How-To

TSE GmbH
timmermann, Software-Entwicklung

Im Zillhardt 4
76646 Bruchsal

Tel. 07257-924-630, Fax 07257-924-9777
Email: bernd.timmermann@tse-informatik.de

Datum: 28.10.06

An: Entwickler

Von: Bernd Timmermann

Betreff: Quellkode-Debugging DILNET-PC

Die Werkzeuge im Embedded-Bereich sind zwar in manchen Dingen immer noch nicht so, wie sie sein sollten, aber es wird besser, und mit Eclipse 3.2 und CDT 3.1.1 kann man auf dem DILNET-PC von SSV inzwischen sogar Quellkode-Debugging betreiben.

Dieser Artikel erklärt in der notwendigen Kürze, wie das geht.

Hinreichende Kenntnisse zu Eclipse, CDT und der Handhabung der SSV-ARM-Toolchain und Konfigurationswerkzeuge werden vorausgesetzt, können aber von C-Entwicklern i.a. rasch erworben werden.

Eclipse 3.2 und CDT 3.1.1 sollten also auf dem Linux-PC installiert sein (s. www.eclipse.org).

Ebenfalls installiert sein sollte auf den Linux-Rechner die SSV-ARM-Toolchain und auf dem DILNET-PC die Shared-Library libthread_db.so (mit Link-Datei von der CD aus sysroot\lib auf /usr/lib auf dem DILNET-PC).

Der root-Account des DILNET-PC muss ein Passwort haben, um alles per FTP übertragen zu können.

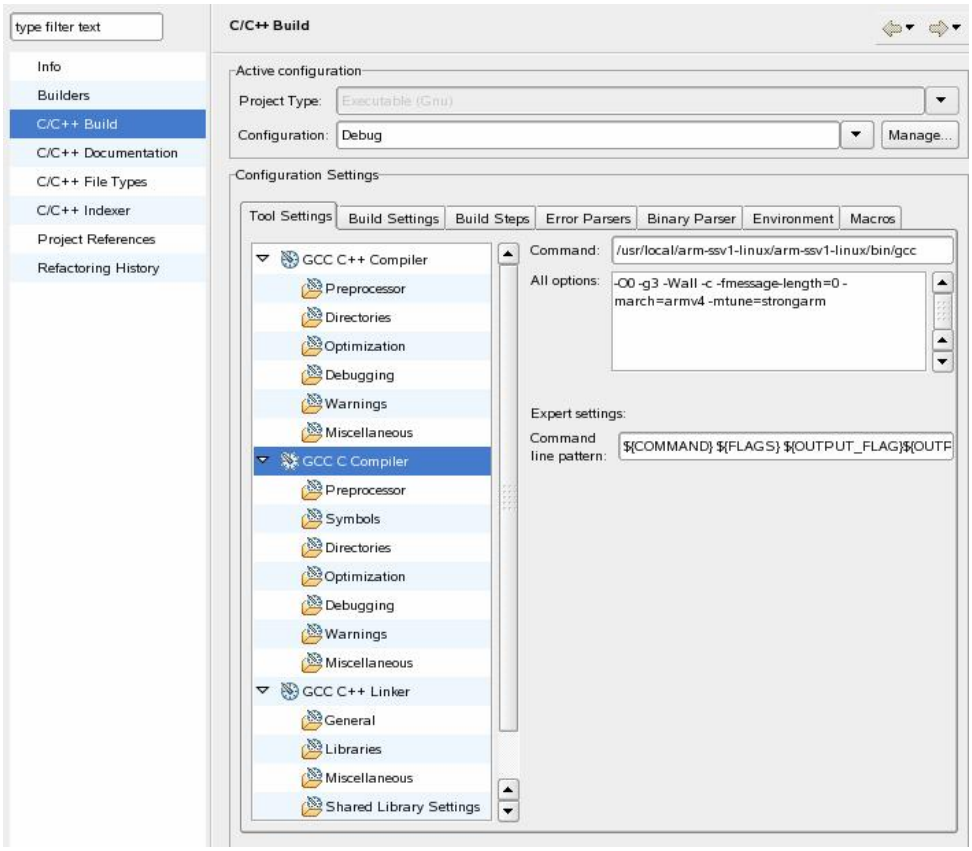
Auf dem DILNET-PC muss/sollte ferner SSH installiert werden (hierfür gibt es von SSV ein sehr gutes Setup). Für den Account, mit dem man arbeitet, muss man die Anmeldung über SSH über einen RSA-Key einrichten, sonst stört die Passwortabfrage: Auf dem Linux-PC in ~/.ssh mit ssh-keygen ein Schlüsselpaar erzeugen, und den Inhalt des öffentlichen Schlüssels (id_rsa.pub) zusätzlich in die Textdatei ~/.ssh/authorized_keys auf dem DILNET-PC stellen).

Anschließend sollte man sich die aktuelle gdb/gdbserver-Version [>= 6.5] für ARM downloaden oder bauen. gdb ist von www.gnu.org und kann dort geladen werden. Beim Übersetzen ist nur wichtig, dass configure für gdb mit --target=arm-linux und für gdbserver mit --host=arm-linux aufgerufen wird. Wer die Version 6.5 braucht, kann sich bei mir melden.

Ob man den ARM-gdb in die ARM-Toolchain oder in ein privates bin-Verzeichnis des Linux-PC abglegt, ist Geschmackssache. Den gdbserver kopiert man jedenfalls mit FTP in das /usr/local/bin-Verzeichnis oder ein anderes Verzeichnis im Path des DILNET-PC.

Nun ist es Zeit, ein Beispiel-CDT-Projekt zu erstellen (HelloWorld oder was immer einem einfällt).

Ist das geschafft, muss man als nächstes die Tools setzen, mit denen CDT arbeitet. Dazu klickt man das Projekt im C/C++-Explorer (ein Eclipse-View von CDT) mit der rechten Maustaste an (Kontextmenü) und wählt „Properties“. In dem erscheinenden Dialog kann man im Reiter „ToolSettings“ den Linker und den Compiler wählen. In allen Tools trägt man unter „Commando“ den Compiler g++ bzw. gcc aus der SSV-ARM-ToolChain ein. Für den C-Compiler-Eintrag sollte man unter „Miscellanous“ noch die „Other Flags“ –march=armv4 und –mtune=strongarm eintragen.



Nun sollte man übersetzen können, und ein ARM-fähiges Programm als Ergebnis erhalten.

Um Debuggen zu können, muss man nun nur noch eine kleine Initialisierungsdatei für den GDB schreiben und die Debug-Konfiguration einrichten.

Die GDB-Initialisierung ist je Projekt zu erstellen und sieht für mein HelloWorld in etwa so aus:

```
shell ftp -u ftp://root:mypass@dil02/SSV/HelloWorld_01 \
      ~/SSV/HelloWorld_01/Debug/HelloWorld_01
shell ssh -t -l root dil02 chmod +x ./SSV/HelloWorld_01
shell ssh -t -l root dil02 ./SSV/gdbserver 192.168.0.121:2222 ./SSV/HelloWorld_01
file ~/SSV/HelloWorld_01/Debug/HelloWorld_01
shell sleep 2
```

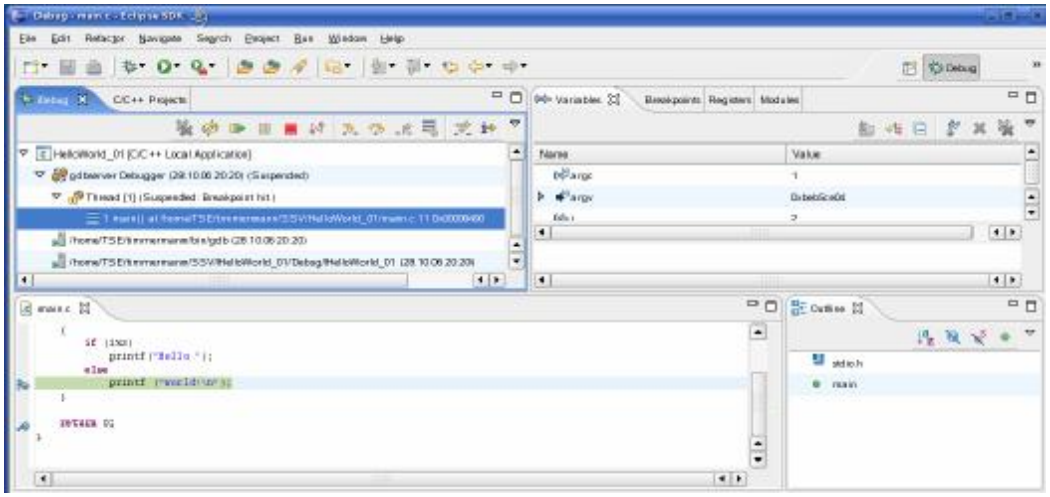
Zur Erklärung:

> GDB kann mit „shell“ bash-Kommandos ausführen.
>
> Das erste Kommando kopiert die ausführbare Datei per FTP auf den DILNET-PC.
>
> Die zweite Zeile setzt die Ausführungsberechtigung.
>
> Die dritte Anweisung startet per SSH den gdbserver auf dem DILNET-PC
>
> Die vierte Zeile lädt die Symbole der ausführbaren Datei in den GDB (geht scheinbar nicht ohne).
>
> Dann wird 2 Sekunden gewartet, bis es weiter geht.

Im letzten Schritt vor dem Debuggen muss nun nur noch eine Debug-Konfiguration eingerichtet werden. In CDT aktiviert man dazu die Aktion „<Bug> Debug…“ und erzeugt eine neue Konfiguration für eine „local Application“. Den Projektnamen und die „ausführbare“ Datei setzt man wie üblich. Als Debugger wählt man explizit den ARM-gdbserver und als GDB-Command File die oben editierte Initialisierungsdatei. Unter Connection gibt man die TCP-Adresse des DILNET-PC und den TCP-Port (ich habe oben wieder 2222 verwendet).

Wenn man nun den Debugger startet, erscheint nach einigen Sekunden der Quellkodedebugger. Das Round-Trip-Debugging funktioniert wie gewohnt.

Zu beachten ist, dass man durch den GDB-Start des GDB-Server keine brauchbare Konsole für „printf“ hat. Wenn man diese braucht, muss man gdbserver manuell in einer Telnet-Konsole starten und den Ablauf von oben entsprechend abändern.



Die Verfahrensbeschreibung ist gegeben „as is“. Bisher habe ich noch keine Erfahrung hinsichtlich Robustheit und Reichweite, jedoch das Verfahren per se funktioniert bei mir zumindest mal.

Wer zu dem Thema etwas neues (z.B. Probleme oder Lösungen) hat, möge dies bitte mir zur Kenntnis bringen.

Viel Spaß damit!