SNAKE GAME

LPC1768 ARM CORTEX MICROCONTROLLER

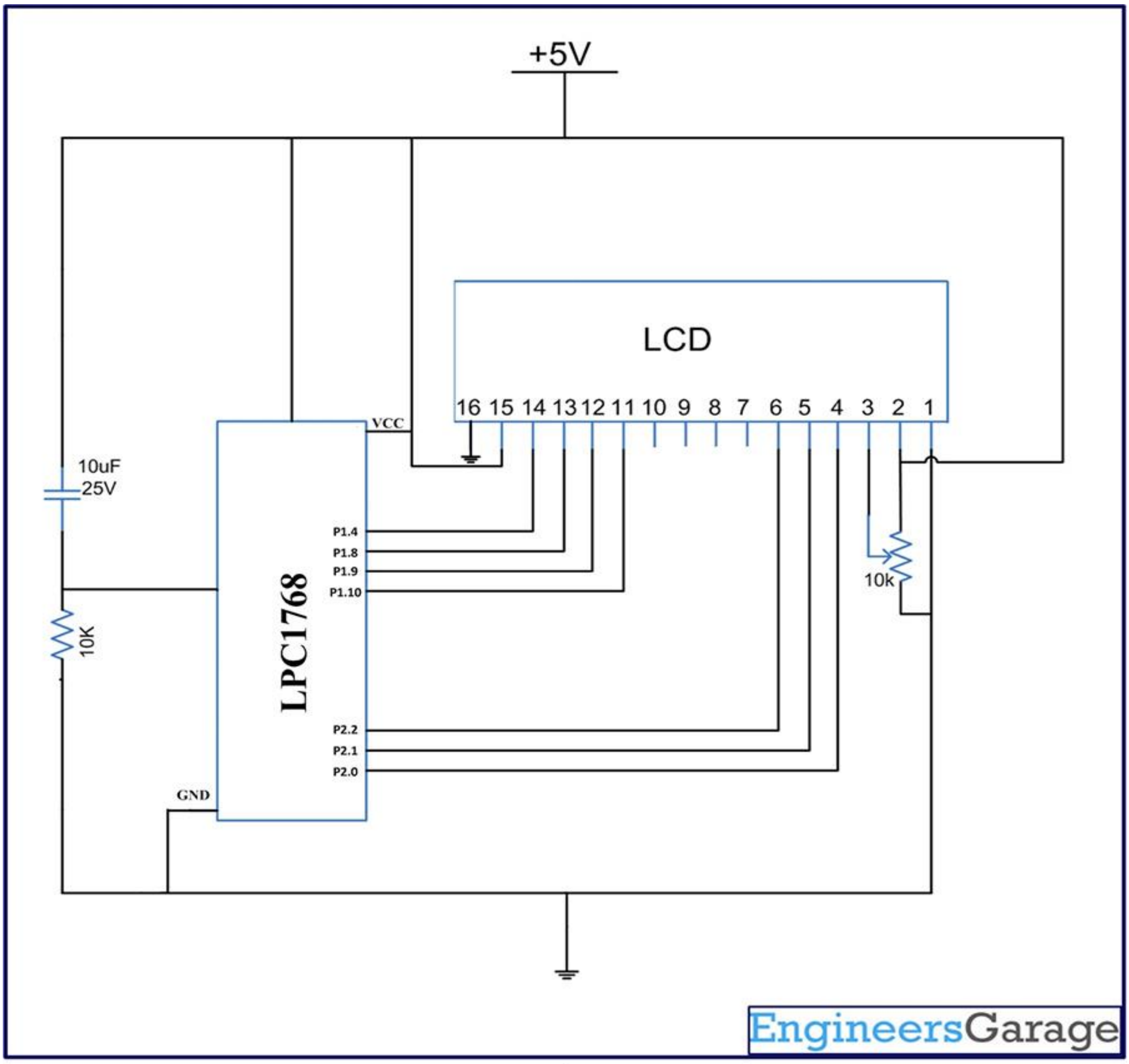
TEAM 11 – CSE D

OBJECTIVE

The main objective of creating a Snake game on the LPC1768 ARM Cortex platform is to gain experience in embedded systems programming and game development, while also creating a fun and engaging game that can be enjoyed by others.

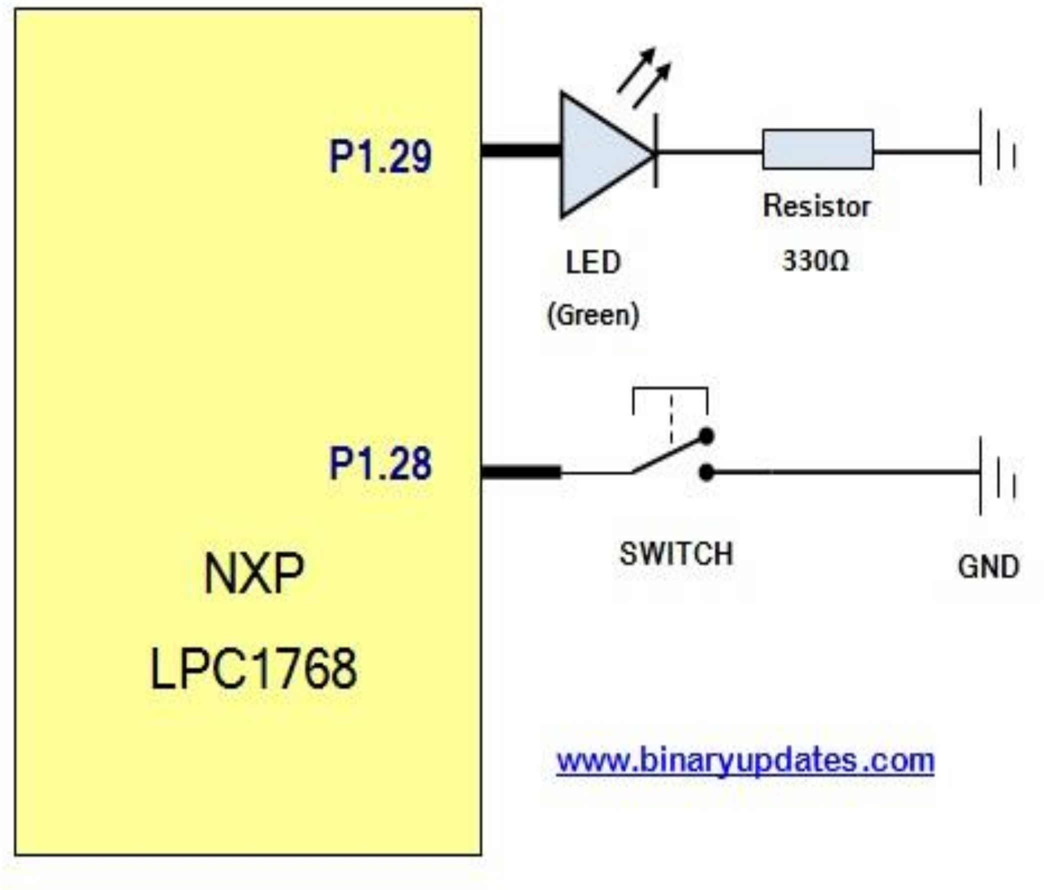
The LPC1768 ARM Cortex Snake game is a classic arcade-style game where the player controls a snake that moves around a 2D game board, eating food items and growing in length. The objective of the game is to eat as much food as possible without colliding with the walls or the snake's own body.
The game is implemented on the LPC1768 ARM Cortex microcontroller platform, which provides a range of features and peripherals to support game development. The game board is displayed on a graphical LCD screen, and the player controls the snake's movement using buttons.

BLOCK DIAGRAM



LCD

BUTTON

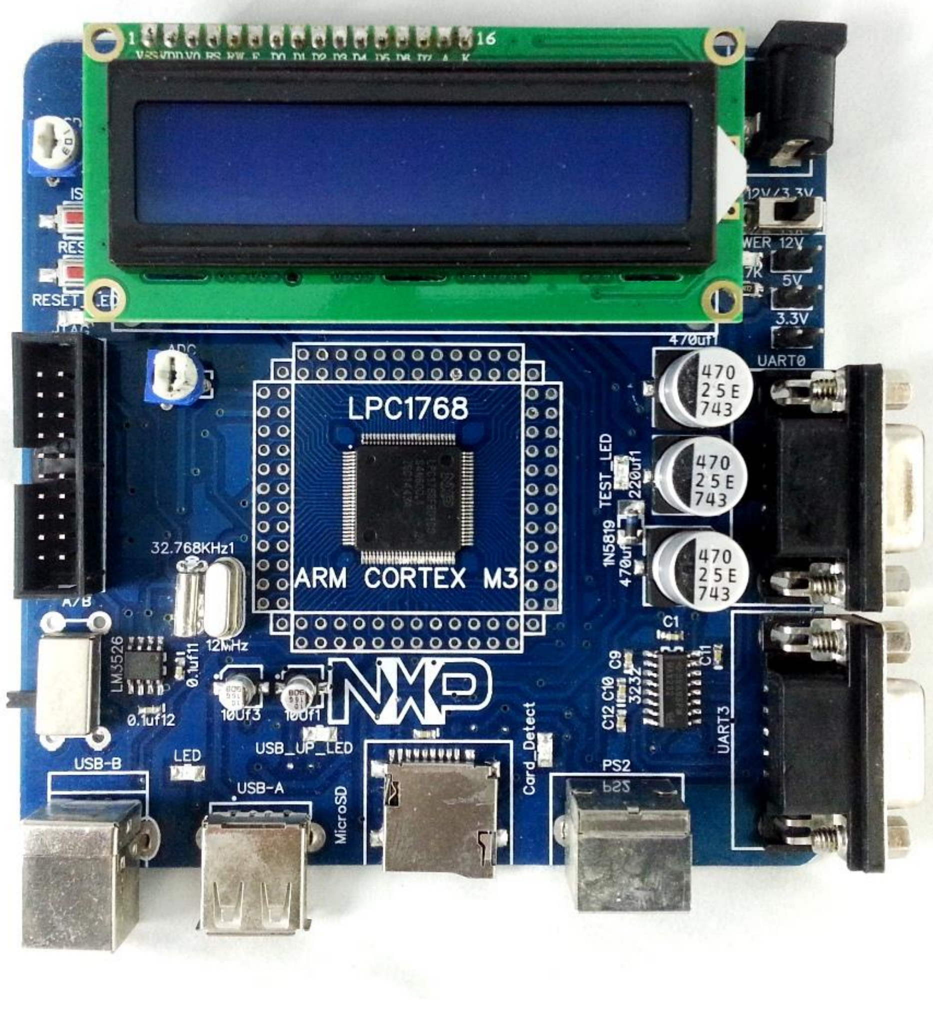


The code is written for the LPC1768 microcontroller from NXP, which is based on the ARM Cortex-M3 processor. The microcontroller is being used to implement a simple snake game, which will be displayed on an LCD module and controlled using four push buttons.

The LCD module used in this code is a standard 16x2 character LCD, which is controlled using a 6-pin interface. The interface consists of 3 control pins (RS, EN, and RW) and 4 data pins (D4-D7). The microcontroller communicates with the LCD module using a 4-bit data transfer protocol, where the upper 4 bits of each 8-bit command or data byte are transferred using pins D4-D7.

The push buttons used in this code are connected to four input pins (P2.10 - P2.13) on the LPC1768 microcontroller. Each button is connected to a different pin, and is configured to operate in a pull-up configuration. This means that when a button is not pressed, the corresponding input pin is held high by a pull-up resistor. When a button is pressed, the input pin is pulled low, indicating that the button has been pressed.

LPC1768
MICROCONTROLLER
BOARD



CODE-SNAKE GAME

```
#include <LPC17xx.h>    // Include LPC1768 header file

#include <stdlib.h>

#include <time.h>


#define SIZE 30         // Size of each block in the game

#define WIDTH 22        // Width of game field

#define HEIGHT 18       // Height of game field

#define SNAKE_SIZE 100  // Maximum size of the snake


#define LCD_RS (1 << 27)    // Register select pin

#define LCD_RW (1 << 28)    // Read/write pin

#define LCD_EN (1 << 29)    // Enable pin

#define LCD_D4 (1 << 23)    // Data pin 4

#define LCD_D5 (1 << 24)    // Data pin 5

#define LCD_D6 (1 << 25)    // Data pin 6

#define LCD_D7 (1 << 26)    // Data pin 7


#define BTN1 (1 << 16)  // Button 1 pin

#define BTN2 (1 << 17)  // Button 2 pin


typedef struct {

   int x, y;          // Coordinates of the block

} Block;


Block snake[SNAKE_SIZE]; // Snake array

Block food;            // Food block

int snake_size = 5;     // Initial size of the snake

int score = 0;         // Initial score
int dir_x = 1;          // Initial direction of the snake (right)
int dir_y = 0;

void init() {
    srand(time(NULL));  // Initialize random number generator
    snake[0].x = WIDTH / 2;     // Initialize the snake at the center
    snake[0].y = HEIGHT / 2;
    for (int i = 1; i < snake_size; i++) {   // Initialize the rest of the snake
        snake[i].x = snake[0].x - i;
        snake[i].y = snake[0].y;
    }
    food.x = rand() % WIDTH;    // Initialize the food at a random location
    food.y = rand() % HEIGHT;
}

void move() {
    int new_x = snake[0].x + dir_x; // Calculate the new head position
    int new_y = snake[0].y + dir_y;
    if (new_x < 0 || new_x >= WIDTH || new_y < 0 || new_y >= HEIGHT) {  // Check for out-of-bounds
        // Game over
    }
    for (int i = 1; i < snake_size; i++) {   // Check for collision with the body
        if (new_x == snake[i].x && new_y == snake[i].y) {
            // Game over
        }
    }
    if (new_x == food.x && new_y == food.y) { // Check for food collision
        score++;
        snake_size++;
        food.x = rand() % WIDTH;
        food.y = rand() % HEIGHT;
    }
    for (int i = snake_size - 1; i > 0; i--) { // Move the body
        snake[i].x = snake[i - 1].x;
        snake[i].y = snake[i - 1].y;
    }
    snake[0].x = new_x;
    snake[0].y = new_y;
}

void lcd_command(unsigned char cmd) {
    LPC_GPIO1->FIOPIN &= ~(LCD_RS);  // RS=0 for command mode
    LPC_GPIO1->FIOPIN &= ~(LCD_RW);  // RW=0 for write mode
        // Send high nibble
    LPC_GPIO1->FIOPIN &= ~(LCD_D7 | LCD_D6 | LCD_D5 | LCD_D4); // Clear data pins
    LPC_GPIO1->FIOPIN |= ((cmd >> 4) & 0x0F) << 23;  // Set data pins
    LPC_GPIO1->FIOPIN |= LCD_EN;    // Set enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay
    LPC_GPIO1->FIOPIN &= ~(LCD_EN);  // Clear enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay

    // Send low nibble
    LPC_GPIO1->FIOPIN &= ~(LCD_D7 | LCD_D6 | LCD_D5 | LCD_D4); // Clear data pins
    LPC_GPIO1->FIOPIN |= (cmd & 0x0F) << 23;   // Set data pins
    LPC_GPIO1->FIOPIN |= LCD_EN;    // Set enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay
    LPC_GPIO1->FIOPIN &= ~(LCD_EN);  // Clear enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay
}

void lcd_data(unsigned char data) {
    LPC_GPIO1->FIOPIN |= LCD_RS;    // RS=1 for data mode
    LPC_GPIO1->FIOPIN &= ~(LCD_RW);  // RW=0 for write mode

    // Send high nibble
    LPC_GPIO1->FIOPIN &= ~(LCD_D7 | LCD_D6 | LCD_D5 | LCD_D4); // Clear data pins
    LPC_GPIO1->FIOPIN |= ((data >> 4) & 0x0F) << 23; // Set data pins
    LPC_GPIO1->FIOPIN |= LCD_EN;    // Set enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay
    LPC_GPIO1->FIOPIN &= ~(LCD_EN);  // Clear enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay

    // Send low nibble
    LPC_GPIO1->FIOPIN &= ~(LCD_D7 | LCD_D6 | LCD_D5 | LCD_D4); // Clear data pins
    LPC_GPIO1->FIOPIN |= (data & 0x0F) << 23;   // Set data pins
    LPC_GPIO1->FIOPIN |= LCD_EN;    // Set enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay
    LPC_GPIO1->FIOPIN &= ~(LCD_EN);  // Clear enable pin
    for (volatile int i = 0; i < 100; i++);   // Delay
}

void lcd_init() {
    LPC_GPIO1->FIODIR |= LCD_RS | LCD_RW | LCD_EN | LCD_D7 | LCD_D6 | LCD_D5 | LCD_D4; // Set all pins
as output
    for (volatile int i = 0; i < 100000; i++); // Power-on delay

    lcd_command(0x28);  // 4-bit mode, 2-line display, 5x8 font
    lcd_command(0x0F);  // Display on
    lcd_command(0x01);  // Clear display
    for (volatile int i = 0; i < 100000; i++); // Clear display delay
    lcd_command(0x06);  // Entry mode set: increment cursor, no display shift
}

void init_buttons() {
    // Set P2.10 - P2.13 as input
    LPC_GPIO2->FIODIR &= ~(1 << 10);
    LPC_GPIO2->FIODIR &= ~(1 << 11);
    LPC_GPIO2->FIODIR &= ~(1 << 12);
    LPC_GPIO2->FIODIR &= ~(1 << 13);

    // Enable pull-up resistors on P2.10 - P2.13
    LPC_PINCON->PINMODE_OD2 &= ~(1 << 10);
    LPC_PINCON->PINMODE_OD2 &= ~(1 << 11);
    LPC_PINCON->PINMODE_OD2 &= ~(1 << 12);
    LPC_PINCON->PINMODE_OD2 &= ~(1 << 13);
    LPC_PINCON->PINMODE_PULLUP2 |= (1 << 10);
    LPC_PINCON->PINMODE_PULLUP2 |= (1 << 11);
    LPC_PINCON->PINMODE_PULLUP2 |= (1 << 12);
    LPC_PINCON->PINMODE_PULLUP2 |= (1 << 13);
}

int read_button(int button) {
    switch (button) {
        case 0: // Left button (P2.10)
            return !(LPC_GPIO2->FIOPIN & (1 << 10));
        case 1: // Up button (P2.11)
            return !(LPC_GPIO2->FIOPIN & (1 << 11));
        case 2: // Down button (P2.12)
            return !(LPC_GPIO2->FIOPIN & (1 << 12));
        case 3: // Right button (P2.13)
            return !(LPC_GPIO2->FIOPIN & (1 << 13));
        default:
            return 0;
    }
}

int main() {
    // Initialize LCD and buttons
    lcd_init();
    init_buttons();
    // Initialize game
    init_game();

    while (1) {
        // Update game
        update_game();

        // Display game
        display_game();

        // Check for button presses
        if (read_button(BUTTON_LEFT)) {
            change_direction(DIRECTION_LEFT);
        }
        if (read_button(BUTTON_UP)) {
            change_direction(DIRECTION_UP);
        }
        if (read_button(BUTTON_DOWN)) {
            change_direction(DIRECTION_DOWN);
        }
        if (read_button(BUTTON_RIGHT)) {
            change_direction(DIRECTION_RIGHT);
        }

        // Delay to control game speed
        for (volatile int i = 0; i < GAME_SPEED; i++);
    }

    return 0;
}
```

TEAM MEMBERS

SHREYANK SHRESTH	**59 -**	**210905360**
KUSH AGRAWAL	**58 -**	**210905358**
KUSHAGRA SARAF	**56 -**	**210905352**
ONKAR KUMAR	**27 -**	**210905161**
VAIBHAV NARANG	**25 -**	**210905148**
SAKSHAM SHARMA	**39 -**	**210905248**