1. Opis urządzenia

Gra Tetris na płytce STM32, wyświetlana na wbudowanym w płytkę wyświetlaczu LCD, sterowana przez UART oraz używając fizycznego kontrolera podłączanego zarobionym prostym kablem RJ12.

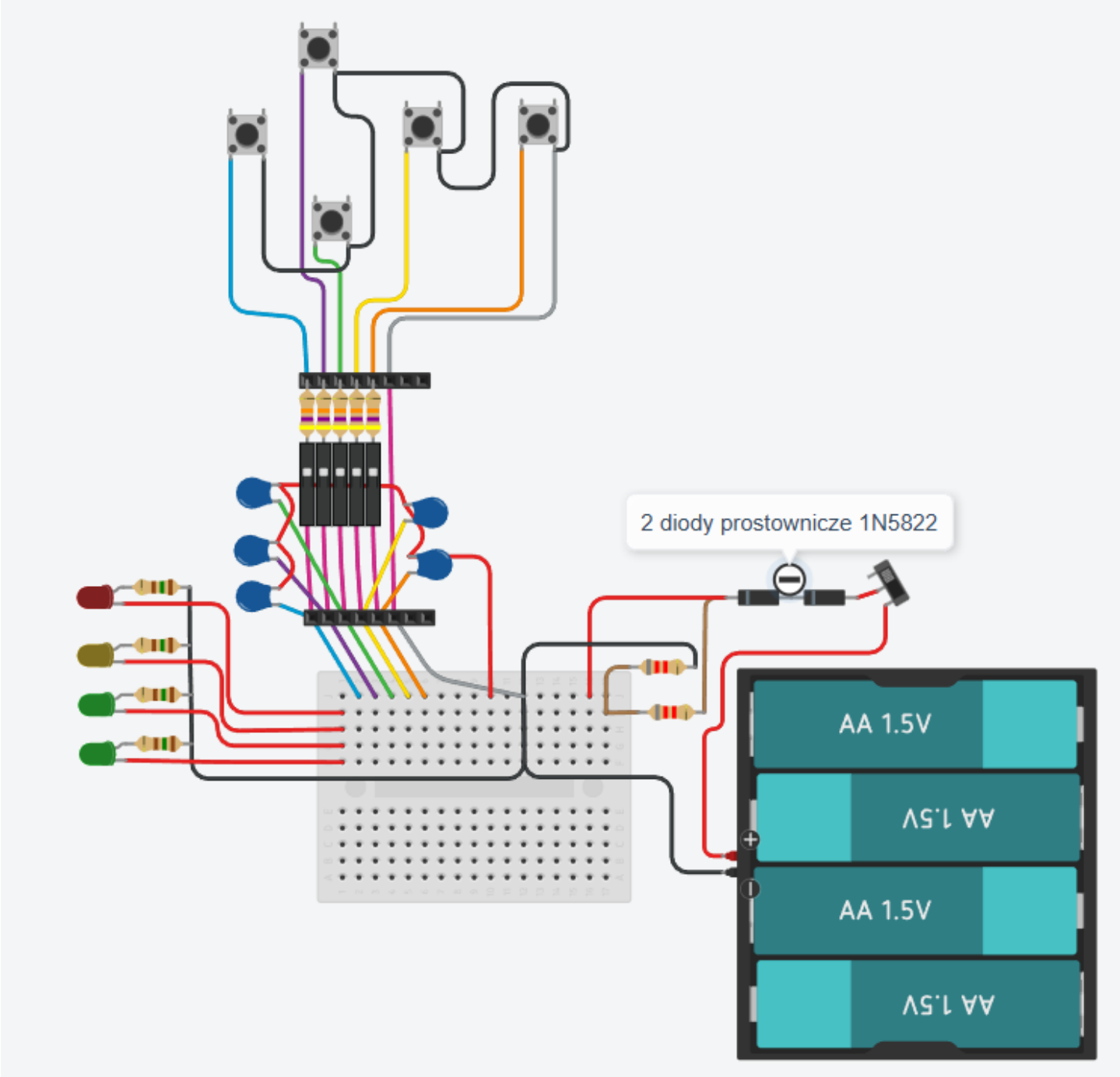
2. Elementy wybrane do budowy urządzenia

Płytka STM32F429I-DISK1,
Przyciski monostabilne,
Gniazda Keystone Jack RJ45,
Przewód RJ12,
Koszyczek na 4 baterie AA,
4 Baterie AA,
Przełącznik suwakowy,
Diody LED, różne kolory,
Diody prostownicze 1N5822,
Rezystory 8,2kOhm,
Rezystory 150Ohm,
Rezystory 47kOhm,
Kondensatory ceramiczne 100nF
Przewody połączeniowe miedziane,
Elementy obudowy konsoli wytworzone z użyciem drukarek 3D,
Elementy obudowy kontrolera wytworzone z użyciem drukarek 3D.

3. Mechanika urządzenia

Na urządzeniu uruchamiana jest gra Tetris wzorowana na projektach offpic/TETRIS-STM32 oraz KenKenMkIISR/picotetris, do wyświetlania używana jest biblioteka dla kontrolera wyświetlacza ILI9341 afiskon/stm32-ili9341. Kod gry przeszedł gruntowne zmiany i udoskonalenia a największym z nich jest odejście od wyświetlania używając znaków i tablic znakowych do poszczególnych pikseli co znacząco wpływa na każdy element gry.
Do Konsoli używając kabla RJ12 podłączany jest Kontroler z 5 przyciskami, Start, Góra, Dół, Lewo, Prawo. Wszystkie niezbędne do gry, możliwe jest również korzystanie z portu szeregowego wysyłając polecenia na port UART używając klawiszy E, W, S, A, D
Konsola zasilana jest 4 bateriami AA podłączonymi szeregowo po dostosowaniu napięcia przez 2 diody prostownicze.
Napięcie zasilania jest monitorowane i stan baterii jest wyświetlany na diodach LED przy każdym włączeniu konsoli.

4. Schemat elektroniczny urządzenia



Płytka stykowa jest użyta jako zamiennik płytki STM32.

5. Oprogramowanie sterujące

```
void updateLEDs(void) {
  uint32_t sum = 0;
  for (int i = 0; i < 10; i++) {
    HAL_ADC_Start(&hadc3);
    HAL_ADC_PollForConversion(&hadc3, HAL_MAX_DELAY);
    sum += HAL_ADC_GetValue(&hadc3);
    HAL_Delay(1);
  }
  uint32_t adcValue = sum / 10;

  float battV = adcValue * 0.00161;         // V_in = ADC * (3.3/4095) * (R1+R2)/R2

  if (battV > 5.4) {                        // 100% // >3350 // PG3 PG2 PG15 PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 1);
  } else if (battV > 4.9) {                 // 75% // 3043-3350 // PG2 PG15 PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  } else if (battV > 4.4) {                 // 50% // 2730-3043 // PG15 PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  } else if (battV > 3.9) {                 // 25% // 2418-2730 // PG14
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  } else {
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 0);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
  }

  HAL_Delay(4000);

  HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 0);
  HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
  HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
  HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
}
```

Fragment kody odpowiadający za monitorowanie i wyświetlanie stanu baterii

```
void clearscreen(void)
{
    ILI9341_FillScreen(COLOR_BG);
}
void gameinit(void)
{
    highscore = 0;
    score = 0;
    total_lines = 0;
}
void gameinit2(void)
{
    score = 0;
    level = 0;
    lines = 0;
    total_lines = 0;
    fallspeed = 25;
    gamestatus = 1;
    clearscreen();
        //GAME AREA
    uint16_t board_frame_x = BOARD_X_OFFSET - 1;
    uint16_t board_frame_y = BOARD_Y_OFFSET - 1;
    uint16_t board_frame_w = BOARD_WIDTH * BLOCK_SIZE + 2; // Width including border pixels
    uint16_t board_frame_h = BOARD_HEIGHT * BLOCK_SIZE + 2; // Height including border pixels
        // Top line
    ILI9341_FillRectangle(board_frame_x, board_frame_y, board_frame_w, 1, COLOR_FRAME);
        // Bottom line
    ILI9341_FillRectangle(board_frame_x, board_frame_y + board_frame_h - 1, board_frame_w, 1, COLOR_FRAME);
        // Left line
    ILI9341_FillRectangle(board_frame_x, board_frame_y, 1, board_frame_h, COLOR_FRAME);
        // Right line
    ILI9341_FillRectangle(board_frame_x + board_frame_w - 1, board_frame_y, 1, board_frame_h, COLOR_FRAME);


        //SCORE AREA
    uint16_t score_frame_x = SCORE_X_OFFSET - 1;
    uint16_t score_frame_y = SCORE_Y_OFFSET - 1;
    uint16_t score_frame_w = 80;
    uint16_t score_frame_h = 5 * (Font_7x10.height + 2) + 4;
        // Top
    ILI9341_FillRectangle(score_frame_x, score_frame_y, score_frame_w, 1, COLOR_FRAME);
        // Bottom
    ILI9341_FillRectangle(score_frame_x, score_frame_y + score_frame_h - 1, score_frame_w, 1, COLOR_FRAME);
        // Left
    ILI9341_FillRectangle(score_frame_x, score_frame_y, 1, score_frame_h, COLOR_FRAME);
        // Right
    ILI9341_FillRectangle(score_frame_x + score_frame_w - 1, score_frame_y, 1, score_frame_h, COLOR_FRAME);
```

Fragment kodu odpowiedzialnego za najniższe poziomy logiki gry, tworzenie planszy do gry, zerowanie i ustawianie najważniejszych zmiennych.

```
void moveblock(void)
{
    _Block tempblock;
    const _Block *blockp;
        // Player Input
        // Rotate
    if (uart_up)
    {
        uart_up = 0;
        eraseblock();

        uint8_t next_angle = (blockangle + 1) % 4;
        if (falling.rot == 1 && next_angle > 1) next_angle = 0;
        if (falling.rot == 0) next_angle = 0;

        if (blockangle == falling.rot)
        {
            blockp = &block[blockno];
            tempblock = *blockp;
        } else {
            tempblock.x1 = -falling.y1; tempblock.y1 = falling.x1;
            tempblock.x2 = -falling.y2; tempblock.y2 = falling.x2;
            tempblock.x3 = -falling.y3; tempblock.y3 = falling.x3;
            tempblock.color = falling.color;
            tempblock.rot = falling.rot;
        }
            // Check rotate
        if (check(&tempblock, blockx, blocky) == 0)
        {
            falling = tempblock;
            blockangle = (blockangle == falling.rot) ? 0 : blockangle + 1;
        }
        putblock();
    }
        // Move Right
    else if (uart_right) {
        uart_right = 0;
        eraseblock();
        if (check(&falling, blockx + 1, blocky) == 0)
        {
            blockx++;
        }
        putblock();
    }
        // Move Left
```

Fragment kodu odpowiedzialnego za ruch klocków na planszy gry

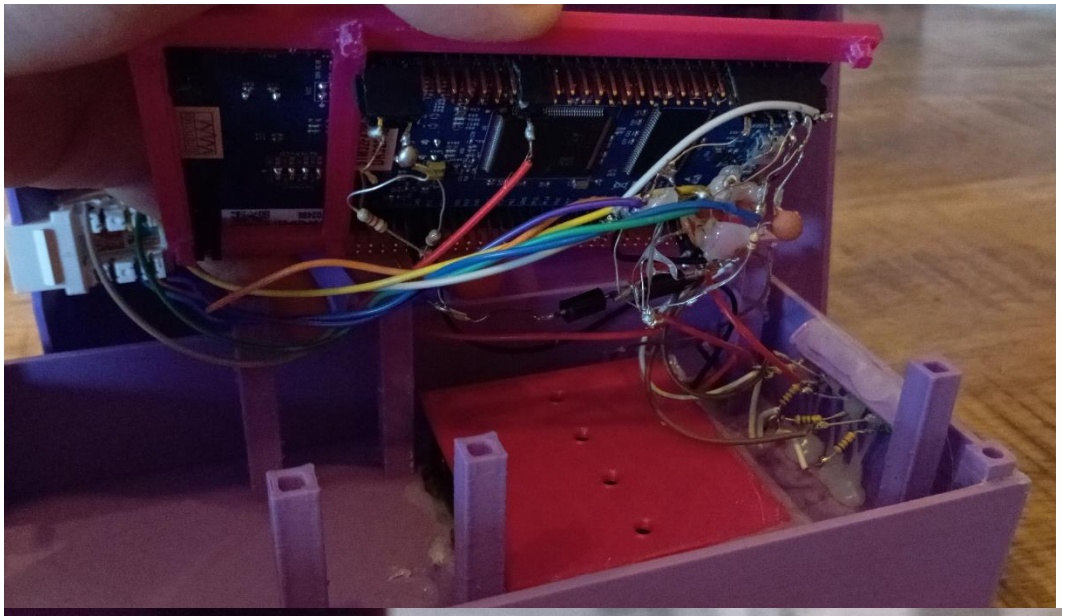
```
}
void game(void)
{
    gameinit2();

    while (gamestatus != 6)
    {
        switch (gamestatus)
        {
        case 1: // Initialize New Level
            gameinit3();
            displaylevel();
            gamestatus = 2;
            break;

        case 2: // Spawn New Block
            if (newblock() != 0)
            {
                gamestatus = 5;
            } else {
                gamestatus = 3;
            }
            break;

        case 3: // Block Falling, Input Check
            // Game tick delay happens here
            HAL_Delay(16); // ~60Hz loop speed

            show();          // Redraw changed parts of the board
            displayscore(); // Update score display
            eraseblock();    // Erase block at current position
            moveblock();     // Check input, apply natural fall, check landing
            putblock();      // Draw block at new position

            break;

        case 4: // Block Landed, Check for Line Clears
            show();
            linecheck(); // Check lines, clear them, update score/lines
            break;
        case 5: // Game Over
            gameover();
            gamestatus = 6;
            break;
        default:
            gamestatus = 6;
            break;
        }
```

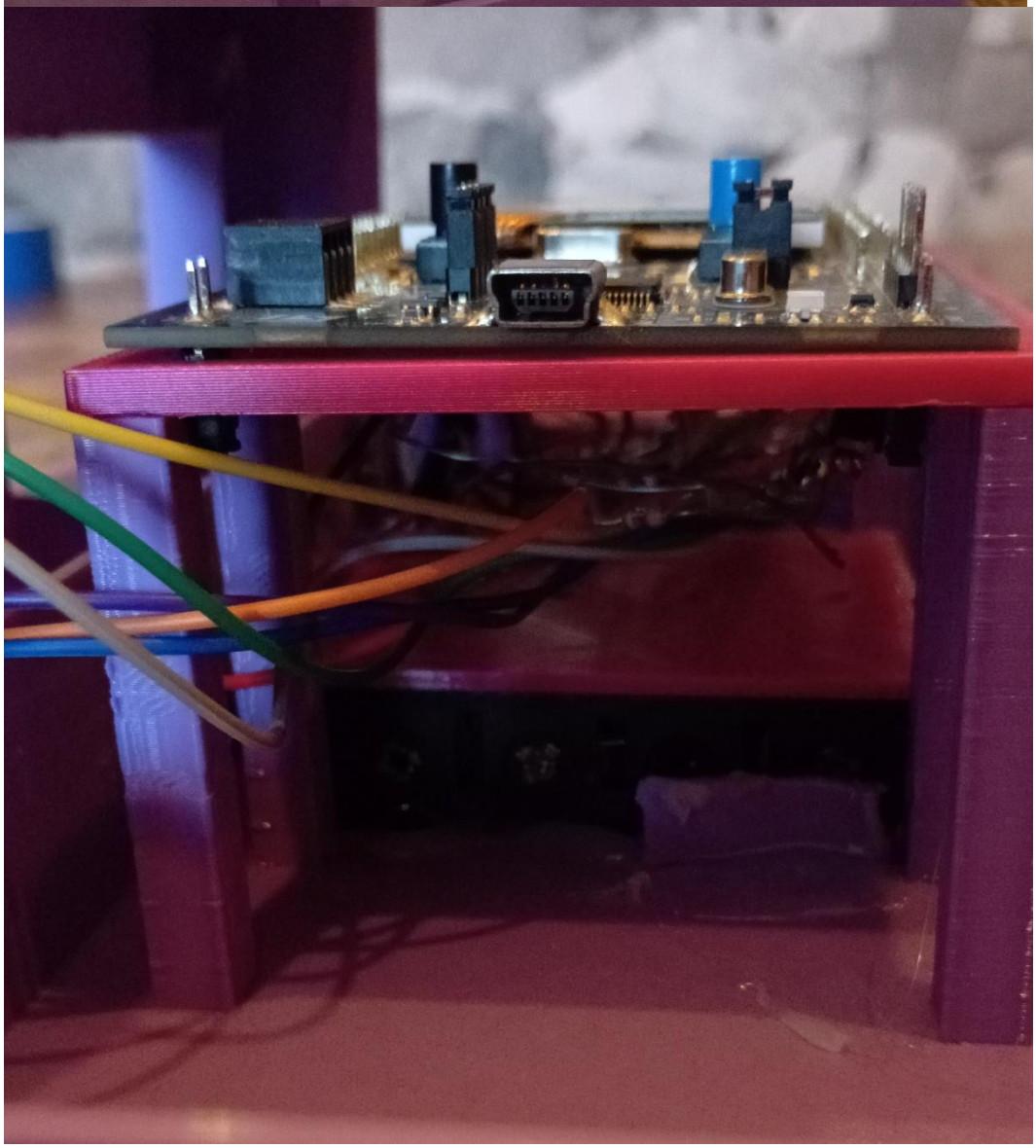
Fragment kodu odpowiedzialnego za główną pętle gry, tutaj wywoływane są wszystkie najważniejsze funkcje z użyciem zmiennej gamestatus.

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    static uint32_t last_interrupt_time_up = 0;
    static uint32_t last_interrupt_time_left = 0;
    static uint32_t last_interrupt_time_right = 0;
    static uint32_t last_interrupt_time_down = 0;
    static uint32_t last_interrupt_time_start = 0;
    uint32_t current_time = HAL_GetTick();
    const uint32_t debounce_delay = 50;

    switch (GPIO_Pin)
    {
        case BTN_UP_Pin: // PC9
            if ((current_time - last_interrupt_time_up) > debounce_delay)
            {
                last_interrupt_time_up = current_time;
                uart_up = 1;
            }
            break;
        case BTN_LEFT_Pin: // PG5
            if ((current_time - last_interrupt_time_left) > debounce_delay)
            {
                last_interrupt_time_left = current_time;
                uart_left = 1;
            }
            break;
        case BTN_RIGHT_Pin: // PG6
            if ((current_time - last_interrupt_time_right) > debounce_delay)
            {
                last_interrupt_time_right = current_time;
                uart_right = 1;
            }
            break;
        case BTN_DOWN_Pin: // PG7
            if ((current_time - last_interrupt_time_down) > debounce_delay)
            {
                last_interrupt_time_down = current_time;
                uart_down = 1;
            }
            break;
        case BTN_START_Pin: // PG8
            if ((current_time - last_interrupt_time_start) > debounce_delay)
            {
                last_interrupt_time_start = current_time;

                if (gamestatus == 3 || gamestatus == 2) {
                    uart_up = 1;
                } else {
                    uart_start = 1;
                }
            }
            break;
        default:
```

Fragment kodu odpowiedzialnego za obsługę fizycznych klawiszy, w tym użycie klawisza start jako obrót kiedy gra jest uruchomiona.

6. Zdjęcia opracowanego urządzenia






TETRIS

Obudowa Konsoli została stworzona w programie Autodesk Fusion wokół modelu płytki https://grabcad.com/library/stm32f429discovery-1
Do modelowania wcięć na moduł Keystone wykorzystany został model: https://www.l-com.com/ethernet-category-3-keystone-jack-110-rj12-eia568-usoc-white