

```
1  index.html
2
3  <!DOCTYPE html>
4  <html lang="ru">
5  <head>
6      <meta charset="UTF-8">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Редактор логических схем</title>
9      <link rel="stylesheet" href="css/styles.css">
10 </head>
11 <body>
12     <div id="app">
13         <div id="menu">
14             <button class="menu-btn" id="btn-new"> Новый</button>
15             <button class="menu-btn" id="btn-save"> Сохранить</button>
16             <button class="menu-btn" id="btn-load"> Загрузить</button>
17             <button class="menu-btn" id="btn-generate-code"> Код</button>
18             <button class="menu-btn" id="btn-project-settings"> Свойства проекта</button>
19         <div class="menu-separator"></div>
20         <div class="zoom-controls">
21             <button class="menu-btn zoom-btn" id="btn-zoom-out">-</button>
22             <span id="zoom-level">100%</span>
23             <button class="menu-btn zoom-btn" id="btn-zoom-in">+</button>
24             <button class="menu-btn" id="btn-zoom-fit">□ Вписать</button>
25             <button class="menu-btn" id="btn-zoom-reset">1:1</button>
26         </div>
27         <input type="file" id="file-input" accept=".json">
28     </div>
29
30     <div id="main">
31         <div id="palette">
32             <h3> Элементы</h3>
33
34             <div class="palette-section">
35                 <div class="palette-section-title">ВХОДЫ</div>
36
37                 <div class="palette-item" data-type="input-signal">
38                     <svg viewBox="0 0 60 40">
39                         <polygon points="0,5 40,5 55,20 40,35 0,35" fill="#0f3460" stroke="#4a90d9" stroke-width="2"/>
40                         <text x="12" y="24" fill="#eee" font-size="10">IN</text>
41                     </svg>
42                     <div class="palette-item-name">Входной сигнал</div>
43                 </div>
44             </div>
45             <div class="palette-section">
46                 <div class="palette-section-title">ВЫХОДЫ</div>
47
48                 <div class="palette-item" data-type="output">
49                     <svg viewBox="0 0 60 40">
50                         <rect x="5" y="5" width="50" height="30" rx="6" fill="none" stroke="#10b981" stroke-width="2" stroke-dasharray="4,2"/>
51                         <text x="12" y="24" fill="#10b981" font-size="9">Выход</text>
52                     </svg>
53                     <div class="palette-item-name">Выход</div>
54                 </div>
55             </div>
56
57             <div class="palette-section">
58                 <div class="palette-section-title">ЛОГИЧЕСКИЕ</div>
59
60                 <div class="palette-item" data-type="and">
61                     <svg viewBox="0 0 60 40">
```

```
62          <rect x="5" y="5" width="50" height="30" rx="5"
63      fill="#0f3460" stroke="#a855f7" stroke-width="2"/>
64      <text x="22" y="25" fill="#eee" font-size="12" font-
65      weight="bold">И</text>
66          </svg>
67          <div class="palette-item-name">И (AND)</div>
68      </div>
69
70          <div class="palette-item" data-type="or">
71              <svg viewBox="0 0 60 40">
72                  <rect x="5" y="5" width="50" height="30" rx="5"
73          fill="#0f3460" stroke="#a855f7" stroke-width="2"/>
74          <text x="12" y="25" fill="#eee" font-size="11" font-
75          weight="bold">ИЛИ</text>
76              </svg>
77              <div class="palette-item-name">ИЛИ (OR)</div>
78          </div>
79
80          <div class="palette-item" data-type="not">
81              <svg viewBox="0 0 60 40">
82                  <rect x="5" y="5" width="50" height="30" rx="5"
83          fill="#0f3460" stroke="#a855f7" stroke-width="2"/>
84          <text x="12" y="25" fill="#eee" font-size="11" font-
85          weight="bold">НЕТ</text>
86              </svg>
87              <div class="palette-item-name">НЕТ (NOT)</div>
88          </div>
89      </div>
90
91          <div class="palette-section">
92              <div class="palette-section-title">СРАВНЕНИЕ</div>
93
94              <div class="palette-item" data-type="if">
95                  <svg viewBox="0 0 60 40">
96                      <polygon points="30,3 57,20 30,37 3,20" fill="#0f3460"
97          stroke="#e94560" stroke-width="2"/>
98                  <text x="14" y="24" fill="#eee" font-size="9" font-
99          weight="bold">ЕСЛИ</text>
100                 </svg>
101                 <div class="palette-item-name">ЕСЛИ (IF)</div>
102             </div>
103         </div>
104
105         <div class="palette-section">
106             <div class="palette-section-title">РАЗВЕТВЛЕНИЕ</div>
107
108             <div class="palette-item" data-type="separator">
109                 <svg viewBox="0 0 60 40">
110                     <rect x="5" y="8" width="50" height="24" rx="3"
111             fill="#0f3460" stroke="#f59e0b" stroke-width="2"/>
112             <text x="8" y="25" fill="#f59e0b" font-size="10" font-
113             weight="bold">/>/x</text>
114                 </svg>
115                 <div class="palette-item-name">Сепаратор</div>
116             </div>
117         </div>
118
119         <div class="palette-section">
120             <div class="palette-section-title">ЗНАЧЕНИЯ</div>
121
122             <div class="palette-item" data-type="const">
123                 <svg viewBox="0 0 60 40">
124                     <rect x="10" y="8" width="40" height="24" rx="3"
125             fill="#0f3460" stroke="#3b82f6" stroke-width="2"/>
126             <text x="24" y="25" fill="#3b82f6" font-size="14" font-
```

```
weight="bold">C</text>
116                     </svg>
117                     <div class="palette-item-name">Константа</div>
118                 </div>
119
120             <div class="palette-item" data-type="formula">
121                 <svg viewBox="0 0 60 40">
122                     <rect x="5" y="5" width="50" height="30" rx="5"
123                         fill="#0f3460" stroke="#f59e0b" stroke-width="2"/>
124                     <text x="12" y="25" fill="#f59e0b" font-size="11" font-
125                         weight="bold">f(x)</text>
126                     </svg>
127                     <div class="palette-item-name">Формула</div>
128                 </div>
129             </div>
130
131         <div class="type-legend">
132             <div class="type-legend-item">
133                 <div class="type-legend-dot logic"></div>
134                 <span>Логический</span>
135             </div>
136             <div class="type-legend-item">
137                 <div class="type-legend-dot number"></div>
138                 <span>Числовой</span>
139             </div>
140         </div>
141
142     <div id="workspace-container">
143         <svg id="connections-svg"></svg>
144         <div id="workspace"></div>
145
146         <!-- Мини-карта -->
147         <div id="minimap">
148             <div id="minimap-viewport"></div>
149             <canvas id="minimap-canvas"></canvas>
150         </div>
151
152         <!-- Координаты и информация -->
153         <div id="viewport-info">
154             <span id="cursor-pos">X: 0, Y: 0</span>
155             <span id="selection-info"></span>
156         </div>
157     </div>
158 </div>
159
160 <!-- Модальные окна -->
161 <div id="modal-overlay">
162     <div id="modal">
163         <h3 id="modal-title">Свойства элемента</h3>
164         <div id="modal-content"></div>
165         <div class="modal-buttons">
166             <button class="modal-btn cancel" id="modal-cancel">Отмена</button>
167             <button class="modal-btn save" id="modal-save">Сохранить</button>
168         </div>
169     </div>
170 </div>
171
172 <!-- Модальное окно свойств проекта -->
173 <div id="project-modal-overlay" class="modal-overlay-class">
174     <div id="project-modal" class="modal-class">
175         <h3>Свойства проекта</h3>
176         <div id="project-modal-content"></div>
177         <div class="modal-buttons">
```

```
178             <button class="modal-btn cancel" id="project-modal-cancel">Отмена</
179         button>
180             <button class="modal-btn save" id="project-modal-save">Сохранить</
181         button>
182     </div>
183
184     <div id="code-modal-overlay" class="modal-overlay-class">
185         <div id="code-modal" class="modal-class">
186             <h3>Сгенерированный код</h3>
187             <textarea id="code-output" style="width:100%; height:300px;"></textarea>
188             <div class="modal-buttons">
189                 <button class="modal-btn cancel" id="code-modal-close">Закрыть</
190             button>
191                 </div>
192             </div>
193
194     <div id="context-menu">
195         <div class="context-item" id="ctx-properties"> Свойства</div>
196         <div class="context-item" id="ctx-delete"> Удалить</div>
197     </div>
198
199     <!-- Модули JavaScript -->
200     <!-- Модули JavaScript -->
201     <script src="js/config.js"></script>
202     <script src="js/state.js"></script>
203     <script src="js/utils.js"></script>
204     <script src="js/viewport.js"></script>
205     <script src="js/elements.js"></script>
206     <script src="js/connections.js"></script>
207     <script src="js/outputs.js"></script> <!-- ← Этот файл опционален теперь -->
208     <script src="js/modal.js"></script>
209     <script src="js/project.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) {
```

```
240             Outputs.updateOutputStatus();
241         }
242
243         console.log('Logic Scheme Editor initialized');
244         document.getElementById('btn-generate-code').addEventListener('click', () => {
245             const code = CodeGen.generate();
246             document.getElementById('code-output').value = code;
247             document.getElementById('code-modal-overlay').style.display = 'flex';
248         });
249
250         document.getElementById('code-modal-close').addEventListener('click', () => {
251             document.getElementById('code-modal-overlay').style.display = 'none';
252         });
253     },
254
255     /**
256      * Отмена состояния drag из палитры (helper)
257      */
258     cancelPaletteDrag() {
259         if (AppState.dragPreview) {
260             try { AppState.dragPreview.remove(); } catch (e) { /* ignore */ }
261             AppState.dragPreview = null;
262         }
263         AppState.isDraggingFromPalette = false;
264         AppState.dragType = null;
265     },
266
267     /**
268      * Настройка счётчика выходов в меню
269      */
270     setupOutputCounter() {
271         // Не создавать повторно, если уже есть
272         if (document.getElementById('btn-outputs')) return;
273
274         const menu = document.getElementById('menu');
275
276         // Создаём кнопку с счётчиком выходов
277         const outputBtn = document.createElement('button');
278         outputBtn.className = 'menu-btn output-btn';
279         outputBtn.id = 'btn-outputs';
280         outputBtn.innerHTML =
281             `  Выходы
282             <span id="output-counter" class="output-counter">0</span>
283             `;
284
285         // Вставляем после кнопки свойств проекта
286         const projectBtn = document.getElementById('btn-project-settings');
287         if (projectBtn) {
288             projectBtn.after(outputBtn);
289         } else {
290             menu.appendChild(outputBtn);
291         }
292
293         outputBtn.addEventListener('click', () => {
294             Modal.showProjectPropertiesModal();
295         });
296     },
297
298     /**
299      * Настройка drag & drop из палитры
300      */
301     setupPaletteDragDrop() {
302         document.querySelectorAll('.palette-item').forEach(item => {
303             item.addEventListener('mousedown', (e) => {
304                 // Только левая кнопка мыши должна запускать drag из палитры
```

```
305             if (e.button !== 0) return;
306             e.preventDefault();
307
308             AppState.isDraggingFromPalette = true;
309             AppState.dragType = item.dataset.type;
310
311             AppState.dragPreview = document.createElement('div');
312             AppState.dragPreview.className = 'drag-preview';
313             AppState.dragPreview.textContent =
314               ELEMENT_TYPES[AppState.dragType]?.name || 'Элемент';
315             AppState.dragPreview.style.left = `${e.clientX - 40}px`;
316             AppState.dragPreview.style.top = `${e.clientY - 20}px`;
317             document.body.appendChild(AppState.dragPreview);
318         );
319     },
320
321     /**
322      * Глобальные обработчики мыши
323     */
324   /**
325      * Глобальные обработчики мыши
326   */
327   setupGlobalMouseHandlers() {
328     document.addEventListener('mousemove', (e) => {
329       if (AppState.isDraggingFromPalette && AppState.dragPreview) {
330         AppState.dragPreview.style.left = `${e.clientX - 40}px`;
331         AppState.dragPreview.style.top = `${e.clientY - 20}px`;
332       }
333       if (AppState.resizing) {
334         Elements.handleResize(e);
335         return;
336       }
337       if (AppState.draggingElement) {
338         Elements.handleDrag(e);
339       }
340       if (AppState.tempLine && AppState.connectingFrom) {
341         Connections.drawTempConnection(e);
342       }
343     });
344
345     document.addEventListener('mouseup', (e) => {
346       if (AppState.resizing) {
347         AppState.resizing = null;
348         if (typeof Outputs !== 'undefined') Outputs.updateOutputStatus();
349       }
350
351       if (AppState.isDraggingFromPalette) {
352         try {
353           if (AppState.dragPreview) {
354             AppState.dragPreview.remove();
355             AppState.dragPreview = null;
356           }
357
358           const container = document.getElementById('workspace-container');
359           const rect = container.getBoundingClientRect();
360
361           if (e.clientX >= rect.left && e.clientX <= rect.right &&
362               e.clientY >= rect.top && e.clientY <= rect.bottom) {
363
364             const canvasPos = screenToCanvas(e.clientX, e.clientY);
365             const config = ELEMENT_TYPES[AppState.dragType];
366             if (config) {
367               const defaultWidth = config.minLength || 120;
368               const defaultHeight = config.minLength || 60;
```

```
369
370         // ИСПРАВЛЕНО: addElement возвращает DOM-элемент, его надо
371     // обработать
372         const newElement = Elements.addElement(
373             AppState.dragType,
374             canvasPos.x - defaultWidth / 2,
375             canvasPos.y - defaultHeight / 2
376         );
377
378         if (newElement && typeof Outputs !== 'undefined') {
379             Outputs.updateOutputStatus();
380         }
381     } else {
382         console.error('Неизвестный тип элемента при drop:',
383         AppState.dragType);
384     }
385     }
386     App.cancelPaletteDrag();
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 * Настройка контекстного меню
410 */
411 setupContextMenu() {
412     document.addEventListener('click', (e) => {
413         const menu = document.getElementById('context-menu');
414         if (!menu.contains(e.target)) {
415             menu.style.display = 'none';
416         }
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';
```

```
432         Elements.deleteElement(elemId);
433         // Обновляем выходы только если модуль загружен
434         if (typeof Outputs !== 'undefined' && Outputs.updateOutputStatus) {
435             Outputs.updateOutputStatus();
436         }
437     });
438 },
439 /**
440 * Клик по рабочей области
441 */
442 setupWorkspaceClick() {
443     const workspace = document.getElementById('workspace');
444
445     workspace.addEventListener('click', (e) => {
446         if (e.target === workspace) {
447             Elements.deselectAll();
448         }
449     });
450 }
451 };
452 };
453
454 // Запуск приложения при загрузке страницы
455 document.addEventListener('DOMContentLoaded', () => {
456     App.init();
457 });
458
459 // js/codegen_optimizer.js
460
461 let _depth = 0;
462 const MAX_DEPTH = 200;
463
464 // === Конструкторы ===
465 function Eq0(v) { return { kind: 'cond', type: 'eq0', v }; }
466 function Ne0(v) { return { kind: 'cond', type: 'ne0', v }; }
467 function Cmp(l, op, r) { return { kind: 'cond', type: 'cmp', l, op, r }; }
468 function And(a, b) {
469     if (!a) return b;
470     if (!b) return a;
471     return { kind: 'cond', type: 'and', a, b };
472 }
473 function Or(a, b) {
474     if (!a) return b;
475     if (!b) return a;
476     return { kind: 'cond', type: 'or', a, b };
477 }
478 function Not(x) {
479     if (!x) return null;
480     return { kind: 'cond', type: 'not', x };
481 }
482 const TrueCond = { kind: 'cond', type: 'true' };
483 const FalseCond = { kind: 'cond', type: 'false' };
484
485 function Const(n) { return { kind: 'expr', type: 'const', n }; }
486 function Var(name) { return { kind: 'expr', type: 'var', name }; }
487 function Op(op, l, r) { return { kind: 'expr', type: 'op', op, l, r }; }
488 function When(c, t, e) { return { kind: 'expr', type: 'when', c, t, e }; }
489
490 // === Утилиты ===
491 function atomKey(c) {
492     if (!c) return null;
493     switch (c.type) {
494         case 'eq0': return `eq0:${c.v}`;
495         case 'ne0': return `ne0:${c.v}`;
496         case 'cmp': return `cmp:${c.l}: ${c.op}: ${c.r}`;
```

```
497         case 'true': return 'true';
498         case 'false': return 'false';
499     default: return null;
500 }
501 }
502
503 function negateOp(op) {
504     switch (op) {
505         case '=': return '!=';
506         case '!=': return '=';
507         case '>': return '<';
508         case '<': return '>';
509         case '>=': return '<';
510         case '<=': return '>';
511     default: return null;
512 }
513 }
514
515 function negateAtomKey(key) {
516     if (!key) return null;
517     if (key.startsWith('eq0:')) return 'ne0:' + key.slice(4);
518     if (key.startsWith('ne0:')) return 'eq0:' + key.slice(4);
519     if (key.startsWith('cmp:')) {
520         const parts = key.slice(4).split(':');
521         if (parts.length === 3) {
522             const negOp = negateOp(parts[1]);
523             if (negOp) return `cmp:${parts[0]}:${negOp}:${parts[2]}`;
524         }
525     }
526     return null;
527 }
528
529 function isNegation(a, b) {
530     if (!a || !b) return false;
531     if (a.type === 'eq0' && b.type === 'ne0' && a.v === b.v) return true;
532     if (a.type === 'ne0' && b.type === 'eq0' && a.v === b.v) return true;
533     if (a.type === 'cmp' && b.type === 'cmp' && a.l === b.l && a.r === b.r) {
534         return a.op === negateOp(b.op);
535     }
536     if (a.type === 'not' && condEq(a.x, b)) return true;
537     if (b.type === 'not' && condEq(b.x, a)) return true;
538     return false;
539 }
540
541 function condEq(a, b) {
542     if (a === b) return true;
543     if (!a || !b) return false;
544     if (a.type !== b.type) return false;
545
546     switch (a.type) {
547         case 'eq0':
548         case 'ne0':
549             return a.v === b.v;
550         case 'cmp':
551             return a.l === b.l && a.op === b.op && a.r === b.r;
552         case 'true':
553         case 'false':
554             return true;
555         case 'not':
556             return condEq(a.x, b.x);
557         case 'and':
558         case 'or':
559             return (condEq(a.a, b.a) && condEq(a.b, b.b)) ||
560                     (condEq(a.a, b.b) && condEq(a.b, b.a));
561     default:
```

```
562             return false;
563     }
564 }
565
566 function flattenAnd(c) {
567     if (!c) return [];
568     if (c.type === 'and') return [...flattenAnd(c.a), ...flattenAnd(c.b)];
569     return [c];
570 }
571
572 function flattenOr(c) {
573     if (!c) return [];
574     if (c.type === 'or') return [...flattenOr(c.a), ...flattenOr(c.b)];
575     return [c];
576 }
577
578 function buildAnd(terms) {
579     if (terms.length === 0) return TrueCond;
580     let result = terms[0];
581     for (let i = 1; i < terms.length; i++) {
582         result = And(result, terms[i]);
583     }
584     return result;
585 }
586
587 function buildOr(terms) {
588     if (terms.length === 0) return FalseCond;
589     let result = terms[0];
590     for (let i = 1; i < terms.length; i++) {
591         result = Or(result, terms[i]);
592     }
593     return result;
594 }
595
596 // === Упрощение условий ===
597 function simplifyCond(c) {
598     _depth++;
599     if (_depth > MAX_DEPTH) {
600         _depth--;
601         return c;
602     }
603
604     try {
605         return simplifyCondCore(c);
606     } finally {
607         _depth--;
608     }
609 }
610
611 function simplifyCondCore(c) {
612     if (!c || c.kind !== 'cond') return c;
613
614     switch (c.type) {
615         case 'true':
616         case 'false':
617         case 'eq0':
618         case 'ne0':
619         case 'cmp':
620             return c;
621
622         case 'not': {
623             const x = simplifyCondCore(c.x);
624             if (!x) return TrueCond;
625             if (x.type === 'true') return FalseCond;
626             if (x.type === 'false') return TrueCond;
627         }
628     }
629 }
```

```
627     if (x.type === 'not') return simplifyCondCore(x.x);
628     if (x.type === 'eq0') return Ne0(x.v);
629     if (x.type === 'ne0') return Eq0(x.v);
630     if (x.type === 'cmp') {
631         const neg0p = negate0p(x.op);
632         if (neg0p) return Cmp(x.l, neg0p, x.r);
633     }
634     if (x.type === 'and') return simplifyCondCore(Or(Not(x.a), Not(x.b)));
635     if (x.type === 'or') return simplifyCondCore(And(Not(x.a), Not(x.b)));
636     return Not(x);
637 }
638
639 case 'and': {
640     const a = simplifyCondCore(c.a);
641     const b = simplifyCondCore(c.b);
642
643     if (!a) return b;
644     if (!b) return a;
645     if (a.type === 'false' || b.type === 'false') return FalseCond;
646     if (a.type === 'true') return b;
647     if (b.type === 'true') return a;
648
649     const allTerms = [...flattenAnd(a), ...flattenAnd(b)];
650     const atomMap = new Map();
651     const otherTerms = [];
652
653     for (const t of allTerms) {
654         if (t.type === 'true') continue;
655         if (t.type === 'false') return FalseCond;
656
657         const key = atomKey(t);
658         if (key) {
659             const negKey = negateAtomKey(key);
660             if (negKey && atomMap.has(negKey)) {
661                 return FalseCond;
662             }
663             if (!atomMap.has(key)) {
664                 atomMap.set(key, t);
665             }
666         } else {
667             otherTerms.push(t);
668         }
669     }
670
671     const uniqueAtoms = Array.from(atomMap.values());
672     const result = [...uniqueAtoms, ...otherTerms];
673
674     if (result.length === 0) return TrueCond;
675     if (result.length === 1) return result[0];
676
677     return buildAnd(result);
678 }
679
680 case 'or': {
681     const a = simplifyCondCore(c.a);
682     const b = simplifyCondCore(c.b);
683
684     if (!a) return b;
685     if (!b) return a;
686     if (a.type === 'true' || b.type === 'true') return TrueCond;
687     if (a.type === 'false') return b;
688     if (b.type === 'false') return a;
689
690     const allTerms = [...flattenOr(a), ...flattenOr(b)];
691     const atomMap = new Map();
```

```
692     const otherTerms = [];
693
694     for (const t of allTerms) {
695         if (t.type === 'true') return TrueCond;
696         if (t.type === 'false') continue;
697
698         const key = atomKey(t);
699         if (key) {
700             const negKey = negateAtomKey(key);
701             if (negKey && atomMap.has(negKey)) {
702                 return TrueCond;
703             }
704             if (!atomMap.has(key)) {
705                 atomMap.set(key, t);
706             }
707         } else {
708             otherTerms.push(t);
709         }
710     }
711
712     const uniqueAtoms = Array.from(atomMap.values());
713     const result = [...uniqueAtoms, ...otherTerms];
714
715     if (result.length === 0) return FalseCond;
716     if (result.length === 1) return result[0];
717
718     return buildOr(result);
719 }
720
721     default:
722         return c;
723     }
724 }
725
726 // === Сравнение выражений ===
727 function exprEq(a, b) {
728     if (a === b) return true;
729     if (!a && !b) return true;
730     if (!a || !b) return false;
731     if (a.type !== b.type) return false;
732
733     switch (a.type) {
734         case 'const': return a.n === b.n;
735         case 'var': return a.name === b.name;
736         case 'op': return a.op === b.op && exprEq(a.l, b.l) && exprEq(a.r, b.r);
737         case 'when': return condEq(a.c, b.c) && exprEq(a.t, b.t) && exprEq(a.e, b.e);
738         default: return false;
739     }
740 }
741
742 // === Упрощение выражений ===
743 function simplifyExpr(expr) {
744     _depth++;
745     if (_depth > MAX_DEPTH) {
746         _depth--;
747         return expr;
748     }
749
750     try {
751         return simplifyExprCore(expr);
752     } finally {
753         _depth--;
754     }
755 }
756
```

```
757 function simplifyExprCore(expr) {
758     if (!expr || expr.kind !== 'expr') return expr;
759
760     switch (expr.type) {
761         case 'const':
762         case 'var':
763             return expr;
764
765         case 'op': {
766             const l = simplifyExprCore(expr.l);
767             const r = simplifyExprCore(expr.r);
768
769             if (expr.op === '+') {
770                 if (r?.type === 'const' && r.n === 0) return l;
771                 if (l?.type === 'const' && l.n === 0) return r;
772             }
773             if (expr.op === '*') {
774                 if (l?.type === 'const' && l.n === 0) return Const(0);
775                 if (r?.type === 'const' && r.n === 0) return Const(0);
776                 if (l?.type === 'const' && l.n === 1) return r;
777                 if (r?.type === 'const' && r.n === 1) return l;
778             }
779             return Op(expr.op, l, r);
780         }
781
782         case 'when': {
783             const c = simplifyCond(expr.c);
784             const t = simplifyExprCore(expr.t);
785             const e = simplifyExprCore(expr.e);
786
787             if (c?.type === 'true') return t;
788             if (c?.type === 'false') return e;
789             if (exprEq(t, e)) return t;
790
791             return When(c, t, e);
792         }
793
794         default:
795             return expr;
796     }
797 }
798
799 // === Печать ===
800 function printCond(c) {
801     if (!c) return 'TRUE';
802
803     switch (c.type) {
804         case 'eq0': return `${c.v} = 0`;
805         case 'ne0': return `${c.v} != 0`;
806         case 'cmp': return `${c.l} ${c.op} ${c.r}`;
807         case 'and': return `(${printCond(c.a)} AND ${printCond(c.b)})`;
808         case 'or': return `(${printCond(c.a)} OR ${printCond(c.b)})`;
809         case 'not': return `NOT(${printCond(c.x)})`;
810         case 'true': return 'TRUE';
811         case 'false': return 'FALSE';
812         default: return '?';
813     }
814 }
815
816 function printExpr(e) {
817     if (!e) return '0';
818
819     switch (e.type) {
820         case 'const': return String(e.n);
821         case 'var': return e.name;
```

```
822         case 'op': return `(${printExpr(e.l)}${e.op}${printExpr(e.r)})`;
823         case 'when': return `WHEN(${printCond(e.c)}, ${printExpr(e.t)}, ${
824             printExpr(e.e)})`;
825         default: return '?';
826     }
827 }
828 window.Optimizer = {
829     Eq0, Ne0, Cmp, And, Or, Not, TrueCond, FalseCond,
830     Const, Var, Op, When,
831     simplifyCond, simplifyExpr,
832     printCond, printExpr,
833     condEq, exprEq
834 };
835 // js/codegen.js
836
837 const CodeGen = {
838     _cache: {},
839     _branchCache: {},
840     _resolveCache: {},
841     _visiting: new Set(),
842
843     reset() {
844         this._cache = {};
845         this._branchCache = {};
846         this._resolveCache = {};
847         this._visiting = new Set();
848     },
849
850     toExpr(valueStr) {
851         const s = String(valueStr).trim();
852         if (s === '0') return Optimizer.Const(0);
853         const num = parseFloat(s);
854         if (!isNaN(num) && String(num) === s) return Optimizer.Const(num);
855         return Optimizer.Var(s);
856     },
857
858     exprToName(exprAst) {
859         if (!exprAst) return '0';
860         if (exprAst.type === 'var') return exprAst.name;
861         if (exprAst.type === 'const') return String(exprAst.n);
862         return Optimizer.printExpr(exprAst);
863     },
864
865     mergeCond(a, b) {
866         if (!a && !b) return null;
867         if (!a) return b;
868         if (!b) return a;
869         if (Optimizer.condEq && Optimizer.condEq(a, b)) return a;
870         return Optimizer.And(a, b);
871     },
872
873     getConn(toId, toPort) {
874         return AppState.connections.find(c => c.toElement === toId && c.toPort ===
875             toPort);
876     },
877
878     getConns(toId, prefix) {
879         return AppState.connections.filter(c => c.toElement === toId &&
c.toPort.startsWith(prefix));
880     },
881
882     buildFormulaExpr(elem) {
883         const expression = elem.props.expression || '0';
```

```
884     let result = expression;
885     const formulaRefs = result.match(/formula-\d+/g) || [];
886
887     for (const ref of formulaRefs) {
888         const refElem = AppState.elements[ref];
889         if (refElem && refElem.type === 'formula') {
890             const refExpr = this.buildFormulaExpr(refElem);
891             result = result.replace(new RegExp(ref, 'g'), `(${refExpr})`);
892         }
893     }
894
895     return result;
896 },
897
898 // === Получить ЧИСТУЮ логику элемента ===
899 getPureLogic(id) {
900     const cacheKey = `logic:${id}`;
901     if (cacheKey in this._cache) {
902         return this._cache[cacheKey];
903     }
904
905     const elem = AppState.elements[id];
906     if (!elem) return null;
907
908     let logic = null;
909
910     switch (elem.type) {
911         case 'if':
912             const leftConn = this.getConn(id, 'in-0');
913             const rightConn = this.getConn(id, 'in-1');
914
915             const leftVal = leftConn ? this.getValue(leftConn.fromElement) :
916 Optimizer.Const(0);
917             const rightVal = rightConn ? this.getValue(rightConn.fromElement) :
918 Optimizer.Const(0);
919
920             const op = (elem.props.operator || '=').trim();
921             const leftName = this.exprToName(leftVal);
922             const rightName = this.exprToName(rightVal);
923
924             const leftZero = leftVal.type === 'const' && leftVal.n === 0;
925             const rightZero = rightVal.type === 'const' && rightVal.n === 0;
926
927             // Поддержка всех операторов
928             switch (op) {
929                 case '=':
930                     if (rightZero) {
931                         logic = Optimizer.Eq0(leftName);
932                     } else if (leftZero) {
933                         logic = Optimizer.Eq0(rightName);
934                     } else {
935                         logic = Optimizer.Cmp(leftName, '=', rightName);
936                     }
937                     break;
938                 case '!=':
939                     if (rightZero) {
940                         logic = Optimizer.Ne0(leftName);
941                     } else if (leftZero) {
942                         logic = Optimizer.Ne0(rightName);
943                     } else {
944                         logic = Optimizer.Cmp(leftName, '!=', rightName);
945                     }
946                     break;
947                 case '>':
948                 case '<':
```

```
947         case '>=':
948         case '<=':
949             logic = Optimizer.Cmp(leftName, op, rightName);
950             break;
951         default:
952             logic = Optimizer.TrueCond;
953         }
954         break;
955     }
956
957     case 'and':
958     case 'or': {
959         const isAnd = elem.type === 'and';
960         const count = elem.props.inputCount || 2;
961         let result = null;
962
963         for (let i = 0; i < count; i++) {
964             const conn = this.getConn(id, `in-${i}`);
965             if (!conn) continue;
966
967             const val = this.getPureLogic(conn.fromElement);
968             if (!val) continue;
969
970             if (result === null) {
971                 result = val;
972             } else {
973                 result = isAnd ? Optimizer.And(result, val) :
974 Optimizer.Or(result, val);
975             }
976         }
977         logic = result || Optimizer.FalseCond;
978         break;
979     }
980
981     case 'not': {
982         const conn = this.getConn(id, 'in-0');
983         const inputLogic = conn ? this.getPureLogic(conn.fromElement) : null;
984         logic = Optimizer.Not(inputLogic || Optimizer.FalseCond);
985         break;
986     }
987
988     case 'separator': {
989         const conn = this.getConn(id, 'in-0');
990         logic = conn ? this.getPureLogic(conn.fromElement) :
991 Optimizer.FalseCond;
992         break;
993     }
994
995     default:
996         logic = null;
997     }
998
999     this._cache[cacheKey] = logic;
1000     return logic;
1001 },
1002 // === Получить значение ===
1003 getValue(id) {
1004     const elem = AppState.elements[id];
1005     if (!elem) return Optimizer.Const(0);
1006
1007     if (elem.type === 'input-signal') {
1008         return this.toExpr(elem.props.name || id);
1009     }
1010     if (elem.type === 'const') {
```

```
1010             return Optimizer.Const(Number(elem.props.value) || 0);
1011         }
1012     return Optimizer.Const(0);
1013 },
1014
1015 // === Получить ПОЛНОЕ условие для ветки сепаратора ===
1016 getBranchCondition(sepId, fromPort) {
1017     const cacheKey = `${sepId}:${fromPort}`;
1018     if (cacheKey in this._branchCache) {
1019         return this._branchCache[cacheKey];
1020     }
1021
1022     const sep = AppState.elements[sepId];
1023     if (!sep || sep.type !== 'separator') return null;
1024
1025     const inputLogic = this.getPureLogic(sepId);
1026     const sepContext = this.getConditionFromPort(sepId);
1027
1028     let branchLogic;
1029     if (fromPort === 'out-1') {
1030         branchLogic = inputLogic ? Optimizer.Not(inputLogic) : Optimizer.TrueCond;
1031     } else {
1032         branchLogic = inputLogic || Optimizer.TrueCond;
1033     }
1034
1035     let result;
1036     if (sepContext) {
1037         result = Optimizer.And(sepContext, branchLogic);
1038     } else {
1039         result = branchLogic;
1040     }
1041
1042     this._branchCache[cacheKey] = result;
1043     return result;
1044 },
1045
1046 // === Получить условие от cond-порта ===
1047 getConditionFromPort(id) {
1048     const conn = this.getConn(id, 'cond-0');
1049     if (!conn) return null;
1050
1051     const sourceElem = AppState.elements[conn.fromElement];
1052     if (!sourceElem) return null;
1053
1054     if (sourceElem.type === 'separator') {
1055         return this.getBranchCondition(conn.fromElement, conn.fromPort);
1056     }
1057
1058     return this.getPureLogic(conn.fromElement);
1059 },
1060
1061 // === Основная функция разрешения ===
1062 resolve(id) {
1063     if (id in this._resolveCache) {
1064         return this._resolveCache[id];
1065     }
1066
1067     if (this._visiting.has(id)) {
1068         return null;
1069     }
1070     this._visiting.add(id);
1071
1072     const elem = AppState.elements[id];
1073     if (!elem) {
1074         this._visiting.delete(id);
```

```
1075             return null;
1076         }
1077
1078         let result = null;
1079
1080         try {
1081             switch (elem.type) {
1082                 case 'input-signal':
1083                     result = {
1084                         isValue: true,
1085                         cond: null,
1086                         expr: this.toExpr(elem.props.name || id)
1087                     };
1088                     break;
1089
1090                 case 'const': {
1091                     const cond = this.getConditionFromPort(id);
1092                     result = {
1093                         isValue: true,
1094                         cond: cond,
1095                         expr: Optimizer.Const(Number(elem.props.value) || 0)
1096                     };
1097                     break;
1098                 }
1099
1100                 case 'formula': {
1101                     let cond = this.getConditionFromPort(id);
1102
1103                     const inConns = this.getConns(id, 'in-');
1104                     for (const conn of inConns) {
1105                         const inputNode = this.resolve(conn.fromElement);
1106                         if (inputNode && inputNode.cond) {
1107                             cond = this.mergeCond(cond, inputNode.cond);
1108                         }
1109                     }
1110
1111                     const fullExpr = this.buildFormulaExpr(elem);
1112                     result = {
1113                         isValue: true,
1114                         cond: cond,
1115                         expr: Optimizer.Var(fullExpr)
1116                     };
1117                     break;
1118                 }
1119
1120                 default:
1121                     result = null;
1122             }
1123         } finally {
1124             this._visiting.delete(id);
1125         }
1126
1127         this._resolveCache[id] = result;
1128         return result;
1129     },
1130
1131     // === Генерация ===
1132     generate() {
1133         console.log('== Генерация кода ==');
1134         this.reset();
1135
1136         try {
1137             const outputs = Object.values(AppState.elements).filter(e => e.type ===
1138 'output');
```

```
1139     if (outputs.length === 0) {
1140         return /* Нет выходов */;
1141     }
1142
1143     const allVariants = [];
1144
1145     for (const out of outputs) {
1146         const conns = this.getConns(out.id, 'in-');
1147
1148         for (const conn of conns) {
1149             const node = this.resolve(conn.fromElement);
1150             if (!node || !node.isValue || !node.expr) continue;
1151
1152             const cond = node.cond ? Optimizer.simplifyCond(node.cond) : null;
1153             const isZero = node.expr.type === 'const' && node.expr.n === 0;
1154
1155             if (isZero && !cond) continue;
1156
1157             allVariants.push({
1158                 cond: cond,
1159                 expr: node.expr,
1160                 isZero: isZero,
1161                 source: conn.fromElement
1162             });
1163         }
1164     }
1165
1166     console.log('Варианты:', allVariants.map(v => ({
1167         source: v.source,
1168         cond: Optimizer.printCond(v.cond),
1169         expr: Optimizer.printExpr(v.expr)
1170     })));
1171
1172     if (allVariants.length === 0) {
1173         return '0';
1174     }
1175
1176     const valueVariants = allVariants.filter(v => !v.isZero);
1177
1178     if (valueVariants.length === 0) {
1179         return '0';
1180     }
1181
1182     let result = Optimizer.Const(0);
1183
1184     for (let i = valueVariants.length - 1; i >= 0; i--) {
1185         const v = valueVariants[i];
1186         if (v.cond) {
1187             result = Optimizer.When(v.cond, v.expr, result);
1188         } else {
1189             result = v.expr;
1190         }
1191     }
1192
1193     console.log('До оптимизации:', Optimizer.printExpr(result));
1194
1195     const simplified = Optimizer.simplifyExpr(result);
1196
1197     console.log('После оптимизации:', Optimizer.printExpr(simplified));
1198
1199     return Optimizer.printExpr(simplified);
1200
1201 } catch (err) {
1202     console.error('Ошибка генерации:', err);
1203     return `/* Ошибка: ${err.message} */`;
```

```
1204         }
1205     }
1206 };
1207 windowCodeGen = CodeGen;
1208 config.js
1209
1210 /**
1211  * Конфигурация приложения
1212 */
1213
1214 // Типы сигналов
1215 const SIGNAL_TYPE = {
1216     NUMERIC: 'numeric',      // Числовой сигнал
1217     LOGIC: 'logic',          // Логический (может быть TRUE или FALSE)
1218     TRUE: 'true',            // Явно ИСТИНА
1219     FALSE: 'false',          // Явно ЛОЖЬ
1220     ANY: 'any'               // Любой тип
1221 };
1222
1223 // Типы проекта
1224 const PROJECT_TYPE = {
1225     PARAMETER: 'parameter',
1226     RULE: 'rule'
1227 };
1228
1229 // Конфигурация элементов
1230 const ELEMENT_TYPES = {
1231     'input-signal': {
1232         name: 'Вход',
1233         inputs: 0,
1234         outputs: 1,
1235         outputLabels: ['out'],
1236         outputTypes: [SIGNAL_TYPE.NUMERIC],
1237         color: '#4a90d9',
1238         hasProperties: true,
1239         defaultProps: { name: 'Сигнал', signalType: SIGNAL_TYPE.NUMERIC },
1240         resizable: true,
1241         minWidth: 150,
1242         minHeight: 50
1243     },
1244     'and': {
1245         name: 'И',
1246         inputs: 2, // По умолчанию 2, но может быть изменено
1247         outputs: 1,
1248         inputLabels: ['A', 'B'],
1249         inputTypes: [SIGNAL_TYPE.LOGIC, SIGNAL_TYPE.LOGIC],
1250         outputLabels: ['результат'],
1251         outputTypes: [SIGNAL_TYPE.LOGIC],
1252         color: '#a855f7',
1253         hasProperties: true, // ← Теперь есть свойства (для изменения количества
1254         // входов)
1255         resizable: true,
1256         minWidth: 120,
1257         minHeight: 80,
1258         hasConditionPort: true,
1259         conditionPortType: SIGNAL_TYPE.LOGIC,
1260         defaultProps: {
1261             inputCount: 2 // ← Новое свойство
1262         }
1263     },
1264     'or': {
1265         name: 'ИЛИ',
1266         inputs: 2, // По умолчанию 2
```

```
1268     outputs: 1,
1269     inputLabels: ['A', 'B'],
1270     inputTypes: [SIGNAL_TYPE.LOGIC, SIGNAL_TYPE.LOGIC],
1271     outputLabels: ['результат'],
1272     outputTypes: [SIGNAL_TYPE.LOGIC],
1273     color: '#a855f7',
1274     hasProperties: true, // ← Теперь есть свойства
1275     resizable: true,
1276     minWidth: 120,
1277     minHeight: 80,
1278     hasConditionPort: true,
1279     conditionPortType: SIGNAL_TYPE.LOGIC,
1280     defaultProps: {
1281       inputCount: 2 // ← Новое свойство
1282     }
1283   },
1284   'not': {
1285     name: 'НЕ',
1286     inputs: 1,
1287     outputs: 1,
1288     inputLabels: ['A'],
1289     inputTypes: [SIGNAL_TYPE.LOGIC],
1290     outputLabels: ['¬A'],
1291     outputTypes: [SIGNAL_TYPE.LOGIC],
1292     color: '#a855f7',
1293     hasProperties: false,
1294     resizable: true,
1295     minWidth: 100,
1296     minHeight: 60,
1297     hasConditionPort: true,
1298     conditionPortType: SIGNAL_TYPE.LOGIC
1299   },
1300   'if': {
1301     name: 'ЕСЛИ',
1302     inputs: 2,
1303     outputs: 1, // ← Только один выход!
1304     inputLabels: ['A', 'B'],
1305     inputTypes: [SIGNAL_TYPE.ANY, SIGNAL_TYPE.ANY],
1306     outputLabels: ['результат'], // ← Просто результат
1307     outputTypes: [SIGNAL_TYPE.LOGIC], // ← Выход типа LOGIC
1308     color: '#e94560',
1309     hasProperties: true,
1310     defaultProps: { operator: '=' },
1311     resizable: true,
1312     minWidth: 120,
1313     minHeight: 80,
1314     hasConditionPort: true,
1315     conditionPortType: SIGNAL_TYPE.LOGIC
1316   },
1317   'separator': { // ← НОВЫЙ ЭЛЕМЕНТ
1318     name: 'Сепаратор',
1319     inputs: 1,
1320     outputs: 2,
1321     inputLabels: ['сигнал'],
1322     inputTypes: [SIGNAL_TYPE.LOGIC],
1323     outputLabels: ['ИСТИНА', 'ЛОЖЬ'],
1324     outputTypes: [SIGNAL_TYPE.TRUE, SIGNAL_TYPE.FALSE], // ← TRUE и FALSE
1325     color: '#f59e0b',
1326     hasProperties: false,
1327     resizable: true,
1328     minWidth: 120,
1329     minHeight: 80,
1330     hasConditionPort: true,
1331     conditionPortType: SIGNAL_TYPE.LOGIC
1332   },
```

```
1333     'const': {
1334         name: 'Константа',
1335         inputs: 0,
1336         outputs: 1,
1337         outputLabels: ['out'],
1338         outputTypes: [SIGNAL_TYPE.NUMERIC],
1339         color: '#3b82f6',
1340         hasProperties: true,
1341         defaultProps: { value: 0 },
1342         resizable: true,
1343         minWidth: 120,
1344         minHeight: 60,
1345         hasConditionPort: true,
1346         conditionPortType: SIGNAL_TYPE.LOGIC
1347     },
1348     'formula': {
1349         name: 'Формула',
1350         inputs: 2,
1351         outputs: 1,
1352         inputLabels: ['in1', 'in2'],
1353         inputTypes: [SIGNAL_TYPE.ANY, SIGNAL_TYPE.ANY],
1354         outputLabels: ['результат'],
1355         outputTypes: [SIGNAL_TYPE.NUMERIC],
1356         color: '#f59e0b',
1357         hasProperties: true,
1358         resizable: true,
1359         minWidth: 140,
1360         minHeight: 80,
1361         defaultProps: {
1362             expression: '',
1363             inputCount: 2
1364         },
1365         hasConditionPort: true,
1366         conditionPortType: SIGNAL_TYPE.LOGIC
1367     },
1368     'output': {
1369         name: 'Выход',
1370         inputs: 1,
1371         outputs: 0,
1372         inputLabels: ['сигнал'],
1373         inputTypes: [SIGNAL_TYPE.ANY],
1374         color: '#10b981',
1375         hasProperties: true,
1376         defaultProps: { label: 'Выход', outputGroup: '' },
1377         resizable: true,
1378         minWidth: 150,
1379         minHeight: 60,
1380     },
1381 };
1382
1383 const VIEWPORT_CONFIG = {
1384     minZoom: 0.1,
1385     maxZoom: 3,
1386     zoomStep: 0.1,
1387     panSpeed: 1,
1388     canvasWidth: 5000,
1389     canvasHeight: 5000
1390 };
1391
1392 const MINIMAP_CONFIG = {
1393     width: 200,
1394     height: 150,
1395     padding: 10
1396 };
1397
```

```
1398 connections.js
1399 /**
1400  * Модуль работы с соединениями
1401 */
1402
1403 const Connections = {
1404     /**
1405      * Настройка обработчиков порта
1406      */
1407     setupPortHandlers(port) {
1408         port.addEventListener('mousedown', (e) => {
1409             e.stopPropagation();
1410
1411             if (port.classList.contains('output')) {
1412                 const elemId = port.dataset.element;
1413                 const portName = port.dataset.port;
1414                 const signalType = getOutputPortType(elemId, portName);
1415
1416                 AppState.connectingFrom = {
1417                     element: elemId,
1418                     port: portName
1419                 };
1420                 AppState.connectingFromType = signalType;
1421
1422                 this.highlightCompatiblePorts(signalType);
1423
1424                 const svg = document.getElementById('connections-svg');
1425                 const startPos = this._getPortCanvasCenter(port);
1426
1427                 AppState.tempLine = document.createElementNS('http://www.w3.org/2000/
1428                     svg', 'path');
1429                     AppState.tempLine.setAttribute('class', 'temp-connection');
1430                     AppState.tempLine.setAttribute('d', `M ${startPos.x} ${startPos.y} L ${
1431                     startPos.x} ${startPos.y}`);
1432                     svg.appendChild(AppState.tempLine);
1433                 }
1434             });
1435
1436             port.addEventListener('mouseup', (e) => {
1437                 e.stopPropagation();
1438                 e.preventDefault();
1439
1440                 if (AppState.connectingFrom && port.classList.contains('input')) {
1441                     const toElement = port.dataset.element;
1442                     const toPortName = port.dataset.port;
1443                     const inputType = getInputPortType(toElement, toPortName);
1444
1445                     if (!areTypesCompatible(AppState.connectingFromType, inputType)) {
1446                         this.clearConnectionState();
1447                         return;
1448                     }
1449
1450                     if (AppState.connectingFrom.element !== toElement) {
1451                         const targetElem = AppState.elements[toElement];
1452                         const allowMultipleInputs = targetElem?.type === 'output';
1453
1454                         const exists = AppState.connections.some(c =>
1455                             c.toElement === toElement && c.toPort === toPortName
1456                         );
1457
1458                         if (!exists || allowMultipleInputs) {
1459                             AppState.connections.push({
1460                                 fromElement: AppState.connectingFrom.element,
1461                                 fromPort: AppState.connectingFrom.port,
1462                                 toElement,
```

```
1461                     toPort: toPortName,
1462                     signalType: AppState.connectingFromType
1463                 );
1464
1465             port.classList.add('connected');
1466             this.drawConnections();
1467             this.clearConnectionState();
1468             return;
1469         }
1470     }
1471 }
1472
1473     this.clearConnectionState();
1474 });
1475
1476     port.addEventListener('mouseenter', () => {
1477         if (AppState.connectingFrom && port.classList.contains('input')) {
1478             const toPortName = port.dataset.port;
1479             const inputType = getInputPortType(port.dataset.element, toPortName);
1480
1481             if (!areTypesCompatible(AppState.connectingFromType, inputType)) {
1482                 if (AppState.tempLine) {
1483                     AppState.tempLine.classList.add('invalid');
1484                 }
1485             }
1486         }
1487     });
1488
1489     port.addEventListener('mouseleave', () => {
1490         if (AppState.tempLine) {
1491             AppState.tempLine.classList.remove('invalid');
1492         }
1493     });
1494 },
1495
1496 /**
1497 * Подсветка совместимых портов
1498 */
1499 highlightCompatiblePorts(signalType) {
1500     document.querySelectorAll('.port.input').forEach(port => {
1501         const inputType = getInputPortType(port.dataset.element,
1502         port.dataset.port);
1503
1504         if (areTypesCompatible(signalType, inputType)) {
1505             port.classList.add('compatible-highlight');
1506         } else {
1507             port.classList.add('incompatible');
1508         }
1509     });
1510
1511 /**
1512 * Очистка состояния соединения
1513 */
1514 clearConnectionState() {
1515     if (AppState.tempLine) {
1516         AppState.tempLine.remove();
1517         AppState.tempLine = null;
1518     }
1519     AppState.connectingFrom = null;
1520     AppState.connectingFromType = null;
1521
1522     document.querySelectorAll('.port').forEach(port => {
1523         port.classList.remove('compatible-highlight', 'incompatible');
1524     });
1525 }
```

```
1525 },
1526 /**
1527 * Отрисовка временной линии соединения
1528 */
1529 drawTempConnection(e) {
1530     if (!AppState.tempLine || !AppState.connectingFrom) return;
1531
1532     const fromElem = document.getElementById(AppState.connectingFrom.element);
1533     if (!fromElem) return;
1534
1535     const fromPort = fromElem.querySelector(`[data-port="${AppState.connectingFrom.port}"]`);
1536     if (!fromPort) return;
1537
1538     const startPos = this._getPortCanvasCenter(fromPort);
1539     const endPos = screenToCanvas(e.clientX, e.clientY);
1540
1541     const horizontalDist = Math.abs(endPos.x - startPos.x);
1542     const controlDist = Math.max(horizontalDist * 0.4, 50);
1543
1544     // Тянем всегда от выхода (вектор 1, 0)
1545     const cx1 = startPos.x + controlDist;
1546     const cyl = startPos.y;
1547
1548     // Вторая точка контроля для плавности за курсором
1549     const cx2 = endPos.x - controlDist;
1550     const cy2 = endPos.y;
1551
1552     AppState.tempLine.setAttribute('d', `M ${startPos.x} ${startPos.y} C ${cx1} ${cyl} ${cx2} ${cy2}, ${endPos.x} ${endPos.y}`);
1553     AppState.tempLine.setAttribute('fill', 'none');
1554 },
1555
1556 /**
1557 * Отрисовка всех соединений
1558 */
1559 drawConnections() {
1560     const svg = document.getElementById('connections-svg');
1561
1562     // 1. Очистка старых линий
1563     svg.querySelectorAll('path:not(.temp-connection)').forEach(p => p.remove());
1564
1565     // 2. Сброс визуального состояния портов
1566     document.querySelectorAll('.port.connected').forEach(port => {
1567         port.classList.remove('connected');
1568     });
1569
1570     // 3. Перебор всех соединений из AppState
1571     AppState.connections.forEach(conn => {
1572         const fromElem = document.getElementById(conn.fromElement);
1573         const toElem = document.getElementById(conn.toElement);
1574
1575         if (!fromElem || !toElem) return;
1576
1577         const fromPort = fromElem.querySelector(`[data-port="${conn.fromPort}"]`);
1578         const toPort = toElem.querySelector(`[data-port="${conn.toPort}"]`);
1579
1580         if (!fromPort || !toPort) return;
1581
1582         fromPort.classList.add('connected');
1583         toPort.classList.add('connected');
1584
1585         const startPos = this._getPortCanvasCenter(fromPort);
1586         const endPos = this._getPortCanvasCenter(toPort);
```

```
1588     if (!startPos || !endPos) return;
1589
1590     // Расстояние для изгиба кривой
1591     const horizontalDist = Math.abs(endPos.x - startPos.x);
1592     const verticalDist = Math.abs(endPos.y - startPos.y);
1593     const controlDist = Math.max(horizontalDist * 0.4, 50);
1594
1595     // --- ЛОГИКА ГЕОМЕТРИИ (Вектора касательных) ---
1596     let d;
1597     let cx1 = startPos.x;
1598     let cy1 = startPos.y;
1599     let cx2 = endPos.x;
1600     let cy2 = endPos.y;
1601
1602     // ВЫХОД (Source): Касательная (1, 0) -> Всегда вправо
1603     cx1 = startPos.x + controlDist;
1604     cy1 = startPos.y;
1605
1606     // ВХОД (Target):
1607     if (conn.toPort === 'cond-0') {
1608         // Технический порт: Касательная (0, 1) в декартовой (вверх)
1609         // В экранных координатах Y инвертирован, поэтому отнимаем от Y
1610         cx2 = endPos.x;
1611         cy2 = endPos.y - controlDist; // Линия заходит сверху вертикально
1612     } else {
1613         // Обычный вход: Касательная (-1, 0) -> Слева направо
1614         cx2 = endPos.x - controlDist;
1615         cy2 = endPos.y;
1616     }
1617
1618     d = `M ${startPos.x} ${startPos.y} C ${cx1} ${cy1}, ${cx2} ${cy2}, ${endPos.x}
1619 ${endPos.y}`;
1620
1621     const path = document.createElementNS('http://www.w3.org/2000/svg', 'path');
1622     path.setAttribute('d', d);
1623     path.setAttribute('fill', 'none'); // Чтобы не было черных полигонов
1624
1625     // --- ЛОГИКА ЦВЕТА (Классы) ---
1626     let cssClass = 'connection';
1627     const type = conn.signalType;
1628
1629     // Приоритет новым типам сигналов
1630     if (type === SIGNAL_TYPE.TRUE) cssClass += ' true-conn';
1631     else if (type === SIGNAL_TYPE.FALSE) cssClass += ' false-conn';
1632     else if (type === SIGNAL_TYPE.LOGIC) cssClass += ' logic-conn';
1633     else if (type === SIGNAL_TYPE.NUMERIC) cssClass += ' numeric-conn';
1634     else if (type === SIGNAL_TYPE.ANY) cssClass += ' any-conn';
1635
1636     path.setAttribute('class', cssClass);
1637
1638     // Обработчики событий
1639     path.style.pointerEvents = 'stroke';
1640     path.style.cursor = 'pointer';
1641     path.addEventListener('click', () => this.handleConnectionClick(conn));
1642
1643     svg.appendChild(path);
1644   });
1645
1646   if (typeof Outputs !== 'undefined' && Outputs.updateOutputStatus) {
1647     Outputs.updateOutputStatus();
1648   }
1649   Viewport.updateMinimap();
1650 },
1651 /**

```

```
1652     * Обработка клика по соединению (удаление)
1653     */
1654     handleConnectionClick(conn) {
1655         if (confirm('Удалить соединение?')) {
1656             AppState.connections = AppState.connections.filter(c =>
1657                 !(c.fromElement === conn.fromElement &&
1658                     c.fromPort === conn.fromPort &&
1659                     c.toElement === conn.toElement &&
1660                     c.toPort === conn.toPort)
1661             );
1662             this.drawConnections();
1663         }
1664     },
1665 },
1666 /**
1667 * Получение центра порта в координатах Canvas
1668 */
1669 _getPortCanvasCenter(portEl) {
1670     if (!portEl) return null;
1671
1672     const rect = portEl.getBoundingClientRect();
1673     return screenToCanvas(
1674         rect.left + rect.width / 2,
1675         rect.top + rect.height / 2
1676     );
1677 }
1678 };
1679 );
1680
1681 elements.js
1682
1683 /**
1684 * Модуль работы с элементами схемы
1685 */
1686
1687 const Elements = {
1688     /**
1689     * Генерация HTML для элемента
1690     */
1691     createElementHTML(elemType, elemId, x, y, props = {}, width, height) {
1692         const config = ELEMENT_TYPES[elemType];
1693         if (!config) throw new Error(`Неизвестный тип элемента: ${elemType}`);
1694
1695         const safe = (value, fallback = '') => (value === null || value ===
1696 undefined) ? fallback : String(value);
1697         const w = width ?? config.minLength ?? 120;
1698         const h = height ?? config.minLength ?? 60;
1699
1700         const getPortClass = (signalType, direction) => {
1701             const base = direction === 'output' ? 'port output' : 'port input';
1702             if (signalType === SIGNAL_TYPE.LOGIC) return `${base} logic-port`;
1703             if (signalType === SIGNAL_TYPE.NUMBER) return `${base} number-port`;
1704             return `${base} any-port`;
1705         };
1706
1707         // Эта функция buildConditionPort будет вызываться ИНАЧЕ, а не внутри
1708         innerHTML
1709         // Она тут остается, но ее результат не встраивается в HTML-строку
1710         // напрямую, кроме формулы
1711         const buildConditionPortHTML = () => {
1712             return `
1713                 <div class="condition-port-wrapper">
1714                     <div class="condition-port-label">условие</div>
1715                     <div class="port input condition-port"
1716                         data-port="cond-0"
```

```
1714                     data-element="${elemId}"  
1715                     data-signal-type="${SIGNAL_TYPE.LOGIC}"  
1716                     title="Техническое условие">  
1717                 </div>  
1718             </div>`;  
1719         };  
1720  
1721         const buildInputPorts = (count, types = [], labels = []) => {  
1722             let html = '';  
1723             for (let i = 0; i < count; i++) {  
1724                 const type = types[i] ?? types[types.length - 1] ??  
1725 SIGNAL_TYPE.ANY;  
1726                 html += `<div class="${getPortClass(type, 'input')}" data-  
port="in-${i}" data-element="${elemId}" data-signal-type="${type}" title="${labels[i]}  
|| 'Вход ${i+1}'`></div>`;  
1727             }  
1728             return html;  
1729         };  
1730  
1731         const buildOutputPorts = (count, types = [], labels = []) => {  
1732             let html = '';  
1733             for (let i = 0; i < count; i++) {  
1734                 const type = types[i] ?? types[types.length - 1] ??  
1735 SIGNAL_TYPE.ANY;  
1736                 html += `<div class="${getPortClass(type, 'output')}" data-  
port="out-${i}" data-element="${elemId}" data-signal-type="${type}" title="${labels[i]}  
|| 'Выход ${i+1}'`></div>`;  
1737             }  
1738             return html;  
1739         };  
1740  
1741         const resizeHandles = config.resizable ? `<div class="resize-handle  
handle-se" data-direction="se"></div><div class="resize-handle handle-e" data-  
direction="e"></div><div class="resize-handle handle-s" data-direction="s"></div>` :  
1742         ``;  
1743         // hasCondClass будет добавляться в addElement  
1744         // const hasCondClass = config.hasConditionPort ? 'has-condition-port' :  
1745         ``;  
1746  
1747         let innerHTML = '';  
1748  
1749         if (elemType === 'input-signal') {  
1750             const name = safe(props.name, 'Сигнал');  
1751             const type = props.signalType || SIGNAL_TYPE.NUMBER;  
1752             const symbol = type === SIGNAL_TYPE.LOGIC ? '☒' : '☒';  
1753             innerHTML =`  
1754                 <div class="element-header" style="background:$  
{config.color};">Источник</div>  
1755                 <div class="element-body">  
1756                     <div class="element-symbol">  
1757                         <span class="input-signal-icon">${symbol}</span>  
1758                         <span class="input-signal-name">${name}</span>  
1759                     </div>  
1760                     <div class="ports-right">  
1761                         ${buildOutputPorts(1, [type], ['Выход'])}  
1762                     </div>  
1763                 </div>`;  
1764             }  
1765             else if (elemType === 'const') {  
1766                 innerHTML =`  
1767                     <div class="element-header" style="background:$  
{config.color};">Константа</div>  
1768                     <div class="element-body">  
1769                         <div class="element-symbol">${props.value ?? 0}</div>
```

```
1767             <div class="ports-right">
1768                 ${buildOutputPorts(1, [SIGNAL_TYPE.NUMBER], ['Значение'])}
1769             </div>
1770         </div>`;
1771     }
1772     else if (elemType === 'separator') {
1773         innerHTML = `
1774             <div class="element-header" style="background:$
1775             {config.color};">Сепаратор</div>
1776             <div class="element-body">
1777                 <div class="ports-left">${buildInputPorts(1,
1778 config.inputTypes, config.inputLabels)}</div>
1779                 <div class="element-symbol">/\x</div>
1780                 <div class="ports-right">
1781                     <div class="port output logic-port true-port" data-
1782 port="out-0" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.TRUE}"
1783 title="ИСТИНА"></div>
1784                     <div class="port output logic-port false-port" data-
1785 port="out-1" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.FALSE}"
1786 title="ЛОЖЬ"></div>
1787                 </div>
1788             </div>`;
1789     }
1790     else if (elemType === 'and' || elemType === 'or') {
1791         const gateSymbol = elemType === 'and' ? 'Λ' : '∨';
1792         const inputCount = props.inputCount || config.defaultProps?.inputCount
1793         || 2;
1794         // Генерируем динамические входы
1795         let inputsHTML = '';
1796         for (let i = 0; i < inputCount; i++) {
1797             inputsHTML += `<div class="port input logic-port" data-port="in-$
1798 {i}" data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.LOGIC}" title="Вход $
1799 {i+1}"></div>`;
1800         }
1801         innerHTML = `
1802             <div class="element-header" style="background:${config.color};">$
1803             {config.name}</div>
1804             <div class="element-body">
1805                 <div class="ports-left">
1806                     ${inputsHTML}
1807                 </div>
1808                 <div class="element-symbol">${gateSymbol}</div>
1809                 <div class="ports-right">
1810                     <div class="port output logic-port" data-port="out-0"
1811 data-element="${elemId}" data-signal-type="${SIGNAL_TYPE.LOGIC}" title="Результат"></
1812 div>
1813                 </div>
1814             </div>`;
1815     }
1816     else if (elemType === 'if') {
1817         const op = safe(props.operator, '=');
1818         innerHTML = `
1819             <div class="element-header" style="background:$
1820             {config.color};">Условие</div>
1821             <div class="element-body">
1822                 <div class="ports-left">${buildInputPorts(2,
1823 config.inputTypes, config.inputLabels)}</div>
1824                 <div class="element-symbol">${op}</div>
1825                 <div class="ports-right">
1826                     ${buildOutputPorts(1, [SIGNAL_TYPE.LOGIC], ['результат'])}
1827                 </div>
1828             </div>`;
1829     }
1830 }
```

```
1818         else if (elemType === 'not') {
1819             innerHTML = `
1820                 <div class="element-header" style="background:$
1821 {config.color};">HE</div>
1822                     <div class="element-body">
1823                         <div class="ports-left">${buildInputPorts(1,
1824 [SIGNAL_TYPE.LOGIC], ['A'])}</div>
1825                         <div class="element-symbol">¬</div>
1826                         <div class="ports-right">
1827                             ${buildOutputPorts(1, [SIGNAL_TYPE.LOGIC], ['¬A'])}
1828                         </div>
1829                     </div>`;
1830     }
1831     else if (elemType === 'formula') {
1832         const inputCount = props.inputCount || config.defaultProps?.inputCount
1833         || config.inputs || 2;
1834         const expression = safe(props.expression);
1835         const displayExpression = expression
1836             ? (expression.length > 12 ? `${expression.slice(0, 12)}...` :
1837 expression)
1838             : 'f(x)';
1839         innerHTML = `
1840             ${buildConditionPortHTML()}
1841             <div class="element-header" style="background:$
1842 {config.color};">Формула</div>
1843                 <div class="element-body">
1844                     <div class="ports-left">${buildInputPorts(inputCount,
1845 config.inputTypes, config.inputLabels)}</div>
1846                     <div class="element-symbol">${displayExpression}</div>
1847                     <div class="ports-right">
1848                         ${buildOutputPorts(1, [SIGNAL_TYPE.NUMBER],
1849 ['Результат'])}
1850                         </div>
1851                     </div>`;
1852     }
1853     else if (elemType === 'output') {
1854         innerHTML = `
1855             <div class="element-header" style="background:$
1856 {config.color};">Выход</div>
1857                 <div class="element-body">
1858                     <div class="ports-left">
1859                         ${buildInputPorts(1, [SIGNAL_TYPE.ANY], ['сигнал'])}
1860                     </div>
1861                     <div class="element-symbol">${safe(props.label, 'Выход')}</
1862 div>
1863                     <div class="ports-right"></div>
1864                 </div>`;
1865     } else { // Для любых других (fallback)
1866         innerHTML = `
1867             <div class="element-header" style="background:$
1868 {config.name}</div>
1869                 <div class="element-body">
1870                     <div class="ports-left">${buildInputPorts(config.inputs || 0,
1871 config.inputTypes, config.inputLabels)}</div>
1872                     <div class="element-symbol">${config.name}</div>
1873                     <div class="ports-right">
1874                         ${buildOutputPorts(config.outputs || 0,
1875 config.outputTypes, config.outputLabels)}
1876                         </div>
1877                     </div>`;
1878     }
1879
1880     const html = `
```

```
1871             <div class="element ${elemType}" id="${elemId}"  
1872                 style="left:${x}px; top:${y}px; width:${w}px; height:${h}px;"  
1873             data-type="${elemType}">  
1874                 ${innerHTML}  
1875                 ${resizeHandles}  
1876             </div>`;  
1877  
1878         return { html, width: w, height: h };  
1879     },  
1880  
1881     /**  
1882      * Добавление элемента  
1883      */  
1884     addElement(elemType, x, y, props = {}, elemId = null, customWidth = null,  
1885     customHeight = null) {  
1886         const config = ELEMENT_TYPES[elemType];  
1887         if (!config) {  
1888             console.error(`Неизвестный тип элемента: ${elemType}`);  
1889             return null;  
1890         }  
1891         if (!elemId) {  
1892             elemId = `${elemType}-${++AppState.elementCounter}`;  
1893         }  
1894  
1895         let width = customWidth;  
1896         let height = customHeight;  
1897  
1898         if (width === null || width === undefined) {  
1899             width = config.minLength || 140;  
1900         }  
1901         if (height === null || height === undefined) {  
1902             height = config.minLength || 70;  
1903         }  
1904  
1905         try {  
1906             const result = this.createElementHTML(elemType, elemId, x, y, props,  
1907             width, height);  
1908             if (!result || !result.html) {  
1909                 console.error('createElementHTML вернул пустой результат');  
1910                 return null;  
1911             }  
1912             const workspace = document.getElementById('workspace');  
1913             const wrapper = document.createElement('div');  
1914             wrapper.innerHTML = result.html.trim();  
1915             const element = wrapper.firstChild;  
1916             if (!element) {  
1917                 console.error('Не удалось создать DOM элемент из HTML');  
1918                 return null;  
1919             }  
1920  
1921             // Добавляем класс для отступа  
1922             if (config.hasConditionPort) {  
1923                 element.classList.add('has-condition-port');  
1924             }  
1925             workspace.appendChild(element);  
1926  
1927             AppState.elements[elemId] = {  
1928                 id: elemId,  
1929                 type: elemType,  
1930                 x,  
1931                 y,  
1932                 width: result.width || width,
```

```
1933             height: result.height || height,
1934             props: { ...(config.defaultProps || {}), ... (props || {}) }
1935         );
1936
1937         // ЕСЛИ У ЭЛЕМЕНТА ЕСТЬ COND-ПОРТ (И ОН НЕ ФОРМУЛА, КОТОРАЯ УЖЕ ИМЕЕТ
1938         // ЕГО В HTML)
1939         if (config.hasConditionPort && elemType !== 'formula') {
1940             const condPortWrapper = document.createElement('div');
1941             condPortWrapper.innerHTML =
1942                 `<div class="condition-port-wrapper">
1943                     <div class="condition-port-label">условие</div>
1944                     <div class="port input condition-port"
1945                         data-port="cond-0"
1946                         data-element="${elemId}"
1947                         data-signal-type="${SIGNAL_TYPE.LOGIC}"
1948                         title="Техническое условие">
1949                         </div>
1950                     </div>`;
1951         element.prepend(condPortWrapper.firstChild); // Вставляем в
1952         самое начало элемента
1953     }
1954
1955     this.setupElementHandlers(elemId); // Передаем ID элемента
1956
1957     // Порты инициализируются внутри setupElementHandlers, нет нужды здесь
1958     // element.querySelectorAll('.port').forEach(port => {
1959     //     Connections.setupPortHandlers(port);
1960     // });
1961
1962     Connections.drawConnections(); // Перерисовываем соединения, чтобы
1963     // учесть новые порты
1964     Viewport.updateMinimap();
1965     return elemId;
1966 } catch (err) {
1967     console.error(`Ошибка при добавлении элемента ${elemType}:`, err);
1968     return null;
1969 }
1970
1971 /**
1972 * Обновление входов логического элемента (AND, OR)
1973 */
1974 updateLogicGateInputs(elemId, inputCount) {
1975     const elem = document.getElementById(elemId);
1976     if (!elem) return;
1977
1978     const portsLeft = elem.querySelector('.ports-left');
1979     if (!portsLeft) return;
1980
1981     // Удаляем соединения к портам, которые больше не существуют
1982     AppState.connections = AppState.connections.filter(c => {
1983         if (c.toElement === elemId && c.toPort.startsWith('in-')) {
1984             const portNum = parseInt(c.toPort.split('-')[1], 10);
1985             return portNum < inputCount;
1986         }
1987         return true;
1988     });
1989
1990     // Генерируем новые входы
1991     let inputsHTML = '';
1992     for (let i = 0; i < inputCount; i++) {
1993         inputsHTML += `
1994             <div class="port input logic-port"
1995                 data-port="in-${i}"`
```

```
1995             data-element="${elemId}"  
1996             data-signal-type="${SIGNAL_TYPE.LOGIC}"  
1997             title="Вход ${i+1}">  
1998         </div>  
1999     `;  
2000 }  
2001 portsLeft.innerHTML = inputsHTML;  
2002  
2003 // Переподключаем обработчики  
2004 portsLeft.querySelectorAll('.port').forEach(port =>  
2005     Connections.setupPortHandlers(port)  
2006 );  
2007  
2008     Connections.drawConnections();  
2009 },  
2010  
2011 /**  
2012 * Удаление элемента  
2013 */  
2014 deleteElement(elemId) {  
2015     AppState.connections = AppState.connections.filter(c =>  
2016         c.fromElement !== elemId && c.toElement !== elemId  
2017     );  
2018  
2019     const elem = document.getElementById(elemId);  
2020     if (elem) elem.remove();  
2021  
2022     delete AppState.elements[elemId];  
2023  
2024     if (AppState.selectedElement === elemId) {  
2025         AppState.selectedElement = null;  
2026     }  
2027  
2028     Connections.drawConnections();  
2029     Viewport.updateMinimap();  
2030 },  
2031  
2032 /**  
2033 * Выделение элемента  
2034 */  
2035 selectElement(elemId) {  
2036     if (AppState.selectedElement) {  
2037         const oldElem = document.getElementById(AppState.selectedElement);  
2038         if (oldElem) oldElem.classList.remove('selected');  
2039     }  
2040  
2041     AppState.selectedElement = elemId;  
2042     const elem = document.getElementById(elemId);  
2043     if (elem) elem.classList.add('selected');  
2044  
2045     const elemData = AppState.elements[elemId];  
2046     if (elemData) {  
2047         document.getElementById('selection-info').textContent =  
2048             `Выбрано: ${ELEMENT_TYPES[elemData.type]?.name || elemData.type}`;  
2049     }  
2050 },  
2051  
2052 /**  
2053 * Снять выделение  
2054 */  
2055 deselectAll() {  
2056     if (AppState.selectedElement) {  
2057         const elem = document.getElementById(AppState.selectedElement);  
2058         if (elem) elem.classList.remove('selected');  
2059         AppState.selectedElement = null;
```

```
2060     }
2061     document.getElementById('selection-info').textContent = '';
2062   },
2063
2064   /**
2065    * Настройка обработчиков элемента
2066    */
2067   setupElementHandlers(elemId) {
2068     try {
2069       const elem = document.getElementById(elemId);
2070       if (!elem) return;
2071
2072       elem.addEventListener('mousedown', (e) => {
2073         if (e.target.classList.contains('port')) return;
2074         if (e.target.classList.contains('resize-handle')) return;
2075
2076         e.preventDefault();
2077         e.stopPropagation();
2078
2079         this.selectElement(elemId);
2080
2081         AppState.draggingElement = elemId;
2082         const canvasPos = screenToCanvas(e.clientX, e.clientY);
2083         const elemData = AppState.elements[elemId];
2084         AppState.dragOffset.x = canvasPos.x - elemData.x;
2085         AppState.dragOffset.y = canvasPos.y - elemData.y;
2086       });
2087
2088       elem.addEventListener('dblclick', (e) => {
2089         if (e.target.classList.contains('port')) return;
2090         const config = ELEMENT_TYPES[AppState.elements[elemId].type];
2091         if (config?.hasProperties) {
2092           Modal.showPropertiesModal(elemId);
2093         }
2094       });
2095
2096       elem.addEventListener('contextmenu', (e) => {
2097         e.preventDefault();
2098         this.showContextMenu(e.clientX, e.clientY, elemId);
2099       });
2100
2101       const handles = elem.querySelectorAll('.resize-handle');
2102       handles.forEach(handle => this.setupResizeHandlers(handle, elemId));
2103
2104       const ports = elem.querySelectorAll('.port');
2105       ports.forEach(port => Connections.setupPortHandlers(port));
2106
2107     } catch (err) {
2108       console.error('setupElementHandlers error for', elemId, err);
2109     }
2110   },
2111
2112   /**
2113    * Контекстное меню
2114    */
2115   showContextMenu(x, y, elemId) {
2116     const menu = document.getElementById('context-menu');
2117     menu.style.left = `${x}px`;
2118     menu.style.top = `${y}px`;
2119     menu.style.display = 'block';
2120     menu.dataset.elementId = elemId;
2121   },
2122
2123   /**
2124    * Настройка resize
```

```
2125     */
2126     setupResizeHandlers(handle, elemId) {
2127         handle.addEventListener('mousedown', (e) => {
2128             e.stopPropagation();
2129             e.preventDefault();
2130
2131             const elemData = AppState.elements[elemId];
2132
2133             AppState.resizing = {
2134                 elemId: elemId,
2135                 handle: handle.dataset.direction,
2136                 startX: e.clientX,
2137                 startY: e.clientY,
2138                 startWidth: elemData.width,
2139                 startHeight: elemData.height,
2140                 startLeft: elemData.x,
2141                 startTop: elemData.y
2142             };
2143         });
2144     },
2145
2146     /**
2147      * Обработка resize
2148      */
2149     handleResize(e) {
2150         if (!AppState.resizing) return;
2151
2152         const { elemId, handle, startX, startY, startWidth, startHeight, startLeft,
2153               startTop } = AppState.resizing;
2154         const elem = document.getElementById(elemId);
2155         const elemData = AppState.elements[elemId];
2156         const config = ELEMENT_TYPES[elemData.type];
2157
2158         const dx = (e.clientX - startX) / AppState.viewport.zoom;
2159         const dy = (e.clientY - startY) / AppState.viewport.zoom;
2160
2161         let newWidth = startWidth;
2162         let newHeight = startHeight;
2163         let newLeft = startLeft;
2164         let newTop = startTop;
2165
2166         if (handle.includes('e')) {
2167             newWidth = Math.max(config.minWidth, startWidth + dx);
2168         }
2169         if (handle.includes('w')) {
2170             newWidth = Math.max(config.minWidth, startWidth - dx);
2171             newLeft = startLeft + (startWidth - newWidth);
2172         }
2173         if (handle.includes('s')) {
2174             newHeight = Math.max(config.minLength, startHeight + dy);
2175         }
2176         if (handle.includes('n')) {
2177             newHeight = Math.max(config.minLength, startHeight - dy);
2178             newTop = startTop + (startHeight - newHeight);
2179         }
2180
2181         elem.style.width = `${newWidth}px`;
2182         elem.style.height = `${newHeight}px`;
2183         elem.style.left = `${newLeft}px`;
2184         elem.style.top = `${newTop}px`;
2185
2186         elemData.width = newWidth;
2187         elemData.height = newHeight;
2188         elemData.x = newLeft;
2189         elemData.y = newTop;
```

```
2189     Connections.drawConnections();
2190 },
2191 /**
2192 * Обработка перетаскивания элемента
2193 */
2194 handleDrag(e) {
2195     if (!AppState.draggingElement) return;
2196
2197     const canvasPos = screenToCanvas(e.clientX, e.clientY);
2198     const x = canvasPos.x - AppState.dragOffset.x;
2199     const y = canvasPos.y - AppState.dragOffset.y;
2200
2201     const elemId = AppState.draggingElement;
2202     const elem = document.getElementById(elemId);
2203     const elemData = AppState.elements[elemId];
2204
2205     elem.style.left = `${x}px`;
2206     elem.style.top = `${y}px`;
2207
2208     elemData.x = x;
2209     elemData.y = y;
2210
2211     Connections.drawConnections();
2212 },
2213 /**
2214 * Обновление входов формулы
2215 */
2216 updateFormulaInputs(elemId, inputCount) {
2217     const elem = document.getElementById(elemId);
2218     if (!elem) return;
2219
2220     const portsLeft = elem.querySelector('.ports-left');
2221     if (!portsLeft) return;
2222
2223     AppState.connections = AppState.connections.filter(c => {
2224         if (c.toElement === elemId && c.toPort.startsWith('in-')) {
2225             const portNum = parseInt(c.toPort.split('-')[1], 10);
2226             return portNum < inputCount;
2227         }
2228         return true;
2229     });
2230
2231     let inputsHTML = '';
2232     for (let i = 0; i < inputCount; i++) {
2233         inputsHTML += `
2234             <div class="port input any-port"
2235                 data-port="in-${i}"
2236                 data-element="${elemId}"
2237                 data-signal-type="${SIGNAL_TYPE.ANY}"
2238                 title="in${i} (Любой)">
2239                 </div>
2240             `;
2241     }
2242     portsLeft.innerHTML = inputsHTML;
2243
2244     portsLeft.querySelectorAll('.port').forEach(port =>
2245         Connections.setupPortHandlers(port)
2246     );
2247
2248     Connections.drawConnections();
2249 },
2250
2251 },
2252
2253 }
```

```
2254 	/***
2255   * Рассчитать оптимальный размер элемента на основе количества портов
2256   */
2257 	calculateOptimalHeight(elemId, inputCount, outputCount = 1) {
2258  	const elem = AppState.elements[elemId];
2259  	if (!elem) return null;
2260
2261  	const config = ELEMENT_TYPES[elem.type];
2262  	if (!config || !config.resizable) return null;
2263
2264  	// Базовая высота
2265  	let baseHeight = config.minLength || 60;
2266
2267  	// Каждый порт требует примерно 25-30px высоты
2268  	const portSpacing = 28;
2269  	const maxPorts = Math.max(inputCount, outputCount);
2270
2271  	// Добавляем высоту для портов (кроме первого, который уже в baseHeight)
2272  	const additionalHeight = (maxPorts - 1) * portSpacing;
2273  	const newHeight = Math.max(baseHeight, baseHeight + additionalHeight);
2274
2275  	return newHeight;
2276 },
2277
2278 /**
2279  * Обновление размера элемента при изменении портов
2280  */
2281 updateElementSize(elemId) {
2282  	const elem = document.getElementById(elemId);
2283  	const elemData = AppState.elements[elemId];
2284
2285  	if (!elem || !elemData) return;
2286
2287  	const config = ELEMENT_TYPES[elemData.type];
2288  	if (!config || !config.resizable) return;
2289
2290  	let inputCount = 0;
2291  	let outputCount = config.outputs || 1;
2292
2293  	// Определяем количество входов
2294  	if (elemData.type === 'and' || elemData.type === 'or' || elemData.type ===
2295 'formula') {
2296    	inputCount = elemData.props.inputCount || config.inputs || 2;
2297   } else {
2298    	inputCount = config.inputs || 0;
2299   }
2300
2301  	// Рассчитываем новую высоту
2302  	const newHeight = this.calculateOptimalHeight(elemId, inputCount,
2303  	outputCount);
2304
2305  	if (newHeight && newHeight !== elemData.height) {
2306    	elemData.height = newHeight;
2307    	elem.style.height = `${newHeight}px`;
2308
2309    	// Перерисовываем соединения, т.к. изменился размер элемента
2310    	Connections.drawConnections();
2311    	Viewport.updateMinimap();
2312   }
2313
2314 }
2315
2316 modal.js
```

```
2317
2318 /**
2319  * Модуль модальных окон
2320 */
2321
2322 const Modal = {
2323     /**
2324      * Инициализация модальных окон
2325      */
2326     init() {
2327         // Модальное окно свойств элемента
2328         document.getElementById('modal-save').addEventListener('click', () => {
2329             this.saveElementProperties();
2330         });
2331
2332         document.getElementById('modal-cancel').addEventListener('click', () => {
2333             this.hideModal('modal-overlay');
2334         });
2335
2336         document.getElementById('modal-overlay').addEventListener('click', (e) => {
2337             if (e.target.id === 'modal-overlay') {
2338                 this.hideModal('modal-overlay');
2339             }
2340         });
2341
2342         // Модальное окно свойств проекта
2343         document.getElementById('project-modal-save').addEventListener('click', () =>
2344         {
2345             this.saveProjectProperties();
2346         });
2347
2348         document.getElementById('project-modal-cancel').addEventListener('click', () =>
2349         {
2350             this.hideModal('project-modal-overlay');
2351         });
2352
2353         document.getElementById('project-modal-overlay').addEventListener('click', (e) => {
2354             if (e.target.id === 'project-modal-overlay') {
2355                 this.hideModal('project-modal-overlay');
2356             }
2357         });
2358
2359         /**
2360          * Показать модальное окно
2361          */
2362         showModal(modalId) {
2363             document.getElementById(modalId).style.display = 'flex';
2364         },
2365
2366         /**
2367          * Скрыть модальное окно
2368          */
2369         hideModal(modalId) {
2370             document.getElementById(modalId).style.display = 'none';
2371         },
2372
2373         /**
2374          * Показать свойства элемента
2375          */
2376         showPropertiesModal(elemId) {
2377             const elemData = AppState.elements[elemId];
2378             const elemType = elemData.type;
2379             const props = elemData.props;
```

```
2379     const config = ELEMENT_TYPES[elemType];
2380
2381     const modalOverlay = document.getElementById('modal-overlay');
2382     const modalTitle = document.getElementById('modal-title');
2383     const modalContent = document.getElementById('modal-content');
2384
2385     modalTitle.textContent = `Свойства: ${config.name}`;
2386
2387     let contentHTML = '';
2388
2389     if (elemType === 'input-signal') {
2390         const signalType = props.signalType || SIGNAL_TYPE.NUMBER;
2391         contentHTML =
2392             `

2393                 <label>Название сигнала:</label>
2394                 <input type="text" id="prop-name" value="${props.name} ||
2395 'Сигнал'}>
2396             </div>
2397             <div class="modal-row">
2398                 <label>Тип сигнала:</label>
2399                 <select id="prop-signal-type">
2400                     <option value="${SIGNAL_TYPE.NUMBER}" ${signalType ===
2401 SIGNAL_TYPE.NUMBER ? 'selected' : ''}>Числовой</option>
2402                     <option value="${SIGNAL_TYPE.LOGIC}" ${signalType ===
2403 SIGNAL_TYPE.LOGIC ? 'selected' : ''}>Логический</option>
2404                 </select>
2405             </div>
2406         `;
2407     } else if (elemType === 'if') {
2408         contentHTML =
2409             `

2410                 <label>Оператор сравнения:</label>
2411                 <select id="prop-operator">
2412                     <option value="=" ${props.operator === '=' ? 'selected' : ''}
2413 '>= (равно)</option>
2414                     <option value=">" ${props.operator === '>' ? 'selected' : ''}
2415 '>> (больше)</option>
2416                     <option value="<" ${props.operator === '<' ? 'selected' : ''}
2417 '>< (меньше)</option>
2418                     <option value=">=" ${props.operator === '>=' ? 'selected' :
2419 '>=' (больше или равно)}</option>
2420                     <option value="<=" ${props.operator === '<=' ? 'selected' :
2421 '<=' (меньше или равно)}</option>
2422                     <option value!="!" ${props.operator === '!=' ? 'selected' :
2423 '!>!= (не равно)}</option>
2424                 </select>
2425             </div>
2426         `;
2427     } else if (elemType === 'and' || elemType === 'or') {
2428         contentHTML =
2429             `

2430                 <label>Количество входов:</label>
2431                 <input type="number" id="prop-input-count" value="$
2432 {props.inputCount || 2}" min="2" max="10">
2433             </div>
2434             <div class="modal-row">
2435                 <p style="color: #aaa; font-size: 12px;">
2436                     Измените количество входных портов для этого логического
2437 элемента.
2438                     Лишние соединения будут автоматически удалены.
2439                 </p>
2440             </div>
2441         `;
2442     } else if (elemType === 'const') {
2443         contentHTML =
2444


```

```
2433             <div class="modal-row">
2434                 <label>Значение:</label>
2435                 <input type="number" id="prop-value" value="${props.value ?? 0}"
2436                     step="any">
2437                     </div>
2438             ;
2439         } else if (elementType === 'formula') {
2440             let signalsHTML = '';
2441             AppState.connections.forEach(conn => {
2442                 if (conn.toElement === elemId) {
2443                     const fromElem = AppState.elements[conn.fromElement];
2444                     if (fromElem) {
2445                         const signalName = fromElem.props?.name || fromElem.id;
2446                         signalsHTML += `<div class="signal-item" data-signal="$
2447                             {signalName}">${signalName} (${conn.toPort})</div>`;
2448                     }
2449                 }
2450             });
2451             contentHTML =
2452                 <div class="modal-row">
2453                     <label>Количество входов:</label>
2454                     <input type="number" id="prop-input-count" value="$
2455                         {props.inputCount || 2}" min="1" max="10">
2456                     </div>
2457                     <div class="modal-row">
2458                         <label>Входные сигналы (двойной клик для вставки):</label>
2459                         <div class="signal-list" id="signal-list">
2460                             ${signalsHTML} || '<div style="color:#888;padding:5px;">Нет
2461                             подключённых сигналов</div>'>
2462                         </div>
2463                     <div class="modal-row">
2464                         <label>Выражение:</label>
2465                         <textarea id="prop-expression">${props.expression || ''}</
2466                         textarea>
2467                     </div>
2468             ;
2469         } else if (elementType === 'output') {
2470             contentHTML =
2471                 <div class="modal-row">
2472                     <label>Название выхода:</label>
2473                     <input type="text" id="prop-label" value="${props.label ||
2474                         'Выход'}">
2475                     </div>
2476                     <div class="modal-row">
2477                         <label>Группировка (опционально):</label>
2478                         <input type="text" id="prop-output-group" value="$
2479                         {props.outputGroup || ''}" placeholder="для логической группировки выходов">
2480                     </div>
2481             ;
2482         }
2483     }
2484
2485     // Обработчик вставки сигналов для формулы
2486     if (elementType === 'formula') {
2487         document.querySelectorAll('.signal-item').forEach(item => {
2488             item.addEventListener('dblclick', () => {
2489                 const signal = item.dataset.signal;
2490                 const textarea = document.getElementById('prop-expression');
2491                 textarea.value += signal;
```

```
2491             textarea.focus();
2492         });
2493     });
2494 }
2495 /**
2496 * Сохранить свойства элемента
2497 */
2500 /**
2501 * Сохранить свойства элемента
2502 */
2503 saveElementProperties() {
2504     try {
2505         const modalOverlay = document.getElementById('modal-overlay');
2506         const elemId = modalOverlay.dataset.elementId;
2507         const elemData = AppState.elements[elemId];
2508         const elemType = elemData.type;
2509         const elem = document.getElementById(elemId);
2510
2511         if (elemType === 'input-signal') {
2512             const name = document.getElementById('prop-name').value || 'Сигнал';
2513             const signalType = document.getElementById('prop-signal-type').value;
2514
2515             const oldSignalType = elemData.props.signalType;
2516             elemData.props.name = name;
2517             elemData.props.signalType = signalType;
2518
2519             if (oldSignalType !== signalType) {
2520                 AppState.connections = AppState.connections.filter(conn => {
2521                     if (conn.fromElement === elemId) {
2522                         const toPortIndex = parseInt(conn.toPort.split('-')[1]);
2523                         const inputType = getInputPortType(conn.toElement,
2524                             toPortIndex);
2525                         return areTypesCompatible(signalType, inputType);
2526                     }
2527                     return true;
2528                 });
2529             }
2530             // Перерисовываем элемент
2531             // Перерисовываем элемент
2532             const { html } = Elements.createElementHTML(
2533                 elemType,
2534                 elemId,
2535                 elemData.x,
2536                 elemData.y,
2537                 elemData.props,
2538                 elemData.width,
2539                 elemData.height
2540             );
2541             elem.outerHTML = html;
2542
2543             // Находим новый DOM-элемент
2544             const newElem = document.getElementById(elemId);
2545
2546             // Заново навешиваем обработчики на новый элемент
2547             Elements.setupElementHandlers(elemId);
2548
2549             if (AppState.selectedElement === elemId && newElem) {
2550                 newElem.classList.add('selected');
2551             }
2552
2553             Connections.drawConnections();
2554 }
```

```
2555 } else if (elemType === 'if') {
2556     const operator = document.getElementById('prop-operator').value;
2557     elemData.props.operator = operator;
2558     const symbol = elem.querySelector('.element-symbol');
2559     if (symbol) symbol.textContent = operator;
2560
2561 } else if (elemType === 'const') {
2562     const value = parseFloat(document.getElementById('prop-value').value)
2563     || 0;
2564     elemData.props.value = value;
2565     const symbol = elem.querySelector('.element-symbol');
2566     if (symbol) symbol.textContent = String(value);
2567
2568 } else if (elemType === 'formula') {
2569     const expression = document.getElementById('prop-expression').value;
2570     const inputCount = parseInt(document.getElementById('prop-input-
count').value) || 2;
2571     elemData.props.expression = expression;
2572     elemData.props.inputCount = inputCount;
2573
2574     const symbol = elem.querySelector('.element-symbol');
2575     if (symbol) {
2576         symbol.textContent = expression.length > 12 ? '$
{expression.slice(0, 12)}...` : (expression || 'f(x)');
2577     }
2578
2579     Elements.updateFormulaInputs(elemId, inputCount);
2580     Elements.updateElementSize(elemId); // ← Добавляем это
2581 } else if (elemType === 'and' || elemType === 'or') {
2582     const inputCount = parseInt(document.getElementById('prop-input-
count').value) || 2;
2583     elemData.props.inputCount = inputCount;
2584
2585     Elements.updateLogicGateInputs(elemId, inputCount);
2586     Elements.updateElementSize(elemId); // ← Добавляем это
2587
2588     const symbol = elem.querySelector('.element-symbol');
2589     if (symbol) {
2590         symbol.textContent = elemType === 'and' ? 'Λ' : '∨';
2591     }
2592
2593 } else if (elemType === 'output') {
2594     const label = document.getElementById('prop-label').value || 'Выход';
2595     const outputGroup = document.getElementById('prop-output-group').value
2596     || '';
2597
2598     elemData.props.label = label;
2599     elemData.props.outputGroup = outputGroup;
2600
2601     const symbol = elem.querySelector('.element-symbol');
2602     if (symbol) symbol.textContent = label;
2603
2604     this.hideModal('modal-overlay');
2605
2606 } catch (error) {
2607     console.error('Ошибка при сохранении свойств:', error);
2608     alert('Ошибка сохранения: ' + error.message);
2609 }
2610
2611 /**
2612 * Показать свойства проекта
2613 */
2614
```

```
2615     showProjectPropertiesModal() {
2616         const content = document.getElementById('project-modal-content');
2617         const project = AppState.project;
2618
2619         // Генерируем HTML для списка выходов только если модуль загружен
2620         let outputsHtml = '';
2621         if (typeof Outputs !== 'undefined' && AppState.outputs) {
2622             const logicalOutputsHtml = AppState.outputs.logical.length > 0
2623                 ? AppState.outputs.logical.map(output =>
2624                     <div class="output-item"
2625                         data-element-id="${output.elementId}"
2626                         onmouseenter="Outputs.highlightOutput('${output.elementId}', true)"
2627                         onmouseleave="Outputs.highlightOutput('${output.elementId}', false)"
2628                         onclick="Outputs.navigateToOutput('${output.elementId}')";
2629             Modal.hideModal('project-modal-overlay');"
2630             <span class="output-icon">>${output.portLabel} ==> ${output.elementName}</span>
2631             <span class="output-port">> ${output.portLabel}</span>
2632             </div>
2633             `).join('')
2634             : '<div class="no-outputs">Нет логических выходов</div>';
2635
2636             const numericOutputsHtml = AppState.outputs.numeric.length > 0
2637                 ? AppState.outputs.numeric.map(output =>
2638                     <div class="output-item numeric"
2639                         data-element-id="${output.elementId}"
2640                         onmouseenter="Outputs.highlightOutput('${output.elementId}', true)"
2641                         onmouseleave="Outputs.highlightOutput('${output.elementId}', false)"
2642                         onclick="Outputs.navigateToOutput('${output.elementId}')";
2643             Modal.hideModal('project-modal-overlay');"
2644             <span class="output-icon">> ${output.elementName}</span>
2645             <span class="output-port">> значение</span>
2646             </div>
2647             `).join('')
2648             : '<div class="no-outputs">Нет числовых выходов</div>';
2649
2650             outputsHtml =
2651             <div class="modal-row">
2652                 <label>Выходные сигналы схемы:</label>
2653                 <div class="outputs-container">
2654                     <div class="outputs-section">
2655                         <div class="outputs-section-title">
2656                             <span class="section-icon">> X </span>
2657                             Логические выходы (${AppState.outputs.logical.length})
2658                         </div>
2659                         <div class="outputs-list">
2660                             ${logicalOutputsHtml}
2661                         </div>
2662                     </div>
2663                     <div class="outputs-section">
2664                         <div class="outputs-section-title">
2665                             <span class="section-icon">> Y </span>
2666                             Числовые выходы (${AppState.outputs.numeric.length})
2667                         </div>
2668                         <div class="outputs-list">
2669                             ${numericOutputsHtml}
2670                         </div>
2671                     </div>
2672                 </div>
```



```
2728         btn.classList.add('active');
2729
2730         const type = btn.dataset.type;
2731         document.getElementById('parameter-
2732         fields').classList.toggle('visible', type === PROJECT_TYPE.PARAMETER);
2733         document.getElementById('rule-fields').classList.toggle('visible',
2734         type === PROJECT_TYPE.RULE);
2735     });
2736
2737     this.showModal('project-modal-overlay');
2738 },
2739 /**
2740 * Сохранить свойства проекта
2741 */
2742 saveProjectProperties() {
2743     const activeTypeBtn = document.querySelector('.project-type-btn.active');
2744     const type = activeTypeBtn ? activeTypeBtn.dataset.type :
PROJECT_TYPE.PARAMETER;
2745
2746     AppState.project.code = document.getElementById('project-code').value;
2747     AppState.project.type = type;
2748
2749     if (type === PROJECT_TYPE.PARAMETER) {
2750         AppState.project.dimension = document.getElementById('project-
dimension').value;
2751         AppState.project.possibleCause = '';
2752         AppState.project.guidelines = '';
2753     } else {
2754         AppState.project.dimension = '';
2755         AppState.project.possibleCause = document.getElementById('project-
possible-cause').value;
2756         AppState.project.guidelines = document.getElementById('project-
guidelines').value;
2757     }
2758
2759     this.hideModal('project-modal-overlay');
2760 }
2761 };
2762
2763 output.js
2764 /**
2765 * Модуль управления выходными сигналами
2766 */
2767
2768 const Outputs = {
2769 /**
2770     * Обновление статуса выходных элементов
2771     * Вызывается при каждом изменении схемы
2772     */
2773     updateOutputStatus() {
2774         this.clearAllOutputHighlights();
2775         AppState.outputs.logical = [];
2776         AppState.outputs.numeric = [];
2777         updateFrameChildren();
2778
2779     // Обработка элементов-выходов
2780     Object.values(AppState.elements).forEach(elem => {
2781         if (!elem || elem.type !== 'output') return;
2782
2783         // Проверяем, к чему подключен вход этого выхода
2784         const inputConns = AppState.connections.filter(c =>
2785             c.toElement === elem.id && c.toPort === 'in-0'
2786         );
2787     });
2788 }
```

```
2787 // Каждое соединение к выходу – это отдельный выход
2788 inputConns.forEach((conn, index) => {
2789     const fromElem = AppState.elements[conn.fromElement];
2790     if (!fromElem) return;
2791
2792     const outputType = conn.signalType;
2793     const outputInfo = {
2794         id: `${elem.id}_conn_${index}`,
2795         elementId: elem.id,
2796         sourceElementId: conn.fromElement,
2797         sourcePort: conn.fromPort,
2798         portIndex: 0,
2799         portId: 'in-0',
2800         type: outputType,
2801         label: elem.props?.label || 'Выход',
2802         elementType: 'output',
2803         elementName: elem.props?.label || 'Выход',
2804         name: elem.props?.label || 'Выход'
2805     };
2806
2807     if (outputType === SIGNAL_TYPE.LOGIC) {
2808         AppState.outputs.logical.push(outputInfo);
2809     } else if (outputType === SIGNAL_TYPE.NUMBER) {
2810         AppState.outputs.numeric.push(outputInfo);
2811     }
2812
2813     // Подсветим входной порт
2814     this.highlightOutputPort(elem.id, 0, outputType);
2815 });
2816 });
2817
2818     this.updateOutputCounter();
2819 },
2820
2821 /**
2822 * Очистка всех выделений выходов
2823 */
2824 clearAllOutputHighlights() {
2825     document.querySelectorAll('.port.output-active').forEach(port => {
2826         port.classList.remove('output-active');
2827     });
2828
2829     document.querySelectorAll('.element.has-output').forEach(elem => {
2830         elem.classList.remove('has-output');
2831     });
2832
2833     document.querySelectorAll('.element.output-ambiguous').forEach(el =>
2834 el.classList.remove('output-ambiguous'));
2835     document.querySelectorAll('.element.output-missing').forEach(el =>
2836 el.classList.remove('output-missing'));
2837
2838 /**
2839 * Выделение выходного порта
2840 */
2841 highlightOutputPort(elemId, portIndex, portType) {
2842     const elem = document.getElementById(elemId);
2843     if (!elem) return;
2844
2845     const port = elem.querySelector(`.port.output[data-port="out-${portIndex}]`);
2846     if (port) {
2847         port.classList.add('output-active');
2848     }
2849 }
```

```
2850         // Добавляем класс элементу (даёт общий визуал)
2851         elem.classList.add('has-output');
2852     },
2853
2854     /**
2855      * Обновление счётчика выходов в меню
2856      */
2857     updateOutputCounter() {
2858         const counter = document.getElementById('output-counter');
2859         if (counter) {
2860             const total = AppState.outputs.logical.length +
2861             AppState.outputs.numeric.length;
2862             counter.textContent = total;
2863             counter.style.display = total > 0 ? 'inline-block' : 'none';
2864         },
2865
2866     /**
2867      * Получить все выходы для сохранения в проект
2868      */
2869     getOutputsForSave() {
2870         // Сохраняем информацию о frame/inner для рамок
2871         return {
2872             logical: AppState.outputs.logical.map(o => ({
2873                 id: o.id,
2874                 elementId: o.elementId,
2875                 frameId: o.frameId || null,
2876                 innerElementId: o.innerElementId || null,
2877                 portIndex: o.portIndex ?? o.innerPortIndex ?? null,
2878                 portLabel: o.label
2879             })),
2880             numeric: AppState.outputs.numeric.map(o => ({
2881                 id: o.id,
2882                 elementId: o.elementId,
2883                 frameId: o.frameId || null,
2884                 innerElementId: o.innerElementId || null,
2885                 portIndex: o.portIndex ?? o.innerPortIndex ?? null,
2886                 portLabel: o.label
2887             }))
2888         };
2889     },
2890
2891     /**
2892      * Подсветить конкретный выход (при наведении в списке)
2893      */
2894     highlightOutput(elementId, highlight = true) {
2895         const elem = document.getElementById(elementId);
2896         if (elem) {
2897             if (highlight) {
2898                 elem.classList.add('output-highlighted');
2899             } else {
2900                 elem.classList.remove('output-highlighted');
2901             }
2902         }
2903     },
2904
2905     /**
2906      * Перейти к элементу выхода на схеме (elementId – фокусируемый элемент; для рамок
2907      * это id рамки)
2908      */
2909     navigateToOutput(elementId) {
2910         const elemData = AppState.elements[elementId];
2911         if (!elemData) return;
2912
2913         // Центрируем viewport на элементе
```

```
2913     const container = document.getElementById('workspace-container');
2914     const rect = container.getBoundingClientRect();
2915
2916     const centerX = elemData.x + elemData.width / 2;
2917     const centerY = elemData.y + elemData.height / 2;
2918
2919     AppState.viewport.panX = rect.width / 2 - centerX * AppState.viewport.zoom;
2920     AppState.viewport.panY = rect.height / 2 - centerY * AppState.viewport.zoom;
2921
2922     Viewport.updateTransform();
2923
2924     // Выделяем элемент
2925     Elements.selectElement(elementId);
2926
2927     // Временная подсветка
2928     this.highlightOutput(elementId, true);
2929     setTimeout(() => this.highlightOutput(elementId, false), 2000);
2930   }
2931 };
2932
2933 project.js
2934
2935 /**
2936  * Модуль управления проектом (сохранение, загрузка)
2937 */
2938
2939 const Project = {
2940   /**
2941    * Инициализация
2942    */
2943   init() {
2944     document.getElementById('btn-new').addEventListener('click', () =>
2945       this.newProject());
2946     document.getElementById('btn-save').addEventListener('click', () =>
2947       this.saveProject());
2948     document.getElementById('btn-load').addEventListener('click', () => {
2949       document.getElementById('file-input').click();
2950     });
2951     document.getElementById('btn-project-settings').addEventListener('click', () => {
2952       Modal.showProjectPropertiesModal();
2953     });
2954
2955     document.getElementById('file-input').addEventListener('change', (e) => {
2956       const file = e.target.files[0];
2957       if (file) {
2958         const reader = new FileReader();
2959         reader.onload = (ev) => this.loadProject(ev.target.result);
2960         reader.readAsText(file);
2961       }
2962       e.target.value = '';
2963     });
2964   /**
2965    * Новый проект
2966    */
2967   newProject() {
2968     if (Object.keys(AppState.elements).length > 0) {
2969       if (!confirm('Создать новый проект? Несохранённые изменения будут
потеряны.')) {
2970         return;
2971       }
2972     }
2973   }
```

```
2974     document.getElementById('workspace').innerHTML = '';
2975     document.getElementById('connections-svg').innerHTML = '';
2976 
2977     resetState();
2978     Viewport.updateTransform();
2979 },
2980 
2981 /**
2982 * Сохранение проекта
2983 */
2984 saveProject() {
2985     // Проверяем, заполнены ли свойства проекта
2986     if (!AppState.project.code) {
2987         Modal.showProjectPropertiesModal();
2988         alert('Пожалуйста, укажите код проекта перед сохранением.');
2989         return;
2990     }
2991 
2992     updateFrameChildren();
2993 
2994     const project = {
2995         version: '1.0',
2996         project: AppState.project,
2997         elements: AppState.elements,
2998         connections: AppState.connections,
2999         counter: AppState.elementCounter,
3000         viewport: {
3001             zoom: AppState.viewport.zoom,
3002             panX: AppState.viewport.panX,
3003             panY: AppState.viewport.panY
3004         }
3005     };
3006 
3007     const jsonStr = JSON.stringify(project, null, 2);
3008     const blob = new Blob([jsonStr], { type: 'application/json' });
3009     const url = URL.createObjectURL(blob);
3010 
3011     const filename = `${AppState.project.code || 'scheme'}_${$`AppState.project.type`}.json`;
3012 
3013     const a = document.createElement('a');
3014     a.href = url;
3015     a.download = filename;
3016     a.click();
3017 
3018     URL.revokeObjectURL(url);
3019 },
3020 
3021 /**
3022 * Загрузка проекта
3023 */
3024 loadProject(jsonStr) {
3025     try {
3026         const data = JSON.parse(jsonStr);
3027 
3028         // Очищаем
3029         document.getElementById('workspace').innerHTML = '';
3030         document.getElementById('connections-svg').innerHTML = '';
3031         resetState();
3032 
3033         // Загружаем свойства проекта
3034         if (data.project) {
3035             AppState.project = { ...AppState.project, ...data.project };
3036         }
3037 
```

```
3038     // Загружаем состояние
3039     AppState.elementCounter = data.counter || 0;
3040
3041     // Загружаем viewport
3042     if (data.viewport) {
3043         AppState.viewport.zoom = data.viewport.zoom || 1;
3044         AppState.viewport.panX = data.viewport.panX || 0;
3045         AppState.viewport.panY = data.viewport.panY || 0;
3046     }
3047
3048     // Сначала загружаем рамки
3049     Object.values(data.elements || {})
3050         .filter(e => e.type === 'output-frame')
3051         .forEach(elemData => {
3052             Elements.addElement(
3053                 elemData.type,
3054                 elemData.x,
3055                 elemData.y,
3056                 elemData.props,
3057                 elemData.id,
3058                 elemData.width,
3059                 elemData.height
3060             );
3061         });
3062
3063     // Затем остальные элементы
3064     Object.values(data.elements || {})
3065         .filter(e => e.type !== 'output-frame')
3066         .forEach(elemData => {
3067             Elements.addElement(
3068                 elemData.type,
3069                 elemData.x,
3070                 elemData.y,
3071                 elemData.props,
3072                 elemData.id,
3073                 elemData.width,
3074                 elemData.height
3075             );
3076         });
3077
3078     AppState.connections = data.connections || [];
3079
3080     Viewport.updateTransform();
3081     Connections.drawConnections();
3082
3083 } catch (e) {
3084     alert('Ошибка загрузки: ' + e.message);
3085     console.error(e);
3086 }
3087 }
3088 };
3089
3090 state.js
3091 /**
3092 * Глобальное состояние приложения
3093 */
3094
3095 const AppState = {
3096     // Элементы схемы
3097     elements: {},
3098     connections: [],
3099     elementCounter: 0,
3100
3101     // Выделение
```

```
3103     selectedElement: null,
3104
3105     // Перетаскивание
3106     draggingElement: null,
3107     dragOffset: { x: 0, y: 0 },
3108     isDraggingFromPalette: false,
3109     dragPreview: null,
3110     dragType: null,
3111
3112     // Соединения
3113     connectingFrom: null,
3114     connectingFromType: null,
3115     tempLine: null,
3116
3117     // Resize
3118     resizing: null,
3119
3120     // Viewport (масштабирование и перемещение)
3121     viewport: {
3122         zoom: 1,
3123         panX: 0,
3124         panY: 0,
3125         isPanning: false,
3126         lastMouseX: 0,
3127         lastMouseY: 0
3128     },
3129
3130     // Свойства проекта
3131     project: {
3132         code: '',
3133         type: PROJECT_TYPE.PARAMETER,
3134         // Для параметра
3135         dimension: '',
3136         // Для правила
3137         possibleCause: '',
3138         guidelines: ''
3139     },
3140
3141     // Выходные сигналы (автоматически определяются)
3142     outputs: {
3143         logical: [],    // Логические выходы [{elementId, portIndex, portLabel, ...}]
3144         numeric: []     // Числовые выходы (формулы)
3145     }
3146 };
3147
3148 /**
3149 * Сброс состояния
3150 */
3151 function resetState() {
3152     AppState.elements = {};
3153     AppState.connections = [];
3154     AppState.elementCounter = 0;
3155     AppState.selectedElement = null;
3156     AppState.draggingElement = null;
3157     AppState.connectingFrom = null;
3158     AppState.tempLine = null;
3159     AppState.resizing = null;
3160
3161     AppState.viewport = {
3162         zoom: 1,
3163         panX: 0,
3164         panY: 0,
3165         isPanning: false,
3166         lastMouseX: 0,
3167         lastMouseY: 0
3168     }
3169 }
```

```
3168     };
3169
3170     AppState.project = {
3171         code: '',
3172         type: PROJECT_TYPE.PARAMETER,
3173         dimension: '',
3174         possibleCause: '',
3175         guidelines: ''
3176     };
3177
3178     AppState.outputs = {
3179         logical: [],
3180         numeric: []
3181     };
3182 }
3183
3184 utils.js
3185
3186 /**
3187 * Вспомогательные функции
3188 */
3189
3190 /**
3191 * Генерация уникального ID
3192 */
3193 function generateId() {
3194     AppState.elementCounter++;
3195     return `elem_${AppState.elementCounter}`;
3196 }
3197
3198 function getInputPortType(elementId, portIdentifier) {
3199     const element = AppState.elements[elementId];
3200     if (!element) return SIGNAL_TYPE.ANY;
3201
3202     const config = ELEMENT_TYPES[element.type];
3203     if (!config) return SIGNAL_TYPE.ANY;
3204
3205     let portIndex = portIdentifier;
3206
3207     // Обработка технического порта условия
3208     if (typeof portIdentifier === 'string') {
3209         if (portIdentifier === 'cond-0' && config.hasConditionPort) {
3210             return config.conditionPortType || SIGNAL_TYPE.LOGIC;
3211         }
3212
3213         if (portIdentifier.startsWith('in-')) {
3214             portIndex = parseInt(portIdentifier.split('-')[1], 10);
3215         }
3216     }
3217
3218     if (Number.isNaN(portIndex) || portIndex === null || portIndex === undefined) {
3219         portIndex = 0;
3220     }
3221
3222     // Динамические входы для AND/OR берут тип из конфига
3223     if ((element.type === 'and' || element.type === 'or')) {
3224         return SIGNAL_TYPE.LOGIC; // Логические элементы всегда ожидают LOGIC на
3225         // входе
3226     }
3227
3228     if (element.type === 'formula') {
3229         return SIGNAL_TYPE.ANY;
3230     }
3231
3232     const types = config.inputTypes || [];
```

```
3232     if (types.length === 0) return SIGNAL_TYPE.ANY;
3233
3234     if (portIndex < types.length) {
3235         return types[portIndex] || SIGNAL_TYPE.ANY;
3236     }
3237
3238     return types[types.length - 1] || SIGNAL_TYPE.ANY;
3239 }
3240
3241 function getOutputPortType(elementId, portIdentifier) {
3242     const element = AppState.elements[elementId];
3243     if (!element) return SIGNAL_TYPE.ANY;
3244
3245     const config = ELEMENT_TYPES[element.type];
3246     if (!config) return SIGNAL_TYPE.ANY;
3247
3248     let portIndex = portIdentifier;
3249
3250     if (typeof portIdentifier === 'string') {
3251         if (portIdentifier.startsWith('out-')) {
3252             portIndex = parseInt(portIdentifier.split('-')[1], 10);
3253         }
3254     }
3255
3256     if (Number.isNaN(portIndex) || portIndex === null || portIndex === undefined) {
3257         portIndex = 0;
3258     }
3259
3260     const types = config.outputTypes || [];
3261     if (types.length === 0) return SIGNAL_TYPE.ANY;
3262
3263     if (portIndex < types.length) {
3264         return types[portIndex] || SIGNAL_TYPE.ANY;
3265     }
3266
3267     return types[types.length - 1] || SIGNAL_TYPE.ANY;
3268 }
3269 /**
3270 * Проверка совместимости типов сигналов
3271 *
3272 * Новая логика:
3273 * - ANY совместим со всем
3274 * - TRUE совместим с LOGIC, TRUE, ANY
3275 * - FALSE совместим с LOGIC, FALSE, ANY
3276 * - LOGIC совместим с LOGIC, TRUE, FALSE, ANY
3277 * - NUMERIC совместим с NUMERIC, ANY
3278 */
3279 function areTypesCompatible(outputType, inputType) {
3280     // Если один из типов ANY - совместимы
3281     if (outputType === SIGNAL_TYPE.ANY || inputType === SIGNAL_TYPE.ANY) {
3282         return true;
3283     }
3284
3285     // Если типы одинаковые - совместимы
3286     if (outputType === inputType) {
3287         return true;
3288     }
3289
3290     // TRUE/FALSE совместимы с LOGIC
3291     if ((outputType === SIGNAL_TYPE.TRUE || outputType === SIGNAL_TYPE.FALSE) &&
3292         inputType === SIGNAL_TYPE.LOGIC) {
3293         return true;
3294     }
3295
3296     // LOGIC совместим с TRUE/FALSE (в случае если ожидается конкретный тип)
```

```
3297     if (outputType === SIGNAL_TYPE.LOGIC &&
3298         (inputType === SIGNAL_TYPE.TRUE || inputType === SIGNAL_TYPE.FALSE)) {
3299         return true;
3300     }
3301     return false;
3302 }
3303 */
3304 /**
3305 * Проверка, находится ли элемент внутри рамки
3306 */
3307 function isInsideFrame(elemId, frameId) {
3308     const elem = AppState.elements[elemId];
3309     const frame = AppState.elements[frameId];
3310
3311     if (!elem || !frame || frame.type !== 'output-frame') return false;
3312
3313     const elemCenterX = elem.x + elem.width / 2;
3314     const elemCenterY = elem.y + elem.height / 2;
3315
3316     return elemCenterX > frame.x &&
3317         elemCenterX < frame.x + frame.width &&
3318         elemCenterY > frame.y &&
3319         elemCenterY < frame.y + frame.height;
3320 }
3321 */
3322 /**
3323 * Обновить принадлежность элементов к рамкам
3324 */
3325 */
3326 function updateFrameChildren() {
3327     // Сначала очистим children у рамок и parentFrame у всех элементов
3328     Object.values(AppState.elements).forEach(elem => {
3329         if (elem.type === 'output-frame') {
3330             elem.children = [];
3331         } else {
3332             // удаляем parentFrame по умолчанию (пересчитаем ниже)
3333             if (elem.parentFrame) delete elem.parentFrame;
3334         }
3335     });
3336
3337     // Назначаем принадлежность: для каждого элемента ищем рамку, в которую он
3338     // попадает
3339     Object.values(AppState.elements).forEach(elem => {
3340         if (!elem || elem.type === 'output-frame') return;
3341
3342         Object.values(AppState.elements).forEach(frame => {
3343             if (!frame || frame.type !== 'output-frame') return;
3344
3345             if (isInsideFrame(elem.id, frame.id)) {
3346                 // добавляем в массив детей рамки
3347                 frame.children.push(elem.id);
3348                 // отмечаем у элемента родительскую рамку
3349                 if (AppState.elements[elem.id]) {
3350                     AppState.elements[elem.id].parentFrame = frame.id;
3351                 }
3352             }
3353         });
3354     }
3355
3356 /**
3357 * Преобразование координат экрана в координаты холста
3358 */
3359 function screenToCanvas(screenX, screenY) {
3360     const container = document.getElementById('workspace-container');
```

```
3361     const rect = container.getBoundingClientRect();
3362
3363     const x = (screenX - rect.left - AppState.viewport.panX) / AppState.viewport.zoom;
3364     const y = (screenY - rect.top - AppState.viewport.panY) / AppState.viewport.zoom;
3365
3366     return { x, y };
3367 }
3368
3369 /**
3370 * Преобразование координат холста в координаты экрана
3371 */
3372 function canvasToScreen(canvasX, canvasY) {
3373     const container = document.getElementById('workspace-container');
3374     const rect = container.getBoundingClientRect();
3375
3376     const x = canvasX * AppState.viewport.zoom + AppState.viewport.panX + rect.left;
3377     const y = canvasY * AppState.viewport.zoom + AppState.viewport.panY + rect.top;
3378
3379     return { x, y };
3380 }
3381
3382 /**
3383 * Проверка, является ли порт выходным (не подключен к другим элементам)
3384 */
3385 function isOutputPort(elemId, portIndex) {
3386     const portKey = `out-${portIndex}`;
3387
3388     // Проверяем, есть ли соединения от этого порта
3389     const hasConnection = AppState.connections.some(conn =>
3390         conn.fromElement === elemId && conn.fromPort === portKey
3391     );
3392
3393     return !hasConnection;
3394 }
3395
3396 /**
3397 * Получить информацию о выходном порте
3398 */
3399 function getOutputPortInfo(elemId, portIndex) {
3400     const elem = AppState.elements[elemId];
3401     if (!elem) return null;
3402
3403     const config = ELEMENT_TYPES[elem.type];
3404     if (!config) return null;
3405
3406     return {
3407         elementId: elemId,
3408         elementType: elem.type,
3409         elementName: config.name,
3410         portIndex: portIndex,
3411         portLabel: config.outputLabels?.[portIndex] || `out${portIndex}`,
3412         portType: config.outputTypes?.[portIndex] || SIGNAL_TYPE.ANY,
3413         // Дополнительная информация для идентификации
3414         displayName: `${config.name} → ${config.outputLabels?.[portIndex]} || `out${
3415         {portIndex}`}`
3416     };
3417 }
3418
3419 viewport.js
3420 /**
3421 * Модуль управления viewport (масштабирование и перемещение)
3422 */
3423
3424 const Viewport = {
```

```
3425     /**
3426      * Инициализация viewport
3427      */
3428     init() {
3429         this.setupZoomControls();
3430         this.setupPanning();
3431         this.setupMouseWheel();
3432         this.setupMinimap();
3433         this.setupCursorPosition();
3434         this.updateTransform();
3435     },
3436
3437     /**
3438      * Настройка кнопок масштабирования
3439      */
3440     setupZoomControls() {
3441         document.getElementById('btn-zoom-in').addEventListener('click', () => {
3442             this.setZoom(AppState.viewport.zoom + VIEWPORT_CONFIG.zoomStep);
3443         });
3444
3445         document.getElementById('btn-zoom-out').addEventListener('click', () => {
3446             this.setZoom(AppState.viewport.zoom - VIEWPORT_CONFIG.zoomStep);
3447         });
3448
3449         document.getElementById('btn-zoom-reset').addEventListener('click', () => {
3450             this.setZoom(1);
3451             this.setPan(0, 0);
3452         });
3453
3454         document.getElementById('btn-zoom-fit').addEventListener('click', () => {
3455             this.fitToContent();
3456         });
3457     },
3458
3459     /**
3460      * Настройка перемещения (пан)
3461      */
3462     setupPanning() {
3463         const container = document.getElementById('workspace-container');
3464
3465         container.addEventListener('mousedown', (e) => {
3466             // Средняя кнопка мыши или пробел + левая кнопка
3467             if (e.button === 1 || (e.button === 0 && e.target === container)) {
3468                 e.preventDefault();
3469                 AppState.viewport.isPanning = true;
3470                 AppState.viewport.lastMouseX = e.clientX;
3471                 AppState.viewport.lastMouseY = e.clientY;
3472                 container.style.cursor = 'grabbing';
3473             }
3474         });
3475
3476         document.addEventListener('mousemove', (e) => {
3477             if (AppState.viewport.isPanning) {
3478                 const dx = e.clientX - AppState.viewport.lastMouseX;
3479                 const dy = e.clientY - AppState.viewport.lastMouseY;
3480
3481                 this.setPan(
3482                     AppState.viewport.panX + dx,
3483                     AppState.viewport.panY + dy
3484                 );
3485
3486                 AppState.viewport.lastMouseX = e.clientX;
3487                 AppState.viewport.lastMouseY = e.clientY;
3488             }
3489         });
3490     };
3491 }
```

```
3490
3491     document.addEventListener('mouseup', (e) => {
3492         if (AppState.viewport.isPanning) {
3493             AppState.viewport.isPanning = false;
3494             document.getElementById('workspace-container').style.cursor = '';
3495         }
3496     });
3497
3498     // Клавиша пробел для режима перемещения
3499     document.addEventListener('keydown', (e) => {
3500         if (e.code === 'Space' && !e.repeat) {
3501             document.getElementById('workspace-container').style.cursor = 'grab';
3502         }
3503     });
3504
3505     document.addEventListener('keyup', (e) => {
3506         if (e.code === 'Space') {
3507             document.getElementById('workspace-container').style.cursor = '';
3508         }
3509     });
3510 },
3511
3512 /**
3513 * Настройка масштабирования колесом мыши
3514 */
3515 setupMouseWheel() {
3516     const container = document.getElementById('workspace-container');
3517
3518     container.addEventListener('wheel', (e) => {
3519         e.preventDefault();
3520
3521         const rect = container.getBoundingClientRect();
3522         const mouseX = e.clientX - rect.left;
3523         const mouseY = e.clientY - rect.top;
3524
3525         // Позиция мыши на холсте до масштабирования
3526         const canvasPosBeforeX = (mouseX - AppState.viewport.panX) /
AppState.viewport.zoom;
3527         const canvasPosBeforeY = (mouseY - AppState.viewport.panY) /
AppState.viewport.zoom;
3528
3529         // Новый масштаб
3530         const delta = e.deltaY > 0 ? -VIEWPORT_CONFIG.zoomStep :
VIEWPORT_CONFIG.zoomStep;
3531         const newZoom = Math.max(
3532             VIEWPORT_CONFIG.minZoom,
3533             Math.min(VIEWPORT_CONFIG.maxZoom, AppState.viewport.zoom + delta)
3534         );
3535
3536         // Корректируем pan, чтобы точка под курсором осталась на месте
3537         const newPanX = mouseX - canvasPosBeforeX * newZoom;
3538         const newPanY = mouseY - canvasPosBeforeY * newZoom;
3539
3540         AppState.viewport.zoom = newZoom;
3541         AppState.viewport.panX = newPanX;
3542         AppState.viewport.panY = newPanY;
3543
3544         this.updateTransform();
3545     }, { passive: false });
3546 },
3547
3548 /**
3549 * Установить масштаб
3550 */
3551 setZoom(zoom) {
```

```
3552     const container = document.getElementById('workspace-container');
3553     const rect = container.getBoundingClientRect();
3554
3555     // Центр экрана
3556     const centerX = rect.width / 2;
3557     const centerY = rect.height / 2;
3558
3559     // Позиция центра на холсте
3560     const canvasCenterX = (centerX - AppState.viewport.panX) /
3561     AppState.viewport.zoom;
3561     const canvasCenterY = (centerY - AppState.viewport.panY) /
3562     AppState.viewport.zoom;
3563
3564     // Новый масштаб
3565     const newZoom = Math.max(
3566         VIEWPORT_CONFIG.minZoom,
3567         Math.min(VIEWPORT_CONFIG.maxZoom, zoom)
3568     );
3569
3570     // Корректируем pan
3571     AppState.viewport.panX = centerX - canvasCenterX * newZoom;
3572     AppState.viewport.panY = centerY - canvasCenterY * newZoom;
3573     AppState.viewport.zoom = newZoom;
3574
3575     this.updateTransform();
3576 },
3577 /**
3578 * Установить смещение
3579 */
3580 setPan(x, y) {
3581     AppState.viewport.panX = x;
3582     AppState.viewport.panY = y;
3583     this.updateTransform();
3584 },
3585 /**
3586 * Вписать содержимое в экран
3587 */
3588 fitToContent() {
3589     const elements = Object.values(AppState.elements);
3590     if (elements.length === 0) {
3591         this.setZoom(1);
3592         this.setPan(0, 0);
3593         return;
3594     }
3595
3596     // Находим границы содержимого
3597     let minX = Infinity, minY = Infinity;
3598     let maxX = -Infinity, maxY = -Infinity;
3599
3600     elements.forEach(elem => {
3601         minX = Math.min(minX, elem.x);
3602         minY = Math.min(minY, elem.y);
3603         maxX = Math.max(maxX, elem.x + elem.width);
3604         maxY = Math.max(maxY, elem.y + elem.height);
3605     });
3606
3607     const contentWidth = maxX - minX;
3608     const contentHeight = maxY - minY;
3609
3610     const container = document.getElementById('workspace-container');
3611     const rect = container.getBoundingClientRect();
3612
3613     const padding = 50;
```

```
3615     const availableWidth = rect.width - padding * 2;
3616     const availableHeight = rect.height - padding * 2;
3617
3618     const zoomX = availableWidth / contentWidth;
3619     const zoomY = availableHeight / contentHeight;
3620     const newZoom = Math.min(zoomX, zoomY, 1);
3621
3622     AppState.viewport.zoom = Math.max(VIEWPORT_CONFIG.minZoom, newZoom);
3623     AppState.viewport.panX = padding - minX * AppState.viewport.zoom +
3624     (availableWidth - contentWidth * AppState.viewport.zoom) / 2;
3624     AppState.viewport.panY = padding - minY * AppState.viewport.zoom +
3625     (availableHeight - contentHeight * AppState.viewport.zoom) / 2;
3625
3626     this.updateTransform();
3627 },
3628
3629 /**
3630 * Обновить трансформацию
3631 */
3632 updateTransform() {
3633     const workspace = document.getElementById('workspace');
3634     const svg = document.getElementById('connections-svg');
3635
3636     const transform = `translate(${AppState.viewport.panX}px, ${
3637     AppState.viewport.panY}px) scale(${AppState.viewport.zoom})`;
3638
3639     workspace.style.transform = transform;
3640     svg.style.transform = transform;
3641
3642     // Обновляем отображение масштаба
3643     document.getElementById('zoom-level').textContent = `${Math.round(AppState.viewport.zoom * 100)}%`;
3644
3645     // Обновляем мини-карту
3646     this.updateMinimap();
3647 }
3648
3649 /**
3650 * Настройка мини-карты
3651 */
3652 setupMinimap() {
3653     const minimap = document.getElementById('minimap');
3654     const canvas = document.getElementById('minimap-canvas');
3655
3656     canvas.width = MINIMAP_CONFIG.width;
3657     canvas.height = MINIMAP_CONFIG.height;
3658
3659     // Клик по мини-карте для перемещения
3660     minimap.addEventListener('click', (e) => {
3661         const rect = minimap.getBoundingClientRect();
3662         const x = e.clientX - rect.left;
3663         const y = e.clientY - rect.top;
3664
3665         this.navigateToMinimapPosition(x, y);
3666     });
3667 }
3668
3669 /**
3670 * Обновить мини-карту
3671 */
3672 updateMinimap() {
3673     const canvas = document.getElementById('minimap-canvas');
3674     const ctx = canvas.getContext('2d');
3675     const viewportEl = document.getElementById('minimap-viewport');
```

```
3676 // Очищаем
3677 ctx.fillStyle = '#0a0a1a';
3678 ctx.fillRect(0, 0, canvas.width, canvas.height);
3679
3680 // Масштаб мини-карты
3681 const scale = Math.min(
3682     canvas.width / VIEWPORT_CONFIG.canvasWidth,
3683     canvas.height / VIEWPORT_CONFIG.canvasHeight
3684 );
3685
3686 // Рисуем элементы
3687 Object.values(AppState.elements).forEach(elem => {
3688     const x = elem.x * scale;
3689     const y = elem.y * scale;
3690     const w = Math.max(elem.width * scale, 2);
3691     const h = Math.max(elem.height * scale, 2);
3692
3693     ctx.fillStyle = ELEMENT_TYPES[elem.type]?.color || '#4a90d9';
3694     ctx.fillRect(x, y, w, h);
3695 });
3696
3697 // Рисуем viewport
3698 const container = document.getElementById('workspace-container');
3699 const rect = container.getBoundingClientRect();
3700
3701 const vpX = (-AppState.viewport.panX / AppState.viewport.zoom) * scale;
3702 const vpY = (-AppState.viewport.panY / AppState.viewport.zoom) * scale;
3703 const vpW = (rect.width / AppState.viewport.zoom) * scale;
3704 const vpH = (rect.height / AppState.viewport.zoom) * scale;
3705
3706 viewportEl.style.left = `${vpX}px`;
3707 viewportEl.style.top = `${vpY}px`;
3708 viewportEl.style.width = `${vpW}px`;
3709 viewportEl.style.height = `${vpH}px`;
3710 },
3711 /**
3712 * Перейти к позиции на мини-карте
3713 */
3714 navigateToMinimapPosition(minimapX, minimapY) {
3715     const canvas = document.getElementById('minimap-canvas');
3716     const container = document.getElementById('workspace-container');
3717     const rect = container.getBoundingClientRect();
3718
3719     const scale = Math.min(
3720         canvas.width / VIEWPORT_CONFIG.canvasWidth,
3721         canvas.height / VIEWPORT_CONFIG.canvasHeight
3722     );
3723
3724     const canvasX = minimapX / scale;
3725     const canvasY = minimapY / scale;
3726
3727     // Центрируем viewport на этой точке
3728     AppState.viewport.panX = rect.width / 2 - canvasX * AppState.viewport.zoom;
3729     AppState.viewport.panY = rect.height / 2 - canvasY * AppState.viewport.zoom;
3730
3731     this.updateTransform();
3732 },
3733 /**
3734 * Отслеживание позиции курсора
3735 */
3736 setupCursorPosition() {
3737     const container = document.getElementById('workspace-container');
3738 }
```

```
3741     container.addEventListener('mousemove', (e) => {
3742         const pos = screenToCanvas(e.clientX, e.clientY);
3743         document.getElementById('cursor-pos').textContent =
3744             `X: ${Math.round(pos.x)}, Y: ${Math.round(pos.y)}`;
3745     });
3746 }
3747 };
3748
3749 styles.css
3750
3751 * {
3752     margin: 0;
3753     padding: 0;
3754     box-sizing: border-box;
3755 }
3756
3757 body {
3758     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
3759     background: #1a1a2e;
3760     color: #eee;
3761     overflow: hidden;
3762 }
3763
3764 #app {
3765     display: flex;
3766     flex-direction: column;
3767     height: 100vh;
3768 }
3769
3770 /* ===== MEHIO ===== */
3771 #menu {
3772     background: #16213e;
3773     padding: 10px 20px;
3774     display: flex;
3775     gap: 10px;
3776     align-items: center;
3777     border-bottom: 2px solid #0f3460;
3778     z-index: 100;
3779     flex-wrap: wrap;
3780 }
3781
3782 .menu-btn {
3783     background: #0f3460;
3784     color: #eee;
3785     border: none;
3786     padding: 8px 16px;
3787     border-radius: 5px;
3788     cursor: pointer;
3789     transition: background 0.3s;
3790     font-size: 13px;
3791 }
3792
3793 .menu-btn:hover {
3794     background: #e94560;
3795 }
3796
3797 .menu-separator {
3798     width: 1px;
3799     height: 30px;
3800     background: #0f3460;
3801     margin: 0 10px;
3802 }
3803
3804 .zoom-controls {
3805     display: flex;
```

```
3806     align-items: center;
3807     gap: 8px;
3808     background: #0a0a1a;
3809     padding: 5px 10px;
3810     border-radius: 5px;
3811   }
3812 
3813   .zoom-btn {
3814     width: 30px;
3815     height: 30px;
3816     padding: 0;
3817     font-size: 18px;
3818     font-weight: bold;
3819   }
3820 
3821   #zoom-level {
3822     min-width: 50px;
3823     text-align: center;
3824     font-size: 12px;
3825     color: #aaa;
3826   }
3827 
3828   /* ===== ОСНОВНАЯ ОБЛАСТЬ ===== */
3829   #main {
3830     display: flex;
3831     flex: 1;
3832     overflow: hidden;
3833   }
3834 
3835   /* ===== ПАЛИТРА ===== */
3836   #palette {
3837     width: 200px;
3838     background: #16213e;
3839     padding: 15px;
3840     border-right: 2px solid #0f3460;
3841     overflow-y: auto;
3842     z-index: 10;
3843     flex-shrink: 0;
3844   }
3845 
3846   #palette h3 {
3847     margin-bottom: 15px;
3848     color: #e94560;
3849     text-align: center;
3850     font-size: 14px;
3851   }
3852 
3853   .palette-section {
3854     margin-bottom: 15px;
3855   }
3856 
3857   .palette-section-title {
3858     font-size: 11px;
3859     color: #888;
3860     margin-bottom: 8px;
3861     padding-bottom: 3px;
3862     border-bottom: 1px solid #333;
3863   }
3864 
3865   .palette-item {
3866     background: #0f3460;
3867     padding: 8px;
3868     margin-bottom: 6px;
3869     border-radius: 8px;
3870     cursor: grab;
```

```
3871     text-align: center;
3872     transition: all 0.3s;
3873     border: 2px solid transparent;
3874     user-select: none;
3875   }
3876
3877   .palette-item:hover {
3878     border-color: #e94560;
3879     transform: scale(1.02);
3880   }
3881
3882   .palette-item:active {
3883     cursor: grabbing;
3884   }
3885
3886   .palette-item svg {
3887     width: 50px;
3888     height: 32px;
3889     margin-bottom: 2px;
3890     pointer-events: none;
3891   }
3892
3893   .palette-item-name {
3894     font-size: 10px;
3895     color: #aaa;
3896     pointer-events: none;
3897   }
3898
3899   .type-legend {
3900     margin-top: 15px;
3901     padding-top: 10px;
3902     border-top: 1px solid #333;
3903     font-size: 10px;
3904   }
3905
3906   .type-legend-item {
3907     display: flex;
3908     align-items: center;
3909     gap: 8px;
3910     margin-bottom: 5px;
3911   }
3912
3913   .type-legend-dot {
3914     width: 12px;
3915     height: 12px;
3916     border-radius: 50%;
3917     border: 2px solid #fff;
3918   }
3919   .type-legend-dot.logic { background: #a855f7; }
3920   .type-legend-dot.number { background: #3b82f6; }
3921
3922 /* ===== РАБОЧАЯ ОБЛАСТЬ ===== */
3923 #workspace-container {
3924   flex: 1;
3925   position: relative;
3926   overflow: hidden;
3927   background-color: #0a0a1a;
3928   background-image:
3929     linear-gradient(rgba(255,255,255,0.04) 1px, transparent 1px),
3930     linear-gradient(90deg, rgba(255,255,255,0.04) 1px, transparent 1px);
3931   background-size: 25px 25px;
3932 }
3933
3934 #workspace {
3935   position: absolute;
```

```
3936     transform-origin: 0 0;
3937     width: 5000px;
3938     height: 5000px;
3939 }
3940
3941 #connections-svg {
3942     position: absolute;
3943     transform-origin: 0 0;
3944     pointer-events: none;
3945     z-index: 5;
3946     width: 5000px;
3947     height: 5000px;
3948 }
3949
3950 #connections-svg path {
3951     pointer-events: stroke;
3952 }
3953
3954 /* ===== ЭЛЕМЕНТЫ ===== */
3955 .element {
3956     position: absolute;
3957     background: #0f3460;
3958     border: 2px solid #4a90d9;
3959     border-radius: 8px;
3960     cursor: move;
3961     user-select: none;
3962     z-index: 10;
3963     display: flex;
3964     flex-direction: column;
3965 }
3966
3967 .element.selected {
3968     border-color: #e94560;
3969     box-shadow: 0 0 15px rgba(233, 69, 96, 0.5);
3970 }
3971
3972 .element-header {
3973     background: #4a90d9;
3974     padding: 5px 10px;
3975     border-radius: 5px 5px 0 0;
3976     font-size: 11px;
3977     font-weight: bold;
3978     text-align: center;
3979     white-space: nowrap;
3980     overflow: hidden;
3981     text-overflow: ellipsis;
3982 }
3983
3984 .element-body {
3985     padding: 10px;
3986     display: flex;
3987     justify-content: space-between;
3988     align-items: center;
3989     flex: 1;
3990     gap: 8px;
3991 }
3992
3993 .element-symbol {
3994     font-size: 16px;
3995     font-weight: bold;
3996     flex: 1;
3997     text-align: center;
3998     padding: 0 5px;
3999     word-break: break-all;
4000     color: #eee;
```

```
4001 }
4002
4003 /* ===== ПОРТЫ ===== */
4004 .ports-left, .ports-right {
4005     display: flex;
4006     flex-direction: column;
4007     justify-content: space-around;
4008     gap: 10px;
4009     height: 100%;
4010 }
4011
4012 .port {
4013     width: 14px;
4014     height: 14px;
4015     border-radius: 50%;
4016     border: 2px solid #ffff;
4017     cursor: crosshair;
4018     transition: all 0.2s;
4019     position: relative;
4020     flex-shrink: 0;
4021 }
4022
4023 .port:hover { transform: scale(1.3); }
4024 .port.input { margin-left: -8px; }
4025 .port.output { margin-right: -8px; }
4026 .port.connected { background: #4ade80; }
4027
4028 /* Типы портов */
4029 .port.logic-port { background: #a855f7; border-color: #e9d5ff; }
4030 .port.logic-port:hover { background: #c084fc; }
4031 .port.logic-port.connected { background: #7c3aed; }
4032
4033 .port.number-port { background: #3b82f6; border-color: #bfdbfe; }
4034 .port.number-port:hover { background: #60a5fa; }
4035 .port.number-port.connected { background: #2563eb; }
4036
4037 .port.any-port { background: #6b7280; border-color: #d1d5db; }
4038 .port.any-port:hover { background: #9ca3af; }
4039 .port.any-port.connected { background: #4b5563; }
4040
4041 .port.output.yes-port { background: #4ade80 !important; border-color: #bbf7d0 !important; }
4042 .port.output.no-port { background: #f87171 !important; border-color: #fecaca !important; }
4043
4044 .port.incompatible { opacity: 0.3; cursor: not-allowed; }
4045 .port.compatible-highlight { box-shadow: 0 0 10px 3px #4ade80; }
4046
4047 /* ===== RESIZE HANDLES ===== */
4048 .resize-handle {
4049     position: absolute;
4050     width: 12px;
4051     height: 12px;
4052     background: #e94560;
4053     border: 1px solid #ffff;
4054     border-radius: 3px;
4055     z-index: 20;
4056     opacity: 0;
4057     transition: opacity 0.2s;
4058 }
4059 .element.selected .resize-handle { opacity: 0.8; }
4060 .resize-handle:hover { opacity: 1; }
4061 .resize-handle.handle-se { bottom: -6px; right: -6px; cursor: se-resize; }
4062 .resize-handle.handle-e { top: 50%; right: -6px; transform: translateY(-50%); cursor: ew-resize; }
```

```
4063 .resize-handle.handle-s { bottom: -6px; left: 50%; transform: translateX(-50%);  
4064 cursor: ns-resize; }  
4065  
4066 /* ===== ВХОДНОЙ СИГНАЛ (ТРАПЕЦИЯ) ===== */  
4067 .element.input-signal {  
4068     background: transparent;  
4069     border: none;  
4070 }  
4071  
4072 .element.input-signal .element-header {  
4073     display: none; /* У трапеции нет заголовка */  
4074 }  
4075  
4076 .element.input-signal .element-body {  
4077     padding: 0;  
4078     background: #0f3460;  
4079     border: 2px solid #4a90d9;  
4080     clip-path: polygon(0 0, 80% 0, 100% 50%, 80% 100%, 0 100%);  
4081     display: flex;  
4082     justify-content: space-between;  
4083     align-items: center;  
4084     padding-left: 15px;  
4085     padding-right: 25px;  
4086 }  
4087  
4088 .element.input-signal .element-symbol {  
4089     text-align: left;  
4090     color: #eee;  
4091 }  
4092  
4093 .element.input-signal.selected .element-body {  
4094     border-color: #e94560;  
4095 }  
4096  
4097 /* ===== ЭЛЕМЕНТ ВЫХОДА (ПУНКТИР) ===== */  
4098 .element.output {  
4099     background: rgba(16, 185, 129, 0.1);  
4100     border: 2px dashed #10b981;  
4101 }  
4102  
4103 .element.output .element-header {  
4104     display: none; /* У выхода нет заголовка */  
4105 }  
4106  
4107 .element.output .element-body {  
4108     padding-left: 20px;  
4109 }  
4110  
4111 .element.output .element-symbol {  
4112     color: #10b981;  
4113     font-size: 14px;  
4114 }  
4115  
4116 .element.output.selected {  
4117     border-color: #e94560;  
4118     border-style: dashed;  
4119 }  
4120  
4121  
4122 /* Formula condition port */  
4123 /* Универсальный стиль для технического порта (сверху) */  
4124 .element.has-condition-port {  
4125     margin-top: 30px; /* Даем место порту над элементом */  
4126 }
```

```
4127
4128 .condition-port-wrapper {
4129     position: absolute;
4130     top: -28px;
4131     left: 50%;
4132     transform: translateX(-50%);
4133     display: flex;
4134     flex-direction: column;
4135     align-items: center;
4136     gap: 4px;
4137     pointer-events: none;
4138     z-index: 21;
4139 }
4140
4141 .condition-port-label {
4142     font-size: 10px;
4143     color: #f59e0b;
4144     font-weight: 600;
4145     white-space: nowrap;
4146 }
4147
4148 .port.condition-port {
4149     pointer-events: auto;
4150     width: 16px;
4151     height: 16px;
4152     border-radius: 50%;
4153     border: 2px solid #f59e0b;
4154     background: #fff7ed;
4155     margin: 0; /* Сбрасываем лишние отступы */
4156 }
4157 .element.formula .condition-port:hover { background: #fde68a; }
4158
4159
4160 /* ===== СОЕДИНЕНИЯ ===== */
4161 .connection {
4162     fill: none !important; /* ← добавляем !important */
4163     stroke: #4a90d9;
4164     stroke-width: 2.5;
4165 }
4166 .connection:hover {
4167     stroke: #e94560;
4168     stroke-width: 4;
4169 }
4170
4171 .connection.logic-conn { stroke: #a855f7; }
4172 .connection.numeric-conn { stroke: #3b82f6; }
4173 .connection.any-conn { stroke: #6b7280; }
4174 .connection.true-conn { stroke: #4ade80; }
4175 .connection.false-conn { stroke: #f87171; }
4176
4177 .connection.yes-conn { stroke: #4ade80; }
4178 .connection.no-conn { stroke: #f87171; }
4179
4180 .temp-connection {
4181     fill: none !important; /* ← добавляем !important */
4182     stroke: #e94560;
4183     stroke-width: 2;
4184     stroke-dasharray: 5, 5;
4185 }
4186 .temp-connection.invalid { stroke: #ef4444; }
4187
4188 /* ===== ПРОЧЕЕ ===== */
4189 .drag-preview {
4190     position: fixed;
4191     pointer-events: none;
```

```
4192     opacity: 0.8;
4193     z-index: 1000;
4194     background: #0f3460;
4195     border: 2px solid #e94560;
4196     border-radius: 8px;
4197     padding: 10px 15px;
4198     color: #fff;
4199     font-size: 12px;
4200   }
4201
4202   #minimap {
4203     position: absolute;
4204     bottom: 20px;
4205     right: 20px;
4206     width: 200px;
4207     height: 150px;
4208     background: #16213e;
4209     border: 2px solid #0f3460;
4210     border-radius: 8px;
4211     overflow: hidden;
4212     z-index: 50;
4213   }
4214
4215   #minimap-canvas { width: 100%; height: 100%; }
4216   #minimap-viewport {
4217     position: absolute;
4218     border: 2px solid #e94560;
4219     background: rgba(233, 69, 96, 0.2);
4220     pointer-events: none;
4221   }
4222
4223   #viewport-info {
4224     position: absolute;
4225     bottom: 20px;
4226     left: 20px;
4227     background: rgba(22, 33, 62, 0.9);
4228     padding: 8px 12px;
4229     border-radius: 5px;
4230     font-size: 11px;
4231     color: #888;
4232     z-index: 50;
4233     display: flex;
4234     gap: 15px;
4235   }
4236   #selection-info { color: #e94560; }
4237
4238   #modal-overlay, .modal-overlay-class {
4239     display: none;
4240     position: fixed;
4241     top: 0; left: 0;
4242     width: 100%; height: 100%;
4243     background: rgba(0, 0, 0, 0.7);
4244     z-index: 1000;
4245     justify-content: center;
4246     align-items: center;
4247   }
4248
4249   #modal, .modal-class {
4250     background: #16213e;
4251     border-radius: 10px;
4252     padding: 20px;
4253     min-width: 400px;
4254     max-width: 600px;
4255     max-height: 80vh;
4256     overflow-y: auto;
```

```
4257     border: 2px solid #0f3460;
4258 }
4259
4260 #modal h3, .modal-class h3 { margin-bottom: 15px; color: #e94560; }
4261 .modal-row { margin-bottom: 15px; }
4262 .modal-row label { display: block; margin-bottom: 5px; color: #aaa; font-size: 13px; }
4263 .modal-row input, .modal-row select, .modal-row textarea {
4264     width: 100%;
4265     padding: 10px;
4266     background: #0f3460;
4267     border: 1px solid #4a90d9;
4268     border-radius: 5px;
4269     color: #eee;
4270     font-size: 14px;
4271 }
4272 .modal-row input:focus, .modal-row select:focus, .modal-row textarea:focus { outline: none; border-color: #e94560; }
4273 .modal-row textarea { min-height: 80px; font-family: inherit; resize: vertical; }
4274 .signal-list { max-height: 100px; overflow-y: auto; background: #0f3460; border-radius: 5px; padding: 5px; margin-top: 5px; }
4275 .signal-item { padding: 5px 10px; cursor: pointer; border-radius: 3px; font-size: 12px; }
4276 .signal-item:hover { background: #4a90d9; }
4277 .modal-buttons { display: flex; gap: 10px; justify-content: flex-end; margin-top: 20px; }
4278 .modal-btn { padding: 10px 25px; border: none; border-radius: 5px; cursor: pointer; font-size: 14px; transition: background 0.3s; }
4279 .modal-btn.save { background: #4ade80; color: #000; }
4280 .modal-btn.save:hover { background: #22c55e; }
4281 .modal-btn.cancel { background: #6b7280; color: #fff; }
4282 .modal-btn.cancel:hover { background: #4b5563; }
4283
4284 #context-menu {
4285     display: none;
4286     position: fixed;
4287     background: #16213e;
4288     border: 1px solid #0f3460;
4289     border-radius: 5px;
4290     padding: 5px 0;
4291     z-index: 1001;
4292     min-width: 150px;
4293     box-shadow: 0 5px 20px rgba(0,0,0,0.3);
4294 }
4295 .context-item { padding: 10px 15px; cursor: pointer; font-size: 13px; transition: background 0.2s; }
4296 .context-item:hover { background: #0f3460; }
4297
4298 #file-input { display: none; }
4299
4300 .project-type-selector { display: flex; gap: 10px; margin-bottom: 15px; }
4301 .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; }
4302 .project-type-btn:hover { border-color: #e94560; }
4303 .project-type-btn.active { background: #4a90d9; border-color: #4a90d9; }
4304 .project-type-btn .type-icon { font-size: 24px; margin-bottom: 5px; }
4305 .project-type-btn .type-name { font-weight: bold; }
4306 .project-type-btn .type-desc { font-size: 11px; color: #aaa; margin-top: 3px; }
4307
4308 .conditional-fields { display: none; padding: 15px; background: #0a0a1a; border-radius: 8px; margin-top: 10px; }
4309 .conditional-fields.visible { display: block; }
4310
4311 ::-webkit-scrollbar { width: 8px; height: 8px; }
4312 ::-webkit-scrollbar-track { background: #0a0a1a; }
```

```
4313 ::-webkit-scrollbar-thumb { background: #4a90d9; border-radius: 4px; }
4314 ::-webkit-scrollbar-thumb:hover { background: #e94560; }
4315
4316 /* Стили для выходов */
4317 .output-btn { position: relative; }
4318 .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; }
4319 .output-counter:empty, .output-counter[style*="display: none"] { display: none; }
4320 .element.has-output { box-shadow: 0 0 10px rgba(16, 185, 129, 0.3); }
4321 .element.output-highlighted { box-shadow: 0 0 20px rgba(251, 191, 36, 0.6) !important; border-color: #fbbf24 !important; }
4322 .port.output-active { box-shadow: 0 0 8px 2px rgba(16, 185, 129, 0.8); animation: pulse-output 1.5s infinite; }
4323 @keyframes pulse-output {
4324     0%, 100% { box-shadow: 0 0 8px 2px rgba(16, 185, 129, 0.8); }
4325     50% { box-shadow: 0 0 12px 4px rgba(16, 185, 129, 1); }
4326 }
4327
4328 .outputs-container { background: #0a0ala; border-radius: 8px; padding: 15px; max-height: 250px; overflow-y: auto; }
4329 .outputs-section { margin-bottom: 15px; }
4330 .outputs-section:last-child { margin-bottom: 0; }
4331 .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; }
4332 .outputs-section-title .section-icon { font-size: 16px; }
4333 .outputs-list { display: flex; flex-direction: column; gap: 5px; }
4334 .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; }
4335 .output-item:hover { background: rgba(16, 185, 129, 0.2); border-color: #10b981; transform: translateX(5px); }
4336 .output-item.numeric { background: rgba(59, 130, 246, 0.1); border-color: rgba(59, 130, 246, 0.3); }
4337 .output-item.numeric:hover { background: rgba(59, 130, 246, 0.2); border-color: #3b82f6; }
4338 .output-icon { font-size: 14px; }
4339 .output-name { font-weight: bold; color: #eee; }
4340 .output-port { color: #888; font-size: 12px; margin-left: auto; }
4341 .no-outputs { color: #666; font-style: italic; padding: 10px; text-align: center; }
4342 .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; }
4343 .element.output-ambiguous { box-shadow: 0 0 18px 4px rgba(240, 80, 80, 0.55); border-color: rgba(240, 80, 80, 0.8) !important; }
4344 .element.output-missing { box-shadow: 0 0 14px 3px rgba(250, 200, 30, 0.5); border-color: rgba(250, 200, 30, 0.8) !important; }
4345 /* TRUE/FALSE порты (для сепаратора) */
4346 .port.true-port {
4347     background: #4ade80 !important;
4348     border-color: #bbf7d0 !important;
4349 }
4350 .port.true-port:hover {
4351     background: #22c55e !important;
4352 }
4353 .port.true-port.connected {
4354     background: #16a34a !important;
4355 }
4356
4357 .port.false-port {
4358     background: #f87171 !important;
4359     border-color: #fecaca !important;
4360 }
4361 .port.false-port:hover {
4362     background: #ef4444 !important;
```

```
4363 }
4364 .port.false-port.connected {
4365     background: #dc2626 !important;
4366 }
4367
4368 /* Сепаратор стиль */
4369 .element.separator {
4370     background: #0f3460;
4371     border: 2px solid #f59e0b;
4372 }
4373
4374 .element.separator.selected {
4375     border-color: #e94560;
4376     box-shadow: 0 0 15px rgba(233, 69, 96, 0.5);
4377 }
```