ARM LABORATORY

(EC39006)

SPRING 2024-25

School of Computer Engineering

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of the UGC Act, 1956

Open-Ended Experiment Report

SUBMITTED TO:

DR. V.V.Subrahmanya Kumar Bhajana

Semester: 6th Section : CSSE2

Group Details

2228079	**VINEET GUPTA**
2228077	**VAIBHAV JAIN**
2228074	**SUMIT SAHU**
2228058	**SHAIVEE SAHU**
2228062	**SHREYA MALLIK**

EXPERIMENT :01

OBJECTIVE

The objective of this project is to develop a retro-style gaming system using an embedded microcontroller and a graphical LCD display. The system allows users to navigate a menu and play classic arcade games such as Ping Pong, Snake, and Brick Breaker.

REQUIREMENTS

● **Hardware:**

○ Microcontroller (Arduino or similar)

○ Graphical LCD (128x64 ST7920)

○ Rotary encoder with push button

○ Power supply

○ Connecting wires and resistors

● **Software:**

○ Arduino IDE

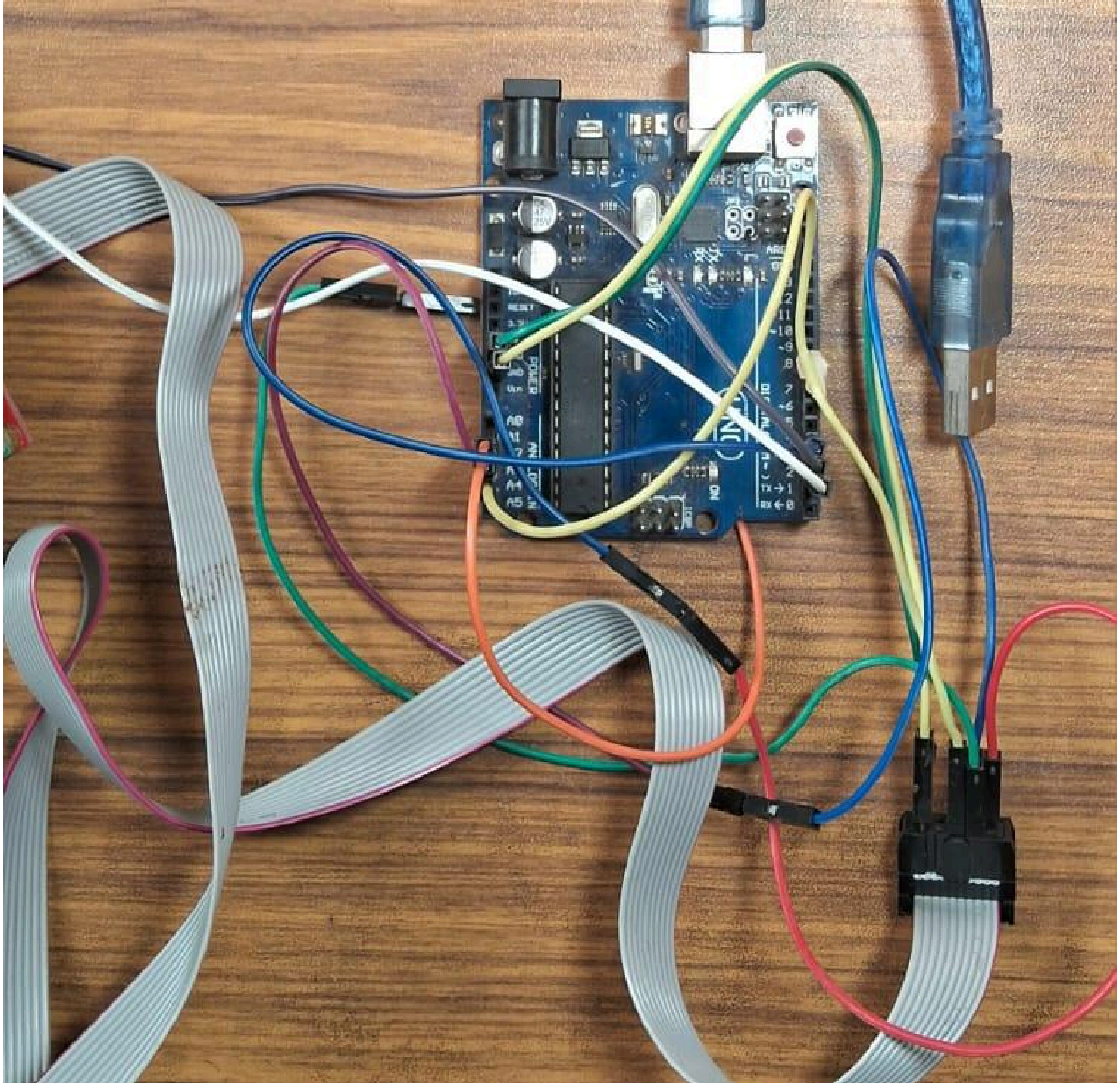
○ U8glib graphics library

PROCEDURE

1. **Setup Hardware:** Connect the graphical LCD and rotary encoder to the microcontroller.
2. **Initialize Display:** Use the U8glib library to control the display and render graphical elements.
3. **Implement Menu System:** Create a navigation menu that allows the user to select from different games.

4. **Game Implementations:** Develop and integrate game logic for Ping Pong, Snake, and Brick Breaker.
5. **User Interaction:** Use a rotary encoder for menu selection and game controls.
6. **Testing & Debugging:** Ensure smooth gameplay and responsiveness.

(RepRapDiscount Full Graphic Smart Controller)



(Uno-Based Circuit with RepRap Display Connection)

PROGRAM

```
#include "U8glib.h"

#define encoderPin1 2

#define encoderPin2 3

#define pinEncButt 4

U8GLIB_ST7920_128X64_1X u8g(17, 15, 16); // SPI Com: en=18,rw=16,di=17

volatile int lastEncoded = 0;

volatile long encoderValue = 0;

int selectedOption = 0;

enum State { MENU, PING_PONG, SNAKE, BRICK_BREAKER };

State currentState = MENU;

// Forward declarations

void drawMenu();

void playPingPong();

void playSnake();

void playBrickBreaker();
void updateEncoder();
void fadeInOutText(const char* text, int delayTime) {

for (int i = 0; i <= 255; i += 25) {

u8g.setContrast(i);

u8g.firstPage();

do {

u8g.setFont(u8g_font_6x12);

u8g.drawStr(32, 32, text); // Center the

text } while (u8g.nextPage());

delay(delayTime);

}

for (int i = 255; i >= 0; i -= 25) {

u8g.setContrast(i);

u8g.firstPage();

do {

u8g.setFont(u8g_font_6x12);

u8g.drawStr(32, 32, text); // Center the

text } while (u8g.nextPage());

delay(delayTime);
```

```
}

}

void displayTeamMembers() {

const char* teamMembers[] = {

"Made BY",

"Vaibhav Jain - 2228077",

"Shaivee Sahu - 2228058",

"Sumit Sahu - 2228074",

"Shreya Mallik - 2228062 ",

"Vineet Gupta - 2228079",

};

int numMembers = sizeof(teamMembers) /

sizeof(teamMembers[0]); for (int i = 0; i < numMembers; i++) {

u8g.firstPage();

do {

// Set a smaller font

u8g.setFont(u8g_font_chikitar);

// Calculate text width and center it horizontally

int textWidth = u8g.getStrWidth(teamMembers[i]);
int x = (128 - textWidth) / 2; // 128 is the screen width

int y = 32; // Center vertically for a single line

u8g.drawStr(x, y, teamMembers[i]);

} while (u8g.nextPage());

delay(2000); // Display each member for 2 seconds

}

}

void setup() {

pinMode(encoderPin1, INPUT);

pinMode(encoderPin2, INPUT);

pinMode(pinEncButt, INPUT);

digitalWrite(encoderPin1, HIGH);

digitalWrite(encoderPin2, HIGH);

digitalWrite(pinEncButt, HIGH);

attachInterrupt(0, updateEncoder, CHANGE);

attachInterrupt(1, updateEncoder, CHANGE);

// Startup sequence

fadeInOutText("Retro Games", 0); // Fade-in and fade-out for the title
```

```
displayTeamMembers(); // Show team member names

}

void loop() {

switch (currentState) {

case MENU:

u8g.firstPage();

do {

drawMenu();

} while (u8g.nextPage());

if (digitalRead(pinEncButt) == LOW) {

if (selectedOption == 0) currentState = PING_PONG; else if

(selectedOption == 1) currentState = SNAKE; else if

(selectedOption == 2) currentState = BRICK_BREAKER;

delay(300); // Debounce button

}

break;

case PING_PONG:

playPingPong();

currentState = MENU;
```

```
break;
case SNAKE:

playSnake();

currentState = MENU;

break;

case BRICK_BREAKER:

playBrickBreaker();

currentState = MENU;

break;

}

}

void drawMenu() {

u8g.setFont(u8g_font_6x12);

const char* menuItems[] = {"1. Ping Pong", "2. Snake", "3. Brick Breaker"};

int numItems = sizeof(menuItems) / sizeof(menuItems[0]); for (int i = 0; i <

numItems; i++) {

if (i == selectedOption) {

u8g.drawBox(0, 12 * i, 128, 12);

u8g.setColorIndex(0);
```

```
u8g.drawStr(2, 12 * (i + 1) - 2, menuItems[i]);
u8g.setColorIndex(1);

} else {

u8g.drawStr(2, 12 * (i + 1) - 2, menuItems[i]);

}

}

selectedOption = constrain(encoderValue / 4, 0, numItems - 1);

}

void playPingPong() {

int playerY = 28; // Initial player paddle position

int aiY = 28; // Initial AI paddle position

int ballX = 64, ballY = 32; // Initial ball position

int ballDX = 1, ballDY = 1; // Ball speed and direction

int playerScore = 0, aiScore = 0;

while (true) {

// Check for "back" button press

if (digitalRead(pinEncButt) == LOW) {

currentState = MENU; // Go back to the MENU
```

```
delay(300); // Debounce button

return;
}

u8g.firstPage();

do {

// Draw paddles and ball

u8g.drawBox(5, playerY, 3, 20); // Player paddle

u8g.drawBox(120, aiY, 3, 20); // AI paddle

u8g.drawDisc(ballX, ballY, 2); // Ball

// Display scores

u8g.setFont(u8g_font_6x12);

char scoreText[20];

sprintf(scoreText, "Player: %d AI: %d", playerScore, aiScore);

u8g.drawStr(20, 10, scoreText);

} while (u8g.nextPage());

// Update ball position

ballX += ballDX;

ballY += ballDY;
```

```
// Ball collision with top and bottom walls

if (ballY <= 0 || ballY >= 63) ballDY *= -1;

// Ball collision with player paddle
if (ballX <= 8 && ballY >= playerY && ballY <= playerY + 20)

{ ballDX *= -1;

ballDX = constrain(ballDX, -2, 2); // Limit ball speed

}

// Ball collision with AI paddle

if (ballX >= 120 && ballY >= aiY && ballY <= aiY + 20) {

ballDX *= -1;

ballDX = constrain(ballDX, -2, 2); // Limit ball speed

}

// Ball out of bounds (game over condition)

if (ballX <= 0) {

aiScore++;

if (aiScore >= 5) break; // AI wins

ballX = 64; ballY = 32; ballDX = 1; ballDY = 1; // Reset

ball }
```

```
if (ballX >= 128) {

playerScore++;

if (playerScore >= 5) break; // Player wins

ballX = 64; ballY = 32; ballDX = -1; ballDY = -1; // Reset ball
}

// Player paddle control

playerY = constrain(encoderValue / 4, 0, 43);

// Reset encoder value dynamically if it exceeds bounds

if (playerY == 0 && encoderValue < 0) encoderValue = 0; // At top if

(playerY == 43 && encoderValue > 172) encoderValue = 172; // At bottom //

AI paddle movement (smooth and responsive)

int aiTargetY = ballY - 10; // Target position for AI paddle

if (aiY < aiTargetY) aiY += 1;

else if (aiY > aiTargetY) aiY -= 1;

aiY = constrain(aiY, 0, 43); // Keep AI paddle in bounds

delay(30); // Adjust for smoother gameplay

}

}

void playSnake() {
```

```
int snakeX[50] = {64}, snakeY[50] = {32};

int snakeLength = 5;

int direction = 0; // 0=right, 1=down, 2=left, 3=up

int foodX = random(0, 128), foodY = random(0, 64);
bool gameOver = false;

while (!gameOver) {

// Update snake

if (digitalRead(pinEncButt) == LOW) {

currentState = MENU;

delay(300); // Debounce button

return;

}

for (int i = snakeLength - 1; i > 0; i--) {

snakeX[i] = snakeX[i - 1];

snakeY[i] = snakeY[i - 1];

}

if (direction == 0) snakeX[0]++;

if (direction == 1) snakeY[0]++;

if (direction == 2) snakeX[0]--;
if (direction == 3) snakeY[0]--;

// Check collisions

if (snakeX[0] < 0 || snakeX[0] >= 128 || snakeY[0] < 0 || snakeY[0] >= 64) gameOver =

true; for (int i = 1; i < snakeLength; i++) {

if (snakeX[0] == snakeX[i] && snakeY[0] == snakeY[i]) gameOver =

true; }

// Check food

if (snakeX[0] == foodX && snakeY[0] == foodY) {

snakeLength++;

foodX = random(0, 128);

foodY = random(0, 64);

}

// Draw

u8g.firstPage();

do {

u8g.drawBox(foodX, foodY, 2, 2);

for (int i = 0; i < snakeLength; i++) {

u8g.drawBox(snakeX[i], snakeY[i], 2, 2);
}

} while (u8g.nextPage());

delay(100);

}

}
void playBrickBreaker() {

int paddleX = 50; // Initial paddle position

int ballX = 64, ballY = 50; // Initial ball position

int ballDX = 1, ballDY = -1; // Ball movement direction

bool bricks[8][4] = {{true, true, true, true},

{true, true, true, true},

{true, true, true, true},

{true, true, true, true},

{true, true, true, true},

{true, true, true, true},

{true, true, true, true},

{true, true, true, true}};

int encoderMaxValue = 98 * 2; // Adjusted encoder range for increased sensitivity
```

```
while (true) {

// Check for "back" button press

if (digitalRead(pinEncButt) == LOW) {

currentState = MENU;

delay(300); // Debounce button

return;
}

u8g.firstPage();

do {

// Draw bricks

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

if (bricks[i][j]) u8g.drawBox(i * 16, j * 8, 14, 6);

}

}

// Draw paddle

u8g.drawBox(paddleX, 58, 30, 4);

// Draw ball

u8g.drawDisc(ballX, ballY, 2);
} while (u8g.nextPage());

// Ball movement

ballX += ballDX;

ballY += ballDY;

// Ball collisions

if (ballX <= 0 || ballX >= 127) ballDX *= -1; // Wall collision
if (ballY <= 0) ballDY *= -1; // Top wall collision

if (ballY >= 56 && ballX >= paddleX && ballX <= paddleX + 30) {

ballDY *= -1; // Paddle collision

// Add spin effect based on where the ball hits the paddle

if (ballX < paddleX + 10) ballDX = -1;

else if (ballX > paddleX + 20) ballDX = 1;

}

// Paddle control

paddleX = constrain(encoderValue / 2, 0, 98);

// Reset encoder value dynamically if it exceeds bounds

if (paddleX == 0 && encoderValue < 0) encoderValue = 0; // Left boundary

if (paddleX == 98 && encoderValue > encoderMaxValue) encoderValue = encoderMaxValue; //
Right boundary
```

```
// Brick collisions

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

if (bricks[i][j] && ballX >= i * 16 && ballX <= i * 16 + 14 &&

ballY >= j * 8 && ballY <= j * 8 + 6) {

bricks[i][j] = false; // Remove brick
ballDY *= -1; // Change ball direction

}

}

}

// Check for ball out of bounds (game over)

if (ballY > 63) break;

// Check if all bricks are destroyed

bool allBricksDestroyed = true;

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 4; j++) {

if (bricks[i][j]) {

allBricksDestroyed = false;

break;
}

}

if (!allBricksDestroyed) break;

}

if (allBricksDestroyed) break; // Player wins

delay(15); // Adjust for smoother gameplay

}

// Display "Game Over" or "You Win" message

u8g.firstPage();

do {

u8g.setFont(u8g_font_6x12);

if (ballY > 63) u8g.drawStr(30, 30, "Game Over!");

else u8g.drawStr(30, 30, "You Win!");

} while (u8g.nextPage());

delay(2000); // Pause before returning to menu

}

void updateEncoder() {

int MSB = digitalRead(encoderPin1);
int LSB = digitalRead(encoderPin2);

int encoded = (MSB << 1) | LSB;

int sum = (lastEncoded << 2) | encoded;

if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue++;

if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue--;

encoderValue = constrain(encoderValue, 0, 400);

lastEncoded = encoded;
}
```

CONCLUSION

This project successfully implements a retro gaming system on an embedded platform. The use of a graphical LCD and rotary encoder allows for a simple yet engaging user experience. The implementation of classic games like Ping Pong, Snake, and Brick Breaker demonstrates effective use of embedded programming concepts, including display handling, user input processing, and game logic. Future improvements could include adding more games, improving graphics, and refining gameplay mechanics.