Anybus® CompactCom™

ホストアプリケーション実装ガイド

HMSI-27-334-JA 1.2 日本語

Connecting Devices™

必ずお読みください

責任の範囲

本ドキュメントは細心の注意を払って作成されています。誤字や脱字があればHMS Industrial Networks ABにご指摘ください。本ドキュメントに記載されているデータや図表は、何ら拘束力を持ちません。HMS Industrial Networks ABは、製品開発に継続的に取り組むという自社のポリシーに基づき、製品に変更を加える権利を留保します。本ドキュメントの内容は予告なく変更される場合があります。また、本ドキュメントの内容はHMS Industrial Networks ABによる何らかの保証を表明するものではありません。HMS Industrial Networks ABは本ドキュメント内の誤りについて一切の責任を負いません。

本製品は様々な用途に応用可能です。本装置の使用者は、必要なあらゆる手段を通じて、本装置の用途が適用される法令、規則、規約、規格の定める性能・安全性に関する要件をすべて満たしていることを検証しなければならないものとします。

HMS Industrial Networks ABは、いかなる場合であっても、本製品のドキュメントに記載されていない機能やタイミング、機能の副作用によって生じた不具合について一切の責任を負いません。本製品のそのような特徴を直接または間接に使用したことで生じる影響（互換性の問題や安定性の問題など）は、本ドキュメントでは定義されていません。

本ドキュメントの例や図表は、説明のみを目的として使用されています。本製品の個々の使用においては様々なバリエーションや要件が存在するため、本ドキュメントの例や図表に基づいて本製品を使用したことに関して、HMS Industrial Networks ABは一切の責任を負いません。

知的所有権

本ドキュメントに記載されている製品に組み込まれた技術に関する知的所有権はHMS Industrial Networks ABに帰属します。この知的所有権には、米国およびその他の国における特許や出願中の特許が含まれます。

Anybus ® は HMS Industrial Networks AB の登録商標です。

CompactCom™ は HMS Industrial Networks AB の登録商標です。

その他の商標は、各所有者に帰属します。

目次

Page

このページは意図的に空白になっています

1 まえがき

1.1 本ドキュメントについて

本ドキュメントでは、サンプル・ホストアプリケーションコードについて説明します。この文書では基本的な実装をガイドし、さらなる複雑な開発のためのヒントを提供します。

追加の関連文書とファイルのダウンロードについては、www.anybus.com/supportをご覧ください。

1.2 関連ドキュメント

ドキュメント	作成者	ドキュメント**ID**
Anybus CompactCom 40 ソフトウェア・デザインガイド	HMS	HMSI-216-125

1.3 ドキュメント更新履歴

バージョン	日付	説明
1.00	2015/11/20	新規作成
1.10	2016/02/05	全面改訂版
1.2	2017/01/10	DOXへの変換 メジャーアップデート

1.4 表記と用語

順番通りに実行されなくてはならない指示については、番号の付いたリストが使用されます。

1. まずこれを行います
2. その後これを行います

順番のない指示については、番号付けのないリスト (箇条書き) が使用されます。

- 項目化された情報
- 任意の順序で実行できる指示

アクションと結果が対になる指示については、以下のように表記します。

- このアクションは...
  - ➡ この結果につながります

Bold typefaceはコネクタ、スイッチなどハードウェア上のインタラクティブな部品、またはグラフィックユーザーインターフェース上のメニューやボタンを示します。

等幅フォントはプログラムコードやコンフィグレーションスクリプトなどのデータ入出力表示などに使用されます。

これはこの文書内の相互参照です: 表記と用語, ページ 4

これは、外部リンク(URL)です:www.hms-networks.com

これはインストールおよび/または操作を容易にする可能性のある追加情報です。

機能の低減および/または機器への損傷のリスクを避けるため、またはネットワークのセキュリティのリスクを避けるために、この指示には従わなければなりません。

注意
個人の負傷のリスクを避けるため、この指示には従わなければなりません。

警告
死亡または重篤な障害のリスクを避けるため、この指示には従わなければなりません。

2 はじめに

Anybus CompactCom 30またはAnybus CompactCom 40の導入を開始するときは、サンプル・ホストアプリケーション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。サンプル・ホストアプリケーションコードには、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である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APIを利用したサンプルアプリケーションのコードも含まれています。

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を追って説明します。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。

この文書は、サンプル・ホストアプリケーションコード・バージョン*3.02*の内容を元にしています。

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限られた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、カスタマイズしたコードを製品に追加することと、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。この後、アプリケーションをさらに拡張、改良することができます。

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可能です。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可能です。

サンプル・ホストアプリケーションコードは、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 / 開発環境に必要なすべてのファイルが含まれています。

プラットフォーム	ツール / プロジェクト	説明
Generic	-	いずれのプラットフォームにも実装可能
Xilinx、MicroZed	GNU	Anybus IPと共にMicroZed評価プラットフォームに使用できます
ST、STM3240-EVAL	Keil µVision	STM3240-EVAL 評価プラットフォームに使用できます
	IAR Embedded Workbench	STM3240-EVAL 評価プラットフォームに使用できます
NXP、TWRP1025	Code Warrior	NXP TWRP1025 評価プラットフォームに使用できます
HMS、USB II ボード	Visual Studio	HMS 製スターターキット (USB ボード) に使用できます

2.1 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変更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に変更が必要な機能などがこれに該当します。下図は、サンプル・ホストアプリケーションコードの構成を示しています。

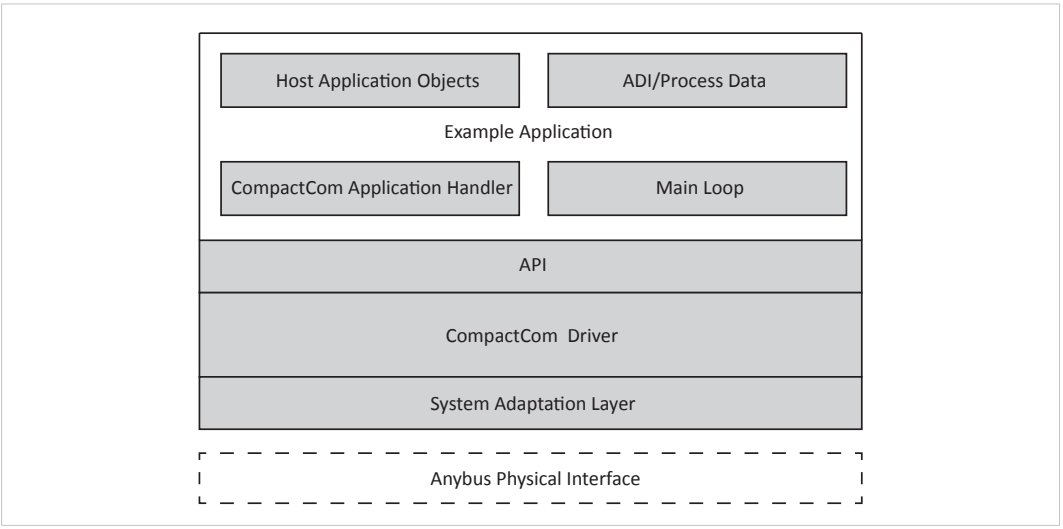


Fig. 1 ソフトウェア概要

サンプル・ホストアプリケーションコードは、機能とユーザーによるファイル修正の要否に応じて5つのフォルダに分けられています。

フォルダ 構造

/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含まれます。 Anybus CompactCom 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含まれます。 Anybus CompactCom 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	設定ファイルが含まれます。 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に対応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	ホストアプリケーションオブジェクトの実装が含まれます。 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可能です。
/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へのメッセージ送信パターンを示すステートマシン。 –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可能です。

2.2 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、実装時の判断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、または複数のモードが使用されるか
- どの通信インターフェースがAnybus CompactComとの通信に使用されるか
- どのようなネットワークが設計に使用されるか
- そのネットワークはAnybus CompactCom 40シリーズで利用可能か、それとも、Anybus 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。
- 最終製品において、ネットワーク上で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
  - 要素数
  - 読み出し/書き込みアクセス
  - アサイクリックアクセス、サイクリックアクセス
  - 最大/最小/デフォルト値
- 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。例ベンダーID、製品コード、ID番号など。

3 ステップ1

3.1 システムへ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、以下のことが実現されているはずです。

- 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サンプル・ホストアプリケーション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。

3.2 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。

3.2.1 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。ビッグエンディアンシステムの場合、 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        /* Big-endian host application */

/* #define ABCC_SYS_BIG_ENDIAN  */ /* Little-endian host application */
```

3.2.2 16ビットの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。16ビットcharシステムの場合、 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 bit char system */

/* #define ABCC_SYS_16_BIT_CHAR */ /* 8 bit char system */
```

3.2.3 バスエンディアンが異なる場合の拡張設定

外部パラレルデータバスのエンディアンが内部データバスののエンディアンと異なっている場合、この定義を有効にします。パラレル16ビットの動作モードが使用されていない場合、この定義は無視されます。

```
#define ABCC_CFG_PAR_EXT_BUS_ENDIAN_DIFF (FALSE)
```

3.2.4 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）

3.3 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。詳細な説明は、`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です。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。

3.3.1 通信インターフェースと動作モード

実装で使用する、ホストアプリケーションとAnybus 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とどのように通信するか、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、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数だけ、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 ) /* Parallel, 8/16-bit, event
mode */

#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 ) /* Serial */

#define ABCC_CFG_DRV_PARALLEL_30 ( TRUE ) /* Parallel, 8-bit, half
duplex */
```

ABCC_CFG_DRV_SERIAL および *ABCC_CFG_DRV_PARALLEL_30* は、処理データとメッセージデータのデータサイズが制限されている*Anybus CompactCom*半二重通信プロトコルを使用します。

- 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 を定義して、 abcc_adapt/abcc_sys_adapt.c の関数 ABCC_SYS_GetOpmode() を実装します。

```
#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。(下記のABCC_CFG_ABCC_OP_MODE_30 および ABCC_CFG_ABCC_OP_MODE_40 を参照。)

- Anybus CompactComホストコネクタ上の動作モードピンがホストプロセッサで操作できる場合は、 ABCC_CFG_OP_MODE_SETTABLE を定義して、関数 ABCC_SYS_SetOpmode() の実装を abcc_adapt/abcc_sys_adapt.cで行います。

```
#define ABCC_CFG_OP_MODE_SETTABLE ( TRUE )
```

定義されていない場合、Anybus 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 (Anybus CompactCom 30 および40) ごとに1つの動作モードだけが使用される場合、動作モードを ABCC_CFG_ABCC_OP_MODE_30 および ABCC_CFG_ABCC_OP_MODE_40で定義します。利用可能な動作モード (ABP_OP_MODE_X) は abcc_abp/abp.hで定義します。

```
#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
```

上記のいずれの定義も設定されていない場合、ABCC_SYS_GetOpmode() を実装して外部ハードウェアから動作モードを取得する必要があります。上記の ABCC_CFG_OP_MODE_GETTABLE を参照してください。

3.3.2 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。

Anybus CompactCom メモリへのダイレクトアクセスが可能な場合 (ホストコントローラーが外部 SRAMにアクセスする専用信号を提供している) は、 ABCC_CFG_MEMORY_MAPPED_ACCESS を TRUE に定義して、ベースアドレスを ABCC_CFG_PARALLEL_BASE_ADR で定義します (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されなくてはなりません) 。

```
#define ABCC_CFG_MEMORY_MAPPED_ACCESS ( TRUE )

#define ABCC_CFG_PARALLEL_BASE_ADR ( 0x00000000 )
```

Anybus CompactComのメモリへのダイレクトアクセスが利用できない場合、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 に実装しなくてはなりません (abcc_drv/inc/abcc_sys_adapt_par.hに記述されています) 。

可能であれば、より簡単に実装が行えるように、*Anybus 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、実装の迅速化につながります) 。

3.3.3 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サイズ (バイト長) は、 ABCC_CFG_SPI_MSG_FRAG_LENに定義されています。

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、フラグメントサイズ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、メッセージのフラグメントサイズ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オクテットまでのフラグメントサイズ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 )
```

3.3.4 モジュールIDとモジュール検出の設定

- Anybus CompactCom ホストコネクタ上のモジュール識別 (MI) ピン がホストプロセッサに接続されていない場合は、ABCC_CFG_MODULE_ID_PINS_CONN を FALSEに設定し、さらに ABCC_CFG_ABCC_MODULE_ID を使用中のデバイスのモジュール ID に対応する正しい値に設定しなければなりません。設定する場合は、 ABP_MODULE_ID_X に abcc_abp/abp.hで定義されている値を設定します。

  ABCC_CFG_MODULE_ID_PINS_CONN が TRUEとして定義されている場合、 abcc_adapt/abcc_sys_adapt.c 内の関数 ABCC_SYS_ReadModuleId() が実装される必要があります。

ホストコネクターのモジュール*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 */

/* #define ABCC_CFG_MODULE_ID_PINS_CONN ( TRUE ) */
```

- ホストコネクターのモジュール検出 (MD) ピン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 は TRUE に設定され、 ABCC_SYS_ModuleDetect() 関数が abcc_adapt/abcc_sys_adapt.c で実装されなくてはなりません。

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

3.3.5 メッセージとプロセスデータの設定

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。

```
#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 )

#define ABCC_CFG_CMD_SEQ_MAX_NUM_RETRIES ( 0 )

#define ABCC_CFG_MAX_NUM_CMD_SEQ ( 2 )
```

3.3.6 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、Anybus 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 を定義して、 abcc_adapt/abcc_sys_adapt.c 内の関数 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。これは ABCC_CFG_INT_ENABLED を FALSE に設定することを意味します。

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 )
```

3.3.7 通信ウォッチドッグの設定

Anybus CompactComウォッチドッグのタイムアウトは、 ABCC_CFG_WD_TIMEOUT_MSに設定されています。タイムアウトが発生した場合 、コールバック関数 ABCC_CbfWdTimeout() がコールされます。

ウォッチドッグ機能をサポートしているのは *SPI* 、シリアル、パラレル*30* (半二重) 動作モードだけです。

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

3.3.8 ADIの設定

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。

```
#define ABCC_CFG_STRUCT_DATA_TYPE ( FALSE )

#define ABCC_CFG_ADI_GET_SET_CALLBACK ( FALSE )

#define ABCC_CFG_64BIT_ADI_SUPPORT ( FALSE )
```

3.3.9 デバッグプリント設定

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、ドライバーのデバッグプリントに関連します。アプリケーションコードから追加のデバッグプリントを行う必要がある場合は、 abcc_adapt/abcc_sw_port.h 内の関数 ABCC_PORT_DebugPrint() を使用します。

- エラー通知コールバック関数 ABCC_CbfDriverError() を ABCC_CFG_ERR_REPORTING_ENABLEDで有効または無効にします。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 )
  ```

- ドライバーによるデバッグプリントのサポートを ABCC_CFG_DEBUG_EVENT_ENABLED で有効/無効にします。ABCC_PORT_DebugPrint() は abcc_adapt/abcc_sw_port.h に記載され、デバッグプリントに使用されます。

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

- ファイル名や行番号などのデバッグ情報のプリントは、ABCC_CbfDriverError() がコールされたとき実行され、ABCC_CFG_DEBUG_ERR_ENABLEDで有効/無効にします。

  ```
  #define ABCC_CFG_DEBUG_ERR_ENABLED ( FALSE )
  ```

- 送受信されたメッセージのプリントを ABCC_CFG_DEBUG_MESSAGINGで有効または無効にします。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プリントされます。

  ```
  #define ABCC_CFG_DEBUG_MESSAGING ( FALSE )
  ```

- コマンドシーケンサーの動作のプリントを ABCC_CFG_DEBUG_CMD_SEQ_ENABLEDで有効または無効にします。

  ```
  #define ABCC_CFG_DEBUG_CMD_SEQ_ENABLED ( FALSE )
  ```

3.3.10 起動時間

Anybus CompactCom IRQ ピンが接続されている場合は、Anybus CompactComが通信可能状態になるまで ABCC_CFG_STARTUP_TIME_MS がタイムアウト時間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 (APPL_MODULE_NOT_ANSWERING) が報告されます。割り込みピンが利用できない場合、Anybus CompactComモジュールと通信を開始するまでの待ち時間として ABCC_CFG_STARTUP_TIME_MS が使用されます。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 )
```

可能な場合、起動割り込み *(SPI* およびパラレル通信インターフェースで利用可能なオプション*)*の使用を推奨します。

3.3.11 Sync設定

40シリーズのみ

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。

```
#define ABCC_CFG_SYNC_ENABLE ( FALSE )

#define ABCC_CFG_SYNC_MEASUREMENT_IP ( FALSE )

#define ABCC_CFG_SYNC_MEASUREMENT_OP ( FALSE )
```

3.4 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。それらの関数は abcc_adapt/abcc_sys_adapt.cで実装され、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

3.4.1 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、Anybus CompactCom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度コールする必要があります。

注： この関数から戻ったときに、Anybus 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、ドライバーの起動および再起動時に、ドライバーによってコールされます。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()

ドライバー終了時にドライバーからコールされます。リソースが ABCC_SYS_Init() にて割り当てられている場合、この関数ではクローズまたは解放することが推奨されます。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Anybus CompactComからの割り込み信号を読み取り、 TRUE を割り込みピンがロー（すなわち割り込みがアクティブ）の場合に返します。割り込みが有効でない場合、この関数を使用して、Anybus 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。

3.4.2 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(ABCC_SYS_SpiDataReceivedCbfType pnDataReceived)

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

例：

```
static ABCC_SYS_SpiDataReceivedCbfType pnDataReadyCbf;

void ABCC_SYS_SpiRegDataReceived( ABCC_SYS_SpiDataReceivedCbfType
pnDataReceived )
{
   pnDataReadyCbf = pnDataReceived;
}
```

ABCC_SYS_SpiSendReceive(void* pxSendDataBuffer, void* pxReceiveDataBuffer, UINT16 iLength)

データの送受信をSPIモードで処理します。

2つのバッファが提供されます。1つは送信用のMOSI データフレーム用バッファ、もう1つは受信用の MISO フレームを格納するバッファです。

3.4.3 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コールされないため、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が使用され、 **`ABCC_CFG_MEMORY_MAPPED_ACCESS`** が定義されている場合も、このセクションは無視してかまいません。詳細については、 パラレル動作モード仕様 の **`ABCC_CFG_MEMORY_MAPPED_ACCESS`**説明部分を参照してください。

ABCC_SYS_ParallelRead()

Anybus 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。

Anybus CompactComのメモリから1オクテットを読み出します。

ABCC_SYS_ParallelRead16()

Anybus CompactComのメモリから1ワードを読み出します。

ABCC_SYS_ParallelWrite()

Anybus 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。

Anybus CompactComのメモリに1オクテットを書き込みます。

ABCC_SYS_ParallelWrite16()

Anybus CompactComのメモリに1ワードを書き込みます。

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書き込むアドレスを取得します。

3.4.4 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コールされないため、このセクションは無視して構いません。

ABCC_SYS_SerRegDataReceived(ABCC_SYS_SerDataReceivedCbfType pnDataReceived)

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。

例：

```
static ABCC_SYS_SerDataReceivedCbfType pnSerDataReadyCbf;

void ABCC_SYS_SerRegDataReceived( ABCC_SYS_SerDataReceivedCbfType
pnDataReceived )
{
   pnSerDataReadyCbf = pnDataReceived;
}
```

ABCC_SYS_SerSendReceive(void* pxTxDataBuffer, void* pxRxDataBuffer, UINT16 iTxSize, UINT16 iRxSize)

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(void)

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。

このコマンドはすべてのバッファを空にして、通信を再起動し、最後に提供された受信テレブラム長にて受信テレグラム待ちを開始します。

3.5 オブジェクトの設定

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

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情報のカスタマイズを行います。

3.6 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。APIは abcc_drv/inc/abcc.hに記述されています。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Anybus 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Anybus CompactComのコンセプト（インターフェース）の確認に使用できるほか、最終製品として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可能です。

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。

3.6.1 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（リードまたはライトプロセスデータ）。

現時点では、 appl_adimap_speed_example.c に説明されているマッピングが使用されます。これは、 /example_app/appl_adi_config.h の APPL_ACTIVE_ADI_SETUP が APPL_ADI_SETUP_SPEED_EXAMPLE として定義されていることを意味します。

- example_app/appl_adimap_speed_example.c - 速度と参照速度のシミュレーションです。
  - ADI 1:「速度」、UINT16、リードプロセスデータにマッピングされている
  - ADI 2:「参照速度」、UINT16、ライトプロセスデータにマッピングされている
  - データは関数 APPL_CyclicalProcessing() で操作されています
  - 構造体またはコールバックは使用しません。

3.6.2 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行われる場所です。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、Anybus CompactComとの通信に必要なハードウェアの起動を行います。起動時の初期化中に、1度コールされます。また、この関数から戻ったときに、Anybus 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、この関数を再度コールする必要はありません。`ABCC_HwInit()` は関数 `ABCC_SYS_HwInit()` を `abcc_adapt/abcc_sys_adapt.c`の中で起動します。この関数は現在のシステムに合わせてカスタマイズされる必要があります。この関数は、メイン関数で最初にコールする関数に含めてください。

- `APPL_HandleAbcc()` - この関数は、Anybus 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コールしなければなりません。この関数をコールするたびに、Anybus CompactComドライバーから状態が返されます。

`APPL_MODULE_NO_ERROR`	Anybus 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`	Anybus CompactCom は検出されませんでした。ユーザーに通知してください。
`APPL_MODULE_NOT_SUPPORTED`	未サポートの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`	考えられる原因：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`	Anybus 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。

- `ABCC_RunTimerSystem()` - この関数は、既知の周期（前回のコールからのミリ秒単位の時間）で定期的にコール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コールするか、タイマー割り込みを設定します。

  Anybus 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コール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。

> 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するようにしてください。

```
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 );
}
```

3.6.3 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。

4 ステップ2

4.1 適合とカスタマイズ

このステップをすべて終えると、以下のことが実現されているはずです。

- ネットワーク識別情報 (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。

4.1.1 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

- 応答を受信せずに送信できるメッセージコマンド数を ABCC_CFG_MAX_NUM_APPL_CMDS で設定することができます。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_APPL_CMDS ( 2 )
  ```

- 応答を送信せずに受信できるメッセージコマンド数を ABCC_CFG_MAX_NUM_ABCC_CMDS で設定することができます。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) を ABCC_CFG_MAX_MSG_SIZEで設定することができます。

---

*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 ( 255 )
```

- 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 )
  ```

- 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 ABCC_CFG_REMAP_SUPPORT_ENABLEDで有効または無効にします。 TRUE の場合、 ABCC_CbfRemapDone() をアプリケーションで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 )
  ```

- メッセージコマンドシーケンサー使用時の同時に使用できるメッセージコマンドシーケンスの最大数です。

  ```
  #define ABCC_CFG_MAX_NUM_CMD_SEQ ( 2 )
  ```

- 利用できるバッファがない場合に、エラーを報告する前にメッセージコマンドシーケンサーが実行する再試行回数を設定します。

  ```
  #define ABCC_CFG_CMD_SEQ_MAX_NUM_RETRIES ( 0 )
  ```

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。

- 割り込みルーチンとともにAnybus CompactComのIRQピンを使用するかどうかを ABCC_CFG_INT_ENABLED で定義します。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、Anybus CompactComの割り込みルーチンの内部からコールしてください。割り込みがエッジでトリガーされる場合、 ABCC_ISR() がコール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。 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 ABCC_CFG_INT_ENABLE_MASK_PAR の定義で設定します。利用可能なオプションは abcc_abp/abp.h に定義されています (割り込みマスクレジスタ) 。イベントがAnybus CompactCom割り込みを介して通知されなかった場合は、ドライバー関数 ABCC_RunDriver() (example_app/APPL_HandleAbcc() でコールされる) で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。割り込み有効マスク (ABCC_CFG_INT_ENABLE_MASK_X) で有効になっているが、ISRで処理するように設定されていないイベントは ABCC_ISR_EVENT_X のビットフィールドの定義に変換され (abcc_drv/inc/abcc.hに定義されている) 、 ABCC_CbfEvent() コールバックを介して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)

ADIの設定

- ADIの構造体データ型のサポートを ABCC_CFG_STRUCT_DATA_TYPEで有効にします。この定義は、 abcc_drv/inc/abcc_ad_if.h 内の AD_AdiEntryType (ユーザーのADIの定義に使用される) に影響を与えます。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、構造体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)

- ADIがリード・ライト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 ABCC_CFG_ADI_GET_SET_CALLBACKで設定します。この定義は、 abcc_drv/inc/abcc_ad_if.h 内の AD_AdiEntryType (ユーザーのADIの定義に使用される) に影響を与えます。ネットワークからADIがリードされると、そのアクションの前にコールバックが実行されます。ネットワークからADIがライトされると、そのアクションの後にコールバックが実行されます。

  #define ABCC_CFG_ADI_GET_SET_CALLBACK (FALSE)

- 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 ABCC_CFG_64BIT_ADI_SUPPORTで有効/無効にします。

  #define ABCC_CFG_64BIT_ADI_SUPPORT (FALSE)

Sync設定

40シリーズのみ。

- ドライバーによるSyncのサポートを有効/無効にします。TRUEの場合、abcc_CbfSyncIsr()をアプリケーションで実装しなくてはなりません。

  ```
  #define ABCC_CFG_SYNC_ENABLE ( FALSE )
  ```

Sync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。

Sync測定関数は、Sync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。

- （Sync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ABCC_CFG_SYNC_MEASUREMENT_IPで有効/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特別なテスト用バージョンをコンパイルするのに使用されます。ABCC_CFG_SYNC_MEASUREMENT_IPがTRUEのとき、WRPDが送信されるとABCC_SYS_GpioReset()がコールされます。SPI動作モードで実行中の場合は、ABCC_SpiRunDriver()がAnybusへのデータ送信終了したときにコールされます。ABCC_CFG_SYNC_MEASUREMENT_IPがTRUEのとき、ABCC_GpioSet()がInput Capture Pointでコールされる必要があります。

  ```
  #define ABCC_CFG_SYNC_MEASUREMENT_IP ( FALSE )
  ```

- （Sync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ABCC_CFG_SYNC_MEASUREMENT_OPで有効/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特別なテスト用バージョンをコンパイルするのに使用されます。ABCC_CFG_SYNC_MEASUREMENT_OPがTRUEのとき、ABCC_SYS_GpioSet()がRDPDI割り込みからコールされます。ABCC_CFG_SYNC_MEASUREMENT_OPがTRUEのとき、ABCC_GpioReset()をOutput Valid Pointでコールする必要があります。

  ```
  #define ABCC_CFG_SYNC_MEASUREMENT_OP ( FALSE )
  ```

4.1.2 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()**

  Anybus 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、Anybus CompactComの割り込みを有効にしたときにドライバーによってコールされます。

  ABCC_CFG_INT_ENABLEDがabcc_adapt/abcc_drv_cfg.hで定義されていない場合、この関数を実装する必要はありません。

- **ABCC_SYS_AbccInterruptDisable()**

  Anybus 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がabcc_adapt/abcc_drv_cfg.hで定義されていない場合、この関数を実装する必要はありません。

4.1.3 ネットワークの識別情報

ここまでは、ネットワークの設定はすべて無効のままになっており、製品はHMSの製品であると認識されています。次はネットワークの識別情報の設定を行います。

ホストアプリケーションオブジェクト ― ネットワーク

各ホストアプリケーションオブジェクトを abcc_adapt/abcc_obj_cfg.hファイルで定義し、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さらなる実装は abcc_obj フォルダで行われます。このフォルダには各オブジェクト用の.cおよび.hファイルが格納されています。

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情報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。そ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情報に関係しています。アトリビュートが有効になっている (TRUE) 場合、その値が使用されます。アトリビュートが無効になっている (FALSE) 場合、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)
*/
#ifndef EPL_IA_VENDOR_ID_ENABLE
   #define EPL_IA_VENDOR_ID_ENABLE                  TRUE
   #define EPL_IA_VENDOR_ID_VALUE                   0xFFFFFFFF
#endif

/*
** Attribute 2: Product Code type (UINT32 - 0x00000000-0xFFFFFFFF)
*/

#ifndef EPL_IA_PRODUCT_CODE_ENABLE
   #define EPL_IA_PRODUCT_CODE_ENABLE               TRUE
   #define EPL_IA_PRODUCT_CODE_VALUE                0xFFFFFFFF
#endif
```

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 では、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

ファイル abcc_adapt/abcc_obj_cfg.h には、サポートされる全ホストオブジェクトの全属性が、既に abcc_adapt/abcc_identification.hに定義されているものを除き含まれています。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。

ファイル *abcc_adapt/abcc_platform_cfg.h* はファイル *abcc_adapt/abcc_obj_cfg.h*、*abcc_adapt/abcc_identification.h* および *abcc_adapt/abcc_drv_cfg.h*のオブジェクトおよびアトリビュートを上書きするために使用できます。

定義を上書きするには、任意の定義を *abcc_adapt/abcc_platform_cfg.h* に追加するか、開発環境のグローバル定義セクションを使用します。

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。

4.1.4 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、プリント関数、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。これらの関数はファイル abcc_adapt/abcc_sw_port.h（ abcc_drv/inc/abcc_port.hに記載）に記載されています。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、実際のプロジェクトでは、目的のプラットフォームに合わせて最適化（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、Anybus 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。以下のリストから最適な実装を選択してください。最初にどの条件が当てはまるかによって、要件を選択します。

1. 以下の条件が当てはまる場合、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_UseCritical()` は、`ABCC_PORT_EnterCritical()` が必要とされるあらゆる宣言の前に使用される必要があります。
   - 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抜ける際、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. `ABCC_PORT_EnterCritical()` および `ABCC_PORT_ExitCritical()` は、以下のいずれかの条件が当てはまる場合、実装する必要があります。
   - `ABCC_RunTimerSystem()` はタイマー割り込みからコールされる。
   - アプリケーションにおいて、上位レベルの機能 (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 Anybus CompactCom ドライバーのメッセージインターフェースがアクセスされる。

     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抜ける際、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()` をコールする前に準備が必要な場合は、このマクロがプラットフォーム固有の必要機能を実装するために使用されます。

ABCC_PORT_EnterCritical()

この関数は、Anybus 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コールされます。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()

割り込みを `ABCC_PORT_EnterCritical()` がコール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。

4.1.5 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ADIマッピング `appl_adimap_speed_example.h` が使用されました。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プはファイル `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。ADIの設定方法詳細は、 `abcc_drv/abcc_ad_if.h`を参照してください。

- `example_app/appl_adimap_speed_example.h` – 速度のシミュレーションおよび参照速度です。

ADI	説明
ADI 1	"Speed"、UINT16 (入力データにマッピングされる)
ADI 2	"Ref Speed"、UINT16 (出力データにマッピングされる)

  - データは関数 APPL_CyclicalProcessing() で操作されています。
  - 構造体またはコールバックは使用しません。

- example_app/appl_adimap_simple16.c- このマップでは32個のワードデータ（16ビット）の折り返しを行います。

ADI	説明
ADI 1	32個のUINT16からなる配列（入力データにマッピングされる）
ADI 2	32個のUINT16からなる配列（出力データにマッピングされる）

  - 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は同じデータエリアを参照するため、データはループ（折り返し）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 コールバックを使用した例です。

ADI	説明
ADI 10	32個のUINT16からなる配列（出力データにマッピングされる）
ADI 11	32個のUINT16からなる配列（入力データにマッピングされる）
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、各方向のプロセスデータにマッピングされます。
  - ネットワークからADI 11がリードされたとき、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。
  - ネットワークからADI 10にライトしたとき、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。

ABCC_CFG_ADI_GET_SET_CALLBACK は、コールバックが使用されているので *abcc_adapt/abcc_drv_cfg.h* で有効になってなくてはなりません。詳細については、*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を使用した例です。

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

ABCC_CFG_STRUCT_DATA_TYPE は構造体データが使用されているので *abcc_adapt/abcc_drv_cfg.h* で有効になっていなくてはなりません。詳細については、*ADIの設定* を参照してください。

この例のデータを操作するための機能は実装されていません。

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 (実装で使用するデータインスタンス) は `AD_AdiEntryType` としてADIエントリ・リストに定義されていなくてはなりません。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	
iInstance	ADIインスタンス番号 (1 ~ 65535) 、0はクラス用に予約。	
pacName	ADI名 (char文字列、ADI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: ABP_SINT8: ABP_SINT16: ABP_SINT32: ABP_UINT8: ABP_UNIT16: ABP_UINT32: ABP_CHAR: ABP_ENUM: ABP_SINT64: ABP_UINT64: ABP_FLOAT: ABP_OCTET ABP_BITS8 ABP_BITS16 ABP_BITS32 ABP_BIT1 ABP_BIT2 : ABP_BIT7 ABP_PAD0 ABP_PAD1 : ABP_PAD16	ブール型 符号つき8ビット整数 符号つき16ビット整数 符号つき32ビット整数 符号なし8ビット整数 符号なし16ビット整数 符号なし32ビット整数 文字型 列挙型 符号つき64ビット整数 符号なし64ビット整数 浮動小数点数 (32ビット) 未定義の8ビットデータ (40シリーズのみ) 8ビットのビットフィールド (40シリーズのみ) 16ビットのビットフィールド (40シリーズのみ) 32ビットのビットフィールド (40シリーズのみ) 1ビットのビットフィールド (40シリーズのみ) 2ビットのビットフィールド (40シリーズのみ) : 7ビットのビットフィールド (40シリーズのみ) 0ビットのパディングフィールド (40シリーズのみ) 1ビットのパディングフィールド (40シリーズのみ) : 16ビットのパディングフィールド (40シリーズのみ)
	注：構造体データ型では無視されます。	
bNumOfElements	配列の場合：bDataTypeで指定されたデータ型の要素数を指定します。 構造体データ型の場合：構造体に含まれる要素数を指定します。	
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する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する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する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するremapサービスが使用可能。 これらの記述子は、論理的にOR結合することが可能です。 このサンプルでは、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	ローカル変数値へのポインター。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	ローカル変数値のプロパティ構造体 (最小 / 最大 / デフォルトを定義) へのポインター。NULLの場合、プロパティは適用されません。データ型はbDataTypeによって決まります。最小 / 最大 / デフォルトをアサイクリックメッセージで使用する場合は、abcc_adapt/abcc_obj_cfg.h. のアプリケーションデータオブジェクト (AD_IA_MIN_MAX_DEFAULT_ENABLE) を有効にする必要があります。 注：構造体データ型では無視されます。	

ADIエントリ項目	説明
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Get時にコールされるABCC_GetAdiValueFuncTypeへのポインター。（任意）
pnSetAdivalue	ADI値のSet時にコールされるABCC_SetAdiValueFuncTypeへのポインター。（任意）

サンプルコード内の様々な*ADI*エントリは *"const"* として定義されており、つまり情報は *ROM* に保存されます。しかし、コンパイル時に*ADI*リストがどのようなものになるか不明なケースがあります。そのようなケースでは、*const* 宣言を削除し、*ADI*エントリは `ABCC_RunDriver()` をコールする前に値が設定されなければなりません。そして、設定値は*RAM*に保存されます。

*ADI*での構造体データ型の使用については、`abcc_drv/inc/abcc_ad_if.h`の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を含む 1 つのリストが存在します。

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（リード/ライト）。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：マッピングのシーケンスは `AD_DEFAULT_MAP_END_ENTRY`で終了され、これは必ずリストの最後に存在しなくてはなりません。Anybus CompactComドライバーは、初期設定シーケンス中で `ABCC_CbfAdiMappingReq()` をコール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}
];
```

使用例を `abcc_drv/inc/abcc_ad_if.h`で参照してください。

プロセスデータ用コールバック

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。一例が example_app/appl_abcc_handler.cに記載されています。

- BOOL ABCC_CbfUpdateWriteProcessData(void* pxWritePd) - 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 (pxWritePd) にコピーしなければなりません。
- void ABCC_CbfNewReadPd(void* pxReadPd) - 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コールされます。関数から戻る前に、（バッファ pxReadPd から）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のサービスをコール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。

例：

```
void ABCC_CbfNewReadPd( void* pxReadPd )

{
  /*
  ** AD_UpdatePdReadData is a general function that updates all ADI:s
  ** according to current map.
  ** If the ADI mapping is fixed there is potential for doing that in a
  ** more optimized way, for example by using memcpy.
  */

   AD_UpdatePdReadData( pxReadPd );
}
BOOL ABCC_CbfUpdateWriteProcessData( void* pxWritePd )
{
  /*
  ** AD_UpdatePdWriteData is a general function that updates all ADI:s
  ** according to current map.
  ** If the ADI mapping is fixed there is potential for doing that in a
  ** more optimized way, for example by using memcpy.
  */

   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、すべてのイベントは `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ファイル `abcc_drv_cfg.h`内の下記定義で設定できます。

```
#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、リードメッセージデータ割り込みとリードプロセスデータ割り込みが有効になります。ただし、リードプロセスデータのコールバックは、割り込みコンテキスト内で直接ドライバーによって実行されます。リードメッセージデータのイベントは関数 `ABCC_CbfEvent()` をコールすることで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内の処理量を減らせます。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。

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次に記載されています *abcc_drv/inc/abcc_port.h*

例：

```
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。

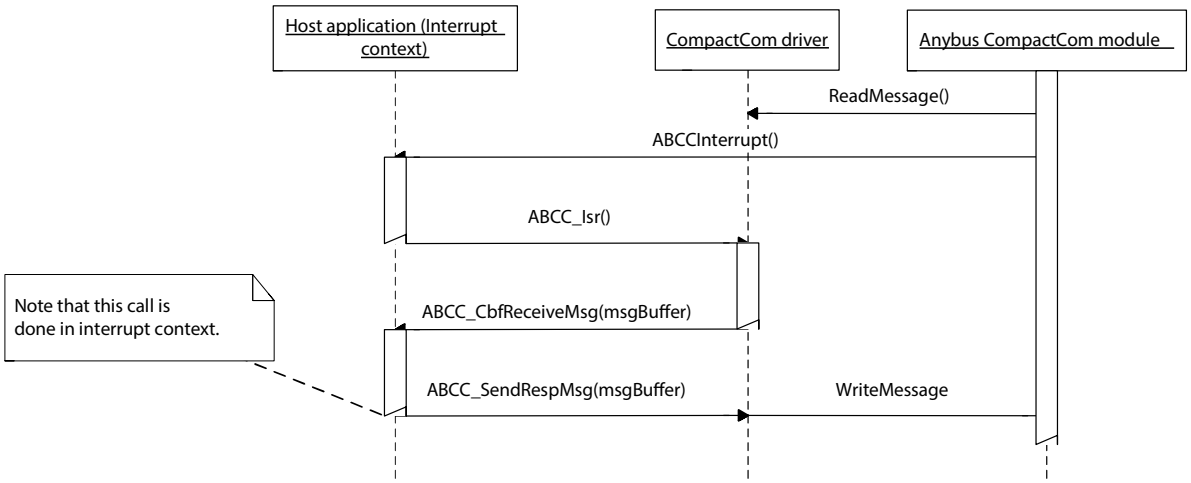
```
volatile BOOL ABCC_fRdMsgEvent = FALSE;

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。

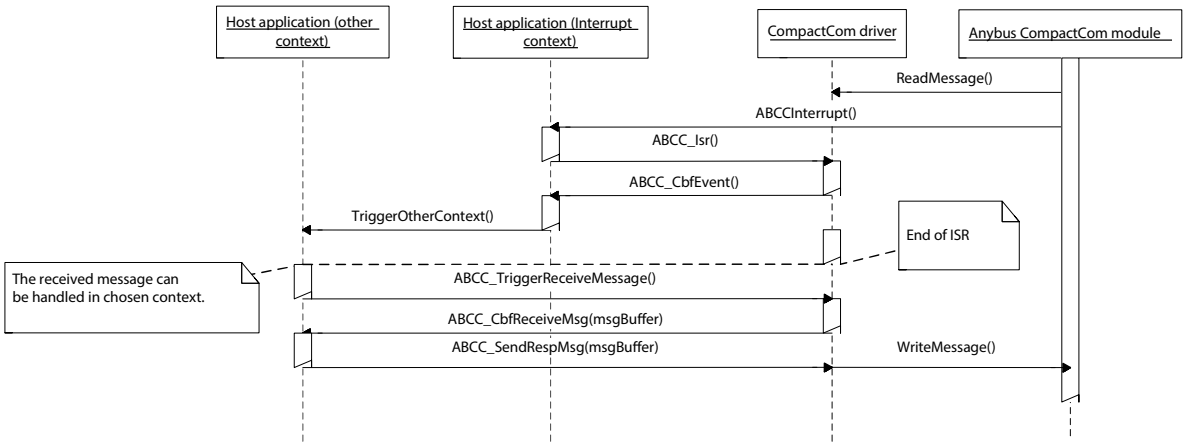
```
#define   ABCC_CFG_INT_ENABLED                ( TRUE )
#define   ABCC_CFG_INT_ENABLE_MASK_PAR        ( ABP_INTMASK_RDMSGIEN )
#define   ABCC_CFG_HANDLE_INT_IN_ISR_MASK     ( ABP_INTMASK_RDMSGIEN )
```



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

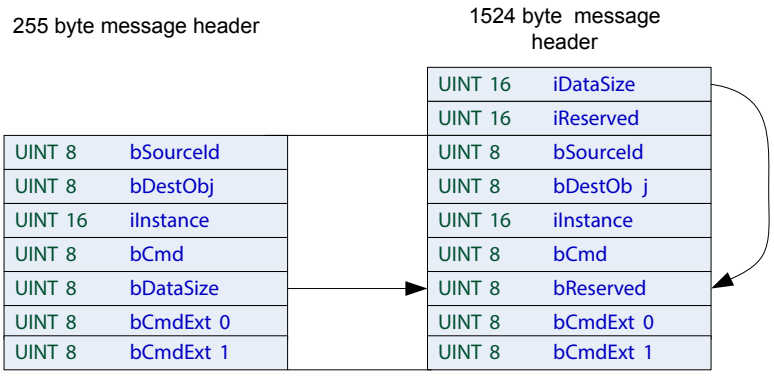
メッセージ処理のインターフェース関数は、 `abcc.h`に記述されています。

コマンドメッセージを送信するには、ユーザーは関数 `ABCC_GetCmdMsgBuffer()` を使用してメッセージメモリバッファを取得しなくてはなりません。応答を受信する際、ユーザーは、応答ハンドラー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。

関数 `ABCC_GetCmdMsgBuffer()` は、これ以上メモリバッファを利用できない場合、NULLポインターを返します。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かはユーザーが判断する必要があります。

注：バッファのリソースはファイル `abcc_adapt/abcc_drv_cfg.h`で設定されています。

注：Anybus 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シリーズとはフォーマットが異なります。ドライバーは、30シリーズデバイスとの通信だけでなく、40シリーズデバイス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フォーマットのみサポートします。30シリーズデバイスを使用する場合、ドライバー内部で従来のメッセージフォーマットに変換されます。以下の図に、2つのメッセージ形式を示します。

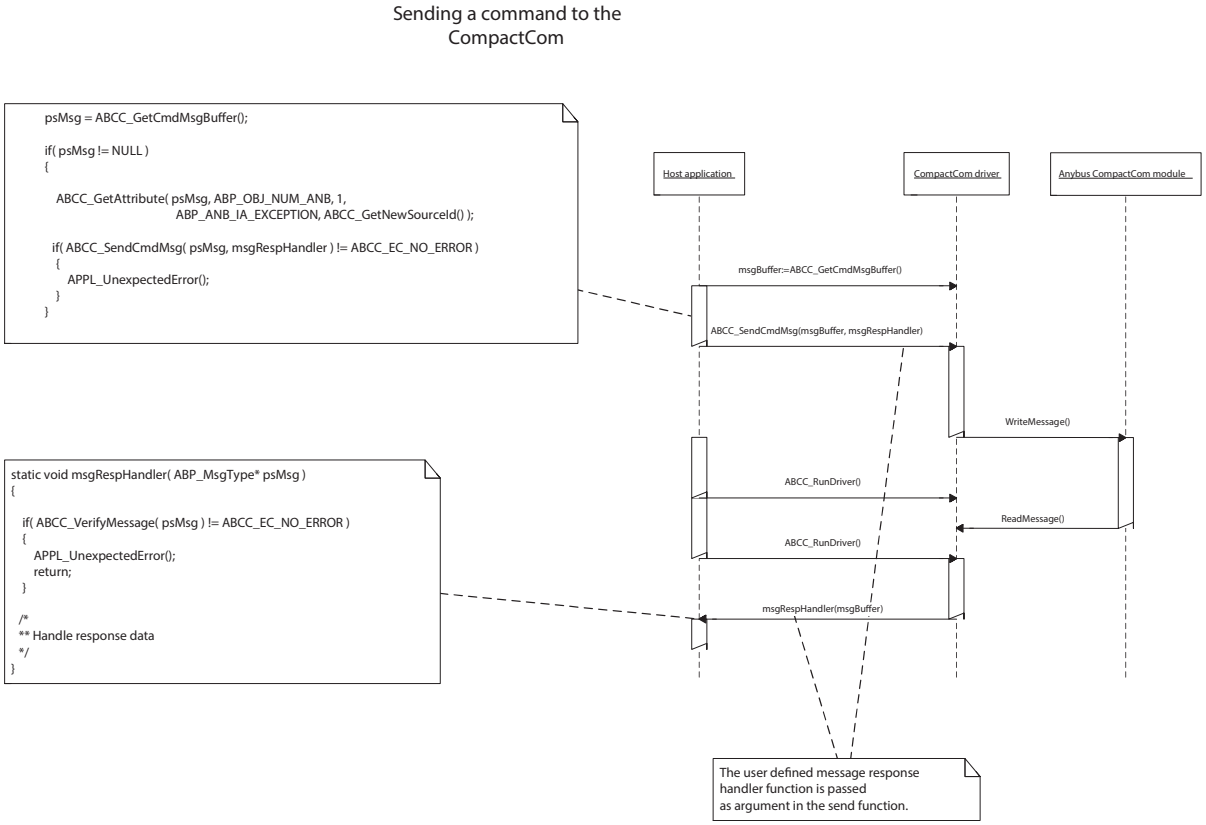


例1: コマンドの送信と応答の受信

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。

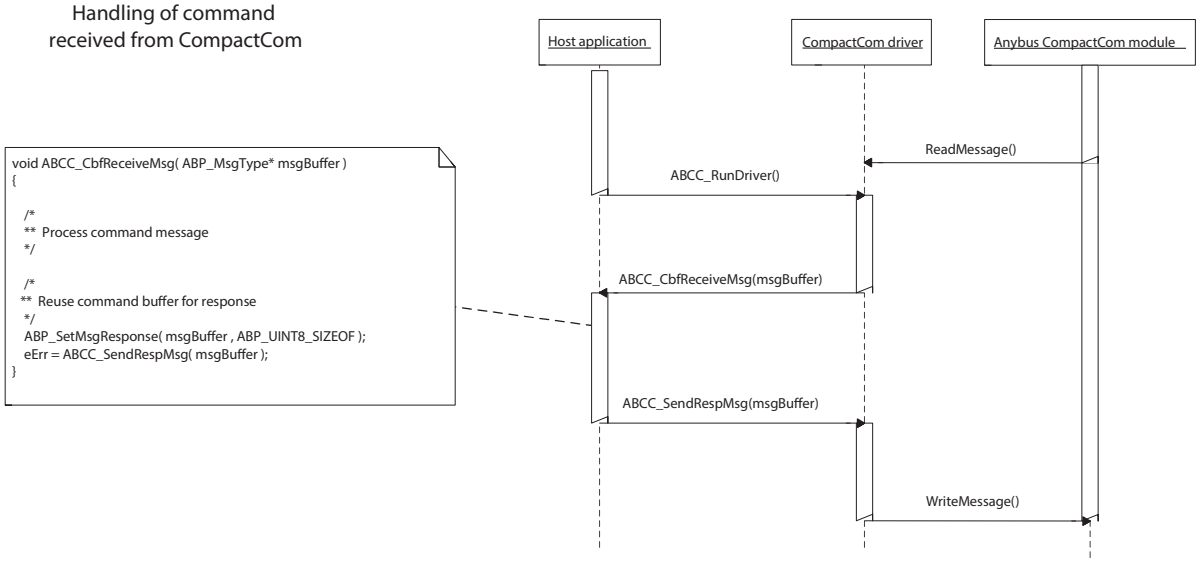
関数 `appl_HandleResp()` は、ソースIDが一致する応答を受信すると、ドライバーにコールされます。

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。



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。



ドライバーは、ノンブロッキング型のAnybus CompactComメッセージ処理に対応しています。これは、ステートマシンを使用してコマンドと応答を監視する必要があることを意味します。

コマンドシーケンサー

Anybus CompactComにメッセージおよびコマンドを送信するもう一つの方法は、コマンドシーケンサーを使用することです。ドライバーがコマンド用バッファの割り当て、リソースの制御、メッセージのシーケンス化を提供します。ユーザーはメッセージを構築し応答を処理する関数を提供しなくてはなりません。

コマンドシーケンサーAPIは `abcc_drv\inc\abcc_cmd_seq.h`に記述されています。

ABCC_CmdSeqTypeの配列が提供され、実行されるコマンドシーケンスを定義します。配列の最終エントリはNULLポインターで示されます。前のコマンドが応答の受信に成功すると、シーケンスの次コマンドが実行されます。

コマンドシーケンス応答ハンドラーが存在する場合、応答はアプリケーションに渡されます。

```
static const ABCC_CmdSeqType appl_asUserInitCmdSeq[] =
{
   ABCC_CMD_SEQ( UpdateIpAddress, NULL ),    /* pnCmdHandler, pnRespHandler */
   ABCC_CMD_SEQ( UpdateNodeAddress, NULL ),
   ABCC_CMD_SEQ( UpdateBaudRate, NULL ),
   ABCC_CMD_SEQ_END()                        /* End of sequence */
};
ABCC_AddCmdSeq( appl_asUserInitCmdSeq, UserInitDone );
```

コマンドシーケンス応答ハンドラーがNULLの場合、アプリケーションには通知されません。エラービットが設定されている場合、アプリケーションへの通知は `ABCC_CbfDriverError()` コールバックにより実行されます。

pnCmdSeqDoneコールバック関数が存在する場合（上記の例のUserInitDone）、コマンドシーケンス全体が終了するとアプリケーションに通知されます。同時に実行するコマンドシーケンスの数は `ABCC_CFG_MAX_NUM_CMD_SEQ` に制限され、これは `abcc_drv_cfg.h`で定義されています。

次の `example_app/appl_abcc_handler.c` には、2つのコマンドシーケンサー使用例があります。

例 1： `ABCC_CbfUserInitReq()` がコールされると、IPアドレスまたはノードアドレスが、`ABCC_UserInitComplete()` がコールされる前に設定されます。

例 2：Anybus CompactComが例外状態を示すと、例外コードを読み出します。

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。Anybus CompactComの新しいファームウェアがリリースされた場合に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なります。
$(ROOT)/abcc_drv/inc	アプリケーションに公開する.hファイルです。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なります。
$(ROOT)/abcc_drv/src	Anybus CompactComドライバーソースコードです。このフォルダに含まれるファイルは読み取り専用となります。
$(ROOT)/abcc_adapt	このフォルダにはドライバーとオブジェクトの適合用ファイルおよび設定ファイルが含まれます。ドライバーやサンプルコードを設定および適合させるには、ユーザーがファイルを変更しなければなりません。
$(ROOT)/abcc_obj	このフォルダにはホストアプリケーションオブジェクトのソースコードが格納されています。これらのファイルはユーザーにより変更することが可能です。
$(ROOT)/example_app	サンプルアプリケーションのソースコードが格納されています。これらのファイルはユーザーにより変更することが可能です。

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ソースコードファイル
$(ROOT)/abcc_versions.h	サンプルコード、ドライバー、ABP（Anybusプロトコル）のバージョン定義が格納されています。

A.3 Anybus 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ドライバーでサポートされているインターフェース
/abcc_drv/inc/abcc_ad_if.h	ADIマッピング用の型定義
/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、ターゲット製品依存インターフェース関数の定義
/abcc_drv/inc/abcc_sys_adapt_spi.h	abcc_spi_drv.cで必要とされる、ターゲット製品依存インターフェース関数の定義
/abcc_drv/inc/abcc_sys_adapt_par.h	abcc_par_drv.cで必要とされる、ターゲット製品依存インターフェース関数の定義
/abcc_drv/inc/abcc_sys_adapt_ser.h	abcc_ser_drv.cで必要とされる、ターゲット製品依存インターフェース関数の定義
/abcc_drv/inc/abcc_cmd_seq_if.h	コマンドシーケンサー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 フォルダ内のファイルの内容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、下位レベルのドライバーインターフェース
/abcc_drv/src/abcc_debug_err.h /abcc_drv/src/abcc_debug_err.c	デバッグおよびエラー出力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処理およびメッセージキュー処理
/abcc_drv/src/abcc_mem.c /abcc_drv/src/abcc_mem.h	abcc_link.cにより使用される、メッセージ用メモリリソースの操作処理
/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	動作中にプロセスデータをremap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実装
abcc_drv\src\abcc_cmd_seq.c abcc_drv\src\abcc_cmd_seq.h	メッセージコマンドシーケンサーの実装

A.4.1 8/16ビットパラレル・イベント処理に固有のファイル

ファイル名	説明
/abcc_drv/src/par/abcc_handler_par.c	ABCC_RunDriver()とABCC_ISR()の実装
/abcc_drv/src/par/abcc_par_drv.c	パラレル動作モードのドライバー実装
/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定義

A.4.2 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()の実装
/abcc_drv/src/spi/abcc_spi_drv.c	SPI動作モードのドライバー実装
/abcc_drv/src/spi/abcc_drv_spi_if.h	SPIドライバーのインターフェース定義
/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

A.4.3 8ビットパラレル・半二重に固有のファイル

ファイル名	説明
/abcc_drv/src/par30/abcc_handler_par30.c	ABCC_RunDriver()とABCC_ISR()の実装
/abcc_drv/src/par30/abcc_par30_drv.c	パラレル30半二重動作モードのドライバー実装
/abcc_drv/src/par30/abcc_drv_par30_if.h	パラレル30半二重ドライバーのインターフェース定義

A.4.4 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()の実装
/abcc_drv/src/serial/abcc_serial_drv.c	シリアル動作モードのドライバー実装
/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定義
/abcc_drv/src/serial/abcc_crc16.c /abcc_drv/src/serial/abcc_crc16.h	シリアルが使用するCRC16の実装

A.5 システム適合用ファイル

ファイル名	説明
/abcc_adapt/abcc_drv_cfg.h	Anybus CompactComドライバーのユーザー設定を行います。設定パラメーターの説明は、ドライバーインターフェースの abcc_cfg.hにあります。
/abcc_adapt/abcc_identification.h	Anybus CompactComモジュールの識別情報のユーザー設定を行います。
/abcc_adapt/abcc_obj_cfg.h	Anybusオブジェクト実装用のユーザー設定を行います。
/abcc_adapt/abcc_sw_port.c	Anybus 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のマクロと関数です。
/abcc_adapt/abcc_sw_port.h	Anybus 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のマクロと関数です。マクロの説明はabcc_port.hに記述されています。abcc_port.hは、Anybus CompactComの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	Anybus CompactComのデータタイプなどの定義
/abcc_adapt/abcc_platform_cfg.h	abcc_adapt/abcc_obj_cfg.h、 abcc_adapt/abcc_drv_cfg.hと abcc_adapt/abcc_identification.hの定義を上書きするプラットフォーム固有の定義です。

B API

B.1 APIについて

Anybus CompactComのAPIレイヤーでは、各ネットワークに対応した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しています。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。 この関数をコールすると、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()実行後、TRUEが返ってくるまでこの関数を繰り返し実行する必要があります。TRUEが返ってきた場合、モジュールとの通信準備が完了し、Anybus CompactComのセットアップシーケンスが開始されることを意味します。
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行います。 この関数からABCC_ShutdownDriver()がコールされます。 注！この関数は、リセットピンをLowにセットする動作のみ行います。十分長いリセット時間（ABCC_HWReset()のコールからABCC_ HWReleaseReset()のコールまでの時間）を確保することは、この関数の呼び出し側の責任になります。
ABCC_HWReleaseReset()	モジュールのハードウェアリセットを解除します。
ABCC_RunTimerSystem()	Anybus 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コール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()	Anybus CompactComドライバーの送受信メカニズムを実行します。ポーリング時、このメインルーチンを周期的にコールする必要があります。
ABCC_UserInitComplete()	ユーザー固有のセットアップにおける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コールする必要があります。これにより、Anybus 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内の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出します。この関数は、パラレル動作モードでのみサポートされます。
ABCC_LedStatus()	LEDの状態を読み出します。SPIおよび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の状態を読み出します。

イベント関連API関数

関数	説明
ABCC_ISR()	Anybus CompactComアプリケーションインターフェースのIRQピンによるトリガーイベントを受付け処理するには、Anybus CompactComの割り込みルーチン内からこの関数をコールする必要があります。
ABCC_TriggerRdPdUpdate()	RdPdの読み出しを開始します。
ABCC_TriggerReceiveMessage()	受信メッセージの読み出しを開始します。
ABCC_TriggerWrPdUpdate()	アプリケーションにて新たなプロセスデータが利用可能になり、Anybus CompactComに送信されることを表します。

イベント関連**API**関数 **(**続き**)**

関数	説明
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。

コールバック**API**

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コールされます。 この関数は、リード / ライトプロセスデータ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SETUP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コールされます。
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()	新しいプロセスデータを受信したときにコールされます。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すべてを受信する関数です。
ABCC_CbfWdTimeout()	この関数は、モジュールとの通信が失われたときにコールされます。
ABCC_CbfWdTimeoutRecovered()	Anybus CompactComのウォッチドッグ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コールされます。
ABCC_CbfAnbStateChanged()	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ステートまたは監視状態が変化したとき) にコールされます。
ABCC_CbfEvent()	処理されないイベントに対してコールされます。 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()	Syncがサポートされている場合、Syncイベント発生時にこの関数がコールされます。

! 上記のコールバックAPIは、すべてアプリケーションで実装する必要があります。

サポート関数

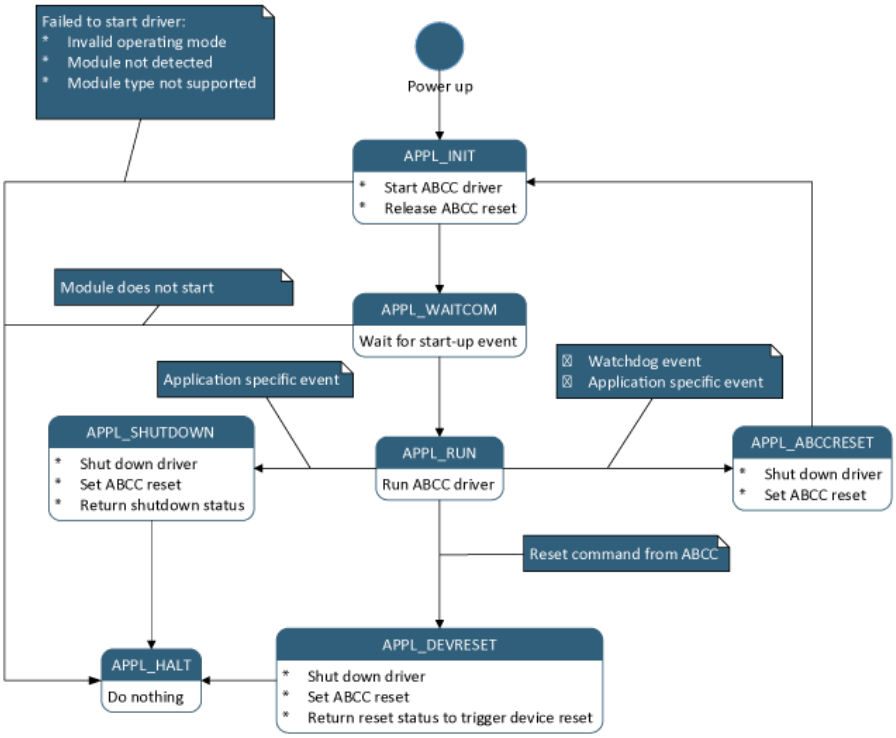
関数	説明
ABCC_NetworkType()	ネットワークタイプを取得します。
ABCC_ModuleType()	モジュールタイプを取得します。
ABCC_DataFormatType()	ネットワークのデータフォーマットを取得します。
ABCC_ParameterSupport()	パラメーターデータのサポート情報を取得します。
ABCC_GetOpmode()	ABCC_SYS_GetOpmode()をコールして、ハードウェアから動作モードを読み出します。
ABCC_GetAttribute()	アトリビュート取得のためのパラメーターをAnybus CompactComメッセージに設定します。
ABCC_SetByteAttribute()	アトリビュート設定のためのパラメーターをAnybus CompactComメッセージに設定します。
ABCC_VerifyMessage()	Anybus CompactCom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。

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の動作は、以下のフローチャートに示すステートマシンで管理されています。

example_app/appl_abcc_handler.hにある APPL_HandleAbcc() は、メインループから周期的にcallされます。この関数内でステートマシンが実装され、各状態でさまざまなタスクの実行を行います。

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



APPL_INIT

- Anybus CompactComが検出されたかをチェックします。
- 指定された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、サンプルアプリケーション, ページ 27で説明した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()をコールします。
- Anybus CompactComのリセットを解除する為に、ABCC_HwReleaseReset()をコールします。
- APPL_WAITCOM状態に遷移します。

APPL_WAITCOM

- Anybus CompactCom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状態に遷移します。

APPL_RUN

- ドライバーを実行する為に、ABCC_RunDriver()をコールします。また、特定のイベントに対してコールバックがコールされます。ドライバーが使用するコールバックにはすべて、ABCC_Cbf*()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、example_app/appl_abcc_handler.cに実装されています。
- 起動中、以下の処理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_CbfAnbStateChanged()がコールされます。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()がコールされます。Genericのサンプルコードでは、デフォルトマッピングに使用する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、Anybus CompactComに情報を設定する、または、Anybus CompactCom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()がコールされます。サンプルコードでは、この関数によってユーザーによる初期化シーケンスが開始され、Anybus CompactComに初期化用の一連のコマンドを送信します。最後のコマンドに対する応答を受信すると、関数ABCC_UserInitComplete()がコールされ、ユーザーによる初期化シーケンスが終了したことをドライバーに通知します。これにより、ドライバーがSETUP_COMPLETEコマンドをAnybus CompactComに送信するよう動作します。ユーザーによる初期化が必要ない場合、ABCC_CbfUserInitReq()からABCC_UserInitComplete()を直接コールすることが可能です。

– セットアップが完了すると、Anybus CompactComはABP_ANB_STATE_NW_INIT状態に遷移します。これは、ABCC_CbfStateChanged()がコールされることを意味します。この状態では、Anybus CompactCom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、どのホストアプリケーション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依存します。必要であれば、どのようなコマンドが受信され、どのように処理されたかを表示するように、ABCC_CbfReceiveMsg()でブレークポイントを設定してください。

– ネットワークの初期化が完了すると、Anybus CompactComはABP_ANB_STATE_WAIT_PROCESS状態に遷移します。ドライバーによってABCC_CbfStateChanged()が再度コールされます。この時点で、ネットワークからIO通信での接続が可能になります。

– IO通信が開始されると、Anybus CompactComはABP_ANB_STATE_PROCESS_ACTIVE状態に遷移します（あるいは、ネットワークによってはABP_ANB_STATE_IDLE状態に遷移します）。Anybus CompactComからホストアプリケーションにプロセスデータが送信されると、関数ABCC_CbfNewReadPd()がコールされます。サンプルコードではAD_UpdatePdReadData()がコールされ、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、該当する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、関数の終了時、ABCC_TriggerWrPdUpdate()がコールされ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コールされます。新しいライトプロセスデータが利用可能になったとき、ABCC_TriggerWrPdUpdate()を必ずコールする必要があります。
注：IO通信でデータのループを行うのは、サンプルADIマッピングとしてappl_adimap_simple16.cを使用した場合になります。

– 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、Anybus CompactComから例外の原因を読み出すことが可能です。RunExceptionSM()は、Anybus CompactComがABP_ANB_STATE_EXCEPTION状態にあるときに、APPL_RUN状態からコールされます。

– Anybus CompactComを再起動する為に、APPL_Reset()がコールされます。この処理は、アプリケーションオブジェクトがAnybus CompactComからリセット要求を受信したときに行われます。すると、ホストアプリケーションのステートマシンがAPPL_ABCCRESET状態になり、APPL_INITから実行し直します。

– 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。この関数がコールされると、ホストアプリケーション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用されていません。電源off/onを避けるために、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()がコールされます。

APPL_SHUTDOWN

- Anybus CompactComのリセットの為に、ABCC_HWReset()がコールされます。
- APPL_HALT状態に遷移します。

APPL_ABCCRESET

- Anybus CompactComのリセットの為に、ABCC_HWReset()がコールされます。
- APPL_INIT状態に遷移します。

APPL_DEVRESET

- Anybus CompactComのリセットの為に、ABCC_HWReset()がコールされます。
- APPL_HALT状態に遷移します。

（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。

Box 4126
300 04 Halmstad, Sweden

info@hms.se
HMSI-27-334-JA 1.2.7272 / 2018-02-02