

# Small Device C Compiler

<http://sdcc.sourceforge.net>

Philipp Klaus Krause

2025-08-16

# Table of Contents

**1** The Small Device C Compiler

**2** Recent Progress

**3** Near Future Plans

# Table of Contents

**1** The Small Device C Compiler

2 Recent Progress

3 Near Future Plans

## 8-Bit architectures

- In between low-end (4-bit) and high-end (32- and 64-bit microcontrollers).
- Typically programmed in C
- Devices cost about 1¢ to 1 €
- Data memory typically in the range of a few B to a few KB
- Program memory typically a few KB
- Market dominated by proprietary architectures, and ancient architectures implemented by many vendors

- Standard C compiler: ISO C90, ISO C99, ISO C11, ISO C23, ISO C2Y
- Freestanding implementation or part of a hosted implementation
- Supporting tools (assembler, linker, simulator, ...)
- Works on many host systems
- Targets various 8-bit architectures, has some unusual optimizations that make sense for these targets
- Latest release: 4.5.0 (2025-01-28)
- User base: embedded developers and retrocomputing/-gaming enthusiasts
- Also used in downstream projects (z88dk, gbdk, devkitSMS)

# Ports

- MCS-51, DS390
- STM8
- HC08, S08
- PDK13, PDK14, PDK15
- MOS 6502, WDC 65C02
- Z80, Z80N, Z180, eZ80, TLCS-90, SM83, Rabbits, R800
- f8
- (PIC14, PIC16)

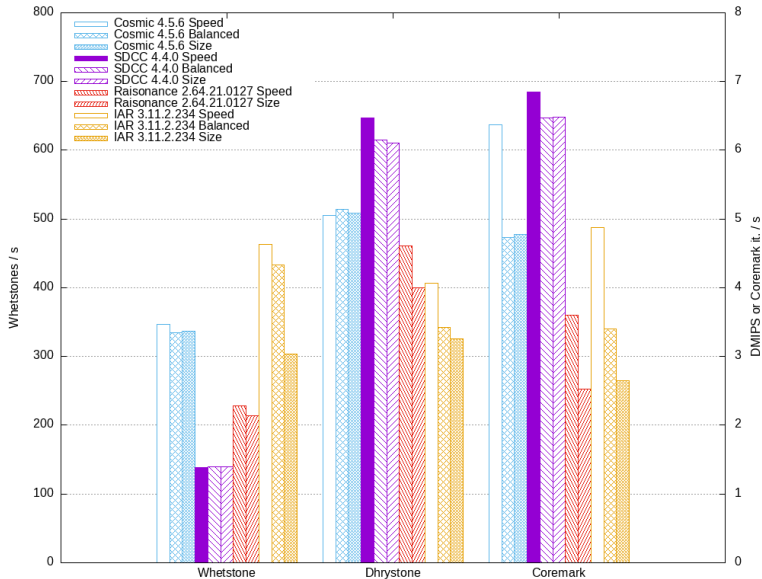
# Optimal Register Allocation in Polynomial Time

- Register allocator based on graph-structure theory
- Optimal register allocation in polynomial time
- Flexible through use of cost function
- Provides substantial improvements in code quality
- Slow compilation for targets with many registers
- Compilation speed / code quality trade-off:
  - max-allocs-per-node

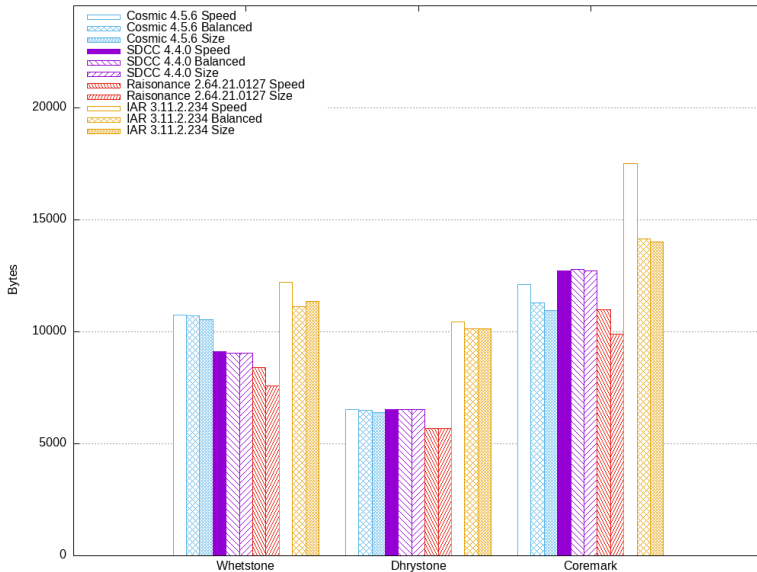
# Bytewise Register Allocation and Spilling

- Decide on the storage of variables bytewise
- Decide for each individual byte in a variable whether to store it in memory or a register
- Consider any byte of any register as a possible storage location

# SDCC vs. non-free compilers: STM8 Benchmark scores



# SDCC vs. non-free compilers: STM8 Code size



# Regression testing

- Regression testing of nightly snapshots
- $\approx$  32000 tests (thrice as many as 2020) compiled and executed on simulators
- Tests mostly from fixed bugs and from GCC
- Targets architectures: MCS-51, DS390, Z80, Z180, eZ80, Rabbit 2000, Rabbit 2000A, Rabbit 3000A, SM83, TLCS-90, HC08, S08, STM8, PDK14, PDK15
- Host OS: GNU/Linux, macOS, “Windows” (cross-compiled on GNU/Linux, tested via wine)
- Host architectures: x86, amd64, ppc64, aarch64

# Project

- Bug and feature request trackers at SourceForge
- Mailing lists sdcc-user and sdcc-devel
- svn repository (and git mirror) at SourceForge
- Wiki
- Compile farm for nightly regression testing
- Mostly volunteer work, but has received some external support (DFG, Prototype Fund, NLnet, hardware vendors)

# Challenges

The C standard is evolving rapidly, SDCC needs to keep up. Many target architectures of SDCC have been discontinued. How can SDCC stay relevant outside of the retrocomputing niche?

- STM8 - SDCC is doing well, and ST put many STM8 devices back to active
- MCS-51 - SDCC port needs major work to be competitive vs. Keil
- PDK - Not an easy target for a C compiler
- S08 - SDCC port needs some work, hard to get into dev community
- eZ80 - hard to get into dev community
- TLCS-870/C1 - no SDCC port yet, hard to get into dev community
- f8

# Table of Contents

1 The Small Device C Compiler

2 Recent Progress

3 Near Future Plans

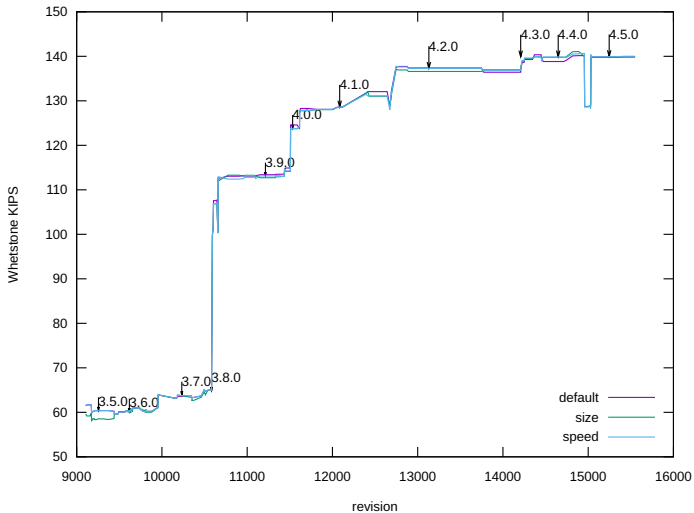
# Standard compliance

- struct / union parameters and return types
- atomic\_flag
- Most of ISO C23
- Bits of C2Y

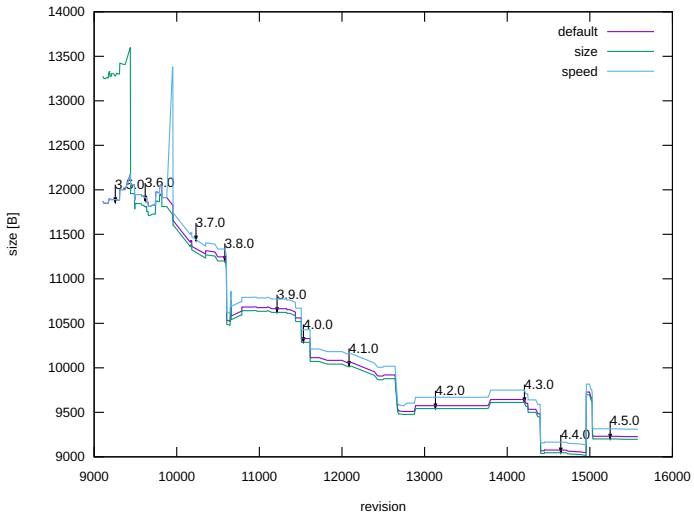
# Generating better code

- Generalized constant propagation
- Various other new optimizations
- More efficient default calling conventions for Rabbits
- New r800 port for ASCII R800
- New mos65c02 port for WDC 65C02
- New f8 port

# STM8 benchmark score history



# STM8 code size history



# Table of Contents

1 The Small Device C Compiler

2 Recent Progress

3 Near Future Plans

# Standard compliance and reliability

- Complete C99-C23 support: compound literals, attributes, atomics, double, long double
- More bits of C2Y (and make C2Y suitable for 8-bit architectures)
- Improve ELF/DWARF support
- More testing (adopt recent GCC tests, command line parameter fuzzing)
- Make SDCC well-suited for safety and security (e.g. guard against introducing timing side-channels)
- Fix bugs

## Generate better code

- Improve Rabbit 2000/3000, eZ80, TLCS-90 support
- New Rabbit 4000, 5000, 6000 ports
- New TLCS-870, TLCS-870/C, TLCS-870/C1 ports
- HC08, S08 improvements
- New PDK16 port
- MCS-51 port rewrite, split into multiple related ports
- Link-time elimination of unused objects and functions
- Efficient allocation of spilt local variables into non-stack memory

## Lessons learned - towards a new 8-bit architecture

- An efficient stackpointer-relative addressing is essential for reentrant functions
- A unified address space is essential for efficient pointer access
- Registers help
- Hardware multithreading can replace peripheral hardware, but it needs good support for atomics, and thread-local storage
- Irregular architectures can be very efficient with tree-decomposition-based register allocation
- A good mixture of 8-bit and 16-bit operations helps
- Pointers should be 16 bits

## Where do we get - f8

- 8/16 bit
- Irregular CISC
- The core becomes bigger than for RISC, but we save so much on code memory that it is worth it
- f8l instruction subset for smaller core
- <https://github.com/f8-arch>

# Summary of near-future plans

Soon:

- Improve support for Rabbit 2000/3000, eZ80, f8, HC08, S08.
- New Rabbit 4000, 5000, 6000 ports
- New f8l port

Then:

- Improve MCS-51 support
- New PDK16 port
- New TLCS-870, TLCS-870/C, TLCS-870/C1 ports
- Improve standard support
- Improve reliability, suitability for safety and security