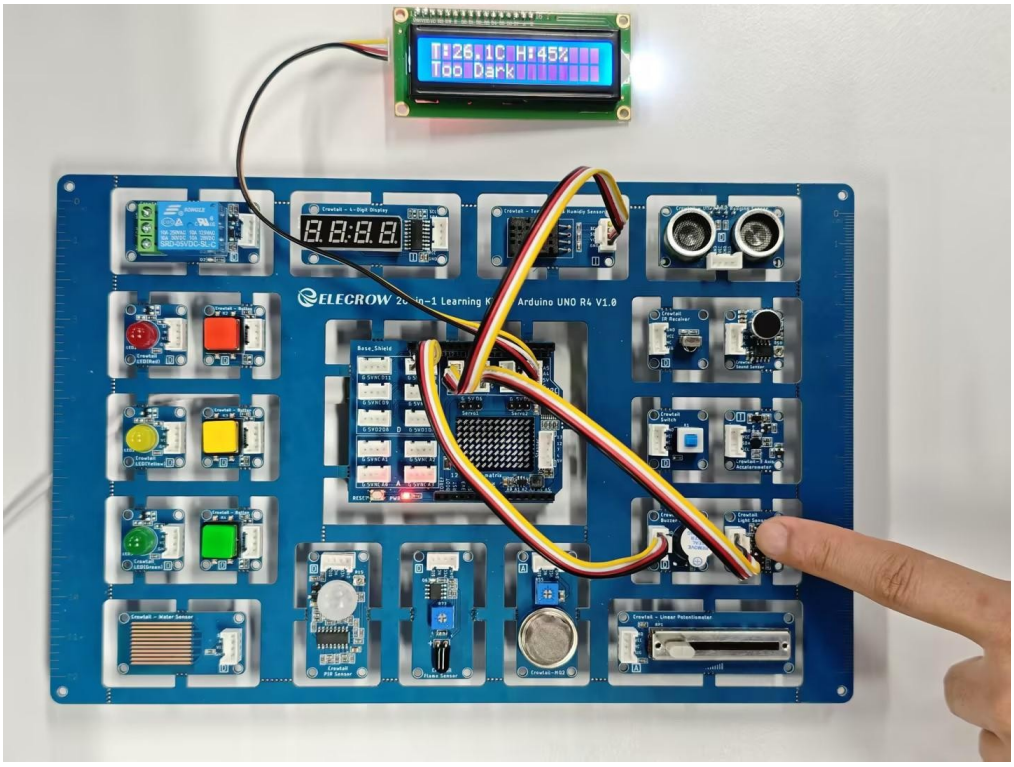
Tento formát zobrazenia robí informácie kompaktnými a intuitívnymi a je veľmi vhodný pre 1602 znakový displej.

V druhom riadku sa zobrazuje stav osvetlenia:

```
lcd.setCursor(0, 1);
if (lux < DARK_LUX) {
  lcd.print("Too Dark       ");
} else {
  lcd.print("Lux:");
  lcd.print(lux, 0);
  lcd.print("          ");
}
```

Keď je intenzita osvetlenia nižšia ako prahová hodnota **DARK_LUX (50 luxov)**, priamo sa zobrazí **„Too Dark“ (Príliš tmavo**). Ide o „displej indikujúci stav“, ktorý viac zodpovedá skutočnému zážitku z používania produktu v porovnaní s jednoduchým zobrazením čísla.

V opačnom prípade: Zobrazí sa konkrétna hodnota intenzity osvetlenia. Pridaním medzery za ňou sa zakryjú staré znaky a zabráni sa duchom, čo je veľmi dôležitý malý detail pri používaní znakových LCD displejov.



④ **Logika teplotného alarmu**

```
87  // ----------- Temperature alarm -----------
88  if (temperature > TEMP_ALARM) {
89    digitalWrite(BUZ, HIGH);
90    delay(100);
91    digitalWrite(BUZ, LOW);
92  }
```

Keď teplota prekročí nastavenú prahovú hodnotu **TEMP_ALARM** (30 °C), bzučiak vydá krátke pípnutie.

Toto riešenie využíva pulzný alarm namiesto nepretržitého zvuku, čím upozorní používateľa bez toho, aby pôsobilo príliš rušivo.

⑤ **Výstup ladenia sériového portu**

```
94  // ----------- Serial port debugging output -----------
95  Serial.print("Temp: ");
96  Serial.print(temperature);
97  Serial.print(" C | Hum: ");
98  Serial.print(humidity);
99  Serial.print(" % | Lux: ");
100 Serial.println(lux);
```

Táto časť nemá vplyv na funkčnosť systému, ale je mimoriadne dôležitá pre vývojárov. Umožňuje zobrazenie údajov zo senzorov v reálnom čase v monitore sériového portu, čo možno využiť na: Overenie, či senzor funguje správne

Určenie, či sú nastavenia prahových hodnôt primerané Identifikáciu

abnormalít zobrazenia alebo alarmu

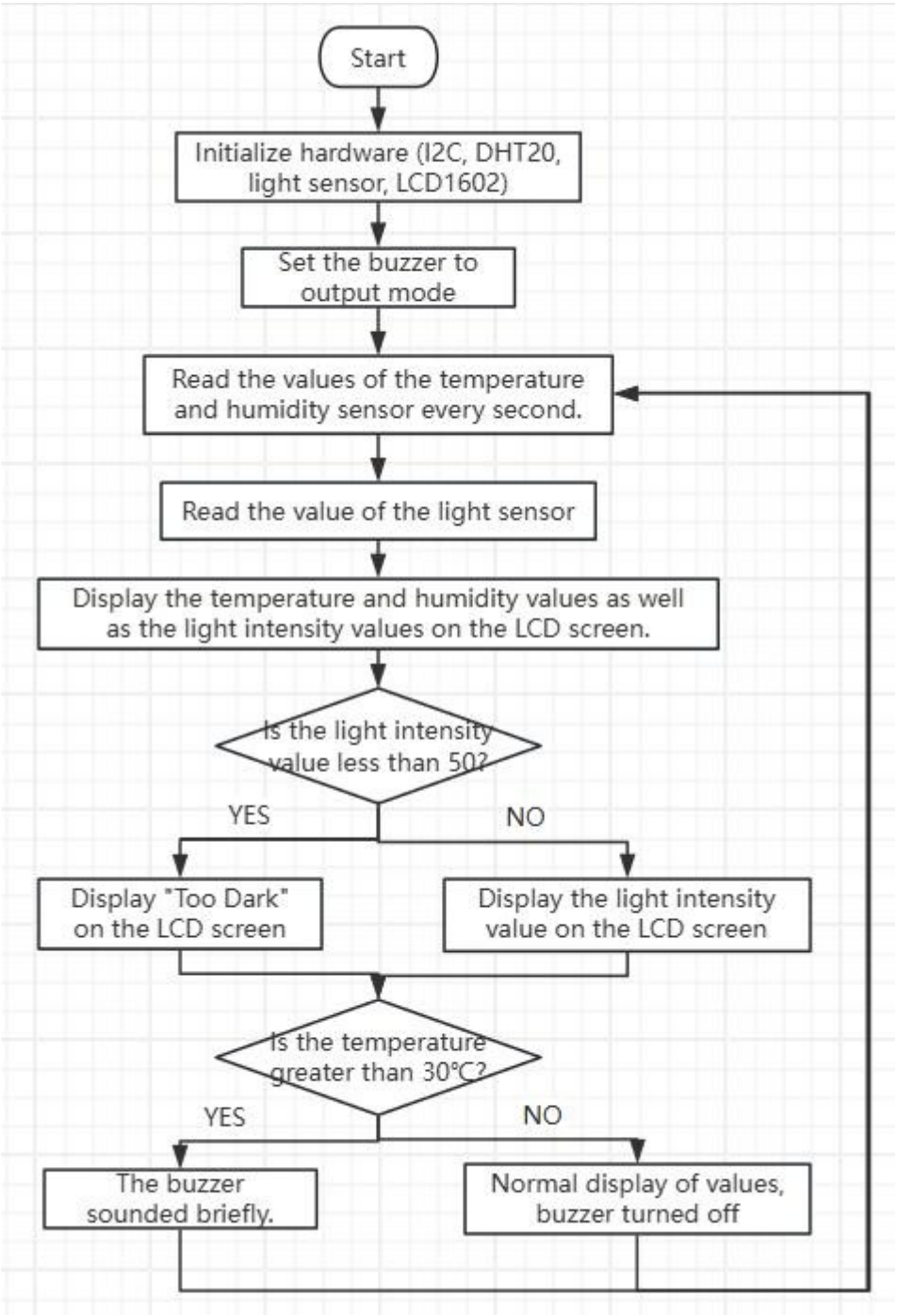
⑥ **Oneskorenie**

```
101
102     delay(500);
103   }
104
```

Pridajte do hlavnej slučky riadenie rytmu 500 ms, aby ste predišli nasledujúcim problémom spôsobeným rýchlym obnovovaním:

- Blikanie LCD displeja
- Sériová aktualizácia obrazovky
- Zvýšená spotreba energie systému

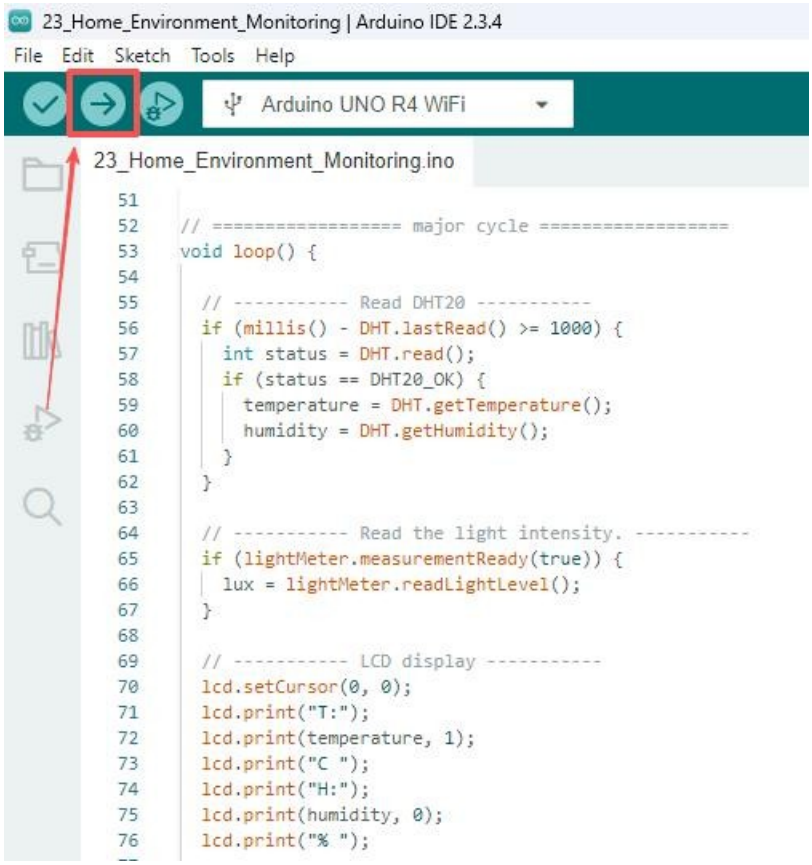
(7) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

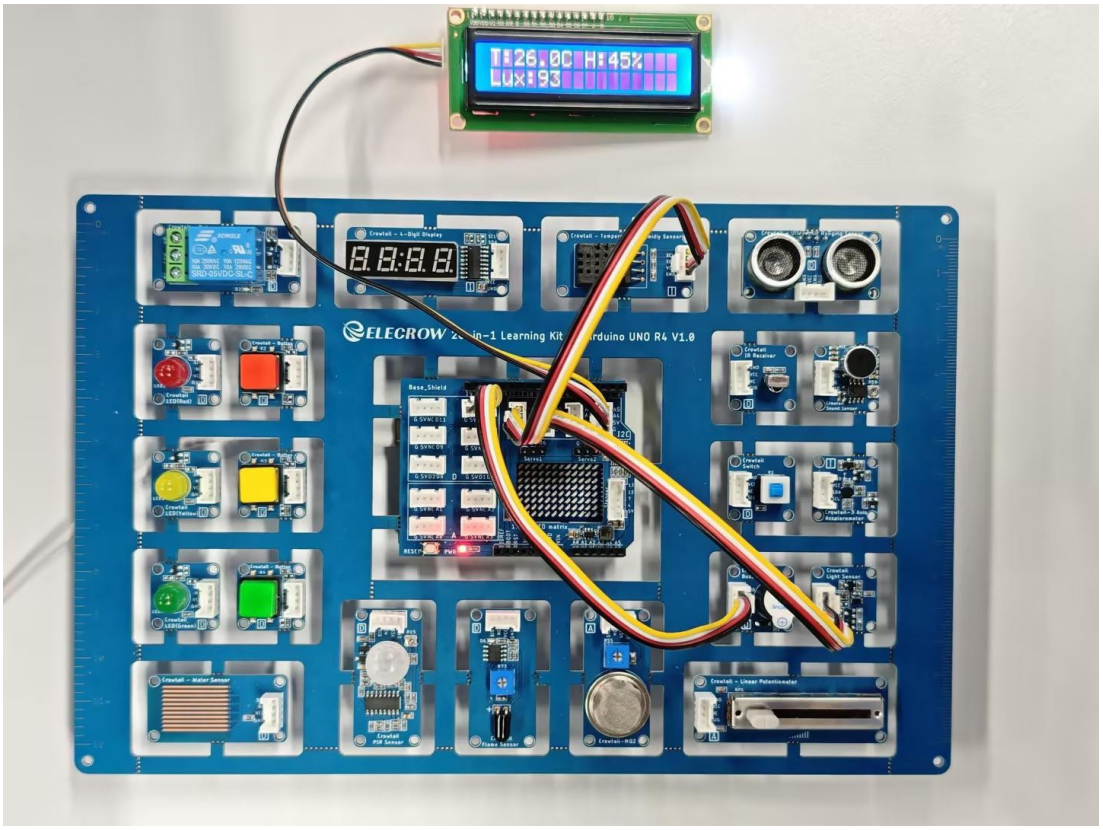
(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

Kliknite na **„Stiahnuť“:**



(2) Uvidíte:

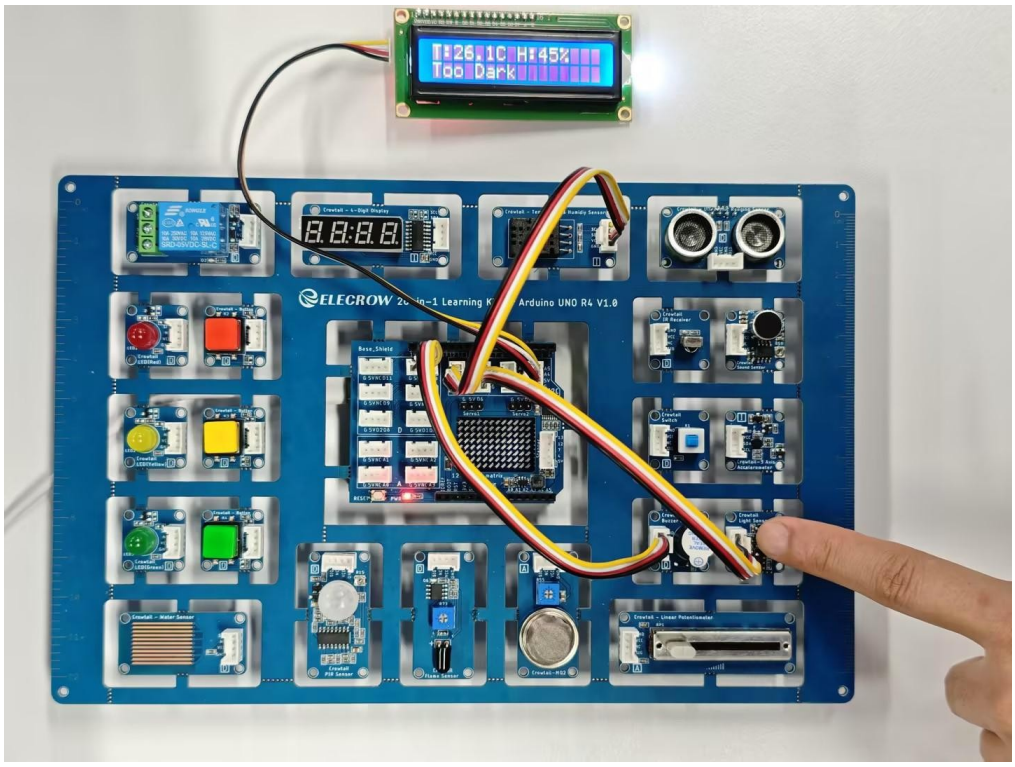
Prvý riadok LCD displeja zobrazuje v reálnom čase: aktuálnu **teplotu** (°C) a aktuálnu **vlhkosť** (%).



Druhý riadok LCD displeja sa dynamicky mení v závislosti od svetelných podmienok:

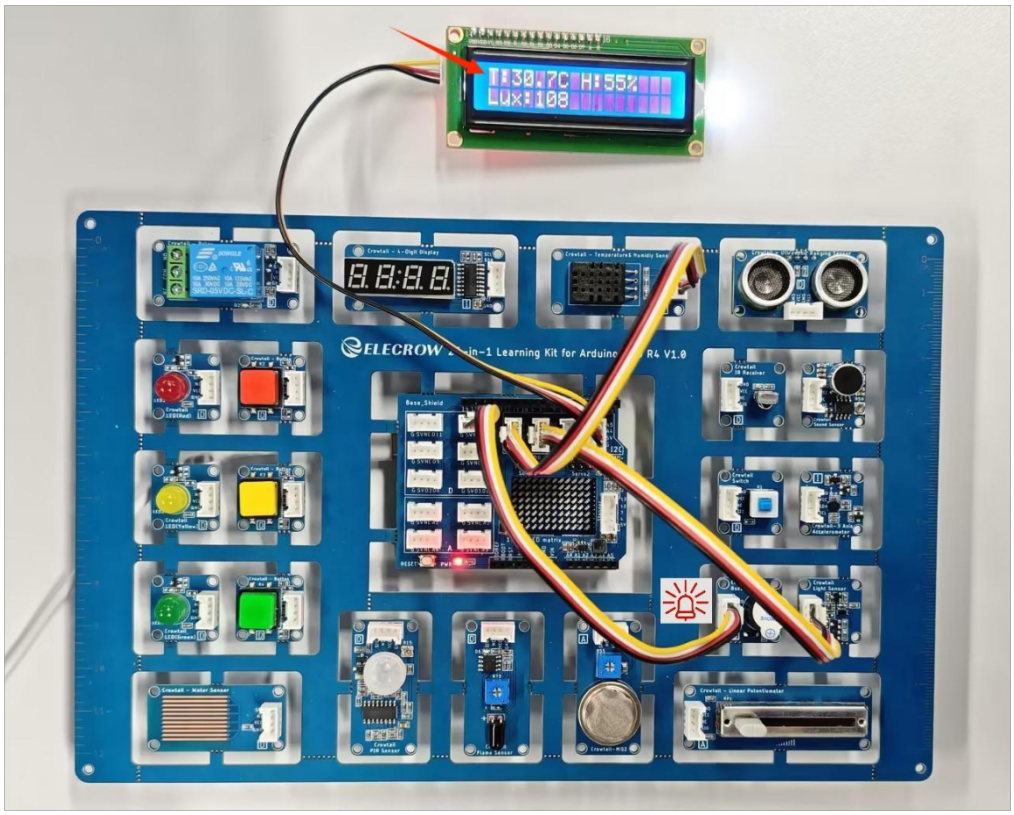
Normálne osvetlenie → Zobrazuje aktuálnu **hodnotu v luxoch**

Príliš málo svetla → Zobrazuje **„Príliš tmavo“**



Keď je teplota vyššia ako $30\ ^{\circ}\ \mathrm{C}$:

Bzučiak vydá krátky varovný signál, aby upozornil na prehriatie.



Lekcia 24 — Hra na hádanie hodnôt

Úvod

V tejto lekcii použijeme lineárny potenciometer a displej LCD1602 na vytvorenie hry „Hádanie čísla“. Systém náhodne vygeneruje cieľovú hodnotu v rozmedzí od 0 do 1023 a zobrazí ju na LCD displeji. Hráč musí v časovom limite otáčať potenciometrom tak, aby sa aktuálna hodnota čo najviac priblížila cieľovej hodnote.

Ak je hodnota potenciometra dostatočne blízko cieľovej hodnote, systém zaznamená úspech a zvýši skóre. Ak sa hodnota príliš líši, zaznamená neúspech.

Celá hra poskytuje spätnú väzbu výlučne prostredníctvom LCD displeja, čím sa stáva klasickou interaktívnou minihrou založenou na displeji.

Ciele vzdelávania

1. Porozumieť rozsahu hodnôt analógových vstupov (analogRead).
2. Naučiť sa používať lineárny potenciometer na získavanie neustále sa meniacich údajov.
3. Zvládnuť základné použitie funkcií random() a randomSeed().
4. Naučiť sa dynamicky zobrazovať údaje a správy na LCD displeji.
5. Používať podmienené príkazy na porovnávanie hodnôt z hľadiska ich blízkosti.
6. Navrhnite jednoduchý systém bodovania na dokončenie plne interaktívnej minihry.

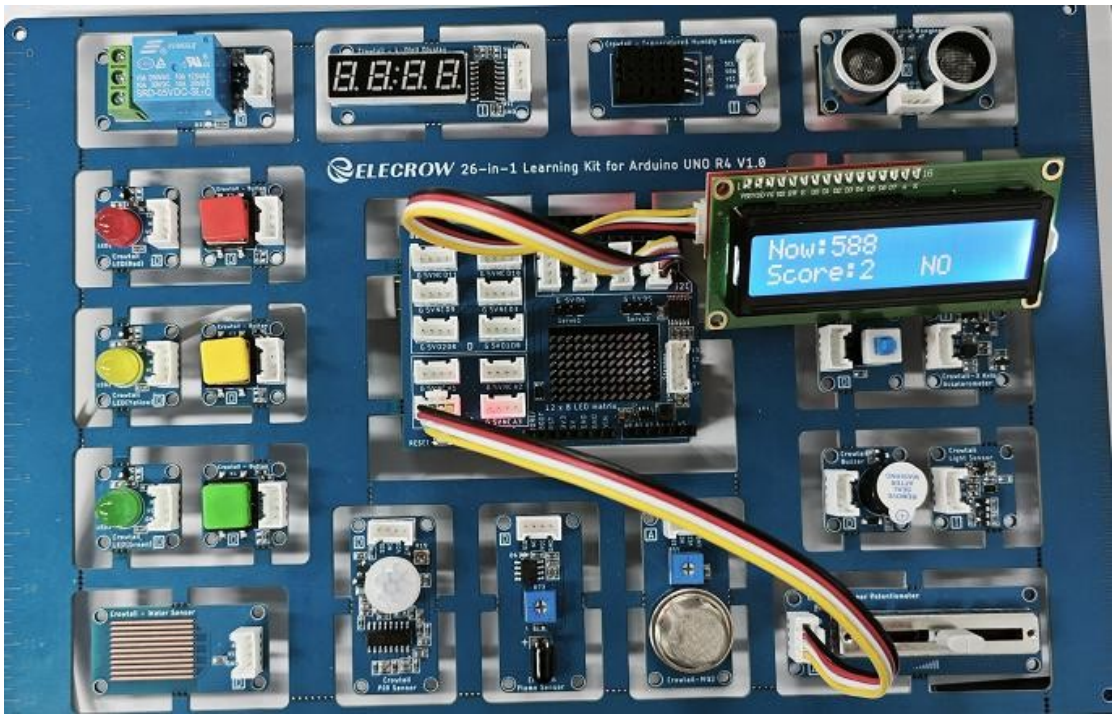
Náhľad výsledku

Pri spustení sa na LCD displeji zobrazí názov hry. Na začiatku každého kola sa na LCD displeji zobrazí cieľová hodnota. Hráč má 3 sekundy na nastavenie potenciometra.

Systém zčíta aktuálnu hodnotu potenciometra (Now). Pravidlá bodovania:

Ak je rozdiel medzi Now a Target ≤ 10 → zobrazí sa OK, pripočíta sa 1 bod. Ak je rozdiel > 10 → zobrazí sa NO.

Hra automaticky prejde do ďalšieho kola a cyklus sa neustále opakuje.



1. Vysvetlenie princípu

Základnou koncepciou tejto hry je „kontrola hodnoty na základe blízkosti“.

Systém zadá cieľovú hodnotu a hráč otáča potenciometrom, aby nastavil aktuálnu hodnotu. Program vypočíta rozdiel medzi týmito dvoma hodnotami a na základe toho určí, či bol pokus úspešný. A:Lineárny potenciometer mení svoje výstupné napätie pri otáčaní.

B: Arduino číta toto napätie pomocou funkcie analogRead() a prevádza ho na celé číslo v rozmedzí od 0 do 1023. C: Program generuje náhodnú cieľovú hodnotu v tom istom rozsahu pomocou funkcie random(0, 1024). D: Aby sa zabezpečilo, že každé kolo má inú cieľovú hodnotu, funkcia randomSeed(analogRead(A1)) inicializuje náhodné semeno.

Po tom, čo hráč nastaví potenciometer, program:

A:Načíta aktuálnu hodnotu.

B:Vypočíta rozdiel s cieľovou hodnotou pomocou abs(currentValue - targetValue).

C:Ak je rozdiel v povolenom rozsahu, počíta sa to ako úspech a pridajú sa body; inak ide o neúspech.

Výsledok a aktuálny stav sa zobrazujú v reálnom čase na LCD displeji, čím sa uzatvára kompletný interaktívny spätnoväzobný cyklus.

2. Požadované moduly

Crowtail -- Lineárny potenciometer × 1

Crowtail - LCD 1602 × 1

3. Spôsob zapojenia

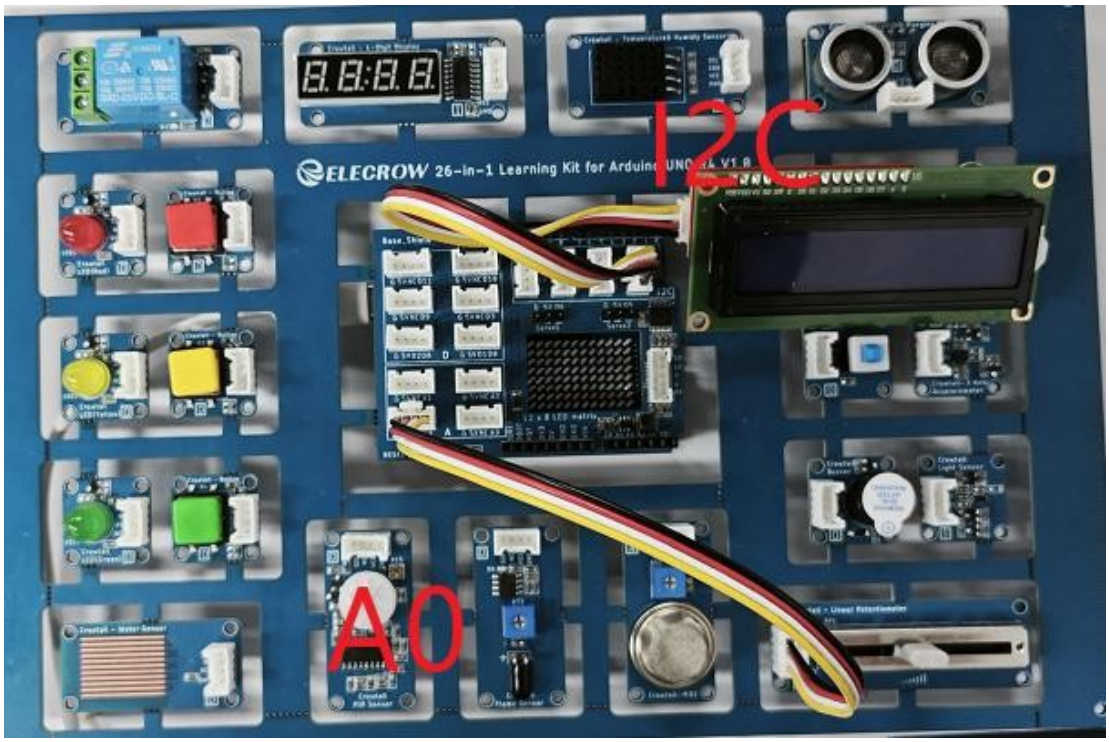
Crowtail -- Lineárny potenciometer → Analógový port A0

Crowtail -- LCD 1602 → port I2C

Špecifikácia štvorportového rozhrania Crowtail:

- GND (čierny) → GND
- VCC (červená) → 5 V
- SDA (biela) → Bez pripojenia
- SCL (žltý) → A0/I2C k 5V

napájaniu.



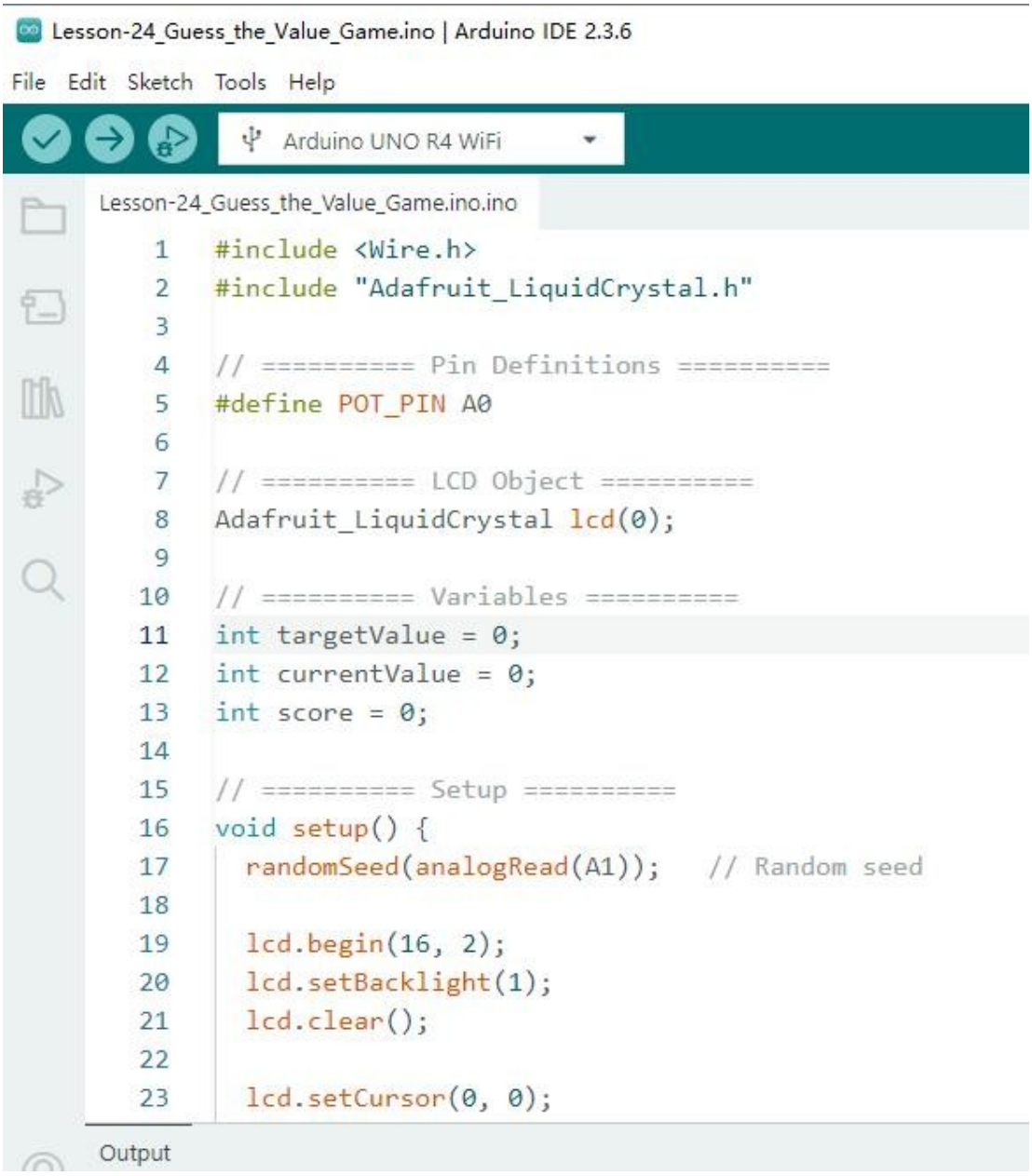
4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-24_Guess_the_Value_Game.ino

Spustite program pomocou Arduino IDE.



Vysvetlenie kódov klávesov

(1) Definície pinov a objektov.

```
#define POT_PIN A0
```

POT_PIN: Analógový vstupný pin, ku ktorému je pripojený potenciometer. Číta hodnoty od 0 do 1023.

```
Adafruit_LiquidCrystal lcd(0);
```

Vytvorí objekt LCD na zobrazenie všetkých informácií o hre.

(2) Parametre hry a základné premenné.

```
int targetValue = 0;
int aktuálnaHodnota =
0; int skóre = 0;
```

targetValue: Náhodne vygenerovaná cieľová hodnota.

currentValue: Aktuálna hodnota z potenciometra. score:

Kumulované skóre hráča.

Tieto premenné tvoria údaje o stave hry.

(3) Nastavenie inicializácie.

```
randomSeed(analogRead(A1));
```

Používa nepripojený analógový pin ako zdroj náhodného počiatočného čísla, čím zabezpečuje odlišné cieľové hodnoty v každom kole a zvyšuje náhodnosť hry.

randomSeed() poskytuje generátoru náhodných čísel počiatočné semeno. Rôzne semená produkujú rôzne sekvencie náhodných čísel.

```
  lcd.begin(16, 2);
  lcd.setBacklight(1);
```

Inicializuje LCD a zapne podsvietenie, ktoré slúži ako zobrazovacie rozhranie hry.

(4) Hlavná slučka.

①Generovanie cieľovej hodnoty

```
targetValue = random(0, 1024);
```

Vygeneruje náhodné číslo v rozmedzí 0 — 1023, čo zodpovedá celému rozsahu potenciometra. Funkcia random() zabezpečuje, že každé kolo má jedinečný cieľ, čím sa zvyšuje náhodnosť a hrateľnosť hry.

②Výzva hráčovi na nastavenie

```
lcd.print("Nastavte do 3 sekúnd");
delay(3000);
```

Dáva hráčovi 3 sekundy na nastavenie potenciometra, čím sa zlepšuje tempo hry.

③Prečítajte aktuálnu hodnotu

```
currentValue = analogRead(POT_PIN);
```

Načíta polohu potenciometra a prevádza ju na číselnú hodnotu.

④Zobrazenie aktuálnej hodnoty a skóre

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Teraz:");
lcd.print(currentValue);


lcd.setCursor(0,     1);
lcd.print("Skóre:");
lcd.print(skóre);
```

⑤Porovnajte hodnoty a zobrazte spätnú väzbu

```
if (abs(currentValue - targetValue) <= 10) { score++;
        lcd.setCursor(10, 1);
        lcd.print("OK");
} inak {
        lcd.setCursor(10, 1);
        lcd.print("NIE");
}
```

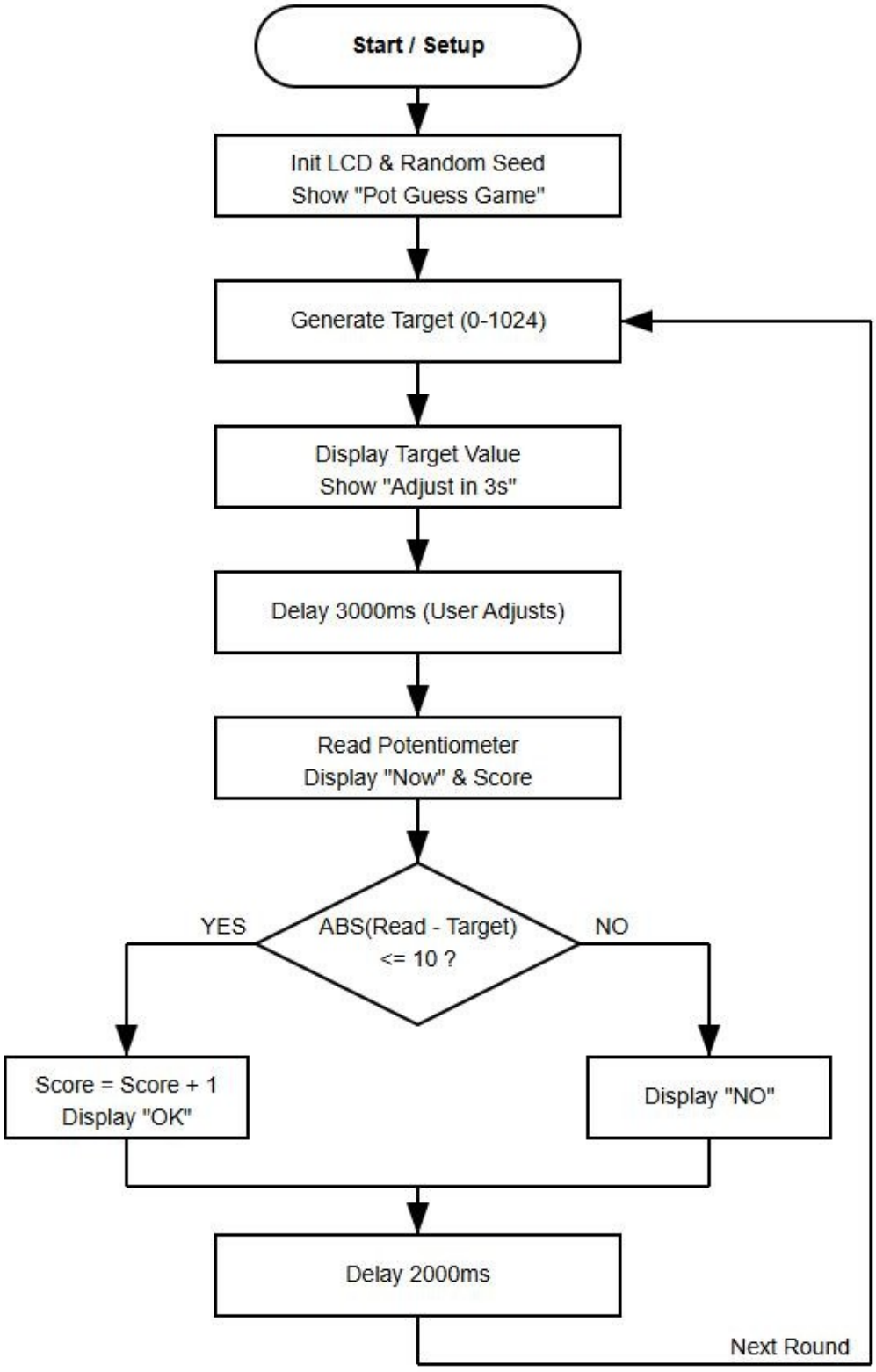
abs() vypočíta absolútny rozdiel medzi currentValue a targetValue.

Ak je rozdiel v povolenom rozsahu (≤10), je to úspech: skóre sa zvýši o 1 a zobrazí sa „OK“.

Ak je rozdiel väčší, ide o chybu: na displeji sa zobrazí „NO“.

Všetka spätná väzba sa zobrazuje na LCD displeji, čím sa zabezpečujú jasné a intuitívne výsledky.

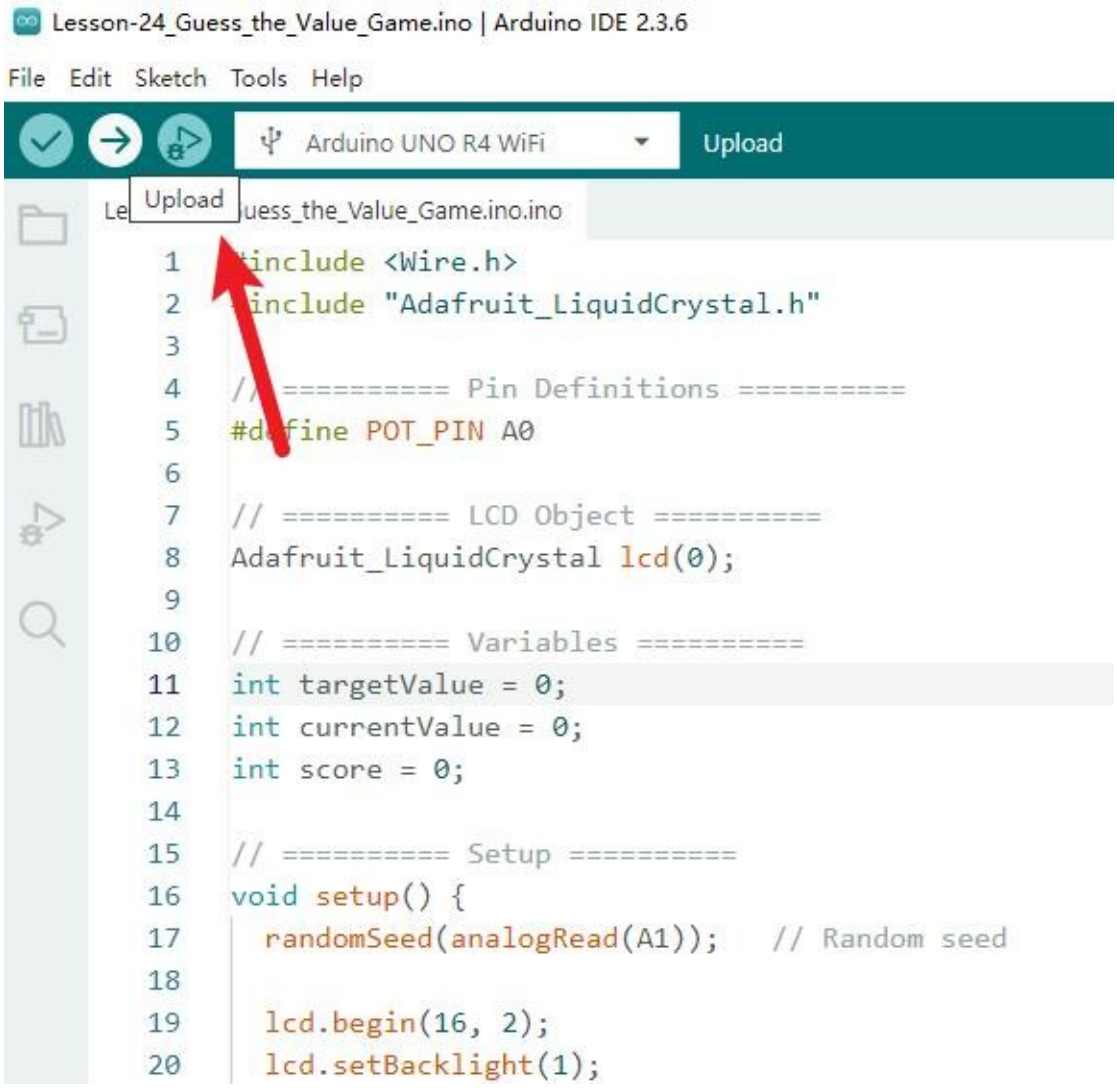
(5) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov obsluhy Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

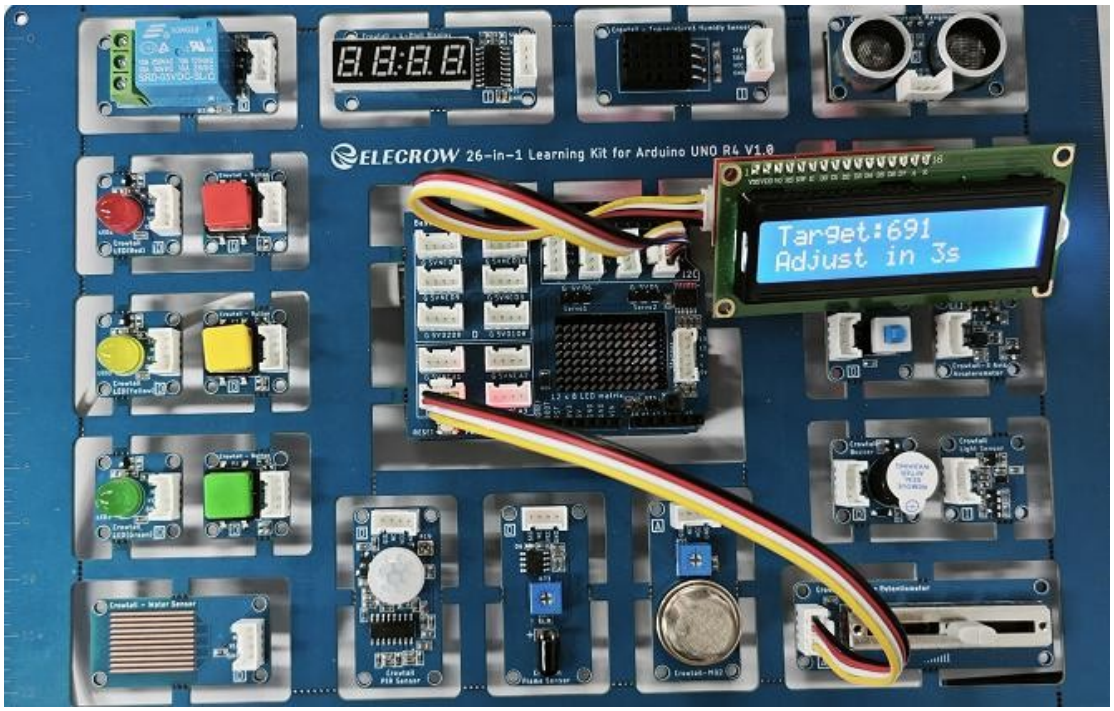
Kliknite na „Stiahnuť“:



(2) Program sa spustí.

Program beží:

Po spustení systému sa na LCD displeji zobrazí názov hry. Následne sa na LCD displeji zobrazí cieľová hodnota a hráč má 3 sekundy na nastavenie potenciometra.



Program vyhodnotí hodnotu potenciometra, uplatní príslušné pravidlá bodovania a hra automaticky prejde do ďalšieho kola, pričom sa tento proces neustále opakuje.

Lekcia 25 — Pamäťová hra s blikajúcou LED

Úvod

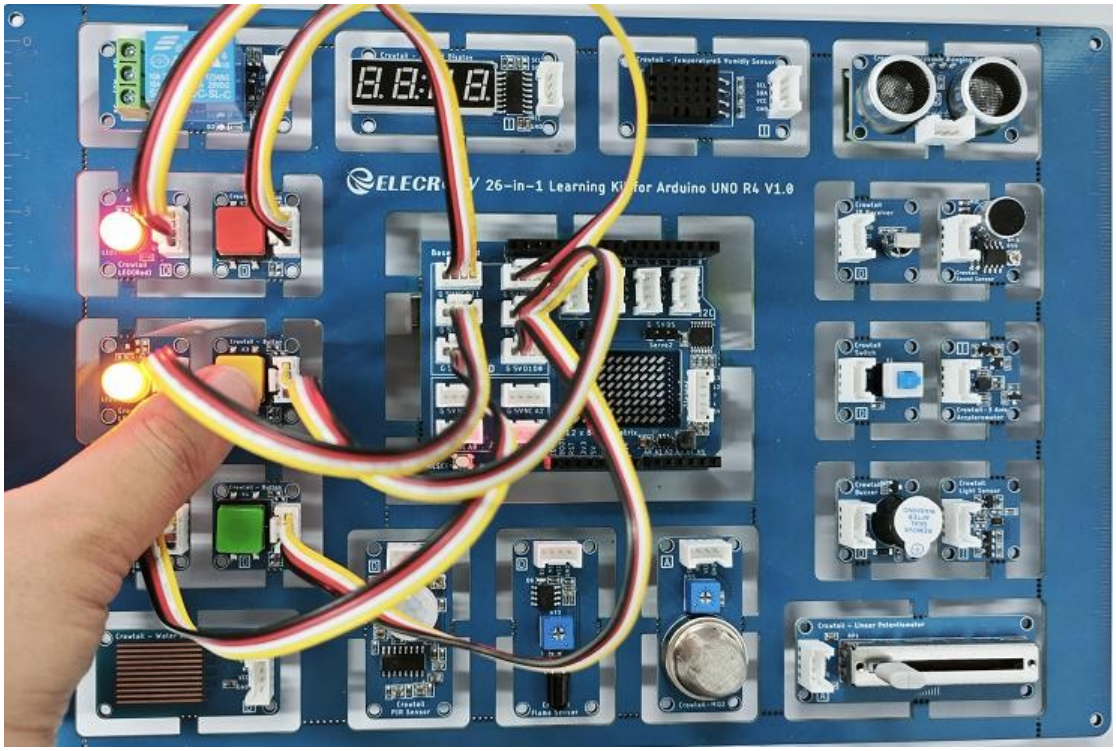
V tejto lekcii použijeme LED diódy a tlačidlové moduly na vytvorenie pamäťovej hry založenej na LED diódach s názvom „Sequence Memory“. Systém rozsvieti LED diódy v prednastavenom poradí a hráč si musí zapamätať túto sekvenciu a reprodukovať ju stlačením tlačidiel v rovnakom poradí.
Tento projekt presahuje jednoduchý koncept „stlač raz, rozsvieti sa“. Zavádza pamäť sekvencií a logické uvažovanie, čo predstavuje dôležitý krok od základného vstupu/výstupu smerom k návrhu minihier založených na logike.

Ciele vzdelávania

1. Naučte sa, ako v programe znázorniť postupnosť.
2. Naučte sa používať LED diódy ako vizuálne signály.
3. Naučte sa používať tlačidlá na sekvenčné zadávanie údajov.
4. Ovládnuť používanie polí na ukladanie sekvenčných údajov.
5. Naučte sa porovnávať výsledky vstupov pomocou podmienečnej logiky.
6. Dokončite kompletnú pamäťovú hru založenú na LED diódach.

Náhľad výsledku

Keď sa systém spustí, LED diódy budú blikať v pevne danom poradí. Po skončení sekvencie hra prejde do fázy zadávania údajov hráčom. Hráč stláča tlačidlá, aby zopakoval sekvenciu LED diód:
Správny postup: Všetky LED diódy svietia súčasne po dobu 1 sekundy.
Nesprávny postup: Všetky LED diódy blikajú s frekvenciou približne 5 Hz po dobu
2 sekúnd.



1. Vysvetlenie princípu

Základnou myšlienkou hry LED Memory Game je „porovnávanie sekvencií“. Program najprv zobrazí sekvenciu blikania LED diód, ktoré sa rozsvietia jedna po druhej, aby ukázali poradie, ktoré si hráč musí zapamätať. Akonáhle všetky LED diódy zablikajú, hráč sa pokúsi replikovať sekvenciu stlačením príslušných tlačidiel v rovnakom poradí.

Počas fázy zadávania program krok za krokom porovnáva každé stlačenie tlačidla s vopred nastavenou sekvenciou LED. Ak sa niektorý krok nezhoduje, hra okamžite zaznamená neúspech. Hra sa považuje za úspešnú len vtedy, ak hráč správne zopakuje celú sekvenciu. V podstate tento proces zahŕňa uloženie sekvencie do poľa a použitie príkazov if na vykonanie porovnania.

2. Požadované moduly

Crowtail -- Tlačidlo × 3

Crowtail – LED × 3

3. Spôsob zapojenia

Crowtail – tlačidlo → DIGITAL D11

Crowtail – tlačidlo → DIGITAL D9

Crowtail – tlačidlo → DIGITAL D8

Crowtail – LED → DIGITAL D10

Crowtail – LED → DIGITAL D3

Crowtail – LED → DIGITAL D0

Špecifikácia rozhrania Crowtail so štyrmi portmi:

- GND (čierna) → GND
- VCC (červená) → 5V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → D11/D9/D8/D10/D3/D0

k napájaniu 5 V.

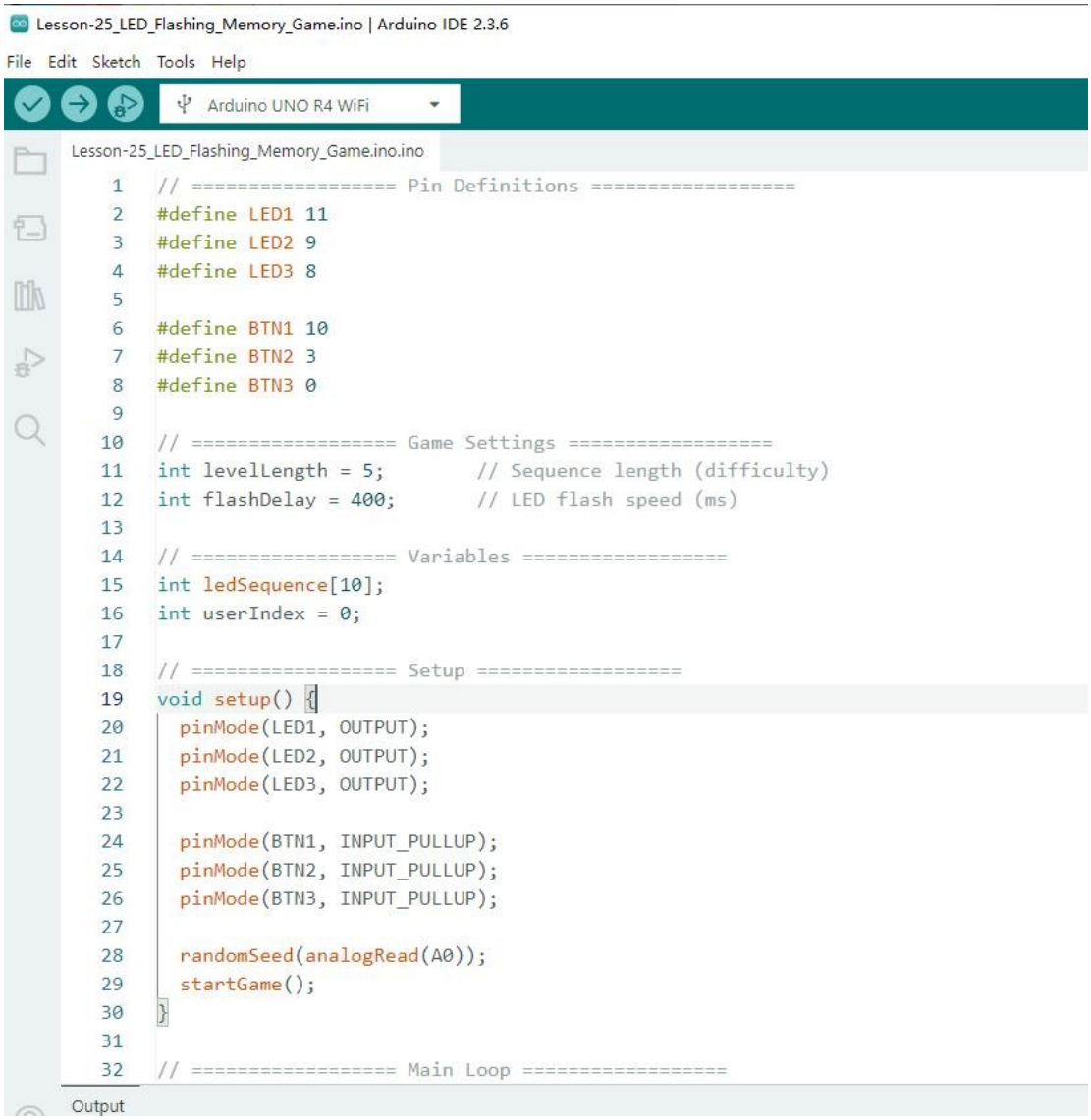
4. Vysvetlenie príkladu

Kliknutím na nižšie uvedený odkaz si stiahnite oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-25_LED_Flashing_Memory_Game.ino

Otvorte program pomocou Arduino IDE.



Vysvetlenie kľúčového kódu

(1) Definície objektov tlačidiel a LED.

```
// ================== Definície pinov ==================
#define LED1 11
#define LED2 9
#define LED3 8
#define BTN1 10
#define BTN2 3
#define BTN3 0
```

LED1 ~ LED3: Tri LED diódy slúžiace na zobrazenie blikajúcej sekvencie.

BTN1 ~ BTN3: Tri tlačidlá, ktoré hráč používa na zadávanie zapamätanej sekvencie. Použitie #define na pomenovanie pinov zvyšuje čitateľnosť a udržateľnosť kódu. Pozorovanie LED1

okamžite vám napovie, že ide o prvú LED diódu, bez toho, aby ste si museli pamätať, že jej zodpovedá pin 11.

(2) Parametre hry a základné premenné.

```
// ================== Nastavenia hry ==================
int levelLength = 5;              // Dĺžka sekvencie (obťažnosť)
int flashDelay = 400;             // Dĺžka blikania LED (ms)
int ledSequence[10];
int userIndex = 0;
```

levelLength: Dĺžka sekvencie LED, určuje obtiažnosť hry. flashDelay: Doba, počas ktorej svieti každá LED, ovláda rýchlosť blikania.

Úpravou týchto hodnôt môžete rýchlo vytvoriť „ľahkú“ alebo „ťažkú“ verziu.

ledSequence[]: Pole ukladajúce systémom generovanú LED sekvenciu. userIndex: Sleduje aktuálny krok, do ktorého hráč vstúpil.

Pole fungujú ako pamäťový kontajner a sú kľúčové pre túto hru. Predstavte si pole ako rad očíslovaných políčok, z ktorých každé obsahuje jeden údaj. V tomto experimente:

Je tu 10 políčok na uloženie čísel sekvencie LED.

Každé políčko obsahuje číslo (1, 2 alebo 3), ktoré predstavuje, ktorá LED svieti.

(3) Nastavenie inicializácie.

```
void setup() { pinMode(LED1,
    OUTPUT); pinMode(LED2,
    OUTPUT); pinMode(LED3,
    OUTPUT);
    pinMode(BTN1,       INPUT_PULLUP);
    pinMode(BTN2,       INPUT_PULLUP);
    pinMode(BTN3, INPUT_PULLUP);
    randomSeed(analogRead(A0));
    startGame();
}
```

Používa nepripojený analógový pin ako zdroj náhodného semena, čím sa zabezpečí odlišný cieľ

Nastavte LED diódy ako výstup a tlačidlá ako INPUT_PULLUP (HIGH, keď nie sú stlačené).

Použite randomSeed() na inicializáciu generátora náhodných čísel, aby sa sekvencia LED menila v každej hre.

Vyvolajte startGame(), aby sa hra spustila ihneď po zapnutí.

Funkcia randomSeed() poskytuje generátoru náhodných čísel počiatočné semeno. Využitím nepripojeného analógového pinu (A0) sa zachytáva šum z okolia, takže semeno sa pri každom spustení líši. Tým sa zaručuje nová sekvencia LED pre každú hru, čo zvyšuje náhodnosť a hrateľnosť.

(4) Hlavná slučka.

①Detekcia stlačenia tlačidla

```
int button = readButton();
```

Detekuje, ktoré tlačidlo hráč stlačil.

Vráti: 1 / 2 / 3 pre číslo LED, 0, ak nie je stlačené žiadne tlačidlo.

②Spätná väzba pre hráča

```
if (button != 0) {
    flashSingleLED(button);
}
```

Každé stlačenie tlačidla okamžite rozsvieti príslušnú LED diódu, čím poskytuje okamžitú spätnú väzbu a zabraňuje omylom pri stlačení.

③Porovnanie sekvencie (základná logika)

```
if (button == ledSequence[userIndex]) { userIndex++;
}
```

Porovnáva aktuálny vstup hráča so sekvenciou systému.

Správne → prejsť na ďalší krok; nesprávne → okamžité zlyhanie. To je podstata porovnávania sekvencií.

④Všetko správne

```
if (userIndex >= levelLength) {
    successLED();
    startGame();
}
```

Keď hráč správne dokončí celú sekvenciu, všetky LED diódy sa na 1 sekundu rozsvietia, čím signalizujú úspech, a potom sa automaticky spustí ďalšie kolo.

⑤Akákoľvek chyba

```
inak {
    failLED();
    startGame();
```

```
}
```

Akýkoľvek nesúlad okamžite spustí poruchu: LED diódy blikajú pri frekvencii 5 Hz po dobu 2 sekúnd, potom sa hra reštartuje.

(5) Funkcie generovania a zobrazenia sekvencie.

①Generovanie náhodnej sekvencie

```
void generateSequence() {
    for (int i = 0; i < levelLength; i++) { ledSequence[i] =
        random(1, 4);
    }
}
```

Používa random(1,4) na generovanie čísel 1 – 3, z ktorých každé reprezentuje jednu LED, uložených v poli ako správna sekvencia.

②Zobrazenie sekvencie LED

```
void showSequence() {
    for (int i = 0; i < levelLength; i++) {
        flashSingleLED(ledSequence[i]);
    }
}
```

Rozsvieti LED diódy v poradí podľa poľa; hráč si musí starostlivo zapamätať postupnosť.

(6) Efekty LED pri úspechu a neúspechu.

①Efekt úspechu

```
void successLED() {
    allLEDOn(); delay(1000);
    allLEDOff();
}
```

Všetky tri LED diódy svietia 1 sekundu, čím jasne signalizujú dokončenie postupu.

②Efekt zlyhania (blikanie s frekvenciou 5 Hz)

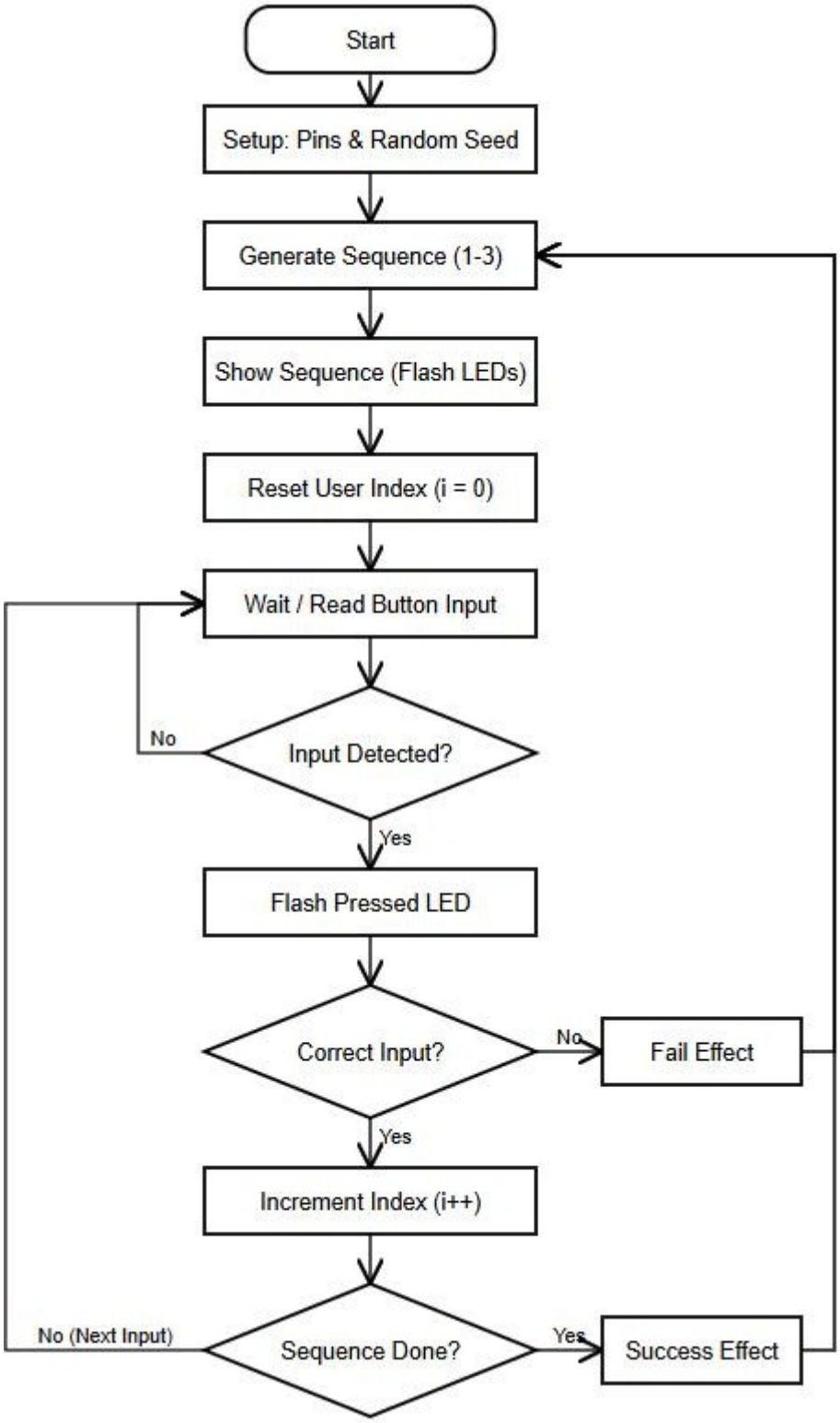
```
void failLED() {
    for (int i = 0; i < 10; i++) {
        allLEDOn(); delay(100);
```

```
        vypnúťVšetkyLEDy();
        počkať(100);
    }
}
```

100 ms zapnuté + 100 ms vypnuté = 1 cyklus 200 ms → 5 Hz.

5 Hz × 2 sekundy = 10 bliknutí, čo jasne signalizuje nesprávny vstup.

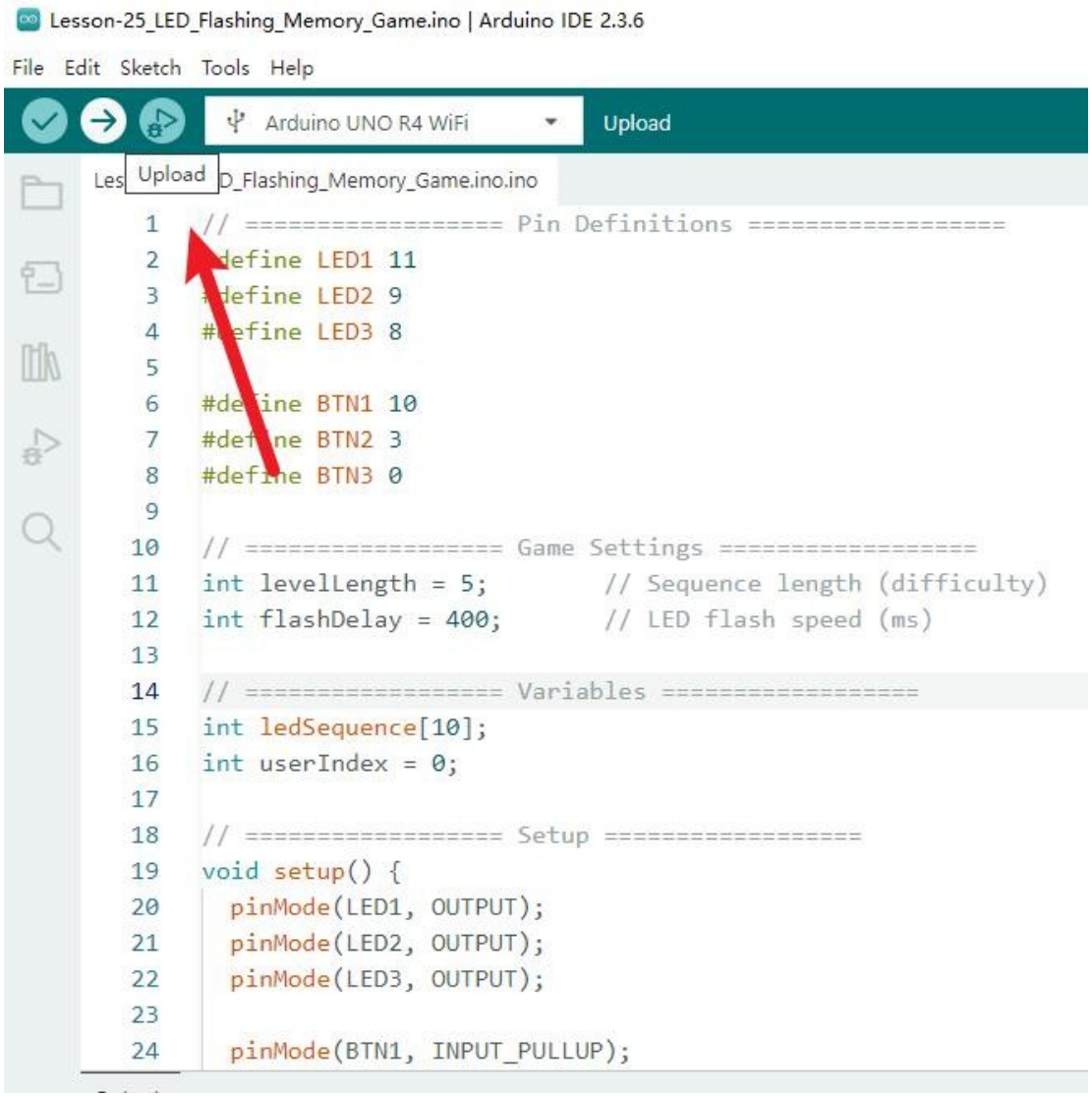
(7) Celkový diagram logiky kódu



5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu Arduino IDE uvedených na strane 8, správne pripojte počítač a dokončite základnú konfiguráciu nahrávania.

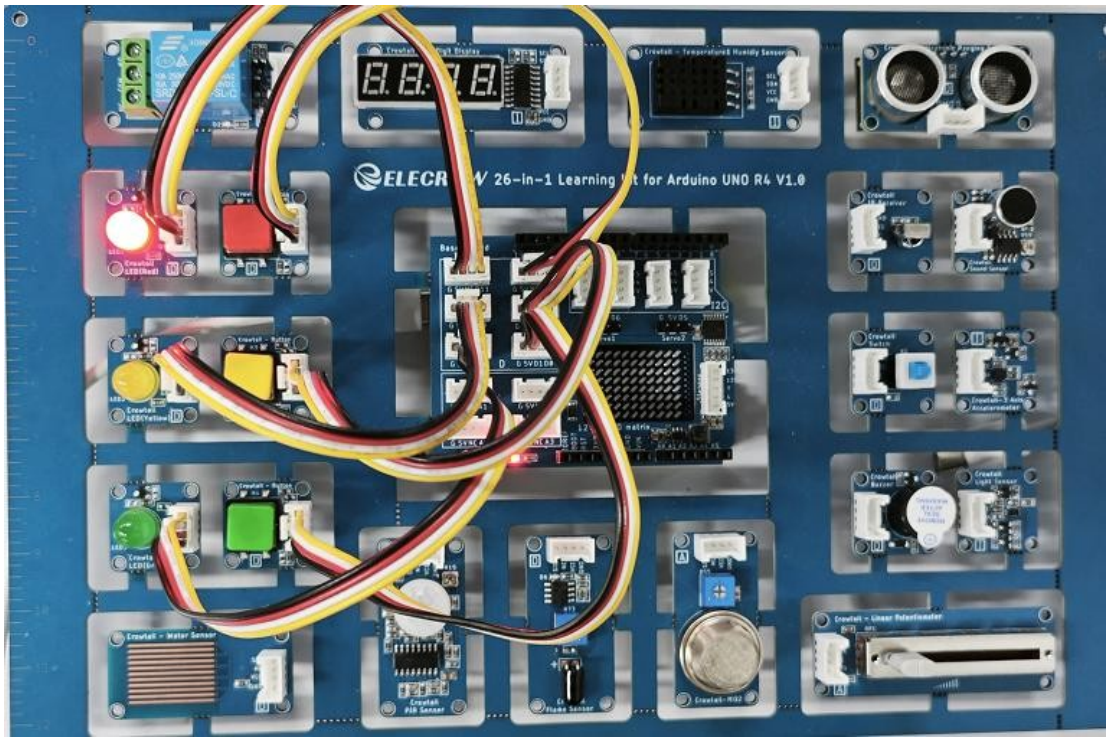
Kliknite na „Stiahnuť“:



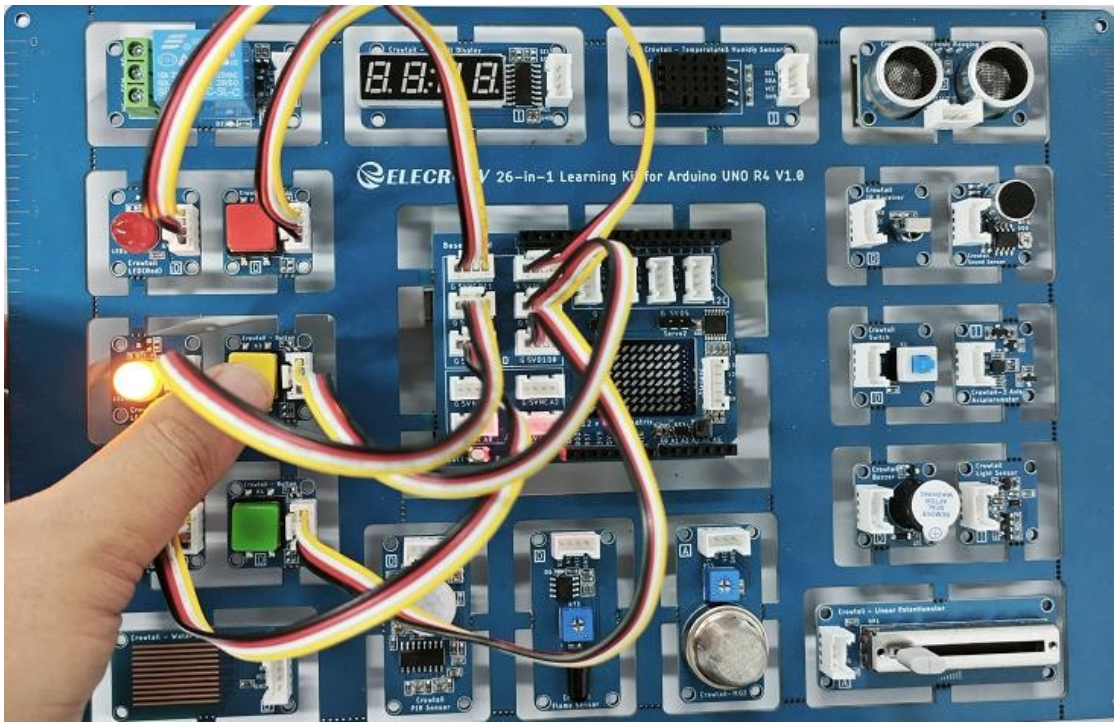
(2) Program sa spustí.

Program beží:

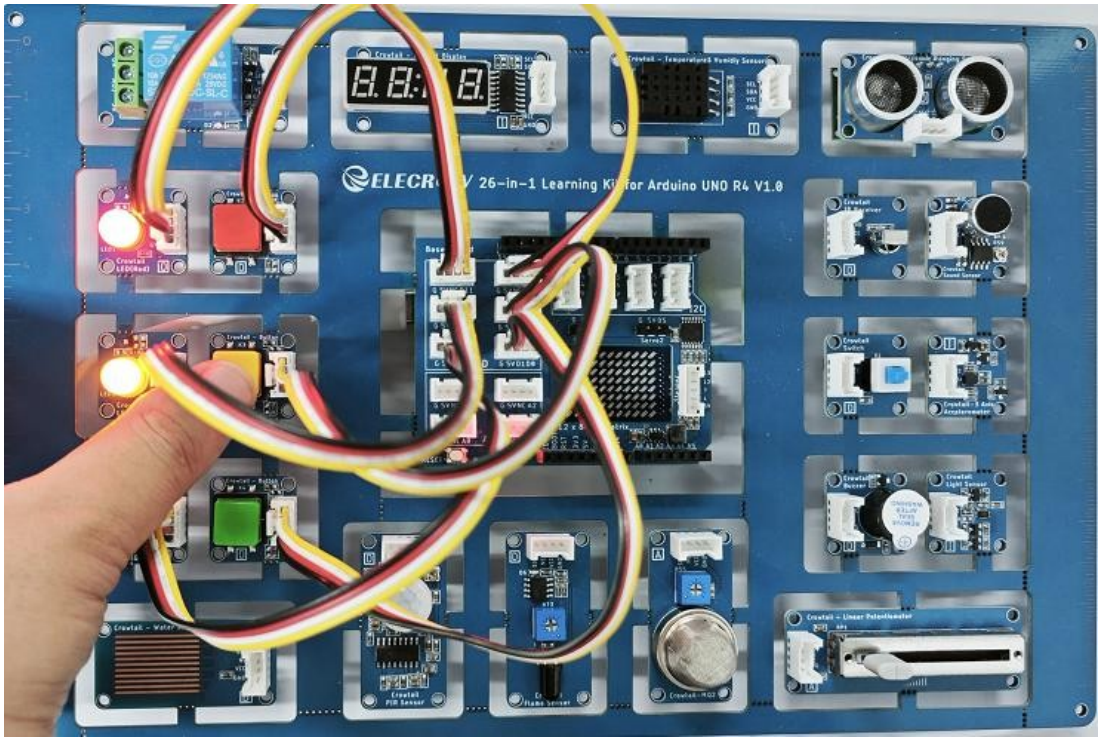
Keď sa hra spustí, LED diódy sa rozsvietia v náhodnom poradí. Vašou úlohou je zapamätať si toto poradie.



Po zapamätaní stlačte tlačidlá v poradí, ktoré ste si zapamätali.



Ak stlačíte tlačidlá v rovnakom poradí ako v pôvodnej sekvencii LED, všetky LED sa rozsvietia súčasne, čo znamená, že ste zadali správny kód.


26-in-1 Learning Kit for Arduino UNO R4 V1.0

Lekcia 26 — Morseovka

Úvod

V tejto lekcii použijeme tlačidlo a displej LCD1602 na vytvorenie dekódovania morseovky

Minihra založená na rozpoznávaní dĺžky stlačenia. Na základe merania dĺžky stlačenia tlačidla dokáže systém rozlíšiť bodku (·) a pomlčku (—) a v reálnom čase zobraziť zadaný morseov kód na LCD displeji, pričom ho nakoniec dekóduje na príslušné anglické písmeno.

Tento projekt presahuje rámec jednoduchého ovládania typu „stlač raz, rozsvieti sa“. Zavádza vnímanie času a pamäť stavu, čo predstavuje dôležitý krok od základného ovládania smerom k inteligentnej interakcii.

Ciele vzdelávania

1Porozumieť základným pravidlám morseovky (bodky a pomlčky).

2. Naučte sa rozlišovať vstupné signály na základe dĺžky stlačenia tlačidla.
3. Ovládnite praktické použitie funkcie millis() na meranie času.
4. Naučte sa kombinovať vstupy pomocou reťazcov (String).
5. Naučte sa zobrazovať dekódované výsledky na LCD1602.
6. Dokončite minihru s dekódovaním morseovky.

Náhľad výsledku

Krátke stlačenie (< 300 ms): Zadajte

bodku . Dlhé stlačenie (≥ 300 ms):

Zadajte pomlčku -

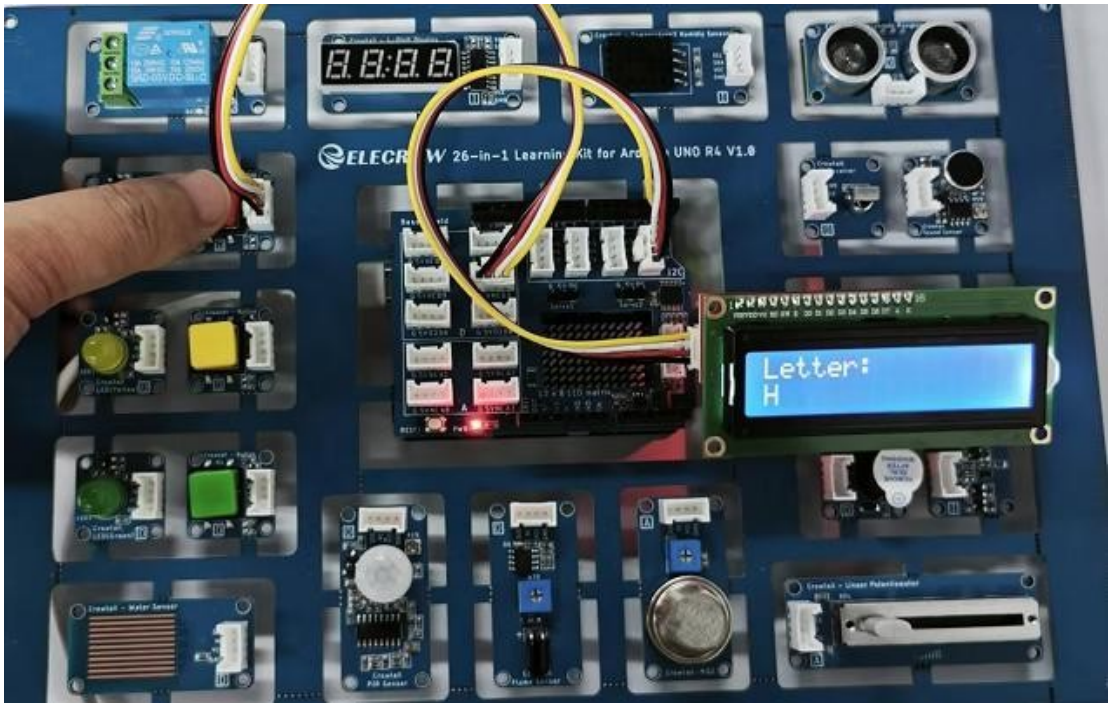
Pauza > 1 sekunda: Označuje koniec zadávania písmena

Prvý riadok LCD: Zobrazuje aktuálne zadaný morseov kód

Druhý riadok LCD: Zobrazuje dekódované anglické

písmeno Príklad:

Vkladanie.....→ LCD displej zobrazí H.



1. Vysvetlenie princípu

Morseovka je komunikačná metóda založená na čase:

Krátky signál → bodka (-)

Dlhý signál → pomlčka (—)

V tomto experimente:

Tlačidlo simuluje „klepnutie“ v telegrafii.

Dĺžka stlačenia rozlišuje medzi bodkami a pomlčkami. Dĺžka

pauzy určuje medzeru medzi písmenami.

Funkcia Arduino millis() vám umožňuje získať aktuálny čas behu bez blokovania programu, čo je ideálne pre implementáciu tohto druhu logiky založenej na časovaní.

2. Požadované moduly

Crowtail -- Tlačidlo × 1

Crowtail - LCD 1602 ×1

3. Spôsob zapojenia

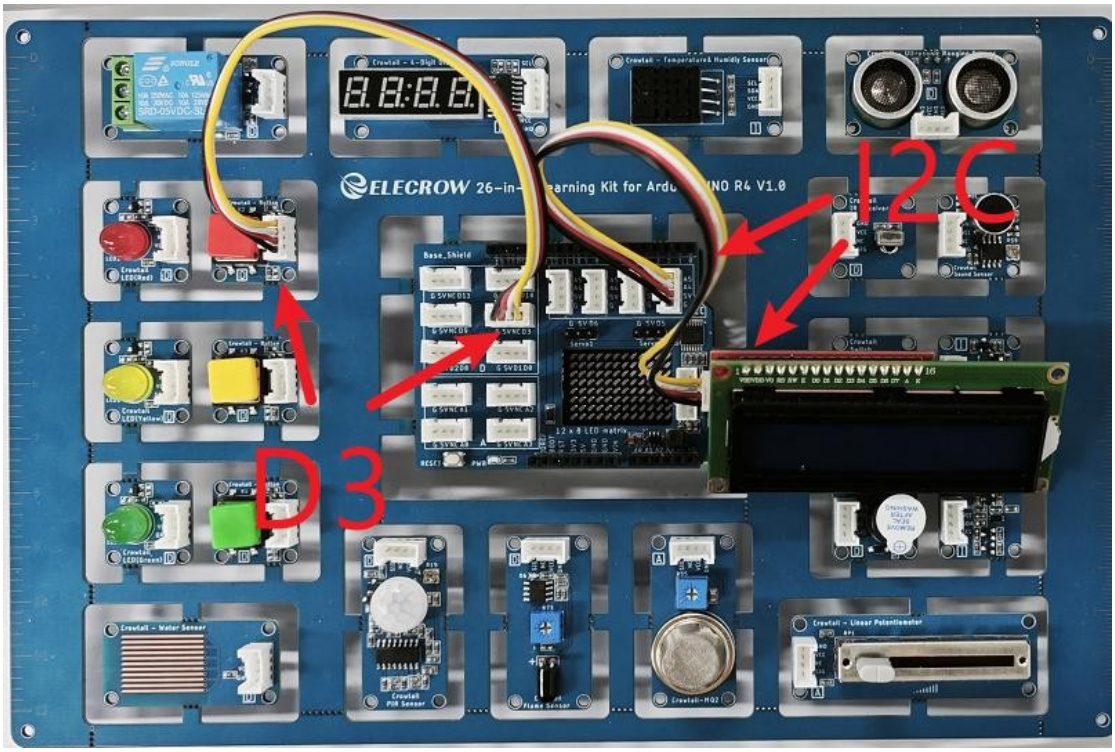
Crowtail – tlačidlo → digitálny port D3

Crowtail LCD1602 → port I2C

Špecifikácia rozhrania Crowtail so štyrmi portmi:

- GND (čierna) → GND
- VCC (červená) → 5V
- SDA (biela) → Bez pripojenia
- SCL (žltá) → I2C/D3 k

napájaniu 5 V.



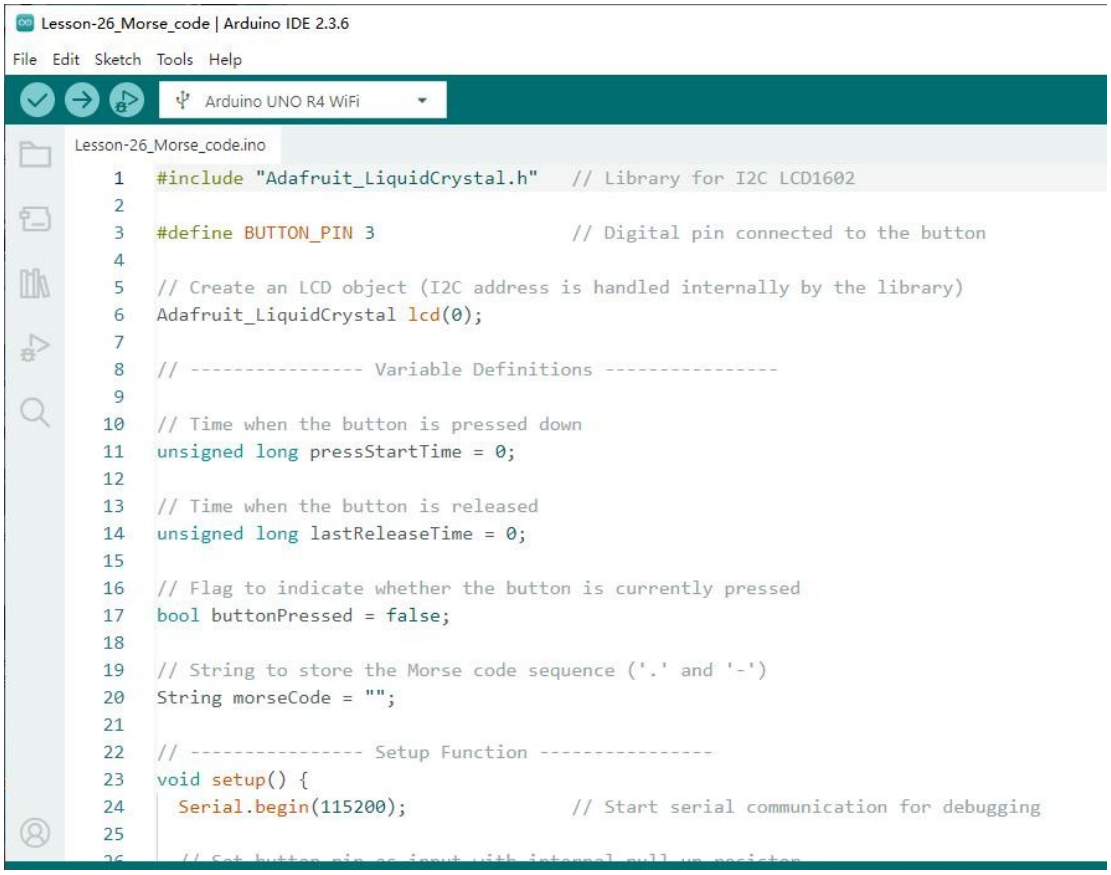
4. Vysvetlenie príkladu

Kliknite na nižšie uvedený odkaz a stiahnite si oficiálny príklad

kódu. Odkaz na GitHub:

https://github.com/Elecrow-RD/26-in-1-Learning-Kit-for-Arduino-UNO-R4/tree/master/lesson_co de/lesson-26_Morse_code

Otvorte program pomocou Arduino IDE.



Vysvetlenie kódov klávesov

(1) Použité knižnice.

```
#include "Adafruit_LiquidCrystal.h"
```

Táto knižnica úplne zapuzdruje komunikáciu s I2C LCD1602, takže sa nemusíme starať o nízkoúrovňové načasovanie I2C.

Operácie s displejom je možné spravovať priamo pomocou funkcií ako print() a setCursor().

(2) Definície objektov tlačidiel a LCD.

```
#define BUTTON_PIN 3

Adafruit_LiquidCrystal lcd(0);
```

BUTTON_PIN: Určuje digitálny pin pripojený k tlačidlu.

lcd(0): Vytvorí objekt na ovládanie LCD. Všetky operácie s displejom prechádzajú cez tento objekt.

(3) Inicializačné premenné.

```
unsigned long pressStartTime = 0; unsigned

long lastReleaseTime = 0;

bool buttonPressed = false;
```

```
String morseCode = "";
```

pressStartTime: Zaznamenáva, kedy je tlačidlo stlačené. lastReleaseTime:

Zaznamenáva, kedy je tlačidlo uvoľnené. buttonPressed: Sleduje, či je

tlačidlo momentálne stlačené. morseCode: Ukladá aktuálnu sekvenciu

morseovho kódu.

Tieto premenné poskytujú programu pamäť stavu, ktorá je kľúčová pre implementáciu časovej logiky.

(4) setup() Inicializácia.

```
void setup() {
    Serial.begin(115200);
    pinMode(BUTTON_PIN, INPUT_PULLUP);


    while (!lcd.begin(16, 2)) {
        delay(50);
    }


    lcd.setBacklight(1);
    lcd.print("Morse Game");
    delay(1500);
    lcd.clear();
}
```

Inicializuje sériový port pre ladenie. Nastaví

tlačidlo do režimu INPUT_PULLUP.

Inicializuje LCD displej a zobrazí uvítací text, aby sa zabezpečilo, že systém sa spustí v známom stave.

(5) Hlavná slučka.

①Detekcia stlačenia tlačidla

```
if (buttonState == LOW && !buttonPressed) {
    buttonPressed = true;
    pressStartTime = millis();
}
```

Keď sa stav tlačidla zmení z nestlačeného na stlačené, zaznamenajte aktuálny čas ako začiatok vstupu.

Funkcia millis() vráti počet milisekúnd, ktoré uplynuli od zapnutia alebo resetovania Arduina, čo je nevyhnutné

pre logiku založenú na časovaní.

②Uvoľnenie tlačidla a detekcia bodky/čiarky

```
if (buttonState == HIGH && buttonPressed) {
    buttonPressed = false;
    unsigned long pressDuration = millis() - pressStartTime;


    if (pressDuration < 300) morseCode += "."; else
    morseCode += "-";


    lastReleaseTime = millis();
    displayMorse();
}
```

Na základe dĺžky stlačenia určí, či ide o bodku alebo čiaru. Výsledok pripojí k

reťazcu morseCode.

Aktualizuje LCD v reálnom čase s aktuálnym vstupom morseovho kódu.

③Dokončenie písmena a dekódovanie

```
if (morseCode.length() > 0 && millis() - lastReleaseTime > 1000) { char
    letter = decodeMorse(morseCode);
    displayLetter(letter);
    morseCode = "";
}
```

Ak uplynie viac ako 1 sekunda bez nového vstupu, systém považuje zadanie písmena za dokončené a dekóduje ho na zobrazenie.

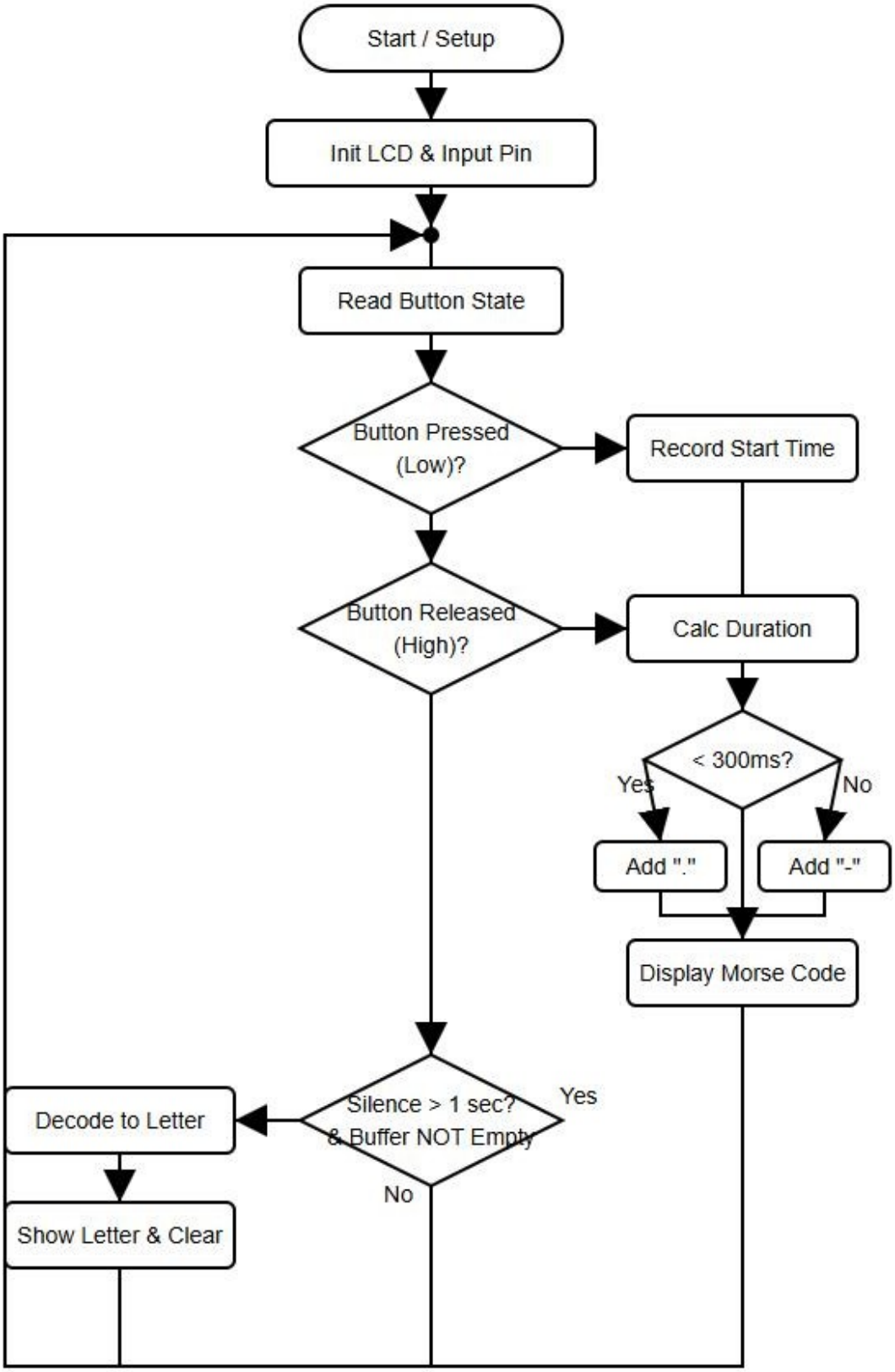
(6) Funkcia dekódovania morseovky.

①Vygenerujte náhodnú sekvenciu

```
char decodeMorse(String code) { if
    (code == ".-") return 'A';
    …
    if (code == "...") return 'S';
    if (code == "---") return 'O';
    return '?';
}
```

Priraďuje reťazce morseovského kódu k anglickým písmenám pomocou jednoduchých if príkazov. Jednoduché a ľahko zrozumiteľné pre začiatočníkov a ľahko rozšíriteľné o ďalšie písmená.

(7) Celkový diagram logiky kódu

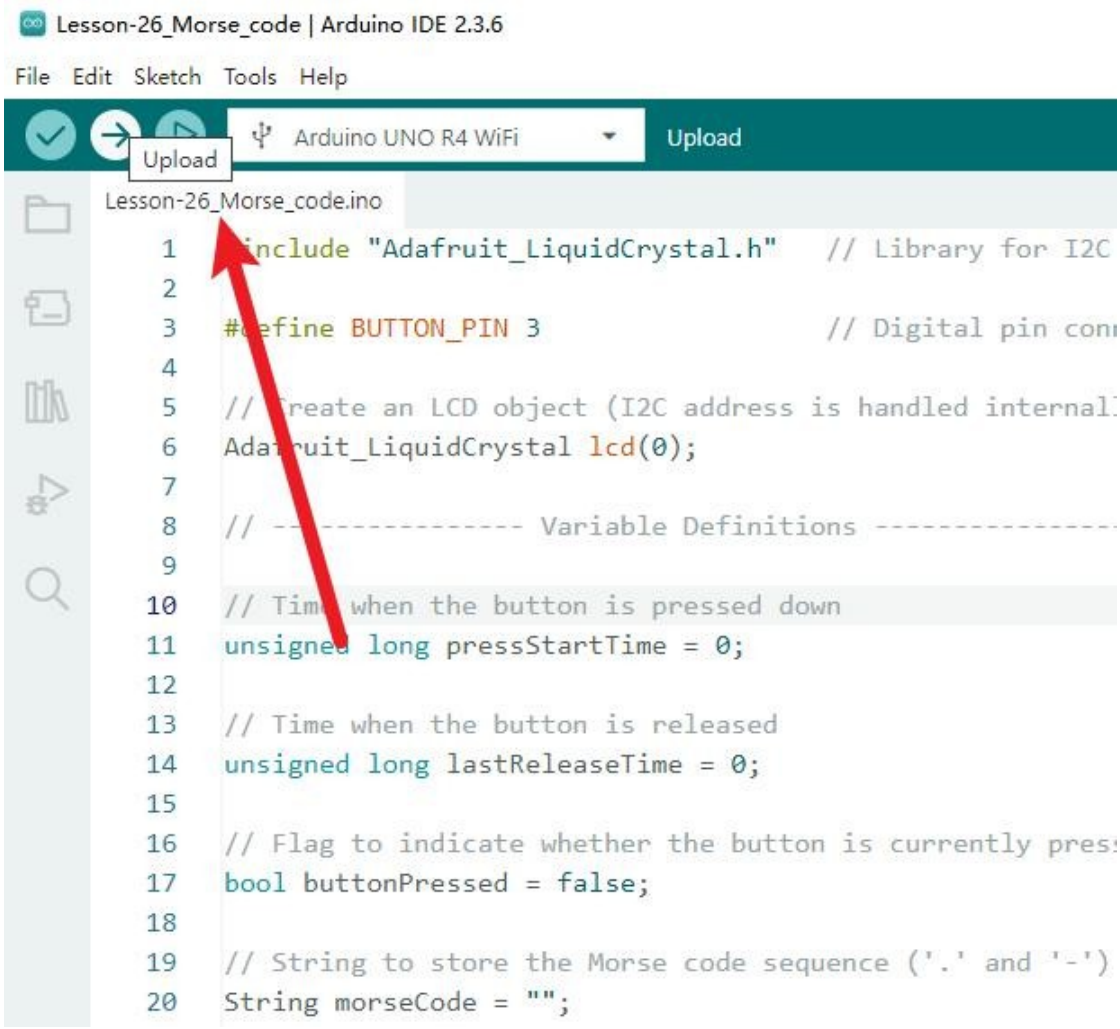


5. Spustite program a sledujte výsledky

(1) Postupujte podľa krokov na obsluhu prostredia Arduino IDE uvedených na strane 8, správne pripojte počítač

a dokončite základnú konfiguráciu nahrávania.

Kliknite na „Stiahnuť“:



(2) Program sa spustí.

Program beží:

Správanie programu:

Krátke stlačenie: LCD zobrazí

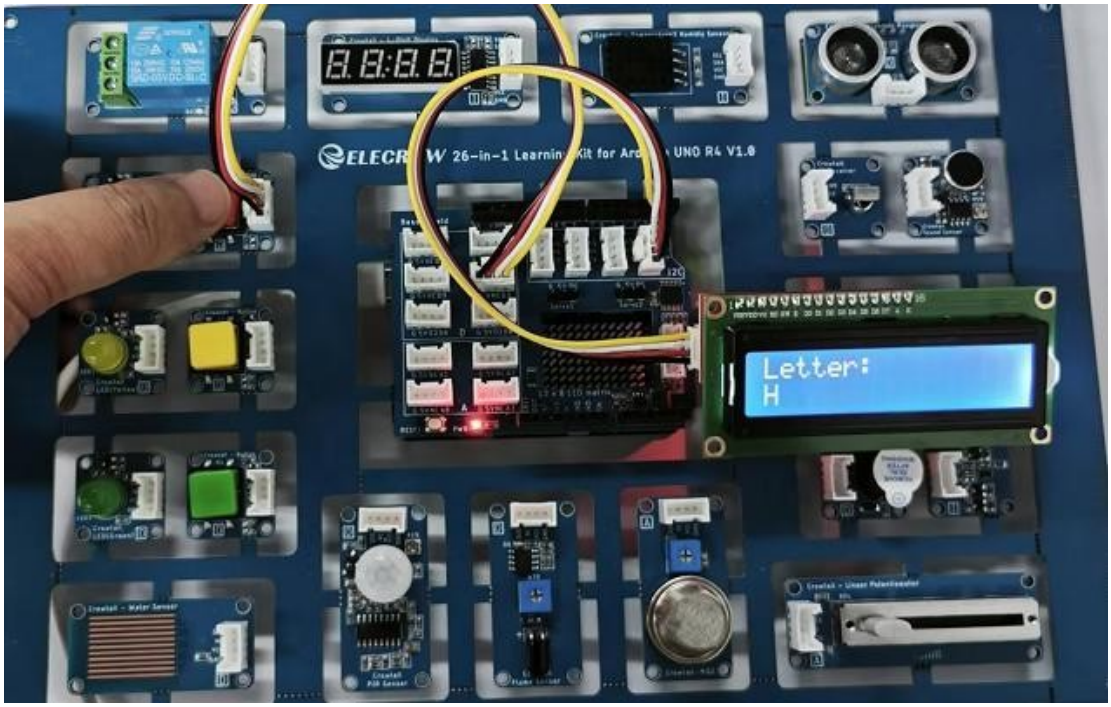
. Dlhé stlačenie: LCD zobrazí

-

Pauza > 1 sekunda: LCD zobrazí dekódované písmeno. Môžete pokračovať v zadávaní viacerých písmen na precvičenie.

Príklad:

Stlačte tlačidlo štyrikrát krátko → LCD zobrazí.....→ dekódované písmeno je H.



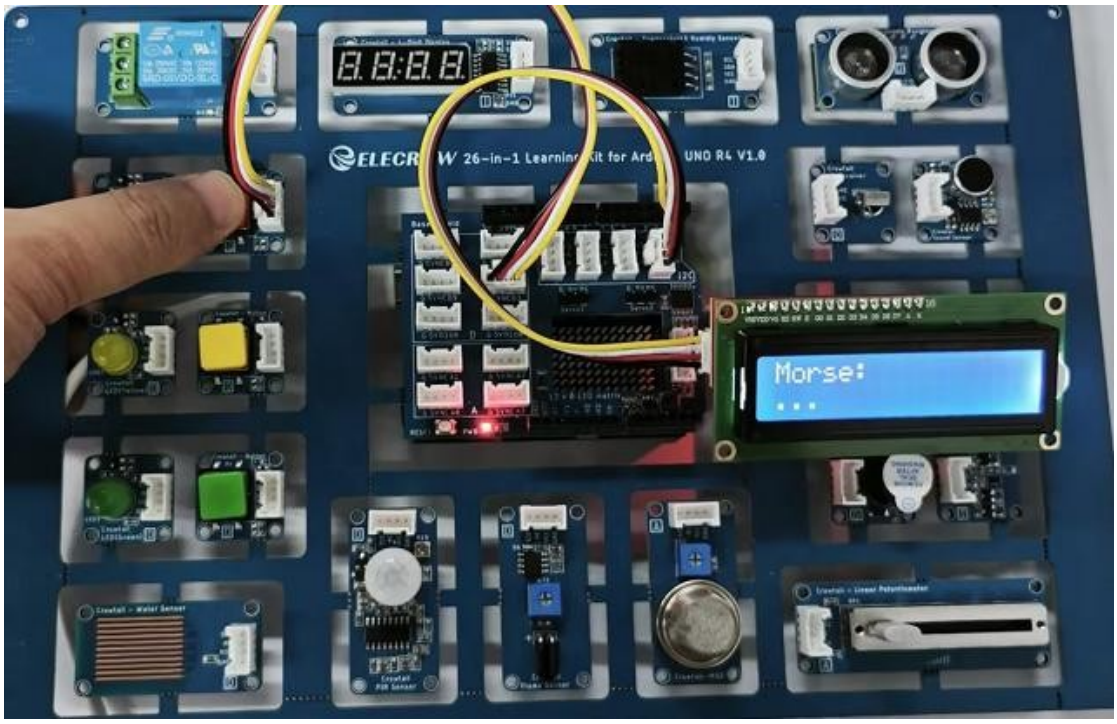
Príklad 2:

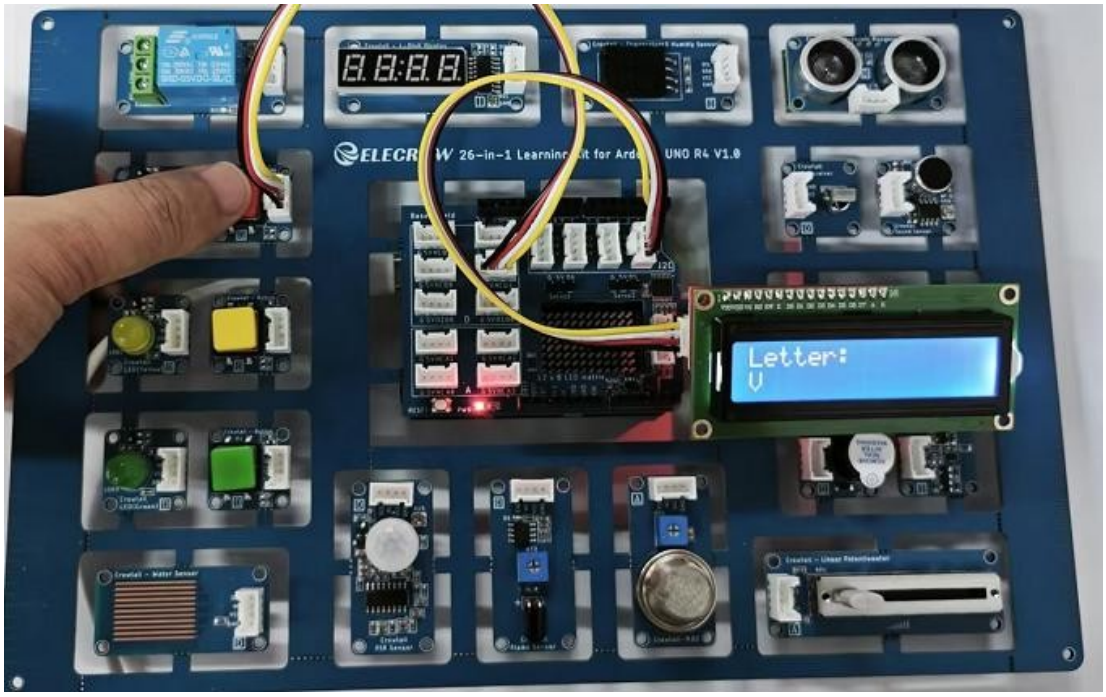
Stlačte tlačidlo šesťkrát krátko → Na displeji LCD sa zobrazí → dekódované písmeno je ?

Je to preto, že pre túto sekvenciu morseovky neexistuje žiadne zodpovedajúce písmeno.

Príklad 3:

Stlačte tlačidlo krátko trikrát → na LCD sa zobrazí ... → dekódované písmeno je V.


UNO R4 V1.0
Morse:
...


UNO R4 V1.0
Letter:
V