Host Application Implementation Guide

Anybus® CompactCom

Doc.Id. HMSI-27-349
Doc. Rev. 1.10

HMS Industrial Networks
Mailing address: Box 4126, 300 04 Halmstad, Sweden
Visiting address: Stationsgatan 37, Halmstad, Sweden

E-mail: info@hms-networks.com
Web: www.anybus.com

目次

前書き

P. 本ドキュメントについて

より詳しい情報や各種ドキュメントは、HMS の Web サイト ‘www.anybus.jp’ から入手いただけます。

P.1 関連ドキュメント

Anybus CompactCom 40 Software Design Guide	HMS
Anybus CompactCom 40 Hardware Design Guide	HMS
Anybus CompactCom 40 Network Guides	HMS

P.2 ドキュメント更新履歴

最近の変更に関する概要 (1.00 ～ 1.10)

変更内容	ページ
ドキュメント改訂	すべて

改訂一覧

改訂	日付	ドキュメント作成者	章	説明
1.00	2015/11/20	KaD	すべて	新規ドキュメント
1.10	2016/02/05	KaD	すべて	全面改訂版

P.3 表記と用語

本マニュアルでは以下の表記を使用します。

- 番号付きリストは手順を表します。
- 番号なしリストは情報を表します。手順ではありません。
- "Anybus" または "CompactCom" は、Anybus CompactCom 40 機器を表します。
- " ホスト " または " ホストアプリケーション " は、Anybus 機器をホストする機器を表します。
- 16 進数は NNNNh または 0xNNNN の形式で表します。ここで、NNNN は 16 進の値を表します。

P.4 サポート

お問い合わせと技術サポートに関する情報は、www.anybus.jp の各ページをご覧ください。

P.5 用語集

項目	説明
ADI	アプリケーションデータインスタンス 詳細については、『Anybus CompactCom 40 Software Design Guide』を参照してください。
API	アプリケーションプログラミングインターフェース
通信インターフェース	通信インターフェースは、通信方法をコードで定義します。シリアル、パラレル 30、パラレル、SPI のインターフェースを利用できます。
動作モード	動作モードは、ホストアプリケーションインターフェースの OM ピンで設定します。

第 1 章

1. はじめに

Anybus CompactCom 30 または Anybus CompactCom 40 の導入を開始するときは、ホストアプリケーションのサンプルコードを使用することで開発プロセスの迅速化が図れます。ホストアプリケーションのサンプルコードには、Anybus CompactCom モジュールとホストアプリケーション間の接着剤として機能するドライバーが含まれています。このドライバーには、ドライバーへの共通インターフェースを定義する API（アプリケーションプログラミングインターフェース）が用意されています。また、最終製品のベースとして使用できる簡単なアプリケーションを構築するための、API を利用したサンプルアプリケーションのサンプルコードも用意されています。

本ガイドでは、Anybus CompactCom のドライバーとサンプルアプリケーションの導入方法をステップバイステップで説明します。プログラマーは、導入を開始する前に、Anybus CompactCom のオブジェクトモデルと通信プロトコルに関する基本的な知識を習得していることが求められます。推奨するドキュメントについては、4 ページの「関連ドキュメント」を参照してください。

本ガイドは 2 つのステップに分かれています。

ステップ 1：ここでは、ターゲットハードウェアに合わせて必要な適合作業を行い、簡単なアプリケーションを開発します。このステップの目標は、ハードウェア固有のコードを正しく動作させることと、ネットワークに接続して少量のデータをやり取りできるようにすることです。

ステップ 2：ターゲット製品に合わせてコードを修正します。このステップの目標は、コードをカスタマイズして製品に追加できるようにすることと、ネットワーク上で送信されるデータを設定できるようにすることです。このアプリケーションは、必要に応じて機能拡張が行えます。

このドライバーは OS から完全に独立しており、必要な場合には、OS なしでも使用することができます。また、Anybus CompactCom 40 モジュールだけでなく、Anybus CompactCom 30 モジュールでも使用できます。このドライバーは複数の動作モードをサポートしており、実行時に実装されたモードのいずれかを選択することができます。

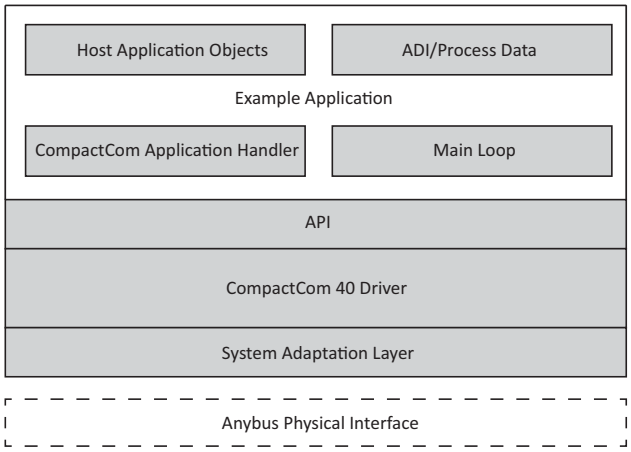
ホストアプリケーションのサンプルコードは、各プラットフォームの各種バージョンに対応しています。本ガイドの作成時点では、以下に示すプラットフォームに対応しています。

各フォルダには、各プラットフォームに必要なすべてのファイルが含まれています。

- Generic - いずれのプラットフォームにも移植可能
- Xilinx, Zynq - MicroZed 評価プラットフォーム用に最適化
- STMicroelectronics, STM32 - STM3240-EVAL 評価プラットフォーム用に最適化
- Freescale - Freescale TWRP1025 評価プラットフォーム用に最適化
- Windows - HMS スターターキットハードウェア（USB board）用に最適化

1.1 概要

ホストアプリケーションのプラットフォームに合わせて、ドライバーコードの一部を修正する必要があります。例えば、Anybus のホストインターフェースにアクセスする機能や、ホストシステムにドライバーを組み込むための修正作業が必要な機能などがこれに該当します。下図は、ホストアプリケーションのサンプルコードの各部分を示しています。



ホストアプリケーションのサンプルコードは、機能の内容とユーザーによるファイル修正の要否に応じて、5 つの異なるフォルダに分けられています。

- **/abcc_abp（ドライバーの一部 - 読み取り専用）**
  - すべての Anybus オブジェクトと通信プロトコルの定義が格納されています。
  - Anybus CompactCom 40 の新しいリリースが利用できるようになると、ファイルが更新される場合があります。
  - **ファイルは読み取り専用です。いかなる方法でも変更してはいけません。**
- **/abcc_drv（ドライバーの一部 - 読み取り専用）**
  - ドライバーのソースファイルとヘッダーファイルが格納されています。
  - Anybus CompactCom 40 の新しいリリースが利用できるようになると、ファイルが更新される場合があります。
  - **ファイルは読み取り専用です。いかなる方法でも変更してはいけません。**
- **/abcc_adapt**
  - 設定ファイルが格納されています。
  - システムの環境に合わせてドライバーやサンプルコードを修正するためには、ユーザーがこれらの設定ファイルを変更しなければなりません。**注**：ある特定のプラットフォーム用に調整済みのサンプルコードを使用する場合は、このフォルダで必要な適合作業のほとんどが既に完了しています。
- **/abcc_obj**
  - Anybus ホストアプリケーションオブジェクトの実装が格納されています。
  - これらのファイルは、最適化や機能拡張のために、必要に応じて変更することができます。
- **/example_app**

  サンプルアプリケーションには以下のものが含まれています。
  - 初期化、再起動、通常状態、エラー状態を処理するメインのステートマシン。
  - Anybus CompactCom のメッセージの送信方法を示すステートマシンのパターン。
  - ドライバーにより要求されるコールバックの実装。
  - ADI（アプリケーションデータインスタンス）の定義、およびデフォルトプロセスデータのマッピングの設定。
  - これらのファイルは、アプリケーションに合わせてプログラマーが修正しなければなりません。また、最適化や機能拡張のための変更を行うことができます。

1.2 準備

作業を続ける前に、以下の質問になるべく多く回答するようにしてください。そうすることで、実装時の決定が簡単に行えるようになります。また、実装時にターゲットハードウェアのハードウェア回路図を入手できるようにしておくとよいでしょう。

ステップ 1

以下の質問項目について検討を行います。

- どのような動作モード（すなわち Anybus CompactCom へのインターフェース）を設計で使用するか。
- どのようなネットワークを設計で使用するか。
- そのネットワークは CompactCom 40 シリーズに対応しているか。または、CompactCom 30 シリーズのモジュールも使用する必要があるか。
- モジュール識別ピンはホストプロセッサーに接続されているか。
- モジュール検出ピンはホストプロセッサーに接続されているか。

ステップ 2

以下の質問項目について検討を行います。

- ハードウェアに割り込み信号が実装されているか。
- 最終製品のネットワークにおいて、どのようなパラメーター / データをやり取りするか。
  - 名前
  - データ型
  - 長さ
  - 読み出し / 書き込みアクセス
  - アサイクリックなアクセス、サイクリックなアクセス
  - 最大 / 最小 / デフォルト
- どのイベント（診断結果）の報告をネットワーク上で行うか。
- どのようなネットワーク識別パラメーターが利用できるか。ベンダー ID、製品コード、ID 番号など。

第 2 章

2. ステップ 1

2.1 システムの適合とアプリケーションの開発

このステップをすべて終えると、以下のことができているはずです。

- Anybus CompactCom との通信に必要なシステム固有の機能の実装。
- デフォルトの設定によるホストアプリケーションのサンプルコードのコンパイル。
- ホストアプリケーションとネットワークマスター / スキャナー間のデータ交換。

2.2 システムの設定

これらの定義は **abcc_adapt/abcc_td.h** に記述されています。

ドライバーで使用する、システム環境の一般設定は、ここで設定されています。

2.2.1 ビッグエンディアンまたはリトルエンディアン

ホストアプリケーションがビッグエンディアンシステムとリトルエンディアンシステムのどちらなのかを設定します。ホストアプリケーションがビッグエンディアンシステムの場合は ABCC_SYS_BIG_ENDIAN を定義します。ホストアプリケーションがリトルエンディアンシステムの場合は定義しないでください（デフォルトのままにしてください）。

```
#define ABCC_SYS_BIG_ENDIAN              /* ビッグエンディアンのホストアプリケーション */
/* #define ABCC_SYS_BIG_ENDIAN */        /* リトルエンディアンのホストアプリケーション */
```

2.2.2 16 ビット Char システム

ホストアプリケーションが 16 ビット char システムと 8 ビット char システムのどちらなのか（すなわち、指定可能な最小の型が 8 ビットと 16 ビットのどちらなのか）を設定します。ホストアプリケーションが 16 ビット char システムの場合は ABCC_SYS_16BIT_CHAR を定義します。ホストアプリケーションが 8 ビット char システムの場合は定義しないでください（デフォルトのままにしてください）。8 ビット char システムに対して 16 ビット char を定義することは推奨されません。

```
#define ABCC_SYS_16_BIT_CHAR             /* 16 ビット char システム */
/* #define ABCC_SYS_16_BIT_CHAR */       /* 8 ビット char システム */
```

2.2.3 データ型

現在のシステムのデータ型を定義します。16 ビット char システムの場合、8 ビット型はすべて 16 ビット型に型変換されます。以下のデータ型を定義しなければなりません。

- BOOL 標準のブールデータ型
- BOOL8 標準のブールデータ型（8 ビット）
- INT8 標準の符号付き 8 ビットデータ型
- INT16 標準の符号付き 16 ビットデータ型
- INT32 標準の符号付き 32 ビットデータ型
- UINT8 標準の符号なし 8 ビットデータ型
- UINT16 標準の符号なし 16 ビットデータ型
- UINT32 標準の符号なし 32 ビットデータ型
- FLOAT32 浮動小数点数（IEC 60559 に基づく）

2.3 Anybus CompactCom の設定

これらの定義と機能は、abcc_adapt/abcc_drv_cfg.h に記述されています。読み取り専用の詳細な記述は、abcc_drv/inc/abcc_cfg.h に記述されています。

Anybus CompactCom の使用方法と、Anybus CompactCom との通信方法に関する設定。動作モード、割り込み処理、メモリ処理などがここで設定されています。

2.3.1 通信インターフェースと動作モード

実装で使用する、アプリケーションと CompactCom 間の通信インターフェースと動作モード（パラレル、SPI、シリアル）を定義します。動作モードの設定については、ホストアプリケーションが Anybus とどのように通信しようとしているか、さらにはユーザーがどのように動作モードを選択するかに応じて、いくつかの選択肢があります。

- 最初に、実装でサポートするすべての通信インターフェースを定義します。使用するすべてのインターフェースをここで定義しなければなりません。そうでないと、後でエラーが報告されます。インターフェースを有効にするたびに、コンパイルされたコードのサイズが大きくなるため、実際に使用するインターフェースだけを定義してください。

40 シリーズのみ。

```
#define ABCC_CFG_DRV_PARALLEL      ( TRUE )  /* パラレル、8/16- ビット、イベントモード */
#define ABCC_CFG_DRV_SPI           ( FALSE ) /* SPI */
```

30 シリーズと 40 シリーズの両方。

```
#define ABCC_CFG_DRV_SERIAL        ( FALSE ) /* シリアル */
#define ABCC_CFG_DRV_PARALLEL_30 ( TRUE )  /* パラレル、8- ビット、半二重 */
```

ABCC_CFG_DRV_SERIAL と ABCC_CFG_DRV_PARALLEL_30 は、Anybus CompactCom 30 シリーズを用いて実装した場合に限り推奨されます。Anybus CompactCom 40 を用いた新しい設計では推奨されません。

- 外部ハードウェアから動作モードを取得します - ホストアプリケーションのプロセッサーに接続されているディップスイッチや、HMI コントローラーなどで動作モードが設定されている場合、ABCC_CFG_OP_MODE_GETTABLE を定義し、abcc_adapt/abcc_sys_adapt.c に関数 ABCC_SYS_GetOpmode() を実装してください。

  #define ABCC_CFG_OP_MODE_GETTABLE (TRUE)

  これらを定義しない場合、特定のモジュールタイプの動作モードを明示的に定義しなければなりません。（11 ページの ABCC_CFG_ABCC_OP_MODE_30 および ABCC_CFG_ABCC_OP_MODE_40 を参照）

- CompactCom のホストコネクタの動作モードピンをホストプロセッサーで制御できる場合、ABCC_CFG_OP_MODE_SETTABLE を定義し、abcc_adapt/abcc_sys_adapt.c に関数 ABCC_SYS_SetOpmode() を実装してください。

  #define ABCC_CFG_OP_MODE_SETTABLE (TRUE)

  これらを定義しない場合、CompactCom のホストコネクタの動作モード信号は、固定または外部ハードウェア（ディップスイッチなど）で制御されるものとみなされます。

- モジュールタイプ（CompactCom 30 および CompactCom 40）ごとに 1 つの動作モードのみ使用する場合、ABCC_CFG_ABCC_OP_MODE_30 および ABCC_CFG_ABCC_OP_MODE_40 を使用して動作モードを定義してください。利用可能な動作モード（ABP_OP_MODE_X）は、abcc_abp/abp.h に記述されています。

  #define ABCC_CFG_ABCC_OP_MODE_30 ABP_OP_MODE_8_BIT_PARALLEL
  #define ABCC_CFG_ABCC_OP_MODE_40 ABP_OP_MODE_16_BIT_PARALLEL

  これらのいずれも定義しない場合、ABCC_SYS_GetOpmode() を実装し、外部ハードウェアから動作モードを取得しなければなりません。11 ページの ABCC_CFG_OP_MODE_GETTABLE を参照してください。

2.3.2 パラレル動作モードの仕様

パラレル動作モード（8 ビットまたは 16 ビット）を使用しない場合、このセクションは無視して構いません。

CompactCom のメモリへのダイレクトアクセスができる場合（外部 SRAM にアクセスするための専用の信号がホストコントローラーに用意されている場合）、ABCC_CFG_MEMORY_MAPPED_ACCESS を TRUE に設定し、ABCC_CFG_PARALLEL_BASE_ADR でベースアドレスを定義してください（このアドレスはホストプラットフォームに合わせて定義しなければなりません）。

#define ABCC_CFG_MEMORY_MAPPED_ACCESS (TRUE)
#define ABCC_CFG_PARALLEL_BASE_ADR (0x00000000)

CompactCom のメモリへのダイレクトアクセスができない場合、データを読み書きするためのいくつかの関数を abcc_adapt/abcc_sys_adapt.c に実装しなければなりません（abcc_drv/inc/abcc_sys_adapt_par.h で記述されています）。17 ページの「パラレル動作モード」を参照してください。

可能であれば、より簡単に実装が行えるように、CompactCom のメモリにダイレクトアクセスできるようにしておくとよいでしょう（こうしておくと、ほとんどの場合、実装の迅速化にもつながります）。

2.3.3 SPI 動作モードの仕様

40 シリーズのみ。SPI 動作モードを使用しない場合、このセクションは無視して構いません。

SPI トランザクションごとの SPI メッセージフラグメントのバイト長は、ABCC_CFG_SPI_MSG_FRAG_LEN で定義されています。

ABCC_CFG_SPI_MSG_FRAG_LEN の値が、送信されるメッセージの最大長より小さい場合、メッセージの送受信は分割され、メッセージがすべて送信されるまで SPI トランザクションが複数実行されることがあります。メッセージが存在するかどうかに関係なく、各 SPI トランザクションはこの長さのメッセージフィールドを持ちます。メッセージが重要な場合には、メッセージが分割されないように、分割長はメッセージの最大長を設定してください。IO データが重要な場合には、SPI トランザクションが高速化できるように、メッセージの分割長をより小さい値に設定してください。

高いパフォーマンスでメッセージを送信できるように、最大 1524 オクテットまでの分割長がサポートされています。メッセージヘッダーは 12 オクテットです。そのため、小さなメッセージを分割せずにサポートするには、16 または 32 オクテットあれば十分です。

```
#define ABCC_CFG_SPI_MSG_FRAG_LEN                ( 16 )
```

2.3.4 モジュール ID とモジュール検出の設定

- CompactCom のホストコネクタのモジュール識別ピン（MI）がホストプロセッサーに接続されていない場合、使用する機器のモジュール ID に対応する正しい CompactCom モジュール ID に合わせて、ABCC_CFG_ABCC_MODULE_ID を定義しなければなりません。これを定義した場合、abcc_abp/abp.h の ABP_MODULE_ID_X を正しく設定する必要があります。

  これを定義しない場合、abcc_adapt/abcc_sys_adapt.c に関数 ABCC_SYS_ReadModuleId() を実装しなければなりません。

アプリケーションコネクタのモジュール ID ピンをホストプロセッサーの GPIO ピンに直接接続し、関数 ABCC_SYS_ReadModuleId() を実装することを推奨します。

```
/* #define ABCC_CFG_ABCC_MODULE_ID       ABP_MODULE_ID_ACTIVE_ABCC40 */
```

- ホストアプリケーションコネクタのモジュール検出ピン（MD）がホストプロセッサーに接続されている場合、ABCC_CFG_MOD_DETECT_PINS_CONN を TRUE に設定し、abcc_adapt/abcc_sys_adapt.c に関数 ABCC_SYS_ModuleDetect() を実装しなければなりません。

  ```
  #define ABCC_CFG_MOD_DETECT_PINS_CONN          ( TRUE )
  ```

2.3.5 メッセージとプロセスデータの設定

ひとまず、以下の定義はデフォルトのままにしておいてください。詳細については、22 ページの「ステップ 2」を参照してください。

```
#define ABCC_CFG_MAX_NUM_APPL_CMDS               ( 2 )
#define ABCC_CFG_MAX_NUM_ABCC_CMDS               ( 2 )
#define ABCC_CFG_MAX_MSG_SIZE                    ( 255 )
#define ABCC_CFG_MAX_PROCESS_DATA_SIZE           ( 512 )
#define ABCC_CFG_REMAP_SUPPORT_ENABLED           ( FALSE )
```

2.3.6 割り込み処理

IRQ ピンが接続されている場合、割り込み無効時でもイベントが発生したかどうかをドライバーでチェックできるように設定することができます。この設定は、CompactCom の電源オンイベントの検出などに使用できます。この機能を有効にするには、ABCC_CFG_POLL_ABCC_IRQ_PIN を定義し、abcc_adapt/abcc_sys_adapt.c. に関数 ABCC_SYS_IsAbccInterruptActive() を実装します。

```
#define ABCC_CFG_POLL_ABCC_IRQ_PIN          ( TRUE )
```

このステップでは割り込み機能を使用しません。すなわち、ABCC_CFG_INT_ENABLED は FALSE に設定します。詳細については、22 ページの「ステップ 2」を参照してください。

IRQ ピンが接続されていない場合、この定義を false に設定しなければなりません。

```
#define ABCC_CFG_INT_ENABLED                ( FALSE )
```

2.3.7 通信ウォッチドッグの設定

CompactCom ウォッチドッグのタイムアウトは、ABCC_CFG_WD_TIMEOUT_MS で定義されています。タイムアウトが発生すると、コールバック関数 ABCC_CbfWdTimeout() が call されます。

注：現在、ウォッチドッグ機能は、SPI、シリアル、パラレル 30（半二重）の動作モードでのみサポートされています。

```
#define ABCC_CFG_WD_TIMEOUT_MS              ( 1000 )
```

2.3.8 ADI の設定

ひとまず、以下の定義はデフォルトのままにしておいてください。詳細については、22 ページの「ステップ 2」を参照してください。

```
#define ABCC_CFG_STRUCT_DATA_TYPE           ( FALSE )
#define ABCC_CFG_ADI_GET_SET_CALLBACK       ( FALSE )
#define ABCC_CFG_64BIT_ADI_SUPPORT          ( FALSE )
```

2.3.9 デバッグイベントの印刷設定

開発用に、さまざまなデバッグ機能が開発者向けに用意されています。以下の定義は、ドライバーからのデバッグ印刷に影響を与えます。アプリケーションコードから追加の印刷を行う必要がある場合は、abcc_adapt/abcc_sw_port.h の移植関数 ABCC_PORT_DebugPrint() を使用します。

- ABCC_CFG_ERR_REPORTING_ENABLED を用いて、エラーを報告するコールバック関数 ABCC_CbfDriverError() を有効 / 無効にします。この関数は abcc_drv/inc/abcc.h に記述されています。

  ```
  #define ABCC_CFG_ERR_REPORTING_ENABLED          ( TRUE )
  ```

- ABCC_CFG_DEBUG_EVENT_ENABLED を用いて、ドライバーによるデバッグイベントの印刷のサポートを有効 / 無効にします。abcc_adapt/abcc_sw_port.h の ABCC_PORT_DebugPrint() は、デバッグ情報の印刷に使用されます。

  ```
  #define ABCC_CFG_DEBUG_EVENT_ENABLED            ( TRUE )
  ```

- ABCC_CbfDriverError() が call されたとき、ABCC_CFG_DEBUG_ERR_ENABLED を用いて、ファイル名や行番号といったデバッグ情報の印刷を有効 / 無効にします。
  #define ABCC_CFG_DEBUG_ERR_ENABLED (TRUE)

- ABCC_CFG_DEBUG_MESSAGING を用いて、送受信されたメッセージの印刷を有効 / 無効にします。バッファの割り当てなどの関連イベントやキューの情報も印刷されます。
  #define ABCC_CFG_DEBUG_MESSAGING (FALSE)

2.3.10 起動時間

CompactCom の IRQ ピンが接続されている場合、ABCC_CFG_STARTUP_TIME_MS は、CompactCom が通信できるようになるまで待機しているときのタイムアウトとして使用されます。このタイムアウト時間内に起動割り込みを受信できなかった場合、エラーが報告されます。割り込みピンを利用できない場合、ABCC_CFG_STARTUP_TIME_MS は、CompactCom との通信を開始するまでの待機時間として機能します。これを定義しない場合、デフォルト値は 1500ms になります。

#define ABCC_CFG_STARTUP_TIME_MS (1500)

可能であれば、起動割り込みを使用することを推奨します。

2.3.11 Sync の設定

40 シリーズのみ。

ひとまず、以下の定義はデフォルトのままにしておいてください。詳細については、22 ページの「ステップ 2」を参照してください。

#define ABCC_CFG_SYNC_ENABLE (FALSE)
#define ABCC_CFG_SYNC_MEASUREMENT_IP (FALSE)
#define ABCC_CFG_SYNC_MEASUREMENT_OP (FALSE)

2.4 システム適合関数

ドライバーが Anybus CompactCom にアクセスできるようにするには、いくつかの関数を実装しなければなりません。この関数は、abcc_adapt/abcc_sys_adapt.c で実装する必要があります。この関数は、以下に示すファイルに動作モードごとに記述されています。

- 汎用関数：abcc_drv/inc/abcc_sys_adapt.h
- SPI 動作モード：abcc_drv/inc/abcc_sys_adapt_spi.h
- パラレル動作モード：abcc_drv/inc/abcc_sys_adapt_par.h
- シリアル動作モード：abcc_drv/inc/abcc_sys_adapt_ser.h

2.4.1 汎用関数

これらの関数は、abcc_drv/inc/abcc_sys_adapt.h に記述されています。

ABCC_SYS_HwInit()

この関数を使用して、CompactCom 機器との通信に必要なハードウェアを起動させることができます（使用するホストプロセッサーのポートピンの入出力の向きや初期値の設定など）。この関数は、起動時の初期化中に、1 度 call する必要があります。

注：この関数から戻ったときに、CompactCom がリセット状態に保たれるようにしてください。

ABCC_SYS_Init()

この関数は、ドライバーの起動 / 再起動時に、ドライバーによって call されます（詳細は、43 ページの「API 関数」の ABCC_StartDriver() を参照してください）。必要に応じて、ハードウェアまたはシステム依存の初期化処理をここで行ってください。この関数を使用しない場合、関数の中身は空のままにしておいてください。

ABCC_SYS_Close()

この関数は、ドライバー終了時にドライバーによって call されます（詳細は、43 ページの「API 関数」の ABCC_ShutDown() を参照してください）。ABCC_SYS_Init() でリソースが割り当てられている場合、この関数でそのリソースを閉じるか解放することを推奨します。この関数を使用しない場合、関数の中身は空のままにしておいてください。

ABCC_SYS_HWReset()

Anybus CompactCom インターフェースのリセットピンをローに落とすには、この関数を実装しなければなりません。

ABCC_SYS_HWReleaseReset()

Anybus CompactCom インターフェースのリセットピンをハイにセットするには、この関数を実装しなければなりません。

ABCC_SYS_AbccInterruptEnable()

現在のところ、割り込みは無効になります。ひとまず、この関数は空のままにしておいてください。

ABCC_SYS_AbccInterruptDisable()

現在のところ、割り込みは無効になります。ひとまず、この関数は空のままにしておいてください。

ABCC_SYS_IsAbccInterruptActive()

割り込みピン（IRQ）がホストプロセッサーに接続されている場合、この関数は CompactCom からの割り込み信号を読み取り、割り込みピンがロー（すなわち割り込みがアクティブ）の場合は TRUE を返します。割り込みが有効でない場合、この関数を使用して、CompactCom インターフェースの割り込みピンのポーリングを有効にすることができます。

2.4.2 SPI 動作モード

40 シリーズのみ。SPI 動作モードを使用しない場合は以下の関数が call されないため、このセクションは無視して構いません。

これらの関数は、abcc_drv/inc/abcc_sys_adapt_spi.h に記述されています。

ABCC_SYS_SpiRegDataReceived()

新たなデータを受信したとき（MISO 受信時）に call するコールバック関数を登録します。

ABCC_SYS_SpiSendReceive()

データの送受信を SPI モードで処理します。

2.4.3 パラレル動作モード

これらの関数は、abcc_drv/inc/abcc_sys_adapt_par.h に記述されています。

パラレル動作モードを使用しない場合は以下の関数が call されないため、このセクションは無視して構いません。

パラレル動作モードを使用する場合でも、ABCC_CFG_MEMORY_MAPPED_ACCESS が定義されている場合は、このセクションは無視して構いません。
ABCC_CFG_MEMORY_MAPPED_ACCESS に関する詳細は、11 ページの「パラレル動作モードの仕様」を参照してください。

ABCC_SYS_ParallelRead()

CompactCom のメモリから一連のオクテットを読み出します。

ABCC_SYS_ParallelRead8()

半二重パラレル動作モードでのみ使用します。

CompactCom のメモリから 1 オクテット読み出します。

ABCC_SYS_ParallelRead16()

CompactCom のメモリから 1 ワード読み出します。

ABCC_SYS_ParallelWrite()

CompactCom のメモリに一連のオクテットを書き込みます。

ABCC_SYS_ParallelWrite8()

半二重パラレル動作モードでのみ使用します。

CompactCom のメモリに 1 オクテット書き込みます。

ABCC_SYS_ParallelWrite16()

CompactCom のメモリに 1 ワード書き込みます。

ABCC_SYS_ParallelGetRdPdBuffer()

受信したリードプロセスデータのアドレスを取得します。

ABCC_SYS_ParallelGetWrPdBuffer()

ライトプロセスデータを保存するアドレスを取得します。

2.4.4 シリアル動作モード

これらの関数は、abcc_drv/inc/abcc_sys_adapt_ser.h に記述されています。

シリアル動作モードを使用しない場合は以下の関数が call されないため、このセクションは無視して構いません。

ABCC_SYS_SerRegDataReceived()

シリアルチャネルで新たな受信テレグラムを受信したことを示すコールバック関数を登録します。

ABCC_SYS_SerSendReceive()

送信テレグラムの送信と、受信テレグラムの受信準備を行います。

ABCC_SYS_SerRestart()

シリアルドライバーを再起動します。通常、テレグラムがタイムアウトしたときに使用します。

2.5 オブジェクトの設定

このステップでは、CompactCom のデフォルト設定を使用します。abcc_adapt/abcc_obj_cfg.h. ではホストアプリケーションオブジェクトは無効になっています。

22 ページの「ステップ 2」で、ターゲット製品に合わせてネットワーク識別属性のカスタマイズを行います。

2.6 サンプルアプリケーション

各ネットワークアプリケーションから Anybus CompactCom ドライバーへの共通インターフェースを定義する API レイヤーが用意されています。この API は、abcc_drv/inc/abcc.h に記述されています。詳細は 43 ページの「API」を参照してください。このサンプルアプリケーションは、標準のアプリケーションが API を使って CompactCom ドライバーを実装する方法の一例を示すためのものです。このサンプルアプリケーションは、そのままで CompactCom のコンセプトの試験に使用できるほか、最終製品にドライバーを組み込む際のベースとしても使用することができます。

ステップ 1 では、サンプルアプリケーションを変更する必要はありません。

2.6.1 ADI とプロセスデータのマッピング

プロセスデータはアプリケーションには不可欠な部分です。ADI（アプリケーションデータインスタンス）を作成してプロセスデータをアプリケーションに追加し、目的のプロセスデータエリアにそれらをマッピングします（読み取りまたは書き込み）。

ひとまず、appl_adimap_simple16.c に記述されているマッピングを使用します。これは、/example_app/appl_adi_config.h の APPL_ACTIVE_ADI_SETUP が APPL_ADI_SETUP_SIMPLE_16 として定義されていることを意味します。

- example_app/appl_adimap_simple16.c - このマッピングは 32 個の 16 ビットワードをループします。
  - ADI 1：32 個の UINT16 の要素からなる配列
  - ADI 2：32 個の UINT16 の要素からなる配列
  - これらの ADI はそれぞれの方向にマッピングされます。
  - この 2 つの ADI は同じデータプレースホルダーを参照するため、データはループされます。
  - 構造体またはコールバックは使用しません。

2.6.2 メインループ

メインループとは、アプリケーションの実行が開始される場所です。一般的なプロジェクトでは、メインループは main.c という名前のファイルにあります。以下に、メインループを実装するためのいくつかのガイドラインを示します。

- ABCC_HwInit() - この関数は、CompactCom との通信に必要なハードウェアを起動します。この関数は、起動時の初期化中に、1 度 call する必要があります。また、この関数から戻ったときに、CompactCom がリセット状態に保たれるようにしてください。ドライバーの再起動時は、この関数を再度 call する必要はありません。ABCC_HwInit() は、abcc_adapt/abcc_sys_adapt.c の関数 ABCC_SYS_HwInit() を起動します。ABCC_SYS_HwInit() は、現在のシステムに合わせてをカスタマイズする必要があります。この関数は、メイン関数で最初に call する関数に含めてください。

- APPL_HandleAbcc() - この関数は、CompactCom のステートマシンを実行し、ドライバーのリセット、実行、シャットダウンを処理します。この関数は、メインループから定期的に call しなければなりません。この関数を call するたびに、CompactCom ドライバーから状態が返されます。

APPL_MODULE_NO_ERROR	CompactCom は正常です。これは、すべて正常に実行されているときの正常な応答です。
APPL_MODULE_NOT_DETECTED	CompactCom を検出できません。ユーザーに通知してください。
APPL_MODULE_NOT_SUPPORTED	サポートされていないモジュールが検出されました。ユーザーに通知してください。
APPL_MODULE_NOT_ANSWERING	以下の理由が考えられます：間違った API が選択されている、モジュールに問題がある
APPL_MODULE_RESET	CompactCom からリセットが要求されました。ネットワークからリセットを受信しました。アプリケーションでシステムを再起動する必要があります。
APPL_MODULE_SHUTDOWN	シャットダウンが要求されました。
APPL_MODULE_UNEXPECTED_ERROR	予期せぬエラーが発生しました。ユーザーに通知してください。必要に応じて、出力をフェールセーフ状態にしてください。

詳細は 46 ページの「ホストアプリケーションのステートマシン」を参照してください。

- ABCC_RunTimerSystem() - この関数は、既知の周期（前回の call からのミリ秒単位の時間）で定期的に call する必要があります。それを実現するには、メインループで既知の遅延を持たせてループの繰り返しごとに関数を call するか、タイマー割り込みを設定します。
CompactCom ドライバーのタイマーは、すべてこの関数で処理する必要があります。タイマー割り込みから定期的にこの関数を call することを推奨します。この関数を実装しない場合、タイムアウトやウォッチドッグの機能は動作しません。

```
int main()
{
        APPL_AbccHandlerStatusType eAbccHandlerStatus = APPL_MODULE_NO_ERROR;

        if( ABCC_HwInit() != ABCC_EC_NO_ERROR )
        {
                return( 0 );
        }
        while( eAbccHandlerStatus == APPL_MODULE_NO_ERROR )
        {
                eAbccHandlerStatus = APPL_HandleAbcc();
#if( !USE_TIMER_INTERRUPT )
                ABCC_RunTimerSystem( APPL_TIMER_MS );
                DelayMs( APPL_TIMER_MS );
#endif
                switch( eAbccHandlerStatus )
                {
                    case APPL_MODULE_RESET:
                        Reset();
                        break;
                    default:
                        break;
                }
        }
return( 0 );
}
```

ヒント：この関数を使用する場合、タイマー割り込みを使用することを推奨します。ただし、実装時のデバッグを容易にするため、はじめのうちはタイマー割り込みをスキップしてください。

2.6.3 コンパイルと実行

プロジェクトをコンパイルするには、Anybus CompactCom 40 のすべてのサンプルコード（ここで説明した 5 つのすべてのフォルダ）が含まれるように、make ファイルを更新してからコンパイルしてください。

- /abcc_abp
- /abcc_drv
- /abcc_adapt
- /abcc_obj
- /example_app

ステップ 2 を開始する前に、以下のことを確認してください。

- プロジェクトのコンパイラーにエラーが発生していない
- ホストアプリケーションが Anybus CompactCom と通信できる
- ネットワークを使用してデータをやり取りできる

第 3 章

3. ステップ 2

3.1 適合とカスタマイズ

このステップをすべて終えると、以下のことができているはずです。

- ネットワーク識別子（ベンダー ID、製品コード、製品名など）のカスタマイズ。
- ターゲット製品向け ADI の作成。
- 周期的に交換される ADI とプロセスデータとのマッピング。

3.1.1 Anybus CompactCom の設定

ステップ 1 では、Anybus CompactCom の一部の設定はデフォルト値のままになっています。ここでは、それらの設定値の見直しを行います。

メッセージとプロセスデータの設定

- ABCC_CFG_MAX_NUM_APPL_CMDS では、応答を受信せずに送信できるメッセージコマンド数を設定することができます。少なくとも 2 つのバッファがドライバーで必要になります。当然ながら、この値を大きくすると使用できるメッセージコマンド数が増えますが、より多くの RAM メモリを消費します。メッセージの送信に関する詳細については、37 ページの「メッセージ処理」を参照してください。
  #define ABCC_CFG_MAX_NUM_APPL_CMDS (2)

- ABCC_CFG_MAX_NUM_ABCC_CMDS では、応答を送信せずに受信できるメッセージコマンド数を設定することがきます。少なくとも 2 つのバッファがドライバーで必要になります。当然ながら、この値を大きくすると使用できるメッセージコマンド数が増えますが、より多くの RAM メモリを消費します。
  #define ABCC_CFG_MAX_NUM_ABCC_CMDS (2)

- 使用するメッセージの最大サイズ（バイト単位）は、ABCC_CFG_MAX_MSG_SIZE で設定します。

Anybus CompactCom 30 は 255 バイトのメッセージをサポートし、Anybus CompactCom 40 は 1524 バイトのメッセージをサポートします。ABCC_CFG_MAX_MSG_SIZE には、送受信されるメッセージの最大サイズを設定してください。不明な場合は、サポートされている最大サイズを設定することを推奨します。

#define ABCC_CFG_MAX_MSG_SIZE (1524)

- プロセスデータの最大サイズ（バイト単位）は、送受信の方向に関係なく、ABCC_CFG_MAX_PROCESS_DATA_SIZE で設定します。最大サイズは、使用するネットワークの種類によって異なります。使用するネットワークのネットワークガイドを参照してください。
  #define ABCC_CFG_MAX_PROCESS_DATA_SIZE (512)

- ABCC_CFG_REMAP_SUPPORT_ENABLED で、ドライバーとアプリケーションデータオブジェクトによる remap コマンドのサポートを有効 / 無効にします。TRUE に設定した場合、アプリケーションで ABCC_CbfRemapDone() を実装する必要があります。この関数は abcc_drv/inc/abcc.h に記述されています。
  #define ABCC_CFG_REMAP_SUPPORT_ENABLED (FALSE)

割り込み処理

Anybus CompactCom ドライバーは、割り込み機能が有効 / 無効のいずれの場合でも使用できます。

- ABCC_CFG_INT_ENABLED を定義し、割り込みルーチンとともに CompactCom の IRQ ピンを使用するかどうかを定義します。IRQ ピンは、パラレルモードと SPI モードのいずれでも使用できます。関数 ABCC_ISR() は、CompactCom の割り込みルーチンの内部から call してください。割り込みがエッジでトリガーされる場合、ABCC_ISR() が call される前に割り込みをアクノリッジする必要があります。
  #define ABCC_CFG_INT_ENABLED (FALSE)

- **パラレルモードを使用しない場合、この定義は無視して構いません。**
  ABCC_CFG_INT_ENABLE_MASK_PAR を定義して、パラレルモード使用時にどの割り込みを有効にするかを設定します。利用可能なオプションは、abcc_abp/abp.h に定義されています（INT MASK レジスタ）イベントを CompactCom の割り込みで通知しない場合、ドライバー関数 ABCC_RunDriver()（この関数は example_app/APPL_HandleAbcc() で call されます）でイベントをポーリングしなければなりません。これを定義しない場合、デフォルトのマスクは 0 になります。
  #define ABCC_CFG_INT_ENABLE_MASK_PAR (ABP_INTMASK_RDPDIEN | ABP_INTMASK_STATUSIEN | ABP_INTMASK_RDMSGIEN | ABP_INTMASK_WRMSGIEN | ABP_INTMASK_ANBRIEN)

- ABCC_CFG_HANDLE_INT_IN_ISR_MASK は、Anybus CompactCom のどの割り込みイベントを割り込みコンテキストで処理するかを定義します。割り込みイネーブルマスク（ABCC_CFG_INT_ENABLE_MASK_X）で許可されているが、ISR で処理するよう設定されていないイベントは、ABCC_ISR_EVENT_X（abcc_drv/inc/abcc.h で定義されている）のビットフィールドに変換され、ABCC_CbfEvent() コールバックによりユーザーに転送されます。8 / 16 ビットパラレル動作モードにのみ適用されます。

  これを定義しない場合、値は 0 になります（すなわち、どのイベントも ISR で処理されません）。
  #define ABCC_CFG_HANDLE_INT_IN_ISR_MASK (ABP_INTMASK_RDPDIEN)

  詳細は 35 ページの「イベント処理」を参照してください。

ADI の設定

- ABCC_CFG_STRUCT_DATA_TYPE で、ADI の構造データ型のサポートを有効にします。この定義は、abcc_drv/inc/abcc_ad_if.h の AD_AdiEntryType に影響を与えます（AD_AdiEntryType は、ユーザー ADI の定義に使用されます）。これを定義すると、必要なメモリ使用量が増加します。そのため、構造データ型が必要な場合のみ定義してください。
  #define ABCC_CFG_STRUCT_DATA_TYPE (FALSE)

- ABCC_CFG_ADI_GET_SET_CALLBACK で、ADI が読み書きされるたびにドライバーがコールバックによる通知を行うかどうかを設定します。この定義は、abcc_drv/inc/abcc_ad_if.h の AD_AdiEntryType に影響を与えます（AD_AdiEntryType は、ユーザー ADI の定義に使用されます）。ネットワークで ADI が読み込まれると、アクションの前にコールバックが call されます。ネットワークで ADI が書き込まれると、アクションの後にコールバックが call されます。34 ページの「プロセスデータのコールバック」を参照してください。
  #define ABCC_CFG_ADI_GET_SET_CALLBACK (FALSE)

- ABCC_CFG_64BIT_ADI_SUPPORT で、アプリケーションデータオブジェクトにおける 64 ビットデータ型のサポートを有効 / 無効にします。
  #define ABCC_CFG_64BIT_ADI_SUPPORT (FALSE)

シンクの設定

40 シリーズのみ。

- ドライバーによるシンクのサポートを有効 / 無効にします。TRUE に設定した場合、アプリケーションで abcc_CbfSyncIsr() を実装しなければなりません。
  #define ABCC_CFG_SYNC_ENABLE (FALSE)

シンクを使用しない場合、または、コードをリリース版としてコンパイルする場合、以下の定義を無効にしてください。

シンク測定関数は、シンクアプリケーションが使用する入力処理時間と出力処理時間の測定に使用します。

- ABCC_CFG_SYNC_MEASUREMENT_IP で、（シンクで使用される）入力処理時間測定のドライバーサポートを有効 / 無効にします。この定義は、開発時に入力処理時間の測定を有効にして製品の特別な試験版をコンパイルするのに使用されます。ABCC_CFG_SYNC_MEASUREMENT_IP を TRUE に設定すると、WRPD が送信されたときに ABCC_SYS_GpioReset() が call されます。SPI 動作モードで実行している場合、WRPD の送信時ではなく、ABCC_SpiRunDriver() が Anybus へのデータ送信の完了時に call されます。ABCC_CFG_SYNC_MEASUREMENT_IP を TRUE に設定した場合、Input Capture Point で ABCC_GpioSet() を call する必要があります。
  #define ABCC_CFG_SYNC_MEASUREMENT_IP (FALSE)

- ABCC_CFG_SYNC_MEASUREMENT_OP で、（シンクで使用される）出力処理時間測定のドライバーサポートを有効 / 無効にします。この定義は、開発時に入力処理時間の測定を有効にして製品の特別な試験版をコンパイルするのに使用されます。ABCC_CFG_SYNC_MEASUREMENT_OP を TRUE に設定すると、RDPDI 割り込みから ABCC_SYS_GpioSet() が call されます。ABCC_CFG_SYNC_MEASUREMENT_OP を TRUE に設定した場合、Output Valid Point で ABCC_GpioReset() を call する必要があります。
  #define ABCC_CFG_SYNC_MEASUREMENT_OP (FALSE)

3.1.2 システム適合関数

これらの関数は abcc_adapt/abcc_sys_adapt.c に記述されています。

ステップ 2 で割り込みを使用する場合、以下の関数を実装してください。

- **ABCC_SYS_AbccInterruptEnable()**

  CompactCom のハードウェア割り込み（アプリケーションインターフェースの IRQ_N ピン）を有効にします。この関数は、CompactCom の割り込みを有効にしたときにドライバーによって call されます。

  ABCC_CFG_INT_ENABLED を定義しない場合、この関数を実装する必要はありません。

- **ABCC_SYS_AbccInterruptDisable()**

  CompactCom のハードウェア割り込み（アプリケーションインターフェースの IRQ_N ピン）を無効にします。

  ABCC_CFG_INT_ENABLED を定義しない場合、この関数を実装する必要はありません。

3.1.3 ネットワークの識別

これまでに、ネットワークの設定はすべて無効のままになっています。また、この製品が HMS 製品であることが認識されています。次はネットワークの識別設定をカスタマイズする番です。

ホストアプリケーションオブジェクト - ネットワーク

abcc_adapt/abcc_obj_cfg.h で各ホストアプリケーションオブジェクトを定義し、実装でサポートすべきネットワークを定義します。ホストアプリケーションオブジェクトの追加の実装は、各々のオブジェクト独自の c- 及び h- ファイルを持たせて、abcc_obj フォルダで行われます。

例：

```
#define PRT_OBJ_ENABLE                  ( TRUE )
#define EIP_OBJ_ENABLE                  ( FALSE )
#define EPL_OBJ_ENABLE                  ( TRUE )
```

ENABLE に設定された各ネットワークオブジェクトの識別に関するアトリビュートは、アプリケーションで設定しなければならないパラメーターです。このアトリビュートはすべて、ネットワーク上での機器の識別のされ方に関係しています。このアトリビュートを有効（TRUE）にすると、その値が使用されます。このアトリビュートを無効（FALSE）にすると、アトリビュートのデフォルト値が使用されます。これらの設定は abcc_adapt/abcc_identification.h に記述されています。

例：

```
/*-----------------------------------------------------------------------------------------------------------------------------
** Ethernet Powerlink (0xE9)
**-----------------------------------------------------------------------------------------------------------------------------
*/
#if EPL_OBJ_ENABLE
/*
** Attribute 1:Vendor ID (UINT32 - 0x00000000-0xFFFFFFFF)
*/
#define EPL_IA_VENDOR_ID_ENABLE         TRUE
#define EPL_IA_VENDOR_ID_VALUE          0xFFFFFFFF

/*
** Attribute 2:Product Code type (UINT32 - 0x00000000-0xFFFFFFFF)
*/
#define EPL_IA_PRODUCT_CODE_ENABLE      TRUE
#define EPL_IA_PRODUCT_CODE_VALUE       0xFFFFFFFF
```

定数の代わりに関数を定義して値を生成することもできます。関数が適しているケースとしては、例えば、シリアル番号が挙げられます。以下の例では、製造時に特定のメモリエリアにシリアル番号が設定されており、ここで同じ番号を取得しています。

```
extern char* GetSerialNumberFromProductionArea(void);

#define PRT_IA_IM_SERIAL_NBR_ENABLE TRUE
#define PRT_IA_IM_SERIAL_NBR_VALUE GetSerialNumberFromProductionArea()
```

ホストアプリケーションオブジェクト - その他

abcc_adapt/abcc_obj_cfg.h において、実装でサポートすべきその他のホストアプリケーションオブジェクトをすべて定義します。

例：

```
#define ETN_OBJ_ENABLE                  TRUE
#define SYNC_OBJ_ENABLE                 FALSE
```

ホストアプリケーションオブジェクト - 高度な設定

abcc_adapt/abcc_obj_cfg.h には、abcc_adapt/abcc_identification.h で既に定義されているアトリビュートを除き、サポートされている各ホストオブジェクトのアトリビュートがすべて含まれています。このファイルで記述されているアトリビュートは、デフォルトではすべて無効になっています。ネットワーク固有のサービスは、デフォルトでは "not supported" と記されており、必要に応じてアプリケーションで実装する必要があります。

abcc_adapt/abcc_platform_cfg.h を使用して、abcc_adapt/abcc_obj_cfg.h と abcc_adapt/abcc_identification.h に記述されているオブジェクトとアトリビュートを上書きすることができます。

任意の定義を abcc_adapt/abcc_platform_cfg.h に追加するだけで、定義が上書きされます。

使用しない場合は、ファイルの中身を空のままにしておいてください。

3.1.4 ソフトウェアプラットフォームの移植

これらの関数は abcc_adapt/abcc_sw_port.h に記述されています。

このドライバーは、メモリコピー関数、印刷関数、クリティカルセクション関数など、さまざまな関数を使用しており、現在のソフトウェアプラットフォームに合わせてそれらの関数を最適化できます。これらの関数は、abcc_adapt/abcc_sw_port.h に記述されています（abcc_drv/inc/abcc_port.h に記述されています）。デフォルトのサンプルコードはそのまま使用できますが、実装プロジェクトでは、目的のプラットフォームに合わせて後で最適化（推奨）するようにしてください。

ABCC_PORT_DebugPrint()

この関数は、ドライバーがイベントやエラーのデバッグ情報などをデバッグプリントする場合に使用します。これを定義しない場合、ドライバーは何もしません。デバッグプリントの結果は、シリアル端末に送信したり、ログファイルへの保存が行えます。

クリティカルセクション関数

クリティカルセクションは、CompactCom の割り込みハンドラーのコンテキストとアプリケーションのスレッドとの間でリソースの衝突や競合が発生するおそれのある場合に使用します。

クリティカルセクションを実装するには 3 つのマクロを使用します。

- ABCC_PORT_UseCritical()
- ABCC_PORT_EnterCritical()
- ABCC_PORT_ExitCritical()

クリティカルセクションの実装では、ドライバーの構成に応じてさまざまな要件があります。以下の番号付きリストから最適な実装を選択してください。最初にどのステートメントが当てはまるかによって、要件を選択します。

1. 以下のいずれかのステートメントが当てはまる場合、3 つのマクロをすべて実装する必要があります。
   - どのメッセージ処理も割り込みコンテキスト内で行われる。

   要件：
   - この実装は、割り込みコンテキストから入るクリティカルセクションをサポートしなければなりません。ABCC_PORT_EnterCritical() で前もって必要とされる宣言に対し、ABCC_PORT_UseCritical() を使用する必要があります。
   - クリティカルセクションに入る際、必要な割り込み（すなわち、ドライバーによるアクセスが発生する割り込み）をすべて無効にしておかなければなりません。クリティカルセクションから離れる際、割り込みの設定を元の状態に戻さなければなりません。

2. 以下のいずれかのステートメントが当てはまる場合、ABCC_PORT_EnterCritical() と ABCC_PORT_ExitCritical() を実装する必要があります。
   - ABCC_RunTimerSystem() がタイマー割り込みから call される。
   - アプリケーションにおいて、上位レベル（セマフォなど）でメッセージインターフェースが保護されていない状態で、複数のプロセスまたはスレッドにより CompactCom のメッセージインターフェースがアクセスされている。

   要件：
   - クリティカルセクションに入る際、必要な割り込み（すなわち、ドライバーによるアクセスが発生する割り込み）をすべて無効にしておかなければなりません。クリティカルセクションから離れる際、割り込みを再度有効にしなければなりません。

3. 上記のいずれも当てはまらない場合、いずれも実装する必要はありません。

ABCC_PORT_UseCritical()

ABCC_PORT_EnterCritical() または ABCC_PORT_ExitCritical() を call する前に何らかの準備が必要な場合、このマクロを使用してプラットフォーム固有の準備を行います。

ABCC_PORT_EnterCritical()

この関数は、CompactCom の割り込みハンドラーとアプリケーションのスレッド（またはメインループ）との間で内部リソースの衝突が発生する可能性がある場合、ドライバーによって call されます。この関数は、割り込みを一時的に無効にして衝突を防ぎます。なお、ドライバーによるアクセスが発生する可能性のある割り込みは、すべて無効にする必要があります。

ABCC_PORT_ExitCritical()

割り込みの状態を ABCC_PORT_EnterCritical() が call される前の状態に戻します。

ABCC_PORT_MemCopy()

一連のオクテットを、ソースポインターからデスティネーションポインターにコピーします。

ABCC_PORT_StrCpyToNative()

パッケージ化された文字列をネイティブ形式の文字列にコピーします。

ABCC_PORT_StrCpyToPacked()

ネイティブ形式の文字列をパッケージ化された文字列にコピーします。

ABCC_PORT_CopyOctets()

オクテット境界のバッファをコピーします。

ABCC_PORT_Copy8()

ソースからデスティネーションに 8 ビットをコピーします。16 ビット char プラットフォームでは、このマクロを移植する際に、オクテット境界のサポート（オクテットのオフセットが奇数）を考慮する必要があります。

ABCC_PORT_Copy16()

ソースからデスティネーションに 16 ビットをコピーします。このマクロを移植する際に、オクテット境界のサポート（オクテットのオフセットが奇数）を考慮する必要があります。

ABCC_PORT_Copy32()

ソースからデスティネーションに 32 ビットをコピーします。このマクロを移植する際に、オクテット境界のサポート（オクテットのオフセットが奇数）を考慮する必要があります。

ABCC_PORT_Copy64()

ソースからデスティネーションに 64 ビットをコピーします。このマクロを移植する際に、オクテット境界のサポート（オクテットのオフセットが奇数）を考慮する必要があります。

3.1.5 サンプルアプリケーション

ADI とプロセスデータのマッピング

ステップ 1 では、サンプルの ADI マッピング appl_adimap_simple16.c を使用しました。このサンプルアプリケーションには、ADI マッピングのサンプルが含まれています。このサンプルでは、さまざまな種類の ADI の例が示されています。

マッピングは 1 度に 1 つだけ使用できます。アプリケーションで現在使用されているマッピングは、example_app/appl_adi_config.h において、使用する ADI マッピングに対して APPL_ACTIVE_ADI_SETUP を定義することで設定します。

- example_app/appl_adimap_simple16.c - このマッピングは 32 個の 16 ビットワードをループします。

ADI	説明
ADI 1	32 個の UINT16 の要素からなる配列（入力データとしてマッピングされる）
ADI 2	32 個の UINT16 の要素からなる配列（出力データとしてマッピングされる）

  - これらの ADI は、各方向のプロセスデータにマッピングされます。
  - この 2 つの ADI は同じデータプレースホルダーを参照するため、データはループされます。
  - 構造体またはコールバックは使用しません。

- example_app/appl_adimap_separate16.c - get/set コールバックの使用例：

ADI	説明
ADI 10	32 個の UINT16 の要素からなる配列（出力データとしてマッピングされる）
ADI 11	32 個の UINT16 の要素からなる配列（入力データとしてマッピングされる）
ADI 12	UINT16（プロセスデータにはマッピングされない）

  - ADI 10 および 11 は、それぞれの方向のプロセスデータにマッピングされます。
  - ネットワークが ADI 11 を読み込んだとき、コールバックが使用されます。このコールバックにより、ADI 12 の値が 1 つ加算されます。
  - ネットワークが ADI 10 に書き込んだとき、コールバックが使用されます。このコールバックにより、ADI 10 の値が ADI 11 にコピーされます。

コールバックを使用するため、abcc_adapt/abcc_drv_cfg.h で ABCC_CFG_ADI_GET_SET_CALLBACK を有効にする必要があります。13 ページの「ADI の設定」を参照してください。

- example_app/appl_adimap_alltypes.c - 構造体データ型およびビットデータ型の使用例：

ADI	説明
ADI 20	UINT32（出力データとしてマッピングされる）
ADI 21	UINT32（入力データとしてマッピングされる）
ADI 22	SINT32（出力データとしてマッピングされる）
ADI 23	SINT32（入力データとしてマッピングされる）
ADI 24	UINT16（出力データとしてマッピングされる）
ADI 25	UINT16（入力データとしてマッピングされる）
ADI 26	SINT16（出力データとしてマッピングされる）
ADI 27	SINT16（入力データとしてマッピングされる）
ADI 28	BITS16（出力データとしてマッピングされる）
ADI 29	BITS16（入力データとしてマッピングされる）
ADI 30	UINT8（出力データとしてマッピングされる）
ADI 31	UINT8（入力データとしてマッピングされる）
ADI 32	SINT8（出力データとしてマッピングされる）
ADI 33	SINT8（入力データとしてマッピングされる）
ADI 34	PAD8（出力データとしてマッピングされる。予約空間。データなし）
ADI 35	PAD8（入力データとしてマッピングされる。予約空間。データなし）
ADI 36	BIT7（出力データとしてマッピングされる）
ADI 37	BIT7（入力データとしてマッピングされる）
ADI 38	Struct（出力データとしてマッピングされる）
ADI 39	Struct（入力データとしてマッピングされる）

構造体を使用するため、abcc_adapt/abcc_drv_cfg.h で ABCC_CFG_STRUCT_DATA_TYPE を有効にする必要があります。13 ページの「ADI の設定」を参照してください。

この例では以下のステップを実行します。このステップは、実際の実装に合わせてカスタマイズする必要があります。

- ADI のエントリリスト - ADI（すなわち、実装で使用するデータインスタンス）のエントリリストにおいて、ADI を AD_AdiEntryType として定義しなければなりません。ADI 関連のパラメーターは、すべてここで指定します。

ADI エントリ項目	説明
インスタンス	ADI インスタンス番号（1 ～ 65535）0 はクラス用に予約されています。
pabName	ADI 名（char 文字列、インスタンスアトリビュート #1）。NULL の場合、長さ 0 の名前が返されます。
bDataType	ABP_BOOL: Boolean ABP_SINT8: 符号つき 8 ビット整数 ABP_SINT16: 符号つき 16 ビット整数 ABP_SINT32: 符号つき 32 ビット整数 ABP_UINT8: 符号なし 8 ビット整数 ABP_UNIT16: 符号なし 16 ビット整数 ABP_UINT32: 符号なし 32 ビット整数 ABP_CHAR: 文字型 ABP_ENUM: 列挙型 ABP_SINT64: 符号つき 64 ビット整数 ABP_UINT64: 符号なし 64 ビット整数 ABP_FLOAT: 浮動小数点数（32 ビット） ABP_OCTET 未定義の 8 ビットデータ（40 シリーズのみ） ABP_BITS8 8 ビットのビットフィールド（40 シリーズのみ） ABP_BITS16 16 ビットのビットフィールド（40 シリーズのみ） ABP_BITS32 32 ビットのビットフィールド（40 シリーズのみ） ABP_BIT1 1 ビットのビットフィールド（40 シリーズのみ） ABP_BIT2 2 ビットのビットフィールド（40 シリーズのみ） : : ABP_BIT7 7 ビットのビットフィールド（40 シリーズのみ） ABP_PAD0 0 パッドのビットフィールド（40 シリーズのみ） ABP_PAD1 1 パッドのビットフィールド（40 シリーズのみ） : : ABP_PAD16 16 パッドのビットフィールド（40 シリーズのみ）
bNumOfElements	配列の場合：データ型の要素数を bDataType で指定します。 構造体データ型の場合：構造体に含まれる要素の数。
bDesc	エントリ記述子。以下の設定に基づくビット値。 ABP_APPD_DESCR_GET_ACCESS：value アトリビュートに対して get サービスが使用可能。 ABP_APPD_DESCR_SET_ACCESS：value アトリビュートに対して set サービスが使用可能。 ABP_APPD_DESCR_MAPPABLE_WRITE_PD：value アトリビュートに対して remap サービスが使用可能。 ABP_APPD_DESCR_MAPPABLE_READ_PD：value アトリビュートに対して remap サービスが使用可能。 これらの記述子は、論理的に OR 結合することができます。 この例では、ALL_ACCESS は上記のすべての記述子が論理的に OR 結合されたものになっています。 注：構造体データ型では無視されます。
pxValuePtr	ローカル値変数へのポインター。データ型は bDataType によって決まります。 注：構造体データ型では無視されます。
pxValuePropPtr	ローカル値プロパティ構造体へのポインター。NULL の場合、プロパティは適用されません（max/min/default）。データ型は bDataType によって決まります。 注：構造体データ型では無視されます。
psStruct	AD_StructDataType へのポインター。非構造体データ型に対しては NULL を設定してください。このフィールドを有効にするには、ABCC_CFG_STRUCT_DATA_TYPE を定義します。（任意。40 シリーズのみ）
pnGetAdiValue	ADI 値の取得時に call される ABCC_GetAdiValueFuncType へのポインター。（任意）
pnSetAdivalue	ADI 値の設定時に call される ABCC_SetAdiValueFuncType へのポインター。（任意）

abcc_drv/inc/abcc_ad_if.h における使用例を参照してください。

- リード / ライトプロセスデータのマッピング - プロセスデータとしてマッピングすべき ADI は、AD_DefaultMapType を用いてマッピングします。リードプロセスデータとライトプロセスデータの両方に対し、結合リストが 1 つだけ存在します。

データマッピング項目	説明
iInstance	マッピングする ADI の ADI 番号（上記の ADI エントリリストを参照）
eDir	マッピングの方向。デフォルトのマッピングリストの終端を示すには、PD_END_MAP を設定します。
bNumElem	マッピングする要素の数。配列または構造体の場合、1 より大きい値のみ設定できます。 AD_DEFAULT_MAP_ALL_ELEM は、すべての要素をマッピングする必要があることを表します。 インスタンスが AD_MAP_PAD_ADI である場合、bNumElem はパッドを挿入するビット数を表します。
bElemStartIndex	配列または構造体における開始要素を表すインデックス。ADI が配列でも構造体でもない場合は 0 を入力します。

マッピングは、ネットワークに現れる順番で行われます。

注：マッピングシーケンスは AD_DEFAULT_MAP_END_ENTRY で終了します。リストの終端には AD_DEFAULT_MAP_END_ENTRY がなければなりません。Anybus CompactCom ドライバーは、設定シーケンスにおいて、ABCC_CbfAdiMappingReq() を call してこの情報を要求します。

例：

```
{ ADI instance no, direction, number of elements in ADI to be mapped, index of
starting element in ADI to be mapped }
AD_DefaultMapType AD_asDefaultMap
[
    { 3, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 5, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 6, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 1, PD_READ, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 2, PD_READ, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 500, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 501, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 502, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 4, PD_READ, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { 503, PD_READ, AD_DEFAULT_MAP_ALL_ELEM ,0 },
    { AD_DEFAULT_MAP_END_ENTRY}
];
```

abcc_drv/inc/abcc_ad_if.h における使用例を参照してください。

プロセスデータのコールバック

ネットワークからリードプロセスデータを受信したことや、ライトプロセスデータを更新する時期が来たことをホストに通知するためには、プロセスデータの更新に関する 2 つのコールバック関数を実装しなければなりません。example_app/appl_abcc_handler.c に、そのサンプルが用意されています。

- ABCC_CbfUpdateWriteProcessData() - 現在のライトプロセスデータを更新します。関数から戻る前に、データをバッファにコピーしなければなりません。
- ABCC_CbfNewReadPd() - ネットワークから新たなプロセスデータを受信したときに call されます。関数から戻る前に、プロセスデータをアプリケーションの ADI にコピーする必要があります。

このサンプルコードでは、以下に示すように、どちらの場合もアプリケーションデータオブジェクト内でサービスを call して情報を更新しています。これらの関数は、通常、どのプロセスデータマッピングでも機能しますが、あらゆるケースに対する考慮が必要であるため、動作は低速です。パフォーマンスを改善するには、各アプリケーションに合わせて更新関数を記述することを検討してください。

```
void ABCC_CbfNewReadPd( void* pxReadPd )
{
  /*
  ** AD_UpdatePdReadData は、現在のマッピングに基づきすべての ADI を更新する汎用関数です。
  ** ADI のマッピングが固定の場合、memcpy を使用するなど、より最適化された方法で処理できる可能性があります。
  */

  AD_UpdatePdReadData( pxReadPd );
}
BOOL ABCC_CbfUpdateWriteProcessData( void* pxWritePd )
{
  /*
  ** AD_UpdatePdWriteData は、現在のマッピングに基づきすべての ADI を更新する汎用関数です。
  ** ADI のマッピングが固定の場合、memcpy を使用するなど、より最適化された方法で処理できる可能性があります。
  */

  return( AD_UpdatePdWriteData( pxWritePd ) );
}
```

イベント処理

40 シリーズのみ。

イベントモードでは、全てのイベントは、以下に示す設定に関する定義（この定義は abcc_drv_cfg.h にあります）を使用することで、ABCC_CbfEvent() インターフェースを介してユーザーに転送されるように設定することができます。

```
#define ABCC_CFG_INT_ENABLE_MASK_PAR (ABP_INTMASK_RDPDIEN | ABP_INTMASK_RDMSGIEN)
#define ABCC_CFG_HANDLE_INT_IN_ISR_MASK (ABP_INTMASK_RDPDIEN)
```

上記の設定では、メッセージ読み込み割り込みとプロセスデータ読み込み割り込みが有効になります。ただし、プロセスデータ読み込みのコールバックは、割り込みコンテキスト内でドライバーによって直接実行されます。関数 ABCC_CbfEvent() を call する事でメッセージ読み込みイベントがアプリケーションに転送されます。

これにより、プロセスデータ処理においてジッターの原因となる、ISR で処理される量が減ります。各イベントのパフォーマンスを改善するために、ユーザーによってこれ以外の設定を選択することも当然可能です。この時点で、ユーザーは選択したどのコンテキストからもイベント処理を起動できます。

注：メッセージ処理が完全にイベント駆動型であり、メッセージが割り込みコンテキスト内で送信される場合、abcc_adapt/abcc_sw_port.h にクリティカルセクションの移植を実装することを検討してください。クリティカルセクションの関数は abcc_drv/inc/abcc_port.h に記述されています。

イベントハンドラーのコールバックのサンプルでは、イベント処理タスクを起動することができます。

```
void ABCC_CbfEvent( UINT16 iEvents )
{
  if( iEvents & ABCC_EVENT_RDMSGI )
  {
    ABCC_fRdMsgEvent = TRUE;
  }
}
```

上記のコードは、ドライバーのイベントコールバックによりタスク（下記）がどのように起動されるかを示しています。

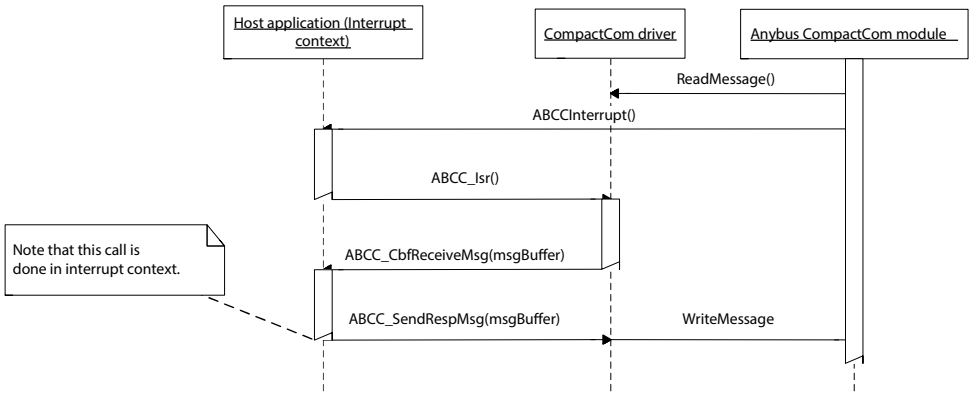
```
volatile BOOL ABCC_fRdMsgEvent = FALSE;

void Task( void )
{
  ABCC_fRdMsgEvent = FALSE;

  while( 1 )
  {
    if( ABCC_fRdMsgEvent )
    {
      ABCC_fRdMsgEvent = FALSE;
      ABCC_TriggerReceiveMessage();
    }
  }
}
```

このコードでは、メッセージ受信イベントを処理するタスクが記述されています。

割り込みコンテキストにおけるイベント処理

40 シリーズのみ。

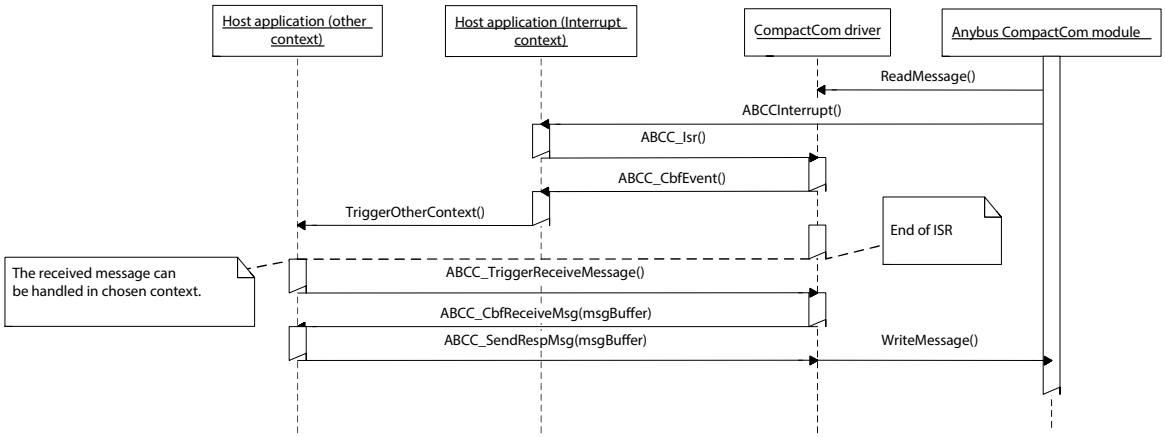
```
#define   ABCC_CFG_INT_ENABLED                ( TRUE )
#define   ABCC_CFG_INT_ENABLE_MASK_PAR        ( ABP_INTMASK_RDMSGIEN )
#define   ABCC_CFG_HANDLE_INT_IN_ISR_MASK     ( ABP_INTMASK_RDMSGIEN )
```



コールバック関数 ABCC_CbfEvent() を用いたイベント処理

40 シリーズのみ。

```
#define   ABCC_CFG_INT_ENABLED                ( TRUE )
#define   ABCC_CFG_INT_ENABLE_MASK_PAR        ( ABP_INTMASK_RDMSGIEN )
#define   ABCC_CFG_HANDLE_INT_IN_ISR_MASK     ( 0 )
```



メッセージ処理

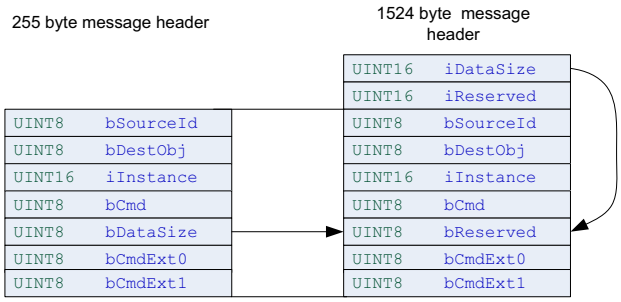
メッセージ処理のインターフェース関数は、abcc.h に記述されています。

コマンドメッセージを送信する場合、ユーザーはメッセージメモリーバッファを取得する為に関数 ABCC_GetCmdMsgBuffer() を使用しなくてはいけません。応答を受信する際、ユーザーは、応答ハンドラ関数のコンテキスト内にある応答バッファから、必要なデータを処理またはコピーしなければなりません。

これ以上メモリバッファを利用できない場合、関数 ABCC_GetCmdMsgBuffer() は NULL ポインターを返す場合があります。ユーザーは、後でメッセージを再送するか、致命的なエラーとして処理する必要があります。

注：バッファのリソースは abcc_drv_cfg.h で設定されています。

注：CompactCom 40 シリーズのモジュールは、最大 1524 バイトのメッセージデータを処理できます。一方、CompactCom 30 シリーズは 255 バイトまでしか処理できません。1524 バイトのメッセージをサポートするメッセージヘッダーは、サイズフィールドが 8 ビットではなく 16 ビットである必要があるため、30 シリーズとは形式が異なります。ドライバーは、30 シリーズのモジュールとの通信だけでなく、40 シリーズのモジュールとの通信もサポートしますが、ドライバー API では新しいメッセージ形式のみサポートします。30 シリーズのモジュールを使用する場合、ドライバー内部で従来のメッセージ形式に変換されます。以下の図に、2 つのメッセージ形式を示します。



例 1：コマンドの送信と応答の受信

ドライバーは、コマンドを送信する際、ソース ID を応答関数（この例では appl_HandleResp()）に結び付けます。

ソース ID が一致する応答を受信すると、ドライバーにより関数 appl_HandleResp() が call されます。

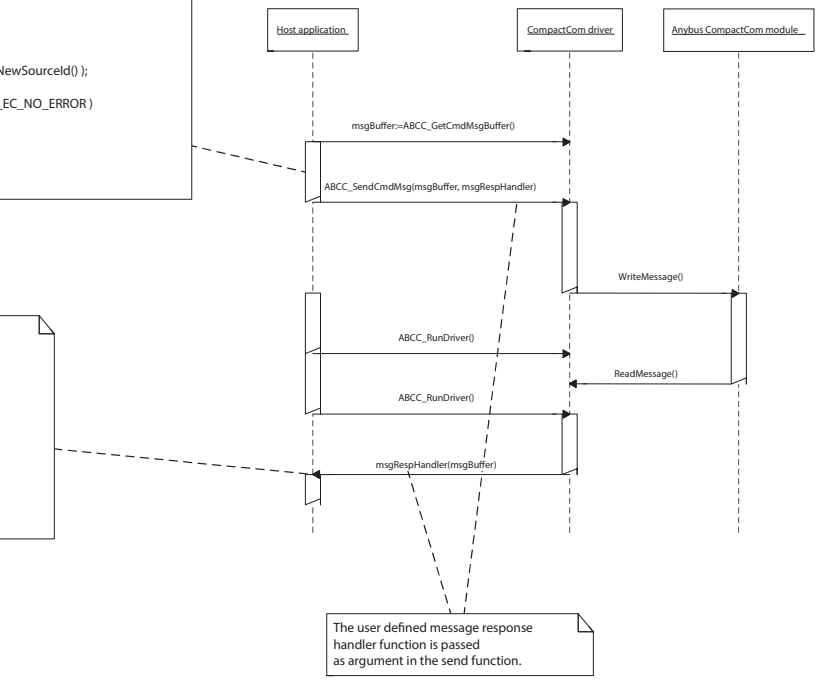
なお、受信メッセージバッファを解放する必要はありません。このバッファは、appl_HandleResp() から戻った後に、ドライバー内部で解放されます。

```
void appl_HandleResp( ABP_MsgType* psMsg )
{
  HandleResponse(psMsg);
}
```

Sending a command to the CompactCom

```
psMsg = ABCC_GetCmdMsgBuffer();

if( psMsg != NULL )
{

  ABCC_GetAttribute( psMsg, ABP_OBJ_NUM_ANB, 1,
                     ABP_ANB_IA_EXCEPTION, ABCC_GetNewSourceId() );

  if( ABCC_SendCmdMsg( psMsg, msgRespHandler ) != ABCC_EC_NO_ERROR )
  {
    APPL_UnexpectedError();
  }
}
```

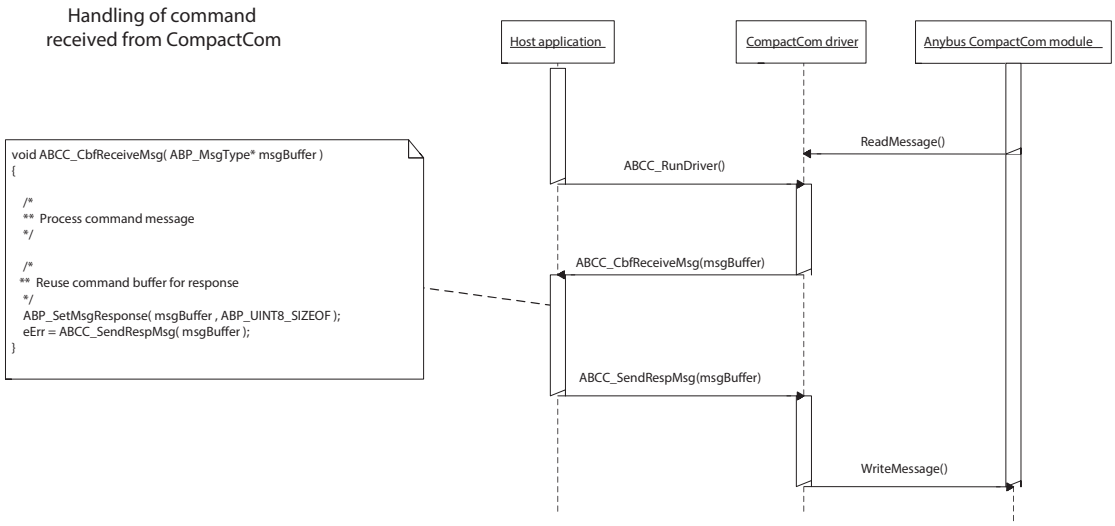
```
static void msgRespHandler( ABP_MsgType* psMsg )
{

  if( ABCC_VerifyMessage( psMsg ) != ABCC_EC_NO_ERROR )
  {
    APPL_UnexpectedError();
    return;
  }

  /*
  ** Handle response data
  */
}
```



例 2：コマンドの受信と応答の送信

注：受信コマンドバッファは応答の送信に再利用されます。

```
void appl_ProcessCmdMsg( ABP_MsgType* psNewMessage )
{
  /* 応答送信用にコマンドバッファを再利用 */
  ABP_SetMsgResponse( psNewMessage, ABP_UINT8_SIZEOF );
  eErr = ABCC_SendRespMsg( psNewMessage );
}
```



ドライバーは、非ブロック形式の Anybus CompactCom メッセージ処理を使用します。これは、ステートマシンを使用してコマンドと応答を追跡する必要があることを意味します。

example_app/appl_abcc_handler.c では、テンプレートとして使用できる 2 つのステートマシンの例が用意されています。

例 1：ABCC_CbfUserInitReq() が call されると、ABCC_UserInitComplete() が call される前に、IP アドレスまたはノードアドレスが設定されます。

例 2：Anybus CompactCom が例外状態を示すと、例外コードが読み込まれます。

Appendix A

A. ソフトウェア概要

A.1 ファイルとフォルダ

フォルダ	説明
$(ROOT)\abcc_abp	このフォルダには、Anybus のプロトコルファイルがすべて格納されています。このフォルダは、Anybus CompactCom の新しいソフトウェアリリースが利用可能になると更新されることがありますが、それ以外の場合は読み取り専用になっています。このフォルダに含まれるファイルは読み取り専用とみなされます。
$(ROOT)\abcc_drv\inc	アプリケーションに公開する .h ファイル。このフォルダには、アプリケーション用のほか、ドライバーのシステム依存部分用のドライバー設定ファイルが格納されています。このフォルダに含まれるファイルは読み取り専用とみなされます。
$(ROOT)\abcc_drv\src	Anybus CompactCom ドライバーの実装。このフォルダに含まれるファイルは読み取り専用とみなされます。
$(ROOT)\abcc_adapt	このフォルダには、ドライバーやオブジェクトの適合用ファイルと設定ファイルがすべて格納されています。これらのファイルは、ドライバーやサンプルコードの設定と適合の為に、ユーザーにより修正されなければいけません。
$(ROOT)\abcc_obj	このフォルダには、Anybus ホストオブジェクトの実装がすべて格納されています。これらのファイルはユーザーにより変更されます。
$(ROOT)\example_app	サンプルアプリケーション。これらのファイルはユーザーにより変更されます。

A.2 ルートファイル

フォルダ	説明
$(ROOT)\main.c	サンプルアプリケーションのメインファイル。
$(ROOT)\abcc_versions.h	サンプルコード、ドライバー、ABP のバージョン定義が格納されています。

A.3 CompactCom ドライバーインターフェース（読み取り専用）

ファイル名	説明
\abcc_drv\inc\abcc.h	Anybus CompactCom ドライバーの公開インターフェース。
\abcc_drv\inc\abcc_ad_if.h	ADI マッピングの型定義。
\abcc_drv\inc\abcc_cfg.h	ドライバーの設定パラメーター。
\abcc_drv\inc\abcc_port.h	Anybus CompactCom を各種プラットフォームに移植するための定義。
\abcc_drv\inc\abcc_sys_adapt.h	すべての動作モードに共通な、ターゲット依存関数のインターフェース。
\abcc_drv\inc\abcc_sys_adapt_spi.h	abcc_spi_drv.c で必要とされる、ターゲット依存関数のインターフェース。
\abcc_drv\inc\abcc_sys_adapt_par.h	abcc_par_drv.c で必要とされる、ターゲット依存関数のインターフェース。
\abcc_drv\inc\abcc_sys_adapt_ser.h	abcc_ser_drv.c で必要とされる、ターゲット依存関数のインターフェース。

A.4 内部ドライバーファイル（読み取り専用）

/abcc_drv/src フォルダに含まれるファイルの中身は変更しないでください。

ファイル名	説明
\abcc_drv\src\abcc_drv_if.h	特定の動作モードを実装するための、下位レベルドライバーのインターフェース。
\abcc_drv\src\abcc_debug_err.h \abcc_drv\src\abcc_debug_err.c	デバッグおよびエラー報告のためのヘルプマクロ。
\abcc_drv\src\abcc_link.c \abcc_drv\src\abcc_link.h	メッセージバッファの処理およびメッセージキューの処理。
\abcc_drv\src\abcc_mem.c \abcc_drv\src\abcc_mem.h	abcc_link.c により使用される、メッセージリソースメモリのサポート。
\abcc_drv\src\abcc_handler.h \abcc_drv\src\abcc_handler.c	Anybus CompactCom ハンドラーの実装（ハンドラーにおける動作モード依存部分）。
\abcc_drv\src\abcc_setup.h \abcc_drv\src\abcc_setup.c	Anybus CompactCom ハンドラーの実装（ステートマシンの設定）。
\abcc_drv\src\abcc_remap.c	実行時にプロセスデータを再マッピングするための、Anybus CompactCom ハンドラーの実装。
\abcc_drv\src\abcc_timer.h \abcc_drv\src\abcc_timer.c	Anybus CompactCom ドライバーのタイムアウト機能のサポート。

8 / 16 ビットパラレルイベントに固有のファイル

ファイル名	説明
\abcc_drv\src\par\abcc_handler_par.c	ABCC_RunDriver() と ABCC_ISR() を実装します。
\abcc_drv\src\par\abcc_par_drv.c	パラレル動作モードのドライバーを実装します。
\abcc_drv\src\par\abcc_drv_par_if.h	パラレルドライバーのインターフェースを実装します。

SPI に固有のファイル

ファイル名	説明
\abcc_drv\src\par\abcc_handler_spi.c	ABCC_RunDriver() と ABCC_ISR() を実装します。
\abcc_drv\src\spi\abcc_spi_drv.c	SPI 動作モードのドライバーを実装します。
\abcc_drv\src\spi\abcc_drv_spi_if.h	SPI ドライバーのインターフェースを実装します。
\abcc_drv\src\spi\abcc_crc32.c \abcc_drv\src\spi\abcc_crc32.h	SPI により使用される CRC32 の実装。

8 ビットパラレルピンポンに固有のファイル

ファイル名	説明
\abcc_drv\src\par30\abcc_handler_par30.c	ABCC_RunDriver() と ABCC_ISR() を実装します。
\abcc_drv\src\par30\abcc_par30_drv.c	パラレル 30 ピンポン動作モードのドライバーを実装します。
\abcc_drv\src\par30\abcc_drv_par30_if.h	パラレル 30 ピンポンドライバーのインターフェースを実装します。

シリアルに固有のファイル

ファイル名	説明
\abcc_drv\src\serial\abcc_handler_ser.c	ABCC_RunDriver() と ABCC_ISR() を実装します。
\abcc_drv\src\serial\abcc_serial_drv.c	シリアル動作モードのドライバーを実装します。
\abcc_drv\src\serial\abcc_drv_ser_if.h	シリアルドライバーのインターフェースを実装します。
\abcc_drv\src\serial\abcc_crc16.c \abcc_drv\src\serial\abcc_crc16.h	シリアルが使用する CRC32 の実装。

A.5 システム適合ファイル

ファイル名	説明
\abcc_adapt\abcc_drv_cfg.h	CompactCom ドライバーのユーザー設定。設定パラメーターは、ドライバーの公開インターフェース interface abcc_cfg.h にドキュメント化されています。
\abcc_adapt\abcc_identification.h	CompactCom モジュールの識別パラメーターをセットするためのユーザー設定。
\abcc_adapt\abcc_obj_cfg.h	Anybus オブジェクト実装用のユーザー設定。
\abcc_adapt\abcc_sw_port.c	CompactCom ドライバーと Anybus オブジェクトの実装で必要とされる、プラットフォーム依存マクロと関数。
\abcc_adapt\abcc_sw_port.h	CompactCom ドライバーと Anybus オブジェクトの実装で必要とされる、プラットフォーム依存マクロと関数。マクロの説明は abcc_port.h に記述されています。abcc_port.h は、CompactCom の公開ドライバーインターフェースにあります。
\abcc_adapt\abcc_sys_adapt.c	-
\abcc_adapt\abcc_td.h	CompactCom の種類の定義。
\abcc_adapt\abcc_platform_cfg.h	abcc_adapt/abcc_obj_cfg.h と abcc_adapt/abcc_identification.h の定義を上書きするための、プラットフォーム固有の定義。

Appendix B

B. API

B.1 API のドキュメント

Anybus CompactCom の API レイヤーでは、各ネットワークアプリケーションから Anybus CompactCom ドライバーへの共通インターフェースが定義されています。このインターフェースは abcc.h に記述されています。

API 関数

関数	説明
ABCC_StartDriver()	ドライバーを起動し、割り込みを有効にし、動作モードを設定します。 この関数を call すると、タイマーシステムを起動できるようになります。 注！この関数はモジュールのリセットを解除しません。
ABCC_IsReadyforCommunication()	ABCC_StartDriver() の後で、この関数が TRUE を返すまでこの関数を繰り返し実行する必要があります。ABCC_StartDriver() が TRUE を返した場合、モジュールの通信準備が完了して CompactCom のセットアップシーケンスが開始されたことを意味します。
ABCC_ShutdownDriver()	ドライバーを停止して SHUTDOWN 状態にします。
ABCC_HWReset()	モジュールのハードウェアリセットをします。 この関数から ABCC_ShutdownDriver() が call されます。 注！この関数は、単にリセットピンをローにするだけです。十分長いリセット時間（ABCC_HWReset() の call から ABCC_HWReleaseReset() の call までの時間）を確保することは、呼び出し側の責任です。
ABCC_HWReleaseReset()	モジュールのリセットを解除します。
ABCC_RunTimerSystem()	CompactCom ドライバーの各タイマーを処理します。タイマー割り込みから定期的にこの関数を call することを推奨します。この関数を実装しない場合、タイムアウトやウォッチドッグの機能は動作しません。
ABCC_RunDriver()	CompactCom ドライバーの送受信メカニズムを駆動します。ポーリング時、このメインルーチンを周期的に call する必要があります。 TRUE：ドライバーが起動し、通信準備ができている。 FALSE：ドライバーが停止している（または起動されていない）。
ABCC_UserInitComplete()	ユーザー固有のセットアップからの最後の応答を受信したとき、アプリケーションからこの関数を call する必要があります。これにより、CompactCom のセットアップシーケンスが終了し、ABCC_SETUP_COMPLETE が送信されます。
ABCC_SendCmdMsg()	モジュールにコマンドメッセージを送信します。
ABCC_SendRespMsg()	モジュールに応答メッセージを送信します。
ABCC_SendRemapRespMsg()	モジュールに remap 応答を送信します。
ABCC_SetAppStatus()	abp.h の ABP_AppStatusType に基づいて、現在のアプリケーション状態を設定します。
ABCC_GetCmdMsgBuffer()	コマンドメッセージバッファを割り当てます。
ABCC_ReturnMsgBuffer()	メッセージバッファを解放します。
ABCC_TakeMsgBufferOwnership()	メッセージバッファの所有権を取得します。
ABCC_ModCap()	モジュールの機能を読み取ります。この関数は、CompactCom パラレル動作モードでのみサポートされます。
ABCC_LedStatus()	LED の状態を読み取ります。SPI および CompactCom パラレル動作モードでのみサポートされます。
ABCC_AnbState()	現在の Anybus の状態を読み取ります。

API のイベントに関する関数

関数	説明
ABCC_ISR()	受信した CompactCom のイベントをアクノリッジして処理するには、CompactCom の割り込みルーチン内からこの関数を call する必要があります。
ABCC_TriggerRdPdUpdate()	RdPd の読み込みを開始します。
ABCC_TriggerReceiveMessage()	メッセージ受信の読み込みを開始します。
ABCC_TriggerWrPdUpdate()	アプリケーションから新たなプロセスデータが利用可能になり、CompactCom に送信されることを表します。
ABCC_TriggerAnbStatusUpdate()	Anybus の状態変化をチェックします。
ABCC_TriggerTransmitMessage()	送信キューをチェックします。

API のコールバック

これらの関数は、すべてアプリケーションで実装する必要があります。

関数	説明
ABCC_CbfAdiMappingReq()	この関数は、ドライバーがプロセスデータの自動マッピングを開始しようとするときに call されます。 この関数は、プロセスデータ読み書き用のマッピング情報を返します。
ABCC_CbfUserInitReq()	この関数は、モジュールがセットアップ状態にあるときにユーザー固有のセットアップを開始するために call されます。
ABCC_CbfUpdateWriteProcessData()	現在のライトプロセスデータを更新します。関数から戻る前に、データをバッファにコピーしなければなりません。
ABCC_CbfNewReadPd()	新たなプロセスデータを受信したときに call されます。関数から戻る前に、プロセスデータをアプリケーションの ADI にコピーする必要があります。
ABCC_CbfReceiveMsg()	モジュールからメッセージを受信しました。これは、モジュールから受信したコマンドをすべて受信する関数です。
ABCC_CbfWdTimeout()	この関数は、モジュールとの通信が失われたときに call されます。
ABCC_CbfWdTimeoutRecovered()	最近 CompactCom のウォッチドッグのタイムアウトが発生したが、現在は通信が再度機能していることを表します。
ABCC_CbfRemapDone()	このコールバックは、REMAP 応答がモジュールに正しく送信されたときに call されます。
ABCC_CbfAnbStatusChanged()	このコールバックは、モジュールの状態が変化したとき（すなわち、Anybus の状態または監視状態が変化したとき）に call されます。
ABCC_CbfEvent()	処理されないイベントに対して call されます。 処理されないイベントとは、ABCC_USER_INT_ENABLE_MASK で有効になっているが、ABCC_USER_HANDLE_IN_ABCC_ISR_MASK で指定されていないイベントのことです。
ABCC_CbfSync_Isr()	シンクがサポートされている場合、シンクイベント発生時にこの関数が call されます。

サポート関数

関数	説明
ABCC_NetworkType()	ネットワークの種類を取得します。
ABCC_ModuleType()	モジュールの種類を取得します。
ABCC_DataFormatType()	ネットワークのエンディアンネスを取得します。
ABCC_ParameterSupport()	パラメーターのサポートを取得します。
ABCC_GetOpmode()	ABCC_SYS_GetOpmode() を call して、ハードウェアから動作モードを読み取ります。
ABCC_GetAttribute()	アトリビュートを取得するためのパラメーターを CompactCom のメッセージに設定します。
ABCC_SetByteAttribute()	アトリビュートを設定するためのパラメーターを CompactCom のメッセージに設定します。
ABCC_VerifyMessage()	CompactCom の応答メッセージを検証します。
ABCC_GetDataTypeSize()	ABP データ型のサイズを返します。

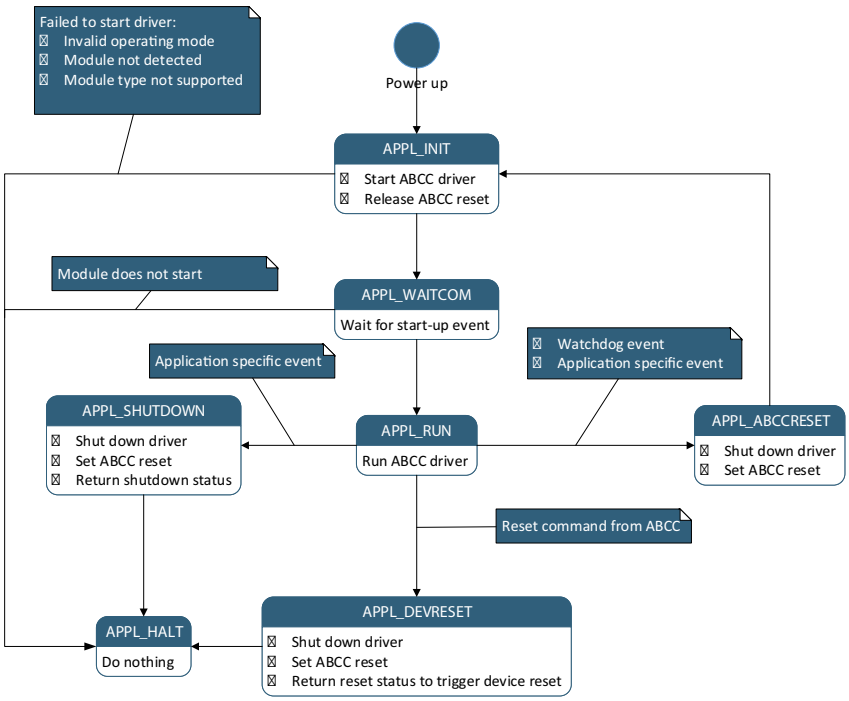
Appendix C

C. ホストアプリケーションのステートマシン

サンプルコードのアプリケーションフローは、以下のフローチャートに示すステートマシンで維持されています。

メインループから周期的に call される関数 APPL_HandleAbcc() によってステートマシンが実行され、この関数により各状態のさまざまなタスクが実行されます。

APPL_HandleAbcc() がはじめて call されたときに、APPL_INIT 状態になります。



APPL_INIT

- Anybus CompactCom 機器が検出されたかをチェックします。
- 任意の ADI マッピングを使用して、アプリケーションデータオブジェクトを起動します。この例では、30 ページの「サンプルアプリケーション」で説明した 3 つの ADI マッピングのいずれかを使用します。
- ドライバーを起動する為に、ABCC_StartDriver() が call されます。
- Anybus CompactCom 機器のリセットを解除する為に、ABCC_HwReleaseReset() が call されます。
- APPL_WAITCOM 状態にします。

APPL_WAITCOM

- Anybus CompactCom 機器から通信準備完了の信号が出力されるのを待ちます。
- APPL_RUN 状態にします。

APPL_RUN

- ドライバーを実行する為に、ABCC_RunDriver() が call されます。特定のイベントに対してコールバックが call されます。ドライバーが使用するコールバックにはすべて、ABCC_Cbf<x>() という名前が付けられています。必要なコールバックはすべて、appl_abcc_handler.c に実装されています。

- 起動処理中、以下のイベントがドライバーによって（記述順に）起動されます。

  - Anybus CompactCom が ABP_ANB_STATE_SETUP 状態になると、ABCC_CbfStateChanged() が call されます。必要であれば、ブレークポイントを設定するか、またはデバッグ関数を使用して、状態の変化を表示してください。

  - Anybus CompactCom 機器がデフォルトのマッピングコマンドを送信できるようになると、ABCC_CbfAdiMappingReq() が call されます。汎用のサンプルコードでは、設定されたデフォルトマッピングのアプリケーションデータオブジェクトを要求します。

  - アプリケーションが、CompactCom 機器に情報を設定するための（または CompactCom 機器から情報を読み取るための）コマンドを送信できるようになると、ABCC_CbfUserInitReq() が call されます。サンプルコードでは、この関数によってユーザーによるステートマシンの初期化が開始され、CompactCom 機器にコマンドシーケンスが送信されます。メッセージに対する最後の応答を受信すると、関数 ABCC-UserInitComplete() が call され、ユーザーによる初期化シーケンスが終了したことがドライバーに通知されます。これにより、内部的にドライバーが起動され、SETUP_COMPLETE コマンドが CompactCom に送信されます。ユーザーによる初期化が必要ない場合、ABCC_CbfUserInitReq() から ABCC_UserInitComplete() を直接 call することができます。

  - セットアップが完了すると、CompactCom 機器は ABP_ANB_STATE_NW_INIT 状態になります。これは、ABCC_CbfStateChanged() が call されることを意味します。この状態では、CompactCom 機器からホストアプリケーションオブジェクトに各種コマンドが送信されます。受信したコマンドは、すべて ABCC_CbfReceiveMsg() で処理されます。コマンドに対する応答は、どのホストオブジェクトが実装されているか、そして abcc_identification.h と abcc_obj_cfg.h でどのような設定が行われているかによって異なります。必要であれば、どのコマンドが送信されて、それがどのように処理されたかが表示されるように、ABCC_CfgReceiveMsg() でブレークポイントを設定ししてください。

  - ネットワークの初期化が完了すると、CompactCom は ABP_ANB_STATE_WAIT_PROCESS 状態になります。ドライバーによって ABCC_CbfStateChanged() が再度 call されます。この時点で、ネットワークから IO 接続を設定できるようになります。

  - IO 接続が設定されると、CompactCom は ABP_ANB_STATE_PROCESS_ACTIVE 状態になります（あるいは、ネットワークによっては ABP_ANB_STATE_IDLE 状態になります）。CompactCom 機器からプロセスデータが送信されると、関数 ABCC_CbfNewReadPd() が call されます。すると、サンプルコードでは AD_UpdatePdReadData() が call されて、受信したデータがアプリケーションデータオブジェクトに転送され、ADI が更新されます。サンプルコードではデータのループのみ行います。そのため、関数本体の終了時、ABCC_TriggerWrPdUpdate() が call されてライトプロセスデータが更新されます。関数 ABCC_TriggerWrPdUpdate() により ABCC_CbfUpdateWriteProcessData() が起動されます。この関数は、ドライバーが新たなプロセスデータを送信できるようになるたびに call されます。
    更新されたプロセスデータが利用可能になったとき、ABCC_TriggerWrPdUpdate() を必ず call する必要があります。

  - ABP_ANB_STATE_EXCEPTION 状態になったとき、例外読み出しのステートマシンをアクティブにすることで、CompactCom 機器から例外の理由を読み出すことができます。RunExceptionSM() は、CompactCom 機器が ABP_ANB_STATE_EXCEPTION 状態にあるときに、APPL_RUN 状態から call されます。

- 機器の再起動を開始する為に、APPL_Reset() が call されます。この処理は、アプリケーションホストオブジェクトが CompactCom 機器からリセット要求を受信したときに行われます。すると、CompactCom ハンドラーのステートマシンが APPL_ABCCRESET 状態になります。

- 機器の再起動を開始する為に、APPL_Reset() と同様に APPL_RestartAbcc() が使用されます。この関数が call されると、CompactCom ハンドラーのステートマシンが APPL_ABCCRESET 状態になります。（現在、この関数はサンプルコードでは使われていません）この関数は電源のオフオンを発生させないため、APPL_Reset() の代わりに使用できます。

- ドライバーをシャットダウンする為に、APPL_Shutdown() が call されます。

APPL_SHUTDOWN

- Anybus CompactCom 機器のリセットの為に、ABCC_HWReset() が call されます。
- APPL_HALT 状態にします。

APPL_ABCCRESET

- Anybus CompactCom 機器のリセットの為に、ABCC_HWReset() が call されます。
- APPL_INIT 状態にします。

APPL_DEVRESET

- Anybus CompactCom 機器のリセットの為に、ABCC_HWReset() が call されます。
- APPL_HALT 状態にします。

（APPL_AbccHandler() からの関数呼び出しにより）メインループに値が返され、機器をリセットすべきであることが示されます。

APPL_HALT

何も動作しません。