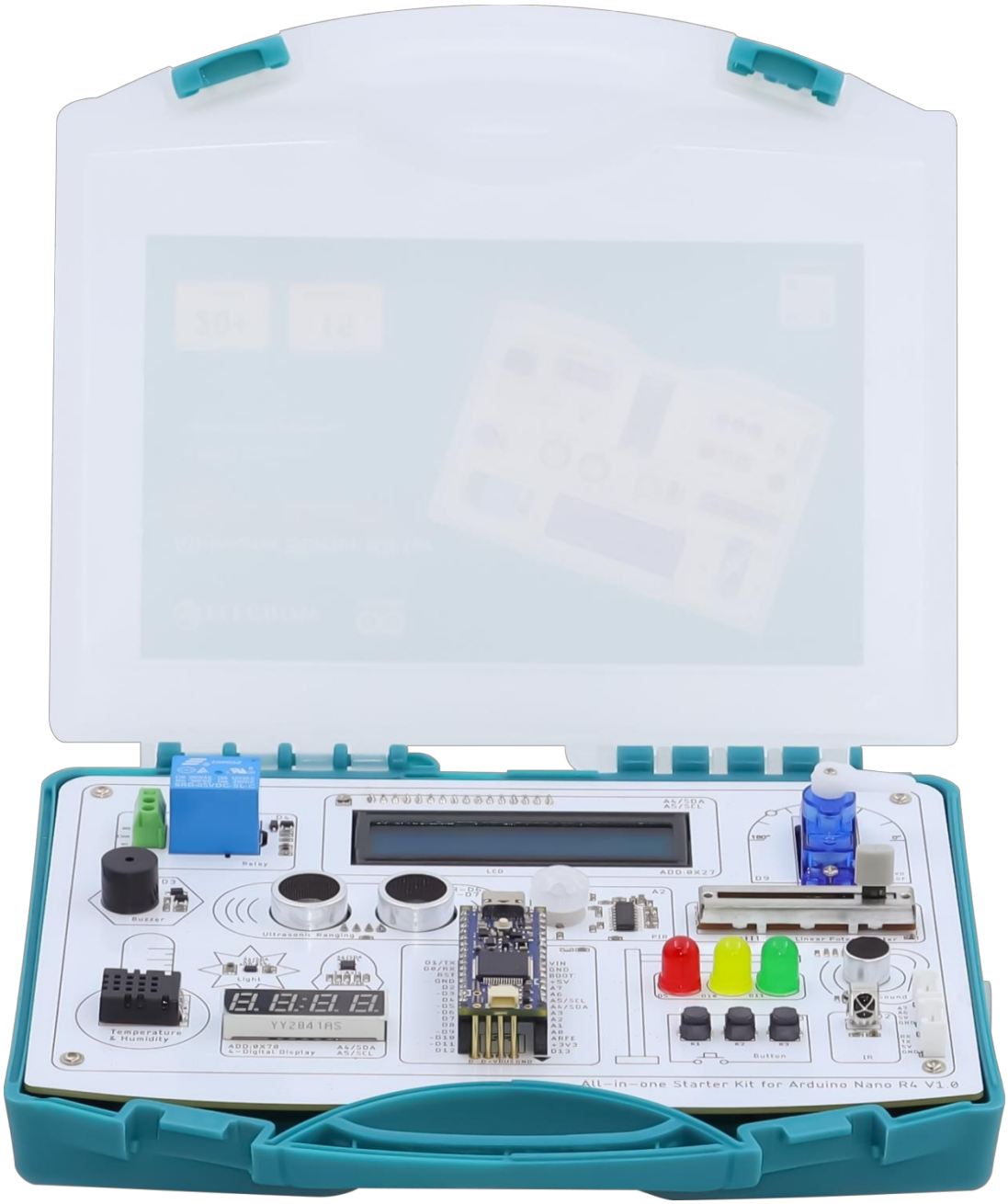
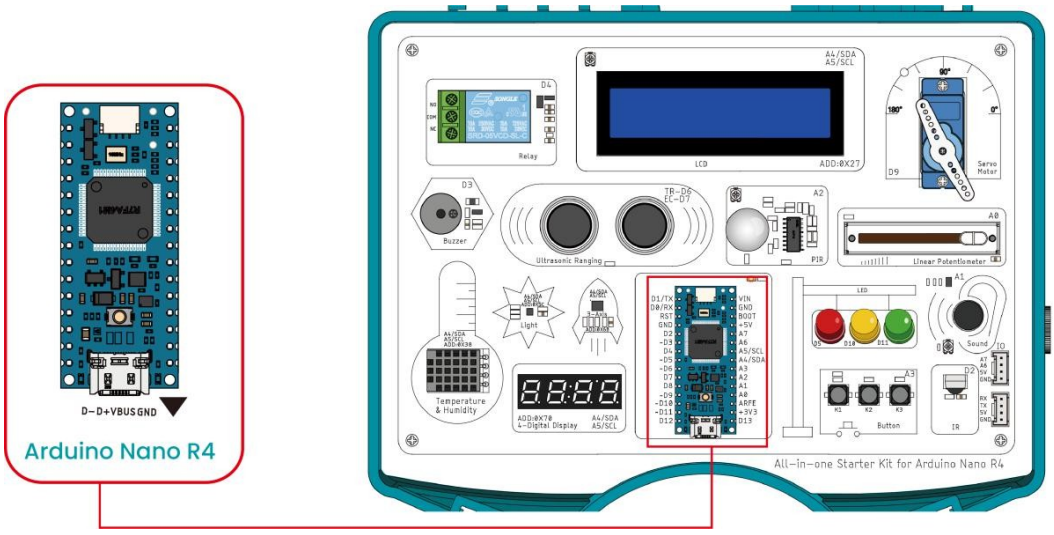
Kompletná štartovacia sada pre Arduino Nano R4



Úvod

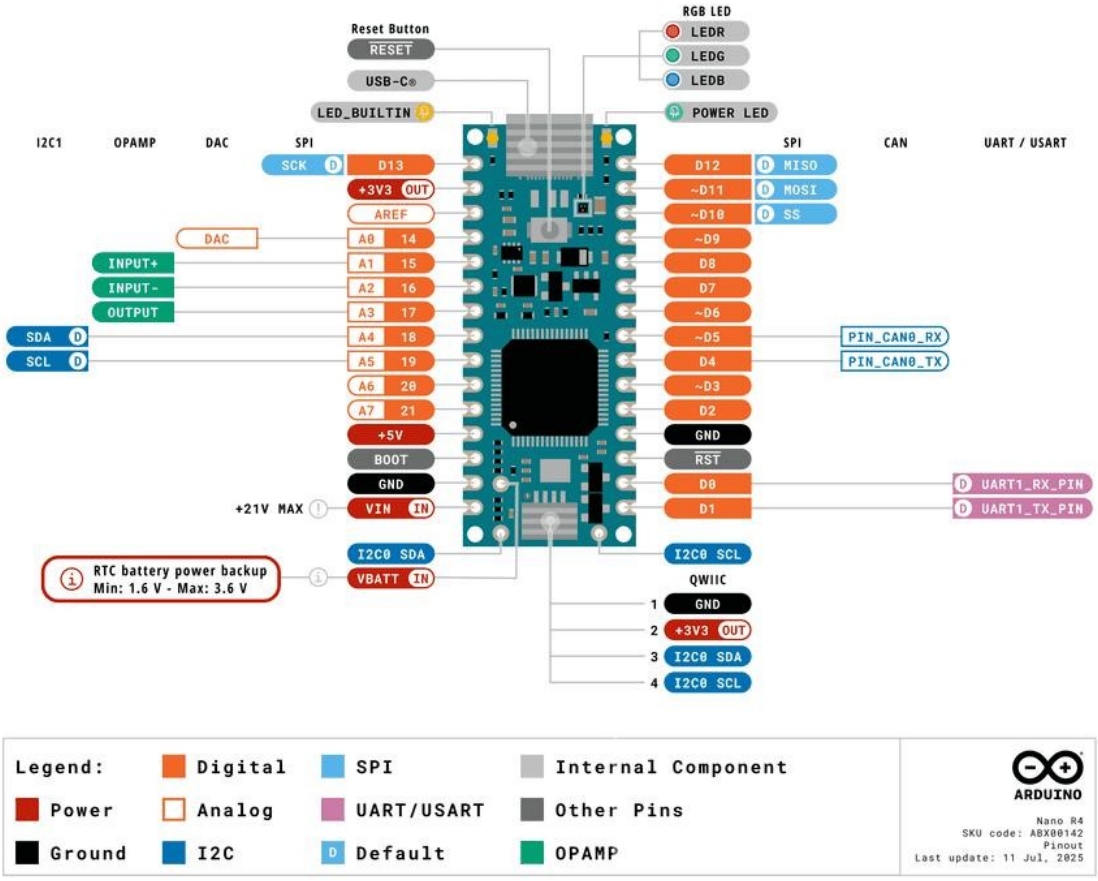
Vitajte v používateľskej príručke k vývojárskej doske Arduino Nano R4. Poďme sa bližšie oboznámiť s doskou Arduino Nano R4 a naučiť sa, ako využívať jej širokú škálu elektronických modulov!
Nebojte sa – táto doska obsahuje 20 ľahko zrozumiteľných, zaujímavých a inšpiratívnych lekcií, ktoré vás prevedú celým procesom krok za krokom. Tu sa zoznámite s rôznymi elektronickými modulmi, rozviniete logické myslenie, podnietite kreativitu a naučíte sa, ako tieto moduly programovať.
Lekcie začínajú základmi, ako je inštalácia softvéru Arduino, potom predstavujú dosku Arduino Nano R4 a jej senzory, nasleduje programovanie týchto modulov a naučenie sa používaných programovacích jazykov. Nakoniec sa naučíte, ako implementovať praktické aplikácie pre tieto senzory. Každý krok je jasne vysvetlený, takže aj začiatočníci môžu rýchlo pochopiť programovanie Arduina.
Doska Arduino Nano R4 obsahuje 15 elektronických modulov, z ktorých každý má jedinečné funkcie a vlastnosti, vďaka čomu je ideálna pre začiatočníkov. Vďaka štúdiu nielenže pochopíte základné princípy senzorov, digitálnych a analógových signálov, analógovo-digitálneho prevodu a programovacej logiky, ale naučíte sa aj pracovať s pokročilejšími modulmi. A čo je najdôležitejšie, oficiálne začnete študovať programovanie Arduina, čím posilníte svoje schopnosti logického myslenia.
Na programovanie použijeme softvér Arduino. Platforma Arduino patrí medzi najpopulárnejšie open-source platformy na svete. Je jednoduchá na používanie, ponúka bohaté hardvérové zdroje (rôzne dosky Arduino) aj softvérové zdroje (Arduino IDE), čo z nej robí jednu z najlepších volieb na štúdium programovania.

Úvod do vývojovej dosky



Pozrime sa bližšie na vývojovú dosku Arduino Nano R4! Mnohí začiatočníci možno

, čo znamenajú čísla a označenia na doske. Ďalej si prejdeme každé označenie jedno po druhom, aby ste pochopili účel každého pinu a rozhrania a mohli dosku využívať efektívnejšie.

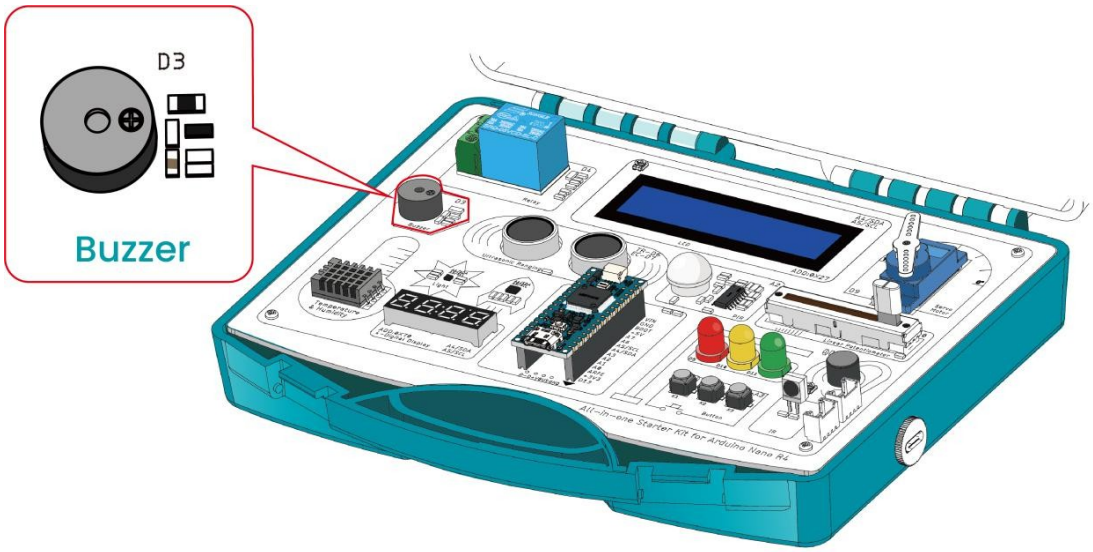


①Inštalácia hlavnej dosky

Najskôr vložte hlavnú dosku Arduino Nano R4 do stredného slotu vývojovej dosky. Uistite sa, že port Type-C smeruje nadol. Na vývojovej doske sú jasne označené polohy všetkých pinov Arduino Nano R4, čo uľahčuje ich pripojenie.

②Porozumenie označeniam pinov a pripojeniam modulov

Na vývojovej doske si všimnete, že každý senzorový modul je označený príslušným číslom pinu. Napríklad modul bzučiaka je označený ako „D3“, čo znamená, že sa pripája k pinu D3 na hlavnej doske Arduino Nano R4.



Keď sa pozrieme na hlavnú dosku, vidíme pin označený ako „~D3“. Symbol „~“ je veľmi dôležitý – označuje, že tento pin podporuje výstup PWM (pulzná šírková modulácia).

To znamená, že môžeme robiť viac než len zapínať alebo vypínať bzučiak; môžeme tiež naprogramovať rôzne frekvencie PWM, aby bzučiak vydával rôzne tóny.

„D4“ označuje digitálny pin. Zvyčajne sa používa na: prijímanie vysokých alebo nízkych signálov z vstupných modulov (napr. stlačenie alebo uvoľnenie tlačidla); ovládanie zapnutého/vypnutého stavu výstupných modulov (napr. zapnutie alebo vypnutie LED); alebo čítanie signálov zo senzorov, ktoré majú len dva stavy (zapnuté/vypnuté).

„~D3“: Tilda „~“ označuje podporu PWM. Takéto piny môžu vysielať digitálne signály s vysokou/nízkou úrovňou ako bežné digitálne piny, ako aj analógové signály PWM, ktoré môžu ovládať jas, rýchlosť alebo výšku tónu. Môžu tiež čítať digitálne signály zo senzorov.

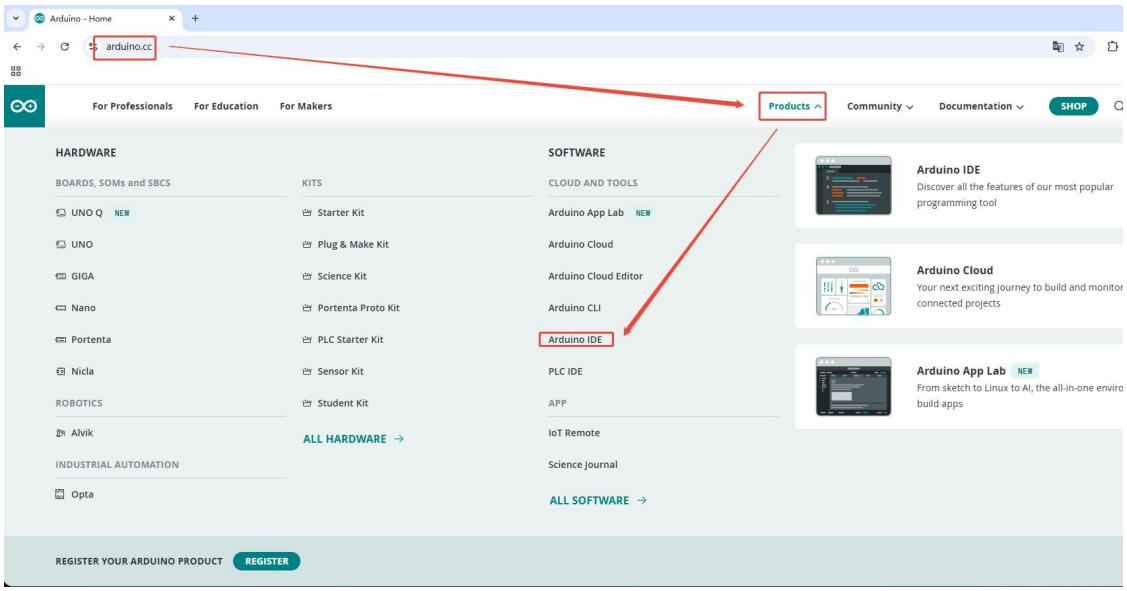
„A2“ označuje analógový vstupný pin. Slúži na snímanie plynulých analógových signálov zo senzorov, napríklad meniacich sa napätí. Je to bežné u senzorov s viacerými stavmi, ako je napríklad posuvný potenciometer. Samozrejme, dá sa použiť aj na jednoduché dvojstavové signály, hoci v takom prípade sa zvyčajne uprednostňuje digitálny pin.

Inštalácia Arduino IDE

Stiahnutie Arduina 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 systém zodpovedajúci vášmu počítaču, napríklad systém Windows.

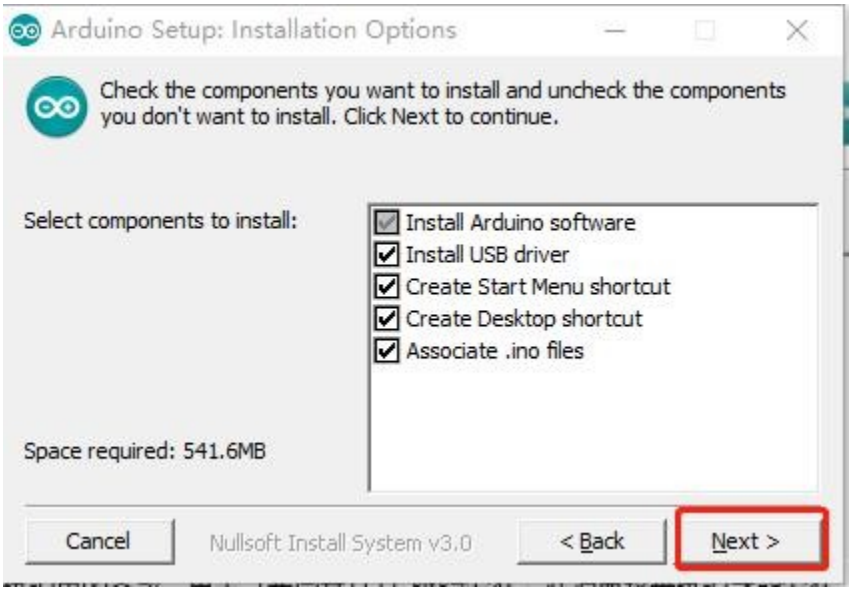
Poznámka: V tomto návode sa používa verzia 2.3.6. Môžete vyskúšať aj iné verzie, ale ak sa počas flashovania vyskytnú akékoľvek problémy, prejdite späť na verziu 2.3.6 a skúste to znovu.

Krok 3:

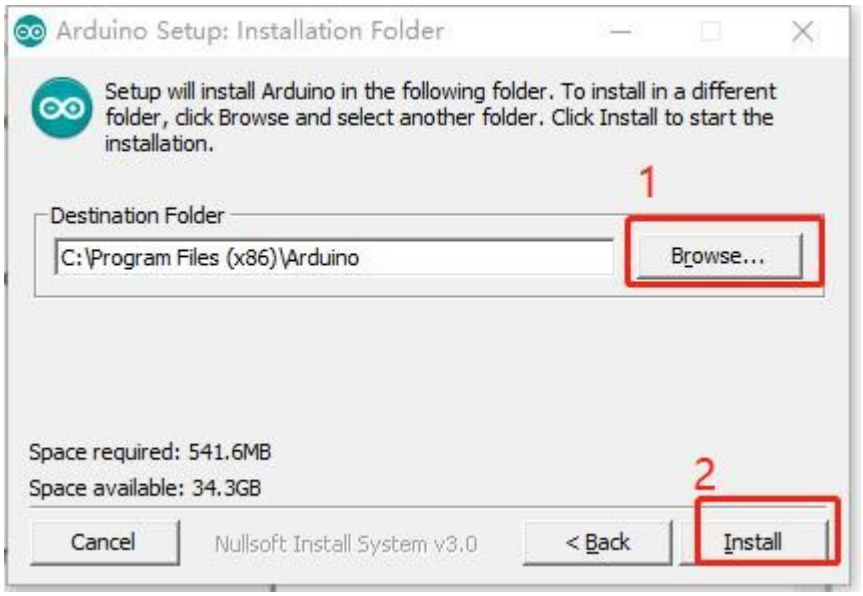
1. Pri inštalácii Arduina vyhľadajte spustiteľný súbor s príponou .exe v priečinku, do ktorého ste ho predtým uložili, teda v inštalačnom balíku Arduina.

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

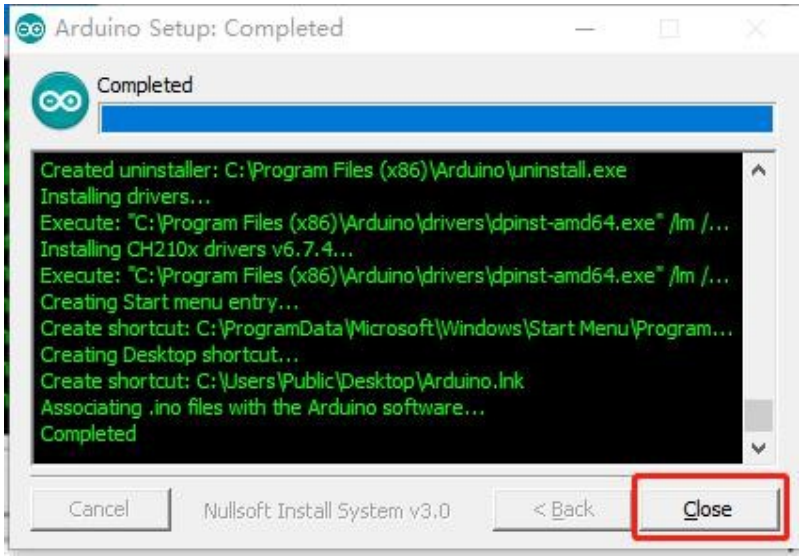
3. Zaškrtnite všetky predvolené možnosti a kliknite na „Ďalej“.



4. Kliknite na „Prehľadávať“ a vyberte umiestnenie inštalácie. Odporúča sa inštalovať na akýkoľvek iný disk ako disk C:. Potom kliknite na „Inštalovať“.



5. Inštalácia bola dokončená



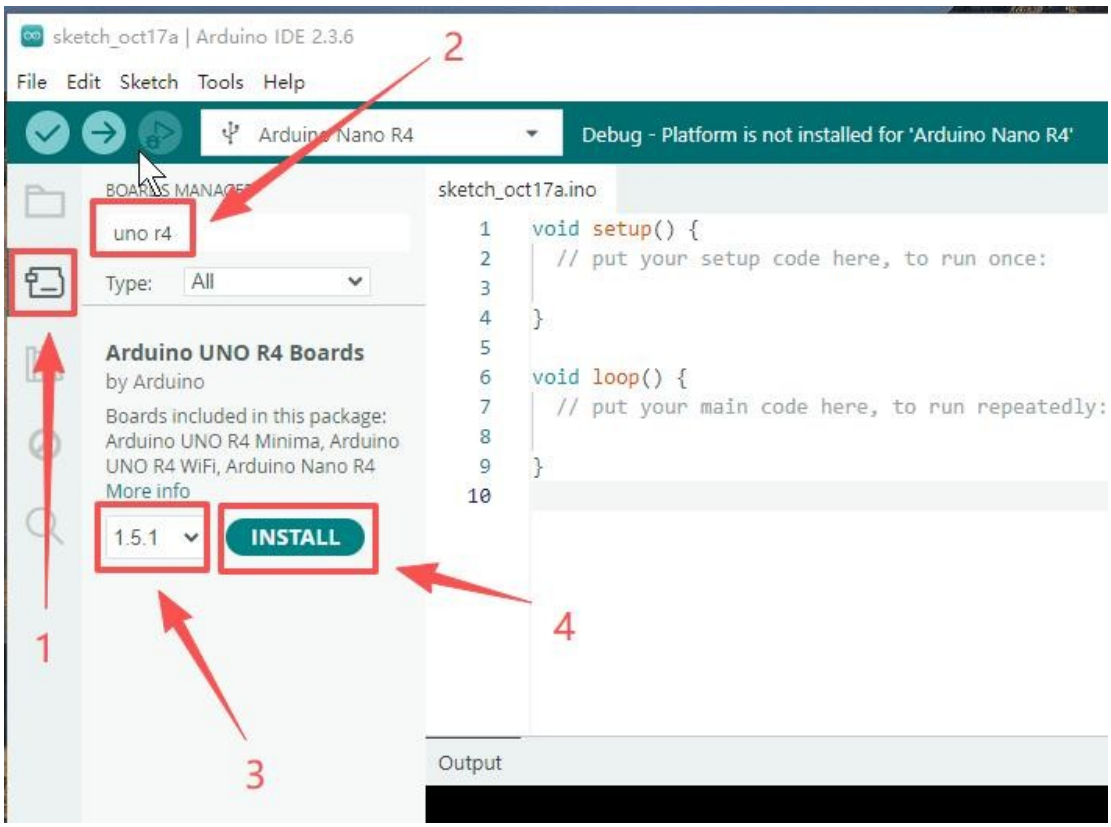
Inštalácia vývojovej dosky Nano R4

①Kliknite na ikonu vývojovej dosky vľavo

②Do vyhľadávacieho poľa napíšte „uno r4“

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

④Kliknite na „INSTALL“ (Inštalovať)



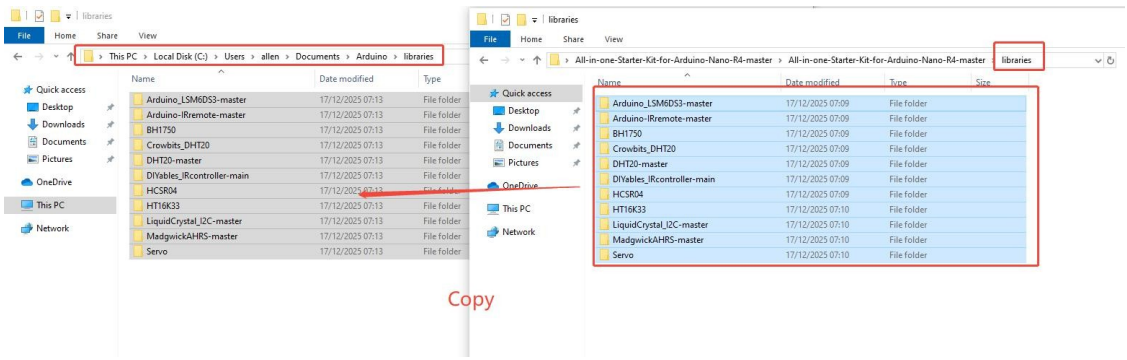
Počas inštalácie sa zobrazí okno – stačí kliknúť na „Yes“.

Importovanie súborov knižnice (dôležité)

Pred začatím je potrebné stiahnuť požadované súbory knižníc, ktoré sú súčasťou sady, a umiestniť ich do príslušného adresára „/Arduino/libraries“ vo vašom počítači.

Odkaz na stiahnutie:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/libraries Skopírujte všetky súbory knižníc zo stiahnutej zložky knižníc do nasledujúceho adresára vo vašom počítači: „C:\Users\Meno používateľa\Documents\Arduino\libraries“

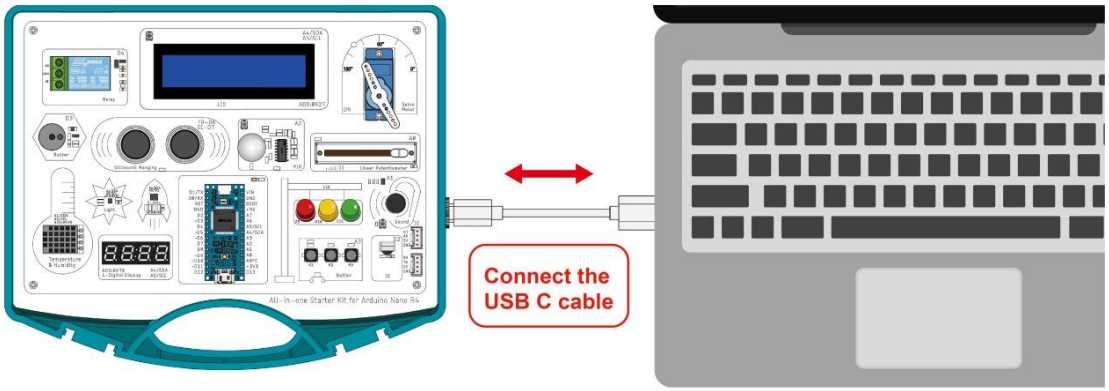


Poznámka: Ak v tomto adresári nie je zložka „libraries“, vytvorte ju ručne.

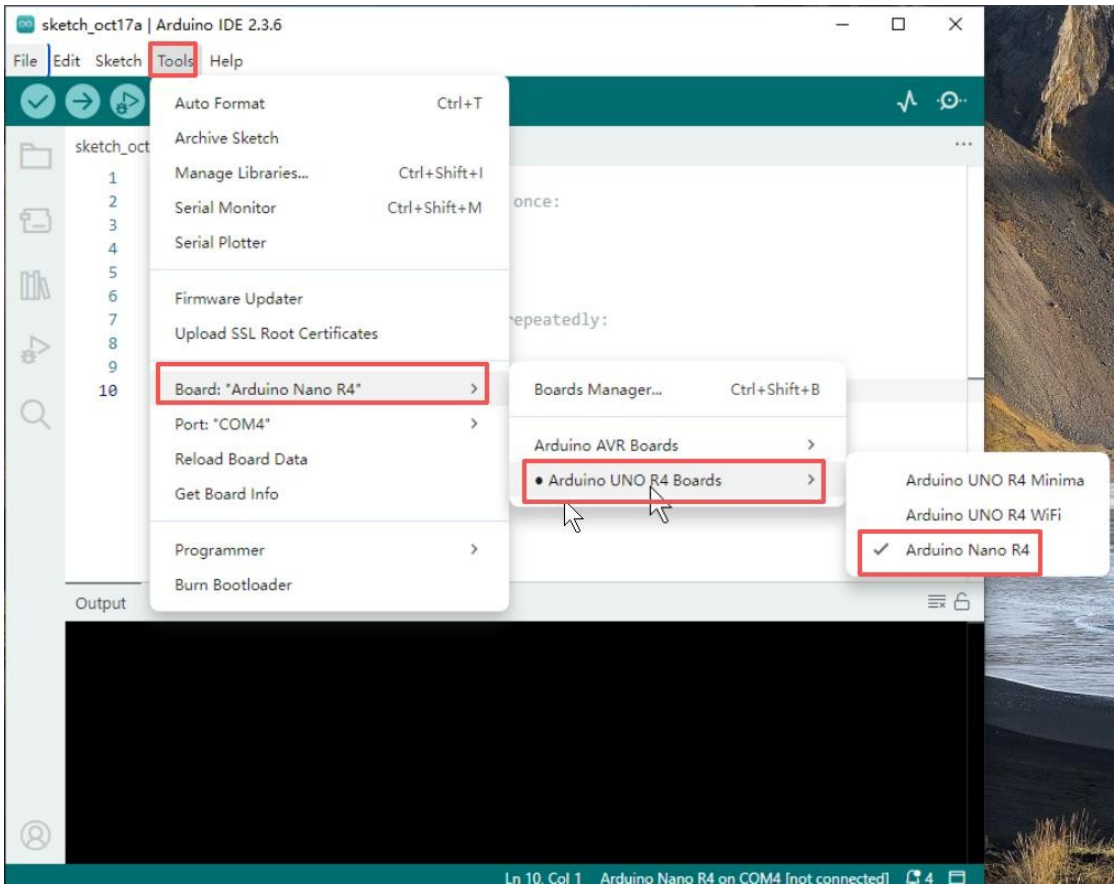
Postup nahrávania (dôležité)

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

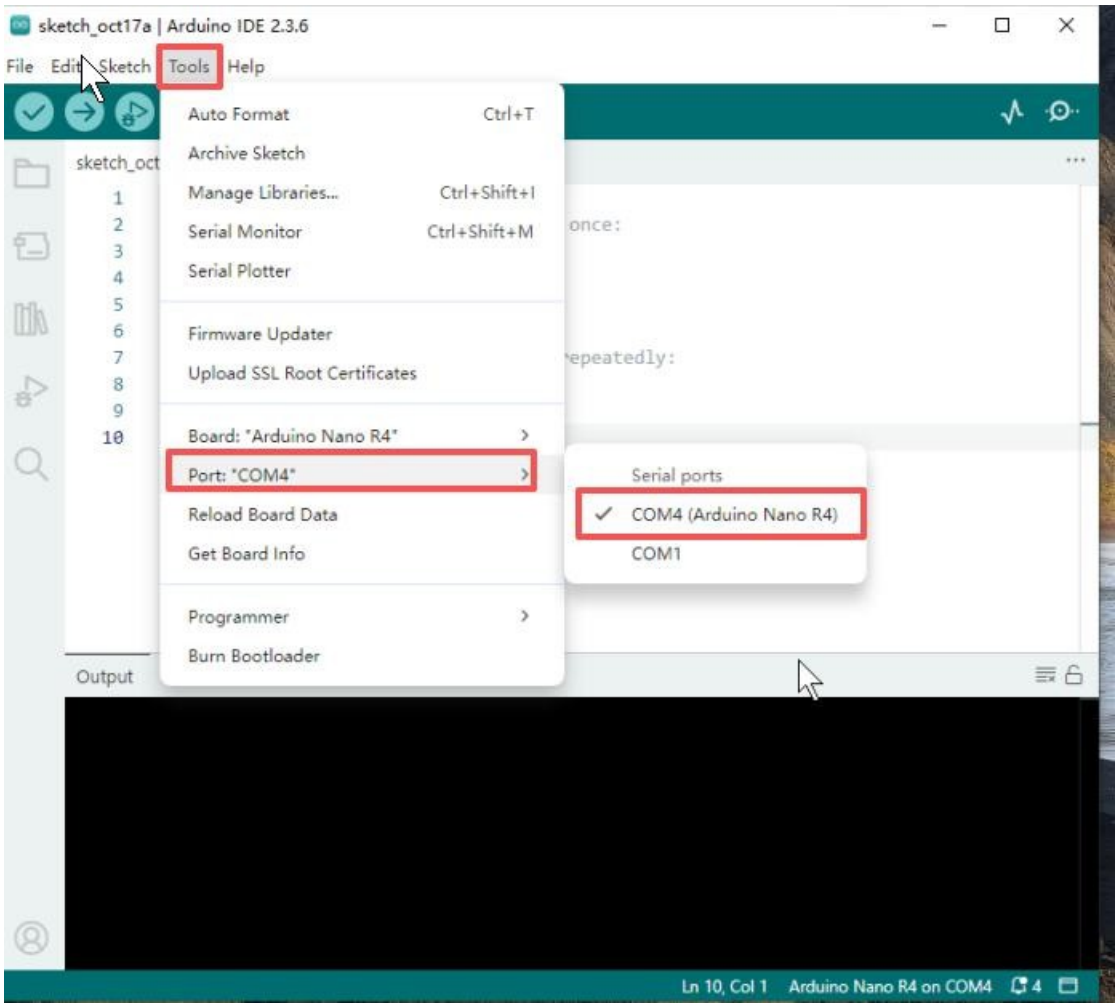
1. Pripojte port na nahrávanie typu C k počítaču.



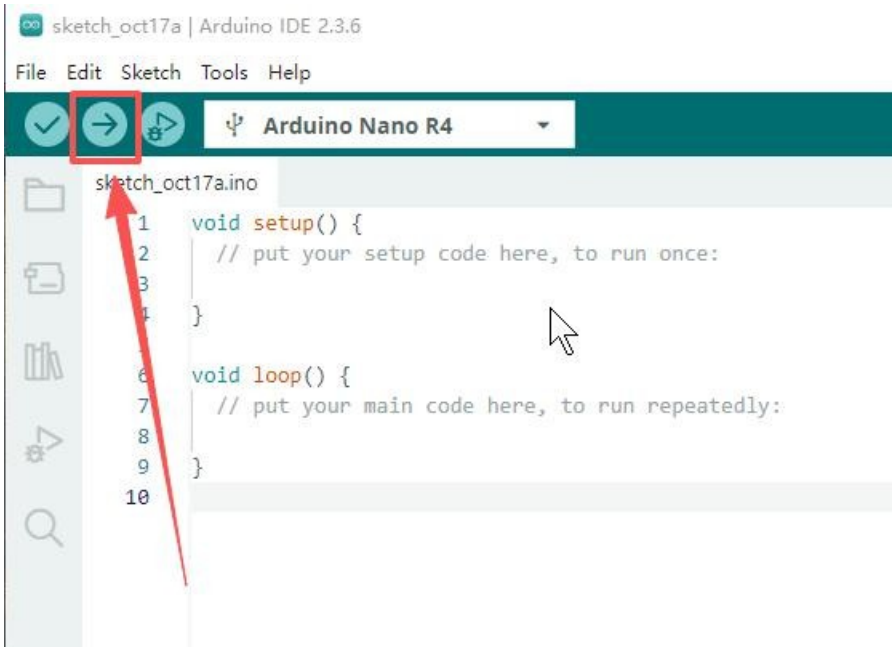
2. Po napísaní kódu vyberte dosku, na ktorú chcete nahrať: „Nástroje“ -> „Doska“ -> „Dosky Arduino UNO R4“ -> „Arduino Nano R4“



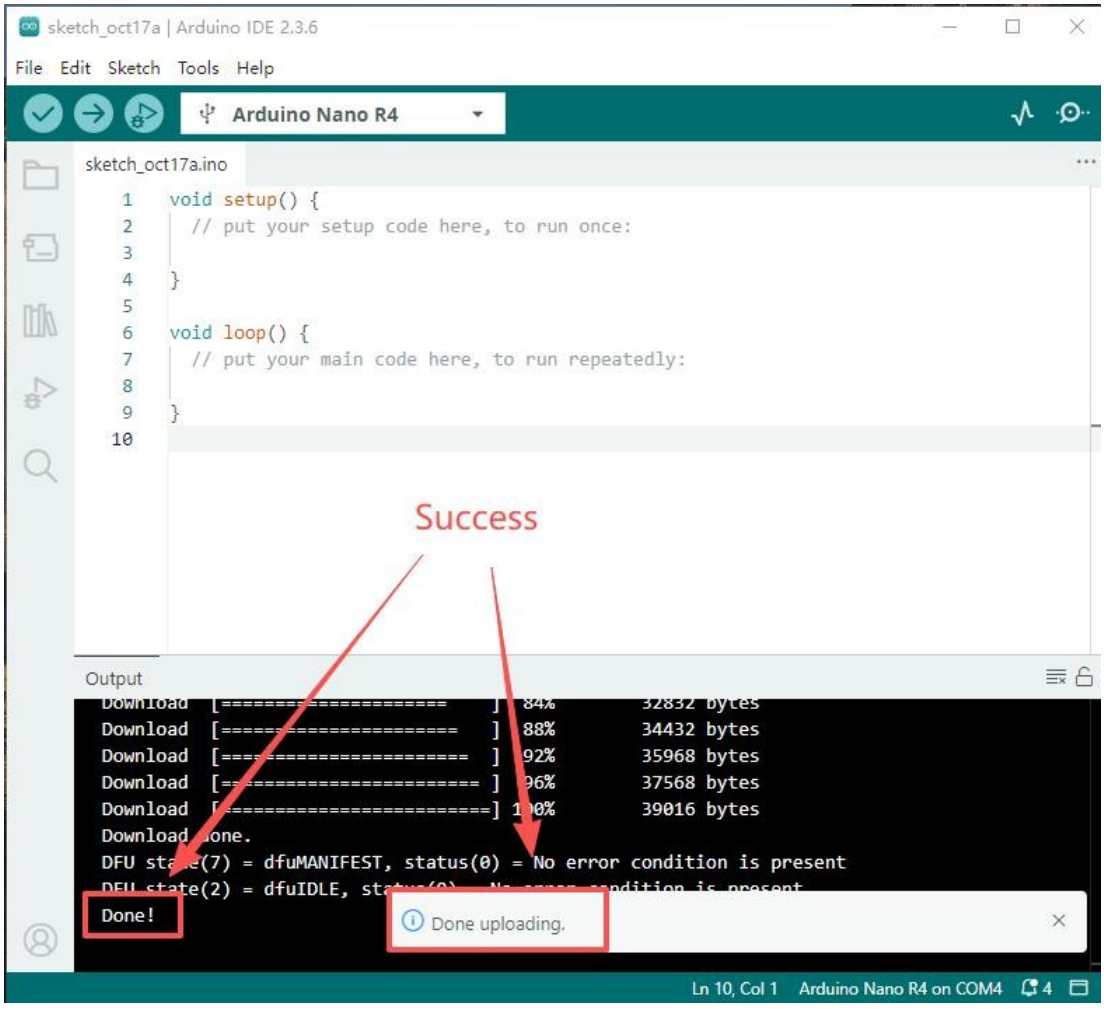
3. Vyberte sériový port pre nahrávanie (poznámka: v tutoriáli je zobrazený COM4, ale u vás môže byť iný – uistite sa, že zodpovedá vašej doske Arduino Nano R4). „Tools“ -> „Port“ -> „COM4 (Arduino Nano R4)“



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



5. Potvrďte, že nahratie prebehlo úspešne.


sketch_oct17a | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Nano R4
sketch_oct17a.ino
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
Success
Output
Download [ ] 84% 32832 bytes
Download [ ] 88% 34432 bytes
Download [ ] 92% 35968 bytes
Download [ ] 96% 37568 bytes
Download [ ] 100% 39016 bytes
Download done.
DFU state(7) = dfuMANIFEST, status(0) = No error condition is present
DFU state(2) = dfuIDLE,
Done!
Done uploading.
Ln 10, Col 1 Arduino Nano R4 on COM4 4

Lekcia 01 --- LED

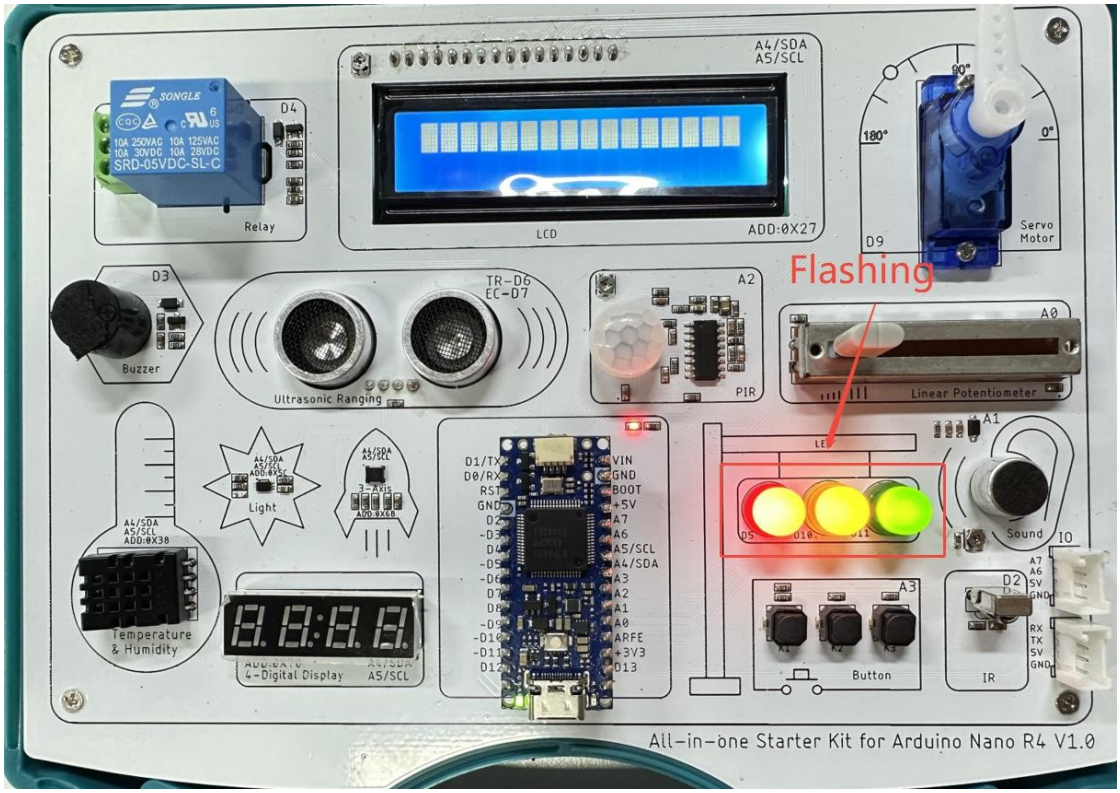
Úvod

V tejto lekcii sa systematicky naučíme, ako pomocou programovacieho jazyka Arduino ovládať tri LED indikátory na vývojárskej doske Arduino Nano R4. Na konci tejto časti budete nielen rozumieť tomu, ako písať programy pre Arduino, ale aj zvládnete základné operácie zapínania a vypínania integrovaných LED diód.

Ciele výučby

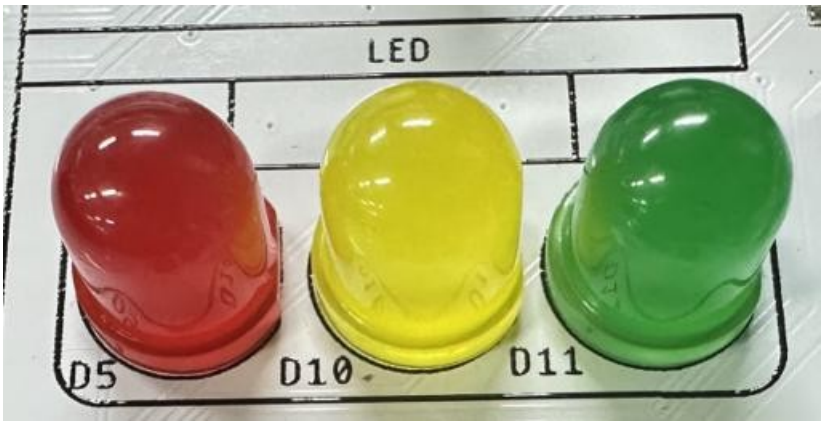
1. Porozumieť programovaciemu rámcu Arduino — funkciám setup() a loop()
2. Naučiť sa používať funkciu pinMode() na konfiguráciu režimov pinov
3. Naučiť sa používať funkciu digitalWrite() na zapnutie LED
4. Naučiť sa používať funkciu delay() na ovládanie blikania LED

Náhľad výsledku



Po nahratí kódu, ako je znázornené na obrázku vyššie, začnú tri LED diódy blikať.

Hardvér použitý v tejto lekcii



Na vývojovej doske Arduino Nano R4 sme nainštalovali tri LED diódy:
Červená LED: pripojená k D5 **Žltá**
LED: pripojená k D10 **Zelená LED:**
pripojená k D11
Všetky tri piny sú na doske označené symbolom „~“, čo znamená, že podporujú výstup PWM (pulzná šírková modulácia). V tejto lekcii však PWM zatiaľ nebudeme používať. Začneme s najzákladnejším ovládaním zapnutia/vypnutia a naučíme sa, ako zapnúť LED diódy a ako ich rozblikať.

Princíp fungovania LED

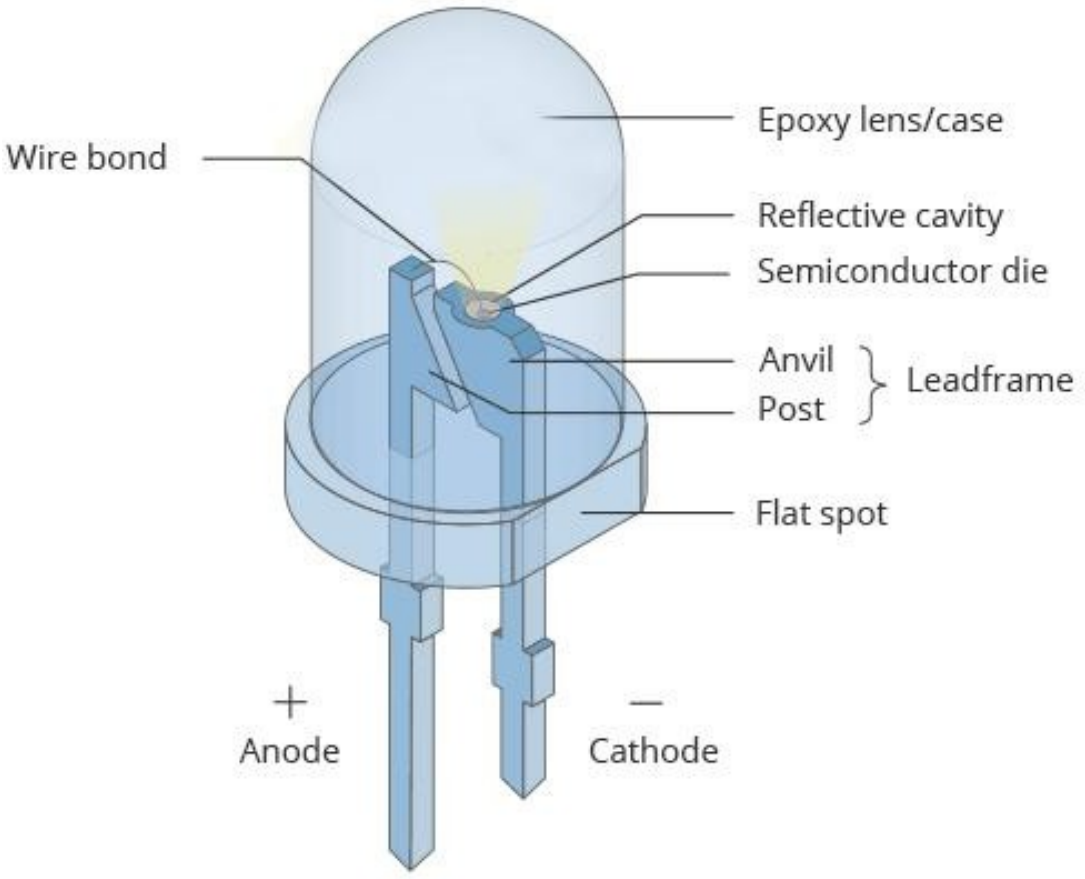
LED je polovodičové zariadenie, ktoré premieňa elektrickú energiu priamo na viditeľné svetlo. LED diódy používané na vývojových doskách Arduino sú zvyčajne vopred zabalené malé žiarovky, vďaka čomu sa dajú ľahko vložiť do experimentálnej dosky alebo pripojiť k pinom.

Zabalená LED dióda má stále dva vývody:

• **Dlhý vodič:** kladný (pripojí sa k výstupnému pinu Arduino Nano R4, napr. D5)

• **Krátky vodič:** záporný (pripojí sa k uzemneniu alebo GND)

Správnym pripojením kladného a záporného vodiča a ovládaním digitálnych pinov na Arduine môžeme dosiahnuť, aby LED svietila alebo blikala.



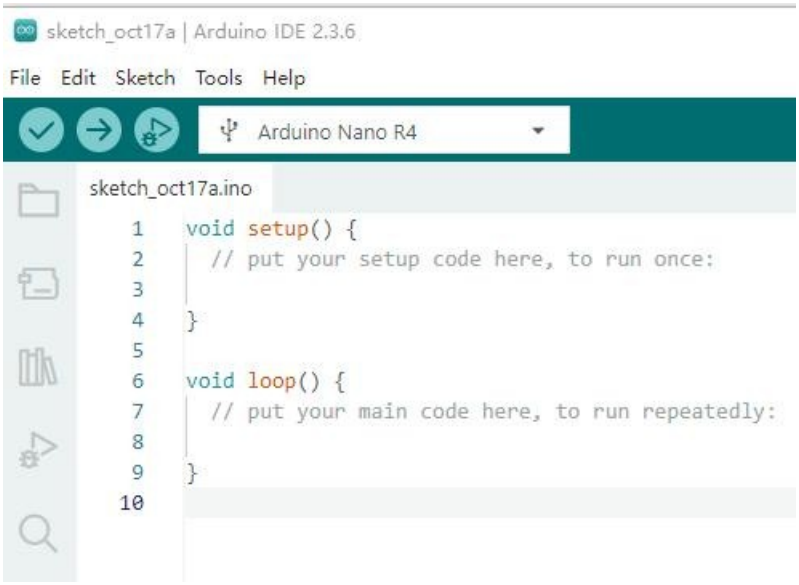
Na vývojovej doske Arduino Nano R4 sú tri LED diódy vopred namontované a pripojené k pinom dosky: D5, D10 a D11.

Blikanie LED

Predtým, ako sa pustíte do kódu, si ho môžete stiahnuť. Tu je odkaz na stiahnutie kompletného kódu: https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/1_LED
Otvorte súbor „1_LED.ino“ nachádzajúci sa v priečinku „1_LED“ pomocou Arduino IDE.

Vysvetlenie kľúčových kódov

V programovaní Arduina má takmer každý program dve základné funkcie:



Funkcia setup()

Táto časť programu sa spustí len raz na začiatku. Zvyčajne sa používa na inicializáciu, napríklad na nastavenie pinov LED ako výstupov, inicializáciu sériovej komunikácie alebo konfiguráciu parametrov senzorov. Ak napríklad chceme rozsvietiť LED, musíme najskôr v funkcii setup() oznámiť Arduinu, že tento pin bude slúžiť ako výstup. Funkciu setup() si predstavte ako „prípravnú fázu“ – spustí sa len raz.

Funkcia loop()

Toto je hlavná slučka programu Arduina, ktorá beží nepretržite. Všetka hlavná logika – napríklad zapínanie LED diód, čítanie senzorov alebo ovládanie bzučiakov – je umiestnená tu. Opakovaným vykonávaním môže Arduino nepretržite ovládať externé zariadenia, čo umožňuje funkcie ako blikanie, monitorovanie alebo reagovanie na akcie. Funkcia loop() je „pracovná fáza“, ktorá beží stále dookola.
Porozumením týchto dvoch funkcií môžeme ovládať rôzny hardvér na Arduine a zabezpečiť, aby fungoval podľa logiky nášho programu.

Poznámka: V kóde Arduina je všetko, čo nasleduje za //, komentárom a program to nevykoná. Komentáre sa používajú na vysvetlenie kódu alebo na zanechanie poznámok pre seba a ostatných, čím sa program stáva ľahšie zrozumiteľným.

Teraz prejdime k príkladu: experiment s blikajúcou LED diódou.

Najskôr musíme definovať tri LED diódy použité v tomto experimente:

```
#define LED_RED 5          // Definujte pin červenej LED ako digitálny pin 5
#define LED_YELLOW 10      // Definujte pin žltej LED ako digitálny pin 10
#define LED_GREEN 11       // Definujte pin zelenej LED ako digitálny pin 11
```

#define je direktíva preprocesora C/C++. Nariaďuje kompilátoru, aby pred kompiláciou kódu nahradil každý výskyt názvu makra jeho zodpovedajúcou hodnotou. Inými slovami, vykonáva nahradenie textu, nie vytvorenie premennej.

V kóde vyššie je „LED_RED“ nahradené číslom 5, „LED_YELLOW“ číslom 10 a „LED_GREEN“ číslom 11. Tieto čísla zodpovedajú pinom pripojeným k červenej, žltej a zelenej LED dióde na vývojovej doske. Vďaka použitiu názvov makier nemusíme čísla pinov zapisovať priamo do program – stačí použiť makro na ovládanie každej LED. Tento prístup má niekoľko výhod:

1. **Lepšia čitateľnosť:** Keď uvidíte LED_RED, okamžite viete, že sa to týka červenej LED, namiesto náhodného čísla, ako je 5.
2. **Ľahšia údržba** ： Ak sa v budúcnosti zmení pin, stačí aktualizovať definíciu makra namiesto toho, aby ste museli kód upravovať všade.
3. **Menej chýb:** Používanie jednotných názvov makier pomáha predchádzať chybám, ako je napríklad zadaní nesprávneho čísla pinu, čo môže spôsobiť nesprávne správanie LED.

Ďalej definujeme časovú premennú s názvom „interval“.

```
int interval = 1000;       // Nastavte interval blikania na 1000 milisekúnd (1 sekunda)
```

Tu sa int používa na definovanie celočíselnej premennej, ktorá ukladá celé čísla. Premenná interval sa používa na ovládanie intervalu blikania LED, meraného v milisekundách. Hodnota 1000 znamená 1000 milisekúnd.

Prečo ukladáme interval blikania do premennej? Z rovnakých dôvodov, ako bolo spomenuté skôr – lepšia čitateľnosť a jednoduchšia údržba. Úpravou hodnoty intervalu môžete ľahko zmeniť rýchlosť blikania LED bez úpravy viacerých riadkov kódu.

Inicializačná funkcia

```
void setup() {
  pinMode(LED_RED, OUTPUT);            // Nastavte pin červenej LED ako
výstup
  pinMode(LED_YELLOW, OUTPUT); // Nastavte pin žltej LED ako výstup
  pinMode(LED_GREEN, OUTPUT);          // Nastavte pin zelenej LED ako výstup
}
```

Už sme spomínali, že **funkcia setup()** sa spustí len raz na začiatku programu a slúži hlavne na inicializáciu. Tu používame pinMode() na nastavenie pinov pre červenú, žltú a zelenú LED diódu do výstupného režimu (OUTPUT). Tým sa doske oznámi, že tieto piny budú vysielať elektrické signály na ovládanie zapnutia alebo vypnutia LED diód. Po správnom nakonfigurovaní pinov môžeme neskôr použiť digitalWrite() vo vnútri funkcie loop() na zapnutie alebo vypnutie každej LED diódy.

Funkcia pinMode() definuje, či sa pin používa ako vstup alebo výstup. Ak sa pokúsite ovládať LED diodu pomocou digitalWrite() bez toho, aby ste najprv zavolali pinMode(), LED dióda nebude fungovať správne.

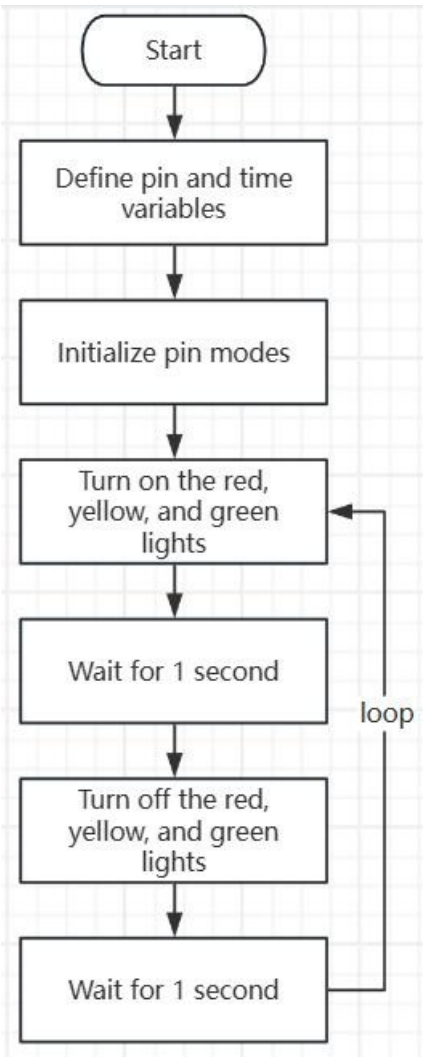
Funkcia loop()

```
void loop() {
  digitalWrite(LED_RED, HIGH);          // Zapnúť červenú LED
  digitalWrite(LED_YELLOW, HIGH); // Zapnúť žltú LED
  digitalWrite(LED_GREEN, HIGH);   // Zapnúť zelenú LED delay(interval);      //
  Počkaj na definovaný interval (1 sekunda)

  digitalWrite(LED_RED, LOW);           // Vypnúť červenú LED
  digitalWrite(LED_YELLOW, LOW);        // Vypni žltú LED
  digitalWrite(LED_GREEN, LOW);   // Vypnúť zelenú LED delay(interval);      //
  Počkať na definovaný interval (1 sekunda)
}
```

Funkcia loop() predstavuje hlavnú slučku programu pre Arduino. Po dokončení funkcie setup() program opakovane vykonáva kód vnútri funkcie loop(). Pomocou príkazu digitalWrite(LED_RED, HIGH) nastavíme pin na vysokú úroveň napätia a pomocou digitalWrite(LED_RED, LOW) na nízku úroveň napätia, čím sa LED rozsvieti alebo zhasne.

Funkcia delay() sa používa na pozastavenie programu. Keď program dosiahne napríklad delay(1000), pozastaví sa na 1000 milisekúnd (1 sekundu), než pokračuje ďalej.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

void setup()	Inicializačná funkcia: Spustí sa len raz, zvyčajne sa používa na konfiguráciu režimov pinov a na vykonanie ďalších úloh súvisiacich s nastavením.
void loop()	Funkcia cyklu: Neustále opakuje základnú funkcionalitu programu.
pinMode()	Nastavenie režimov pinov: Nastavenie pinov ako vstupov alebo výstupov.
digitalWrite()	Zapisuje hodnoty high/low do pinov: Ovláda stavy zapnutia/vypnutia hardvéru.
delay()	Funkcia oneskorenia: Pozastaví program na určitý čas v milisekundách.

Lekcia 02 --- Zvukový senzor

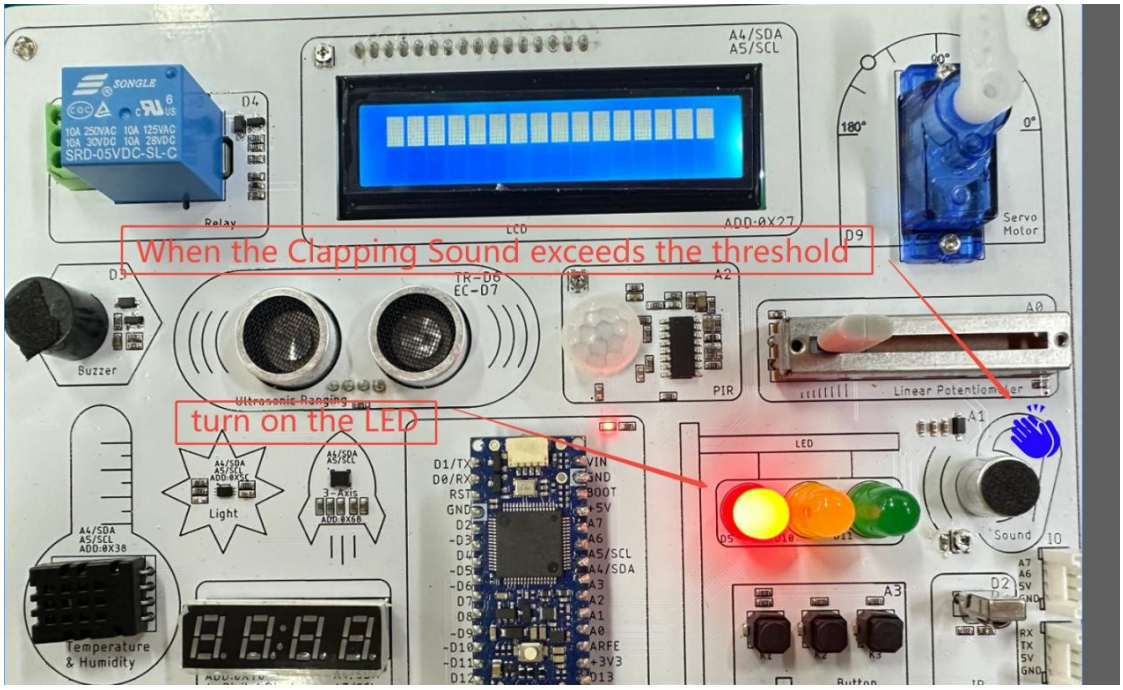
Úvod

V tejto lekcii sa naučíme, ako z čítať analógové hodnoty zo zvukového senzora a ako pomocou detekcie prahovej hodnoty vytvoriť „osvetlenie chodby aktivované zvukom“. Keď sa v okolí zaznamená zvuk, LED dióda sa rozsvieti; keď je ticho, LED dióda sa automaticky zhasne. Prostredníctvom tohto experimentu sa zoznámite s čítaním analógových signálov a naučíte sa, ako vykonávať jednoduché logické riadenie na základe údajov zo senzora.

Ciele výučby

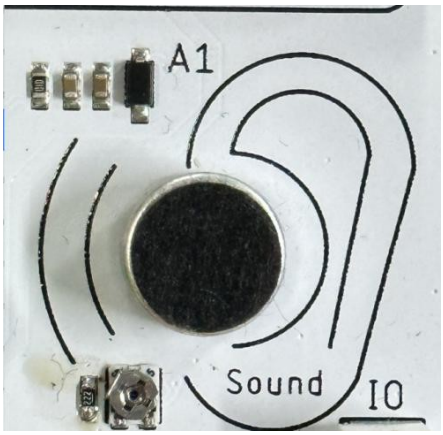
1. Vlastnosti analógového výstupu zo zvukového senzora
2. Naučiť sa, kedy používať analógové piny a kedy digitálne piny
3. Ovládnuť funkciu „analogRead()“ na čítanie analógových hodnôt zo senzora
4. Naučte sa používať sériový monitor na ladenie
5. Naučte sa používať príkazy if na logické rozhodnutia
6. Vykonajte experiment s analógovým osvetlením chodby aktivovaným zvukom

Náhľad výsledku



Po nahratí kódu, keď zvukový senzor zistí analógovú hodnotu vyššiu ako prah nastavený v kóde, červená LED dióda sa rozsvieti na 3 sekundy a potom zhasne. Naopak, ak je hodnota nižšia ako prah, LED dióda sa nerozsvieti. Ako je znázornené na obrázku, po zapnutí napájania potlesk spustí rozsvietenie LED diódy na 3 sekundy.

Hardvér použitý v tejto lekcii



Na vývojovej doske Arduino Nano R4 je zvukový senzor umiestnený na pravej strane a je pripojený k hlavnej doske cez pin A1. A1 je analógový vstupný pin, ktorý sa používa na čítanie nepretržite sa meniacich analógových napäťových hodnôt, na rozdiel od digitálnych pinov (D pinov), ktoré dokážu čítať len vysoké alebo nízke úrovne.

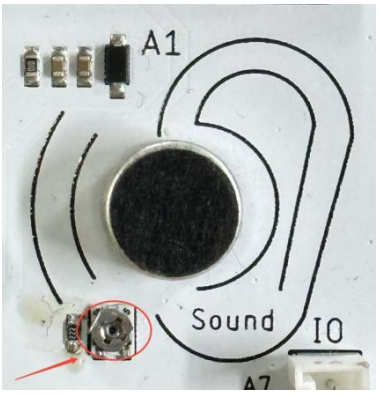
Kedy zvoliť analógový pin a kedy digitálny pin:

• Ak je možné detekovať len dva stavy, napríklad v prípade tlačidlového modulu, použite digitálny pin. Digitálne piny musia detekovať len vysoké alebo nízke napäťové úrovne.

• Ak potrebujete získať neustále sa meniace údaje, napríklad intenzitu zvuku alebo svetla, použite analógový pin. Analógové piny dokážu čítať neustále sa meniace napätia, čím poskytujú presnejšie numerické informácie.

V ľavom dolnom rohu zvukového senzora sa nachádza gombík na nastavenie citlivosti. Otáčaním v smere hodinových ručičiek sa zvyšuje citlivosť senzora, takže aj malé zvuky budú generovať vyššie hodnoty. Otáčaním proti smeru hodinových ručičiek sa citlivosť znižuje, čo vyžaduje hlasnejšie zvuky na spustenie výstupu.

Poznámka: Pri nastavovaní neotáčajte gombíkom nad jeho limit, pretože by mohlo dôjsť k poškodeniu vnútorného mechanizmu.



Princíp fungovania zvukového modulu

Zvukový senzor je malá elektronická súčiastka, ktorá slúži na „počúvanie“ zvukov. Jeho jadrom je malý mikrofón, ktorý premieňa okolité zvuky na slabé elektrické signály. Tieto signály sú potom zosilnené a spracované obvodom, ktorý nakoniec vydáva analógové napätie. Arduino číta

túto analógovú hodnotu a určuje úroveň zvuku v prostredí – čím je zvuk hlasnejší, tým vyššie výstupné napätie; čím je zvuk tichší, tým je nižšie výstupné napätie.

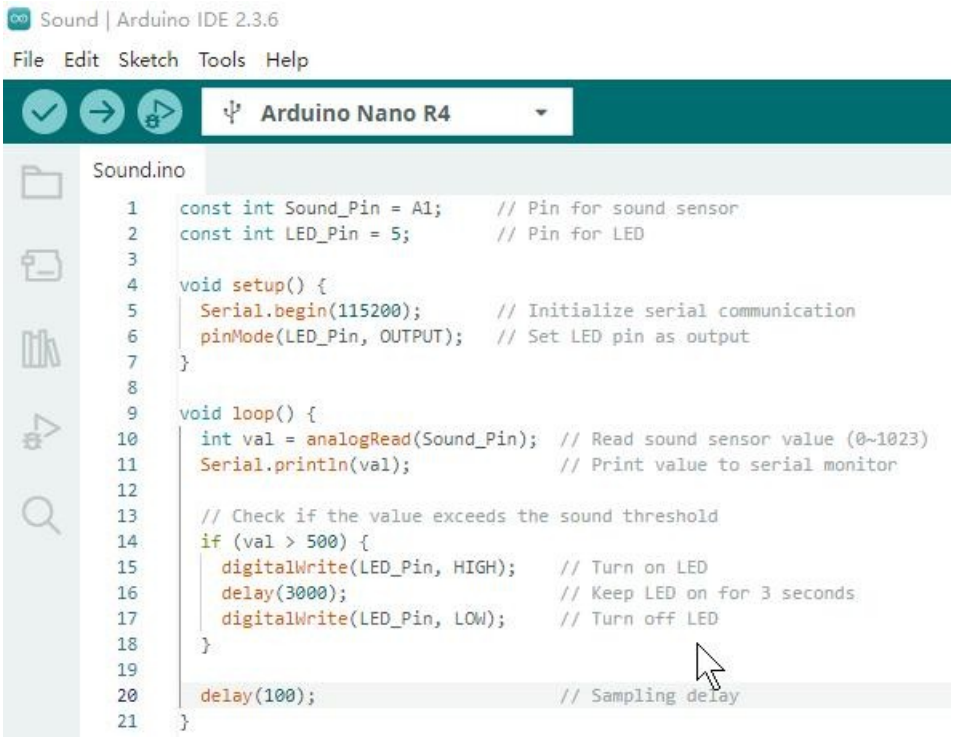
Simulované osvetlenie chodby aktivované zvukom

Môžete si ho stiahnuť. Tu je odkaz na stiahnutie kompletného kódu: https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_c ode/2_Sound

Otvorte súbor „2_Sound.ino“ nachádzajúci sa v priečinku „2_Sound“ pomocou prostredia Arduino IDE.

Vysvetlenie kódov klávesov

Teraz si prejdime príklad: analógové osvetlenie chodby aktivované zvukom.



Najskôr začneme definovaním pinov, ktoré sa budú používať

```
const int Sound_Pin = A1;          // Pin pre zvukový senzor
const int LED_Pin = 5;             // Pin pre LED
```

V Arduine (C/C++) sa **kľúčové slovo const** používa na to, aby bola premenná len na čítanie, čo znamená, že jej hodnota je pevne stanovená už pri kompilácii a počas behu programu ju nie je možné zmeniť. Tým sa zvyšuje bezpečnosť kódu, pretože sa zabráni náhodným zmenám dôležitých hodnôt, ako sú napríklad čísla pinov. Odstránenie kľúčového slova const nespôsobí zlyhanie vášho kódu.

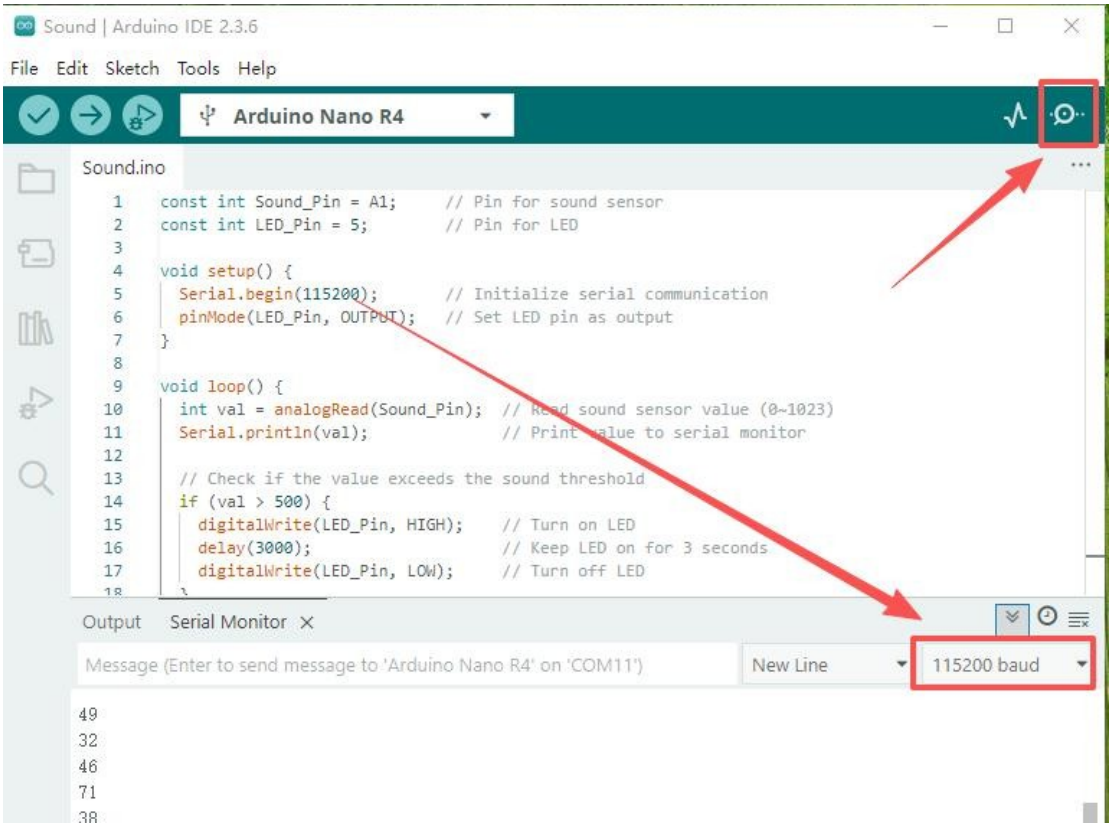
int sa používa na definovanie celočíselnej premennej. Možno sa čudujete, prečo môžeme použiť int na definovanie „A1“, keďže to nevyzerá ako číslo. Dôvodom je, že v kompilátore Arduina je „A1“ v skutočnosti celočíselná konštanta. Názvy A0, A1 atď. sa používajú jednoducho preto, aby vývojári mohli ľahšie identifikovať a zapamätať si analógové vstupné piny.

Inicializačná funkcia

```
void setup() {
  Serial.begin(115200);          // Inicializácia sériovej komunikácie
  pinMode(LED_Pin, OUTPUT);      // Nastavenie pinu LED ako výstup
}
```

V inicializačnej funkcii zavádzame nový pojem: Serial.begin. Týmto sa inicializuje sériová komunikácia Arduina a nastaví prenosová rýchlosť na 115 200. Upozorňujeme, že tu nastavená prenosová rýchlosť sa musí zhodovať s rýchlosťou zvolenou v sériovom monitore; v opačnom prípade sa môže zobraziť skreslený výstup.

Ako otvoriť sériový monitor:



Kliknite na ikonu sériového monitora v pravom hornom rohu. Po otvorení vyberte v pravom dolnom rohu rovnakú prenosovú rýchlosť, aká je nastavená vo vašom kóde.

Prečítajte analógovú hodnotu senzora a urobte rozhodnutie

```
void loop() {
  int val = analogRead(Sound_Pin); // Načítanie hodnoty zvukového senzora
  (0~1023) Serial.println(val);          // Vypíšte hodnotu na sériový monitor
```

Vo funkcii loop() je naša hlavná logika nasledovná: najprv prečítame analógovú hodnotu zo zvukového senzora a potom ju vytlačíme na sériový monitor. Pozorovaním týchto hodnôt môžeme určiť intenzitu zvuku a nastaviť vhodnú prahovú hodnotu, aby sme rozhodli, pri akej úrovni zvuku sa má LED rozsvietiť, čím dosiahneme funkciu osvetlenia aktivovaného zvukom.

Ako bolo spomenuté skôr, **int** sa používa na definovanie celočíselných premenných

Funkcia **analogRead()** sa používa špeciálne na čítanie hodnoty napätia z analógového vstupného pinu. Keď senzor vydáva viac ako dve úrovne variácií – napríklad intenzitu zvuku, jas svetla alebo teplotu – môžete ho pripojiť k analógovému pinu a použiť analogRead(), aby ste získali
nepretržite sa meniace hodnoty. To umožňuje programu robiť presnejšie posúdenia a riadiť rozhodnutia.

Príklad: podmienený príkaz if

```
if (soundValue > threshold) {
        digitalWrite(ledPin, HIGH);
}
```

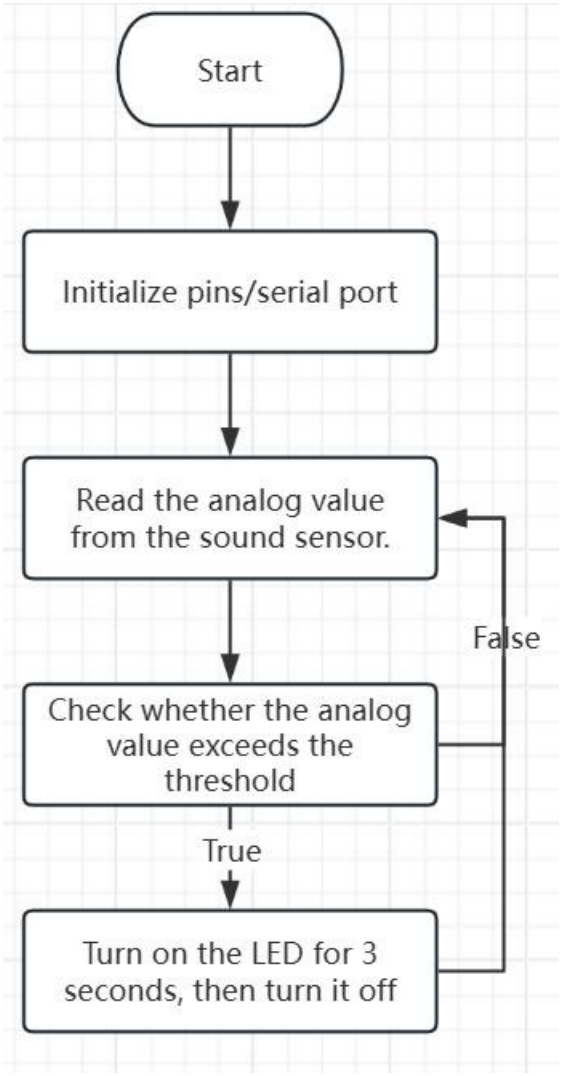
Kód vnútri zložených zátvoriek sa vykoná len vtedy, ak je podmienka v zátvorkách (soundValue > threshold) pravdivá, čím sa zapne LED. Ak je podmienka nepravdivá, kód vnútri zložených zátvoriek sa nevykoná a LED zostane vypnutá.

Podmienený príkaz if v tomto príklade

```
  // Skontroluj, či hodnota prekračuje prah zvuku if (val
  > 500) {
    digitalWrite(LED_Pin, HIGH);          // Zapni LED delay(3000);
                                          // Nechaj LED svietiť 3 sekundy
    digitalWrite(LED_Pin, LOW);           // Vypni LED
  }
  delay(100);                             // Oneskorenie vzorkovania
}
```

Keď prečítame analógovú hodnotu zo senzora, porovnáme ju s hodnotou 500. Ak prekročí túto hranicu, vykonáme digitalWrite(LED_Pin, HIGH);, aby sme zapli LED. Na simuláciu svetla v chodbe aktivovaného zvukom použijeme funkciu delay(), aby LED svietila 3 sekundy, a potom vykonáme digitalWrite(LED_Pin, LOW);, aby sme ju vypli. Konečné delay(100) spôsobí pauzu na 0,1 sekundy pred opätovným načítaním analógovej hodnoty; toto možno ponechať alebo odstrániť podľa potreby.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

Serial.begin	Inicializácia prenosovej rýchlosti sériového portu
analogRead	Čítanie analógových hodnôt, zvyčajne používané pre senzory s viacerými stavmi
if	podmienené príkazy if, používané v prípade, že program potrebuje vykonať akcie v závislosti od toho, či je splnená podmienka

Lekcia 03 – Senzor PIR

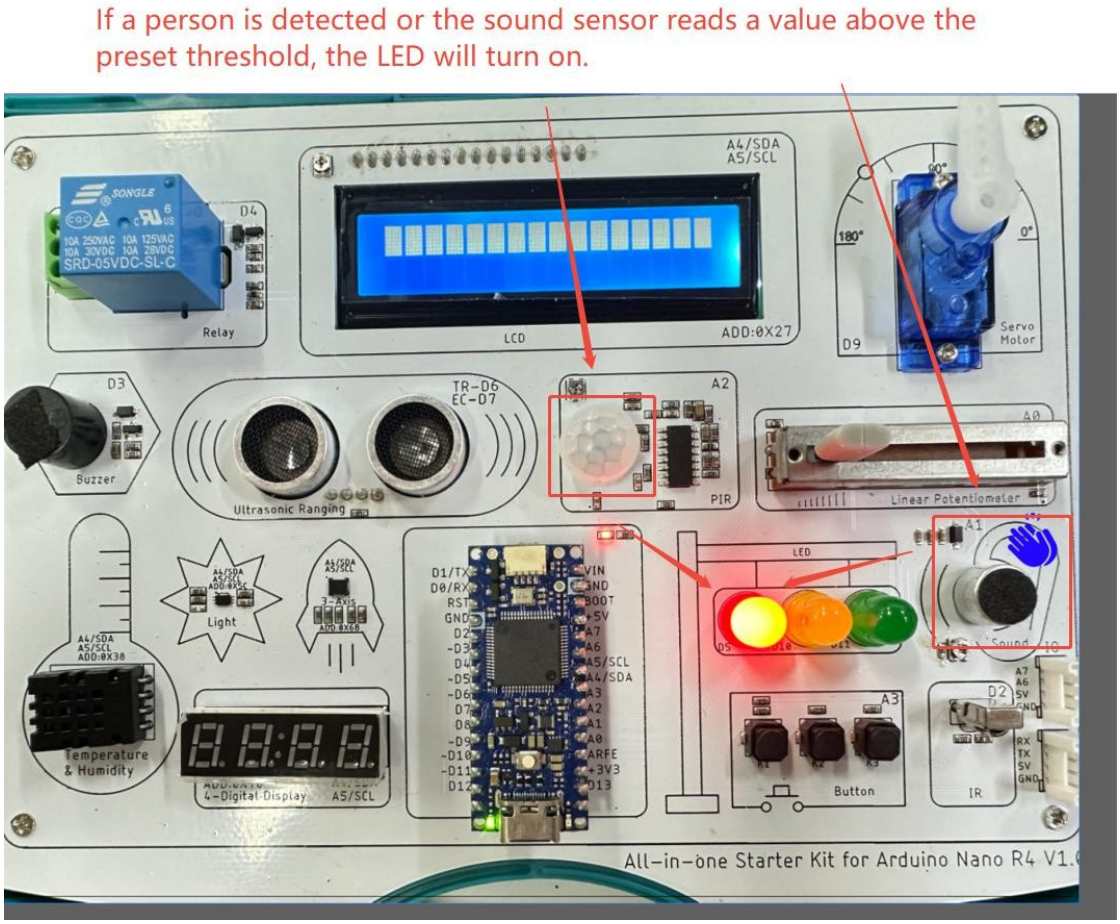
Úvod

V tejto lekcii sa naučíme jeden z najčastejšie používaných podmienených operátorov v programovaní Arduina – logický operátor OR ||. Jeho účelom je umožniť, aby viacero podmienok spustilo ďalšiu časť programu, pokiaľ je aspoň jedna z nich pravdivá. Po osvojení si tohto konceptu budete schopní napísať riadiacu logiku, ktorá reaguje na viacero podmienok, a zostaviť inteligentné osvetlenie chodby, ktoré prijíma rozhodnutia pomocou viacerých senzorov súčasne.

Ciele

1. Porozumieť fungovaniu PIR senzora
2. Ovládať logický operátor OR „||“
3. Naučiť sa čítať úrovne napätia pomocou analógového pinu
4. Dokončiť príklad simulácie inteligentného osvetlenia chodby s viacerými senzormi

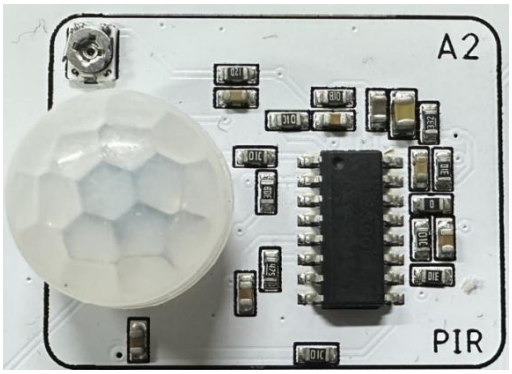
Náhľad výsledku

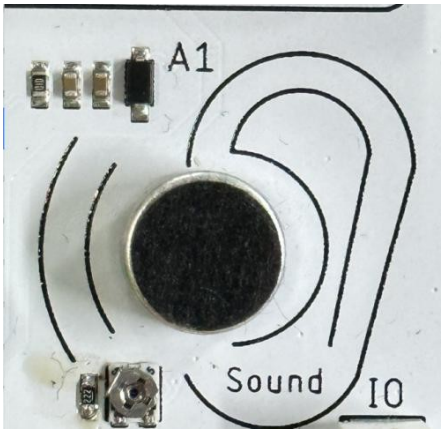


Po nahratí kódu, ak zvukový senzor zaznamená analógovú hodnotu vyššiu ako prahová hodnota

nastavenú v programe alebo PIR senzor zistí prítomnosť osoby, červená LED dióda sa rozsvieti na 3 sekundy a potom zhasne. V tomto simulovanom projekte inteligentného osvetlenia chodby sa môže stať, že ak osoba vydá len veľmi tichý zvuk, svetlo sa nemusí rozsvietiť včas. Preto pridávame PIR senzor na detekciu prítomnosti človeka. Pokiaľ je splnená jedna z týchto dvoch podmienok, LED dióda sa rozsvieti a poskytne osvetlenie.

Hardvér použitý v tejto lekcii

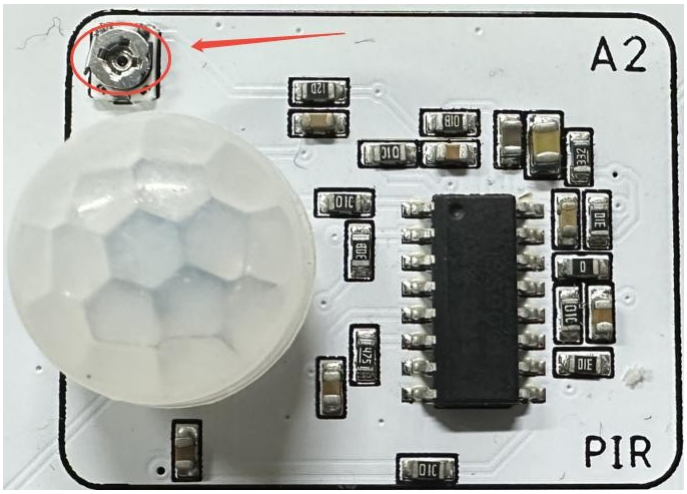




Senzor PIR sa nachádza v strednej časti vývojovej dosky a je pripojený k analógovému vstupnému pinu A2. Je dôležité poznamenať, že analógové piny dokážu čítať ako digitálne úrovne (HIGH/LOW), tak aj analógové hodnoty, zatiaľ čo digitálne piny dokážu čítať len HIGH alebo LOW a nedokážu čítať analógové hodnoty. Je to preto, lebo analógové piny obsahujú analógovo-digitálny prevodník (ADC), ktorý

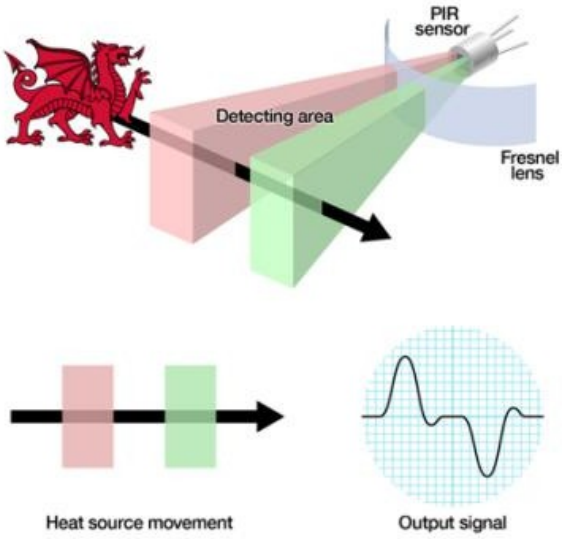
umožňuje funkcii analogRead() čítať analógové hodnoty v rozsahu od 0 do 1023.

V ľavom hornom rohu senzora PIR sa nachádza gombík na nastavenie citlivosti. Otáčaním v smere hodinových ručičiek sa citlivosť senzora zvyšuje, zatiaľ čo otáčaním proti smeru hodinových ručičiek sa citlivosť znižuje. Vyhnite sa prílišnému otáčaniu proti smeru hodinových ručičiek, inak senzor nemusí detekovať osoby, aj keď sú prítomné.
Poznámka: Neotáčajte gombíkom za jeho limit. Mohlo by to poškodiť vnútorný nastavovací mechanizmus.



Princíp fungovania modulu PIR

PIR snímač pohybu dokáže detekovať zmenu množstva infračerveného žiarenia odrážajúceho sa na ňom, čo závisí od teploty a povrchových vlastností objektu pred snímačom. Keď sa objekt, napríklad osoba, pohybuje v detekčnom rozsahu PIR snímača, teplota v danom bode v zornom poli snímača stúpne z izbovej teploty na telesnú teplotu a táto zmena sa vráti k snímaču a bude detekovaná.

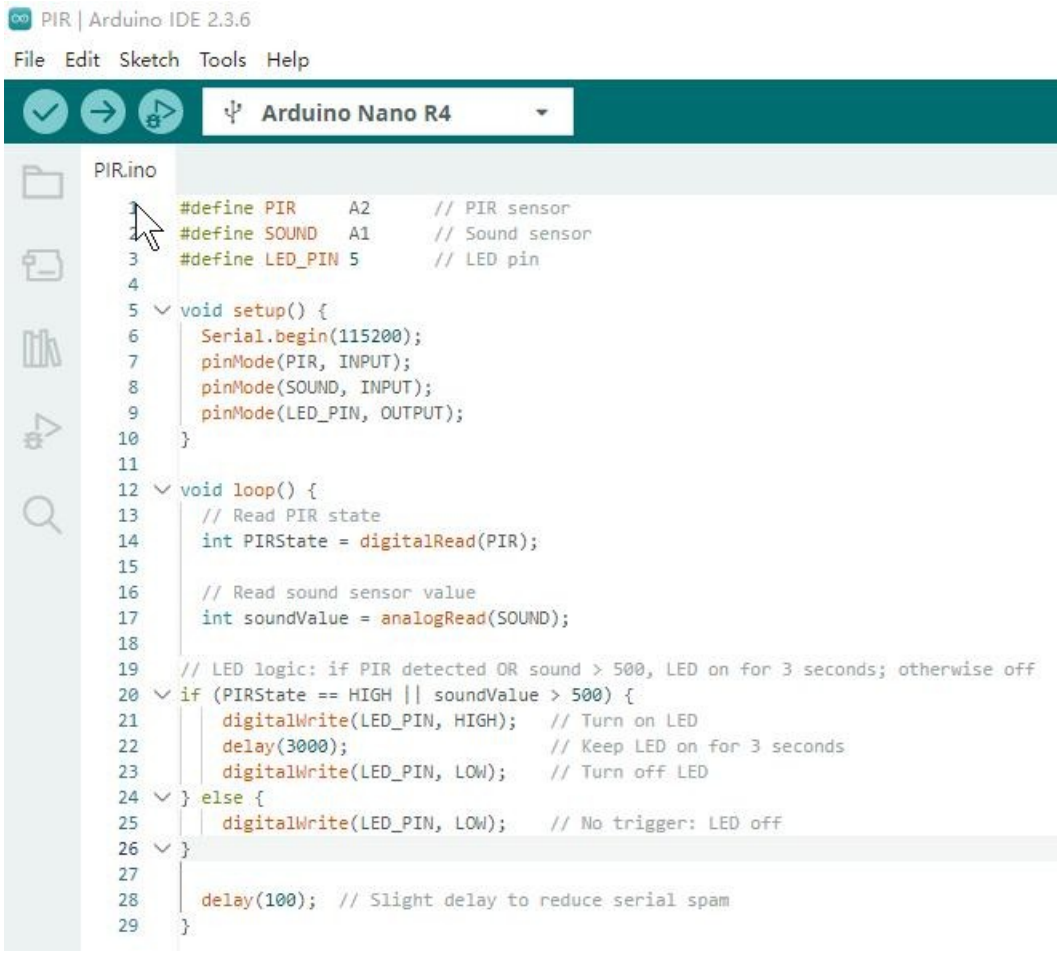


Inteligentné osvetlenie chodby

Predtým, ako sa pustíte do kódu, si ho môžete stiahnuť. Tu je odkaz na stiahnutie kompletného kódu:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/3_PIR
Otvorte súbor „3_PIR.ino“ v priečinku „3_PIR“ pomocou Arduino IDE.

Vysvetlenie kódu

Teraz si prejdime príklad: Inteligentné osvetlenie chodby aktivované zvukom



Najskôr opäť definujeme piny, ktoré potrebujeme použiť:

```
#define PIR A2 // Senzor PIR
#define SOUND A1 // Zvukový
senzor #define LED_PIN 5 // Pin LED
```

Tu definujeme tri piny:

- Senzor PIR je pripojený k analógovému pinu A2, ale keďže senzor PIR má len dva stavy (zistená prítomnosť osoby alebo nie), môžeme na zistenie jeho úrovne HIGH alebo LOW jednoducho použiť funkciu digitalRead.
- Zvukový senzor je pripojený k analógovému pinu A1 a ako sme sa naučili v predchádzajúcej lekcii,

na získanie jeho analógovej hodnoty používame funkciu analogRead.

• LED je pripojená k D5 a zapíname ju zápisom HIGH a vypíname zápisom LOW.

Inicializačná funkcia

```
void setup() {
  pinMode(PIR, INPUT);
  pinMode(SOUND, INPUT);
  pinMode(LED_PIN, OUTPUT);
}
```

V inicializačnej funkcii nakonfigurujeme tri piny do príslušných režimov. Senzory sú nastavené do vstupného režimu, zatiaľ čo akčný člen (LED) je nastavený do výstupného režimu.

Vnútri funkcie loop()

```
void loop() {
  // Načítanie stavu PIR
  int PIRState = digitalRead(PIR);

  // Načítanie hodnoty zvukového senzora
  int soundValue = analogRead(SOUND);
```

V rámci funkcie loop musíme pri vyhodnocovaní podmienok najprv načítať hodnoty vrátené senzorom, aby sme určili aktuálny stav. Tieto kľúčové body sme vysvetlili už skôr, takže sa k nim nebudeme vracať.

Kľúčový kód podmieneného posúdenia

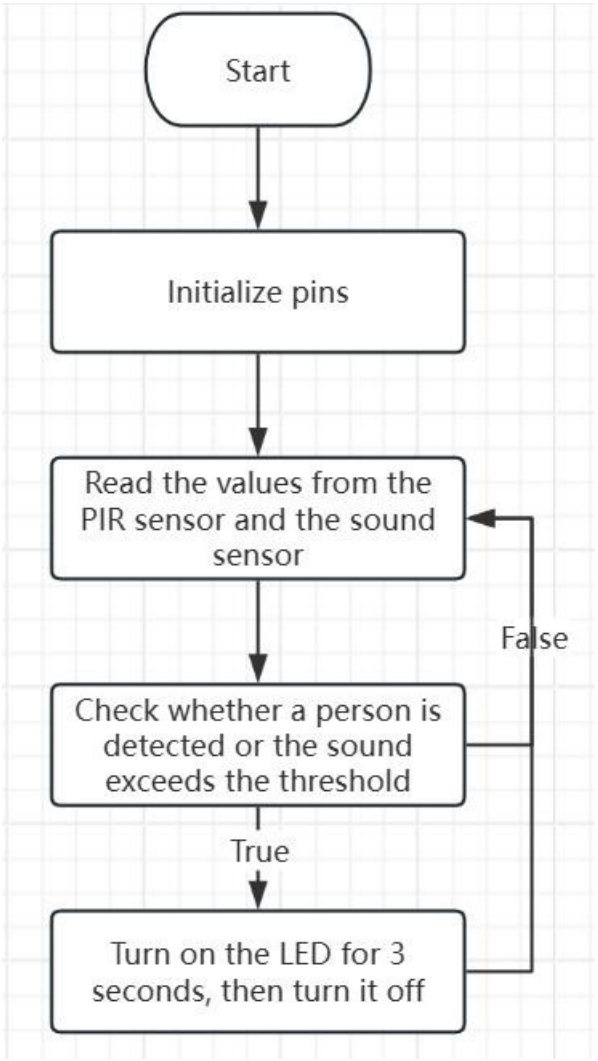
```
// Logika LED: ak PIR zaznamená pohyb ALEBO hladina zvuku je vyššia ako 500, LED svieti
3 sekundy; inak nesvieti
if (PIRState == HIGH || soundValue > 500) {
    digitalWrite(LED_PIN, HIGH);      // Zapnúť LED
    delay(3000);                      // Nechaj LED svietiť 3 sekundy
    digitalWrite(LED_PIN, LOW);       // Vypnúť LED
} inak {
    digitalWrite(LED_PIN, LOW);       // Žiadny spúšťač: LED
}                                     vypnutá

  delay(100); // Krátke oneskorenie na zníženie
} sériového spamu
```

Symbol „ | | “ je logický operátor OR, ktorý si môžete predstaviť ako „alebo“ – Ak je splnená aspoň jedna z týchto dvoch podmienok, celá podmienka je splnená.

• „PIRState == HIGH || soundValue > 500“: Ak senzor PIR zistí úroveň HIGH (niekto je prítomný) alebo ak senzor zvuku zistí hodnotu väčšiu ako 500, LED sa rozsvieti.

• Vykonávací kód tu nebude podrobne vysvetlený. Jeho hlavnou funkciou je nastaviť LED na HIGH na 3 sekundy a potom ju vrátiť späť na LOW. Ak sa podmienka spustí, LED sa rozsvieti na 3 sekundy a potom zhasne.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

||	Logický operátor OR považuje celú podmienku za splnenú, pokiaľ je splnená aspoň jedna z viacerých podmienok.

Lekcia 04 --- Bzučiak

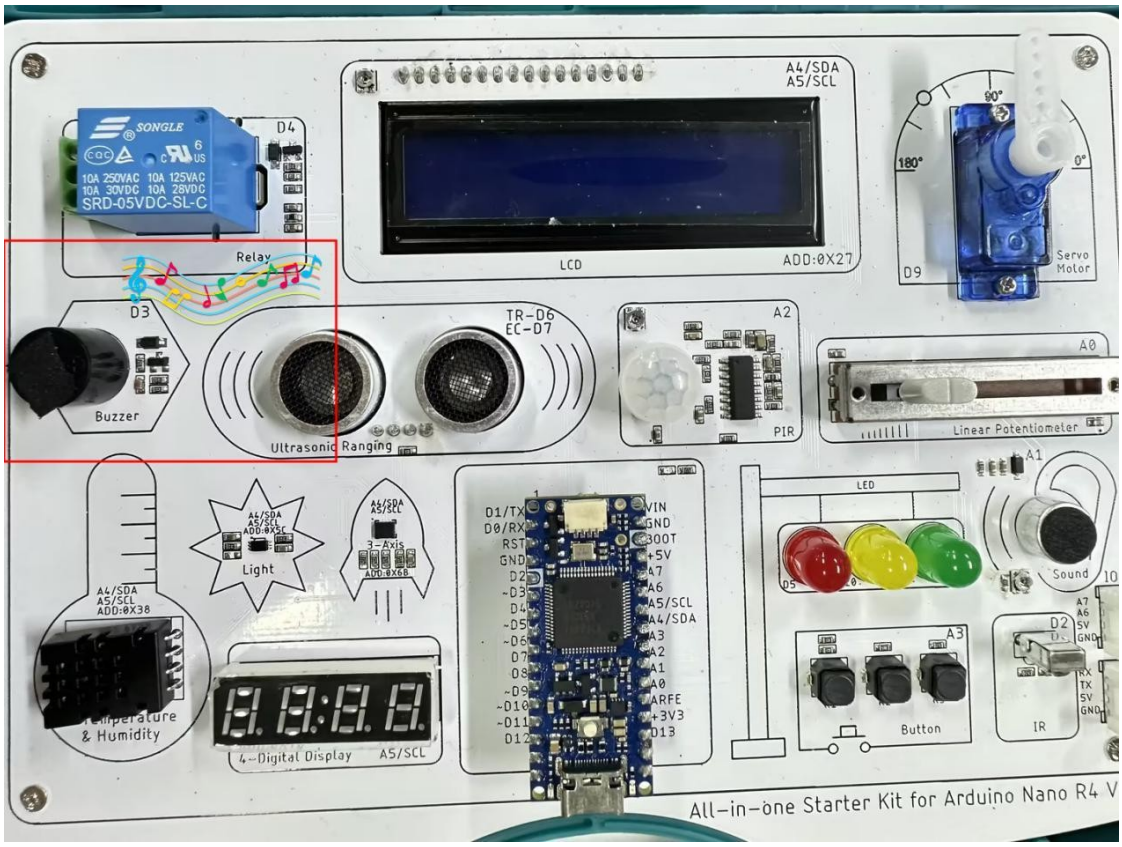
Úvod

V tejto lekcii sa naučíme, ako ovládať bzučiak tak, aby hral rôzne melódie. Naučíte sa zistiť, čo je pasívny bzučiak, a pomocou cyklov for a funkcií prehrávať sedem základných tónov. Na konci tejto lekcie budete vedieť, ako definovať funkcie, zjednodušiť kód pomocou cyklov for a meniť výšku tónu úpravou rôznych frekvencií vstupného signálu.

Ciele výučby

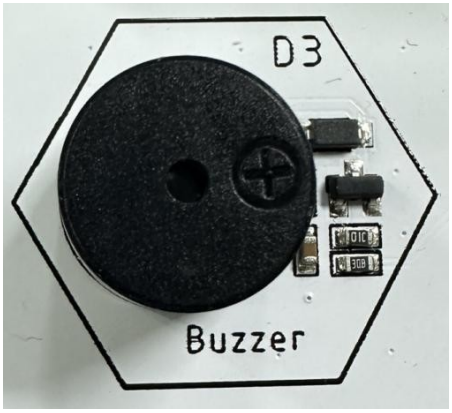
1. Pochopiť, čo je pasívny bzučiak
2. Ovládať používanie polí
3. Ovládať používanie cyklov for
4. Dokončiť príklad prehrávania siedmich základných tónov

Náhľad výsledku



Po nahratí kódu bude bzučiak v slučke prehrávať sedem štandardných tónov.

Hardvér použitý v tejto lekcii



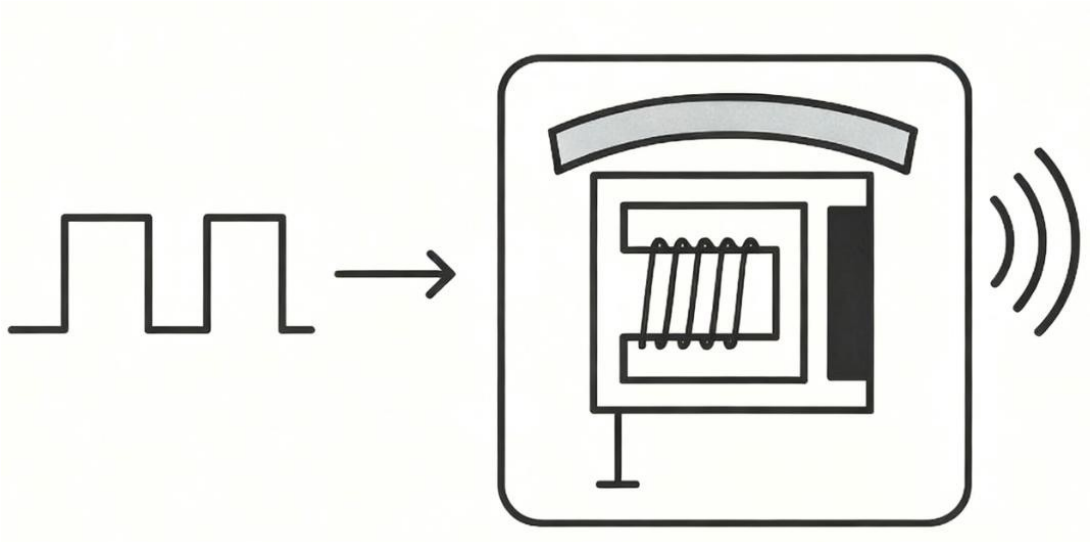
Modul bzučiaka je výstupné zariadenie umiestnené na ľavej strane vývojovej dosky Arduino Nano R4：
Bzučiaky možno rozdeliť do dvoch typov:

1. Aktívny bzučiak: Je jednoduchý na používanie, na zapnutie alebo vypnutie vyžaduje iba signály HIGH alebo LOW. Frekvenciu tónu nie je možné nastaviť.
2. Pasívny bzučiak: Jeho používanie je zložitejšie, frekvenciu zvuku je možné nastaviť. Zvyčajne sa využíva vstavaná funkcia tone(), do ktorej zadáte parametre na nastavenie požadovanej frekvencie

V tejto lekcii používame pasívny bzučiak, ktorý je možné naprogramovať tak, aby prehrával rôzne melódie

Princíp fungovania bzučiaka

Ako je znázornené na obrázku, signál s obdĺžnikovým priebehom na ľavej strane predstavuje riadiaci signál poskytovaný externým obvodom. Tento riadiaci signál sa prenáša prostredníctvom vodičov do elektromagnetickej cievky vnútri pasívneho bzučiaka. Keď cievkou preteká prúd, vytvára magnetické pole.Cievka je zvyčajne pripojená k vibrujúcej membráne. Keď sa mení prúd v cievke, výsledné magnetické pole priťahuje alebo odpudzuje permanentný magnet, čím spôsobuje vibrácie membrány. Keďže riadiaci signál je obdĺžnikový, nastavením frekvencie a pracovného cyklu signálu je možné ovládať frekvenciu vibrácií, čím sa dosiahne nastaviteľná výška tónu.

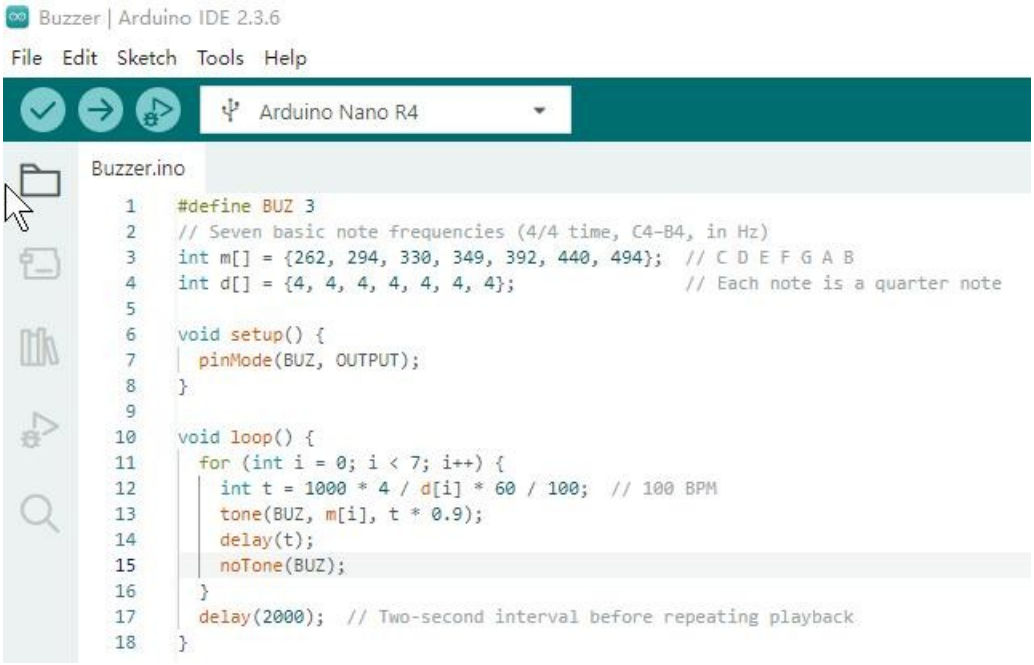


Hranie siedmich základných tónov

Predtým, ako sa pustíte do kódu, si ho môžete stiahnuť. Tu je odkaz na stiahnutie kompletného kódu: https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_c ode/4_Buzzer
Otvorte súbor „4_Buzzer.ino“ v priečinku „4_Buzzer“ pomocou prostredia Arduino IDE.

Vysvetlenie kódov klávesov

Teraz si prejdime príklad: Prehrávanie siedmich základných tónov



Najskôr definujeme piny, ktoré budeme používať:

```
#define BUZ 3
```

V tejto lekcii je bzučiak pripojený k pinu D3 vývojovej dosky Arduino Nano R4. Na pomenovanie pinu používame „#define“, čo uľahčuje neskoršiu úpravu a ladenie kódu.

Ďalej použijeme pole na definovanie frekvenčných hodnôt siedmich základných tónov

```
// Sedem základných frekvencií tónov (4/4 takt, C4–B4, v Hz)
int m[] = {262, 294, 330, 349, 392, 440, 494}; // C D E F G A B
int d[] = {4, 4, 4, 4, 4, 4, 4};                // Každá nota je štvrťová nota
```

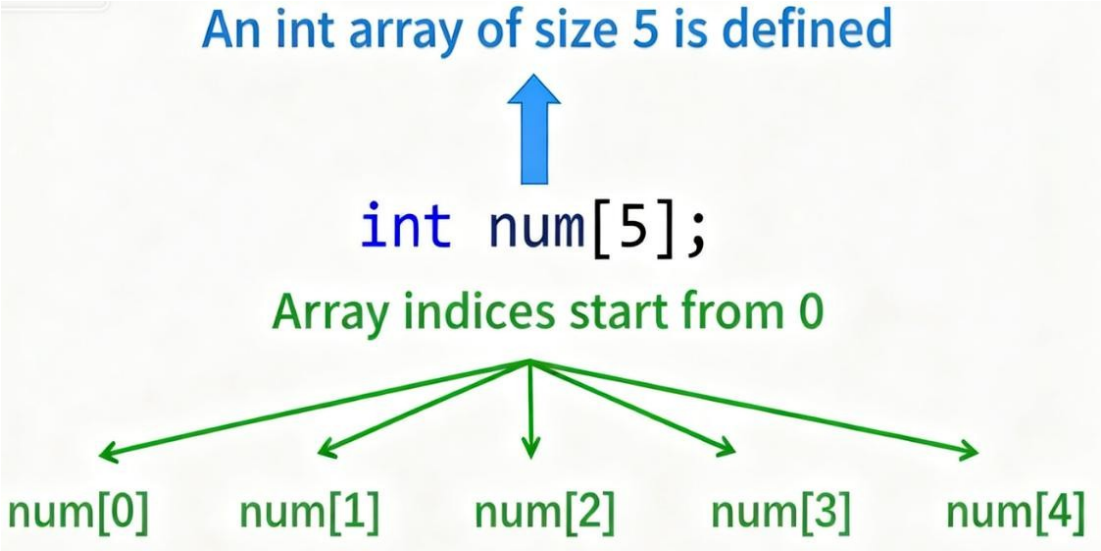
Toto je kľúčový bod tejto lekcie — polia

Existuje mnoho spôsobov, ako definovať polia, ale tu predstavíme dva najčastejšie používané spôsoby:

1. Pri špecifikovaní počtu prvkov

Definujte pole typu int, kde int označuje, že všetky prvky v poli sú celé čísla.

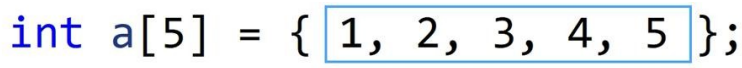
int num[5]:Týmto sa definuje pole celých čísel s názvom num s piatimi prvkami, ale konkrétne hodnoty prvkov ešte neboli priradené.

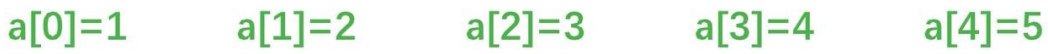


Ako je znázornené na obrázku, prvky poľa „num“ sú num[0], num[1], num[2], num[3] a num[4]

Poznámka: Indexovanie začína od 0, takže „num[0]“ je prvý prvok a „num[1]“ je druhý prvok poľa.

int a[5] = {1,2,3,4,5}:Toto definuje celočíselné pole s názvom „a“ s piatimi prvkami: 1, 2, 3, 4 a 5





Ako je znázornené na obrázku, „a[0]“ je prvý prvok poľa „a“ a má hodnotu 1. Po definovaní celého poľa môžeme na jeho prvky odkazovať priamo pomocou zápisu „Array[]“.

2. Keď je počet prvkov neistý

int a[] = {1,2,3}: Pri definovaní poľa týmto spôsobom sa [] ponechá prázdne a dĺžka poľa sa automaticky určí podľa počtu zadaných prvkov. Táto metóda je užitočná, ak budete neskôr potrebovať pridať ďalšie prvky, pretože nemusíte meniť počiatočnú dĺžku poľa.

V príklade tejto lekcie používame na definovanie polí druhú metódu, čo znamená, že neskôr môžeme priamo pridávať prvky, aby sme zahrnuli ďalšie noty, ktoré chceme prehrať.

Inicializačná funkcia

```
void setup() { pinMode(BUZ,
  OUTPUT);
}
```

V inicializačnej funkcii nastavíme pin bzučiaka do výstupného režimu

Vnútri funkcie loop()

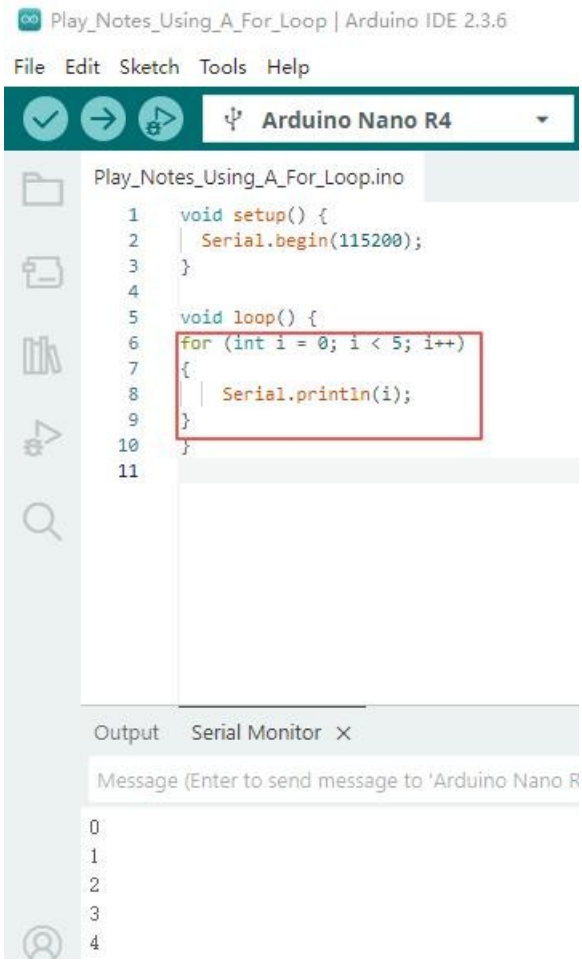
```
void loop() {
  for (int i = 0; i < 7; i++) {
    int t = 1000 * 4 / d[i] * 60 / 100; // 100 BPM
    tone(BUZ, m[i], t * 0.9);
    delay(t); noTone(BUZ);
  }
  delay(2000); // Dvojsekundový interval pred opakovaným prehrávaním
}
```

Vo vnútri funkcie loop() používame cyklus for na zjednodušenie kódu, vďaka čomu program prehrá všetkých sedem štandardných tónov v rámci jediného cyklu for.

Cyklus for

Cyklus for nám umožňuje automaticky a opakovane vykonávať kód určitý počet krát. V kombinácii s poľami nám umožňuje spracovávať viacero prvkov naraz, čo výrazne zjednodušuje kód a robí program efektívnejším a prehľadnejším.

Napríklad:



Rozložme si hlavnú štruktúru cyklu for:

1. Inicializácia: int i = 0

Týmto sa definuje premenná cyklu i a inicializuje sa na hodnotu 0.

2. Podmienka cyklu: i < 5

Pokiaľ je táto podmienka pravdivá, vykoná sa kód vnútri tela cyklu (Serial.println(i)). Keď je podmienka nepravdivá, cyklus sa ukončí

3. Telo cyklu: Serial.println(i)

Pri každom opakovaní sa vypíše aktuálna hodnota premennej i

4. Aktualizácia počítadla: i++

Po každej iterácii sa i automaticky zvýši o 1. Potom sa vráti k kontrole podmienky, aby sa rozhodlo, či pokračovať

Výsledok vykonania: 0, 1, 2, 3, 4

Keď i dosiahne hodnotu 5, podmienka i < 5 je nepravdivá a cyklus sa ukončí

Späť k nášmu kódu funkcie loop():

Potrebujeme vykonať 7 cyklov, pričom index začína na 0. Inicializujeme teda i na 0. Podmienka i < 7 zabezpečuje, že cyklus pokračuje, kým i nedosiahne hodnotu 7. i++ znamená, že pri každej iterácii sa i zvýši o 1. Počas prvej iterácie sa teda cyklus vykonáva tak, ako je znázornené na obrázku nižšie:

```
int t = 1000 * 4 / d[0] * 60 / 100;  // Calculate note duration at 100 BPM
tone(BUZ, m[0], t * 0.9);           // Play the note
delay(t);                            // Wait for the note to finish
noTone(BUZ);
```

Počas druhej iterácie:

```
int t = 1000 * 4 / d[1] * 60 / 100;  // Calculate note duration at 100 BPM
tone(BUZ, m[1], t * 0.9);           // Play the note
delay(t);                            // Wait for the note to finish
noTone(BUZ);
```

Všimnete si, že pri každej iterácii cyklu sa mení iba hodnota indexu podľa premennej „i“, čím sa dosiahne požadovaný efekt.

Vysvetlenie parametrov funkcie „tone()“

Funkcia „tone()“ môže mať tri parametre:

pin: Určuje hardvérový pin pripojený k bzučiaku.

frequency: Určuje frekvenciu tónu, ktorý bude bzučiak hrať. Vyššia frekvencia = vyšší tón, nižšia frekvencia = nižší tón.

duration: Určuje, ako dlho bude bzučiak znieť, v milisekundách. Ak sa tento parameter vynechá, bzučiak bude znieť, kým sa nezavolá „noTone()“.

tone(3,262,500): Bzučiak na pine 3 prehráva frekvenciu 262 Hz po dobu 500 milisekúnd. Tu číslo 262 zodpovedá tónu „C“ na klavíri.

Vysvetlenie funkcie „noTone()“

Funkcia „noTone()“ prijíma len jeden parameter, číslo pinu, a zastaví bzučiak na tomto pine.

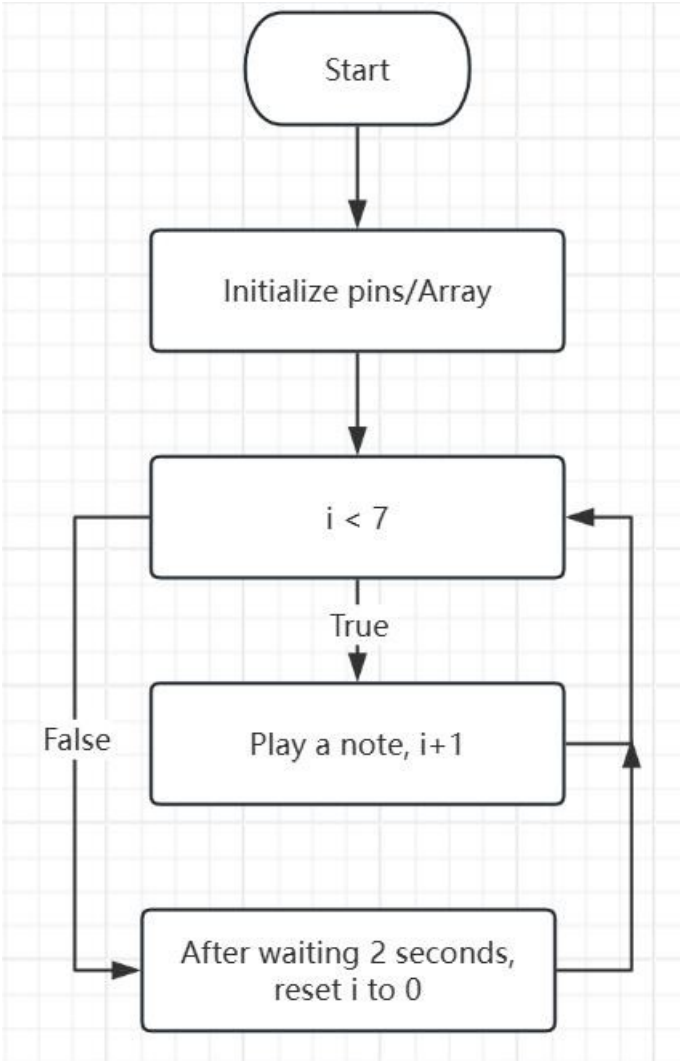
noTone(3): Vypne bzučiak na pine 3.

Pozrime sa na tento kód. Musíme pochopiť, čo predstavujú premenné „m“, „d“ a „t0“ a odkiaľ pochádzajú ich hodnoty.
V kóde na prehrávanie melódií pomocou bzučiaka je potrebné venovať pozornosť niekoľkým kľúčovým premenným: m: Predstavuje frekvenciu tónu, ktorá určuje výšku tónu bzučiaka. Napríklad hodnota 262 zodpovedá tónu C.
d: Predstavuje hodnotu trvania tónu. Tu číslo 4 znamená štvrťovú notu, ktorá sa používa na výpočet skutočnej dĺžky prehrávania.
t0: Skutočná dĺžka tónu, vypočítaná na základe hodnoty „d“ (štvrťová nota, osminová nota atď.).
Frekvencie nôt pochádzajú z vopred definovaných hudobných partitúr. V nasledujúcich častiach budeme diskutovať aj o konkrétnych frekvenciách pre ostatné štandardné noty.

Kľúčový pojem:
t0 = 1000 * 4 / d * 60 / 100:Tento vzorec vypočíta skutočnú dĺžku prehrávania noty. Stačí zmeniť premennú „d“, aby ste získali rôzne dĺžky nôt, napríklad:
d = 4 → štvrťová nota d
= 8 → osminová nota d
= 2 → polová nota

tone(BUZ,m,t0*0.9): Prehrá notu s frekvenciou „262“ na bzučiaku pripojenom k pinu 3 po dobu t0*0,9 milisekúnd.
Prečo sa používa t0*0.9:
Keď program dosiahne tento riadok, nečaká, kým bzučiak dohrá, a pokračuje ďalej. Bzučiak prijme príkaz a program pokračuje v vykonávaní ďalších riadkov.
Po tomto riadku nasleduje oneskorenie (delay(t0)), ktoré pozastaví program na to milisekúnd. Ak by sme v tone() použili „t0“ ako trvanie, bzučiak by znel po celú dobu bez medzery medzi tónmi.
Napríklad, ak je t0 = 1000 ms, bzučiak znie 1000*0,9 = 900 ms a program sa oneskorí o 1000 ms, čím vznikne medzera 100 ms medzi tónmi. Táto medzera dodáva rytmus, vďaka čomu melódia znie prirodzenejšie.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Postup nahratia“ na strane 8.

Kľúčové body:

tone()	Používa sa na vydávanie zvuku bzučiaka; môžete špecifikovať pin, frekvenciu tónu a trvanie.
noTone()	Používa sa na zastavenie zvuku bzučiaka.
int a[3] = {1,2,3}	Definuje celočíselné pole a s tromi prvkami: 1, 2, 3.
int a[] = {1,2,3}	Definuje celočíselné pole a, ktorého dĺžka sa automaticky nastaví na 3 na základe počtu prvkov:
for(int i = 0; i < 7; i++)	Opakované 7-krát, pričom "i" sa zvyšuje z 0 na 6, každýkrát o 1.

Lekcia 05 --- Relé

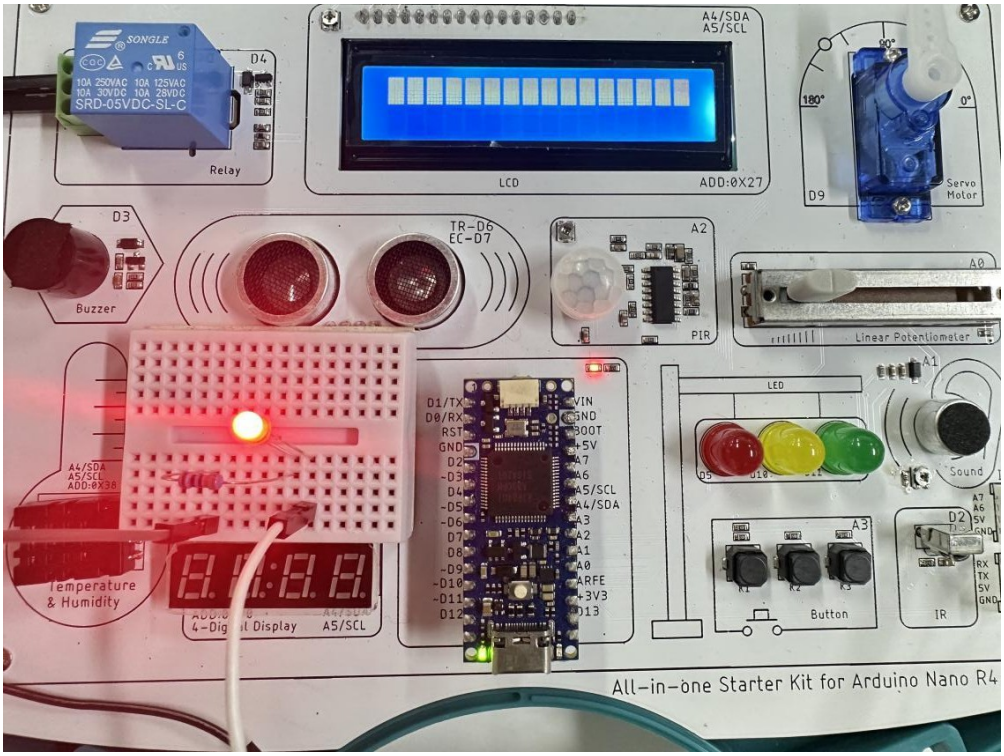
Úvod

V tejto lekcii sa budeme venovať relé, so zameraním na pochopenie ich fungovania. Prostredníctvom experimentu s LED osvetlením vizuálne predvedieme a podrobne vysvetlíme proces fungovania relé. Na konci tejto lekcie budú študenti rozumieť základným princípom relé a zvládnu ich základné použitie.
Poznámka: Doska na prototypy, prepojovacie vodiče, rezistory a sada LED spomenuté v tomto kurze nie sú súčasťou balenia a je potrebné si ich zabezpečiť samostatne.

Ciele výučby

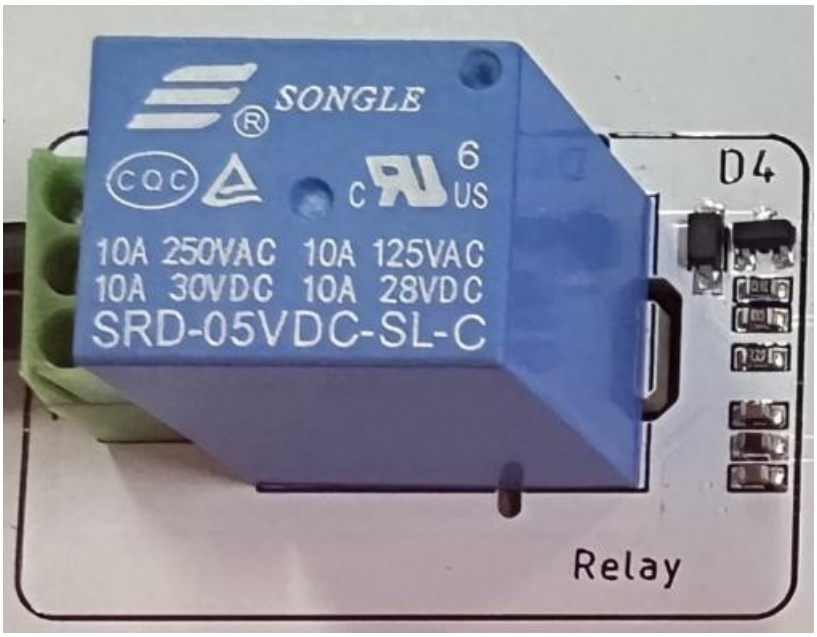
1. Porozumieť fungovaniu relé
2. Ovládnuť princíp fungovania relé

Náhľad výsledku



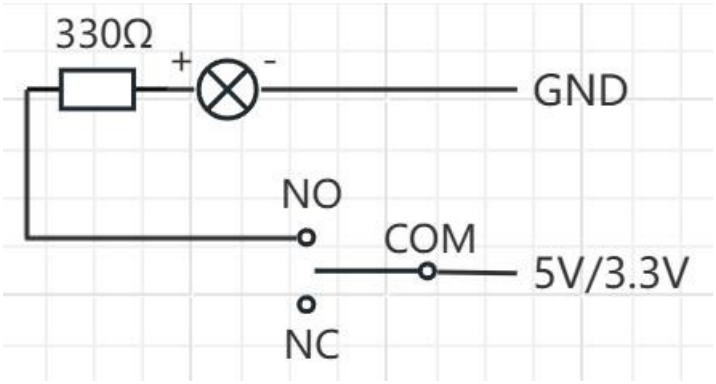
Po nahratí kódu bude LED dióda na doske ovládaná relé tak, že sa na 3 sekundy rozsvieti a potom na 3 sekundy zhasne.
Ak nie je pripojená žiadna externá LED dióda, po nahratí kódu bude relé vydávať len „klikavý“ zvuk. Tento zvuk je spôsobený prepínaním spojení medzi svorkami NO a NC.

Hardvér použitý v tejto lekcii



Reléový modul je výstupné zariadenie umiestnené v ľavom hornom rohu vývojovej dosky Arduino Nano R4.

V tejto lekcii použijeme aj prototypovaciu dosku na zostavenie obvodu s LED diódou. Nastavenie obvodu je znázornené na obrázku nižšie:

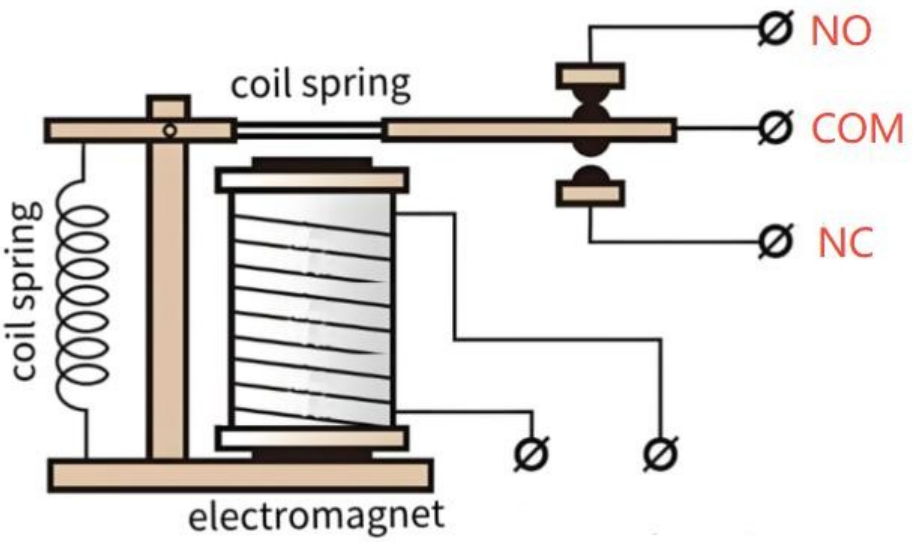


LED je zapojená do série s 330-ohmovým rezistorom a pripojená k NO svorkám relé. Druhý koniec LED je pripojený k GND (tu používame GND pin v pravom dolnom rohu dosky Arduino Nano R4). COM svorky relé sú pripojené k 5V (pomocou 5V pinu v pravom dolnom rohu dosky Arduino Nano R4).

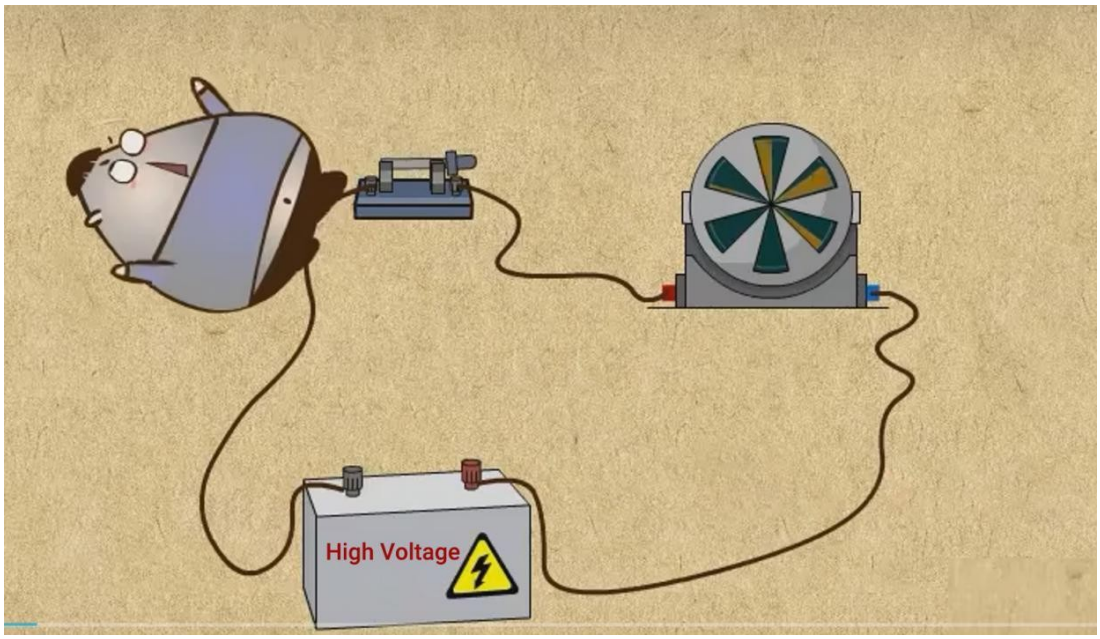
Princíp fungovania reléového modulu

Relé je elektricky ovládaný spínač s vnútornými cievkami a kontaktmi. Medzi bežné kontakty patria COM (spoločný), NC (normálne uzavretý) a NO (normálne otvorený). V normálnom stave bez napájania je COM pripojený k NC a odpojený od NO. Keď je cievka pod napätím, generuje magnetickú silu, ktorá pohybuje kontaktmi a prepína COM z NC na NO. V tomto bode je COM pripojený k NO a odpojený od NC. Vďaka tomuto mechanizmu umožňuje relé maloprúdovému nízkonapäťovému riadiacemu signálu bezpečne ovládať vysokoprúdové alebo vysokonapäťové zaťaženia,

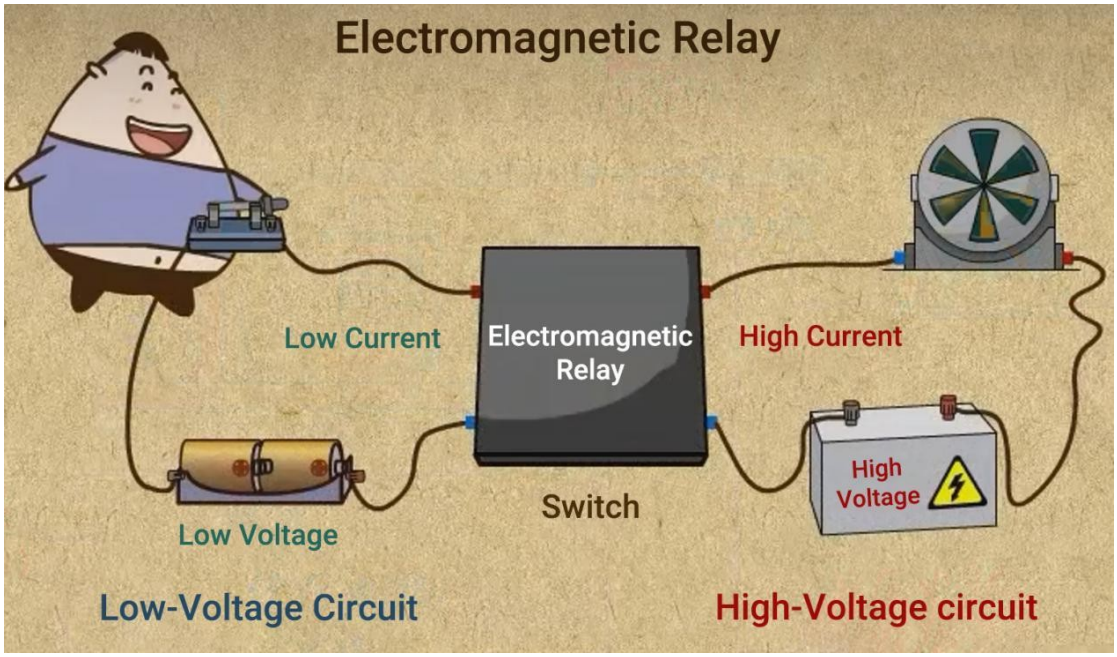
čo umožňuje zapínanie, vypínanie alebo presmerovanie obvodov.



Bez relé je priame ovládanie vysokonapäťových zariadení mimoriadne nebezpečné:



Vďaka použitiu relé na ovládanie vysokonapäťových zariadení je to pre ľudí bezpečné:



Príklad LED riadeného relé

Pred prečítaním kódu si ho môžete stiahnuť. Tu je odkaz na stiahnutie kompletného kódu:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/5_Relay
Otvorte súbor „5_Relay.ino“ v priečinku „5_Relay“ pomocou Arduino IDE.

Vysvetlenie kľúčových častí kódu

Teraz si prejdime tento príklad: Príklad LED riadenej relé

Najskôr začneme definovaním pinov, ktoré sa budú používať.

```
#define Relé 4
```

Na vývojovej doske „Arduino Nano R4“ je relé pripojené k pinu D4 hlavného ovládača. Pomocou #define pomenujeme pin D4 ako „Relay“.

Inicializačná funkcia

```
void setup() {
  pinMode(Relay, OUTPUT); // Nastavte pin relé ako výstup
}
```

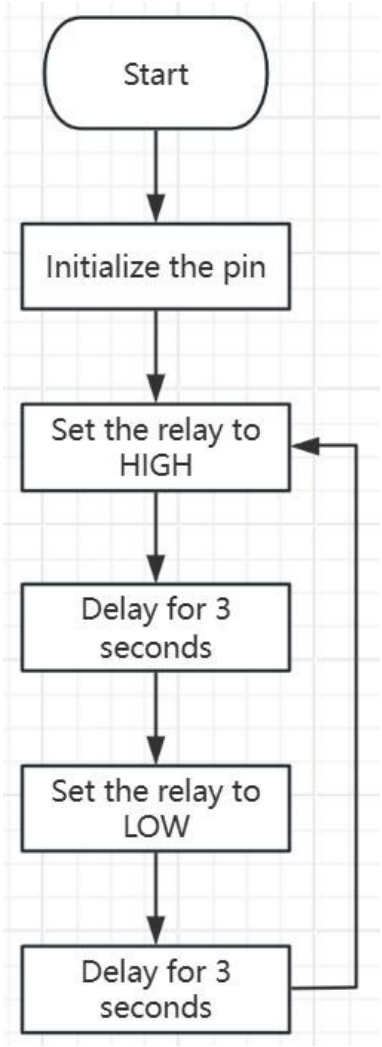
Nastavte pin „Relay“ do výstupného

režimu. Funkcia slučky

```
void loop() {
  digitalWrite(Relay, HIGH); // Zapnite relé
  delay(3000);                    // Udržujte relé zapnuté po dobu 3
  sekúnd digitalWrite(Relay, LOW); // Vypnite relé
  delay(3000);                    // Udržujte relé vypnuté po dobu 3 sekúnd
}
```

Použite „digitalWrite“ na nastavenie pinu na HIGH alebo LOW na ovládanie relé. Keď je aplikovaná úroveň HIGH, terminál „COM“ relé sa pripojí k terminálu „NO“. Naopak, keď je aplikovaná úroveň LOW, terminál „COM“ relé sa pripojí k terminálu „NC“.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Kroky na nahratie“ na strane 8.

Kľúčové body:

Princíp fungovania relé	Keď je na relé privedený signál HIGH, svorka COM sa spojí so svorkou NO. Keď je privedený signál LOW, svorka COM s terminálom NC.

Lekcia 06 --- Lineárny potenciometer

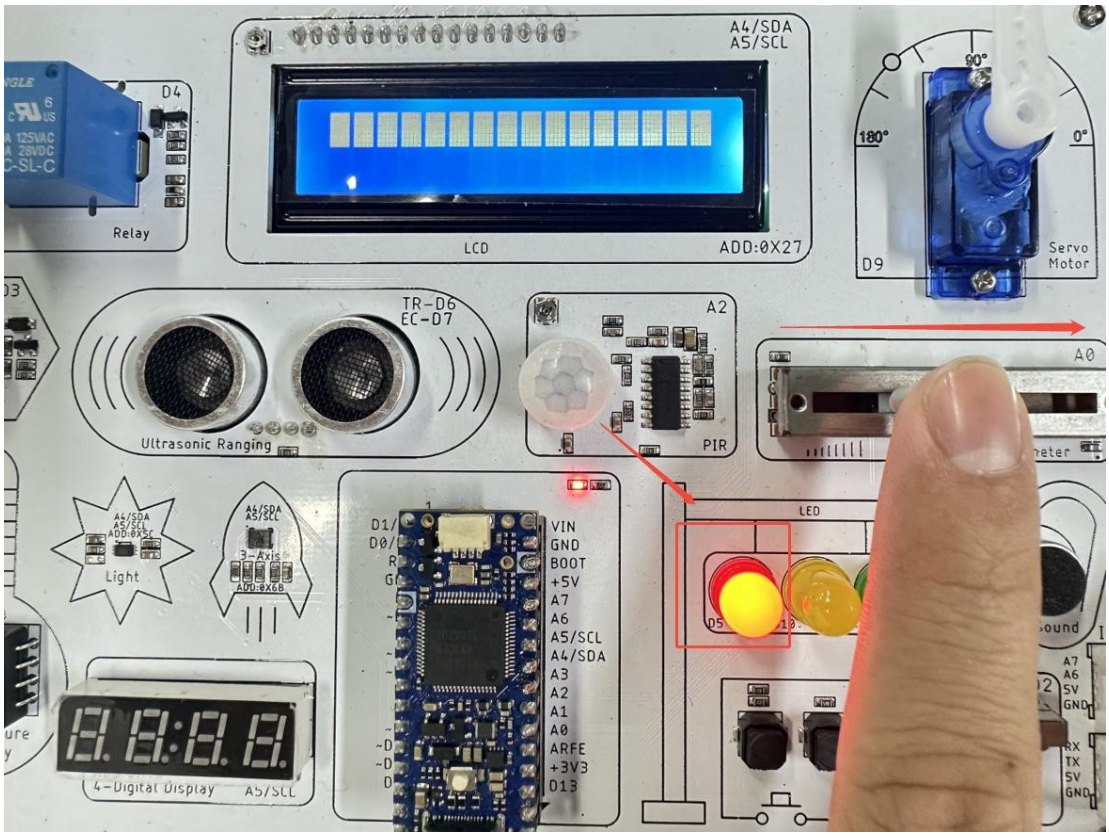
Úvod

V tejto lekcii sa naučíte, ako pomocou PWM ovládať jas LED diódy pomocou modulu „Lineárny potenciometer“. Prostredníctvom praktického príkladu získate pochopenie toho, ako funguje PWM, a naučíte sa, ako nastaviť jas LED diódy pomocou „analogWrite“.

Ciele výučby

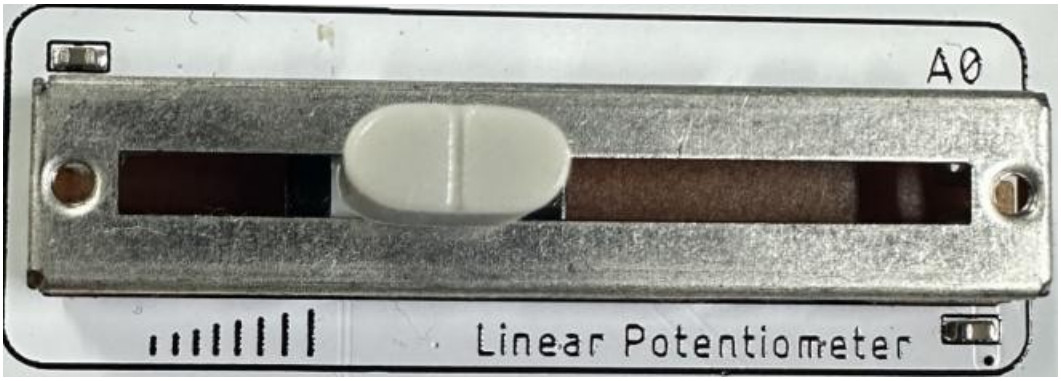
1. Porozumieť fungovaniu posuvného potenciometra
2. Naučte sa, ako používať operátory v programe a ako ich využiť na premietnutie hodnôt z jedného rozsahu do druhého
3. Porozumejte princípom PWM riadenia
4. Naučte sa ovládať jas LED pomocou funkcie „analogWrite“
5. Vykonajte experiment, v ktorom pomocou posuvného potenciometra plynule regulujete jas LED

Náhľad výsledku



Po nahratí kódu sa pri posúvaní potenciometra zľava doprava bude červená LED postupne zosvetľovať.

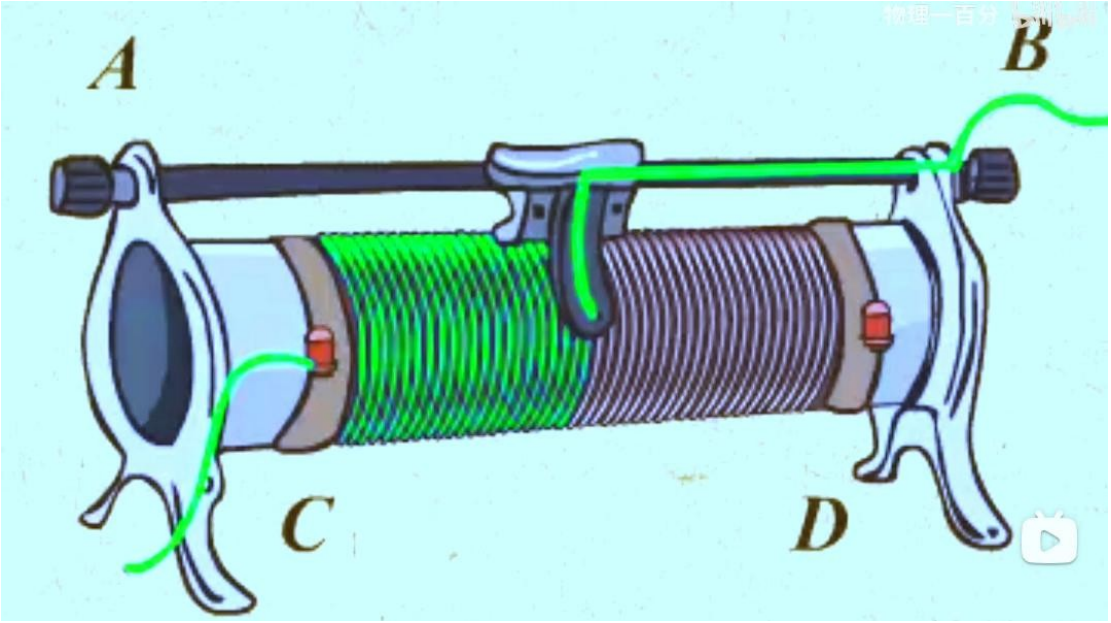
Hardvér použitý v tejto lekcii



Modul lineárneho potenciometra sa nachádza na pravej strane dosky Arduino Nano R4:
Jeho signálny pin je pripojený k pinu A0 dosky Arduino Nano R4, ktorý číta analógové hodnoty v rozsahu od 0 do 1023. Keď je lineárny potenciometer posunutý úplne doľava, analógová hodnota je 0. Keď je posunutý úplne doprava, hodnota dosahuje 1023.

Princíp fungovania modulu lineárneho potenciometra

V tejto lekcii funguje modul „Lineárny potenciometer“ ako kovový stierač, ktorý sa pohybuje po odporovej dráhe. Ako je znázornené na schéme, rozsah posuvu siaha od bodu A po bod B. Efektívny odpor je zvýraznený zelenou farbou a predstavuje časť odporovej dráhy medzi bodom C a stieračom. Keď je stierač posunutý úplne doprava, dĺžka odporovej dráhy medzi bodmi C a B je maximálna, čo má za následok najvyšší odpor.



Posuvný potenciometer – príklad riadenia jasu LED

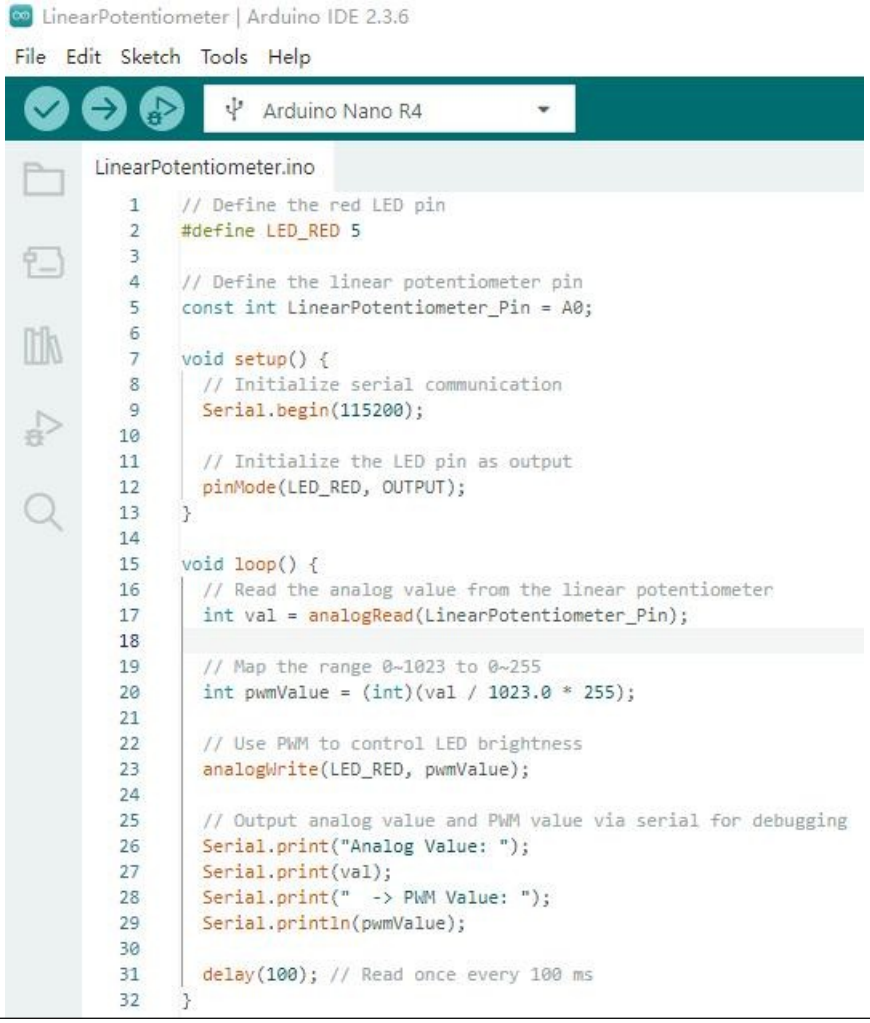
Než sa pustíte do prehliadania kódu, môžete si ho najskôr stiahnuť. Odkaz na stiahnutie kompletného kódu:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/6_Linear_Potentionmeter
Otvorte priečinok „6_Linear_Potentionmeter“ v prostredí Arduino IDE a potom otvorte súbor „6_Linear_Potentionmeter.ino“.

Vysvetlenie kľúčového kódu

Teraz si prejdime príklad: použitie posuvného potenciometra na ovládanie jasu LED.

Najprv definujme piny, ktoré budeme používať:




```
#define LED_RED 5
const int LinearPotentiometer_Pin = A0;
```

V tomto príklade použijeme jednu LED diódu a posuvný potenciometer. LED dióda je červená LED pripojená k pinu 5 a posuvný potenciometer je pripojený k analógovému pinu A0.

Inicializačná funkcia

```
void setup() { Serial.begin(115200);
  pinMode(LED_RED, OUTPUT);
```

```
}
```

Inicializujte sériový port a nastavte pin LED do výstupného režimu – oboje sme prebrali v predchádzajúcich lekciách. Uistite sa, že prenosová rýchlosť, ktorú tu nastavíte, zodpovedá prenosovej rýchlosti v Serial ; inak uvidíte skreslený výstup. Vo funkcii loop

```
void loop() {
  int val = analogRead(LinearPotentiometer_Pin);
```

Načítajte hodnotu vrátenú posuvným potenciometrom. Jeho výstupný rozsah je od 0 do 1023, takže ho definujeme ako int

```
  int pwmValue = (int)(val / 1023.0 * 255);
```

Tento krok je kľúčový: musíme previesť rozsah z 0 — 1023 na 0 — 255. O chvíľu vysvetlíme, prečo je táto konverzia potrebná. Teraz sa zameriame na to, ako to urobiť.

1. Ak chcete previesť rozsah (0 — 1) na (0 — 255), stačí vynásobiť číslom 255: (0-1)*255 -> (0–255)
2. Na prevod rozsahu (0 — 1023) na (0 — 1) vydelíte číslom 1023: (0-1023)/1023 -> (0-1)
3. Vzorec na prevod (0 — 1023) na (0 — 255) je teda: (0-1023)/1023,0 * 255

Konečný výsledok priraďte celočíselnej premennej „pwmValue“.

```
  analogWrite(LED_RED, pwmValue);
```

Funkcia „analogWrite“ vysiela PWM signál na pin, čo jej umožňuje simulovať rôzne úrovne napätia a vytvárať rôzne intenzity výstupu.

Najprv sa pozrime na to, ako funguje PWM:

PWM je skratka pre Pulse Width Modulation (modulácia šírky impulzu). Hoci pin môže vysielať iba signál HIGH alebo LOW, môžeme simulovať rôzne úrovne napätia rýchlym prepínaním medzi HIGH a LOW a nastavením dĺžky trvania signálu HIGH počas každého cyklu.

Pracovný cyklus: označuje percentuálny podiel času, počas ktorého signál zostáva na úrovni HIGH v

rámci jedného kompletného cyklu. Ako je znázornené na obrázku:

Doba HIGH = 0 → Pracovný cyklus = 0 % → LED svieti

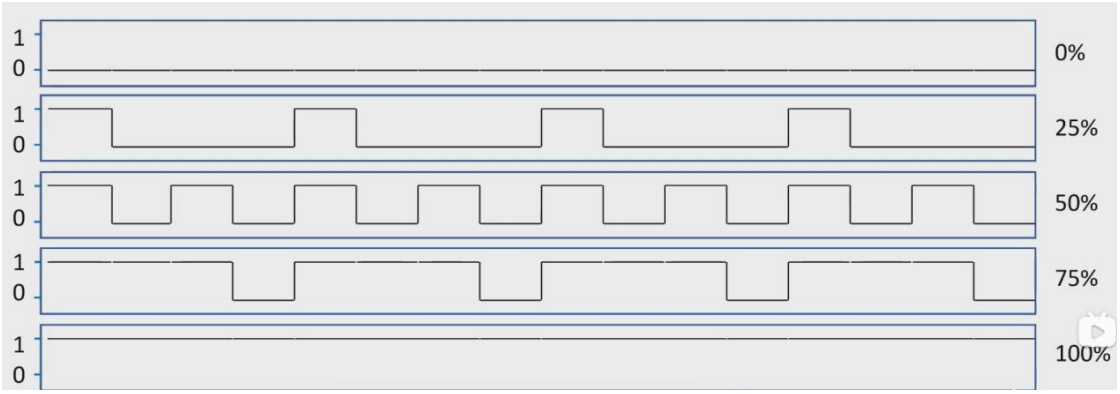
Doba zapnutia = 25 % → Pracovný cyklus = 25 % → Jas LED je 25 %

Doba zapnutia = 50 % → Pracovný cyklus = 50 % → Jas LED je 50 %

Doba zapnutia = 75 % → Pracovný cyklus = 75 % → Jas LED je 75 %

Doba zapnutia = 100 % → Pracovný cyklus = 100 % → LED svieti naplno

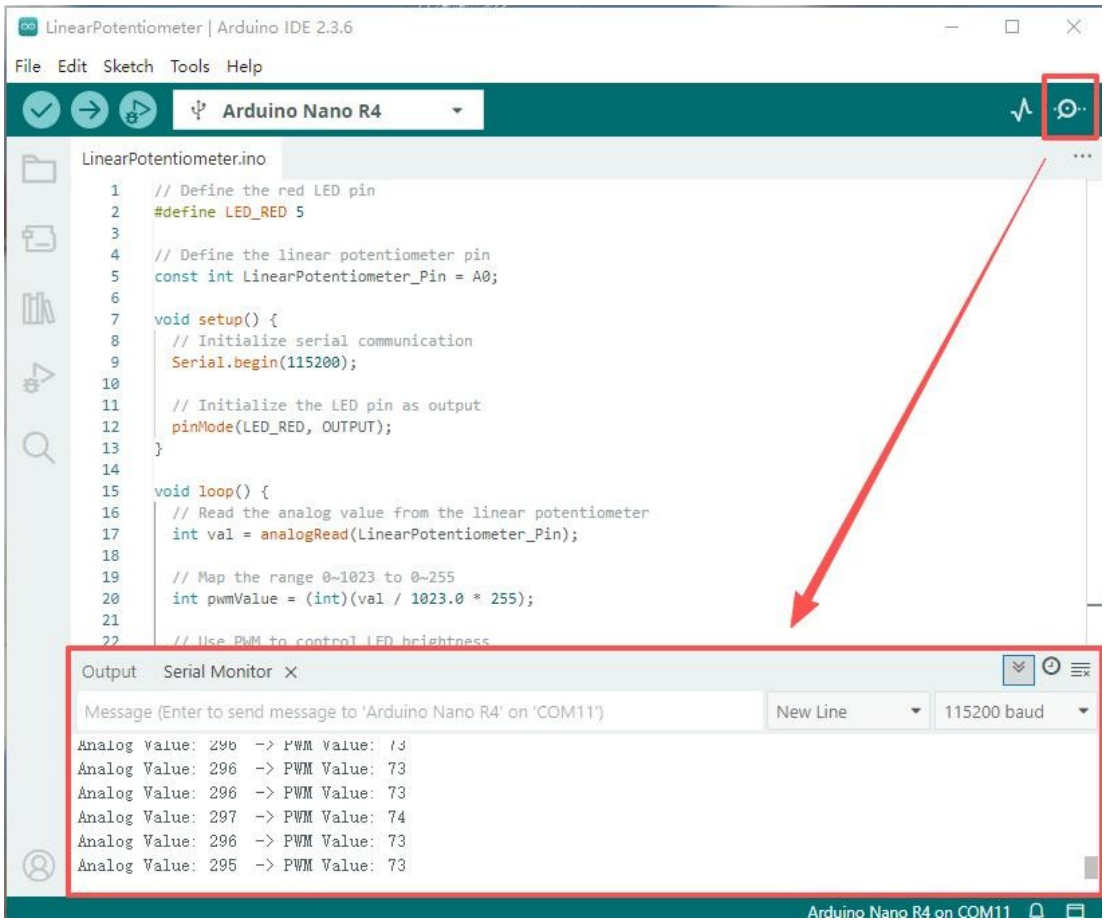
Zvýšením pracovného cyklu sa LED stáva jasnejšou. Znížením pracovného cyklu sa LED stáva tmavšou.

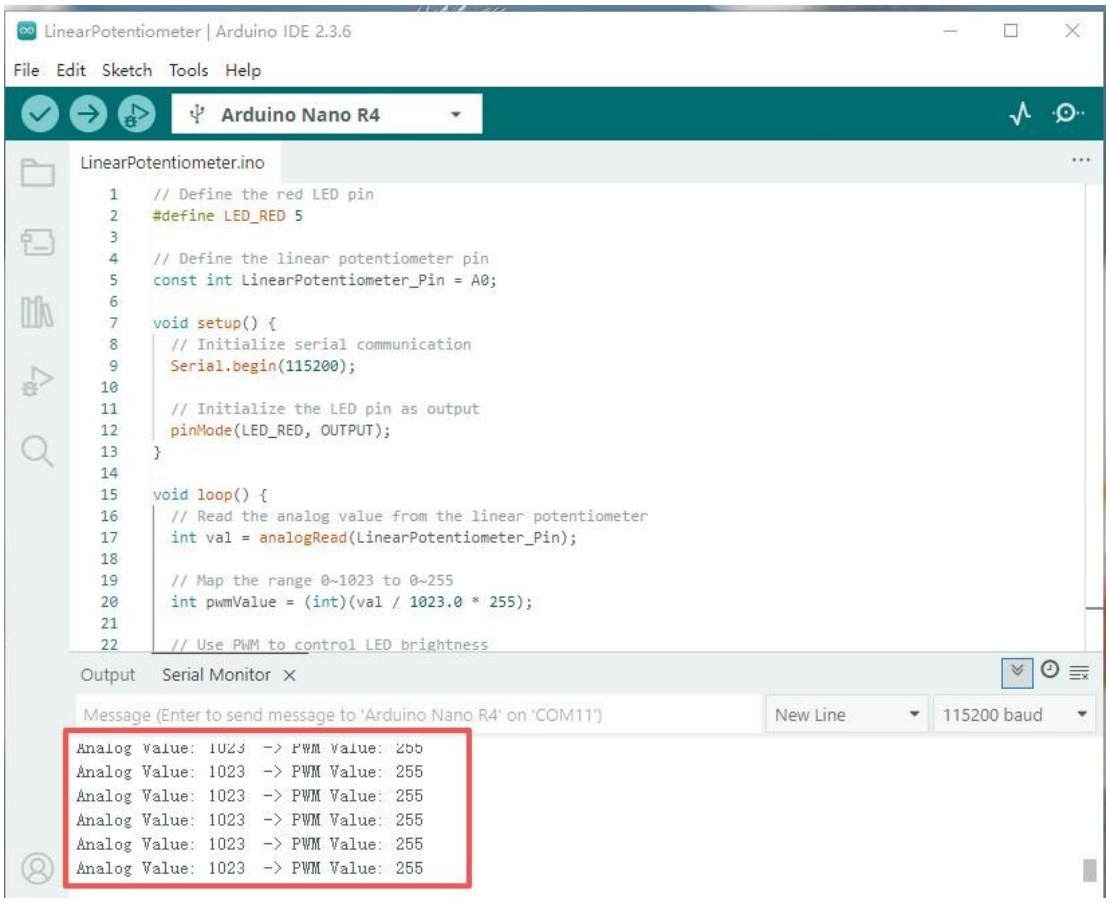


PWM piny prijímajú hodnoty v rozsahu 0 – 255: preto musíme previesť predchádzajúce hodnoty do tohto rozsahu. Mapovaním vstupného rozsahu 0 – 1023 na 0 – 255, posuvný potenciometer môže priamo ovládať jas LED diódy. Keď je potenciometer otočený úplne doľava a hodnota je 0, zapísaná hodnota PWM je 0 a LED dióda svieti najslabšie. Keď je otočený úplne doprava a hodnota dosiahne 1023, zapísaná hodnota PWM je 255 a LED dióda svieti najsilnejšie.

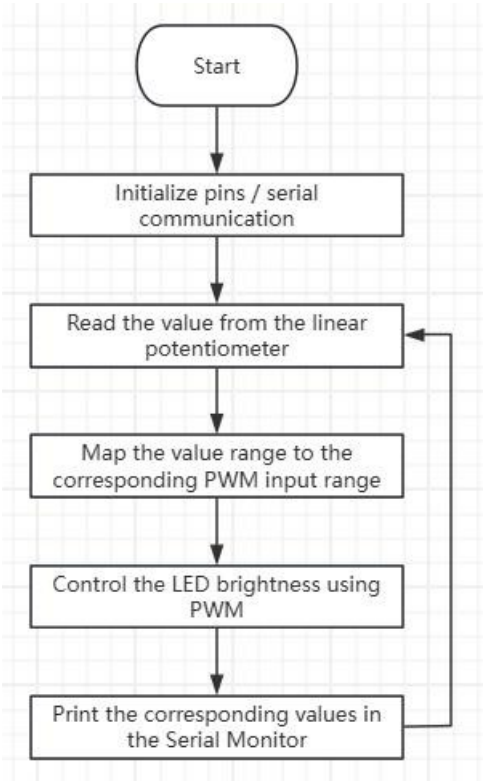
```
  Serial.print("Analógová
  hodnota: "); Serial.print(val);
  Serial.print(" -> Hodnota PWM: ");
  Serial.println(pwmValue);
  delay(100); // Čítať raz za 100 ms
}
```

Nakoniec môžete pomocou sériového monitora sledovať, ako sa hodnoty načítané z posuvného potenciometra premietajú do zodpovedajúcich hodnôt PWM.


LinearPotentiometer | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Nano R4
LinearPotentiometer.ino
// Define the red LED pin
#define LED_RED 5
// Define the linear potentiometer pin
const int LinearPotentiometer_Pin = A0;
void setup() {
// Initialize serial communication
Serial.begin(115200);
// Initialize the LED pin as output
pinMode(LED_RED, OUTPUT);
}
void loop() {
// Read the analog value from the linear potentiometer
int val = analogRead(LinearPotentiometer_Pin);
// Map the range 0~1023 to 0~255
int pwmValue = (int)(val / 1023.0 * 255);
Output Serial Monitor
Message (Enter to send message to 'Arduino Nano R4' on 'COM11')
New Line
115200 baud
Analog Value: 296 -> PWM Value: 73
Analog Value: 296 -> PWM Value: 73
Analog Value: 297 -> PWM Value: 74
Analog Value: 296 -> PWM Value: 73
Analog Value: 295 -> PWM Value: 73
Arduino Nano R4 on COM11



Celkový diagram logického toku kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Kroky na nahratie“ na strane 8.

Kľúčové body:

analogWrite()	Analógový výstup: Bežne sa používa na ovládanie úrovne napájania hardvéru alebo jasu prostredníctvom signálov PWM
*	Operátor násobenia: Používa sa vo výrazoch na násobenie dvoch hodnôt
/	Operátor delenia: Používa sa vo výrazoch na vydelenie jednej hodnoty druhou

Lekcia 07 --- Ovládanie LED pomocou tlačidla

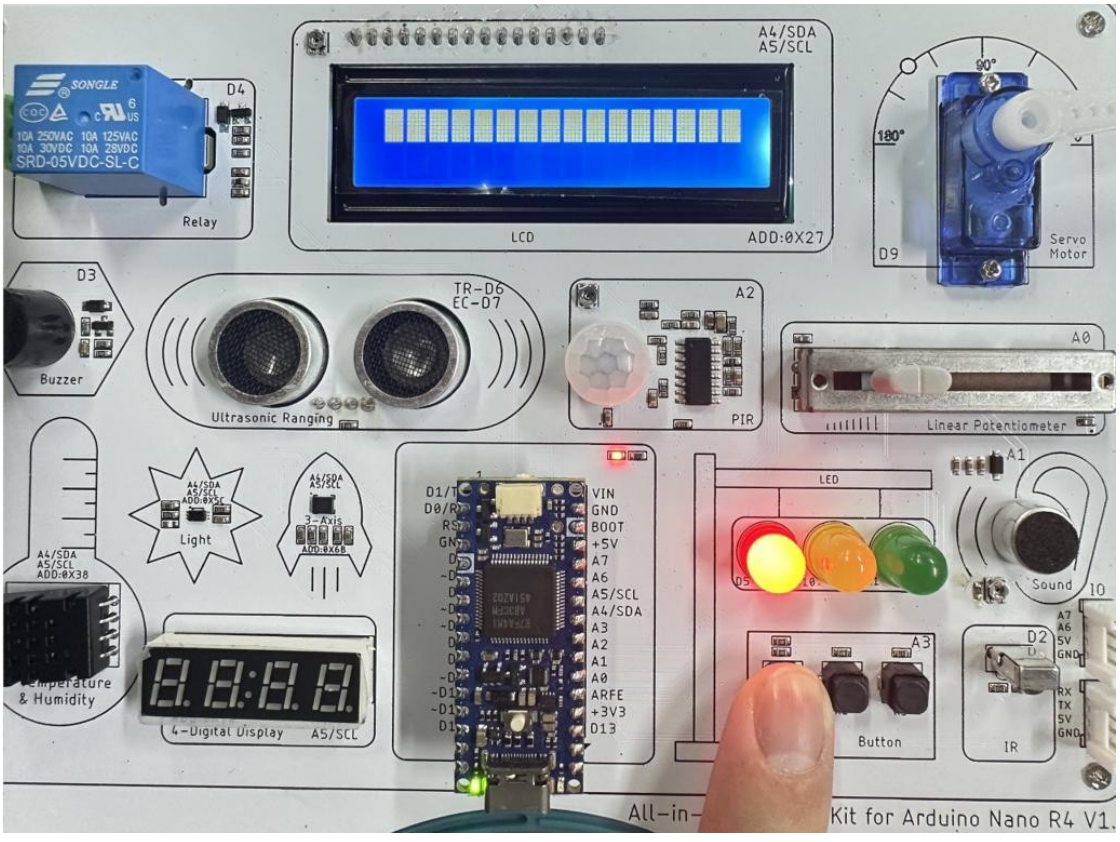
Úvod

V predchádzajúcich lekciách sme sa naučili, ako používať podmienené príkazy „if“ na riadenie priebehu programu na základe jednej podmienky. V tejto lekcii budeme na týchto základoch stavať a naučíme sa príkazy s viacerými podmienkami, pričom pochopíme, ako Arduino určuje a rozlišuje, ktoré tlačidlo bolo stlačené, keď je prítomných viacero vstupov z tlačidiel. Na konci tejto lekcie dokončíte praktický príklad: použitie troch modulov s tlačidlami na nezávislé ovládanie troch LED diód. Prostredníctvom tejto praxe si osvojíte základnú štruktúru logiky s viacerými podmienkami a spôsob, akým rôzne podmienky spúšťajú rôzne výsledky.

Ciele výučby

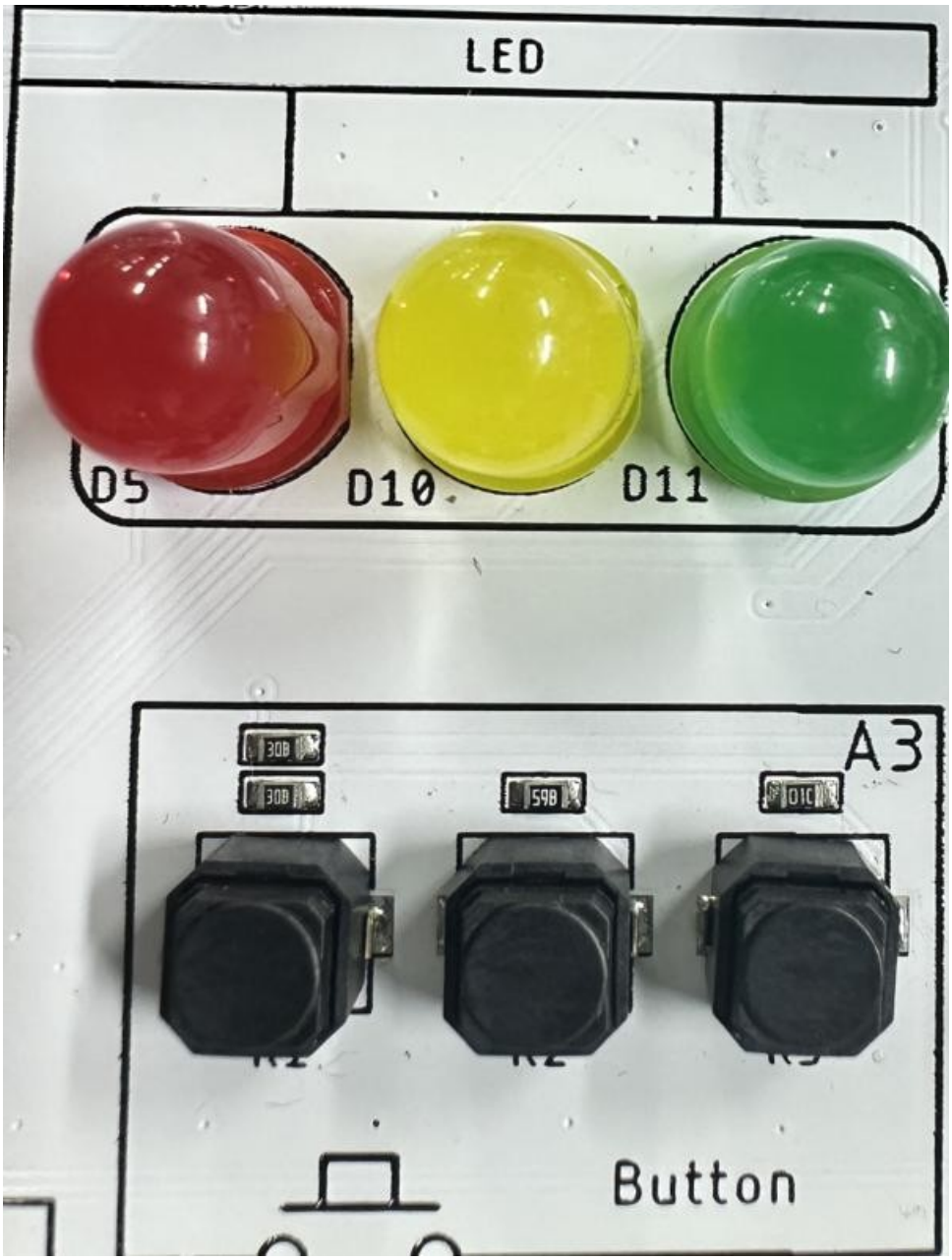
1. Porozumieť princípu fungovania tlačidlových modulov
2. Ovládnuť logiku vykonávania príkazov if - else if - else
3. Používať príkazy if - else if - else na určenie, ktorý z troch tlačidlových modulov je stlačený, a rozsvietiť príslušnú LED diódu
4. Ovládajte logické operátory používané v podmienených príkazoch

Náhľad výsledku



Po nahratí kódu stlačením tlačidla zapnete príslušnú LED diódu nad ním.

Hardvér použitý v tejto lekcii

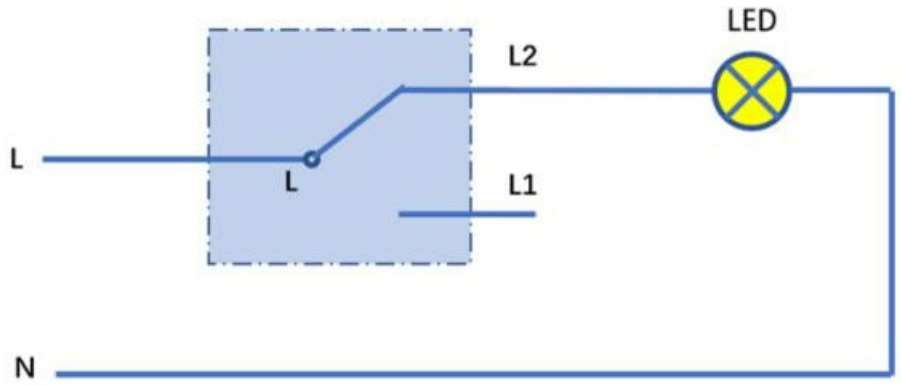


Na vývojovej doske Arduino Nano R4 sa nachádzajú tri tlačidlá：
Všetky tri tlačidlá sú pripojené k vývodu A3 hlavného ovládača Arduino Nano R4. A3 je analógový vstupný vývod, ktorý dokáže z vývodu získať hodnoty analógového napätia. Pri týchto troch tlačidlách stlačenie rôznych tlačidiel spôsobuje, že vývod A3 vráti rôzne analógové hodnoty. Na základe týchto hodnôt môže program zistiť, ktoré tlačidlo bolo stlačené, a následne vykonať príslušnú akciu.

Princíp fungovania tlačidlového modulu

Tlačidlový modul, ktorý sme použili v tejto lekcii, je ekvivalentom jednopólového dvojpolohového spínača. Ako

je znázornené na obrázku nižšie: Keď je tlačidlo stlačené, je to ekvivalentné tomu, že prepínač dosiahne koniec L2 a obvod sa zapne; keď je tlačidlo uvoľnené, je to ekvivalentné tomu, že prepínač dosiahne koniec L1 a obvod sa odpojí.



LED ovládaná tlačidlom

Než sa pustíte do prezerania kódu, môžete si ho stiahnuť. Tu je odkaz na stiahnutie kompletného kódu: https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/7_Button_Control_LED
Otvorte súbor „7_Button_Control_LED.ino“ v priečinku „7_Button_Control_LED“ pomocou Arduino IDE.

Vysvetlenie kódu

Teraz si prejdime príklad: LED ovládaná tlačidlom

Najskôr začneme definovaním pinov, ktoré sa budú používať：

```
#define LED_RED 5
#define LED_YELLOW 10
#define LED_GREEN 11
#define Button_Pin A3
```

Tri LED diódy sú pripojené k vývodom 5, 10 a 11. Nachádza sa tu aj vývod A3 pre senzor tlačidla, o ktorom sme sa dozvedeli v tejto lekcii.

Inicializačná funkcia

```
void setup() { pinMode(LED_RED,
  OUTPUT);
```

```
  pinMode(LED_YELLOW, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(Button_Pin, INPUT);
  Serial.begin(115200);
}
```

Inicializujte príslušné režimy pinov: všetky piny LED sú nastavené na výstupný režim, zatiaľ čo senzor tlačidla potrebuje zbierať vrátené údaje, takže je nastavený na vstupný režim.

Vo funkcii loop

```
void loop() {
  int val = analogRead(Button_Pin);
  Serial.print("Hodnota tlačidla: ");
  Serial.println(val);
```

Na načítanie analógovej hodnoty vrátenej senzorom tlačidla použite metódu „analogRead“. Po otvorení sériového monitora vidíme, že analógové hodnoty troch tlačidiel sú približne 511, 684 a 853. Medzi jednotlivými hardvérovými jednotkami môžu existovať mierne rozdiely, čo je normálne.

```
  // Skontrolujte rozsah analógových hodnôt a rozsvietite príslušnú LED
  diodu if (val >= 500 && val <= 520) {
    digitalWrite(LED_RED, HIGH);
  }
  else if (val >= 680 && val <= 690) {
    digitalWrite(LED_YELLOW, HIGH);
  }
  inak ak (val >= 845 && val <= 860) {
    digitalWrite(LED_GREEN, HIGH);
  }
  inak{
  // Vypnúť všetky LED diódy ako predvolené
  nastavenie digitalWrite(LED_RED, LOW);
  digitalWrite(LED_YELLOW, LOW);
  digitalWrite(LED_GREEN, LOW);
  }

  delay(100); // Čítaj každých 100 ms
}
```

Toto je kľúčový bod tejto lekcie: if-else if-else. V predchádzajúcich lekciách sme sa naučili príkaz if, ktorý je kontrolou jednej podmienky. Ak je podmienka pravdivá, kód vnútri príkazu if sa vykoná.

if (val >= 500 && val <= 520) :Táto podmienka zodpovedá stlačeniu prvého tlačidla. Skôr sme zistili, že stlačenie prvého tlačidla vráti analógovú hodnotu 511, takže tu nastavíme rozsah, ktorý pokrýva túto hodnotu. Rôzny hardvér môže vrátiť mierne odlišné hodnoty, čo je normálne.

Ako referenčnú hodnotu použite analógové hodnoty, ktoré poskytuje váš vlastný hardvér.

&&: znamená „a“. Je to logický operátor. Keď spája dve podmienky, celkový výsledok je True len vtedy, ak sú splnené obe podmienky.

Zamyslite sa: Prečo tu nepoužijeme priamo podmienku typu „**if (val == 511)**“?
Pretože, ako bolo spomenuté skôr, hardvér môže produkovať drobné chyby. Niekedy vrátená hodnota nemusí byť presne 511, ale 512 alebo iná podobná hodnota. Preto je použitie rozsahu pre kontrolu podmienky spoľahlivejšie a bezpečnejšie.

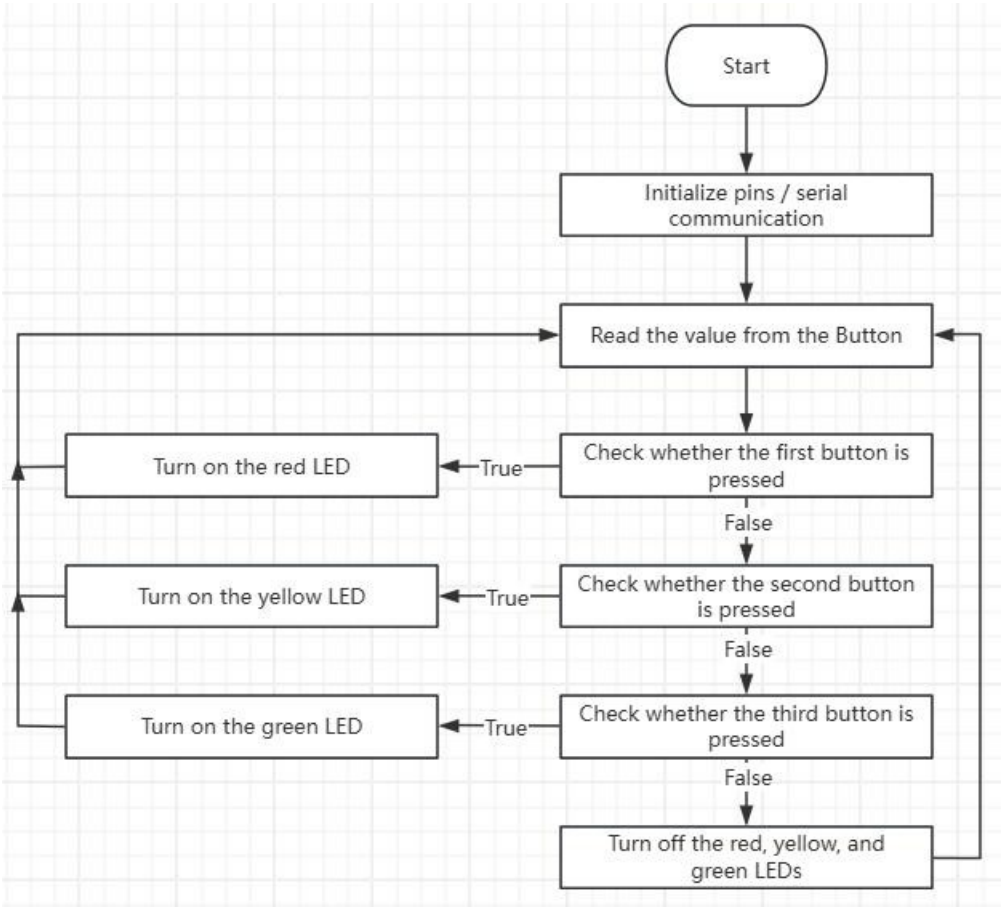
else if (val >= 680 && val <= 690):Až keď nebude splnená predchádzajúca podmienka if, program prejde k tomuto príkazu else if, aby skontroloval, či je táto podmienka splnená. Inými slovami, program skontroluje, či je stlačené druhé tlačidlo, až potom, čo zistí, že prvé tlačidlo nie je stlačené.

else if (val >= 845 && val <= 860):Až keď nie je stlačené ani druhé tlačidlo, program skontroluje, či bolo stlačené tretie tlačidlo.

else: Nakoniec, ak nie je splnená žiadna z predchádzajúcich podmienok, program prejde do tejto časti a vykoná kód v nej.

Popíšme si vyššie uvedený kód slovami:
Skontrolujte, či je vrátená analógová hodnota v rozmedzí 500 – 520 (či je stlačené prvé tlačidlo). Ak áno, zapnite červenú LED.
Ak nie, skontrolujte, či je hodnota v rozmedzí 680 – 690 (či je stlačené druhé tlačidlo). Ak áno, rozsvieťte žltú LED.
Ak nie, skontrolujte, či je hodnota v rozmedzí 845 – 860 (či je stlačené tretie tlačidlo). Ak áno, rozsvietite zelenú LED.
Ak nie je splnená žiadna z týchto podmienok, program prejde do bloku else a vypne všetky tri LED diódy.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Postup nahratia“ na strane 8.

Kľúčové body:

&&	Logický operátor, ktorý predstavuje „a“. Keď spája dve podmienky, celý výraz je pravdivý len vtedy, ak sú pravdivé obe podmienky.
if	Ak je podmienka pravdivá, vykoná sa kód vnútri „{}“ nasledujúci za príkazom „if“ , bude vykonaný.
else if	Ak je predchádzajúca podmienka „if“ False, program prejde k ďalšej podmienke v poradí. Ak je táto podmienka True, kód vnútri všetko, čo nasleduje za „{}“, sa vykoná.
inak	Ak sú všetky predchádzajúce podmienky False, vykoná sa kód vnútri „{}“ nasledujúci po „else“ bude vykonaný.

Lekcia 08 --- Servo

Úvod

V tejto lekcii sa naučíte, ako používať modul servomotora, a získate prvý úvod do externých knižníc. Uvidíte, ako importovať externú knižnicu, deklarovať objekt knižnice a volať jej metódy na ovládanie serva. Prostredníctvom tohto príkladu získate úplné pochopenie

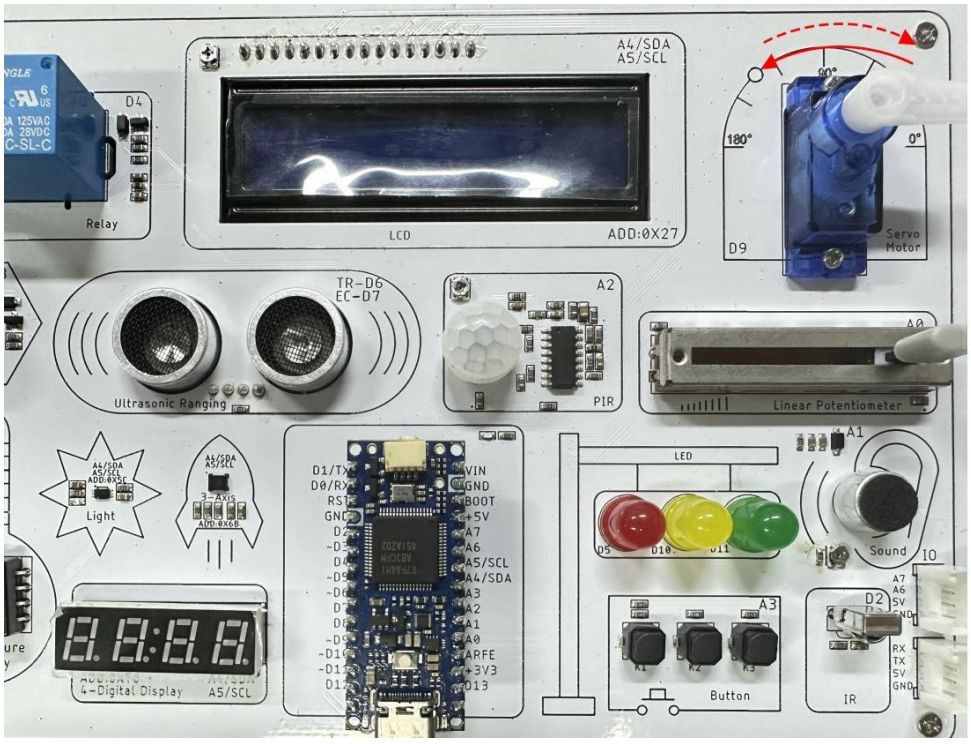
postup pri používaní externých knižníc – od importu a vytvorenia objektu až po volanie metód

— čím si vytvoríte pevný základ pre prácu s ďalšími modulmi v budúcich lekciách.

Ciele výučby

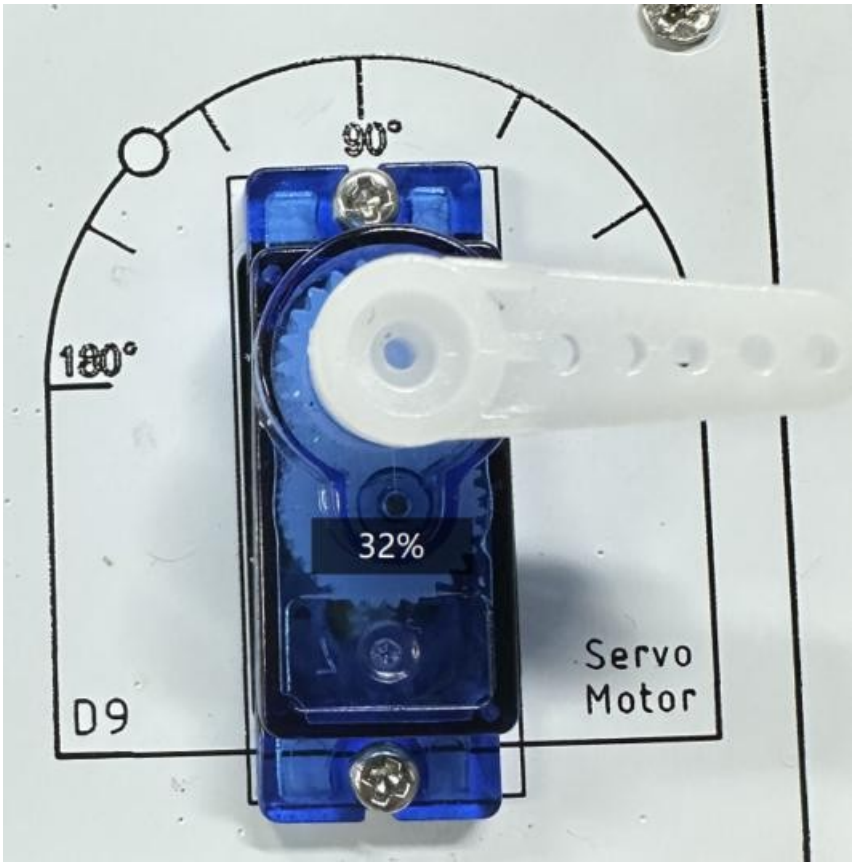
1. Zoznámte sa s princípom fungovania modulu servomotora
2. Naučte sa, ako používať externú knižnicu na ovládanie modulu
3. Vykonajte experiment, pri ktorom sa servomotor otáča v rozmedzí 0 až 180 stupňov.

Náhľad výsledku



Po nahratí kódu sa servomotor bude opakovane otáčať tam a späť v rozmedzí 0 180 stupňov a 180 — 0 stupňov.

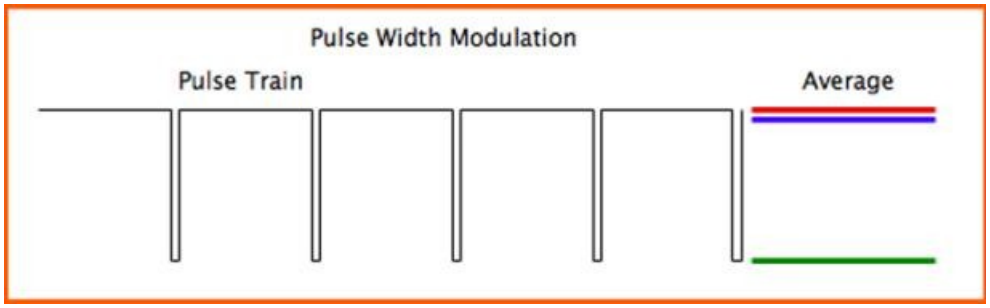
Hardvér použitý v tejto lekcii



Servomotor sa nachádza v pravom hornom rohu dosky Arduino Nano R4 a je pripojený k pinu D9. D9 je digitálny výstup, ktorý podporuje PWM, čo umožňuje presné ovládanie uhla otáčania serva.

Princíp fungovania servomodulu

Modul servomotora je tiež poháňaný signálom PWM (pulzná šírková modulácia). V predchádzajúcich lekciách sme sa naučili, že PWM ovláda motory alebo iné moduly úpravou šírky impulzov. V prípade serva je uhol otáčania určený šírkou impulzu signálu PWM: dlhšia šírka impulzu vedie k väčšiemu uhlu otáčania, zatiaľ čo kratšia šírka impulzu vedie k menšiemu uhlu. Tento mechanizmus umožňuje presné ovládanie polohy serva.



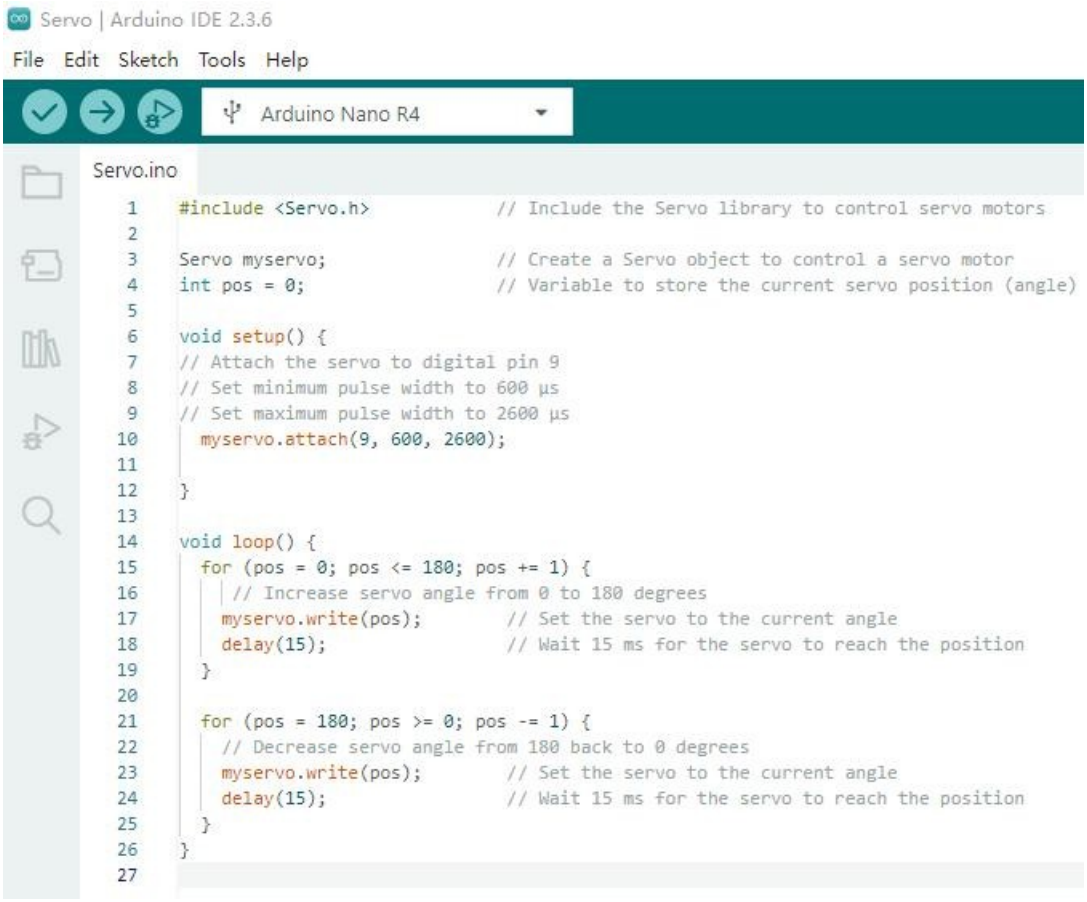
Otáčanie serva

Predtým, ako prejdeme kódom, si ho môžete najskôr stiahnuť. Odkaz na stiahnutie kompletného kódu:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/8_Servo
Otvorte priečinok „8_Servo“ v prostredí Arduino IDE a potom otvorte súbor „8_Servo.ino“.

Vysvetlenie kľúčových častí kódu

Teraz si prejdime tento príklad: otáčanie servomotora.



Zahrnutie knižnice Arduino Servo Control

```
#include <Servo.h>
```

„Servo.h“ je oficiálna knižnica na ovládanie serva, ktorú poskytuje Arduino. Obsahuje nízkoúrovňové spracovanie signálu PWM, čo vám umožňuje ovládať otáčanie serva v rozsahu od 0 do 180 stupňov priamo, bez toho, aby ste museli sami generovať PWM vlny.

„#include“ je direktíva preprocesora. Pred kompiláciou doslova skopíruje obsah iného súboru do vášho programu.

Z tohto dôvodu by sa „#include“ malo používať opatrne – vkladanie veľkých alebo nepotrebných knižníc môže zvýšiť veľkosť kódu a potenciálne spomaliť váš projekt. Najlepšie je vyhnúť sa vkladaniu knižnice, ktoré nie sú relevantné pre vašu aplikáciu.

Vytvorenie objektu serva

```
Servo myservo;
```

Vytvorte objekt triedy „Servo“ s názvom „myservo“. Tento objekt reprezentuje jednu inštanciu serva a všetky následné ovládanie serva sa bude vykonávať prostredníctvom „myservo“.

Definovanie premennej uhla

```
int pos = 0;
```

Použite celočíselnú premennú s názvom „pos“ na reprezentáciu aktuálnej uhlovej polohy serva a inicializujte ju na hodnotu 0. Neustálou aktualizáciou tejto premennej môžete meniť uhol serva a dosiahnuť plynulý, nepretržitý pohyb.

Inicializačná funkcia

```
void setup() {
   myservo.attach(9, 600, 2600);
}
```

V inicializačnej funkcii používame metódu attach na špecifikovanie pinu serva, ako aj minimálnych a maximálnych limitov otáčania. Tento krok je veľmi dôležitý.

Podrobnosti o parametroch:

9: Pin na ovládanie serva. Na doske Arduino Nano R4 je toto priradenie pinov pevné. **600**: Šírka impulzu (v mikrosekundách) zodpovedajúca minimálnemu uhlu (0°). **2600**: Šírka impulzu (v mikrosekundách) zodpovedajúca maximálnemu uhlu (180°).

Ak máte pocit, že minimálny uhol serva nie je dostatočne malý, môžete znížiť hodnotu 600. Ak maximálny uhol nie je dostatočne veľký, môžete zvýšiť hodnotu 2600.

V slučke funkcie

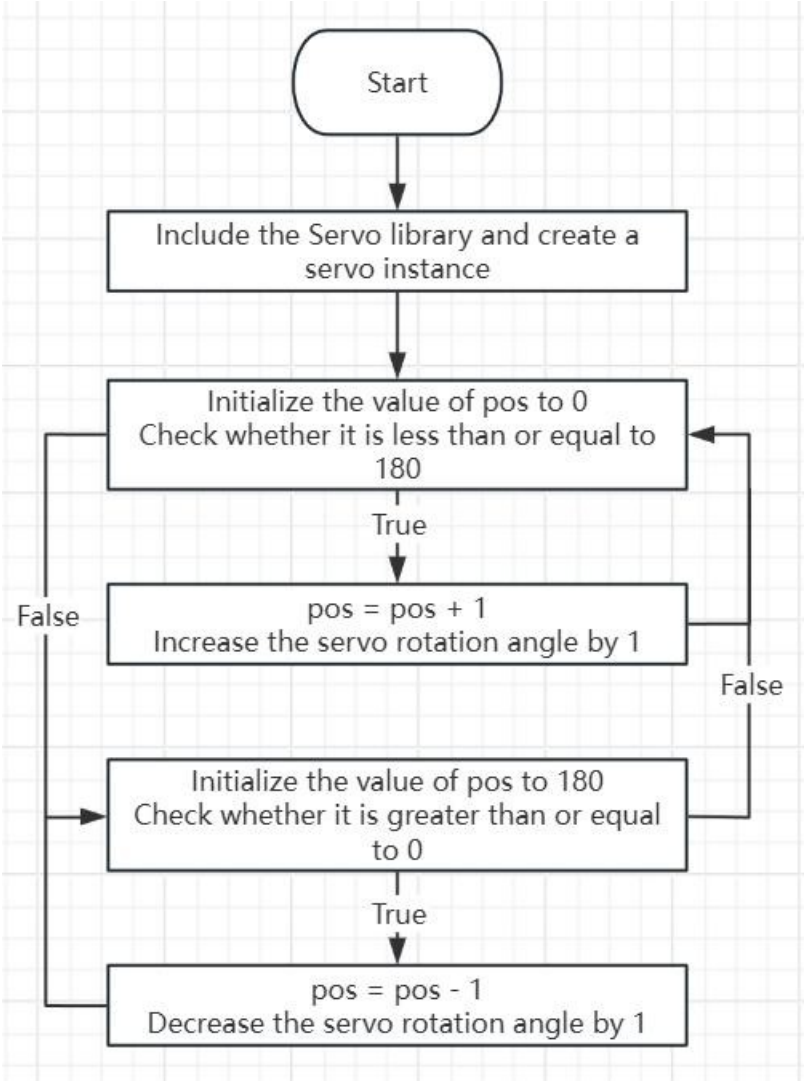
```
void loop() {
  for (pos = 0; pos <= 180; pos += 1) {
    myservo.write(pos);
    delay(15);
  }
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

for (pos = 0; pos <= 180; pos += 1): Premenná pos je inicializovaná na hodnotu 0. Pri každej iterácii cyklu podmienka skontroluje, či je pos stále menšia alebo rovná 180. Ak áno, pos sa zvýši o 1. To spôsobí, že pos sa bude postupne zvyšovať od 0, až kým nedosiahne 181, v čom sa cyklus ukončí.

myservo.write(pos): Tento riadok sa spustí 180-krát, čím sa servu nariadi, aby sa plynulo pohybovalo od 0 do 180 stupňov.

Nasledujúca slučka for funguje rovnakým spôsobom a poháňa servo späť z 180 stupňov na 0.

Celkový diagram logického toku kódu



Kroky na nahratie programu

Tento projekt vyžaduje ďalšie externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

#include	Keď importujete externú knižnicu, jej obsah sa skopíruje priamo do váš program v pôvodnej podobe pred kompiláciou.

Lekcia 09 – Ultrazvukový senzor

Úvod

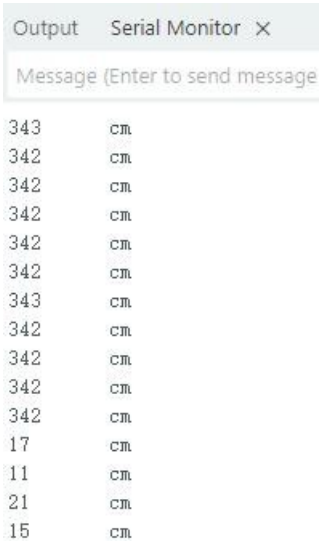
V tejto lekcii sa naučíme, ako používať ultrazvukový senzor. Ako všetci vieme, ultrazvukové moduly sa bežne používajú na meranie vzdialenosti – ale zamysleli ste sa niekedy nad tým, ako vlastne ako zistiť vzdialenosť? Na konci tejto lekcie pochopíte princíp fungovania ultrazvukového snímania vzdialenosti a pomocou kódu budete ovládať hardvér tak, aby vysielal zvukové vlny, prijímal odrazený signál a na základe odozvy vypočítal vzdialenosť.

Ciele výučby

1. Porozumieť princípu fungovania ultrazvukového merania vzdialenosti
2. Naučiť sa definovať a volať funkcie
3. Napísať a spustiť funkciu ultrazvukového merania vzdialenosti, aby bolo možné snímať vzdialenosť v reálnom čase.

Náhľad výsledku

Po nahratí programu do Arduina Nano R4 začne systém v reálnom čase monitorovať vzdialenosť pred vozidlom. Ultrazvukový senzor nepretržite meria vzdialenosť medzi sebou a akoukoľvek prekážkou a výsledky sa zobrazujú v reálnom čase v sériovom monitore.



Hardvér použitý v tejto lekcii

Ultrazvukový modul používa dva hlavné ovládacie piny: D6 a D7

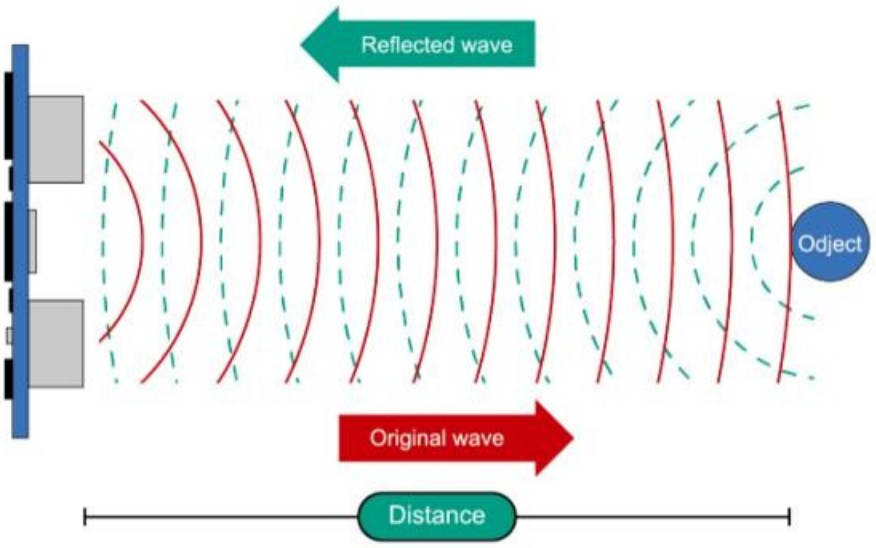
D6 (Trig) je spúšťací pin, zodpovedný za vysielanie ultrazvukových signálov: Keď mu program pošle krátky impulz HIGH, modul vysiela sériu ultrazvukových vĺn smerom dopredu.

D7 (Echo) je pin pre echo, ktorý prijíma ultrazvukové signály odrazené späť od prekážky: Modul vypočíta vzdialenosť k prekážke na základe časového intervalu medzi odoslaním a prijatím signálu.

V tejto lekcii využijeme tieto dva piny na prenos a príjem signálu a pomocou vzorca pre výpočet času letu vypočítame skutočnú vzdialenosť medzi ultrazvukovým senzorom a prekážkou pred ním.

Princíp fungovania modulu ultrazvukového senzora

Ako je znázornené na nižšie uvedenom obrázku, keď je ultrazvukový senzor v prevádzke, najskôr vysiela sériu ultrazvukových vĺn cez vysielací pin (Trig). Keď tieto zvukové vlny narazia na prekážku pred senzorom, odrazia sa späť a zachytí ich prijímací pin (Echo).



Ako funguje ultrazvukový senzor:

1. Fáza vysielania

Program aktivuje vysielací pin senzora, aby vyslal krátky ultrazvukový impulz, a zaznamená presný okamih, kedy je tento impulz vyslaný (čas prenosu).

2. Šírenie a odraz

Ultrazvuková vlna sa šíri vzduchom. Keď narazí na objekt pred senzorom, odrazí sa späť.

3. Fáza príjmu

Hneď ako prijímací pin zaznamená spätný odraz, program okamžite zaznamená aktuálny čas (čas prijatia).

4. Výpočet časového rozdielu

Výpočtom rozdielu medzi časom vysielania a časom prijatia získame celkový čas prechodu ultrazvukovej vlny tam a späť.

Vzorec na výpočet vzdialenosti: Vzdialenosť = (Rýchlosť zvuku * Celkový čas preletu) / 2

Je dôležité poznamenať, že zvuková vlna prekonáva cestu tam a späť – k prekážke a späť –, takže výsledok je potrebné vydeliť 2, aby sme získali skutočnú vzdialenosť od objektu. Rýchlosť zvuku vo vzduchu je približne 343 m/s.

Ultrazvukové meranie vzdialenosti

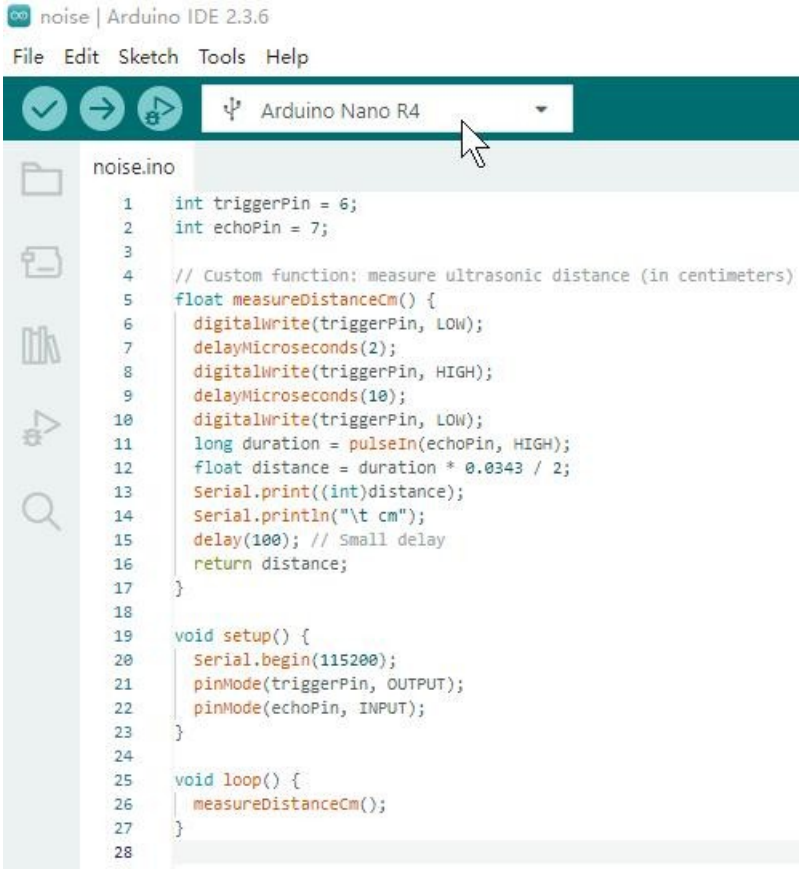
Než sa pustíte do vysvetľovania kódu, môžete si najskôr stiahnuť kompletný program. Odkaz na stiahnutie kompletného kódu:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/9_Ultrasonic_Sensor

Otvorte priečinok „9_Ultrasonic_Sensor“ v prostredí Arduino IDE a potom otvorte súbor „9_Ultrasonic_Sensor.ino“, ktorý sa v ňom nachádza.

Vysvetlenie kľúčových kódov

Teraz si prejdime príklad projektu: **Ultrazvukové meranie vzdialenosti**



Najskôr definujeme piny potrebné pre túto lekciu:

```
int triggerPin = 6;
int echoPin = 7;
```

triggerPin: Toto je spúšťací (vysielací) pin, pripojený k D6 na Arduino Nano R4. echoPin: Toto je echo (prijímací) pin, pripojený k D7 na Arduino Nano R4.

Definujte funkciu merania vzdialenosti pomocou ultrazvuku pomocou typu float (kľúčové zameranie).

```
float measureDistanceCm() {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2); digitalWrite(triggerPin,
  HIGH); delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  long duration = pulseIn(echoPin, HIGH); float
  distance = duration * 0.0343 / 2;
  Serial.print((int)distance); Serial.println("\t
  cm");
  delay(100); // Malé oneskorenie
  return distance;
}
```

Kľúčové body, ktoré je potrebné pochopiť:

Z kódu môžeme vidieť nasledujúce kroky:

①Najskôr sa „triggerPin“ nastaví na úroveň LOW a udrží sa po dobu 2 mikrosekúnd. Tým sa zabezpečí, že pin je v stabilnom stave LOW pred odoslaním spúšťacieho impulzu.
②Ďalej sa „triggerPin“ nastaví na HIGH a udrží sa na tejto úrovni po dobu 10 mikrosekúnd. Tento krok iniciuje ultrazvukový impulz. Pre väčšinu ultrazvukových modulov je 10 mikrosekúnd minimálna požadovaná šírka spúšťacieho

impulzu.
③Po uplynutí 10 mikrosekúnd sa výstup opäť nastaví na stav LOW, čím sa impulz ukončí a dokončí sa jeden ultrazvukový prenos.

delayMicroseconds(2): odloží vykonanie o 2 mikrosekundy delay(2):

odloží vykonanie o 2 milisekundy

long duration = pulseIn(echoPin, HIGH);

"long" môže ukladať väčšie celé čísla ako "int", preto používame "long" na definovanie premennej "duration", keďže pracujeme s číslami na úrovni mikrosekúnd.

pulseIn() je funkcia Arduina, ktorá meria, ako dlho pin zostáva na určitej úrovni napätia. Tu ju používame s „echoPin“ a „HIGH“, čo znamená, že meria dobu, počas ktorej „echoPin“ zostáva na úrovni HIGH. Potom, čo ultrazvukový modul vysiela impulz, „echoPin“ prejde do stavu HIGH a keď sa odrazená zvuková vlna vráti, „echoPin“ prejde do stavu LOW. Meranie doby trvania stavu HIGH „echoPin“ tak poskytuje celkový čas, ktorý ultrazvukový impulz potrebuje na cestu k prekážke a späť.

float distance = duration * 0.0343 / 2;

Typ float sa používa na definovanie premennej s plávajúcou desatinnou čiarkou (čísla s desatinnými miestami). Keďže výpočty zahŕňajú desatinné hodnoty, definujeme premennú ako float. Použitie typov int alebo long by umožnilo ukladať iba celé čísla, pričom by sa zlikvidovala desatinná časť a došlo by k strate presnosti. Použitie typu float zachováva desatinné miesta, čím sa meranie vzdialenosti stáva presnejším.

duration je čas, za ktorý ultrazvukový impulz prejde k prekážke a späť.

0,0343 predstavuje rýchlosť zvuku prepočítanú na centimetre za mikrosekundu: 343 metrov za sekundu sa rovná 0,0343 cm/µs.

vrátiť vzdialenosť;

Funkcia vráti hodnotu – konečnú nameranú vzdialenosť. Po volaní funkcie „measureDistanceCm“ môžeme jej vrátenú hodnotu uložiť do premennej, čím získame

zmeranú vzdialenosť.

Inicializačná funkcia

```
void setup() { Serial.begin(115200);
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
```

Poznámka: V tomto prípade je „triggerPin“ výstupný pin, zatiaľ čo „echoPin“ je vstupný pin. Pri nastavovaní režimov pinov sa uistite, že ich nezamieňate.

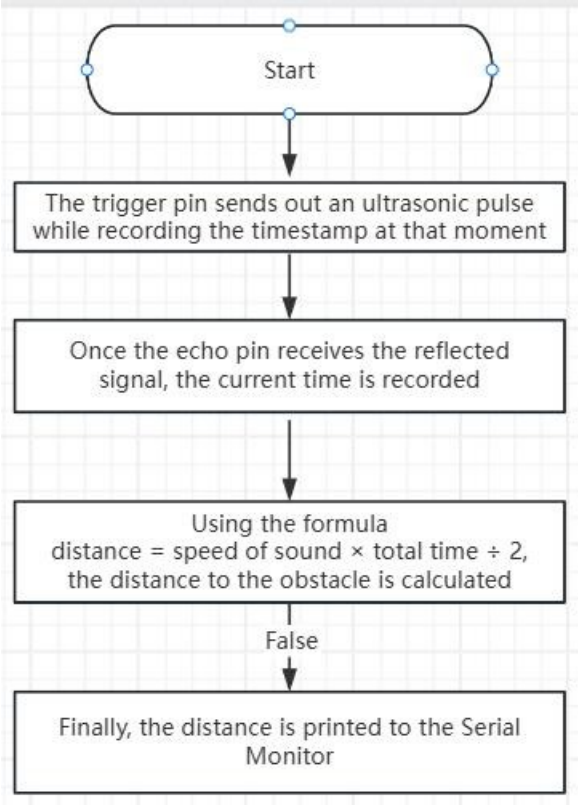
Funkcia loop

```
void loop() {
```

```
  measureDistanceCm();
}
```

V funkcii loop stačí len zavolať predtým definovanú funkciu „measureDistanceCm“, aby sa spustil kód na meranie vzdialenosti pomocou ultrazvuku.

Celkový diagram logického toku kódu



Postup nahratia programu

Podrobné pokyny na nahratie nájdete v časti „Kroky na nahratie“ na strane 8.

Kľúčové body:

delayMicroseconds()	Používa sa na pozastavenie vykonávania na určitý počet mikrosekúnd; je presnejšie ako delay()
pulseIn()	Meranie trvania impulzu na pine; vrátená hodnota je v mikrosekundách
long	Typ Integer používaný na ukladanie veľkých celých čísel
float	Typ s pohyblivou desatinnou čiarkou používaný na ukladanie čísel s desatinnými miestami
void	Používa sa na definovanie typu návratovej hodnoty funkcie. void znamená, že funkcia nevráti žiadnu hodnotu
float function()	Definujte funkciu, ktorá vráti hodnotu s plávajúcou desatinnou čiarkou: Názov funkcie je funkcia
return distance	Vráti hodnotu premennej distance z vnútra funkcie

Lekcia 10 --- Digitálny displej

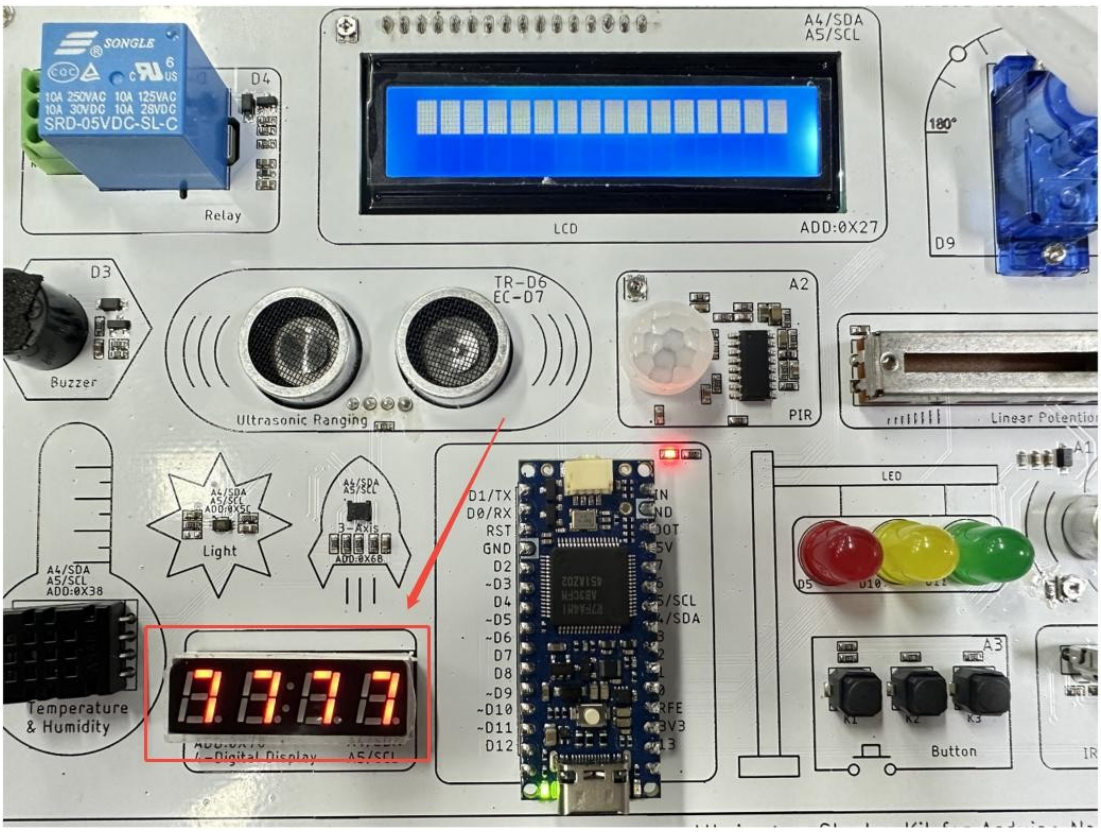
Úvod

V tejto lekcii sa naučíme, ako ovládať 7-segmentový displej na vývojovej doske „Arduino Nano R4“. Na ovládanie displeja použijeme externú knižnicu „HT16K33.h“ a prejdeme si komplexný praktický projekt: 7-segmentový odpočítavací časovač. Na konci tejto lekcie pochopíte základy zobrazovania čísel na 7-segmentovom displeji a naučíte sa, ako implementovať bežné efekty zobrazenia, ako je prepínanie číslic, blikanie a rozsvietenie desatinných bodiek.

Ciele výučby

1. Porozumieť princípom fungovania 7-segmentového displeja
2. Naučiť sa rozdiel medzi uint8_t a int 3. Pochopiť,
čo je I2C
4. Ovládnuť viacero funkcií na ovládanie 7-segmentového displeja
5. Dokončite projekt odpočítavania s 7-segmentovým displejom

Náhľad výsledku



Po spustení programu sa najskôr inicializuje 7-segmentový displej a potom začne odpočítavať od

8888 do 1111.

Hardvér použitý v tejto lekcii

7-segmentový displej sa nachádza v ľavom dolnom rohu vývojovej dosky. Je pripojený k doske cez I2C s adresou 0x70.

I2C je protokol synchrónnej sériovej zbernice používaný na komunikáciu medzi integrovanými obvodmi. Na prenos dát využíva jednu hodinovú linku (SCL) a jednu dátovú linku (SDA). Hlavné zariadenie generuje hodinový signál a riadi komunikačný proces, zatiaľ čo podriadené zariadenia komunikujú len po tom, čo ich hlavné zariadenie vyberie na základe ich adries. Viacero zariadení môže zdieľať tú istú zbernicu I2C a rozlišujú sa podľa svojich jedinečných adries, čo umožňuje komunikáciu viacerých zariadení s použitím len veľmi malého počtu pinov.

Prečo používať I2C? Aké sú jeho výhody?

Keď máme veľa periférnych zariadení, ale obmedzený počet dostupných pinov na vývojovej doske, I2C sa stáva ideálnym riešením. Komunikácia I2C vyžaduje len dva piny (SDA a SCL) na komunikáciu s viacerými zariadeniami. Priradením rôznych adries každému zariadeniu môže vývojová doska komunikovať s viacerými modulmi na tej istej zbernici. Táto schopnosť ovládať viacero zariadení s použitím minimálneho počtu pinov je najväčšou výhodou I2C.

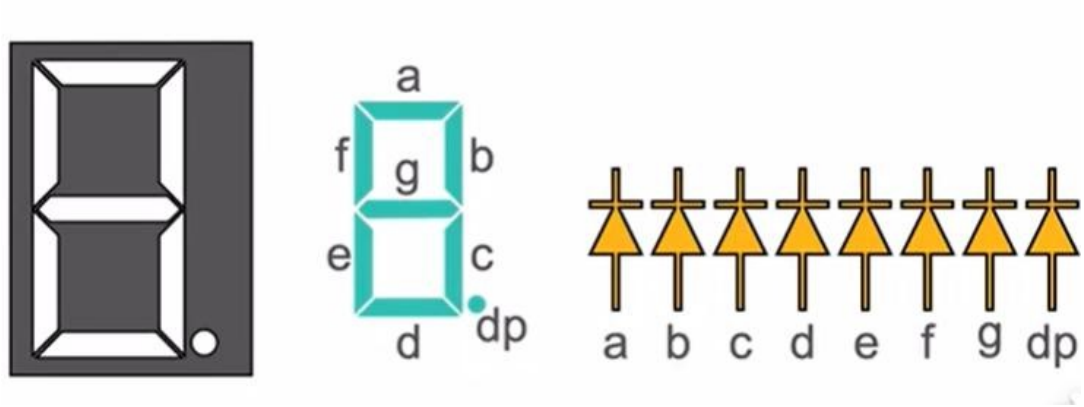
Adresa I2C

Ako teda I2C rozlišuje medzi rôznymi zariadeniami? Na zbernici I2C je každému zariadeniu pridelená jedinečná adresa I2C. Keď je k tej istej zbernici pripojených viacero zariadení, práve tieto adresy umožňujú systému rozlíšiť ich. Počas inicializácie zvyčajne definujeme a inicializujeme adresu zariadenia spolu s príslušným objektom. Následne môže Arduino pomocou adresy I2C presne komunikovať s konkrétnym zariadením.

Princíp fungovania modulu digitálneho displeja

7-segmentový displej je zobrazovacie zariadenie pozostávajúce zo siedmich osvetlených segmentov (a, b, c, d, e, f, g) a jednej desatinnej čiarky (dp). Číslica v tvare „8“ znázornená na obrázku je vytvorená kombináciou týchto siedmich segmentov. Každý segment je nezávislá svetelná jednotka (napríklad LED), ktorú

možno ovládať individuálne – zapínať alebo vypínať podľa potreby.



Princíp fungovania: „Rozsvietením konkrétnych segmentov sa vytvárajú čísla“
7-segmentový displej zobrazuje rôzne čísla zapínaním konkrétnych segmentov tak, aby vytvorili tvar každej číslice.

Na zobrazenie čísla „8“: rozsvietite všetkých sedem segmentov – a, b, c, d, e, f, g. **Na zobrazenie čísla „0**“: rozsvietite segmenty a, b, c, d, e, f a segment g nechajte zhasnutý. **Na zobrazenie čísla „1**“: rozsvietite len segmenty b a c.

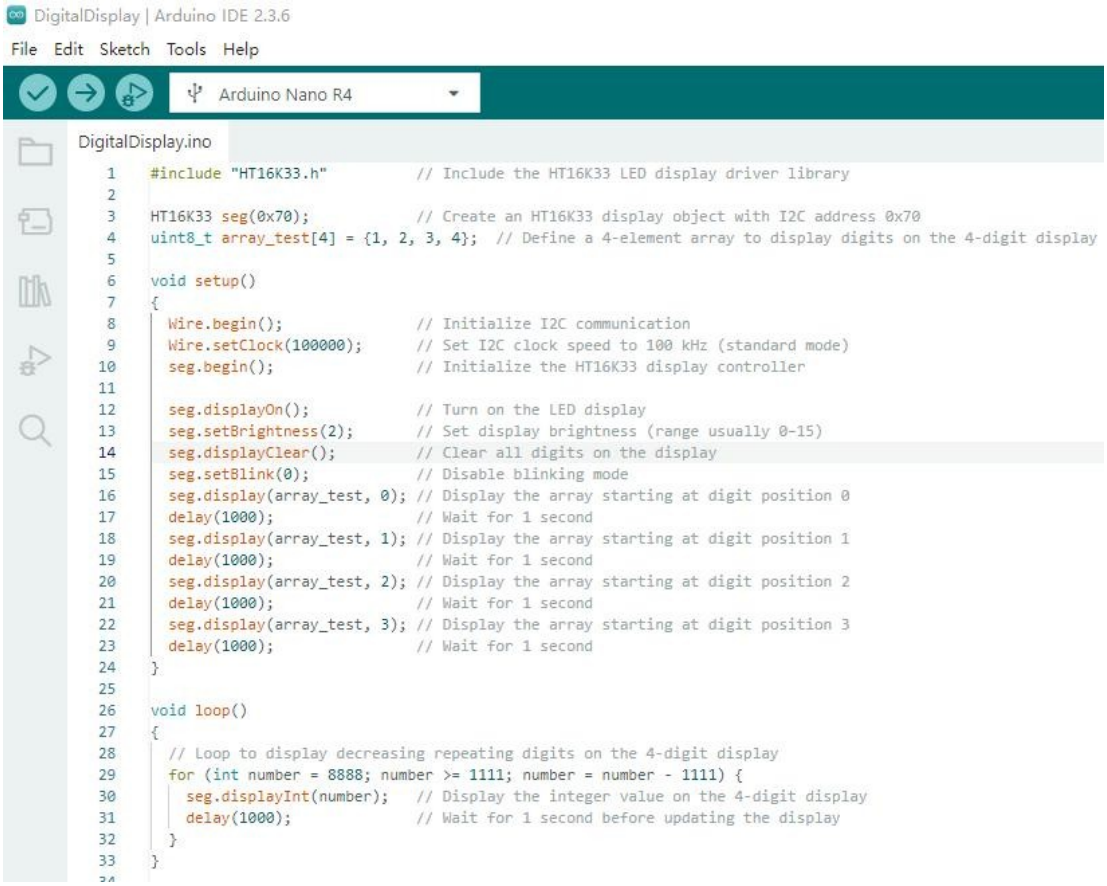
Na **zobrazenie desatinnej čiarky**: jednoducho rozsvietite segment „dp“.

Odpočítavanie na 7-segmentovom displeji

Než sa pustíme do vysvetľovania kódu, môžete si najskôr stiahnuť celý program. Odkaz na stiahnutie kompletného kódu:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/10_DigitalDisplay
Otvorte priečinok „10_DigitalDisplay“ v prostredí Arduino IDE a potom otvorte súbor „10_DigitalDisplay.ino“, ktorý sa v ňom nachádza.

Vysvetlenie kľúčových častí kódu

Teraz si prejdime príklad projektu: Odpočítavanie na 7-segmentovom displeji.



Najskôr naimportujeme knižnicu ovládača 7-segmentového displeja:

```
#include "HT16K33.h"
```

Knižnica ovládača displeja „HT16K33.h“ zapuzdruje nízkoúrovňovú komunikáciu I2C s čipom „HT16K33“ a predstavuje kľúčový súbor pre túto lekciu. Všetky metódy použité v kóde na ovládanie 7-segmentového displeja poskytuje práve táto knižnica.

Vytvorenie objektu displeja

```
HT16K33 seg(0x70);
```

Vytvorte objekt s názvom „seg“ na ovládanie 7-segmentového displeja „HT16K33“ s adresou I2C „0x70“. Týmto sa Arduino IDE informuje, že vždy, keď použijeme názov „seg“, ovládame displej „HT16K33“ umiestnený na adrese „0x70“.

Definujte testovacie pole

```
uint8_t array_test[4] = {1, 2, 3, 4};
```

Tu si musíme jasne uvedomiť rozdiel medzi typmi „uint8_t“ a „int“:

1. Rozsah hodnôt

„uint8_t“: 0–255

„int“: –32768–32767

2. Využitie pamäte

„uint8_t“ využíva menej pamäte ako **„int“**.

3. Rozdiel v znamienku

„uint8_t“ je bez znamienka, čo znamená, že môže reprezentovať iba kladné hodnoty.

„int“ je typ so znamienkom a môže reprezentovať kladné aj záporné hodnoty.
Ak sú hodnoty malé a nikdy nie sú záporné, „uint8_t“ je dobrá voľba a bežne sa používa pre údaje súvisiace so hardvérom.
Keď môžu byť hodnoty väčšie a môžu zahŕňať kladné aj záporné čísla, na numerické výpočty sa zvyčajne používa typ „int“.
Tu definujeme pole špeciálne na ovládanie 7-segmentového displeja. Keďže každý prvok musí reprezentovať iba hodnoty od **0 do 9**, používame typ **„uint8_t**“. To plne spĺňa požadovaný rozsah hodnôt a zároveň umožňuje základným knižničným funkciám pracovať efektívne a

priamo – hlavne preto, že funkcia **„display“** prijíma iba dáta vo formáte **„uint8_t**“.

Inicializačná funkcia

```
void setup()
{
  Serial.begin(115200);
  Wire.begin();
  Wire.setClock(100000);
```

Wire.begin(): Inicializuje zbernicu I2C v Arduine. Túto funkciu je potrebné vyvolať pred použitím modulu „HT16K33“.
Wire.setClock(100000): Nastaví rýchlosť hodín I2C na 100 kHz.
Tieto dva riadky slúžia na inicializáciu zbernice I2C. Potom pokračujeme v inicializácii ovládača 7-segmentového displeja.

Inicializujte modul 7-segmentového displeja, aby bol pripravený na ďalšie použitie

```
  seg.begin();
```

Zapnite 7-segmentový displej

```
  seg.displayOn();
```

Nastavte jas displeja na 2. Úroveň jasu je možné nastaviť v rozmedzí od 0 do 10

```
  seg.setBrightness(2);
```

Pred spustením nového cyklu zobrazenia vymažte predchádzajúcu vyrovnávaciu pamäť zobrazenia

```
  seg.displayClear();
```

Nastavte režim blikania na neblikanie. Ak je nastavené na 1, displej bude blikať.

```
  seg.setBlink(0);
```

Otestujte údaje v poli, vrátane špeciálnych symbolov, ako sú úvodzovky, na 7-segmentovom displeji. (Dôležité)

```
  seg.display(array_test, 0);
  delay(1000);
  seg.display(array_test, 1);
  delay(1000);
  seg.display(array_test, 2);
  delay(1000);
```

```
  seg.display(array_test, 3);
  delay(1000);
}
```

Tu sa líši len posledný parameter. Všetky predchádzajúce parametre určujú zobrazenie údajov z poľa „array_test“, zatiaľ čo posledný parameter určuje, ktorý symbol sa zobrazí na 7-segmentovom displeji sa zobrazí. Hodnoty od 0 do 3 zodpovedajú rôznym interpunkčným znamienkam, ako je znázornené nižšie.

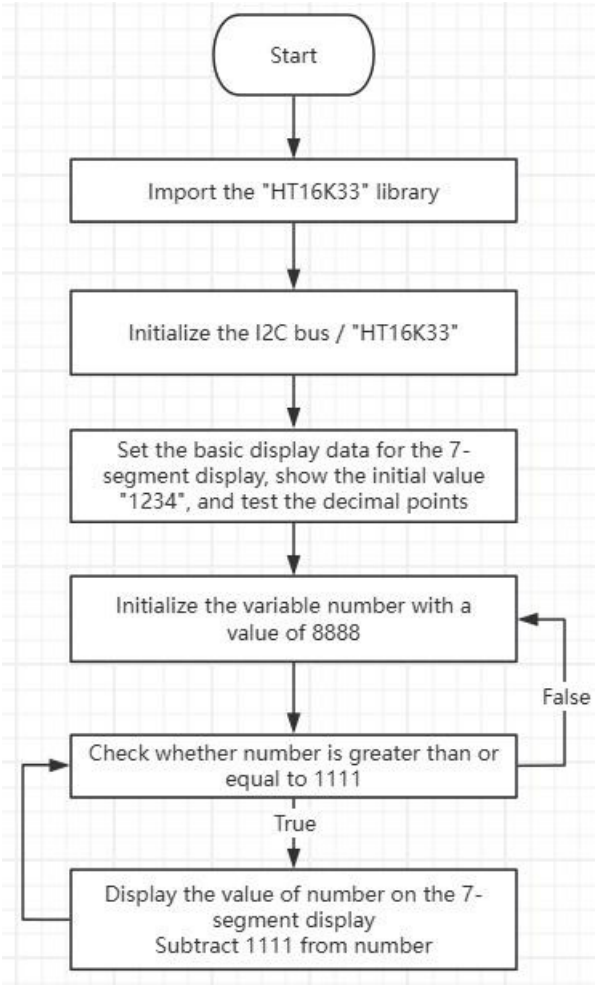
Poznámka: Interpunkčný znak úplne vľavo sa nepoužíva.

Funkcia hlavnej slučky

```
void loop()
{
  // Slučka na zobrazenie klesajúcich opakujúcich sa číslic na 4-miestnom
  displeji for (int number = 8888; number >= 1111; number = number - 1111) {
    seg.displayInt(number);
    delay(1000);
  }
}
```

V hlavnej slučke používame slučku for a metódu „displayInt“ na odpočítavanie od 8888 do 1111. Metóda „displayInt“ prijíma ako parameter hodnotu typu „int“, čo umožňuje pohodlne znižovať hodnotu o 1111 pri každej iterácii slučky.

Celkový diagram logiky kódu



Kroky na nahratie programu

Tento projekt vyžaduje dodatočné externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

Wire.begin()	Inicializácia zbernice I2C
seg.displayOn()	Zapnutie 7-segmentového displeja
seg.setBrightness(2)	Nastavte jas displeja na 2
seg.displayClear()	Vymazať vyrovnávaciu pamäť displeja
seg.setBlink(0)	Nastavte displej do režimu bez blikania
seg.display(array,0)	Zobraziť hodnoty prednastaveného poľa na 7-segmentovom displeji, pričom sa zobrazí index interpunkčného znamienka 0
seg.displayInt(8888)	Zobraziť celočíselnú hodnotu 8888 na 7-segmentovom displeji

Lekcia 11 --- LCD

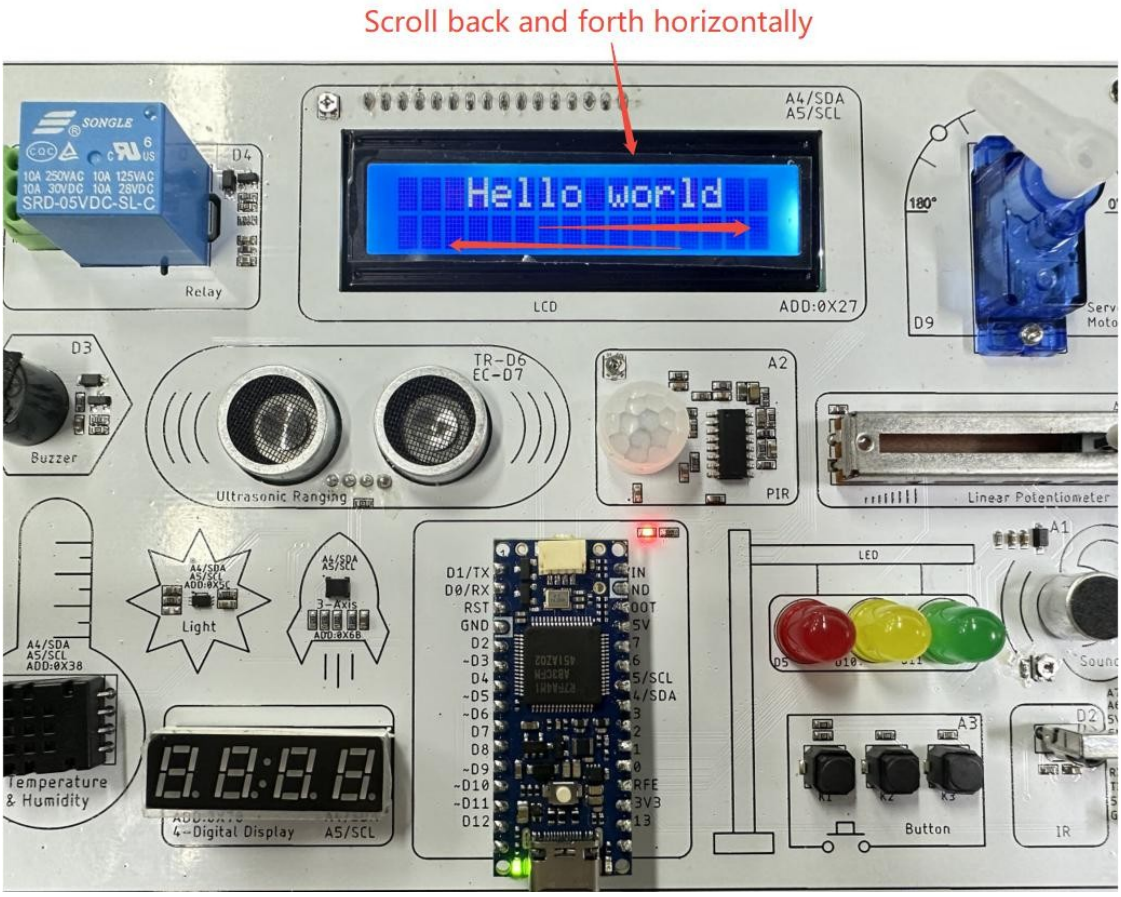
Úvod

V tejto lekcii sa naučíme, ako ovládať LCD displej na vývojovej doske „Arduino Nano R4“. Použijeme cyklus for, ktorý ste sa naučili skôr, aby sme text posúvali doľava a doprava po LCD obrazovke. Na konci tejto lekcie pochopíte, ako ovládať LCD displej, a zoznámite sa s niekoľkými bežnými technikami ovládania LCD.

Ciele

1. Porozumieť fungovaniu LCD displeja
2. Ovládať viacero metód ovládania LCD displeja
3. Vykonajte experiment, pri ktorom sa na LCD obrazovke bude text „Hello World“ posúvať doľava a doprava

Náhľad výsledku



Po spustení programu sa text „Hello World“ bude posúvať sem a tam po LCD obrazovke.

Hardvér použitý v tejto lekcii

LCD displej sa nachádza v hornej časti vývojovej dosky a je pripojený cez rozhranie I2C dosky s adresou 0x27.

V ľavom hornom rohu LCD displeja sa nachádza otočný gombík. **Tento gombík slúži hlavne na nastavenie kontrastu obrazovky.**

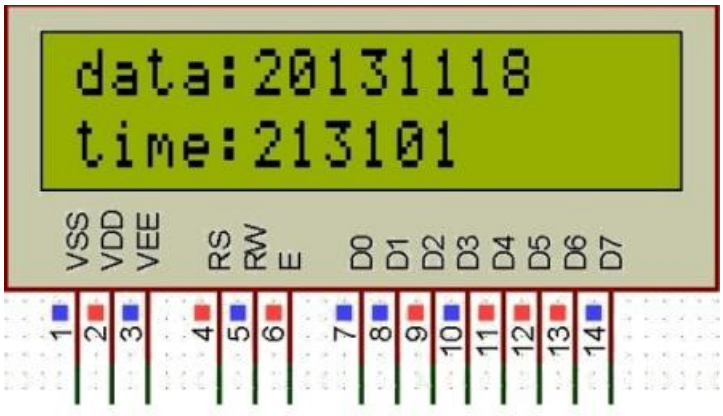
Otáčaním gombíka v smere hodinových ručičiek (dopredu) sa kontrast postupne znižuje. Znaky sa stanú svetlejšími a nakoniec môžu zmiznúť, čím bude obrazovka vyzerať prázdne.
Otáčaním gombíka proti smeru hodinových ručičiek (zadný chod) sa kontrast zvyšuje. Znaky budú tmavšie, ale ak budete gombíkom otáčať ďalej, na obrazovke sa namiesto čitateľného textu môžu zobraziť súvislé bloky.

Princíp fungovania LCD modulu

LCD displej na doske Arduino Nano R4 je modul riadený rozhraním I2C. Funguje to takto: mikrokontrolér posiela dáta do čipu rozširujúceho vstupy a výstupy cez zbernicu I2C pomocou iba dvoch vedení, SCL a SDA. Rozširujúci čip prevádza sériové dáta I2C na paralelný signál, ktorý ovláda vedenia RS, RW, E

a dátové linky D4 – D7. Ovládač LCD potom používa tieto signály na rozlíšenie medzi

príkazy a znakové dáta, zapíše kódy znakov do svojej internej pamäte a nakoniec ovláda pixely z tekutých kryštálov, aby zmenili svoju priepustnosť svetla. Pri zapnutom podsvietení sa na obrazovke zobrazia príslušné znaky. Táto konštrukcia umožňuje ovládať LCD 1602 pomocou oveľa menšieho počtu I/O pinov.

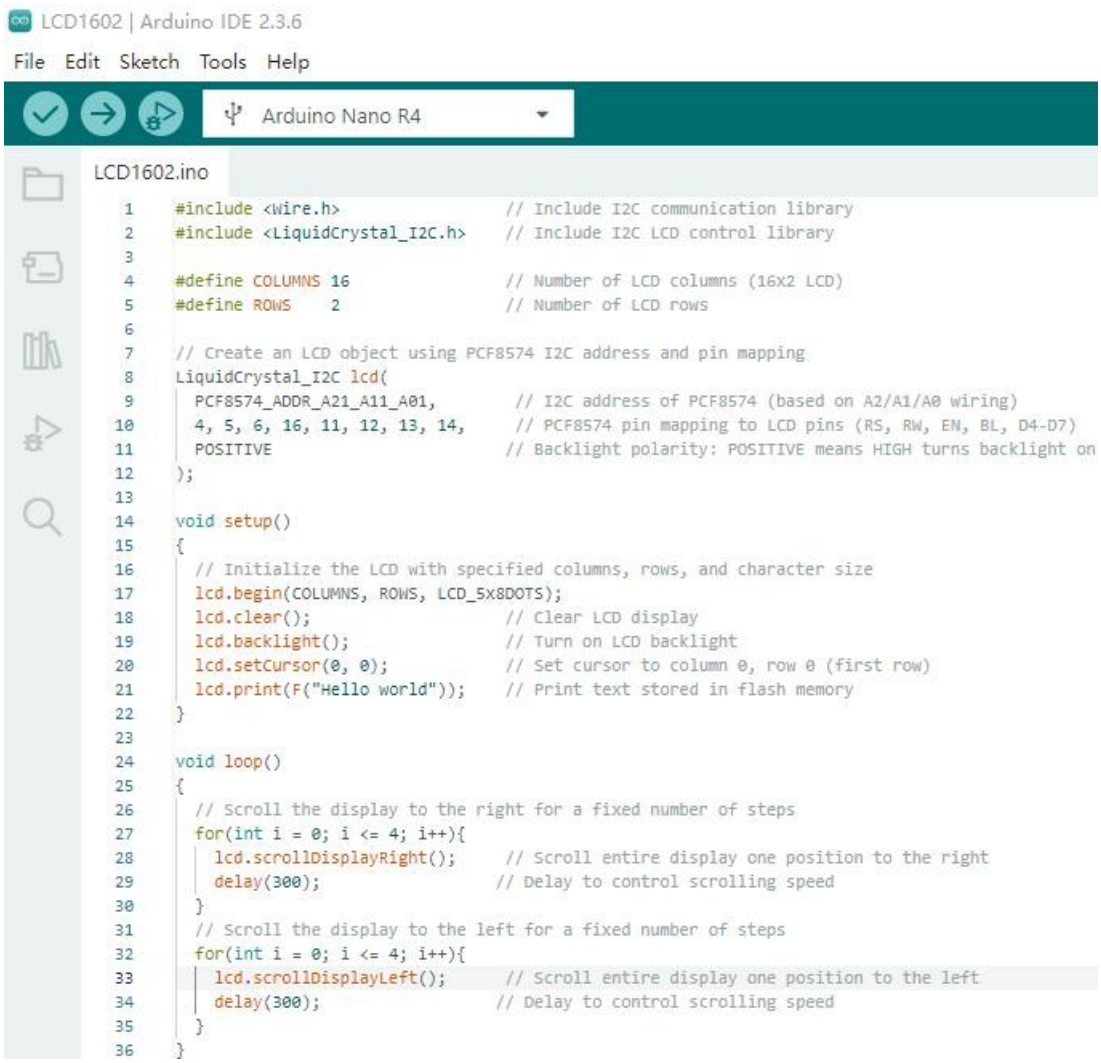


Posúvanie textu na LCD

Než sa pustíte do kódu, môžete si najskôr stiahnuť program. Kompletný kód je k dispozícii na nasledujúcom odkaze:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_c ode/11_LCD
Otvorte súbor „11_LCD.ino“ nachádzajúci sa v priečinku „11_LCD“ pomocou prostredia Arduino IDE.

Vysvetlenie kódov klávesov

Teraz si prejdime príklad: posúvanie textu na LCD.



Najprv zahrnieme potrebné hlavičkové súbory. V tejto lekcii budeme pokračovať v používaní I2C, takže musíme importovať komunikačnú knižnicu I2C aj knižnicu LCD.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Knižnica „LiquidCrystal_I2C.h“ je špeciálne navrhnutá pre znakové LCD displeje, ktoré používajú rozhranie I2C.

Ďalej definujeme počet stĺpcov a riadkov pre LCD

```
#define COLUMNS 16
#define ROWS 2
```

„COLUMNS“ a „ROWS“ predstavujú rozmery LCD displeja – 16 stĺpcov a 2 riadky. Vytvorte

objekt LCD (tento krok je kľúčový)

```
LiquidCrystal_I2C lcd(
  PCF8574_ADDR_A21_A11_A01, 4, 5,
  6, 16, 11, 12, 13, 14, POSITIVE
);
```

Táto časť kódu je mimoriadne dôležitá, pretože všetky nasledujúce operácie s LCD sa vykonávajú prostredníctvom nej. Prejdime si parametre jeden po druhom: **PCF8574_ADDR_A21_A11_A01:** Predstavuje adresu I2C čipu PCF8574. Môžete ju tiež napísať priamo ako 0x27. Makro „PCF8574_ADDR_A21_A11_A01“ je už definované ako 0x27 v súbore „LiquidCrystal_I2C.h“.

(4, 5, 6, 16, 11, 12, 13, 14): Tieto hodnoty priraďujú piny rozširujúcej dosky k riadiacim a dátovým pinom LCD displeja. Toto priradenie musí zodpovedať skutočnému zapojeniu hardvéru; v opačnom prípade sa obraz na LCD displeji nebude zobrazovať správne. Na doske Arduino Nano R4 je toto priradenie pinov pevne dané a nesmie sa meniť.

POSITIVE:Toto zapne podsvietenie. Ak je nastavené na „NEGATIVE“, podsvietenie sa vypne. Inicializačná

funkcia

```
void setup()
{
  lcd.begin(COLUMNS, ROWS, LCD_5x8DOTS);
  lcd.clear();                          // Vymazať displej LCD
  lcd.backlight();                      // Zapnúť podsvietenie LCD
  lcd.setCursor(0, 0);                  // Nastav kurzor na stĺpec 0, riadok 0 (prvý
  riadok) lcd.print(F("Hello world"));                    // Vytlač text uložený vo
  flash pamäti
}
```

V inicializačnej funkcii nakonfigurujeme LCD tak, aby bolo pripravené na správne zobrazenie textu.

lcd.begin(): Informuje LCD, že veľkosť obrazovky je 16 × 2 znaky, s bodovou maticou 5 × 8 na znak.

lcd.clear(): Vymaže akýkoľvek zvyšný obsah zo zapnutia; inak sa môžu zobraziť náhodné alebo nečitateľné znaky.

lcd.backlight(): Zapne podsvietenie LCD displeja. Bez neho nemusí byť text viditeľný, aj keď sa zobrazuje.

lcd.setCursor(0,0): Nastaví polohu kurzora, začínajúc v stĺpci 0, riadku 0.

lcd.print(F("Hello world")): Zobrazí reťazec „Hello world“ na LCD displeji. Makro „F“ ukladá reťazec do pamäte Flash, takže nespotrebúva pamäť SRAM.

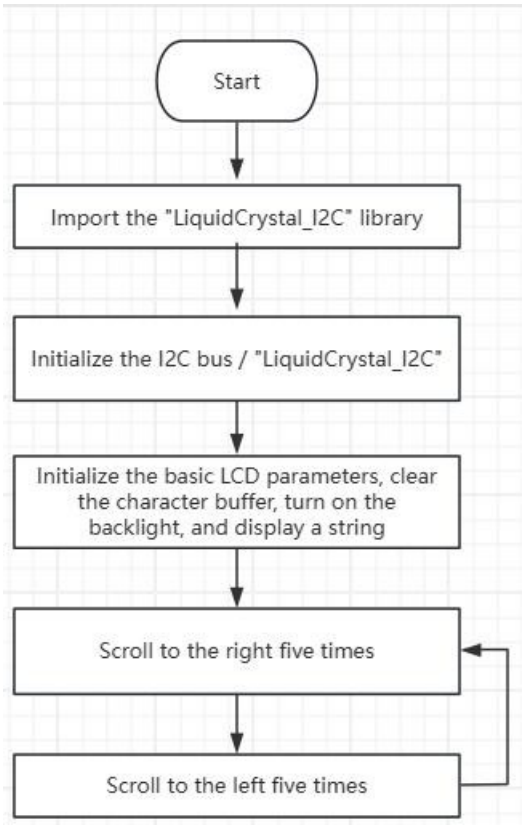
Funkcia slučky

```
void loop()
{
  for(int i = 0; i <= 4; i++){
    lcd.scrollDisplayRight();          // Posuň celý displej o jednu pozíciu doprava
    delay(300);                        // Oneskorenie na ovládanie rýchlosti posúvania
  }
  for(int i = 0; i <= 4; i++){
    lcd.scrollDisplayLeft();
    delay(300);                        // Posunúť celý displej o jednu pozíciu doľava
  }                                    // Oneskorenie na reguláciu rýchlosti posúvania
}
```

Hlavná slučka zodpovedá za vytvorenie efektu posúvania doľava a doprava. V každom smere sa vykoná 5 cyklov, pretože text „Hello world“ obsahuje 11 znakov a LCD displej má 16 stĺpcov. To

znamená, že text narazí na okraje obrazovky po posunutí o 5 pozícií doľava alebo doprava.

Celkový diagram logiky kódu



Kroky na nahratie programu

Tento projekt vyžaduje dodatočné externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Postup nahratia“ na strane 8.

Kľúčové body:

lcd.clear()	Vymazať vyrovnávaciu pamäť LCD
lcd.backlight()	Zapnutie podsvietenia LCD
lcd.setCursor(0, 0)	Nastaviť kurzor na stĺpec 0, riadok 0
lcd.print()	Zobraziť reťazec na LCD displeji
lcd.scrollDisplayRight()	Posuňte text doprava
lcd.scrollDisplayLeft()	Posúva text doľava

Lekcia 12 ---6-osový modul

Úvod

V tejto lekcii sa naučíme pracovať so šesťosovým senzorom na vývojovej doske „Arduino Nano R4“. Pomocou externých knižníc „Arduino_LSM6DS3“ a „MadgwickAHRS“ budeme čítať údaje z akcelerometra a gyroskopu, spájať údaje zo senzorov a v reálnom čase zobrazovať uhly orientácie. Na konci tejto lekcie budete vedieť používať šesťosový senzor na základnej úrovni, porozumiete základom odhadu polohy a budete vedieť implementovať periodické vzorkovanie a sériový výstup orientačných údajov prostredníctvom kódu.

Ciele výučby

1. Porozumieť princípom fungovania šesťosového senzora
2. Ovládať rôzne použitia funkcie „Serial.print“
3. Získajte surové údaje o zrýchlení a uhlovej rýchlosti a pomocou Madgwickovho algoritmu vypočítajte a vygenerujte uhly orientácie

Náhľad výsledku

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Nano R4' on 'COM12')

```
Roll =    2 °, Pitch =    5 °, Yaw =  105 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =   26 °, Pitch =    5 °, Yaw =  227 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =   13 °, Pitch =  -14 °, Yaw =  349 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =    4 °, Pitch =    7 °, Yaw =  109 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =   26 °, Pitch =    3 °, Yaw =  231 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =   11 °, Pitch =  -13 °, Yaw =  353 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =    6 °, Pitch =    8 °, Yaw =  114 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =   26 °, Pitch =    0 °, Yaw =  236 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =    9 °, Pitch =  -11 °, Yaw =  358 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =    9 °, Pitch =    9 °, Yaw =  118 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =   25 °, Pitch =   -2 °, Yaw =  240 °
ax =  0.1 g, ay =  1.0 g, az =  4.0 g, gx =    43.2 °/s, gy =   500.0 °/s, gz =  1979.7 °/s
Roll =    7 °, Pitch =   -9 °, Yaw =    2 °
```

Po spustení programu a otvorení sériového monitora uvidíte aktuálne údaje o zrýchlení, uhlovej rýchlosti a vypočítané údaje o uhle orientácie.

Hardvér použitý v tejto lekcii

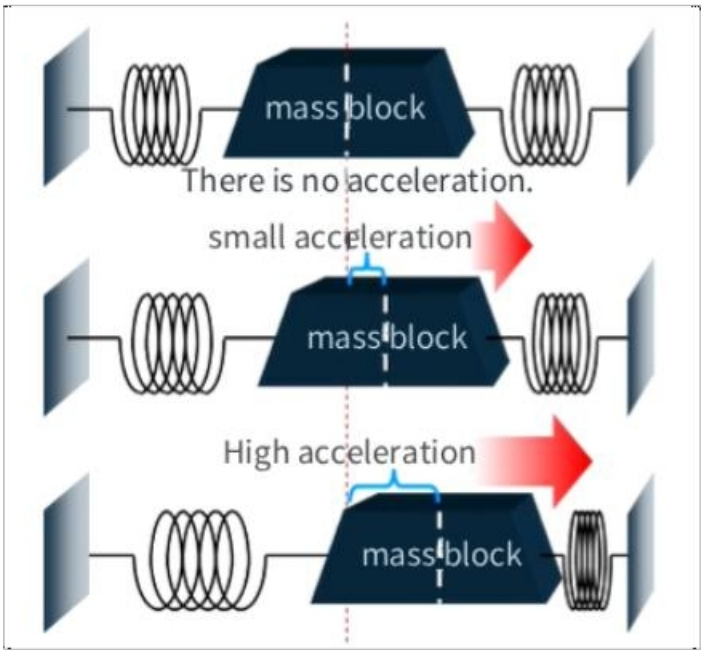
6-osový senzor sa nachádza v ľavom dolnom rohu vývojovej dosky. Je pripojený cez rozhranie I2C dosky a používa adresu 0x6B.

Princíp fungovania 6-osového modulu

Na obrázku nižšie je znázornený princíp fungovania „jednoosového akcelerometra“ vo vnútri šesťosového senzora. Funguje na základe vnútornej referenčnej hmoty a pružín: keď senzor zrýchľuje, zotrvačnosť hmoty spôsobuje deformáciu pružín. Rozdiel v deformácii zodpovedá veľkosti zrýchlenia (bez zrýchlenia je deformácia rovnaká; pri malom zrýchlení je rozdiel malý; pri väčšom zrýchlení sa rozdiel zväčšuje).

šesťosový senzor obsahuje tri takéto osi akcelerometra – po jednej v smere X, Y a Z

(všetky založené na rovnakom princípe znázornenom na obrázku, len orientované inak) – plus tri osí gyroskopu, ktoré merajú uhlovú rýchlosť. Táto viacosová konštrukcia spolu umožňuje detekciu trojrozmerného zrýchlenia a uhlovej rýchlosti.

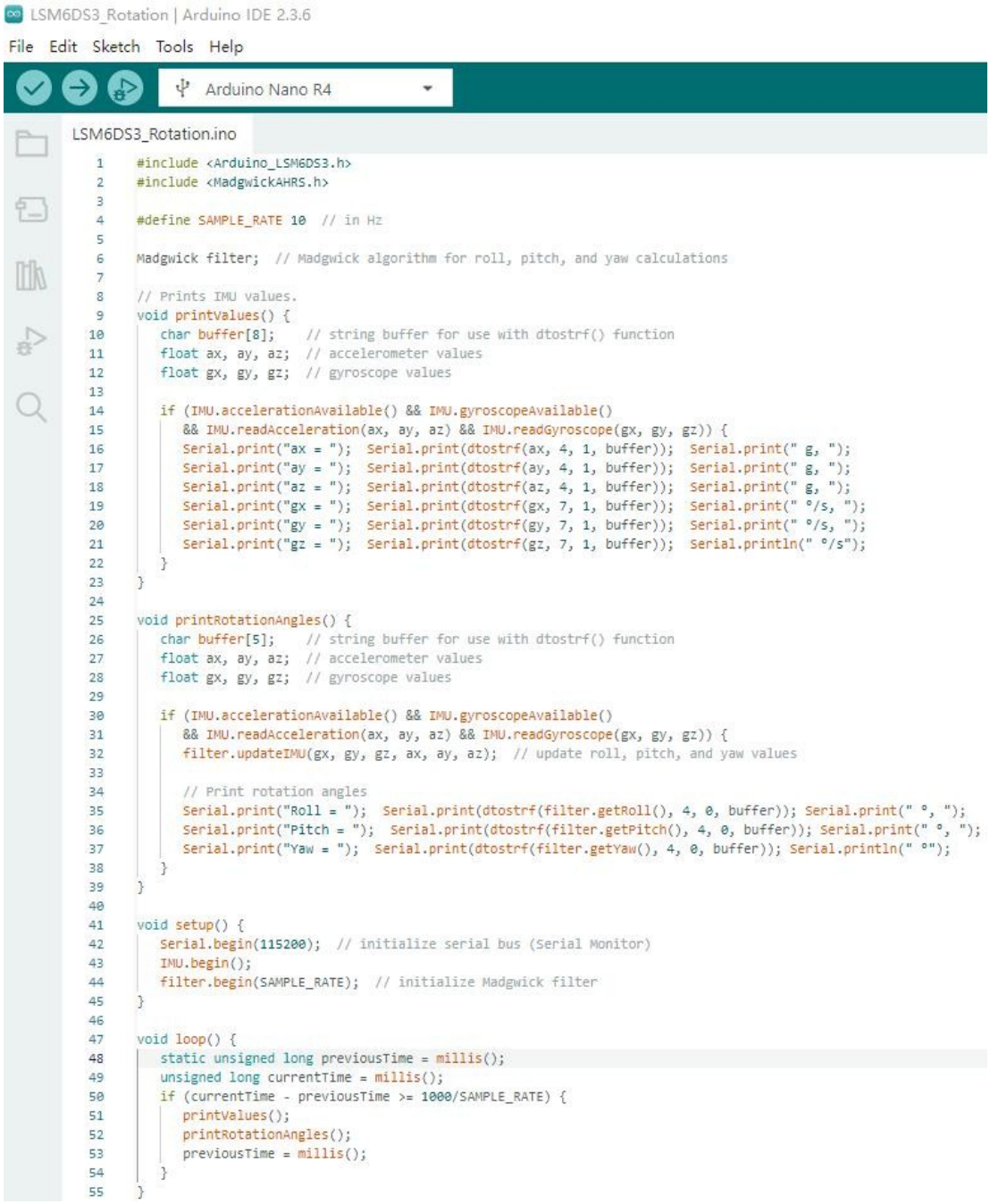


Získavanie a spracovanie údajov zo 6-osového senzora

Než sa pustíte do prehľadu kódu, môžete si najskôr stiahnuť kompletný zdrojový kód. Odkaz na stiahnutie kompletného kódu:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/12_6-Axis
Otvorte súbor „12_6-Axis.ino“ nachádzajúci sa v priečinku „12_6-Axis“ pomocou prostredia Arduino IDE.

Vysvetlenie kľúčových častí kódu

Teraz si prejdime príklad: získavanie a spracovanie údajov zo 6-osového senzora.




```ino
#include <Arduino_LSM6DS3.h>
#include <MadgwickAHRS.h>

#define SAMPLE_RATE 10  // in Hz

Madgwick filter;  // Madgwick algorithm for roll, pitch, and yaw calculations

// Prints IMU values.
void printValues() {
  char buffer[8];    // string buffer for use with dtostrf() function
  float ax, ay, az;  // accelerometer values
  float gx, gy, gz;  // gyroscope values

  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
    && IMU.readAcceleration(ax, ay, az) && IMU.readGyroscope(gx, gy, gz)) {
    Serial.print("ax = ");  Serial.print(dtostrf(ax, 4, 1, buffer));  Serial.print(" g, ");
    Serial.print("ay = ");  Serial.print(dtostrf(ay, 4, 1, buffer));  Serial.print(" g, ");
    Serial.print("az = ");  Serial.print(dtostrf(az, 4, 1, buffer));  Serial.print(" g, ");
    Serial.print("gx = ");  Serial.print(dtostrf(gx, 7, 1, buffer));  Serial.print(" °/s, ");
    Serial.print("gy = ");  Serial.print(dtostrf(gy, 7, 1, buffer));  Serial.print(" °/s, ");
    Serial.print("gz = ");  Serial.print(dtostrf(gz, 7, 1, buffer));  Serial.println(" °/s");
  }
}

void printRotationAngles() {
  char buffer[5];    // string buffer for use with dtostrf() function
  float ax, ay, az;  // accelerometer values
  float gx, gy, gz;  // gyroscope values

  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
    && IMU.readAcceleration(ax, ay, az) && IMU.readGyroscope(gx, gy, gz)) {
    filter.updateIMU(gx, gy, gz, ax, ay, az);  // update roll, pitch, and yaw values

    // Print rotation angles
    Serial.print("Roll = ");  Serial.print(dtostrf(filter.getRoll(), 4, 0, buffer)); Serial.print(" °, ");
    Serial.print("Pitch = ");  Serial.print(dtostrf(filter.getPitch(), 4, 0, buffer)); Serial.print(" °, ");
    Serial.print("Yaw = ");  Serial.print(dtostrf(filter.getYaw(), 4, 0, buffer)); Serial.println(" °");
  }
}

void setup() {
  Serial.begin(115200);  // initialize serial bus (Serial Monitor)
  IMU.begin();
  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();
  }
}
```

Najskôr načítajú oficiálnu knižnicu IMU pre Arduino, ktorá slúži na ovládanie senzora „LSM6DS3“ – známeho aj ako „6-osový“ senzor. Následne načítajú knižnicu „MadgwickAHRS.h“ na implementáciu algoritmu odhadu polohy.

```
#include <Arduino_LSM6DS3.h>
#include <MadgwickAHRS.h>
```

Ďalej definujeme vzorkovaciu frekvenciu, ktorá je potrebná neskôr na výpočet uhlov orientácie.

```
#define SAMPLE_RATE 10 // v Hz
```

Vytvorte objekt filtra „Madgwick“.

```
Madgwick filter;
```

Vytvorte objekt „filter“, ktorý neskôr použijeme na vkladanie údajov z „IMU“ a získavanie výsledkov uhlov orientácie.

Definujte funkciu „printValues“. Jej účel je jednoduchý: vytlačiť surové údaje z „IMU“, aby sme ich mohli ľahko zobraziť v sériovom monitore.

```
void printValues() {
  char buffer[8];       // reťazcový buffer na použitie s funkciou
  dtostrf() float ax, ay, az; // hodnoty akcelerometra
  float gx, gy, gz; // hodnoty gyroskopu
```

Najskôr deklarujeme premenné. Premenná „buffer“ je tu definovaná ako reťazec typu „char“ a slúži hlavne na ukladanie textu. Údaje zo šesťosového senzora sú deklarované ako typ „float“, keďže tieto hodnoty obsahujú desatinné miesta.

```
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
      && IMU.readAcceleration(ax, ay, az) && IMU.readGyroscope(gx, gy, gz)) { Serial.print("ax
      = "); Serial.print(dtostrf(ax, 4, 1, buffer));   Serial.print(" g, "); Serial.print("ay = ");
      Serial.print(dtostrf(ay, 4, 1, buffer)); Serial.print(" g, "); Serial.print("az = ");
      Serial.print(dtostrf(az, 4, 1, buffer));   Serial.print(" g, "); Serial.print("gx = ");
      Serial.print(dtostrf(gx, 7, 1, buffer)); Serial.print(" °/s, "); Serial.print("gy = ");
      Serial.print(dtostrf(gy, 7, 1, buffer));   Serial.print(" °/s, "); Serial.print("gz = ");
      Serial.print(dtostrf(gz, 7, 1, buffer)); Serial.println(" °/s");
  }
}
```

Najprv pomocou podmienky if skontrolujeme, či sú údaje pripravené. Senzor vyžaduje inicializačné obdobie, takže pokiaľ nie je pripravený, nevykonávame žiadny výpis údajov. Akonáhle sú údaje pripravené, pomocou „Serial.print“ vyvedieme hodnoty do sériového monitora.

Serial.print(""): priamo vytlačí reťazec vnútri úvodzoviek.
Serial.print(dtostrf(ax, 4, 1, buffer)): Používa funkciu „dtostrf()“ na nastavenie šírky a desatinného rozlíšenia údajov. Číslo 4 znamená, že údaj zaberá celkovo najmenej štyri znaky, a číslo 1 znamená, že za desatinnou čiarkou zostane jedna desatinná číslica. „buffer“ je miesto, kde sa ukladá naformátovaný výsledok, ktoré bolo definované už skôr v kóde.

Ďalej definujeme druhú funkciu „printRotationAngles“. Účelom tejto funkcie je použiť algoritmus „Madgwick“ na výpočet a výstup orientačných uhlov na základe údajov „IMU“.

```
void printRotationAngles() {
  char buffer[5];       // reťazcový buffer na použitie s funkciou
  dtostrf() float ax, ay, az; // hodnoty akcelerometra
  float gx, gy, gz; // hodnoty gyroskopu
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()
      && IMU.readAcceleration(ax, ay, az) && IMU.readGyroscope(gx, gy, gz)) { filter.updateIMU(gx, gy, gz,
      ax, ay, az); // aktualizácia hodnôt náklonu, sklonu a odklonu
      // Vytlač uhly otáčania
```

```
        Serial.print("Naklonenie = "); Serial.print(dtostrf(filter.getRoll(), 4, 0, buffer)); Serial.print(" °, ");
        Serial.print("Sklon = "); Serial.print(dtostrf(filter.getPitch(), 4, 0, buffer)); Serial.print(" °, ");
        Serial.print("Odchýlka = "); Serial.print(dtostrf(filter.getYaw(), 4, 0, buffer)); Serial.println(" °");
    }
}
```

Metóda výstupu je podobná tej, ktorá sa používa vo funkcii „printValues“, takže sa tu už nebudeme podrobne zaoberať.

filter.getRoll(): získava náklon vľavo – vpravo (roll)

filter.getPitch(): získa sklon dopredu – dozadu (pitch)
filter.getYaw(): získa horizontálnu rotáciu (yaw) Inicializačná

funkcia

```
void setup() {
    Serial.begin(115200); // inicializuje sériovú zbernicu (sériový monitor)
    IMU.begin();
    filter.begin(SAMPLE_RATE); // inicializuje Madgwickov filter
}
```

Nastavte prenosovú rýchlosť sériového rozhrania na 115200 a pri otvorení sériového monitora nezabudnite zvoliť rovnakú prenosovú rýchlosť.

Inicializujte „IMU“ na spustenie senzora LSM6DS3.

Inicializujte Madgwickov filter a zadajte predtým definovanú vzorkovaciu frekvenciu; inak bude výpočet orientácie nepresný.

Funkcia slučky

```
void loop() {
    static unsigned long previousTime = millis(); unsigned
    long currentTime = millis();
    if (currentTime - previousTime >= 1000/SAMPLE_RATE) {
        printValues();
        printRotationAngles();
        previousTime = millis();
    }
}
```

Táto časť kódu využíva prístup „softvérového časovača“ na spúšťanie úloh v pevných intervaloch bez blokovania programu.

unsigned long: typ bez znamienka s dlhým celým číslom. Hodnota vrátená funkciou „millis()“ je veľké číslo a môže byť iba kladná, preto je tento typ povinný.

static: zaručuje, že premenná je inicializovaná len raz a zachováva si svoju hodnotu aj pri viacerých volaniach funkcie. Za normálnych okolností sa funkcia „loop()“ vykonáva nepretržite a premenné by sa pri každom spustení cyklu priradili nanovo. Ak by k tomu došlo, premenná „previousTime“ by bola vždy nastavená na aktuálny čas a podmienka na meranie času by sa nikdy nesplnila. Vďaka použitiu „static“ je premenná inicializovaná len pri prvom spustení funkcie „loop()“ a svoju hodnotu si zachová aj naďalej.

currentTime: táto premenná sa aktualizuje pri každom spustení slučky, čo nám umožňuje zistiť aktuálny

systémový čas.

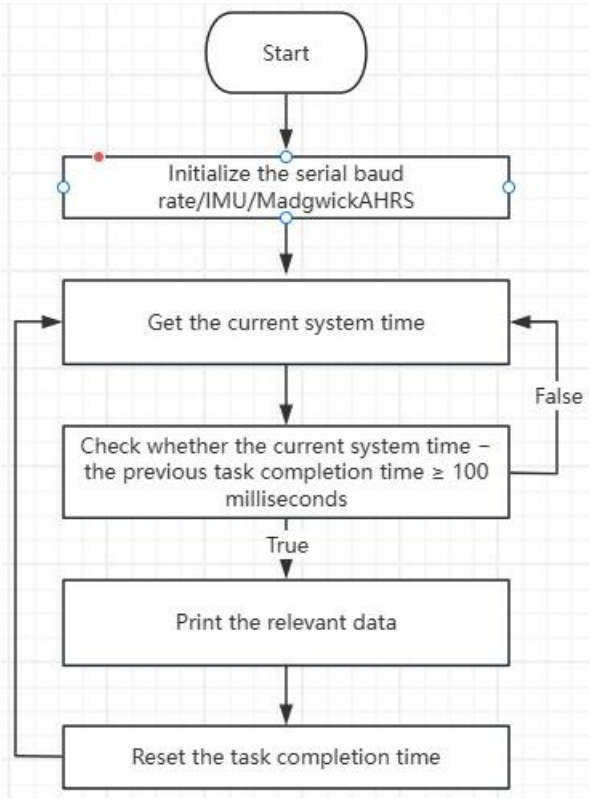
currentTime - previousTime: rozdiel medzi týmito dvoma hodnotami predstavuje, koľko času uplynulo.

1000/SAMPLE_RATE: 1000 milisekúnd vydelených vzorkovacou frekvenciou (koľko vzoriek sa odoberie za sekundu) dáva časový interval pre každú vzorku.

Ak je rozdiel medzi aktuálnym časom a časom posledného merania väčší alebo rovný definovanému intervalu merania, vykonajú sa dve nižšie uvedené funkcie a príslušné údaje sa vypíšu na sériový monitor.

previousTime = millis(): po dokončení úlohy sa aktuálny systémový čas priradí k „previousTime“, aby sa pripravila ďalšia kontrola času.

Celkový diagram logiky kódu



Kroky na nahratie programu

Tento projekt vyžaduje dodatočné externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

char	Definujte dátový typ ako reťazec
millis()	Aktuálny systémový čas

static	Tento kód sa spustí len raz v rámci funkcie cyklu a slúži na získanie aktuálneho systémového času na meranie času a kontrolu podmienok

Lekcia 13 --- Svetelný senzor

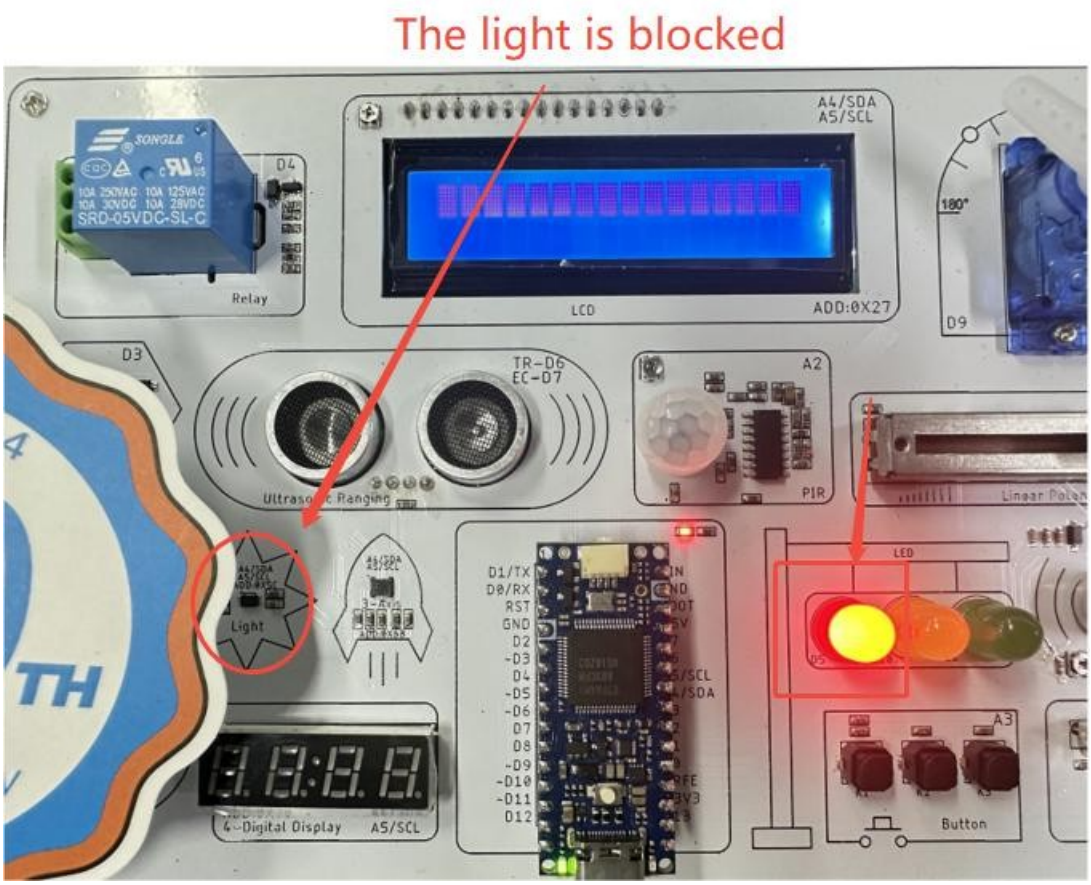
Úvod

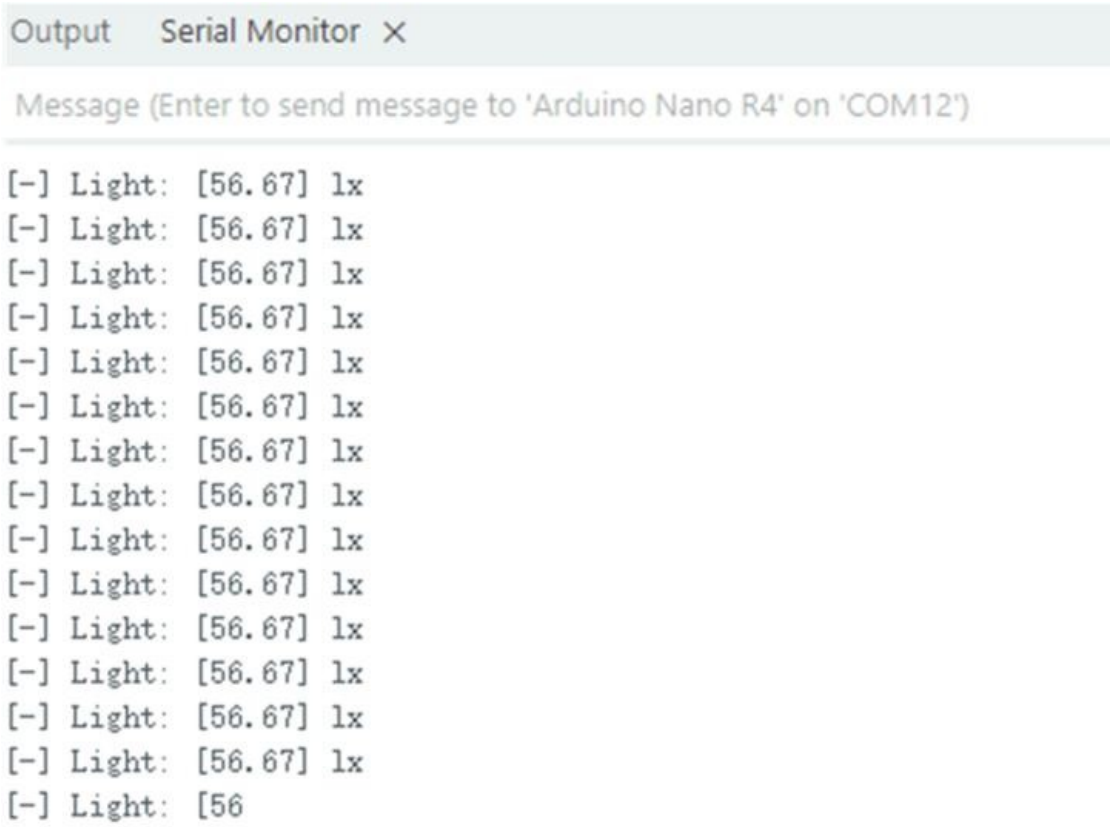
V tejto lekcii sa naučíme, ako používať „svetelný senzor“ na vývojovej doske „Arduino Nano R4“. Budeme naďalej používať rozhranie I2C na ovládanie senzora, čítanie údajov o intenzite okolitého osvetlenia zo „svetelného senzora“ a vytvorenie jednoduchého simulovaného inteligentného pouličného osvetlenia systém – automatické zapínanie osvetlenia v noci a vypínanie cez deň na základe podmienok okolitého osvetlenia.

Ciele výučby

1. Porozumieť fungovaniu „svetelného senzora“
2. Zopakovať si, ako používať príkazy if – else
3. Vykonajte simulovaný experiment s pouličným osvetlením, ktorý zmeria intenzitu okolitého osvetlenia a určí, či by sa malo svetlo zapnúť

Náhľad výsledku





Po spustení programu, ak zakryjete svetlo dopadajúce na „svetelný senzor“ a otvoríte sériový monitor, uvidíte, že aktuálna hodnota intenzity okolitého osvetlenia je nižšia ako 100. Červená LED dióda na doske Arduino Nano R4 sa rozsvieti. Ak je hodnota intenzity okolitého osvetlenia vyššia ako 100, červená LED dióda zhasne.

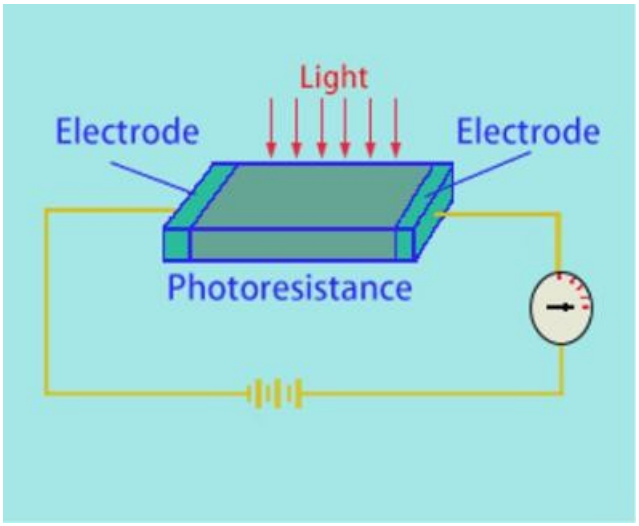
Hardvér použitý v tejto lekcii

„Svetelný senzor“ sa nachádza v ľavom dolnom rohu vývojovej dosky. Je pripojený k doske cez I2C s adresou 0x5C.

Princíp fungovania modulu svetelného senzora

Ako je znázornené na obrázku nižšie, hlavnou súčasťou svetelného senzora je fotorezistor v strede. Keď svetlo dopadá na povrch fotorezistora, jeho odpor sa mení v závislosti od intenzity svetla.

intenzita – čím silnejšie svetlo, tým nižší odpor. Keď je fotorezistor zapojený do obvodu s napájacím zdrojom a ampérmetrom, prúd sa mení úmerne so zmenou odporu (nižší odpor vedie k vyššiemu prúdu, zatiaľ čo vyšší odpor vedie k nižšiemu prúdu). Týmto spôsobom sa pôvodný svetelný signál v obvode premieňa na elektrický signál. Zmena hodnoty na ampérmetri odráža zmenu intenzity svetla. Toto je základný princíp fungovania svetelného senzora, ktorý „využíva svetlo na riadenie elektriny“.

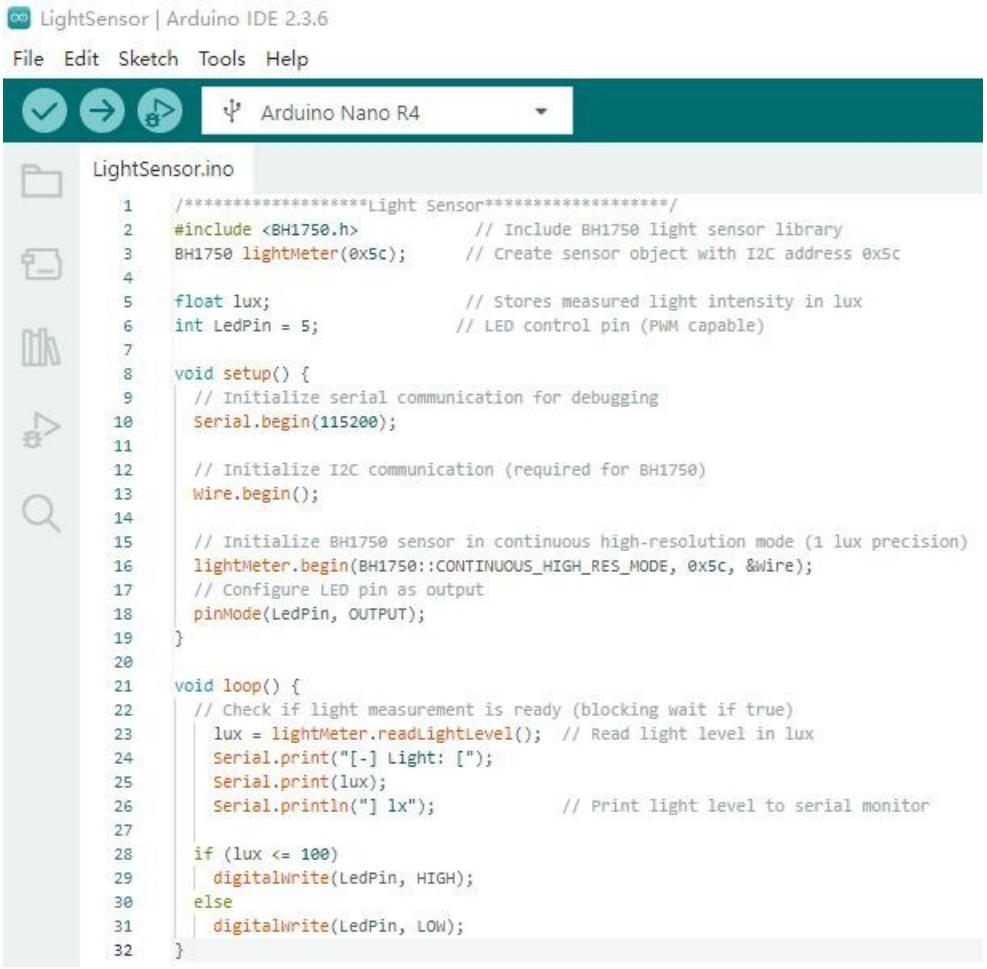


Inteligentné pouličné osvetlenie

Než prejdeme k kódu, môžete si ho najskôr stiahnuť. Odkaz na stiahnutie kompletného kódu:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/13_Light_Sensor
Otvorte súbor „13_Light_Sensor.ino“ nachádzajúci sa v priečinku „13_Light_Sensor“ pomocou prostredia Arduino IDE.

Vysvetlenie kľúčových častí kódu

Teraz si prejdime príklad: Inteligentné pouličné osvetlenie.



Najskôr importujte knižnicu senzora „BH1750“, ktorá sa používa na ovládanie „svetelného senzora“.

```
#include <BH1750.h>
```

Vytvorte objekt senzora BH1750.

```
BH1750 lightMeter(0x5c);
```

Určite adresu I2C „svetelného senzora“ ako „0x5C“ a vytvorte objekt senzora „BH1750“ s názvom „lightMeter“.

Definujte premennú intenzity osvetlenia s názvom „lux“ na ukladanie úrovne okolitého osvetlenia.

```
float lux;
```

Túto premennú definujeme pomocou typu float, aby sa zachovali desatinné hodnoty a nedošlo k strate presnosti.

Definujte ovládací pin pre LED.

```
int LedPin = 5;
```

Inicializácia funkcie.

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

```
  lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire); pinMode(LedPin,
  OUTPUT);
}
```

Nastavte sériovú prenosovú rýchlosť na 115200. Musíme tiež otvoriť sériový monitor, aby sme mohli zobraziť vrátené hodnoty intenzity osvetlenia.

Wire.begin(): inicializuje komunikáciu I2C, ktorú sme podrobne rozoberali v predchádzajúcich lekciách.

lightMeter.begin: inicializuje senzor „BH1750“. **BH1750::CONTINUOUS_HIGH_RES_MODE**: nastaví senzor do režimu nepretržitého merania s vysokým rozlíšením. V tomto režime senzor pracuje nepretržite a nie je potrebné ho opakovane prebúdzať.

pinMode(): nastavuje režim pinu – v tomto prípade nastaví pin LED ako výstup.

Funkcia loop

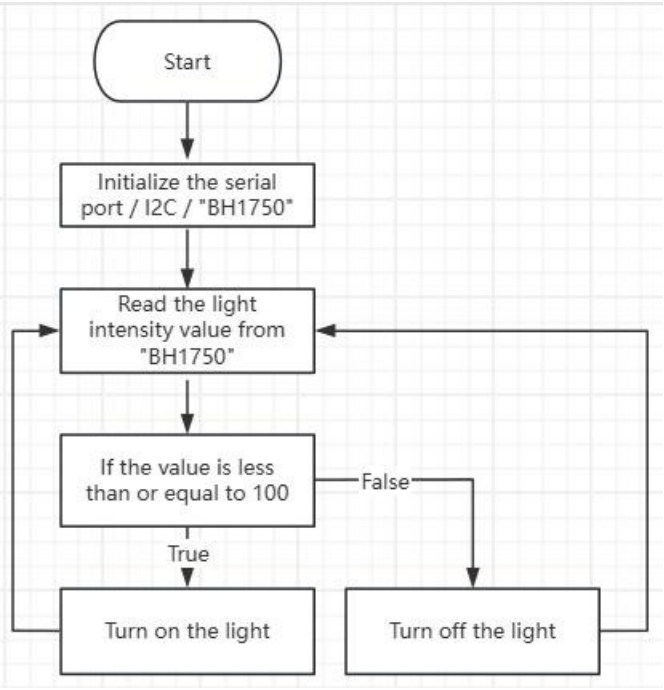
```
void loop() {
  // Skontroluje, či je meranie osvetlenia pripravené (blokujúce
    čakanie, ak je hodnota true) lux = lightMeter.readLightLevel(); //
    Prečíta úroveň osvetlenia v luxoch Serial.print("[-] Svetlo: [");
    Serial.print(lux);
    Serial.println("] lx");                    // Vypíše úroveň osvetlenia na sériový monitor

  if (lux <= 100)
    digitalWrite(LedPin, HIGH);
  inak
    digitalWrite(LedPin, LOW);
}
```

lightMeter.readLightLevel():číta aktuálnu úroveň osvetlenia zo senzora BH1750 a priradí vrátenú hodnotu premennej „lux“, čím sa uľahčuje jej neskoršie vytlačenie a vyhodnotenie.

Serial.print(): Serial.print() sa používa na výstup údajov do sériového monitora; nebudeme sa tu o tom podrobne rozpisovať
, pretože jeho hlavným účelom je jednoducho zobraziť hodnoty.

if - else: Ak je hodnota „lux“ menšia alebo rovná 100, LED sa nastaví na HIGH a rozsvieti sa; v opačnom prípade sa nastaví na LOW a zhasne.

Celkový diagram logického toku kódu



Kroky na nahratie programu

Tento projekt vyžaduje dodatočné externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

readLightLevel()	Toto je metóda poskytovaná knižnicou BH1750 a jej primárnou Cieľom je zistiť hodnotu intenzity okolitého osvetlenia.

Lekcia 14 --- Teplota a vlhkosť

Úvod

V tejto lekcii sa naučíme, ako používať senzor teploty a vlhkosti na vývojovej doske „Arduino Nano R4“. Naučíte sa, ako používať knižnicu DHT20 a jej vstavané metódy na čítanie údajov o teplote a vlhkosti a zobrazenie hodnôt v sériovom monitore.

Ciele

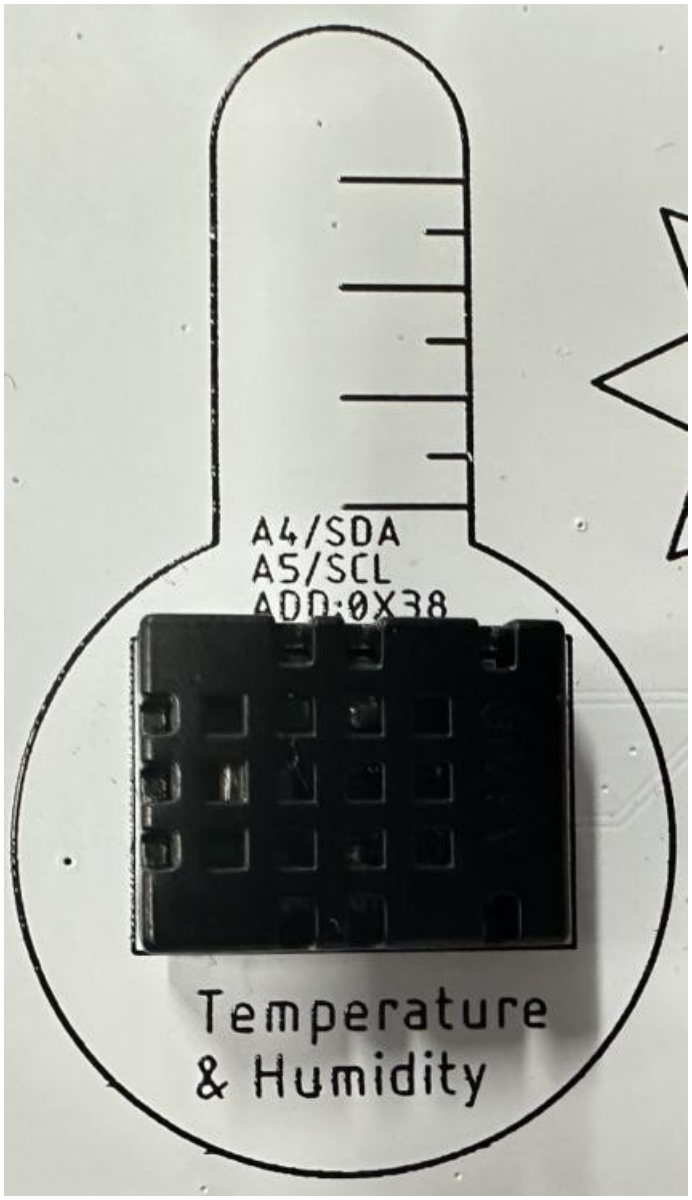
1. Porozumieť fungovaniu senzora teploty a vlhkosti
2. Vykonajte experiment, ktorý načítava údaje o teplote a vlhkosti a vypíše hodnoty do sériového monitora

Náhľad výsledku

```
Temperature: 27.8 °C   Humidity: 55.3 %
Temperature: 27.8 °C   Humidity: 55.2 %
Temperature: 27.8 °C   Humidity: 55.2 %
Temperature: 27.8 °C   Humidity: 55.2 %
Temperature: 27.8 °C   Humidity: 55.2 %
Temperature: 27.7 °C   Humidity: 55.1 %
Temperature: 27.7 °C   Humidity: 55.1 %
Temperature: 27.8 °C   Humidity: 55.1 %
Temperature: 27.7 °C   Humidity: 55.0 %
Temperature: 27.7 °C   Humidity: 55.0 %
Temperature: 27.7 °C   Humidity: 55.1 %
Temperature: 27.7 °C   Humidity: 55.1 %
Temperature: 27.7 °C   Humidity: 55.0 %
Temperature: 27.7 °C   Humidity: 55.0 %
Temperature: 27.8 °C   Humidity: 55.0 %
```

Po spustení programu a otvorení sériového monitora uvidíte aktuálne hodnoty teploty a vlhkosti, ktoré snímač odosiela.

Hardvér použitý v tejto lekcii

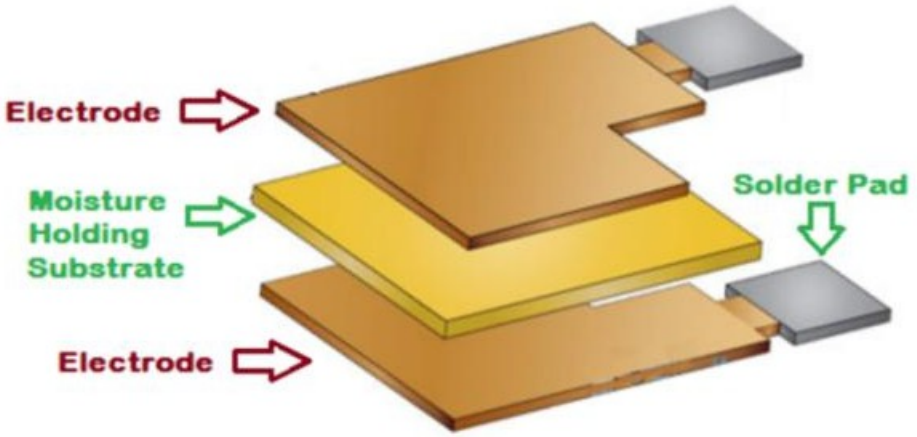


Senzor teploty a vlhkosti sa nachádza v ľavom dolnom rohu vývojovej dosky. Je pripojený k doske cez rozhranie I2C s adresou 0x38.

Princíp fungovania modulu snímača teploty a vlhkosti

Na nižšie uvedenom schéme je znázornená štruktúra snímača teploty a vlhkosti.
Jeho jadro tvorí horná a spodná elektróda s hygroskopickým dielektrickým substrátom medzi nimi, pričom kontaktné plôšky zabezpečujú elektrické spojenie medzi elektródami a vonkajším obvodom.
Počas merania vlhkosti sa vodná para vo vzduchu absorbuje alebo uvoľňuje dielektrickým substrátom. Tento proces mení dielektrickú konštantu substrátu, čo následne mení kapacitu vytvorenú elektródami a substrátom, čím sa umožňuje premena vlhkosti na elektrický signál. Zároveň integrovaný teplotne citlivý prvok meria teplotu okolia. Nezávisle odosiela údaje o teplote a tiež uplatňuje teplotnú

kompenzáciu na korekciu chýb merania vlhkosti. Nakoniec sa prostredníctvom obvodu na úpravu signálu generujú presné elektrické signály teploty a vlhkosti.



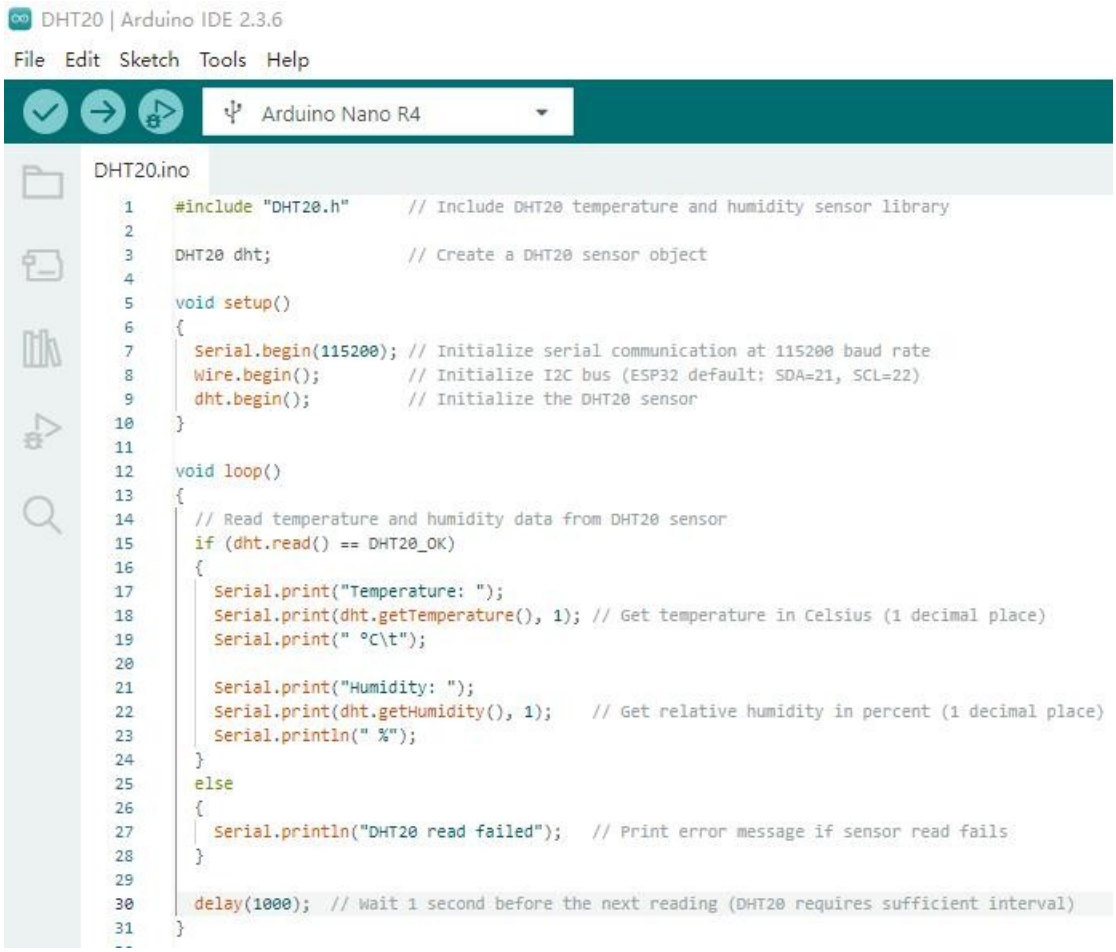
Získavanie údajov zo senzora teploty a vlhkosti

Pred prechádzaním kódu si ho môžete najskôr stiahnuť. Odkaz na stiahnutie kompletného kódu: https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/14_Tem%26Hum

Pomocou prostredia Arduino IDE otvorte súbor „14_Tem&Hum.ino“, ktorý sa nachádza v priečinku „14_Tem&Hum“.

Vysvetlenie kľúčových kódov

Teraz si prejdime príklad: Získavanie údajov zo senzora teploty a vlhkosti.



Najskôr importujte knižnicu „DHT20“, ktorá sa používa na ovládanie snímača teploty a vlhkosti.

```
#include "DHT20.h"
```

Ďalej vytvorte objekt triedy DHT20 s názvom „dht“.

```
DHT20 dht;
```

Táto inštancia senzora „dht“ sa používa na ovládanie a čítanie údajov zo senzora „DHT20“. Inicializačná

funkcia

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

V tejto funkcii vykonávame tri úkony: inicializujeme prenosovú rýchlosť sériového rozhrania, inicializujeme zbernicu I2C a inicializujeme snímač teploty a vlhkosti.

Funkcia Loop

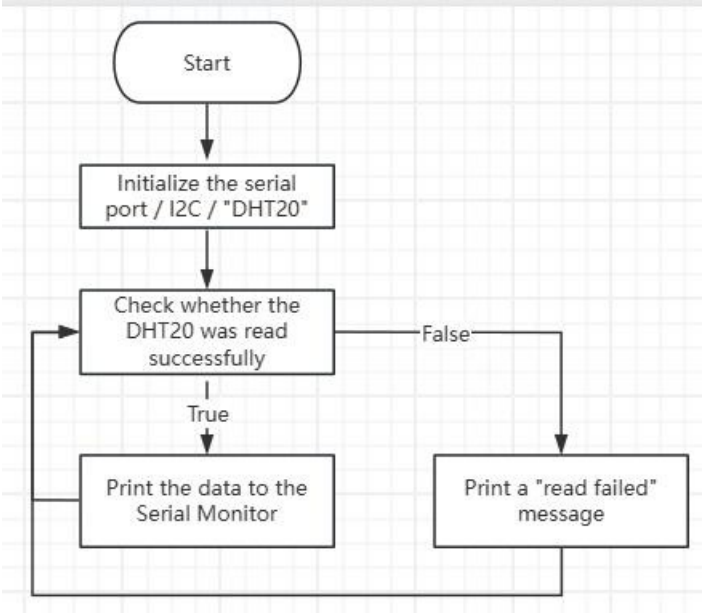
```
void loop()
{
  if (dht.read() == DHT20_OK)
```

V rámci funkcie loop najprv pomocou príkazu if skontrolujeme, či sa údaje prečítali úspešne. Tento krok je veľmi dôležitý, pretože aktualizuje údaje zo senzora a zároveň overuje, či bola operácia čítania úspešná.

```
  {
    Serial.print("Teplota: ");
    Serial.print(dht.getTemperature(), 1); // Získať teplotu v stupňoch Celzia (1 desatinné miesto)
    Serial.print(" °C\t");

    Serial.print("Vlhkosť: ");
    Serial.print(dht.getHumidity(), 1);          // Získať relatívnu vlhkosť v percentách (1 desatinné
    miesto) Serial.println(" %");
  }
  else
  {
    Serial.println("Čítanie DHT20 zlyhalo");      // Vypíše chybovú správu, ak zlyhá čítanie senzora
  }
  delay(1000); // Počkaj 1 sekundu pred ďalším čítaním (DHT20 vyžaduje dostatočný interval)
}
```

Vypíše údaje o teplote a vlhkosti načítané zo senzora na sériový monitor. Ak sa čítanie nezdaří, vypíše „DHT20 read failed“.

Celkový diagram logického toku kódu



Kroky na nahratie programu

Tento projekt vyžaduje dodatočné externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Postup nahratia“ na strane 8.

Kľúčové body:

dht.getTemperature()	Načítanie údajov o teplote zo senzora DHT20.
dht.getHumidity()	Načítanie údajov o vlhkosti zo senzora DHT20.

Lekcia 15 --- IR modul

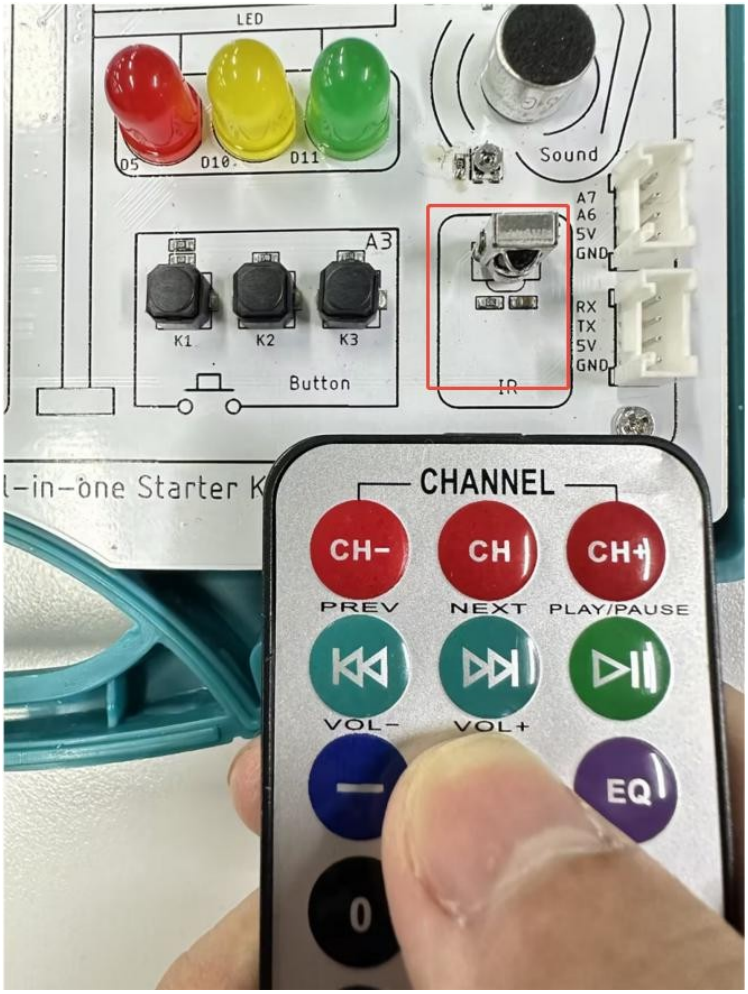
Úvod

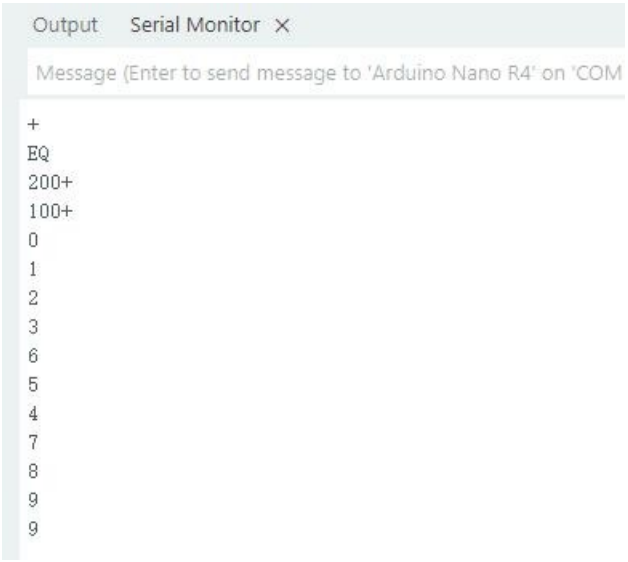
V tejto lekcii sa naučíme, ako používať „IR modul“ na vývojovej doske Arduino Nano R4. Počas celého kurzu budeme používať príkaz „switch-case“ na spracovanie rôznych signálov tlačidiel z diaľkového ovládača a dokončíme praktický projekt, ktorý vytlačí hodnoty tlačidiel na sériový monitor. Vďaka tomuto príkladu získate jasnú predstavu o tom, ako používať štruktúru „switch-case“ na efektívne a logické spracovanie viacerých podmienok pri práci s mnohými možnými vstupnými možnosťami.

Ciele vzdelávania

1. Porozumieť fungovaniu „IR modulu“
2. Naučte sa, kedy a ako používať príkaz „switch-case“
3. Dokončite projekt, ktorý vypíše hodnoty tlačidiel na sériový monitor

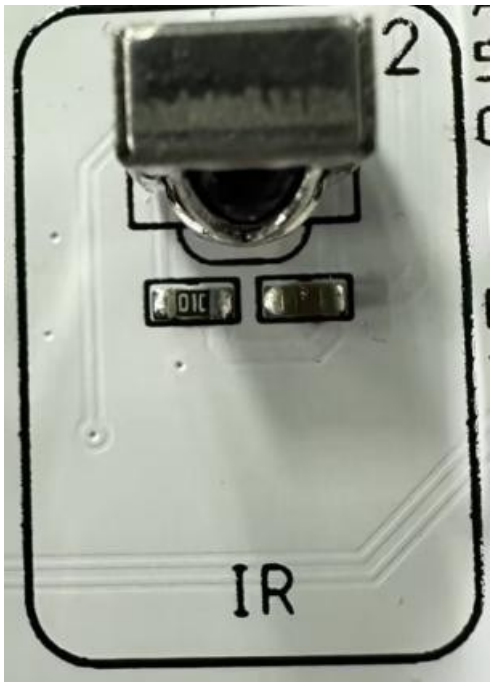
Náhľad výsledku





Po spustení programu, keď stlačíte tlačidlo na diaľkovom ovládači, sériový monitor zobrazí príslušné informácie o tlačidle. Poznámka: Uistite sa, že diaľkový ovládač smeruje k „IR modulu“, a skontrolujte, či sú batérie v diaľkovom ovládači správne vložené.

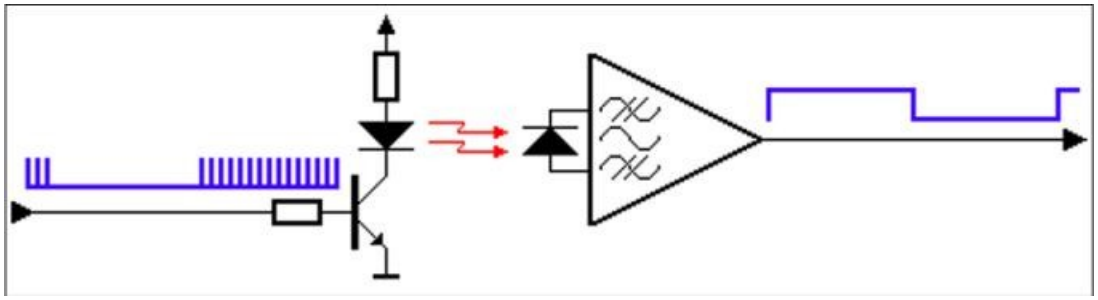
Hardvér použitý v tejto lekcii



„IR modul“ sa nachádza v pravom dolnom rohu vývojovej dosky a je pripojený k pinu D2.

Princíp fungovania infračerveného modulu

Ako je znázornené na nižšie uvedenom schéme, princíp fungovania infračerveného diaľkového ovládača je nasledovný: kódovaný elektrický signál vysielaný diaľkovým ovládačom riadi tranzistor a infračervenú LED diódu v obvode vysielača, čím sa elektrický signál prevádza na infračervené svetelné impulzy. Tieto infračervené signály sú následne zachytené infračerveným prijímačom, spätne prevedené na elektrické signály a spracované zosilňovacím obvodom. Nakoniec sa rekonštruuje pôvodný riadiaci signál, čo umožňuje diaľkové ovládanie zariadenia.



Tlač informácií o tlačidlách diaľkového ovládača

Než sa pustíme do vysvetľovania kódu, najprv si stiahnime program. Kompletný kód nájdete na nižšie uvedenom odkaze:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/15_IR_Module
Otvorte súbor „15_IR_Module.ino“ nachádzajúci sa v priečinku „15_IR_Module“ pomocou Arduino IDE.

Vysvetlenie kódov tlačidiel

Teraz si prejdime projekt: Tlač informácií o tlačidlách diaľkového ovládača


```ino
#include <DIYables_IRcontroller.h>              // Include the DIYables IR controller library

#define IR_RECEIVER_PIN 2                       // Define the Arduino pin connected to the IR receiver

DIYables_IRcontroller_21 irController(         // Create an IR controller object for a 21-key remote
  IR_RECEIVER_PIN,                              // Specify the pin used by the IR receiver
  200                                           // Set debounce time to 200 milliseconds
);

void setup() {                                  // Arduino setup function, runs once
  Serial.begin(115200);                         // Initialize serial communication at 115200 baud rate
  irController.begin();                         // Initialize the IR controller
}

void loop() {                                   // Arduino loop function, runs repeatedly
  Key21 command = irController.getKey();        // Read the pressed key from the IR remote

  if (command != Key21::NONE) {                 // Check if a valid key was received
    switch (command) {                          // Determine which key was pressed

      case Key21::KEY_CH_MINUS:                 // If the CH- key is pressed
        Serial.println("CH-");                  // Print "CH-" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_CH:                       // If the CH key is pressed
        Serial.println("CH");                   // Print "CH" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_CH_PLUS:                  // If the CH+ key is pressed
        Serial.println("CH+");                  // Print "CH+" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_PREV:                     // If the previous (<<) key is pressed
        Serial.println("<<");                   // Print "<<" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_NEXT:                     // If the next (>>) key is pressed
        Serial.println(">>");                   // Print ">>" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_PLAY_PAUSE:               // If the play/pause key is pressed
        Serial.println(">||");                  // Print ">||" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_VOL_MINUS:                // If the volume down key is pressed
        Serial.println("-");                    // Print "-" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_VOL_PLUS:                 // If the volume up key is pressed
        Serial.println("+");                    // Print "+" to the serial monitor
        break;                                  // Exit the switch case

      case Key21::KEY_EQ:                       // If the EQ key is pressed
        Serial.println("EQ");                   // Print "EQ" to the serial monitor
        break;                                  // Exit the switch case
```

Najskôr importujte knižnicu „DIYables_IRcontroller.h“, ktorá sa používa na ovládanie „IR modulu“.

```
#include <DIYables_IRcontroller.h>
```

Ďalej definujte pin infračerveného prijímača

```
#define IR_RECEIVER_PIN 2
```

Oznámte programu, že signálny pin infračerveného prijímača je pripojený k pinu D2 Arduina. Potom

vytvorte objekt infračerveného diaľkového ovládača

```
DIYables_IRcontroller_21 irController(
```

```
  IR_RECEIVER_PIN,
  200
);
```

Vytvorte objekt infračerveného diaľkového ovládača s pinom 2 a dobou odozvy 200 milisekúnd, aby sa dal ľahko používať v ďalšej časti programu

Inicializačná funkcia

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

Inicializujte prenosovú rýchlosť sériového portu, aby sme mohli zobraziť výstup tlačidla v sériovom monitore. Inicializujte infračervený ovládač, aby ste povolili príjem infračerveného signálu.

Funkcia hlavnej slučky

```
void loop() {
  Key21 command = irController.getKey();
```

Načíta vstup z tlačidla z infračerveného prijímača. V tomto prípade je „Key21“ dátový typ, „command“ je premenná a vrátená hodnota funkcie „irController.getKey()“ je tiež typu „Key21“. Medzi bežné návratové hodnoty patria „Key21::KEY_1“, „Key21::KEY_CH_PLUS“, „Key21::NONE“ a podobne.

Ďalej skontrolujte, či bolo skutočne stlačené platné tlačidlo.

```
  if (command != Key21::NONE) {
```

Ak sa nezistí žiadne stlačenie tlačidla, jednoducho preskočte zvyšok logiky; inak by sériový monitor nepretržite vypisoval bezvýznamné údaje.

Určite, ktoré konkrétne tlačidlo bolo stlačené

```
    switch (command) {
```

Vykonaj rôzny kód v závislosti od hodnoty tlačidla

```
      case Key21::KEY_CH_MINUS:                    // Ak je stlačené tlačidlo CH-
        Serial.println("CH-"); break;              // Vypíše „CH-“ na sériový monitor
```

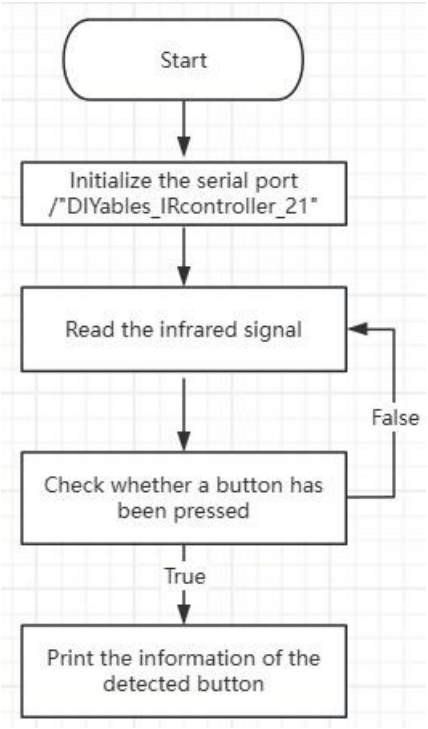
Napríklad, ak je vrátená hodnota „Key21::KEY_CH_MINUS“, sériový monitor vypíše „CH-“ Ak sa

vyskytne nedefinované tlačidlo

```
      predvolené:
        Serial.println("WARNING: undefined command:");
        break;
```

výstup „WARNING: undefined command:“

Celkový diagram logického toku kódu



Kroky na nahratie programu

Tento projekt vyžaduje ďalšie externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Kľúčové body:

switch-case	Používa sa na výber a vykonanie konkrétneho bloku kódu na základe hodnoty premennej.

Lekcia 16 --- Detektor šumu

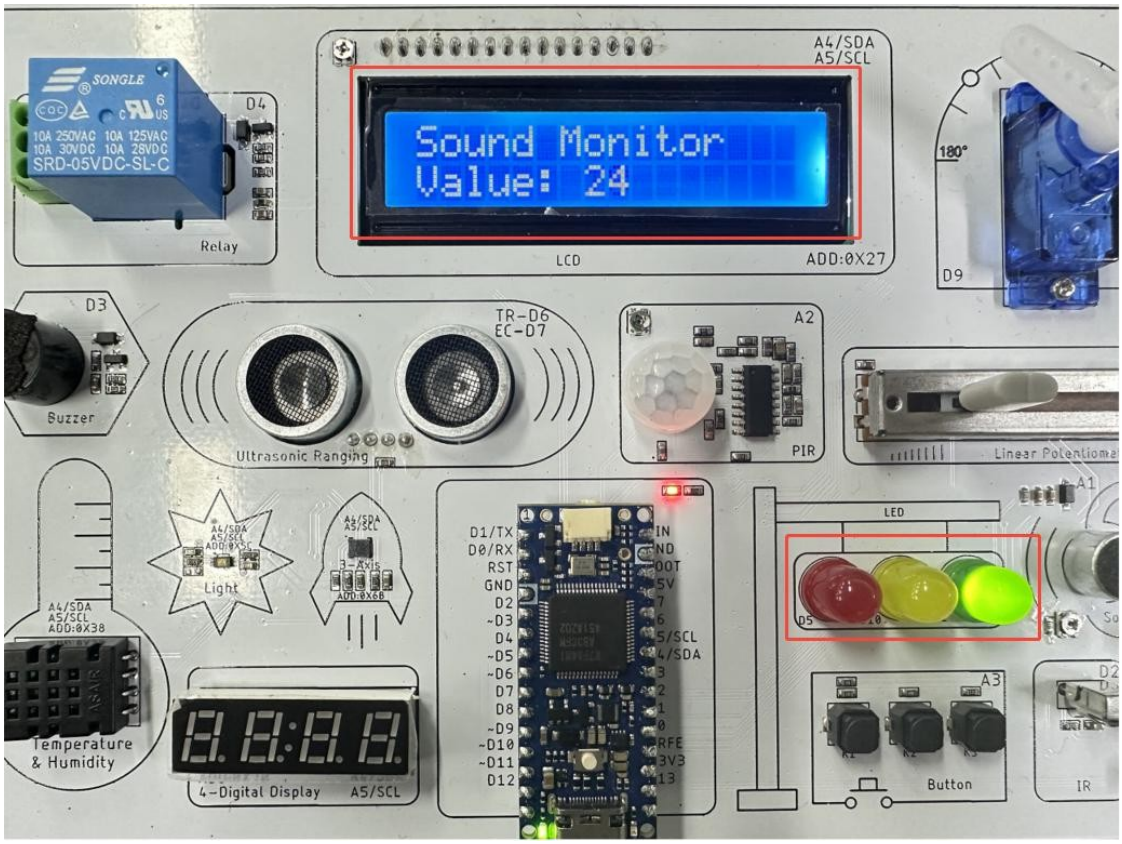
Úvod

V tejto lekcii integrujeme tri moduly – zvukový senzor, LED diódy a LCD displej – s cieľom vytvoriť praktický projekt detektora okolitého hluku. V rámci tejto lekcie si zopakujete a
upevniť si už osvojené pojmy, pochopiť, ako rôzne moduly spolupracujú, a naučiť sa, ako implementovať koordinovanú prevádzku viacerých senzorov a výstupných zariadení, čím si ďalej zdokonalíte svoje celkové aplikačné zručnosti.

Ciele vzdelávania

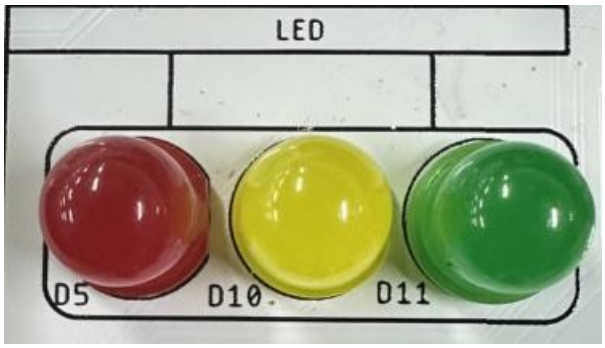
1. Prejdite si základné operácie používania Arduina na ovládanie hardvéru
2. Prezrite si, ako Arduino číta hodnoty vrátené hardvérovými senzormi
3. Vykonajte experiment s detektorom hluku, ktorý meria hladiny okolitého zvuku a rozsvieti rôzne LED diódy na základe zistenej intenzity zvuku

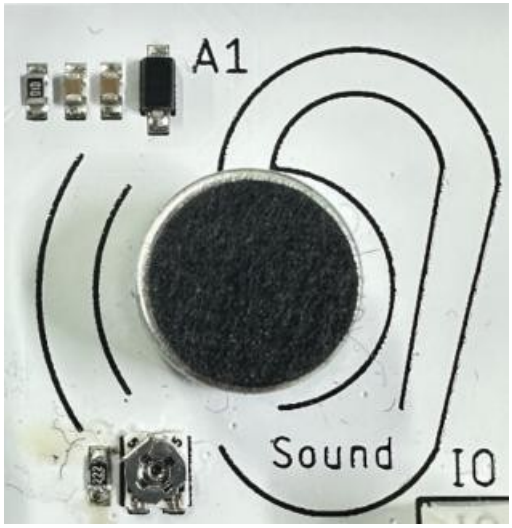
Náhľad výsledku



Po spustení programu Arduino Nano R4 monitoruje úroveň okolitého hluku. Ak je prostredie tiché, rozsvieti sa zelená LED dióda. Ak je úroveň hluku stredne vysoká, rozsvieti sa žltá LED dióda. Ak je prostredie veľmi hlučné, rozsvieti sa červená LED dióda

Hardvér použitý v tejto lekcii





Detektor šumu

Predtým, ako sa pustíte do prechádzania kódu, si ho môžete najprv stiahnuť. Odkaz na stiahnutie kompletného kódu: https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/16_Noise_Detector

Pomocou prostredia Arduino IDE otvorte súbor „16_Noise_Detector.ino“ nachádzajúci sa v priečinku „16_Noise_Detector“.

Vysvetlenie kľúčových častí kódu

Teraz si prejdime príklad: Detektor hluku.


```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// ===== LCD parameters =====
#define COLUMNS 16
#define ROWS    2

LiquidCrystal_I2C lcd(
  PCF8574_ADDR_A21_A11_A01,
  4, 5, 6, 16, 11, 12, 13, 14,
  POSITIVE
);
// ===== Pin definitions =====
const int Sound_Pin = A1;
const int RED_LED    = 5;
const int YELLOW_LED = 10;
const int GREEN_LED  = 11;

void setup() {
  // Initialize LED pins as output
  pinMode(RED_LED, OUTPUT);
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);

  // Initialize LCD
  lcd.begin(COLUMNS, ROWS, LCD_5x8DOTS);
  lcd.clear();
  lcd.backlight();

  lcd.setCursor(0, 0);
  lcd.print("Sound Monitor");
}

void loop() {
  int soundValue = analogRead(Sound_Pin);
  // ===== LCD display =====
  lcd.setCursor(0, 1);
  lcd.print("Value: ");
  lcd.print(soundValue);
  lcd.print("   ");   // Clear remaining charac
  // ===== LED status control =====
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
  if (soundValue < 100) {
    digitalWrite(GREEN_LED, HIGH);
  }
  else if (soundValue <= 300) {
    digitalWrite(YELLOW_LED, HIGH);
  }
  else {
    digitalWrite(RED_LED, HIGH);
  }

  delay(200);  // Refresh rate
}
```

Najskôr načítajte hlavičkové súbory potrebné pre tento projekt.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

V tomto príklade vyžaduje externú knižnicu na ovládanie len modul LCD, preto zahrnieme hlavičkový súbor „LiquidCrystal_I2C.h“ spolu so súborom „Wire.h“ na podporu komunikácie I2C.

Ďalej definujeme veľkosť LCD.

```
#define COLUMNS 16
#define ROWS        2
```

LCD má 16 znakov na riadok a celkom 2 riadky. Vytvorte objekt LCD

```
LiquidCrystal_I2C lcd(
  PCF8574_ADDR_A21_A11_A01, 4, 5,
  6, 16, 11, 12, 13, 14, POSITIVE
);
```

Vytvorte objekt LCD pomocou adresy I2C zariadenia.

Definujte piny

```
const int Sound_Pin = A1;
const int RED_LED         = 5;
const int YELLOW_LED = 10; const
int GREEN_LED = 11;
```

Tu definujeme vývody pre tri LED diódy a jeden zvukový senzor

Funkcia inicializácie

```
void setup() { pinMode(RED_LED,
  OUTPUT);
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT); lcd.begin(COLUMNS,
  ROWS, LCD_5x8DOTS);
  lcd.clear();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Sound Monitor");
}
```

Nastavte tri piny LED do výstupného režimu, inicializujte modul LCD, vymažte obrazovku LCD a zapnite podsvietenie. Potom, začínajúc v pozícii (0, 0) na prvom riadku, zobrazte „Sound Monitor“.

V hlavnej slučke

```
void loop() {
  int soundValue = analogRead(Sound_Pin);
```

Neustále čítaj analógovú hodnotu zo zvukového senzora a priraď ju celočíselnej premennej s názvom „soundValue“

Zobrazte hodnotu zvuku na LCD displeji

```
  lcd.setCursor(0, 1);
  lcd.print("Hodnota: ");
  lcd.print(soundValue);
```

```
  lcd.print("       ");     // Vymazať zostávajúce znaky
```

Vytlačte analógovú hodnotu vrátenú zvukovým senzorom na druhý riadok LCD Vypnite všetky

LED diódy

```
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
```

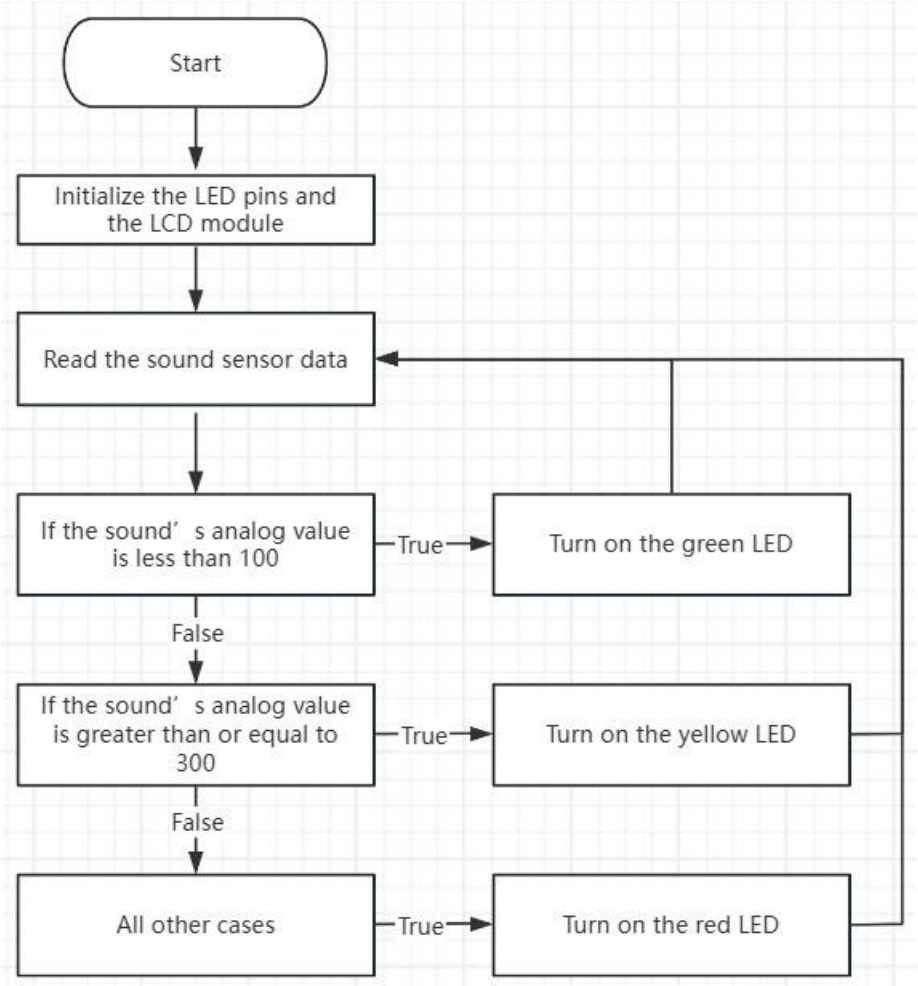
Tým sa zabezpečí, že počas každého cyklu svieti len jedna LED. Na

základe úrovne zvuku urč, ktorá LED má svietiť.

```
  if (soundValue < 100) {
    digitalWrite(GREEN_LED, HIGH);
  }
  else if (soundValue <= 300) {
    digitalWrite(YELLOW_LED, HIGH);
  }
  inak {
    digitalWrite(RED_LED, HIGH);
  }

  delay(200); // Obnovovacia frekvencia
}
```

Ak je hodnota zvuku menšia ako 100, zapnite zelenú LED. Ak je hodnota zvuku menšia alebo rovná 300, zapnite žltú LED. Všetky ostatné hodnoty zapnú červenú LED. Na riadenie obnovovacej frekvencie sa používa oneskorenie 200 ms.

Celkový diagram logiky kódu



Kroky na nahratie programu

Tento projekt vyžaduje ďalšie externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Lekcia 17 – Ultrazvukový senzor

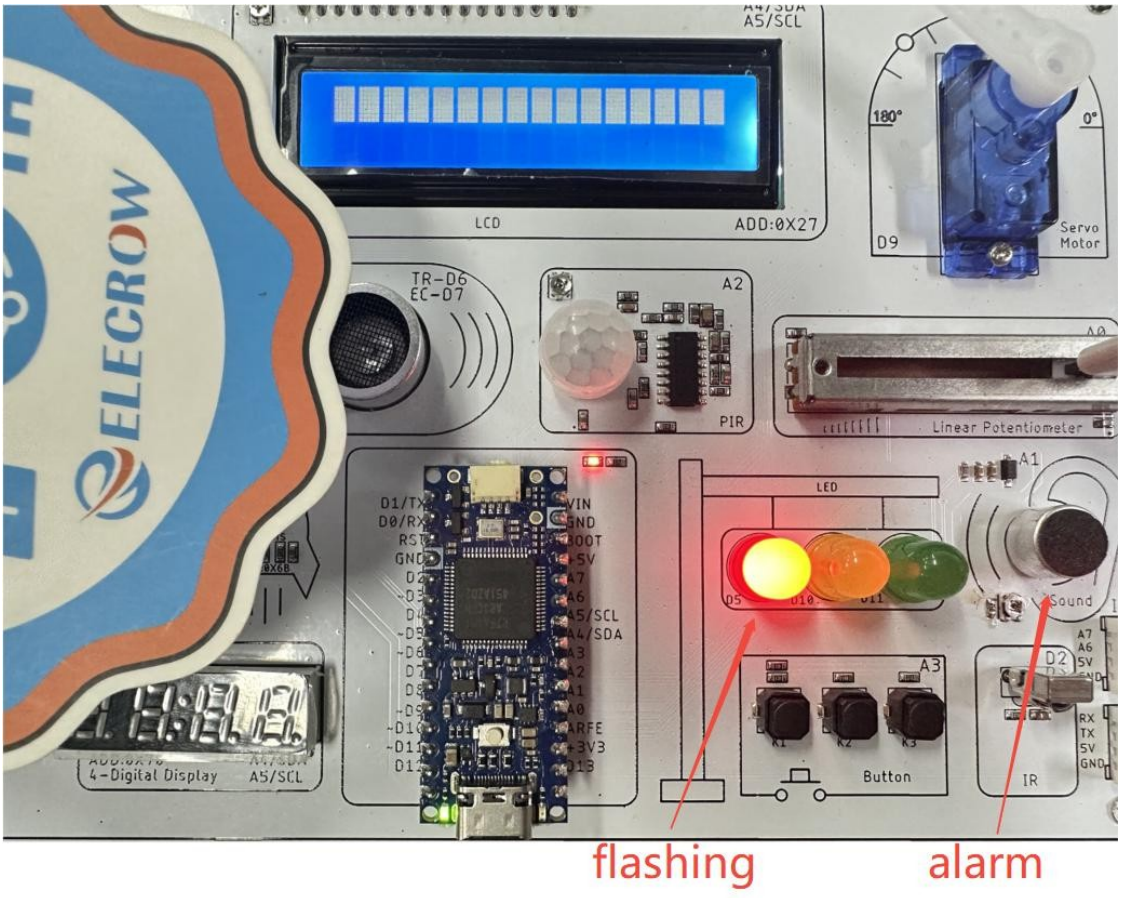
Úvod

V tejto lekcii skombinujeme ultrazvukový senzor, bzučiak a LED diódy, aby sme vytvorili systém varovania pri cúvaní. Počas implementácie si zopakujeme, ako sa volajú funkcie, a zameriame sa na naučenie sa, ako používať parametre funkcií. Odovzdávaním rôznych parametrov do funkcií môžeme flexibilne ovládať správanie bzučiaka a LED diód. V tejto lekcii sa naučíte, ako používať údaje zo senzorov ako parametre funkcií, a získate hlbšie pochopenie dôležitosti funkcií v štruktúre programu a opätovnom použití kódu.

Ciele výučby

1. Prečítajte si, ako používať ultrazvukový senzor
2. Prehľad definovania a volania funkcií
3. Naučte sa a osvojte si používanie parametrov funkcií
4. Vykonajte experiment s ultrazvukovým parkovacím senzorom.

Náhľad výsledku



Po nahratí programu do Arduino Nano R4 systém začne v reálnom čase monitorovať

vzdialenosť pred vozidlom v reálnom čase. Ultrazvukový senzor nepretržite meria vzdialenosť medzi sebou a akoukoľvek prekážkou a výsledky sa zobrazujú v reálnom čase v sériovom monitore.

• **Ak je vzdialenosť väčšia ako 20 cm**: Systém považuje situáciu za bezpečnú. Zvukový signál neznie a červená LED svieti.

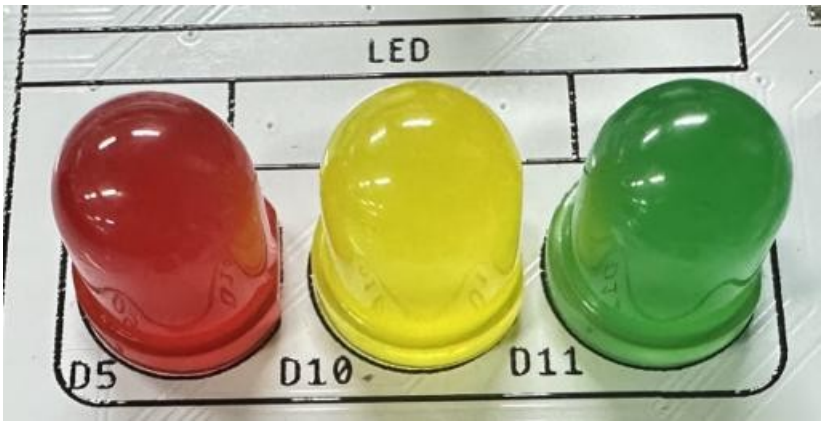
• **Keď je vzdialenosť medzi 11 cm a 20 cm**: Systém zistí, že prekážka je relatívne blízko. Zvukový signál znie so strednou frekvenciou a červená LED bliká strednou rýchlosťou ako varovanie.

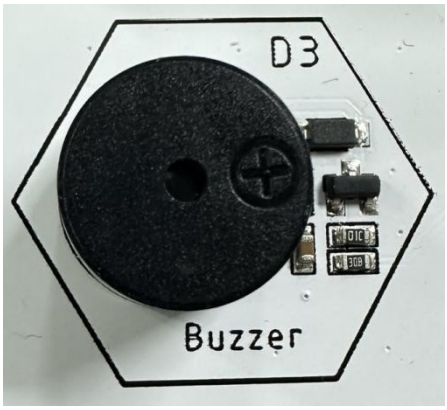
• **Keď je vzdialenosť menšia ako 11 cm**: Systém určí, že vzdialenosť je nebezpečná. Bzučiak znie s vysokou frekvenciou a červená LED rýchlo bliká, aby vydala varovanie.

Na dokončenie tohto projektu parkovacieho alarmu pri cúvaní musíme definovať dve funkčné rutiny:

1. Funkcia, ktorá pomocou ultrazvukového modulu meria vzdialenosť a vráti nameranú hodnotu v reálnom čase.
2. Funkcia, ktorá ovláda bzučiak a červenú LED diódu na generovanie rôznych typov upozornení na základe nameranej vzdialenosti.

Hardvér použitý v tejto lekcii



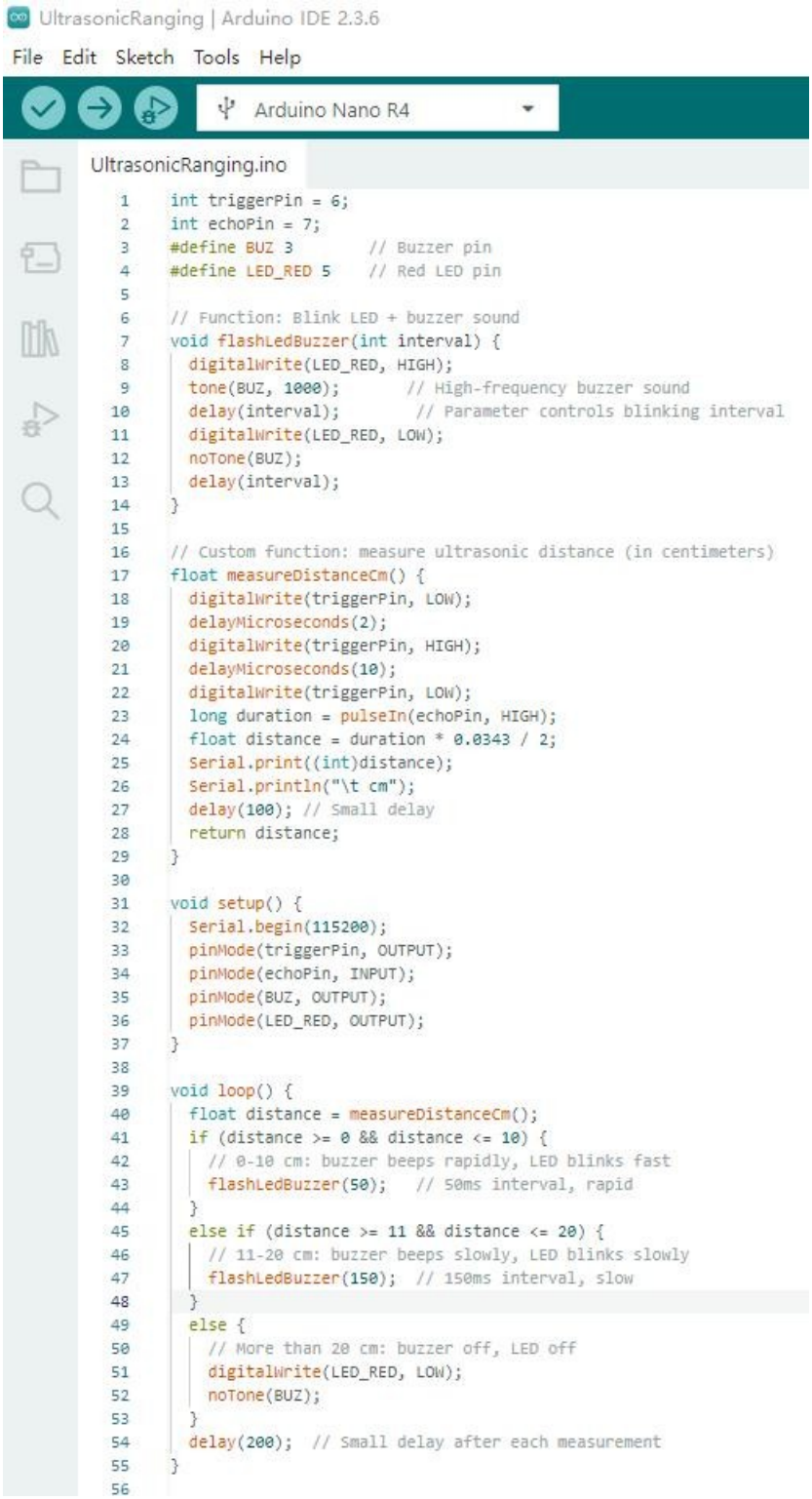


Ultrazvukový parkovací alarm

Než sa pustíte do vysvetľovania kódu, môžete si najskôr stiahnuť kompletný program. Odkaz na stiahnutie kompletného kódu:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/17_Reverse_Parking_Alarm_Radar
Otvorte priečinok „17_Reverse_Parking_Alarm_Radar“ v prostredí Arduino IDE a potom otvorte súbor „17_Reverse_Parking_Alarm_Radar.ino“, ktorý sa v ňom nachádza.

Vysvetlenie kľúčových kódov

Teraz si prejdime príklad projektu: **Ultrazvukový parkovací senzor pre cúvanie**.




```
int triggerPin = 6;
int echoPin = 7;
#define BUZ 3        // Buzzer pin
#define LED_RED 5    // Red LED pin

// Function: Blink LED + buzzer sound
void flashLedBuzzer(int interval) {
  digitalWrite(LED_RED, HIGH);
  tone(BUZ, 1000);       // High-frequency buzzer sound
  delay(interval);        // Parameter controls blinking interval
  digitalWrite(LED_RED, LOW);
  noTone(BUZ);
  delay(interval);
}

// Custom function: measure ultrasonic distance (in centimeters)
float measureDistanceCm() {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  float distance = duration * 0.0343 / 2;
  Serial.print((int)distance);
  Serial.println("\t cm");
  delay(100); // Small delay
  return distance;
}

void setup() {
  Serial.begin(115200);
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(BUZ, OUTPUT);
  pinMode(LED_RED, OUTPUT);
}

void loop() {
  float distance = measureDistanceCm();
  if (distance >= 0 && distance <= 10) {
    // 0-10 cm: buzzer beeps rapidly, LED blinks fast
    flashLedBuzzer(50);   // 50ms interval, rapid
  }
  else if (distance >= 11 && distance <= 20) {
    // 11-20 cm: buzzer beeps slowly, LED blinks slowly
    flashLedBuzzer(150);  // 150ms interval, slow
  }
  else {
    // More than 20 cm: buzzer off, LED off
    digitalWrite(LED_RED, LOW);
    noTone(BUZ);
  }
  delay(200);  // Small delay after each measurement
}
```

Najskôr definujeme piny potrebné pre túto lekciu:

```
int triggerPin = 6; int
echoPin = 7;
#define BUZ 3            // Pin bzučiaka
#define LED_RED 5        // Pin červenej LED
```

triggerPin: Toto je pin spúšťača (vysielania), pripojený k D6 na Arduino Nano R4. echoPin: Toto je pin ozveny (prijímania), pripojený k D7 na Arduino Nano R4.

Na definovanie funkcie alarmu použite void

```
void flashLedBuzzer(int interval) {
  digitalWrite(LED_RED, HIGH);
  tone(BUZ, 1000);              // Vysokofrekvenčný zvuk bzučiaka
  delay(interval);              // Parameter ovláda interval blikania
  digitalWrite(LED_RED, LOW);
  noTone(BUZ);
  delay(interval);
}
```

Naše upozornenie funguje takto: červená LED sa rozsvieti a zaznie bzučiak → potom sa červená LED zhasne a bzučiak prestane znieť. Parameter „interval“ je kľúčom k ovládaniu rôznych frekvencií upozornení – čím kratší interval, tým naliehavejšie je upozornenie; čím dlhší interval, tým jemnejšie je upozornenie.

void: Určuje typ návratovej hodnoty funkcie. Použitie „void“ znamená, že funkcia nevracia žiadnu hodnotu.
flashLedBuzzer: Toto je názov funkcie. Názov funkcie nemôže začínať číslom, ale môže obsahovať písmená, čísla a podčiarknutia, pričom sa rozlišujú veľké a malé písmená. Najdôležitejšie je, aby jasný názov funkcie pomohol čitateľom okamžite pochopiť, čo funkcia robí.
int interval: Toto definuje parameter, ktorý sa odovzdáva do funkcie delay, čo znamená, že sa používa na riadenie dĺžky oneskorenia pri volaní funkcie.
Pri volaní funkcie odovzdávame argument v zátvorkách. Tento argument nahrádza parameter v celom tele funkcie.
Parameter: Premenná uvedená v zátvorkách pri definovaní funkcie. Slúži na prijímanie hodnôt odovzdaných zvonku funkcie a je to lokálna premenná, čo znamená, že je platná len v rámci samotnej funkcie.
Argument: Skutočná hodnota alebo premenná uvedená v zátvorkách pri volaní funkcie. Tieto hodnoty sa priradia k parametrom, aby ich funkcia mohla použiť.

Definujte funkciu na meranie vzdialenosti pomocou ultrazvuku s použitím typu float (kľúčové zameranie).

```
float measureDistanceCm() {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2); digitalWrite(triggerPin,
  HIGH); delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  long duration = pulseIn(echoPin, HIGH); float
  distance = duration * 0.0343 / 2;
  Serial.print((int)distance); Serial.println("\t
  cm");
  delay(100); // Malé oneskorenie
  return distance;
}
```

Kľúčové body, ktoré je potrebné pochopiť:

Z kódu môžeme vidieť nasledujúce kroky:
①Najskôr sa pin „triggerPin“ nastaví na úroveň LOW a udrží sa v tomto stave po dobu 2 mikrosekúnd. Tým sa zabezpečí, že pin je v stabilnom stave LOW pred odoslaním spúšťacieho impulzu.
②Ďalej sa „triggerPin“ nastaví na HIGH a udrží sa na tejto úrovni po dobu 10 mikrosekúnd. Tento krok iniciuje ultrazvukový impulz. Pre väčšinu ultrazvukových modulov je 10 mikrosekúnd minimálna požadovaná šírka spúšťacieho

.
③Po uplynutí 10 mikrosekúnd sa pin opäť nastaví na LOW, čím sa impulz ukončí a dokončí sa jeden ultrazvukový prenos.

delayMicroseconds(2): odloží vykonanie o 2 mikrosekundy delay(2):

odloží vykonanie o 2 milisekundy

long duration = pulseIn(echoPin, HIGH);

Typ „long“ dokáže uchovávať väčšie celé čísla ako typ „int“, a preto používame typ „long“ na definovanie premennej „duration“, keďže pracujeme s číslami v mikrosekundovom rozsahu.

pulseIn() je funkcia Arduina, ktorá meria, ako dlho zostáva pin na určitej úrovni napätia. Tu ju používame s „echoPin“ a „HIGH“, čo znamená, že meria dobu, počas ktorej „echoPin“ zostáva na úrovni HIGH. Po tom, čo ultrazvukový modul vysiela impulz, „echoPin“ prejde do stavu HIGH a keď sa odrazená zvuková vlna vráti, „echoPin“ prejde do stavu LOW. Meranie doby trvania stavu HIGH „echoPin“ tak poskytuje celkový čas, ktorý ultrazvukový impulz potrebuje na to, aby sa dostal k prekážke a späť.

float distance = duration * 0.0343 / 2;

Typ float sa používa na definovanie premennej s plávajúcou desatinnou čiarkou (čísla s desatinnými miestami). Keďže výpočty zahŕňajú desatinné hodnoty, definujeme premennú ako float. Použitie typov int alebo long by uložilo iba celé čísla, pričom by sa zlikvidovala desatinná časť a došlo by k strate presnosti. Použitie typu float zachováva desatinné miesta, čím sa meranie vzdialenosti stáva presnejším.

duration je čas, za ktorý ultrazvukový impulz prejde k prekážke a späť.

0,0343 predstavuje rýchlosť zvuku prepočítanú na centimetre za mikrosekundu: 343 metrov za sekundu sa rovná 0,0343 cm/µs.

vrátiť vzdialenosť;

Funkcia vráti hodnotu – konečnú nameranú vzdialenosť. Po volaní funkcie „measureDistanceCm“ môžeme jej vrátenú hodnotu uložiť do premennej, čím získame

zmeranú vzdialenosť.

Inicializačná funkcia

```
void setup() { Serial.begin(115200);
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(BUZ, OUTPUT);
  pinMode(LED_RED, OUTPUT);
}
```

Poznámka: Tu je „triggerPin“ výstupný pin, zatiaľ čo „echoPin“ je vstupný pin. Dávajte pozor, aby ste

nezamieňate pri nastavovaní režimov pinov.

Funkcia loop

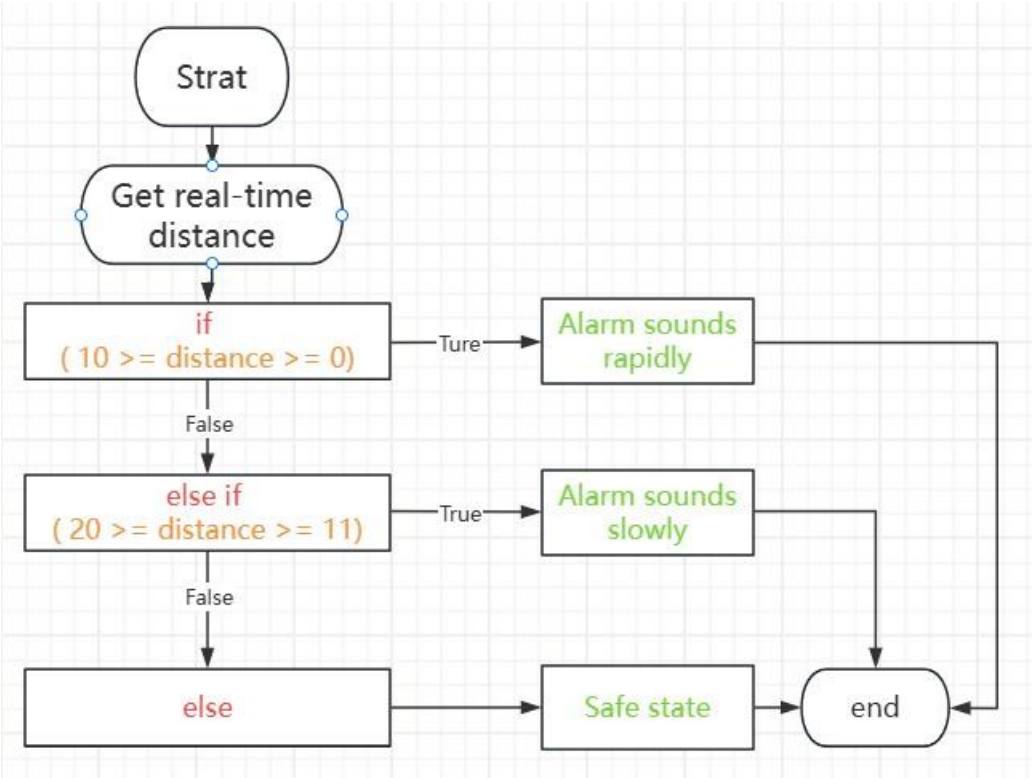
```
void loop() {
  float distance = measureDistanceCm(); if
  (distance >= 0 && distance <= 10) {
    // 0–10 cm: bzučiak rýchlo pípne, LED rýchlo bliká
    flashLedBuzzer(50);         // 50 ms interval, rýchlo
  }
  inak ak (vzdialenosť >= 11 a vzdialenosť <= 20) {
    // 11–20 cm: bzučiak pomaly pípne, LED pomaly bliká flashLedBuzzer(150);
    // interval 150 ms, pomaly
  }
  else {
    // Viac ako 20 cm: bzučiak vypnutý, LED
    vypnutá digitalWrite(LED_RED, LOW);
    noTone(BUZ);
  }
  delay(200); // Krátke oneskorenie po každom meraní
}
```

Keďže sme už definovali funkcie ako samostatné funkcie – napríklad „measureDistanceCm“ spracováva meranie vzdialenosti a „flashLedBuzzer“ môže vytvárať rôzne výstražné zvuky jednoduchým odovzdaním rôznych parametrov – naša hlavná slučka sa stáva veľmi stručnou.

float distance = measureDistanceCm();
Ako sme už spomínali, „measureDistanceCm“ vráti nameranú vzdialenosť, ktorú uložíme do premennej distance. Vďaka tomu je ľahké ju použiť v nasledujúcom kóde na rozhodnutia založené na vzdialenosti.

Ak je vzdialenosť 0–10 cm, bzučiak rýchlo pípne a červená LED bliká s vysokou frekvenciou.
Ak je vzdialenosť 11–20 cm, bzučiak pípne pomaly a červená LED bliká strednou rýchlosťou.
Keď je vzdialenosť väčšia ako 20 cm, systém je v bezpečnom stave: bzučiak mlčí a červená LED svieti.

Celkový diagram logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Postup nahratia“ na strane 8.

Lekcia 18 – Pamäťová hra s blikajúcim LED svetlom

Úvod

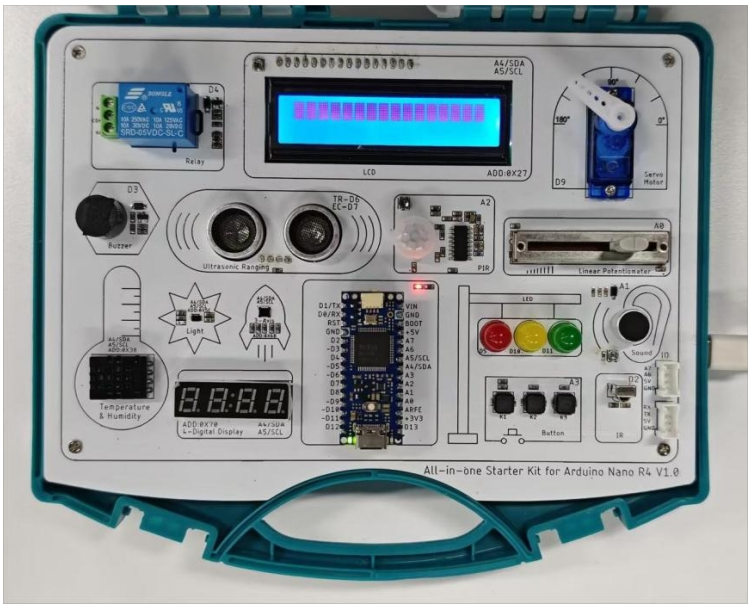
V tejto lekcii sa naučíme, ako pomocou LED diód a tlačidiel vytvoriť jednoduchú, ale veľmi klasickú pamäťovú hru.
V tomto projekte bude Arduino najprv ovládať tri LED diódy, ktoré budú rýchlo blikať v určitom poradí. Študenti si musia zapamätať poradie blikania LED diód a potom stlačením príslušných tlačidiel musia reprodukovať proces blikania LED diód v správnom poradí. Ak je poradie úplne správne, úloha je úspešná; ak sa v ktoromkoľvek bode vyskytne chyba, úloha zlyhá a študenti musia začať od začiatku.
Vďaka tomuto kurzu sa študenti nenaučia len „rozsvietiť LED diódu“, ale budú schopní pochopiť základné princípy náhodných čísel, polí, sekvenčného posudzovania a
interakcie človeka s počítačom. Ide o veľmi dôležitý komplexný kurz zameraný na zdokonalenie vedomostí.

Ciele výučby

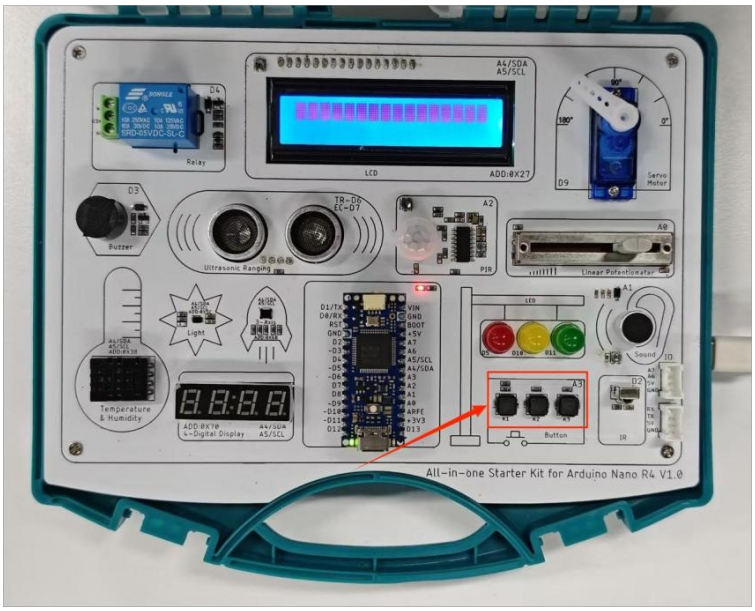
1. Porozumieť základnému konceptu implementácie sekvenčných pamäťových hier.
2. Ovládať ovládanie blikania LED a zapuzdrenie funkcií.
3. Naučiť sa používať polia na ukladanie údajov v poradí.
4. Zjednoťte metódu simulácie stlačenia klávesov (delenie napätia na rezistore) tak, aby bolo možné čítať stlačenie viacerých klávesov.
5. Byť schopný nastaviť obtiažnosť hry (rýchlosť a frekvenciu blikania) prostredníctvom premenných.

Náhľad výsledku

Po úspešnom nahratí kódu uvidíte nasledujúci efekt:
Po zapnutí budú tri LED diódy blikať v náhodnom poradí jedna po druhej (napríklad: žltá -> zelená -> zelená -> červená -> zelená)



Akonáhle LED prestane blikať, je na rade vaša operácia.
Musíte použiť tlačidlá podľa svojej pamäti a stlačiť ich v presne rovnakom poradí, ako blikala LED.
(Napríklad: postupujte podľa sekvencie uvedenej v predchádzajúcom príklade: žltá -> zelená -> zelená -> červená -> zelená)



Ak sú všetky správne:

➢ Tri LED diódy sa rozsvietia súčasne na 1 sekundu, čo znamená, že ste úlohu úspešne splnili.

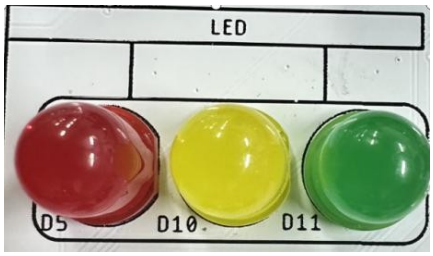
Ak počas procesu urobíte chybu:

➢ Tri LED diódy budú rýchlo blikať, čo znamená, že úloha nebola úspešná.

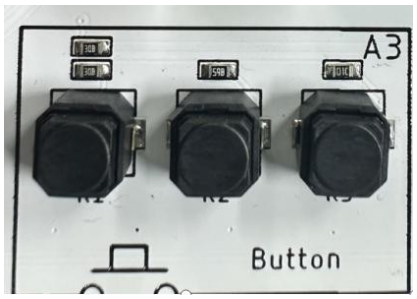
Potom sa hra automaticky reštartuje a prejde do ďalšieho kola.

Hardvér použitý v tejto lekcii

Trojfarebné LED diódy



Tri tlačidlá



Puzdro na pamäťovú hru s blikajúcimi LED diódami

Pred vysvetlením kódu si môžeme kód stiahnuť. Odkaz na stiahnutie celého kódu je:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/18_LED_Light_Flashing_Memory_Game
Otvorte priečinok „18_LED_Light_Flashing_Memory_Game“ v prostredí Arduino IDE a vyberte súbor s kódom „18_LED_Light_Flashing_Memory_Game.ino“

Vysvetlenie kódov klávesov

Po otvorení kódu uvidíte kód k tejto lekcii:

18_LED_Light_Flashing_Memory_Game.ino

```
// ================== Pin Definitions ==================
#define LED_RED     5
#define LED_YELLOW  10
#define LED_GREEN   11

#define Button_Pin  A3   // Analog button input

// ================== Game Settings ==================
#define MAX_LEVEL     10

int levelLength = 5;      // Sequence length (difficulty)
int flashDelay  = 400;    // LED flash speed (ms)

// ================== Variables ==================
int ledSequence[MAX_LEVEL];
int userIndex = 0;

// ================== Setup ==================
void setup() {
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(Button_Pin, INPUT);

  Serial.begin(115200);
  randomSeed(analogRead(A0));

  startGame();
}

// ================== Main Loop ==================
void loop() {
  int button = readButton();

  if (button != 0) {
    flashSingleLED(button);

    if (button == ledSequence[userIndex]) {
      userIndex++;

      // All correct
```

Definícia pinov

```
#define LED_RED          5
#define LED_YELLOW       10
#define LED_GREEN        11
#define Pin_tlačidla     A3      // Analógový vstup tlačidla
```

LED_RED, LED_YELLOW a LED_GREEN zodpovedajú digitálnym pinom 5, 10 a 11 a slúžia na ovládanie červenej, žltej a zelenej LED diódy. Vďaka tomu bude kód v nasledujúcich funkciách pinMode() a digitalWrite() veľmi intuitívny na čítanie – keď uvidíte LED_RED, hneď viete, že ide o ovládanie červenej LED diódy, bez toho, aby ste museli opakovane kontrolovať, „čo je pripojené na

pin 5?“.

Button_Pin je definovaný ako A3 a špecificky označený ako analógový vstup tlačidla, čo naznačuje, že nejde o jednoduché digitálne tlačidlo, ale o viacero tlačidiel, ktoré zdieľajú jeden analógový vstupný pin prostredníctvom delenia napätia rezistorom. Neskôr v programe sa pomocou analogRead(A3) načítajú rôzne rozsahy napätia, aby sa určilo, ktoré tlačidlo bolo stlačené.

Definícia premenných

```
#define MAX_LEVEL          10
int levelLength = 5;       // Dĺžka sekvencie (náročnosť)
int flashDelay     = 400;   // Rýchlosť blikania LED (ms)
int ledSequence[MAX_LEVEL];
int userIndex = 0;
```

#define MAX_LEVEL 10 Definuje konštantu pre maximálnu dĺžku úrovne, ktorá nielen obmedzuje hornú hranicu obtiažnosti hry, ale tiež určuje maximálnu kapacitu poľa ledSequence, aby sa predišlo chybám prekročenia hraníc poľa. Ide o dôležitý bezpečnostný prvok v programovaní vstavaných systémov.

levelLength = 5 Určuje dĺžku svetelnej sekvencie, ktorú je potrebné si zapamätať v aktuálnom kole hry. Čím väčšia je hodnota, tým viac krokov si hráč musí zapamätať.

Preto ide v podstate o parameter na ovládanie obtiažnosti hry. Ak chcete v budúcnosti hru sťažiť, stačí zmeniť túto hodnotu.

flashDelay = 400 Slúži na nastavenie dĺžky svietenia jednotlivých LED diód, pričom jednotkou sú milisekundy. Priamo ovplyvňuje rytmus, v akom hráč „jasne vidí svetlá“. Čím je hodnota nižšia, tým je blikanie rýchlejšie a tým je reakcia náročnejšia.

Nasledujúce ledSequence[MAX_LEVEL] je pole celých čísel, ktoré slúži na ukladanie náhodne generovanej sekvencie LED systémom. Každé číslo v poli predstavuje svetlo (napríklad 1 = červené svetlo, 2 = žlté svetlo, 3 = zelené svetlo). Ide o základnú dátovú štruktúru celej logiky pamäte. userIndex = 0 Je premenná ukazovateľa pokroku, ktorá slúži na zaznamenávanie kroku, ktorý hráč doteraz správne zadal. Zakaždým, keď hráč odpovie správne, tento index sa zvýši o jednotku. Akonáhle dosiahne hodnotu levelLength, znamená to, že hráč si úspešne zapamätal celú úroveň.

Sekcia nastavenia

```
void setup() { pinMode(LED_RED,
    OUTPUT);
    pinMode(LED_YELLOW, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
    pinMode(Button_Pin, INPUT);

    Serial.begin(115200);
    randomSeed(analogRead(A0));

    startGame();
}
```

Táto funkcia setup() je zodpovedná za inicializáciu celej hry s pamäťovým svetlom pri jej zapnutí alebo resete. Môžeme ju chápať ako „prípravnú fázu predtým, ako sa hra

oficiálne začne“.

Najprv sa pomocou funkcií pinMode(LED_RED, OUTPUT), pinMode(LED_YELLOW, OUTPUT) a pinMode(LED_GREEN, OUTPUT) nastavia piny pripojené k trom LED diódam do výstupného režimu. To umožňuje Arduinu aktívne vysielať na tieto piny signály s vysokou alebo nízkou úrovňou, čím sa ovláda stav zapnutia/vypnutia svetiel;
zatiaľ čo pinMode(Button_Pin, INPUT) nastaví analógový pin, na ktorom sa nachádza tlačidlo, do vstupného režimu, ktorý sa používa na čítanie operácií hráča. Toto je vstupný bod pre interakciu človeka s počítačom.

Potom Serial.begin(115200) inicializuje sériovú komunikáciu, ktorá sa používa hlavne na ladenie a sledovanie stavu behu programu, napríklad na kontrolu analógovej hodnoty tlačidla alebo priebehu hry. To je veľmi dôležité počas fázy učenia a riešenia problémov.

„randomSeed(analogRead(A0))“ je kľúčový, ale často prehliadaný detail. Inicializuje počiatočnú hodnotu generátora náhodných čísel načítaním hodnoty z nepripojeného alebo rušeného analógového pinu, čím sa zabezpečí, že svetelná sekvencia generovaná funkciou „random()“ bude pri každom zapnutí napájania iná. V opačnom prípade by sa herná sekvencia opakovala, čím by sa stratil zmysel „náhodnej výzvy“.
„randomSeed()“ je funkcia v Arduine používaná na inicializáciu sekvencie náhodných čísel a „analogRead(A0)“ číta hodnotu napätia na analógovom pine A0. V skutočnom obvode, ak nie je pin A0 pripojený k žiadnemu stabilnému signálu, bude ovplyvnený šumom z okolia a hodnota načítaná pri každom zapnutí bude mierne odlišná. Práve táto „nepredvídateľná malá odchýlka“ slúži
ako zdroj náhodného semena. Nasledujúce volania „random()“ vygenerujú cieľové hodnoty, ktoré nebudú opakovať rovnakú sadu čísel pri každom zapnutí napájania, čo je mimoriadne dôležité pre zvýšenie spravodlivosti a zábavnosti hry.

Posledná volaná funkcia, startGame(), je vlastne „príkazom na začatie hry“. Vynuluje stav hráčových vstupov, vygeneruje novú náhodnú svetelnú sekvenciu a zobrazí ju hráčovi. To zodpovedá prepojeniu všetkých predchádzajúcich hardvérových a systémových príprav a oficiálnemu začatiu prvého kola hry.

loop

```
void loop() {
  int button = readButton();

  if (button != 0) {
    flashSingleLED(button);

    if (button == ledSequence[userIndex]) {
      userIndex++;

      // Všetko správne
      if (userIndex >= levelLength) {
```

```
            successLED();
            delay(500);
            startGame();
        }
    } inak {
        failLED();
        delay(500);
        startGame();
    }

    delay(300); // Zabráni tomu, aby bolo stlačenie jedného tlačidla zaznamenané viackrát
  }
}
```

Táto funkcia loop() je základnou logickou slučkou celej hry s pamäťovými svetlami, ktorú možno považovať za „systém nepretržitého monitorovania vstupov, ktorý okamžite posudzuje správnosť alebo nesprávnosť a slúži ako rozhodca“.

Na začiatku programu funkcia readButton() zistí, ktoré tlačidlo stlačil aktuálny hráč, a výsledok uloží do premennej button. Vrátené hodnoty 1/2/3 tu predstavujú červené, žlté a zelené svetlo, zatiaľ čo hodnota 0 znamená, že momentálne nie je stlačené žiadne tlačidlo. Len ak je button != 0, znamená to, že hráč skutočne vykonal nejakú operáciu, a program bude pokračovať v spracovaní, čím sa zabráni nečinnému behu a nesprávnemu posúdeniu situácie.

```
31  // ================== Main Loop ==================
32  void loop() {
33    int button = readButton();
```

Keď sa zistí stlačenie tlačidla, najskôr sa volá funkcia flashSingleLED(button). Tá spôsobí, že príslušná farebná LED dióda okamžite raz zabliká. Tento krok slúži ako vizuálna spätná väzba, ktorá informuje hráča, že „tvoje tlačidlo bolo rozpoznané“, a zároveň robí proces ovládania intuitívnejším a podobnejším hre.

```
35    if (button != 0) {
36      flashSingleLED(button);
```

Ďalej sa dostávame k najdôležitejšej logike posudzovania: if (button == ledSequence[userIndex]) – tu sa aktuálna klávesa stlačená hráčom porovná so správnou odpoveďou na „aktuálnej pozícii“ v predgenerovanej svetelnej sekvencii systémom; ak sú rovnaké, znamená to, že tento vstup je správny, takže sa vykoná userIndex++, čo znamená, že hráč úspešne dokončil jednu pozíciu v aktuálnej sekvencii a systém je pripravený skontrolovať ďalšiu.

```
38      if (button == ledSequence[userIndex]) {
39        userIndex++;
```

Nasledujúce if (userIndex >= levelLength) sa používa na určenie, či boli dokončené všetky svetelné sekvencie tohto kola. Keď hráč správne usporiadal celú sekvenciu, program vyvolá funkciu successLED(), aby prehral svetelný efekt úspechu, čím poskytne hráčovi jasnú pozitívnu spätnú väzbu; potom sa delay(500) na chvíľu pozastaví, aby bol efekt úspechu jasne viditeľný. Nakoniec sa vyvolá startGame(), čím sa vygeneruje nová

svetelnú sekvenciu a spustí ďalšie kolo výziev, čo je ekvivalentné „prejdeniu úrovne a vstupu do nového kola“.

```
41      // All correct
42      if (userIndex >= levelLength) {
43        successLED();
44        delay(500);
45        startGame();
46      }
```

Ak klávesa stlačená hráčom nezodpovedá správnemu poradí, program prejde do vetvy „else“, čím signalizuje nesprávny vstup. V tomto momente program volá funkciu failLED(), ktorá prostredníctvom rýchleho blikania alebo iným spôsobom upozorní na neúspech. Po krátkom oneskorení 500 milisekúnd volá aj funkciu startGame(), čím priamo reštartuje hru a umožní hráčovi ďalší pokus. Tento dizajn je jednoduchý a zároveň zodpovedá intuitívnemu chápaniu herného procesu zo strany začiatočníka.

```
47      } else {
48        failLED();
49        delay(500);
50        startGame();
51      }
```

Záverečné „delay(300)“ je veľmi praktický malý detail. Slúži ako softvérový mechanizmus proti chveniu a duplicite, ktorý zabraňuje tomu, aby program opakovane čítal stlačenie jedného tlačidla v dôsledku toho, že prst sa včas neuvoľnil, čím sa zabraňuje chybám pri zadávaní.

```
53      delay(300); // Prevent a single key press from being read multiple times
54    }
55  }
```

Celkovo je štruktúra tejto funkcie loop() jasná: načítať vstup → poskytnúť spätnú väzbu → určenie správnosti → rozhodnutie o úspechu alebo neúspechu → riadenie tempa hry.

Funkcia startGame

```
void startGame() {
  userIndex = 0;
  generateSequence();
  showSequence();
}
```

Túto funkciu s názvom „startGame()“ možno považovať za „inicializáciu a otvárací ceremoniál každej hernej relácie“. Volá sa vždy, keď sa hra spustí, hráč úspešne dokončí úroveň alebo zlyhá a musí to skúsiť znova. Jej úlohou je obnoviť stav systému do čistého a kontrolovateľného východiskového bodu.

Prvý riadok funkcie, „userIndex = 0;“, slúži na vynulovanie aktuálneho indexu pozície vstupu hráča, čo znamená, že hráč ešte nezačal zadávať žiadnu svetelnú sekvenciu. Je to rovnaké, ako keby sme programu povedali: „Začni znova porovnávať od prvého svetla.“

Potom sa volá funkcia „generateSequence()“. Tento riadok je kľúčovou súčasťou generovania obsahu pre túto hernú reláciu, ktorou je náhodné generovanie sekvencie LED diód a jej uloženie do

pole „ledSequence“. Vďaka použitiu náhodných čísel tento krok zaručuje, že kombinácia rozsvietenia LED diód je pri každom spustení hry iná, čím sa zvyšuje náročnosť a hrateľnosť hry a zabraňuje sa tomu, aby hráči „podvádzali“ tým, že by sa spoliehali na zapamätanie si predchádzajúcej sekvencie. Nakoniec zavolajte funkciu showSequence(); Ide o veľmi dôležitú „prezentačnú fázu“ v hre. Systém postupne rozsvieti LED diódy v práve vygenerovanom poradí a hráčovi tak v plnom rozsahu predstaví správnu odpoveď.

To znamená, že funkcia startGame() vykonáva všetky tri úlohy – „vymazanie stavu → generovanie otázok → prezentovanie otázok hráčovi“ – naraz, čím pripravuje základ pre vstup hráča a logiku posudzovania v nasledujúcej slučke loop().

Funkcia generateSequence

```
void generateSequence() {
  for (int i = 0; i < levelLength; i++) {
    ledSequence[i] = random(1, 4); // 1 = červená, 2 = žltá, 3 = zelená
  }
}
```

Hlavnou funkciou tejto funkcie generateSequence() je generovanie „náhodnej sekvencie svetiel“ pre aktuálne kolo pamäťovej hry. Môžeme ju považovať za proces, ktorým systém automaticky „nastavuje otázku“ na začiatku každej úrovne.

Celá štruktúra funkcie je veľmi prehľadná: vonkajšia vrstva je cyklus for, počet iterácií je určený premennou levelLength a táto premenná sama o sebe predstavuje „koľko svetiel je potrebné si v tejto úrovni zapamätať“, čo je priamym odrazom obtiažnosti hry – čím väčšia je hodnota, tým dlhšia je sekvencia, ktorú si hráč musí zapamätať, a tým väčšia je výzva.

V rámci cyklu je najdôležitejší riadok „ledSequence[i] = random(1, 4);“. Tu sa používa funkcia random() poskytovaná platformou Arduino. Význam výrazu „random(1, 4)“ spočíva vo vygenerovaní náhodného celého čísla, ktoré je väčšie alebo rovné 1 a menšie ako 4. Inými slovami, výsledkom môže byť iba 1, 2 alebo 3. Pri návrhu programu boli týmto trom číslam umelo priradené významy: 1 predstavuje červenú LED, 2 predstavuje žltú LED a 3 predstavuje zelenú LED. Jednoduché číslo tak môže priamo vyjadrovať, „ktoré svetlo by malo svietiť ako ďalšie“.

Počas každého cyklu sa vygeneruje nová náhodná hodnota, ktorá sa postupne ukladá do poľa ledSequence, pričom jej pozícia je určená indexom i. Výsledkom je, že pole nielen ukladá informáciu o tom, „ktoré svetlá svietia“, ale aj úplne zachováva poradie, v akom sa svetlá rozsvietili. V nasledujúcom programe funkcia showSequence() rozsvieti LED diódy v poradí tohto poľa jedna po druhej, aby si ich hráč ľahšie zapamätal; zatiaľ čo pri zadávaní hráčom sa toto pole bude krok za krokom porovnávať s operáciou hráča, aby sa určilo, či je správne.

Je veľmi dôležité zapuzdriť „generovanie náhodných sekvencií“ do samostatnej funkcie: na jednej strane to robí hlavný proces (napríklad startGame()) intuitívnejším na čítanie; na druhej strane to ponecháva priestor pre budúce rozšírenie hry, ako je pridanie nových farieb LED, dynamické zvyšovanie levelLength a dokonca zavedenie rôznych režimov obtiažnosti. Všetko toto môže

dosiahnuť jednoducho niekoľkými úpravami v tejto časti.

funkcia showSequence

```
void   showSequence()   {
   delay(800);
   for  (int  i  =  0;  i  <  levelLength;  i++)  {
      flashSingleLED(ledSequence[i]);
      delay(200);
   }
}
```

Funkcia showSequence() slúži na prezentáciu náhodne generovanej svetelnej sekvencie, ktorú práve vygeneroval systém, hráčom. Môžeme ju považovať za fázu v pamäťovej hre, kde sa „predkladajú otázky a prehrávajú nápovedy“. Počas celého priebehu hry plní úlohu „ukážky učiteľa“: najprv vám povie, aká je sekvencia, a potom je na rade vaša operácia.

Inštrukcia „delay(800);“ na začiatku funkcie nebola pridaná náhodne. Jej účelom je poskytnúť hráčovi časovú rezervu pred spustením svetelného efektu. Zvyčajne po skončení predchádzajúceho kola alebo na začiatku nového kola potrebuje hráč nejaký čas na prechod z textu výzvy alebo mentálneho stavu. Táto 800-milisekundová pauza je ekvivalentom odpočítavania na „prípravu na štart“, vďaka čomu je svetelná show menej náhla.

Ďalej nasleduje cyklus for a počet iterácií je tiež určený premennou levelLength, čo je presne to isté ako vo funkcii generateSequence(). Tým sa zabezpečí, že „počet generovaných svetiel je rovnaký ako počet zobrazených svetiel“. Kľúčovým príkazom v rámci cyklu je flashSingleLED(ledSequence[i]);

Tu je kľúčový bod: Funkcia ledSequence[i] nevyvolá konkrétne pevne dané svetlo, ale i-té svetlo, ktoré bolo náhodne vygenerované a predtým uložené do poľa.
To znamená, že systém prehráva celú svetelnú sekvenciu presne v poradí, v akom bola vygenerovaná, bez toho, aby niečo vynechal alebo preskočil. Funkcia flashSingleLED() je zodpovedná za konkrétne hardvérové akcie – zapnutie príslušnej LED, oneskorenie a následné vypnutie. Funkcia showSequence() sa teda nemusí zaoberať základnými detailmi; stačí, ak „určí, ktoré svetlo má blikať“.

Nasledujúci riadok „delay(200);“ predstavuje časový interval medzi dvoma susednými svetlami. Bez tohto oneskorenia by blikanie viacerých LED diód bolo nepretržité, čo by pôsobilo veľmi chaoticky a hráč by takmer nedokázal rozoznať sekvenciu. Vďaka tejto 200-milisekundovej medzere sa každé svetlo stáva zreteľným a jasným „bodom v pamäti“, čo výrazne zlepšuje čitateľnosť a herný zážitok.

Funkcia readButton

```
int readButton() {
   int val = analogRead(Button_Pin);
```

```
    Serial.println(val);

    if (val >= 500 && val <= 520) return 1;          // Červená
    if (val >= 680 && val <= 690) return 2;          // Žltá if
    (val >= 845 && val <= 860) return 3;             // Zelená

    vráť 0;
}
```

Funkcia „readButton()“ slúži na zistenie, ktoré tlačidlo hráč stlačil, a na jeho prevod na číslo, s ktorým môže hra pracovať. Ide o „kľúčové rozhranie na detekciu vstupov od hráča“ v celej pamäťovej hre a možno ju chápať ako „prekladač akcií hráča“: keď hráč stlačí fyzické tlačidlo, táto funkcia premení túto akciu na digitálne čísla 1, 2 alebo 3, ktorým program rozumie.

Najprv riadok „int val = analogRead(Button_Pin);“ prečíta hodnotu napätia cez analógový pin. Keďže tri tlačidlá sú pripojené k rovnakému analógovému vstupu cez rezistory na delenie napätia, každé stlačenie tlačidla generuje iné napätie, zodpovedajúce inému číselnému rozsahu. Použitie analogRead nám umožňuje získať celočíselnú hodnotu v rozmedzí od 0 do 1023.

Nasledujúci príkaz Serial.println(val); je ladiaci príkaz, ktorý vypíše aktuálnu analógovú hodnotu na sériový monitor. To je veľmi užitočné pri ladení citlivosti tlačidiel, chyby rezistorov alebo hardvérových pripojení. Pri stlačení každého tlačidla môžete priamo vidieť zodpovedajúce hodnoty napätia, čo je výhodné pre kalibráciu.

Ďalej nasledujú tri podmienky typu „if“:

```
87    if (val >= 500 && val <= 520) return 1;   // Red
88    if (val >= 680 && val <= 690) return 2;   // Yellow
89    if (val >= 845 && val <= 860) return 3;   // Green
90
```

Tieto riadky slúžia na priradenie analógového napätia k číslu tlačidla. Každé tlačidlo má primeraný rozsah napätia. Napríklad, keď je stlačené tlačidlo červeného svetla, hodnota napätia je približne medzi 500 a 520 a funkcia vráti 1; žlté svetlo vráti 2 a zelené svetlo vráti 3. Prostredníctvom tohto priradenia môže program interne zistiť, ktoré tlačidlo hráč stlačil, bez toho, aby priamo spracovával hodnotu napätia.

Záverečné „return 0;“ je predvolená návratová hodnota, ktorá označuje, že nebolo stlačené žiadne tlačidlo. Keď nie je stlačené žiadne z troch tlačidiel, funkcia vráti 0, čím dá hlavnej slučke vedieť, že „v tomto momente nie je žiadny vstup“.

Funkcia flashSingleLED

```
void flashSingleLED(int num) {
    allLEDOff();

    if (num == 1) digitalWrite(LED_RED, HIGH);
```

```
    if (num == 2) digitalWrite(LED_YELLOW, HIGH); if
    (num == 3) digitalWrite(LED_GREEN, HIGH);

    delay(flashDelay);
    allLEDOff();
}
```

Funkcia flashSingleLED(int num) slúži na jednorazové zablikanie jednej LED diódy, čo sa používa na upozornenie hráča alebo zobrazenie stavu hry. Dá sa to považovať za „zapnutie svetla určitej farby na krátku dobu a následné vypnutie“ a je to kľúčová funkcia pre vizuálnu spätnú väzbu v hre.

Prvý riadok: allLEDOff(); Zabezpečuje, aby boli všetky LED diódy vypnuté pred začatím blikania, aby sa zabránilo tomu, že zostatkový stav z predchádzajúceho blikania ovplyvní aktuálne zobrazenie, a aby bolo blikanie jasne rozoznateľné.

Nasledujúce tri príkazy if určujú, ktorá LED dióda svieti, na základe vstupného parametra num:

```
98    if (num == 1) digitalWrite(LED_RED, HIGH);
99    if (num == 2) digitalWrite(LED_YELLOW, HIGH);
100   if (num == 3) digitalWrite(LED_GREEN, HIGH);
```

- Ak je num rovné 1 → Zapni červené svetlo
- Keď num je rovné 2 → Zapni žlté svetlo
- Keď je num rovné 3 → Zapni zelené svetlo

V tomto prípade je použitie digitalWrite(..., HIGH) štandardnou metódou v Arduine na zapnutie a rozsvietenie LED.

Potom volanie delay(flashDelay); pozastaví vykonávanie programu na určitú dobu, ktorú určuje flashDelay a ktorá je zvyčajne nastavená na niekoľko sto milisekúnd. To umožňuje hráčom jasne pozorovať efekt blikania LED. (Tu je to 400 ms)

Na záver opäť vyvolajte funkciu allLEDOff(), aby ste LED diódy vypli a dokončili tak celý cyklus blikania.

Funkcia successLED

```
void successLED() { digitalWrite(LED_RED,
    HIGH); digitalWrite(LED_YELLOW, HIGH);
    digitalWrite(LED_GREEN, HIGH);
    delay(1000);
    allLEDOff();
}
```

Použitím digitalWrite(..., HIGH) na súčasné zapnutie červenej, žltej a zelenej LED diódy sa dosiahne výrazný blikajúci efekt, ktorý signalizuje „úspech“. Ide o najintuitívnejšiu pozitívnu spätnú väzbu v hre, ktorá hráčovi dáva vedieť, že jeho operácia je správna.
Program sa pozastaví na 1000 milisekúnd (1 sekundu), aby sa zabezpečilo, že tri LED diódy zostanú svietiť určitý čas, čo umožní hráčovi plne si všimnúť tento „signál úspechu“ a nenechať

nepremeškal v okamihu.
Na vypnutie všetkých troch LED diód sa volá skôr definovaná funkcia allLEDOff(), čím sa pripraví systém na ďalšiu hru alebo ďalšie kolo blikania.

Funkcia failLED

```
void failLED() {
   for (int i = 0; i < 10; i++) {
      allLEDOn(); delay(100);
      allLEDOff(); delay(100);
   }
}
```

Vo vnútri funkcie sa používa cyklus for: spustí sa 10-krát. Pri každom spustení sa všetky LED diódy raz rozsvietia a potom raz zhasnú, čím sa vytvorí efekt nepretržitého blikania, ktorý simuluje dojem „alarmu“ alebo „chybového hlásenia“.

- allLEDOn(): Zapne červené, žlté a zelené LED diódy, čo umožňuje hráčom okamžite zistiť chybu.
- delay(100): Udrží svetlá zapnuté po dobu 100 milisekúnd.
- allLEDOff(): Vypne všetky LED diódy.
- delay(100): Udrží ich vypnuté po dobu 100 milisekúnd, čím sa vytvorí interval blikania.

Funkcia allLEDOn

```
void allLEDOn() { digitalWrite(LED_RED,
   HIGH); digitalWrite(LED_YELLOW,
   HIGH); digitalWrite(LED_GREEN, HIGH);
}
```

Volaním funkcie allLEDOn() môže program rozsvietiť všetky LED diódy naraz bez toho, aby musel zakaždým opakovať tri príkazy digitalWrite. To zjednodušuje kód a umožňuje opätovné použitie logiky, čím sa dosiahne efekt stručného kódu a opätovného použitia logiky. Je to obzvlášť vhodné na volanie na viacerých miestach, kde je potrebná indikácia „plného zapnutia“, napríklad v funkciách successLED() alebo failLED().

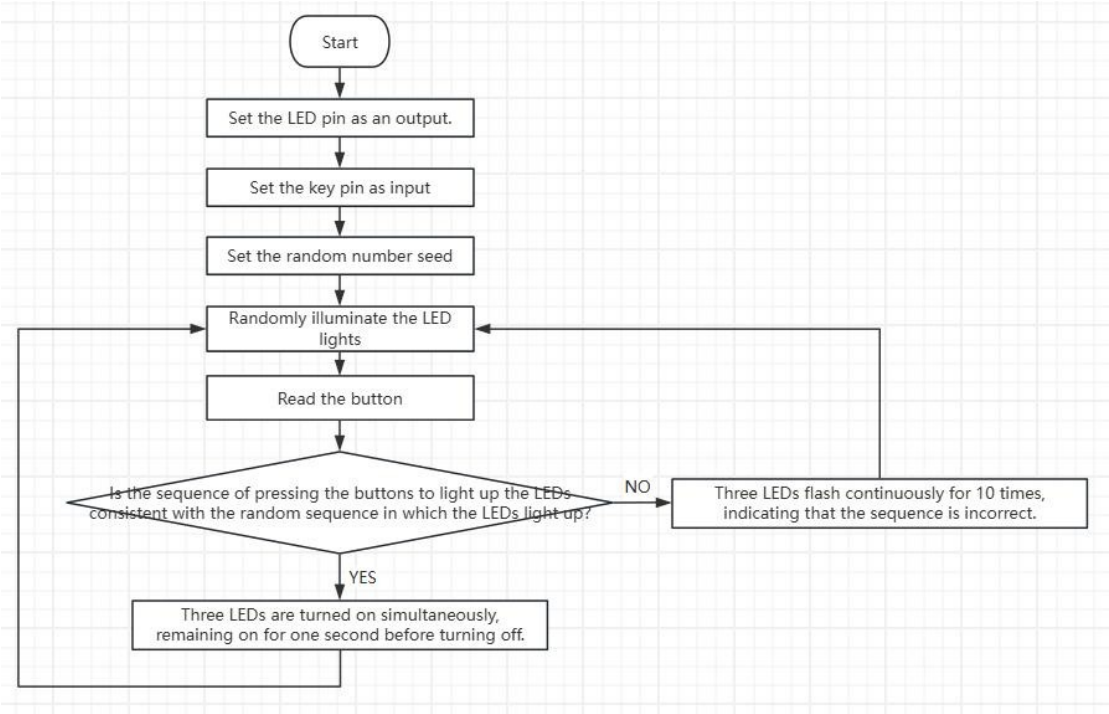
Funkcia allLEDOff

```
void allLEDOff() { digitalWrite(LED_RED,
   LOW); digitalWrite(LED_YELLOW, LOW);
   digitalWrite(LED_GREEN, LOW);
}
```

Volaním funkcie allLEDOff() môže program vypnúť všetky LED diódy naraz, bez toho, aby bolo potrebné opakovane zapisovať tri príkazy digitalWrite. To uľahčuje vynulovanie alebo vymazanie stavu výzvy v hernej logike, čím sa zvyšuje čitateľnosť a opätovná použiteľnosť kódu. Napríklad vo funkciách ako flashSingleLED(), successLED() alebo failLED() sa táto funkcia používa

na ovládanie zapínania a vypínania osvetlenia.

Celkový diagram toku logiky kódu



Kroky na nahratie programu

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Lekcia 19 – Hádanka s posuvným rezistorom

Úvod

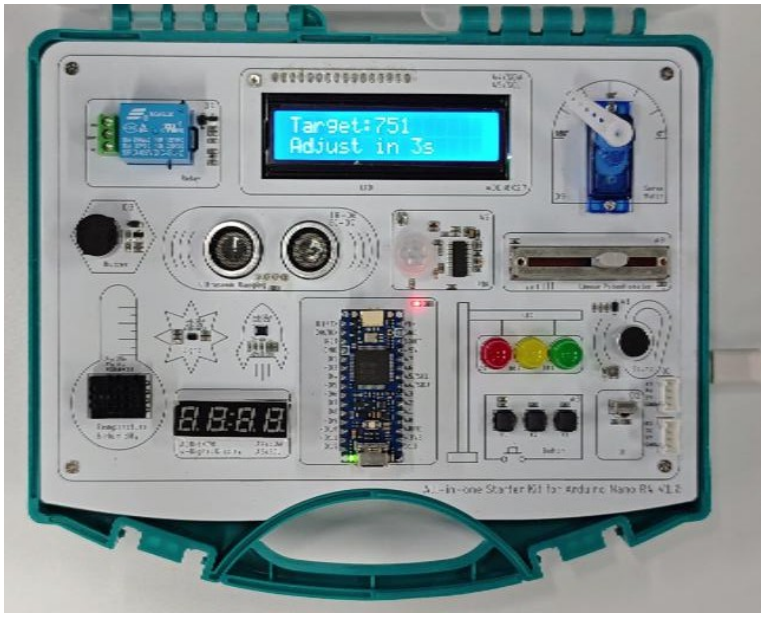
V tejto lekcii sa prostredníctvom zaujímavej minihry naučíme komplexné využitie potenciometrov, displejov LCD1602 a LED indikátorov.
Pravidlá hry: Displej LCD1602 náhodne zobrazí cieľovú hodnotu v rozmedzí od 0 do 1023. Hráč musí do 3 sekúnd nastaviť potenciometer tak, aby sa hodnota odporu čo najviac priblížila cieľovej hodnote. Po uplynutí času systém odčíta hodnotu odporu a vykoná porovnanie. Zelené svetlo signalizuje úspech, červené svetlo neúspech.
V tejto lekcii si osvojíte základnú logiku čítania analógových signálov, programovanie náhodných čísel a spoluprácu viacerých modulov.

Ciele výučby

1. Upevniť si metódu čítania analógových hodnôt pomocou funkcie analogRead().
2. Naučte sa používať funkciu randomSeed na generovanie počiatočných hodnôt pre náhodné čísla a zistite, ako generovať náhodné čísla.
3. Zobrazte dynamické údaje na LCD1602.
4. V tejto časti dokončite výrobu hry na hádanie čísel s posuvným potenciometrom.

Náhľad výsledku

Po nahratí programu sa na displeji LCD1602 zobrazí nasledujúci proces:
1. LCD displej zobrazí náhodnú cieľovú hodnotu (0 až 1023).
2. Zobrazí sa výzva na nastavenie potenciometra do 3 sekúnd.

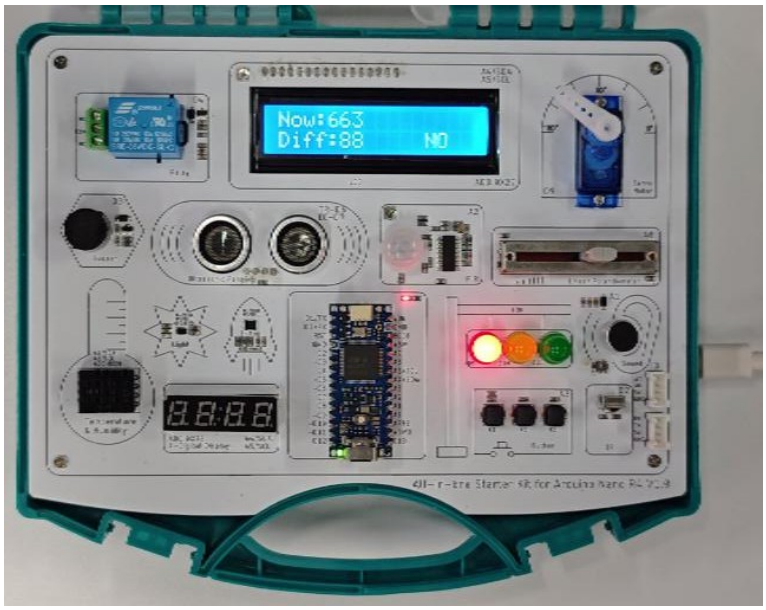


3. Po uplynutí tohto času:

- LCD displej zobrazí aktuálnu hodnotu posuvného reostatu.
- Zobrazí sa tiež rozdiel od cieľovej hodnoty (Diff).

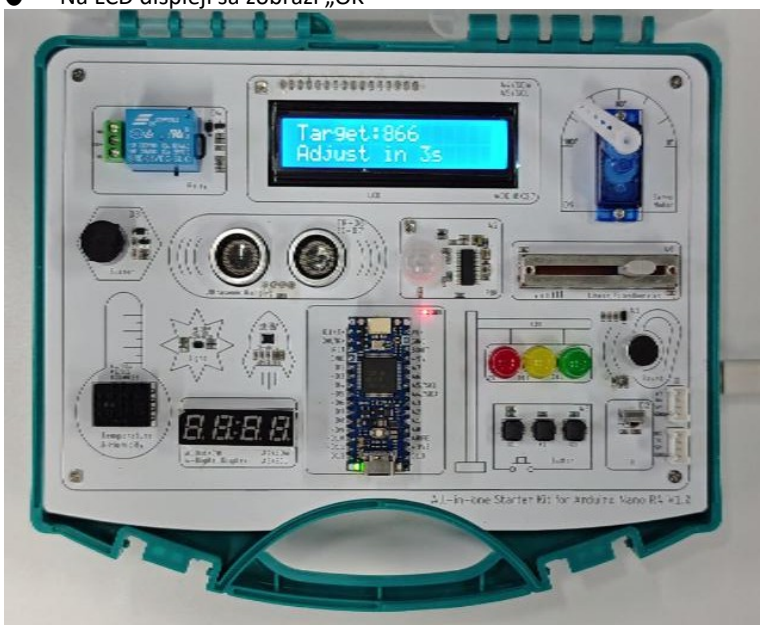
4. Ak je rozdiel veľký:

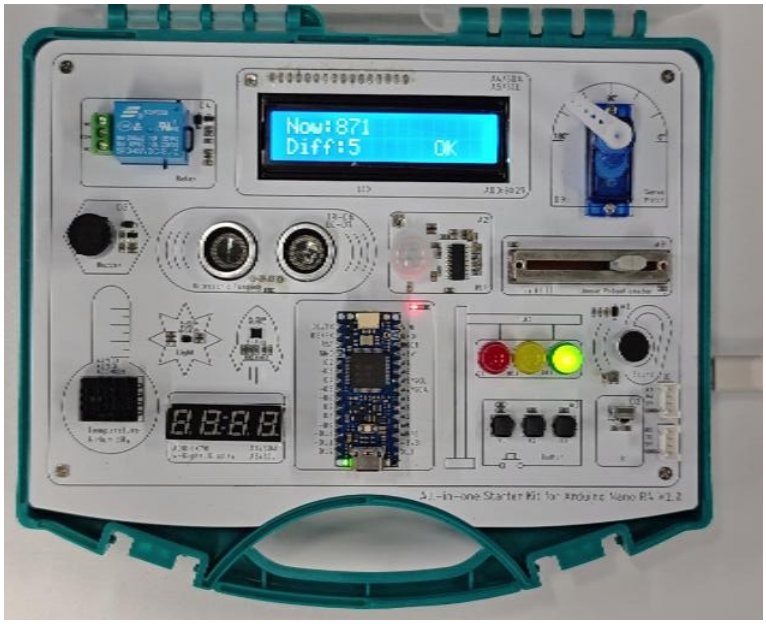
- Svieti červené svetlo.
- Na LCD displeji sa zobrazí „NO“.



5. Ak je rozdiel dostatočne malý:

- Svieti zelené svetlo
- Na LCD displeji sa zobrazí „OK“

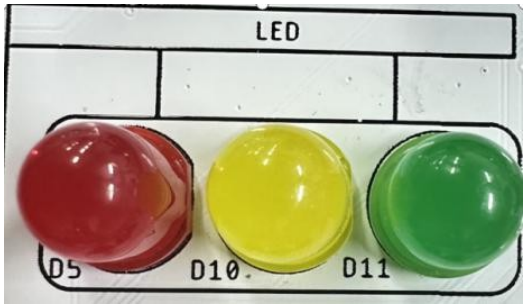




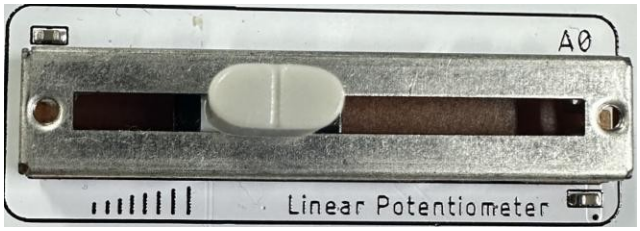
6. Systém zobrazí aktuálny výsledok a automaticky prejde do ďalšieho kola hry.

Hardvér použitý v tejto lekcii

Trojfarebné LED diódy



Lineárny potenciometer



Displej LCD1602

Puzdro na hru s odhadovaním čísel s posuvným odporom

Pred vysvetlením kódu si ho môžeme stiahnuť. Odkaz na stiahnutie celého kódu je:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/19_Sliding_Resistor_Guessing_Game
Otvorte priečinok „19_Sliding_Resistor_Guessing_Game“ v prostredí Arduino IDE a potom otvorte súbor s kódom „19_Sliding_Resistor_Guessing_Game.ino“ na adrese .

Vysvetlenie kľúčového kódu

Po otvorení kódu uvidíte kód pre túto lekciu:

19_Sliding_Resistor_Guessing_Game.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

/* ========== LCD Parameters ========== */
#define COLUMNS 16
#define ROWS    2

LiquidCrystal_I2C lcd(
  PCF8574_ADDR_A21_A11_A01,
  4, 5, 6, 16, 11, 12, 13, 14,
  POSITIVE
);

/* ========== Hardware Pins ========== */
#define POT_PIN A0

#define LED_RED     5
#define LED_GREEN   11

/* ========== Game Variables ========== */
int targetValue = 0;
int currentValue = 0;
int diffValue = 0;
int score = 0;

/* ========== Setup ========== */
void setup()
{
  Serial.begin(115200);

  lcd.begin(COLUMNS, ROWS, LCD_5x8DOTS);
  lcd.clear();
  lcd.backlight();

  randomSeed(analogRead(A1));

  // LED initialization
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);

  digitalWrite(LED_RED, LOW);
  digitalWrite(LED_GREEN, LOW);

```

Úvod do knižničných súborov

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Dva riadky kódu #include <Wire.h> a #include <LiquidCrystal_I2C.h> slúžia na poskytovanie komunikačných schopností I2C a ovládacieho rozhrania pre I2C obrazovku z tekutých kryštálov pre Arduino: Wire.h je zodpovedný za základnú komunikáciu zbernice I2C, umožňujúcu prenos dát medzi hlavným a podriadeným zariadením;

LiquidCrystal_I2C.h potom na tomto základe poskytuje pokročilú enkapsuláciu pre ovládanie LCD, čo nám umožňuje priamo používať intuitívne funkcie, ako sú lcd.begin(), lcd.print() na zobrazenie textu, bez toho, aby sme sa museli zaoberať konkrétnymi časovými intervalmi komunikácie a hardvérovými detailmi. Je to

základom pre normálnu prevádzku celého LCD.

Definícia parametrov displeja LCD1602 a inicializácia rozhrania I2C

```
#define COLUMNS 16          // Počet stĺpcov LCD
#define ROWS          2     // Počet riadkov LCD
LiquidCrystal_I2C lcd(
   PCF8574_ADDR_A21_A11_A01,        // I2C adresa PCF8574 4, 5,
   6, 16,                           // RS, RW, EN, podsvietenie
   11, 12, 13, 14,                  // D4, D5, D6, D7
   KLADNÁ                           // Polarita podsvietenia
);
```

Najskôr pomocou príkazov #define COLUMNS 16 a #define ROWS 2 jasne oznámime programu, že aktuálny displej je znakový LCD displej s rozmermi 16 stĺpcov × 2 riadky. Takto môže knižničná funkcia pri neskoršom použití lcd.begin(COLUMNS, ROWS) správne nakonfigurovať vyrovnávaciu pamäť displeja a systém kurzora. Potom sa vytvorí objekt LiquidCrystal_I2C lcd(...). Tu neovládame LCD priamo pomocou paralelných portov, ale nepriamo ovládame všetky piny LCD cez rozširujúci čip PCF8574 I2C.

PCF8574_ADDR_A21_A11_A01 sa používa na špecifikovanie adresy tohto rozširujúceho čipu na zbernici I2C. Nasledujúce RS, RW, EN, podsvietenie, D4~D7 predstavujú zodpovedajúce vzťahy medzi výstupnými pinmi PCF8574 a riadiacimi a dátovými linkami LCD. Posledný signál POSITIVE označuje, že podsvietenie je na vysokej úrovni a svieti, a táto metóda zápisu umožňuje Arduinu stabilne ovládať LCD displej len pomocou dvoch vodičov SDA/SCL, čo je veľmi klasické a veľmi úsporné riešenie zobrazenia v embedded projektoch.

Definícia pinov

```
#define POT_PIN A0
#define LED_RED          5
#define LED_GREEN        11
```

POT_PIN je definovaný ako A0, čo znamená, že potenciometer (premenný rezistor) je pripojený k analógovému vstupnému pinu A0 dosky Arduino, ktorý slúži na snímanie analógových napäťových hodnôt v rozsahu od 0 do 1023; LED_RED a LED_GREEN sú definované ako digitálne piny 5 a 11, ktoré zodpovedajú červenej a zelenej LED dióde.

Týmto spôsobom je možné v nasledujúcom kóde len pomocou POT_PIN, LED_RED a LED_GREEN intuitívne pochopiť funkcie každého pinu.

Definujte premenné

```
int targetValue = 0; int
currentValue = 0; int
diffValue = 0;
int score = 0;
```

Tieto štyri celočíselné premenné slúžia na ukladanie kľúčových stavov údajov počas hry: targetValue predstavuje „cieľovú hodnotu“ náhodne generovanú systémom, ktorá je štandardom, ku ktorému sa hráči musia v tomto kole priblížiť;

currentValue sa používa na ukladanie aktuálnej skutočnej analógovej hodnoty načítanej z potenciometra

(POT_PIN), ktorá odráža výsledok činnosti hráča;
diffValue je rozdiel medzi aktuálnou hodnotou a cieľovou hodnotou (zvyčajne sa berie absolútna hodnota), ktorý sa používa na určenie, či je hráč dostatočne blízko k cieľu;
premenná score slúži na zaznamenávanie počtu úspešných zásahov hráča.

Sekcia Setup

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

  lcd.begin(COLUMNS, ROWS, LCD_5x8DOTS);
  lcd.clear();
  lcd.backlight();

  randomSeed(analogRead(A1));

  // Inicializácia LED
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);

  digitalWrite(LED_RED, LOW);
  digitalWrite(LED_GREEN, LOW);

  lcd.setCursor(0, 0);
  lcd.print("Hra na hádanie
  čísla"); lcd.setCursor(0, 1);
  lcd.print("Pripravte sa...");
  delay(1500);
  lcd.clear();
}
```

Táto funkcia setup() je zodpovedná za inicializáciu celej hry, ktorú možno považovať za „prípravnú fázu po zapnutí systému“.
Najskôr otvorí sériovú komunikáciu pomocou Serial.begin(115200), aby uľahčil ladenie a sledovanie výstupu údajov;
Potom inicializuje LCD displej pomocou lcd.begin(), nastaví počet stĺpcov, riadkov a režim písma pre zobrazenie, vymaže obrazovku a zapne podsvietenie pomocou lcd.clear() a lcd.backlight(), aby zabezpečil normálny stav zobrazenia.

Funkcia riadku kódu „randomSeed(analogRead(A1))“ je nastaviť náhodné semeno pre generátor náhodných čísel, čím sa zabezpečí, že cieľové hodnoty generované v hre majú skutočnú náhodnosť.

Ďalej sa vykoná konfigurácia režimu pre vývody červených a zelených LED diód, ktoré sú spočiatku

nastavené na nízku úroveň, aby sa zabezpečilo, že všetky kontrolky sú pri spustení systému vypnuté.

Nakoniec sa na LCD displeji zobrazí text spúšťacej výzvy „Pot Guess Game“ a „Get Ready...“, čím sa hráčom poskytne čas na prípravu. Po oneskorení 1500 milisekúnd (1,5 sekundy) sa obrazovka vymaže, aby sa pripravilo prostredie pre oficiálny herný cyklus.

smyčka

```
void loop()
{
   // Vypnúť LED diódy na začiatku každého kola
   digitalWrite(LED_RED, LOW);
   digitalWrite(LED_GREEN, LOW);

   /* ===== Vygenerovanie cieľovej hodnoty ===== */
   targetValue = random(0, 1024);

   lcd.clear(); lcd.setCursor(0,
   0); lcd.print("Cieľ:");
   lcd.print(targetValue);

   lcd.setCursor(0, 1);
   lcd.print("Nastavte do 3
   sekúnd");

   delay(3000);

   /* ===== Načítanie hodnoty potenciometra =====
   */ currentValue = analogRead(POT_PIN); diffValue
   = abs(currentValue - targetValue);

   lcd.clear(); lcd.setCursor(0,
   0); lcd.print("Teraz:");
   lcd.print(currentValue);

   lcd.setCursor(0, 1);
   lcd.print("Rozdiel:");
   lcd.print(diffValue);

   /* ===== Úspech / Neúspech ===== */
   if (diffValue <= 10) {
      skóre++;
      digitalWrite(LED_GREEN, HIGH);          // Úspech → zapnúť zelenú LED
      lcd.setCursor(12, 1);
```

```
    lcd.print("OK");
  } else {
    digitalWrite(LED_RED, HIGH);          // Neúspech → zapnúť červenú
    lcd.setCursor(12, 1);                 LED
    lcd.print("NO");
  }


  delay(2000);


  /* ===== Zobrazenie skóre =====
  */
  lcd.clear(); lcd.setCursor(0,
  0); lcd.print("Skóre:");
  lcd.print(skóre);
  lcd.setCursor(0, 1);
  lcd.print("Ďalšie kolo");


  delay(1500);
}
```

Táto funkcia loop() nie je len „hlavným procesom“ hry, ale aj klasickým príkladom, ktorý spája náhodné čísla, simulované vstupy, matematické operácie a interakciu človeka s počítačom.

Hneď ako program vstúpi do funkcie loop(), najprv vypne červenú aj zelenú LED diódu pomocou digitalWrite(LED_RED, LOW); a digitalWrite(LED_GREEN, LOW);

Toto je veľmi dôležitý krok „resetovania stavu“, ktorého cieľom je zabrániť zotrvávaniu stavu úspechu alebo neúspechu z predchádzajúceho kola a zabezpečiť, aby každé kolo začínalo v čistom počiatočnom stave.

```
54  void loop()
55  {
56    // Turn off LEDs at the start of each round
57    digitalWrite(LED_RED, LOW);
58    digitalWrite(LED_GREEN, LOW);
59
```

Ďalej nasleduje generovanie cieľových hodnôt. targetValue = random(0, 1024); Používa sa funkcia random() zabudovaná v Arduine, ktorá generuje pseudonáhodné celé číslo v rozsahu od 0 do 1023 (okrem 1024). Tento rozsah je presne rovnaký ako rozsah vrátených hodnôt funkcie analogRead(), pretože analógový vstup Arduina má 10-bitovú presnosť, čo zodpovedá hodnotám od 0 do 1023.

Tento návrh je veľmi dômyselný: cieľová hodnota a hodnota potenciometra sú v rovnakých „jednotkách“, takže neskoršie porovnanie dáva zmysel. Následne sa cieľová hodnota zobrazí na LCD displeji a zároveň sa hráčovi zobrazí výzva, že má 3 sekundy na otočenie potenciometra. Funkcia delay(3000) tu zodpovedá „operačnému oknu“, ktoré je poskytnuté hráčovi.

```
60    /* ===== Generate target value ===== */
61    targetValue = random(0, 1024);
62
63    lcd.clear();
64    lcd.setCursor(0, 0);
65    lcd.print("Target:");
66    lcd.print(targetValue);
67
68    lcd.setCursor(0, 1);
69    lcd.print("Adjust in 3s");
70
71    delay(3000);
```

Po 3 sekundách program prejde do fázy snímania vstupu od hráča. currentValue = analogRead(POT_PIN);
Táto funkcia prečíta analógové napätie na pine A0 a premení ho na digitálnu hodnotu v rozmedzí od 0 do 1023. Táto digitálna hodnota predstavuje aktuálnu polohu, do ktorej je potenciometer otočený.

```
73    /* ===== Read potentiometer value ===== */
74    currentValue = analogRead(POT_PIN);
```

Potom je veľmi dôležitý tento riadok: diffValue = abs(currentValue - targetValue);
Tu sa používa funkcia abs(), ktorá slúži na výpočet absolútnej hodnoty. Prečo je potrebné používať absolútnu hodnotu?
Hodnota hráča môže byť totiž väčšia alebo menšia ako cieľová hodnota. Ak by sme ich jednoducho od seba odpočítali, výsledok by mohol byť záporný, pričom nás skutočne zaujíma skôr „o koľko sa líšia“, ako „smer tohto rozdielu“. Funkcia abs() zaručuje, že bez ohľadu na to, či je hodnota vyššia alebo nižšia, konečný výsledok bude vždy kladná hodnota chyby. To je nevyhnutné pri určovaní, „či je hodnota dostatočne blízko cieľu“.

```
75    diffValue = abs(currentValue - targetValue);
```

Následne LCD displej zobrazí aktuálnu hodnotu potenciometra (Now) a rozdiel od cieľovej hodnoty (Diff), čo hráčovi umožní jasne vidieť, kde „zaostáva“.

```
77    lcd.clear();
78    lcd.setCursor(0, 0);
79    lcd.print("Now:");
80    lcd.print(currentValue);
81
82    lcd.setCursor(0, 1);
83    lcd.print("Diff:");
84    lcd.print(diffValue);
85
```

Potom nasleduje logika na určenie výsledku:
Ak (diffValue <= 10), znamená to, že pokiaľ je chyba v rozmedzí 10, považuje sa, že hráč vykonal dostatočne presnú úpravu. Táto hodnota 10 v skutočnosti predstavuje kľúčový parameter „náročnosti hry“. Čím je hodnota menšia, tým je hra náročnejšia; čím je hodnota väčšia, tým je ľahšie uspieť.
V prípade úspechu program poskytne tri typy spätnej väzby: skóre (score++) sa zvýši,

rozsvieti sa zelená LED dióda a na LCD displeji sa zobrazí „OK“; v prípade neúspechu svieti červená LED dióda a zobrazí sa „NO“. Vďaka dvojitej vizuálnej a číselnej spätnej väzbe sa posilňuje vnímanie hráča.

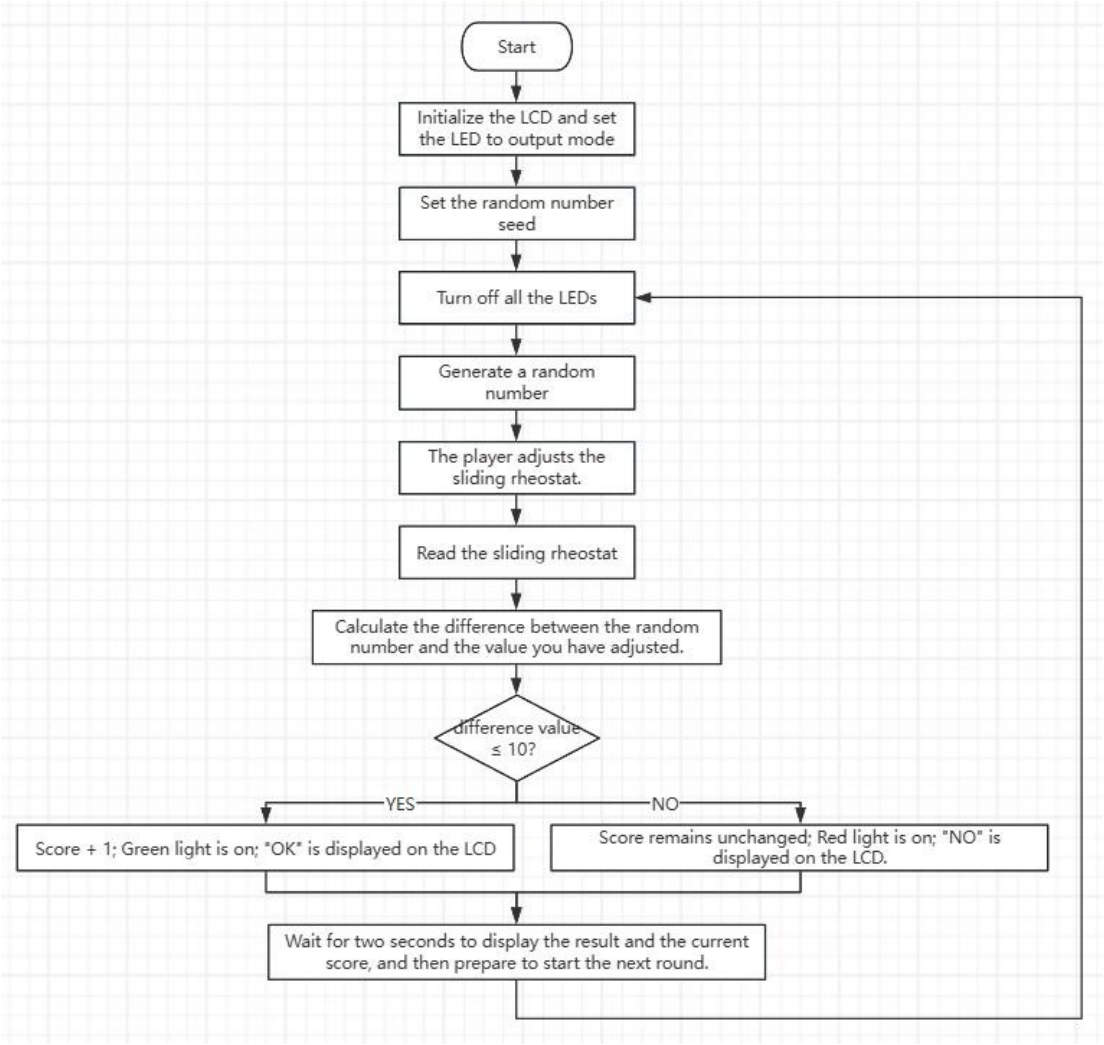
```
/* ===== Success / Failure ===== */
if (diffValue <= 10) {
  score++;
  digitalWrite(LED_GREEN, HIGH);   // Success → turn on green LED
  lcd.setCursor(12, 1);
  lcd.print("OK");
} else {
  digitalWrite(LED_RED, HIGH);     // Failure → turn on red LED
  lcd.setCursor(12, 1);
  lcd.print("NO");
}

```

Záverečnou časťou je zhrnutie kola a prechod. Program sa na 2 sekundy pozastaví, aby hráči mohli jasne vidieť výsledok, potom vymaže obrazovku a zobrazí aktuálne kumulatívne skóre, ako aj výzvu „Ďalšie kolo“. Po 1,5-sekundovom oneskorení automaticky prejde do ďalšieho kola. Tento rytmický dizajn zabezpečuje, že hra nie je príliš rýchla, aby bola pre hráčov príliš náročná, ani príliš pomalá, aby pôsobila ťažkopádne.

```
  delay(2000);

  /* ===== Display score ===== */
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Score:");
  lcd.print(score);
  lcd.setCursor(0, 1);
  lcd.print("Next round");

  delay(1500);
}
```

Celkovo vzaté, táto funkcia „loop()“ nie je len obyčajnou slučkou; skôr v sebe plne zahŕňa: použitie náhodných čísel, zber analógových signálov, matematické operácie s absolútnymi hodnotami, podmienené rozhodovanie a spätnú väzbu v rámci interakcie človek-počítač. Je mimoriadne vhodná ako klasický príklad pre začiatočníkov, aby pochopili, „ako funguje kompletný projekt s Arduinom“.

Celkový diagram logiky kódu



Kroky na nahratie programu

Tento projekt vyžaduje dodatočné externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.

Lekcia 20 – Hra na dekódovanie morseovky

Úvod

V tejto hodine sa budeme venovať veľmi zaujímavému a náročnému projektu – hre na dešifrovanie morseovky.
Morseovka je spôsob prenosu informácií pomocou dlhých a krátkych signálov. V minulosti sa široko používala v telegrafickej komunikácii.
V tomto projekte budeme používať:

- Tlačidlá (Button) na zadávanie krátkych a dlhých stlačení
- LED indikátory na zobrazenie rôznych prevádzkových stavov
- Displej LCD1602 na zobrazenie dekódovaných anglických znakov v reálnom čase

Po absolvovaní tohto kurzu už nebudete len „stláčať tlačidlá a rozsvietiť LED diódy“, ale budete môcť pomocou tlačidiel skutočne „písať“ a na LCD displeji uvidíte anglické slová, ktoré zadáte, čím dokončíte kompletnú minihru zameranú na interakciu človeka s počítačom.

Ciele výučby

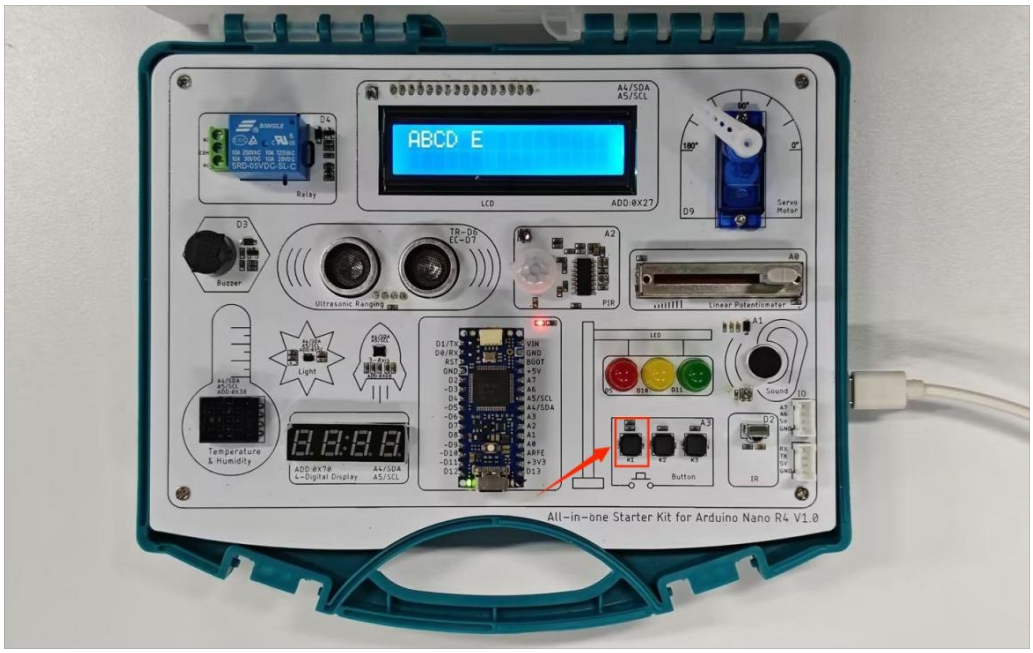
1. Porozumieť základným pravidlám kódovania morseovky
2. Ovládať metódy implementácie krátkeho stlačenia / dlhého stlačenia v programe
3. Upevniť používanie funkcie analogRead na čítanie viacerých tlačidiel
4. Upevniť logiku zobrazenia reťazcov a obnovovania na LCD1602
5. Porozumieť celému priebehu programu „vstup → spracovanie → zobrazenie“

Náhľad výsledku

Keď sa program úspešne nahraje, uvidíte nasledujúci efekt:

➢ Stlačte prvé tlačidlo

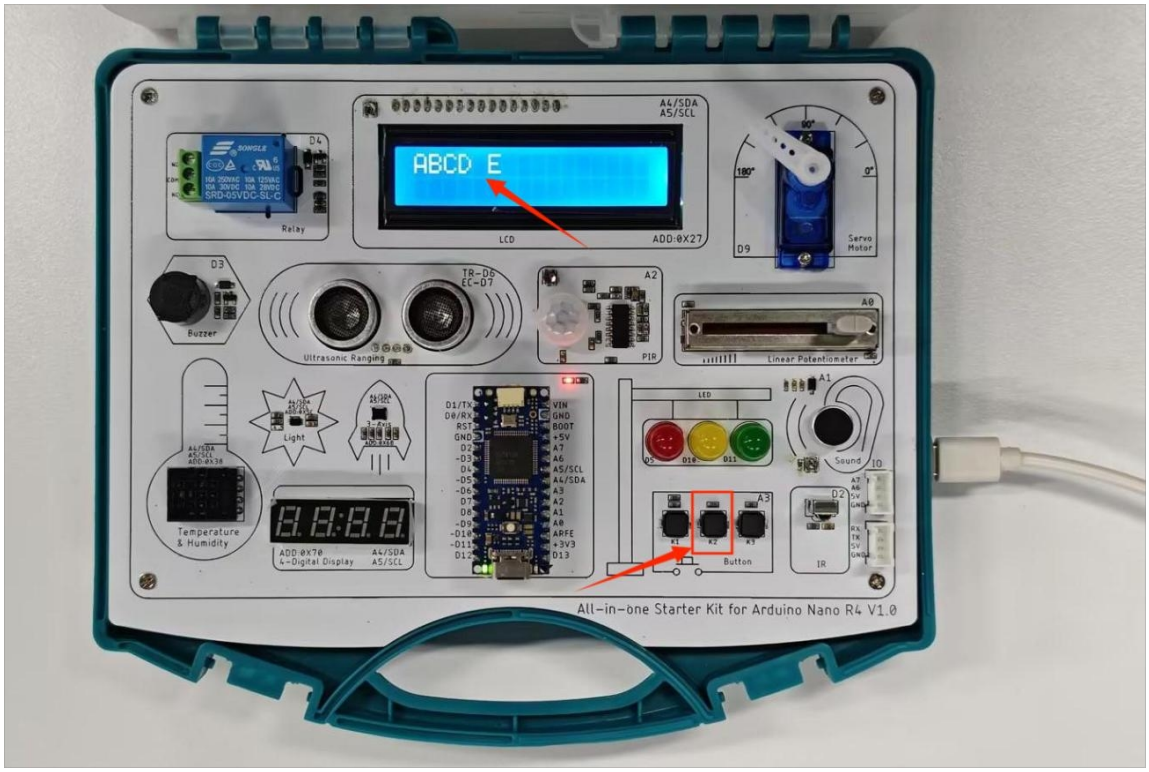
Krátke stlačenie → Zadajte „.“ (V tomto prípade sa rozsvieti žlté svetlo, po ktorom nasleduje zelené svetlo, ktoré slúži ako výzva)
Dlhé stlačenie → Zadajte znak „-“ (V tomto momente sa rozsvieti žlté svetlo, po ktorom nasleduje červené svetlo, ktoré slúži ako výzva)
Po uvoľnení tlačidla na chvíľu program automaticky dekóduje morseovku na písmeno a zobrazí ho na LCD displeji.



➢ Stlačte druhé tlačidlo
Zadajte medzeru

Toto sa používa na oddelenie slov

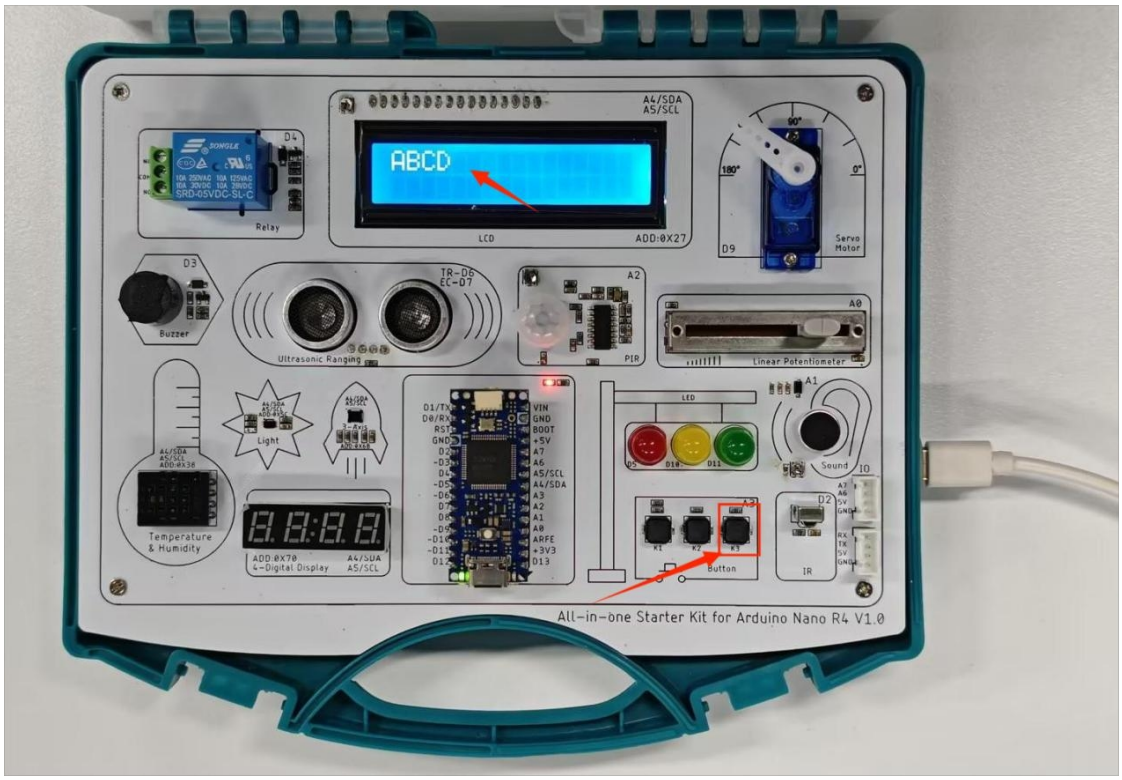
(ako výzva svieti zelené svetlo)



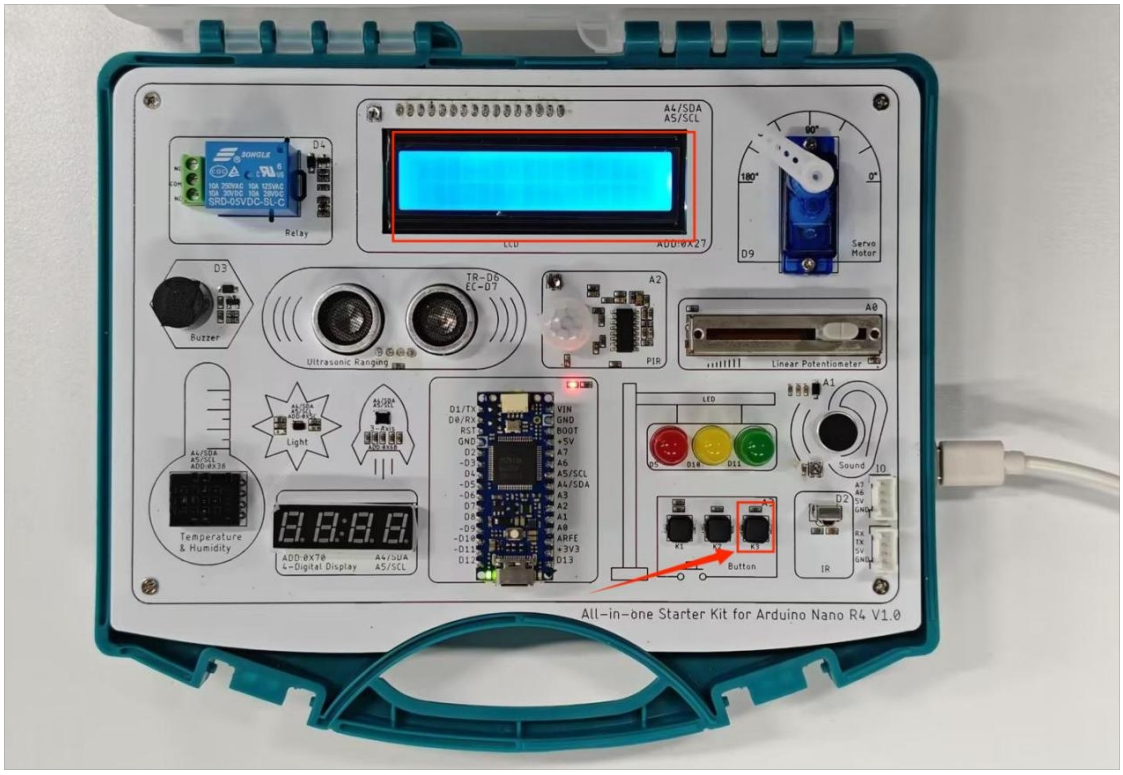
➢ Použite tretie tlačidlo

Krátke stlačenie → Vymazať posledný znak (ako pripomienka svieti červené svetlo)

(Tu je v porovnaní s predchádzajúcim popisom vidieť, že písmeno E bolo vymazané)

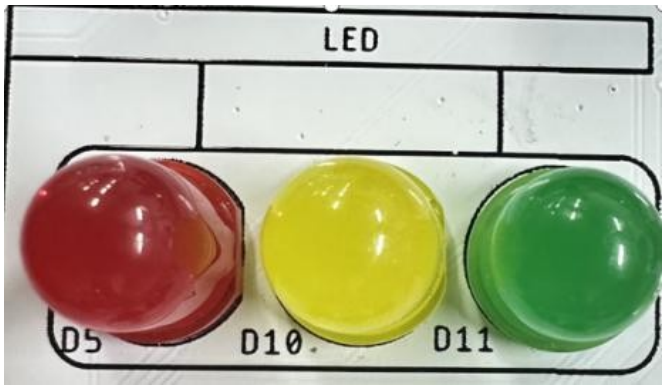


Dlhé stlačenie → Vymazať celú obrazovku
(Blikajúce červené svetlo slúži ako pripomienka)

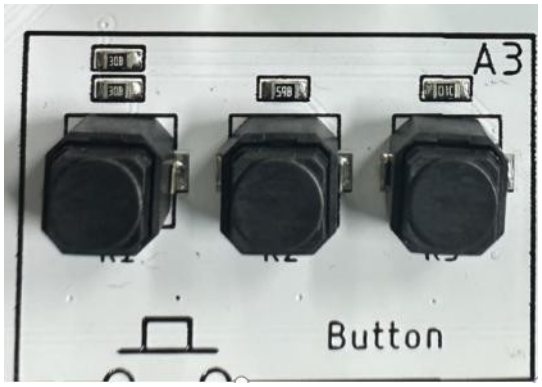


Hardvér použitý v tejto lekcii

Trojfarebné LED svetlá



Tri tlačidlá



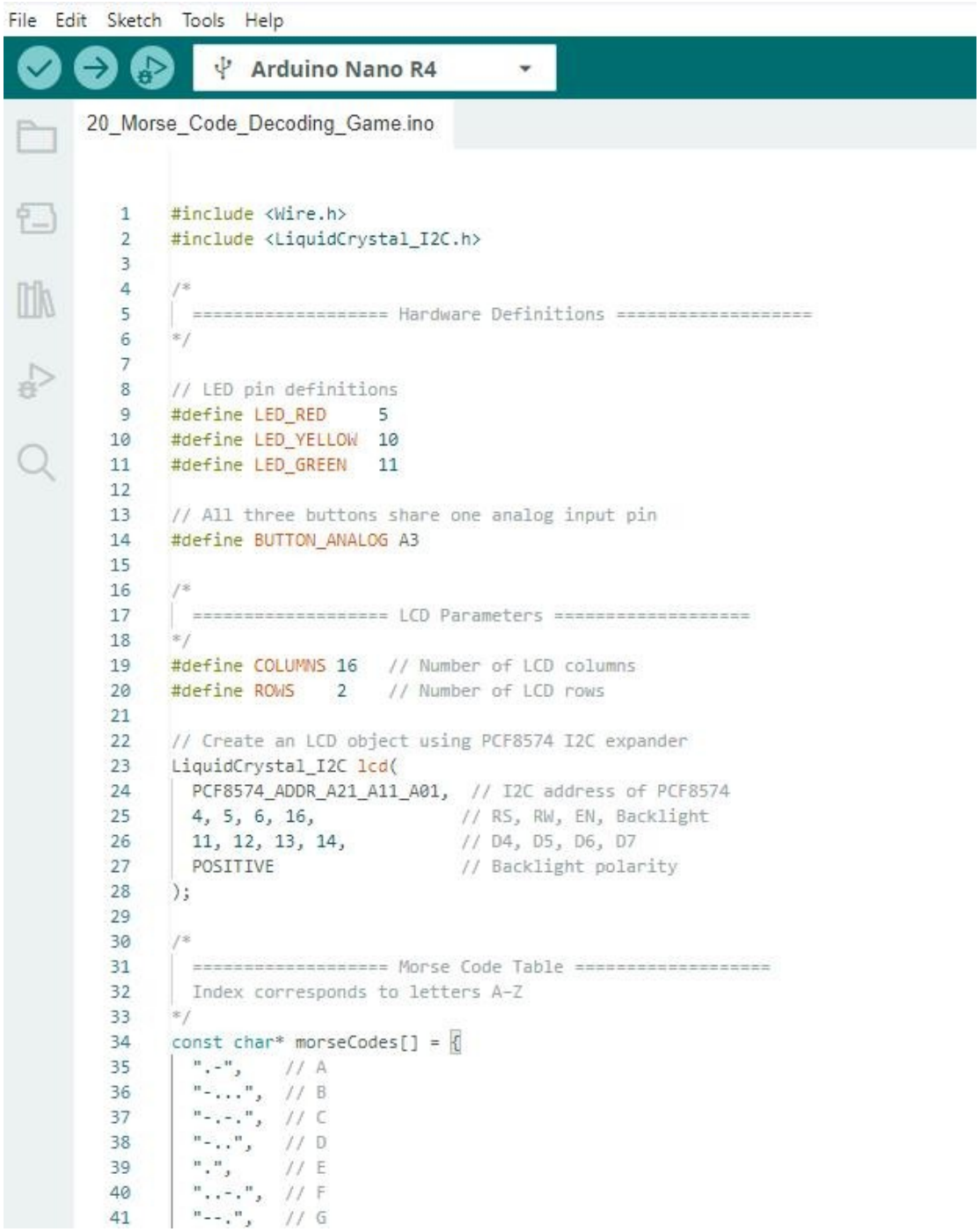
Displej LCD1602

Prípad hry s morseovkou

Pred vysvetlením kódu si ho môžeme stiahnuť. Odkaz na stiahnutie celého kódu je:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Nano-R4/tree/master/lesson_code/20_Morse_Code_Decoding_Game
Otvorte v prostredí Arduino IDE priečinok „20_Morse_Code_Decoding_Game“ a potom otvorte súbor s kódom „20_Morse_Code_Decoding_Game.ino“

Vysvetlenie kódov

Po otvorení kódu uvidíte kód pre túto lekciu:



Úvod do súborov knižnice

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Wire.h je zodpovedný za aktiváciu ovládača zbernice I2C Arduina, čo umožňuje mikrokontroléru komunikovať s externými čipmi prostredníctvom dvoch liniek SDA a SCL;

Zatiaľ čo LiquidCrystal_I2C.h je vopred zabalená knižnica na ovládanie LCD postavená na knižnici Wire. Skryje všetky zložité načasovanie I2C a základné ovládanie signálu, čo nám umožňuje priamo používať intuitívne funkcie, ako sú lcd.begin(), lcd.setCursor() a lcd.print(), na ovládanie obsahu LCD displeja.

V tomto projekte komunikuje LCD s Arduinom prostredníctvom rozširujúceho čipu PCF8574 I2C.

Preto sú obe tieto knižnice nevyhnutné – prvá zodpovedá za „tok dát“ a druhá za „to, čo sa zobrazuje na obrazovke“, čím spoločne zabezpečujú, že výsledok dekódovania morseovky sa môže stabilne a jasne zobrazovať na LCD obrazovke.

Definícia pinov

```
#define LED_RED          5
#define LED_YELLOW       10
#define LED_GREEN        11
#define TLAČIDLO_ANALOG A3
```

„LED_RED“, „LED_YELLOW“, „LED_GREEN“ zodpovedajú digitálnym pinom (5, 10, 11) pripojeným k červenej, žltej a zelenej LED dióde. Použitím tejto metódy „definície makra“ stačí v nasledujúcom kóde namiesto konkrétnych čísel napísať len názvy funkcií LED diód, čo výrazne zlepšuje čitateľnosť a udržiavateľnosť programu;

Zatiaľ čo „BUTTON_ANALOG A3“ označuje, že v tomto projekte tri rôzne funkcie tlačidiel nezaberajú každý samostatný digitálny pin, ale namiesto toho zdieľajú analógový vstupný pin A3 prostredníctvom odporového deliča. Arduino číta rôzne hodnoty napätia prostredníctvom „analogRead(A3)“ a potom na základe rozsahu napätia určí, ktoré tlačidlo je stlačené.

Definícia parametrov displeja LCD1602 a inicializácia rozhrania I2C

```
#define COLUMNS 16         // Počet stĺpcov LCD
#define ROWS        2      // Počet riadkov LCD
LiquidCrystal_I2C lcd(
   PCF8574_ADDR_A21_A11_A01,        // I2C adresa PCF8574 4, 5,
   6, 16,                           // RS, RW, EN, podsvietenie
   11, 12, 13, 14,                  // D4, D5, D6, D7
   KLADNÁ                           // Polarita podsvietenia
);
```

Najprv pomocou #define COLUMNS 16 a #define ROWS 2 jasne informujeme program, že aktuálny displej je 16-stĺpcový × 2-riadkový znakový LCD displej. Týmto spôsobom, keď neskôr použijeme lcd.begin(COLUMNS, ROWS), knižničná funkcia môže správne nakonfigurovať vyrovnávaciu pamäť displeja a systém kurzora.

Potom sa vytvorí objekt LiquidCrystal_I2C lcd(...). Tu neovládame LCD priamo pomocou paralelných portov, ale nepriamo ovládame všetky piny LCD cez rozširujúci čip PCF8574 I2C.

PCF8574_ADDR_A21_A11_A01 slúži na určenie adresy tohto rozširujúceho čipu na zbernici I2C. Nasledujúce signály RS, RW, EN, podsvietenie a D4~D7 predstavujú zodpovedajúce vzťahy medzi výstupnými pinmi čipu PCF8574 a riadiacimi a dátovými linkami LCD displeja. Posledný signál POSITIVE označuje, že podsvietenie je na vysokej úrovni a svieti, a tento spôsob zápisu umožňuje Arduinu stabilne ovládať LCD displej len pomocou dvoch vodičov SDA/SCL, čo je veľmi klasické a veľmi úsporné riešenie zobrazenia v embedded projektoch.

Tabuľka zodpovedností písmen morseovky k anglickým písmenám

```
const char* morseCodes[] = { ".-
   ",          // A
```

```
    "-...",     // B
    "-.-.",     // C
    "-..",      // D
    ".",        // E
    "..-.",     // F
    "--.",      // G
    "....",     // H
    "..",       // I
    ".---",     // J
    "-.-",      // K
    ".-..",     // L
    "--",       // M
    "-.",       // N
    "---",      // O
    ".--.",     // P
    "--.-",     // Q
    ".-.",      // R
    "...",      // S
    "-",        // T
    "..-",      // U
    "...-",     // V
    ".--",      // W
    "-..-",     // X
    "-.--",     // Y
    "--.."      // Z
};


const char letters[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

const char* morseCodes[] definuje pole ukazovateľov na reťazce, kde každý prvok reťazca ukladá morseovku zodpovedajúcu písmenu (zloženú z bodiek a pomlčiek). Indexy poľa prísne zodpovedajú písmenám A až Z. Napríklad index 0 je morseovka pre A (".-"), index 1 je morseovka pre B ("-..."), a tak ďalej.

Hneď za tým nasleduje konštanta `char letters[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"`, ktorá definuje pole znakov, v ktorom sú samotné písmená uložené v rovnakom poradí. V programe je tak možné pomocou rovnakých indexových hodnôt priradiť zadaný reťazec morseovky k príslušnému anglickému písmenu. Táto „metóda vyhľadávacej tabuľky s indexovým zarovnaním“ je nielen logicky prehľadná, ale vyznačuje sa aj vysokou efektivitou spracovania.

Kľúčové parametre „pravidiel určovania času“

```
const int SHORT_PRESS_THRESHOLD = 500;                  // Krátke stlačenie < 500 ms → bodka
const int CHAR_INTERVAL = 1500;                         // Čas nečinnosti na automatické
dekódovanie znaku const int CLEAR_SCREEN_THRESHOLD = 1000;                      // Dlhé
stlačenie > 1 s → vymazať obrazovku const int BLINK_INTERVAL = 200;     // Interval blikania LED
(ms)
```

V podstate ide o to, že programu oznamuje, „čo predstavuje každá z rôznych dĺžok stlačenia a čakacích časov

znamená“:

SHORT_PRESS_THRESHOLD = 500 znamená, že ak je klávesa stlačená menej ako 500 milisekúnd, program to klasifikuje ako „bodku (.)“, čo je základný časový prah na rozlíšenie bodiek (.) od pomlčiek (-);

CHAR_INTERVAL = 1500 sa používa na určenie „či bol znak zadaný“; ak po uvoľnení predchádzajúcej klávesy nedôjde k novému zadaniu viac ako 1,5 sekundy, program to považuje za ukončenie aktuálnej morseovej sekvencie a môže automaticky vykonať dekódovanie abecedy;

CLEAR_SCREEN_THRESHOLD = 1000 sa používa na rozlíšenie funkcie tlačidla na vymazanie obrazovky. Ak doba stlačenia presiahne 1 sekundu, už sa nevymaže jeden znak, ale priamo sa vymaže celý obsah displeja;

Nakoniec, BLINK_INTERVAL = 200 riadi rytmus blikania LED, čím je svetelná spätná väzba jasná, ale nie príliš rýchla, čo uľahčuje rozpoznávanie ľudským okom.

Intervaly identifikácie pre každé tlačidlo

```
const int MORSE_BTN_RANGE[2]        = {500, 520};      // Tlačidlo pre
morseovský vstup const int SPACE_BTN_RANGE[2]      = {680, 690};         //
Tlačidlo medzerníka
const int DELETE_BTN_RANGE[2] = {845, 860};           // Tlačidlo Odstrániť / Vymazať
```

Viacero tlačidiel zdieľa jeden analógový vstupný pin. Ako môže program rozlíšiť, ktoré tlačidlo bolo skutočne stlačené?

Keďže hardvér používa metódu deliča odporov (odporový rebrík), pri stlačení rôznych tlačidiel bude hodnota napätia načítaná funkciou analogRead() spadáť do rôznych číselných rozsahov.

MORSE_BTN_RANGE = {500, 520} označuje, že keď analógová hodnota spadá do rozsahu 500 až 520, program považuje stlačenie „tlačidla Morseovho vstupu“, čo je prvá klávesa;

SPACE_BTN_RANGE = {680, 690} zodpovedá „medzerníku“, čo je druhá klávesa; DELETE_BTN_RANGE = {845, 860} predstavuje „tlačidlo na vymazanie/vyčistenie obrazovky“, čo je tretia klávesa.

Program neurčuje pevnú hodnotu, ale skôr rozsah. Je to preto, že skutočné napätie ovplyvňujú faktory, ako sú chyby odporu a kolísanie napájania. Použitím tejto metódy „posudzovania intervalu“ je možné pomocou jediného analógového portu čítať stavy viacerých tlačidiel, čo nielen šetrí zdroje IO, ale aj zabezpečuje stabilné rozpoznávanie.

„Premenné prevádzkového stavu“ celého systému morseovského vstupu

```
String currentMorse = "";          // Aktuálna zadávaná morseovská sekvencia
Reťazec displayText    = "";       // Celý text zobrazený na LCD (história zachovaná)


unsigned long buttonPressTime = 0;                    // Časová pečiatka stlačenia tlačidla
unsigned long lastButtonReleaseTime = 0;          // Časová pečiatka posledného
uvoľnenia tlačidla

bool morseBtnPressed        = false;       // Stav tlačidla Morse
bool spaceBtnPressed        = false;       // Stav tlačidla
medzerníka
```

```
bool stlačené tlačidlo Delete = false;  // Stav tlačidla Delete
```

Slúžia ako „pamäť“ a „základ pre rozhodovanie“ programu, používajú sa na ukladanie aktuálneho obsahu vstupu, informácií o čase a stavu stlačenia klávesov.

Premenná „currentMorse“ sa používa na dočasné ukladanie zadávaného morseovho kódu (napr. .- a --.) a priebežne sa dopĺňa, kým používateľ nedokončí písmeno;

Premenná „displayText“ ukladá kompletný textový obsah, ktorý bol úspešne dekódovaný a zobrazený na LCD displeji; aj keď sa nepretržite zadávajú nové znaky, predchádzajúci obsah zostane zachovaný.

Dve premenné typu unsigned long „buttonPressTime“ a „lastButtonReleaseTime“ sa používajú špeciálne na meranie času. Spolu s funkciou millis() určujú, „ako dlho bolo stlačené“ a „ako dlho bolo uvoľnené“, čím rozlišujú medzi krátkym stlačením a dlhým stlačením a či bola prekročená doba nečinnosti pre automatické dekódovanie.

Tri premenné typu bool (morseBtnPressed, spaceBtnPressed, deleteBtnPressed) sú indikátory stavu klávesov, ktoré sa používajú na zaznamenanie, či je určitá klávesa momentálne v stave „už stlačená“, aby sa zabránilo opakovanému rozpoznaniu jedného stlačenia.

Sekcia nastavenia

```
void setup() {
  Serial.begin(115200);
  // Inicializácia pinov LED
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);

  // Inicializácia LCD
  lcd.begin(COLUMNS, ROWS, LCD_5x8DOTS);
  lcd.clear();
  lcd.backlight();

  // Spúšťacia správa lcd.setCursor(0,
  0); lcd.print(F("Morse Decoder"));
  lcd.setCursor(0, 1);
  lcd.print(F("Ready"));

  delay(1500); lcd.clear();
}
```

Funkcia setup() slúži na vykonanie kompletnej inicializácie celého „Morse Input System“ pri zapnutí, čo možno považovať za „samokontrolu pri zapnutí + prípravnú fázu“ systému.

Najskôr sa pomocou Serial.begin(115200) otvorí sériová komunikácia, hlavne na účely ladenia a sledovania stavu behu programu, čo uľahčuje kontrolu správneho fungovania systému na strane počítača;

Potom sa pomocou pinMode nastavia piny červeného, žltého a zeleného LED diodu do výstupného režimu. Tieto LED diódy sa neskôr použijú ako spätná väzba vstupu a indikátory stavu.
Ďalej nasleduje inicializácia LCD: lcd.begin(COLUMNS, ROWS, LCD_5x8DOTS) informuje systém, že ide o 16×2 znakové LCD a používa štandardné 5×8 bodové maticové písmo. lcd.clear() zabezpečí čistú obrazovku a lcd.backlight() zapne podsvietenie, aby bol obsah viditeľný.
Potom program zobrazí na LCD štartovaciu výzvu „Morse Decoder“ a „Ready“. Ide o typický dizajn interakcie človek-počítač, ktorý informuje používateľa, že systém sa úspešne spustil a je v stave pripravenosti.
Nakoniec sa pomocou delay(1500) zabezpečí, aby sa výzva zobrazovala 1,5 sekundy, potom sa obrazovka vymaže a pripraví sa na formálne rozhranie pre zadávanie morseovky.

loop

```
void loop() {
   // Prečíta hodnotu zdieľaného analógového
   tlačidla int val =
   analogRead(BUTTON_ANALOG);

   // Spracovanie každého tlačidla na základe
   analógovej hodnoty handleMorseButton(val);
   handleSpaceButton(val); handleDeleteButton(val);

   delay(10);      // Jednoduché oneskorenie na odstránenie odskoku
}
```

Táto funkcia loop() je hlavnou bežiacou slučkou celého systému dekódovania Morseovej abecedy. Môžeme ju považovať za hlavný riadiaci mozog systému, ktorý neustále „skenuje a reaguje na operácie používateľa“.
Najprv funkcia analogRead(BUTTON_ANALOG) prečíta hodnotu napätia z toho istého analógového pinu. Je to preto, že tri tlačidlá zdieľajú ten istý analógový pin prostredníctvom metódy odporového deliča (odporového rebríka). Rôzne tlačidlá zodpovedajú rôznym rozsahom napätia a prečítaná hodnota val je ekvivalentom pôvodného základu na určenie „ktorá klávesa je momentálne stlačená“.
Ďalej program nespracováva logiku priamo v slučke loop(), ale namiesto toho odovzdáva tú istú analógovú hodnotu trom funkciám: handleMorseButton(val), handleSpaceButton(val) a handleDeleteButton(val) na posúdenie a spracovanie. Ide o veľmi typický a jasný koncept modulárneho dizajnu: hlavná slučka je zodpovedná len za „plánovanie“ a konkrétne funkcie vykonávajú špecializované funkcie.
Záverečné delay(10) je jednoduché softvérové oneskorenie na potlačenie odrazov, ktoré slúži na zníženie nesprávnych posúdení spôsobených chvením tlačidiel a chvením analógového odčítania, pričom výrazne neovplyvňuje rýchlosť odozvy systému.

Funkcia handleMorseButton

```
void handleMorseButton(int val) {
   bool active = (val >= MORSE_BTN_RANGE[0] && val <= MORSE_BTN_RANGE[1]);

   // Stlačenie tlačidla
```

```
    if (active && !morseBtnPressed) {
        morseBtnPressed = true;
        buttonPressTime = millis();
        digitalWrite(LED_YELLOW, HIGH);          // Žltá LED indikuje stlačenie tlačidla
    }
    // Tlačidlo uvoľnené
    inak ak (!aktívne && morseBtnPressed) {
        morseBtnPressed = false;
        digitalWrite(LED_YELLOW, LOW);

        unsigned long duration = millis() - buttonPressTime;
        lastButtonReleaseTime = millis();

        // Určenie bodky alebo čiarky
        if (trvanie < SHORT_PRESS_THRESHOLD) {
            currentMorse += "."; digitalWrite(LED_GREEN,
            HIGH);
        } inak {
            currentMorse += "-";
            digitalWrite(LED_RED, HIGH);
        }

        delay(100); digitalWrite(LED_GREEN,
        LOW); digitalWrite(LED_RED, LOW);
    }

    // Automatické dekódovanie v prípade uplynutia časového limitu nečinnosti
    if (!active && currentMorse.length() > 0) {
        if (millis() - lastButtonReleaseTime > CHAR_INTERVAL) { char c =
            decodeMorse(currentMorse);
            displayText += c; updateLCDDisplay();
            resetMorseInput();
        }
    }
}
```

Celá táto funkcia „handleMorseButton(int val)“ v skutočnosti implementuje kompletný „stavový stroj pre morseovský vstup“: je zodpovedná za identifikáciu toho, „či bolo stlačené morseovské tlačidlo“, „ako dlho bolo stlačené“, „či ide o bodku alebo pomlčku“ a „kedy môže byť automaticky dekódované na písmeno“.
Na začiatku, pomocou „val >= MORSE_BTN_RANGE[0] && val <= MORSE_BTN_RANGE[1]“,
hodnota analógového napätia prevádza na logický stav „aktívny“, čo je kľúčový krok od analógového tlačidla → digitálny stav. V podstate nahrádza „digitálnu úroveň pinov high/low“ „rozsahom napätia

rozsah“. Keď sa zistí „active && !morseBtnPressed“, znamená to, že tlačidlo bolo práve stlačené. Ide o detekciu hrany a program okamžite zaznamená čas stlačenia „buttonPressTime = millis()“ a zapne žlté svetlo, aby poskytol používateľovi spätnú väzbu v reálnom čase o tom, že „tlačidlo je stlačené“.

```
149  void handleMorseButton(int val) {
150    bool active = (val >= MORSE_BTN_RANGE[0] && val <= MORSE_BTN_RANGE[1]);
151
152    // Button pressed
153    if (active && !morseBtnPressed) {
154      morseBtnPressed = true;
155      buttonPressTime = millis();
156      digitalWrite(LED_YELLOW, HIGH);  // Yellow LED indicates button hold
157    }
```

Keď sa vyskytne „!active && morseBtnPressed“, znamená to, že tlačidlo bolo práve uvoľnené. Ide o detekciu klesajúcej hrany. V tomto bode sa vypočíta trvanie „duration“ stlačenia pomocou „millis() - buttonPressTime“. Toto je fyzikálny základ pre rozlišovanie medzi „bodkami“ a „čiarkami“: kratšie ako „SHORT_PRESS_THRESHOLD“ sa považuje za bodku (.), dlhšie ako prahová hodnota sa považuje za čiarku (-), a „.“ alebo „-“ sa príslušne pripojí k reťazcu „currentMorse“. Zároveň sa ako vizuálna spätná väzba používa krátke blikanie zeleného alebo červeného svetla. Potom sa zaznamená „lastButtonReleaseTime“, aby sa pripravilo následné „posúdenie časového limitu automatického dekódovania“.

```
158    // Button released
159    else if (!active && morseBtnPressed) {
160      morseBtnPressed = false;
161      digitalWrite(LED_YELLOW, LOW);
162
163      unsigned long duration = millis() - buttonPressTime;
164      lastButtonReleaseTime = millis();
165
166      // Determine dot or dash
167      if (duration < SHORT_PRESS_THRESHOLD) {
168        currentMorse += ".";
169        digitalWrite(LED_GREEN, HIGH);
170      } else {
171        currentMorse += "-";
172        digitalWrite(LED_RED, HIGH);
173      }
174
175      delay(100);
176      digitalWrite(LED_GREEN, LOW);
177      digitalWrite(LED_RED, LOW);
178    }
```

Posledný úsek kódu stelesňuje koncepciu interakcie človek-počítač riadenú časom: Ak nie sú stlačené žiadne klávesy a bol zadaný morseov kód, a ak doba nečinnosti od posledného uvoľnenia klávesy do súčasnosti presiahne hodnotu „CHAR_INTERVAL“, považuje sa to za ukončenie zadávania celého znaku. V tomto bode sa volá funkcia „decodeMorse()“, ktorá preloží morseovskú sekvenciu, napríklad „.-“, na písmená, a potom ju pripojí k „displayText“, obnoví LCD

a prostredníctvom „resetMorseInput()“ vymaže aktuálny stav zadávania, čím sa pripraví na ďalší znak.

```
180     // Auto-decode if idle timeout reached
181     if (!active && currentMorse.length() > 0) {
182       if (millis() - lastButtonReleaseTime > CHAR_INTERVAL) {
183         char c = decodeMorse(currentMorse);
184         displayText += c;
185         updateLCDDisplay();
186         resetMorseInput();
187       }
188     }
189   }
```

Celkovo táto funkcia integruje určenie analógového vstupu, detekciu hrany, meranie času, riadenie stavového automatu a spätnú väzbu pre používateľa. Je to najpodstatnejšia a „inžinierska“ časť celého systému dekódovania morseovky.

Funkcia handleSpaceButton

```
void handleSpaceButton(int val) {
    bool active = (val >= SPACE_BTN_RANGE[0] && val <= SPACE_BTN_RANGE[1]);

    if (active && !spaceBtnPressed) {
        spaceBtnPressed = true;
        digitalWrite(LED_GREEN, HIGH);          // Zelená LED indikuje stlačenie medzerníka
    }
    inak ak (!active && spaceBtnPressed) {
        spaceBtnPressed = false;
        digitalWrite(LED_GREEN, LOW);

        displayText += " ";
        updateLCDDisplay();
        delay(100);
    }
}
```

Táto funkcia handleSpaceButton(int val) implementuje kompletnú logiku vstupu pre „medzerník“. Logika je veľmi podobná predchádzajúcemu Morseovmu kľúču, ale je jednoduchšia z hľadiska funkčnosti. Hlavný cieľ je len jeden: vložiť medzeru do textu na displeji v správnom čase.

Najprv sa pomocou val >= SPACE_BTN_RANGE[0] && val <= SPACE_BTN_RANGE[1]
sa hodnota analógového vstupu prevedie na booleovskú premennú active. Ide o typické určenie rozsahu analógovej klávesy, ktoré sa používa na určenie, či je v danom momente stlačená „medzerník“.

Pri detekcii active && !spaceBtnPressed to znamená, že medzerník bol práve stlačený. Ide o udalosť detekcie stlačenia klávesy. Program okamžite nastaví spaceBtnPressed na true a zapne zelenú LED diódu, ktorá slúži ako spätná väzba pre používateľa, aby jasne informovala používateľa, že „medzerník je stlačený“.

```
195  void handleSpaceButton(int val) {
196    bool active = (val >= SPACE_BTN_RANGE[0] && val <= SPACE_BTN_RANGE[1]);
197
198    if (active && !spaceBtnPressed) {
199      spaceBtnPressed = true;
200      digitalWrite(LED_GREEN, HIGH);  // Green LED indicates space key
201    }
```

Pri vstupe do vetvy !active && spaceBtnPressed sa signalizuje, že klávesa bola uvoľnená. Ide o detekciu uvoľnenia klávesy. V tomto momente program zhasne zelené svetlo a logicky vykoná akciu „vstup medzerníka“: volaním displayText += " " pripojí medzerník k aktuálne zobrazenému reťazcu a následne volaním updateLCDDisplay() obnoví LCD displej, čím sa obsah obrazovky okamžite aktualizuje.

```
202    else if (!active && spaceBtnPressed) {
203      spaceBtnPressed = false;
204      digitalWrite(LED_GREEN, LOW);
205
206      displayText += " ";
207      updateLCDDisplay();
208      delay(100);
209    }
210  }
```

V tomto prípade sa doba stlačenia nepoužíva na posúdenie. Je to preto, že klávesa medzerníka sama nerozlišuje medzi krátkym a dlhým stlačením; zaujíma ju len kompletná udalosť „stlačenie → uvoľnenie“. Záverečné delay(100) slúži ako jednoduchá funkcia odskoku a proti blikaniu, ktorá zabraňuje vloženiu viacerých medzier za sebou v dôsledku simulovaného kolísania hodnoty alebo nečistého zdvihnutia prsta.

Funkcia handleDeleteButton

```
void handleDeleteButton(int val) {
  bool active = (val >= DELETE_BTN_RANGE[0] && val <= DELETE_BTN_RANGE[1]);

  if (active && !deleteBtnPressed) {
    deleteBtnPressed = true;
    buttonPressTime = millis();
    digitalWrite(LED_RED, HIGH);
  }
  else if (deleteBtnPressed) {
    unsigned long pressTime = millis() - buttonPressTime;

    // Dlhé stlačenie: vymazať celý displej
    if (pressTime >= CLEAR_SCREEN_THRESHOLD) { displayText =
      "";
      lcd.clear(); blinkRedLED(3);
      deleteBtnPressed = false;
```

```
            digitalWrite(LED_RED, LOW);
        }
        // Krátke stlačenie: vymazať posledný
        znak else if (!active) {
            if (dĺžkaTextu.dĺžka() > 0) { dĺžkaTextu.odstrániť(dĺžkaTextu.dĺžka() - 1);
                aktualizovaťLCDDisplej();
            }
            deleteBtnPressed = false;
            digitalWrite(LED_RED, LOW);
            delay(100);
        }
    }
}
```

Táto funkcia handleDeleteButton(int val) implementuje logiku tlačidla na vymazanie s „rozlíšením medzi krátkym a dlhým stlačením“. Komplexne využíva tri základné poznatky: detekciu simulovaných klávesov, uvažovanie pomocou stavového automatu a meranie času.

Funkcia tiež najskôr prevádza simulovanú vstupnú hodnotu pomocou podmienky val >= DELETE_BTN_RANGE[0] && val <= DELETE_BTN_RANGE[1] na hodnotu „active“, ktorá slúži na určenie, či je tlačidlo na vymazanie stlačené; ak sa zistí stav „active“ a zároveň „!deleteBtnPressed“, znamená to, že tlačidlo na vymazanie bolo práve stlačené, čo je spúšťačom udalosti stlačenia. Program okamžite nastaví deleteBtnPressed na true a pomocou buttonPressTime = millis() zaznamená čas, kedy bolo tlačidlo stlačené, pričom zároveň rozsvieti červenú LED diódu, aby poskytol používateľovi intuitívnu spätnú väzbu, že „prebieha operácia vymazania“.

```
217  void handleDeleteButton(int val) {
218    bool active = (val >= DELETE_BTN_RANGE[0] && val <= DELETE_BTN_RANGE[1]);
219
220    if (active && !deleteBtnPressed) {
221      deleteBtnPressed = true;
222      buttonPressTime = millis();
223      digitalWrite(LED_RED, HIGH);
224    }
```

Nasledujúca vetva else if (deleteBtnPressed) označuje: Pokiaľ je tlačidlo na vymazanie v stave „stlačené“, program bude nepretržite kontrolovať, ako dlho je stlačené. Tu sa pressTime vypočíta pomocou millis() - buttonPressTime, čo je typická metóda merania trvania stlačenia klávesy na základe systémového času. Ak je pressTime >= CLEAR_SCREEN_THRESHOLD, znamená to, že ide o operáciu dlhého stlačenia. Program potom vymaže displayText, vymaže obrazovku LCD a volá blinkRedLED(3), aby červené svetlo trikrát zablikalo ako výrazná výzva na „úspešné vymazanie celej obrazovky“, a potom resetuje stav deleteBtnPressed a vypne červené svetlo, aby sa zabránilo opakovanému spúšťaniu.

```
225     else if (deleteBtnPressed) {
226       unsigned long pressTime = millis() - buttonPressTime;
227
228       // Long press: clear entire display
229       if (pressTime >= CLEAR_SCREEN_THRESHOLD) {
230         displayText = "";
231         lcd.clear();
232         blinkRedLED(3);
233         deleteBtnPressed = false;
234         digitalWrite(LED_RED, LOW);
235       }
```

Ak nie je dosiahnutá prahová hodnota pre dlhé stlačenie a zistí sa, že stav je !active, znamená to krátke stlačenie a uvoľnenie prsta. V tomto momente program vymaže len posledný znak: Najskôr skontrolujte, či je reťazec prázdny, aby ste sa vyhli chybám prekročenia hraníc. Potom použite displayText.remove(displayText.length() - 1) na presné vymazanie posledného znaku. Následne zavolajte funkciu updateLCDDisplay() na obnovenie displeja. Nakoniec vynulujte stav, vypnite červené svetlo a pridajte krátke oneskorenie (delay(100)) ako spracovanie odskoku.

```
236       // Short press: delete last character
237       else if (!active) {
238         if (displayText.length() > 0) {
239           displayText.remove(displayText.length() - 1);
240           updateLCDDisplay();
241         }
242         deleteBtnPressed = false;
243         digitalWrite(LED_RED, LOW);
244         delay(100);
245       }
246     }
247   }
```

Funkcia decodeMorse

```
char decodeMorse(String morse) { for
   (int i = 0; i < 26; i++) {
      if (morse == morseCodes[i]) return letters[i];
   }
   vráť '?';        // Neznáma morseovská sekvencia
}
```

Funkcia „decodeMorse(String morse)“ je kľúčovou funkciou pre prevod „Morseovej abecedy → písmen“ v celom programe. V podstate implementuje vyhľadávaciu tabuľku + proces sekvenčného porovnávania.
Vstupný parameter „morse“ tejto funkcie je celý reťazec morseovských signálov, ktorý práve zadal používateľ (napríklad „.-“ a „-...“). Použitie typu „String“ slúži na uľahčenie porovnávania reťazcov;
V rámci funkcie sa pomocou cyklu for (int i = 0; i < 26; i++), ktorý prechádza vopred definovaným poľom „morseCodes[]“, každý prvok tohto poľa jednoznačne zodpovedá písmenu anglického abecedy (index 0 zodpovedá A, 1 zodpovedá B atď.). Preto premenná i tu slúži ako index poľa a zároveň implicitne reprezentuje pozíciu písmena.

Keď je splnená podmienka „if (morse == morseCodes[i])“ pomocou preťaženého operátora „==“ triedy „String“ Arduina, obsah „aktuálne zadaného morseovho kódu“ sa porovná s „určitým štandardným morseovým kódom v slovníku“. Akonáhle je zhoda úspešná, funkcia okamžite vráti „letters[i]“, čím vráti príslušné písmeno volajúcemu. Tu je „letters[]“ presne tá istá tabuľka znakov s rovnakými indexmi ako „morseCodes[]“.

Ak sa po ukončení cyklu nenájdu žiadne zhodné položky, znamená to, že sekvencia morseovského kódu zadaná používateľom nie je platným alebo definovaným písmenom (napr. chyba zadania, abnormálny rytmus atď.). V tomto bode funkcia vráti '?' ako náhradný výsledok. Ide o návrh na riešenie chýb a indikáciu výnimiek, ktorý môže zabrániť zrúteniu programu a jasne informovať používateľa na úrovni displeja, že „aktuálny vstup nie je možné rozpoznať“.

Funkcia resetMorseInput

```
void resetMorseInput() {
    currentMorse = "";
    lastButtonReleaseTime = millis();
}
```

Najprv currentMorse = ""; Tento riadok vymaže aktuálny reťazec morseovského kódu. Logicky to zodpovedá výroku „Po dekódovaní jedného písmena by sa ďalší vstup mal začať od začiatku“. Ak sa tento krok nevykoná, bodky a čiarky v nasledujúcom vstupe sa omylom prekrývajú s predchádzajúcim písmenom, čo vedie k chaotickému výsledku dekódovania.

Tento riadok „lastButtonReleaseTime = millis();“ teda nie je napísaný náhodne. Využíva systémovú časovú funkciu millis() platformy Arduino na zaznamenanie „aktuálneho okamihu“ ako najnovšieho času uvoľnenia tlačidla. Cieľom je umožniť, aby následné „posúdenie doby nečinnosti“ (CHAR_INTERVAL) reštartovalo časovač, čím sa zabráni tomu, aby práve dekódovaný časový interval bol nesprávne vyhodnotený ako nová podmienka časového limitu.

Funkcia updateLCDDisplay

```
void updateLCDDisplay() {
    lcd.clear();
    if (displayText.length() <= COLUMNS) {
        lcd.setCursor(0, 0);
        lcd.print(displayText);
    } else {
        lcd.setCursor(0, 0);
        lcd.print(displayText.substring(0, COLUMNS));
        lcd.setCursor(0, 1);
        lcd.print(displayText.substring(COLUMNS));
    }
}
```

Táto funkcia updateLCDDisplay() stelesňuje kompletný návrhový koncept „bezpečného a kontrolovateľného mapovania stavu reťazca v pamäti na 16×2 LCD obrazovku“.

Funkcia začína použitím lcd.clear(), čo je veľmi dôležitá operácia zobrazenia. Jej významom nie je „lenivé vymazanie obrazovky“, ale zabrániť tomu, aby na obrazovke zostali zvyškové znaky, keď je zobrazený obsah dlhší ako aktuálny obsah, čím sa zabezpečí, že LCD

sa vždy obnoví s „čistým obrazom“.

Potom sa prostredníctvom displayText.length() určí dĺžka aktuálneho reťazca, ktorý sa má zobraziť. Tento krok odráža aktívne prispôsobenie programu fyzickým obmedzeniam hardvéru: keďže každý riadok LCD môže zobraziť najviac COLUMNS (tu 16) znakov, je potrebné najprv určiť, či sa reťazec zmestí do jedného riadku. Ak je dĺžka menšia alebo rovná 16, kurzor sa presunie na počiatočnú pozíciu prvého riadku pomocou lcd.setCursor(0, 0) a celý reťazec sa vypíše, čo zodpovedá najjednoduchšej a najintuitívnejšej situácii zobrazenia;
Ak dĺžka reťazca presiahne 16 znakov, kód prejde do vetvy „else“. Pomocou funkcií substring(0, COLUMNS) a substring(COLUMNS) sa logicky dlhý reťazec rozdelí na dva segmenty. Prvých 16 znakov sa zobrazí v prvom riadku a zvyšné znaky v druhom riadku.

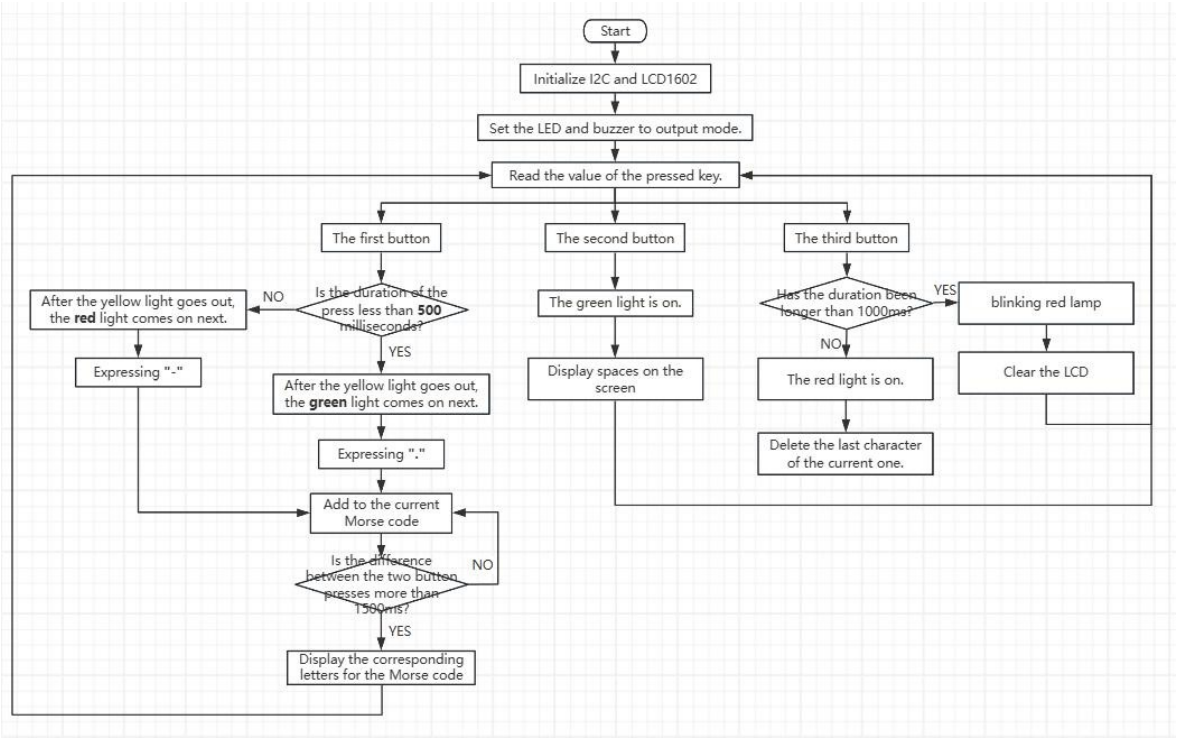
Funkcia blinkRedLED

```
void blinkRedLED(int times) { for
    (int i = 0; i < times; i++) {
        digitalWrite(LED_RED, HIGH);
        delay(BLINK_INTERVAL);
        digitalWrite(LED_RED, LOW);
        delay(BLINK_INTERVAL);
    }
}
```

Existencia parametra funkcie „times“ je mimoriadne dôležitá. Nie je nastavená len na niekoľkokrát bliknutie, ale abstrahuje „počet bliknutí“ do premennej. Týmto spôsobom je možné v rôznych scenároch (napríklad pri signalizácii chyby, úspechu alebo varovaní používateľa) znovu použiť ten istý kód zadávaním rôznych hodnôt, čo demonštruje dobrý koncept modularizácie.

Vo vnútri funkcie sa používa cyklus for, čo je typická riadiaca štruktúra cyklu typu poradia. int i = 0; i < times; 'i++' znamená, že celý proces blikania sa vykoná 'times'-krát a každý cyklus zodpovedá kompletnému cyklu „zapnuté → vypnuté“. V tele cyklu 'digitalWrite(LED_RED, HIGH)' nastaví pin červenej LED na vysokú úroveň, čo znamená, že dodáva napätie do LED, čím ju rozsvieti; hneď za tým nasleduje 'delay(BLINK_INTERVAL)', ktoré sa používa na udržanie rozsvieteného stavu po určitú dobu. Táto doba je jednotne riadená konštantou 'BLINK_INTERVAL', aby sa zabezpečila konzistentnosť rytmu blikania. Následne 'digitalWrite(LED_RED, LOW)' stiahne pin na nízku úroveň, čím preruší prúd a LED zhasne. Potom sa použije ďalší identický 'delay(BLINK_INTERVAL)' na udržanie stavu vypnutia, čím „zapnutie + vypnutie“ spolu tvoria kompletný, vizuálne rozpoznateľný blikajúci cyklus.

Celkový diagram logiky kódu



Kroky na nahratie programu

Tento projekt vyžaduje ďalšie externé knižnice. Pozrite si časť „Importovanie súborov knižníc“ na strane 7 a na začiatku importujte všetky potrebné externé knižnice. Ak ste ich už importovali, môžete tento krok preskočiť.

Podrobné pokyny na nahratie nájdete v časti „Kroky nahratia“ na strane 8.