Anybus-S(ABS)
プログラミング参考資料
A00

エイチエムエス・インダストリアルネットワークス株式会社
〒222-0033
神奈川県横浜市港北区新横浜 3-19-5
新横浜第 2 センタービル 6F
TEL：045-478-5340
FAX：045-476-0315

URL
www.anybus.jp

EMAIL
セールス:jp-sales@hms-networks.com
サポート:jp-support@hms-networks.com

EVOLUTION OF THE DOCUMENT

Issue	Date	Author	Motive and nature of the modifications
A00	2012/01/06	YOH	First release.

This document contains: 31 pages.

1. Anybus-S 概要

- Anybus-S は、メモリーと高性能プロセッサを搭載した、組み込み型フィールドバス通信用モジュールです。フィールドバス通信に要求されるすべてのソフトウェアならびにハードウェア機能が組み込まれていて、エンジニアはアプリケーションや他のタスクに専念することができます。

- データは DPRAM を介して通信されます。そのため、高速通信を実現し、一般的にアプリケーションに対して僅かな負荷しか与えません。

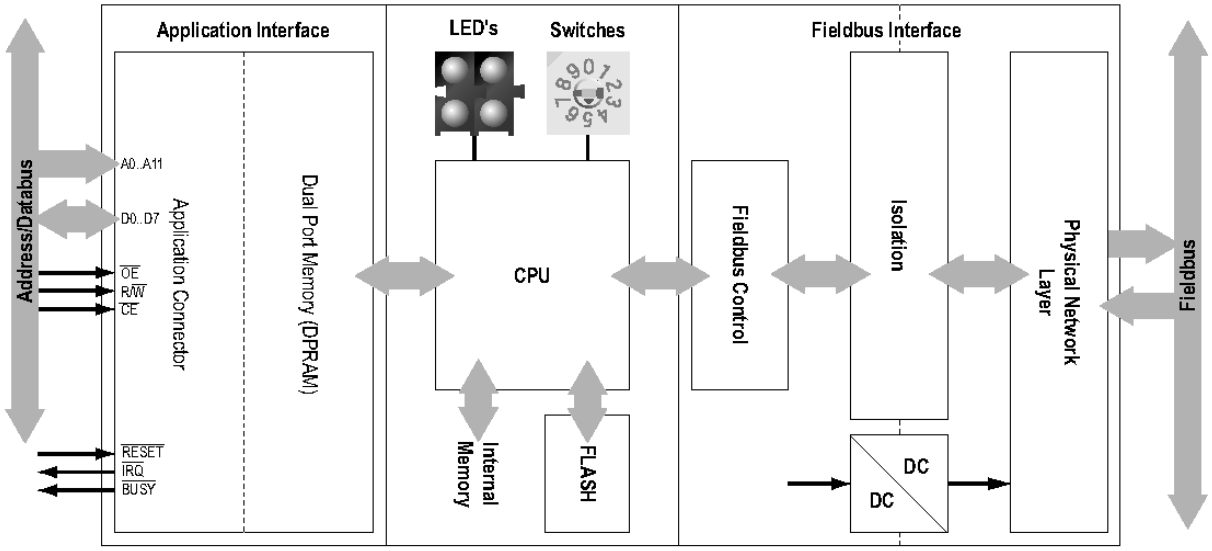
- Anybus-S モジュールは、ハードウェア/ ソフトウェアとも共通したインターフェースのため、異なるモジュールとの互換性を実現します。 また、同一基板を異なるフィールドバス・システムで使用することも可能です。

- 代表的なアプリケーションとして、周波数インバータ、HMI、表示機器、計測器、スケール、ロボット、PLC など、インテリジェントな計測機器があげられます。

1.1. 特徴

- 内部交換可能 (各フィールドバスに共通したインターフェース)。
- スレーブバージョンを提供（マスターバージョンとして Anybus-M があります)。
- 共通アプリケーション･インターフェースにより主要フィールドバス･システムをサポート。
- CPU を搭載していますためネットワーク対応を迅速に実現。
- すべてのフィールドバス・ネットワークの認証取得済み。
- メールボックス・インターフェース。
- 2K バイトデュアル・ポート RAM (DPRAM) アーキテクチャ。
- 最大 2048 バイトの入/出力データ対応。
- オン・ボード・コンフィギュレーション・スイッチ。
- オン・ボード LED 表示。
- 絶縁処理済みフィールドバス・インターフェース。
- CE, UL & cUL 承認済み。

1.2. ブロック図



1.3. アプリケーション・インターフェース

8 ビット並列ポート・インターフェースを装備した、アドレス/ データ・バスを持つマイクロプロセッサ搭載システムに容易に組み込むことのできるアプリケーション・インターフェースが共通しています。さらにこのアプリケーション・インターフェースはまた、リセット・ピン、ビジー・シグナル、そして割り込み要求シグナル機能を提供します。

1.4. フィールドバス・インターフェース

Anybus-S モジュールのフィールドバス・インターフェースは絶縁処理がされていて、各フィールドバス標準向けにデザインされています。フィールドバス・プロトコルは Anybus-S 側で扱われ、アプリケーションによる処理を必要としません。Anybus-S モジュールは、各フィールドバスの可能性を最大限に利用できるよう、各フィールドバス向け仕様に対応しています。

2. メモリー構成

2.1. データ種別

I/O データ
このタイプのデータは通常高速フィールドバス・データ(サイクリック・データ)に該当します。

メッセージ・データ
このタイプのデータは通常低速フィールドバス・データ（アサイクリック・データ）に該当します。

2.2. DPRAM

DPRAM は Anybus モジュールとアプリケーションプログラムの共有メモリーとなっています為に、個々の領域ごとにアクセス権を確保してからアクセスする必要が有る。以下のメモリー マップ（Parallel Interface Design Guide Anybus-S Slave & Master から抜粋）を参照下さい。

アドレス	領域	アクセス	注意
000h-1FFh	入力データ領域	R/W	7-1"フィールドバス・データ交換"を参照して下さい。
200h-3FFh	出力データ領域	RO	7-1"フィールドバス・データ交換"を参照して下さい。
400h-51Fh	メールボックス入力領域	R/W	8-1"メールボックス・インターフェース"を参照して下さい。
520h-63Fh	メールボックス出力領域	RO	8-1"メールボックス・インターフェース"を参照して下さい。
640h-7BFh	フィールドバス個別領域	-	（個別 Fieldbus Appendix を参照して下さい）。
7C0h-7DFh	制御レジスタ領域	R/W	4-1"制御レジスタ領域"を参照して下さい。
7FEh-7FFh	ハンドシェーク・レジスタ	R/W	5-1"ハンドシェーキング&インジケーションレジスタ"を参照して下さい。

これらの領域はアクセス前に割り当てなければなりません。詳細については 5-1"ハンドシェーキング&インジケーションレジスタ"を参照して下さい。

これらの領域は直接アクセスできます。

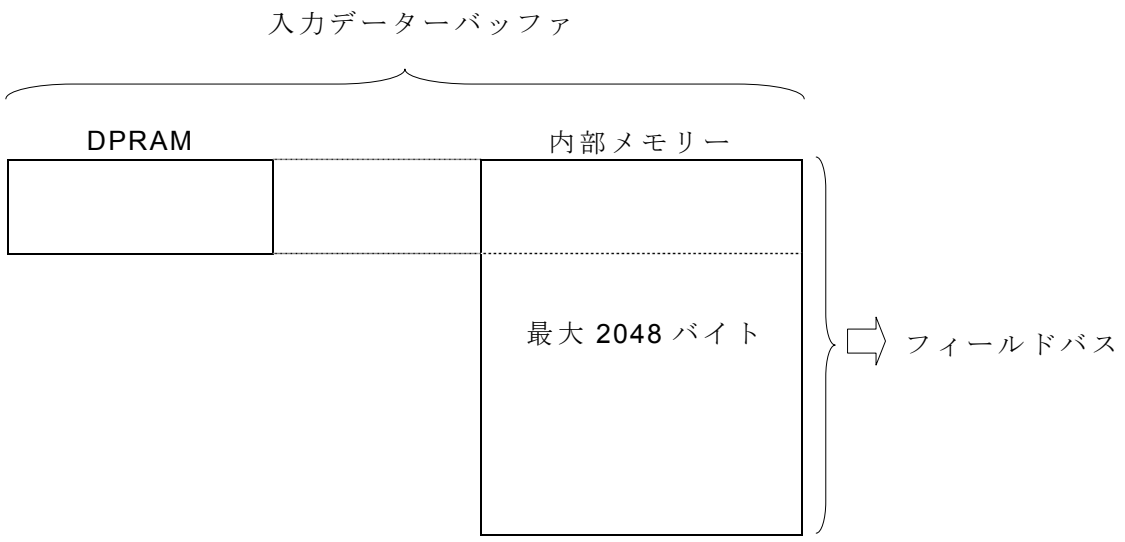
2.3. データバッファ

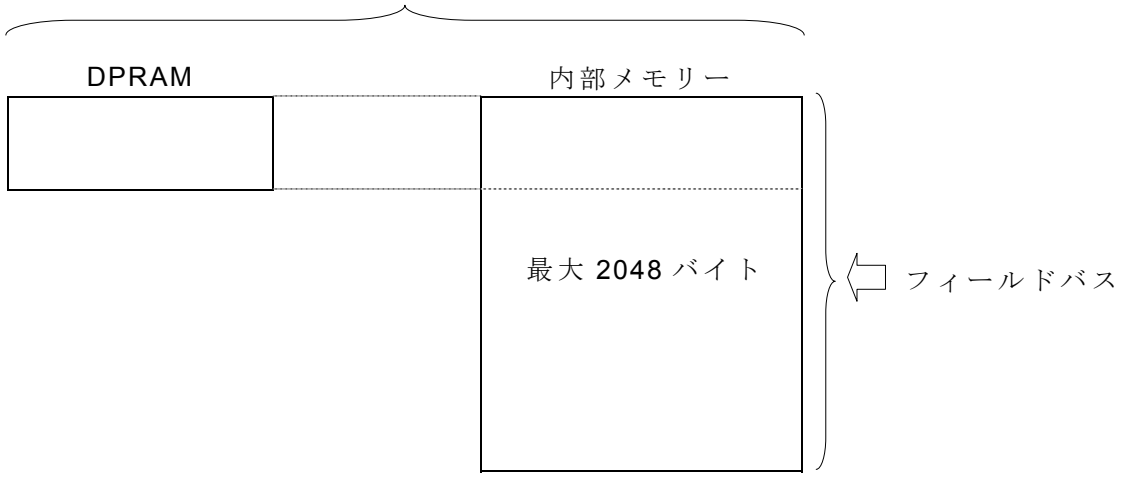
モジュールは入力/出力で独立した２つのデータ・バッファ（入力データバッファ/出力データバッファ）を介してフィールドバス上でデータを交換します。

入力データ・バッファ
・　このバッファに書き込まれたデータはフィールドバスへ送信されます。

出力データ・バッファ
・　このバッファはフィールドバスから受信したデータを含みます。

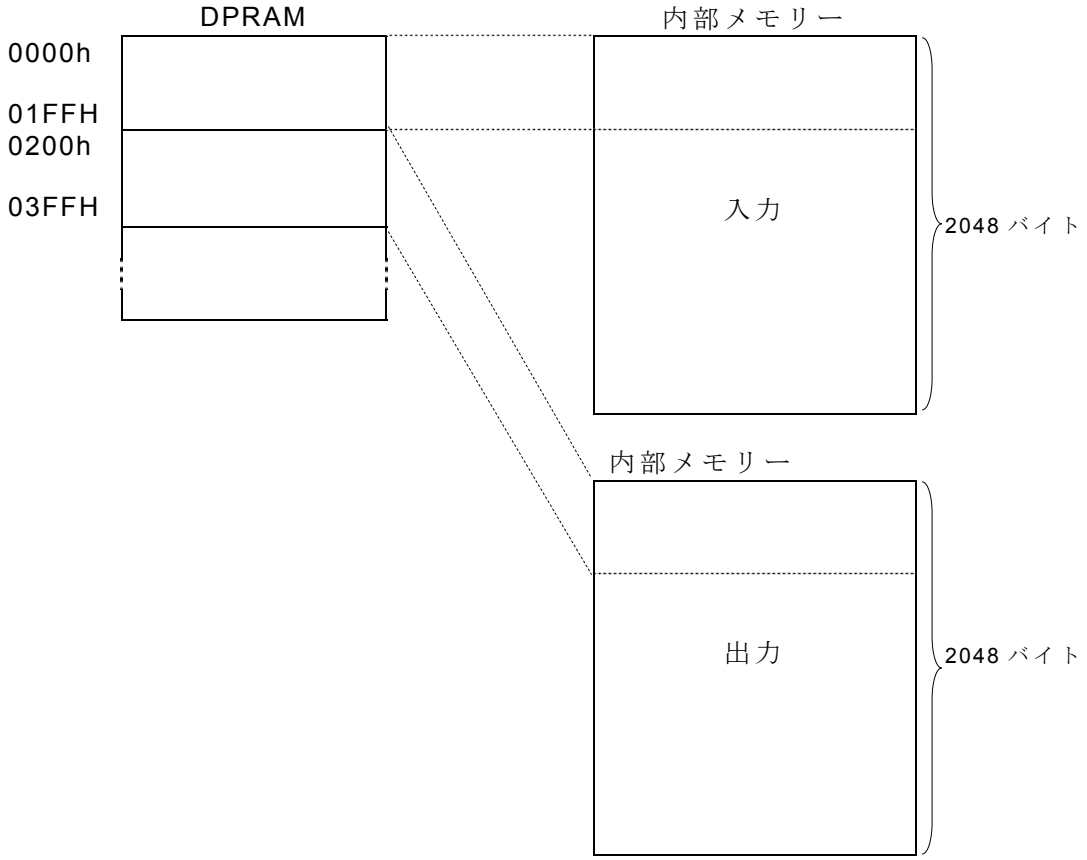
基本的にはフィールドバス上でデータ交換するためにアプリケーションがしなければならない事は、これらの 2 つのバッファ間のデータを読み込み又は書き込みをすることです。





2.4. DPRAM と内部メモリー

- データバッファは DPRAM と内部メモリーとを組み合わせる事で高速アクセスと大容量を実現します。
- DPRAM は直接アクセス可能な為に高速でアクセス出来ますが、内部メモリーはメールボックス経由でのアクセスとなる為に低速でのアクセスになります。
- 内部メモリーの最大サイズは入力／出力各々2048 バイトです(*1)。
- 2048 バイトの内で DPRAM にマッピング出来る最大サイズは 512 バイト。
- DPRAM 内にどれだけのデータを置くか、内部メモリー内にどれだけのデータを置くかは初期化時に設定します(*2)。



注記）
(*1)： フィールドバスの仕様、モジュールの機種(Anybus-M)等によっては実際にデーター交換に使用できるサイズは 2048 バイトより小さい場合があります。
(*2)： 内部メモリーの容量と構成はモジュールが初期化時に決定されます。従ってモジュールが初めに正しく初期化されていなければモジュールはデータ交換を行うことができません。

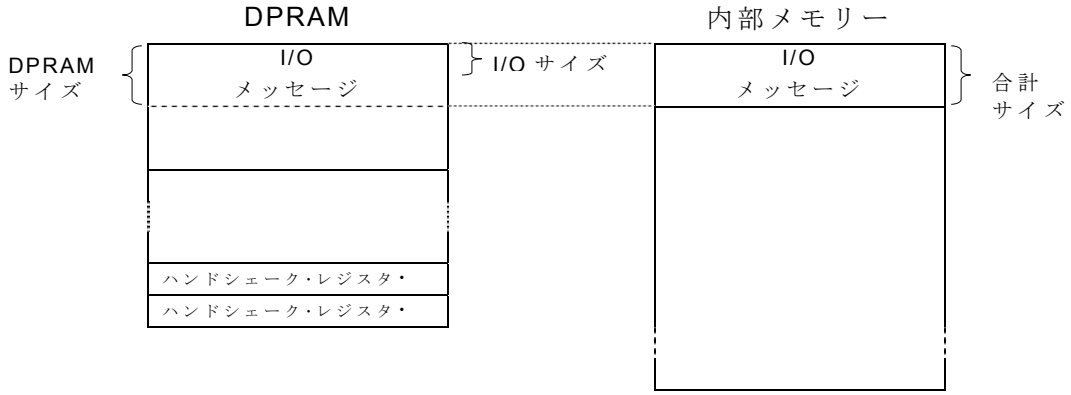
2.4.1.データ構成

I/O データとメッセージ・データの構成はメールボックス・コマンド‘Anybus Init’でモジュールの初期化中に決定されます。

I/O サイズ
- このパラメータで定義したサイズが I/O データとして確保されます。

DPRAM サイズ
- このパラメータは DPRAM 内に割り当てられたデータサイズを定義します。
- DPRAM サイズは 512 バイトを超えることはできません。
- メッセージデータは合計サイズから I/O データサイズを引いた残りとなります。

合計サイズ
- このパラメータは I/O データ＋メッセージ・データの合計サイズを定義します。
- 先頭から I/O データ領域が確保されてます。
- 最大合計サイズは 2048 バイトです。

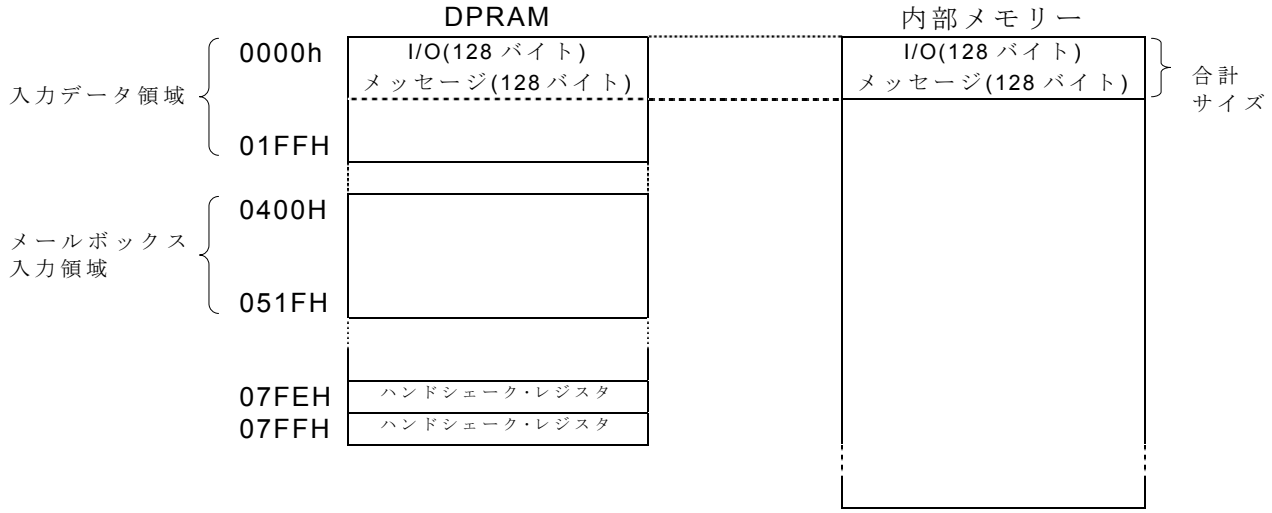


2.4.2.データ構成例

例 1

Input Data として I/O データ（サイクリックデータ）128 バイト、メッセージデータ（アサイクリックデータ）128 バイトを DPRAM 領域を使用してアクセスを行なう場合。

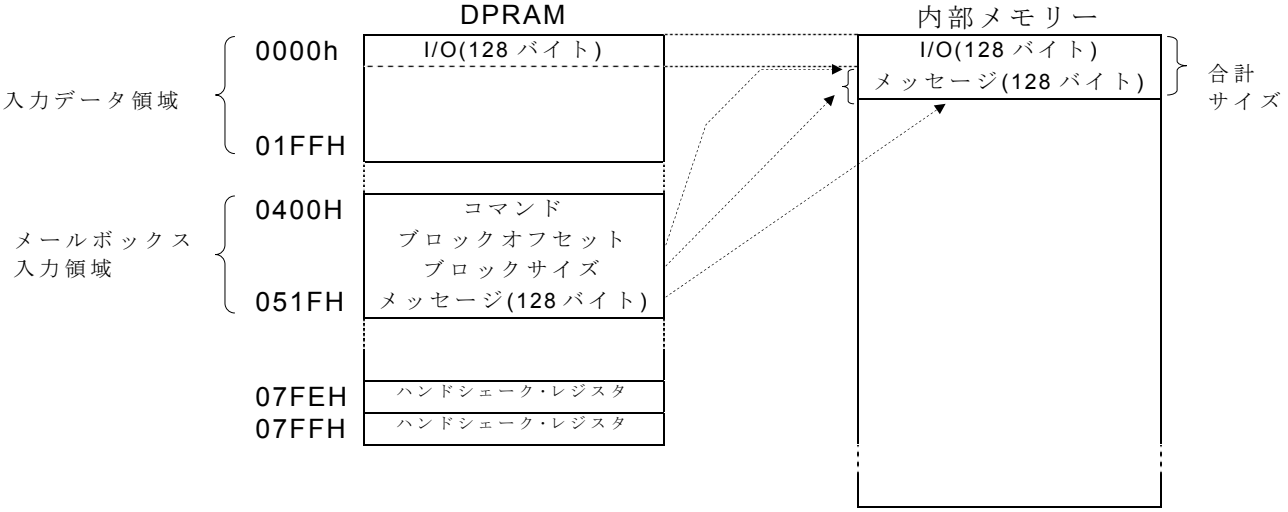
Input I/O Length(I/O サイズ): 128 バイト
Input DPRAM Length（DPRAM サイズ）: 256 バイト
Input Total Length（合計サイズ）: 256 バイト



例 2

Input Data として I/O データ（サイクリックデータ）128 バイトを DPRAM 領域を使用してアクセス、メッセージデータ（アサイクリックデータ）128 バイトをメールボックスを使用してアクセスを行なう場合。

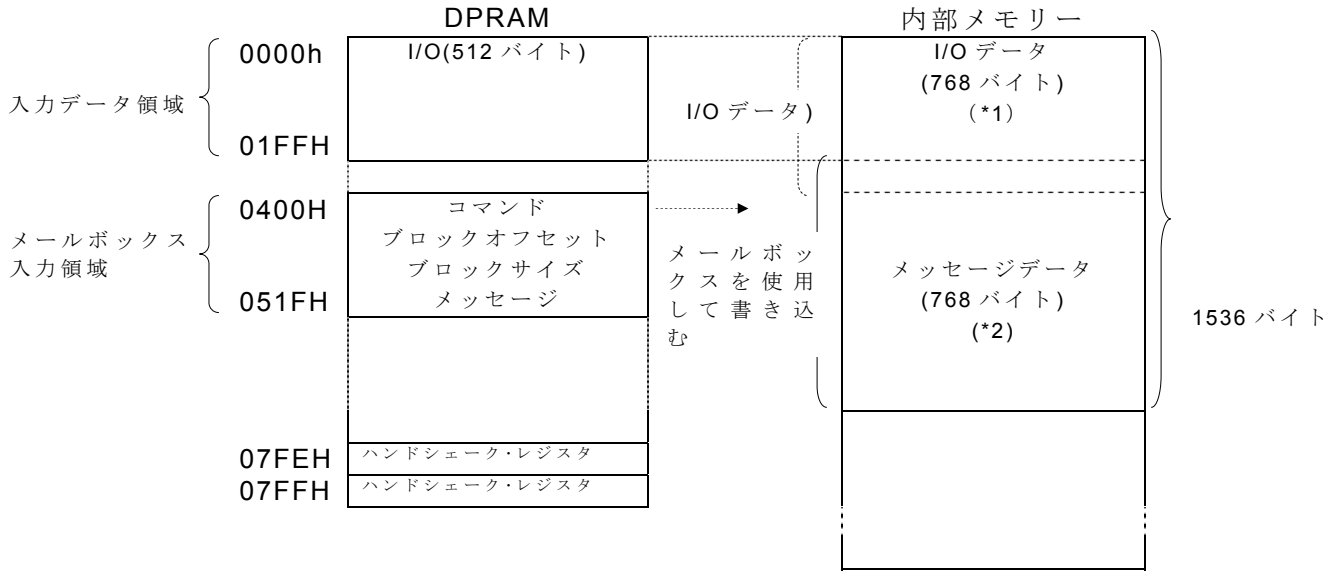
Input I/O Length(I/O サイズ): 128 バイト
Input DPRAM Length（DPRAM サイズ）: 128 バイト
Input Total Length（合計サイズ）: 256 バイト



例 3

Input Data として I/O データ（サイクリックデータ）768 バイトを DPRAM とメールボックスを使用してアクセス、メッセージデータ（アサイクリックデータ）768 バイトをメールボックスを使用してアクセスを行なう場合。

Input I/O Length(I/O サイズ): 768 バイト
Input DPRAM Length（DPRAM サイズ）: 512 バイト
Input Total Length（合計サイズ）: 1536 バイト



注記）

(*1) DPRAM の入力データ領域にマッピングされていない I/O データの書き込みはメールボックスを使用して行なわれます。上記例の場合、I/O データとして合計 768 バイトを書き込むようになっており、DPRAM へは 512 バイト（DPRAM の入力データ領域は最大 512 バイト）がマッピングされている為､残りの 128 バイトはメールボックスを介して内部メモリーの先頭アドレス(ブロックオフセット）及びデータサイズ（ブロックサイズ）を指定することにより I/O データ（メッセージ）を書き込みます。

(*2) DPRAM の入力データ領域にマッピングされていないメッセージデータの書き込みの場合も同様にメールボックスを介して内部メモリーの先頭アドレス（ブロックオフセット）及びデータサイズ（ブロックサイズ）を指定することによりメッセージデータ（メッセージ）を書き込みます。上記の例の場合、メールボックスを使用したメッセージデータの最大書き込みバイト数は 256 バイトである為､768 バイトの書き込みにはメールボックスを複数回使用して書き込みます。

3. Control Register

・ Anybus モジュールのバージョン、初期化パラメータ、フィールドバス・タイプとステータスなどの情報が含まれています。また、ウォッチドッグやイベント通知のレジスタも含んでいます。

・ 一般的に、制御レジスタ領域はアクセス前にアクセス権を確保する必要があります(Watchdog Counter 更新時にも必要)。ただし、モジュール初期化中は、 ソフトウェアのバージョン、フィールドバス・タイプ、モジュール・タイプなど、ハンドシェークを伴わないスタティック・データはアクセス権を確保しないで呼び出すことができます。

・ 以下参照（Parallel Interface Design Guide Anybus-S Slave & Master から抜粋)。

アドレス	領域	アクセス	注意
7C0h-7C1h	ブート・ローダー・バージョン	RO	
7C2h-7C3h	アプリケーション･インターフェース･ソフトウェア・バージョン(a)	RO	
7C4h-7C5h	フィールドバス・ソフトウェア・バージョン(a)	RO	
7C6h-7C9h	モジュール・シリアル番号	RO	独自シリアル番号
7CAh-7CBh	ベンダ ID	RO	メーカーID 番号（HMS,他）
7CCh-7CDh	フィールドバス・タイプ	RO	フィールドバス･タイプ識別子
7CEh-7CFh	モジュール・ソフトウェ・バージョン	RO	ソフトウェアレビジョン
7D0h-7D1h	(リザーブド)	-	
7D2h-7D3h	Watchdog カウンタ入力	R/W	アプリケーションに制御された Watchdog カウンタ
7D4h-7D5h	Watchdog カウンタ出力	RO	カウンタ、各 1ms でインクリメント
7D6h- 7D9h	(リザーブド)	RO	
7DAh-7DDh	LED ステータス	RO	各フィールドバス･ステータス表示の現在のステータス
7DEh-7DFh	(リザーブド)	-	
7E0h-7E1h	モジュール・タイプ	RO	モジュール･タイプ、マスター、スレーブ、その他
7E2h-7E3h	モジュール・ステータス	RO	ビット情報：フリーズ、クリア等
7E4h-7EBh	更新 データ･フィールド	RO	ビット･フィールド、出力データ内の変更表示 デュアル･ポート･メモリ内の領域
7ECh-7EDh	イベント通知発生要因	R/W	イベント発生レジスタ
7EEh-7EFh	イベント通知ソース	RO	イベント通知のためのコンフィグレーション･レジスタ
7F0h-7F1h	入力 I/O 長	RO	入力 I/O サイズ
7F2h-7F3h	入力 DPRAM 長	RO	デュアル･ポート･メモリ内の入力 I/O バイト数
7F4h-7F5h	入力合計長	RO	合計入力データ･サイズ
7F6h-7F7h	出力 I/O 長	RO	出力 I/O サイズ
7F8h-7F9h	出力 DPRM 長	RO	デュアル･ポート･メモリ内の出力 I/O バイト数
7FAh-7FBh	出力合計長	RO	合計出力データ･サイズ
7FCh-7FDh	(リザーブド)	-	

(a) : アプリケーション･インターフェース側のバージョン 2.00 前のモジュールではこのレジスタは予約済みで'0'です。

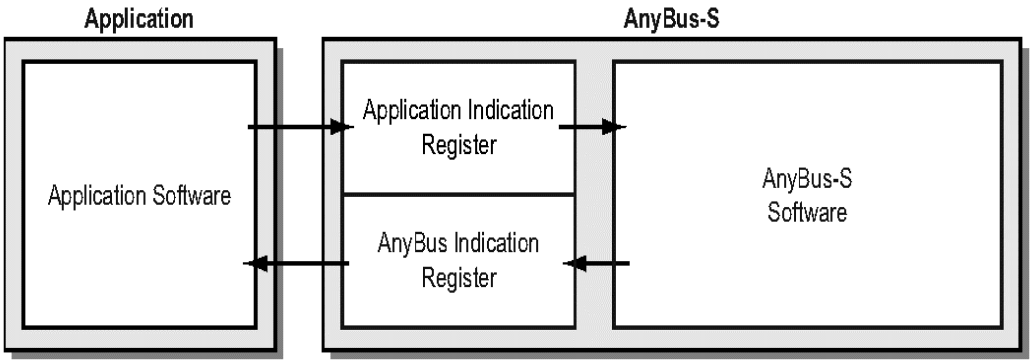
4. ハンドシェーク

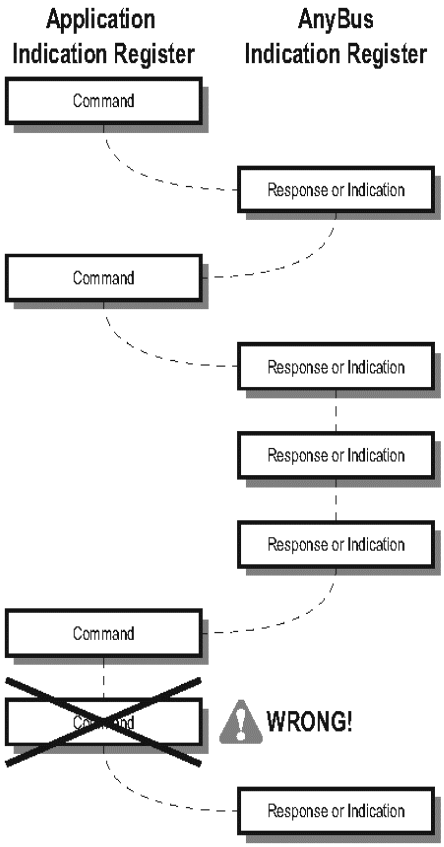
・ アプリケーションプログラムと Anybus モジュール間の通信は Indication Register を用いてハンドシェークを行います。

・ Indication Register はアプリケーション側、Anybus モジュール側双方に独立して実装されています。

・ アプリケーションはモジュールが前回のコマンドに対してレスポンスするまで新しいコマンドを送信することができません。

・ モジュールがレスポンスあるいは情報を送信する毎に、UPDATED ビットがトグルされ、ハードウェアの割り込みが発生します。





4.1. Indication Register

Application Indication Register (7FEh、RO)

- DPRAM アクセス権要求/解放 (リクエスト/リリース)。
- イベント承認。
- メールボックス・ハンドシェイク。

以下参照（Parallel Interface Design Guide Anybus-S Slave & Master から抜粋)。

B7	B6	B5	B4	B3	B2	B1	B0
AP_Min	AP_Mout	AP_EVNT	ACTION	LOCK	AP_IN	AP_OUT	AP_FBCTRL

ビット	機能	説明	デフォルト
AP_Min	メールボックス通知	メールボックス･メッセージを送信するために使用。8-2"メールボックス通知ビット"を参照して下さい。	0
AP_Mout	メールボックス通知	受診したメールボックスメッセージがある事を知らせるために使用されます。8-2"メールボックス通知ビット"を参照。	0
AP_EVNT	イベント承認	本ビットはイベント通知イベントが処理された事を知らせるためにトグルすべきです。6-2"イベント通知（ソフトウェア割込み）"を参照して下さい。	0
ACTION	領域リクエスト/リリース これらのビットはデュアル･ポート内に1つあるいは複数の領域のリクエスト/リリースするために使用されます。	このビットは現在の"ACTION"がリクエストかリリースかを表示します。 １：リクエスト ０：リリース 詳細は 4.3 項参照。	0
LOCK		このビットは"ACTION"がロックされているかいないかを表示します。 １：ロックされている ０：ロックされていない 詳細は 4.3 項参照。	0
AP_IN		このビットは入力データ領域を表します。 １：アクションはこの領域に影響します。 ０：アクションはこの領域に影響しません。	0
AP_OUT		このビットは出力データ領域を表します。 １：アクションはこの領域に影響します。 ０：アクションはこの領域に影響しません	0
AP_FBCTRL		このビットはフィールドバス個別領域と制御レジスタを表します。 １：アクションはこの領域に影響します。 ０：アクションはこの領域に影響しません	0

注意１：このレジスタにサイクリックにアクセスすることは推奨しません。入力されるイベントへのレスポンスとリクエストのときにのみ使用されるべきです。
注意２：このレジスタにアクセスする場合、シングル･メモリ･アクセスを使用し希望する操作に関連する全てのビットを変更することが重要です。そうでなければ、操作はモジュールによる複数の操作として誤解される可能性があります。

Anybus Indication Register (7FFh、R/W)

- DPRAM の各領域のアクセス権。
- イベント通知。
- メールボックス・ハンドシェイク。
- 初期化ステータス。

以下参照（Parallel Interface Design Guide Anybus-S Slave & Master から抜粋）。

B7	B6	B5	B4	B3	B2	B1	B0
AB_Min	AB_Mout	AB_EVNT	INIT	UPDATED	AB_IN	AB_OUT	AB_FBCTRL

ビット	機能	説明	デフォルト
AB_Min	メールボックス通知	モジュールはメールボックス･メッセージを受信した事を知らせるためにこのビットをトグルします。 8-2"メールボックス通知ビット"を参照して下さい。	0
AB_Mout	メールボックス通知	モジュールはこのビットをメールボックス送信領域で新しいメッセージが送信可能なことを示すためにトグルします。 8-2"メールボックス通知ビット"を参照して下さい。	0
AB_EVNT	イベント承認	モジュールは新しいイベント通知が発生するとこのビットをトグルします。	0
INIT	モジュール初期化	モジュールが初期化されるとこのビットが設定されます。 １：モジュールは初期化されいません ０：モジュールは初期化されています(a)	0
UPDATED	レジスタ更新	このビットは本レジスタの内容が変更される度にモジュールによりトグルされます。	0
AB_IN	これらのビットはデュアルポート内の各領域のオーナーを表示します。	このビットは入力データ領域を表します。 １：領域はアプリケーションにより所有されています ０：領域は Anybus により所有されています	0
AB_OUT		このビットは出力データ領域を表します。 １：領域はアプリケーションにより所有されています ０：領域は Anybus により所有されています	0
AB_FBCTRL		このビットはフィールドバス個別領域と制御レジスタを表します。 １：領域はアプリケーションにより所有されています ０：領域は Anybus により所有されています	0

(a)：フィールドバス･システムによっては、このフラグはフィールドバスが完全に初期化されデータ交換されるということを示さないことに注意して下さい。
詳細については個別の fieldbus appendix を参照して下さい。

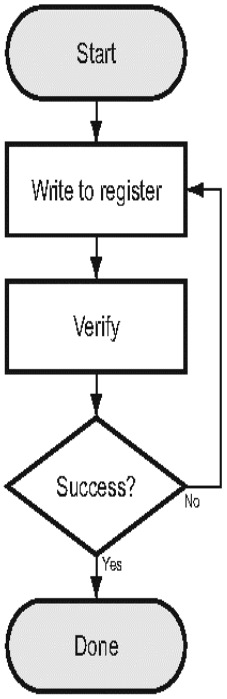
注意：このレジスタリードオンリーです。このレジスタに書き込みはしないで下さい。予測不可能な結果を引き起こします。

4.2. 衝突回避

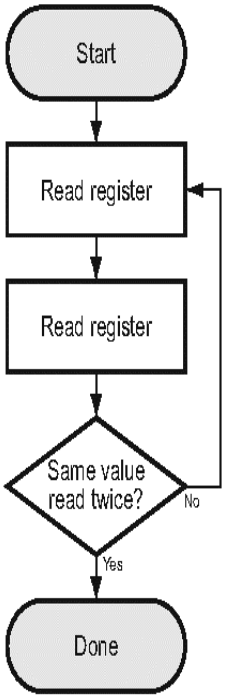
Indication Register は DPRAM に配置されており、アプリケーション、モジュール双方からアクセス出来るので衝突を回避する手段を講じる必要が有る。

- ２度読み/ ２度書き（右記フロー参照、必須）
- アプリケーションへの/IRQ シグナルのインプリメント(オプション、推奨)
- アプリケーションへの/BUSY シグナルのインプリメント (オプション、推奨)

Application Indication Register Access:



AnyBus Indication Register Access:



4.3. アクセス権確保と解放

・ 下記の DPRAM 領域は、アプリケーションとモジュールの衝突を防ぐために、各領域のアクセス権を確保してからアクセスする必要があります。そして、使用後は速やかにアクセス権を解放します。

– Input Data Area
– Output Data Area
– Control Register Area (Fieldbus Specific Area を含む)

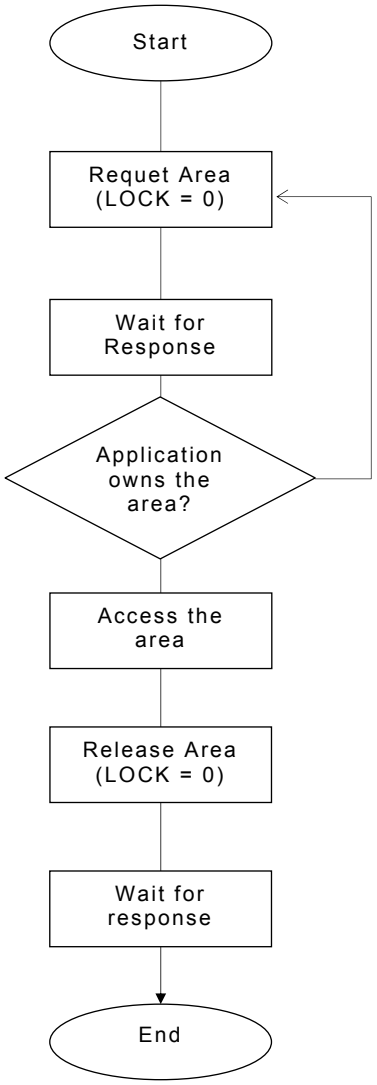
・ アクセス権の確保の方法には同期方式と非同期方式が有る。

・ 一般的にはアクセス権確保(Request)は同期方式、アクセス権解放(Release)は非同期方式の方がアプリケーションの負荷が少なくて高速に処理が出来る。

	Lock(=1)	Unlock(=0)
Request（アプリケーション側が DPRAM の領域確保を行いたい場合に使用）	アプリケーション側より一度 Lock Request(Action: Request, Lock=1) で DPRAM に対してアクセス要求を行なった場合 実際に領域が確保できた場合は ABS より割込みが発生してアプリケーション側に通知します。	アプリケーション側より Unlock Request(Action: Request, Lock=0) で DPRAM に対してアクセス要求を行なった場合、アプリケーション側は、“Anybus Indication Register”を読み出し、アクセス要求を出した DPRAM の領域がアプリケーション側に取得されたかどうか確認します。 もし、取得されていなければ再度 Unlock Request を出して、DPRAM の領域がアプリケーション側に取得されるまで要求を繰り返します。
Release（アプリケーション側が DPRAM の領域解放を行いたい場合に使用）(*1)	アプリケーション側より Lock Release (Action: Release, Lock=1) で DPRAM に対して領域解放を行なった場合、解放された領域は ABS 側で確保されます。 注記） Profibus 等、フィールドバスの種類によっては一旦 DPRAM の領域が ABS 側に確保された場合、プロセスデータに変化が生じない限り、ABS 側から確保された DPRAM の領域を開放しない場合があります。	アプリケーション側より Unlock Release (Action: Release, Lock=0) で DPRAM に対して領域解放を行なった場合、解放された領域はフリー（アプリケーション側からも ABS 側からも確保されない）状態となる。 よって、アプリケーション側より Unlock Release 直後に Request を出してアプリケーション側に領域確保を行なうことができます。

注記）
(*1) アプリケーション側から Lock Request、又は Unlock Request で領域を確保した後, Release 時は Lock Release でも Unlock Release でもどちらでも使用できます。Release 時は Unlock Release を使用することをお勧め致します。

4.4. 非同期方式のデータ交換



1. 領域へのアクセス・リクエストト (LOCK = 0)
アプリケーションはアプリケーション・インジケーション・レジスタ内で ACTION ビットだけでなく、その領域に該当するビットも設定します。

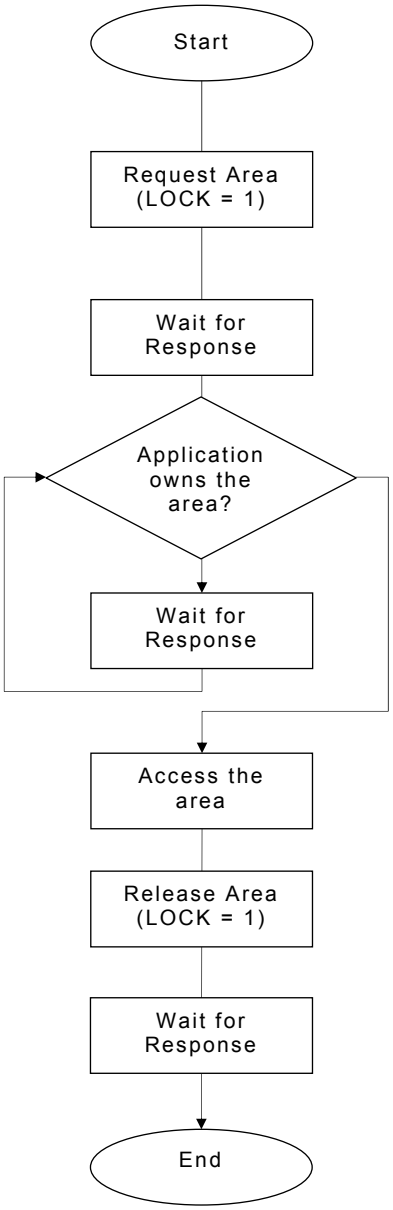
2. レスポンス待ち。

3. アクセス成功かどうかを見るためのレスポンスチェック要求が成功したか知るために、アプリケーションが Anybus Indication Register 内でオーナーシップ領域をチェックします。領域の所有権が Anybus 側で有れば、step1 に戻ります。

4. 領域のリリース (LOCK = 0)
アプリケーションは、アプリケーション・インジケーション・レジスタ内で ACTION ビットだけでなくその領域に該当するビットを設定します。

5. レスポンス待ち。

4.5. 同期方式のデータ交換



ロックドリクエスト

ロックリクエストは、領域がフリーになると直ぐにアクセス権を得る事を確実にします。

1. 領域へのアクセス・リクエスト (LOCK = 1)。

2. 最初のレスポンス待ち。
最初のレスポンスはアクセス・リクエストに対する応答です。

3. 領域のオーナーシップ・チェック
アプリケーションが所有 - 領域は自由に使用できます。ステップ 6 へ。Anybus が所有 - 領域は現在使用中です。ステップ 4 へ。

4. 追加レスポンス待ち
最初のレスポンス時に領域が確保されていなかった場合は、実際に領域が確保された時に再度レスポンスが発生します。

5. 領域のオーナーシップ・チェック
領域のオーナーシップはアプリケーションが所有しています。

6. アプリケーションから領域へのアクセス可能。

ロックリリース

領域をリリースするときに LOCK ビットを使用することにより、アプリケーションは Anybus モジュールのために領域をリザーブすることができます。

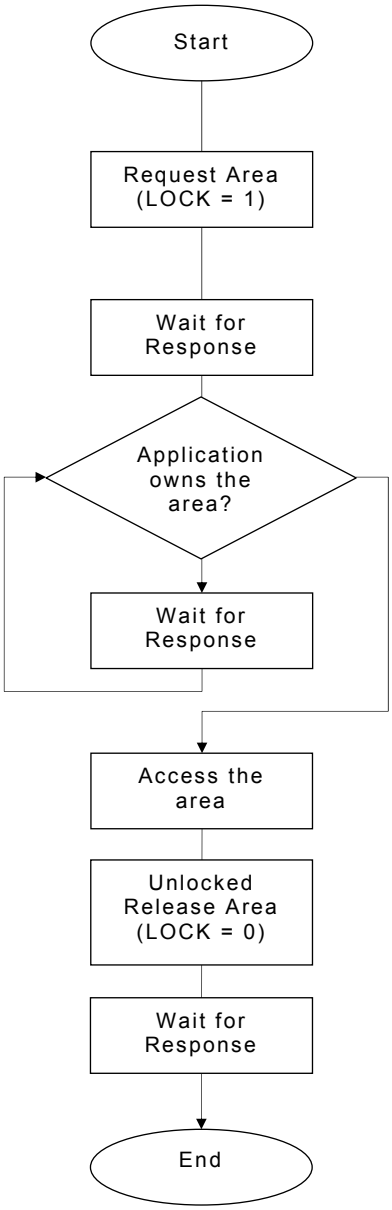
1. 領域のリリース (LOCK = 1)。

2. レスポンス待ち。

3. 終了。
領域は Anybus モジュールのためにリザーブしました。すなわちアプリケーションはモジュールがデータを更新するまで新たに領域を確保出来ません。

注意
出力データ領域に対してロックリリースを行った場合、出力データ領域の更新が行われないと、次回の出力データ領域のリクエストに対してレスポンスが帰らなくなりますので注意が必要です。

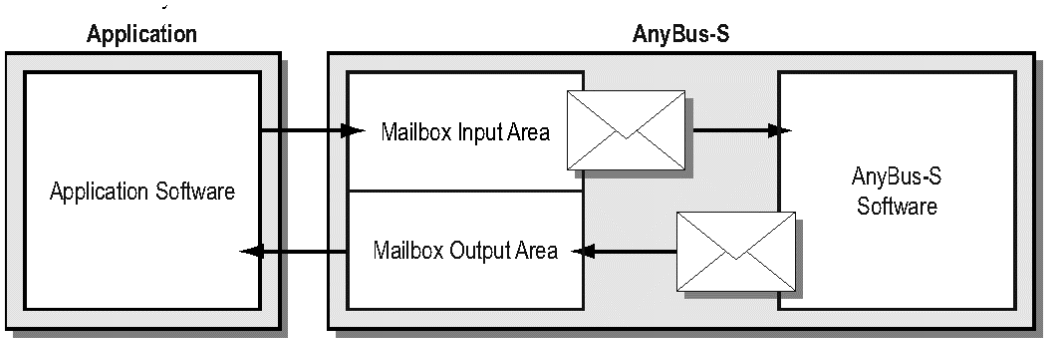
4.6. プログラム例



```
#define INDREG_MIN              0x80
#define INDREG_MOUT             0x40
#define APPINDREG_R_R           0x10
#define APPINDREG_L             0x08
#define INDREG_IN               0x04
#define INDREG_OUT              0x02
#define INDREG_FB_CTRL          0x01
#define APPIND_REQ_BITS         0x1F
unsigned char AbIndCopy;
unsigned char ProcessInData;
unsigned char ProcessOutData;

void ExchangeData( void )
{
   unsigned char AppIndCopy;
   unsigned char DoubleReadTemp;

   // Locked Request output area
   AppIndCopy  = XBYTE[ DPRAM_BASE + 0x07FE ];
   AppIndCopy &= ~APPIND_REQ_BITS;
   AppIndCopy |= APPINDREG_R_R;
   AppIndCopy |= APPINDREG_L;
   AppIndCopy |= INDREG_OUT;
   do {
      XBYTE[ DPRAM_BASE + 0x07FE ] = AppIndCopy;
   }
   while( XBYTE[ DPRAM_BASE + 0x07FE ] != AppIndCopy );

   // Response Wait
   do {
      while( ABINT ) {
      }
      DoubleReadTemp = XBYTE[ DPRAM_BASE + 0x07FF ];
      AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
      while( AbIndCopy != DoubleReadTemp ) {
         DoubleReadTemp = AbIndCopy;
         AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
      }
   }

   // Output Data Area の所有権が確保されるまでループ
   while( !( AbIndCopy & INDREG_OUT ) );
   // Output Data Area からプロセスデータを読み込み
   ProcessInData = XBYTE[ DPRAM_BASE + 0x0200 ];


   // Unlocked Release output area
   AppIndCopy  = XBYTE[ DPRAM_BASE + 0x07FE ];
   AppIndCopy &= ~APPIND_REQ_BITS;
   AppIndCopy |= INDREG_OUT;
   do {
      XBYTE[ DPRAM_BASE + 0x07FE ] = AppIndCopy;
   }
   while( XBYTE[ DPRAM_BASE + 0x07FE ] != AppIndCopy );

   // Response Wait
   do {
      while( ABINT ) {
      }
      DoubleReadTemp = XBYTE[ DPRAM_BASE + 0x07FF ];
      AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
      while( AbIndCopy != DoubleReadTemp ) {
         DoubleReadTemp = AbIndCopy;
         AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
      }
   }
   // Output Data Area の所有権が解放されるまでループ
   while( !( AbIndCopy & INDREG_OUT ) );
```

```
    // データ処理
    ProcessOutData = ~ProcessInData;


    // Locked Request Input Data Area
    AppIndCopy  = XBYTE[ DPRAM_BASE + 0x07FE ];
    AppIndCopy &= ~APPIND_REQ_BITS;
    AppIndCopy |= APPINDREG_R_R;
    AppIndCopy |= APPINDREG_L;
    AppIndCopy |= INDREG_IN;
    do {
       XBYTE[ DPRAM_BASE + 0x07FE ] = AppIndCopy;
    }
    while( XBYTE[ DPRAM_BASE + 0x07FE ] != AppIndCopy );

    // Response Wait
    do {
       while( ABINT ) {
       }
       DoubleReadTemp = XBYTE[ DPRAM_BASE + 0x07FF ];
       AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
       while( AbIndCopy != DoubleReadTemp ) {
          DoubleReadTemp = AbIndCopy;
          AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
       }
    }

    // Input Data Area の所有権が確保されるまでループ
    while( !( AbIndCopy & INDREG_IN ) );
    // Input Data Area へデータを書き込み
    XBYTE[ DPRAM_BASE + 0x0000 ] = ProcessOutData;
    // Unlocked Release Input Data Area
    AppIndCopy  = XBYTE[ DPRAM_BASE + 0x07FE ];
    AppIndCopy &= ~APPIND_REQ_BITS;
    AppIndCopy |= INDREG_IN;
    do {
       XBYTE[ DPRAM_BASE + 0x07FE ] = AppIndCopy;
    }
    while( XBYTE[ DPRAM_BASE + 0x07FE ] != AppIndCopy );

    // Response Wait
    do {
       while( ABINT ) {
       }
       DoubleReadTemp = XBYTE[ DPRAM_BASE + 0x07FF ];
       AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
       while( AbIndCopy != DoubleReadTemp ) {
          DoubleReadTemp = AbIndCopy;
          AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
       }
    }

    // Input Data Area の所有権が解放されるまでループ
    while( !( AbIndCopy & INDREG_IN ) );
}
```

5. メールボックス

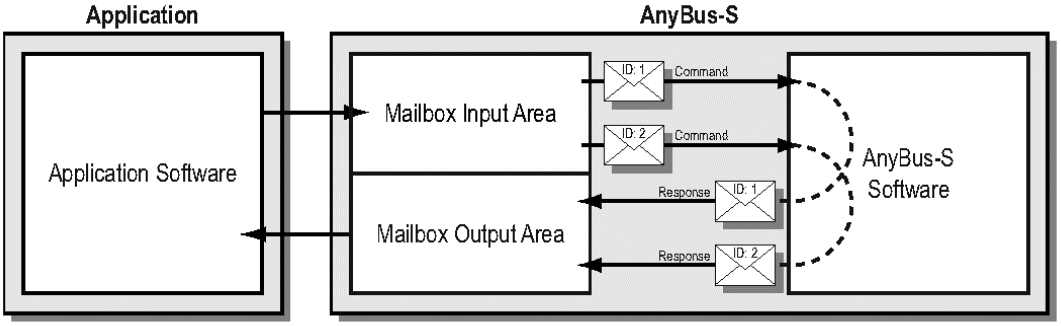


・ モジュールに特定タスクの命令やデータのリクエストをします。

・ モジュールによるイベントの表示や アプリケーションから送信されたリクエストにレスポンスします。

・ メールボックス通信はメールボックス入出力領域を通して処理され、フィールドバス・データ交換に影響しません（フィールドバスの設定をする場合を除く）。

・ メールボックス・インターフェースは次のタイプの通信をサポートしています。
– コマンド – レスポンス
– インジケーション



・ メールボックス・インターフェースはレスポンスを受信する前に複数のメッセージをモジュールに送信できるようにデザインされています。

・ コマンドとレスポンスのペアを区別するために独自の ID が各メッセージ/レスポンスに使用されます。

5.1. メールボックス のハンドシェーク

Anybus インジケーションレジストとアプリケーションインジケーションレジスタを使用してハンドシェークを行います。

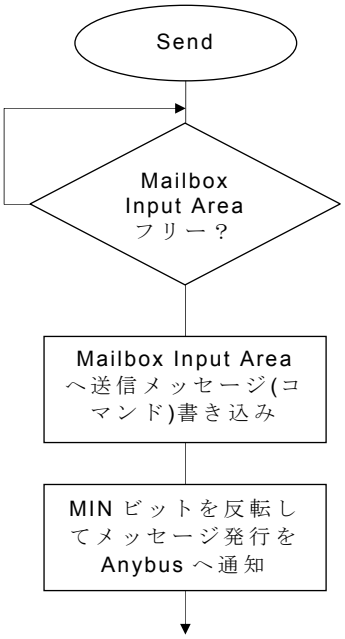
制御ビット

AP_MIN	メールボックス入力領域に書き込まれたメッセージを送信するためにトグルします。
AP_MOUT	メールボックス・メッセージが読まれたことを確認するためにトグルします。
AB_MIN	メールボックス・メッセージが読まれたとき Anubus モジュールによりトグルされます。
AB_MOUT	メールボックス出力領域で新しいメッセージが待機する毎にトグルされます。

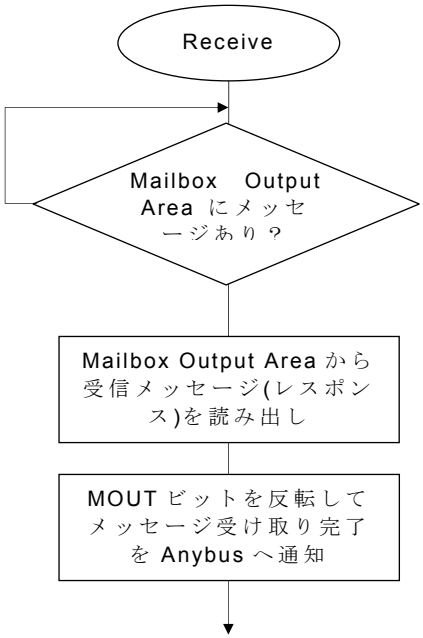
ハンドシェーク

AP_MIN ＝ AB_MIN	メールボックス入力領域がフリーです。
AP_MIN ≠ AB_MIN	メールボックス入力領域は現在使用中です。
AP_MOUT ＝ AB_MOUT	メールボックス出力領域では使用可能なメッセージはありません。
AP_MOUT ≠ AB_MOUT	メールボックス出力領域で新しいメッセージが使用可能です。

5.2. ハンドシェーク手順

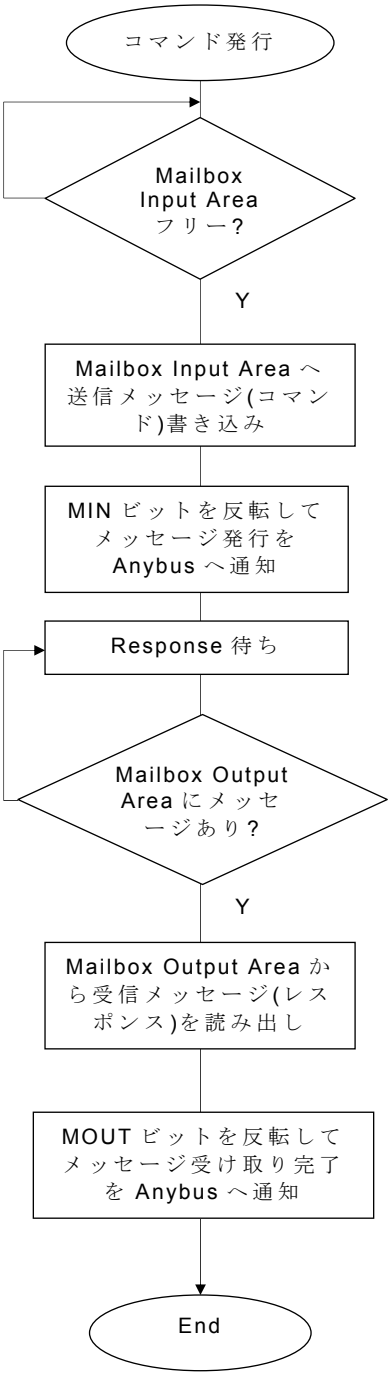


- メールボックス・メッセージの送信手順。
  1. メールボックス入力領域が空いていますかどうかチェックします。
  2. メールボックス入力領域へのメッセージの書き込み。
  3. アプリケーション・インジケータ・レジスタ（7FEh）の AP_MIN ビット をトグルします。



- メールボックス・メッセージの受信手順。
  1. メールボックス出力領域にメッセージが待っていないかチェックします。
  2. メールボックス出力領域からのメッセージの呼び出し。
  3. アプリケーション・インジケーション・レジスタ（7FEh）の AP_MOUT ビットをトグルします。

5.3. プログラム例



```
#define INDREG_MIN              0x80
#define INDREG_MOUT             0x40
#define MBH_INFO_ERR_MASK       0x80
unsigned char AppIndCopy;
unsigned char AnybusInitResponse[16];
unsigned char AnybusInitDataResponse[18];
unsigned char CommandExample( void )
{
   unsigned char AbIndCopy;
   unsigned char DoubleReadTemp;
   // Message Input areaはフリー?
   while ( ( ( AbIndCopy ^ XBYTE[ DPRAM_BASE + 0x07FE ] ) & INDREG_MIN ) ) {
      do {
         while( ABINT ) {
         }
         DoubleReadTemp = XBYTE[ DPRAM_BASE + 0x07FF ];
         AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
         while( AbIndCopy != DoubleReadTemp ) {
            DoubleReadTemp = AbIndCopy;
            AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
         }
      }
   }

   // Message Input areaにコマンドを書き込み
   for( Byte = 0; Byte < 16; Byte++ )
   {
      XBYTE[ DPRAM_BASE + 0x0400 + Byte ] = AnybusInit[ Byte ];
   }
   for( Byte = 0; Byte < 18; Byte++ )
   {
      XBYTE[ DPRAM_BASE + 0x0420 + Byte ] = AnybusInitData[ Byte ];
   }

   // MINビットを反転してコマンドを発行
   AppIndCopy  = XBYTE[ DPRAM_BASE + 0x07FE ];
   AppIndCopy ^=  INDREG_MIN;
   do {
      XBYTE[ DPRAM_BASE + 0x07FE ] = AppIndCopy;
   }
   while( XBYTE[ DPRAM_BASE + 0x07FE ] != AppIndCopy );
```

```
   // Response 待ち
   do {
      while( ABINT ) {
      }
      DoubleReadTemp = XBYTE[ DPRAM_BASE + 0x07FF ];
      AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
      while( AbIndCopy != DoubleReadTemp ) {
         DoubleReadTemp = AbIndCopy;
         AbIndCopy      = XBYTE[ DPRAM_BASE + 0x07FF ];
      }
   }
   // Mailbox Output Area に有効なメッセージが有ればループを抜ける
   while( !( ( AbIndCopy ^ XBYTE[ DPRAM_BASE + 0x07FE ] ) & INDREG_MOUT ) );

   // Message Output area からメッセージ(レスポンス)を読み出し
   for( Byte = 0; Byte < 16; Byte++ ) {
      AnybusInitResponse[ Byte ] = XBYTE[ DPRAM_BASE + 0x0520 + Byte ];
   }
   for( Byte = 0; Byte < 18; Byte++ ) {
      AnybusInitDataResponse[ Byte ] = XBYTE[ DPRAM_BASE + 0x0540 + Byte ];
   }

   // MOUT ビットを反転してレスポンスを受け取った事を Anybus へ通知
   AppIndCopy  = XBYTE[ DPRAM_BASE + 0x07FE ];
   AppIndCopy ^=  INDREG_MOUT;
   do {
      XBYTE[ DPRAM_BASE + 0x07FE ] = AppIndCopy;
   }
   while( XBYTE[ DPRAM_BASE + 0x07FE ] != AppIndCopy );

   // レスポンスデータのエラー判定
   if( AnybusInitResponse[ 2 ] & MBH_INFO_ERR_MASK )
   {
      return( 0 );
   }
   return(1);
}
```

6. スタートアップ・シーケンス

・ 電源投入時／ハードウェアリセット時は Anybus モジュールの起動を確認してから DPRAM にアクセスする必要があります。

・ アプリケーションの実装の仕方によって、スタートアップ手順は異なります。

ハードウェア割り込みが実装されていない場合

パワーオン/ リセットの後、アプリケーションは、モジュールによって適切に更新されていますことを検出するため、Watchdog Counter Out レジスタ (7D4h -7D5h) を約 10ms 周期でポーリングします。最低 10 回このレジスタがモジュールによって更新されればモジュールは動作可能です。

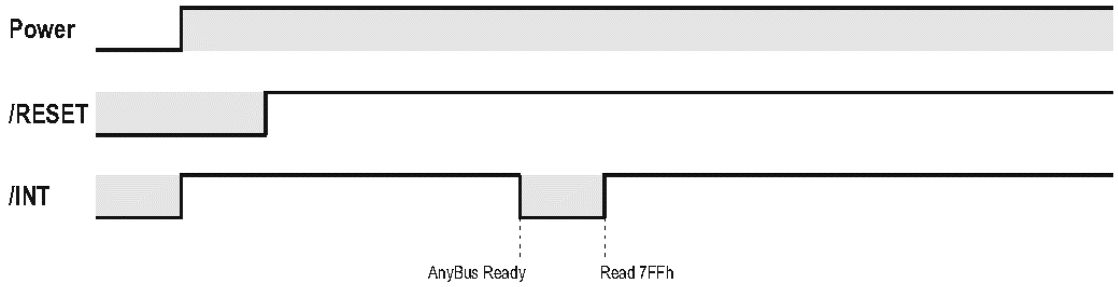
ハードウェア割り込みが実装されている場合

パワーオンリセットの後、Anybus モジュールは割り込みによって動作可能であることを示します。割り込みの前、アプリケーションは DPRAM にデータを書き込むことができません。

このロジックはハードウェア・リセットではなくパワーオンリセットによって開始されます。そのため適切な機能を保証するために注意が必要です。詳細については Parallel Interface Design Guide Anybus-S Slave & Master の「C-1 割り込み信号とハードウェア・リセット」 を参照して下さい。

Normal Power On-Reset

The interrupt line (/INT) is initialized to a high level after a power-on reset.



7. 初期化例

多くのモジュールでは下記の3つのメールボックスコマンドを実行するだけで基本的な I/O 通信の為の初期化が可能です。

1. Start_init
2. Anybus_init
3. End_init

```
START_INIT
      0x00, 0x01,
      0x40, 0x01,
      0x00, 0x01,
      0x00, 0x00,
      0x00, 0x01,
      0x00, 0x01,
      0x00, 0x00,
      0x00, 0x00

Anybus_INIT
      0x00, 0x0A,     //
      0x40, 0x01,     //
      0x00, 0x02,     //
      0x00, 0x12,     //
      0x00, 0x01,     //
      0x00, 0x01,     //
      0x00, 0x00,     //
      0x00, 0x00,     //
      0x00, 0xc8,     // Input I/O Length
      0x00, 0xc8,     // Input DPRAM Length
      0x00, 0xc8,     // Input Total Length
      0x00, 0xc8,     // Output I/O Length
      0x00, 0xc8,     // Output DPRAM Length
      0x00, 0xc8,     // Output Total Length
      0x00, 0x00,     // Operation Mode
      0x00, 0x00,     // Event Notification Config.
      0x00, 0x00      // Watchdog Timeout Value

END_INIT
     0x00, 0x01,
     0x00, 0x01,
     0x00, 0x03,
     0x00, 0x00,
     0x00, 0x01,
     0x00, 0x01,
     0x00, 0x00,
     0x00, 0x00
```

フィールドバス個別の設定は Anybus モジュールのデフォルト設定になります。一般的にはユーザー仕様及びコンフォーマンステストの為にフィールドバス個別のコマンドを使用して設定をする必要があります。

8. 開発環境

AnyBus-S Evaluation Board

- 8051 ベースのマイクロコントローラの環境。
- 統合された AnyBus- S のスロット。
- オンボード LCD ディスプレイ。
- パラレルとシリアル両方の AnyBus- S インタフェースをサポート 。
- C 言語のサンプルソフトウェア。
- PC 上で動作する Windows ベースのモニターツール(Keil を利用可能)。
- すべて AnyBus- S インターフェイスに使用可能。

8.1. サンプルプログラム

- 評価ボードにはスレーブ用とマスター用のサンプルプログラムが付属しています。
- スレーブ用のサンプルプログラムは、割り込みを使用したサンプルプログラムと、割り込みを使用しないサンプルプログラムが付属しています。
- フィールドバスの区別は無くて、Anybus モジュールの初期化は Anybus_Init コマンドのみで行っており、I/O サイズは入出力共に 2 バイトに設定しています。
- フィールドバス固有の初期化コマンドは実装していません。但し、単純にプロセスデータのデータ交換を行うだけで有れば、多くのフィールドバスでデフォルト値のままで動作します。
- 各関数の大まかな説明。

– WaitAnybusStart() 電源投入時の Anybus モジュール起動待ち
– WaitAnybusResponse() コマンド送信後の Anybus からの応答待ち
– WriteAppInsBits() アプリケーションインジケータレジスタの更新
– RequestReleaseArea() エリアの確保／解放要求
– InitialiseAnybus() Anybus モジュール初期化（Anybus_Init）
– ExchangeData() プロセスデータへのアクセス
– main() main 処理

以上