第8章 リンカ周りはコツが要る

今回のLLVM/Clang環境におけるクロス・コンパイル方法

村井 和夫

第7章で，ARM用クロス・コンパイラをインストールし，最新のGCC/LLVMを使ってARMの開発ができる環境を構築しました．本稿では，構築した環境を使ってプログラムを開発するためのクロス・コンパイル方法やコツを解説します．

LLVM/Clang開発環境でまだまだなこと

● その1：クロス・コンパイラのプロセッサ対応状況

LLVMは，GCCと違ってプロセッサごとにコンパイラを用意する必要はありません．CPU種別をパラメータで指定するだけで，クロス・コンパイラとして動作させることができます．しかし，ARMの対応状況についてはまだ完全な状態ではなく，以下に示す内容について注意する必要があります．**表1**にLLVMの各種プロセッサへの対応状況を示します．

● その2：アセンブラ&リンカ

LLVMでは，組み込みに必須となるリンカ・スクリプトによるCPU上のメモリ割り当て機能のサポート状況や，ARMのアセンブラ機能の互換性について，まだドキュメント化されていないようです．

▶今回アセンブラとリンカはGNUのBinutilsを使う

このため今回構築した環境では，アセンブラやリンカはGNUのBinutilsを使うことでGNUの開発環境との互換性を保つようにしました．

クロス・コンパイルの注意点

LLVM/Clangでクロス・コンパイルを実施するにあたり，主な注意点を次に示します．

● LLVMのバージョンによって対応が異なる

LLVMは現在も盛んに開発が進められており，ほぼ1年に2回程度のペースでバージョン・アップが行われています．バージョン3.5.0は，2014年9月4日にリリースされた原稿執筆時の最新版です．クロス・コンパイルの手順もLLVMのバージョンによっても変わります．ここでは，最新の3.5.0と，一つ前にリリースされた3.4でのクロス・コンパイルの手順について説明します．

● やらないといけないこと

以下，順番にLLVM/Clangでの対処方法と，クロス・コンパイルで動作するオブジェクト・ファイルの作成手順を説明します．

▶ARMアーキテクチャの指定

LLVMのバージョンによってARMの個別CPUアーキテクチャの指定方法が微妙に異なります．また，Cortex-Mは，`clang`コマンドだけでは中間言語の出力までしかできません．アセンブラ以降は個別に

表1(1) コミュニティが公開しているLLVMの各種プロセッサへの対応状況
筆者調査時の情報で最新ではない可能性があります

機　能	ARM	MIPS	PowerPC	SPARC	x86	xCORE
安定度	○	○	○	○	○	○
アセンブラ	×	×	×	×	○	×
逆アセンブラ	○	×	×	○	○	○
インライン・アセンブラ	○	×	○	不明	○	○
JITコンパイラ	○注	○	○	不明	○	×
オブジェクト・ファイル出力	×	×	×	×	○	×
テイル・コール	○	×	○	不明	○	×
スタック・セグメント	×	×	×	×	○注	×

注▶部分的にサポート