Driver User Manual

Anybus® CompactCom 40

Doc.Id. HMSI-27-273
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
www.hms-networks.com

必ずお読みください

本ドキュメントは、Anybus CompactCom 通信モジュールを用いた開発を容易にするためのドライバーソフトウェアを十分理解していただくためのものです。本ドキュメントでは、パッシブモジュールについては扱っていません。また、ネットワーク特有の機能についても扱っていません。それらの情報は別途ドキュメント（Network Guides）として用意されています。

本ドキュメントの読者は、上位 / 下位レベルのソフトウェア設計や通信システム全般に関して高い知識を備えていることが求められます。

本ドキュメントは、オンラインで使うのに最適なように作られています。可能な場合には、ファンクションコールなどがクリック可能なハイパーリンクで示されています。

- Anybus CompactCom のコンセプトがどのように機能するかについて理解するのは開発者の責任であり、ドライバーがその役目を負うものではありません（すなわち、ユーザーは『Anybus CompactCom 40 Software Design Guide』を読んでその内容を*理解*することが求められます）。
- 本ドライバーを使用する前に、ハードウェアの実装が期待どおりに機能することを確認してください。
- ユーザーは、C プログラミング言語に関する知識を備えていることが求められます。
- ユーザーは、本プロジェクトで使用される開発環境に関する知識を備えていることが求められます。

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.com から入手いただけます。

責任の範囲

本マニュアルは細心の注意を払って作成されています。誤字や脱字があった場合は、HMS Industrial Networks AB にお知らせください。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。

本ソフトウェアの使用者は、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。

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ソフトウェア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著作権、特許や出願中の特許が含まれます。

商標

Anybus ® 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。

ソースコードのライセンス

Anybus CompactCom 40 ドライバーソースコードは、以下の著作権宣言に従います。

本コードは、HMS Industrial Networks AB の所有物です。事前の許可なしに本ソースコードを複製、配布、使用することを禁じます。本コードを HMS の製品とともに使用する場合は、何らの制限なく、バイナリ形式で変更、複製、配布できます。

本コードは " 現状のまま " 提供され、いかなる保証も行いません。HMS は、本コードの機能が利用者の要求を満たすことを保証しません。また、本コードの動作に中断やエラーがないことや、本コードの不具合が修正されることも保証しません。

目次

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

ドキュメント	作成者
Anybus CompactCom M40 Hardware Design Guide	HMS
Anybus CompactCom 40 Software Design Guide	HMS
Anybus CompactCom 40 Drive Profile Design Guide	HMS
Anybus CompactCom 40 Network Guides（各ネットワークシステムに関する別途ドキュメント）	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
サポート情報を更新	6
ドライバーのフォルダー構造を改訂	11
Startdriver コマンドにパラメータを追加	30

リビジョンリスト

リビジョン	日付	作成者	章	説明
0.10	2013/12/19	KeL, KaD	すべて	新規ドキュメント
0.50	2014/02/06	KaD	すべて	改訂ドキュメント
1.00	2014/03/25	KaD	すべて	改訂ドキュメント
1.10	2014/06/05	KaD	P, 4, B	大幅に更新

P.3 表記と用語

本ドキュメント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 " モジュール " は、Anybus CompactCom モジュールを表します。
- " ホスト "、" ホストシステム "、または " ホストアプリケーション " は、Anybus CompactCom モジュールをホストするハードウェアとソフトウェアを表します。
- 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。
- 特に指定がない限り、Intel 形式のバイトオーダーを想定します。
- ホストシステムのメモリ空間 = ホストソフトウェアにより直接アクセス可能なメモリ。
- パラレルインターフェースのメモリ空間 = パラレルインターフェースが存在するメモリ。ホストシステムのメモリ空間で直接利用可能とは限りません。

P.3.1 略語

略語	説明
ABCC	Anybus CompactCom
ABP	Anybus Protocol
ISR	Interrupt Service Routine
WRPD	Write process data
RDPD	Read process data
WRMSG	Write message
RDMSG	Read message

P.4 サポート

お問い合わせと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。

1. Anybus CompactCom 40 Driver について

1.1 概要

Anybus CompactCom 40 ネットワーク通信モジュールは、要求が厳しい産業用途向けの高性能な通信ソリューションです。このモジュールは、サーボ駆動システムなどの高性能かつ同期が必要な用途に適しています。代表的な用途は、PLC、HMI、ロボット、AC/DC ドライブ、サーボドライブです。

Anybus CompactCom 40 のソフトウェアインターフェースは、ネットワークプロトコルとは独立して設計されているため、ホストアプリケーションは、機能を損なうことなく、同じソフトウェアドライバーを使用してあらゆる主要ネットワークシステムに対応することができます。

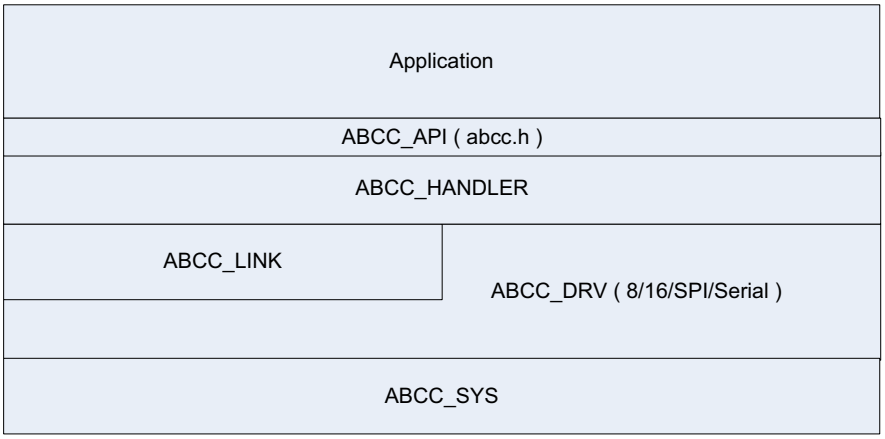
柔軟性と拡張性を提供するため、ホストアプリケーションと Anybus モジュールとの間はオブジェクト指向のアドレス指定方式が使用されています。この方式では、メモリマップドデータではなくオブジェクトに対して明示的にリクエストするため、Anybus モジュールはホストアプリケーションの通信プロトコルを使用してホストアプリケーションから直接情報を取得できるようになり、非常に高いレベルの統合化が可能となります。

HMS は、開発プロセスのスピードの向上に自由に活用できる、無償のソースレベル（C 言語）のソフトウェアドライバーを提供しています。このドライバーは、Anybus モジュールとホストアプリケーションとの " 接着剤 " の役目を果たします。すなわち、下位レベルのホストインターフェース通信がホストアプリケーションのソフトウェアから分離されるため、Anybus に関連する共通のタスクのための使いやすいファンクションコールを提供します。このドライバーは、1 つのアプリケーションにつき 1 つの物理インターフェースをサポートします。

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。またこのドライバーは、CompactCom 40 モジュールに加えて CompactCom 30 モジュールでも使用できます。本マニュアルでは、Anybus CompactCom 40 Driver について説明します。また、ドライバーから起動されるアプリケーションの実装方法について、ステップバイステップで説明します。

24 ページ「ステップバイステップのユーザーガイド」では、ドライバーの使い方に関するガイドが示されています。必要な参照ドキュメントは、前の章にすべて記載されています。

2. ABCC ドライバースタック



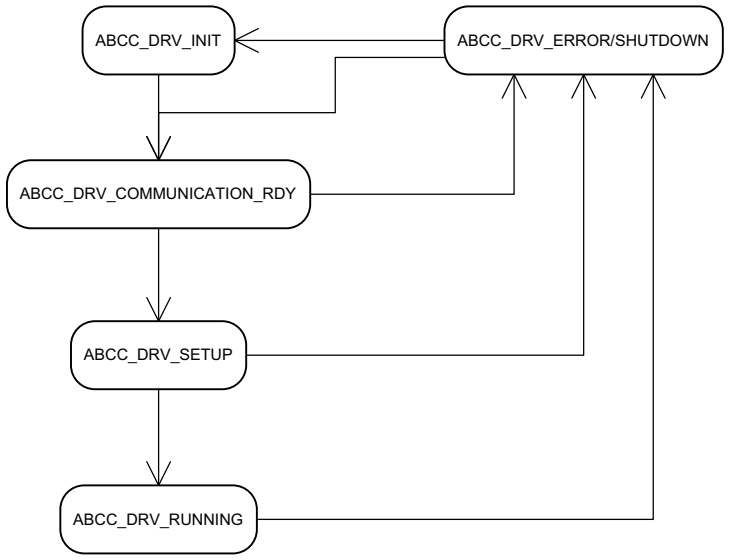
ドライバー部品の説明

- ABCC_API - Anybus CompactCom ドライバーを使用するアプリケーションへの共通インターフェースを提供します。インターフェースの説明は abcc.h を参照してください。
- ABCC ハンドラーレイヤー— イベントまたはポーリングにより駆動される Anybus CompactCom のメインのステートマシンを処理します。このレイヤーは、動作モードに依存する部分と依存しない部分に分かれています。
- ABCC_LINK レイヤー — メッセージフローコントロールとメッセージバッファハンドリングを行う内部レイヤーです。インターフェースの説明は abcc_link.h を参照してください。
- ABCC_DRV - 動作モード固有のドライバーです。インターフェースの説明は abcc_drv_if.h を参照してください。
- ABCC_SYS - 動作モードとターゲットに固有の物理インターフェースです。ABCC_SYS には、ドライバーを動作させるのに必要な、ハードウェアに依存するコードが含まれています。インターフェースの説明は、abcc_sys.h、abcc_sys_par.h、abcc_sys_spi.h、abcc_sys_ser.h、および abcc_sys_par30.h を参照してください。

3. Anybus CompactCom ハンドラーのメインステートマシン

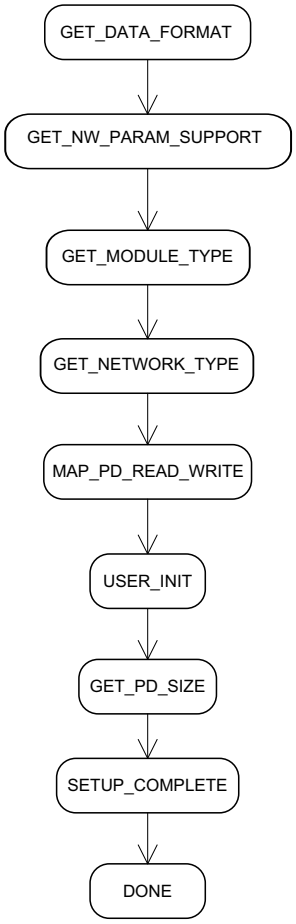
本章では、Anybus CompactCom ハンドラーの主な状態について説明します。このステートマシンは、アプリケーションのメインループから ABCC_RunDriver() 関数を呼ぶことで駆動されます。または、Anybus CompactCom モジュールによりイベントがトリガーされたときに ABCC_ISR() 関数を呼ぶことで駆動されます。

以下の図に、ハンドラーの主な状態を制御するステートマシンを示します。



状態	説明
ABCC_DRV_INIT	ABCC_StartDriver() 関数が呼ばれると、以下の手順が実行される。 - すべての内部パラメータと変数がリセットされる - モジュール検出が行われる - モジュール ID が読み取られる - 動作モードがセットされる - 割り込みが有効になる 完了すると、ハンドラーは ABCC_DRV_COMMUNICATION_RDY 状態になる。
ABCC_DRV_COMMUNICATION_RDY	電源オン割り込みにより Anybus モジュールが通信可能であることが示された場合、または、ABCC_USER_STARTUP_TIME_MS タイムアウト時間が経過した場合、ABCC_isReadyForCommunication() コマンドにより ABCC_DRV_SETUP に移行する。
ABCC_DRV_SETUP	Anybus CompactCom をセットアップする。このセットアップは、ABCC_RunDriver() または ABCC_ISR() により駆動される。セットアップが完了すると、RUNNING 状態に移行する。 Anybus CompactCom のセットアップに関するサブステートマシンについては、10 ページ「Anybus のセットアップステートマシン」を参照。
ABCC_DRV_RUNNING	Anybus CompactCom が稼働中。送信されてくるイベント（ABCC_RunDriver() または ABCC_ISR()）は、処理されるかアプリケーションに送信される。
ABCC_DRV_ERROR	ハンドラーを強制的にこの状態に移行させるエラーが発生した。 ABCC_HwReset() 関数を呼び出すと、ハンドラーが ABCC_DRV_SHUTDOWN 状態に移行する。 他のイベントはすべて無視される。
ABCC_DRV_SHUTDOWN	ABCC_StartDriver() 関数を呼び出すと、ドライバーが再起動される。
すべての状態	エラーが検出されると、そのエラーが通知され、ハンドラーが ABCC_DRV_ERROR 状態に移行する。 ABCC_Shutdown() 関数を呼び出すと、ハンドラーが ABCC_DRV_SHUTDOWN 状態に移行する。

Anybus のセットアップステートマシン



状態	説明
GET_DATA_FORMAT	データ形式を読むためのメッセージが送信される。応答が正しく返されると、GET_NW_PARAM_SUPPORT 状態に移行する。
GET_NW_PARAM_SUPPORT	ネットワークパラメータのサポート状態を読むためのメッセージが送信される。応答が正しく返されると、GET_MODULE_TYPE 状態に移行する。
GET_MODULE_TYPE	モジュールタイプを読むためのメッセージが送信される。応答が正しく返されると、GET_NETWORK_TYPE 状態に移行する。
GET_NETWORK_TYPE	ネットワークタイプを読むためのメッセージが送信される。応答が正しく返されると、MAP_PD_READ_WRITE 状態に移行する。
MAP_PD_READ_WRITE	ABCC_CbfAdiMappingReq() 関数を呼び出すと、ADI マッピング情報が取得される。すべてのリード / ライトマッピングが実行されると、USER_INIT 状態に移行する。
USER_INIT	この状態では、ユーザー固有のセットアップが行われる。このセットアップは、Anybus CompactCom ドライバーが ABCC_CbfUserInitReq() を呼び出すことでトリガーされる。アプリケーションにより ABCC_CbfUserInitComplete() が呼ばれると、GET_PD_SIZE 状態に移行する。
GET_PD_SIZE	この状態では、Anybus オブジェクトからプロセスデータサイズが読み出される。完了すると、SETUP_COMPLETE 状態に移行する。
SETUP_COMPLETE	セットアップ完了を確定するためのメッセージが送信される。応答が正しく返されると、メイン状態である RUNNING 状態に移行する。
すべての状態	エラーまたはメッセージの間違いが検出されると、該当するエラーが通知され、メイン状態である ERROR 状態に移行する。エラーは ABCC_CbfDriverError() により通知される。

4. ソフトウェア概要

4.1 概要

Anybus CompactCom 40 Driver は、Anybus CompactCom 40 モジュールにアクセスするためのソフトウェアコンポーネントで、ユーザーの製品やアプリケーションに完全に組み込むことができます。

ホスト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（ポーティング）する必要があります。これには、通常、Anybus ホストインターフェースにアクセスする関数や、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関数が含まれます。これらの関数は、sys_example フォルダーのファイルにすべて含まれています。

4.1.1 ファイルとフォルダー

フォルダー	説明
$(ROOT)\abcc_app_drv\inc	アプリケーションに公開されている .h ファイル。このフォルダーには、アプリケーションのためのドライバーコンフィグレーションファイルや、ドライバーのシステム依存部分のためのドライバーコンフィグレーションファイルが格納されている。
$(ROOT)\abcc_app_drv\src	Anybus CompactCom ドライバーの実装。
$(ROOT)\abcc_app_drv\src\spi	SPI に固有のファイル。
$(ROOT)\abcc_app_drv\src\par	8/16 ビットイベントに固有のファイル。
$(ROOT)\abcc_app_drv\src\par30	8/16 ビットピンポンに固有のファイル。
$(ROOT)\abcc_app_drv\src\serial	シリアルピンポンに固有のファイル。
$(ROOT)\sys_example\sys_template	ドライバーのシステム依存部分に関する実装例（SDK および空のサンプル）、およびドライバーコンフィグレーションファイル。
$(ROOT)\abp\inc	ABP のヘッダーファイル。

4.1.2 ドライバーの構築

ドライバーを構築するには以下のファイルが必要です。

公開されている ABCC ドライバーインターフェース

ファイル名	説明
\abcc_app_drv\inc\abcc.h	Anybus CompactCom Driver のパブリックインターフェース。
\abcc_app_drv\inc\abcc_ad_if.h	ADI マッピングの型定義。
\abcc_app_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_app_drv\inc\abcc_sys_adapt_spi.h	abcc_spi_drv.c で必要なターゲット依存関数のインターフェース。
\abcc_app_drv\inc\abcc_sys_adapt_par.h	abcc_par_drv.c で必要なターゲット依存関数のインターフェース。
\abcc_app_drv\inc\abcc_sys_adapt_ser.h	abcc_ser_drv.c で必要なターゲット依存関数のインターフェース。

ドライバーの内部ファイル

ファイル名	説明
\abcc_app_drv\src\abcc_drv_if.h	特定の動作モードを実現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_app_drv\src\abcc_debug_err.h	デバッグとエラーレポートのためのヘルプマクロ。

ファイル名	説明
\abcc_app_drv\src\abcc_link.c \abcc_app_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_app_drv\src\abcc_mem.c \abcc_app_drv\src\abcc_mem.h	abcc_link.c で使用されるメッセージリソースメモリのサポート。
\abcc_app_drv\src\abcc_handler.c	動作モードに依存しない部分の Anybus CompactCom ハンドラーの実装。
\abcc_app_drv\src\abcc_setup.c	セットアップステートマシンを含む Anybus CompactCom ハンドラーの実装。
\abcc_app_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_app_drv\src\par\abcc_handler_par.c	ABCC_RunDriver() および ABCC_ISR() を実装する。
\abcc_app_drv\src\par\abcc_par_drv.c	パラレル動作モード用の abcc_drv_if.h を実装する。

SPI に固有のファイル

ファイル名	説明
\abcc_app_drv\src\par\abcc_handler_spi.c	ABCC_RunDriver() および ABCC_ISR() を実装する。
\abcc_app_drv\src\spi\abcc_spi_drv.c	SPI 動作モード用の abcc_drv_if.h を実装する。
\abcc_app_drv\src\spi\abcc_crc32.c \abcc_app_drv\src\spi\abcc_crc32.h	SPI で使用される CRC32 の実装。

8 ビットピンポンに固有のファイル

ファイル名	説明
\abcc_app_drv\src\par\abcc_handler_par30.c	ABCC_RunDriver() および ABCC_ISR() を実装する。
\abcc_app_drv\src\par\abcc_par30_drv.c	パラレル 30 ピンポン動作モード用の abcc_drv_if.h を実装する。

シリアルに固有のファイル

ファイル名	説明
\abcc_app_drv\src\par\abcc_handler_ser.c	ABCC_RunDriver() および ABCC_ISR() を実装する。
\abcc_app_drv\src\serial\abcc_serial_drv.c	シリアル動作モード用の abcc_drv_if.h を実装する。
\abcc_app_drv\src\serial\abcc_crc16.c \abcc_app_drv\src\serial\abcc_crc16.h	シリアルで使用される CRC16 の実装。

システム適応インターフェースファイル

ファイル名	説明
\abcc_app_drv\sys_example\sys_template\abcc_td.h	Anybus CompactCom で使用される型。
\abcc_app_drv\sys_example\sys_template\abcc_user_def.h	ドライバーのコントロールとコンフィグレーションに関するユーザー定義。

システム適応実装ファイル

ファイル名	説明
\abcc_app_drv\sys_example\sys_template\abcc_sys_adapt.c	システム適応に関する空の実装サンプル。
\abcc_app_drv\sys_example\sys_template\abcc_cbf.c	コールバック関数のテンプレート。
\abcc_app_drv\sys_example\sys_pctransportprovider\abcc_sys_adapt.c	PC の transport provider に適応するための実装。

5. API

Anybus CompactCom 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 ドライバーにアクセスするための共通インターフェースを定義します。このインターフェースは abcc.h に格納されています。

API 関数

関数	説明
ABCC_StartDriver()	ドライバーを初期化し、割り込みを許可し、動作モードをセットする。 この関数を呼び出すと、タイマーシステムが起動可能になる。 注！この関数はモジュールのリセットを解除しない。
ABCC_ShutdownDriver()	ドライバーを停止して SHUTDOWN 状態にする。
ABCC_IsReadyforCommunication()	ABCC_StartDriver() を呼び出した後に、TRUE が返されるまでこの関数をポーリングする必要がある。TRUE が返された場合、モジュールが通信可能となり ABCC のセットアップシーケンスが開始されたことを表す。
ABCC_RunTimerSystem()	ABCC ドライバーの各タイマーを処理する。タイマー割り込みにてこの関数を定期的に呼び出すことを推奨する。この関数を使用しないと、タイムアウトとウォッチドッグの機能が動作しない。
ABCC_RunDriver()	ABCC ドライバーの送受信メカニズムを駆動する。 TRUE: ドライバーが起動されて通信可能 FALSE: ドライバーが停止された、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最後の応答が返されたときに、アプリケーションからこの関数を呼び出す必要がある。これにより、ABCC のセットアップシーケンスが終了し、ABCC_SETUP_COMPLETE が送信される。
ABCC_SendCmdMsg()	モジュールにコマンドメッセージを送信する。
ABCC_SendRespMsg()	モジュールに応答メッセージを送信する。
ABCC_SendRemapRespMsg()	モジュールに再マッピング応答を送信する。
ABCC_GetCmdQueueSize()	コマンドキューに残っているエントリの数を取得する。
ABCC_SetAppStatus()	abp.h の ABP_AppStatusType に従って、現在のアプリケーションの状態を設定する。
ABCC_MsgAlloc()	メッセージバッファを割り当てる。メッセージバッファを割り当てる際は、必ずこの関数を使用すること。
ABCC_MsgFree()	他のメッセージを送信できるように、メッセージバッファをドライバーに返す。
ABCC_HwReset()	モジュールのハードウェアをリセットする。 この関数から ABCC_ShutdownDriver() が呼び出される。 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う。十分長いリセット時間（ABCC_HwReset() の呼び出しから ABCC_HwReleaseReset() の呼び出しまでの時間）の確保は呼び出し側にて行うこと。
ABCC_HwReleaseReset()	モジュールのリセットを解除する。
ABCC_ReadModuleId()	モジュール ID を読み出す。 この関数は、ドライバーの起動とリセットの解除前に使用することが可能。
ABCC_ModuleDetect()	モジュールがあるかどうかを検出する。 この関数は、ドライバーの起動とリセットの解除前に使用することが可能。
ABCC_ModCap()	モジュールの機能を読み出す。この関数は、ABCC40 のパラレル動作モードでのみサポートされている。
ABCC_LedStatus()	LED ステータスを読み出す。SPI および ABCC40 のパラレル動作モードでのみサポートされている。
ABCC_AnbState()	Anybus の現在の状態を読み出す。

API イベントに関する関数

関数	説明
ABCC_ISR()	受信した ABCC イベント（ABCC アプリケーションインターフェースの IRQ_n によりトリガーされたイベント）を認識して処理するため、ABCC 割り込みルーチン内でこの関数を呼び出す必要がある。
ABCC_RdPdEvent()	RdPd 読み込みをトリガーする。
ABCC_RdMsgEvent()	メッセージ受信読み込みをトリガーする。
ABCC_NewWrPdEvent()	アプリケーションからの新規プロセスデータが存在し、それらが ABCC に送信されることを表す。

API コールバック

この関数はすべて、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()	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この関数が呼び出される。 この関数は、プロセスデータのリードライトのマッピング情報を返す。
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トリガーするために呼び出される。
ABCC_CbfUpdateWriteProcessData()	現在のライトプロセスデータを更新する。この関数から戻る前に、データをバッファにコピーしておく必要がある。
ABCC_CbfNewReadPd()	この関数は、新規プロセスデータを受信したときに呼ばれる。この関数から戻る前に、プロセスデータをアプリケーションの ADI にコピーしておく必要がある。
ABCC_CbfReceiveMsg()	モジュールからメッセージを受信した。これは、モジュールから受信したすべてのコマンドの受信関数である。
ABCC_CbfWdTimeout()	モジュールとの通信が切断されたときにこの関数が呼び出される。
ABCC_CbfWdTimeoutRecovered()	ABCC のウォッチドッグタイムアウトが発生したが、現在は通信が再開されていることを表す。
ABCC_CbfDriverError()	ドライバーがエラーを検出した。
ABCC_CbfRemapDone()	このコールバックは、REMAP 応答がモジュールに送信されたときに呼び出される。
ABCC_CbfAnbStatusChanged()	このコールバックは、モジュールが状態を変化させるとき、すなわち、Anybus の状態または監視の状態が変化したときに呼び出される。
ABCC_CbfEvent()	処理されないイベントに対して呼び出される。 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 で選択されていないイベントのことである。

サポート関数

関数	説明
ABCC_NetworkType()	ネットワークの種類を取得する。
ABCC_ModuleType()	モジュールの種類を取得する。
ABCC_NetFormatType()	ネットワークの形式を取得する。
ABCC_DataFormatType()	ネットワークのエンディアン形式を取得する。
ABCC_ParameterSupport()	パラメータのサポートを取得する。
ABCC_GetOpmode()	ABCC_SYS_GetOpmode() を呼び出してハードウェアから動作モードを読み出す。
ABCC_GetAttribute()	ABCC のメッセージにパラメータを設定してアトリビュートを取得する。
ABCC_SetByteAttribute()	ABCC のメッセージにパラメータを設定してアトリビュートを設定する。
ABCC_VerifyMessage()	ABCC の応答メッセージを照合する。
ABCC_GetDataTypeSize()	ABP データ型のサイズを返す。

6. コンフィグレーションパラメータ

abcc_user_def.h には、システムの設定で使用する定義が数多く用意されています。それらを以下の表に示します。

定義名	説明
ABCC_USER_DRV_SPI	SPI 動作モードを使用。
ABCC_USER_DRV_PARALLEL	パラレルイベントモード（8/16 ビット）を使用。
ABCC_USER_DRV_PARALLEL_30	パラレルピンポン（半二重）モードを使用。
ABCC_USER_DRV_SERIAL	シリアルピンポン（半二重）モードを使用。
ABCC_USER_STARTUP_TIME_MS	CompactCom モジュールと通信を開始するまでの待ち時間。 高速セットアップを使用しないときのみ有効。 abcc_user_def.h にて定義しない場合、デフォルト値は 1500ms。
ABCC_USER_REMAP_SUPPORT_ENABLED	再マッピングコマンドをサポートする場合は定義すること。 これを定義した場合、アプリケーションにて ABCC_CbfRemapDone() を実装する必要がある。
ABCC_USER_MAX_NUM_APPL_CMDS	応答を受信せずに送信可能なコマンドの数。
ABCC_USER_MAX_NUM_ABCC_CMDS	応答を送信せずに受信可能なコマンドの数。
ABCC_USER_MAX_NUM_MSG_RESOURCES	メッセージリソースの総数。 abcc_user_def.h にて定義しない場合、デフォルト値が使用される。 デフォルト値は (ABCC_USER_MAX_NUM_APPL_CMDS + ABCC_USER_MAX_NUM_ABCC_CMDS)
ABCC_USER_MAX_MSG_SIZE	使用されるメッセージの最大サイズ（単位 : バイト） 注！ CompactCom 30 は 255 バイトのメッセージをサポートし、CompactCom 40 は 1524 バイトのメッセージをサポートする。 ABCC_USER_MAX_MSG_SIZE に送受信される最大サイズを設定すること。最大サイズが不明な場合、サポートされている最大値を設定することを推奨する。
ABCC_USER_MAX_PROCESS_DATA_SIZE	送受信で使用されるプロセスデータの最大サイズ（単位 : バイト）
ABCC_USER_MEMORY_MAPPED_ACCESS	Anybus CompactCom のインターフェースはメモリマップ形式である。 これを定義すると、sys_adapt_par.h の直接メモリアクセスマクロが使用可能になる。
ABCC_USER_PARALLEL_BASE_ADR	メモリマップドインターフェースを使用する場合、Anybus CompactCom のベースアドレスを定義する。
ABCC_USER_INT_ENABLED	Anybus CompactCom の割り込み（IRQ_N ピン）を使用する場合は定義する。
ABCC_USER_INT_ENABLE_MASK	ABCC のどの割り込みを有効にするか定義する（abp.h の ABP_INTMASK_X を参照）。 abcc_user_def.h にて定義しない場合、デフォルトのマスクは 0。

定義名	説明
ABCC_USER_HANDLE_IN_ABCC_ISR_MASK	割り込みコンテキストにてどのイベントを処理するか定義する。割り込みで処理しない場合、ABCC_RunDriver() を使用してポーリングする必要がある。 abcc_user_def.h にて定義しない場合、デフォルト値は以下のとおり : ** パラレル 16/8: ABCC_USER_INT_ENABLE_MASK ** 他の動作モードは適用外
ABCC_USER_POLL_WRPD	これを定義すると、ABCC_RunDriver() が呼び出されるたびに WRPD が更新（ABCC_CbfUpdateWriteProcessData()）される。これを定義しない場合、ユーザーが関数 ABCC_NewWrPdEvent() を呼び出して WrPd の更新をトリガーする必要がある。
ABCC_USER_WD_TIMEOUT_MS	半二重（ピンポン）ウォッチドッグのタイムアウト。 ping の応答を正しく受信するまでの待ち時間（Par30, Ser）。SPI の場合、MISO フレームを正しく受信するまでの最大待ち時間を表す。PAR インターフェースは適用外。
ABCC_USER_ERR_REPORTING_ENABLED	エラー通知コールバック関数を有効にする。（ABCC_CbfDriverError()）
ABCC_USER_DEBUG_PRINT	ドライバーによりマクロを用いたデバッグ印刷（イベントやエラーのデバッグ情報などの印刷）が行われる。定義しない場合、ドライバーは何も出力しない。
ABCC_USER_64BIT_ADI_SUPPORT	64 ビット ADI をサポート。
ABCC_USER_DEBUG_EVENT_ENABLED	デバッグ印刷が有効。
ABCC_USER_DEBUG_ERR_ENABLED	ABCC_CbfDriverError() が呼び出されたときにデバッグ情報の印刷が可能。
ABCC_USER_BIG_ENDIAN	ビッグエンディアンのホストを使用。定義しない場合、リトルエンディアンとみなされる。

定義（SPI 固有）	説明
ABCC_USER_SPI_MSG_FRAG_LEN	各トランザクションにおける SPI メッセージフラグメントの長さ（単位 : バイト） 最大メッセージ長より短い場合、メッセージの送受信は複数の SPI トランザクションにて行われる。メッセージがあるかどうかにかかわらず、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つ。メッセージが重要な場合、メッセージが分割されるのを防ぐため、最大メッセージ長をフラグメント長に設定すること。IO データが重要な場合、メッセージのフラグメント長はそれより小さな値にしてもよい。
ABCC_USER_SPI_MAX_PD_LEN	SPI フレームのプロセスフィールドに割り当てる 16 ビットワードの最大数。 **注** : 実際にマッピングされたプロセスデータ長のみ SPI フレームで送信される。MOSI と MISO の SPI フレームは同じ長さでなければならないため、マッピングされた WrPd と RdPd の最大サイズが選択される。

7. ホストプラットフォーム依存の実装

プラットフォーム依存の関数はすべて、abcc_sys_adapt* ファイルに収録されています。これらの関数はドライバーでのみ使用され、アプリケーションでは使用されません。

Anybus CompactCom システム依存の関数

関数	説明
ABCC_SYS_Init()	この関数は、ABCC_StartDriver() の先頭でドライバーにより呼び出される。必要に応じて、ハードウェアやシステムに依存する初期化をここで行うこと。 なお、ABCC_StartDriver() はドライバー再起動時にも呼び出される。
ABCC_SYS_Close()	この関数は、ABCC_ShutDown() の最後でドライバーにより呼び出される。 なお、ABCC_StartDriver() を呼び出すことでドライバーを再起動できる。
ABCC_SYS_SetOpmode()	ABCC インターフェースの ABCC 動作モードピンを設定する。動作モードが固定されている場合、または、動作モードがディップスイッチにより設定される場合、この関数を空のままにするか、期待される動作モードの確認に使用できる。
ABCC_SYS_GetOpmode()	ハードウェアから ABCC の現在の動作モードを読み出す。
ABCC_SYS_HWReset()	モジュールをリセットする。ABCC インターフェースのリセットピンが Low に設定される。
ABCC_SYS_HWReleaseReset()	モジュールのリセットを解除する。ABCC インターフェースのリセットピンが High に設定される。
ABCC_SYS_ReadModuleId()	ホストコネクターのモジュール識別ピンを読み出す。 識別ピンが接続されていない場合、デバイスに応じて正しい値を返すこと。 0 0 (0) アクティブ CompactCom 30 シリーズ 0 1 (1) パッシブ CompactCom 1 0 (2) アクティブ CompactCom 40 シリーズ 1 1 (3) ユーザー固有
ABCC_SYS_ModuleDetect()	ABCC インターフェースのモジュール検出ピンを読み、モジュールが存在するかどうかを検出する。検出ピンが接続されていない場合は true を返すこと。
ABCC_SYS_AbccInterruptEnable()	ABCC 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）を有効にする。 この関数は、ABCC の割り込みが有効にされたときにドライバーにより呼び出される。
ABCC_SYS_AbccInterruptDisable()	ABCC 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）を無効にする。
ABCC_SYS_EnterCritical()	ABCC 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）との間に内部リソース衝突の可能性がある場合、ドライバーによりこの関数が呼び出される。この関数は、衝突を回避するために割り込みを一時的に無効にする。なお、ドライバーのアクセスが発生する可能性があるすべての割り込みを無効にする必要がある。割り込みが使用されておらず、1 つのスレッドまたはメインループからドライバーがアクセスされる場合、この関数を実装する必要はない。この関数を使用すると、すべての割り込みが無効になるため、複数のスレッドがドライバーに同時にアクセスすることで起こる衝突を回避できる。

関数	説明
ABCC_SYS_ExitCritical()	割り込みの状態を ABCC_SYS_EnterCritical() が呼び出されたときの状態に戻す。
ABCC_SYS_UseCritical()	ABCC_SYS_EnterCritical() または ABCC_SYS_ExitCritical() を呼び出す前に何らかの準備が必要な場合は、このマクロを使用してハードウェア固有の処理を追加する。

パラレル（8/16）固有の関数

関数	説明
ABCC_SYS_ParallelRead()	ABCC メモリから一定バイト数のデータを読み出す。
ABCC_SYS_ParallelRead8()	ABCC メモリから 1 バイトのデータを読み出す。 メモリマップドシステムの場合、この関数を実装する必要はない。
ABCC_SYS_ParallelRead16()	ABCC メモリから 1 ワードのデータを読み出す。 メモリマップドシステムの場合、この関数を実装する必要はない。
ABCC_SYS_READ8() ABCC_SYS_READ16()	ドライバーは、ABCC_SYS_READ8 マクロおよび ABCC_SYS_READ16 マクロを使用して ABCC のレジスターにアクセスする。
ABCC_SYS_ParallelWrite()	ABCC メモリに一定バイト数のデータを書き込む。
ABCC_SYS_ParallelWrite8()	ABCC メモリに 1 バイトのデータを書き込む。 メモリマップドシステムの場合、この関数を実装する必要はない。
ABCC_SYS_ParallelWrite16()	ABCC メモリに 1 ワードのデータを書き込む。 メモリマップドシステムの場合、この関数を実装する必要はない。
ABCC_SYS_CopyMemBytes()	ソースポインターの場所からデスティネーションポインターの場所に一定バイト数のデータをコピーする。
ABCC_SYS_WRITE8() ABCC_SYS_WRITE16()	ドライバーは、ABCC_SYS_WRITE8 マクロおよび ABCC_SYS_WRITE16 マクロを使用して ABCC のレジスターにアクセスする。
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する。
ABCC_SYS_ParallelGetWrPdBuffer()	保存するライトプロセスデータのアドレスを取得する。

SPI 固有の関数

関数	説明
ABCC_SYS_SpiRegDataReceived()	SPI モードで使用する。新規データを受信したときに呼び出すコールバック関数を登録する。
ABCC_SYS_SpiSendReceive()	SPI モードで使用する。SPI モードのデータ送受信を行う。

シリアル固有の関数

関数	説明
ABCC_SYS_SerRegDataReceived()	シリアルモードで使用する。シリアルチャネルで新規データが受信されたことを通知するコールバック関数を登録する。
ABCC_SYS_SerSendReceive()	シリアルモードで使用する。TX テレグラムを送信し、RX テレグラムの受信準備を行う。
ABCC_SYS_SerRestart()	シリアルモードで使用する。シリアルドライバーを再起動する。通常、テレグラムがタイムアウトしたときに使用する。

8. メッセージ処理

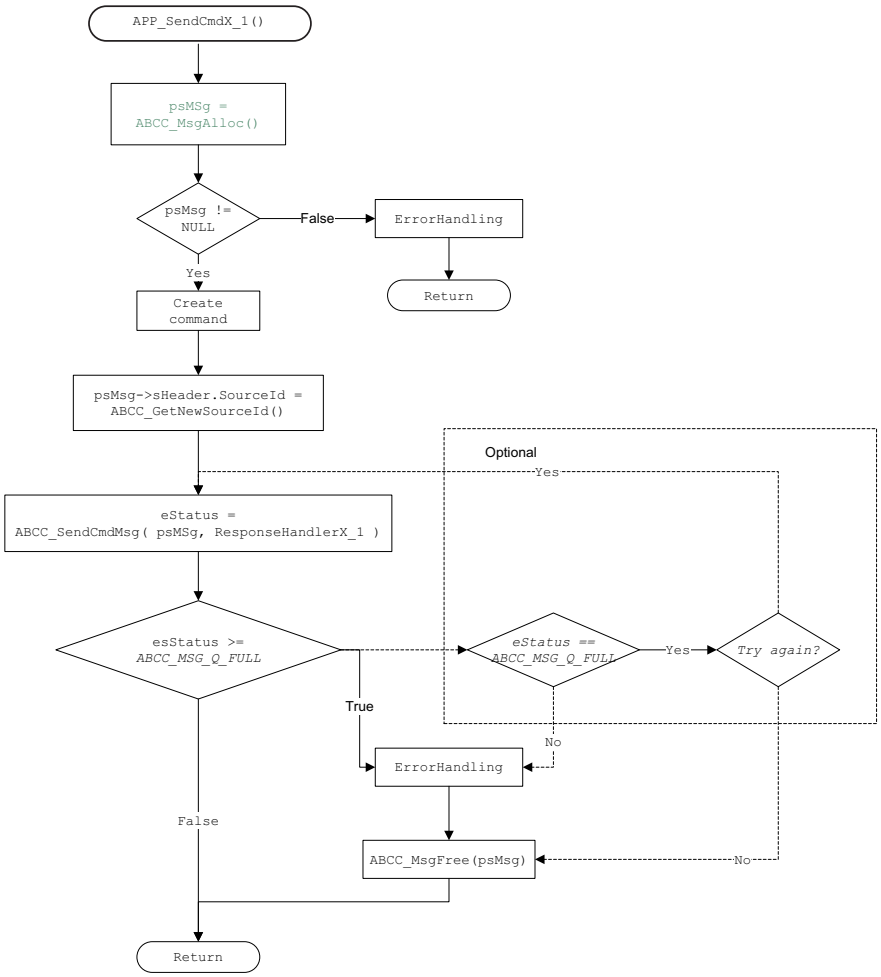
ドライバーにはメッセージ送受信メカニズムが用意されています。CompactCom モジュールにメッセージを直接送信できない場合、そのメッセージはキューに入れられ、モジュールがメッセージを受信できるようになり次第、送信されます。メッセージキューには、応答メッセージ用とコマンドメッセージ用の 2 つのキューがあります。応答キューの方が優先順位が高く、最初にチェックされます。各キューは FIFO 方式で処理されます。

8.1 コマンドの送信

コマンドメッセージをモジュールに送信する場合、ドライバーの API で用意されている ABCC_MsgAlloc() 関数（abcc.h）を使用してメッセージバッファを割り当てる必要があります。詳細は 13 ページ「API 関数」の ABCC_SendCmdMsg() を参照してください。以下のフローチャートにその例を示します。

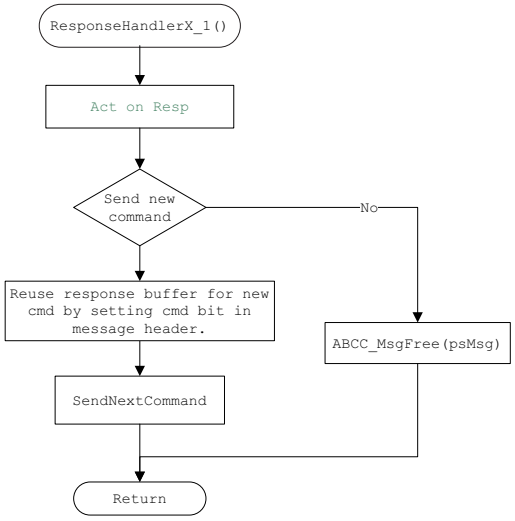
APP_SendCmdX_1() などの関数を実装します。この例では、ABCC_GetNewSourceId() 関数を使用してソース ID が作成されます。このとき、一意のソース ID を使用してください。応答を受信すると、ドライバーにより呼び出される ResponseHandlerX_1() 関数にソース ID が関連付けられます。

ドライバーが同時に処理可能な数よりも多くのメッセージリソースがある場合、送信キューが一杯になる可能性があります。これについては、フローチャートのオプションの箇所に記述されています。いずれにせよ、メッセージがドライバーにより受け付けられなかった場合は、ユーザーがバッファを解放するか、そのメッセージを後で再送する必要があります。



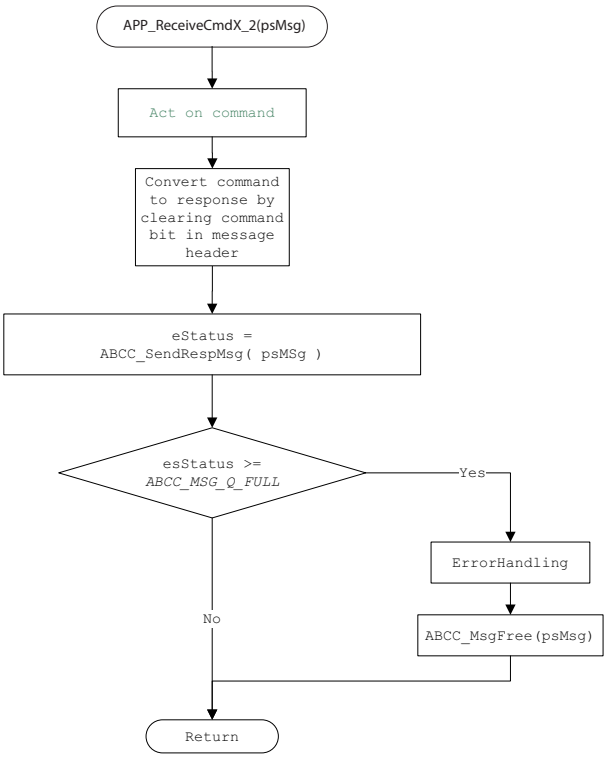
8.2 応答の受信

コマンドを送信するたびに、応答を処理する関数（応答ハンドラー）を用意する必要があります。以下のフローチャートに、応答に対する処理の簡単な例を示します。応答により新たなコマンドがトリガーされた場合、そのメッセージバッファを再利用できます。新たなコマンドがない場合、ABCC_MsgFree() を呼び出して受信バッファのメモリを解放する必要があります。



8.3 コマンドの受信

以下のフローチャートに、コマンド受信の例を示します。受信コマンドに対して使用したバッファは、応答の送信で再利用できます。応答がドライバーにより受け付けられなかった場合、メモリリークを防ぐために関数を呼び出してメッセージバッファを解放する必要があります。



9. ADI マッピング

ADI マッピングの作成に必要な型定義は、abcc_ad_if.h に用意されています。

9.1 ADI エントリリスト

ADI マッピングテーブルの例を以下に示します。

```
const AD_AdiEntryType AD_asADIEntryList[] =
{
/* Idx:0 */ { 1, “RadOnOff”,                  ABP_BOOL,   1, ALL_ACCESS, &VAPP_fRadOnOff,          &VAPP_BOOL8Props_fRadOnOff },
/* Idx:1 */ { 2, “RadSetTemp”,                ABP_FLOAT,  1, ALL_ACCESS, &VAPP_rRadSetTemp,        &VAPP_FLOAT32Props_fRadSetTemp },
/* Idx:2 */ { 3, “RadErrCode”,                ABP_ENUM,   1, ALL_ACCESS, &VAPP_eRadErrCode,        &VAPP_ENUMProps_eRadErrCode },
/* Idx:3 */ { 4, “FanOnOff”,                  ABP_BOOL,   1, ALL_ACCESS, &VAPP_fFanOnOff,          &VAPP_BOOL8Props_fFanOnOff },
/* Idx:4 */ { 5, “FanOpErrCode”,              ABP_ENUM,   1, ALL_ACCESS, &VAPP_eFanErrCode,        &VAPP_ENUMProps_eFanErrCode },
/* Idx:5 */ { 6, “ActTemp”,                   ABP_FLOAT,  1, ALL_ACCESS, &VAPP_rActTemp,           &VAPP_FLOAT32Props_rActTemp },
/* Idx:6 */ { 500, “ActTempErrCode”,          ABP_ENUM,   1, ALL_ACCESS, &VAPP_eActTempErrCode,    &VAPP_ENUMProps_ActTempErrCode },
/* Idx:7 */ { 501, “InputTemp”,               ABP_FLOAT,  1, ALL_ACCESS, &VAPP_rInputTemp,         NULL },
/* Idx:8 */ { 502, “HeatEffect_deg/sec”,      ABP_FLOAT,  1, ALL_ACCESS, &VAPP_rHeatEff,           NULL },
/* Idx:9 */ { 503, “SystemTripBits”,          ABP_UINT8,  1, ALL_ACCESS, &VAPP_bTripBits,          NULL },
};
```

各エントリは以下の構造になっています。

```
/* Idx:x */ { iInstance, pabName, bDataType, bNumOfElements, bDesc, pxValuePtr, pxValueProps }
```

エントリの項目について以下の表で説明します。

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。
pabName	ADI 名（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 の名前が返される。
bDataType	ABP_BOOL: Boolean ABP_SINT8: 8 ビット符号つき整数 ABP_SINT16: 16 ビット符号つき整数 ABP_SINT32: 32 ビット符号つき整数 ABP_UINT8: 8 ビット符号なし整数 ABP_UNIT16: 16 ビット符号なし整数 ABP_UINT32: 32 ビット符号なし整数 ABP_CHAR: 文字型 ABP_ENUM: 列挙型 ABP_SINT64: 64 ビット符号つき整数 ABP_UINT64: 64 ビット符号なし整数 ABP_FLOAT: 浮動小数点数（32 ビット）
bNumOfElements	bDataType で指定したデータ型の要素数。
bDesc	エントリ記述子。以下のコンフィグレーションに基づくビット値。 ABP_APPD_DESCR_GET_ACCESS: 値のアトリビュートに対する Get サービスが許可されている。 ABP_APPD_DESCR_SET_ACCESS: 値のアトリビュートに対する Set サービスが許可されている。 ABP_APPD_DESCR_MAPPABLE_WRITE_PD: 値のアトリビュートに対する Remap サービスが許可されている。 ABP_APPD_DESCR_MAPPABLE_READ_PD: 値のアトリビュートに対する Remap サービスが許可されている。 各記述子は論理 OR が可能です。 この例では、ALL_ACCESS は上記のすべての記述子を論理 OR したものです。
pxValuePtr	ローカル値の変数に対するポインター。型は bDataType により決まる。
pxValueProps	ローカル値のプロパティ構造体に対するポインター。NULL の場合はプロパティは適用されない（最大 / 最小 / デフォルト）。型は bDataType により決まる。

9.2 ライト / リードプロセスデータマッピング

マッピングされる ADI は、0xFFFF で終了するリストで定義します。リードプロセスデータとライトプロセスデータに対して 1 つの複合リストが用意されます。このリストの各要素は、AD_asADIEntryList の ADI に対する ADI インデックス（注 : インスタンス番号ではない）、およびマッピングがリード方向かライト方向かの情報で構成されます。各リストのエントリは、AD_asADIEntryList の ADI を指します。

```
static const AD_DefaultMapType AD_asDefaultMap[] =
                                {{0, PD WRITE},
                                 {3, PD READ}
                                 {1, PD WRITE},
                                 {4, PD READ},
                                 {2, PD WRITE},
                                 {5, PD READ},
                                 {0xFFFF}};
```

セットアップシーケンス時、Anybus CompactCom ドライバーは、ABCC_CbfAdiMappingReq() を呼び出してこの情報を要求します。

9.3 実装例

上記のマッピング例の場合、実装は以下のようになります。

```
void ABCC_CbfAdiMappingReq(   const AD_AdiEntryType** const ppsAdiEntry,
                        const AD_DefaultMapType**  const ppsDefaultMap )
{
  *ppsAdiEntry = AD_asADIEntryList;
  *ppsDefaultMap = AD_asDefaultMap;
}
```

セットアップ時にマッピングが必要ない場合、実装は以下のようになります。

```
void ABCC_CbfAdiMappingReq(   const AD_AdiEntryType** const ppsAdiEntry,
                        const AD_DefaultMapType**  const ppsDefaultMap )
{
  *ppsAdiEntry = NULL;
  *ppsDefaultMap = NULL;
}
```

10. ステップバイステップのユーザーガイド

本章では、Anybus CompactCom 40 Driver を用いた簡単なサンプルアプリケーションを比較的少ない労力でセットアップする方法について、ステップバイステップで説明します。これは完全な実装を示すものではありません。ドライバーを使用して独自のアプリケーションを実装する手順をより深く理解するための第一歩とお考えください。

10.1 概要

多くのアプリケーションでは、大幅に変更することなく標準のドライバーを使用できます。以下の手順を行うと、実際に機能する簡単なアプリケーションを作成できます。各手順の詳細は、本章の後ろのセクションで説明します。

作業を始める前に、いくつかアドバイスがあります。

- C コンパイラーは C90 互換であること。
- 『Anybus CompactCom 40 Software Design Guide』を読み、Anybus CompactCom のコンセプトについて理解すること。
- ハードウェアが期待どおりに機能することを確認すること。
- C プログラミング言語に関する知識を備えていること。
- 製品の開発環境に関する知識を備えていること。
- プログラミングを行う際は、本マニュアルの参照箇所や、ソースコード（主に abcc.h、abcc_sys_adapt.h、abcc_ad_if.h）のコメントを十分活用すること。

10.2 ステップバイステップ

以下の手順で作業を行います。

1. プロジェクトを準備する。
2. ドライバーを設定する。
3. システム固有のファイルを実装する。または、既存の実装を選択する。
4. 使用していないコードを削除する。
5. アプリケーションを記述する。
   - デフォルトの ADI をマッピングする（必要な場合）。
   - ユーザー固有のセットアップを実装する（任意）。
6. コンパイルを行ってアプリケーションを実行する。

手順 1: 準備

ドライバーファイルを変更する前に、プロジェクトを正しくセットアップする必要があります。

- **空のプロジェクトを作成する**
- **ドライバーファイルをプロジェクトフォルダーにコピーする**
- **プロジェクトにドライバーファイルを追加する**

- **abcc_user_def_example.h の名前を abcc_user_def.h に変更する**
  古いバージョンのドライバーをアップデートする場合は、古い abcc_user_def.h と新しい abcc_user_def.h の差分をとることを推奨します。

- **新規ファイルを作成して単純なメイン関数を追加する**

```
#include “abcc_td.h”
#include “abcc.h”
#include “abp.h”
#include “abcc_ad_if.h”

main()
{
  //nothing yet
}
```

- **‘abcc_td.h’ を変更する**
  このファイルには、ドライバーで使用するさまざまな型が用意されています。標準の 8、16、32 ビットのデータ型（符号つきおよび符号なし）が、お使いのコンパイラーに適合しているか確認してください。

- **コンパイル**
  プロジェクトがエラーなしにコンパイルされたことを確認してください。

手順 2: ドライバーを設定する

abcc_user_def.h で記述するユーザー定義の #define は、システムの設定や定数（タイムアウト、バッファサイズ、キューサイズ、ポーリング方法、割り込み方法）の定義に使用します。必要に応じてドライバーファイルにて abcc_user_def.h をインクルードします。

アプリケーションにインクルードする基本定義

- **スタートアップ時間の設定**
  #define ABCC_USER_STARTUP_TIME_MS は、リセット解除からモジュールが通信可能となるまでの時間を指定します。1500ms に設定してください。モジュールが通信可能なことを示す電源オン割り込みを使用しない場合のみ有効です。

- **割り込み通信の設定**
  #define ABCC_USER_INT_ENABLE は、Anybus CompactCom の割り込み（IRQ_N pin）を使うことを指定します。シリアル通信では使用しないでください。

- **バッファサイズとキューサイズの設定**
  以下の定義によりバッファとキューのサイズが決定されます。
  - #define ABCC_USER_MAX_NUM_APPL_CMDS
  - #define ABCC_USER_MAX_NUM_ABCC_CMDS
  - #define ABCC_USER_MAX_MSG_SIZE
  - #define ABCC_USER_MAX_PROCESS_DATA_SIZE

- **ポーリング方法の決定**
  - #define ABCC_USER_POLL_WRPD
    これを定義すると、ABCC_RunDriver() が呼び出されるたびに、ABCC_CbfUpdateWriteProcessData() により WRPD が更新されます。これを定義しない場合、ユーザーが ABCC_NewWrPdEvent() を呼び出して WRPD の更新をトリガーする必要があります。

ABCC_USER_POLL_WRPD を定義し、ABCC_RunDriver() を使用して新規 WRPD データを要求することを推奨します。

特定の動作モードに関する定義

- **パラレルモード（8/16 ビット、イベント駆動型）**
  - #define ABCC_USER_DRV_PARALLEL を定義する
  - #define ABCC_USER_INT_ENABLE_MASK を設定し、どの割り込みを許可するかを定義する

  詳細は 16 ページ「コンフィグレーションパラメータ」および 29 ページ「イベントベースのアプリケーション」を参照してください。

- **SPI モード（イベント駆動型）**
  - #define ABCC_USER_DRV_SPI を定義する
  - #define ABCC_USER_ABCC_INT_ENABLE を定義する

  以下の #define を定義して SPI 通信を設定する :
  - #define ABCC_USER_SPI_MSG_FRAG_LEN
  - #define ABCC_USER_WD_TIMEOUT_MS

  詳細は 17 ページ「定義（SPI 固有）」を参照してください。

- **シリアルモード（UART、半二重）**
  - #define ABCC_USER_DRV_SERIAL を定義する
  - #define ABCC_USER_WD_TIMEOUT_MS を設定し、適切なタイムアウト値を設定する詳細は 17 ページ「ABCC_USER_WD_TIMEOUT_MS」を参照してください。

- **パラレル 30 モード（8 ビット、半二重）**
  - #define ABCC_USER_DRV_PARALLEL_30 を定義する
  - 必要に応じて #define ABCC_USER_INT_ENABLE を定義する
  - #define ABCC_USER_WD_TIMEOUT_MS を設定し、適切なタイムアウト値を設定する詳細は 17 ページ「ABCC_USER_WD_TIMEOUT_MS」を参照してください。

手順 3: システム固有の設定を実装する

- **ホストプラットフォームの適応ファイルの変更**

  12 ページ「システム適応インターフェースファイル」および 18 ページ「ホストプラットフォーム依存の実装」を参照してください。‘abcc_sys_adapt.h’ の関数と、‘abcc_sys_adapt_par.h’、‘abcc_sys_adapt_spi.h’、‘abcc_sys_adapt_par30.h’、または ‘abcc_sys_adapt_ser.h’ の関数は、使用されているターゲットのハードウェア固有の部分に対するインターフェースを定義します。

  これらのファイルを変更しない場合、システム適応の実装サンプルの状態のままになります。sys_template/abcc_sys_adapt.c に、空の実装サンプルが用意されています。

  完成された実装が利用できる場合はそれを使用してください。

- **コンパイル**

  プロジェクトがエラーなしにコンパイルされたことを確認してください。

手順 4: 使用していないコードを削除する

ドライバーには、特定のアプリケーションでは使用されないコードが多く含まれています。例えば、使用しない動作モードやその適応ファイルがそれに該当します（11 ページ「ファイルとフォルダー」を参照）。このコードをプロジェクトから削除してください。プロジェクトがエラーなしにコンパイルされることを確認してください。

手順 5: アプリケーションを記述する

デフォルト ADI のマッピング

独自の ADI をマッピングする必要がある場合は、ここでマッピングを行います。22 ページ「ADI マッピング」を参照してください。

ユーザー固有のセットアップの実装

ANB_SETUP フェーズにて、ユーザー固有の部分を実装します（Profibus スレーブアドレスの設定など）。ネットワークから読み取り可能なデータは、遅くてもここで初期化してください。Anybus のセットアップステートマシンがここから先の状態に遷移すると、エンドユーザーはいつでもデータを読むことができます。

（ユーザー固有の実装は必須ではありません。ただし。ABCC_UserInitComplete() を必ず呼び出す必要があることに留意してください）詳細は 10 ページ「Anybus のセットアップステートマシン」を参照してください。

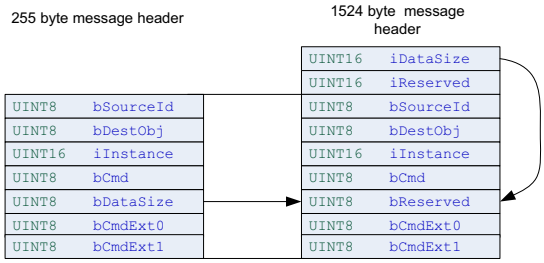
これで、必要なすべての #define が設定され、アプリケーションの記述が可能となりました。

手順 6: コンパイルと実行を行う

A. メッセージヘッダーのフォーマット変換

メッセージヘッダーのフォーマット変換はドライバー内部で行われるため、ユーザーは何もする必要はありません。この変換は単純で、データサイズフィールドをヘッダーの正しい位置にコピーするだけです。

40 シリーズのモジュールでは、最大 1524 バイトのメッセージデータを処理できるように改善されました。この改善により、メッセージヘッダーのデータサイズインジケーターを 16 ビットに拡張する必要があります。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 16 ビットでなければならないため、古いフォーマットとは異なります。このドライバーは、40 シリーズのモジュールに加えて 30 シリーズのモジュールとの通信をサポートしますが、ドライバーの API は新しいメッセージフォーマットのみサポートします。Anybus CompactCom 30 モジュールが使用されている場合、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 つのメッセージフォーマットを示します。



B. ドライバーの使用

B.1 タイマーシステム

ABCC_RunTimerSystem() を定期的に呼び出すタイマー割り込みを設定することを推奨します。

```
interrupt MyTenMSTimerInterrupt( void )
{
  ABCC_RunTimerSystem( TimeInMsSinceLastCall );
}
```

または、メインのポーリングループ内で行ってください。

```
while( True )
{
  Delay(10)
  ABCC_RunTimerSystem( 10 );
..
}
```

B.2 イベントベースのアプリケーション

イベントベースのアプリケーションを動作させるには、動作モードをパラレル 8/16 ビットにする必要があります。

イベントベースのアプリケーションでは、ABCC の割り込みを扱う割り込み処理ルーチンから ABCC_ISR() を呼び出す必要があります。

user_def.h にて以下の定義を行う必要があります。

```
#define ABCC_USER_INT_ENABLED
#define ABCC_USER_INT_ENABLE_MASK ( ABP_INTMASK_RDPDIEN | ABP_INTMASK_RDMSGIEN
                                    ABP_INTMASK_WRMSGIEN | ABP_INTMASK_ANBRIEN
                                ABP_INTMASK_ANBRIEN | ABP_INTMASK_STATUSIEN )
```

ABCC_USER_INT_ENABLED は、ドライバーの割り込み処理を有効にします。

ABCC_USER_INT_ENABLE_MASK は、ABCC_ISR() を呼び出す実際の割り込みを制御します。割り込みマスクは abp.h で定義されています。

デフォルトでは、ABCC_USER_INT_ENABLE_MASK で定義されるすべての割り込みが ABCC_ISR() で直接処理されます。一部の割り込みのみ処理したい場合は、ABCC_ISR() によって処理されるイベントを定義できます。

```
#define ABCC_USER_HANDLE_IN_ISR_MASK      ABP_INTMASK_RDPDIEN
```

上記の例では、リードプロセスデータが割り込みで処理されます。その他のすべてのイベントは、ABCC_CbfEvent() を呼び出すことでアプリケーションに渡されます。

ドライバーは、エッジまたはレベルでトリガーされた割り込みをサポートします。
(ABCC_ISR() 関数は、許可されたイベントの割り込みステータスレジスタがすべてクリアになるまで実行されます)

B.3 アプリケーションのメインループ

このセクションでは、メインループの簡単な例を示します。

```
void main( void )
{

/* The operation mode is set */
  ABCC_StartDriver( opMode, fPingPong );

/* Release reset of the Anybus CompactCom module */
  ABCC_HwReleaseReset();

/* Enable timer interrupt */

  while( !isReadyForCommunication() );

/* The SETUP sequence has started and interrupts are enabled if configured */
```

ドライバーをポーリングする場合、ここからポーリングを開始します。

```
/*
** Start running the driver
*/
  while(ABCC_RunDriver();
  {

    Delay( 1ms );

  }

/*
** Stop the driver
*/

/* Disable the 10 ms interrupt */
  ABCC_ShutdownDriver();

/*Reset of the Anybus CompactCom module */
  ABCC_HwReset();
```

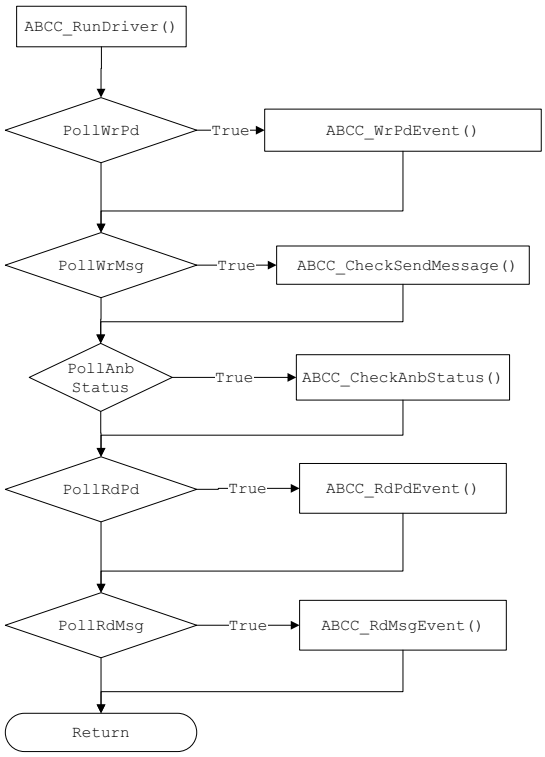
完全なイベントベースのドライバーの場合、ABCC_ISR() がすべてのタスクとイベントを処理するため、ABCC_RunDriver() の呼び出しをスキップできます。

B.4 フローチャート

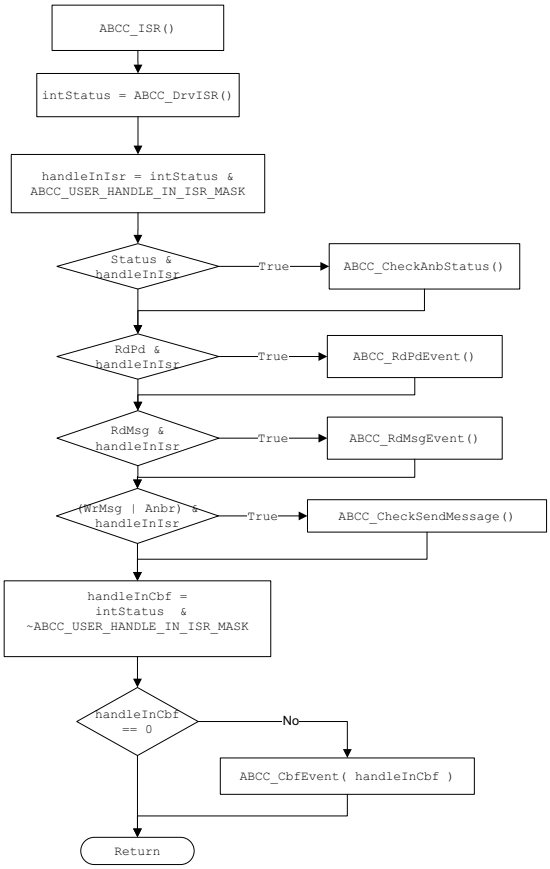
以下に、主な関数のフローチャートを示します。

- イベント駆動型パラレル RunDriver のフロー
- イベント駆動型パラレル割り込みサービスルーチン
- パラレル 30 およびシリアル（割り込みなし）の RunDriver のフロー
- パラレル 30（割り込みあり）の RunDriver のフロー
- パラレル 30（割り込みあり）の割り込みサービスルーチン
- SPI の RunDriver のフロー
- SPI の割り込みサービスルーチン
- メッセージデータ読み込みのフロー
- プロセスデータ読み込みのフロー
- プロセスデータ書き込みのフロー

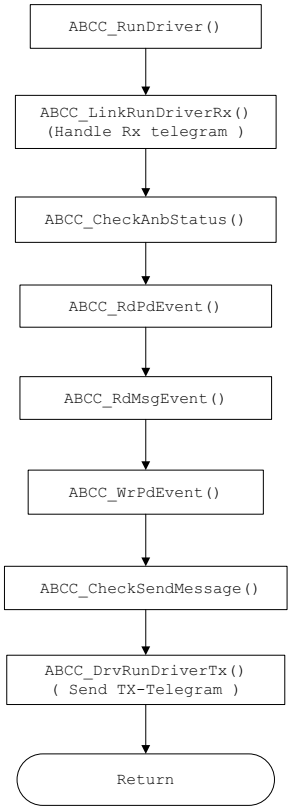
イベント駆動型パラレル RunDriver のフロー



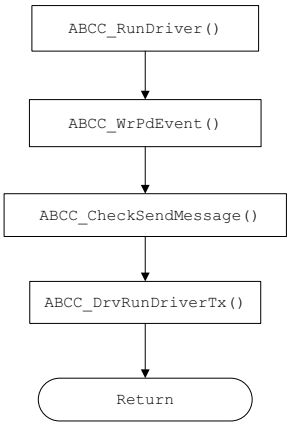
イベント駆動型パラレル割り込みサービスルーチン



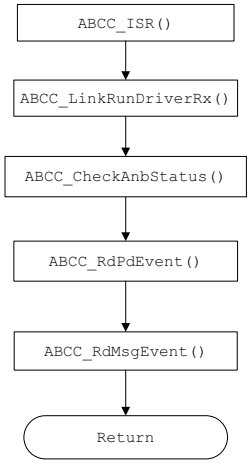
パラレル 30 およびシリアル（割り込みなし）の RunDriver のフロー



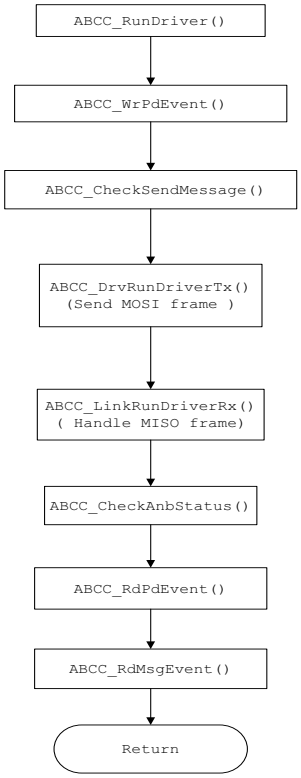
パラレル 30（割り込みあり）の RunDriver のフロー



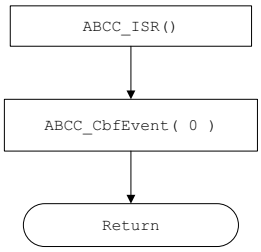
パラレル 30（割り込みあり）の割り込みサービスルーチン



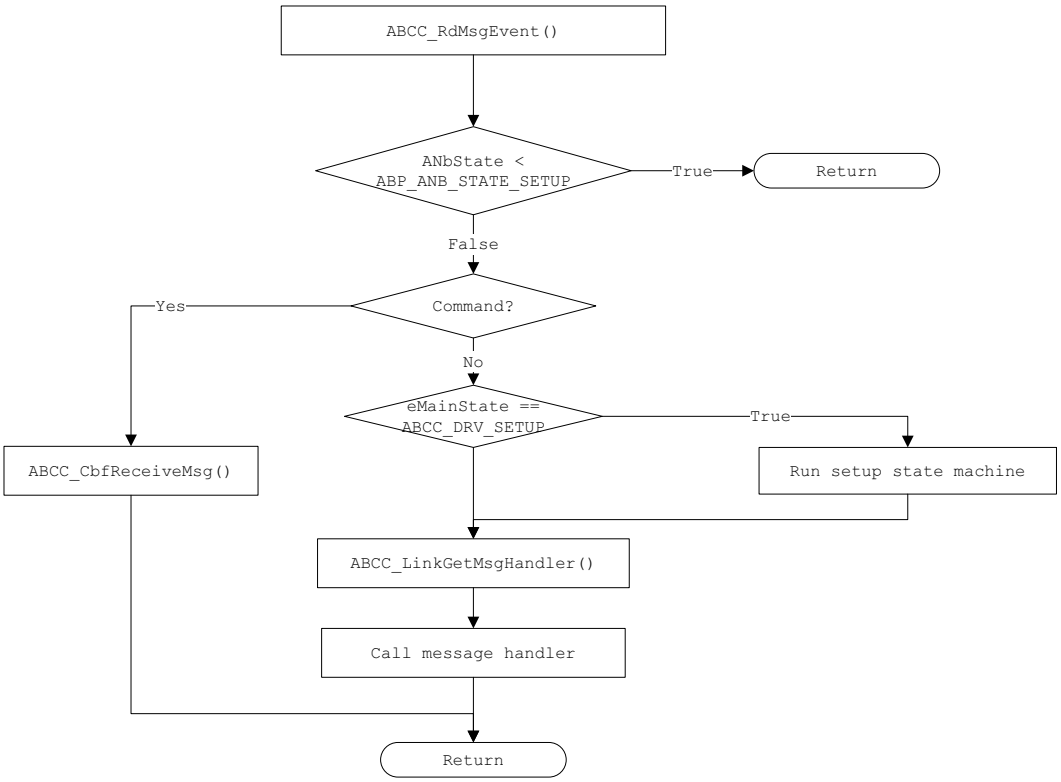
SPI の RunDriver のフロー



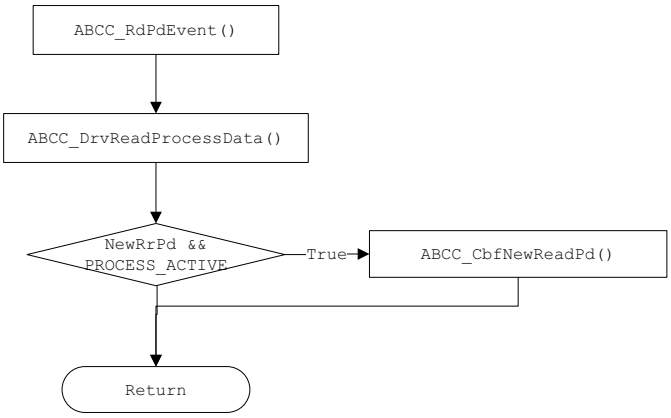
SPI の割り込みサービスルーチン



メッセージデータ読み込みのフロー



プロセスデータ読み込みのフロー



プロセスデータ書き込みのフロー