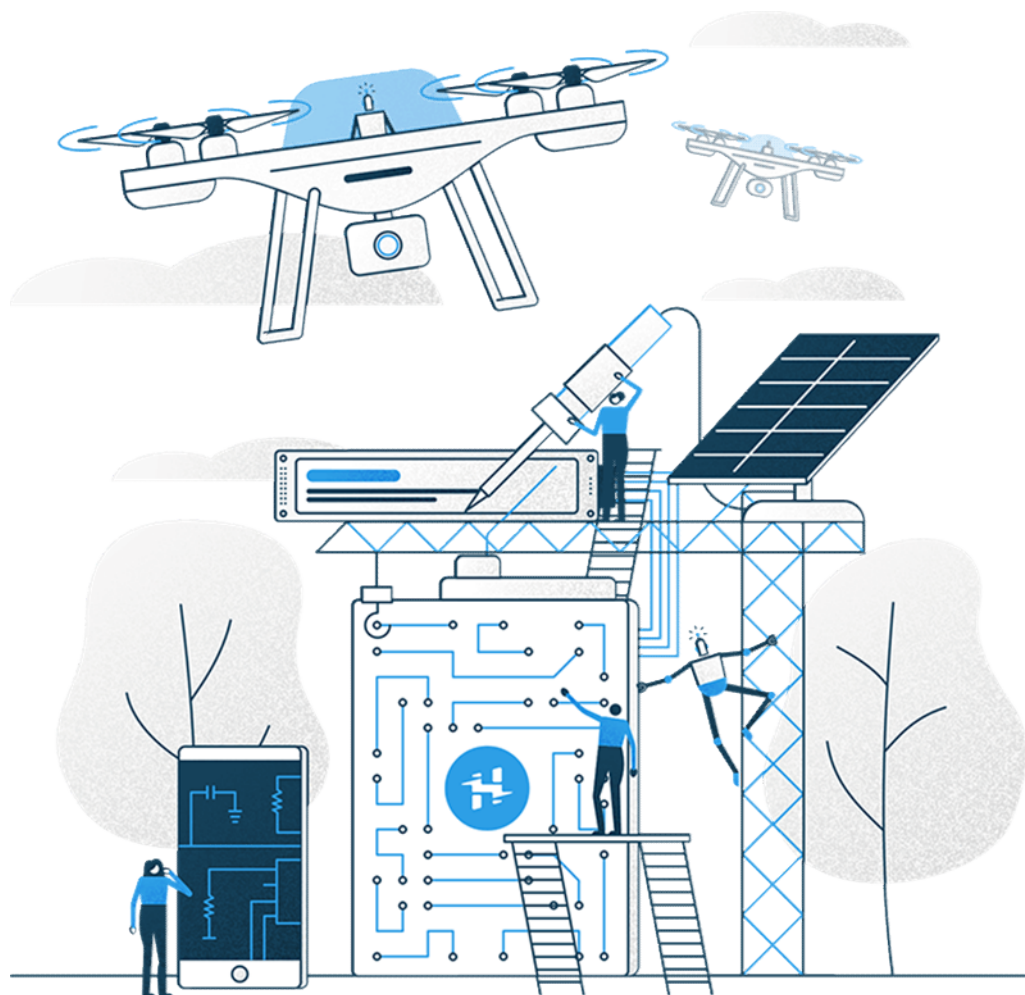
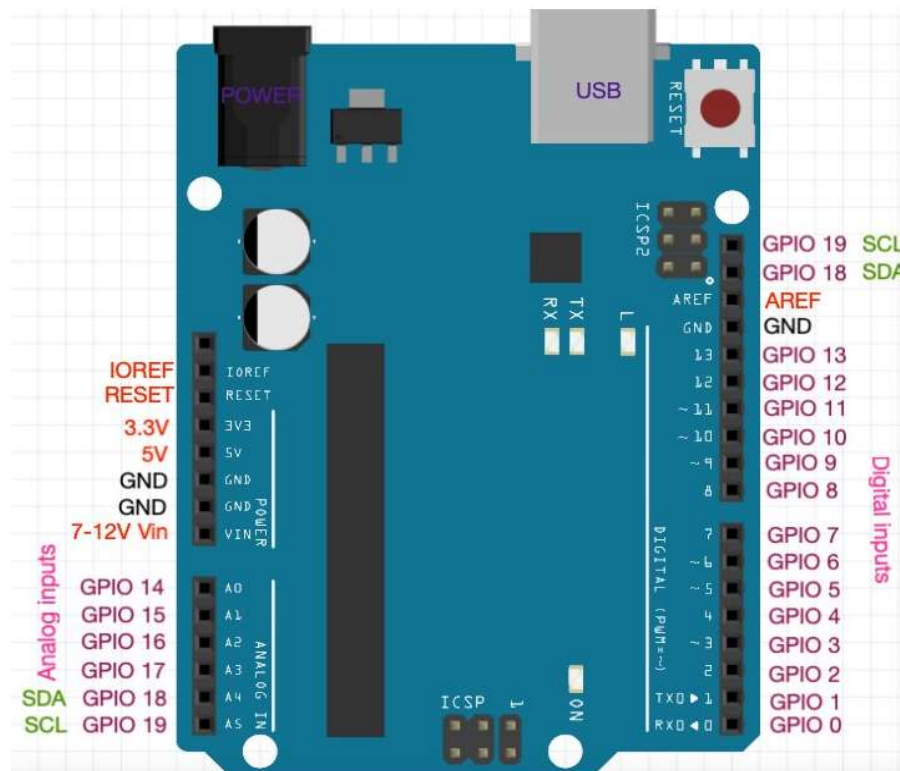


МІКРОПРОЦЕСОРНІ СИСТЕМИ УПРАВЛІННЯ



Lesson 3

Порти введення/виведення AVR



Мікроконтролер — це логічний пристрій, який створено для управління іншими пристроями за допомогою логічних (цифрових) сигналів. Це означає, що максимум можна зняти з порта 40 мА, а рекомендується не більше 20 мА. Що станеться, якщо зняти з порта більше, ніж він може віддати? Він зламається. Що буде, якщо зняти з декількох портів більше, ніж може віддати мікроконтролер в цілому? Згорить мікроконтролер. Цифрові порти служать для подачі команд іншим пристроям, наприклад реле / транзисторів для комутації навантажень.

Порти введення/виведення AVR

За замовчуванням усі порти Arduino визначаються як входи, і немає потреби описувати це в коді. Порти зазвичай прописуються в функції ініціалізації змінних. Ініціалізація порту вводу-виводу Arduino:

pinMode (pin, mode). Параметри:

- pin: номер виводу, режим роботи якого задається;
- mode: приймає значення: INPUT — вхід, у цьому режимі відбувається зчитування даних з датчиків, стану кнопок, аналогового та цифрового сигналу. Порт знаходиться в так званому високоімпедансному стані, тобто на вході високий опір. OUTPUT — вихід, залежно від команди прописаної в коді, порт приймає значення 1 або 0. Вихід стає свого роду керованим джерелом живлення та видає максимальний струм (20 мА) у навантаження, що до нього підключене. INPUT_PULLUP — порт працює як вхід, але до нього підключається «PushUp» резистор з номіналом 20 – 50 кОм;
- значення, що повертаються – немає.

digitalWrite (pin, value). Параметри:

- pin: номер виводу;
- value: значення HIGH або LOW;
- значення, що повертаються — немає.

digitalRead (pin). Параметри:

- pin: номер цифрового виводу, з якого необхідно зчитати значення (int);
- значення, що повертаються HIGH або LOW.

Порти введення/виведення AVR

Для зміни режимів роботи портів D2-D13, A0-A5 можна використати фрагмент програми:

```
for ( byte i = 2; i <= 19; i ++ ) {  
    pinMode (i, OUTPUT ) ; // робимо виходами }
```

Приклад, в якому порти ініціалізуються як виходи, і на них подається сигнал:

```
void setup () {  
    pinMode (10, OUTPUT )    ; // D10 як вихід  
    pinMode (A3, OUTPUT )    ; // A3 як вихід  
    pinMode (19, OUTPUT )    ; // A5 як вихід (Nano / UNO)  
    digitalWrite (10, HIGH) ; // високий рівень на D10  
    digitalWrite (A3, 1)     ; // високий рівень на A3  
    digitalWrite (19, 1)     ; // високий рівень на A5  
}  
void loop () {}
```

Приклад читання стану порта

```
void setup () {  
    Serial. begin ( 9600 ) ;  
}  
void loop () {  
    Serial. println ( digitalRead ( 5 ) ) ;  
}
```

Порти введення/виведення AVR

Регістри портів дозволяють низькорівневі високошвидкісні маніпуляції з портами мікроконтролера. Мікроконтролери, що використовуються в Arduino мають три порти : B (D8-D13), C(A0-A7), D(D0-D7).

Кожен порт контролюється трьома регістрами, кожен з яких відповідає за певний стан. Регістр DDR визначає, який біт порта вхідний, а який вихідний. Регістр PORT встановлює біт порта у відповідний стан HIGH або LOW, регістр PIN читає стан вхідного порта.

Регістри DDR та PORT можуть бути як прочитані, так і записані. Регістр PIN відповідає за стан вхідних портів, тому може бути лише прочитаний.

PORTD відповідає за виводи 0 - 7.

DDRD — регістр напрямку порту D;

PORTD — регістр даних порту D;

PIND — регістр вхідних даних порту D

PORTC відповідає за аналогові виводи 0 - 5.

DDRC — регістр напрямку порту C;

PORTC — регістр даних порту C;

PINC — регістр вхідних даних порту C

PORTB відповідає за виводи 8 - 13. Два старших біта (6 та 7), що відповідають за виводи кварцу, не використовуються.

DDRB — регістр напрямку порту B;

PORTB — регістр даних порту B;

PINB — регістр вхідних даних порту B

Порти введення/виведення AVR

Приклад роботи з портом D

```
// призначаємо виводи Arduino 1-7 вихідними, вивід 0-вхідним
DDRD = B11111110;
// виводи з 2 по 7 вихідні, стан виводів 0 та 1 не змінюється
DDRD = DDRD | B11111100;
// встановлюємо рівень HIGH на цифрових виводах 7,5,3
PORTD = B10101000;
```

Приклад роботи з портом B

```
void setup() {
  //виставляємо всі біти порту B як вихід, PB4 як вхід
  DDRB = B11101111;
  PORTB = B00000000; //скидаємо всі біти порту B
}
void loop() {
  if (PINB==B00010000) {
    PORTB |= 1 << 5 // PB5=1
    delay (1000) // очікуємо секунду
    PORTB &= ~(1 << 5) // PB5=0
    delay (1000) }
}
```

Порти введення/виведення AVR

При роботі з портами можна використовувати вбудовані в Arduino функції для роботи з бітами порта `bitRead()`, `bitWrite()`, `bitSet()`, `bitClear()`.

`bitRead(x, n)` зчитує стан зазначеного біта числа, де `x`: регістр вхідних даних порту (PINB, PIND, PINC) біт якого буде зчитуватись; `n`: номер біта, стан якого необхідно зчитати (стан біту (0 або 1)).

`bitWrite(x, n, b)` змінює стан зазначеного біта змінної, де `x`: числова змінна, у якій необхідно змінити біт (DDRx або PORTx); `n`: номер біта, стан якого необхідно змінити; `b`: нове значення біта (0 або 1).

`bitSet(x, n)` встановлює зазначений біт (записує 1) числової змінної, де `x`: числова змінна, у якій необхідно змінити біт (DDRx або PORTx); `n`: номер біта, стан якого необхідно змінити.

`bitClear(x, n)` скидає вказаний біт (записує 0) числової змінної, де `x`: числова змінна, у якій необхідно змінити біт (DDRx або PORTx); `n`: номер біта, стан якого необхідно змінити.

Використання регістрів портів суттєво зменшує розмір коду та збільшує швидкодію на декілька порядків

Порти введення/виведення AVR

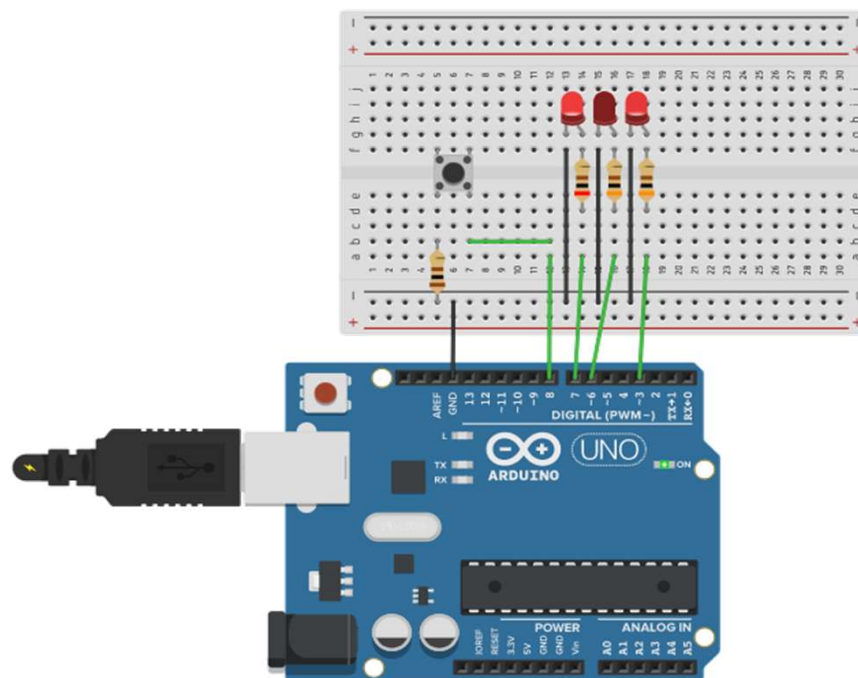
Приклад читання стану цифрового порта (перевірка натиснення кнопки):

```
void setup(){
  pinMode (8, INPUT_PULLUP);
  DDRD=0xFF;
  PORTD = 0b010101010;}

void loop(){
  if (digitalRead(8)==0){
    delay (200);
    if (digitalRead(8)==1)
      PORTD = ~PORTD; }
}
```

```
void setup(){
  pinMode (8, INPUT_PULLUP);
  DDRD=0xFF;
  PORTD = 0b010101010;}

void loop(){
  while(digitalRead(8)==1){}
  while(digitalRead(8)==0){}
  PORTD = ~PORTD;
}
```



Таймери/лічильники. Модуль переривань

У ATmega328 передбачені три таймери/лічильники, на яких реалізовано функції часу, які використовують для формування та вимірювання часових інтервалів. Основні функції часу:

`delay (ms)` Параметри:

- ms — **кількість мілісекунд, на які необхідно призупинити програму;**
- значення, що повертаються — немає;
- опис: припиняє виконання програми на вказаний проміжок часу (в мілісекундах).

`delayMicroseconds (us)`. Параметри:

- us - **кількість мікросекунд, на які необхідно призупинити програму;**
- значення, що повертаються — немає;
- опис: припиняє виконання програми на вказаний проміжок часу (в мікросекундах). Найбільше число для формування затримки — **16383**.

`millis ()`:

- параметри — немає;
- значення, що повертаються — **кількість мілісекунд, що пройшли з моменту старту програми;**
- опис: повертає кількість мілісекунд, що пройшли з моменту старту програми Arduino. Число, що повертається, скинеться в 0) через приблизно **50 днів**.

`micros()`:

- параметри — не має;
- значення, що повертаються — **кількість мікросекунд, що минули з моменту старту програми;**
- опис: повертає кількість мікросекунд, що минули з моменту початку виконання програми Arduino. Число, що повертається, скинеться в 0 через **приблизно 70 хвилин**. Роздільна здатність цієї функції становить чотири мікросекунди.

Таймери/лічильники. Модуль переривань

pulseIn (pin, value) / pulseIn (pin, value, timeout). Параметри:

- pin : номер виводу, на якому буде очікуватися сигнал;
- value: тип імпульсу (HIGH або LOW);
- timeout (опціонально): час очікування імпульсу в мікросекундах (значення за замовчуванням - одна секунда);
- значення, що повертаються — тривалість імпульсу (в мікросекундах) або 0 в разі відсутності імпульсу протягом таймаута;
- опис: зчитує тривалість імпульсу (будь-якого – HIGH або LOW) на виведення. Наприклад, якщо задане значення value = HIGH, то функція pulseIn () очікує появи на виведення сигналу HIGH, потім вимірює час та чекає перемикання в стан LOW, після чого зупиняє відлік часу. Функція повертає тривалість імпульсу в мікросекундах, або 0 в разі відсутності імпульсу протягом певного часу очікування. Функція працює з імпульсами тривалістю від 10 мікросекунд до 3 хвилин.

Переривання – це сигнали, що переривають нормальний перебіг програми. Вони використовуються для апаратних пристроїв, що вимагають негайної реакції на появу подій. Обробка переривань у мікроконтролері відбувається за допомогою модуля переривань, який приймає запити переривання й організовує перехід до виконання визначеної програми. Запити переривання можуть надходити як від зовнішніх джерел, так і від джерел, розташованих у різних внутрішніх модулях мікроконтролера.

Таймери/лічильники. Модуль переривань

Функції переривання Arduino:

attachInterrupt (interrupt, function, mode). Параметри:

- interrupt: номер переривання (0 – pin D2, 1 – pin D3);
- function: функція, яку необхідно викликати при виникненні переривання; ця функція повинна бути без параметрів і не повертати ніяких значень (таку функцію іноді називають оброблювачем переривання);
- mode: визначає умову, за якої має спрацьовувати переривання. Може приймати одне з чотирьох визначених значень: **LOW** — переривання буде спрацьовувати щоразу, коли на виводі присутній низький рівень сигналу, **CHANGE** — переривання буде спрацьовувати щоразу, коли змінюється стан виводу, **RISING** — переривання спрацює, коли стан виводу зміниться з низького рівня на високий, **FALLING** — переривання спрацює, коли стан виводу зміниться з високого рівня на низький;
- значення, що повертаються – немає.

detachInterrupt (pin):

- параметри — немає;
- значення, що повертаються — немає;
- опис: забороняє задане переривання. Забороняє переривання, для того, щоб відключити доступ до даних в процесі виконання переривання.

interrupts ():

- параметри — немає;
- значення, що повертаються — немає;
- опис: повторно дозволяє переривання.

Таймери/лічильники. Модуль переривань

Приклад програми з використанням зовнішнього переривання

```
// Interrupt INT1 (D3)
int led = 5;
volatile int state = LOW;

void setup(){
  pinMode(led, OUTPUT);    // порт як вихід
  attachInterrupt(1, blink, CHANGE);
  // прив'язуємо 1-е переривання до функції blink() ;
}
void loop()
{
  digitalWrite(led, state);    // виводимо state
}
void blink()
{
  state = !state;
}
```

Таймери, як і зовнішні переривання, працюють незалежно від основної програми. У стандартних платах Arduino є три таймера Timer0, Timer1 і Timer2. Timer0 є 8 бітним таймером, це означає, що його рахунковий регістр може зберігати числа до 255. Timer0 використовується стандартними часовими функціями Arduino такими як delay() і millis(), так що краще його не використовувати у своїх проектах.

Timer1 це 16 бітний таймер з максимальним значенням 65535. Timer2 — 8 бітний і дуже схожий на Timer0. Він використовується в функції tone () Arduino.

Таймери/лічильники. Модуль переривань

Для обробки переривань у мові програмування Arduino використовується функція `ISR ()`. Arduino (ATmega328P) використовує такі варіанти параметра функції `ISR ()` для роботи з таймерами:

- ☐ `TIMER2_COMPA_vect` — переривання від Timer2 при збігу з A;
- ☐ `TIMER2_COMPB_vect` — переривання від Timer2 при збігу з B;
- ☐ `TIMER2_OVF_vect` — переривання переповнення Timer2;
- ☐ `TIMER1_CAPT_vect` — переривання від Timer1 (режим захоплення);
- ☐ `TIMER1_COMPA_vect` — переривання від Timer1 при збігу з A;
- ☐ `TIMER1_COMPB_vect` — переривання від Timer1 при збігу з B;
- ☐ `TIMER1_OVF_vect` — переривання переповнення Timer1;
- ☐ `TIMER0_COMPA_vect` — переривання від Timer0 при збігу з A;
- ☐ `TIMER0_COMPB_vect` — переривання від Timer0 при збігу з B;
- ☐ `TIMER0_OVF_vect` — переривання переповнення Timer0.

Для того щоб використовувати таймери в AVR є регістри налаштувань. Таймери містять безліч таких регістрів. Два з них — регістри управління таймера / лічильника містять установчі змінні й називаються `TCCRxA` і `TCCRxB`, де `x` — номер таймера (`TCCR1A` і `TCCR1B`). Кожен регістр містить 8 біт і кожен біт зберігає конфігураційну змінну. Найбільш важливими є три останні біта в `TCCR1B`: `CS12`, `CS11` і `CS10`. Вони визначають тактову частоту таймера (табл. 1). За замовчуванням ці біти не встановлені.

Таймери/лічильники. Модуль переривань

Таблиця 1 — Біти конфігурації частоти роботи таймера/лічильника

CS12	CS11	CS10	Опис
0	0	0	Таймер лічильник 1 зупинений
0	0	1	СК
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Зовнішній вхід T1, спадаючий фронт
1	1	1	Зовнішній вхід T1, наростаючий фронт

TIMSK1 це регістр маски переривань Timer1. Він контролює переривання, які таймер може викликати. Установка біта 0 біту (TOIE1) вказує таймеру, що дозволено переривання коли таймер переповнюється (дораховує до максимального значення з частотою, що визначена бітами CS12, CS11, CS10). Якщо частота предільника Timer1 встановлена у значення 001, то при тактовій частоті 16 МГц Atmega328 переривання виникне приблизно через 0,0041 секунд (65535/16МГц).

Таймери/лічильники. Модуль переривань

Приклад програми з використанням переривання від Timer1 у режимі переповнення:

```
#define LEDPIN 13
int count = 100;
void setup(){
    pinMode(LEDPIN, OUTPUT);
    // Timer1 module overflow interrupt configuration
    TCCR1A = 0;
    TCCR1B = 1; // enable Timer1 with prescaler = 1
    TCNT1 = 0; // set Timer1 preload value to 0
    TIMSK1 = 1; // enable Timer1 overflow interrupt

    ISR(TIMER1_OVF_vect) {
        digitalWrite(LEDPIN, !digitalRead(LEDPIN));
    }
}
void loop(){
    if(digitalRead(button) == 0){
        count++; // increment 'count' by 1
        if(count == 9999){
            count = 0;
        }
        delay(200); // wait 200 milliseconds
    }
}
```

TCNT1 це регістр, який рахує імпульси. У програмі йому присвоєне значення 0, а це значить, що переривання виникне, коли він дорахує до значення 65535. Запуск лічби починається встановленням біту CS10 (TCCR1B = 1) і, як тільки виникає переривання у режимі переповнення, викликається ISR (TIMER1_OVF_vect). Це відбувається завжди коли таймер переповнюється.

Таймери/лічильники. Модуль переривань

Функція `random(max)` / `random(min, max)` генерує псевдо-випадкові числа від `min` до `max-1` (від 0 до 4 294 967 295)

```
long randNumber;

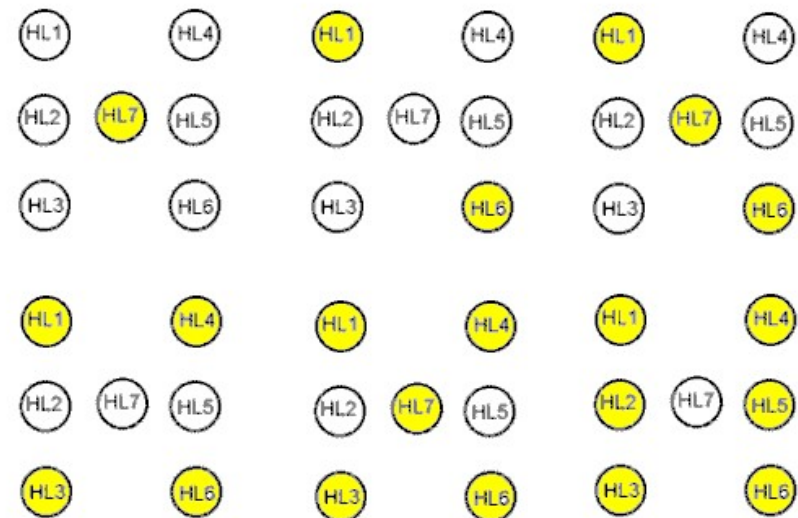
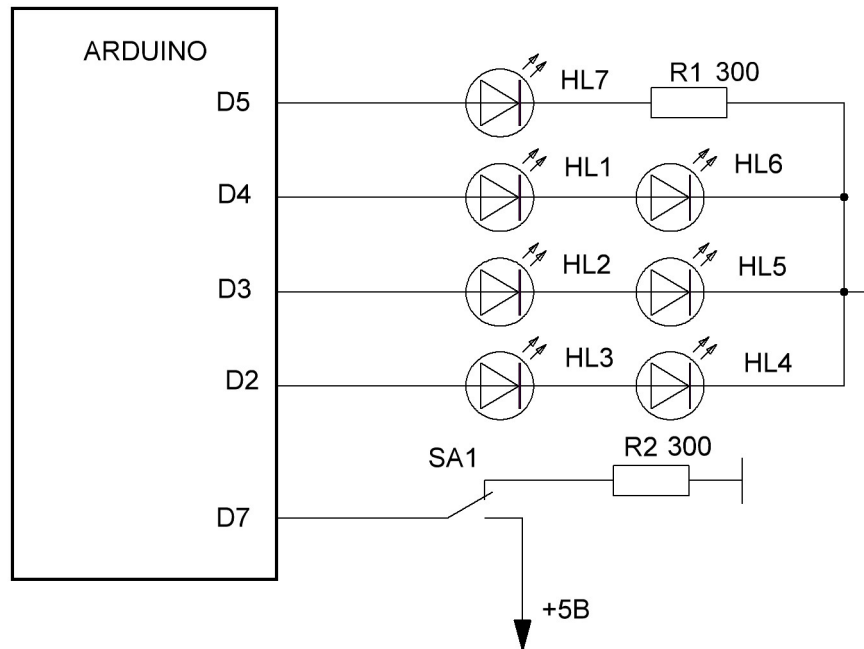
void setup() {
  Serial.begin(9600);
}

void loop() {
  // виводимо випадкове число у діапазоні від 0 до 299
  randNumber = random(300);
  Serial.println(randNumber);

  // виводимо випадкове число у діапазоні від 10 до 19
  randNumber = random(10, 20);
  Serial.println(randNumber);
  delay(50);
}
```

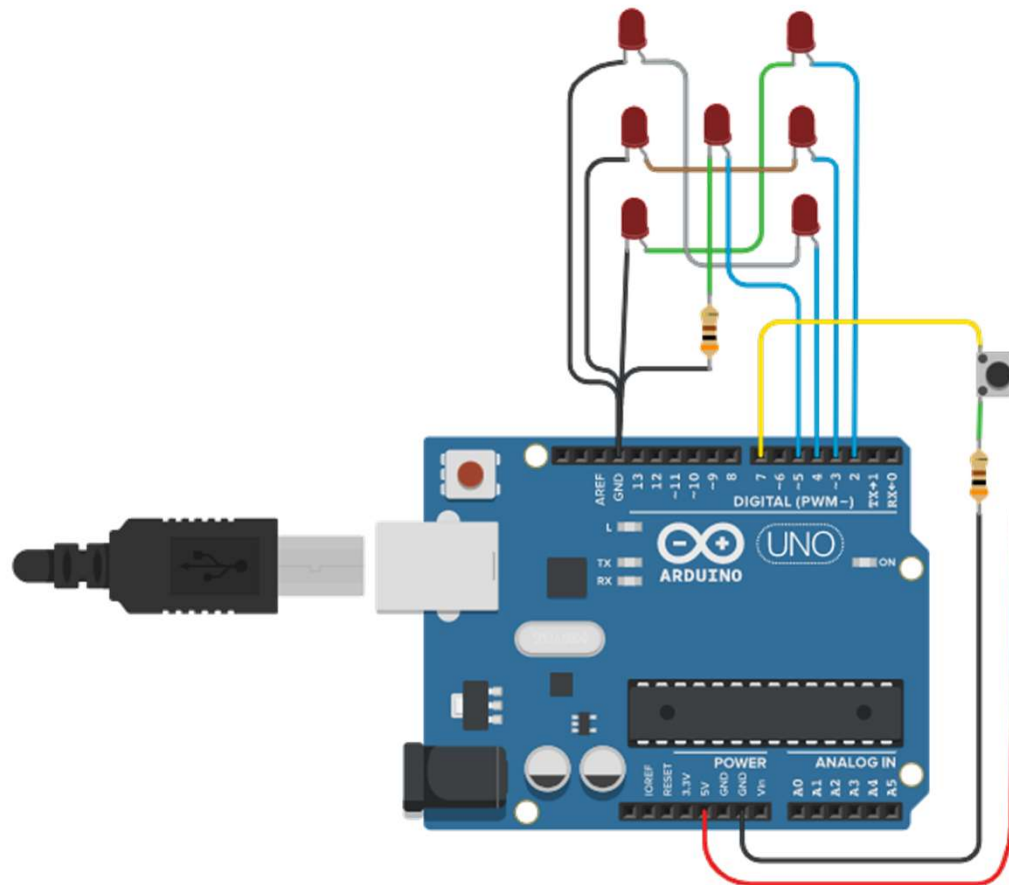

Індивідуальне завдання

Реалізувати електронний кубик на платформі
Arduino



Індивідуальне завдання

Реалізувати електронний кубик на платформі
Arduino



Етапи реалізації

1. Скласти схему у TinkerCad
2. Написати програму (code 1), щоб по черзі відображались цифри 1, 2, 3, 4, 5, 6 (використати `bitRead`, `digitalRead`).
3. Ввести зміни у програму (code 2), щоб з кожним натисненням кнопки по черзі відображались цифри 0 (усе погашено), 1, 2, 3, 4, 5, 6, 0....
4. Ввести зміни у програму (code 3), щоб генерувалось випадкова комбінація `count = random(1,7)`.
5. Внести зміни у схему та програму, щоб використати Interrupt (code4)
5. Оформити звіт. Він містить code 1, code 2, code 3, code 4 та скріни роботи електронного кубика. За можливістю надати посилання на проект у TinkerCad