The most important thing we build is trust

ADVANCED ELECTRONIC SOLUTIONS | AVIATION SERVICES | COMMUNICATIONS AND CONNECTIVITY | MISSION SYSTEMS

Definition and Design of software components for LEON3FT Microcontroller and LEON-REX Instruction Set Architecture

TEC-ED & TEC-SW Final Presentation Days - December 2016

Daniel Cederman (Presenter)

Daniel Hellström, Arne Samuelsson

Jorge Lopez Trescastro (ESA Technical Officer)

Overview

Quick summary

- LEON-REX: 16-bit instructions for the LEON3/4 processors
  - More compact code
  - Internally developed at Cobham Gaisler
- Available in the LEON3FT microcontroller (GR716)
  - Funded by ESA
  - Developed by Cobham Gaisler

- New compiler toolchain required
- Goal of activity: Define the design of new toolchain

- Complete compiler toolchain for LEON-REX available
- Will be part of BCC2 with BSP for GR716

Overview

Activity

- One year activity (Jan 2016 – Dec 2016)

- MTR-1 (Mar 2016)
  - Evaluation of LEON-REX and listing implications to other parts of compilation toolchain [D11]
- MTR-2 (Jun 2016)
  - New ABI standard [D12]
  - Assembly language reference [D13]
  - High level requirements for all part of compilation toolchain [D14]
- AR (Dec 2016)
  - Design documents for Binutils, Newlib, and LLVM [D15, D16, D18, D19]
  - Test plans [D17, D20]
  - Final report

Overview

Presentation outline

- LEON-REX instruction set
- Toolchain components - ABI, LLVM Backend, Linker
- Evaluation
- Conclusions

LEON-REX

LEON-REX

New 16-bit instruction set

- 16-bit versions of common instructions
  - add %i0, %i2, %i0 <-> radd %i2, %i0

    [0xb0, 0x06, 0x00, 0x1a] [0xa0, 0x0a]

- Only two registers and no immediates for most instructions
- Can only access half of available registers

- Designed for compact code at low hardware cost
  - LEON-REX instructions translated to standard SPARC
  - Does not affect maximum frequency
  - Available for LEON3/4
  - Compatible with SPARC ABI
- Developed by Cobham Gaisler

LEON-REX

REX mode and standard SPARC

- New instructions only available in REX mode

- All functions start in standard SPARC mode
- Enter REX mode in function prologue with SAVEREX or ADDREX
  - save %sp, -96, %sp -> saverex %sp, -96, %sp
  - add %sp, -80, %sp -> addrex %sp, -80, %sp
- Can use REX instructions until end of function

- Most 32-bit instructions available (including floating point)
- Exceptions are: UNIMP and SETHI
- Immediate field only 7 bits instead of 13

LEON-REX

Function calls

- CALL instruction available in REX mode
- No delay slot

- Call sets %o7 to PC-4
  - Returns go to %o7+8 to skip delay slot, -4 negates this

- Least significant bit in PC signifies REX mode
  - Unused since instructions are 16-bit aligned
  - Returning from function reenables REX mode or SPARC mode depending on return address

LEON-REX

Reduced register set

- All registers still accessible using 32-bit instructions in REX mode
- Values can be moved between registers using 16-bit RMOV

- %O0-O3
  - Generic use and outgoing parameters
- %I0-I7
  - Generic use and incoming parameters
  - %i6 holds the frame pointer
  - %i7 holds the return address
- %L4-L7
  - Generic use
- Loads and stores relative %sp possible

Instruction set overview

16-bit instructions

- Arithmethic
  - add, sub, and, sll, …
  - Not multiplication/division
- Bitmask operations
  - Set, clear, invert, or test individual bit
  - Mask away upper bits
- Negation
  - Negate or invert register

- Register to register copy
  - All registers accessible
- Branch instructions
  - Integer and floating point (±256 bytes)

LEON-REX

Instruction set overview

- Return instructions
  - RRETREST -> jmpl %i7+8, %g0 ; restore
  - RRETL -> jmpl %o7+8; nop

- Load/store with single register addressing
  - rld [rs], rd
  - For 8, 16, 32, 64, single, double
- Autoincrementing load/store
  - rld [rs], rd -> ld [rs], rd ; add rs, 4, rs

- Load/store with fixed register plus immediate
  - Supported registers are %o0, %i0, %sp, and %fp
  - 32-bit values only

LEON-REX

Instruction set overview

- Miscellaneous
  - Push and pop to stack
  - Software trap 0-7
  - Leave REX mode
  - Get value of PC

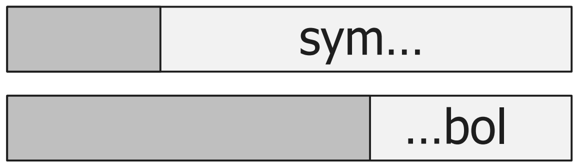
LEON-REX

Instruction set overview

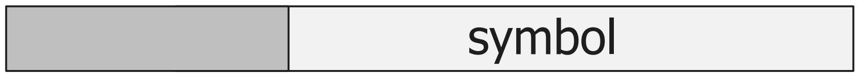
- 48-bit instructions

Instruction format

- `sethi %hi(symbol), %i0`
- `or %i0, %lo(symbol), %i0`



- `rset32 symbol, %i0`



- RSET32 and RSET32PC
- RLD32 and RLD32PC

Summary

LEON-REX instruction set

- 16-bit versions of many instructions
- Requires entering REX mode
- Most 32-bit instructions still available

- Available for LEON3/4

- Updated assembly language reference [D1 2]
- Available as technical note at Cobham Gaisler webpage

ABI

ABI

Application binary interface

- LEON-REX was designed to be compatible with the SPARC ABI
- All functions begin in standard SPARC mode

- Only minor extensions needed
- New code examples
  - Prologue and epilogue generation
- New linker relocations
- Two possible methods for returning structs

- New ABI standard [D12]

LLVM BACKEND

LLVM

LEON-REX Backend

- Getting more popular as an alternative to GCC
- Modern code base – modular and easy to read C++ code
- BSD license – not GPL
- Has a SPARC backend
  - Improved and extended with LEON specific features in concurrent ESA activity
- Good choice for implementing a new architecture

- Lacks GCC's long heritage
- Generates larger binaries than GCC for SPARC

- Used to implement assembler and C compiler backend
- Backend specification and design document [D18, D19]

LLVM

Compiling for LEON-REX

- Compiling application
  - sparc-bcc-elf-clang -qbsp=gr716 -mcpu=leon3 **-mrex** *.c

- Compiling object file
  - sparc-bcc-elf-clang -c -mcpu=leon3 **-mrex** hello.c

- Compiling function

```
__attribute__((target("rex")))
void hello(void) {
...
}
```

BINUTILS

Binutils

Overview

- Assembler
  - Internal LLVM assembler used
- Utilities
  - Use llvm-objdump instead of objdump
- Linker
  - Standard relocations possible with existing relocation types
  - New types for position independent code and thread local storage

- Design document for LEON-REX components in Binutils [D15]
- Test plan for Binutils [D17]

GRMON2

LEON-REX support

- The debug monitor software for LEON processors

- All standard functionality available also in REX mode

- Disassemble REX code
- Instructions trace
  - Shows actual SPARC instructions performed
  - Marked as running in REX mode
- Hardware breakpoints

EVALUATION

Evaluation

Correctness tests

- Unit tests

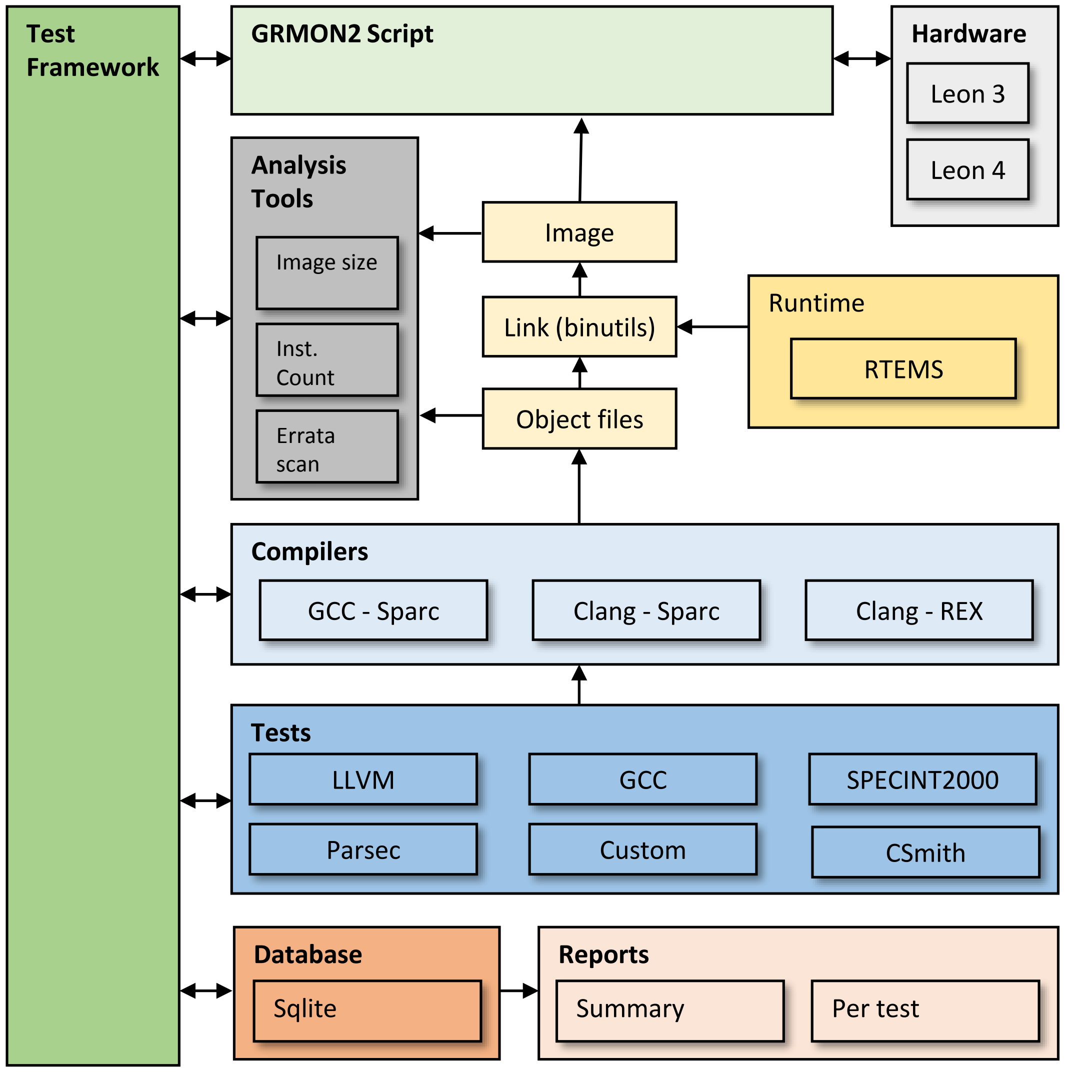
- Autogenerated code
  - Csmith
  - LLVM-stress

- RTEMS and Newlib

- LEON Toolchain test suite
  - From ”LLVM Compiler for in flight SW development and validation process” activity

LEON Toolchain test suite

Overview



LEON Toolchain test suite

Test programs

- LLVM
  - 146 benchmarks
  - 374 test programs
- GCC
  - 3919 Compile only
  - 2071 test programs
- Parsec
  - Open source
  - Multi-core
  - 9 benchmarks
- SPECINT2000
  - Commercial
  - 4 benchmarks

Correctness

Results

- 6547/6600 test programs compiled and executed correctly
- Failing tests
  - Tests that fail on standard Sparc for LLVM
    - Vector
    - GCC specific
  - C++ stack unwinding

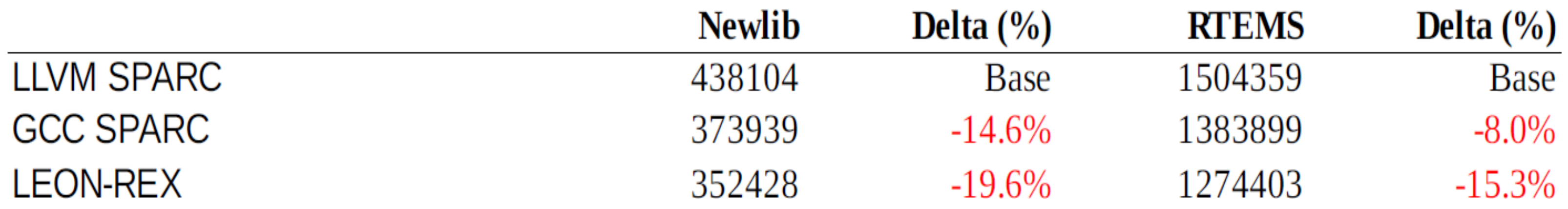
- Newlib and RTEMS compiles successfully

- Newlib C library test suite run successfully
- RTEMS test suite show only 15 of 525 failing
  - RTEMS master (Nov 2016)

Results

Object file size comparison

- Newlib and RTEMS
- Comparing LLVM SPARC, GCC SPARC, LEON-REX
- Total size of all text sections in object files



	Newlib	Delta (%)	RTEMS	Delta (%)
LLVM SPARC	438104	Base	1504359	Base
GCC SPARC	373939	-14.6%	1383899	-8.0%
LEON-REX	352428	-19.6%	1274403	-15.3%

CONCLUSIONS

Conclusions

Current status

- BCC2
  - Will include compiler for LEON-REX
  - BSP for GR716 for both GCC and LLVM
- GRLIB
  - Available for LEON3 and LEON4
- Official LLVM repository
  - First patches submitted
  - Hard to get attention for new backend

Conclusions

Conclusions

- The project was completed on time with all deadlines met

- Fully working implementation of a LEON-REX compiler
- ABI compatibility makes porting code to LEON-REX easy

- Possible to compile and run RTEMS and Newlib with LEON-REX
- 20% code size decrease compared to LLVM standard SPARC
- Big step forward generating size efficient code for the GR716

- LLVM generates larger object files than GCC for SPARC
- Room for improvements both for SPARC and LEON-REX backend

THANK YOU FOR LISTENING!

EXTRA SLIDES

LEON-REX LLVM Backend

- New instruction formats

```
// For 16-bit REX instruction with rs and rd operands
class RexFR<bits<2> opval, bits<4> rop3val, bits<1> rop3lval,
            dag outs, dag ins, string asmstr, list<dag> pattern>
    : InstRex16<outs, ins, asmstr, pattern> {
  bits<4> rs;
  bits<4> rd;
  let op           = opval;
  let Inst{13-10} = rd;
  let Inst{9}      = 0;
  let Inst{8-5}    = rop3val;
  let Inst{4}      = rop3lval;
  let Inst{3-0}    = rs;
}
```

LEON-REX LLVM Backend

- New instructions

```
let Constraints = "$rd = $rsc" in
def RADD : RexFR<0b10, 0b0000, 0, (outs RexIntRegsOp:$rd),
            (ins RexIntRegsOp:$rs, RexIntRegsOp:$rsc),
            "radd $rs, $rd", [(set i32:$rd, (add i32:$rsc, i32:$rs))]>;
```

LEON-REX LLVM Backend

- New reduced register sets
- New code generation patterns

- New prologue and epilogue generation
- Single register addressing format
- Single immediate addressing format

- Post-incrementing load and store instructions

- Assignment of constants
  - RSET instructions instead of SETHI

LEON-REX LLVM Backend

COBHAM

- Support for 48 and 16 bit instructions
- Frame index elimination
- LEON-REX encoding of standard SPARC format 3 and floating point instructions
- Support long and short branches
- Enable or disable REX for individual functions via attributes
- Backend specification [D18]
- Backend design document [D19]

LLVM SPARC vs REX

Compiled with -Os

specint2000

LLVM

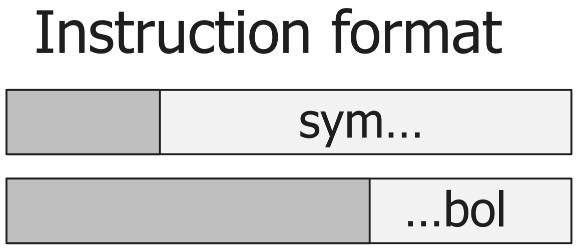
Parsec

		LLVM SPARC	LLVM REX	Delta (%)
vpr	Total execution time (s)	512	568	11.0%
	Code size (bytes)	130494	114912	-11.9%
	L1 instruction cache miss	767121	735217	-4.2%
gcc	Total execution time (s)	89	94	5.7%
	Code size (bytes)	1386967	1010661	-27.1%
	L1 instruction cache miss	83843201	54369859	-35.2%
parser	Total execution time (s)	173	190	10.1%
	Code size (bytes)	97960	70112	-28.4%
	L1 instruction cache miss	785602	467636	-40.5%
espresso	Total execution time (s)	61	67	10.4%
	Code size (bytes)	142034	98622	-30.6%
	L1 instruction cache miss	1827268	2304666	26.1%
x264	Total execution time (s)	436	664	52.2%
	Code size (bytes)	447519	421889	-5.7%
	L1 instruction cache miss	72875937	80894328	11.0%

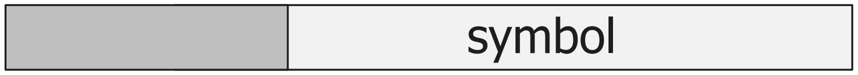
Binutils

Linker relocations - Standard

- Standard relocations can be done using existing type



- sethi %hi(symbol), %i0
- or %i0, %lo(symbol), %i0
- R_SPARC_HI22/R_SPARC_LO10



- rset32 symbol, %i0
- R_SPARC_32

- No change in relocation for calls

Binutils

Linker relocations - Position independent code

- Symbol offsets stored in Global offset table by runtime linker

- Standard SPARC

```
call .L1
sethi %hi(_GLOBAL_OFFSET_TABLE_), %i0
.L1:    or %i0, %lo(_GLOBAL_OFFSET_TABLE_), %i0
add %i0, %o7, %i0

ld [%i0 + %got(foo)], %i1
```

- REX

```
rset32pc _GLOBAL_OFFSET_TABLE_, %i0

rset21 %got21(foo), %i1
```

Binutils

Linker Relocations - Thread local storage

- General Dynamic
- Local Dynamic
- Initial Executable
- Local Executable

- Sample sequence on standard SPARC:

```
sethi %tle_hix22(levar), %i0 -- R_SPARC_TLS_LE_HIX22
xor %i0, %tle_lox10(levar), %i0 -- R_SPARC_TLS_LE_LOX10
ld [%g7+%i0], %i0
```

- LEON-REX

```
rset32 %tle_32(levar), %o1 -- R_SPARC_TLS_LE_REX_32
ld     [%g7 + %o1], %o0
```