Improving SDCC for MCS-51

Philipp Klaus Krause

June 30, 2016

Why free tools?

Availability of FLOSS tools for an architecture is essential. Great successes such as those of the ARM and AVR architectures would not have happened without FLOSS compiler and tools support. FLOSS tools make the architecture a viable option for many developers that would not otherwise consider it:

- FLOSS ensures continued availability independent of tool vendor support.
- FLOSS allows easy implementation of features and combination with other software as desired by the user independent of tool vendor policies.
- FLOSS is easily available by simple download and by being included in OS distributions.
- For some projects and developers the use of non-free software is unacceptable.

The compiler is a key component of the free tool chain. For MCS-51 this is SDCC. SDCC is improving steadily, but slowly. Since vendors of MCS-51-compatible hardware would benefit a lot from a better free toolchain, they might be interested in funding some work to speed up SDCC improvements.

Comparing to other compilers

Standard Compliance

The input language to be accepted by C compilers is defined by international standards, with the ANSI C89/ISO C90, ISO C99 and ISO C11 being the most important. No current compiler targeting the MCS-51 fully supports any of the standards. However all of the compilers at least support large subsets of some standards. Supported standards according to compiler documentation:

- SDCC: C90, C95, C99, C11
- Keil: C90
- IAR: C90, C99, EC++
- Raisonance: C90
- Wickenhäuser: C90

SDCC is already doing very well, but needs to catch up on IAR wrt. improving C99 support. Also some C90 compilers have better `struct` / `union` support.

Code Quality

SDCC passes its regression tests, which include most of GCC tests.

To compare code size and speed to other compilers and to check code speed on real hardware we could:

- Get evaluation boards:
  - Need: One for original 8051
  - Need: One for the most common dual-dptr variant (probably Axsem, since it seems well-documented, and Axsem recommends SDCC)
  - Maybe: One for Infineon XC800 (to ensure that SDCC does well vs. the fork, and for developing MDU and Cordic support)
  - Maybe: One for Cypress FX2 (popular for logic analyzers that use SDCC for firmware)
  - Maybe: One for Silicon Labs (there is a FreeRTOS port that uses SDCC)
  - Maybe: One for TI (there is a Contiki port that uses SDCC)
  - Maybe: One for Z8051 (Zilog recommends SDCC)

  All boards should have at least 8 KB of XRAM to run benchmarks, and an UART to report back results. For all boards it should be possible to write the .ihx onto the board using free software only.
- Compile and run Whetstone, Dhrystone, Coremark on evaluation boards.
- Compare code size and benchmark scores to other compilers (might be hard, as some compiler vendors only offer very restricted eval versions - maybe ask someone who has full versions to compile the benchmark?).

Improving SDCC

Standard Compliance

To put SDCC ahead of all other MCS-51 compilers, in order of importance:

- C90: `struct` and `union` assignment
- C90: `struct` and `union` passing
- C90: `struct` and `union` returning
- C99: `long long`
- C99: Separate `_Bool` from `__bit`
- C90/C99/C11: Annex J.3 documentation requirements on implemention-defined behaviour

- C99: Declarations in `for` loop
- C99: Intermingling of declarations and statements
- C99: Compound literals
- C90/C99: Some further stuff that is either not that important (K&R declarations, not fully braced `struct` init) or a lot of work (`double`, `long double`, VLAs)

Debug info

- ELF/DWARFv2 output - should be quick to implement, as some other backends already have it.
- ELF/DWARFv4 output - more work.

Code size and performance

- Find some low-hanging fruit in terms of improving SDCC MCS-51 code size / performance and go for it.
- More agressive leaf function detection - anything that does not call non-reentrant functions could be considered a leaf for variable overlay.
- Implement special case 16 x 8 -> 16 bit multiplication - useful to speed up array indexing
- 24- and 48-bit integers
- `printf()` family rewrite with compile-time analysis of string literal to choose variant, compiler optimizations for `printf()` calls() - combine them, replace by `puts()`, etc.
- Implement mcs51 backend support for dual dptr. New mcs51-variant backends: c521, z8051, xc866.
- Implement mcs51 library support for MDU. New mcs51-variant backends: c517, xc822.
- Reimplement 32-bit `float` using more 8-bit-friendly format.
- Implement 48-bit `double` and `long double`.
- Implement mcs51 library support for Cordic: xc888.
- ?

Other

- Reorganize library (the current handling of stuff shared between different library builds is a bit of a mess and makes the library harder to maintain)
- Emit a warning when calling non-reentrant functions from reentrant function
- Make all compiler support functions reentrant (as most other backends already do)
- Create tutorials for various board on how to get started with MCS-51 SDCC (without depending on any IDE, etc); Something like the STM8 tutorials `http://colecovision.eu/stm8/`.