Figma Design-to-Code Conversion Issues: DivRiots Plugin Analysis

Executive Summary

This README documents the specific challenges encountered when using the **DivRiots figma.to.website plugin** (https://www.figma.com/community/plugin/1329237288766226289/figma-to-website-by-divriots) for Figma-to-HTML conversion and provides actionable solutions for creating designs that work better with both this tool and AI-powered code generation systems.

Table of Contents

The Problem with Current Figma-to-Code Tools {#the-problem}

What Went Wrong with DivRiots figma.to.website Plugin

When using the **DivRiots figma.to.website plugin**, our design conversion resulted in:

- **Thousands of lines of unreadable code**: Files ranging from 5,889 to 8,275+ lines
- **Massive inline JavaScript**: Complex animation and interaction code embedded directly in HTML (F2W_REACTIONS system)
- **Cryptic class names**: Auto-generated IDs like `I8854_1806_8802_1423` instead of semantic names
- **Unnecessary nested divs**: Over-complicated DOM structure with excessive wrapper elements
- **Bloated CSS**: Inline styles mixed with generated CSS classes creating maintenance nightmares
- **Figma-specific optimization conflicts**: The plugin optimizes for Figma's visual system rather than clean, maintainable HTML

Why This Specific Plugin Has These Issues

The DivRiots plugin is actually one of the **better** Figma-to-website tools available, with 58,300+ users and active development. However, it still faces fundamental challenges:

1. **It's optimized for visual accuracy over code quality**: The plugin prioritizes making the website look exactly like your Figma design
2. **Complex animation system**: Uses a proprietary `F2W_REACTIONS` system for handling Figma prototype interactions
3. **Performance over semantic HTML**: Generates optimized but not human-readable code structure
4. **Platform dependency**: Creates websites that work best within their hosting ecosystem

Example of Generated Code Issues

```
<!-- What we got: -->
<div id="I8854_1806_8802_1423" class="auto-generated-wrapper-div">
  <div class="Frame_106_nested_container">
    <div class="unnecessary_wrapper_element">
      <span style="color: #32becd; font-size: 14px; line-height: 1.2;">Text
content</span>
    </div>
  </div>
</div>

<!-- What we wanted: -->
<section class="hero">
  <h1>Text content</h1>
</section>
```

DivRiots Plugin: Features vs. Code Quality Trade-offs

What DivRiots Does Well

Based on their documentation and user feedback, the plugin offers:

- ☑ **Full Figma feature support**: Auto Layout, Components, Variants, Interactions, Variables
- ☑ **Performance optimization**: Optimized for Core Web Vitals, global CDN
- ☑ **Responsive design support**: Multiple frames for different breakpoints
- ☑ **SEO features**: Automatic sitemap generation, Open Graph data
- ☑ **Integration capabilities**: Forms, analytics, custom embeds
- ☑ **Export functionality**: Can export HTML/CSS/JS for hosting elsewhere

The Code Quality Trade-off

However, these features come at the cost of code maintainability:

- **Visual accuracy prioritized**: Code structure serves visual fidelity over semantic meaning
- **Proprietary systems**: Custom animation and interaction frameworks
- **Platform optimization**: Code optimized for their hosting platform, not general web standards
- **Black box conversion**: No control over the HTML structure generation process

Common Issues Users Report with DivRiots

Based on user feedback from the Figma community, common problems include:

1. **Publishing failures**: Some users report getting "white page and undefined" errors when publishing
2. **Domain setup challenges**: Difficulty with custom domain configuration
3. **Image limitations**: Issues with image uploads and optimization
4. **Export restrictions**: HTML/CSS export only available in paid plans
5. **Platform dependency**: Websites work best within DivRiots ecosystem

6. **Limited customization**: Hard to modify generated structure without breaking functionality

Note: These issues don't mean the plugin is bad—they're typical challenges with any automated design-to-code tool trying to bridge the gap between visual design and structured HTML.

Analysis of Generated Code Issues {#code-analysis}

1. Complex Animation Systems

The generated code includes massive JavaScript objects for handling interactions:

```
window.F2W_REACTIONS = (() => {
  const e = [
    [{ key: "background-color", from: "#32becd", to: "#29ad3c" }],
    [{ key: "color", from: "#fff", to: "#e3eee3" }],
    // ... thousands more lines of animation definitions
  ];
})();
```

Problem: This approach creates maintenance nightmares and performance issues.

2. Non-Semantic HTML Structure

Generated HTML lacks proper semantic structure:

- No `<header>`, `<nav>`, `<main>`, `<section>`, `<article>` elements
- Everything wrapped in generic `<div>` elements
- No accessibility considerations (missing ARIA labels, alt texts, etc.)
- Non-descriptive class names that don't indicate purpose

3. Inline Styles Everywhere

The conversion tool generates extensive inline styles:

```
<div style="display: flex; flex-direction: column; padding: 32px 16px 0px;
background-image: linear-gradient(0deg, rgba(0,0,0,0.2), rgba(0,0,0,0.2)),
url('https://long-cloudinary-url...');">
```

Problem: Makes styling changes difficult and CSS overrides nearly impossible.

Understanding Figma Design Structure Problems {#design-problems}

Real-World Example: How This Figma Should Have Been Structured

Based on the actual Figma file that caused the conversion issues, here's how it should have been organized for optimal DivRiots plugin compatibility:

❌ Current Problematic Structure (What We Had)

```
About - Desktop
├── FIXED
│   └── CTA Button
├── SCROLLS
│   ├── Rectangle 110
│   ├── Headerexpanded
│   ├── Frame 28
│   ├── Artboard 209 1
│   ├── Responsive Logo Line (duplicate)
│   ├── Responsive Logo Line (duplicate)
│   ├── Responsive Logo Line (duplicate)
│   ├── Partners Horizontal
│   ├── Frame 85
│   └── OurJourney
```

☑ How It Should Have Been Structured

```
About Page - Desktop
├── Header Section
│   ├── Navigation Bar
│   │   ├── Logo
│   │   ├── Menu Items
│   │   └── CTA Button
│   └── Hero Section
│       ├── Main Heading
│       ├── Subtitle
│       └── Hero CTA
├── Main Content
│   ├── About Section
│   │   ├── Section Title
│   │   ├── Description Text
│   │   └── Feature Grid
│   ├── Impact Section
│   │   ├── Stats Grid
│   │   └── Impact Metrics
│   ├── Partnership Section
│   │   ├── Section Header
│   │   ├── Partner Logos Grid
│   │   └── Partnership Benefits
│   └── Journey Section
│       ├── Timeline Container
│       └── Journey Steps
└── Footer Section
    ├── Contact Info
    ├── Social Links
    └── Copyright
```

Key Problems with Original Structure

1. **Generic Layer Names**: `Rectangle 110`, `Frame 28`, `Artboard 209 1`
   - **Solution**: Use semantic names like `Hero Background`, `Content Container`, `About Section`
2. **Duplicate Components**: Multiple `Responsive Logo Line` without variants
   - **Solution**: Create one `Logo Line` component with variants for different states
3. **No Semantic Grouping**: Elements not grouped by purpose or HTML section
   - **Solution**: Group by semantic meaning (`Header Section`, `Main Content`, `Footer Section`)
4. **Non-Descriptive Containers**: `FIXED`, `SCROLLS` don't indicate content
   - **Solution**: Use HTML-semantic names that indicate the actual page structure

DivRiots-Specific Best Practices

1. Use Semantic Frame Names

```
☑ Good Examples:
- Header Section
- Hero Banner
- Features Grid
- Contact Form
- Footer Navigation

❌ Avoid:
- Frame 1, Frame 2
- Rectangle 110
- Group 45
- Artboard 209 1
```

2. Create Component Variants Instead of Duplicates

```
Instead of:
├── Responsive Logo Line
├── Responsive Logo Line (copy)
├── Responsive Logo Line (copy 2)

Use:
└── Partner Logos
    ├── Desktop Variant
    ├── Tablet Variant
    └── Mobile Variant
```

3. Structure with HTML in Mind

```
Page Structure:
├── Header
│   ├── Navigation
│   └── Hero
├── Main
│   ├── Section 1
│   ├── Section 2
│   └── Section 3
└── Footer
    ├── Links
    └── Copyright
```

4. Consistent Naming Convention

```
Format: [Element Type] [Purpose] [State/Variant]

Examples:
- Button Primary Default
- Button Primary Hover
- Card Product Featured
- Section Hero Desktop
- Logo Main Light
```

Expected Code Improvement

With proper Figma structure, instead of:

```
<div id="I8854_1806_8802_1423" class="auto-generated-wrapper-div">
  <div class="Frame_106_nested_container">
    <!-- Complex nested structure -->
  </div>
</div>
```

You would get:

```
<header class="header-section">
  <nav class="navigation-bar">
    <img src="logo.png" alt="Company Logo" class="logo">
    <div class="menu-items">
      <!-- Navigation items -->
    </div>
  </nav>
</header>
```

Why Conversion Tools Fail

1. **Figma is a Visual Tool, Not a Code Structure Tool**
    - Figma frames don't translate directly to semantic HTML elements
    - Visual hierarchy ≠ DOM hierarchy
    - Design components aren't necessarily code components
2. **Lack of Design System Constraints**
    - Inconsistent naming conventions
    - No semantic meaning in layer names
    - Missing component architecture thinking
3. **Complex Visual Effects**
    - Heavy reliance on gradients, shadows, and complex backgrounds
    - Multiple overlapping elements creating unnecessary layers
    - Animations defined visually rather than programmatically

Common Figma Design Patterns That Break Conversion

❌ Problematic Patterns:

1. **Unnamed or generically named layers**

```
Frame 1
├── Rectangle 2
├── Group 3
└── Frame 4
```

2. **Complex overlapping elements**
    - Multiple backgrounds layered for visual effects
    - Text on complex image overlays
    - Unnecessary grouped elements
3. **Inconsistent component usage**
    - One-off designs instead of reusable components
    - Variations created by copying instead of using variants

☑ Better Patterns:

1. **Semantic layer naming**

```
Header
├── Logo
├── Navigation
└── CTA Button
```

2. **Component-based architecture**
    - Consistent button components
    - Reusable card patterns
    - Standardized text styles

How to Design for Better AI Code Generation {#best-practices}

1. Use Semantic Naming Conventions

Frame Names Should Reflect HTML Elements:

```
Header
Main Content
  ├── Hero Section
  ├── Features Section
  └── CTA Section
Footer
```

Layer Names Should Describe Purpose:

```
Primary Button
Secondary Button
Heading Large
Body Text
Navigation Link
```

2. Leverage Figma's Auto Layout Properly

Auto Layout is Figma's closest equivalent to CSS Flexbox/Grid:

```
☑ Use Auto Layout for:
- Navigation bars (horizontal flow)
- Card layouts (vertical flow)
- Grid systems (grid flow)
- Button groups
- Content sections

❌ Avoid:
- Manually positioned elements
- Complex overlapping layouts
- Pixel-perfect absolute positioning
```

3. Create Consistent Design Systems

Component Architecture:

- Define reusable components for all UI elements
- Use component variants instead of creating new components
- Establish consistent spacing and sizing systems
- Create semantic color and typography systems

Design Tokens:

```
Colors:
├── Primary (brand colors)
├── Secondary (accent colors)
├── Neutral (grays, whites)
└── Semantic (success, warning, error)

Typography:
├── Heading 1 (32px, bold)
├── Heading 2 (24px, semibold)
├── Body Large (18px, regular)
└── Body Small (14px, regular)
```

4. Structure for CSS Grid/Flexbox

Design layouts that naturally translate to CSS Grid and Flexbox:

For CSS Grid:

- Use regular, grid-like layouts
- Consistent spacing between elements
- Clear column/row structure

For Flexbox:

- Linear arrangements (horizontal/vertical)
- Clear alignment patterns
- Consistent spacing systems

5. Simplify Visual Complexity

Background and Images:

- Use single background images where possible
- Avoid complex layering effects
- Use CSS-friendly gradients
- Keep shadows simple and consistent

Text and Content:

- Avoid text on complex backgrounds
- Use consistent text containers

- Ensure sufficient contrast
- Group related content logically

Technical Recommendations {#technical-recommendations}

For Designers

1. **Before Starting Design:**
    - Establish a design system with semantic naming
    - Create component libraries with proper naming
    - Define spacing and sizing systems (8px grid, etc.)
2. **During Design:**
    - Name every layer semantically
    - Use Auto Layout extensively
    - Think in terms of HTML structure
    - Avoid unnecessary grouping and nesting
3. **Before Handoff:**
    - Review layer structure for logical hierarchy
    - Ensure consistent component usage
    - Document any complex interactions separately
    - Provide style guide with design tokens

For Developers Using DivRiots Plugin

1. **Strategic Use of the Plugin:**
    - Use DivRiots for **rapid prototyping and client previews**
    - Export the generated code as a **reference, not final product**
    - Leverage their hosting for quick client approvals, then rebuild properly
    - Use the plugin's responsive preview to understand breakpoint behavior
2. **DivRiots-Specific Workflow:**

```
1. Design in Figma with DivRiots best practices
2. Use plugin to generate rapid prototype
3. Show clients the live preview on figweb.site
4. Export the code to understand structure
5. Rebuild with semantic HTML structure
6. Use exported CSS as reference for styling
7. Host on your own infrastructure
```

3. **When to Use DivRiots Directly:**
    - Landing pages that don't need much customization

- Client presentations and quick mockups
- Testing responsive design behavior
- Projects with very tight deadlines

Recommended Figma-to-DivRiots Workflow

Based on the structural issues identified in the example Figma file:

1. **Pre-Design Planning:**
   - Map out HTML structure before opening Figma
   - Define semantic naming conventions for the team
   - Create a component library with proper naming
   - Establish Auto Layout patterns for common elements
2. **During Design Phase:**
   - Name every layer semantically (avoid `Frame 28`, `Rectangle 110`)
   - Use components with variants instead of duplicates
   - Group elements by HTML sections (`Header`, `Main`, `Footer`)
   - Apply Auto Layout extensively for responsive behavior
3. **Pre-Export Checklist:**
   - Review layer names for semantic meaning
   - Ensure no duplicate components (consolidate with variants)
   - Check that grouping reflects intended HTML structure
   - Test component behavior in different breakpoints
4. **DivRiots Export Process:**
   - Export a small section first to test code quality
   - Review generated HTML structure
   - Adjust Figma structure if code is too complex
   - Re-export and iterate until satisfied
5. **Post-Export Optimization:**
   - Use generated code as reference, not final product
   - Rebuild with semantic HTML structure
   - Apply clean CSS architecture
   - Test responsive behavior across devices

Key Success Metrics:

- Generated class names should be meaningful
- HTML structure should be shallow (not deeply nested)
- Components should map to reusable code patterns
- Responsive behavior should work without major CSS overrides

For General Development (Alternative to DivRiots)

1. **Instead of Direct Conversion Tools:**
    - Use Figma as visual reference, not source of truth for code
    - Implement semantic HTML structure first
    - Build component libraries in code
    - Use CSS Grid/Flexbox for layouts
2. **Better Workflow:**

```
1. Analyze Figma design for components and patterns
2. Create semantic HTML structure
3. Implement design system in CSS
4. Build reusable components
5. Match visual design through CSS
```

For AI-Assisted Development

1. **Figma MCP (Model Context Protocol) Server:**
    - Provides better context to AI coding tools
    - Enables design-system-aware code generation
    - Works with tools like Claude, Cursor, etc.
2. **Better Prompting for AI:**

```
Instead of: "Convert this Figma design to HTML"

Try: "Create a semantic HTML structure for a hero section with:
- Main heading
- Subtitle text
- Primary CTA button
- Background image
- Responsive layout using CSS Grid"
```

Alternative Workflows {#alternatives}

1. Design-System-First Approach

1. **Create Design System in Figma:**
    - Build comprehensive component library
    - Establish design tokens (colors, typography, spacing)
    - Use semantic naming throughout
2. **Mirror in Code:**
    - Build corresponding CSS/component library

- Use tools like Storybook for component documentation
- Maintain design-code synchronization

2. Progressive Enhancement Workflow

1. **Start with HTML Structure:**
   - Write semantic HTML first
   - Focus on content hierarchy and accessibility
   - Ensure proper document outline
2. **Add Design Layer by Layer:**
   - Implement layout with CSS Grid/Flexbox
   - Add typography and spacing
   - Include colors and backgrounds
   - Add interactions and animations last

3. Component-Driven Development

1. **Identify Design Patterns:**
   - Break design into reusable components
   - Define component APIs and variants
   - Document usage patterns
2. **Build Component Library:**
   - Start with basic atoms (buttons, inputs)
   - Combine into molecules (forms, cards)
   - Build organism-level components (headers, sections)

4. DivRiots + AI-Assisted Hybrid Workflow

1. **Use DivRiots for Initial Conversion:**
   - Generate initial website with the plugin
   - Export the HTML/CSS code (available in paid plans)
   - Use as reference for understanding layout structure
   - Test responsive behavior on their preview
2. **Combine with AI for Clean Rebuild:**
   - Feed the DivRiots-generated code to AI (Claude/GPT)
   - Ask AI to "refactor this into semantic HTML structure"
   - Request modular CSS that maintains visual fidelity
   - Generate accessible, maintainable version
3. **Best of Both Worlds:**
   - Speed of DivRiots conversion
   - Quality of AI-assisted refactoring

- Maintain design accuracy
- Get clean, maintainable code

5. Traditional AI-Assisted Workflow

1. **Use Figma MCP Server:**
   - Enables AI to understand design context
   - Provides better code generation results
   - Maintains design system consistency
2. **Combine with Claude/GPT:**
   - Provide design context through MCP
   - Ask for semantic HTML structure first
   - Request modular CSS implementation
   - Generate accessible, maintainable code

Conclusion

The DivRiots figma.to.website plugin represents one of the best available Figma-to-code conversion tools, but even it faces fundamental challenges when trying to generate maintainable, semantic HTML. The issue isn't poor engineering—it's the inherent complexity of translating visual design systems into structured code.

Key Takeaways for DivRiots Users:

1. **Use DivRiots strategically**: Great for prototypes, client previews, and quick launches
2. **Design with the plugin's strengths in mind**: Follow their best practices for Auto Layout and component structure
3. **Export and refactor**: Use the generated code as a starting point, then refactor for maintainability
4. **Hybrid approach works best**: Combine DivRiots speed with AI-assisted code improvement
5. **Know when to rebuild**: For complex, long-term projects, use DivRiots output as reference only

The DivRiots plugin is actually quite good at what it does—creating visually accurate, performant websites quickly. The challenge comes when you need the underlying code to be maintainable, semantic, and customizable beyond the plugin's ecosystem.

For your specific use case: The plugin worked correctly in generating a functional website, but the code quality issues you encountered are inherent to the visual-first approach. Consider using the DivRiots output as a reference while rebuilding with semantic HTML structure for long-term maintainability.

Last updated: December 28, 2025