Lero THE IRISH SOFTWARE RESEARCH CENTRE

LLVM LEON Backend Final Presentation

Emil Vassev

December 6, 2016

University of Limerick Ollscoil Luimnigh
Trinity College Dublin
DCU
UCD Dublin
Ollscoil na Gaillimh NUI Galway
Dundalk Institute of Technology
UCC
Maynooth University National University of Ireland Maynooth

Outline

Introduction

- Project Objectives
- LLVM & Problem Statement

Results of the Development Activities

- Software Requirements
- Software Design
- Implementation
- Unit Tests
- Test Suite
- Code Check-ins

Conclusion

- Overall Assessment
- Challenges
- Future Improvements

LLVM LEON Backend: Project Objectives

Build an LLVM Backend for LEON2/3/4-FT :

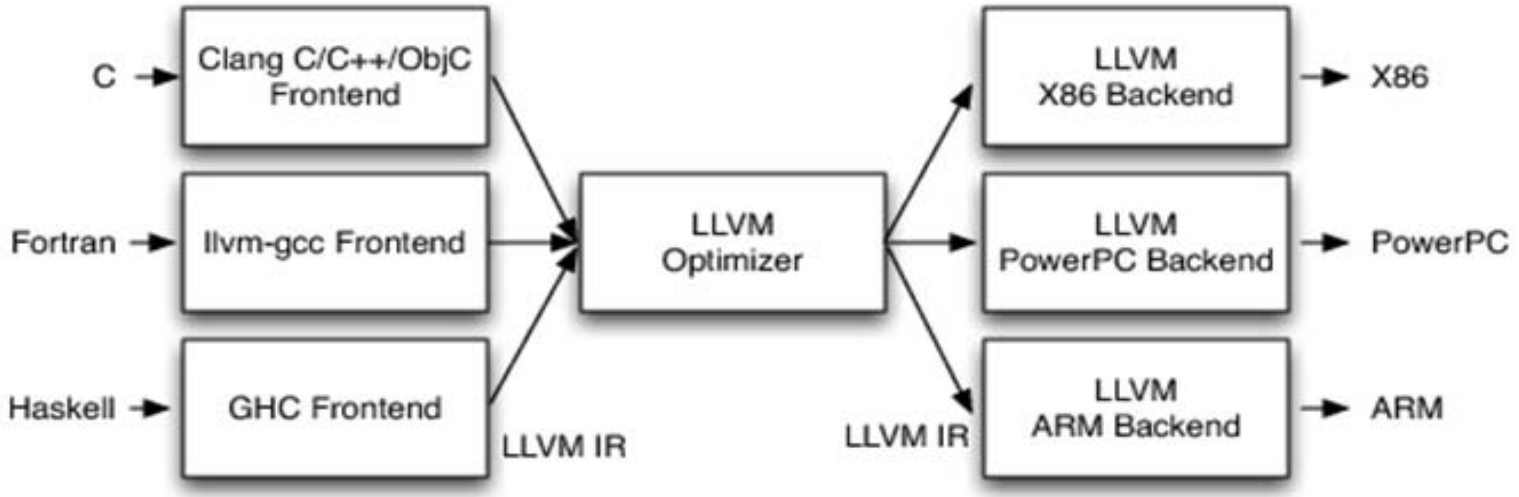
- ✓ specify the upgrades needed in the LLVM SPARC backend to support the LEON2/3/4-FT processors
- ✓ design and develop the LLVM backend for LEON2/3/4-FT processors
- ✓ implement a test suite running on the LEON3/4-FT architecture - comparison between LLVM and GNU GCC with respect to performance, accuracy, etc.
- ✓ study and prototype novel techniques to improve the determinism of application execution and WCET reduction
- ✓ identification of new use cases enabled by the LLVM LEON backend

LLVM Framework

Compiler stages:

- ✓ parse source language to an IR
- ✓ transformations of the IR
- ✓ generation of machine code from IR

LLVM - modular and reusable libraries built around the LLVM IR



LLVM LEON Backend: Problem Statement

LLVM LEON Backend:

- × cope with existing LLVM Framework and LLVM SPARC backend
- × open source, code needs to be checked-in
- × support to LEON 2 FT, LEON 3 FT, LEON 4 FT
- × different LEON instruction latencies, errata, SPARC architecture

Solution:

- ✓ subtargets for the current SPARC backend
- ✓ processor itineraries (LEON 2/3/4) and extended instruction set
- ✓ subtarget features to parameterize the LEON backend
- ✓ erratum fixes and optimizations

LEON Backend Results: Software Requirements

Elicitation and specification of backend requirements:

- ✓ multiple iterations – 80 refined requirements
- ✓ all requirements are addressed by the backend design and unit test plan
- ✓ traceability b
- ✓ traceability b

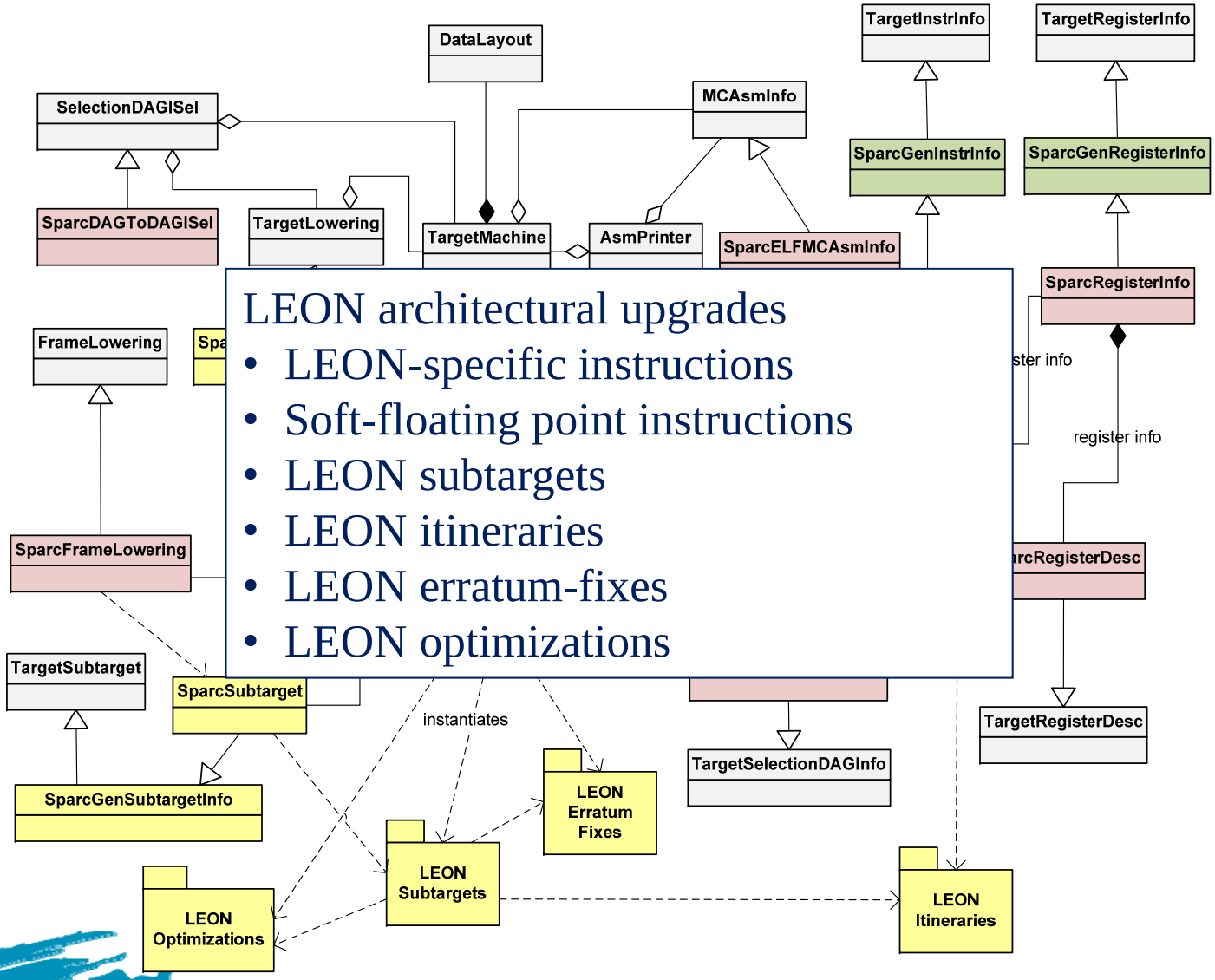
Performed studies:

- Implementation trade-off
- SPARC backend
- SPARC lowering
- SPARC ABI
- LEON Errata
- Cache and multicore optimization

Requirement #LBR21

- *Description*: For … e %i0 (the caller's %o0) to return the result.
- *Exception condition*: This is not required where a procedure is implemented as an optimized lead procedure.
- *Responsible*: Task 120, Task 210
- *Verification*: code inspection, empirical study

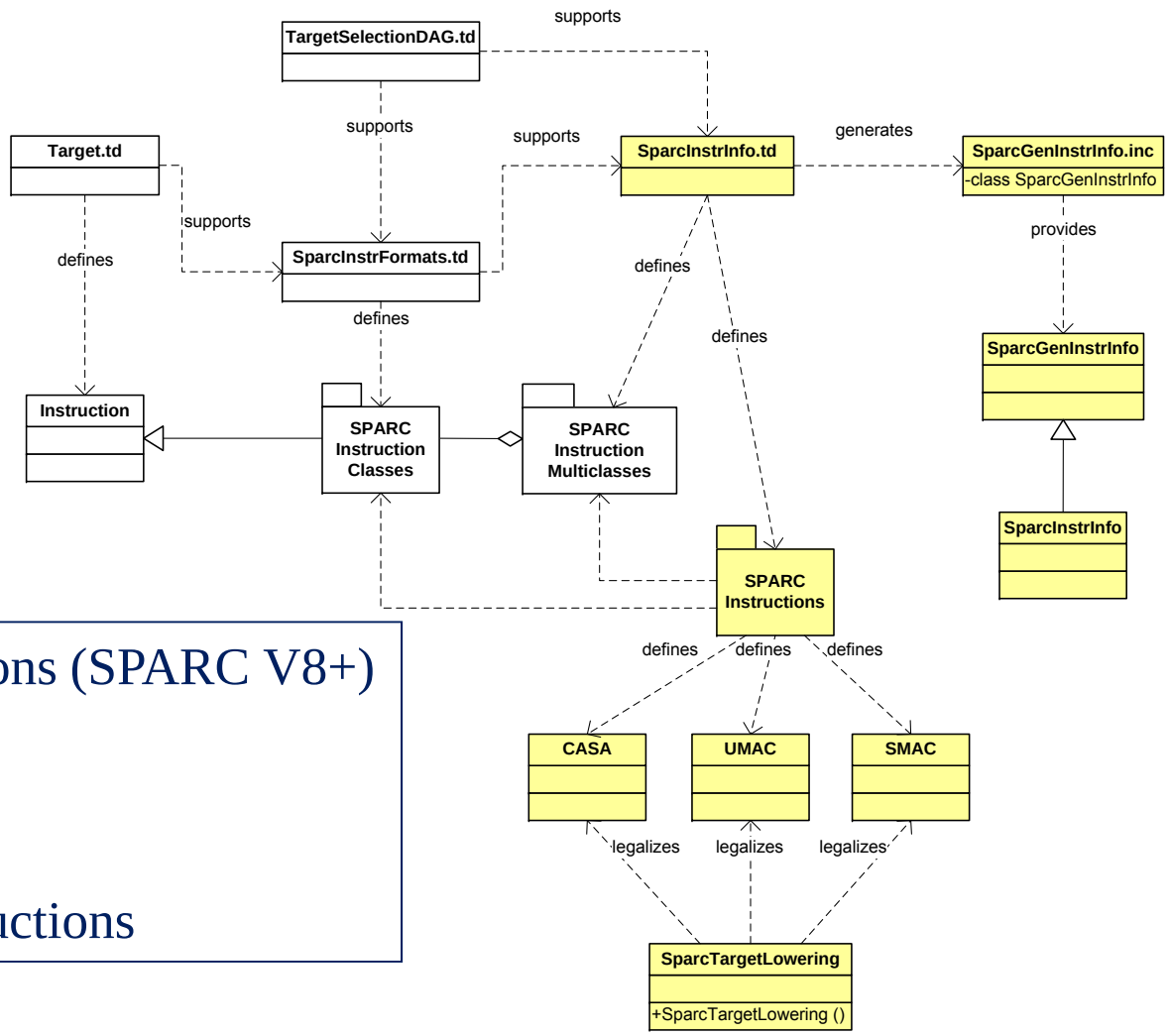
LEON Backend Results – Software Design



LEON architectural upgrades

- LEON-specific instructions
- Soft-floating point instructions
- LEON subtargets
- LEON itineraries
- LEON erratum-fixes
- LEON optimizations

LEON Backend Results – Software Design



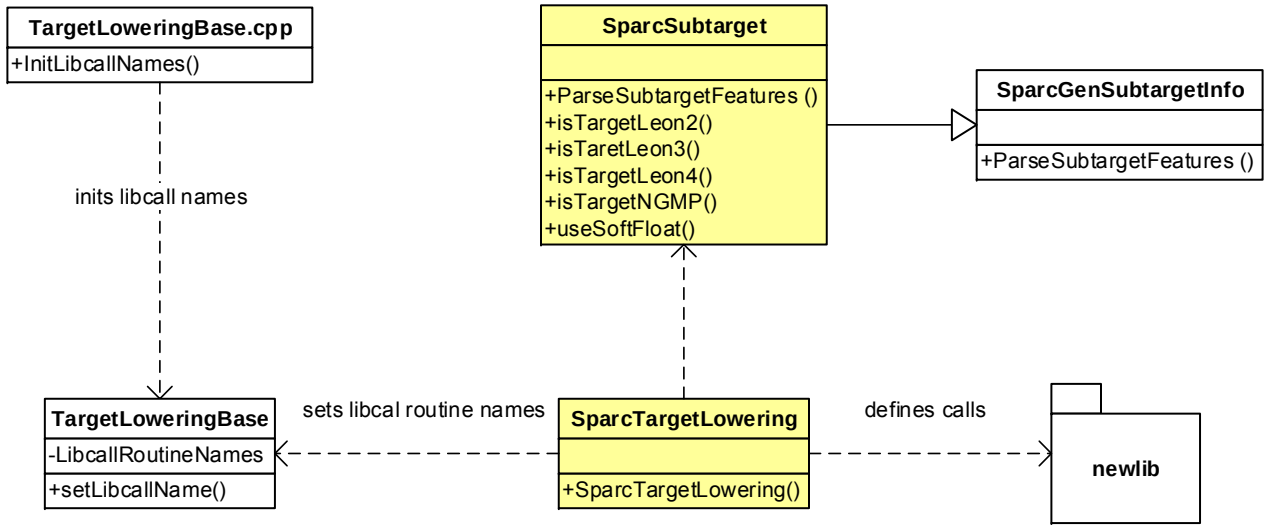
LEON specific instructions (SPARC V8+)

- UMAC/SMAC
- CASA

- + all SPARC V8 instructions

LEON Backend Results – Software Design

Software FP instructions
- software FP library calls
- mathematical library (libm), e.g, Newlib



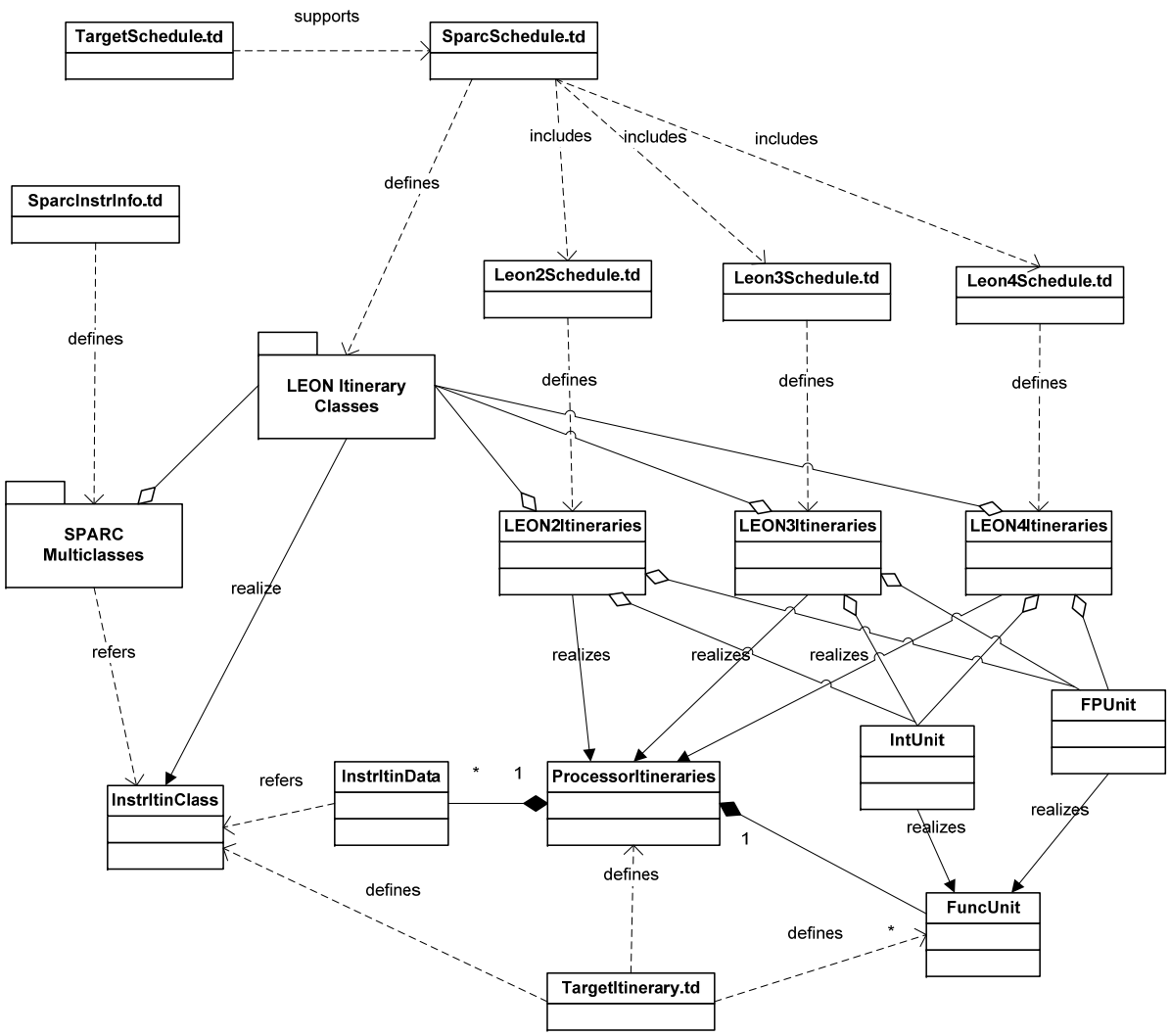
LEON Backend Results: Software Design

LEON subtargets

- ✓ leon2 - generic LEON 2 FT, LEON 2 itineraries, no erratum fixes
- ✓ at697e - AT697E / LEON 2 FT, LEON 2 itineraries, erratum fixes for LEON 2 AT697E
- ✓ at697f - AT697F / LEON 2 FT, LEON 2 itineraries, erratum fixes for LEON 2 AT697F
- ✓ leon3 – generic LEON 3 FT, LEON 3 itineraries, UMAC and SMAC instructions, no erratum fixes
- ✓ ut699 - UT699 / LEON 3 FT, LEON 3 itineraries, UMAC and SMAC instructions, erratum fixes for LEON 3 UT699
- ✓ gr712rc - GR712RC / LEON 3 FT, LEON 3 itineraries, UMAC and SMAC instructions, no erratum fixes, CASA instruction
- ✓ leon4 - generic LEON 4 FT, LEON 4 itineraries, UMAC and SMAC instructions, no erratum fixes, CASA instruction
- ✓ gr740 - GR740 / LEON 4 FT, LEON 4 itineraries, UMAC and SMAC instructions, no erratum fixes, CASA instruction

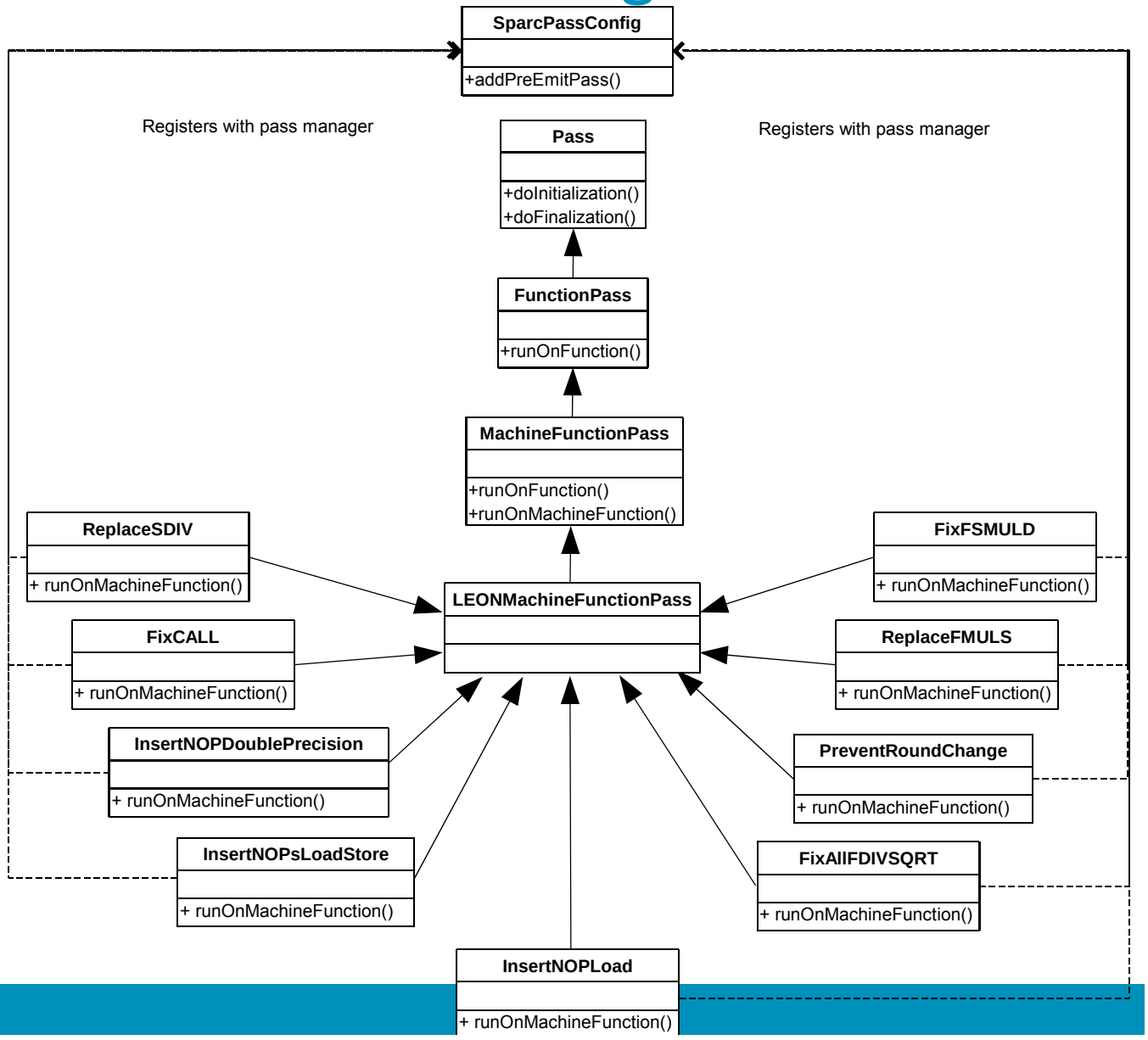
LEON Backend Results – Software Design

LEON itineraries



LEON Backend Results – Software Design

LEON erratum fixes



LEON Backend Results: Software Design

LEON Optimizations

New optimizations:

- ✓ Delay Slots Optimization

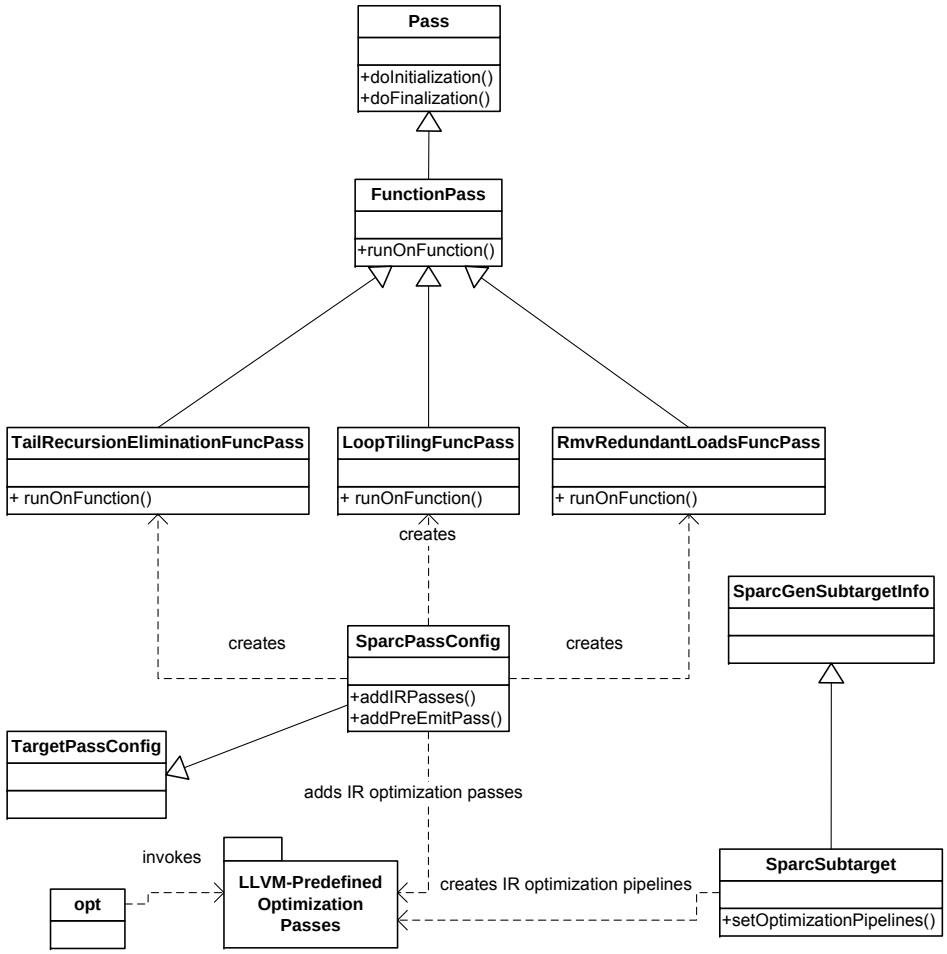
  (partially realized by DelaySlotFiller)

- ✓ Delay Slot after Unused Modified Register Optimization

  (realized for FixAllFDIVSQRTPass erratum fix)

Note:

Initially planned optimizations were not implemented: the focus was on the correctness of the generated code.

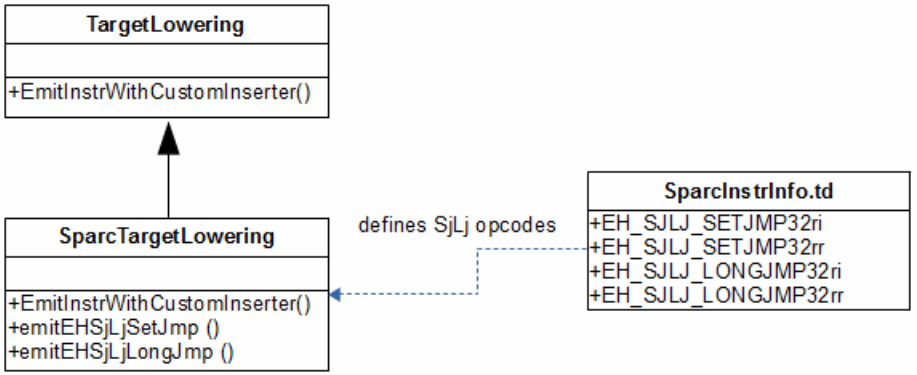


LEON Backend Results: Software Design

SPARC backend patches/fixes

Builtin intrinsics:

- ✓ __builtin_setjmp
- ✓ __builtin_longjmp



LEON Backend Results: Implementation

LEON backend implementation:
- multiple iterations (impl-UT-testing)
- followed and changed the design
- impacted by code check-in

- ✓ LEON subtargets
- ✓ Missing instructions (LEON-specific, coprocessor instr., etc.)
- ✓ LEON itineraries
- ✓ LEON subtarget features
- ✓ LEON erratum fixes
- ✓ LEON optimizations
- ✓ SjLj inrinsics
- ✓ Unplanned fixes and patches

LEON Backend Results: Unit Tests

LLVM UTs developed to test particular backend features:
-UT plan
-UT implementation

UTs – sc

UTs targ

- ✓ lower
- ✓ imple proce
- ✓ genel
- ✓ encoc

every sing

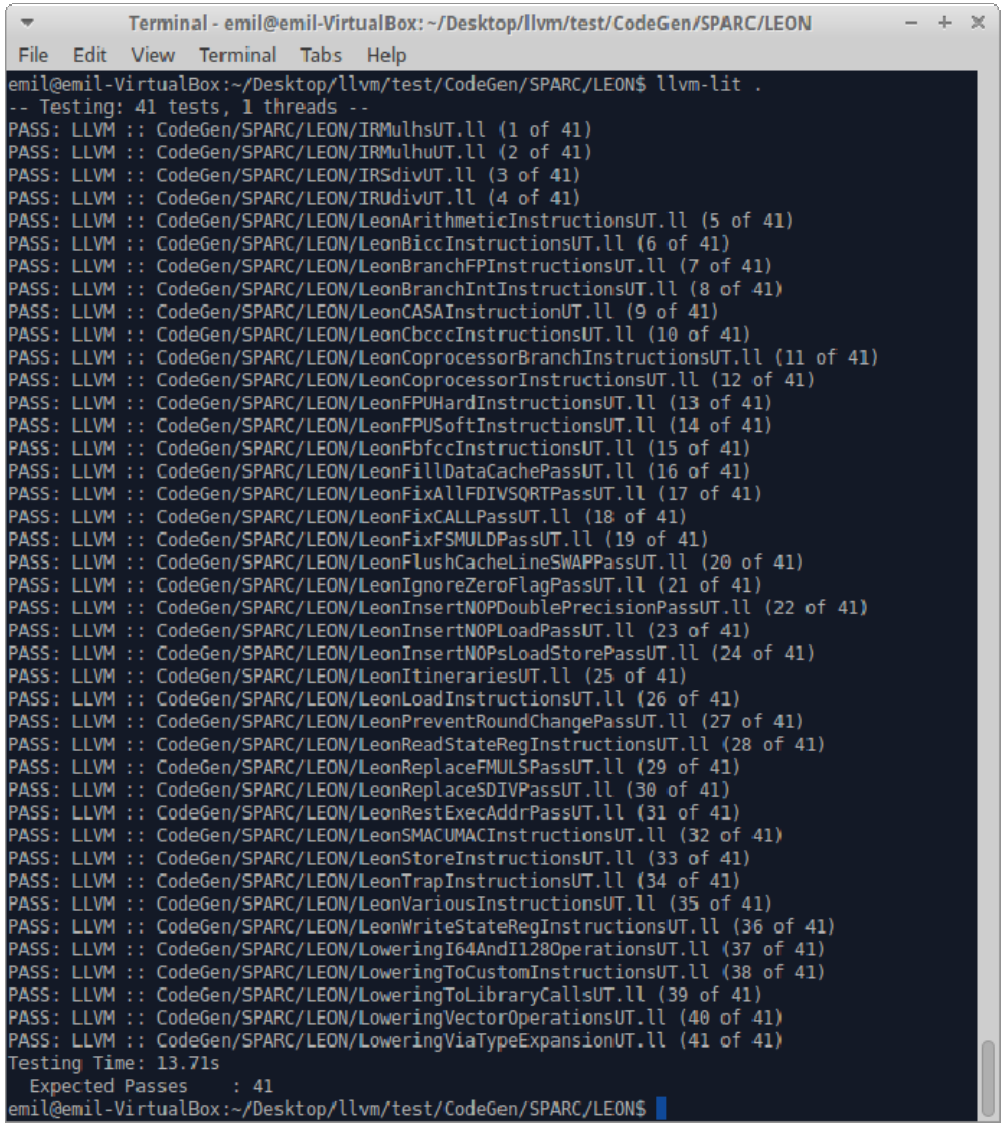
```
; RUN: llc < %s -march=sparc -mcpu=leon2    | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=at697e   | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=at697f   | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=leon3    | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=ut699    | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=gr712rc  | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=leon4    | FileCheck %s
; RUN: llc < %s -march=sparc -mcpu=gr740    | FileCheck %s

; testing ADD - Add
; CHECK-LABEL: test_add
; CHECK:       add
define i8 @test_add(i8 %a, i8 %b) nounwind {
  %1 = and i8 %a, %b

  ret i8 %1
}
```

nd LEON 3

LEON Backend Results: Unit Tests

UT Results – over 1000 test cases in 39 UT files

- ✓ developed and executed in multiple iterations
- ✓ final result: all the unit tests passed, confirming the correctness of the tested features



LEON Backend Results: Test Suite

Cobham Gaisler Test Suite:
- test the correctness of the LLVM LEON backend
- measure performance compared to GCC LEON backend

Includes over 6600 tests written in C/C++

Over 6500 tests coming from GCC

Targets LEON 3 and LEON 4:

- ✓ instruction generation
- ✓ erratum fixes

Tests executed on two LEON systems:

- ✓ LEON 3
- ✓ quad core LEON 4 with a L2 cache

LEON Backend Results: Test Suite

Test Suite Results

- ✓ Testing performed in multiple iterations
- ✓ Final result: only 11 GCC-specific tests do not pass out of over 6600 tests (over 1100 were not passing at the start of the project)
- ✓ The 11 non-passing tests are mostly related:
  - to the use of the GCC's __vector language extension of C++
  - to the use of inline assembly code with GCC syntax
- ✓ All errata-related tests successfully passed

LLVM LEON backend is a competitive alternative of GCC LEON backend

LEON Backend Results: Code Check-ins

Upload the LEON backend code into the LLVM Repository

Significant check-ins in the project:

- ✓ multiple iterations – upload-review-modification
- ✓ over 15 check-ins for the LEON backend implementation:
  - ➢ planned implementation
  - ➢ code fixes and patches
- ✓ problems mainly related to processing inline assembly and erratum fixes

LEON Backend Overall Assessment

LEON backend - complete solution and good competitor of GCC LEON backend

Multiple-iteration software process:

- ✓ comprehensive requirements specifications
- ✓ efficient and comprehensive backend design and implementation
- ✓ comprehensive and effective unit test plan
- ✓ efficient and exhaustive integration testing

Comprehensive and regular reporting

Three major review meetings

LEON Backend Challenges

LEON backend – extremely challenging project

- ✓ initial learning curve
  - ➢ learning about the SPARC instruction set
  - ➢ learning curve for the LLVM system
- ✓ TableGen language
- ✓ IR language
- ✓ coding support
- ✓ coding cycle time:
  - ➢ LLVM compilation time
  - ➢ test cycle time

LEON Backend Future Improvements

LEON backend – good solution with only few possible improvements

- ✓ LLVM LEON backend:
  - implement LEON itineraries following the NEW way of implementation
  - improvements in efficiency of the generated code
- ✓ Test suite:
  - categorization of the different test programs
  - by measuring the coverage of the compiler code
  - gathering more information from the execution environment
  - using the TSIM simulator
  - new runtimes (currently RTEMS)

LEON Backend Future Improvements

Improvements based on enabled use cases

- ✓ visual modeling for LLVM
- ✓ automatic code generation for LLVM
- ✓ automatic verification with model checking for LLVM
- ✓ static code analysis for LLVM
- ✓ automatic generation of unit tests for LLVM
- ✓ requirements specification for LLVM
- ✓ software design for LLVM
- ✓ visual modeling and code generation to C++/LLVM's IR
- ✓ formal specification, verification and code generation to C++/LLVM's IR
- ✓ specifying autonomy behavior and code generation to C++/LLVM's IR
- ✓ programming by mixing different programming languages and code generation to LLVM's IR

Summary

LLVM LEON backend

- LLVM Backend for LEON 2, LEON 3, and LEON 4
- Upgrade to LLVM SPARC backend
- Extensive set of various features
- Unit testing with LLVM UTs and LIT
- Integration testing with Cobham Gaisler test suite (6600+ tests)
- Code check-ins to LLVM repository

Science Foundation Ireland

Ireland's European Structural and Investment Funds Programmes 2014-2020
Co-funded by the Irish Government and the European Union

European Union
European Regional Development Fund

SOUTHERN Regional Assembly
Promoting our Region