

この資料は、更新された最新の英語版が存在します。こちらの日本語版は参考用としてご利用下さい。設計の際は、必ず最新の英語版で内容をご確認下さい。

NI152003-1.0

はじめに

この章では、Nios® II プロセッサ用の HAL(Hardware Abstraction Layer) システム・ライブラリについて説明します。

HAL システム・ライブラリは、シンプルなデバイス・ドライバ・インタフェースを提供する軽量のランタイム環境です。プログラムはこのインタフェースを使用して基礎となるハードウェアと通信します。HAL アプリケーション・プログラム・インタフェース (API) は、ANSI C 標準ライブラリと統合されています。HAL API を使用すると、`printf()`、`fopen()`、`fwrite()` など、一般的な C ライブラリ関数によってデバイスおよびファイルにアクセスできます。

HAL は Nios II プロセッサ・システムのボード・サポート・パッケージとして機能し、エンベデッド・システムのペリフェラルへの一貫したインタフェースを提供します。SOPC Builder と Nios II 統合開発環境 (IDE) は緊密に統合されているため、HAL システム・ライブラリを自動的に生成できます。SOPC Builder がハードウェア・システムを生成すると、Nios II IDE は、そのハードウェア・コンフィギュレーションに合わせて、独自の HAL システム・ライブラリを生成できます。さらに、ハードウェア・コンフィギュレーションを変更すると、HAL デバイス・ドライバのコンフィギュレーションに自動的に反映されます。このため、基本ハードウェアをわずかに変更したことによって発生するやっかいなバグがなくなります。

HAL によってデバイス・ドライバが抽象化されるため、アプリケーションとデバイス・ドライバ・ソフトウェアが明確に区別されます。このようなドライバの抽象化によって、基本ハードウェアの変更に対応する再利用可能なアプリケーション・コードを容易に記述できます。さらに、既存のペリフェラル・ドライバに対応する新しいハードウェア・ペリフェラル用のドライバも簡単に記述できます。

使用開始にあたって

HAL の使用を開始する最も簡単な方法は、Nios II IDE に付属するオンライン・チュートリアルを実行することです。Nios II IDE で新規プロジェクトを作成するプロセスでは、HAL システム・ライブラリも同時に作成します。HAL ファイルを作成したりコピーしたりする必要はなく、また HAL ソース・コードを編集する必要もまったくありません。HAL システム・ライブラリは、Nios II IDE によって自動的に生成および管理されます。

HAL システム・ライブラリは、特定の SOPC Builder システム上に構築する必要があります。SOPC Builder システムとは、ペリフェラルおよびメモリと統合された Nios II プロセッサ・コア（SOPC Builder によって生成）を意味します。独自の SOPC Builder システムがない場合は、アルテラが提供するハードウェア・システムの例をベースにして、プロジェクトを作成できます。最初に、アルテラの Nios 開発ボードをターゲットとするプロジェクトを開発し、その後でプロジェクトのターゲットをカスタム・ボードに変更できます。ターゲットの SOPC Builder システムは、後から簡単に変更できます。



新規プロジェクトの開始に関する詳細は、Nios II IDE のオンライン・ヘルプを参照してください。

HAL アーキテクチャ

このセクションでは、HAL アーキテクチャの基本的な要素について説明します。

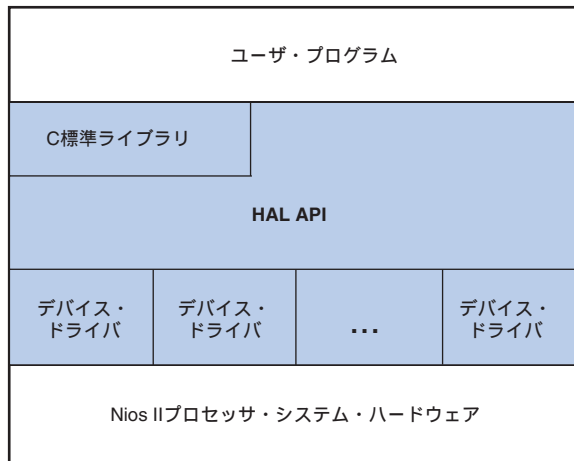
サービス

HAL システム・ライブラリは、次のサービスを提供します。

- newlib ANSI C 標準ライブラリとの統合
使い慣れた C 標準ライブラリ関数が利用できるようになります。
- デバイス・ドライバ
システム内の各デバイスへのアクセスが可能になります。
- HAL API
デバイス・アクセス、割り込み処理、アラーム機能など、HAL サービスへの一貫した標準インタフェースを提供します。
- システムの初期化
`main()` を実行する前に、プロセッサおよびランタイム環境用の初期化タスクを実行します。
- デバイスの初期化
`main()` を実行する前に、システムの各デバイスをインスタンス化および初期化します。

図 3-1 は、ハードウェア・レベルからユーザ・プログラムまでの HAL ベースのシステムのレイヤを示します。

図 3-1. HAL ベースのシステムのレイヤ



アプリケーションとドライバ

プログラムは、アプリケーション開発者とデバイス・ドライバ開発者の2つの明確なグループに区分されます。アプリケーション開発者にはユーザの大部分が該当し、各種ルーチンの中でも特にシステムの `main()` ルーチンの記述を担当します。アプリケーションは、C 標準ライブラリまたは HAL システム・ライブラリ API を介してシステム・リソースと通信します。デバイス・ドライバ開発者の役割は、アプリケーション開発者がデバイス・リソースを利用できるようにすることです。デバイス・ドライバは、低レベルのハードウェア・アクセス・マクロにより、ハードウェアと直接通信します。

このため、主要な HAL の資料は、次の 2 つの章に大きく分割されています。

- **第 4 章 HAL を使用したプログラムの開発**では HAL を活用し、基本ハードウェアを意識しないでプログラムを記述する方法について説明しています。
- **第 5 章 HAL 用デバイス・ドライバの開発**では、ハードウェアと直接通信する方法、および抽象化された HAL API を介してハードウェア・リソースを利用可能にする方法について説明しています。

汎用デバイス・モデル

HAL は、タイマ、Ethernet MAC/PHY チップ、キャラクタ・データを伝送する I/O ペリフェラルなど、エンベデッド・システムで広く使用されるペリフェラルのクラスに対応する汎用デバイス・モデルを提供します。汎用デバイス・モデルは、HAL システム・ライブラリの中核となる機能です。汎用デバイス・モジュールを利用すれば、基本ハードウェアに関係なく、一貫した API を使用してプログラムを記述することができます。

デバイス・モデル・クラス

HAL は、次のデバイスのクラス用のモデルを提供します。

- **キャラクタ・モード・デバイス**
UART など、キャラクタをシリアルに送受信するハードウェア・ペリフェラルです。
- **タイマ**
クロックをカウントし、周期的な割り込み要求を生成できるハードウェア・ペリフェラルです。
- **ファイル・サブシステム**
物理デバイス内に格納されたファイルにアクセスするためのメカニズムを提供します。内部実装に応じて、ファイル・サブシステム・ドライバは、基本デバイスに直接アクセスしたり、別のデバイス・ドライバを使用したりすることができます。例えば、フラッシュ・メモリ・デバイス用の HAL API を使用して、フラッシュにアクセスするフラッシュ・ファイル・サブシステム・ドライバを記述できます。
- **イーサネット・デバイス**
アルテラが提供する軽量 IP プロトコル・スタックに対応したイーサネット接続へのアクセスを可能にします。
- **DMA デバイス**
データ・ソースからディスティネーションへのバルク・データ転送を実行するペリフェラルです。イーサネット接続など、メモリやその他のデバイスをソースおよびディスティネーションにすることができます。
- **フラッシュ・メモリ・デバイス**
専用のプログラミング・プロトコルを使用してデータを格納する不揮発性メモリ・デバイスです。

アプリケーション開発者の利点

HAL システム・ライブラリは、デバイスの各クラスの初期化とアクセスに使用する関数のセットを定義します。この API は、デバイス・ハードウェアの基本実装状態に関係なく、一貫性が維持されています。例えば、キャラクタ・モードのデバイスおよびファイル・サブシステムにアクセスする場合には、`printf()` や `fopen()` などの C 標準ライブラリ関数が使用できます。アプリケーション開発者の場合、これらペリフェラルのクラスに対するハードウェアとの基本的な通信を確立するためだけに、低レベルのルーチンを記述する必要はありません。

デバイス・ドライバ開発者の利点

各デバイス・モデルは、デバイスの特定のクラスを操作するのに必要なドライバ関数のセットを定義します。新しいペリフェラル用のドライバを記述する場合は、このドライバ関数のセットを提供するだけで十分です。結果として、ドライバ開発作業は事前定義され、記録されます。さらに、既存の HAL 関数とアプリケーションを使用してデバイスにアクセスし、ソフトウェア開発の労力を軽減させることもできます。HAL システム・ライブラリは、ドライバ関数をコールしてハードウェアにアクセスします。アプリケーション・プログラムは、ドライバ・ルーチンを直接呼び出すのではなく、ANSI C または HAL API を呼び出してハードウェアにアクセスします。したがって、ドライバの使用法は HAL API の一部として記録されます。

C 標準ライブラリ — Newlib

HAL システム・ライブラリでは、ANSI C 標準ライブラリがランタイム環境に統合されています。HAL は、C 標準ライブラリのオープン・ソース実装である newlib を使用しています。newlib は、エンベデッド・システムで使用するための C ライブラリであり、HAL および Nios II プロセッサに最適です。newlib のライセンスでは、ソース・コードのリリースや newlib ベースのプロジェクトに対するロイヤリティは不要です。

ANSI C 標準ライブラリに関する文献は豊富にあります。最もよく知られた参考文献は、Prentice Hall から出版された、B.W. カーニハン / D.M. リッチー著のプログラミング言語 C でしょう。この文献は、20 以上の言語に翻訳されています。また、Redhat 社は、<http://sources.redhat.com/newlib> で newlib のオンライン資料を提供しています。

サポート されている ペリフェラル

アルテラは、Nios II プロセッサ・システムで使用する多数のペリフェラルを提供しています。大部分のアルテラ製ペリフェラルでは、HAL API を介してハードウェアへのアクセスを可能にする HAL デバイス・ドライバが利用できます。次のアルテラ製ペリフェラルは、HAL を完全にサポートしています。

- キャラクタ・モード・デバイス：
 - UART コア
 - JTAG UART コア
 - LCD 16207 ディスプレイ・コントローラ
- フラッシュ・メモリ・デバイス
 - 共通フラッシュ・インタフェース準拠のフラッシュ・チップ
 - アルテラの EPCS シリアル・コンフィギュレーション・デバイス・コントローラ
- ファイル・サブシステム
 - リード・オンリ zip ファイル・システム
- タイマ・デバイス
 - タイマ・コア
- DMA デバイス
 - DMA コントローラ・コア
- イーサネット・デバイス
 - LAN91C111 Ethernet MAC/PHY Controller



LAN91C111 コンポーネントには、MicroC/OS- II ランタイム環境が必要です。詳細については、[9-1 ページの「イーサネットと Lightweight IP」](#)を参照してください。



その他にも、ここに記載していないペリフェラルが、サードパーティ・ベンダから提供されています。Nios II プロセッサで利用可能なその他のペリフェラルは、アルテラのWebサイトwww.altera.comをご覧ください。

(アルテラおよびサードパーティ・ベンダの両方が提供する)すべてのペリフェラルは、ハードウェアに対するペリフェラルの低レベル・インタフェースを定義したヘッダ・ファイルを提供する必要があります。このため、すべてのペリフェラルはある程度 HAL をサポートしています。ただし、デバイス・ドライバを提供していないペリフェラルもあります。ドライバを入手できない場合は、ヘッダ・ファイルに提供された定義のみを使用して、ハードウェアにアクセスしてください。ハード・コード化されたアドレスやその他の「マジック・ナンバ」を使用してペリフェラルにアクセスすることは、絶対にしないでください。

特定のペリフェラルには、汎用 API では捕捉できない使用条件を持つハードウェア固有の機能が必ずあります。HAL システム・ライブラリは、UNIX 形式の `ioctl()` 関数を提供することによって、ハードウェア固有の要求に対応しています。ハードウェア機能はペリフェラルに依存するため、`ioctl()` オプションは、各ペリフェラルの説明書に記載されています。

一部のペリフェラルには、HAL 汎用デバイス・モデルをベースにしている専用のアクセス関数が用意されています。例えば、アルテラは、Nios II プロセッサ・システムで使用するための汎用パラレル I/O (PIO) コアを提供しています。この PIO ペリフェラルは、HAL が提供する汎用デバイス・モデルのどのクラスにも適合しないため、ヘッダ・ファイルと少数の専用アクセス関数のみが用意されています。



ペリフェラルのソフトウェア・サポートの詳細については、当該ペリフェラルの説明書を参照してください。アルテラ提供のペリフェラルの詳細については、Nios II プロセッサ・リファレンス・ハンドブックを参照してください。

