```
index.html:
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Редактор логических схем</title>
    <link rel="stylesheet" href="css/styles.css">
</head>
<body>
    <div id="app">
        <div id="menu">
            <button class="menu-btn" id="btn-new">📄 Новый</button>
            <button class="menu-btn" id="btn-save">💾 Сохранить</button>
            <button class="menu-btn" id="btn-load">📂 Загрузить</button>
            <button class="menu-btn" id="btn-generate-code">🧩 Код</button>
            <button class="menu-btn" id="btn-project-settings">⚙️ Свойства проекта</button>
            <div class="menu-separator"></div>
            <div class="zoom-controls">
                <button class="menu-btn zoom-btn" id="btn-zoom-out">−</button>
                <span id="zoom-level">100%</span>
                <button class="menu-btn zoom-btn" id="btn-zoom-in">+</button>
                <button class="menu-btn" id="btn-zoom-fit">⊡ Вписать</button>
                <button class="menu-btn" id="btn-zoom-reset">1:1</button>
            </div>
            <input type="file" id="file-input" accept=".json">
        </div>

        <div id="main">
            <div id="palette">
                <h3>📦 Элементы</h3>

                <div class="palette-section">
                    <div class="palette-section-title">ВХОДЫ</div>

                    <div class="palette-item" data-type="input-signal">
                        <svg viewBox="0 0 60 40">
                            <polygon points="0,5 40,5 55,20 40,35 0,35" fill="#0f3460" stroke="#4a90d9" stroke-width="2"/>
                            <text x="12" y="24" fill="#eee" font-size="10">IN</text>
                        </svg>
                        <div class="palette-item-name">Входной сигнал</div>
                    </div>
                </div>
                <div class="palette-section">
                    <div class="palette-section-title">ВЫХОДЫ</div>

                    <div class="palette-item" data-type="output">
                        <svg viewBox="0 0 60 40">
                            <rect x="5" y="5" width="50" height="30" rx="6" fill="none" stroke="#10b981" stroke-width="2" stroke-dasharray="4,2"/>
                            <text x="12" y="24" fill="#10b981" font-size="9">ВЫХОД</text>
                        </svg>
                        <div class="palette-item-name">Выход</div>
                    </div>
                </div>

                <div class="palette-section">
                    <div class="palette-section-title">ЛОГИЧЕСКИЕ</div>

                    <div class="palette-item" data-type="and">
                        <svg viewBox="0 0 60 40">
                            <rect x="5" y="5" width="50" height="30" rx="5"
```

```
    fill="#0f3460" stroke="#a855f7" stroke-width="2"/>
 62                             <text x="22" y="25" fill="#eee" font-size="12" font-
    weight="bold">И</text>
 63                         </svg>
 64                         <div class="palette-item-name">И (AND)</div>
 65                     </div>
 66
 67                     <div class="palette-item" data-type="or">
 68                         <svg viewBox="0 0 60 40">
 69                             <rect x="5" y="5" width="50" height="30" rx="5"
    fill="#0f3460" stroke="#a855f7" stroke-width="2"/>
 70                             <text x="12" y="25" fill="#eee" font-size="11" font-
    weight="bold">ИЛИ</text>
 71                         </svg>
 72                         <div class="palette-item-name">ИЛИ (OR)</div>
 73                     </div>
 74
 75                     <div class="palette-item" data-type="not">
 76                         <svg viewBox="0 0 60 40">
 77                             <rect x="5" y="5" width="50" height="30" rx="5"
    fill="#0f3460" stroke="#a855f7" stroke-width="2"/>
 78                             <text x="12" y="25" fill="#eee" font-size="11" font-
    weight="bold">НЕТ</text>
 79                         </svg>
 80                         <div class="palette-item-name">НЕТ (NOT)</div>
 81                     </div>
 82                 </div>
 83
 84                 <div class="palette-section">
 85                     <div class="palette-section-title">СРАВНЕНИЕ</div>
 86
 87                     <div class="palette-item" data-type="if">
 88                         <svg viewBox="0 0 60 40">
 89                             <polygon points="30,3 57,20 30,37 3,20" fill="#0f3460"
    stroke="#e94560" stroke-width="2"/>
 90                             <text x="14" y="24" fill="#eee" font-size="9" font-
    weight="bold">ЕСЛИ</text>
 91                         </svg>
 92                         <div class="palette-item-name">ЕСЛИ (IF)</div>
 93                     </div>
 94                 </div>
 95
 96                 <div class="palette-section">
 97                     <div class="palette-section-title">РАЗВЕТВЛЕНИЕ</div>
 98
 99                     <div class="palette-item" data-type="separator">
100                         <svg viewBox="0 0 60 40">
101                             <rect x="5" y="8" width="50" height="24" rx="3"
    fill="#0f3460" stroke="#f59e0b" stroke-width="2"/>
102                             <text x="8" y="25" fill="#f59e0b" font-size="10" font-
    weight="bold">✓/✗</text>
103                         </svg>
104                         <div class="palette-item-name">Сепаратор</div>
105                     </div>
106                 </div>
107
108                 <div class="palette-section">
109                     <div class="palette-section-title">ЗНАЧЕНИЯ</div>
110
111                     <div class="palette-item" data-type="const">
112                         <svg viewBox="0 0 60 40">
113                             <rect x="10" y="8" width="40" height="24" rx="3"
    fill="#0f3460" stroke="#3b82f6" stroke-width="2"/>
114                             <text x="24" y="25" fill="#3b82f6" font-size="14" font-
    weight="bold">C</text>
```

```
                        </svg>
                        <div class="palette-item-name">Константа</div>
                    </div>

                    <div class="palette-item" data-type="formula">
                        <svg viewBox="0 0 60 40">
                            <rect x="5" y="5" width="50" height="30" rx="5"
    fill="#0f3460" stroke="#f59e0b" stroke-width="2"/>
                            <text x="12" y="25" fill="#f59e0b" font-size="11" font-
    weight="bold">f(x)</text>
                        </svg>
                        <div class="palette-item-name">Формула</div>
                    </div>
                </div>

                <div class="type-legend">
                    <div class="type-legend-item">
                        <div class="type-legend-dot logic"></div>
                        <span>Логический</span>
                    </div>
                    <div class="type-legend-item">
                        <div class="type-legend-dot number"></div>
                        <span>Числовой</span>
                    </div>
                </div>
            </div>

            <div id="workspace-container">
                <svg id="connections-svg"></svg>
                <div id="workspace"></div>

                <!-- Мини-карта -->
                <div id="minimap">
                    <div id="minimap-viewport"></div>
                    <canvas id="minimap-canvas"></canvas>
                </div>

                <!-- Координаты и информация -->
                <div id="viewport-info">
                    <span id="cursor-pos">X: 0, Y: 0</span>
                    <span id="selection-info"></span>
                </div>
            </div>
        </div>
    </div>

    <!-- Модальные окна -->
    <div id="modal-overlay">
        <div id="modal">
            <h3 id="modal-title">Свойства элемента</h3>
            <div id="modal-content"></div>
            <div class="modal-buttons">
                <button class="modal-btn cancel" id="modal-cancel">Отмена</button>
                <button class="modal-btn save" id="modal-save">Сохранить</button>
            </div>
        </div>
    </div>

    <!-- Модальное окно свойств проекта -->
    <div id="project-modal-overlay" class="modal-overlay-class">
        <div id="project-modal" class="modal-class">
            <h3>Свойства проекта</h3>
            <div id="project-modal-content"></div>
            <div class="modal-buttons">
                <button class="modal-btn cancel" id="project-modal-cancel">Отмена</
```

```
    button>
178                 <button class="modal-btn save" id="project-modal-save">Сохранить</
    button>
179             </div>
180         </div>
181     </div>
182
183     <div id="code-modal-overlay" class="modal-overlay-class">
184         <div id="code-modal" class="modal-class">
185             <h3>Сгенерированный код</h3>
186             <textarea id="code-output" style="width:100%; height:300px;"></textarea>
187             <div class="modal-buttons">
188                 <button class="modal-btn cancel" id="code-modal-close">Закрыть</
    button>
189             </div>
190         </div>
191     </div>
192
193     <div id="context-menu">
194         <div class="context-item" id="ctx-properties">⚙️ Свойства</div>
195         <div class="context-item" id="ctx-delete">🗑️ Удалить</div>
196     </div>
197
198    <!-- Модули JavaScript -->
199     <!-- Модули JavaScript -->
200     <script src="js/config.js"></script>
201     <script src="js/state.js"></script>
202     <script src="js/utils.js"></script>
203     <script src="js/viewport.js"></script>
204     <script src="js/elements.js"></script>
205     <script src="js/connections.js"></script>
206     <script src="js/outputs.js"></script>  <!-- ← Этот файл опционален теперь -->
207     <script src="js/modal.js"></script>
208     <script src="js/project.js"></script>
209     <script src="js/codegen_graph.js"></script>
210     <script src="js/codegen_optimizer.js"></script>
211     <script src="js/codegen.js"></script>
212
213     <script src="js/app.js"></script>
214 </body>
215 </html>
216
217 app.js:
218 /**
219  * Главный модуль приложения
220  */
221
222 const App = {
223     /**
224      * Инициализация приложения
225      */
226     init() {
227         this.setupPaletteDragDrop();
228         this.setupGlobalMouseHandlers();
229         this.setupContextMenu();
230         this.setupWorkspaceClick();
231         this.setupOutputCounter();
232
233         // Инициализация модулей
234         Viewport.init();
235         Modal.init();
236         Project.init();
237
238         // Первоначальное определение выходов (только если модуль загружен)
239         if (typeof Outputs !== 'undefined' && Outputs.updateOutputStatus) {
```

```
            Outputs.updateOutputStatus();
        }

        console.log('Logic Scheme Editor initialized');
        document.getElementById('btn-generate-code').addEventListener('click', () => {
            const code = CodeGen.generate();
            document.getElementById('code-output').value = code;
            document.getElementById('code-modal-overlay').style.display = 'flex';
        });

        document.getElementById('code-modal-close').addEventListener('click', () => {
            document.getElementById('code-modal-overlay').style.display = 'none';
        });
    },

    /**
     * Отмена состояния drag из палитры (helper)
     */
    cancelPaletteDrag() {
        if (AppState.dragPreview) {
            try { AppState.dragPreview.remove(); } catch (e) { /* ignore */ }
            AppState.dragPreview = null;
        }
        AppState.isDraggingFromPalette = false;
        AppState.dragType = null;
    },

    /**
     * Настройка счётчика выходов в меню
     */
    setupOutputCounter() {
        // Не создавать повторно, если уже есть
        if (document.getElementById('btn-outputs')) return;

        const menu = document.getElementById('menu');

        // Создаём кнопку с счётчиком выходов
        const outputBtn = document.createElement('button');
        outputBtn.className = 'menu-btn output-btn';
        outputBtn.id = 'btn-outputs';
        outputBtn.innerHTML = `
            📤 Выходы
            <span id="output-counter" class="output-counter">0</span>
        `;

        // Вставляем после кнопки свойств проекта
        const projectBtn = document.getElementById('btn-project-settings');
        if (projectBtn) {
            projectBtn.after(outputBtn);
        } else {
            menu.appendChild(outputBtn);
        }

        outputBtn.addEventListener('click', () => {
            Modal.showProjectPropertiesModal();
        });
    },

    /**
     * Настройка drag & drop из палитры
     */
    setupPaletteDragDrop() {
        document.querySelectorAll('.palette-item').forEach(item => {
            item.addEventListener('mousedown', (e) => {
                // Только левая кнопка мыши должна запускать drag из палитры
```

```
                if (e.button !== 0) return;
                e.preventDefault();

                AppState.isDraggingFromPalette = true;
                AppState.dragType = item.dataset.type;

                AppState.dragPreview = document.createElement('div');
                AppState.dragPreview.className = 'drag-preview';
                AppState.dragPreview.textContent =
    ELEMENT_TYPES[AppState.dragType]?.name || 'Элемент';
                AppState.dragPreview.style.left = `${e.clientX - 40}px`;
                AppState.dragPreview.style.top = `${e.clientY - 20}px`;
                document.body.appendChild(AppState.dragPreview);
            });
        });
    },

    /**
     * Глобальные обработчики мыши
     */
/**
 * Глобальные обработчики мыши
 */
setupGlobalMouseHandlers() {
    document.addEventListener('mousemove', (e) => {
        if (AppState.isDraggingFromPalette && AppState.dragPreview) {
            AppState.dragPreview.style.left = `${e.clientX - 40}px`;
            AppState.dragPreview.style.top = `${e.clientY - 20}px`;
        }
        if (AppState.resizing) {
            Elements.handleResize(e);
            return;
        }
        if (AppState.draggingElement) {
            Elements.handleDrag(e);
        }
        if (AppState.tempLine && AppState.connectingFrom) {
            Connections.drawTempConnection(e);
        }
    });

    document.addEventListener('mouseup', (e) => {
        if (AppState.resizing) {
            AppState.resizing = null;
            if (typeof Outputs !== 'undefined') Outputs.updateOutputStatus();
        }

        if (AppState.isDraggingFromPalette) {
            try {
                if (AppState.dragPreview) {
                    AppState.dragPreview.remove();
                    AppState.dragPreview = null;
                }

                const container = document.getElementById('workspace-container');
                const rect = container.getBoundingClientRect();

                if (e.clientX >= rect.left && e.clientX <= rect.right &&
                    e.clientY >= rect.top && e.clientY <= rect.bottom) {

                    const canvasPos = screenToCanvas(e.clientX, e.clientY);
                    const config = ELEMENT_TYPES[AppState.dragType];
                    if (config) {
                        const defaultWidth = config.minWidth || 120;
                        const defaultHeight = config.minHeight || 60;
```

```
369
370                         // ИСПРАВЛЕНО: addElement возвращает DOM-элемент, его надо
    обработать
371                         const newElement = Elements.addElement(
372                             AppState.dragType,
373                             canvasPos.x - defaultWidth / 2,
374                             canvasPos.y - defaultHeight / 2
375                         );
376
377                         if (newElement && typeof Outputs !== 'undefined') {
378                             Outputs.updateOutputStatus();
379                         }
380                     } else {
381                         console.error('Неизвестный тип элемента при drop:',
    AppState.dragType);
382                     }
383                 }
384             } finally {
385                 App.cancelPaletteDrag();
386             }
387         }
388
389         if (AppState.draggingElement) {
390             AppState.draggingElement = null;
391         }
392
393         Connections.clearConnectionState();
394     });
395
396     document.addEventListener('keydown', (e) => {
397         if (e.key === 'Delete' && AppState.selectedElement) {
398             Elements.deleteElement(AppState.selectedElement);
399             if (typeof Outputs !== 'undefined') Outputs.updateOutputStatus();
400         }
401         if (e.key === 'Escape') {
402             Elements.deselectAll();
403             Connections.clearConnectionState();
404             if (AppState.isDraggingFromPalette) App.cancelPaletteDrag();
405         }
406     });
407 },
408
409     /**
410      * Настройка контекстного меню
411      */
412     setupContextMenu() {
413         document.addEventListener('click', (e) => {
414             const menu = document.getElementById('context-menu');
415             if (!menu.contains(e.target)) {
416                 menu.style.display = 'none';
417             }
418         });
419
420         document.getElementById('ctx-properties').addEventListener('click', () => {
421             const elemId = document.getElementById('context-menu').dataset.elementId;
422             document.getElementById('context-menu').style.display = 'none';
423             const config = ELEMENT_TYPES[AppState.elements[elemId]?.type];
424             if (config?.hasProperties) {
425                 Modal.showPropertiesModal(elemId);
426             }
427         });
428
429         document.getElementById('ctx-delete').addEventListener('click', () => {
430             const elemId = document.getElementById('context-menu').dataset.elementId;
431             document.getElementById('context-menu').style.display = 'none';
```

```
            Elements.deleteElement(elemId);
            // Обновляем выходы только если модуль загружен
            if (typeof Outputs !== 'undefined' && Outputs.updateOutputStatus) {
                Outputs.updateOutputStatus();
            }
        });
    },

    /**
     * Клик по рабочей области
     */
    setupWorkspaceClick() {
        const workspace = document.getElementById('workspace');

        workspace.addEventListener('click', (e) => {
            if (e.target === workspace) {
                Elements.deselectAll();
            }
        });
    }
};

// Запуск приложения при загрузке страницы
document.addEventListener('DOMContentLoaded', () => {
    App.init();
});

codegen_graph.js:
// js/codegen_graph.js

const CodeGenGraph = {
        /**
     * Собрать все условия вверх по цепочке cond-портов (до корня).
     * Возвращает null или объединённое через AND условие.
     */
/**
 * Собрать ВСЕ условия: и через cond-порты, и через контекст обычных входов
 */
    collectAllCond(graph) {
        if (!graph) return null;

        let c = null;
        const elem = graph.elem;

        // 1. Собираем условия через cond-порт (как было)
        if (graph.condInput) {
            const condConn = graph.condInput.conn;
            const fromGraph = graph.condInput.fromGraph;
            const oneCond = this.evalConditionFromPort(fromGraph, condConn.fromPort);
            c = oneCond;

            // Рекурсивно идём вверх по cond-цепочке
            const upCond = this.collectAllCond(fromGraph);
            if (upCond) {
                c = c ? Optimizer.And(c, upCond) : upCond;
            }
        }

        // 2. НОВОЕ: если это separator — учитываем контекст его входа
        if (elem.type === 'separator' && graph.inputs.length > 0) {
            const inputGraph = graph.inputs[0].fromGraph;
            const inputContext = this.collectAllCond(inputGraph);
            if (inputContext) {
                c = c ? Optimizer.And(c, inputContext) : inputContext;
            }
```

```
        }

        return c;
    },
    buildDependencyGraph(elementId) {
        const graph = {
            nodeId: elementId,
            elem: AppState.elements[elementId],
            inputs: [],
            condInput: null,
        };

        if (!graph.elem) return null;

        const inConns = AppState.connections.filter(c =>
            c.toElement === elementId && c.toPort.startsWith('in-')
        );
        inConns.forEach(conn => {
            graph.inputs.push({
                conn,
                fromGraph: this.buildDependencyGraph(conn.fromElement)
            });
        });

        const condConn = AppState.connections.find(c =>
            c.toElement === elementId && c.toPort === 'cond-0'
        );
        if (condConn) {
            graph.condInput = {
                conn: condConn,
                fromGraph: this.buildDependencyGraph(condConn.fromElement)
            };
        }

        return graph;
    },

    /**
     * Получить ЛОГИКУ из графа (для IF/AND/OR/NOT/SEPARATOR)
     */
    evalLogic(graph) {
        if (!graph) return Optimizer.TrueCond;
        const elem = graph.elem;

        switch (elem.type) {
            case 'if': {
                const left = graph.inputs[0]?.fromGraph;
                const right = graph.inputs[1]?.fromGraph;

                const leftVal = left ? this.evalValue(left) : Optimizer.Const(0);
                const rightVal = right ? this.evalValue(right) : Optimizer.Const(0);

                const op = elem.props.operator || '=';
                return this.buildIfLogic(leftVal, op, rightVal);
            }

            case 'and': {
                let result = null;
                for (const inp of graph.inputs) {
                    const inLogic = this.evalLogic(inp.fromGraph);
                    result = result ? Optimizer.And(result, inLogic) : inLogic;
                }
                return result || Optimizer.TrueCond;
            }

```

```
562             case 'or': {
563                 let result = null;
564                 for (const inp of graph.inputs) {
565                     const inLogic = this.evalLogic(inp.fromGraph);
566                     result = result ? Optimizer.Or(result, inLogic) : inLogic;
567                 }
568                 return result || Optimizer.FalseCond;
569             }
570 
571             case 'not': {
572                 const inLogic = this.evalLogic(graph.inputs[0]?.fromGraph);
573                 return Optimizer.Not(inLogic);
574             }
575 
576             case 'separator': {
577                 return this.evalLogic(graph.inputs[0]?.fromGraph);
578             }
579 
580             default:
581                 return Optimizer.TrueCond;
582         }
583     },
584 
585     /**
586      * Получить ЗНАЧЕНИЕ из графа (для INPUT/CONST/FORMULA)
587      */
588     evalValue(graph) {
589         if (!graph) return Optimizer.Const(0);
590         const elem = graph.elem;
591 
592         switch (elem.type) {
593             case 'input-signal':
594                 return Optimizer.Var(elem.props.name || graph.nodeId);
595 
596             case 'const':
597                 return Optimizer.Const(Number(elem.props.value) || 0);
598 
599             case 'formula': {
600                 const expr = this.buildFormulaExpr(elem);
601                 return Optimizer.Var(expr);
602             }
603 
604             case 'separator':
605                 return this.evalValue(graph.inputs[0]?.fromGraph);
606 
607             default:
608                 return Optimizer.Const(0);
609         }
610     },
611 
612     // js/codegen_graph.js
613 
614     /**
615      * Рекурсивно собрать полный контекст условий для элемента
616      * через всю цепочку cond-портов вверх
617      */
618   // В codegen_graph.js, в evalFullContext добавь:
619 
620     evalFullContext(graph) {
621         if (!graph) return null;
622 
623         let context = null;
624         const elem = graph.elem;
625 
626         console.log(`evalFullContext для ${elem.id} (${elem.type})`);
```

```
627
628         // 1. Если сам элемент имеет cond-порт — собираем его условие
629         if (graph.condInput) {
630             const condConn = graph.condInput.conn;
631             console.log(`  → имеет cond-0 от ${graph.condInput.fromGraph.elem.id}.$
    {condConn.fromPort}`);
632
633             const condLogic = this.evalConditionFromPort(
634                 graph.condInput.fromGraph,
635                 condConn.fromPort
636             );
637             console.log(`  → условие от cond-0: ${Optimizer.printCond(condLogic)}`);
638             context = condLogic;
639
640             // 2. Рекурсивно собираем контекст элемента, на который указывает cond-
    порт
641             const upstreamContext = this.evalFullContext(graph.condInput.fromGraph);
642             if (upstreamContext) {
643                 console.log(`  → upstreamContext: $
    {Optimizer.printCond(upstreamContext)}`);
644                 context = context ? Optimizer.And(context, upstreamContext) :
    upstreamContext;
645             }
646         } else {
647             console.log(`  → нет cond-0`);
648         }
649
650         console.log(`  → итоговый контекст: ${Optimizer.printCond(context)}`);
651         return context;
652     },
653
654     /**
655      * Получить УСЛОВИЕ для cond-порта элемента
656      * Учитывает цепочку сепараторов с TRUE/FALSE ветвлением
657      */
658     evalConditionFromPort(graph, fromPort) {
659         if (!graph) return null;
660         const elem = graph.elem;
661
662         // Если это сепаратор — вычисляем его вход и применяем ветвление
663         if (elem.type === 'separator') {
664             const inputLogic = this.evalLogic(graph.inputs[0]?.fromGraph);
665
666             if (fromPort === 'out-0') {
667                 return inputLogic;
668             } else if (fromPort === 'out-1') {
669                 return Optimizer.Not(inputLogic);
670             }
671         }
672
673         // Если это логический элемент (AND/OR/NOT/IF) — просто вычисляем логику
674         if (elem.type === 'and' || elem.type === 'or' || elem.type === 'not' ||
    elem.type === 'if') {
675             return this.evalLogic(graph);
676         }
677
678         return null;
679     },
680
681     /**
682      * Главная функция: получить {cond, expr} для элемента
683      */
684     evalGraphValue(graph) {
685
686         if (!graph) return { cond: null, expr: Optimizer.Const(0) };
```

```
687
688         const elem = graph.elem;
689         //let cond = null;
690
691         // ← НОВОЕ: собираем полный контекст через цепочку cond-портов
692         let cond = this.collectAllCond(graph);
693
694         let expr = null;
695
696         switch (elem.type) {
697             case 'input-signal':
698                 expr = Optimizer.Var(elem.props.name || graph.nodeId);
699                 break;
700
701             case 'const':
702                 expr = Optimizer.Const(Number(elem.props.value) || 0);
703                 break;
704
705             case 'formula': {
706                 // Для формулы также собираем условия от всех входных элементов
707                 const inputConds = graph.inputs.map(inp => {
708                     const inResult = this.evalGraphValue(inp.fromGraph);
709                     return inResult.cond;
710                 }).filter(c => c);
711
712                 // Объединяем cond-порт с условиями от входов
713                 for (const inCond of inputConds) {
714                     cond = cond ? Optimizer.And(cond, inCond) : inCond;
715                 }
716
717                 expr = Optimizer.Var(this.buildFormulaExpr(elem));
718                 break;
719             }
720
721             case 'separator':
722                 // Сепаратор — просто пробрасываем значение дальше
723                 return this.evalGraphValue(graph.inputs[0]?.fromGraph);
724
725             // Логические элементы не должны здесь быть
726             case 'and':
727             case 'or':
728             case 'not':
729             case 'if':
730             default:
731                 expr = Optimizer.Const(0);
732         }
733
734         return { cond, expr };
735     },
736
737     buildIfLogic(leftVal, op, rightVal) {
738         const leftName = leftVal.type === 'var' ? leftVal.name : String(leftVal.n);
739         const rightName = rightVal.type === 'var' ? rightVal.name :
    String(rightVal.n);
740
741         const leftZero = leftVal.type === 'const' && leftVal.n === 0;
742         const rightZero = rightVal.type === 'const' && rightVal.n === 0;
743
744         switch (op) {
745             case '=':
746                 if (rightZero) return Optimizer.Eq0(leftName);
747                 if (leftZero) return Optimizer.Eq0(rightName);
748                 return Optimizer.Cmp(leftName, '=', rightName);
749             case '!=':
750                 if (rightZero) return Optimizer.Ne0(leftName);
```

```
                if (leftZero) return Optimizer.Ne0(rightName);
                return Optimizer.Cmp(leftName, '!=', rightName);
            case '>':
            case '<':
            case '>=':
            case '<=':
                return Optimizer.Cmp(leftName, op, rightName);
            default:
                return Optimizer.TrueCond;
        }
    },

    buildFormulaExpr(elem) {
        let result = elem.props.expression || '0';
        const formulaRefs = result.match(/formula-\d+/g) || [];

        for (const ref of formulaRefs) {
            const refElem = AppState.elements[ref];
            if (refElem && refElem.type === 'formula') {
                const refExpr = this.buildFormulaExpr(refElem);
                result = result.replace(new RegExp(ref, 'g'), `(${refExpr})`);
            }
        }

        return result;
    }
};

window.CodeGenGraph = CodeGenGraph;

codegen_optimizer.js:
// js/codegen_optimizer.js

let _depth = 0;
const MAX_DEPTH = 200;

// === Конструкторы ===
function Eq0(v) { return { kind: 'cond', type: 'eq0', v }; }
function Ne0(v) { return { kind: 'cond', type: 'ne0', v }; }
function Cmp(l, op, r) { return { kind: 'cond', type: 'cmp', l, op, r }; }
function And(a, b) {
    if (!a) return b;
    if (!b) return a;
    return { kind: 'cond', type: 'and', a, b };
}
function Or(a, b) {
    if (!a) return b;
    if (!b) return a;
    return { kind: 'cond', type: 'or', a, b };
}
function Not(x) {
    if (!x) return null;
    return { kind: 'cond', type: 'not', x };
}
const TrueCond = { kind: 'cond', type: 'true' };
const FalseCond = { kind: 'cond', type: 'false' };

function Const(n) { return { kind: 'expr', type: 'const', n }; }
function Var(name) { return { kind: 'expr', type: 'var', name }; }
function Op(op, l, r) { return { kind: 'expr', type: 'op', op, l, r }; }
function When(c, t, e) { return { kind: 'expr', type: 'when', c, t, e }; }

// === Утилиты ===
function atomKey(c) {
    if (!c) return null;
```

```
    switch (c.type) {
        case 'eq0': return `eq0:${c.v}`;
        case 'ne0': return `ne0:${c.v}`;
        case 'cmp': return `cmp:${c.l}:${c.op}:${c.r}`;
        case 'true': return 'true';
        case 'false': return 'false';
        default: return null;
    }
}

function negateOp(op) {
    switch (op) {
        case '=': return '!=';
        case '!=': return '=';
        case '>': return '<=';
        case '<': return '>=';
        case '>=': return '<';
        case '<=': return '>';
        default: return null;
    }
}

// Преобразует cmp-условие в интервал по одной переменной
// Возвращает { varName, min, minInc, max, maxInc } или null
function cmpToInterval(c) {
    if (!c || c.type !== 'cmp') return null;

    const lNum = parseNumberLiteral(c.l);
    const rNum = parseNumberLiteral(c.r);

    let varName, op, val;

    if (lNum == null && rNum != null) {
        // var OP const
        varName = c.l;
        op = c.op;
        val = rNum;
    } else if (lNum != null && rNum == null) {
        // const OP var  ->  var (OP') const
        varName = c.r;
        op = reverseOp(c.op);
        if (!op) return null;
        val = lNum;
    } else {
        // Либо обе стороны числа, либо обе не числа — не трогаем
        return null;
    }

    // Интересуют только упорядочивающие операторы
    switch (op) {
        case '<':
        case '<=':
        case '>':
        case '>=':
        case '=':
            break;
        default:
            return null;
    }

    let min = Number.NEGATIVE_INFINITY;
    let max = Number.POSITIVE_INFINITY;
    let minInc = false;
    let maxInc = false;

```

```
    switch (op) {
        case '<':
            max = val; maxInc = false; break;
        case '<=':
            max = val; maxInc = true; break;
        case '>':
            min = val; minInc = false; break;
        case '>=':
            min = val; minInc = true; break;
        case '=':
            min = val; minInc = true;
            max = val; maxInc = true;
            break;
    }

    return { varName, min, minInc, max, maxInc };
}

function intervalSubset(a, b) {
    if (!a || !b) return false;

    // Нижняя граница: a.min >= b.min
    const amin = a.min, bmin = b.min;
    if (amin === Number.NEGATIVE_INFINITY) {
        if (bmin !== Number.NEGATIVE_INFINITY) return false;
        // оба -∞ — ок
    } else if (bmin === Number.NEGATIVE_INFINITY) {
        // b начинается “раньше” — ок
    } else if (amin > bmin) {
        // a стартует правее b — ок
    } else if (amin < bmin) {
        // a захватывает меньшее значение — не подмножество
        return false;
    } else {
        // amin === bmin
        if (a.minInc && !b.minInc) {
            // a включает границу, а b — нет → в a есть точка, не входящая в b
            return false;
        }
    }

    // Верхняя граница: a.max <= b.max
    const amax = a.max, bmax = b.max;
    if (amax === Number.POSITIVE_INFINITY) {
        if (bmax !== Number.POSITIVE_INFINITY) return false;
    } else if (bmax === Number.POSITIVE_INFINITY) {
        // b идёт дальше — ок
    } else if (amax < bmax) {
        // a заканчивается раньше — ок
    } else if (amax > bmax) {
        return false;
    } else {
        // amax === bmax
        if (a.maxInc && !b.maxInc) {
            return false;
        }
    }

    return true;
}

// Удаляет избыточные cmp-условия в массиве атомов
// mode: 'and' | 'or'
function removeRedundantCmpAtoms(atoms, mode) {
    if (!atoms || atoms.length < 2) return atoms;
```

```
946
947       const keep = new Array(atoms.length).fill(true);
948
949       for (let i = 0; i < atoms.length; i++) {
950           if (!keep[i]) continue;
951           const a = atoms[i];
952           if (!a || a.type !== 'cmp') continue;
953
954           for (let j = 0; j < atoms.length; j++) {
955               if (i === j || !keep[j]) continue;
956               const b = atoms[j];
957               if (!b || b.type !== 'cmp') continue;
958
959               const rel = cmpImplicationRelation(a, b);
960               if (!rel) continue;
961
962               if (rel === 'a_in_b') {
963                   if (mode === 'or') {
964                       // A ⊆ B → A OR B = B  → A лишнее
965                       keep[i] = false;
966                       break;
967                   } else if (mode === 'and') {
968                       // A ⊆ B → A AND B = A → B лишнее
969                       keep[j] = false;
970                   }
971               } else if (rel === 'b_in_a') {
972                   if (mode === 'or') {
973                       // B ⊆ A → A OR B = A → B лишнее
974                       keep[j] = false;
975                   } else if (mode === 'and') {
976                       // B ⊆ A → A AND B = B → A лишнее
977                       keep[i] = false;
978                       break;
979                   }
980               }
981           }
982       }
983
984       return atoms.filter((_, idx) => keep[idx]);
985   }
986
987   // Отношение между двумя cmp-условиями через интервалы
988   // 'a_in_b'  — A ⊆ B
989   // 'b_in_a'  — B ⊆ A
990   // 'equal'   — одинаковые интервалы (редко используем)
991   // null      — не можем определить
992   function cmpImplicationRelation(c1, c2) {
993       const i1 = cmpToInterval(c1);
994       const i2 = cmpToInterval(c2);
995       if (!i1 || !i2) return null;
996       if (i1.varName !== i2.varName) return null;
997
998       const aInB = intervalSubset(i1, i2);
999       const bInA = intervalSubset(i2, i1);
1000
1001      if (aInB && bInA) return 'equal';
1002      if (aInB) return 'a_in_b';
1003      if (bInA) return 'b_in_a';
1004      return null;
1005  }
1006
1007  // Разворот оператора при перестановке аргументов (левый/правый)
1008  function reverseOp(op) {
1009      switch (op) {
1010          case '<':  return '>';
```

```
            case '>':  return '<';
            case '<=': return '>=';
            case '>=': return '<=';
            case '=':
            case '!=':
                return op;
            default:
                return null;
        }
    }

    // Аккуратный парсер числового литерала.
    // Возвращает число или null, если строка не чисто числовая.
    function parseNumberLiteral(s) {
        if (typeof s !== 'string') return null;
        const trimmed = s.trim().replace(',', '.');

        // Только простые вещи: -123, 45, 3.14
        if (!/^-?\d+(\.\d+)?$/.test(trimmed)) return null;

        const n = Number(trimmed);
        return Number.isFinite(n) ? n : null;
    }


    function negateAtomKey(key) {
        if (!key) return null;
        if (key.startsWith('eq0:')) return 'ne0:' + key.slice(4);
        if (key.startsWith('ne0:')) return 'eq0:' + key.slice(4);
        if (key.startsWith('cmp:')) {
            const parts = key.slice(4).split(':');
            if (parts.length === 3) {
                const negOp = negateOp(parts[1]);
                if (negOp) return `cmp:${parts[0]}:${negOp}:${parts[2]}`;
            }
        }
        return null;
    }

    function isNegation(a, b) {
        if (!a || !b) return false;
        if (a.type === 'eq0' && b.type === 'ne0' && a.v === b.v) return true;
        if (a.type === 'ne0' && b.type === 'eq0' && a.v === b.v) return true;
        if (a.type === 'cmp' && b.type === 'cmp' && a.l === b.l && a.r === b.r) {
            return a.op === negateOp(b.op);
        }
        if (a.type === 'not' && condEq(a.x, b)) return true;
        if (b.type === 'not' && condEq(b.x, a)) return true;
        return false;
    }

    function condEq(a, b) {
        if (a === b) return true;
        if (!a || !b) return false;
        if (a.type !== b.type) return false;

        switch (a.type) {
            case 'eq0':
            case 'ne0':
                return a.v === b.v;
            case 'cmp':
                return a.l === b.l && a.op === b.op && a.r === b.r;
            case 'true':
            case 'false':
                return true;
```

```
        case 'not':
            return condEq(a.x, b.x);
        case 'and':
        case 'or':
            return (condEq(a.a, b.a) && condEq(a.b, b.b)) ||
                   (condEq(a.a, b.b) && condEq(a.b, b.a));
        default:
            return false;
    }
}

function flattenAnd(c) {
    if (!c) return [];
    if (c.type === 'and') return [...flattenAnd(c.a), ...flattenAnd(c.b)];
    return [c];
}

function flattenOr(c) {
    if (!c) return [];
    if (c.type === 'or') return [...flattenOr(c.a), ...flattenOr(c.b)];
    return [c];
}

function buildAnd(terms) {
    if (terms.length === 0) return TrueCond;
    let result = terms[0];
    for (let i = 1; i < terms.length; i++) {
        result = And(result, terms[i]);
    }
    return result;
}

function buildOr(terms) {
    if (terms.length === 0) return FalseCond;
    let result = terms[0];
    for (let i = 1; i < terms.length; i++) {
        result = Or(result, terms[i]);
    }
    return result;
}

// Поглощение для AND: X AND (X OR Y) = X
function applyAndAbsorption(terms) {
    if (!terms || terms.length < 2) return terms;

    const keep = new Array(terms.length).fill(true);

    for (let i = 0; i < terms.length; i++) {
        if (!keep[i]) continue;
        const ti = terms[i];
        if (!ti || ti.type !== 'or') continue;

        const orParts = flattenOr(ti);
        let drop = false;

        outer:
        for (const part of orParts) {
            for (let j = 0; j < terms.length; j++) {
                if (j === i || !keep[j]) continue;
                if (condEq(part, terms[j])) {
                    drop = true;
                    break outer;
                }
            }
        }
```

```
            if (drop) {
                keep[i] = false;
            }
        }

        return terms.filter((_, idx) => keep[idx]);
    }

    // Поглощение для OR: X OR (X AND Y) = X
    function applyOrAbsorption(terms) {
        if (!terms || terms.length < 2) return terms;

        const keep = new Array(terms.length).fill(true);

        for (let i = 0; i < terms.length; i++) {
            if (!keep[i]) continue;
            const ti = terms[i];
            if (!ti || ti.type !== 'and') continue;

            const andParts = flattenAnd(ti);
            let drop = false;

            outer:
            for (const part of andParts) {
                for (let j = 0; j < terms.length; j++) {
                    if (j === i || !keep[j]) continue;
                    if (condEq(part, terms[j])) {
                        drop = true;
                        break outer;
                    }
                }
            }

            if (drop) {
                keep[i] = false;
            }
        }

        return terms.filter((_, idx) => keep[idx]);
    }

    // === Упрощение условий ===
    function simplifyCond(c) {
        _depth++;
        if (_depth > MAX_DEPTH) {
            _depth--;
            return c;
        }

        try {
            return simplifyCondCore(c);
        } finally {
            _depth--;
        }
    }

    function simplifyCondCore(c) {
        if (!c || c.kind !== 'cond') return c;

        switch (c.type) {
            case 'true':
            case 'false':
            case 'eq0':
            case 'ne0':
```

```
        case 'cmp':
            return c;

        case 'not': {
            const x = simplifyCondCore(c.x);
            if (!x) return TrueCond;
            if (x.type === 'true') return FalseCond;
            if (x.type === 'false') return TrueCond;
            if (x.type === 'not') return simplifyCondCore(x.x);
            if (x.type === 'eq0') return Ne0(x.v);
            if (x.type === 'ne0') return Eq0(x.v);
            if (x.type === 'cmp') {
                const negOp = negateOp(x.op);
                if (negOp) return Cmp(x.l, negOp, x.r);
            }
            if (x.type === 'and') return simplifyCondCore(Or(Not(x.a), Not(x.b)));
            if (x.type === 'or') return simplifyCondCore(And(Not(x.a), Not(x.b)));
            return Not(x);
        }

case 'and': {
    const a = simplifyCondCore(c.a);
    const b = simplifyCondCore(c.b);

    if (!a) return b;
    if (!b) return a;
    if (a.type === 'false' || b.type === 'false') return FalseCond;
    if (a.type === 'true') return b;
    if (b.type === 'true') return a;

    const allTerms = [...flattenAnd(a), ...flattenAnd(b)];

    // === НОВОЕ: Сразу собираем все eq0/ne0 для быстрой проверки ===
    const eq0Vars = new Map(); // var -> term
    const ne0Vars = new Map(); // var -> term
    const cmpTerms = [];
    const otherTerms = [];

    for (const t of allTerms) {
        if (t.type === 'true') continue;
        if (t.type === 'false') return FalseCond;

        if (t.type === 'eq0') {
            // Проверка на противоречие сразу
            if (ne0Vars.has(t.v)) {
                console.log(`Противоречие найдено: ${t.v} = 0 AND ${t.v} != 0`);
                return FalseCond;
            }
            eq0Vars.set(t.v, t);
        } else if (t.type === 'ne0') {
            // Проверка на противоречие сразу
            if (eq0Vars.has(t.v)) {
                console.log(`Противоречие найдено: ${t.v} != 0 AND ${t.v} = 0`);
                return FalseCond;
            }
            ne0Vars.set(t.v, t);
        } else if (t.type === 'cmp') {
            cmpTerms.push(t);
        } else if (t.type === 'or') {
            // === НОВОЕ: Проверяем каждую ветку OR на противоречие с контекстом ===
            const orTerms = flattenOr(t);
            const validBranches = [];

            for (const branch of orTerms) {
                let branchValid = true;
```

```
1271
1272                 if (branch.type === 'ne0' && eq0Vars.has(branch.v)) {
1273                     console.log(`OR ветка ${branch.v} != 0 противоречит контексту $
     {branch.v} = 0`);
1274                     branchValid = false;
1275                 } else if (branch.type === 'eq0' && ne0Vars.has(branch.v)) {
1276                     console.log(`OR ветка ${branch.v} = 0 противоречит контексту $
     {branch.v} != 0`);
1277                     branchValid = false;
1278                 }
1279
1280                 if (branchValid) {
1281                     validBranches.push(branch);
1282                 }
1283             }
1284
1285             if (validBranches.length === 0) {
1286                 console.log(`Все ветки OR противоречат контексту → FALSE`);
1287                 return FalseCond;
1288             } else if (validBranches.length === 1) {
1289                 // Если осталась только одна ветка OR, добавляем её напрямую
1290                 const singleBranch = validBranches[0];
1291                 if (singleBranch.type === 'eq0') {
1292                     if (ne0Vars.has(singleBranch.v)) return FalseCond;
1293                     eq0Vars.set(singleBranch.v, singleBranch);
1294                 } else if (singleBranch.type === 'ne0') {
1295                     if (eq0Vars.has(singleBranch.v)) return FalseCond;
1296                     ne0Vars.set(singleBranch.v, singleBranch);
1297                 } else {
1298                     otherTerms.push(singleBranch);
1299                 }
1300             } else {
1301                 // Перестраиваем OR только с валидными ветками
1302                 otherTerms.push(buildOr(validBranches));
1303             }
1304         } else {
1305             otherTerms.push(t);
1306         }
1307     }
1308
1309     // Собираем уникальные атомы
1310     const atomMap = new Map();
1311
1312     for (const [v, term] of eq0Vars) {
1313         const key = atomKey(term);
1314         if (key) atomMap.set(key, term);
1315     }
1316
1317     for (const [v, term] of ne0Vars) {
1318         const key = atomKey(term);
1319         if (key) atomMap.set(key, term);
1320     }
1321
1322     for (const term of cmpTerms) {
1323         const key = atomKey(term);
1324         if (key) {
1325             const negKey = negateAtomKey(key);
1326             if (negKey && atomMap.has(negKey)) {
1327                 return FalseCond;
1328             }
1329             if (!atomMap.has(key)) {
1330                 atomMap.set(key, term);
1331             }
1332         }
1333     }
```

```

    let uniqueAtoms = Array.from(atomMap.values());
    uniqueAtoms = removeRedundantCmpAtoms(uniqueAtoms, 'and');

    let result = [...uniqueAtoms, ...otherTerms];

    // Поглощение: X AND (X OR Y) = X
    result = applyAndAbsorption(result);

    if (result.length === 0) return TrueCond;
    if (result.length === 1) return result[0];

    return buildAnd(result);
}

        case 'or': {
            const a = simplifyCondCore(c.a);
            const b = simplifyCondCore(c.b);

            if (!a) return b;
            if (!b) return a;
            if (a.type === 'true' || b.type === 'true') return TrueCond;
            if (a.type === 'false') return b;
            if (b.type === 'false') return a;

            const allTerms = [...flattenOr(a), ...flattenOr(b)];
            const atomMap = new Map();
            const otherTerms = [];

            for (const t of allTerms) {
                if (t.type === 'true') return TrueCond;
                if (t.type === 'false') continue;

                const key = atomKey(t);
                if (key) {
                    const negKey = negateAtomKey(key);
                    if (negKey && atomMap.has(negKey)) {
                        return TrueCond;
                    }
                    if (!atomMap.has(key)) {
                        atomMap.set(key, t);
                    }
                } else {
                    otherTerms.push(t);
                }
            }

            let uniqueAtoms = Array.from(atomMap.values());
            uniqueAtoms = removeRedundantCmpAtoms(uniqueAtoms, 'or');

            let result = [...uniqueAtoms, ...otherTerms];

            // Поглощение: X OR (X AND Y) = X
            result = applyOrAbsorption(result);

            if (result.length === 0) return FalseCond;
            if (result.length === 1) return result[0];

            return buildOr(result);
        }

        default:
            return c;
    }
}
```

```
1399
1400  // === Сравнение выражений ===
1401  function exprEq(a, b) {
1402      if (a === b) return true;
1403      if (!a && !b) return true;
1404      if (!a || !b) return false;
1405      if (a.type !== b.type) return false;
1406
1407      switch (a.type) {
1408          case 'const': return a.n === b.n;
1409          case 'var': return a.name === b.name;
1410          case 'op': return a.op === b.op && exprEq(a.l, b.l) && exprEq(a.r, b.r);
1411          case 'when': return condEq(a.c, b.c) && exprEq(a.t, b.t) && exprEq(a.e, b.e);
1412          default: return false;
1413      }
1414  }
1415
1416  // === Упрощение выражений ===
1417  function simplifyExpr(expr) {
1418      _depth++;
1419      if (_depth > MAX_DEPTH) {
1420          _depth--;
1421          return expr;
1422      }
1423
1424      try {
1425          return simplifyExprCore(expr);
1426      } finally {
1427          _depth--;
1428      }
1429  }
1430
1431  function simplifyExprCore(expr) {
1432      if (!expr || expr.kind !== 'expr') return expr;
1433
1434      switch (expr.type) {
1435          case 'const':
1436          case 'var':
1437              return expr;
1438
1439          case 'op': {
1440              const l = simplifyExprCore(expr.l);
1441              const r = simplifyExprCore(expr.r);
1442
1443              if (expr.op === '+') {
1444                  if (r?.type === 'const' && r.n === 0) return l;
1445                  if (l?.type === 'const' && l.n === 0) return r;
1446              }
1447              if (expr.op === '*') {
1448                  if (l?.type === 'const' && l.n === 0) return Const(0);
1449                  if (r?.type === 'const' && r.n === 0) return Const(0);
1450                  if (l?.type === 'const' && l.n === 1) return r;
1451                  if (r?.type === 'const' && r.n === 1) return l;
1452              }
1453              return Op(expr.op, l, r);
1454          }
1455
1456          case 'when': {
1457              const c = simplifyCond(expr.c);
1458              const t = simplifyExprCore(expr.t);
1459              const e = simplifyExprCore(expr.e);
1460
1461              if (c?.type === 'true') return t;
1462              if (c?.type === 'false') return e;
1463              if (exprEq(t, e)) return t;
```

```
1464
1465                return When(c, t, e);
1466            }
1467
1468            default:
1469                return expr;
1470        }
1471    }
1472
1473    // === Печать ===
1474    function printCond(c) {
1475        if (!c) return 'TRUE';
1476
1477        switch (c.type) {
1478            case 'eq0': return `${c.v} = 0`;
1479            case 'ne0': return `${c.v} != 0`;
1480            case 'cmp': return `${c.l} ${c.op} ${c.r}`;
1481            case 'and': return `(${printCond(c.a)} AND ${printCond(c.b)})`;
1482            case 'or':  return `(${printCond(c.a)} OR ${printCond(c.b)})`;
1483            case 'not': return `NOT(${printCond(c.x)})`;
1484            case 'true': return 'TRUE';
1485            case 'false': return 'FALSE';
1486            default: return '?';
1487        }
1488    }
1489
1490    function printExpr(e) {
1491        if (!e) return '0';
1492
1493        switch (e.type) {
1494            case 'const': return String(e.n);
1495            case 'var': return e.name;
1496            case 'op': return `(${printExpr(e.l)}${e.op}${printExpr(e.r)})`;
1497            case 'when': return `WHEN(${printCond(e.c)}, ${printExpr(e.t)}, $
      {printExpr(e.e)})`;
1498            default: return '?';
1499        }
1500    }
1501
1502    window.Optimizer = {
1503        Eq0, Ne0, Cmp, And, Or, Not, TrueCond, FalseCond,
1504        Const, Var, Op, When,
1505        simplifyCond, simplifyExpr,
1506        printCond, printExpr,
1507        condEq, exprEq
1508    };
1509
1510    codegen.js:
1511    // js/codegen.js
1512
1513    const CodeGen = {
1514        _cache: {},
1515        _branchCache: {},
1516        _resolveCache: {},
1517        _visiting: new Set(),
1518
1519        reset() {
1520            this._cache = {};
1521            this._branchCache = {};
1522            this._resolveCache = {};
1523            this._visiting = new Set();
1524        },
1525
1526        toExpr(valueStr) {
1527            const s = String(valueStr).trim();
```

```
1528          if (s === '0') return Optimizer.Const(0);
1529          const num = parseFloat(s);
1530          if (!isNaN(num) && String(num) === s) return Optimizer.Const(num);
1531          return Optimizer.Var(s);
1532      },
1533
1534      exprToName(exprAst) {
1535          if (!exprAst) return '0';
1536          if (exprAst.type === 'var') return exprAst.name;
1537          if (exprAst.type === 'const') return String(exprAst.n);
1538          return Optimizer.printExpr(exprAst);
1539      },
1540
1541      mergeCond(a, b) {
1542          if (!a && !b) return null;
1543          if (!a) return b;
1544          if (!b) return a;
1545          if (Optimizer.condEq && Optimizer.condEq(a, b)) return a;
1546          return Optimizer.And(a, b);
1547      },
1548
1549      getConn(toId, toPort) {
1550          return AppState.connections.find(c => c.toElement === toId && c.toPort ===
      toPort);
1551      },
1552
1553      getConns(toId, prefix) {
1554          return AppState.connections.filter(c => c.toElement === toId &&
      c.toPort.startsWith(prefix));
1555      },
1556
1557      buildFormulaExpr(elem) {
1558          const expression = elem.props.expression || '0';
1559          let result = expression;
1560          const formulaRefs = result.match(/formula-\d+/g) || [];
1561
1562          for (const ref of formulaRefs) {
1563              const refElem = AppState.elements[ref];
1564              if (refElem && refElem.type === 'formula') {
1565                  const refExpr = this.buildFormulaExpr(refElem);
1566                  result = result.replace(new RegExp(ref, 'g'), `(${refExpr})`);
1567              }
1568          }
1569
1570          return result;
1571      },
1572
1573      // === Получить ЧИСТУЮ логику элемента ===
1574      getPureLogic(id) {
1575          const cacheKey = `logic:${id}`;
1576          if (cacheKey in this._cache) {
1577              return this._cache[cacheKey];
1578          }
1579
1580          const elem = AppState.elements[id];
1581          if (!elem) return null;
1582
1583          let logic = null;
1584
1585          switch (elem.type) {
1586              case 'if': {
1587                  const leftConn = this.getConn(id, 'in-0');
1588                  const rightConn = this.getConn(id, 'in-1');
1589
1590                  const leftVal = leftConn ? this.getValue(leftConn.fromElement) :
```

```
      Optimizer.Const(0);
1591                  const rightVal = rightConn ? this.getValue(rightConn.fromElement) :
      Optimizer.Const(0);
1592
1593                  const op = (elem.props.operator || '=').trim();
1594                  const leftName = this.exprToName(leftVal);
1595                  const rightName = this.exprToName(rightVal);
1596
1597                  const leftZero = leftVal.type === 'const' && leftVal.n === 0;
1598                  const rightZero = rightVal.type === 'const' && rightVal.n === 0;
1599
1600                  switch (op) {
1601                      case '=':
1602                          if (rightZero) {
1603                              logic = Optimizer.Eq0(leftName);
1604                          } else if (leftZero) {
1605                              logic = Optimizer.Eq0(rightName);
1606                          } else {
1607                              logic = Optimizer.Cmp(leftName, '=', rightName);
1608                          }
1609                          break;
1610                      case '!=':
1611                          if (rightZero) {
1612                              logic = Optimizer.Ne0(leftName);
1613                          } else if (leftZero) {
1614                              logic = Optimizer.Ne0(rightName);
1615                          } else {
1616                              logic = Optimizer.Cmp(leftName, '!=', rightName);
1617                          }
1618                          break;
1619                      case '>':
1620                      case '<':
1621                      case '>=':
1622                      case '<=':
1623                          logic = Optimizer.Cmp(leftName, op, rightName);
1624                          break;
1625                      default:
1626                          logic = Optimizer.TrueCond;
1627                  }
1628                  break;
1629              }
1630
1631              case 'and':
1632              case 'or': {
1633                  const isAnd = elem.type === 'and';
1634                  const count = elem.props.inputCount || 2;
1635                  let result = null;
1636
1637                  for (let i = 0; i < count; i++) {
1638                      const conn = this.getConn(id, `in-${i}`);
1639                      if (!conn) continue;
1640
1641                      const val = this.getPureLogic(conn.fromElement);
1642                      if (!val) continue;
1643
1644                      if (result === null) {
1645                          result = val;
1646                      } else {
1647                          result = isAnd ? Optimizer.And(result, val) :
      Optimizer.Or(result, val);
1648                      }
1649                  }
1650                  logic = result || Optimizer.FalseCond;
1651                  break;
1652              }
```

```
1653
1654              case 'not': {
1655                  const conn = this.getConn(id, 'in-0');
1656                  const inputLogic = conn ? this.getPureLogic(conn.fromElement) : null;
1657                  logic = Optimizer.Not(inputLogic || Optimizer.FalseCond);
1658                  break;
1659              }
1660
1661              case 'separator': {
1662                  const conn = this.getConn(id, 'in-0');
1663                  logic = conn ? this.getPureLogic(conn.fromElement) :
      Optimizer.FalseCond;
1664                  break;
1665              }
1666
1667              default:
1668                  logic = null;
1669          }
1670
1671          // ↓ новая часть: добавляем контекст с cond-порта для логических элементов
1672          if (elem.type === 'if' || elem.type === 'and' || elem.type === 'or' ||
      elem.type === 'not') {
1673              const ctx = this.getConditionFromPort(id);
1674              if (ctx) {
1675                  if (logic) {
1676                      logic = Optimizer.And(ctx, logic);
1677                  } else {
1678                      logic = ctx;
1679                  }
1680              }
1681          }
1682
1683          this._cache[cacheKey] = logic;
1684          return logic;
1685      },
1686
1687      // === Получить значение ===
1688      getValue(id) {
1689          const elem = AppState.elements[id];
1690          if (!elem) return Optimizer.Const(0);
1691
1692          switch (elem.type) {
1693              case 'input-signal':
1694                  // Имя сигнала или id как Var(...)
1695                  return this.toExpr(elem.props.name || id);
1696
1697              case 'const':
1698                  return Optimizer.Const(Number(elem.props.value) || 0);
1699
1700              case 'formula': {
1701                  // Используем текст формулы как выражение
1702                  const exprStr = this.buildFormulaExpr(elem) || '0';
1703                  return this.toExpr(exprStr);
1704              }
1705
1706              default:
1707                  // На всякий случай — даём символическое имя, а не 0
1708                  if (elem.props && typeof elem.props.name === 'string') {
1709                      return this.toExpr(elem.props.name);
1710                  }
1711                  return this.toExpr(id);
1712          }
1713      },
1714
1715      // === Получить ПОЛНОЕ условие для ветки сепаратора ===
```

```
    getBranchCondition(sepId, fromPort) {
        const cacheKey = `${sepId}:${fromPort}`;
        if (cacheKey in this._branchCache) {
            return this._branchCache[cacheKey];
        }

        const sep = AppState.elements[sepId];
        if (!sep || sep.type !== 'separator') return null;

        const inputLogic = this.getPureLogic(sepId);
        const sepContext = this.getConditionFromPort(sepId);

        let branchLogic;
        if (fromPort === 'out-1') {
            branchLogic = inputLogic ? Optimizer.Not(inputLogic) : Optimizer.TrueCond;
        } else {
            branchLogic = inputLogic || Optimizer.TrueCond;
        }

        let result;
        if (sepContext) {
            result = Optimizer.And(sepContext, branchLogic);
        } else {
            result = branchLogic;
        }

        this._branchCache[cacheKey] = result;
        return result;
    },

    // === Получить условие от cond-порта ===
    getConditionFromPort(id) {
        const conn = this.getConn(id, 'cond-0');
        if (!conn) return null;

        const sourceElem = AppState.elements[conn.fromElement];
        if (!sourceElem) return null;

        if (sourceElem.type === 'separator') {
            return this.getBranchCondition(conn.fromElement, conn.fromPort);
        }

        return this.getPureLogic(conn.fromElement);
    },

    // === Основная функция разрешения ===
    resolve(id) {
        if (id in this._resolveCache) {
            return this._resolveCache[id];
        }

        if (this._visiting.has(id)) {
            return null;
        }
        this._visiting.add(id);

        const elem = AppState.elements[id];
        if (!elem) {
            this._visiting.delete(id);
            return null;
        }

        let result = null;

        try {
```

```
            switch (elem.type) {
                case 'input-signal':
                    result = {
                        isValue: true,
                        cond: null,
                        expr: this.toExpr(elem.props.name || id)
                    };
                    break;

                case 'const': {
                    const cond = this.getConditionFromPort(id);
                    result = {
                        isValue: true,
                        cond: cond,
                        expr: Optimizer.Const(Number(elem.props.value) || 0)
                    };
                    break;
                }

                case 'formula': {
                    let cond = this.getConditionFromPort(id);

                    const inConns = this.getConns(id, 'in-');
                    for (const conn of inConns) {
                        const inputNode = this.resolve(conn.fromElement);
                        if (inputNode && inputNode.cond) {
                            cond = this.mergeCond(cond, inputNode.cond);
                        }
                    }

                    const fullExpr = this.buildFormulaExpr(elem);
                    result = {
                        isValue: true,
                        cond: cond,
                        expr: Optimizer.Var(fullExpr)
                    };
                    break;
                }

                default:
                    result = null;
            }
        } finally {
            this._visiting.delete(id);
        }

        this._resolveCache[id] = result;
        return result;
    },

    generate() {
        console.log('=== Генерация кода (граф) ===');
        this.reset();

        try {
            const outputs = Object.values(AppState.elements).filter(e => e.type ===
    'output');

            if (outputs.length === 0) {
                return '/* Нет выходов */';
            }

            const allVariants = [];

            for (const out of outputs) {
```

```
                const conns = this.getConns(out.id, 'in-');

                for (const conn of conns) {
                    console.log(`\n=== Обработка выхода ${out.id}, вход от ${conn.fromElement} ===`);
                    const graph = CodeGenGraph.buildDependencyGraph(conn.fromElement);
                    const result = CodeGenGraph.evalGraphValue(graph);
                    console.log(`Результат: cond=${Optimizer.printCond(result.cond)}, expr=${Optimizer.printExpr(result.expr)}`);

                    if (!result || !result.expr) continue;

                    const cond = result.cond ? Optimizer.simplifyCond(result.cond) : null;
                    const isZero = result.expr.type === 'const' && result.expr.n === 0;

                    if (isZero && !cond) continue;

                    allVariants.push({
                        cond,
                        expr: result.expr,
                        isZero
                    });
                }
            }

            console.log('Варианты:', allVariants.map(v => ({
                cond: Optimizer.printCond(v.cond),
                expr: Optimizer.printExpr(v.expr)
            })));

            if (allVariants.length === 0) return '0';

            const valueVariants = allVariants.filter(v => !v.isZero);
            if (valueVariants.length === 0) return '0';

            let result = Optimizer.Const(0);

            for (let i = valueVariants.length - 1; i >= 0; i--) {
                const v = valueVariants[i];
                if (v.cond) {
                    result = Optimizer.When(v.cond, v.expr, result);
                } else {
                    result = v.expr;
                }
            }

            const simplified = Optimizer.simplifyExpr(result);
            return Optimizer.printExpr(simplified);

        } catch (err) {
            console.error('Ошибка:', err);
            return `/* Ошибка: ${err.message} */`;
        }
    }
};

window.CodeGen = CodeGen;

config.js:
/**
 * Конфигурация приложения
 */

```

```
1906  // Типы сигналов
1907  const SIGNAL_TYPE = {
1908      NUMERIC: 'numeric',   // Числовой сигнал
1909      LOGIC: 'logic',       // Логический (может быть TRUE или FALSE)
1910      TRUE: 'true',         // Явно ИСТИНА
1911      FALSE: 'false',       // Явно ЛОЖЬ
1912      ANY: 'any'            // Любой тип
1913  };
1914
1915  // Типы проекта
1916  const PROJECT_TYPE = {
1917      PARAMETER: 'parameter',
1918      RULE: 'rule'
1919  };
1920
1921  // Конфигурация элементов
1922  const ELEMENT_TYPES = {
1923      'input-signal': {
1924          name: 'Вход',
1925          inputs: 0,
1926          outputs: 1,
1927          outputLabels: ['out'],
1928          outputTypes: [SIGNAL_TYPE.NUMERIC],
1929          color: '#4a90d9',
1930          hasProperties: true,
1931          defaultProps: { name: 'Сигнал', signalType: SIGNAL_TYPE.NUMERIC },
1932          resizable: true,
1933          minWidth: 150,
1934          minHeight: 50
1935      },
1936      'and': {
1937          name: 'И',
1938          inputs: 2,  // По умолчанию 2, но может быть изменено
1939          outputs: 1,
1940          inputLabels: ['A', 'B'],
1941          inputTypes: [SIGNAL_TYPE.LOGIC, SIGNAL_TYPE.LOGIC],
1942          outputLabels: ['результат'],
1943          outputTypes: [SIGNAL_TYPE.LOGIC],
1944          color: '#a855f7',
1945          hasProperties: true,  // ← Теперь есть свойства (для изменения количества
      входов)
1946          resizable: true,
1947          minWidth: 120,
1948          minHeight: 80,
1949          hasConditionPort: true,
1950          conditionPortType: SIGNAL_TYPE.LOGIC,
1951          defaultProps: {
1952              inputCount: 2  // ← Новое свойство
1953          }
1954      },
1955      'or': {
1956          name: 'ИЛИ',
1957          inputs: 2,  // По умолчанию 2
1958          outputs: 1,
1959          inputLabels: ['A', 'B'],
1960          inputTypes: [SIGNAL_TYPE.LOGIC, SIGNAL_TYPE.LOGIC],
1961          outputLabels: ['результат'],
1962          outputTypes: [SIGNAL_TYPE.LOGIC],
1963          color: '#a855f7',
1964          hasProperties: true,  // ← Теперь есть свойства
1965          resizable: true,
1966          minWidth: 120,
1967          minHeight: 80,
1968          hasConditionPort: true,
1969          conditionPortType: SIGNAL_TYPE.LOGIC,
```

```
            defaultProps: {
                inputCount: 2  // ← Новое свойство
            }
        },
        'not': {
            name: 'НЕ',
            inputs: 1,
            outputs: 1,
            inputLabels: ['A'],
            inputTypes: [SIGNAL_TYPE.LOGIC],
            outputLabels: ['¬A'],
            outputTypes: [SIGNAL_TYPE.LOGIC],
            color: '#a855f7',
            hasProperties: false,
            resizable: true,
            minWidth: 100,
            minHeight: 60,
            hasConditionPort: true,
            conditionPortType: SIGNAL_TYPE.LOGIC
        },
        'if': {
            name: 'ЕСЛИ',
            inputs: 2,
            outputs: 1,  // ← Только один выход!
            inputLabels: ['A', 'B'],
            inputTypes: [SIGNAL_TYPE.ANY, SIGNAL_TYPE.ANY],
            outputLabels: ['результат'],  // ← Просто результат
            outputTypes: [SIGNAL_TYPE.LOGIC],  // ← Выход типа LOGIC
            color: '#e94560',
            hasProperties: true,
            defaultProps: { operator: '=' },
            resizable: true,
            minWidth: 120,
            minHeight: 80,
            hasConditionPort: true,
            conditionPortType: SIGNAL_TYPE.LOGIC
        },
        'separator': {  // ← НОВЫЙ ЭЛЕМЕНТ
            name: 'Сепаратор',
            inputs: 1,
            outputs: 2,
            inputLabels: ['сигнал'],
            inputTypes: [SIGNAL_TYPE.LOGIC],
            outputLabels: ['ИСТИНА', 'ЛОЖЬ'],
            outputTypes: [SIGNAL_TYPE.TRUE, SIGNAL_TYPE.FALSE],  // ← TRUE и FALSE
            color: '#f59e0b',
            hasProperties: false,
            resizable: true,
            minWidth: 120,
            minHeight: 80,
            hasConditionPort: true,
            conditionPortType: SIGNAL_TYPE.LOGIC
        },
        'const': {
            name: 'Константа',
            inputs: 0,
            outputs: 1,
            outputLabels: ['out'],
            outputTypes: [SIGNAL_TYPE.NUMERIC],
            color: '#3b82f6',
            hasProperties: true,
            defaultProps: { value: 0 },
            resizable: true,
            minWidth: 120,
            minHeight: 60,
```

```
        hasConditionPort: true,
        conditionPortType: SIGNAL_TYPE.LOGIC
    },
    'formula': {
        name: 'Формула',
        inputs: 2,
        outputs: 1,
        inputLabels: ['in₁', 'in₂'],
        inputTypes: [SIGNAL_TYPE.ANY, SIGNAL_TYPE.ANY],
        outputLabels: ['результат'],
        outputTypes: [SIGNAL_TYPE.NUMERIC],
        color: '#f59e0b',
        hasProperties: true,
        resizable: true,
        minWidth: 140,
        minHeight: 80,
        defaultProps: {
            expression: '',
            inputCount: 2
        },
        hasConditionPort: true,
        conditionPortType: SIGNAL_TYPE.LOGIC
    },
    'output': {
        name: 'Выход',
        inputs: 1,
        outputs: 0,
        inputLabels: ['сигнал'],
        inputTypes: [SIGNAL_TYPE.ANY],
        color: '#10b981',
        hasProperties: true,
        defaultProps: { label: 'Выход', outputGroup: '' },
        resizable: true,
        minWidth: 150,
        minHeight: 60,
    },
};

const VIEWPORT_CONFIG = {
    minZoom: 0.1,
    maxZoom: 3,
    zoomStep: 0.1,
    panSpeed: 1,
    canvasWidth: 5000,
    canvasHeight: 5000
};

const MINIMAP_CONFIG = {
    width: 200,
    height: 150,
    padding: 10
};

connections.js:
/**
 * Модуль работы с соединениями
 */

const Connections = {
    /**
     * Настройка обработчиков порта
     */
    setupPortHandlers(port) {
        port.addEventListener('mousedown', (e) => {
            e.stopPropagation();
```

```
2100
2101            if (port.classList.contains('output')) {
2102                const elemId = port.dataset.element;
2103                const portName = port.dataset.port;
2104                const signalType = getOutputPortType(elemId, portName);
2105
2106                AppState.connectingFrom = {
2107                    element: elemId,
2108                    port: portName
2109                };
2110                AppState.connectingFromType = signalType;
2111
2112                this.highlightCompatiblePorts(signalType);
2113
2114                const svg = document.getElementById('connections-svg');
2115                const startPos = this._getPortCanvasCenter(port);
2116
2117                AppState.tempLine = document.createElementNS('http://www.w3.org/2000/
     svg', 'path');
2118                AppState.tempLine.setAttribute('class', 'temp-connection');
2119                AppState.tempLine.setAttribute('d', `M ${startPos.x} ${startPos.y} L $
     {startPos.x} ${startPos.y}`);
2120                svg.appendChild(AppState.tempLine);
2121            }
2122        });
2123
2124        port.addEventListener('mouseup', (e) => {
2125            e.stopPropagation();
2126            e.preventDefault();
2127
2128            if (AppState.connectingFrom && port.classList.contains('input')) {
2129                const toElement = port.dataset.element;
2130                const toPortName = port.dataset.port;
2131                const inputType = getInputPortType(toElement, toPortName);
2132
2133                if (!areTypesCompatible(AppState.connectingFromType, inputType)) {
2134                    this.clearConnectionState();
2135                    return;
2136                }
2137
2138                if (AppState.connectingFrom.element !== toElement) {
2139                    const targetElem = AppState.elements[toElement];
2140                    const allowMultipleInputs = targetElem?.type === 'output';
2141
2142                    const exists = AppState.connections.some(c =>
2143                        c.toElement === toElement && c.toPort === toPortName
2144                    );
2145
2146                    if (!exists || allowMultipleInputs) {
2147                        AppState.connections.push({
2148                            fromElement: AppState.connectingFrom.element,
2149                            fromPort: AppState.connectingFrom.port,
2150                            toElement,
2151                            toPort: toPortName,
2152                            signalType: AppState.connectingFromType
2153                        });
2154
2155                        port.classList.add('connected');
2156                        this.drawConnections();
2157                        this.clearConnectionState();
2158                        return;
2159                    }
2160                }
2161            }
2162
```

```
2163              this.clearConnectionState();
2164          });
2165
2166          port.addEventListener('mouseenter', () => {
2167              if (AppState.connectingFrom && port.classList.contains('input')) {
2168                  const toPortName = port.dataset.port;
2169                  const inputType = getInputPortType(port.dataset.element, toPortName);
2170
2171                  if (!areTypesCompatible(AppState.connectingFromType, inputType)) {
2172                      if (AppState.tempLine) {
2173                          AppState.tempLine.classList.add('invalid');
2174                      }
2175                  }
2176              }
2177          });
2178
2179          port.addEventListener('mouseleave', () => {
2180              if (AppState.tempLine) {
2181                  AppState.tempLine.classList.remove('invalid');
2182              }
2183          });
2184      },
2185
2186      /**
2187       * Подсветка совместимых портов
2188       */
2189      highlightCompatiblePorts(signalType) {
2190          document.querySelectorAll('.port.input').forEach(port => {
2191              const inputType = getInputPortType(port.dataset.element,
      port.dataset.port);
2192
2193              if (areTypesCompatible(signalType, inputType)) {
2194                  port.classList.add('compatible-highlight');
2195              } else {
2196                  port.classList.add('incompatible');
2197              }
2198          });
2199      },
2200
2201      /**
2202       * Очистка состояния соединения
2203       */
2204      clearConnectionState() {
2205          if (AppState.tempLine) {
2206              AppState.tempLine.remove();
2207              AppState.tempLine = null;
2208          }
2209          AppState.connectingFrom = null;
2210          AppState.connectingFromType = null;
2211
2212          document.querySelectorAll('.port').forEach(port => {
2213              port.classList.remove('compatible-highlight', 'incompatible');
2214          });
2215      },
2216
2217      /**
2218       * Отрисовка временной линии соединения
2219       */
2220      drawTempConnection(e) {
2221      if (!AppState.tempLine || !AppState.connectingFrom) return;
2222
2223      const fromElem = document.getElementById(AppState.connectingFrom.element);
2224      if (!fromElem) return;
2225
2226      const fromPort = fromElem.querySelector(`[data-port="$
```

```
      {AppState.connectingFrom.port}"]`);
2227      if (!fromPort) return;
2228
2229      const startPos = this._getPortCanvasCenter(fromPort);
2230      const endPos = screenToCanvas(e.clientX, e.clientY);
2231
2232      const horizontalDist = Math.abs(endPos.x - startPos.x);
2233      const controlDist = Math.max(horizontalDist * 0.4, 50);
2234
2235      // Тянем всегда от выхода (вектор 1, 0)
2236      const cx1 = startPos.x + controlDist;
2237      const cy1 = startPos.y;
2238
2239      // Вторая точка контроля для плавности за курсором
2240      const cx2 = endPos.x - controlDist;
2241      const cy2 = endPos.y;
2242
2243      AppState.tempLine.setAttribute('d', `M ${startPos.x} ${startPos.y} C ${cx1} $
      {cy1}, ${cx2} ${cy2}, ${endPos.x} ${endPos.y}`);
2244      AppState.tempLine.setAttribute('fill', 'none');
2245  },
2246
2247      /**
2248       * Отрисовка всех соединений
2249       */
2250      drawConnections() {
2251      const svg = document.getElementById('connections-svg');
2252
2253      // 1. Очистка старых линий
2254      svg.querySelectorAll('path:not(.temp-connection)').forEach(p => p.remove());
2255
2256      // 2. Сброс визуального состояния портов
2257      document.querySelectorAll('.port.connected').forEach(port => {
2258          port.classList.remove('connected');
2259      });
2260
2261      // 3. Перебор всех соединений из AppState
2262      AppState.connections.forEach(conn => {
2263          const fromElem = document.getElementById(conn.fromElement);
2264          const toElem = document.getElementById(conn.toElement);
2265
2266          if (!fromElem || !toElem) return;
2267
2268          const fromPort = fromElem.querySelector(`[data-port="${conn.fromPort}"]`);
2269          const toPort = toElem.querySelector(`[data-port="${conn.toPort}"]`);
2270
2271          if (!fromPort || !toPort) return;
2272
2273          fromPort.classList.add('connected');
2274          toPort.classList.add('connected');
2275
2276          const startPos = this._getPortCanvasCenter(fromPort);
2277          const endPos = this._getPortCanvasCenter(toPort);
2278
2279          if (!startPos || !endPos) return;
2280
2281          // Расстояние для изгиба кривой
2282          const horizontalDist = Math.abs(endPos.x - startPos.x);
2283          const verticalDist = Math.abs(endPos.y - startPos.y);
2284          const controlDist = Math.max(horizontalDist * 0.4, 50);
2285
2286          // --- ЛОГИКА ГЕОМЕТРИИ (Вектора касательных) ---
2287          let d;
2288          let cx1 = startPos.x;
2289          let cy1 = startPos.y;
```

```
2290          let cx2 = endPos.x;
2291          let cy2 = endPos.y;
2292
2293          // ВЫХОД (Source): Касательная (1, 0) -> Всегда вправо
2294          cx1 = startPos.x + controlDist;
2295          cy1 = startPos.y;
2296
2297          // ВХОД (Target):
2298          if (conn.toPort === 'cond-0') {
2299              // Технический порт: Касательная (0, 1) в декартовой (вверх)
2300              // В экранных координатах Y инвертирован, поэтому отнимаем от Y
2301              cx2 = endPos.x;
2302              cy2 = endPos.y - controlDist; // Линия заходит сверху вертикально
2303          } else {
2304              // Обычный вход: Касательная (-1, 0) -> Слева направо
2305              cx2 = endPos.x - controlDist;
2306              cy2 = endPos.y;
2307          }
2308
2309          d = `M ${startPos.x} ${startPos.y} C ${cx1} ${cy1}, ${cx2} ${cy2}, ${endPos.x}
      ${endPos.y}`;
2310
2311          const path = document.createElementNS('http://www.w3.org/2000/svg', 'path');
2312          path.setAttribute('d', d);
2313          path.setAttribute('fill', 'none'); // Чтобы не было черных полигонов
2314
2315          // --- ЛОГИКА ЦВЕТА (Классы) ---
2316          let cssClass = 'connection';
2317          const type = conn.signalType;
2318
2319          // Приоритет новым типам сигналов
2320          if (type === SIGNAL_TYPE.TRUE) cssClass += ' true-conn';
2321          else if (type === SIGNAL_TYPE.FALSE) cssClass += ' false-conn';
2322          else if (type === SIGNAL_TYPE.LOGIC) cssClass += ' logic-conn';
2323          else if (type === SIGNAL_TYPE.NUMERIC) cssClass += ' numeric-conn';
2324          else if (type === SIGNAL_TYPE.ANY) cssClass += ' any-conn';
2325
2326          path.setAttribute('class', cssClass);
2327
2328          // Обработчики событий
2329          path.style.pointerEvents = 'stroke';
2330          path.style.cursor = 'pointer';
2331          path.addEventListener('click', () => this.handleConnectionClick(conn));
2332
2333          svg.appendChild(path);
2334      });
2335
2336      if (typeof Outputs !== 'undefined' && Outputs.updateOutputStatus) {
2337          Outputs.updateOutputStatus();
2338      }
2339      Viewport.updateMinimap();
2340  },
2341      /**
2342       * Обработка клика по соединению (удаление)
2343       */
2344      handleConnectionClick(conn) {
2345          if (confirm('Удалить соединение?')) {
2346              AppState.connections = AppState.connections.filter(c =>
2347                  !(c.fromElement === conn.fromElement &&
2348                    c.fromPort === conn.fromPort &&
2349                    c.toElement === conn.toElement &&
2350                    c.toPort === conn.toPort)
2351              );
2352
2353              this.drawConnections();
```

```
        }
    },

    /**
     * Получение центра порта в координатах Canvas
     */
    _getPortCanvasCenter(portEl) {
        if (!portEl) return null;

        const rect = portEl.getBoundingClientRect();
        return screenToCanvas(
            rect.left + rect.width / 2,
            rect.top + rect.height / 2
        );
    }
};

elements.js:
/**
 * Модуль работы с элементами схемы
 */

const Elements = {
    /**
     * Генерация HTML для элемента
     */
        createElementHTML(elemType, elemId, x, y, props = {}, width, height) {
            const config = ELEMENT_TYPES[elemType];
            if (!config) throw new Error(`Неизвестный тип элемента: ${elemType}`);

            const safe = (value, fallback = '') => (value === null || value ===
undefined) ? fallback : String(value);
            const w = width ?? config.minWidth ?? 120;
            const h = height ?? config.minHeight ?? 60;

            const getPortClass = (signalType, direction) => {
                const base = direction === 'output' ? 'port output' : 'port input';
                if (signalType === SIGNAL_TYPE.LOGIC) return `${base} logic-port`;
                if (signalType === SIGNAL_TYPE.NUMBER) return `${base} number-port`;
                return `${base} any-port`;
            };

            // Эта функция buildConditionPort будет вызываться ИНАЧЕ, а не внутри
innerHTML
            // Она тут остается, но ее результат не встраивается в HTML-строку
напрямую, кроме формулы
            const buildConditionPortHTML = () => {
                return `
                    <div class="condition-port-wrapper">
                        <div class="condition-port-label">условие</div>
                        <div class="port input condition-port"
                            data-port="cond-0"
                            data-element="${elemId}"
                            data-signal-type="${SIGNAL_TYPE.LOGIC}"
                            title="Техническое условие">
                        </div>
                    </div>`;
            };


            const buildInputPorts = (count, types = [], labels = []) => {
                let html = '';
                for (let i = 0; i < count; i++) {
                    const type = types[i] ?? types[types.length - 1] ??
SIGNAL_TYPE.ANY;
```

```
2415                     html += `<div class="${getPortClass(type, 'input')}" data-
      port="in-${i}" data-element="${elemId}" data-signal-type="${type}" title="${labels[i]
      || `Вход ${i+1}`}"></div>`;
2416                 }
2417                 return html;
2418             };
2419 
2420             const buildOutputPorts = (count, types = [], labels = []) => {
2421                 let html = '';
2422                 for (let i = 0; i < count; i++) {
2423                     const type = types[i] ?? types[types.length - 1] ??
      SIGNAL_TYPE.ANY;
2424                     html += `<div class="${getPortClass(type, 'output')}" data-
      port="out-${i}" data-element="${elemId}" data-signal-type="${type}" title="${labels[i]
      || `Выход ${i+1}`}"></div>`;
2425                 }
2426                 return html;
2427             };
2428 
2429             const resizeHandles = config.resizable ? `<div class="resize-handle
      handle-se" data-direction="se"></div><div class="resize-handle handle-e" data-
      direction="e"></div><div class="resize-handle handle-s" data-direction="s"></div>` :
      '';
2430             // hasCondClass будет добавляться в addElement
2431             // const hasCondClass = config.hasConditionPort ? 'has-condition-port' :
      '';
2432 
2433             let innerHTML = '';
2434 
2435             if (elemType === 'input-signal') {
2436                 const name = safe(props.name, 'Сигнал');
2437                 const type = props.signalType || SIGNAL_TYPE.NUMBER;
2438                 const symbol = type === SIGNAL_TYPE.LOGIC ? '🔀' : '🔢';
2439                 innerHTML = `
2440                     <div class="element-header" style="background:$
      {config.color};">Источник</div>
2441                     <div class="element-body">
2442                         <div class="element-symbol">
2443                             <span class="input-signal-icon">${symbol}</span>
2444                             <span class="input-signal-name">${name}</span>
2445                         </div>
2446                         <div class="ports-right">
2447                             ${buildOutputPorts(1, [type], ['Выход'])}
2448                         </div>
2449                     </div>`;
2450             }
2451             else if (elemType === 'const') {
2452                 innerHTML = `
2453                     <div class="element-header" style="background:$
      {config.color};">Константа</div>
2454                     <div class="element-body">
2455                         <div class="element-symbol">${props.value ?? 0}</div>
2456                         <div class="ports-right">
2457                             ${buildOutputPorts(1, [SIGNAL_TYPE.NUMBER], ['Значение'])}
2458                         </div>
2459                     </div>`;
2460             }
2461             else if (elemType === 'separator') {
2462                 innerHTML = `
2463                     <div class="element-header" style="background:$
      {config.color};">Сепаратор</div>
2464                     <div class="element-body">
2465                         <div class="ports-left">${buildInputPorts(1,
      config.inputTypes, config.inputLabels)}</div>
2466                         <div class="element-symbol">✓/✗</div>
```

```
                        <div class="ports-right">
                            <div class="port output logic-port true-port" data-port="out-0" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.TRUE}" title="ИСТИНА"></div>
                            <div class="port output logic-port false-port" data-port="out-1" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.FALSE}" title="ЛОЖЬ"></div>
                        </div>
                    </div>`;
            }
            else if (elemType === 'and' || elemType === 'or') {
                const gateSymbol = elemType === 'and' ? '∧' : '∨';
                const inputCount = props.inputCount || config.defaultProps?.inputCount || 2;

                // Генерируем динамические входы
                let inputsHTML = '';
                for (let i = 0; i < inputCount; i++) {
                    inputsHTML += `<div class="port input logic-port" data-port="in-${i}" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.LOGIC}" title="Вход ${i+1}"></div>`;
                }

                innerHTML = `
                    <div class="element-header" style="background:${config.color};">${config.name}</div>
                    <div class="element-body">
                        <div class="ports-left">
                            ${inputsHTML}
                        </div>
                        <div class="element-symbol">${gateSymbol}</div>
                        <div class="ports-right">
                            <div class="port output logic-port" data-port="out-0" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.LOGIC}" title="Результат"></div>
                        </div>
                    </div>`;
            }
            else if (elemType === 'if') {
                const op = safe(props.operator, '=');
                innerHTML = `
                    <div class="element-header" style="background:${config.color};">Условие</div>
                    <div class="element-body">
                        <div class="ports-left">${buildInputPorts(2, config.inputTypes, config.inputLabels)}</div>
                        <div class="element-symbol">${op}</div>
                        <div class="ports-right">
                            ${buildOutputPorts(1, [SIGNAL_TYPE.LOGIC], ['результат'])}
                        </div>
                    </div>`;
            }
            else if (elemType === 'not') {
                innerHTML = `
                    <div class="element-header" style="background:${config.color};">НЕ</div>
                    <div class="element-body">
                        <div class="ports-left">${buildInputPorts(1, [SIGNAL_TYPE.LOGIC], ['A'])}</div>
                        <div class="element-symbol">¬</div>
                        <div class="ports-right">
                            ${buildOutputPorts(1, [SIGNAL_TYPE.LOGIC], ['¬A'])}
                        </div>
                    </div>`;
            }
```

```
2518              else if (elemType === 'formula') {
2519                  const inputCount = props.inputCount || config.defaultProps?.inputCount
      || config.inputs || 2;
2520                  const expression = safe(props.expression);
2521                  const displayExpression = expression
2522                      ? (expression.length > 12 ? `${expression.slice(0, 12)}…` :
      expression)
2523                      : 'f(x)';
2524
2525                  innerHTML = `
2526                      ${buildConditionPortHTML()}
2527                      <div class="element-header" style="background:$
      {config.color};">Формула</div>
2528                      <div class="element-body">
2529                          <div class="ports-left">${buildInputPorts(inputCount,
      config.inputTypes, config.inputLabels)}</div>
2530                          <div class="element-symbol">${displayExpression}</div>
2531                          <div class="ports-right">
2532                              ${buildOutputPorts(1, [SIGNAL_TYPE.NUMBER],
      ['Результат'])}
2533                          </div>
2534                      </div>`;
2535              }
2536              else if (elemType === 'output') {
2537                  innerHTML = `
2538                      <div class="element-header" style="background:$
      {config.color};">Выход</div>
2539                      <div class="element-body">
2540                          <div class="ports-left">
2541                              ${buildInputPorts(1, [SIGNAL_TYPE.ANY], ['сигнал'])}
2542                          </div>
2543                          <div class="element-symbol">${safe(props.label, 'Выход')}</
      div>
2544                          <div class="ports-right"></div>
2545                      </div>`;
2546              } else { // Для любых других (fallback)
2547                  innerHTML = `
2548                      <div class="element-header" style="background:${config.color};">$
      {config.name}</div>
2549                      <div class="element-body">
2550                          <div class="ports-left">${buildInputPorts(config.inputs || 0,
      config.inputTypes, config.inputLabels)}</div>
2551                          <div class="element-symbol">${config.name}</div>
2552                          <div class="ports-right">
2553                              ${buildOutputPorts(config.outputs || 0,
      config.outputTypes, config.outputLabels)}
2554                          </div>
2555                      </div>`;
2556              }
2557
2558
2559              const html = `
2560                  <div class="element ${elemType}" id="${elemId}"
2561                      style="left:${x}px; top:${y}px; width:${w}px; height:${h}px;"
      data-type="${elemType}">
2562                      ${innerHTML}
2563                      ${resizeHandles}
2564                  </div>`;
2565
2566              return { html, width: w, height: h };
2567          },
2568
2569      /**
2570       * Добавление элемента
2571       */
```

```
2572          addElement(elemType, x, y, props = {}, elemId = null, customWidth = null,
      customHeight = null) {
2573              const config = ELEMENT_TYPES[elemType];
2574              if (!config) {
2575                  console.error(`Неизвестный тип элемента: ${elemType}`);
2576                  return null;
2577              }
2578
2579              if (!elemId) {
2580                  elemId = `${elemType}-${++AppState.elementCounter}`;
2581              }
2582
2583              let width = customWidth;
2584              let height = customHeight;
2585
2586              if (width === null || width === undefined) {
2587                  width = config.minWidth || 140;
2588              }
2589              if (height === null || height === undefined) {
2590                  height = config.minHeight || 70;
2591              }
2592
2593              try {
2594                  const result = this.createElementHTML(elemType, elemId, x, y, props,
      width, height);
2595                  if (!result || !result.html) {
2596                      console.error('createElementHTML вернул пустой результат');
2597                      return null;
2598                  }
2599
2600                  const workspace = document.getElementById('workspace');
2601                  const wrapper = document.createElement('div');
2602                  wrapper.innerHTML = result.html.trim();
2603                  const element = wrapper.firstElementChild;
2604                  if (!element) {
2605                      console.error('Не удалось создать DOM элемент из HTML');
2606                      return null;
2607                  }
2608
2609                  // Добавляем класс для отступа
2610                  if (config.hasConditionPort) {
2611                      element.classList.add('has-condition-port');
2612                  }
2613
2614                  workspace.appendChild(element);
2615
2616                  AppState.elements[elemId] = {
2617                      id: elemId,
2618                      type: elemType,
2619                      x,
2620                      y,
2621                      width: result.width || width,
2622                      height: result.height || height,
2623                      props: { ...(config.defaultProps || {}), ...(props || {}) }
2624                  };
2625
2626                  // ЕСЛИ У ЭЛЕМЕНТА ЕСТЬ COND-ПОРТ (И ОН НЕ ФОРМУЛА, КОТОРАЯ УЖЕ ИМЕЕТ
      ЕГО В HTML)
2627                  if (config.hasConditionPort && elemType !== 'formula') {
2628                      const condPortWrapper = document.createElement('div');
2629                      condPortWrapper.innerHTML = `
2630                          <div class="condition-port-wrapper">
2631                              <div class="condition-port-label">условие</div>
2632                              <div class="port input condition-port"
2633                                  data-port="cond-0"
```

```
                                data-element="${elemId}"
                                data-signal-type="${SIGNAL_TYPE.LOGIC}"
                                title="Техническое условие">
                            </div>
                        </div>`;
                    element.prepend(condPortWrapper.firstElementChild); // Вставляем в самое начало элемента
                }


                this.setupElementHandlers(elemId); // Передаем ID элемента

                // Порты инициализируются внутри setupElementHandlers, нет нужды здесь
                // element.querySelectorAll('.port').forEach(port => {
                //     Connections.setupPortHandlers(port);
                // });

                Connections.drawConnections(); // Перерисовываем соединения, чтобы учесть новые порты
                Viewport.updateMinimap();
                return elemId;
            } catch (err) {
                console.error(`Ошибка при добавлении элемента ${elemType}:`, err);
                return null;
            }
        },

    /**
     * Обновление входов логического элемента (AND, OR)
     */
    updateLogicGateInputs(elemId, inputCount) {
        const elem = document.getElementById(elemId);
        if (!elem) return;

        const portsLeft = elem.querySelector('.ports-left');
        if (!portsLeft) return;

        // Удаляем соединения к портам, которые больше не существуют
        AppState.connections = AppState.connections.filter(c => {
            if (c.toElement === elemId && c.toPort.startsWith('in-')) {
                const portNum = parseInt(c.toPort.split('-')[1], 10);
                return portNum < inputCount;
            }
            return true;
        });

        // Генерируем новые входы
        let inputsHTML = '';
        for (let i = 0; i < inputCount; i++) {
            inputsHTML += `
                <div class="port input logic-port"
                    data-port="in-${i}"
                    data-element="${elemId}"
                    data-signal-type="${SIGNAL_TYPE.LOGIC}"
                    title="Вход ${i+1}">
                </div>
            `;
        }
        portsLeft.innerHTML = inputsHTML;

        // Переподключаем обработчики
        portsLeft.querySelectorAll('.port').forEach(port =>
            Connections.setupPortHandlers(port)
        );

```

```
        Connections.drawConnections();
    },

    /**
     * Удаление элемента
     */
    deleteElement(elemId) {
        AppState.connections = AppState.connections.filter(c =>
            c.fromElement !== elemId && c.toElement !== elemId
        );

        const elem = document.getElementById(elemId);
        if (elem) elem.remove();

        delete AppState.elements[elemId];

        if (AppState.selectedElement === elemId) {
            AppState.selectedElement = null;
        }

        Connections.drawConnections();
        Viewport.updateMinimap();
    },

    /**
     * Выделение элемента
     */
    selectElement(elemId) {
        if (AppState.selectedElement) {
            const oldElem = document.getElementById(AppState.selectedElement);
            if (oldElem) oldElem.classList.remove('selected');
        }

        AppState.selectedElement = elemId;
        const elem = document.getElementById(elemId);
        if (elem) elem.classList.add('selected');

        const elemData = AppState.elements[elemId];
        if (elemData) {
            document.getElementById('selection-info').textContent =
                `Выбрано: ${ELEMENT_TYPES[elemData.type]?.name || elemData.type}`;
        }
    },

    /**
     * Снять выделение
     */
    deselectAll() {
        if (AppState.selectedElement) {
            const elem = document.getElementById(AppState.selectedElement);
            if (elem) elem.classList.remove('selected');
            AppState.selectedElement = null;
        }
        document.getElementById('selection-info').textContent = '';
    },

    /**
     * Настройка обработчиков элемента
     */
    setupElementHandlers(elemId) {
        try {
            const elem = document.getElementById(elemId);
            if (!elem) return;

            elem.addEventListener('mousedown', (e) => {
```

```
                if (e.target.classList.contains('port')) return;
                if (e.target.classList.contains('resize-handle')) return;

                e.preventDefault();
                e.stopPropagation();

                this.selectElement(elemId);

                AppState.draggingElement = elemId;
                const canvasPos = screenToCanvas(e.clientX, e.clientY);
                const elemData = AppState.elements[elemId];
                AppState.dragOffset.x = canvasPos.x - elemData.x;
                AppState.dragOffset.y = canvasPos.y - elemData.y;
            });

            elem.addEventListener('dblclick', (e) => {
                if (e.target.classList.contains('port')) return;
                const config = ELEMENT_TYPES[AppState.elements[elemId].type];
                if (config?.hasProperties) {
                    Modal.showPropertiesModal(elemId);
                }
            });

            elem.addEventListener('contextmenu', (e) => {
                e.preventDefault();
                this.showContextMenu(e.clientX, e.clientY, elemId);
            });

            const handles = elem.querySelectorAll('.resize-handle');
            handles.forEach(handle => this.setupResizeHandlers(handle, elemId));

            const ports = elem.querySelectorAll('.port');
            ports.forEach(port => Connections.setupPortHandlers(port));

        } catch (err) {
            console.error('setupElementHandlers error for', elemId, err);
        }
    },

    /**
     * Контекстное меню
     */
    showContextMenu(x, y, elemId) {
        const menu = document.getElementById('context-menu');
        menu.style.left = `${x}px`;
        menu.style.top = `${y}px`;
        menu.style.display = 'block';
        menu.dataset.elementId = elemId;
    },

    /**
     * Настройка resize
     */
    setupResizeHandlers(handle, elemId) {
        handle.addEventListener('mousedown', (e) => {
            e.stopPropagation();
            e.preventDefault();

            const elemData = AppState.elements[elemId];

            AppState.resizing = {
                elemId: elemId,
                handle: handle.dataset.direction,
                startX: e.clientX,
                startY: e.clientY,
```

```
                startWidth: elemData.width,
                startHeight: elemData.height,
                startLeft: elemData.x,
                startTop: elemData.y
            };
        });
    },

    /**
     * Обработка resize
     */
    handleResize(e) {
        if (!AppState.resizing) return;

        const { elemId, handle, startX, startY, startWidth, startHeight, startLeft,
    startTop } = AppState.resizing;
        const elem = document.getElementById(elemId);
        const elemData = AppState.elements[elemId];
        const config = ELEMENT_TYPES[elemData.type];

        const dx = (e.clientX - startX) / AppState.viewport.zoom;
        const dy = (e.clientY - startY) / AppState.viewport.zoom;

        let newWidth = startWidth;
        let newHeight = startHeight;
        let newLeft = startLeft;
        let newTop = startTop;

        if (handle.includes('e')) {
            newWidth = Math.max(config.minWidth, startWidth + dx);
        }
        if (handle.includes('w')) {
            newWidth = Math.max(config.minWidth, startWidth - dx);
            newLeft = startLeft + (startWidth - newWidth);
        }
        if (handle.includes('s')) {
            newHeight = Math.max(config.minHeight, startHeight + dy);
        }
        if (handle.includes('n')) {
            newHeight = Math.max(config.minHeight, startHeight - dy);
            newTop = startTop + (startHeight - newHeight);
        }

        elem.style.width = `${newWidth}px`;
        elem.style.height = `${newHeight}px`;
        elem.style.left = `${newLeft}px`;
        elem.style.top = `${newTop}px`;

        elemData.width = newWidth;
        elemData.height = newHeight;
        elemData.x = newLeft;
        elemData.y = newTop;

        Connections.drawConnections();
    },

    /**
     * Обработка перетаскивания элемента
     */
    handleDrag(e) {
        if (!AppState.draggingElement) return;

        const canvasPos = screenToCanvas(e.clientX, e.clientY);
        const x = canvasPos.x - AppState.dragOffset.x;
        const y = canvasPos.y - AppState.dragOffset.y;
```

```
2891
2892            const elemId = AppState.draggingElement;
2893            const elem = document.getElementById(elemId);
2894            const elemData = AppState.elements[elemId];
2895
2896            elem.style.left = `${x}px`;
2897            elem.style.top = `${y}px`;
2898
2899            elemData.x = x;
2900            elemData.y = y;
2901
2902            Connections.drawConnections();
2903        },
2904
2905        /**
2906         * Обновление входов формулы
2907         */
2908        updateFormulaInputs(elemId, inputCount) {
2909            const elem = document.getElementById(elemId);
2910            if (!elem) return;
2911
2912            const portsLeft = elem.querySelector('.ports-left');
2913            if (!portsLeft) return;
2914
2915            AppState.connections = AppState.connections.filter(c => {
2916                if (c.toElement === elemId && c.toPort.startsWith('in-')) {
2917                    const portNum = parseInt(c.toPort.split('-')[1], 10);
2918                    return portNum < inputCount;
2919                }
2920                return true;
2921            });
2922
2923            let inputsHTML = '';
2924            for (let i = 0; i < inputCount; i++) {
2925                inputsHTML += `
2926                    <div class="port input any-port"
2927                         data-port="in-${i}"
2928                         data-element="${elemId}"
2929                         data-signal-type="${SIGNAL_TYPE.ANY}"
2930                         title="in${i} (Любой)">
2931                    </div>
2932                `;
2933            }
2934            portsLeft.innerHTML = inputsHTML;
2935
2936            portsLeft.querySelectorAll('.port').forEach(port =>
2937                Connections.setupPortHandlers(port)
2938            );
2939
2940            Connections.drawConnections();
2941        },
2942
2943        /**
2944         * Рассчитать оптимальный размер элемента на основе количества портов
2945         */
2946        calculateOptimalHeight(elemId, inputCount, outputCount = 1) {
2947            const elem = AppState.elements[elemId];
2948            if (!elem) return null;
2949
2950            const config = ELEMENT_TYPES[elem.type];
2951            if (!config || !config.resizable) return null;
2952
2953            // Базовая высота
2954            let baseHeight = config.minHeight || 60;
2955
```

```
2956          // Каждый порт требует примерно 25-30px высоты
2957          const portSpacing = 28;
2958          const maxPorts = Math.max(inputCount, outputCount);
2959
2960          // Добавляем высоту для портов (кроме первого, который уже в baseHeight)
2961          const additionalHeight = (maxPorts - 1) * portSpacing;
2962          const newHeight = Math.max(baseHeight, baseHeight + additionalHeight);
2963
2964          return newHeight;
2965      },
2966
2967      /**
2968       * Обновление размера элемента при изменении портов
2969       */
2970      updateElementSize(elemId) {
2971          const elem = document.getElementById(elemId);
2972          const elemData = AppState.elements[elemId];
2973
2974          if (!elem || !elemData) return;
2975
2976          const config = ELEMENT_TYPES[elemData.type];
2977          if (!config || !config.resizable) return;
2978
2979          let inputCount = 0;
2980          let outputCount = config.outputs || 1;
2981
2982          // Определяем количество входов
2983          if (elemData.type === 'and' || elemData.type === 'or' || elemData.type ===
      'formula') {
2984              inputCount = elemData.props.inputCount || config.inputs || 2;
2985          } else {
2986              inputCount = config.inputs || 0;
2987          }
2988
2989          // Рассчитываем новую высоту
2990          const newHeight = this.calculateOptimalHeight(elemId, inputCount,
      outputCount);
2991
2992          if (newHeight && newHeight !== elemData.height) {
2993              elemData.height = newHeight;
2994              elem.style.height = `${newHeight}px`;
2995
2996              // Перерисовываем соединения, т.к. изменился размер элемента
2997              Connections.drawConnections();
2998              Viewport.updateMinimap();
2999          }
3000      }
3001
3002
3003  };
3004
3005  modal.js:
3006  /**
3007   * Модуль модальных окон
3008   */
3009
3010  const Modal = {
3011      /**
3012       * Инициализация модальных окон
3013       */
3014      init() {
3015          // Модальное окно свойств элемента
3016          document.getElementById('modal-save').addEventListener('click', () => {
3017              this.saveElementProperties();
3018          });
```

```
3019
3020          document.getElementById('modal-cancel').addEventListener('click', () => {
3021              this.hideModal('modal-overlay');
3022          });
3023
3024          document.getElementById('modal-overlay').addEventListener('click', (e) => {
3025              if (e.target.id === 'modal-overlay') {
3026                  this.hideModal('modal-overlay');
3027              }
3028          });
3029
3030          // Модальное окно свойств проекта
3031          document.getElementById('project-modal-save').addEventListener('click', () =>
     {
3032              this.saveProjectProperties();
3033          });
3034
3035          document.getElementById('project-modal-cancel').addEventListener('click', ()
     => {
3036              this.hideModal('project-modal-overlay');
3037          });
3038
3039          document.getElementById('project-modal-overlay').addEventListener('click', (e)
     => {
3040              if (e.target.id === 'project-modal-overlay') {
3041                  this.hideModal('project-modal-overlay');
3042              }
3043          });
3044      },
3045
3046      /**
3047       * Показать модальное окно
3048       */
3049      showModal(modalId) {
3050          document.getElementById(modalId).style.display = 'flex';
3051      },
3052
3053      /**
3054       * Скрыть модальное окно
3055       */
3056      hideModal(modalId) {
3057          document.getElementById(modalId).style.display = 'none';
3058      },
3059
3060      /**
3061       * Показать свойства элемента
3062       */
3063      showPropertiesModal(elemId) {
3064          const elemData = AppState.elements[elemId];
3065          const elemType = elemData.type;
3066          const props = elemData.props;
3067          const config = ELEMENT_TYPES[elemType];
3068
3069          const modalOverlay = document.getElementById('modal-overlay');
3070          const modalTitle = document.getElementById('modal-title');
3071          const modalContent = document.getElementById('modal-content');
3072
3073          modalTitle.textContent = `Свойства: ${config.name}`;
3074
3075          let contentHTML = '';
3076
3077          if (elemType === 'input-signal') {
3078              const signalType = props.signalType || SIGNAL_TYPE.NUMBER;
3079              contentHTML = `
3080                  <div class="modal-row">
```

```
                    <label>Название сигнала:</label>
                    <input type="text" id="prop-name" value="${props.name ||
'Сигнал'}">
                </div>
                <div class="modal-row">
                    <label>Тип сигнала:</label>
                    <select id="prop-signal-type">
                        <option value="${SIGNAL_TYPE.NUMBER}" ${signalType ===
SIGNAL_TYPE.NUMBER ? 'selected' : ''}>Числовой</option>
                        <option value="${SIGNAL_TYPE.LOGIC}" ${signalType ===
SIGNAL_TYPE.LOGIC ? 'selected' : ''}>Логический</option>
                    </select>
                </div>
            `;
        } else if (elemType === 'if') {
            contentHTML = `
                <div class="modal-row">
                    <label>Оператор сравнения:</label>
                    <select id="prop-operator">
                        <option value="=" ${props.operator === '=' ? 'selected' : ''}
>=  (равно)</option>
                        <option value=">" ${props.operator === '>' ? 'selected' : ''}
>>  (больше)</option>
                        <option value="<" ${props.operator === '<' ? 'selected' : ''}
><  (меньше)</option>
                        <option value=">=" ${props.operator === '>=' ? 'selected' :
''}>= (больше или равно)</option>
                        <option value="<=" ${props.operator === '<=' ? 'selected' :
''}>= (меньше или равно)</option>
                        <option value="!=" ${props.operator === '!=' ? 'selected' :
''}>!= (не равно)</option>
                    </select>
                </div>
            `;
        } else if (elemType === 'and' || elemType === 'or') {
            contentHTML = `
                <div class="modal-row">
                    <label>Количество входов:</label>
                    <input type="number" id="prop-input-count" value="$
{props.inputCount || 2}" min="2" max="10">
                </div>
                <div class="modal-row">
                    <p style="color: #aaa; font-size: 12px;">
                        Измените количество входных портов для этого логического
элемента.
                        Лишние соединения будут автоматически удалены.
                    </p>
                </div>
            `;
        } else if (elemType === 'const') {
            contentHTML = `
                <div class="modal-row">
                    <label>Значение:</label>
                    <input type="number" id="prop-value" value="${props.value ?? 0}"
step="any">
                </div>
            `;
        } else if (elemType === 'formula') {
            let signalsHTML = '';
            AppState.connections.forEach(conn => {
                if (conn.toElement === elemId) {
                    const fromElem = AppState.elements[conn.fromElement];
                    if (fromElem) {
                        const signalName = fromElem.props?.name || fromElem.id;
                        signalsHTML += `<div class="signal-item" data-signal="$
```

```
      {signalName}">${signalName} (${conn.toPort})</div>`;
3134                      }
3135                  }
3136              });
3137
3138              contentHTML = `
3139                  <div class="modal-row">
3140                      <label>Количество входов:</label>
3141                      <input type="number" id="prop-input-count" value="$
      {props.inputCount || 2}" min="1" max="10">
3142                  </div>
3143                  <div class="modal-row">
3144                      <label>Входные сигналы (двойной клик для вставки):</label>
3145                      <div class="signal-list" id="signal-list">
3146                          ${signalsHTML || '<div style="color:#888;padding:5px;">Нет
      подключённых сигналов</div>'}
3147                      </div>
3148                  </div>
3149                  <div class="modal-row">
3150                      <label>Выражение:</label>
3151                      <textarea id="prop-expression">${props.expression || ''}</
      textarea>
3152                  </div>
3153              `;
3154          } else if (elemType === 'output') {
3155              contentHTML = `
3156                  <div class="modal-row">
3157                      <label>Название выхода:</label>
3158                      <input type="text" id="prop-label" value="${props.label ||
      'Выход'}">
3159                  </div>
3160                  <div class="modal-row">
3161                      <label>Группировка (опционально):</label>
3162                      <input type="text" id="prop-output-group" value="$
      {props.outputGroup || ''}" placeholder="для логической группировки выходов">
3163                  </div>
3164              `;
3165          }
3166
3167
3168          modalContent.innerHTML = contentHTML;
3169          modalOverlay.dataset.elementId = elemId;
3170          this.showModal('modal-overlay');
3171
3172          // Обработчик вставки сигналов для формулы
3173          if (elemType === 'formula') {
3174              document.querySelectorAll('.signal-item').forEach(item => {
3175                  item.addEventListener('dblclick', () => {
3176                      const signal = item.dataset.signal;
3177                      const textarea = document.getElementById('prop-expression');
3178                      textarea.value += signal;
3179                      textarea.focus();
3180                  });
3181              });
3182          }
3183      },
3184
3185      /**
3186       * Сохранить свойства элемента
3187       */
3188  /**
3189   * Сохранить свойства элемента
3190   */
3191      saveElementProperties() {
3192          try {
```

```
3193              const modalOverlay = document.getElementById('modal-overlay');
3194              const elemId = modalOverlay.dataset.elementId;
3195              const elemData = AppState.elements[elemId];
3196              const elemType = elemData.type;
3197              const elem = document.getElementById(elemId);
3198
3199              if (elemType === 'input-signal') {
3200                  const name = document.getElementById('prop-name').value || 'Сигнал';
3201                  const signalType = document.getElementById('prop-signal-type').value;
3202
3203                  const oldSignalType = elemData.props.signalType;
3204                  elemData.props.name = name;
3205                  elemData.props.signalType = signalType;
3206
3207                  if (oldSignalType !== signalType) {
3208                      AppState.connections = AppState.connections.filter(conn => {
3209                          if (conn.fromElement === elemId) {
3210                              const toPortIndex = parseInt(conn.toPort.split('-')[1]);
3211                              const inputType = getInputPortType(conn.toElement,
      toPortIndex);
3212                              return areTypesCompatible(signalType, inputType);
3213                          }
3214                          return true;
3215                      });
3216                  }
3217
3218                                  // Перерисовываем элемент
3219                  // Перерисовываем элемент
3220                  const { html } = Elements.createElementHTML(
3221                      elemType,
3222                      elemId,
3223                      elemData.x,
3224                      elemData.y,
3225                      elemData.props,
3226                      elemData.width,
3227                      elemData.height
3228                  );
3229                  elem.outerHTML = html;
3230
3231                  // Находим новый DOM-элемент
3232                  const newElem = document.getElementById(elemId);
3233
3234                  // Заново навешиваем обработчики на новый элемент
3235                  Elements.setupElementHandlers(elemId);
3236
3237                  if (AppState.selectedElement === elemId && newElem) {
3238                      newElem.classList.add('selected');
3239                  }
3240
3241                  Connections.drawConnections();
3242
3243              } else if (elemType === 'if') {
3244                  const operator = document.getElementById('prop-operator').value;
3245                  elemData.props.operator = operator;
3246                  const symbol = elem.querySelector('.element-symbol');
3247                  if (symbol) symbol.textContent = operator;
3248
3249              } else if (elemType === 'const') {
3250                  const value = parseFloat(document.getElementById('prop-value').value)
      || 0;
3251                  elemData.props.value = value;
3252                  const symbol = elem.querySelector('.element-symbol');
3253                  if (symbol) symbol.textContent = String(value);
3254
3255              } else if (elemType === 'formula') {
```

```
3256                 const expression = document.getElementById('prop-expression').value;
3257                 const inputCount = parseInt(document.getElementById('prop-input-
      count').value) || 2;
3258 
3259                 elemData.props.expression = expression;
3260                 elemData.props.inputCount = inputCount;
3261 
3262                 const symbol = elem.querySelector('.element-symbol');
3263                 if (symbol) {
3264                     symbol.textContent = expression.length > 12 ? `$
      {expression.slice(0, 12)}…` : (expression || 'f(x)');
3265                 }
3266 
3267                 Elements.updateFormulaInputs(elemId, inputCount);
3268                 Elements.updateElementSize(elemId);  // ← Добавляем это
3269             } else if (elemType === 'and' || elemType === 'or') {
3270                 const inputCount = parseInt(document.getElementById('prop-input-
      count').value) || 2;
3271                 elemData.props.inputCount = inputCount;
3272 
3273                 Elements.updateLogicGateInputs(elemId, inputCount);
3274                 Elements.updateElementSize(elemId);  // ← Добавляем это
3275 
3276                 const symbol = elem.querySelector('.element-symbol');
3277                 if (symbol) {
3278                     symbol.textContent = elemType === 'and' ? '∧' : '∨';
3279                 }
3280 
3281         } else if (elemType === 'output') {
3282                 const label = document.getElementById('prop-label').value || 'Выход';
3283                 const outputGroup = document.getElementById('prop-output-group').value
      || '';
3284 
3285                 elemData.props.label = label;
3286                 elemData.props.outputGroup = outputGroup;
3287 
3288                 const symbol = elem.querySelector('.element-symbol');
3289                 if (symbol) symbol.textContent = label;
3290             }
3291 
3292             this.hideModal('modal-overlay');
3293 
3294         } catch (error) {
3295             console.error('❌ Ошибка при сохранении свойств:', error);
3296             alert('Ошибка сохранения: ' + error.message);
3297         }
3298     },
3299 
3300     /**
3301      * Показать свойства проекта
3302      */
3303     showProjectPropertiesModal() {
3304         const content = document.getElementById('project-modal-content');
3305         const project = AppState.project;
3306 
3307         // Генерируем HTML для списка выходов только если модуль загружен
3308         let outputsHtml = '';
3309         if (typeof Outputs !== 'undefined' && AppState.outputs) {
3310             const logicalOutputsHtml = AppState.outputs.logical.length > 0
3311                 ? AppState.outputs.logical.map(output => `
3312                     <div class="output-item"
3313                         data-element-id="${output.elementId}"
3314                         onmouseenter="Outputs.highlightOutput('${output.elementId}',
      true)"
3315                         onmouseleave="Outputs.highlightOutput('${output.elementId}',
```

```
false)"
                        onclick="Outputs.navigateToOutput('${output.elementId}');
Modal.hideModal('project-modal-overlay');">
                        <span class="output-icon">${output.portLabel === 'Да' ? '✅' :
'❌'}</span>
                        <span class="output-name">${output.elementName}</span>
                        <span class="output-port">→ ${output.portLabel}</span>
                    </div>
                `).join('')
                : '<div class="no-outputs">Нет логических выходов</div>';

            const numericOutputsHtml = AppState.outputs.numeric.length > 0
                ? AppState.outputs.numeric.map(output => `
                    <div class="output-item numeric"
                        data-element-id="${output.elementId}"
                        onmouseenter="Outputs.highlightOutput('${output.elementId}',
true)"
                        onmouseleave="Outputs.highlightOutput('${output.elementId}',
false)"
                        onclick="Outputs.navigateToOutput('${output.elementId}');
Modal.hideModal('project-modal-overlay');">
                        <span class="output-icon">🔢</span>
                        <span class="output-name">${output.elementName}</span>
                        <span class="output-port">→ значение</span>
                    </div>
                `).join('')
                : '<div class="no-outputs">Нет числовых выходов</div>';

            outputsHtml = `
                <div class="modal-row">
                    <label>Выходные сигналы схемы:</label>
                    <div class="outputs-container">
                        <div class="outputs-section">
                            <div class="outputs-section-title">
                                <span class="section-icon">🔀</span>
                                Логические выходы (${AppState.outputs.logical.length})
                            </div>
                            <div class="outputs-list">
                                ${logicalOutputsHtml}
                            </div>
                        </div>
                        <div class="outputs-section">
                            <div class="outputs-section-title">
                                <span class="section-icon">📐</span>
                                Числовые выходы (${AppState.outputs.numeric.length})
                            </div>
                            <div class="outputs-list">
                                ${numericOutputsHtml}
                            </div>
                        </div>
                    </div>
                    <div class="outputs-hint">
                        💡 Выходами автоматически становятся элементы, чьи выходные
порты не подключены к другим элементам.
                        Кликните на выход, чтобы перейти к нему на схеме.
                    </div>
                </div>
            `;
        }

        content.innerHTML = `
            <div class="modal-row">
                <label>Код проекта:</label>
                <input type="text" id="project-code" value="${project.code || ''}"
placeholder="Уникальный идентификатор">
```

```
3373             </div>
3374
3375             <div class="modal-row">
3376                 <label>Тип проекта:</label>
3377                 <div class="project-type-selector">
3378                     <div class="project-type-btn ${project.type ===
     PROJECT_TYPE.PARAMETER ? 'active' : ''}" data-type="${PROJECT_TYPE.PARAMETER}">
3379                         <div class="type-icon">📊</div>
3380                         <div class="type-name">Параметр</div>
3381                         <div class="type-desc">Вычисляемое значение</div>
3382                     </div>
3383                     <div class="project-type-btn ${project.type ===
     PROJECT_TYPE.RULE ? 'active' : ''}" data-type="${PROJECT_TYPE.RULE}">
3384                         <div class="type-icon">📋</div>
3385                         <div class="type-name">Правило</div>
3386                         <div class="type-desc">Логическое условие</div>
3387                     </div>
3388                 </div>
3389             </div>
3390
3391             <div id="parameter-fields" class="conditional-fields ${project.type ===
     PROJECT_TYPE.PARAMETER ? 'visible' : ''}">
3392                 <div class="modal-row">
3393                     <label>Размерность:</label>
3394                     <input type="text" id="project-dimension" value="$
     {project.dimension || ''}" placeholder="Например: м/с, кг, °C">
3395                 </div>
3396             </div>
3397
3398             <div id="rule-fields" class="conditional-fields ${project.type ===
     PROJECT_TYPE.RULE ? 'visible' : ''}">
3399                 <div class="modal-row">
3400                     <label>Возможная причина:</label>
3401                     <textarea id="project-possible-cause" placeholder="Описание
     возможной причины срабатывания правила">${project.possibleCause || ''}</textarea>
3402                 </div>
3403                 <div class="modal-row">
3404                     <label>Методические указания:</label>
3405                     <textarea id="project-guidelines" placeholder="Инструкции и
     рекомендации при срабатывании правила">${project.guidelines || ''}</textarea>
3406                 </div>
3407             </div>
3408
3409             ${outputsHtml}
3410         `;
3411
3412         // Обработчики переключения типа
3413         content.querySelectorAll('.project-type-btn').forEach(btn => {
3414             btn.addEventListener('click', () => {
3415                 content.querySelectorAll('.project-type-btn').forEach(b =>
     b.classList.remove('active'));
3416                 btn.classList.add('active');
3417
3418                 const type = btn.dataset.type;
3419                 document.getElementById('parameter-
     fields').classList.toggle('visible', type === PROJECT_TYPE.PARAMETER);
3420                 document.getElementById('rule-fields').classList.toggle('visible',
     type === PROJECT_TYPE.RULE);
3421             });
3422         });
3423
3424         this.showModal('project-modal-overlay');
3425     },
3426
3427     /**
```

```
3428        * Сохранить свойства проекта
3429        */
3430       saveProjectProperties() {
3431           const activeTypeBtn = document.querySelector('.project-type-btn.active');
3432           const type = activeTypeBtn ? activeTypeBtn.dataset.type :
      PROJECT_TYPE.PARAMETER;
3433
3434           AppState.project.code = document.getElementById('project-code').value;
3435           AppState.project.type = type;
3436
3437           if (type === PROJECT_TYPE.PARAMETER) {
3438               AppState.project.dimension = document.getElementById('project-
      dimension').value;
3439               AppState.project.possibleCause = '';
3440               AppState.project.guidelines = '';
3441           } else {
3442               AppState.project.dimension = '';
3443               AppState.project.possibleCause = document.getElementById('project-
      possible-cause').value;
3444               AppState.project.guidelines = document.getElementById('project-
      guidelines').value;
3445           }
3446
3447           this.hideModal('project-modal-overlay');
3448       }
3449   };
3450
3451   output.js:
3452   /**
3453    * Модуль управления выходными сигналами
3454    */
3455
3456   const Outputs = {
3457       /**
3458        * Обновление статуса выходных элементов
3459        * Вызывается при каждом изменении схемы
3460        */
3461       updateOutputStatus() {
3462           this.clearAllOutputHighlights();
3463           AppState.outputs.logical = [];
3464           AppState.outputs.numeric = [];
3465           updateFrameChildren();
3466
3467           // Обработка элементов-выходов
3468           Object.values(AppState.elements).forEach(elem => {
3469               if (!elem || elem.type !== 'output') return;
3470
3471               // Проверяем, к чему подключен вход этого выхода
3472               const inputConns = AppState.connections.filter(c =>
3473                   c.toElement === elem.id && c.toPort === 'in-0'
3474               );
3475
3476               // Каждое соединение к выходу — это отдельный выход
3477               inputConns.forEach((conn, index) => {
3478                   const fromElem = AppState.elements[conn.fromElement];
3479                   if (!fromElem) return;
3480
3481                   const outputType = conn.signalType;
3482                   const outputInfo = {
3483                       id: `${elem.id}_conn_${index}`,
3484                       elementId: elem.id,
3485                       sourceElementId: conn.fromElement,
3486                       sourcePort: conn.fromPort,
3487                       portIndex: 0,
3488                       portId: 'in-0',
```

```
                    type: outputType,
                    label: elem.props?.label || 'Выход',
                    elementType: 'output',
                    elementName: elem.props?.label || 'Выход',
                    name: elem.props?.label || 'Выход'
                };

                if (outputType === SIGNAL_TYPE.LOGIC) {
                    AppState.outputs.logical.push(outputInfo);
                } else if (outputType === SIGNAL_TYPE.NUMBER) {
                    AppState.outputs.numeric.push(outputInfo);
                }

                // Подсветим входной порт
                this.highlightOutputPort(elem.id, 0, outputType);
            });
        });

        this.updateOutputCounter();
    },

    /**
     * Очистка всех выделений выходов
     */
    clearAllOutputHighlights() {
        document.querySelectorAll('.port.output-active').forEach(port => {
            port.classList.remove('output-active');
        });

        document.querySelectorAll('.element.has-output').forEach(elem => {
            elem.classList.remove('has-output');
        });

        document.querySelectorAll('.element.output-ambiguous').forEach(el =>
    el.classList.remove('output-ambiguous'));
        document.querySelectorAll('.element.output-missing').forEach(el =>
    el.classList.remove('output-missing'));
    },

    /**
     * Выделение выходного порта
     */
    highlightOutputPort(elemId, portIndex, portType) {
        const elem = document.getElementById(elemId);
        if (!elem) return;

        const port = elem.querySelector(`.port.output[data-port="out-${portIndex}"]`);
        if (port) {
            port.classList.add('output-active');
        }

        // Добавляем класс элементу (даёт общий визуал)
        elem.classList.add('has-output');
    },

    /**
     * Обновление счётчика выходов в меню
     */
    updateOutputCounter() {
        const counter = document.getElementById('output-counter');
        if (counter) {
            const total = AppState.outputs.logical.length +
    AppState.outputs.numeric.length;
            counter.textContent = total;
            counter.style.display = total > 0 ? 'inline-block' : 'none';
```

```
        }
    },

    /**
     * Получить все выходы для сохранения в проект
     */
    getOutputsForSave() {
        // Сохраняем информацию о frame/inner для рамок
        return {
            logical: AppState.outputs.logical.map(o => ({
                id: o.id,
                elementId: o.elementId,
                frameId: o.frameId || null,
                innerElementId: o.innerElementId || null,
                portIndex: o.portIndex ?? o.innerPortIndex ?? null,
                portLabel: o.label
            })),
            numeric: AppState.outputs.numeric.map(o => ({
                id: o.id,
                elementId: o.elementId,
                frameId: o.frameId || null,
                innerElementId: o.innerElementId || null,
                portIndex: o.portIndex ?? o.innerPortIndex ?? null,
                portLabel: o.label
            }))
        };
    },

    /**
     * Подсветить конкретный выход (при наведении в списке)
     */
    highlightOutput(elementId, highlight = true) {
        const elem = document.getElementById(elementId);
        if (elem) {
            if (highlight) {
                elem.classList.add('output-highlighted');
            } else {
                elem.classList.remove('output-highlighted');
            }
        }
    },

    /**
     * Перейти к элементу выхода на схеме (elementId — фокусируемый элемент; для рамок это id рамки)
     */
    navigateToOutput(elementId) {
        const elemData = AppState.elements[elementId];
        if (!elemData) return;

        // Центрируем viewport на элементе
        const container = document.getElementById('workspace-container');
        const rect = container.getBoundingClientRect();

        const centerX = elemData.x + elemData.width / 2;
        const centerY = elemData.y + elemData.height / 2;

        AppState.viewport.panX = rect.width / 2 - centerX * AppState.viewport.zoom;
        AppState.viewport.panY = rect.height / 2 - centerY * AppState.viewport.zoom;

        Viewport.updateTransform();

        // Выделяем элемент
        Elements.selectElement(elementId);

```

```
        // Временная подсветка
        this.highlightOutput(elementId, true);
        setTimeout(() => this.highlightOutput(elementId, false), 2000);
    }
};

project.js:
/**
 * Модуль управления проектом (сохранение, загрузка)
 */

const Project = {
    /**
     * Инициализация
     */
    init() {
        document.getElementById('btn-new').addEventListener('click', () =>
this.newProject());
        document.getElementById('btn-save').addEventListener('click', () =>
this.saveProject());
        document.getElementById('btn-load').addEventListener('click', () => {
            document.getElementById('file-input').click();
        });
        document.getElementById('btn-project-settings').addEventListener('click', ()
=> {
            Modal.showProjectPropertiesModal();
        });

        document.getElementById('file-input').addEventListener('change', (e) => {
            const file = e.target.files[0];
            if (file) {
                const reader = new FileReader();
                reader.onload = (ev) => this.loadProject(ev.target.result);
                reader.readAsText(file);
            }
            e.target.value = '';
        });
    },

    /**
     * Новый проект
     */
    newProject() {
        if (Object.keys(AppState.elements).length > 0) {
            if (!confirm('Создать новый проект? Несохранённые изменения будут
потеряны.')) {
                return;
            }
        }

        document.getElementById('workspace').innerHTML = '';
        document.getElementById('connections-svg').innerHTML = '';

        resetState();
        Viewport.updateTransform();
    },

    /**
     * Сохранение проекта
     */
    saveProject() {
        // Проверяем, заполнены ли свойства проекта
        if (!AppState.project.code) {
            Modal.showProjectPropertiesModal();
            alert('Пожалуйста, укажите код проекта перед сохранением.');
```

```
            return;
        }

        updateFrameChildren();

        const project = {
            version: '1.0',
            project: AppState.project,
            elements: AppState.elements,
            connections: AppState.connections,
            counter: AppState.elementCounter,
            viewport: {
                zoom: AppState.viewport.zoom,
                panX: AppState.viewport.panX,
                panY: AppState.viewport.panY
            }
        };

        const jsonStr = JSON.stringify(project, null, 2);
        const blob = new Blob([jsonStr], { type: 'application/json' });
        const url = URL.createObjectURL(blob);

        const filename = `${AppState.project.code || 'scheme'}_${AppState.project.type}.json`;

        const a = document.createElement('a');
        a.href = url;
        a.download = filename;
        a.click();

        URL.revokeObjectURL(url);
    },

    /**
     * Загрузка проекта
     */
    loadProject(jsonStr) {
        try {
            const data = JSON.parse(jsonStr);

            // Очищаем
            document.getElementById('workspace').innerHTML = '';
            document.getElementById('connections-svg').innerHTML = '';
            resetState();

            // Загружаем свойства проекта
            if (data.project) {
                AppState.project = { ...AppState.project, ...data.project };
            }

            // Загружаем состояние
            AppState.elementCounter = data.counter || 0;

            // Загружаем viewport
            if (data.viewport) {
                AppState.viewport.zoom = data.viewport.zoom || 1;
                AppState.viewport.panX = data.viewport.panX || 0;
                AppState.viewport.panY = data.viewport.panY || 0;
            }

            // Сначала загружаем рамки
            Object.values(data.elements || {})
                .filter(e => e.type === 'output-frame')
                .forEach(elemData => {
                    Elements.addElement(
```

```
                        elemData.type,
                        elemData.x,
                        elemData.y,
                        elemData.props,
                        elemData.id,
                        elemData.width,
                        elemData.height
                    );
                });

            // Затем остальные элементы
            Object.values(data.elements || {})
                .filter(e => e.type !== 'output-frame')
                .forEach(elemData => {
                    Elements.addElement(
                        elemData.type,
                        elemData.x,
                        elemData.y,
                        elemData.props,
                        elemData.id,
                        elemData.width,
                        elemData.height
                    );
                });

            AppState.connections = data.connections || [];

            Viewport.updateTransform();
            Connections.drawConnections();

        } catch (e) {
            alert('Ошибка загрузки: ' + e.message);
            console.error(e);
        }
    }
};

state.js:

/**
 * Глобальное состояние приложения
 */

const AppState = {
    // Элементы схемы
    elements: {},
    connections: [],
    elementCounter: 0,

    // Выделение
    selectedElement: null,

    // Перетаскивание
    draggingElement: null,
    dragOffset: { x: 0, y: 0 },
    isDraggingFromPalette: false,
    dragPreview: null,
    dragType: null,

    // Соединения
    connectingFrom: null,
    connectingFromType: null,
    tempLine: null,

    // Resize
```

```
    resizing: null,

    // Viewport (масштабирование и перемещение)
    viewport: {
        zoom: 1,
        panX: 0,
        panY: 0,
        isPanning: false,
        lastMouseX: 0,
        lastMouseY: 0
    },

    // Свойства проекта
    project: {
        code: '',
        type: PROJECT_TYPE.PARAMETER,
        // Для параметра
        dimension: '',
        // Для правила
        possibleCause: '',
        guidelines: ''
    },

    // Выходные сигналы (автоматически определяются)
    outputs: {
        logical: [],    // Логические выходы [{elementId, portIndex, portLabel, ...}]
        numeric: []     // Числовые выходы (формулы)
    }
};

/**
 * Сброс состояния
 */
function resetState() {
    AppState.elements = {};
    AppState.connections = [];
    AppState.elementCounter = 0;
    AppState.selectedElement = null;
    AppState.draggingElement = null;
    AppState.connectingFrom = null;
    AppState.tempLine = null;
    AppState.resizing = null;

    AppState.viewport = {
        zoom: 1,
        panX: 0,
        panY: 0,
        isPanning: false,
        lastMouseX: 0,
        lastMouseY: 0
    };

    AppState.project = {
        code: '',
        type: PROJECT_TYPE.PARAMETER,
        dimension: '',
        possibleCause: '',
        guidelines: ''
    };

    AppState.outputs = {
        logical: [],
        numeric: []
    };
}
```

```
3870
3871  utils.js:
3872  /**
3873   * Вспомогательные функции
3874   */
3875
3876  /**
3877   * Генерация уникального ID
3878   */
3879  function generateId() {
3880      AppState.elementCounter++;
3881      return `elem_${AppState.elementCounter}`;
3882  }
3883
3884  function getInputPortType(elementId, portIdentifier) {
3885      const element = AppState.elements[elementId];
3886      if (!element) return SIGNAL_TYPE.ANY;
3887
3888      const config = ELEMENT_TYPES[element.type];
3889      if (!config) return SIGNAL_TYPE.ANY;
3890
3891      let portIndex = portIdentifier;
3892
3893      // Обработка технического порта условия
3894      if (typeof portIdentifier === 'string') {
3895          if (portIdentifier === 'cond-0' && config.hasConditionPort) {
3896              return config.conditionPortType || SIGNAL_TYPE.LOGIC;
3897          }
3898
3899          if (portIdentifier.startsWith('in-')) {
3900              portIndex = parseInt(portIdentifier.split('-')[1], 10);
3901          }
3902      }
3903
3904      if (Number.isNaN(portIndex) || portIndex === null || portIndex === undefined) {
3905          portIndex = 0;
3906      }
3907
3908      // Динамические входы для AND/OR берут тип из конфига
3909      if ((element.type === 'and' || element.type === 'or')) {
3910          return SIGNAL_TYPE.LOGIC;  // Логические элементы всегда ожидают LOGIC на
      входе
3911      }
3912
3913      if (element.type === 'formula') {
3914          return SIGNAL_TYPE.ANY;
3915      }
3916
3917      const types = config.inputTypes || [];
3918      if (types.length === 0) return SIGNAL_TYPE.ANY;
3919
3920      if (portIndex < types.length) {
3921          return types[portIndex] || SIGNAL_TYPE.ANY;
3922      }
3923
3924      return types[types.length - 1] || SIGNAL_TYPE.ANY;
3925  }
3926
3927  function getOutputPortType(elementId, portIdentifier) {
3928      const element = AppState.elements[elementId];
3929      if (!element) return SIGNAL_TYPE.ANY;
3930
3931      const config = ELEMENT_TYPES[element.type];
3932      if (!config) return SIGNAL_TYPE.ANY;
3933
```

```
    let portIndex = portIdentifier;

    if (typeof portIdentifier === 'string') {
        if (portIdentifier.startsWith('out-')) {
            portIndex = parseInt(portIdentifier.split('-')[1], 10);
        }
    }

    if (Number.isNaN(portIndex) || portIndex === null || portIndex === undefined) {
        portIndex = 0;
    }

    const types = config.outputTypes || [];
    if (types.length === 0) return SIGNAL_TYPE.ANY;

    if (portIndex < types.length) {
        return types[portIndex] || SIGNAL_TYPE.ANY;
    }

    return types[types.length - 1] || SIGNAL_TYPE.ANY;
}
/**
 * Проверка совместимости типов сигналов
 *
 * Новая логика:
 * - ANY совместим со всем
 * - TRUE совместим с LOGIC, TRUE, ANY
 * - FALSE совместим с LOGIC, FALSE, ANY
 * - LOGIC совместим с LOGIC, TRUE, FALSE, ANY
 * - NUMERIC совместим с NUMERIC, ANY
 */
function areTypesCompatible(outputType, inputType) {
    // Если один из типов ANY - совместимы
    if (outputType === SIGNAL_TYPE.ANY || inputType === SIGNAL_TYPE.ANY) {
        return true;
    }

    // Если типы одинаковые - совместимы
    if (outputType === inputType) {
        return true;
    }

    // TRUE/FALSE совместимы с LOGIC
    if ((outputType === SIGNAL_TYPE.TRUE || outputType === SIGNAL_TYPE.FALSE) &&
        inputType === SIGNAL_TYPE.LOGIC) {
        return true;
    }

    // LOGIC совместим с TRUE/FALSE (в случае если ожидается конкретный тип)
    if (outputType === SIGNAL_TYPE.LOGIC &&
        (inputType === SIGNAL_TYPE.TRUE || inputType === SIGNAL_TYPE.FALSE)) {
        return true;
    }

    return false;
}

/**
 * Проверка, находится ли элемент внутри рамки
 */
function isInsideFrame(elemId, frameId) {
    const elem = AppState.elements[elemId];
    const frame = AppState.elements[frameId];

    if (!elem || !frame || frame.type !== 'output-frame') return false;
```

```
3999
4000      const elemCenterX = elem.x + elem.width / 2;
4001      const elemCenterY = elem.y + elem.height / 2;
4002
4003      return elemCenterX > frame.x &&
4004             elemCenterX < frame.x + frame.width &&
4005             elemCenterY > frame.y &&
4006             elemCenterY < frame.y + frame.height;
4007  }
4008
4009  /**
4010   * Обновить принадлежность элементов к рамкам
4011   */
4012  function updateFrameChildren() {
4013      // Сначала очистим children у рамок и parentFrame у всех элементов
4014      Object.values(AppState.elements).forEach(elem => {
4015          if (elem.type === 'output-frame') {
4016              elem.children = [];
4017          } else {
4018              // удаляем parentFrame по умолчанию (пересчитаем ниже)
4019              if (elem.parentFrame) delete elem.parentFrame;
4020          }
4021      });
4022
4023      // Назначаем принадлежность: для каждого элемента ищем рамку, в которую он
      попадает
4024      Object.values(AppState.elements).forEach(elem => {
4025          if (!elem || elem.type === 'output-frame') return;
4026
4027          Object.values(AppState.elements).forEach(frame => {
4028              if (!frame || frame.type !== 'output-frame') return;
4029
4030              if (isInsideFrame(elem.id, frame.id)) {
4031                  // добавляем в массив детей рамки
4032                  frame.children.push(elem.id);
4033                  // отмечаем у элемента родительскую рамку
4034                  if (AppState.elements[elem.id]) {
4035                      AppState.elements[elem.id].parentFrame = frame.id;
4036                  }
4037              }
4038          });
4039      });
4040  }
4041
4042  /**
4043   * Преобразование координат экрана в координаты холста
4044   */
4045  function screenToCanvas(screenX, screenY) {
4046      const container = document.getElementById('workspace-container');
4047      const rect = container.getBoundingClientRect();
4048
4049      const x = (screenX - rect.left - AppState.viewport.panX) / AppState.viewport.zoom;
4050      const y = (screenY - rect.top - AppState.viewport.panY) / AppState.viewport.zoom;
4051
4052      return { x, y };
4053  }
4054
4055  /**
4056   * Преобразование координат холста в координаты экрана
4057   */
4058  function canvasToScreen(canvasX, canvasY) {
4059      const container = document.getElementById('workspace-container');
4060      const rect = container.getBoundingClientRect();
4061
4062      const x = canvasX * AppState.viewport.zoom + AppState.viewport.panX + rect.left;
```

```
4063      const y = canvasY * AppState.viewport.zoom + AppState.viewport.panY + rect.top;
4064
4065      return { x, y };
4066  }
4067
4068  /**
4069   * Проверка, является ли порт выходным (не подключен к другим элементам)
4070   */
4071  function isOutputPort(elemId, portIndex) {
4072      const portKey = `out-${portIndex}`;
4073
4074      // Проверяем, есть ли соединения от этого порта
4075      const hasConnection = AppState.connections.some(conn =>
4076          conn.fromElement === elemId && conn.fromPort === portKey
4077      );
4078
4079      return !hasConnection;
4080  }
4081
4082  /**
4083   * Получить информацию о выходном порте
4084   */
4085  function getOutputPortInfo(elemId, portIndex) {
4086      const elem = AppState.elements[elemId];
4087      if (!elem) return null;
4088
4089      const config = ELEMENT_TYPES[elem.type];
4090      if (!config) return null;
4091
4092      return {
4093          elementId: elemId,
4094          elementType: elem.type,
4095          elementName: config.name,
4096          portIndex: portIndex,
4097          portLabel: config.outputLabels?.[portIndex] || `out${portIndex}`,
4098          portType: config.outputTypes?.[portIndex] || SIGNAL_TYPE.ANY,
4099          // Дополнительная информация для идентификации
4100          displayName: `${config.name} → ${config.outputLabels?.[portIndex] || `out$
      {portIndex}`}`
4101      };
4102  }
4103
4104  viewport.js:
4105  /**
4106   * Модуль управления viewport (масштабирование и перемещение)
4107   */
4108
4109  const Viewport = {
4110      /**
4111       * Инициализация viewport
4112       */
4113      init() {
4114          this.setupZoomControls();
4115          this.setupPanning();
4116          this.setupMouseWheel();
4117          this.setupMinimap();
4118          this.setupCursorPosition();
4119          this.updateTransform();
4120      },
4121
4122      /**
4123       * Настройка кнопок масштабирования
4124       */
4125      setupZoomControls() {
4126          document.getElementById('btn-zoom-in').addEventListener('click', () => {
```

```
            this.setZoom(AppState.viewport.zoom + VIEWPORT_CONFIG.zoomStep);
        });

        document.getElementById('btn-zoom-out').addEventListener('click', () => {
            this.setZoom(AppState.viewport.zoom - VIEWPORT_CONFIG.zoomStep);
        });

        document.getElementById('btn-zoom-reset').addEventListener('click', () => {
            this.setZoom(1);
            this.setPan(0, 0);
        });

        document.getElementById('btn-zoom-fit').addEventListener('click', () => {
            this.fitToContent();
        });
    },

    /**
     * Настройка перемещения (pan)
     */
    setupPanning() {
        const container = document.getElementById('workspace-container');

        container.addEventListener('mousedown', (e) => {
            // Средняя кнопка мыши или пробел + левая кнопка
            if (e.button === 1 || (e.button === 0 && e.target === container)) {
                e.preventDefault();
                AppState.viewport.isPanning = true;
                AppState.viewport.lastMouseX = e.clientX;
                AppState.viewport.lastMouseY = e.clientY;
                container.style.cursor = 'grabbing';
            }
        });

        document.addEventListener('mousemove', (e) => {
            if (AppState.viewport.isPanning) {
                const dx = e.clientX - AppState.viewport.lastMouseX;
                const dy = e.clientY - AppState.viewport.lastMouseY;

                this.setPan(
                    AppState.viewport.panX + dx,
                    AppState.viewport.panY + dy
                );

                AppState.viewport.lastMouseX = e.clientX;
                AppState.viewport.lastMouseY = e.clientY;
            }
        });

        document.addEventListener('mouseup', (e) => {
            if (AppState.viewport.isPanning) {
                AppState.viewport.isPanning = false;
                document.getElementById('workspace-container').style.cursor = '';
            }
        });

        // Клавиша пробел для режима перемещения
        document.addEventListener('keydown', (e) => {
            if (e.code === 'Space' && !e.repeat) {
                document.getElementById('workspace-container').style.cursor = 'grab';
            }
        });

        document.addEventListener('keyup', (e) => {
            if (e.code === 'Space') {
```

```
4192                 document.getElementById('workspace-container').style.cursor = '';
4193             }
4194         });
4195     },
4196
4197     /**
4198      * Настройка масштабирования колесом мыши
4199      */
4200     setupMouseWheel() {
4201         const container = document.getElementById('workspace-container');
4202
4203         container.addEventListener('wheel', (e) => {
4204             e.preventDefault();
4205
4206             const rect = container.getBoundingClientRect();
4207             const mouseX = e.clientX - rect.left;
4208             const mouseY = e.clientY - rect.top;
4209
4210             // Позиция мыши на холсте до масштабирования
4211             const canvasPosBeforeX = (mouseX - AppState.viewport.panX) /
     AppState.viewport.zoom;
4212             const canvasPosBeforeY = (mouseY - AppState.viewport.panY) /
     AppState.viewport.zoom;
4213
4214             // Новый масштаб
4215             const delta = e.deltaY > 0 ? -VIEWPORT_CONFIG.zoomStep :
     VIEWPORT_CONFIG.zoomStep;
4216             const newZoom = Math.max(
4217                 VIEWPORT_CONFIG.minZoom,
4218                 Math.min(VIEWPORT_CONFIG.maxZoom, AppState.viewport.zoom + delta)
4219             );
4220
4221             // Корректируем pan, чтобы точка под курсором осталась на месте
4222             const newPanX = mouseX - canvasPosBeforeX * newZoom;
4223             const newPanY = mouseY - canvasPosBeforeY * newZoom;
4224
4225             AppState.viewport.zoom = newZoom;
4226             AppState.viewport.panX = newPanX;
4227             AppState.viewport.panY = newPanY;
4228
4229             this.updateTransform();
4230         }, { passive: false });
4231     },
4232
4233     /**
4234      * Установить масштаб
4235      */
4236     setZoom(zoom) {
4237         const container = document.getElementById('workspace-container');
4238         const rect = container.getBoundingClientRect();
4239
4240         // Центр экрана
4241         const centerX = rect.width / 2;
4242         const centerY = rect.height / 2;
4243
4244         // Позиция центра на холсте
4245         const canvasCenterX = (centerX - AppState.viewport.panX) /
     AppState.viewport.zoom;
4246         const canvasCenterY = (centerY - AppState.viewport.panY) /
     AppState.viewport.zoom;
4247
4248         // Новый масштаб
4249         const newZoom = Math.max(
4250             VIEWPORT_CONFIG.minZoom,
4251             Math.min(VIEWPORT_CONFIG.maxZoom, zoom)
```

```
        );

        // Корректируем pan
        AppState.viewport.panX = centerX - canvasCenterX * newZoom;
        AppState.viewport.panY = centerY - canvasCenterY * newZoom;
        AppState.viewport.zoom = newZoom;

        this.updateTransform();
    },

    /**
     * Установить смещение
     */
    setPan(x, y) {
        AppState.viewport.panX = x;
        AppState.viewport.panY = y;
        this.updateTransform();
    },

    /**
     * Вписать содержимое в экран
     */
    fitToContent() {
        const elements = Object.values(AppState.elements);
        if (elements.length === 0) {
            this.setZoom(1);
            this.setPan(0, 0);
            return;
        }

        // Находим границы содержимого
        let minX = Infinity, minY = Infinity;
        let maxX = -Infinity, maxY = -Infinity;

        elements.forEach(elem => {
            minX = Math.min(minX, elem.x);
            minY = Math.min(minY, elem.y);
            maxX = Math.max(maxX, elem.x + elem.width);
            maxY = Math.max(maxY, elem.y + elem.height);
        });

        const contentWidth = maxX - minX;
        const contentHeight = maxY - minY;

        const container = document.getElementById('workspace-container');
        const rect = container.getBoundingClientRect();

        const padding = 50;
        const availableWidth = rect.width - padding * 2;
        const availableHeight = rect.height - padding * 2;

        const zoomX = availableWidth / contentWidth;
        const zoomY = availableHeight / contentHeight;
        const newZoom = Math.min(zoomX, zoomY, 1);

        AppState.viewport.zoom = Math.max(VIEWPORT_CONFIG.minZoom, newZoom);
        AppState.viewport.panX = padding - minX * AppState.viewport.zoom +
    (availableWidth - contentWidth * AppState.viewport.zoom) / 2;
        AppState.viewport.panY = padding - minY * AppState.viewport.zoom +
    (availableHeight - contentHeight * AppState.viewport.zoom) / 2;

        this.updateTransform();
    },

    /**
```

```
4315        * Обновить трансформацию
4316        */
4317       updateTransform() {
4318           const workspace = document.getElementById('workspace');
4319           const svg = document.getElementById('connections-svg');
4320
4321           const transform = `translate(${AppState.viewport.panX}px, $
      {AppState.viewport.panY}px) scale(${AppState.viewport.zoom})`;
4322
4323           workspace.style.transform = transform;
4324           svg.style.transform = transform;
4325
4326           // Обновляем отображение масштаба
4327           document.getElementById('zoom-level').textContent = `$
      {Math.round(AppState.viewport.zoom * 100)}%`;
4328
4329           // Обновляем мини-карту
4330           this.updateMinimap();
4331       },
4332
4333       /**
4334        * Настройка мини-карты
4335        */
4336       setupMinimap() {
4337           const minimap = document.getElementById('minimap');
4338           const canvas = document.getElementById('minimap-canvas');
4339
4340           canvas.width = MINIMAP_CONFIG.width;
4341           canvas.height = MINIMAP_CONFIG.height;
4342
4343           // Клик по мини-карте для перемещения
4344           minimap.addEventListener('click', (e) => {
4345               const rect = minimap.getBoundingClientRect();
4346               const x = e.clientX - rect.left;
4347               const y = e.clientY - rect.top;
4348
4349               this.navigateToMinimapPosition(x, y);
4350           });
4351       },
4352
4353       /**
4354        * Обновить мини-карту
4355        */
4356       updateMinimap() {
4357           const canvas = document.getElementById('minimap-canvas');
4358           const ctx = canvas.getContext('2d');
4359           const viewportEl = document.getElementById('minimap-viewport');
4360
4361           // Очищаем
4362           ctx.fillStyle = '#0a0a1a';
4363           ctx.fillRect(0, 0, canvas.width, canvas.height);
4364
4365           // Масштаб мини-карты
4366           const scale = Math.min(
4367               canvas.width / VIEWPORT_CONFIG.canvasWidth,
4368               canvas.height / VIEWPORT_CONFIG.canvasHeight
4369           );
4370
4371           // Рисуем элементы
4372           Object.values(AppState.elements).forEach(elem => {
4373               const x = elem.x * scale;
4374               const y = elem.y * scale;
4375               const w = Math.max(elem.width * scale, 2);
4376               const h = Math.max(elem.height * scale, 2);
4377
```

```
            ctx.fillStyle = ELEMENT_TYPES[elem.type]?.color || '#4a90d9';
            ctx.fillRect(x, y, w, h);
        });

        // Рисуем viewport
        const container = document.getElementById('workspace-container');
        const rect = container.getBoundingClientRect();

        const vpX = (-AppState.viewport.panX / AppState.viewport.zoom) * scale;
        const vpY = (-AppState.viewport.panY / AppState.viewport.zoom) * scale;
        const vpW = (rect.width / AppState.viewport.zoom) * scale;
        const vpH = (rect.height / AppState.viewport.zoom) * scale;

        viewportEl.style.left = `${vpX}px`;
        viewportEl.style.top = `${vpY}px`;
        viewportEl.style.width = `${vpW}px`;
        viewportEl.style.height = `${vpH}px`;
    },

    /**
     * Перейти к позиции на мини-карте
     */
    navigateToMinimapPosition(minimapX, minimapY) {
        const canvas = document.getElementById('minimap-canvas');
        const container = document.getElementById('workspace-container');
        const rect = container.getBoundingClientRect();

        const scale = Math.min(
            canvas.width / VIEWPORT_CONFIG.canvasWidth,
            canvas.height / VIEWPORT_CONFIG.canvasHeight
        );

        const canvasX = minimapX / scale;
        const canvasY = minimapY / scale;

        // Центрируем viewport на этой точке
        AppState.viewport.panX = rect.width / 2 - canvasX * AppState.viewport.zoom;
        AppState.viewport.panY = rect.height / 2 - canvasY * AppState.viewport.zoom;

        this.updateTransform();
    },

    /**
     * Отслеживание позиции курсора
     */
    setupCursorPosition() {
        const container = document.getElementById('workspace-container');

        container.addEventListener('mousemove', (e) => {
            const pos = screenToCanvas(e.clientX, e.clientY);
            document.getElementById('cursor-pos').textContent =
                `X: ${Math.round(pos.x)}, Y: ${Math.round(pos.y)}`;
        });
    }
};

styles.css:
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
```

```
    background: #1a1a2e;
    color: #eee;
    overflow: hidden;
}

#app {
    display: flex;
    flex-direction: column;
    height: 100vh;
}

/* ====== МЕНЮ ====== */
#menu {
    background: #16213e;
    padding: 10px 20px;
    display: flex;
    gap: 10px;
    align-items: center;
    border-bottom: 2px solid #0f3460;
    z-index: 100;
    flex-wrap: wrap;
}

.menu-btn {
    background: #0f3460;
    color: #eee;
    border: none;
    padding: 8px 16px;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
    font-size: 13px;
}

.menu-btn:hover {
    background: #e94560;
}

.menu-separator {
    width: 1px;
    height: 30px;
    background: #0f3460;
    margin: 0 10px;
}

.zoom-controls {
    display: flex;
    align-items: center;
    gap: 8px;
    background: #0a0a1a;
    padding: 5px 10px;
    border-radius: 5px;
}

.zoom-btn {
    width: 30px;
    height: 30px;
    padding: 0;
    font-size: 18px;
    font-weight: bold;
}

#zoom-level {
    min-width: 50px;
    text-align: center;
```

```
    font-size: 12px;
    color: #aaa;
}

/* ====== ОСНОВНАЯ ОБЛАСТЬ ====== */
#main {
    display: flex;
    flex: 1;
    overflow: hidden;
}

/* ====== ПАЛИТРА ====== */
#palette {
    width: 200px;
    background: #16213e;
    padding: 15px;
    border-right: 2px solid #0f3460;
    overflow-y: auto;
    z-index: 10;
    flex-shrink: 0;
}

#palette h3 {
    margin-bottom: 15px;
    color: #e94560;
    text-align: center;
    font-size: 14px;
}

.palette-section {
    margin-bottom: 15px;
}

.palette-section-title {
    font-size: 11px;
    color: #888;
    margin-bottom: 8px;
    padding-bottom: 3px;
    border-bottom: 1px solid #333;
}

.palette-item {
    background: #0f3460;
    padding: 8px;
    margin-bottom: 6px;
    border-radius: 8px;
    cursor: grab;
    text-align: center;
    transition: all 0.3s;
    border: 2px solid transparent;
    user-select: none;
}

.palette-item:hover {
    border-color: #e94560;
    transform: scale(1.02);
}

.palette-item:active {
    cursor: grabbing;
}

.palette-item svg {
    width: 50px;
    height: 32px;
```

```
    margin-bottom: 2px;
    pointer-events: none;
}

.palette-item-name {
    font-size: 10px;
    color: #aaa;
    pointer-events: none;
}

.type-legend {
    margin-top: 15px;
    padding-top: 10px;
    border-top: 1px solid #333;
    font-size: 10px;
}

.type-legend-item {
    display: flex;
    align-items: center;
    gap: 8px;
    margin-bottom: 5px;
}

.type-legend-dot {
    width: 12px;
    height: 12px;
    border-radius: 50%;
    border: 2px solid #fff;
}
.type-legend-dot.logic { background: #a855f7; }
.type-legend-dot.number { background: #3b82f6; }

/* ====== РАБОЧАЯ ОБЛАСТЬ ====== */
#workspace-container {
    flex: 1;
    position: relative;
    overflow: hidden;
    background-color: #0a0a1a;
    background-image:
        linear-gradient(rgba(255,255,255,0.04) 1px, transparent 1px),
        linear-gradient(90deg, rgba(255,255,255,0.04) 1px, transparent 1px);
    background-size: 25px 25px;
}

#workspace {
    position: absolute;
    transform-origin: 0 0;
    width: 5000px;
    height: 5000px;
}

#connections-svg {
    position: absolute;
    transform-origin: 0 0;
    pointer-events: none;
    z-index: 5;
    width: 5000px;
    height: 5000px;
}

#connections-svg path {
    pointer-events: stroke;
}

```

```
/* ====== ЭЛЕМЕНТЫ ====== */
.element {
    position: absolute;
    background: #0f3460;
    border: 2px solid #4a90d9;
    border-radius: 8px;
    cursor: move;
    user-select: none;
    z-index: 10;
    display: flex;
    flex-direction: column;
}

.element.selected {
    border-color: #e94560;
    box-shadow: 0 0 15px rgba(233, 69, 96, 0.5);
}

.element-header {
    background: #4a90d9;
    padding: 5px 10px;
    border-radius: 5px 5px 0 0;
    font-size: 11px;
    font-weight: bold;
    text-align: center;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}

.element-body {
    padding: 10px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    flex: 1;
    gap: 8px;
}

.element-symbol {
    font-size: 16px;
    font-weight: bold;
    flex: 1;
    text-align: center;
    padding: 0 5px;
    word-break: break-all;
    color: #eee;
}

/* ====== ПОРТЫ ====== */
.ports-left, .ports-right {
    display: flex;
    flex-direction: column;
    justify-content: space-around;
    gap: 10px;
    height: 100%;
}

.port {
    width: 14px;
    height: 14px;
    border-radius: 50%;
    border: 2px solid #fff;
    cursor: crosshair;
    transition: all 0.2s;
```

```
4703      position: relative;
4704      flex-shrink: 0;
4705  }
4706
4707  .port:hover { transform: scale(1.3); }
4708  .port.input { margin-left: -8px; }
4709  .port.output { margin-right: -8px; }
4710  .port.connected { background: #4ade80; }
4711
4712  /* Типы портов */
4713  .port.logic-port { background: #a855f7; border-color: #e9d5ff; }
4714  .port.logic-port:hover { background: #c084fc; }
4715  .port.logic-port.connected { background: #7c3aed; }
4716
4717  .port.number-port { background: #3b82f6; border-color: #bfdbfe; }
4718  .port.number-port:hover { background: #60a5fa; }
4719  .port.number-port.connected { background: #2563eb; }
4720
4721  .port.any-port { background: #6b7280; border-color: #d1d5db; }
4722  .port.any-port:hover { background: #9ca3af; }
4723  .port.any-port.connected { background: #4b5563; }
4724
4725  .port.output.yes-port { background: #4ade80 !important; border-color: #bbf7d0 !
      important; }
4726  .port.output.no-port { background: #f87171 !important; border-color: #fecaca !
      important; }
4727
4728  .port.incompatible { opacity: 0.3; cursor: not-allowed; }
4729  .port.compatible-highlight { box-shadow: 0 0 10px 3px #4ade80; }
4730
4731  /* ====== RESIZE HANDLES ====== */
4732  .resize-handle {
4733      position: absolute;
4734      width: 12px;
4735      height: 12px;
4736      background: #e94560;
4737      border: 1px solid #fff;
4738      border-radius: 3px;
4739      z-index: 20;
4740      opacity: 0;
4741      transition: opacity 0.2s;
4742  }
4743  .element.selected .resize-handle { opacity: 0.8; }
4744  .resize-handle:hover { opacity: 1; }
4745  .resize-handle.handle-se { bottom: -6px; right: -6px; cursor: se-resize; }
4746  .resize-handle.handle-e { top: 50%; right: -6px; transform: translateY(-50%); cursor:
      ew-resize; }
4747  .resize-handle.handle-s { bottom: -6px; left: 50%; transform: translateX(-50%);
      cursor: ns-resize; }
4748
4749
4750  /* ====== ВХОДНОЙ СИГНАЛ (ТРАПЕЦИЯ) ====== */
4751  .element.input-signal {
4752      background: transparent;
4753      border: none;
4754  }
4755
4756  .element.input-signal .element-header {
4757      display: none; /* У трапеции нет заголовка */
4758  }
4759
4760  .element.input-signal .element-body {
4761      padding: 0;
4762      background: #0f3460;
4763      border: 2px solid #4a90d9;
```

```
    clip-path: polygon(0 0, 80% 0, 100% 50%, 80% 100%, 0 100%);
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding-left: 15px;
    padding-right: 25px;
}

.element.input-signal .element-symbol {
    text-align: left;
    color: #eee;
}

.element.input-signal.selected .element-body {
    border-color: #e94560;
}

/* ====== ЭЛЕМЕНТ ВЫХОДА (ПУНКТИР) ====== */
.element.output {
    background: rgba(16, 185, 129, 0.1);
    border: 2px dashed #10b981;
}

.element.output .element-header {
    display: none; /* У выхода нет заголовка */
}

.element.output .element-body {
    padding-left: 20px;
}

.element.output .element-symbol {
    color: #10b981;
    font-size: 14px;
}

.element.output.selected {
    border-color: #e94560;
    border-style: dashed;
}


/* Formula condition port */
/* Универсальный стиль для технического порта (сверху) */
.element.has-condition-port {
    margin-top: 30px; /* Даем место порту над элементом */
}

.condition-port-wrapper {
    position: absolute;
    top: -28px;
    left: 50%;
    transform: translateX(-50%);
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 4px;
    pointer-events: none;
    z-index: 21;
}

.condition-port-label {
    font-size: 10px;
    color: #f59e0b;
    font-weight: 600;
```

```
    white-space: nowrap;
}

.port.condition-port {
    pointer-events: auto;
    width: 16px;
    height: 16px;
    border-radius: 50%;
    border: 2px solid #f59e0b;
    background: #fff7ed;
    margin: 0; /* Сбрасываем лишние отступы */
}
.element.formula .condition-port:hover { background: #fde68a; }


/* ====== СОЕДИНЕНИЯ ====== */
.connection {
    fill: none !important;  /* ← добавляем !important */
    stroke: #4a90d9;
    stroke-width: 2.5;
}
.connection:hover {
    stroke: #e94560;
    stroke-width: 4;
}

.connection.logic-conn { stroke: #a855f7; }
.connection.numeric-conn { stroke: #3b82f6; }
.connection.any-conn { stroke: #6b7280; }
.connection.true-conn { stroke: #4ade80; }
.connection.false-conn { stroke: #f87171; }

.connection.yes-conn { stroke: #4ade80; }
.connection.no-conn { stroke: #f87171; }

.temp-connection {
    fill: none !important;  /* ← добавляем !important */
    stroke: #e94560;
    stroke-width: 2;
    stroke-dasharray: 5, 5;
}
.temp-connection.invalid { stroke: #ef4444; }

/* ====== ПРОЧЕЕ ====== */
.drag-preview {
    position: fixed;
    pointer-events: none;
    opacity: 0.8;
    z-index: 1000;
    background: #0f3460;
    border: 2px solid #e94560;
    border-radius: 8px;
    padding: 10px 15px;
    color: #fff;
    font-size: 12px;
}

#minimap {
    position: absolute;
    bottom: 20px;
    right: 20px;
    width: 200px;
    height: 150px;
    background: #16213e;
    border: 2px solid #0f3460;
```

```
4894      border-radius: 8px;
4895      overflow: hidden;
4896      z-index: 50;
4897  }
4898
4899  #minimap-canvas { width: 100%; height: 100%; }
4900  #minimap-viewport {
4901      position: absolute;
4902      border: 2px solid #e94560;
4903      background: rgba(233, 69, 96, 0.2);
4904      pointer-events: none;
4905  }
4906
4907  #viewport-info {
4908      position: absolute;
4909      bottom: 20px;
4910      left: 20px;
4911      background: rgba(22, 33, 62, 0.9);
4912      padding: 8px 12px;
4913      border-radius: 5px;
4914      font-size: 11px;
4915      color: #888;
4916      z-index: 50;
4917      display: flex;
4918      gap: 15px;
4919  }
4920  #selection-info { color: #e94560; }
4921
4922  #modal-overlay, .modal-overlay-class {
4923      display: none;
4924      position: fixed;
4925      top: 0; left: 0;
4926      width: 100%; height: 100%;
4927      background: rgba(0, 0, 0, 0.7);
4928      z-index: 1000;
4929      justify-content: center;
4930      align-items: center;
4931  }
4932
4933  #modal, .modal-class {
4934      background: #16213e;
4935      border-radius: 10px;
4936      padding: 20px;
4937      min-width: 400px;
4938      max-width: 600px;
4939      max-height: 80vh;
4940      overflow-y: auto;
4941      border: 2px solid #0f3460;
4942  }
4943
4944  #modal h3, .modal-class h3 { margin-bottom: 15px; color: #e94560; }
4945  .modal-row { margin-bottom: 15px; }
4946  .modal-row label { display: block; margin-bottom: 5px; color: #aaa; font-size: 13px; }
4947  .modal-row input, .modal-row select, .modal-row textarea {
4948      width: 100%;
4949      padding: 10px;
4950      background: #0f3460;
4951      border: 1px solid #4a90d9;
4952      border-radius: 5px;
4953      color: #eee;
4954      font-size: 14px;
4955  }
4956  .modal-row input:focus, .modal-row select:focus, .modal-row textarea:focus { outline:
      none; border-color: #e94560; }
4957  .modal-row textarea { min-height: 80px; font-family: inherit; resize: vertical; }
```

```
4958  .signal-list { max-height: 100px; overflow-y: auto; background: #0f3460; border-
      radius: 5px; padding: 5px; margin-top: 5px; }
4959  .signal-item { padding: 5px 10px; cursor: pointer; border-radius: 3px; font-size:
      12px; }
4960  .signal-item:hover { background: #4a90d9; }
4961  .modal-buttons { display: flex; gap: 10px; justify-content: flex-end; margin-top:
      20px; }
4962  .modal-btn { padding: 10px 25px; border: none; border-radius: 5px; cursor: pointer;
      font-size: 14px; transition: background 0.3s; }
4963  .modal-btn.save { background: #4ade80; color: #000; }
4964  .modal-btn.save:hover { background: #22c55e; }
4965  .modal-btn.cancel { background: #6b7280; color: #fff; }
4966  .modal-btn.cancel:hover { background: #4b5563; }
4967
4968  #context-menu {
4969      display: none;
4970      position: fixed;
4971      background: #16213e;
4972      border: 1px solid #0f3460;
4973      border-radius: 5px;
4974      padding: 5px 0;
4975      z-index: 1001;
4976      min-width: 150px;
4977      box-shadow: 0 5px 20px rgba(0,0,0,0.3);
4978  }
4979  .context-item { padding: 10px 15px; cursor: pointer; font-size: 13px; transition:
      background 0.2s; }
4980  .context-item:hover { background: #0f3460; }
4981
4982  #file-input { display: none; }
4983
4984  .project-type-selector { display: flex; gap: 10px; margin-bottom: 15px; }
4985  .project-type-btn { flex: 1; padding: 15px; background: #0f3460; border: 2px solid
      #4a90d9; border-radius: 8px; color: #eee; cursor: pointer; text-align: center;
      transition: all 0.3s; }
4986  .project-type-btn:hover { border-color: #e94560; }
4987  .project-type-btn.active { background: #4a90d9; border-color: #4a90d9; }
4988  .project-type-btn .type-icon { font-size: 24px; margin-bottom: 5px; }
4989  .project-type-btn .type-name { font-weight: bold; }
4990  .project-type-btn .type-desc { font-size: 11px; color: #aaa; margin-top: 3px; }
4991
4992  .conditional-fields { display: none; padding: 15px; background: #0a0a1a; border-
      radius: 8px; margin-top: 10px; }
4993  .conditional-fields.visible { display: block; }
4994
4995  ::-webkit-scrollbar { width: 8px; height: 8px; }
4996  ::-webkit-scrollbar-track { background: #0a0a1a; }
4997  ::-webkit-scrollbar-thumb { background: #4a90d9; border-radius: 4px; }
4998  ::-webkit-scrollbar-thumb:hover { background: #e94560; }
4999
5000  /* Стили для выходов */
5001  .output-btn { position: relative; }
5002  .output-counter { display: inline-block; background: #e94560; color: white; font-size:
      11px; font-weight: bold; padding: 2px 6px; border-radius: 10px; margin-left: 5px; min-
      width: 18px; text-align: center; }
5003  .output-counter:empty, .output-counter[style*="display: none"] { display: none; }
5004  .element.has-output { box-shadow: 0 0 10px rgba(16, 185, 129, 0.3); }
5005  .element.output-highlighted { box-shadow: 0 0 20px rgba(251, 191, 36, 0.6) !important;
      border-color: #fbbf24 !important; }
5006  .port.output-active { box-shadow: 0 0 8px 2px rgba(16, 185, 129, 0.8); animation:
      pulse-output 1.5s infinite; }
5007  @keyframes pulse-output {
5008      0%, 100% { box-shadow: 0 0 8px 2px rgba(16, 185, 129, 0.8); }
5009      50% { box-shadow: 0 0 12px 4px rgba(16, 185, 129, 1); }
5010  }
```

```
5011
5012  .outputs-container { background: #0a0a1a; border-radius: 8px; padding: 15px; max-
      height: 250px; overflow-y: auto; }
5013  .outputs-section { margin-bottom: 15px; }
5014  .outputs-section:last-child { margin-bottom: 0; }
5015  .outputs-section-title { color: #10b981; font-weight: bold; font-size: 13px; margin-
      bottom: 10px; padding-bottom: 5px; border-bottom: 1px solid #333; display: flex;
      align-items: center; gap: 8px; }
5016  .outputs-section-title .section-icon { font-size: 16px; }
5017  .outputs-list { display: flex; flex-direction: column; gap: 5px; }
5018  .output-item { display: flex; align-items: center; gap: 10px; padding: 8px 12px;
      background: rgba(16, 185, 129, 0.1); border: 1px solid rgba(16, 185, 129, 0.3);
      border-radius: 5px; cursor: pointer; transition: all 0.2s; }
5019  .output-item:hover { background: rgba(16, 185, 129, 0.2); border-color: #10b981;
      transform: translateX(5px); }
5020  .output-item.numeric { background: rgba(59, 130, 246, 0.1); border-color: rgba(59,
      130, 246, 0.3); }
5021  .output-item.numeric:hover { background: rgba(59, 130, 246, 0.2); border-color:
      #3b82f6; }
5022  .output-icon { font-size: 14px; }
5023  .output-name { font-weight: bold; color: #eee; }
5024  .output-port { color: #888; font-size: 12px; margin-left: auto; }
5025  .no-outputs { color: #666; font-style: italic; padding: 10px; text-align: center; }
5026  .outputs-hint { margin-top: 10px; padding: 10px; background: rgba(59, 130, 246, 0.1);
      border-radius: 5px; font-size: 12px; color: #888; line-height: 1.4; }
5027  .element.output-ambiguous { box-shadow: 0 0 18px 4px rgba(240, 80, 80, 0.55); border-
      color: rgba(240,80,80,0.8) !important; }
5028  .element.output-missing { box-shadow: 0 0 14px 3px rgba(250, 200, 30, 0.5); border-
      color: rgba(250,200,30,0.8) !important; }
5029  /* TRUE/FALSE порты (для сепаратора) */
5030  .port.true-port {
5031      background: #4ade80 !important;
5032      border-color: #bbf7d0 !important;
5033  }
5034  .port.true-port:hover {
5035      background: #22c55e !important;
5036  }
5037  .port.true-port.connected {
5038      background: #16a34a !important;
5039  }
5040
5041  .port.false-port {
5042      background: #f87171 !important;
5043      border-color: #fecaca !important;
5044  }
5045  .port.false-port:hover {
5046      background: #ef4444 !important;
5047  }
5048  .port.false-port.connected {
5049      background: #dc2626 !important;
5050  }
5051
5052  /* Сепаратор стиль */
5053  .element.separator {
5054      background: #0f3460;
5055      border: 2px solid #f59e0b;
5056  }
5057
5058  .element.separator.selected {
5059      border-color: #e94560;
5060      box-shadow: 0 0 15px rgba(233, 69, 96, 0.5);
5061  }
```