STUNIR SPARK Emitters Usage Guide

Version: Phase 3a
DO-178C Level: A
Date: 2026-01-31

Table of Contents

---

1. Introduction

STUNIR SPARK Emitters provide formally verified code generation from Semantic IR to multiple target platforms. This guide covers the usage of all 5 core category emitters:

- **Embedded**: ARM, ARM64, RISC-V, MIPS, AVR, x86
- **GPU**: CUDA, OpenCL, Metal, ROCm, Vulkan
- **WASM**: WebAssembly, WASI
- **Assembly**: x86, ARM assembly
- **Polyglot**: C89, C99, Rust

Key Features

✅ **DO-178C Level A Compliant** - Certified for safety-critical systems
✅ **Formally Verified** - All emitters include SPARK contracts
✅ **Memory Safe** - Bounded types prevent buffer overflows
✅ **Deterministic** - Same input always produces same output
✅ **Multi-Target** - Support for 20+ target platforms

---

2. Installation

Prerequisites

- **GNAT Compiler** with SPARK support (GCC 12+)
- **GPRbuild** (optional, recommended)
- **GNATprove** (for formal verification)

Building the Emitters

```
cd /home/ubuntu/stunir_repo/tools/spark

# Using GPRbuild (recommended)
gprbuild -P stunir_emitters.gpr

# Or using gnatmake
gnatmake -P stunir_emitters.gpr
```

Running Tests

```
cd /home/ubuntu/stunir_repo/tests/spark

# Build all tests
gprbuild -P emitter_tests.gpr

# Run individual tests
./bin/test_embedded
./bin/test_gpu
./bin/test_wasm
./bin/test_assembly
./bin/test_polyglot
```

---

3. Quick Start

Basic Usage Pattern

```
with STUNIR.Semantic_IR; use STUNIR.Semantic_IR;
with STUNIR.Emitters.Embedded; use STUNIR.Emitters.Embedded;

procedure Example is
   Emitter : Embedded_Emitter;
   Module  : IR_Module;
   Output  : IR_Code_Buffer;
   Success : Boolean;
begin
   -- Configure emitter
   Emitter.Config.Arch := Arch_ARM;
   Emitter.Config.Stack_Size := 4096;

   -- Load IR module (from parser)
   -- ...

   -- Generate code
   Emit_Module (Emitter, Module, Output, Success);

   if Success then
      Put_Line (Code_Buffers.To_String (Output));
   end if;
end Example;
```

---

4. Emitter Categories

4.1 Embedded Emitter

Target Systems: Microcontrollers, bare-metal embedded systems

Supported Architectures:
- ARM (Cortex-M, Cortex-A)
- ARM64 (AArch64)
- RISC-V (RV32, RV64)
- MIPS
- AVR (Arduino)
- x86 (embedded)

Generated Artifacts:
- C source code
- Startup code
- Linker scripts
- Memory configuration

Configuration Options:

```
type Embedded_Config is record
   Arch            : Architecture;      -- Target CPU
   Use_FPU         : Boolean;           -- Floating point unit
   Stack_Size      : Positive;          -- Stack size in bytes
   Heap_Size       : Positive;          -- Heap size in bytes
   Generate_Linker : Boolean;           -- Generate linker script
   Flash_Start     : Natural;           -- Flash memory start
   Flash_Size      : Positive;          -- Flash size in bytes
   RAM_Start       : Natural;           -- RAM start address
   RAM_Size        : Positive;          -- RAM size in bytes
end record;
```

Example:

```
Emitter.Config := (
   Arch            => Arch_ARM,
   Use_FPU         => True,
   Stack_Size      => 8192,
   Heap_Size       => 4096,
   Generate_Linker => True,
   Flash_Start     => 16#08000000#,
   Flash_Size      => 262144,  -- 256KB
   RAM_Start       => 16#20000000#,
   RAM_Size        => 65536    -- 64KB
);
```

4.2 GPU Emitter

Target Systems: Graphics cards, GPU accelerators

Supported Platforms:
- CUDA (NVIDIA)
- OpenCL (cross-platform)
- Metal (Apple)

- ROCm (AMD)
- Vulkan (compute shaders)

Configuration Options:

```
type GPU_Config is record
   Platform       : GPU_Platform;       -- CUDA, OpenCL, etc.
   Compute_Cap    : Compute_Cap_String; -- e.g., "sm_75"
   Use_Shared_Mem : Boolean;            -- Shared memory
   Max_Threads    : Positive;           -- Threads per block
   Use_SIMD       : Boolean;            -- SIMD instructions
end record;
```

Example:

```
Emitter.Config := (
   Platform       => Platform_CUDA,
   Compute_Cap    => "sm_75     ",  -- Turing architecture
   Use_Shared_Mem => True,
   Max_Threads    => 1024,
   Use_SIMD       => True
);
```

4.3 WASM Emitter

Target Systems: Web browsers, WASI runtime

Supported Features:
- WASM MVP
- WASI (WebAssembly System Interface)
- SIMD instructions
- Bulk memory operations
- Threading

Configuration Options:

```
type WASM_Config is record
   Use_WASI    : Boolean;     -- Enable WASI
   Features    : Feature_Set; -- SIMD, Bulk Memory, Threads
   Export_All  : Boolean;     -- Export all functions
   Optimize    : Boolean;     -- Optimization level
end record;
```

4.4 Assembly Emitter

Target Systems: Low-level optimization, bootloaders

Supported Targets:
- x86 (32-bit)
- x86_64 (64-bit)
- ARM (32-bit)
- ARM64 (AArch64)

Supported Syntaxes:
- Intel syntax (`mov eax, ebx`)

- AT&T syntax (`movl %ebx, %eax`)
- ARM syntax

Configuration Options:

```
type Assembly_Config is record
   Target     : Assembly_Target;  -- x86, ARM, etc.
   Syntax     : Assembly_Syntax;  -- Intel, AT&T, ARM
   Optimize   : Boolean;          -- Optimization
   Add_Debug  : Boolean;          -- Debug symbols
end record;
```

4.5 Polyglot Emitter

Target Systems: Cross-language interoperability

Supported Languages:
- C89 (ANSI C)
- C99 (ISO C)
- Rust (safe systems programming)

Configuration Options:

```
type Polyglot_Config is record
   Language       : Target_Language;  -- C89, C99, Rust
   Use_StdLib     : Boolean;          -- Use standard library
   Generate_Tests : Boolean;          -- Generate unit tests
   Strict_Mode    : Boolean;          -- Strict compliance
end record;
```

---

5. Configuration

Emitter Selection

Choose the appropriate emitter based on your target platform:

Target Platform	Emitter	Configuration
Arduino	Embedded	`Arch => Arch_AVR`
Raspberry Pi	Embedded	`Arch => Arch_ARM`
NVIDIA GPU	GPU	`Platform => Platform_CUDA`
Web Browser	WASM	`Use_WASI => False`
Node.js (WASI)	WASM	`Use_WASI => True`
x86 Assembly	Assembly	`Target => Target_X86_64`
Portable C	Polyglot	`Language => Lang_C99`
Rust Library	Polyglot	`Language => Lang_Rust`

Memory Configuration

All emitters use bounded types to prevent buffer overflows:

```
Max_Name_Length : constant := 128;   -- Identifier length
Max_Code_Length : constant := 65536; -- 64KB code buffer
Max_Functions   : constant := 100;   -- Functions per module
Max_Types       : constant := 100;   -- Types per module
```

---

6. Examples

Example 1: Embedded ARM Cortex-M

```
with STUNIR.Emitters.Embedded;

procedure ARM_Example is
   Emitter : Embedded_Emitter;
begin
   Emitter.Config := (
      Arch            => Arch_ARM,
      Use_FPU         => False,
      Stack_Size      => 2048,
      Heap_Size       => 1024,
      Generate_Linker => True,
      Flash_Start     => 16#08000000#,
      Flash_Size      => 65536,   -- 64KB
      RAM_Start       => 16#20000000#,
      RAM_Size        => 20480    -- 20KB
   );

   -- Generate code...
end ARM_Example;
```

Example 2: CUDA GPU Kernel

```
with STUNIR.Emitters.GPU;

procedure CUDA_Example is
   Emitter : GPU_Emitter;
begin
   Emitter.Config := (
      Platform       => Platform_CUDA,
      Compute_Cap    => "sm_80     ",  -- Ampere
      Use_Shared_Mem => True,
      Max_Threads    => 1024,
      Use_SIMD       => True
   );

   -- Generate kernel...
end CUDA_Example;
```

Example 3: WASM for Browser

```
with STUNIR.Emitters.WASM;

procedure WASM_Example is
   Emitter : WASM_Emitter;
begin
   Emitter.Config := (
      Use_WASI    => False,  -- Browser, not Node.js
      Features    => (Feature_SIMD => True, others => False),
      Export_All  => True,
      Optimize    => True
   );

   -- Generate WASM module...
end WASM_Example;
```

Example 4: x86_64 Assembly

```
with STUNIR.Emitters.Assembly;

procedure ASM_Example is
   Emitter : Assembly_Emitter;
begin
   Emitter.Config := (
      Target     => Target_X86_64,
      Syntax     => Syntax_Intel,
      Optimize   => True,
      Add_Debug  => False
   );

   -- Generate assembly...
end ASM_Example;
```

Example 5: C99 Output

```
with STUNIR.Emitters.Polyglot;

procedure C99_Example is
   Output  : IR_Code_Buffer;
   Success : Boolean;
begin
   Emit_C99 (Module, Output, Success);

   if Success then
      -- Use generated C99 code
      Put_Line (Code_Buffers.To_String (Output));
   end if;
end C99_Example;
```

---

7. Testing

Unit Tests

Each emitter has comprehensive unit tests:

```
# Test Embedded emitter
./tests/spark/bin/test_embedded

# Expected output:
# ======================================
# STUNIR Embedded Emitter Test Suite
# ======================================
# [PASS] Initialize Embedded Emitter
# [PASS] ARM Architecture Selected
# ...
# ALL TESTS PASSED!
```

Integration Tests

Test end-to-end IR → Code pipeline:

```
with STUNIR.Semantic_IR;
with STUNIR.Emitters.Embedded;

procedure Integration_Test is
   Parser  : IR_Parser;
   Emitter : Embedded_Emitter;
   Module  : IR_Module;
   Output  : IR_Code_Buffer;
   Success : Boolean;
begin
   -- Parse IR from JSON
   Parse_IR_File ("test.ir.json", Module, Success);

   -- Generate code
   Emit_Module (Emitter, Module, Output, Success);

   -- Verify output
   pragma Assert (Success);
end Integration_Test;
```

---

8. Troubleshooting

Common Issues

Issue: Buffer Overflow

Error: `Code_Buffers.Length (Output) = Max_Code_Length`

Solution: Increase `Max_Code_Length` in `stunir-semantic_ir.ads` or split module into smaller units.

Issue: Compilation Fails

Error: `constraint_error: index check failed`

Solution: Ensure all array indices are within bounds. Check `Field_Cnt`, `Func_Cnt`, etc.

Issue: GNATprove Fails

Error: `postcondition might fail`

Solution: Review SPARK contracts and add necessary loop invariants.

Debugging Tips

1. **Enable Debug Output:**
   `ada`
   `Put_Line ("Emitting function: " & Get_Function_Name (Func));`

2. **Check Configuration:**
   `ada`
   `Put_Line ("Architecture: " & Get_Arch_Name (Emitter.Config.Arch));`

3. **Verify IR Module:**
   `ada`
   `pragma Assert (Is_Valid_Module (Module));`

Getting Help

- **Documentation:** `docs/SPARK_EMITTER_ARCHITECTURE.md`
- **Examples:** `examples/outputs/spark/`
- **Tests:** `tests/spark/emitters/`

Appendix A: Type Mappings

C Type Mapping

IR Type	C89/C99 Type	Notes
`i8`	`int8_t`	Signed 8-bit
`i16`	`int16_t`	Signed 16-bit
`i32`	`int32_t`	Signed 32-bit
`i64`	`int64_t`	Signed 64-bit
`u8`	`uint8_t`	Unsigned 8-bit
`u32`	`uint32_t`	Unsigned 32-bit
`f32`	`float`	32-bit float
`f64`	`double`	64-bit float
`bool`	`bool`	C99 only
`string`	`char*`	Null-terminated

Rust Type Mapping

IR Type	Rust Type	Notes
`i32`	`i32`	Signed 32-bit
`u32`	`u32`	Unsigned 32-bit
`f32`	`f32`	32-bit float
`bool`	`bool`	Boolean
`string`	`String`	Owned string
`void`	`()`	Unit type

Appendix B: Command Reference

Build Commands

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

# Build tests
gprbuild -P tests/spark/emitter_tests.gpr

# Clean
gprclean -P tools/spark/stunir_emitters.gpr
```

Verification Commands

```
# Run SPARK verification
gnatprove -P tools/spark/stunir_emitters.gpr --level=2

# Generate proof reports
gnatprove -P tools/spark/stunir_emitters.gpr --report=all
```

END OF GUIDE