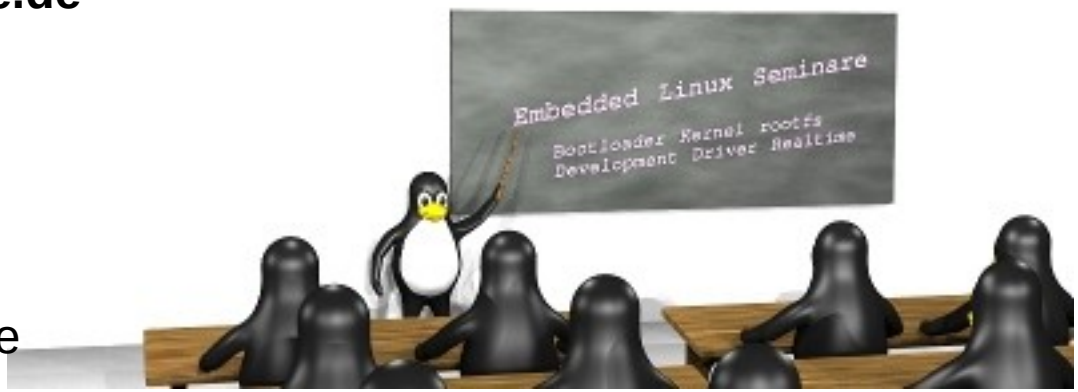
Embedded-Linux-Seminare

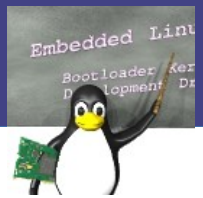
Toolchains

http://www.embedded-linux-seminare.de

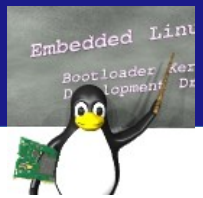
Diplom-Physiker Peter Börner
Spandauer Weg 4
37085 Göttingen

Tel.: 0551-7703465
Mail: info@embedded-linux-seminare.de



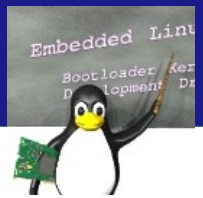


Kopier-Rechte

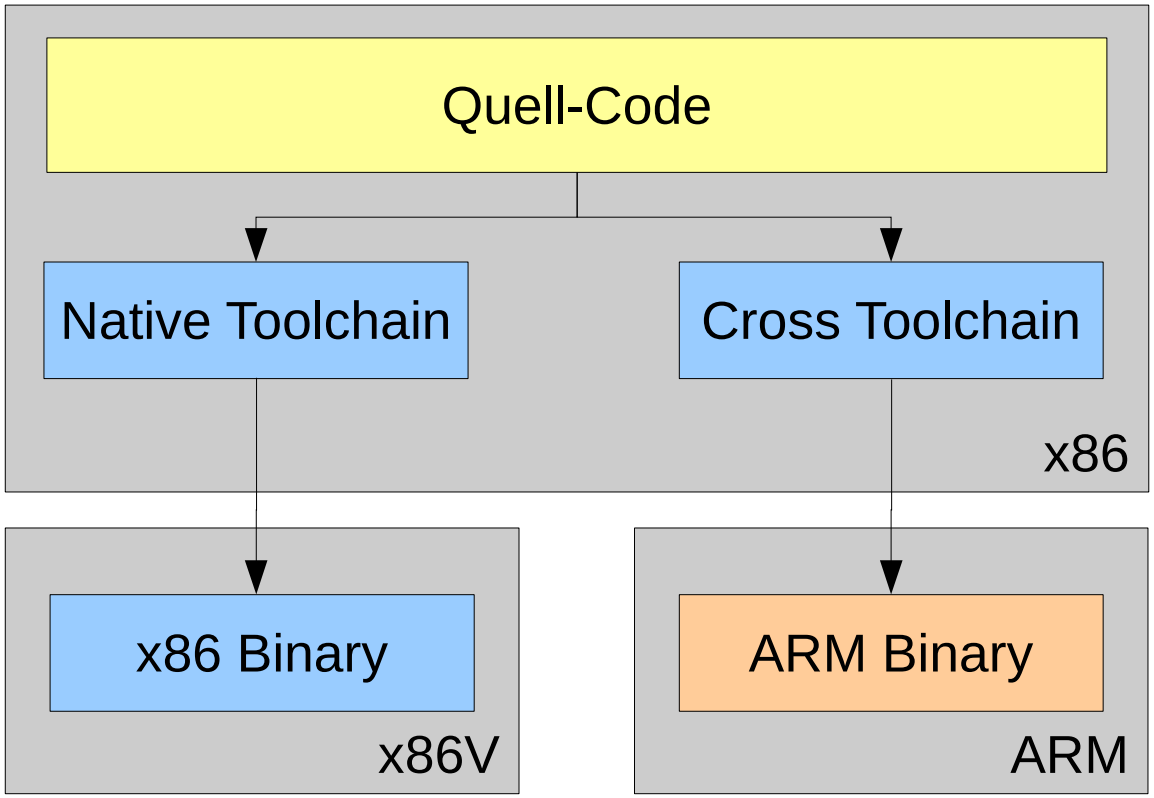


Definition

- Die normalen Entwicklungstools auf einem Linux System bilden die *Native-Toolchain.*
- Diese Toolchain läuft auf dem Rechner mit einer bestimmten Architektur (meist x86) und generiert Code für die gleich Architektur.
- Für Embedded Systeme ist es meist nicht sinnvoll eine Native-Toolchain zu verwenden
  - Das Target System hat zu wenig Ressourcen und zu geringe Performance
  - Die Entwicklungs-Tools sollen nicht auf dem Target installiert werden
- Da die Architektur des Embedded System meist vom Entwicklungsrechner abweicht, wird eine *Cross-Toolchain* benötigt, welche Code für die Target Architektur erzeugt.

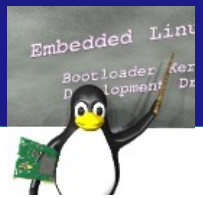


Definition



Entwicklungs Rechner

Ziel Rechner



Komponenten

Binutils

as, ld, ar, ranlib, strip, ...

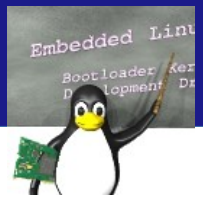
Kernel Header

C/C++ Bibliotheken

GCC Compiler

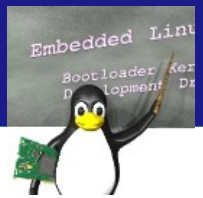
GDB Debugger

(optional)

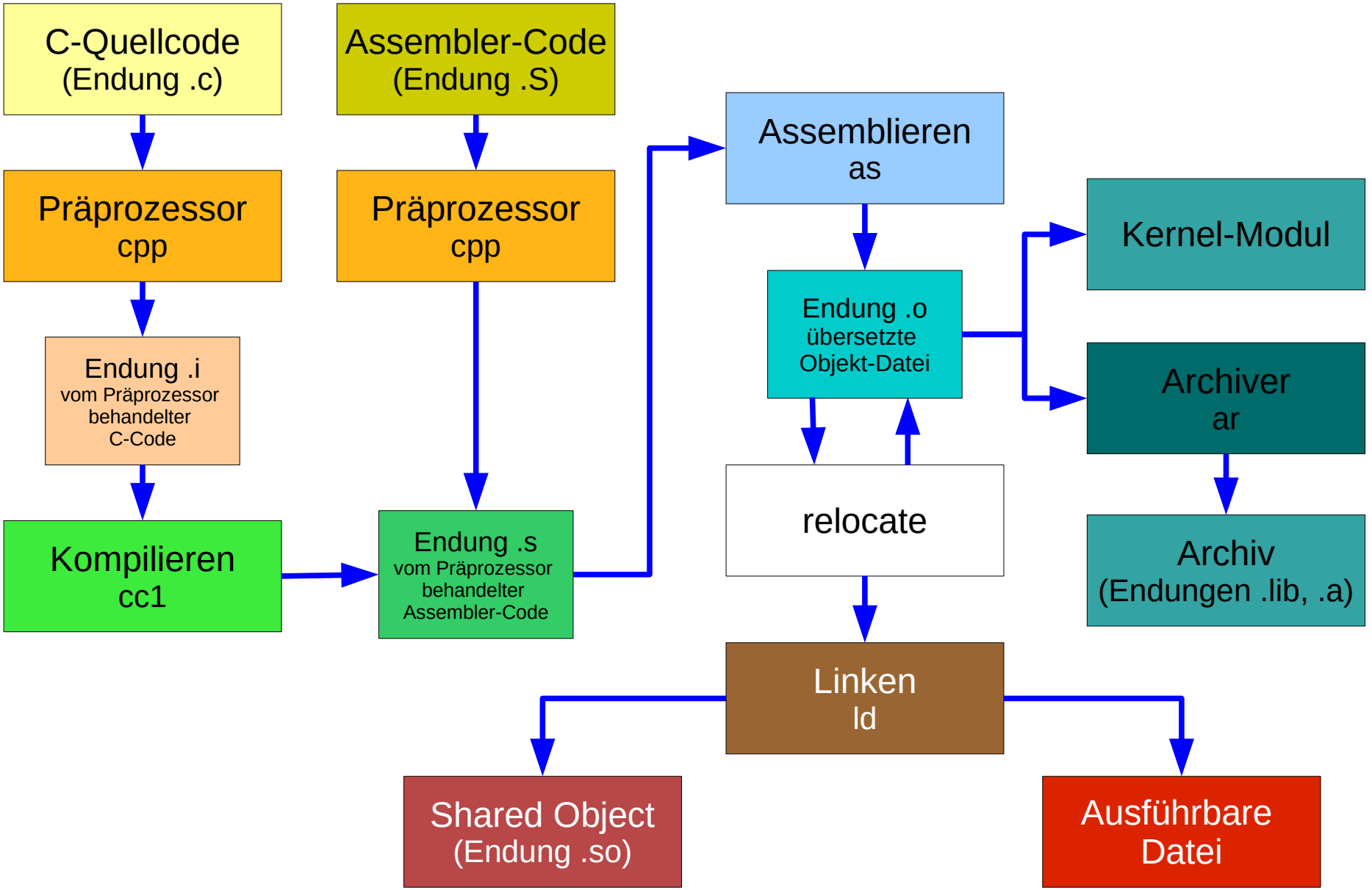


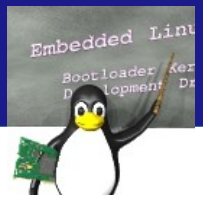
GCC

- GNU Compiler Collection
- Besteht aus mehreren Compiler für verschiedene Sprachen und dazugehörigen Bibliotheken
  - C gcc
  - C++ g++ libstdc++
  - Java gcj libgcj
  - Fortran gfortan libgfortran
  - Weitere: Ada, Go, Objective-C
- gcc ist der C Compiler
- gcc erkennt auch andere Sprachen und ruft den dazugehörigen Compiler auf



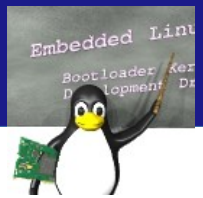
gcc


C-Quellcode
(Endung .c)
Assembler-Code
(Endung .S)
Präprozessor
cpp
Präprozessor
cpp
Endung .i
vom Präprozessor
behandelter
C-Code
Kompilieren
cc1
Endung .s
vom Präprozessor
behandelter
Assembler-Code
Assemblieren
as
Endung .o
übersetzte
Objekt-Datei
relocate
Linken
ld
Kernel-Modul
Archiver
ar
Archiv
(Endungen .lib, .a)
Shared Object
(Endung .so)
Ausführbare
Datei



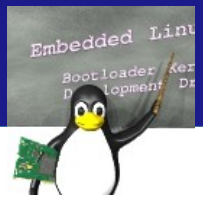
Binutils

- Die Binutils sind eine Sammlung von Tools, um Ausführbare Dateien für eine bestimmte CPU zu erzeugen und zu verändern.
  - **as** Assembler, generiert ausführbaren Binär-Code aus Assembler-Quellcode
  - **ld** Linker
  - **ar, ranlib** Erzeugen .a Archive, werden für Bibliotheken verwendet
  - **objdump, readelf, size, nm, strings** können zum analysieren des Binär-Codes genutzt werden.
  - **strip** Entfernt unnötige Teile aus dem Binär-Code (z.B. Debugging Informationen) und reduziert damit die Größe.
- Stehen unter der GPL
- http://www.gnu.org/software/binutils



Kernel Header

- Die C-Bibliothek und kompilierte Programme müssen mit dem Kernel interagieren
  - Vorhandene System-Calls und deren ID
  - Konstanten Definitionen
  - Daten Strukturen, etc.
- Deshalb werden zum Kompilieren der C-Bibliothek die Kernel Header benötigt
- Einige Applikationen brauchen die Kernel Header direkt
- Verfügbar in `<linux/...>` und `<asm/...>`



Kernel Header

- System-Call Ids in <asm/unistd.h>

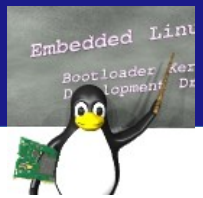
```
#define __NR_exit                1
#define __NR_fork                2
#define __NR_read                3
```

- Konstanten Definitionen z.B. in <asm-generic/fcntl.h>. eingebunden von <asm/fcntl.h>, eingebunden von <linux/fcntl.h>

```
#define O_RDWR                   00000002
```

- Daten Strukturen, z.B. in <asm/stat.h>

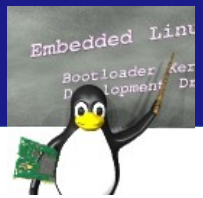
```
struct stat {
        unsigned long st_dev;
        unsigned long st_ino;
        […]
};
```



Kernel Header

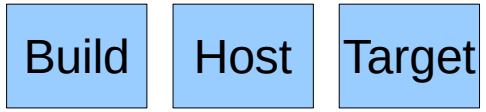
- Kernel – User-Space ABI ist rückwärts kompatibel
  - Binär-Code, der mit einer Toolchain erzeugt wurde, die ältere Kernel Header verwendet als der laufende Kernel, funktioniert. Er kann aber neue System-Calls nicht verwenden.
  - Binär-Code, der mit einer Toolchain erzeugt wurde, der neuere Kernel Header verwendet als der laufende Kernel, wird laufen, solange keine neuen Funktionen genutzt werden
  - Es ist nicht nötig die neuesten Kernel Header zu verwenden, es sei denn neue Features werden benötigt.
- Die Kernel Header werden aus dem Quell-Code Pfad des Kernels extrahiert mit

```
make headers_install
```



Cross-Toolchain Bauen

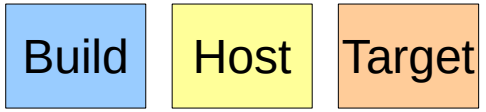
- Es werden drei Maschinen unterschieden, wenn über das Generieren einer Cross-Toolchain gesprochen wird
  - Build: Auf dieser wird die Toolchain gebaut
  - Host: Auf dieser wird die Toolchain ausgeführt
  - Target: Auf dieser sollen die erzeugten Binaries laufen
- 4 Variationen des Bauvorgangs sind möglich



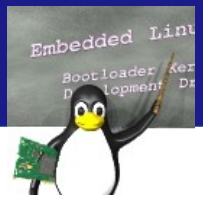
Native Build
für den Bau des normalen gcc

Cross Build
für den Bau einer Toolchain, die auf dem Entwicklungsrechner läuft und Code für das Target erzeugt.
Häufigste Variante bei Embedded Systemen

Cross-Native Build
für den Bau einer Toolchain, die auf dem Target läuft und Code für das Target erzeugt.

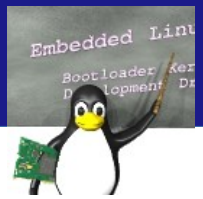


Canadian Build
Toolchain wird auf A gebaut, läuft auf B und erzeugt Code für C



Cross-Toolchain Bauen

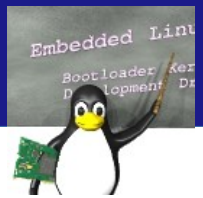
- Es müssen einige Entscheidungen vorab getroffen werden
  - Welche C-Bibliothek?
  - Versionen der verschiedenen Komponenten
  - Konfiguration der Toolchain
    - Welche ABI? Für ARM ist z.B. OABI (Old ABI) oder EABI (Embedded ABI) möglich
    - Soll Software Flaoting Point Operationen genutzt werden oder unterstützt die Hardware dies?
    - Soll locales, Ipv6 oder andere spezifische Features unterstützt werden



Cross-Toolchain Bauen

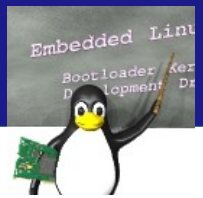
Cross-Toolchain Bau-Report http://kegel.com/crosstool/crosstool-0.43/build.logs

gcc-4.1-20050709 cgcc-3.3.6 glibc-2.3.2 binutils-2.15 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050709 cgcc-3.3.6 glibc-2.3.2 binutils-2.15 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	alpha
ICE gdb FAIL	ICE gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm9tdmi
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	arm-iwmmxt
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	arm-softfloat
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm-xscale
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	armeb
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	armv5b-softfloat
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	i686
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ia64
kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	m68k
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	mips
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	mipsel
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-405
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-750
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-860
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	powerpc-970
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	s390



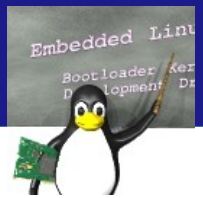
Bauvorgang

- Kernel Header installieren
- Binutils konfigurieren, kompilieren und installieren
- Erste Version des gcc die Code für das Target erzeugt konfigurieren, kompilieren und installieren. Wird benötigt, um die C-Bibliothek zu kompilieren
- C-Bibliothek konfigurieren und mit gerade erzeugtem gcc kompilieren
- Endgültigen gcc konfigurieren und kompilieren
- Achtung: Viele Fallstricke, setzt einiges an Know-How voraus.



Fertige Cross-Toolchains

- Einfacher ist es vorgefertigte Toolchains zu verwenden
  - CodeSourcery
    - Bietet nur glibc Toolchains
    - http://www.codesourcery.com
  - Siehe http://elinux.org/Toolchains
- Alternativ können auch fertige Tools zum Bauen einer Toolchain genutzt werden
  - Crosstool
    - von Dan Kegel
    - http://www.kegel.com/crosstool
  - Crosstool-ng
    - Neu geschriebene Crosstool
    - Unterstützt µClibc, glibc, eglibc
    - http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool



Tools zum Bauen der Toolchain

- Es gibt zahlreiche root-Filesystem Bausysteme, die das Erstellen einer Cross-Toolchain ermöglichen
  - Buildroot
    - Makefile basiert
    - Nur µClibc
    - Von Community betreut
    - http://buildroot.uclibc.org
  - PTXdist
    - Makefile basiert
    - µClibc und glibc
    - Von Pengutronix betreut
    - http://www.pengutronix.de/software/ptxdist
  - OpenEmbedded
    - Breite Unterstützung, aber komplex
    - http://www.openembedded.org