Path to STUNIR v1.0.0 Release

Current Version: v0.7.0 (Pre-release) ✅ **WEEK 11 COMPLETE**
Target Version: v1.0.0 (Production Ready)
Target Date: March 7, 2026 (6 weeks)
Current Completion: 95% (+10% in 2 weeks!)

Release Philosophy

STUNIR v1.0.0 will be a **production-ready deterministic code generation framework** with:
- ✅ Multiple verified implementation languages
- ✅ Real function body generation (not stubs)
- ✅ Multi-file project support
- ✅ Formal verification guarantees (SPARK)
- ✅ Comprehensive test coverage

Critical Path to v1.0.0

✅ Week 10: SPARK Multi-File + Rust Function Bodies (COMPLETED - Jan 31)

SPARK Multi-File Support (3 days) ✅ DONE
- [x] Modify `tools/spark/src/stunir_spec_to_ir.adb`
- [x] Add `--spec-root` argument parsing
- [x] Implement `Find_All_Spec_Files` procedure
- [x] Implement `Merge_Functions` procedure
- [x] Test with ardupilot_test (2 files → 11 functions)
- [x] Verify IR output matches Python/Rust

Rust Function Body Emission (4 days) ✅ DONE
- [x] Create `translate_steps_to_rust()` in `ir_to_code.rs`
- [x] Implement step operation handlers (assign, return, call)
- [x] Add Rust type inference system
- [x] Update Rust code templates
- [x] Test with ardupilot_test
- [x] Verify generated Rust compiles with rustc

Deliverable: v0.6.0 release ✅ SHIPPED

✅ Week 11: SPARK Function Bodies (COMPLETED - Jan 31) 🎉

SPARK IR-to-Code Enhancement (7 days) ✅ DONE
- [x] Design SPARK-safe step translation (no buffer overflows)
- [x] Implement `Translate_Steps_To_C` in `stunir_ir_to_code.adb` (~200 lines)

- [x] Add type inference system (`Infer_C_Type_From_Value`)
- [x] Handle all IR step operations (assign, return, nop)
- [x] Update IR parsing to load steps array
- [x] Test with ardupilot_test (11 functions with bodies!)
- [x] Verify generated C compiles with gcc -std=c99

Key Achievement: Complete Feature Parity 🎉
- [x] All 3 pipelines (Python, Rust, SPARK) now generate function bodies
- [x] Type inference working in Ada SPARK
- [x] Local variable tracking implemented
- [x] Step translation validated against Python/Rust

Deliverable: v0.7.0 release ✅ SHIPPED

Week 12-13: Control Flow Support (Feb 15-28)

IR Schema Extension (3 days)
- [ ] Define control flow operations in `stunir_ir_v1` schema
- `if` operation with condition and branches
- `while` operation with condition and body
- `for` operation with iterator and body
- [ ] Update IR validation logic
- [ ] Add control flow examples to test specs
- [ ] Document IR schema changes

Python Implementation (2 days)
- [ ] Add control flow parsing in `spec_to_ir.py`
- [ ] Implement control flow code generation in `ir_to_code.py`
- [ ] Test with control flow specs
- [ ] Verify generated C has correct if/while/for syntax

Rust Implementation (3 days)
- [ ] Add control flow parsing in `spec_to_ir.rs`
- [ ] Implement control flow code generation in `ir_to_code.rs`
- [ ] Test and verify

SPARK Implementation (5 days)
- [ ] Add control flow parsing in SPARK
- [ ] Implement with proof obligations
- [ ] Run GNATprove on control flow code
- [ ] Test and verify

Deliverable: v0.8.0 release

Week 14: Final Testing & Polish (Mar 1-7)

Comprehensive Testing (3 days)
- [ ] Run full test suite on all pipelines
- [ ] Cross-pipeline validation (IR hashes match)
- [ ] Performance benchmarking

- [ ] Memory profiling
- [ ] Security analysis (static analysis, fuzzing)

Documentation (2 days)
- [ ] Update README with v1.0.0 features
- [ ] Create user guide
- [ ] Create API reference
- [ ] Add migration guide from v0.x
- [ ] Update examples

Bug Fixes & Polish (2 days)
- [ ] Fix any issues found in testing
- [ ] Code cleanup and refactoring
- [ ] Final optimization passes
- [ ] Update version numbers

Release (1 day)
- [ ] Create v1.0.0 release notes
- [ ] Tag v1.0.0 in Git
- [ ] Build release binaries
- [ ] Publish documentation
- [ ] Announce release

Deliverable: v1.0.0 release ✅

Detailed Task Breakdown

Task 1: SPARK Multi-File Support

Files to Modify:
- `tools/spark/src/stunir_spec_to_ir.adb`
- `tools/spark/src/stunir_spec_to_ir.ads`

Implementation:

```ada
-- Add to stunir_spec_to_ir.ads
procedure Process_Spec_Directory
  (Spec_Root : String;
   Output_Path : String;
   Result : out Conversion_Result)
  with Pre => Spec_Root'Length > 0 and Output_Path'Length > 0;

-- Add to stunir_spec_to_ir.adb
procedure Process_Spec_Directory
  (Spec_Root : String;
   Output_Path : String;
   Result : out Conversion_Result)
is
  Spec_Files : File_List;
  Merged_IR : IR_Module;
begin
  -- Find all .json files in Spec_Root
  Find_All_Spec_Files(Spec_Root, Spec_Files);

  -- Process first file
  Convert_Spec_To_IR(Spec_Files(1), Merged_IR);

  -- Merge additional files
  for I in 2 .. Spec_Files.Length loop
    Merge_IR_Functions(Merged_IR, Spec_Files(I));
  end loop;

  -- Write output
  Write_IR_JSON(Output_Path, Merged_IR, Result);
end Process_Spec_Directory;
```

Testing:

```
./tools/spark/bin/stunir_spec_to_ir_main --spec-root spec/ardupilot_test --out ir.json
# Expected: 11 functions merged from 2 files
```

Task 2: Rust Function Body Emission

Files to Modify:

- `tools/rust/src/ir_to_code.rs`
- `templates/rust/module.template` (if needed)

Implementation:

```
fn translate_steps_to_rust(steps: &[Step], ret_type: &str) -> String {
    let mut lines = Vec::new();
    let mut local_vars = HashSet::new();

    for step in steps {
        match step.op.as_str() {
            "assign" => {
                let target = &step.target;
                let value = &step.value;

                if !local_vars.contains(target) {
                    local_vars.insert(target.clone());
                    let var_type = infer_rust_type(value);
                    lines.push(format!("    let {}: {} = {};", target, var_type, value
));
                } else {
                    lines.push(format!("    {} = {};", target, value));
                }
            },
            "return" => {
                let value = &step.value;
                lines.push(format!("    return {};", value));
            },
            "call" => {
                // Function call implementation
                // ...
            },
            "nop" => {
                lines.push("    // nop".to_string());
            },
            _ => {
                lines.push(format!("    // UNKNOWN OP: {}", step.op));
            }
        }
    }

    lines.join("\n")
}

fn infer_rust_type(value: &str) -> &'static str {
    if value == "true" || value == "false" {
        "bool"
    } else if value.contains('.') {
        "f64"
    } else if value.starts_with('-') {
        "i32"
    } else {
        "u32"
    }
}
```

Testing:

```
./tools/rust/target/release/stunir_ir_to_code \
  --ir test_outputs/rust_multifile/ir.json \
  --lang rust \
  --out test_outputs/rust_bodies/

rustc --crate-type lib test_outputs/rust_bodies/mavlink_handler.rs
# Expected: Compiles successfully
```

Task 3: SPARK Function Body Emission

Files to Modify:
- `tools/spark/src/stunir_ir_to_code.adb`
- `tools/spark/src/stunir_ir_to_code.ads`

Implementation:

```ada
function Translate_Steps_To_C
  (Steps : Step_Array;
   Ret_Type : String)
  return String
  with Pre => Steps'Length > 0
is
  Result : Bounded_String;
  Local_Vars : Variable_Set;
begin
  for I in Steps'Range loop
    case Steps(I).Op is
      when Op_Assign =>
        if not Contains(Local_Vars, Steps(I).Target) then
          Insert(Local_Vars, Steps(I).Target);
          Append(Result, "  ");
          Append(Result, Infer_C_Type(Steps(I).Value));
          Append(Result, " ");
          Append(Result, Steps(I).Target);
          Append(Result, " = ");
          Append(Result, Steps(I).Value);
          Append(Result, ";");
          Append(Result, ASCII.LF);
        else
          -- Assignment to existing variable
          Append(Result, "  ");
          Append(Result, Steps(I).Target);
          Append(Result, " = ");
          Append(Result, Steps(I).Value);
          Append(Result, ";");
          Append(Result, ASCII.LF);
        end if;

      when Op_Return =>
        Append(Result, "  return ");
        Append(Result, Steps(I).Value);
        Append(Result, ";");
        Append(Result, ASCII.LF);

      when Op_Nop =>
        Append(Result, "  /* nop */");
        Append(Result, ASCII.LF);

      when others =>
        -- Unknown operation
        Append(Result, "  /* UNKNOWN OP */");
        Append(Result, ASCII.LF);
    end case;
  end loop;

  return To_String(Result);
end Translate_Steps_To_C;
```

Formal Verification:

```
cd tools/spark
gprbuild -P stunir_tools.gpr
gnatprove -P stunir_tools.gpr --level=2 --timeout=60
# Expected: All proof obligations discharged
```

Task 4: Control Flow Support

IR Schema Addition:

```
{
  "op": "if",
  "condition": "x > 0",
  "then_branch": [
    {"op": "assign", "target": "result", "value": "1"}
  ],
  "else_branch": [
    {"op": "assign", "target": "result", "value": "0"}
  ]
}
```

Code Generation (C):

```
if (x > 0) {
  result = 1;
} else {
  result = 0;
}
```

Success Metrics

v1.0.0 Acceptance Criteria

Criterion	Target	Current	Status
Functional Pipelines	3/3	3/3	✅
Multi-File Support	3/3	2/3	🚧 67%
Function Body Emission	3/3	1/3	🚧 33%
Control Flow	3/3	0/3	❌ 0%
Test Pass Rate	100%	100%	✅
Code Compiles	Yes	Yes	✅
SPARK Verification	Pass	Partial	🚧
Documentation	Complete	85%	🚧

Quality Gates

Code Quality:
- [ ] All tests pass (100%)
- [ ] No compiler warnings
- [ ] SPARK proof obligations discharged
- [ ] Code coverage > 80%
- [ ] Static analysis clean (clippy, pylint, gnatcheck)

Performance:
- [ ] IR generation: <1s for 100 functions
- [ ] Code emission: <2s for 100 functions
- [ ] Memory usage: <100 MB
- [ ] Binary size: <5 MB (Rust/SPARK)

Documentation:
- [ ] User guide complete
- [ ] API reference complete
- [ ] Examples work
- [ ] README up-to-date
- [ ] CHANGELOG accurate

Risk Mitigation

High-Priority Risks

Risk 1: SPARK Function Bodies Too Complex
- **Mitigation**: Start simple, iterate, consult SPARK community
- **Fallback**: Accept partial implementation, defer full support to v1.1.0
- **Timeline Impact**: +1 week

Risk 2: Control Flow IR Design Flawed
- **Mitigation**: Prototype in Python first, review with stakeholders
- **Fallback**: Defer control flow to v1.1.0, focus on basic function bodies
- **Timeline Impact**: -2 weeks on critical path

Risk 3: Schedule Slips
- **Mitigation**: Weekly checkpoints, re-prioritize if behind
- **Fallback**: Release v0.9.0 as “feature-complete beta”, v1.0.0 delayed 2 weeks
- **Timeline Impact**: +2 weeks total

Alternative Scenarios

Scenario A: Aggressive Timeline (4 weeks)

Skip:
- Control flow support (defer to v1.1.0)
- Haskell pipeline (already deferred)

Focus:
- SPARK multi-file (Week 10)
- Rust function bodies (Week 10)
- SPARK function bodies (Week 11)
- Final testing (Week 12)

v1.0.0: February 21, 2026

Scenario B: Conservative Timeline (8 weeks)

Add:
- Error handling support
- Semantic type system
- Extensive user testing

v1.0.0: March 21, 2026

Scenario C: MVP Timeline (2 weeks)

Minimum Viable Product:
- Python pipeline only (already has function bodies + multi-file)

- Declare v1.0.0-python
- Rust/SPARK as v1.0.0-rust, v1.0.0-spark later

v1.0.0-python: February 7, 2026
v1.0.0-complete: March 21, 2026

Post-v1.0.0 Roadmap

v1.1.0 (April 2026)

- Advanced control flow (switch, break, continue)
- Error handling (try/catch/throw)
- Pointer arithmetic support
- Inline assembly

v1.2.0 (May 2026)

- Generics/templates
- Function pointers
- Variadic functions
- Preprocessor macros

v1.3.0 (June 2026)

- Concurrency primitives (threads, mutexes)
- Async/await support
- Coroutines

v2.0.0 (Q4 2026)

- LLVM IR backend
- GPU code generation (CUDA, OpenCL)
- WebAssembly native target
- JIT compilation support

Team Assignments (If Multi-Person)

Primary Responsibilities

SPARK Developer:
- SPARK multi-file support (Week 10)
- SPARK function bodies (Week 11)
- SPARK formal verification (Week 11-12)

Rust Developer:
- Rust function bodies (Week 10)
- Rust control flow (Week 12)
- Performance optimization (Week 13)

Python Maintainer:
- Python control flow (Week 12)

- Testing infrastructure (Week 13)
- Documentation (Week 14)

Release Manager:
- Version coordination
- CI/CD pipeline
- Release notes
- Binary distribution

Communication Plan

Weekly Status Updates

Every Friday at 5 PM:
- Progress on assigned tasks
- Blockers and risks
- Next week's priorities
- Decision items

Critical Milestones

Week 10 End (Feb 7):
- Demo: SPARK multi-file + Rust function bodies
- Release: v0.6.0

Week 11 End (Feb 14):
- Demo: SPARK function bodies with formal verification
- Release: v0.7.0

Week 13 End (Feb 28):
- Demo: Control flow in all pipelines
- Release: v0.8.0

Week 14 End (Mar 7):
- Demo: Full v1.0.0 feature set
- Release: **v1.0.0** 🎉

Decision Log

Decision 1: Defer Haskell Pipeline

- **Date**: January 31, 2026
- **Rationale**: No toolchain, limited ROI, 3 pipelines sufficient
- **Impact**: -2 weeks saved on critical path
- **Approved**: Yes

Decision 2: Implement Function Bodies in Python First

- **Date**: January 31, 2026
- **Rationale**: Fastest iteration, reference implementation
- **Impact**: +1 week to Rust/SPARK timeline (learning from Python)

- **Approved**: Yes

Decision 3: Control Flow as Stretch Goal

- **Date**: January 31, 2026 (Pending)
- **Rationale**: Basic function bodies are v1.0.0 MVP
- **Impact**: Can defer to v1.1.0 if needed
- **Approved**: TBD

Resources

Documentation

- STUNIR IR Schema: `docs/ir_schema_v1.md`
- Ada SPARK Guide: `docs/SPARK_DEVELOPMENT.md`
- Rust Development: `tools/rust/README.md`
- Python Reference: `tools/README.md`

External References

- Ada SPARK Documentation (https://docs.adacore.com/spark2014-docs/html/ug/)
- Rust Book (https://doc.rust-lang.org/book/)
- C99 Standard (https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf)
- DO-178C (https://en.wikipedia.org/wiki/DO-178C)

Tools

- GNATprove: SPARK formal verification
- rustc: Rust compiler
- GCC: C compiler
- Python 3.11+: Reference implementation

Conclusion

The path to STUNIR v1.0.0 is **clear and achievable** within 6 weeks. The project has strong momentum with:
- ✅ 85% completion
- ✅ 3 functional pipelines
- ✅ Function body emission (1/3 pipelines)
- ✅ Multi-file support (2/3 pipelines)
- ✅ Comprehensive testing

Key to success:
1. Focus on critical path items
2. SPARK as priority (formal verification)
3. Defer nice-to-haves if needed
4. Maintain quality standards
5. Test continuously

Confidence Level: **80%** for March 7 target

Document Version: 1.0
Last Updated: January 31, 2026
Next Review: February 7, 2026 (Week 10)