Arduino code :

1) LED:

```
#define LED PA0
void setup()
{
pinMode(LED, OUTPUT);
}
void loop()
{
digitalWrite(LED, HIGH);
delay(1000);
digitalWrite(LED, LOW);
delay(1000);
}
```

2) LED WITH PUSH BUTTON

```
#define LED_PIN PA5
#define BUTTON_PIN PC3

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  // If button is pressed (LOW)
  if (digitalRead(BUTTON_PIN) == LOW) {

    // Blink LED 10 times
    for (int i = 0; i < 10; i++) {
      digitalWrite(LED_PIN, HIGH);
      delay(1000);
      digitalWrite(LED_PIN, LOW);
      delay(1000);
    }

    // Wait until button is released to avoid repeat triggering
    while (digitalRead(BUTTON_PIN) == LOW);
    delay(50);  // small debounce
  }
}
```

3) 7 SEGMENT

```
// 7-segment pins (common cathode)
#define SEG_A PD0
#define SEG_B PD1
#define SEG_C PD2
#define SEG_D PD3
```

```
#define SEG_E PD4
#define SEG_F PD5
#define SEG_G PD6


#define ena1 PC2
#define ena2 PC3
#define ena3 PC4
#define ena4 PC5
// Array of segment pins for easy looping
int segments[] = {SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G};

// Digit patterns for common-cathode display
// 1 = ON, 0 = OFF
byte digits[10][7] = {
  // a, b, c, d, e, f, g
  {0,0,0,0,0,0,0}, // 0 dummy (not used)
  {0,1,1,0,0,0,0}, // 1
  {1,1,0,1,1,0,1}, // 2
  {1,1,1,1,0,0,1}, // 3
  {0,1,1,0,0,1,1}, // 4
  {1,0,1,1,0,1,1}, // 5
  {1,0,1,1,1,1,1}, // 6
  {1,1,1,0,0,0,0}, // 7
  {1,1,1,1,1,1,1}, // 8
  {1,1,1,1,0,1,1}  // 9
};

void setup() {
  pinMode(ena1,OUTPUT);
pinMode(ena2,OUTPUT);
pinMode(ena3,OUTPUT);
pinMode(ena4,OUTPUT);

digitalWrite(ena1,HIGH);
digitalWrite(ena2,HIGH);
digitalWrite(ena3,HIGH);
digitalWrite(ena4,HIGH);

  // Set all segment pins as OUTPUT
  for (int i = 0; i < 7; i++) {
    pinMode(segments[i], OUTPUT);

  }
}

// Function to display a single digit
void displayDigit(int num) {
  for (int i = 0; i < 7; i++) {
    digitalWrite(segments[i], digits[num][i] ? HIGH : LOW);
  }
}

void loop() {
  // Show digits 1 to 9 one by one
  for (int i = 1; i <= 9; i++) {
    displayDigit(i);
    delay(1000);  // 1 second per number
  }
}
```

4) 4X4 MATRIX KEYPAD

```
#define ROW0 PB0
#define ROW1 PB1
#define ROW2 PB2
#define ROW3 PB3
#define COL0 PB4
#define COL1 PB5
#define COL2 PB6
#define COL3 PB7
#define SEG_A PD0
#define SEG_B PD1
#define SEG_C PD2
#define SEG_D PD3
#define SEG_E PD4
#define SEG_F PD5
#define SEG_G PD6
#define CTRL0 PC0
#define CTRL1 PC1
#define CTRL2 PC2
#define CTRL3 PC3
const byte segDigits[10][7] = {
{1,1,1,1,1,1,0},
{0,1,1,0,0,0,0},
{1,1,0,1,1,0,1},
{1,1,1,1,0,0,1},
{0,1,1,0,0,1,1},
{1,0,1,1,0,1,1},
{1,0,1,1,1,1,1},
{1,1,1,0,0,0,0},
{1,1,1,1,1,1,1},
{1,1,1,1,0,1,1}
};
char lastKey = '0';
void setup() {
Serial.begin(9600);
pinMode(ROW0, OUTPUT); digitalWrite(ROW0, HIGH);
pinMode(ROW1, OUTPUT); digitalWrite(ROW1, HIGH);
pinMode(ROW2, OUTPUT); digitalWrite(ROW2, HIGH);
pinMode(ROW3, OUTPUT); digitalWrite(ROW3, HIGH);
pinMode(COL0, INPUT_PULLUP);
pinMode(COL1, INPUT_PULLUP);
pinMode(COL2, INPUT_PULLUP);
pinMode(COL3, INPUT_PULLUP);
pinMode(SEG_A, OUTPUT); digitalWrite(SEG_A, HIGH);
pinMode(SEG_B, OUTPUT); digitalWrite(SEG_B, HIGH);
pinMode(SEG_C, OUTPUT); digitalWrite(SEG_C, HIGH);
pinMode(SEG_D, OUTPUT); digitalWrite(SEG_D, HIGH);
pinMode(SEG_E, OUTPUT); digitalWrite(SEG_E, HIGH);
pinMode(SEG_F, OUTPUT); digitalWrite(SEG_F, HIGH);
pinMode(SEG_G, OUTPUT); digitalWrite(SEG_G, HIGH);
pinMode(CTRL0, OUTPUT); digitalWrite(CTRL0, HIGH);
pinMode(CTRL1, OUTPUT); digitalWrite(CTRL1, HIGH);
pinMode(CTRL2, OUTPUT); digitalWrite(CTRL2, HIGH);
pinMode(CTRL3, OUTPUT); digitalWrite(CTRL3, HIGH);
displayDigit('0');
}
char scanKeypad() {
char keys[4][4] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};
digitalWrite(ROW0, LOW);
if (digitalRead(COL0) == LOW) { digitalWrite(ROW0, HIGH); delay(50); return keys[0][0]; }
if (digitalRead(COL1) == LOW) { digitalWrite(ROW0, HIGH); delay(50); return keys[0][1]; }
if (digitalRead(COL2) == LOW) { digitalWrite(ROW0, HIGH); delay(50); return keys[0][2]; }
if (digitalRead(COL3) == LOW) { digitalWrite(ROW0, HIGH); delay(50); return keys[0][3]; }
digitalWrite(ROW0, HIGH);
digitalWrite(ROW1, LOW);
if (digitalRead(COL0) == LOW) { digitalWrite(ROW1, HIGH); delay(50); return keys[1][0]; }
if (digitalRead(COL1) == LOW) { digitalWrite(ROW1, HIGH); delay(50); return keys[1][1]; }
if (digitalRead(COL2) == LOW) { digitalWrite(ROW1, HIGH); delay(50); return keys[1][2]; }
if (digitalRead(COL3) == LOW) { digitalWrite(ROW1, HIGH); delay(50); return keys[1][3]; }
digitalWrite(ROW1, HIGH);
digitalWrite(ROW2, LOW);
if (digitalRead(COL0) == LOW) { digitalWrite(ROW2, HIGH); delay(50); return keys[2][0]; }
if (digitalRead(COL1) == LOW) { digitalWrite(ROW2, HIGH); delay(50); return keys[2][1]; }
if (digitalRead(COL2) == LOW) { digitalWrite(ROW2, HIGH); delay(50); return keys[2][2]; }
if (digitalRead(COL3) == LOW) { digitalWrite(ROW2, HIGH); delay(50); return keys[2][3]; }
digitalWrite(ROW2, HIGH);
digitalWrite(ROW3, LOW);
if (digitalRead(COL0) == LOW) { digitalWrite(ROW3, HIGH); delay(50); return keys[3][0]; }
if (digitalRead(COL1) == LOW) { digitalWrite(ROW3, HIGH); delay(50); return keys[3][1]; }
if (digitalRead(COL2) == LOW) { digitalWrite(ROW3, HIGH); delay(50); return keys[3][2]; }
if (digitalRead(COL3) == LOW) { digitalWrite(ROW3, HIGH); delay(50); return keys[3][3]; }
digitalWrite(ROW3, HIGH);
return '\0';
}
void displayDigit(char key) {
if (key >= '0' && key <= '9') {
int digit = key - '0';
digitalWrite(SEG_A, !segDigits[digit][0]);
digitalWrite(SEG_B, !segDigits[digit][1]);
digitalWrite(SEG_C, !segDigits[digit][2]);
digitalWrite(SEG_D, !segDigits[digit][3]);
digitalWrite(SEG_E, !segDigits[digit][4]);
digitalWrite(SEG_F, !segDigits[digit][5]);
digitalWrite(SEG_G, !segDigits[digit][6]);
} else {
digitalWrite(SEG_A, HIGH);
digitalWrite(SEG_B, HIGH);
digitalWrite(SEG_C, HIGH);
digitalWrite(SEG_D, HIGH);
digitalWrite(SEG_E, HIGH);
digitalWrite(SEG_F, HIGH);
digitalWrite(SEG_G, HIGH);
}
}
void loop() {
char key = scanKeypad();
if (key != '\0')

 {
lastKey = key;
displayDigit(key);
}
else {
displayDigit(lastKey);
}
delay(50);
}
```

5) DC MOTOR

```
#define IN1 PA0
#define IN2 PA1
void setup() {
pinMode(IN1, OUTPUT);
pinMode(IN2, OUTPUT);
}
void loop() {
 digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
delay(2000);

digitalWrite(IN1, LOW);
digitalWrite(IN2, LOW);
delay(1000);

 digitalWrite(IN1, LOW);
digitalWrite(IN2, HIGH);
 delay(2000);

digitalWrite(IN1, LOW);
```

```
 digitalWrite(IN2, LOW);
  delay(1000);
 }
```

6) STEPPER MOTOR

```
// 4 wires of stepper motor connected to STM32 pins
int IN1 = PD0;
int IN2 = PD1;
int IN3 = PD2;
int IN4 = PD3;

// Full-step sequence (clockwise)
int stepSequence[4][4] = {
  {1, 0, 0, 1},
  {1, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 1}
};

int stepDelay = 5;  // delay between steps (ms) - controls speed

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  // Turn all coils off initially
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

// Function to rotate clockwise for given steps
void stepClockwise(int steps) {
  for (int i = 0; i < steps; i++) {
    for (int step = 0; step < 4; step++) {
      digitalWrite(IN1, stepSequence[step][0]);
      digitalWrite(IN2, stepSequence[step][1]);
      digitalWrite(IN3, stepSequence[step][2]);
      digitalWrite(IN4, stepSequence[step][3]);
      delay(stepDelay);
    }
  }
}

void loop() {
  stepClockwise(200);  // rotate 200 steps clockwise
  delay(1000);         // 1 second wait
}
```

7) DC MOTOR WITH PWM

```
#define PWM_PIN Pb2
#define DIR_PIN1 PB7
#define DIR_PIN2 PB8
void setup() {
pinMode(PWM_PIN, OUTPUT);
pinMode(DIR_PIN1, OUTPUT);
pinMode(DIR_PIN2, OUTPUT);

digitalWrite(DIR_PIN1, HIGH);
digitalWrite(DIR_PIN2, LOW);
}
void loop() {

for (int duty = 0; duty <= 255; duty++) {
analogWrite(PWM_PIN, duty);
delay(10);
}
delay(1000);

for (int duty = 255; duty >= 0; duty--) {
analogWrite(PWM_PIN, duty);
delay(10);
}
analogWrite(PWM_PIN, 0);
delay(1000);

digitalWrite(DIR_PIN1, LOW);
digitalWrite(DIR_PIN2, HIGH);

for (int duty = 0; duty <= 255; duty++) {
analogWrite(PWM_PIN, duty);
delay(10);
}
delay(1000);
analogWrite(PWM_PIN, 0);
delay(1000);
}
```

8) SEMAPHORE:

```
#include <STM32FreeRTOS.h>
#include <semphr.h>


SemaphoreHandle_t xLEDSemaphore;

void Task1(void *pvParameters);
void Task2(void *pvParameters);

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

```
    xLEDSemaphore = xSemaphoreCreateBinary();
  xSemaphoreGive(xLEDSemaphore);  // Make available initially

    xTaskCreate(Task1, "LED_1s", 128, NULL, 2, NULL);
  xTaskCreate(Task2, "LED_2s", 128, NULL, 1, NULL);
  vTaskStartScheduler();
}

void loop() {
 }

void Task1(void *pvParameters) {
  for (;;) {
    if (xSemaphoreTake(xLEDSemaphore, portMAX_DELAY) == pdTRUE) {
      Serial.println("Task 1: LED ON (1 sec)");
      digitalWrite(LED_PIN, HIGH);
      vTaskDelay(1000 / portTICK_PERIOD_MS);
      digitalWrite(LED_PIN, LOW);
      Serial.println("Task 1: LED OFF");
      xSemaphoreGive(xLEDSemaphore);
    }
    vTaskDelay(3000 / portTICK_PERIOD_MS);
  }
}

void Task2(void *pvParameters) {
  for (;;) {
    if (xSemaphoreTake(xLEDSemaphore, portMAX_DELAY) == pdTRUE) {
      Serial.println("Task 2: LED ON (2 sec)");
      digitalWrite(LED_PIN, HIGH);
      vTaskDelay(2000 / portTICK_PERIOD_MS);
      digitalWrite(LED_PIN, LOW);
      Serial.println("Task 2: LED OFF");
      xSemaphoreGive(xLEDSemaphore);
    }
    vTaskDelay(5000 / portTICK_PERIOD_MS);
  }
}
```

9) WITHOUT SEMAPHORE:

```
#include <STM32FreeRTOS.h>
#define LED1 PB0
#define LED2 PB1
void Task_LED1(void *pvParameters);
void Task_LED2(void *pvParameters);

void setup() {

  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  xTaskCreate(Task_LED1, "LED1 Task", 128, NULL, 1, NULL);
  xTaskCreate(Task_LED2, "LED2 Task", 128, NULL, 1, NULL);
  vTaskStartScheduler();
}

void loop() {
  }

void Task_LED1(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    digitalWrite(LED1, HIGH);
    vTaskDelay(500 / portTICK_PERIOD_MS);
    digitalWrite(LED1, LOW);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void Task_LED2(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    digitalWrite(LED2, HIGH);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    digitalWrite(LED2, LOW);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

KEIL CODES:

10) LED:

```
#include "stm32f407xx.h"

void delay_ms(uint32_t ms);

int main(void)
{
    // 1. Enable clock for GPIOD
    RCC->AHB1ENR |= (1 << 3);   // GPIOD clock enable

    // 2. Configure PD5 as OUTPUT
    GPIOD->MODER &= ~(3 << (5 * 2));   // Clear mode bits
    GPIOD->MODER |=  (1 << (5 * 2));   // 01 = output mode

    while(1)
    {
        // LED ON
        GPIOD->BSRR = (1 << 5);
        delay_ms(500);
        // LED OFF
        GPIOD->BSRR = (1 << 5) << 16;
        delay_ms(500);
    }
}

void delay_ms(uint32_t ms)
{
    while(ms--)
    {
        int count = 16000;
        while(count--)
        {
            __NOP();
        }
    }
}
```

11) LED PUSH BUTTON (KEIL) :

```
#include "stm32f407xx.h"

void delay_ms(uint32_t ms);

int main(void)
{
    // 1. Enable clock for GPIOD & GPIOE
    RCC->AHB1ENR |= (1 << 3);   // GPIOD clock enable
    RCC->AHB1ENR |= (1 << 4);   // GPIOE clock enable

    // 2. Configure PD5 as OUTPUT
    GPIOD->MODER &= ~(3 << (5 * 2));   // Clear mode
    GPIOD->MODER |=  (1 << (5 * 2));   // 01 = output

    // 3. Configure PE4 as INPUT
    GPIOE->MODER &= ~(3 << (4 * 2));   // 00 = input

    // 4. Optional pull-up (button active LOW)
    GPIOE->PUPDR &= ~(3 << (4 * 2));
    GPIOE->PUPDR |=  (1 << (4 * 2));   // 01 = pull-up

    while (1)
```

```
  {
    if (!(GPIOE->IDR & (1 << 4)))   // Button pressed (PE4 LOW)
    {
      GPIOD->BSRR = (1 << 5);     // LED ON (PD5)
    }
    else
    {
      GPIOD->BSRR = (1 << 5) << 16; // LED OFF (PD5)
    }
  }
}

void delay_ms(uint32_t ms)
{
  while (ms--)
  {
    int count = 16000;
    while (count--)
    {
      __NOP();
    }
  }
}
```

12) DC MOTOR (DIRECTION CONTROL) :

```
#include "stm32f407xx.h"

void delay_ms(uint32_t ms);

int main(void)
{
  // 1. Enable clock for GPIOD
  RCC->AHB1ENR |= (1 << 3);  // GPIOD clock enable

  // 2. Configure PD0 and PD1 as OUTPUT
  GPIOD->MODER &= ~(0xF << (0 * 2));   // Clear PD0 & PD1 mode bits
  GPIOD->MODER |=  (0x5 << (0 * 2));   // 01 01 = output for PD0, PD1

  while(1)
  {
    // -------- CLOCKWISE --------
    GPIOD->BSRR = (1 << 0);      // IN1 = HIGH
    GPIOD->BSRR = (1 << 1) << 16; // IN2 = LOW
    delay_ms(2000);

    // -------- STOP --------
```

```
        GPIOD->BSRR = (1 << 0) << 16; // IN1 = LOW
        GPIOD->BSRR = (1 << 1) << 16; // IN2 = LOW
        delay_ms(1000);

        // -------- ANTICLOCKWISE --------
        GPIOD->BSRR = (1 << 1);       // IN2 = HIGH
        GPIOD->BSRR = (1 << 0) << 16; // IN1 = LOW
        delay_ms(2000);

        // -------- STOP --------
        GPIOD->BSRR = (1 << 0) << 16;
        GPIOD->BSRR = (1 << 1) << 16;
        delay_ms(1000);
    }
}

void delay_ms(uint32_t ms)
{
    while(ms--)
    {
        int count = 16000;
        while(count--)
        {
            __NOP();
        }
    }
}
```