Small Device C Compiler: Standardkompatibilität und Usability

SDCC-STD-UX

Philipp Krause, Felix Salfelder, Benedikt Freisen

27. September 2022

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS22S19 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

1 Aufgabenstellung und Motivation

Die Informations- und Kommunikationstechnik trägt bei Herstellung und im Betrieb wesentlich zum globalen Ressourcen- und Energieverbrauch bei. Es sollten Einstiegshürden beim Entwickeln sparsamer Systeme verringert werden.

Mikrocontroller (µC) sind in nahezu allen elektronischen Geräten enthalten, und werden dort eingesetzt, wo es starke Einschränkungen bei Kosten, Größe und Energieverbrauch gibt. Sie sind maßgeblich für das Internet of Things (IoT) und Sensornetzwerke. Obwohl der einzelne µC in Herstellung und Betrieb wenig Ressourcen erfordert, ist der Gesamtaufwand aufgrund der Stückzahlen enorm. Beispielsweise wurden 2020 weltweit 8-Bit-µC für circa sieben Milliarden Euro verkauft, bei einem Stückpreis von oft unter einem Cent. Die Wahl eines µC kann den Ressourcen- und Energieverbrauch entscheidend beeinflussen. Wenn statt eines einfacheren 8-Bit µC ein 32-Bit µC eingesetzt werden muss, oder aus anderen Gründen ein vorhandener µC nicht mehr den Anforderungen genügt, und ersetzt werden muss, entstehen Sprungkosten im Verbrauch.

Aktuelle Software, insbesondere auch FOSS, verwendet oft Features aus aktuellen Sprachstandards[1], die von GCC und LLVM unterstützt werden. Um solche Software auch auf einfacheren 8-Bit µC einsetzen zu können, ist ein benutzerfreundlicher Compiler für 8-Bit µC nötig, der aktuelle Standards unterstützt.

Der Small Device C Compiler (SDCC) ist ein ein Freier / Open Source (FOSS) Compiler, der in der Sprache C geschriebene Programme für 8-Bit Mikrocontroller µC übersetzt. Er schließt somit eine Lücke unterhalb der bekannten FOSS Compiler GCC und

[1] „Standard“ wird hier, wie in der Informationstechnik üblich, im Sinne des englischen „standard“ verwendet, was in der Bedeutung dem deutschen „Norm“ entspricht

LLVM, die vorwiegend Systeme ab 32 Bit unterstützen, wie man sie in PCs, Servern und Smartphones findet.

Das Projekt beabsichtigte Unterstützung für C-Sprachstandards in SDCC auszubauen um Interoperabilität und Benutzbarkeit zu verbessern. Es sollte Unterstützung für ISO C90, C99 vervollständigt, für C2X hinzugefügt werden. Es sollte standardkonforme Ausgabe von Debuginformationen implementiert werden, die Integration in Entwicklungsumgebungen verbessert werden.

2 Beitrag zu den Zielen der Förderinitiative „Software-Sprint“

SDCC ist der bedeutendste freie Compiler für 8-Bit-µC, und somit ein essentieller Bestandteil der Softwareinfrastruktur für diese, eingebettete Systeme, das Internet of Things (IoT) und Sensornetzwerke. Die erreichten Verbesserungen in der Unterstützung für die C-Sprachstandards in SDCC stärken daher die Softwareinfrastruktur in diesem Bereich deutlich; SDCC übertrifft bezüglich der C-Standardunterstützung nun auch alle in diesem Bereich verbreiteten nicht-freien Compiler deutlich.

3 Einflüsse und Ergebnisse Anderer

Ein Ziel des Projekts war, die Unterstützung für den zukünftigen C-Standard C2X zu verbessern. Bei Beantragung des Projekts war noch kaum abzusehen, welche Änderungen dieser Standard gegenüber dem aktuellen C17 aufweisen würde. Dies wurde durch das Normierungsgremium ISO SC22/WG14 entschieden, insbesondere auch auf zwei einwöchigen Sitzungen im Mai und Juli, die in die Projektlaufzeit fielen. ISO SC22/WG14 beinflusste also direkt die Ziele des Projekts. Philipp Klaus Krause nahm als von der DIN entsandter Experte an den Sitzungen von ISO SC22/WG14 im Projektzeitraum teil.

4 Einhaltung der Planung

Da über viele Aspekte des zukünftigen C-Standards C2X erst relativ spät während der Projektlaufzeit vom Normierungsgremium ISO SC22/WG14 entschieden wurde, stand für diese relativ wenig Zeit zur Implementierung zur Verfügung. Auch war der Implementierungsaufwand für manche Aspekte von C2X etwas größer als erwartet. Im Bereich der Usability (insbesondere bei der Integration von SDCC in die integrierte Entwicklungsumgebung Code::Blocks) gab es Beiträge aus der Community). Bezüglich der Ziele der Förderinitiative erschien uns die Unterstützung der C-Standards als wichtigster Aspekt. Daher haben wir vorrangig an der Verbesserung der Unterstützung der C-Standards gearbeitet, unter Vernachlässigung von Verbesserungen der Usability und Debuginformationen.

Zu Projektanfang wurde eine priorisierte (1 (am wichtigsten) bis 5) Liste `https://sourceforge.net/p/sdcc/wiki/SDCC-STD-UX/` von Teilzielen erstellt, die jeweils nach den Sitzungen von ISO SC22/WG14 ergänzt wurde, was schließlich 59 Teilziele ergab.

Zum Projektende waren von den Teilzielen jeweils im SDCC Subversion-Repository `https://sourceforge.net/p/sdcc/code/HEAD/tree/trunk/` und damit bereits den Nutzern von SDCC verfügbar: 100% der Teilziele der Priorität 1, 60% der Teilziele der Priorität 2, 61% der Teilziele der Priorität 3, 52% der Teilziele der Priorität 4, 50% der Teilziele der Priorität 5. Einige weitere Teilziele wurden im Projekt bereits soweit entwickelt, dass sie voraussichtlich innerhalb weniger Wochen nach Projektende erreicht werden.

5 Ergebnisse des Projekts

Es wurden entscheidende Verbesserungen in der Standardunterstützung erzielt; zu Projektende dürfte SDCC der Compiler mit der besten Unterstützung für C2X sein, wobei davon auszugehen ist, dass insbesondere GCC und LLVM hier bald auf- und überholen werden. Gegenüber nichtfreien Compilern dürfte der erreichte Vorsprung dagegen noch auf viele Jahre hinaus weiterbestehen.

Die wichtigsten im Projekt erreichten Teilziele waren (dabei bezieht sich C90 auf den ISO C-Standard von 1990, C99 auf den von 1999, C11 auf den von 2011, C2X auf den nächsten):

- Unterstützung für `struct` und `union` als Rückgabetyp von Funktionen (relevant für C90 bis C2X).
- Unterstützung für `struct` und `union` als Parametertyp von Funktionen (relevant für C90 bis C2X).
- Zahlreiche Bugfixes bei `_Generic` (relevant für C11 bis C2X).
- Zahlreiche Bugfixes im Typsystem (relevant für C90 bis C2X).
- Zahlreiche Bugfixes bei Funktionsdeklarationen (relevant für C90 bis C2X).
- Unterstützung für Variably-modified Types (relevant für C99 und C2X).
- Präprozessormodernisierung (relevant für C11 bis C2X).
- Neues `assert` in der Standardbibliothek (relevant für C2X und Usability).
- `typeof` und `typeof_unqual` (relevant für C2X).
- Refactoring des Parsers (relevant für C2X).
- Attribute, außer bei Sprungmarken (relevant für C2X).
- Bit-precise integer types (relevant für C2X).
- Übernahme der Tests von GCC 11 (relevant für C90 bis C2X und Usability).
- Übernahme der Tests von QuickC (relevant für C90 bis C2X und Usability).

- Übernahme der Tests von tinycc (relevant für C90 bis C2X und Usability).
- GCC-kompatible Syntax für Kommandozeilenoptionen (relevant für Usability).
- `auto` (relevant für C2X).
- Bit utilities in der Standardbibliothek (relevant für C2X).
- Checked integer arithmetic in der Standardbibliothek für alle Typen außer (unsigned) long long (relevant für C2X).
- Warnungen bei ungültigen Unicode-Sequenzen (relevant für C2X und Usability).
- Initialisierung mittels `{}` (relevant für C2X).
- Neue Schlüsselwörter (relevant für C2X).
- `unreachable` (relevant für C2X).
- UTF8-Literals vom Typ `char8_t` (relevant für C2X).
- `memalignment` der Standardbibliothek (relevant für C2X).
- Neue regeln für `enum`-Typen (relevant für C90 bis C2X).
- `nullptr` und `nullptr_t` (relevant für C2X).
- Neue Tag-compability-Regeln (relevant für C2X).
- Sprungmarken am Ende von Compound Statements (relevant für C2X).
- `call_once` der Standardbibliothek (relevant für C2X).

Daneben gab es zahlreiche diverse Bugfixes. Es wurde ein SDCC developer meeting organisiert und durchgeführt, es wurden dort, sowie auf der FSiC und der FrOSCon Vorträge zu SDCC gehalten.

Leider ist es bei einigen Teilzielen nicht gelungen, diese im Projektzeitraum zu erreichen, insbesondere:

- Compound Literals (relevant für C99 bis C2X).
- Bugfixes bei Debuginformationen (relevant für Debugging).
- ELF/DWARF-Unterstützung im mcs51-Backend (relevant für Debugging).
- Enhanced Enumerations (relevant für C2X).
- Checked integer arithmetic in der Standardbibliothek für (unsigned) long long (relevant für C2X).
- Qualifier-preserving functions in der Standardbibliothek (relevant für C2X).
- `constexpr` (relevant für C2X).
- ELF/DWARF-Unterstützung im sm83-Backend (relevant für Debugging).

6 Zielgruppe und Nutzen für diese

Die Zielgruppe sind Programmierer von µC, etwa Entwickler eingebetteter Systeme, und Nutzer von Geräten, die Ihr Recht auf Reparatur wahrnehmen. Dieses Projekt addressiert sowohl solche, die bereits SDCC verwenden, als auch solche, die bisher nur andere Compiler (nicht-freie Compiler für 8-Bit-µC oder beliebige Compiler für größere 32-Bit-µC) verwenden. Durch die verbesserte Standardunterstützung und Usability wird diese um einige Programmierer, die bisher nur für größere Systeme programmierten, erweitert. Über die neuen Features wird durch die Ankündigungen der jährlichen SDCC-Releases (auf den SDCC Mailinglisten, in Foren zu µC-Programmierung) informiert. Es gab Vorträge auf einschlägigen Konferenzen (FSiC, FrOSCon) dazu. Beim SDCC developer meeting (im Hackerspace haxko e.V., Andernach) befasste sich einer der drei öffentlichen Vorträge mit dem Projekt, und dessen Fortschritt zum damaligen Zeitpunkt.

7 Angestrebte Weiterentwicklung

Die erreichten umfangreichen Verbesserungen in der Unterstützung der C-Standards sollen in den nächsten Monaten noch geringfügig ergänzt werden. Es ist zu erwarten, dass mit steigender Verwendung von C2X noch Bugs durch Nutzer gefunden werden, die zu beheben sind.

Um Nutzer nichtfreier Compiler zum Umstieg auf SDCC zu bewegen erscheint es mittelfristig zielführend, hauptsächlich an Effizienz (im Bezug auf Größe und Geschwindigkeit des erzeugten Codes), Bugfixes und Verbesserungen der Debugunterstützung zu arbeiten. Hierzu wurden Daten von Benutzern und potentiellen Nutzern eingeholt und ausgewertet. Zur Beschleunigung der nötigen Arbeiten wird eine Förderung durch NGI Zero oder den Sovereign Tech Fund angestrebt.