

この資料は英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。こちらの日本語版は参考用としてご利用ください。設計の際には、最新の英語版で内容をご確認ください。

NI152010-8.0.0

### はじめに

この章では、HAL (Hardware Abstraction Layer) アプリケーション・プログラミング・インタフェース (API) 内のすべての関数をアルファベット順に記載しています。各関数について C プロトタイプを示し、簡単に説明します。また、マルチ・スレッド環境で実行するときにスレッド・セーフかどうか、割り込みサービス・ルーチン (ISR) から呼び出せるかどうかについても示します。

ここでは、HAL が提供する機能のみを記載します。HAL システム内部からはすべての **newlib** API も利用できることに注意してください。例えば、**newlib** はここで説明していない `printf()` や、その他の標準 I/O 関数を提供しています。



**newlib** API の詳細は、**newlib** の資料を参照してください。Windows の **Start** メニューから、**Programs**、**Altera**、**Nios II <version>**、**Nios II Documentation** の順にクリックしてください。

この章は、以下の項で構成されています。

- 12-1 ページの「HAL API 関数」

### HAL API 関数

HAL API 関数を以降に示します。

## `_exit()`

プロトタイプ:	<code>void _exit (int exit_code)</code>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<p>Newlib の <code>exit()</code> 関数は、<code>_exit()</code> 関数を呼び出して、現在のプロセスを終了します。一般に、<code>main()</code> が完了したときです。HAL システム内にプロセスは 1 つしかないため、HAL 実装は永久にブロックします。</p> <p>割り込みはディセーブルされないため、ISR は動作し続けることに注意してください。</p> <p>入力引数 <code>exit_code</code> は、無視されます。</p>
戻り値:	—
関連項目:	newlib の資料。Windows の <b>Start</b> メニューから、 <b>Programs</b> 、 <b>Altera</b> 、 <b>Nios II &lt;version&gt;</b> 、 <b>Nios II Documentation</b> の順にクリックしてください。

## `_rename()`

プロトタイプ:	<code>int _rename(char *existing, char* new)</code>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	<input type="radio"/>
ISR からの利用:	<input type="radio"/>
インクルード:	<code>&lt;stdio.h&gt;</code>
説明:	<code>_rename()</code> 関数は、newlib の互換性を確保します。
戻り値:	常にリターン・コード <code>-1</code> を返すと共に、 <code>errno</code> を <code>ENOSYS</code> に設定して失敗します。
関連項目:	newlib の資料。Windows の <b>Start</b> メニューから、 <b>Programs</b> 、 <b>Altera</b> 、 <b>Nios II</b> <code>&lt;version&gt;</code> 、 <b>Nios II Documentation</b> の順にクリックしてください。

## alt\_alarm\_start()

プロトタイプ: 

```
int alt_alarm_start (alt_alarm* alarm,
                    alt_u32   nticks,
                    alt_u32 (*callback) (void* context),
                    void*     context)
```

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ○

インクルード: `<sys/alt_alarm.h>`

説明: `alt_alarm_start()` 関数は、アラーム・コールバックをスケジュールします。「Nios® II ソフトウェア開発ハンドブック」の「HAL を使用したプログラムの開発」の章の「アラーム」の項を参照してください。入力引数 `ntick` は、`callback` 関数を呼び出すまで積算されるシステム・クロック・チック数です。入力引数 `context` は、コールバックが発生したときに入力引数として `callback` に渡されます。

入力 `alarm` は、このアラームを表す構造体へのポインターです。このポインターは、ユーザーが作成する必要があり、少なくともアラームと同じ間有効である必要があります。ただし、`alarm` が指し示す構造体の内容をユーザーが初期化する必要はありません。この処理は、`alt_alarm_start()` 呼び出しによって実行されます。

戻り値: `alt_alarm_start()` の戻り値は、成功した場合はゼロ、それ以外は負の値です。この関数は、システム・クロックが利用できない場合は失敗します。

関連項目: 

```
alt_alarm_stop()
alt_nticks()
alt_sysclk_init()
alt_tick()
alt_ticks_per_second()
gettimeofday()
settimeofday()
times()
usleep()
```

## alt\_alarm\_stop()

プロトタイプ:	<code>void alt_alarm_stop (alt_alarm* alarm)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_alarm.h&gt;</code>
説明:	<code>alt_alarm_stop()</code> 関数を呼び出すと、 <code>alt_alarm_start()</code> 呼び出しによって以前に登録されたアラームをキャンセルできます。入力引数は、以前の <code>alt_alarm_start()</code> 呼び出しに使用したアラーム構造体へのポインターです。  復帰時にアラームがまだアクティブの場合、アラームはキャンセルされます。
戻り値:	—
関連項目:	<code>alt_alarm_start()</code> <code>alt_ticks()</code> <code>alt_sysclk_init()</code> <code>alt_tick()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code> <code>usleep()</code>

## alt\_dcache\_flush()

プロトタイプ: `void alt_dcache_flush (void* start, alt_u32 len)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ○

インクルード: `<sys/alt_cache.h>`

説明: `alt_dcache_flush()` 関数は、アドレス `start` から `len` バイトの長さのメモリ領域に対するデータ・キャッシュを消去（つまり、ダーティ・データをライト・バックして無効化）します。

データ・キャッシュのないプロセッサでは、効果はありません。

戻り値: -

関連項目: `alt_dcache_flush_all()`  
`alt_icache_flush()`  
`alt_icache_flush_all()`  
`alt_remap_cached()`  
`alt_remap_uncached()`  
`alt_uncached_free()`  
`alt_uncached_malloc()`

## alt\_dcache\_flush\_all()

プロトタイプ: `void alt_dcache_flush_all (void)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ○

インクルード: `<sys/alt_cache.h>`

説明: `alt_dcache_flush_all()` 関数は、データ・キャッシュのすべての内容を消去（つまりダーティ・データをライト・バックして無効化）します。

データ・キャッシュのないプロセッサでは、効果はありません。

戻り値: -

関連項目: `alt_dcache_flush()`  
`alt_icache_flush()`  
`alt_icache_flush_all()`  
`alt_remap_cached()`  
`alt_remap_uncached()`  
`alt_uncached_free()`  
`alt_uncached_malloc()`

## alt\_dev\_reg()

プロトタイプ:	<code>int alt_dev_reg(alt_dev* dev)</code>
一般的な呼び出し元:	デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dev.h&gt;</code>
説明:	<p><code>alt_dev_reg()</code> 関数は、デバイスをシステムに登録します。登録が完了すると、標準 I/O 関数を使用して、そのデバイスにアクセスできます。「HAL を使用したプログラムの開発」の章 (Nios II ソフトウェア開発ハンドブック) を参照してください。</p> <p>デバイスが既存のデバイスまたはファイル・システムと競合する名前に登録された場合、システムの動作は不定になります。</p> <p><code>alt_dev_reg()</code> 関数が呼び出された時点で、デバイス・リストを使用しているスレッドが他に存在してはならないという意味で、<code>alt_dev_reg()</code> はスレッド・セーフではありません。実際には、<code>alt_dev_reg()</code> は、シングル・スレッド・モードで動作している間のみ呼び出す必要があります。この関数は <code>alt_sys_init()</code> で起動されたデバイス初期化関数によってのみ呼び出され、このデバイス初期化関数はシングル・スレッド C スタートアップ・コードによってのみ呼び出されることを想定しています。</p>
戻り値:	ゼロの戻り値は成功を示します。負の戻り値は失敗を示します。
関連項目:	<code>alt_fs_reg()</code>

## alt\_dma\_rxchan\_close()

プロトタイプ:	<code>int alt_dma_rxchan_close (alt_dma_rxchan rxchan)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dma.h&gt;</code>
説明:	<code>alt_dma_rxchan_close()</code> 関数は、アプリケーションがダイレクト・メモリ・アクセス (DMA) 受信チャネル <code>rxchan</code> の使用を完了したことをシステムに通知します。現在の実装は常に成功します。
戻り値:	戻り値は、成功した場合はゼロ、それ以外は負の値です。
関連項目:	<code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

## alt\_dma\_rxchan\_depth()

プロトタイプ: `alt_u32 alt_dma_rxchan_depth(alt_dma_rxchan dma)`  
一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ  
スレッド・セーフ: ○  
ISR からの利用: ×  
インクルード: `<sys/alt_dma.h>`  
説明: `alt_dma_rxchan_depth()` 関数は、指定されたDMA送信チャンネルdmaに送信できる受信要求の最大数を返します。

この関数がスレッド・セーフかどうか、またはISRから呼び出せるかどうかは、基本デバイス・ドライバによって決まります。一般には、デバイスに依存しないことを前提とします。

戻り値: 送信できる受信要求の最大数を返します。

関連項目: `alt_dma_rxchan_close()`  
`alt_dma_rxchan_ioctl()`  
`alt_dma_rxchan_open()`  
`alt_dma_rxchan_prepare()`  
`alt_dma_rxchan_reg()`  
`alt_dma_txchan_close()`  
`alt_dma_txchan_ioctl()`  
`alt_dma_txchan_open()`  
`alt_dma_txchan_reg()`  
`alt_dma_txchan_send()`  
`alt_dma_txchan_space()`

## alt\_dma\_rxchan\_ioctl()

プロトタイプ :	<pre>int alt_dma_rxchan_ioctl (alt_dma_rxchan dma,                           int req,                           void* arg)</pre>
一般的な呼び出し元 :	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ :	説明を参照。
ISR からの利用 :	説明を参照。
インクルード :	<b>&lt;sys/alt_dma.h&gt;</b>
説明 :	alt_dma_rxchan_ioctl() 関数は、DMA 受信チャンネル dma 上で DMA I/O 操作を実行します。I/O 操作はデバイス固有です。例えば、一部の DMA ドライバは転送操作の幅を制御するオプションをサポートしています。入力引数 req は要求される操作の列挙、arg は要求に対する追加引数です。arg の解釈は要求によって異なります。
	<b>表 12-1</b> に、 <b>&lt;sys/alt_dma.h&gt;</b> で定義されている一般的な要求を示します。DMA デバイスはこれらの要求をサポートできます。
	alt_dma_rxchan_ioctl の呼び出しがスレッド・セーフかどうか、または ISR から呼び出せるかどうかは、デバイスによって決まります。一般には、デバイスに依存しないことを前提とします。
	DMA 転送のペンディング中に alt_dma_rxchan_ioctl() 関数を呼び出さないでください。予期できない動作が発生することがあります。
	Altera® DMA コントローラ・コアに関するデバイス固有の情報については、「Quartus® II ハンドブック Volume 5」の「Avalon インタフェース対応 DMA コントローラ・コア」の章を参照してください。
戻り値 :	負の戻り値は失敗を示します。それ以外の場合、戻り値の解釈は要求によって異なります。
関連項目 :	<pre>alt_dma_rxchan_close() alt_dma_rxchan_depth() alt_dma_rxchan_open() alt_dma_rxchan_prepare() alt_dma_rxchan_reg() alt_dma_txchan_close() alt_dma_txchan_ioctl() alt_dma_txchan_open() alt_dma_txchan_reg() alt_dma_txchan_send() alt_dma_txchan_space()</pre>

表 12-1. 一般的な要求	
要求	意味
ALT_DMA_SET_MODE_8	8 ビット単位でデータを送信します。arg の値は無視されます。
ALT_DMA_SET_MODE_16	16 ビット単位でデータを送信します。arg の値は無視されます。
ALT_DMA_SET_MODE_32	32 ビット単位でデータを送信します。arg の値は無視されます。
ALT_DMA_SET_MODE_64	64 ビット単位でデータを送信します。arg の値は無視されます。
ALT_DMA_SET_MODE_128	128 ビット単位でデータを送信します。arg の値は無視されます。
ALT_DMA_GET_MODE	送信幅を返します。arg の値は無視されます。
ALT_DMA_TX_ONLY_ON (1)	The ALT_DMA_TX_ONLY_ON 要求によって、DMA チャンネルはトランスミッタのみがソフトウェアで制御されるモードで動作します。他端は1つの位置から連続して書き込みます。書き込み先のアドレスはこの要求の引数です。
ALT_DMA_TX_ONLY_OFF (1)	DMA の受信側と送信側の両方をソフトウェアで制御可能なデフォルト・モードに戻ります。
ALT_DMA_RX_ONLY_ON (1)	ALT_DMA_RX_ONLY_ON 要求によって、DMA チャンネルはレシーバのみがソフトウェアで制御されるモードで動作します。他端は1つの位置から連続して読み込まれます。読み込むアドレスはこの要求の引数です。
ALT_DMA_RX_ONLY_OFF (1)	DMA の受信側と送信側の両方をソフトウェアで制御可能なデフォルト・モードに戻ります。

## 表 12-1 の注：

- (1) Nios II エンベデッド・デザイン・スイート (EDS) のバージョン 1.1 では、これらのマクロ名は変更されています。古い名前 (ALT\_DMA\_TX\_STREAM\_ON、ALT\_DMA\_TX\_STREAM\_OFF、ALT\_DMA\_RX\_STREAM\_ON、および ALT\_DMA\_RX\_STREAM\_OFF) は現在でも有効ですが、新規デザインでは新しい名前を使用してください。

## alt\_dma\_rxchan\_open()

プロトタイプ:	<code>alt_dma_rxchan alt_dma_rxchan_open (const char* name)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dma.h&gt;</code>
説明:	<code>alt_dma_rxchan_open()</code> 関数は、DMA受信チャンネルの <code>alt_dma_rxchan</code> ディスクリプタを取得します。入力引数 <code>name</code> は、 <code>/dev/dma_0</code> など、関連付けられた物理デバイスの名前です。
戻り値:	戻り値は、失敗した場合は <code>null</code> 、そうでない場合は <code>null</code> 以外の値です。エラーが発生した場合、 <code>errno</code> は <code>ENODEV</code> に設定されます。
関連項目:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

## alt\_dma\_rxchan\_prepare()

プロトタイプ: 

```
int alt_dma_rxchan_prepare (alt_dma_rxchan dma,
                             void* data,
                             alt_u32 length,
                             alt_rxchan_done* done,
                             void* handle)
```

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: 説明を参照。

ISR からの利用: 説明を参照。

インクルード: `<sys/alt_dma.h>`

説明: `alt_dma_rxchan_prepare()` は、DMA 受信チャンネルに受信要求を送信します。入力引数 `dma` は使用するチャンネル、`data` はデータの受信先を示すポインタ、`length` は受信するデータのバイト単位の最大長、`done` はデータが受信されると呼び出しされるコールバック関数、`handle` は `done` に渡される不定値です。

この関数がスレッド・セーフかどうか、または ISR から呼び出せるかどうかは、基本デバイス・ドライバによって決まります。一般には、デバイスに依存しないことを前提とします。

戻り値: 戻り値は、要求を送信できない場合は負の値、それ以外はゼロです。

関連項目: `alt_dma_rxchan_close()`  
`alt_dma_rxchan_depth()`  
`alt_dma_rxchan_ioctl()`  
`alt_dma_rxchan_open()`  
`alt_dma_rxchan_reg()`  
`alt_dma_txchan_close()`  
`alt_dma_txchan_ioctl()`  
`alt_dma_txchan_open()`  
`alt_dma_txchan_reg()`  
`alt_dma_txchan_send()`  
`alt_dma_txchan_space()`

## alt\_dma\_rxchan\_reg()

プロトタイプ: `int alt_dma_rxchan_reg (alt_dma_rxchan_dev* dev)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<sys/alt_dma_dev.h>`

説明: `alt_dma_rxchan_reg()` 関数は、DMA 受信チャンネルをシステムに登録します。登録が完了すると、「Nios II ソフトウェア開発ハンドブック」の /HAL を使用したプログラムの開発」の章の「DMA 受信チャンネル」で説明する関数を使用して、デバイスにアクセスできます。

チャンネルが既存のチャンネルと競合する名前で登録された場合、システムの動作は不定になります。

`alt_dma_rxchan_reg()` 関数が呼び出された時点で他のスレッドがチャンネル・リストを使用している場合、`alt_dma_rxchan_reg()` はスレッド・セーフではありません。実際には、`alt_dma_rxchan_reg()` は、シングル・スレッド・モードで動作している間のみ呼び出します。`alt_sys_init()` によって起動されたデバイス初期化関数でのみ呼び出してください。デバイス初期化関数は、シングル・スレッド C スタートアップ・コードでのみ呼び出してください。

戻り値: ゼロの戻り値は成功を示します。負の戻り値は失敗を示します。

関連項目: `alt_dma_rxchan_close()`  
`alt_dma_rxchan_depth()`  
`alt_dma_rxchan_ioctl()`  
`alt_dma_rxchan_open()`  
`alt_dma_rxchan_prepare()`  
`alt_dma_txchan_close()`  
`alt_dma_txchan_ioctl()`  
`alt_dma_txchan_open()`  
`alt_dma_txchan_reg()`  
`alt_dma_txchan_send()`  
`alt_dma_txchan_space()`

## alt\_dma\_txchan\_close()

プロトタイプ:	<code>int alt_dma_txchan_close (alt_dma_txchan txchan)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dma.h&gt;</code>
説明:	<code>alt_dma_txchan_close</code> 関数は、アプリケーションが DMA 送信チャンネル <code>txchan</code> の使用を完了したことをシステムに通知します。現在の実装は常に成功します。
戻り値:	戻り値は、成功した場合はゼロ、それ以外は負の値です。
関連項目:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

## alt\_dma\_txchan\_ioctl()

プロトタイプ：  

```
int alt_dma_txchan_ioctl (alt_dma_txchan dma,
                          int req,
                          void* arg)
```

一般的な呼び出し元：  
C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ：  
説明を参照。

ISR からの利用：  
説明を参照。

インクルード：  
<sys/alt\_dma.h>

説明：  
alt\_dma\_txchan\_ioctl() 関数は、DMA 送信チャンネル dma 上でデバイス固有の I/O 操作を実行します。例えば、一部のドライバは転送操作の幅を制御するオプションをサポートしています。入力引数 req は要求される操作の列挙、arg は要求に対する追加引数です。arg の解釈は要求によって異なります。

デバイスがサポートできる一般的な要求については、表 12-1 を参照してください。

alt\_dma\_txchan\_ioctl() の呼び出しがスレッド・セーフかどうか、または ISR から呼び出せるかどうかは、デバイスによって決まります。一般には、デバイスに依存しないことを前提とします。

DMA 転送のベンディング中に alt\_dma\_rxchan\_ioctl() 関数を呼び出さないでください。予期できない動作が発生することがあります。

戻り値：  
負の戻り値は失敗を示します。それ以外の場合、戻り値の解釈は要求によって異なります。

関連項目：  
alt\_dma\_rxchan\_close()  
alt\_dma\_rxchan\_depth()  
alt\_dma\_rxchan\_ioctl()  
alt\_dma\_rxchan\_open()  
alt\_dma\_rxchan\_prepare()  
alt\_dma\_rxchan\_reg()  
alt\_dma\_txchan\_close()  
alt\_dma\_txchan\_open()  
alt\_dma\_txchan\_reg()  
alt\_dma\_txchan\_send()  
alt\_dma\_txchan\_space()

## alt\_dma\_txchan\_open()

プロトタイプ:	<code>alt_dma_txchan alt_dma_txchan_open (const char* name)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dma.h&gt;</code>
説明:	<code>alt_dma_txchan_open()</code> 関数は、DMA送信チャンネルの <code>alt_dma_txchan()</code> ディスクリプタを取得します。入力引数 <code>name</code> は、 <code>/dev/dma_0</code> など、関連付けられた物理デバイスの名前です。
戻り値:	戻り値は、失敗した場合は <code>null</code> 、そうでない場合は <code>null</code> 以外の値です。エラーが発生した場合、 <code>errno</code> は <code>ENODEV</code> に設定されます。
関連項目:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

## alt\_dma\_txchan\_reg()

プロトタイプ:	<code>int alt_dma_txchan_reg (alt_dma_txchan_dev* dev)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dma_dev.h&gt;</code>
説明:	<p><code>alt_dma_txchan_reg()</code> 関数は、DMA 送信チャンネルをシステムに登録します。登録が完了すると、「Nios II ソフトウェア開発ハンドブック」の「HAL を使用したプログラムの開発」の章の「DMA 送信チャンネル」で説明する関数を使用して、デバイスにアクセスできます。</p> <p>チャンネルが既存のチャンネルと競合する名前で登録された場合、システムの動作は不定になります。</p> <p><code>alt_dma_txchan_reg()</code> 関数が呼び出された時点で他のスレッドがチャンネル・リストを使用している場合、<code>alt_dma_txchan_reg()</code> はスレッド・セーフではありません。<code>alt_dma_txchan_reg()</code> は、シングル・スレッド・モードで動作している間のみ呼び出してください。<code>alt_sys_init()</code> によって起動されたデバイス初期化関数でのみ呼び出してください。このデバイス初期化関数は、シングル・スレッド C スタートアップ・コードでのみ呼び出してください。</p>
戻り値:	ゼロの戻り値は成功を示します。負の戻り値は失敗を示します。
関連項目:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_send()</code> <code>alt_dma_txchan_space()</code>

## alt\_dma\_txchan\_send()

プロトタイプ:

```
int alt_dma_txchan_send (alt_dma_txchan  dma,
                        const void*      from,
                        alt_u32          length,
                        alt_txchan_done* done,
                        void*            handle)
```

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: 説明を参照。

ISR からの利用: 説明を参照。

インクルード: <sys/alt\_dma.h>

説明: alt\_dma\_txchan\_send() 関数は、DMA送信チャンネルに送信要求を送信します。入力引数 dma は使用するチャンネル、from は送信するデータの先頭を指すポインター、length は送信するデータのバイト単位の長さ、done はデータが送信されると呼び出されるコールバック関数、handle は done に渡される不定値です。

この関数がスレッド・セーフかどうか、または ISR から呼び出せるかどうかは、基本デバイス・ドライバによって決まります。一般には、デバイスに依存しないことを前提とします。

戻り値: 戻り値は、要求を送信できない場合は負の値、それ以外はゼロです。

関連項目:

```
alt_dma_rxchan_close()
alt_dma_rxchan_depth()
alt_dma_rxchan_ioctl()
alt_dma_rxchan_open()
alt_dma_rxchan_prepare()
alt_dma_rxchan_reg()
alt_dma_txchan_close()
alt_dma_txchan_ioctl()
alt_dma_txchan_open()
alt_dma_txchan_reg()
alt_dma_txchan_space()
```

## alt\_dma\_txchan\_space()

プロトタイプ:	<code>int alt_dma_txchan_space (alt_dma_txchan dma)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	説明を参照。
ISR からの利用:	説明を参照。
インクルード:	<code>&lt;sys/alt_dma.h&gt;</code>
説明:	<code>alt_dma_txchan_space()</code> 関数は、指定されたDMA送信チャンネルdmaに送信できる送信要求の数を返します。負の戻り値は数を決定できないことを示します。  この関数がスレッド・セーフかどうか、またはISRから呼び出せるかどうかは、基本デバイス・ドライバによって決まります。一般には、デバイスに依存しないことを前提とします。
戻り値:	送信できる転送要求の数を返します。
関連項目:	<code>alt_dma_rxchan_close()</code> <code>alt_dma_rxchan_depth()</code> <code>alt_dma_rxchan_ioctl()</code> <code>alt_dma_rxchan_open()</code> <code>alt_dma_rxchan_prepare()</code> <code>alt_dma_rxchan_reg()</code> <code>alt_dma_txchan_close()</code> <code>alt_dma_txchan_ioctl()</code> <code>alt_dma_txchan_open()</code> <code>alt_dma_txchan_reg()</code> <code>alt_dma_txchan_send()</code>



## alt\_exception\_cause\_generated\_bad\_addr()

プロトタイプ:	<pre>int alt_exception_cause_generated_bad_addr     ( alt_exception_cause cause)</pre>
一般的な呼び出し元:	命令関連の例外ハンドラ
スレッド・セーフ:	
ISR からの利用:	
インクルード:	<code>&lt;sys/alt_exceptions.h&gt;</code>
説明:	<p>この関数は、命令関連の例外ハンドラに対する <code>bad_addr</code> 引数を検証します。ハンドラの <code>cause</code> 引数を解析して、例外を発生させたアドレスが <code>bad_addr</code> レジスタに格納されているかどうかを判断します。例外が <code>bad_addr</code> に有効なアドレスを生成するタイプのものの場合、この関数はゼロ以外の値を返します。そうでない場合は、ゼロを返します。Nios II プロセッサ・コアで <code>cause</code> レジスタが未実装の場合、この関数は常にゼロを返します。</p> <p>命令関連の例外ハンドラを介して高度な例外がサポートされます。命令関連の例外ハンドラを実装するには、<code>hal.enable_instruction_related_exceptions_api</code> BSP 設定を有効にする必要があります。高度な例外のサポートは、Nios II ソフトウェア・ビルド・ツールの開発フローでのみ利用できます。</p>
戻り値:	<p>ゼロ以外の値は、<code>bad_addr</code> に例外を発生させたアドレスが格納されていることを示します。</p> <p>ゼロは、<code>bad_addr</code> の値を無視できることを示します。</p>
関連項目:	<code>alt_instruction_exception_register()</code>

## alt\_flash\_close\_dev()

プロトタイプ:	<code>void alt_flash_close_dev(alt_flash_fd* fd)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_flash.h&gt;</code>
説明:	<p><code>alt_flash_close_dev()</code> 関数は、フラッシュ・デバイスをクローズします。それ以降は、このフラッシュ・デバイスに対する <code>alt_write_flash()</code>、<code>alt_read_flash()</code>、<code>alt_get_flash_info()</code>、<code>alt_erase_flash_block()</code>、および <code>alt_write_flash_block()</code> 呼び出しはすべて失敗します。</p> <p><code>alt_flash_close_dev()</code> 関数は、シングル・スレッド・モードで動作しているときにのみ呼び出してください。</p> <p><code>fd</code> パラメータの値は、<code>alt_flash_open_dev</code> 関数から返された値のみ有効です。その他の値が渡された場合、この関数の動作は不定です。</p>
戻り値:	—
関連項目:	<code>alt_erase_flash_block()</code> <code>alt_flash_open_dev()</code> <code>alt_get_flash_info()</code> <code>alt_read_flash()</code> <code>alt_write_flash()</code> <code>alt_write_flash_block()</code>

## alt\_flash\_open\_dev()

プロトタイプ:	<code>alt_flash_fd* alt_flash_open_dev(const char* name)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_flash.h&gt;</code>
説明:	<p><code>alt_flash_open_dev()</code> 関数は、フラッシュ・デバイスをオープンします。オープンしたフラッシュ・デバイスに対しては、<code>alt_write_flash()</code> を使用した書き込みや、<code>alt_read_flash()</code> を使用した読み込みが可能です。また個々のフラッシュ・ブロックは、<code>alt_get_flash_info()</code>、<code>alt_erase_flash_block()</code>、または <code>alt_write_flash_block()</code> 関数を使用して制御できます。</p> <p><code>alt_flash_open_dev</code> 関数は、シングル・スレッド・モードで動作しているときにのみ呼び出してください。</p>
戻り値:	ゼロの戻り値は失敗を示します。それ以外の値は成功を示します。
関連項目:	<code>alt_erase_flash_block()</code> <code>alt_flash_close_dev()</code> <code>alt_get_flash_info()</code> <code>alt_read_flash()</code> <code>alt_write_flash()</code> <code>alt_write_flash_block()</code>

## alt\_fs\_reg()

プロトタイプ:	<code>int alt_fs_reg (alt_dev* dev)</code>
一般的な呼び出し元:	デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_dev.h&gt;</code>
説明:	<p><code>alt_fs_reg()</code> 関数は、ファイル・システムを HAL に登録します。登録が完了すると、標準 I/O 関数を使用してファイル・システムにアクセスできます。「HAL を使用したプログラムの開発」の章 (Nios II ソフトウェア開発ハンドブック) を参照してください。</p> <p>ファイル・システムが、既存のデバイスまたはファイル・システムと競合する名前登録された場合、システムの動作は不定になります。</p> <p><code>alt_fs_reg()</code> が呼び出された時点で他のスレッドがデバイス・リストを使用している場合、<code>alt_fs_reg()</code> はスレッド・セーフではありません。実際には、<code>alt_fs_reg()</code> は、シングル・スレッド・モードで動作している間のみ呼び出してください。この関数は <code>alt_sys_init()</code> で起動されたデバイス初期化関数によってのみ呼び出され、このデバイス初期化関数はシングル・スレッド C スタートアップ・コードによってのみ呼び出されることを想定しています。</p>
戻り値:	ゼロの戻り値は成功を示します。負の戻り値は失敗を示します。
関連項目:	<code>alt_dev_reg()</code>

## alt\_get\_flash\_info()

プロトタイプ: `int alt_get_flash_info(alt_flash_fd* fd,  
flash_region** info,  
int* number_of_regions)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<sys/alt_flash.h>`

説明: `alt_get_flash_info()` 関数は、フラッシュ部分の消去領域の詳細情報を取得します。フラッシュ部分は記述子 `fd` によって指定され、`flash_region` 構造体の開始位置へのポインターは `info` パラメータ内に返され、さらにフラッシュ領域の数は `number of regions` に返されます。

この関数は、シングル・スレッド・モードで動作しているときにのみ呼び出ししてください。

`fd` パラメータの値は、`alt_flash_open_dev` 関数から返された値のみ有効です。その他の値が渡された場合、この関数の動作は不定です。

戻り値: ゼロの戻り値は成功を示します。負の戻り値は失敗を示します。

関連項目: `alt_erase_flash_block()`  
`alt_flash_close_dev()`  
`alt_flash_open_dev()`  
`alt_read_flash()`  
`alt_write_flash()`  
`alt_write_flash_block()`

## alt\_icache\_flush()

プロトタイプ: `void alt_icache_flush (void* start, alt_u32 len)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ○

インクルード: `<sys/alt_cache.h>`

説明: `alt_icache_flush()` 関数は、アドレス `start` から始まる長さ `len` バイトのメモリ領域の命令キャッシュを無効化します。

命令キャッシュのないプロセッサでは、効果はありません。

戻り値: -

関連項目: `alt_dcache_flush()`  
`alt_dcache_flush_all()`  
`alt_icache_flush_all()`  
`alt_remap_cached()`  
`alt_remap_uncached()`  
`alt_uncached_free()`  
`alt_uncached_malloc()`

## alt\_icache\_flush\_all()

プロトタイプ: `void alt_icache_flush_all (void)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ○

インクルード: `<sys/alt_cache.h>`

説明: `alt_icache_flush_all()` 関数は、命令キャッシュのすべての内容を無効化します。

命令キャッシュのないプロセッサでは、効果はありません。

戻り値: -

関連項目: `alt_dcache_flush()`  
`alt_dcache_flush_all()`  
`alt_icache_flush()`  
`alt_remap_cached()`  
`alt_remap_uncached()`  
`alt_uncached_free()`  
`alt_uncached_malloc()`

## alt\_instruction\_exception\_register()

プロトタイプ:	<pre>void alt_instruction_exception_register (     alt_exception_result (*handler)     ( alt_exception_cause cause,       alt_u32             exception_pc,       alt_u32             bad_addr ))</pre>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	○
インクルード:	<sys/alt_exceptions.h>
説明:	<p>HAL API 関数 <code>alt_instruction_exception_register()</code> は、命令関連の例外ハンドラを登録します。handler 引数は、命令関連の例外ハンドラへのポインターです。</p> <p>このAPI関数はBSPの<code>hal.enable_instruction_related_exceptions_api</code>設定を有効にしている場合にのみ使用できます。詳細は、「<a href="#">Nios IIソフトウェア開発ハンドブック</a>」の「<a href="#">Nios II Software Build Tools Reference</a>」の章の「<a href="#">Settings</a>」を参照してください。</p> <p>命令関連の例外ハンドラは、できるだけ早く関数<code>main()</code>で登録してください。これによって、起動時の異常な状態を処理できます。</p> <p>例外ハンドラは<code>alt_main()</code>関数から登録できます。</p> <p><code>alt_instruction_exception_register()</code> を呼び出すと、以前に登録された例外ハンドラがある場合は、それを置き換えます。handler を null に設定した場合、命令関連の例外ハンドラは削除されます。</p> <p>詳細な使用方法については、「<a href="#">Nios IIソフトウェア開発ハンドブック</a>」の「<a href="#">例外処理</a>」の章を参照してください。</p> <p>命令関連の例外ハンドラを介して高度な例外がサポートされます。命令関連の例外ハンドラを実装するには、<code>hal.enable_instruction_related_exceptions_api</code> BSP 設定を有効にする必要があります。高度な例外のサポートは、Nios II ソフトウェア・ビルド・ツールの開発フローでのみ利用できます。</p>
戻り値:	—
関連項目:	<code>alt_irq_register()</code> <code>alt_exception_cause_generated_bad_addr()</code>

## alt\_irq\_disable()

プロトタイプ: `int alt_irq_disable (alt_u32 id)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ×

インクルード: `<sys/alt_irq.h>`

説明: `alt_irq_disable()` 関数は 1 つの割り込みをディセーブルします。

戻り値: 戻り値はゼロです。

関連項目: `alt_irq_disable_all()`  
`alt_irq_enable()`  
`alt_irq_enable_all()`  
`alt_irq_enabled()`  
`alt_irq_register()`

## alt\_irq\_disable\_all()

プロトタイプ:	<code>alt_irq_context alt_irq_disable_all (void)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_irq.h&gt;</code>
説明:	<code>alt_irq_disable_all()</code> 関数はすべての割り込みをディセーブルします。
戻り値:	これに続く <code>alt_irq_enable_all()</code> 呼び出しに、戻り値を入力引数として渡します。
関連項目:	<code>alt_irq_disable()</code> <code>alt_irq_enable()</code> <code>alt_irq_enable_all()</code> <code>alt_irq_enabled()</code> <code>alt_irq_register()</code>

## alt\_irq\_enable()

プロトタイプ: `int alt_irq_enable (alt_u32 id)`

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ×

インクルード: `<sys/alt_irq.h>`

説明: `alt_irq_enable()` 関数は 1 つの割り込みをイネーブルします。

戻り値: 戻り値はゼロです。

関連項目: `alt_irq_disable()`  
`alt_irq_disable_all()`  
`alt_irq_enable_all()`  
`alt_irq_enabled()`  
`alt_irq_register()`

## alt\_irq\_enable\_all()

プロトタイプ:	<code>void alt_irq_enable_all (alt_irq_context context)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_irq.h&gt;</code>
説明:	<code>alt_irq_enable_all()</code> 関数は、 <code>alt_irq_disable_all()</code> によって前にディセーブルしたすべての割り込みをイネーブルします。入力引数 <code>context</code> は、前の <code>alt_irq_disable_all()</code> の呼び出しによって返された値です。 <code>context</code> を使用することにより、 <code>alt_irq_disable_all()</code> および <code>alt_irq_enable_all()</code> のネストした呼び出しを実行できます。その結果 <code>alt_irq_enable_all()</code> は、 <code>alt_irq_disable()</code> によって明示的にディセーブルした割り込みなど、すべての割り込みをイネーブルするわけではありません。
戻り値:	—
関連項目:	<code>alt_irq_disable()</code> <code>alt_irq_disable_all()</code> <code>alt_irq_enable()</code> <code>alt_irq_enabled()</code> <code>alt_irq_register()</code>

## alt\_irq\_enabled()

プロトタイプ:	<code>int alt_irq_enabled (void)</code>
一般的な呼び出し元:	デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_irq.h&gt;</code>
説明:	The <code>alt_irq_enabled()</code> 関数
戻り値:	割り込みがディセーブルされた場合はゼロを返し、そうでない場合はゼロ以外の値を返します。
関連項目:	<code>alt_irq_disable()</code> <code>alt_irq_disable_all()</code> <code>alt_irq_enable()</code> <code>alt_irq_enable_all()</code> <code>alt_irq_register()</code>

## alt\_irq\_register()

プロトタイプ: 

```
int alt_irq_register (alt_u32 id,
                    void* context,
                    void (*isr)(void*, alt_u32))
```

一般的な呼び出し元: デバイス・ドライバ

スレッド・セーフ: ○

ISR からの利用: ×

インクルード: `<sys/alt_irq.h>`

説明: `alt_irq_register()` 関数は ISR を登録します。この関数が成功すると、要求された割り込みは関数の戻り時にイネーブルになります。入力引数 `id` はイネーブルする割り込み、`isr` は割り込みがアクティブなときに呼び出される関数、`context` および `id` は、`isr` の 2 つの入力引数です。

`alt_irq_register()` が呼び出されると、登録された割り込み `id` のハンドラと置き換わります。

`irq_handler` が `null` に設定されている場合、割り込みはディセーブルされません。

戻り値: `alt_irq_register()` 関数は成功した場合はゼロを返し、そうでない場合はゼロ以外の値を返します。

関連項目: 

```
alt_irq_disable()
alt_irq_disable_all()
alt_irq_enable()
alt_irq_enable_all()
alt_irq_enabled()
```

## alt\_llist\_insert()

プロトタイプ:	<pre>void alt_llist_insert(alt_llist* list,                     alt_llist* entry)</pre>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_llist.h&gt;</code>
説明:	<code>alt_llist_insert()</code> 関数は、二重のリンク・リスト・エントリ <code>entry</code> をリスト <code>list</code> に挿入します。これは、リエントリ操作できません。例えば、リストが他のスレッドから操作される、またはアプリケーション・コードと ISR の両方から操作される可能性がある場合、リストへのアクセスを保護する何らかのメカニズムが必要です。割り込みはロックすることができ、また MicroC/OS-II では <code>mutex</code> が使用できます。
戻り値:	-
関連項目:	<code>alt_llist_remove()</code>

## alt\_llist\_remove()

プロトタイプ:	<code>void alt_llist_remove (alt_llist* entry)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_llist.h&gt;</code>
説明:	<code>alt_llist_remove()</code> 関数は、二重のリンク・リスト・エントリ <code>entry</code> を、現在そのエントリがメンバーであるリストから除去します。これは、リエントリ操作できません。例えば、リストが他のスレッドから操作される、またはアプリケーション・コードと ISR の両方から操作される可能性がある場合、リストへのアクセスを保護する何らかのメカニズムが必要です。割り込みはロックすることができ、また MicroC/OS-II では <code>mutex</code> が使用できます。
戻り値:	—
関連項目:	<code>alt_llist_insert()</code>



## alt\_nticks()

プロトタイプ:	<code>alt_u32 alt_nticks (void)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_alarm.h&gt;</code>
説明:	<code>alt_nticks()</code> 関数
戻り値:	リセット時点から経過したシステム・クロック・チック数を返します。システム・クロックが利用できない場合はゼロを返します。
関連項目:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_sysclk_init()</code> <code>alt_tick()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code> <code>usleep()</code>

## alt\_read\_flash()

プロトタイプ:	<pre>int alt_read_flash(alt_flash_fd* fd,                   int          offset,                   void*        dest_addr,                   int          length)</pre>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<sys/alt_flash.h>
説明:	<p>alt_read_flash() 関数は、フラッシュからデータを読み出します。フラッシュ fd から length バイト（フラッシュの先頭から offset バイトで）が読み込まれ、dest_addr 位置に書き込まれます。</p> <p>この関数は、シングル・スレッド・モードで動作しているときのみ呼び出してください。</p> <p>fd パラメータの値は、alt_flash_open_dev 関数から返された値のみ有効です。その他の値が渡された場合、この関数の動作は不定です。</p>
戻り値:	戻り値は、成功した場合はゼロ、そうでない場合はゼロ以外の値です。
関連項目:	<pre>alt_erase_flash_block() alt_flash_close_dev() alt_flash_open_dev() alt_get_flash_info() alt_write_flash() alt_write_flash_block()</pre>

## alt\_remap\_cached()

プロトタイプ:	<pre>void* alt_remap_cached (volatile void* ptr,                         alt_u32      len);</pre>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_cache.h&gt;</code>
説明:	<p>alt_remap_cached() 関数は、キャッシュ・アクセスのためのメモリ領域を再マップします。マップするメモリは、アドレス ptr から len バイトです。</p> <p>データ・キャッシュのないプロセッサは、非キャッシュ・メモリを返します。</p>
戻り値:	この関数の戻り値は再マップされたメモリ領域です。
関連項目:	<pre>alt_dcachel_flush() alt_dcachel_flush_all() alt_icachel_flush() alt_icachel_flush_all() alt_remap_uncached() alt_uncached_free() alt_uncached_malloc()</pre>

## alt\_remap\_uncached()

プロトタイプ:	<pre>volatile void* alt_remap_uncached (void* ptr,                                    alt_u32 len);</pre>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<b>&lt;sys/alt_cache.h&gt;</b>
説明:	<p>alt_remap_uncached() 関数は、非キャッシュ・アクセスのためのメモリ領域を再マップします。マップするメモリは、アドレス ptr から len バイトです。</p> <p>データ・キャッシュのないプロセッサは、非キャッシュ・メモリを返します。</p>
戻り値:	この関数の戻り値は、再マップされたメモリ領域です。
関連項目:	<pre>alt_dcache_flush() alt_dcache_flush_all() alt_icache_flush() alt_icache_flush_all() alt_remap_cached() alt_uncached_free() alt_uncached_malloc()</pre>

## alt\_sysclk\_init()

プロトタイプ:	<code>int alt_sysclk_init (alt_u32 nticks)</code>
一般的な呼び出し元:	デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_alarm.h&gt;</code>
説明:	<code>alt_sysclk_init()</code> 関数は、システム・クロック・ドライバを登録します。入力引数は、システム・クロック動作時の 1 秒あたりのチック数です。  この関数は、 <code>alt_sys_init()</code> から、つまりシステムがシングル・スレッド・モードで動作している間のみ呼び出されると想定されます。この関数を同時に呼び出すと、予測できない結果になることがあります。
戻り値:	この関数は、成功した場合はゼロを返し、それ以外は負の値を返します。システム・クロック・ドライバが既に登録されている場合、呼び出しは失敗します。
関連項目:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_nticks()</code> <code>alt_tick()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code> <code>usleep()</code>

## alt\_tick()

プロトタイプ:	<code>void alt_tick (void)</code>
一般的な呼び出し元:	デバイス・ドライバ
スレッド・セーフ:	×
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_alarm.h&gt;</code>
説明:	システム・クロック・ドライバしか <code>alt_tick()</code> 関数を呼び出してはなりません。ドライバは、 <code>alt_sys_init()</code> の呼び出しで指定されたレートで、周期的にこの関数を呼び出す必要があります。この関数は、システム・クロック・チックが発生したシステムに通知します。この関数は、システム・クロック・ドライバに対する ISR の一部として動作します。
戻り値:	—
関連項目:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_ticks()</code> <code>alt_sysclk_init()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code> <code>usleep()</code>

## alt\_ticks\_per\_second()

プロトタイプ:	<code>alt_u32 alt_ticks_per_second (void)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/alt_alarm.h&gt;</code>
説明:	<code>alt_ticks_per_second()</code> 関数は、1 秒ごとに経過するシステム・クロック・チック数を返します。システム・クロックが利用できない場合、戻り値はゼロです。
戻り値:	1 秒ごとに経過するシステム・クロック・チック数を返します。
関連項目:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_nticks()</code> <code>alt_sysclk_init()</code> <code>alt_tick()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code> <code>usleep()</code>

## alt\_timestamp()

プロトタイプ:	<code>alt_u32 alt_timestamp (void)</code>
一般的な呼び出し元:	C/C++ プログラム
スレッド・セーフ:	説明を参照。
ISR からの利用:	説明を参照。
インクルード:	<code>&lt;sys/alt_timestamp.h&gt;</code>
説明:	<p><code>alt_timestamp()</code> 関数はタイムスタンプ・カウンタの現在の値を返します。「Nios II ソフトウェア開発ハンドブック」の /HAL を使用したプログラムの開発」の章の「高精度時間測定」の項を参照してください。この関数は、タイムスタンプ・ドライバによって実装されます。したがって、この関数がスレッド・セーフかどうか、割り込みレベルで利用できるかどうかは、基本ドライバによって決まります。</p> <p><code>alt_timestamp_start()</code> 関数は、必ず <code>alt_timestamp()</code> を呼び出す前に呼び出してください。そうしない場合、<code>alt_timestamp()</code> の動作は不定になります。</p>
戻り値:	タイムスタンプ・カウンタの現在の値を返します。
関連項目:	<code>alt_timestamp_freq()</code> <code>alt_timestamp_start()</code>

## alt\_timestamp\_freq()

プロトタイプ:	<code>alt_u32 alt_timestamp_freq (void)</code>
一般的な呼び出し元:	C/C++ プログラム
スレッド・セーフ:	説明を参照。
ISR からの利用:	説明を参照。
インクルード:	<code>&lt;sys/alt_timestamp.h&gt;</code>
説明:	<code>alt_timestamp_freq()</code> 関数は、タイムスタンプ・カウンタが増加するレートを返します。「Nios II ソフトウェア開発ハンドブック」の「HAL を使用したプログラムの開発」の章の「高精度時間測定」の項を参照してください。この関数は、タイムスタンプ・ドライバによって実行されます。したがって、この関数がスレッド・セーフかどうか、割り込みレベルで利用できるかどうかは、基本ドライバによって決まります。
戻り値:	戻り値は、1 秒あたりのカウンタ・チック数です。
関連項目:	<code>alt_timestamp()</code> <code>alt_timestamp_start()</code>

## alt\_timestamp\_start()

プロトタイプ:	<code>int alt_timestamp_start (void)</code>
一般的な呼び出し元:	C/C++ プログラム
スレッド・セーフ:	説明を参照。
ISR からの利用:	説明を参照。
インクルード:	<code>&lt;sys/alt_timestamp.h&gt;</code>
説明:	<p><code>alt_timestamp_start()</code> 関数はシステム・タイムスタンプ・カウンタを開始します。「Nios II ソフトウェア開発ハンドブック」の「HAL を使用したプログラムの開発」の章の「高精度時間測定」の項を参照してください。この関数は、タイムスタンプ・ドライバによって実行されます。したがって、この関数がスレッド・セーフかどうか、割り込みレベルで利用できるかどうかは、基本ドライバによって決まります。</p> <p>この関数は、カウンタをゼロにリセットして、カウンタの動作を開始します。</p>
戻り値:	戻り値は、成功した場合はゼロ、そうでない場合はゼロ以外の値です。
関連項目:	<code>alt_timestamp()</code> <code>alt_timestamp_freq()</code>

## alt\_uncached\_free()

プロトタイプ:	<code>void alt_uncached_free (volatile void* ptr)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_cache.h&gt;</code>
説明:	<code>alt_uncached_free()</code> 関数を実行すると、 <code>ptr</code> が示すメモリが割り当て解除されます。すなわち、 <code>alt_uncached_malloc()</code> 呼び出しによってメモリ割り当てが可能になります。入力ポインタ <code>ptr</code> は、以前に <code>alt_uncached_malloc()</code> 呼び出しによって割り当てられたメモリの領域を指します。そうでない場合、動作は不安定です。
戻り値:	—
関連項目:	<code>alt_dcache_flush()</code> <code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code> <code>alt_icache_flush_all()</code> <code>alt_remap_cached()</code> <code>alt_remap_uncached()</code> <code>alt_uncached_malloc()</code>

## alt\_uncached\_malloc()

プロトタイプ:	<code>volatile void* alt_uncached_malloc (size_t size)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;sys/alt_cache.h&gt;</code>
説明:	<code>alt_uncached_malloc()</code> 関数は、長さ <code>size</code> バイトの非キャッシュ・メモリ領域を割り当てます。これにより割り当てられたメモリの領域は、 <code>alt_uncached_free()</code> 関数を使用して解放できます。  データ・キャッシュのないプロセッサは、非キャッシュ・メモリを返します。
戻り値:	この関数は、十分なメモリを割り当てできない場合は <code>null</code> を返し、それ以外は割り当てた空間へのポインターを返します。
関連項目:	<code>alt_dcache_flush()</code> <code>alt_dcache_flush_all()</code> <code>alt_icache_flush()</code> <code>alt_icache_flush_all()</code> <code>alt_remap_cached()</code> <code>alt_remap_uncached()</code> <code>alt_uncached_free()</code>

## alt\_write\_flash()

プロトタイプ: 

```
int alt_write_flash(alt_flash_fd* fd,
                    int           offset,
                    const void*   src_addr,
                    int           length)
```

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<sys/alt_flash.h>`

説明: `alt_write_flash()` 関数は、データをフラッシュに書き込みます。書き込まれるデータは `src_addr` アドレスにあり、`length` バイト分のデータがフラッシュ `fd` (フラッシュの先頭から `offset` バイト) に書き込まれます。

この関数は、シングル・スレッド・モードで動作しているときにのみ呼び出してください。この関数は、この書き込みによってフラッシュ・セクタの非書き込み領域が影響を受ける場合でも、その領域の内容を保存しません。「Nios II ソフトウェア開発ハンドブック」の「HAL を使用したプログラムの開発」の章の「シンプル・フラッシュ・アクセス」の項を参照してください。

`fd` パラメータの値は、`alt_flash_open_dev` 関数から返された値のみ有効です。その他の値が渡された場合、この関数の動作は不定です。

戻り値: 戻り値は、成功した場合はゼロ、そうでない場合はゼロ以外の値です。

関連項目: `alt_erase_flash_block()`  
`alt_flash_close_dev()`  
`alt_flash_open_dev()`  
`alt_get_flash_info()`  
`alt_read_flash()`  
`alt_write_flash_block()`

## alt\_write\_flash\_block()

プロトタイプ: 

```
int alt_write_flash_block(alt_flash_fd* fd,
                          int           block_offset,
                          int           data_offset,
                          const void    *data,
                          int           length)
```

一般的な呼び出し元: C/C++ プログラム  
デバイス・ドライバ

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: **<sys/alt\_flash.h>**

説明: alt\_write\_flash\_block() 関数は、フラッシュの 1 つの消去ブロックに書き込みます。フラッシュ・デバイスは fd で指定され、block\_offset はフラッシュ内におけるこのブロックの先頭位置のオフセット、data\_offset はフラッシュ内でデータの書き込みを開始する位置のオフセット、data は書き込むデータ、length は書き込むデータ量です。どのパラメータにもチェックが行われないことに注意してください。「Nios II ソフトウェア開発ハンドブック」の「HAL を使用したプログラムの開発」の章の「高精度フラッシュ・アクセス」の項を参照してください。

この関数は、シングル・スレッド・モードで動作しているときにのみ呼び出ししてください。

fd パラメータの値は、alt\_flash\_open\_dev 関数から返された値のみ有効です。その他の値が渡された場合、この関数の動作は不定です。

戻り値: 戻り値は、成功した場合はゼロ、そうでない場合はゼロ以外の値です。

関連項目: alt\_erase\_flash\_block()  
alt\_flash\_close\_dev()  
alt\_flash\_open\_dev()  
alt\_get\_flash\_info()  
alt\_read\_flash()  
alt\_write\_flash()

## close()

プロトタイプ:	<code>int close (int fd)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<code>close()</code> 関数は、ファイル・ディスクリプタ <code>fd</code> をクローズする標準 UNIX 形式の <code>close()</code> 関数です。  操作対象のドライバが提供する <code>close()</code> の実装がスレッド・セーフである場合のみ、 <code>close()</code> 呼び出しはスレッド・セーフになります。  <code>fd</code> パラメータの有効な値は、 <code>stdout</code> 、 <code>stdin</code> 、および <code>stderr</code> 、または <code>open()</code> 呼び出しから返された値です。
戻り値:	戻り値は、成功した場合はゼロ、それ以外は <code>-1</code> です。エラーが発生した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>open()</code> <code>read()</code> <code>stat()</code> <code>write()</code>  newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## execve()

プロトタイプ:	<pre>int execve(const char *path,            char *const argv[],            char *const envp[])</pre>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<unistd.h>
説明:	execve() 関数は、newlibとの互換性を確保するためだけに提供されています。
戻り値:	execve() 呼び出しは、常にリターン・コード -1 で、errno を ENOSYS に設定して失敗します。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## fcntl()

プロトタイプ: `int fcntl(int fd, int cmd)`

一般的な呼び出し元: C/C++ プログラム

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<unistd.h>`  
`<fcntl.h>`

説明: `fcntl()` 関数は、オープン・ファイル・ディスクリプタに関連付けられたフラグの状態を変更できる標準 `fcntl()` システム・コールの限定された実装です。通常、これらのフラグは `open()` 呼び出しの間にセットされます。この関数の主な用途は、デバイスの状態をブロックから非ブロックに変更することです(この機能をサポートしているデバイス・ドライバの場合)。

入力引数 `fd` は操作対象のファイル・ディスクリプタです。`cmd` は実行するコマンドで、`F_GETFL` (フラグの現在の値を返す) または `F_SETFL` (フラグの値をセット) のいずれかです。

戻り値: `cmd` が `F_SETFL` の場合、引数 `arg` はフラグの新しい値です。そうでない場合、`arg` は無視されます。`fcntl()` の呼び出しによって、フラグ `O_APPEND` および `O_NONBLOCK` のみ更新できます。その他のフラグは変更されません。戻り値は、成功した場合はゼロ、それ以外は `-1` です。

`cmd` が `F_GETFL` の場合、戻り値はフラグの現在の値です。エラーがある場合、`-1` が返されます。

エラーが発生した場合、`errno` はエラーの原因を示す値に設定されます。

関連項目: `close()`  
`fstat()`  
`ioctl()`  
`isatty()`  
`lseek()`  
`read()`  
`stat()`  
`write()`  
`newlib` の資料。Windows の Start メニューから、Programs、Altera、Nios II `<version>`、**Nios II Documentation** の順にクリック

## fork()

プロトタイプ:	pid_t fork (void)
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<unistd.h>
説明:	fork() 関数は、newlib との互換性を確保するためだけに提供されています。
戻り値:	fork() 呼び出しは、常にリターン・コード-1 で、errno を ENOSYS に設定して失敗します。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## fstat()

プロトタイプ:	<code>int fstat (int fd, struct stat *st)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<b>&lt;sys/stat.h&gt;</b>
説明:	<p><code>fstat()</code> 関数は、オープン・ファイル・ディスクリプタの機能に関する情報を取得します。基本デバイス・ドライバは、入力 <code>st</code> 構造体にその機能に関する情報を入れます。利用可能なオプションについては、コンパイラとともに提供されるヘッダ・ファイル <b>sys/stat.h</b> を参照してください。</p> <p>基本ドライバが <code>fstat()</code> 関数の独自の実装を提供しない場合、デフォルトではファイル・ディスクリプタはキャラクタ・デバイスとしてマークされます。</p> <p>操作対象のドライバが提供する <code>fstat()</code> の実装がスレッド・セーフである場合のみ、<code>fstat()</code> 呼び出しはスレッド・セーフになります。</p> <p><code>fd</code>パラメータの有効な値は、<code>stdout</code>、<code>stdin</code>、および <code>stderr</code>、または <code>open()</code> 呼び出しから返された値です。</p>
戻り値:	戻り値は、成功した場合はゼロ、それ以外は <code>-1</code> です。呼び出しが失敗した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>close()</code> <code>fcntl()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>open()</code> <code>read()</code> <code>stat()</code> <code>write()</code> newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## getpid()

プロトタイプ:	pid_t getpid (void)
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<unistd.h>
説明:	getpid() 関数は、newlib との互換性を確保するために提供されており、現在のプロセス id を取得します。
戻り値:	HAL システムは複数のプロセスを含むことができないため、getpid() は常に同じ id 番号を返します。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## gettimeofday()

プロトタイプ:	<pre>int gettimeofday(struct timeval *ptimeval,                  struct timezone *ptimezone)</pre>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	○
インクルード:	<b>&lt;sys/time.h&gt;</b>
説明:	<p>gettimeofday() 関数は、現在のウォール・クロック・タイムを示す時間構造体を取得します。この時刻は、システム・クロック・チックの経過数、および最新の settimeofday() 呼び出しによって設定された現在の時刻を使用して計算されます。</p> <p>この関数が、settimeofday() 呼び出しと同時に呼び出された場合、gettimeofday() が返す値の信頼性は低くなりますが、gettimeofday() は同時に呼び出しても問題ありません。</p>
戻り値:	戻り値は、成功した場合はゼロ、それ以外は -1 です。呼び出しが失敗した場合、errno はエラーの原因を示す値に設定されます。
関連項目:	<pre>alt_alarm_start() alt_alarm_stop() alt_nticks() alt_sysclk_init() alt_tick() alt_ticks_per_second() settimeofday() times() usleep()</pre> <p>newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II &lt;version&gt;、<b>Nios II Documentation</b> の順にクリック</p>

## ioctl()

プロトタイプ:	<code>int ioctl (int fd, int req, void* arg)</code>
一般的な呼び出し元:	C/C++ プログラム
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;sys/ioctl.h&gt;</code>
説明:	<p><code>ioctl()</code> 関数を使用すれば、アプリケーション・コードから、デバイス・ドライバの I/O 機能をドライバ固有の方法で操作できます。この関数は、UNIX の標準 <code>ioctl()</code> 関数に相当します。入力引数 <code>fd</code> は操作するデバイスのオープン・ファイル・ディスクリプタ、<code>req</code> は操作要求を定義する列挙です。また <code>arg</code> の解釈は、要求によって異なります。</p> <p>一般に、この実装では、要求を適切なドライバ <code>ioctl()</code> 関数（ドライバの <code>alt_dev</code> 構造体での登録に従って）に振り分けます。ただし、デバイスの場合（ファイル・サブシステムとは異なり）、<code>TIOCEXCL</code> 要求と <code>TIOCXXCL</code> 要求は、ドライバを参照しないで処理されます。これらの要求は、デバイスを排他的アクセスに対してロックおよびリリースします。</p> <p>操作対象のドライバが提供する <code>ioctl()</code> の実装がスレッド・セーフである場合のみ、<code>ioctl()</code> 呼び出しはスレッド・セーフになります。</p> <p><code>fd</code> パラメータの有効な値は、<code>stdout</code>、<code>stdin</code>、および <code>stderr</code>、または <code>open()</code> 呼び出しから返された値です。</p>
戻り値:	戻り値の解釈は要求によって異なります。呼び出しが失敗した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>isatty()</code> <code>lseek()</code> <code>open()</code> <code>read()</code> <code>stat()</code> <code>write()</code> <p><code>newlib</code> の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code>、<b>Nios II Documentation</b> の順にクリック</p>

## isatty()

プロトタイプ:	<code>int isatty(int fd)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<code>isatty()</code> 関数は、オープン・ファイル・ディスクリプタ <code>fd</code> に関連付けられたデバイスが、ターミナル・デバイスかどうかを判断します。この実装では、ドライバ <code>fstat()</code> 関数を使用してその応答を判断します。  操作対象のドライバが提供する <code>fstat()</code> の実装がスレッド・セーフである場合のみ、 <code>isatty()</code> 呼び出しはスレッド・セーフになります。
戻り値:	戻り値は、デバイスがキャラクタ・デバイスの場合は 1、それ以外はゼロです。エラーが発生した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>lseek()</code> <code>open()</code> <code>read()</code> <code>stat()</code> <code>write()</code>  newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## kill()

プロトタイプ:	<code>int kill(int pid, int sig)</code>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;signal.h&gt;</code>
説明:	<p><code>kill()</code> 関数は、プロセスにシグナルを送信するために newlib が使用します。入力引数 <code>pid</code> はシグナルを送信するプロセスの <code>id</code>、<code>sig</code> は送信するシグナルです。HAL にはプロセスが 1 つしか存在しないため、<code>pid</code> に対して有効な値は、<code>getpid()</code> で返される現在のプロセス <code>id</code>、またはブロードキャスト値のみです。つまり、<code>pid</code> はゼロ以下でなければなりません。</p> <p>シグナル SIGABRT、SIGALRM、SIGFPE、SIGILL、SIGKILL、SIGPIPE、SIGQUIT、SIGSEGV、SIGTERM、SIGUSR1、SIGUSR2、SIGBUS、SIGPOLL、SIGPROF、SIGSYS、SIGTRAP、SIGVTALRM、SIGXCPU、または SIGXFSZ が発生すると、システムは <code>exit()</code> を呼び出すことなく、すぐにシャットダウンします。</p> <p>シグナル SIGCHLD および SIGURG は無視されます。</p> <p>その他のシグナルはすべてエラーとして扱われます。</p>
戻り値:	戻り値は、成功した場合はゼロ、それ以外は <code>-1</code> です。呼び出しが失敗した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code> 、 <b>Nios II Documentation</b> の順にクリック

## link()

プロトタイプ:	<pre>int link(const char *_path1,          const char *_path2)</pre>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	<input type="radio"/>
ISR からの利用:	<input type="radio"/>
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	link() 関数は、newlib との互換性を確保するためだけに提供されています。
戻り値:	link() 呼び出しは、常にリターン・コード -1 で、errno を ENOSYS に設定して失敗します。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## lseek()

プロトタイプ:	<code>off_t lseek(int fd, off_t ptr, int whence)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<p><code>lseek()</code> 関数は、ファイル・ディスクリプタ <code>fd</code> に関連付けられた読み出し/書き込みポインタを移動します。この関数は、ファイル・ディスクリプタに関連付けられたドライバが提供する <code>lseek()</code> 関数に呼び出しを振り分けます。ドライバが <code>lseek()</code> の実装を提供しない場合は、エラーになります。</p> <p><code>lseek()</code> は UNIX の標準 <code>lseek()</code> 関数に相当します。</p> <p>入力パラメータ <code>whence</code> に対して、次の値が使用できます。</p> <ul style="list-style-type: none"><li>● <code>whence</code> の値</li><li>● インタプリテーション</li><li>● <code>SEEK_SET</code>— オフセットが <code>ptr</code> バイトに設定されます。</li><li>● <code>SEEK_CUR</code>— オフセットが <code>ptr</code> バイトだけ増加されます。</li><li>● <code>SEEK_END</code>— オフセットがファイルの終端から <code>ptr</code> バイト加えた位置に設定されます。</li></ul> <p>操作対象のドライバが提供する <code>lseek()</code> の実装がスレッド・セーフである場合のみ、<code>lseek()</code> 呼び出しはスレッド・セーフになります。</p> <p><code>fd</code> パラメータの有効な値は、<code>stdout</code>、<code>stdin</code>、および <code>stderr</code>、または <code>open()</code> 呼び出しから返された値です。</p>
戻り値:	成功した場合、戻り値は負でないファイル・ポインタです。エラーが発生した場合、戻り値は <code>-1</code> です。呼び出しが失敗した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>open()</code> <code>read()</code> <code>stat()</code> <code>write()</code> <p>newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II &lt;version&gt;、<b>Nios II Documentation</b> の順にクリック</p>

## open()

プロトタイプ:	<code>int open (const char* pathname, int flags, mode_t mode)</code>
一般的な呼び出し元:	C/C++ プログラム
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code> <code>&lt;fcntl.h&gt;</code>
説明:	<p><code>open()</code> 関数は、ファイルまたはデバイスをオープンし、ファイル・ディスクリプタ (読み出し、書き込みなどに使用する負でない小さな整数) を返します。</p> <p><code>flags</code> は、それぞれリード・オンリー、ライト・オンリー、リード/ライトでファイルのオープンを要求する <code>O_RDONLY</code>、<code>O_WRONLY</code>、<code>O_RDWR</code> のいずれかです。</p> <p>ビット単位の OR フラグを <code>O_NONBLOCK</code> と組み合わせて、ファイルを非ブロック・モードでオープンすることもできます。返されたファイル・ディスクリプタに対する <code>open()</code> 操作または後続の操作によって、呼び出し中のプロセスが待機することはありません。</p> <p>必ずしもすべてのファイル・システム / デバイスが、このオプションを認識しないことに注意してください。</p> <p><code>mode</code> は、新規ファイルが作成された場合に使用許可を指定します。これは現在のファイル・システムでは使用されていませんが、互換性のために維持されています。</p> <p>操作対象のドライバが提供する <code>open()</code> の実装がスレッド・セーフである場合のみ、<code>open()</code> 呼び出しはスレッド・セーフになります。</p>
戻り値:	戻り値は、成功した場合は新しいファイル・ディスクリプタ、それ以外は <code>-1</code> です。エラーが発生した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>read()</code> <code>stat()</code> <code>write()</code> <p><code>newlib</code> の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code>、<b>Nios II Documentation</b> の順にクリック</p>

## read()

プロトタイプ:	<code>int read(int fd, void *ptr, size_t len)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<p><code>read()</code> 関数は、ファイルまたはデバイスからデータのブロックを読み出します。この関数は、入力のオープン・ファイル・ディスクリプタ <code>fd</code> に関連付けられたデバイス・ドライバに要求を振り分けます。入力引数 <code>ptr</code> は読み込まれたデータを格納する位置、<code>len</code> は読み込むデータのバイト長です。</p> <p>操作対象のドライバが提供する <code>read()</code> の実装がスレッド・セーフである場合のみ、<code>read()</code> 呼び出しはスレッド・セーフになります。</p> <p><code>fd</code> パラメータの有効な値は、<code>stdout</code>、<code>stdin</code>、および <code>stderr</code>、または <code>open()</code> 呼び出しから返された値です。</p>
戻り値:	<p>戻り引数は読み込まれたバイト数ですが、これは要求した長さより短い場合もあります。</p> <p>戻り値の <code>-1</code> はエラーを示します。エラーが発生した場合、<code>errno</code> はエラーの原因を示す値に設定されます。</p>
関連項目:	<code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>open()</code> <code>stat()</code> <code>write()</code> <p>newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code>、<b>Nios II Documentation</b> の順にクリック</p>

## sbrk()

プロトタイプ:	<code>caddr_t sbrk(int incr)</code>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	×
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<code>sbrk()</code> 関数はアプリケーションのデータ・セグメントを動的に拡張します。入力引数 <code>incr</code> は割り当てるブロックのサイズです。 <code>sbrk()</code> は直接呼び出さないでください。メモリを動的に割り当てる場合は、 <code>malloc()</code> 関数を使用します。
戻り値:	—
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code> 、 <b>Nios II Documentation</b> の順にクリック

## settimeofday()

プロトタイプ:	<pre>int settimeofday (const struct timeval *t,                   const struct timezone *tz)</pre>
一般的な呼び出し元:	C/C++ プログラム
スレッド・セーフ:	×
ISR からの利用:	○
インクルード:	<b>&lt;sys/time.h&gt;</b>
説明:	settimeofday() 関数が gettimeofday() と同時に呼び出された場合、gettimeofday() が返す値の信頼性は低下します。
戻り値:	戻り値は、成功した場合はゼロ、それ以外は -1 です。現在の実装は常に成功します。
関連項目:	<pre>alt_alarm_start() alt_alarm_stop() alt_nticks() alt_sysclk_init() alt_tick() alt_ticks_per_second() gettimeofday() times() usleep()</pre>

## stat()

プロトタイプ: `int stat(const char *file_name,  
struct stat *buf);`

一般的な呼び出し元: C/C++ プログラム  
newlib C ライブラリ

スレッド・セーフ: 説明を参照。

ISR からの利用: ×

インクルード: **<sys/stat.h>**

説明: `stat()` 関数は、`fstat()` 関数と同様に、ファイルに関するステータス情報を取得します。`fstat()` のように、オープン・ファイル・ディスクリプタを使用する代わりに、`stat()` はファイルの名前を入力引数として受け取ります。

操作対象のドライバが提供する `stat()` の実装がスレッド・セーフである場合のみ、`stat()` 呼び出しはスレッド・セーフになります。

内部では、`stat()` 関数は `fstat()` 呼び出しとして実行されます。[12-58 ページの「fstat\(\)」](#)を参照してください。

戻り値: -

関連項目: `close()`  
`fcntl()`  
`fstat()`  
`ioctl()`  
`isatty()`  
`lseek()`  
`open()`  
`read()`  
`write()`  
newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、**Nios II Documentation** の順にクリック

## times()

プロトタイプ:	<code>clock_t times (struct tms *buf)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/times.h&gt;</code>
説明:	<p>この <code>times()</code> 関数は、Newlib との互換性を確保するために提供されています。リセット後のクロック・チック数を返します。さらに、入力パラメータ <code>buf</code> が示す構造体に時間計算に関する情報を格納します。tms 構造体の定義は以下のとおりです。</p> <pre>typedef struct {     clock_t tms_utime;     clock_t tms_stime;     clock_t tms_cutime;     clock_t tms_cstime; };</pre> <p>この構造体には次の要素が含まれています。</p> <ul style="list-style-type: none"> <li>● <code>tms_utime</code>: ユーザー命令の実行に要した CPU 時間</li> <li>● <code>tms_stime</code>: プロセスに代わってシステムが実行に要した CPU 時間</li> <li>● <code>tms_cutime</code>: 子プロセスのすべての <code>tms_utime</code> および <code>tms_cutime</code> の合計</li> <li>● <code>tms_cstime</code>: 子プロセスの <code>tms_stimes</code> および <code>tms_cstimes</code> の合計</li> </ul> <p>実際には、すべての経過時間はシステム時間として計算されます。ユーザー時間として考慮される時間はありません。また、子プロセスは HAL から生成できないため、子プロセスに時間は割り当てられません。</p>
戻り値:	システム・クロックが利用できない場合、戻り値はゼロです。
関連項目:	<pre>alt_alarm_start() alt_alarm_stop() alt_nticks() alt_sysclk_init() alt_tick() alt_ticks_per_second() gettimeofday() settimeofday() usleep()</pre> <p>newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II &lt;version&gt;、<b>Nios II Documentation</b> の順にクリック</p>

## unlink()

プロトタイプ:	<code>int unlink(char *name)</code>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<unistd.h>
説明:	unlink() 関数は、newlibとの互換性を確保するためだけに提供されています。
戻り値:	unlink() 呼び出しは、常にリターン・コード -1 で、errno を ENOSYS に設定して失敗します。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <version>、 <b>Nios II Documentation</b> の順にクリック

## usleep()

プロトタイプ:	<code>int usleep (int us)</code>
一般的な呼び出し元:	C/C++ プログラム デバイス・ドライバ
スレッド・セーフ:	○
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<code>usleep()</code> 関数は、us マイクロ秒単位で実行を遅延します。
戻り値:	<code>usleep()</code> 関数は、成功した場合はゼロ、それ以外は -1 を返します。エラーが発生した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。現在の実装は常に成功します。
関連項目:	<code>alt_alarm_start()</code> <code>alt_alarm_stop()</code> <code>alt_nticks()</code> <code>alt_sysclk_init()</code> <code>alt_tick()</code> <code>alt_ticks_per_second()</code> <code>gettimeofday()</code> <code>settimeofday()</code> <code>times()</code>

## wait()

プロトタイプ:	<code>int wait(int *status)</code>
一般的な呼び出し元:	newlib C ライブラリ
スレッド・セーフ:	○
ISR からの利用:	○
インクルード:	<code>&lt;sys/wait.h&gt;</code>
説明:	Newlib は <code>wait()</code> 関数を使用して、すべての子プロセスが終了するのを待ちます。HAL は子プロセスの生成をサポートしていないため、この関数はすぐに復帰します。
戻り値:	復帰すると、 <code>status</code> の内容はゼロに設定され、子プロセスが存在しないことを示します。  戻り値は常に <code>-1</code> になり、 <code>errno</code> は <code>ECHILD</code> に設定され、待機する子プロセスが存在しないことを示します。
関連項目:	newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code> 、 <b>Nios II Documentation</b> の順にクリック

## write()

プロトタイプ:	<code>int write(int fd, const void *ptr, size_t len)</code>
一般的な呼び出し元:	C/C++ プログラム newlib C ライブラリ
スレッド・セーフ:	説明を参照。
ISR からの利用:	×
インクルード:	<code>&lt;unistd.h&gt;</code>
説明:	<code>write()</code> 関数は、ファイルまたはデバイスにデータのブロックを書き込みます。この関数は、要求を入力ファイル・ディスクブタ <code>fd</code> に関連付けられたデバイス・ドライバに振り分けます。入力引数 <code>ptr</code> は書き込むデータ、 <code>len</code> はデータのバイト長です。  操作対象のドライバが提供する <code>write()</code> の実装がスレッド・セーフである場合のみ、 <code>write()</code> 呼び出しはスレッド・セーフになります。  <code>fd</code> パラメータに対する有効な値は、 <code>stdout</code> 、 <code>stdin</code> 、および <code>stderr</code> 、または <code>open()</code> 呼び出しから返された値です。
戻り値:	戻り引数は書き込まれたバイト数ですが、これは要求した長さより短い場合もあります。  戻り値の <code>-1</code> はエラーを示します。エラーが発生した場合、 <code>errno</code> はエラーの原因を示す値に設定されます。
関連項目:	<code>close()</code> <code>fcntl()</code> <code>fstat()</code> <code>ioctl()</code> <code>isatty()</code> <code>lseek()</code> <code>open()</code> <code>read()</code> <code>stat()</code>  newlib の資料。Windows の Start メニューから、Programs、Altera、Nios II <code>&lt;version&gt;</code> 、 <b>Nios II Documentation</b> の順にクリック

## 標準型

移植性を高めるために、HAL は ANSI C 組み込み型の代わりに標準型定義のセットを使用しています。表 12-2 にヘッダー `alt_types.h` で定義されるこれらの型を示します。

型	説明
<code>alt_8</code>	符号付き 8 ビット整数
<code>alt_u8</code>	符号なし 8 ビット整数
<code>alt_16</code>	符号付き 16 ビット整数
<code>alt_u16</code>	符号なし 16 ビット整数
<code>alt_32</code>	符号付き 32 ビット整数
<code>alt_u32</code>	符号なし 32 ビット整数
<code>alt_64</code>	符号付き 64 ビット整数
<code>alt_u64</code>	符号なし 64 ビット整数

## 参考資料

この章では以下のドキュメントを参照しています。

- Newlib ANSI C standard library documentation installed with the Nios II EDS

## 改訂履歴

表 12-3 に、本資料の改訂履歴を示します。

表 12-3. 改訂履歴		
日付および ドキュメント・ バージョン	変更内容	概要
2008年5月 v8.0.0	命令関連の例外ハンドラのための <code>alt_instruction_exception_register()</code> および <code>alt_exception_cause_generated_bad_addr()</code> を追加。	<ul style="list-style-type: none"> <li>● Nios II コアに高度な例外を追加。</li> <li>● 命令関連の例外処理を HAL に追加。</li> </ul>
2007年10月 v7.2.0	前バージョンからの内容の変更はありません。	
2007年5月 v7.1.0	<ul style="list-style-type: none"> <li>● 10章から11章に変更。</li> <li>● 「はじめに」の項に目次を追加。</li> <li>● 「参考資料」の項を追加。</li> </ul>	
2007年3月 v7.0.0	前バージョンからの内容の変更はありません。	
2006年11月 v6.1.0	関数 <code>open()</code> には <code>fcntl.h</code> 必要。	
2006年5月 v6.0.0	前バージョンからの内容の変更はありません。	
2005年10月 v5.1.0	以前に誤って削除されていた <code>[alt_irq_disable()]</code> および <code>[alt_irq_enable()]</code> のための API エントリを追加。	
2005年5月 v5.0.0	<ul style="list-style-type: none"> <li>● <code>alt_load_section()</code> 関数を追加。</li> <li>● <code>fcntl()</code> 関数を追加。</li> </ul>	
2004年12月 v1.2	DMA の一般的な要求の名前を更新。	
2004年9月 v1.1	<ul style="list-style-type: none"> <li>● <code>open()</code> を追加。</li> <li>● <code>alt_dma_txchan_open()</code> に <code>ERRNO</code> 情報を追加。</li> <li>● <code>ALT_DMA_TX_STREAM_ON</code> の定義を訂正。</li> <li>● <code>ALT_DMA_RX_STREAM_ON</code> の定義を訂正。</li> <li>● <code>alt_dma_rxchan_ioctl()</code> および <code>alt_dma_txchan_ioctl()</code> に情報を追加。</li> </ul>	
2004年5月 v1.0	初版	

