SPARK Emitter Architecture for Semantic IR

Document Version: 1.0
Created: 2026-01-31
DO-178C Level: A
SPARK Compliance: Required

---

1. Executive Summary

This document defines the architecture for STUNIR's SPARK-based code emitters that consume Semantic IR (v1) and generate code for multiple target platforms. The architecture ensures:

- **DO-178C Level A compliance** through formal verification
- **Memory safety** via SPARK contracts and bounded types
- **Deterministic code generation** with cryptographic receipts
- **Multi-target support** across 5 core categories

---

2. Architecture Overview

2.1 High-Level Data Flow

```
Semantic IR (JSON) → IR Parser → Semantic IR Tree → Emitter Visitor → Target Code
                                          ↓
                                   SPARK Contracts
                                          ↓
                                   Formal Verification
```

2.2 Core Components

1. **Base Emitter Package** (`emitters.ads`)
   - Abstract interface for all emitters
   - Common types and contracts
   - Memory safety guarantees

2. **IR Visitor** (`emitters-visitor.ads/adb`)
   - Traverses Semantic IR tree
   - Invokes emitter callbacks
   - Maintains verification invariants

3. **Code Generator** (`emitters-codegen.ads/adb`)
   - Bounded string buffers
   - Indentation management
   - Comment generation

4. **Category Emitters**
   - Embedded: `emitters-embedded.ads/adb`

- GPU: `emitters-gpu.ads/adb`
- WASM: `emitters-wasm.ads/adb`
- Assembly: `emitters-assembly.ads/adb`
- Polyglot: `emitters-polyglot.ads/adb`

---

3. Semantic IR Data Model (SPARK)

3.1 Core Types

```
package STUNIR.Semantic_IR is
   pragma SPARK_Mode (On);

   -- Bounded strings for memory safety
   Max_Name_Length : constant := 128;
   Max_Type_Length : constant := 64;
   Max_Doc_Length  : constant := 1024;
   Max_Code_Length : constant := 65536;

   subtype IR_Name_String is String (1 .. Max_Name_Length);
   subtype IR_Type_String is String (1 .. Max_Type_Length);
   subtype IR_Doc_String  is String (1 .. Max_Doc_Length);
   subtype IR_Code_Buffer is String (1 .. Max_Code_Length);

   -- Primitive types
   type IR_Primitive_Type is
     (Type_String, Type_Int, Type_Float, Type_Bool, Type_Void,
      Type_I8, Type_I16, Type_I32, Type_I64,
      Type_U8, Type_U16, Type_U32, Type_U64,
      Type_F32, Type_F64);

   -- Field definition
   type IR_Field is record
      Name     : IR_Name_String;
      Type_Ref : IR_Type_String;
      Optional : Boolean;
      Name_Len : Natural range 0 .. Max_Name_Length;
      Type_Len : Natural range 0 .. Max_Type_Length;
   end record
   with Dynamic_Predicate =>
     Name_Len <= Max_Name_Length and Type_Len <= Max_Type_Length;

   -- Type definition
   Max_Fields : constant := 50;
   type Field_Array is array (Positive range <>) of IR_Field;

   type IR_Type_Def is record
      Name      : IR_Name_String;
      Docstring : IR_Doc_String;
      Fields    : Field_Array (1 .. Max_Fields);
      Name_Len  : Natural range 0 .. Max_Name_Length;
      Doc_Len   : Natural range 0 .. Max_Doc_Length;
      Field_Cnt : Natural range 0 .. Max_Fields;
   end record
   with Dynamic_Predicate =>
     Name_Len <= Max_Name_Length and
     Doc_Len <= Max_Doc_Length and
     Field_Cnt <= Max_Fields;

   -- Function argument
   type IR_Arg is record
      Name     : IR_Name_String;
      Type_Ref : IR_Type_String;
      Name_Len : Natural range 0 .. Max_Name_Length;
      Type_Len : Natural range 0 .. Max_Type_Length;
   end record
   with Dynamic_Predicate =>
     Name_Len <= Max_Name_Length and Type_Len <= Max_Type_Length;

   -- Statement (simplified for Phase 3a)
   type IR_Statement_Kind is
     (Stmt_Assign, Stmt_Call, Stmt_Return, Stmt_If, Stmt_Loop);
```

```
   type IR_Statement is record
      Kind : IR_Statement_Kind;
      Data : IR_Code_Buffer;
      Len  : Natural range 0 .. Max_Code_Length;
   end record
   with Dynamic_Predicate => Len <= Max_Code_Length;

   -- Function definition
   Max_Args       : constant := 20;
   Max_Statements : constant := 100;
   type Arg_Array is array (Positive range <>) of IR_Arg;
   type Statement_Array is array (Positive range <>) of IR_Statement;

   type IR_Function is record
      Name        : IR_Name_String;
      Docstring   : IR_Doc_String;
      Args        : Arg_Array (1 .. Max_Args);
      Return_Type : IR_Type_String;
      Statements  : Statement_Array (1 .. Max_Statements);
      Name_Len    : Natural range 0 .. Max_Name_Length;
      Doc_Len     : Natural range 0 .. Max_Doc_Length;
      Arg_Cnt     : Natural range 0 .. Max_Args;
      Ret_Len     : Natural range 0 .. Max_Type_Length;
      Stmt_Cnt    : Natural range 0 .. Max_Statements;
   end record
   with Dynamic_Predicate =>
     Name_Len <= Max_Name_Length and
     Doc_Len <= Max_Doc_Length and
     Arg_Cnt <= Max_Args and
     Ret_Len <= Max_Type_Length and
     Stmt_Cnt <= Max_Statements;

   -- Module (top-level IR)
   Max_Types     : constant := 100;
   Max_Functions : constant := 100;
   type Type_Array is array (Positive range <>) of IR_Type_Def;
   type Function_Array is array (Positive range <>) of IR_Function;

   type IR_Module is record
      IR_Version  : String (1 .. 2);  -- "v1"
      Module_Name : IR_Name_String;
      Docstring   : IR_Doc_String;
      Types       : Type_Array (1 .. Max_Types);
      Functions   : Function_Array (1 .. Max_Functions);
      Name_Len    : Natural range 0 .. Max_Name_Length;
      Doc_Len     : Natural range 0 .. Max_Doc_Length;
      Type_Cnt    : Natural range 0 .. Max_Types;
      Func_Cnt    : Natural range 0 .. Max_Functions;
   end record
   with Dynamic_Predicate =>
     Name_Len <= Max_Name_Length and
     Doc_Len <= Max_Doc_Length and
     Type_Cnt <= Max_Types and
     Func_Cnt <= Max_Functions;

end STUNIR.Semantic_IR;
```

4. Base Emitter Interface

4.1 Abstract Emitter Class

```
package STUNIR.Emitters is
   pragma SPARK_Mode (On);

   type Target_Category is
     (Category_Embedded, Category_GPU, Category_WASM,
      Category_Assembly, Category_Polyglot);

   type Emitter_Status is (Success, Error_Parse, Error_Generate, Error_IO);

   -- Abstract emitter interface
   type Base_Emitter is abstract tagged record
      Category : Target_Category;
      Status   : Emitter_Status;
   end record;

   -- Abstract methods (must be overridden)
   procedure Emit_Module
     (Self   : in out Base_Emitter;
      Module : in     STUNIR.Semantic_IR.IR_Module;
      Output :    out IR_Code_Buffer;
      Length :    out Natural)
   is abstract
   with
     Pre'Class  => Module.Func_Cnt > 0,
     Post'Class => Length <= Max_Code_Length;

   procedure Emit_Type
     (Self   : in out Base_Emitter;
      T      : in     STUNIR.Semantic_IR.IR_Type_Def;
      Output :    out IR_Code_Buffer;
      Length :    out Natural)
   is abstract
   with
     Pre'Class  => T.Field_Cnt > 0,
     Post'Class => Length <= Max_Code_Length;

   procedure Emit_Function
     (Self   : in out Base_Emitter;
      Func   : in     STUNIR.Semantic_IR.IR_Function;
      Output :    out IR_Code_Buffer;
      Length :    out Natural)
   is abstract
   with
     Pre'Class  => Func.Name_Len > 0,
     Post'Class => Length <= Max_Code_Length;

end STUNIR.Emitters;
```

5. Visitor Pattern for IR Traversal

5.1 Visitor Interface

```
package STUNIR.Emitters.Visitor is
   pragma SPARK_Mode (On);

   type Visit_Result is (Continue, Skip, Abort_Visit);

   type Visitor_Callbacks is record
      On_Module_Start  : access procedure (Module : IR_Module);
      On_Module_End    : access procedure (Module : IR_Module);
      On_Type_Start    : access procedure (T : IR_Type_Def);
      On_Type_End      : access procedure (T : IR_Type_Def);
      On_Function_Start: access procedure (Func : IR_Function);
      On_Function_End  : access procedure (Func : IR_Function);
   end record;

   procedure Traverse_Module
     (Module    : in IR_Module;
      Callbacks : in Visitor_Callbacks;
      Result    : out Visit_Result)
   with
     Pre  => Module.Func_Cnt > 0,
     Post => Result in Visit_Result;

end STUNIR.Emitters.Visitor;
```

6. Code Generation Utilities

6.1 Bounded Buffers and Indentation

```
package STUNIR.Emitters.CodeGen is
   pragma SPARK_Mode (On);

   type Indent_Level is range 0 .. 10;

   type Code_Generator is record
      Buffer       : IR_Code_Buffer;
      Current_Pos  : Natural range 0 .. Max_Code_Length;
      Indent       : Indent_Level;
      Indent_Width : Positive range 2 .. 8;
   end record
   with Dynamic_Predicate => Current_Pos <= Max_Code_Length;

   procedure Initialize
     (Gen : out Code_Generator;
      Indent_Width : Positive := 4)
   with
     Post => Gen.Current_Pos = 0 and Gen.Indent = 0;

   procedure Append_Line
     (Gen  : in out Code_Generator;
      Line : in     String;
      Success : out Boolean)
   with
     Pre  => Gen.Current_Pos < Max_Code_Length,
     Post => (if Success then
                Gen.Current_Pos <= Max_Code_Length);

   procedure Increase_Indent (Gen : in out Code_Generator)
   with
     Pre  => Gen.Indent < Indent_Level'Last,
     Post => Gen.Indent = Gen.Indent'Old + 1;

   procedure Decrease_Indent (Gen : in out Code_Generator)
   with
     Pre  => Gen.Indent > 0,
     Post => Gen.Indent = Gen.Indent'Old - 1;

   procedure Get_Output
     (Gen    : in  Code_Generator;
      Output : out String;
      Length : out Natural)
   with
     Pre  => Gen.Current_Pos <= Max_Code_Length,
     Post => Length = Gen.Current_Pos;

end STUNIR.Emitters.CodeGen;
```

7. Category-Specific Emitters

7.1 Embedded Emitter

Targets: ARM, ARM64, RISC-V, MIPS, AVR, x86

Outputs: Startup code, linker scripts, memory management

```
package STUNIR.Emitters.Embedded is
   pragma SPARK_Mode (On);

   type Architecture is
     (Arch_ARM, Arch_ARM64, Arch_RISCV, Arch_MIPS, Arch_AVR, Arch_X86);

   type Memory_Region is record
      Name   : IR_Name_String;
      Start  : Natural;
      Length : Natural;
   end record;

   type Embedded_Config is record
      Arch          : Architecture;
      Use_FPU       : Boolean;
      Stack_Size    : Positive;
      Heap_Size     : Positive;
      Generate_Linker : Boolean;
   end record;

   type Embedded_Emitter is new Base_Emitter with record
      Config : Embedded_Config;
   end record;

   overriding procedure Emit_Module
     (Self   : in out Embedded_Emitter;
      Module : in     IR_Module;
      Output :    out IR_Code_Buffer;
      Length :    out Natural);

   procedure Emit_Startup_Code
     (Self   : in out Embedded_Emitter;
      Output :    out IR_Code_Buffer;
      Length :    out Natural)
   with
     Pre  => Self.Config.Stack_Size > 0,
     Post => Length <= Max_Code_Length;

   procedure Emit_Linker_Script
     (Self   : in out Embedded_Emitter;
      Output :    out IR_Code_Buffer;
      Length :    out Natural)
   with
     Pre  => Self.Config.Generate_Linker,
     Post => Length <= Max_Code_Length;

end STUNIR.Emitters.Embedded;
```

7.2 GPU Emitter

Targets: CUDA, OpenCL, Metal, ROCm, Vulkan

Outputs: Kernel code, memory management, synchronization

```
package STUNIR.Emitters.GPU is
   pragma SPARK_Mode (On);

   type GPU_Platform is
     (Platform_CUDA, Platform_OpenCL, Platform_Metal,
      Platform_ROCm, Platform_Vulkan);

   type GPU_Config is record
      Platform       : GPU_Platform;
      Compute_Cap    : String (1 .. 10);  -- e.g., "sm_75"
      Use_Shared_Mem : Boolean;
      Max_Threads    : Positive;
   end record;

   type GPU_Emitter is new Base_Emitter with record
      Config : GPU_Config;
   end record;

   overriding procedure Emit_Module
     (Self   : in out GPU_Emitter;
      Module : in     IR_Module;
      Output :    out IR_Code_Buffer;
      Length :    out Natural);

   procedure Emit_Kernel
     (Self   : in out GPU_Emitter;
      Func   : in     IR_Function;
      Output :    out IR_Code_Buffer;
      Length :    out Natural)
   with
     Pre  => Func.Name_Len > 0,
     Post => Length <= Max_Code_Length;

end STUNIR.Emitters.GPU;
```

7.3 WASM Emitter

Targets: WASM, WASI

Outputs: WebAssembly modules with SIMD and bulk memory ops

```
package STUNIR.Emitters.WASM is
   pragma SPARK_Mode (On);

   type WASM_Feature is (Feature_SIMD, Feature_Bulk_Memory, Feature_Threads);
   type Feature_Set is array (WASM_Feature) of Boolean;

   type WASM_Config is record
      Use_WASI    : Boolean;
      Features    : Feature_Set;
      Export_All  : Boolean;
   end record;

   type WASM_Emitter is new Base_Emitter with record
      Config : WASM_Config;
   end record;

   overriding procedure Emit_Module
     (Self   : in out WASM_Emitter;
      Module : in     IR_Module;
      Output :    out IR_Code_Buffer;
      Length :    out Natural);

end STUNIR.Emitters.WASM;
```

7.4 Assembly Emitter

Targets: x86, ARM assembly

Outputs: Optimized assembly code

```
package STUNIR.Emitters.Assembly is
   pragma SPARK_Mode (On);

   type Assembly_Target is (Target_X86, Target_X86_64, Target_ARM, Target_ARM64);
   type Assembly_Syntax is (Syntax_Intel, Syntax_ATT, Syntax_ARM);

   type Assembly_Config is record
      Target     : Assembly_Target;
      Syntax     : Assembly_Syntax;
      Optimize   : Boolean;
      Add_Debug  : Boolean;
   end record;

   type Assembly_Emitter is new Base_Emitter with record
      Config : Assembly_Config;
   end record;

   overriding procedure Emit_Module
     (Self   : in out Assembly_Emitter;
      Module : in     IR_Module;
      Output :    out IR_Code_Buffer;
      Length :    out Natural);

end STUNIR.Emitters.Assembly;
```

7.5 Polyglot Emitter

Targets: C89, C99, Rust

Outputs: Idiomatic code for each language

```
package STUNIR.Emitters.Polyglot is
   pragma SPARK_Mode (On);

   type Target_Language is (Lang_C89, Lang_C99, Lang_Rust);

   type Polyglot_Config is record
      Language       : Target_Language;
      Use_StdLib     : Boolean;
      Generate_Tests : Boolean;
   end record;

   type Polyglot_Emitter is new Base_Emitter with record
      Config : Polyglot_Config;
   end record;

   overriding procedure Emit_Module
     (Self   : in out Polyglot_Emitter;
      Module : in     IR_Module;
      Output :    out IR_Code_Buffer;
      Length :    out Natural);

   procedure Emit_C89
     (Module : in IR_Module;
      Output : out IR_Code_Buffer;
      Length : out Natural)
   with
     Pre  => Module.Func_Cnt > 0,
     Post => Length <= Max_Code_Length;

   procedure Emit_C99
     (Module : in IR_Module;
      Output : out IR_Code_Buffer;
      Length : out Natural)
   with
     Pre  => Module.Func_Cnt > 0,
     Post => Length <= Max_Code_Length;

   procedure Emit_Rust
     (Module : in IR_Module;
      Output : out IR_Code_Buffer;
      Length : out Natural)
   with
     Pre  => Module.Func_Cnt > 0,
     Post => Length <= Max_Code_Length;

end STUNIR.Emitters.Polyglot;
```

---

8. Formal Verification Strategy

8.1 SPARK Contracts

All emitters must include:

1. **Preconditions** - Input validation
2. **Postconditions** - Output guarantees
3. **Type Invariants** - Data consistency
4. **Loop Invariants** - Progress guarantees

8.2 Verification Levels

Level	Verification Goal	GNATprove Mode
1	No runtime errors	`--level=1`
2	Type safety	`--level=2`
3	Overflow protection	`--level=3`
4	Full functional correctness	`--level=4`

8.3 Proof Obligations

- **Memory Safety**: All array accesses within bounds
- **No Buffer Overflows**: String operations checked
- **Type Correctness**: Type conversions validated
- **Determinism**: Same input → Same output

---

9. DO-178C Level A Compliance

9.1 Requirements Traceability

Each emitter must trace to:
- Requirements Document
- Design Document (this doc)
- Test Cases
- Verification Results

9.2 Code Standards

- **Coding Standard**: MISRA Ada 2012
- **Complexity**: Cyclomatic complexity ≤ 10
- **Coverage**: 100% MC/DC coverage
- **Documentation**: All public APIs documented

9.3 Verification Artifacts

1. Static Analysis Reports (GNATprove)
2. Test Coverage Reports
3. Code Review Checklists
4. Traceability Matrix

---

10. Build and Integration

10.1 GNAT Project Structure

```
tools/spark/
├── stunir_emitters.gpr      # New emitter project
├── src/
│   ├── emitters/
│   │   ├── stunir-semantic_ir.ads/adb
│   │   ├── stunir-emitters.ads
│   │   ├── stunir-emitters-base.adb
│   │   ├── stunir-emitters-visitor.ads/adb
│   │   ├── stunir-emitters-codegen.ads/adb
│   │   ├── stunir-emitters-embedded.ads/adb
│   │   ├── stunir-emitters-gpu.ads/adb
│   │   ├── stunir-emitters-wasm.ads/adb
│   │   ├── stunir-emitters-assembly.ads/adb
│   │   └── stunir-emitters-polyglot.ads/adb
│   └── (existing files)
├── obj/                     # Build artifacts
└── bin/                     # Executables
```

10.2 Build Commands

```
# Build all emitters
gprbuild -P tools/spark/stunir_emitters.gpr

# Verify with SPARK
gnatprove -P tools/spark/stunir_emitters.gpr --level=4 --mode=all

# Run tests
gnatmake -P tests/spark/emitter_tests.gpr
./tests/spark/bin/run_all_tests
```

---

11. Testing Strategy

11.1 Test Levels

1. **Unit Tests** - Individual emitter procedures
2. **Integration Tests** - Full IR → Code pipeline
3. **Property Tests** - Determinism verification
4. **Regression Tests** - Against known outputs

11.2 Test Coverage Metrics

- **Statement Coverage**: 100%
- **Branch Coverage**: 100%
- **MC/DC Coverage**: 100%
- **Proof Coverage**: 100%

---

12. Migration Path from Hash-Based IR

12.1 Compatibility Layer

To support gradual migration, provide a compatibility shim:

```
package STUNIR.IR_Compat is
   pragma SPARK_Mode (On);

   procedure Convert_Hash_IR_To_Semantic
     (Hash_IR   : in  String;  -- Old format
      Semantic  : out IR_Module;
      Success   : out Boolean)
   with
     Pre  => Hash_IR'Length > 0,
     Post => (if Success then Semantic.Func_Cnt > 0);

end STUNIR.IR_Compat;
```

12.2 Deprecation Timeline

- **Phase 3a**: Both formats supported
- **Phase 3b**: Semantic IR recommended
- **Phase 4**: Hash-based IR deprecated
- **Phase 5**: Hash-based IR removed

---

13. Performance Considerations

13.1 Memory Bounds

Structure	Max Size	Rationale
Module Name	128 bytes	Reasonable identifier length
Type Count	100	Typical program complexity
Function Count	100	Typical module size
Code Buffer	64 KB	Large enough for most outputs

13.2 Optimization Goals

- **Parsing**: < 100ms for typical IR
- **Code Generation**: < 500ms per emitter
- **Memory Usage**: < 10MB per emitter instance

---

14. Security Considerations

14.1 Input Validation

- All JSON inputs validated against schema
- Buffer overflows prevented via bounded types
- No dynamic memory allocation in critical paths

14.2 Deterministic Hashing

All generated code includes SHA-256 hash:

```
function Compute_Code_Hash
  (Code : IR_Code_Buffer;
   Len  : Natural)
  return Hash_String
with
  Pre  => Len <= Max_Code_Length,
  Post => Compute_Code_Hash'Result'Length = 64;  -- SHA-256 hex
```

---

15. Documentation Requirements

Each emitter must include:

1. **API Documentation** - All public procedures
2. **Usage Examples** - At least 3 per emitter
3. **Verification Report** - GNATprove results
4. **Test Report** - Coverage and results
5. **User Guide** - Target-specific instructions

---

16. Revision History

Version	Date	Author	Changes
1.0	2026-01-31	STUNIR Team	Initial architecture design

---

17. References

- DO-178C: Software Considerations in Airborne Systems
- MISRA Ada 2012: Guidelines for Safety-Critical Systems
- SPARK User's Guide: https://docs.adacore.com/spark2014-docs/html/ug/
- STUNIR IR Schema: `schemas/stunir_ir_v1.schema.json`

END OF DOCUMENT