

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。

NII52008-1.1

はじめに

この章では、Nios® II プロセッサ用の MicroC/OS-II リアルタイム・カーネルについて説明します。

概要

MicroC/OS-II は、Micrium Inc. 社製の定評のあるリアルタイム・カーネルで、解説書としては Jean J. Labrosse 著の「MicroC/OS-II - The Real Time Kernel (CMP Books)」があります。この書籍では MicroC/OS-II は、移植性、ROM 化の可能性、拡張性を備えた、プリエンティブ、リアルタイム、マルチタスク対応のカーネルとして解説されています。MicroC/OS-II は、1992 年の発売以来、数百もの商業アプリケーションで使用され、Nios II プロセッサを含めて 40 種類以上のプロセッサ・アーキテクチャに移植されています。MicroC/OS-II は次のサービスを提供します。

- タスク (スレッド)
- イベント・フラグ
- メッセージ受け渡し
- メモリ管理
- セマフォ
- 時間管理

MicroC/OS-II カーネルは、Nios II プロセッサ用の HAL (Hardware Abstraction Layer) システム・ライブラリ上で動作します。HAL の特性により、MicroC/OS-II をベースにするプログラムは、他の Nios II ハードウェア・システムよりも移植性が高く、また基本ハードウェアが変更されても大きな影響を受けません。また、MicroC/OS-II プログラムはすべての HAL サービスにアクセスし、使い慣れた HAL API (Advanced Programming Interface) 関数を呼び出すことが可能です。

その他の情報

この章では、Nios II プロセッサ用の MicroC/OS-II の使い方の詳細についてのみ説明します。MicroC/OS-II の機能と使用方法に関する詳しい内容については、「MicroC/OS-II - The Real-Time Kernel」を参照してください。また、Micrium 社の Web サイト www.micrium.com でも詳しい情報を入手できます。

ライセンス

アルテラは、Nios II 開発キットの MicroC/OS-II を評価目的として配布しています。市販製品で MicroC/OS-II の使用を計画されている場合は、Licensing@Micrium.com または <http://www.micrium.com> から Micrium 社に問い合わせ、ライセンスを取得する必要があります。

 Micrium社は大学および学生向けに無償ライセンスを提供しています。詳細については、Micrium 社にお問い合わせください。

その他の RTOS プロバイダ

アルテラは、使いやすいリアルタイム・オペレーティング・システム (RTOS) をすぐに体験いただくために、MicroC/OS-II を提供しています。MicroC/OS-II の他にも、多数の RTOS がサードパーティ・ベンダから市販されています。



Nios II プロセッサをサポートする RTOS の全リストは、www.altera.com/nios2にあるNios IIのページにアクセスしてください。

アルテラによる MicroC/OS-II の 移植

アルテラは MicroC/OS-II を Nios II プロセッサに移植しました。アルテラは Nios II 開発キットで MicroC/OS-II を配布し、MicroC/OS-II カーネルの Nios II への移植をサポートします。Nios II 開発キットとともに、MicroC/OS-II プログラムがすぐに使える実用的なサンプルとしてインストールされます。実際、Nios 開発ボードは、MicroC/OS-II をベースにした Web サーバ・リファレンス・デザインおよび Lightweight IP TCP/IP スタックを使用して事前にプログラムされています。

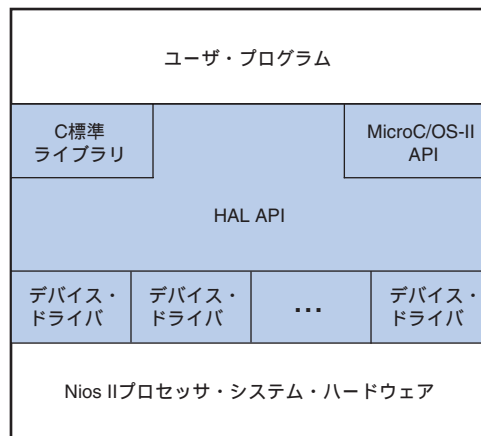
アルテラ版 MicroC/OS-II は、Nios II IDE から使いやすいように設計されています。Nios II IDE を使用すれば、すべての RTOS のモジュールに対するコンフィギュレーションを制御できます。カーネル機能をイネーブルまたはディセーブルするために、ソース・ファイルを直接修正する必要はありません。しかし、ユーザが確認できるように、アルテラは Nios II プロセッサ専用のソース・コードを提供しています。このコードはディレクトリ <Nios II インストール・パス>/components/altera_nios/UCOSII に格納されています。プロセッサに依存しないコードは、<Nios II インストール・パス>/components/micrium_uc_osii にあります。MicroC/OS-II ソフトウェア・コンポーネントは、SOPC Builder ハードウェア・コンポーネントのドライバのように動作します。つまり、MicroC/OS-II が Nios II 統合開発環境 (IDE) プロジェクトに含まれている場合、components/micrium_uc_osii からのヘッダ・ファイルとソース・ファイルがプロジェクト・パスに含まれるため、MicroC/OS-II カーネルはコンパイルされてプロジェクトにリンクされます。

MicroC/OS-II アーキテクチャ

アルテラ版 Nios II プロセッサ用 MicroC/OS-II は、基本的には HAL のスーパーセットです。つまり、MicroC/OS-II スケジューラおよび関連の MicroC/OS-II API を統合して HAL 環境を拡張したものです。すべての HAL API が MicroC/OS-II プロジェクトから利用できます。

図 8-1 に、MicroC/OS-II をベースとしたプログラムのアーキテクチャ、および HAL との関係を示します。

図 8-1. MicroC/OS-II プログラムのアーキテクチャ



一部の HAL 関数はマルチ・スレッド環境の影響を受けます。



マルチ・スレッド環境で特定の HAL 関数を呼び出した結果の詳細については、10-1 ページの「HAL API リファレンス」を参照してください。

MicroC/OS-II のスレッド対応デバッグ

MicroC/OS-II アプリケーションをデバッグするときに、デバッガでは、バックトレースやレジスタ値など、アプリケーション内のすべてのスレッドの現在の状態を表示できます。デバッガを使用して現在のスレッドを変更することはできません。したがって、スレッドの変更や他のスレッドのシングル・ステップ実行を目的として、デバッガを使用することはできません。



スレッド対応デバッグによって、ターゲット・アプリケーションの動作が変化することは一切ありません。

MicroC/OS-II デバイス・ドライバ

各ペリフェラル（つまり、SOPC Builder コンポーネント）は、コンポーネントの HAL ディレクトリの inc サブディレクトリおよび src サブディレクトリにインクルード・ファイルおよびソース・ファイルを提供できます（5-1 ページの「HAL 用デバイス・ドライバの開発」を参照）。HAL ディレクトリの他にも、MicroC/OS-II 環境専用のコードを格納する UCOSII ディレクトリを提供するようにコンポーネントを選択できます。HAL ディレクトリと同様に、UCOSII ディレクトリには inc サブディレクトリと src サブディレクトリが含まれています。これらのディレクトリは、Nios II IDE で MicroC/OS-II プロジェクトをビルドするときに、ソース検索パスとインクルード検索パスに自動的に追加されます。

UCOSII ディレクトリを使用すると、マルチ・スレッド環境でのみ使用されるコードを取得できます。これらの追加検索ディレクトリを除いて、MicroC/OS-II デバイス・ドライバを提供するためのメカニズムは、5-1 ページの「HAL 用デバイス・ドライバの開発」で説明したプロセスと同じです。

HAL システム初期化プロセスでは、`alt_sys_init()` の前に MicroC/OS-II 関数 `OSInit()` を呼び出し、この関数によってシステム内の各デバイスのインスタンス化と初期化を行います。したがって、システムはプログラムが `main()` から `OSStart()` を呼び出すまでシングル・スレッド・モードで動作し続けますが、デバイス・ドライバからはすべての MicroC/OS-II API が利用できます。

スレッド・セーフ HAL ドライバ

同じドライバを HAL 環境と MicroC/OS-II 環境との間で移植できるようにするために、アルテラはオペレーティング・システムの機能へアクセスを提供する OS に依存しないマクロのセットを定義しています。MicroC/OS-II プロジェクト用にコンパイルすると、マクロは MicroC/OS-II API 呼び出しに展開されます。シングル・スレッド HAL プロジェクトに対してコンパイルされると、マクロは悪影響のない空の実装部に展開されます。これらのマクロは、アルテラが提供するデバイス・ドライバ・コードで使用され、同様の移植性を持つデバイス・ドライバを記述する必要がある場合にも使用できます。

表 8-1 に、利用可能なマクロとそれらの機能を示します。



MicroC/OS-II 環境での機能の詳細については、「MicroC/OS-II – The Real-Time Kernel」を参照してください。

ヘッダ・ファイルに対してリストされたパスは、<Nios II インストール・パス >/components/micrium_uc_osii/OCOSII/inc ディレクトリを基準とする相対パスです。

表 8-1. スレッド・セーフ HAL ドライバ用の OS に依存しないマクロ (1 / 2)

マクロ	定義するヘッダ	MicroC/OS-II 実装	シングル・スレッド HAL 実装
ALT_FLAG_GRP(group)	os/alt_flag.h	group という名前のフラグ・グループを指すポインタを作成します。	空のステートメント。
ALT_EXTERN_FLAG_GRP(group)	os/alt_flag.h	group という名前のフラグ・グループを指すポインタへの外部参照を作成します。	空のステートメント。
ALT_STATIC_FLAG_GRP(group)	os/alt_flag.h	group という名前のフラグ・グループを指す静的なポインタを作成します。	空のステートメント。
ALT_FLAG_CREATE(group, flags)	os/alt_flag.h	OSFlagCreate() を呼び出して、flags(フラグ値)でフラグ・グループ・ポインタ group を初期化します。エラー・コードは、マクロの戻り値です。	0 を返します(成功)。
ALT_FLAG_PEND(group, flags, wait_type, timeout)	os/alt_flag.h	最初の4つの入力引数をそれぞれ group、flags、wait_type、および timeout に設定して、OSFlagPend() を呼び出します。エラー・コードは、マクロの戻り値です。	0 を返します(成功)。
ALT_FLAG_POST(group, flags, opt)	os/alt_flag.h	最初の3つの入力引数をそれぞれ group、flags、および opt に設定して、OSFlagPost() を呼び出します。エラー・コードは、マクロの戻り値です。	0 を返します(成功)。
ALT_SEM(sem)	os/alt_sem.h	sem という名前の OS_EVENT ポインタを作成します。	空のステートメント。
ALT_EXTERN_SEM(sem)	os/alt_sem.h	sem という名前の OS_EVENT ポインタへの外部参照を作成します。	空のステートメント。

表 8-1. スレッド・セーフ HAL ドライバ用の OS に依存しないマクロ (2 / 2)

マクロ	定義するヘッダ	MicroC/OS-II 実装	シングル・スレッド HAL 実装
ALT_STATIC_SEM(sem)	os/alt_sem.h	sem という名前の静的な OS_EVENT ポインタを作成します。	空のステートメント。
ALT_SEM_CREATE(sem, value)	os/alt_sem.h	value (引数) で OS_SemCreate() を呼び出して、OS_EVENT ポインタ sem を初期化します。戻り値は、成功時はゼロ、それ以外は負の値です。s w z	0 を返します (成功)。
ALT_SEM_PEND(sem, timeout)	os/alt_sem.h	最初の2つの引数をそれぞれ sem および timeout に設定して、OS_SemPend() を呼び出します。エラー・コードは、マクロの戻り値です。	0 を返します (成功)。
ALT_SEM_POST(sem)	os/alt_sem.h	入力引数 sem を使用して OS_SemPost() を呼び出します。	0 を返します (成功)。

Newlib ANSI C 標準ライブラリ

MicroC/OS-II をベースとしたプログラムでは、ANSI C 標準ライブラリ関数も呼び出せます。マルチ・スレッド環境では、C 標準ライブラリ関数が確実にスレッド・セーフとなるように、何らかの配慮が必要です。Newlib C ライブラリは、すべてのグローバル変数をポインタ `_impure_ptr` を介して参照される 1 つの構造体内に格納します。ただし、アルテラ版 MicroC/OS-II は、タスクごとに構造体の新しいインスタンスを作成します。コンテキストの切り替え時に、`_impure_ptr` の値は、この構造体の現在のタスクのバージョンを指し示すように更新されます。このようにして、`_impure_ptr` が指し示す構造体の内容は、スレッド・ローカルとして扱われます。例えば、このメカニズムを利用すると、各タスクは独自のバージョンの `errno` を持ちます。

このスレッド・ローカルなデータは、タスクのスタックの先頭に割り当てられます。したがって、スタックにメモリを割り当てるときは余分に領域が必要です。一般に、`_reent` 構造体は、標準 C ライブラリでは約 900 バイトのデータを消費し、フットプリントを縮小した C ライブラリでも 90 バイトのデータを消費します。



`_reent` 構造体の内容に関する詳細は、(Windows のスタート・メニューから) **プログラム** > **Altera** > **Nios II Development Kit** > **Nios II Documentation** の順にクリックして、Newlib の資料を参照してください。

さらに、MicroC/OS-II の移植版では適切なタスク・ロックが利用できるため、ヒープ・アクセスつまり、`malloc()` コールと `free()` コールも確実にスレッド・セーフにすることができます。

Nios II IDE での MicroC/OS-II プロジェクトの 実装

MicroC/OS-II をベースにしたプログラムを作成するには、まずシステム・ライブラリのプロパティを MicroC/OS-II プロジェクトに設定する必要があります。ここから、Nios II IDE は RTOS オプションを提供し、ユーザは MicroC/OS-II カーネルのコンフィギュレーションを制御することができます。

従来は、ファイル `OS_CFG.h` 内で `#define` ディレクティブを使用して、MicroC/OS-II をコンフィギュレーションする必要がありました。これに代わって、Nios II IDE では GUI を使用して各オプションをコンフィギュレーションすることが可能です。したがって、MicroC/OS-II の機能をコンフィギュレーションするために、ヘッダ・ファイルやソース・コードを編集する必要はありません。GUI 設定はシステム・ライブラリの `system.h` ファイルに反映され、`OS_CFG.h` は単に `system.h` を指定するだけです。

次のセクションでは、Nios II IDE で利用可能な MicroC/OS-II 設定を定義しています。各設定の意味は、すべて「MicroC/OS-II – The Real-Timer Kernel」の 17 章「MicroC/OS-II Configuration Manual」に記載されています。



Nios II IDE で MicroC/OS-II プロジェクトを作成する方法に関するステップごとの説明は、「Using the MicroC/OS-II RTOS with the Nios II Processor Tutorial」を参照してください。

MicroC/OS-II の一般オプション

表 8-2 に、一般オプションを示します。

オプション	機能説明
タスクの最大数	#define OS_MAX_TASKS にマップします。最低でも 2 つが必要。
割り当て可能な最低優先順位	#define OS_LOWEST_PRIO にマップします。割り当て可能な最大値は 63 です。
イベント・フラグのコード生成イネーブル	#define OS_FLAG_EN にマップします。これをディセーブルすると、イベント・フラグ設定もディセーブルされます (8-8 ページの「イベント・フラグ設定」を参照)。
相互排除セマフォのコード生成イネーブル	#define OS_MUTEX_EN にマップします。これをディセーブルすると、相互排除セマフォ設定もディセーブルされます (8-9 ページの「相互排除の設定」を参照)。
セマフォのコード生成イネーブル	#define OS_SEM_EN にマップします。これをディセーブルすると、セマフォ設定もディセーブルされます (8-9 ページの「セマフォの設定」を参照)。
メールボックスのコード生成イネーブル	#define OS_MBOX_EN にマップします。これをディセーブルすると、メールボックス設定もディセーブルされます (8-10 ページの「メールボックスの設定」を参照)。
キューのコード生成イネーブル	#define OS_Q_EN にマップします。これをディセーブルすると、キュー設定もディセーブルされます (8-10 ページの「キューの設定」を参照)。
メモリ管理のコード生成のイネーブル	#define OS_MEM_EN にマップします。これをディセーブルすると、メモリ管理設定もディセーブルされます (8-11 ページの「メモリ管理の設定」を参照)。

イベント・フラグ設定

表 8-3 に、イベント・フラグ設定を示します。

設定	機能説明
クリア・イベント・フラグを待機するコードのインクルード	#define OS_FLAG_WAIT_CLR_EN にマップします。
OSFlagAccept() のコードのインクルード	#define OS_FLAG_ACCEPT_EN にマップします。

表 8-3. イベント・フラグ設定 (2 / 2)

設定	機能説明
OSFlagDel() のコードのインクルード	#define OS_FLAG_DEL_EN にマップします。
OSFlagQuery() のコードのインクルード	#define OS_FLAG_QUERY_EN にマップします。
イベント・フラグ・グループの最大数	#define OS_MAX_FLAGS にマップします。
イベント・フラグ・グループの名前のサイズ	#define OS_FLAG_NAME_SIZE にマップします。

相互排除の設定

表 8-4 に、相互排除の設定を示します。

表 8-4. 相互排除の設定

設定	機能説明
OSMutexAccept() のコードのインクルード	#define OS_MUTEX_ACCEPT_EN にマップします。
OSMutexDel() のコードのインクルード	#define OS_MUTEX_DEL_EN にマップします。
OSMutexQuery() のコードのインクルード	#define OS_MUTEX_QUERY_EN にマップします。

セマフォの設定

表 8-5 に、セマフォの設定を示します。

表 8-5. セマフォの設定

設定	機能説明
OSSemAccept() のコードのインクルード	#define OS_SEM_ACCEPT_EN にマップします。
OSSemSet() のコードのインクルード	#define OS_SEM_SET_EN にマップします。
OSSemDel() のコードのインクルード	#define OS_SEM_DEL_EN にマップします。
OSSemQuery() のコードのインクルード	#define OS_SEM_QUERY_EN にマップします。

メールボックスの設定

表 8-6 に、メールボックスの設定を示します。

設定	機能説明
OSMboxAccept() のコードのインクルード	#define OS_MBOX_ACCEPT_EN にマップします。
OSMBoxDel() のコードのインクルード	#define OS_MBOX_DEL_EN にマップします。
OSMboxPost() のコードのインクルード	#define OS_MBOX_POST_EN にマップします。
OSMboxPostOpt() のコードのインクルード	#define OS_MBOX_POST_OPT_EN にマップします。
OSMBoxQuery() のコードのインクルード	#define OS_MBOX_QUERY_EN にマップします。

キューの設定

表 8-7 に、キューの設定を示します。

設定	機能説明
OSQAccept() のコードのインクルード	#define OS_Q_ACCEPT_EN にマップします。
OSQDel() のコードのインクルード	#define OS_Q_DEL_EN にマップします。
OSQFlush() のコードのインクルード	#define OS_Q_FLUSH_EN にマップします。
OSQPost() のコードのインクルード	#define OS_Q_POST_EN にマップします。
OSQPostFront() のコードのインクルード	#define OS_Q_POST_FRONT_EN にマップします。
OSQPostOpt() のコードのインクルード	#define OS_Q_POST_OPT_EN にマップします。
OSQQuery() のコードのインクルード	#define OS_Q_QUERY_EN にマップします。
キュー・コントロール・ブロックの最大数	#define OS_MAX_QS にマップします。

メモリ管理の設定

表 8-8 に、メモリ管理の設定を示します。

表 8-8. メモリ管理の設定	
設定	機能説明
OSMemQuery() のコードのインクルード	#define OS_MEM_QUERY_EN にマップします。
メモリ・パーティションの最大数	#define OS_MAX_MEM_PART にマップします。
メモリ・パーティション名のサイズ	#define OS_MEM_NAME_SIZE にマップします。

その他の設定

表 8-9 に、その他の設定を示します。

表 8-9. その他の設定	
設定	機能説明
引数チェックの有効化	#define OS_ARG_CHK_EN にマップします。
uCOS-II フックの有効化	#define OS_CPU_HOOKS_EN にマップします。
デバッグ変数の有効化	#define OS_DEBUG_EN にマップします。
OSSchedLock() および OSSchedUnlock() コードのインクルード	#define OS_SCHED_LOCK_EN にマップします。
uCOS-View に対するチック・ステップ機能の有効化	#define OS_TICK_STEP_EN にマップします。
静的タスクの有効化	#define OS_TASK_STAT_EN にマップします。
静的タスクからのタスク・スタックのチェック	#define OS_TASK_STAT_STK_CHK_EN にマップします。
静的タスク・スタック・サイズ	#define OS_TASK_STAT_STK_SIZE にマップします。
アイドル・タスク・スタック・サイズ	#define OS_TASK_IDLE_STK_SIZE にマップします。
イベント・コントロール・ブロックの最大数	#define OS_MAX_EVENTS 60 にマップします。
セマフォ、相互排除、メールボックス、またはキュー名のサイズ	#define OS_EVENT_NAME_SIZE にマップします。

タスク管理の設定

表 8-10 に、タスク管理の設定を示します。

表 8-10. タスク管理の設定	
設定	機能説明
OSTaskChangePrio() のコードのインクルード	#define OS_TASK_CHANGE_PRIO_EN にマップします。
OSTaskCreate() のコードのインクルード	#define OS_TASK_CREATE_EN にマップします。
OSTaskCreateExt() のコードのインクルード	#define OS_TASK_CREATE_EXT_EN にマップします。
OSTaskDel() のコードのインクルード	#define OS_TASK_DEL_EN にマップします。
プロファイリングを目的とした変数の OS_TCB へのインクルード	#define OS_TASK_PROFILE_EN にマップします。
OSTaskQuery() のコードのインクルード	#define OS_TASK_QUERY_EN にマップします。
OSTaskSuspend() および OSTaskResume() コードのインクルード	#define OS_TASK_SUSPEND_EN にマップします。
OSTaskSwHook() のコードのインクルード	#define OS_TASK_SW_HOOK_EN にマップします。
タスク名のサイズ	#define OS_TASK_NAME_SIZE にマップします。

時間管理の設定

表 8-11 に、時間管理の設定を示します。

表 8-11. 時間管理の設定	
設定	機能説明
OSTimeDlyHMSM() のコードのインクルード	#define OS_TIME_DLY_HMSM_EN にマップします。
OSTimeDlyResume() のコードのインクルード	#define OS_TIME_DLY_RESUME_EN にマップします。
OSTimeGet() および OSTimeSet() コードのインクルード	#define OS_TIME_GET_SET_EN にマップします。
OSTimeTickHook() のコードのインクルード	#define OS_TIME_TICK_HOOK_EN にマップします。