ＦＣＡＤ４１６－ＤＳＵＢ
取扱説明書

ＵＳＢ２．０ハイスピード対応

バッファメモリ内蔵

高速アナログ変換ユニット

２０１３年2月　ECO20121126からの修正(ECO20130110参照)ドライバーバージョン１．０．６とする
２０１２年１１月　不要関数見直しにかかる修正(ECO20121126参照)
２０１１年１１月　デバイスドライバー整理及び６４ビットインポートライブラリ名称変更（１．０．５）
２０１１年１０月　動作確認プログラム追加機能（リアルタイム描画機能）説明を追加
２０１１年4月２０日　ＬａＢＤＡＱ対応に際し記載内容の見直し等を行う
２０１１年1月１６日　デバイスドライバーをWindows7対応とし、同時に６４ビット対応とする（１．０．４）
２０１０年7月１０日　デバイスドライバーを改変し、そのレビジョンを１．０２とする
２０１０年7月７日　説明内容見直しによる変更
２０１０年6月１０日　外部クロック動作仕様見直しによる変更
２００９年１２月７日　ベンダーリクエスト説明書を追加
２００９年8月２６日　トリガ設定に関するシーケンス記述ミスを修正（74,75ページ）
２００９年8月２０日　４－５－【7】以降一部ページ番号の誤りを修正
２００９年8月７日　サンプルプログラム内容追加修正に伴う修正
２００９年6月１０日　Ｗｉｎｄｏｗｓ７ＲＣ上で動作確認
２００９年6月１日発行　第1版

アプリケーション応用例
（６４チャンネル　マスタースレーブ動作例）

ユニット内部

《目　次》

本製品の使用・適用についての注意事項

（1）　本製品は、ＩＢＭＰＣ／ＡＴ互換機のＵＳＢポート、または同パソコンのＵＳＢポートに接続された、セルフパワーハブに接続して使用するものです。

（2）　本製品が組み込まれたシステムの運用対象・方法・場所・環境等によって、故障・誤動作等が生じた場合に起こり得る、身体・生命・財産等に対する損害の回避処置は同システムの設計・製作に別途付加・反映させて下さい。本製品自体には、前述の機能はなく、従って弊社では本製品が組み込まれたシステムの運用により発生した故障・誤動作・事故に起因する身体・生命・財産等の損害に対する責任は負えません。これは本製品の故障・誤動作が原因となった場合も含み、理由の如何を問いません。

（3）　本製品付属のソフトウェアは本製品利用の方法を示す例、またオプションの関連ソフトウェアは本製品利用の一般的便宜をはかるものであり、現在未発見のバグ存在の可能性も含めて、運用結果についての責任は一切負えません。
これらのソフトウェアには自身が組み込まれたシステムに故障・誤動作・事故等が生じた場合に起こり得る身体・生命・財産等に対する損害の回避機能はありません。御利用の場合は同システムの設計・製作で配慮・付加・反映させて下さい。

（4）　本製品（付属ソフトウェアを含む）、及びオプションの関連ソフトウェアは医用・航空機器用・その他高信頼性・高安全性を必要とするシステムに使用しないで下さい。

（5）　本製品付属のソフトウェアについて弊社は著作権を保持しますが、第３者の権利を侵害しない限りにおいて購入者は自身が製作するシステム等に自由に組み込み、販売する事もできます。但し、弊社製ソフトウェアのソースコードを含むソフトウェアを第３者に販売・移転するときは弊社の文書による事前許可を必要とします。

（6）　弊社では、本製品の販売・サポート・保証の範囲を日本国内に限っています。

（7）　本製品は改良のため仕様変更、価格改定を行うことがあります。

故障・修理・サポート方法について

（1）　納入後１年間は自然故障、及び弊社製造上の問題に起因した事が明らかな故障製品に対して無償修理を行います。

（2）　保証期間中であっても、次の場合は有償修理となります。

ア　取扱上の不注意、誤用による故障および損傷
イ　弊社以外での修理、改造、分解掃除等による故障および損傷
ウ　泥、砂、水などのかぶり、落下、衝撃等が原因で発生した故障および損傷
エ　火災、地震、水害、落雷その他の天変地異、公害や異常電圧による故障および損傷
オ　保管上の不備（高温多湿の場所等）や手入れの不備による故障
カ　接続している他の機器に起因して生じた故障
キ　その他使用者側の責に帰する故障

（3）　修理は宅配便によるセンドバックで行います。尚運賃は互いに発送する側が負担するものとします。
　出張修理は行っておりません。簡単な故障であれば一週間程度で修理・返却が可能ですが、故障状況によっては更に日数を要します。

（4）　本製品使用上の質問・トラブル対応・故障修理などは入手経路の如何にかかわらず、弊社宛に直接御相談下さい。その際は、客観情報の整理・評価を行うために必ずレポートを御送付下さい。特にEmailで情報を頂く事で解決速度が大幅に向上する事が期待できます。（support@flexcore.jp）また、内容が複雑になっている場合はfax使用も有効です。

（5）　有償修理の場合の修理費用は、基本料金￥８，０００円＋部品代となります。簡単な故障では凡そ1万円以下程度だと思われます。修理費用限度額がある場合は、お申し付け頂ければ超過する場合に御連絡致します。

（6）　修理品送付先

〒301-0853
茨城県龍ヶ崎市松ヶ丘３丁目１８番地３
フレックスコア
品質管理部　林
E-mail: support@flexcore.jp
Fax:050-3488-3354

ＦＣＡＤ４１６－ＤＳＵＢ仕様一覧

アナログ入力部	
入力数（ソフト選択）	１６チャンネルシングルエンド入力/８チャンネル差動入力 各入力ラインに高インピーダンス信号源対応のためバッファアンプ実装
入力接続（ソフト選択）	チャンネル毎に、実入力チャンネルを自由に接続可能
入力範囲（ソフト選択）	±１０．２４Ｖ／±５．１２Ｖ／±２．５６Ｖ／±１．２８Ｖ(混合は不可)
入力保護機能	絶対最大定格　±３５Ｖ
入力インピーダンス	１０MΩ（１０MΩ抵抗にて終端)
AD変換部【注】正確度：内部雑音を含まず	
分解能	１６ＢＩＴ
単Chサンプリング速度	最高２５０KHｚ（最低３０５Hｚ）
複Chサンプリング速度	最高４ｕＳ×Ｃｈ数（最低３．２７６ｍＳ×Ｃｈ数)
非直線性	±０．００４%ＦＳｍａｘ
正確度	±０．０２%ＦＳ／常温で製造調整時
内部雑音	±４ＬＳＢ　Type（対全入力レンジ、弊社製造システムにて)
温度ドリフト	±１０ｐｐｍ／℃　Type
ADデータ・コード	バイナリ、または2の補数（ソフト指定)
制御部・その他	
クロック	内部２０MHｚ／内部１６．３８４MHｚ／外部ＴＴＬ入力（1MHｚ最大)
分周機能	３２ＢＩＴプログラマブルカウンタ（バイナリ)
トリガ条件	
内部トリガ	プログラム上からの即トリガ アナログ入力（スキャン先頭チャンネル）の指定エッジ、レベル、またはレンジ ポストトリガ時は、スキャン先頭チャンネルのみを最高速度で連続監視 プリトリガ時は全チャンネルを指定時間間隔でサンプリングしながら先頭チャンネルを監視、何れの場合も、実際の先頭チャンネルは、入力接続機能により任意のチャンネルを指定可能
外部トリガ	外部ＴＴＬ入力の指定エッジによりサンプリングを開始、サンプリング停止は指定サンプリング数、或いは外部からの操作による。プリトリガ・ポストトリガ何れも可能
バッファメモリ	８M語ＦＩＦＯ（プリトリガ時、ソフトウェア設定により１M語～７M語の範囲をリングバッファとしてトリガ検出前データの保存が可能)
ADデータ転送	ＵＳＢバスを経由し、サンプリング済みデータを継続的に取得、プリトリガバッファについては、サンプリング終了後ＦＩＦＯデータを読み出した後に取得
デジタル入出力	４ビット汎用ＴＴＬ入力、４ビット汎用ＴＴＬ出力、クロック出力、同期信号出力、制御入力、マスタースレーブ方式による複数台（最大スレーブ１５台）同期サンプリング動作可能
ユニット寸法	２０５mm×１６０mm×５１mm（W×D×H）／突出部を含まず
ユニット質量	５２０ｇ、導電・電磁シールド対策済み樹脂ケース採用
動作環境	ＵＳＢ２．０ハイスピード（４８０Mビット／秒）、周囲温度：０～＋４０℃（結露しないこと）、保存温度：－１０～＋８０℃（結露しないこと）
電源消費	＋５Ｖ、０．４５A_{Max}（ＵＳＢバスパワー動作)
製品構成	本体ADユニット（ＲｏＨＳ対応）、入出力用プラグ（ＲｏＨＳ対応）、（ＷｉｎｄｏｗｓＸＰからＷｉｎｄｏｗｓ７及びWindowsServer２００８Ｒ２用のドライバー／制御関数Ｄｌｌライブラリ、添付ソフト、（全て３２/６４ビット対応）、取扱説明書ＰＤＦファイルは、ホームページからのダウンロードにより提供)
推奨ケーブル	ＡＥＣ－UM2（ＵＳＢ２．０ケーブル、ケーブル長2m、ＡＣＲＯ‘S製）又は同等品（弊社出荷試験では、ケーブル長５mのＵＳＢ２．０ケーブルを採用しています)

ＦＣＡＤ４１６－ＤＳＵＢユニットの構成

ＤＳＵＢコネクタ仕様

付属品　アナログ入力コネクタキット（17JE-23370-02(D8A)-CG　コネクタヘッダ、コネクタハウジング）

デジタル入出力コネクタキット（DX40-36P(55)　コネクタヘッダ、DX-36-CV1　コネクタハウジング）

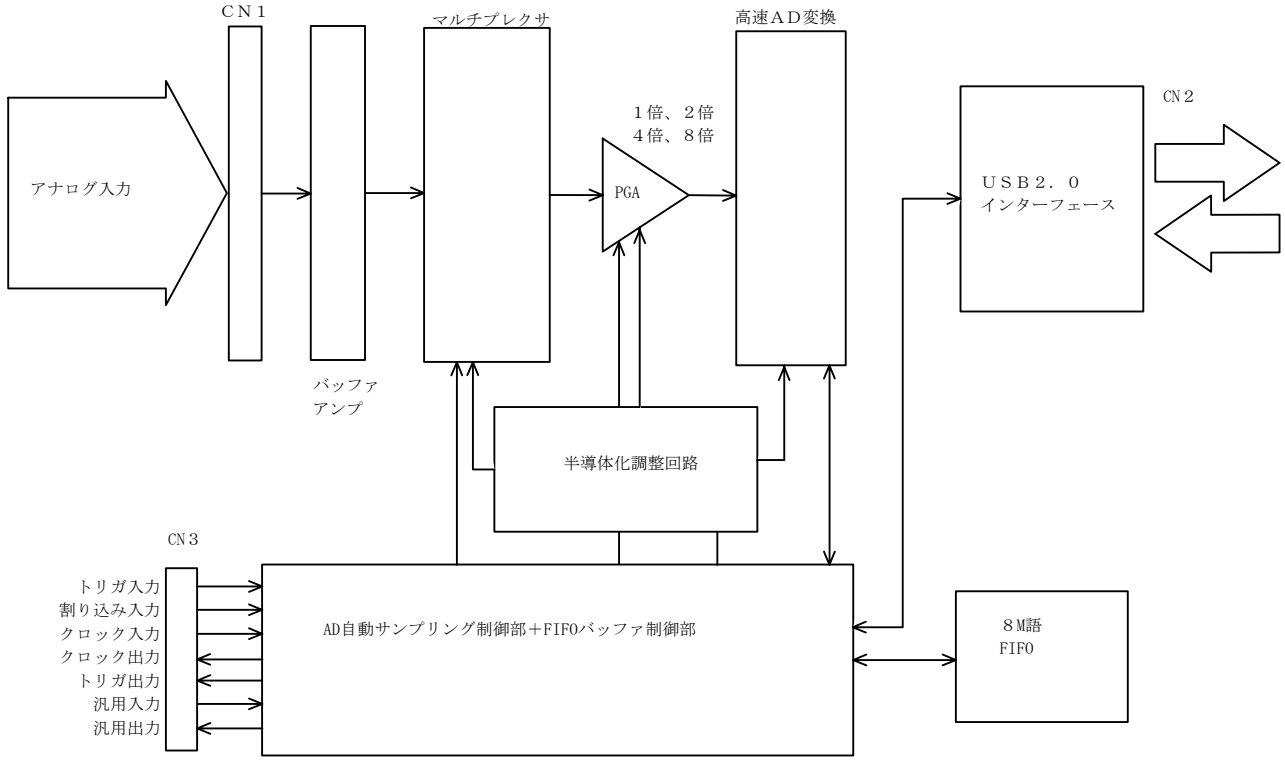
第１章. 導入・試運転

１－１. 本製品の概要

ＦＣＡＤ４１６－ＤＳＵＢはポストトリガ、プリトリガの２系統に運用できるバッファメモリを搭載した、１６チャンネル逐次サンプリングＡＤユニットです。
内部に８M語のメモリを持ち、ソフトウェアの設定によって８M語全てをＦＩＦＯとするモード（ポストトリガモード）及び１M語～７M語のリングバッファＲＡＭと残りをＦＩＦＯとするモード（プリトリガモード）の２種類を使用することができるため、種々の応用が可能です。

- １６チャンネルシングルエンド入力又は８チャンネル差動入力モード
- ±１０．２４Ｖ～±１．２８Ｖまで４種類の入力感度を実現
- ＷｉｎｄｏｗｓＸＰ／Ｖｉｓｔａ共用のデバイスドライバー、ハンドラー関数ライブラリＤｌｌ（３２ビット版）
- 最高２５０ＫＨｚサンプリング（単チャンネル、複数チャンネルでは最高４μＳ×チャンネル数）
- 最長３．２７６ｍＳ周期スキャン可能
- 最大１５台のスレーブユニットと同期運転可能（プリトリガ・ポストトリガ共に対応）
- マルチ・クロック源：内部２０MHｚ、内部１６．３８４MHｚ、外部ＴＴＬ入力（最大１MHｚ）
- マルチ・トリガ源：ソフトトリガ（即トリガ）、アナログ（エッジ・レベル・レンジ）、外部ＴＴＬエッジ入力
- トリガ前のデータも得られるプリトリガ機能
- プリトリガバッファＲＡＭ（最大７M語、最低１M語）にも分割使用が可能なＦＩＦＯバッファ（８M語）搭載
- 外部停止制御：外部停止制御エッジ入力
- ＵＳＢ通信を経由して各ユニットへのサービスリクエストを検出し、メッセージを生成する事で、ある程度の時系列的な応答が可能
- 半導体化調整回路及び高精度部品の使用により、入力範囲切り替え時の再調整不要
- ＲｏＨＳ対応設計
- 合成樹脂ケース（導電、電磁シールド対策済み）使用（ＲｏＨＳ対応）

図１　ＦＣＡＤ４１６－ＤＳＵＢ機能ブロック



１－１－１　インストールパッケージの構造

今回、Windows7　64 ビット環境までを対応ＯＳとしたため、ソフトウェアパッケージを大幅に見直しています。全体の構成は、以下のようになっています。
基本的に、ドライバーパッケージ部分が、ＯＳの種類によって細分化された事が、これまでのパッケージとの相違点になります。また、インポートライブラリも、それぞれに付属しています。また、これらＯＳの内、ハードウェア・ウィザードによるインストールを行うもの（Windows2000、WindowsXP32 ビット版及び WindowsServer2003　32 ビット版）については、旧来の個別パッケージ（sys ファイル、dll ファイル、cat ファイル及び inf ファイル）を用意しています。更に動作確認プログラムも x86 版、x64 版の両ＯＳ対応版を揃えました。

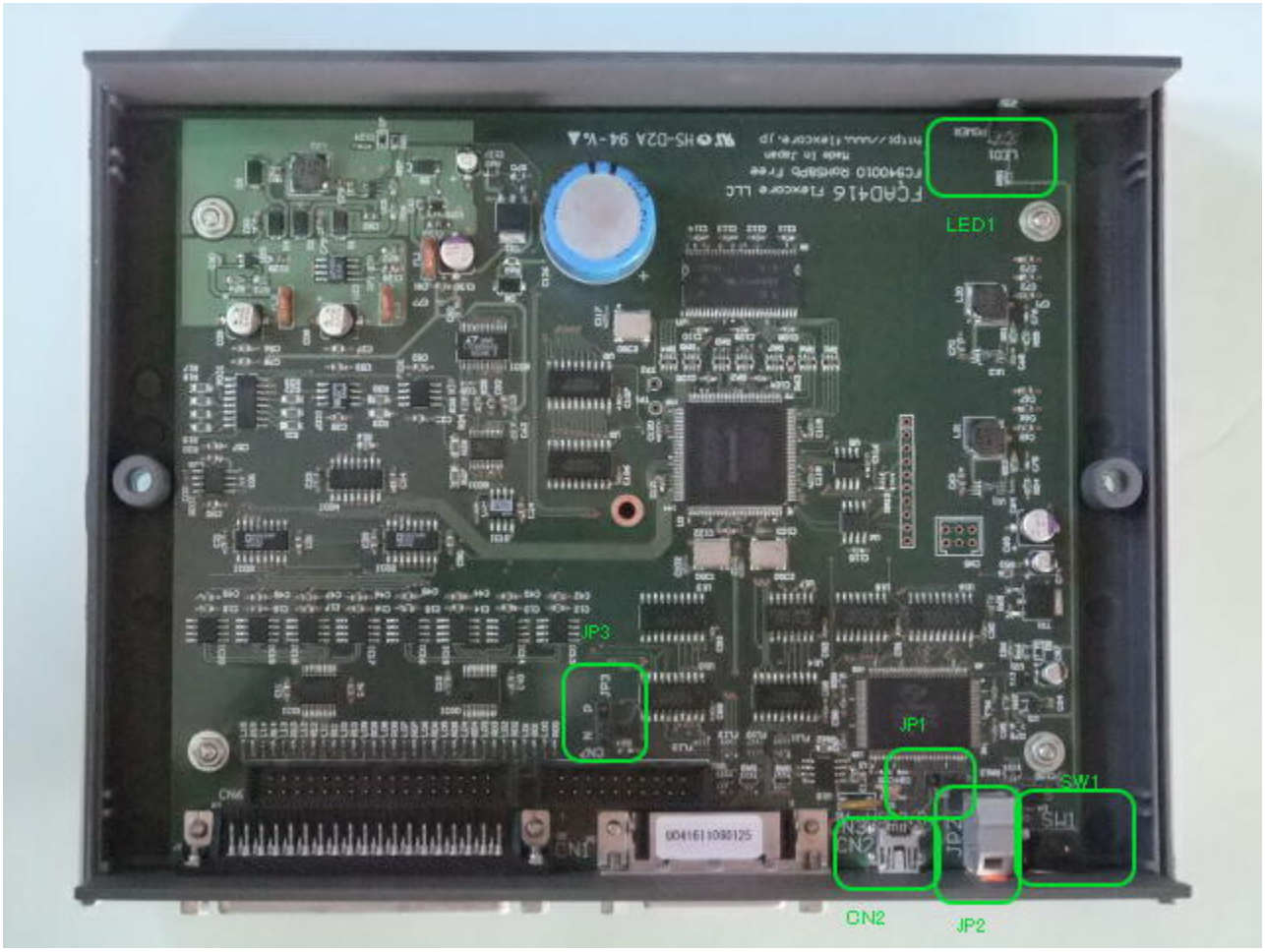
```
Flexcore（トップフォルダ）
 ├----Fcad416
 │    ├----C++Builder2009
 │    ├----CBuilderPlus10
 │    ├----Delphi2009
 │    ├----InstFcad416
 │    │    ├----w2k
 │    │    │    └----x86      個別ファイル形式ドライバー,fcad416.lib
 │    │    ├----wlh
 │    │    │    ├----x64      fcad416_64-Wlh.msi   ,fcad416_64.lib
 │    │    │    └----x86      fcad416_86-Wlh.msi   ,fcad416.lib
 │    │    └----wxp
 │    │         ├----x64      fcad416_64-Wxp.msi   ,fcad416_64.lib
 │    │         └----x86      個別ファイル形式ドライバー,fcad416.lib
 │    ├----LABVIEW-SAMPLEfor8.6J
 │    ├----Tools
 │    ├----VB6
 │    ├----VB2008Plus
 │    ├----VC6Plus
 │    ├---VC2008Plus
 │    ├----動作確認
 │         ├----x64   VC6Plus_64.exe
 │         └----x86   VC6Plus.exe
```

１－１－２　対応ＯＳ一覧

ドライバーパッケージと、Windows 各ＯＳとの関係は、以下の表に示す関係になっています。
ここで、Windows2000 の６４ビットと、WindowsServer2008R2 の３２ビットは、ＯＳが存在しないため、該当パッケージは存在しません。また、Windows2000、WindowsXP(32 ビット)及び WindowsServer2003(32 ビット)については、個別ファイルによるインストール（注１）となります。詳細については、１－５－２項　WindowsXP での操作を参照して下さい。

Windows バージョン	32 ビット	64 ビット
Windows2000	注１	該当ＯＳなし
WindowsXP	注１	fcad416_64-Wxp.msi
WindowsServer2003	注１	fcad416_64-Wxp.msi
WindowsVista	fcad416_86-Wlh.msi	fcad416_64-Wlh.msi
WindowsServer2008	fcad416_86-Wlh.msi	fcad416_64-Wlh.msi
Windows7	fcad416_86-Wlh.msi	fcad416_64-Wlh.msi
WindowsServer2008R2	該当ＯＳなし	fcad416_64-Wlh.msi

１－２．ユニット上の設定

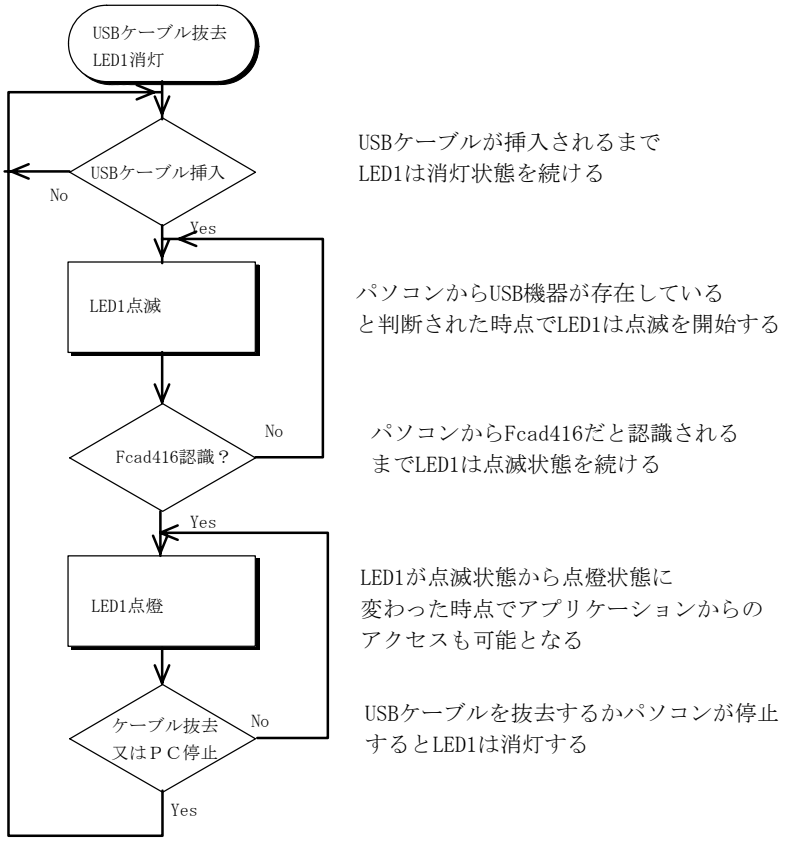


・ＪＰ１　　フレームグランドと回路グランドの接続/遮断切り替え【出荷時：接続】
・ＪＰ２　　フレームグランド外部接続用ターミナル【出荷時：無接続】
・ＪＰ３　　汎用デジタル出力の極性選択【出荷時：Ｎ（負論理）】
・ＳＷ１　　ユニット番号設定スイッチ【出荷時：０】
・ＣＮ１　　アナログ入力コネクタ（３７ピンＤ－ＳＵＢ）
・ＣＮ２　　ＵＳＢミニＢコネクタ
・ＣＮ３　　デジタル入出力コネクタ（３６ピン・ハーフピッチ）
・ＬＥＤ１　　回路動作表示用ＬＥＤ

ＪＰ１は、回路グランドと筐体フレームグランドとを接続するかどうかを切り替えるためのジャンパーです。
出荷時は接続状態となっていますが、実動作時にオープンとしたほうが良い場合も考えられるため、このような構成になっています。
ＪＰ２は、本ユニットのフレームグランドを外部と接続する際に使用するターミナルです。
ＪＰ３は、汎用デジタル出力の極性を選択するジャンパーで、出荷時には負極性（１出力が０Ｖ出力）に設定してあります。
ＳＷ１は、本ユニットのＩＤを設定するためのロータリースイッチで出荷時には“０”に設定しています。
ＣＮ１は、アナログ入力接続用のＤＳＵＢコネクタ（３７ピン）です。
ＣＮ２は、ＵＳＢケーブルを接続するミニＢタイプのコネクタです。
ＣＮ３は、本ユニットと外部を接続するデジタル信号を受け持つコネクタです。

１－２－１　ＬＥＤ１

ＬＥＤ１は、本ユニットの動作状態をモニターするための表示器です。本ユニットが正常動作を行っているかどうかは、このＬＥＤが点燈状態かどうかで判断できます。本ユニットがパソコンから認識されるまでは、このＬＥＤは消灯しています。そしてパソコンからＵＳＢ機器として認識されると点滅を開始します。そして、パソコンからＦＣＡＤ４１６－ＤＳＵＢとして認識されると点燈状態に遷移します。これ以降パソコンをシャットダウンするかＵＳＢケーブルを外すまでの間、ＬＥＤ１は点燈状態を保持し続ける事で、本ユニットが使用可能状態である事を示します。ＵＳＢケーブルを取り外すと、本LEDは消灯し、再挿入することで、数秒の待ち時間の後、再度点滅状態から動作を再開します。



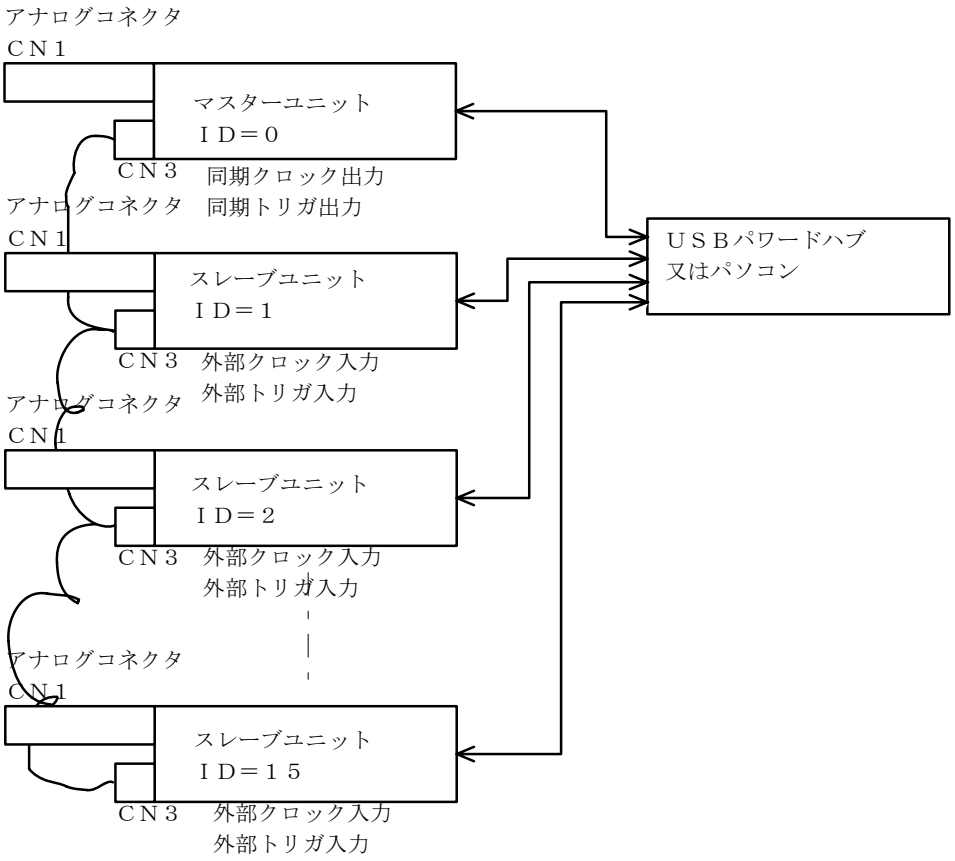
※　本ユニットはＵＳＢバス製品なので、活線挿抜にも対応しています。ユニット運用中にＵＳＢケーブルを挿抜する必要が生じた場合は、サンプリング中であった場合一旦サンプリングを停止していただく必要があります。また、ケーブルを抜去した後は、確保されていたリソースが全て開放されるため、再度ユニットの初期化から操作を行う必要があります。

※　パソコン、或いはセルフパワーハブの種類によっては、パソコンをシャットダウンした後、本ＬＥＤが点燈し続けたり、点滅をし続けたりする場合がありますが、いずれの場合であっても、パソコンを再起動する事により上の図で示したシーケンスをＵＳＢケーブル挿入の段階から実行しますので、機能上特に問題はありません。

※　ＦＣＡＤ４１６－ＤＳＵＢとＦＣＡＤ４１６とは、ソフトウェア上同一とみなされ、汎用入出力点数を除きソフト的に差異は存在しません。そのため、実際的な意味はあまりありませんが、両者を混在使用する事も理論上可能です。

１－２－２　ＳＷ１

ＳＷ１は、ユニットＩＤを設定するロータリースイッチです。
このスイッチをユニット毎に異なる値にすることで、ＩＤ＝０のユニットをマスターとする、マスタースレーブ接続動作を行うことができます。
スレーブユニットは最大１５台まで使用できますが、その際の条件としては
１　夫々のユニットで異なるＩＤを設定してあること
２　ユニットＩＤ＝０をマスターユニットとし、スレーブユニットにはマスターユニットから同期トリガ出力及び同期クロック出力を接続しておくこと
の二点が必要です。スレーブユニットのＩＤとしては、０でない夫々異なるＩＤという条件を満たしていれば、動作します。例えば、ＩＤ＝０->ＩＤ＝３->ＩＤ＝１->ＩＤ＝８、、のように設定しても、問題なく動作します。



１－３．入出力コネクタ・ピン接続

ＣＮ１　アナログ入力コネクタ

図１－３Ａ．アナログ入力コネクタ（ＣＮ１）ピン接続（嵌合面視）

コネクタ型式　17LE-13370-27(D4AB)-FA (DDK)

対応ヘッダ　17JE-23370-02(D8A)-CG (DDK)

信号名	ピン番号	ピン番号	信号名
ＣＨ０アナログ入力（差動ＣＨ０＋側）	1	○	
		○	20　ＡＧ（アナログ・グランド）
ＣＨ１アナログ入力（差動ＣＨ０－側）	2	○	
		○	21　ＡＧ（　〃　　〃　）
ＣＨ２アナログ入力（差動ＣＨ１＋側）	3	○	
		○	22　ＡＧ（　〃　　〃　）
ＣＨ３アナログ入力（差動ＣＨ１－側）	4	○	
		○	23　ＡＧ（　〃　　〃　）
ＣＨ４アナログ入力（差動ＣＨ２＋側）	5	○	
		○	24　ＡＧ（　〃　　〃　）
ＣＨ５アナログ入力（差動ＣＨ２－側）	6	○	
		○	25　ＡＧ（　〃　　〃　）
ＣＨ６アナログ入力（差動ＣＨ３＋側）	7	○	
		○	26　ＡＧ（　〃　　〃　）
ＣＨ７アナログ入力（差動ＣＨ３－側）	8	○	
		○	27　ＡＧ（　〃　　〃　）
ＣＨ８アナログ入力（差動ＣＨ４＋側）	9	○	
		○	28　ＡＧ（　〃　　〃　）
ＣＨ９アナログ入力（差動ＣＨ４－側）	10	○	
		○	29　ＡＧ（　〃　　〃　）
ＣＨ10アナログ入力（差動ＣＨ５＋側）	11	○	
		○	30　ＡＧ（　〃　　〃　）
ＣＨ11アナログ入力（差動ＣＨ５－側）	12	○	
		○	31　ＡＧ（　〃　　〃　）
ＣＨ12アナログ入力（差動ＣＨ６＋側）	13	○	
		○	32　ＡＧ（　〃　　〃　）
ＣＨ13アナログ入力（差動ＣＨ６－側）	14	○	
		○	33　ＡＧ（　〃　　〃　）
ＣＨ14アナログ入力（差動ＣＨ７＋側）	15	○	
		○	34　ＡＧ（　〃　　〃　）
ＣＨ15アナログ入力（差動ＣＨ７－側）	16	○	
		○	35　ＡＧ（　〃　　〃　）
（空き）	17	○	
		○	36　（空き）
（内部使用：使用不可）	18	○	
		○	37　ＤＧ（デジタル・グランド）
（空き）	19	○	
（筐体上面側）			（筐体底面側）

【注】アナログ・グランドＡＧとデジタル・グランドＤＧはボード内部で接続されています。

ＣＮ２　ＵＳＢコネクタ

ＵＳＢミニＢコネクタ（ＣＮ２）ピン接続

コネクタ型式　UX60A-MB-5ST（ヒロセ）

信号名	ピン番号
Vbus	1
－Data	2
＋Data	3
ID(NC)	4
GND	5

本ユニットでは、ＩＤピンは使用しておりません。

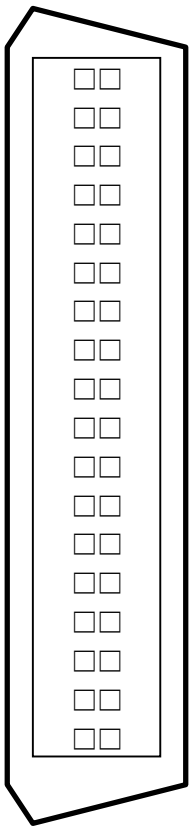
ＣＮ３　デジタル入出力コネクタ

図１－３Ｂ．デジタル入出力コネクタ（ＣＮ３）ピン接続（嵌合面視）

コネクタ形式　DX10A-36S(50)（ヒロセ）

対応ヘッダ　DX40-36P(55)+DX-36-CV1(ヒロセ)

信号名	ピン番号	ピン番号	信号名
（デジタル入力ビット０）　Ｉ０	1	19	ＤＧ　（デジタル・グランド）
（　〃　〃　〃　１）　Ｉ１	2	20	ＤＧ　（　〃　・　〃　）
（　〃　〃　〃　２）　Ｉ２	3	21	ＤＧ　（　〃　・　〃　）
（　〃　〃　〃　３）　Ｉ３	4	22	ＤＧ　（　〃　・　〃　）
（デジタル出力ビット０）　Ｑ０	5	23	ＤＧ　（　〃　・　〃　）
（　〃　〃　〃　１）　Ｑ１	6	24	ＤＧ　（　〃　・　〃　）
（　〃　〃　〃　２）　Ｑ２	7	25	ＤＧ　（　〃　・　〃　）
（　〃　〃　〃　３）　Ｑ３	8	26	ＤＧ　（　〃　・　〃　）
（ｻｰﾋﾞｽﾘｸｴｽﾄ入力）　ＩＮＴ－ＩＮ	9	27	ＤＧ　（　〃　・　〃　）
（トリガ入力）　ＴＲＧ－ＩＮ	10	28	ＤＧ　（　〃　・　〃　）
（クロック入力）　ＣＬＫ－ＩＮ	11	29	ＤＧ　（　〃　・　〃　）
（ストップ入力）　ＳＴＰ－ＩＮ	12	30	ＤＧ　（　〃　・　〃　）
（空き）	13	31	（空き）
（トリガ入力）　ＴＲＧ－ＩＮ	14	32	ＤＧ　（　〃　・　〃　）
（クロック入力）　ＣＬＫ－ＩＮ	15	33	ＤＧ　（　〃　・　〃　）
（空き）	16	34	ＤＧ　（　〃　・　〃　）
同期トリガ出力　ＳＹＮＣ－ＴＲＧ	17	35	ＤＧ　（　〃　・　〃　）
同期クロック出力　ＣＬＫ－ＯＵＴ	18	36	ＤＧ　（　〃　・　〃　）
（筐体上面側）			（筐体底面側）

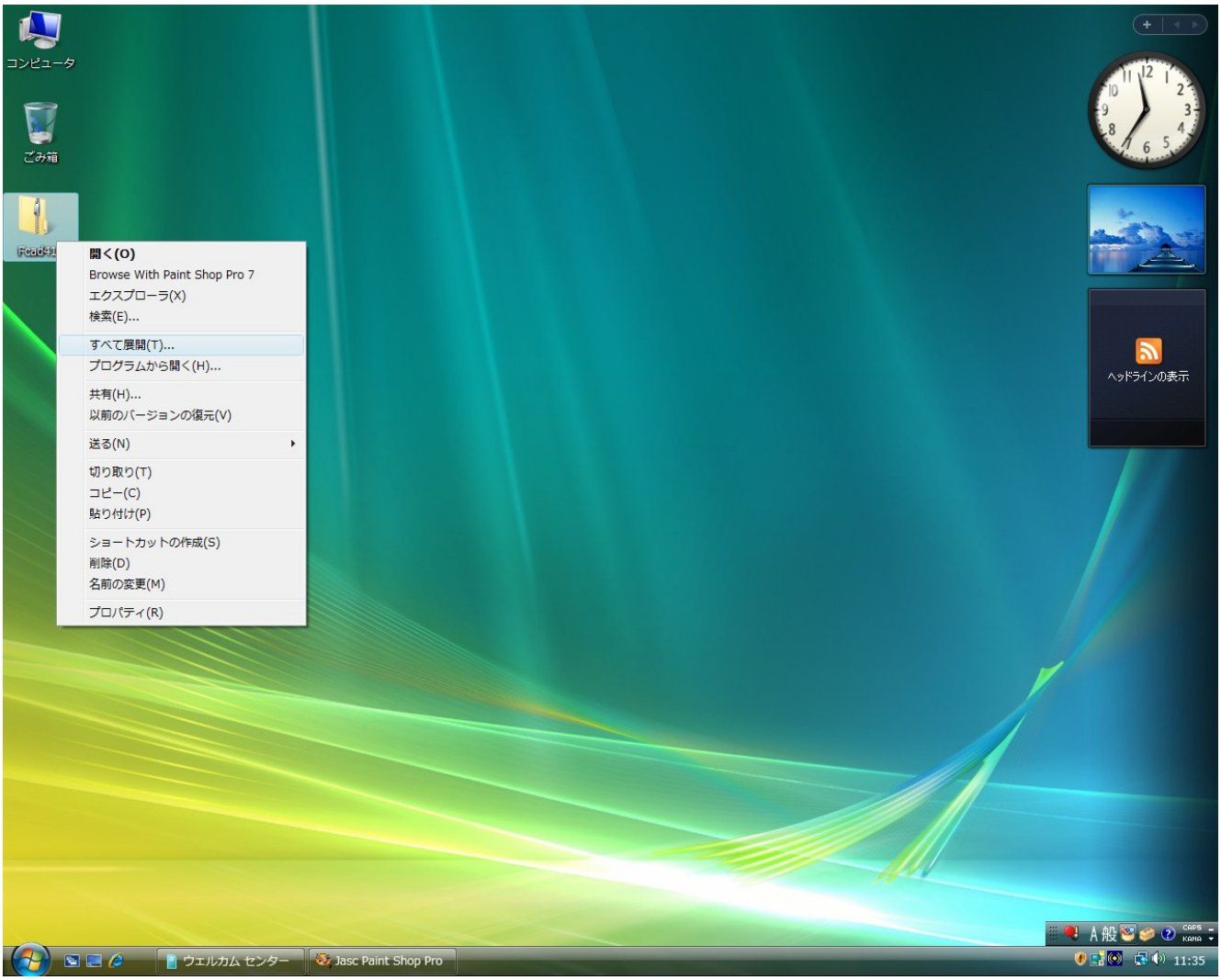


※　１４ピン、１５ピン入力は、夫々１０ピン、１１ピン入力と同じ信号を受けており、マスタースレーブ接続時にコネクタ端での渡り配線用として使用するために実装されているものです。
また、ＣＮ１、ＣＮ３は、コネクタ嵌合面からユニット内部へ向かってコネクタ端子を見たときの状態で表示しています。

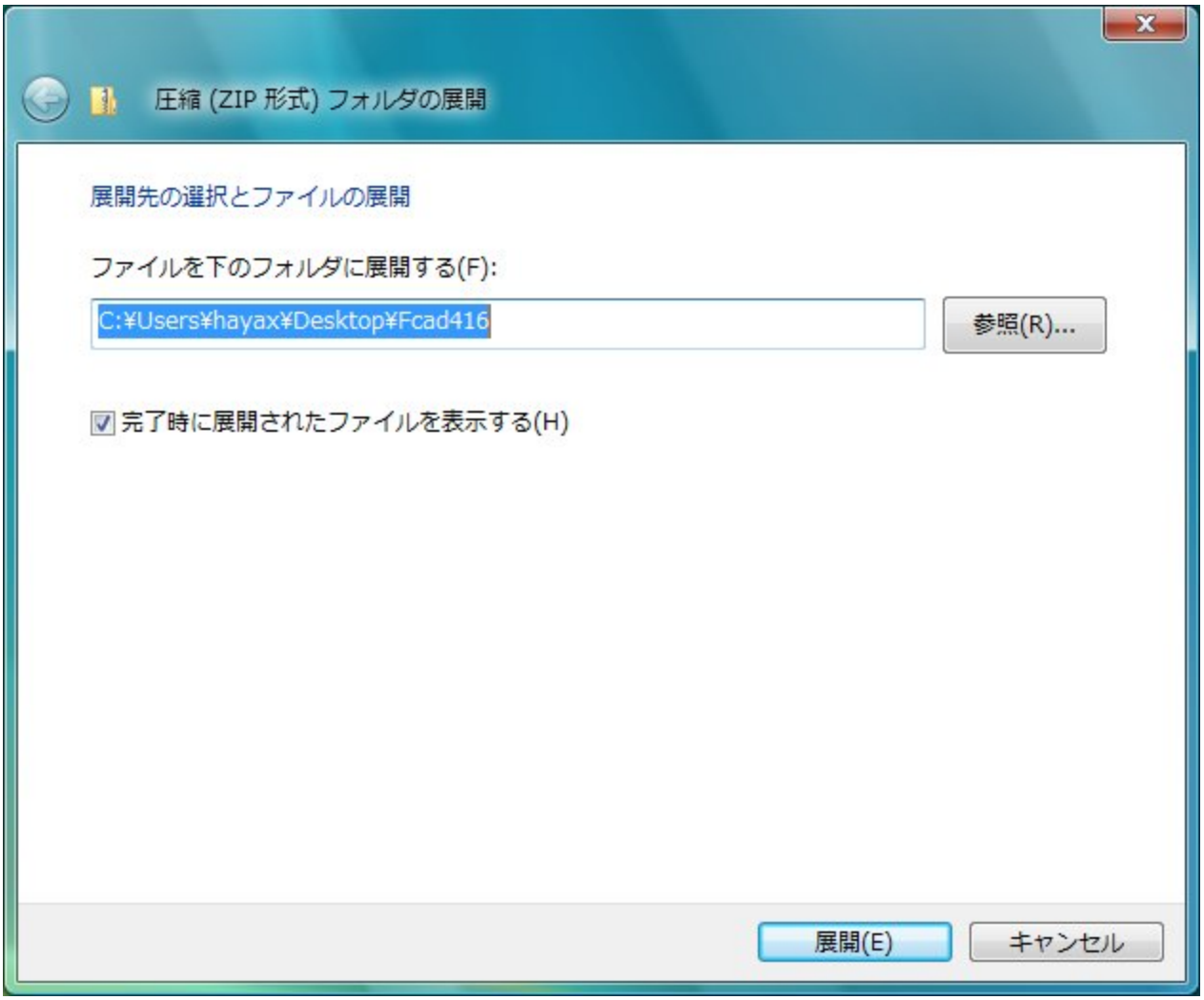
１－４．ソフトウェアの解凍・展開

１－４－１　Windows7 から WindowsVISTA までの操作

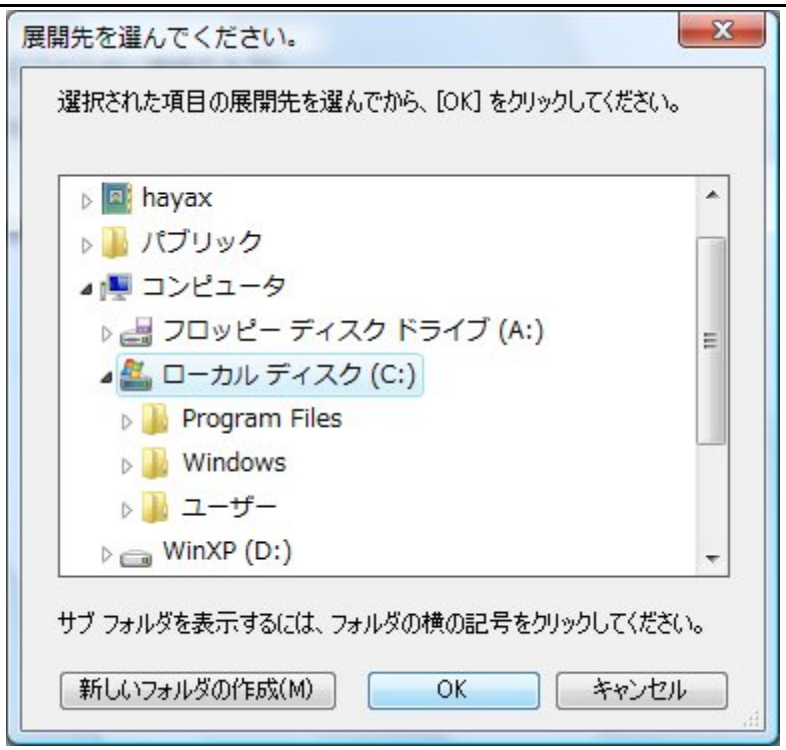
本製品用のソフトウェアは弊社ホームページよりダウンロードして頂き御使用になるパソコンにあらかじめ展開しておく必要があります。（この作業は、インストール操作ではありません。また本ユニットのインストールに先立って行っておく必要があります。）
弊社ホームページより圧縮形のソフトウェアセット（Fcad416.zip）をダウンロードして頂き、デスクトップ等の上で右クリックします。



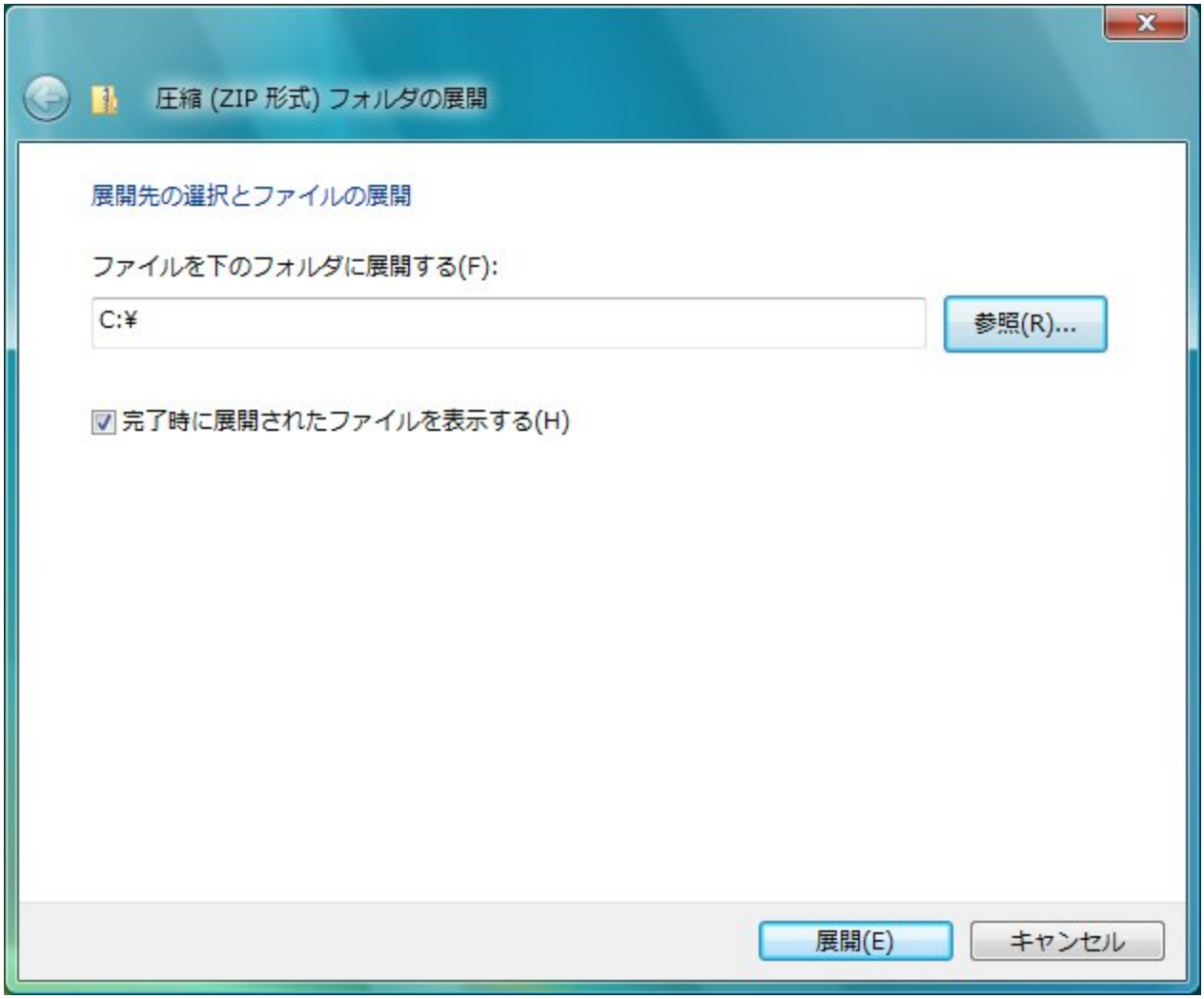
あらわれるドロップダウンメニューの中から、“すべて展開“を選んでクリックします。
するとＺＩＰファイル解凍メニューが表示されます。（次ページ）



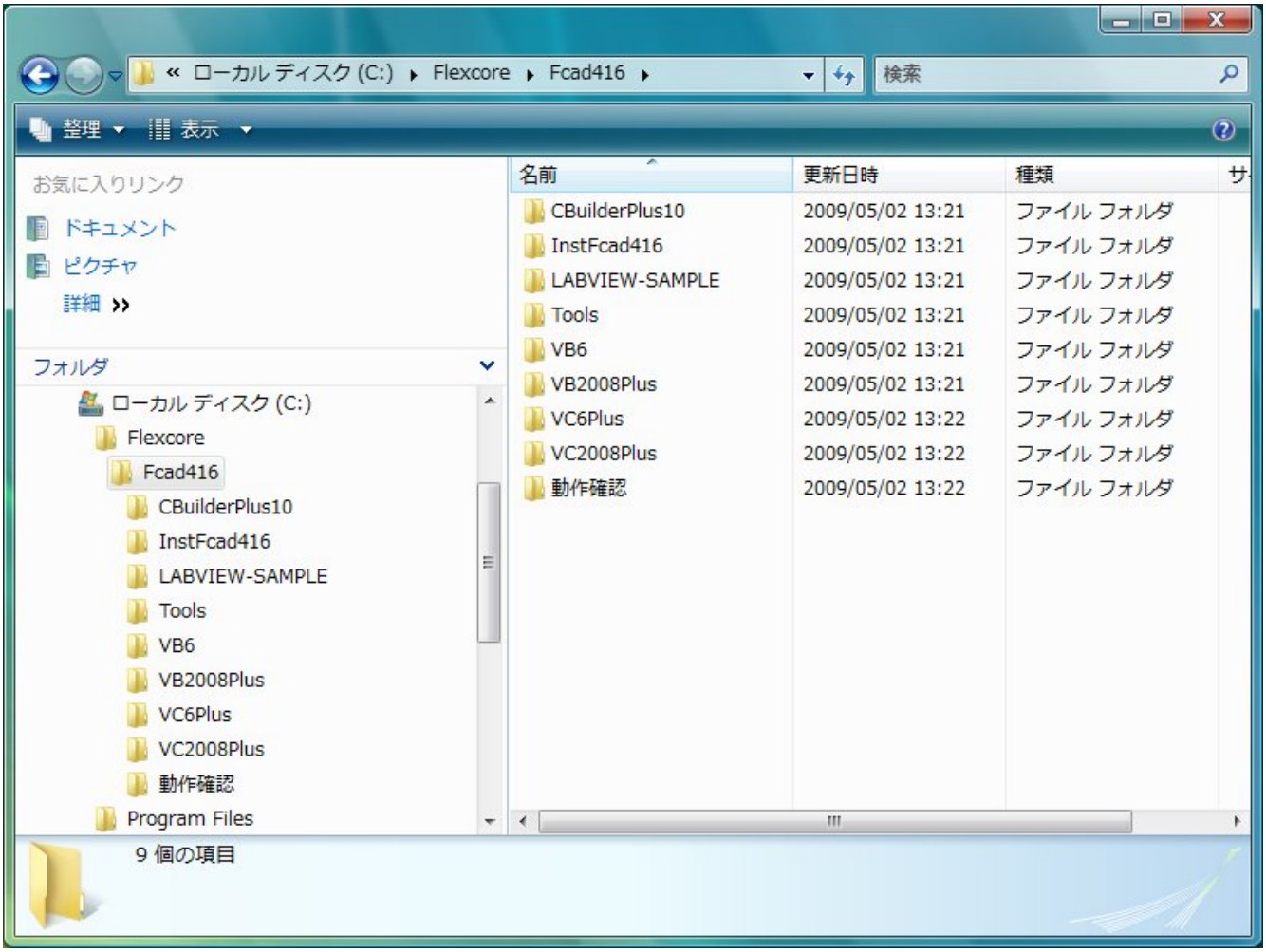
ここでは“参照“ボタンを押して展開する場所を選択して下さい。デフォルト（使用ユーザーのデスクトップ）のままで問題はありませんが、ドライバーインストール、サンプルプログラム操作など、このパスを使用する局面は多いので分かりやすい場所がお勧めだと思います。（次ページ）



ここでは例として“Ｃドライブ”を選択し、“ＯＫ”ボタンを押します。すると展開先が決定されますので、



“展開”ボタンを押してしばらく待っていると、次のように必要なファイルが展開されます。

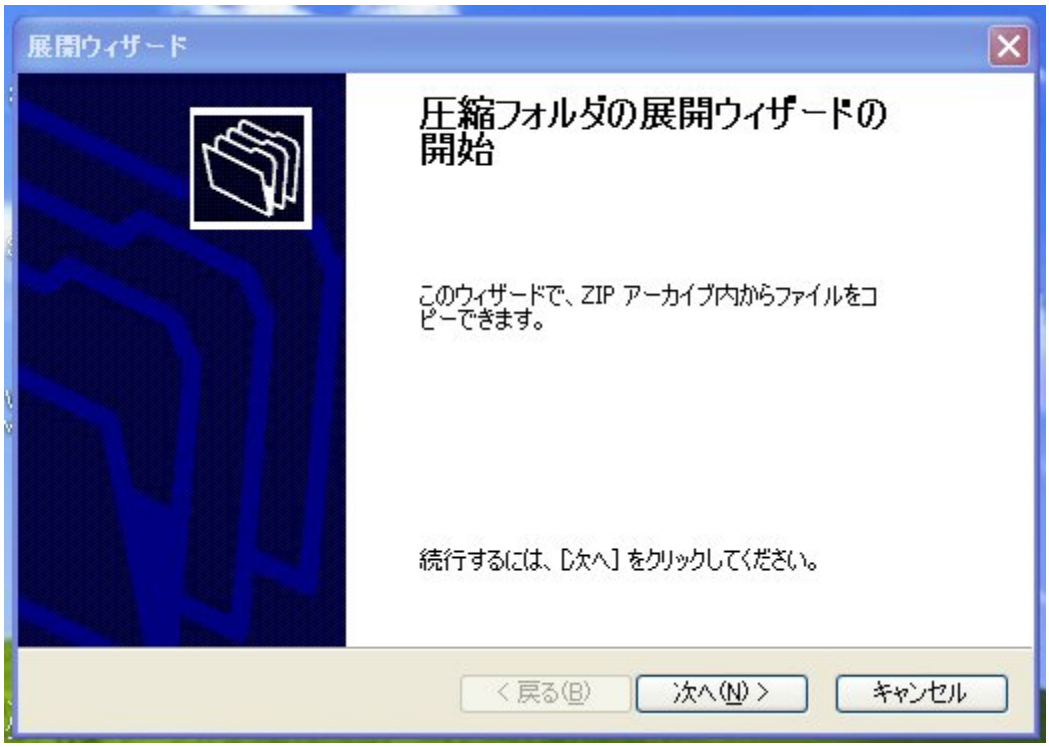


この処理が終了した後、ダウンロードファイル（Fcad416.zip）は削除して頂いても問題ありません。

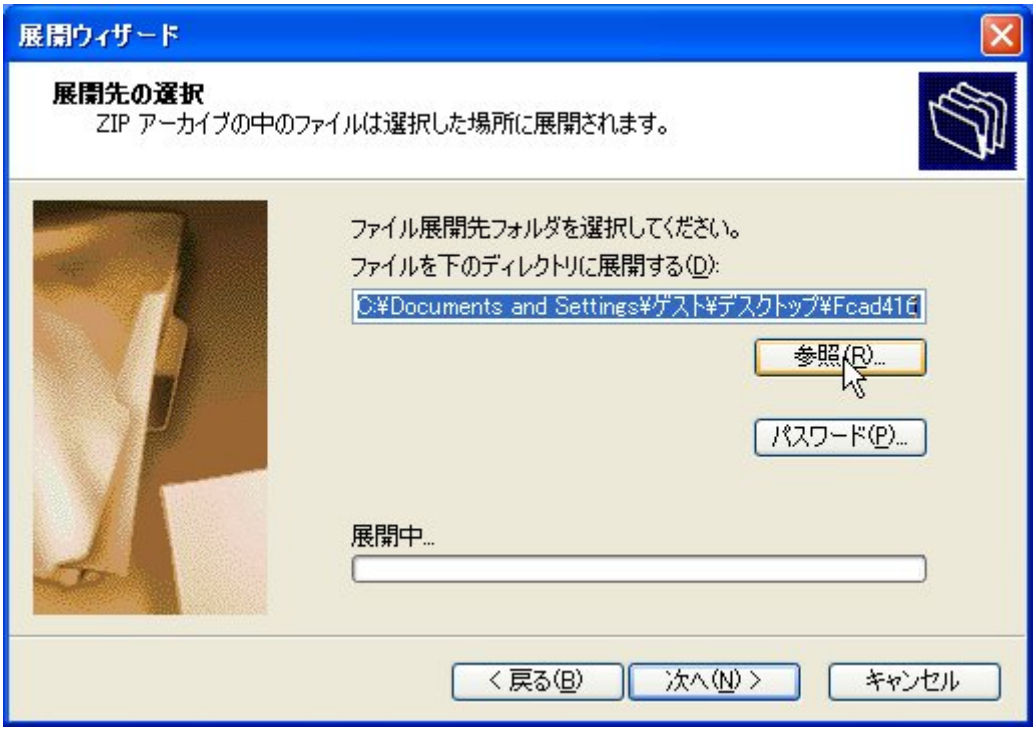
１－４－２　WindowsXP での操作

弊社ホームページより、ソフトウェアキット（Ｆｃａｄ４１６．ｚｉｐ）ファイルをダウンロードして頂きます。ダウンロードする場所はどこでも構いません。一般的にはデスクトップ上等が簡単かと思います。
ダウンロードしていただいた、Ｆｃａｄ４１６．ｚｉｐファイルを右クリックしていただくと、下の図のようなドロップダウンリストが表示されます。ここでは、この中から“すべて展開”コマンドにカーソルを合わせ、マウスの左ボタンをクリックして下さい。

すると、次のようなウィンドーが表示されます。

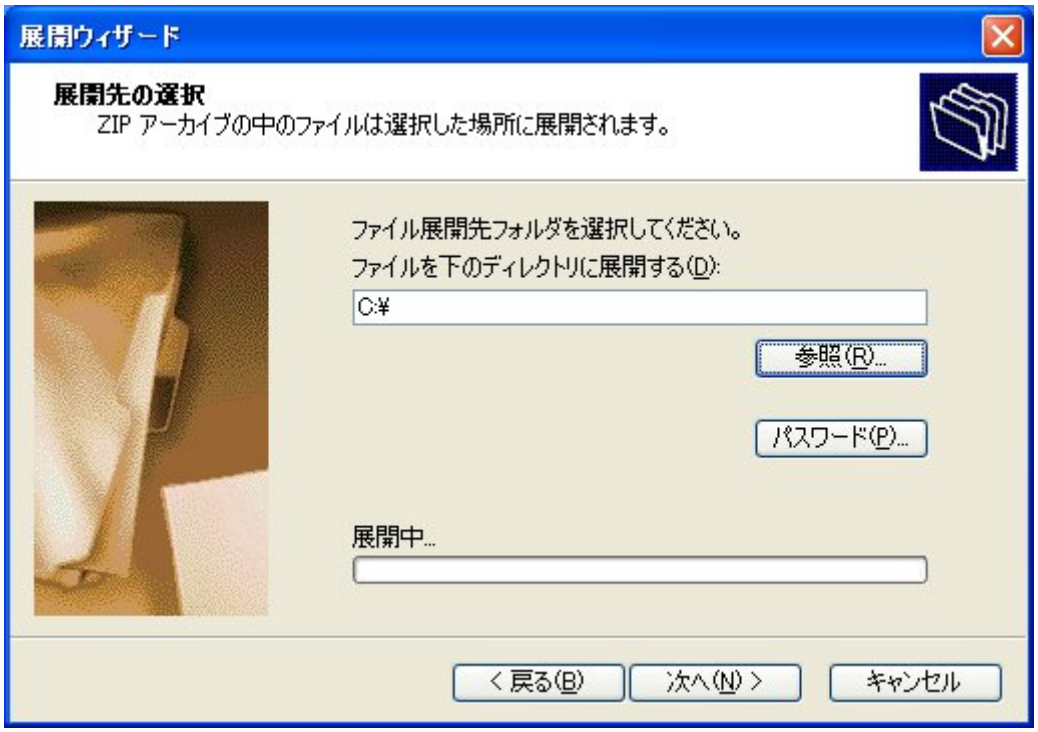


ここでは、“次へ”ボタンを（左ボタンで）クリックして下さい。すると、次のウィンドーがあらわれます。（次頁）

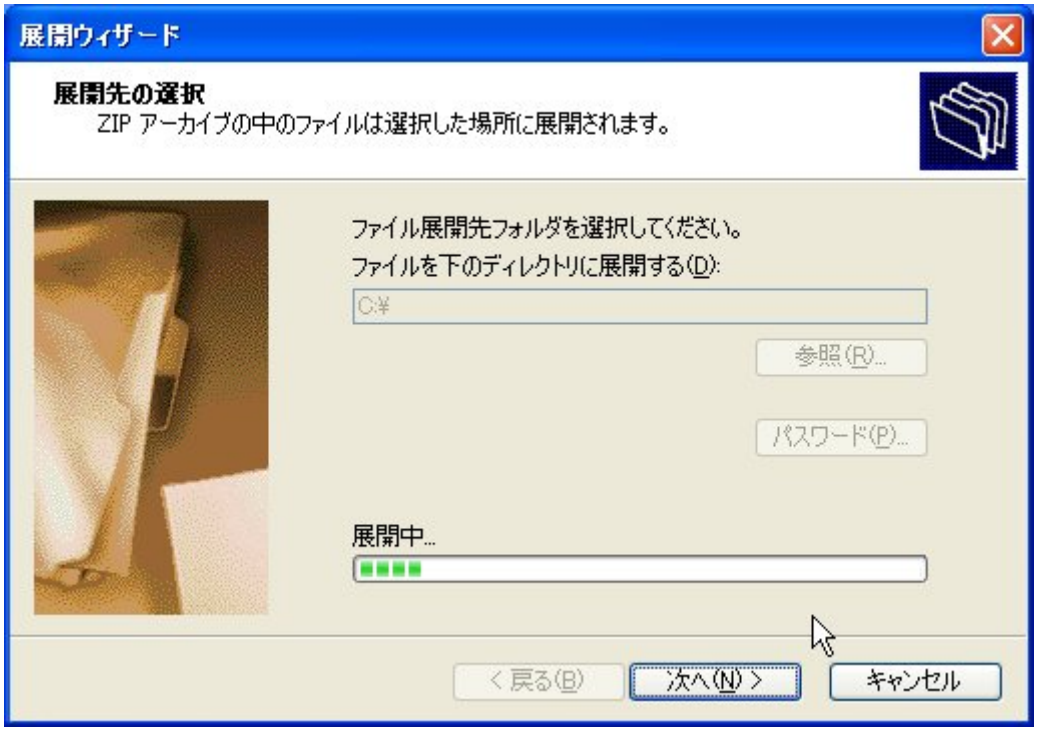


この段階で、圧縮ファイルを解凍する場所が選択できます。デフォルト（使用ユーザーのデスクトップ）でも問題はありませんが、この後、ドライバーインストールやサンプルソフトウェアの動作確認、更には修正など色々な作業を行う事になるため、ご自分で分かりやすい場所を選ばれるのがよいと思います。ここでは例として“Ｃドライブ”を指定してみます。
“参照“ボタンをクリックし、下の図のようにして”Ｃ“ドライブを指定します。

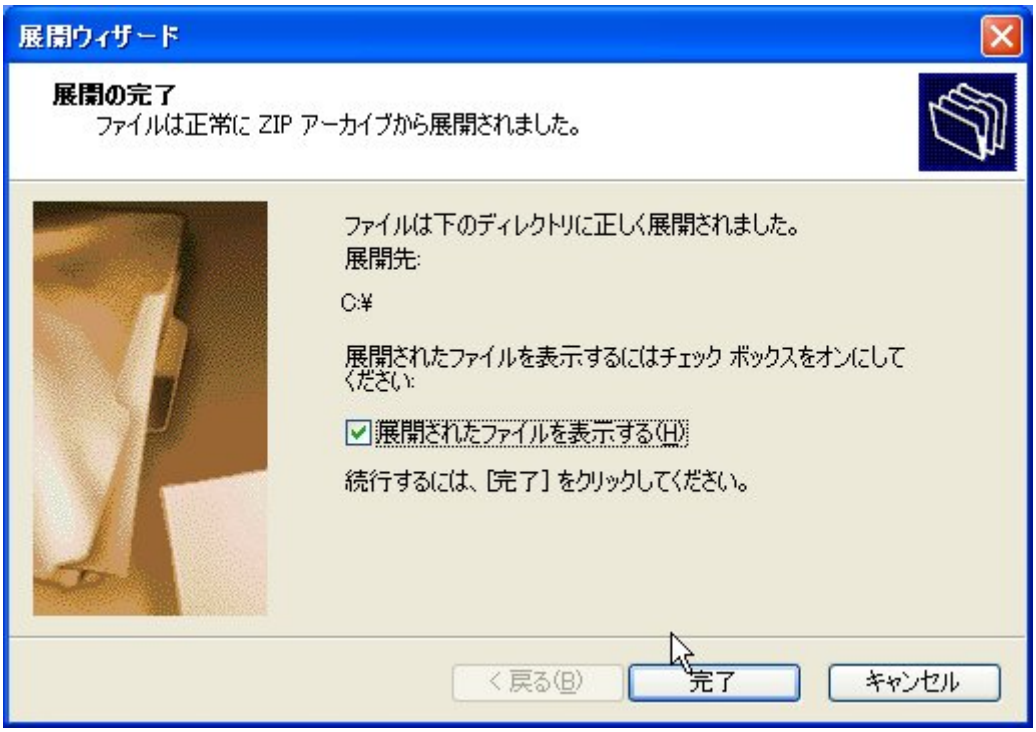
“ＯＫ”ボタンをクリックすると、解凍先が“Ｃ“ドライブに変わります。（次頁）



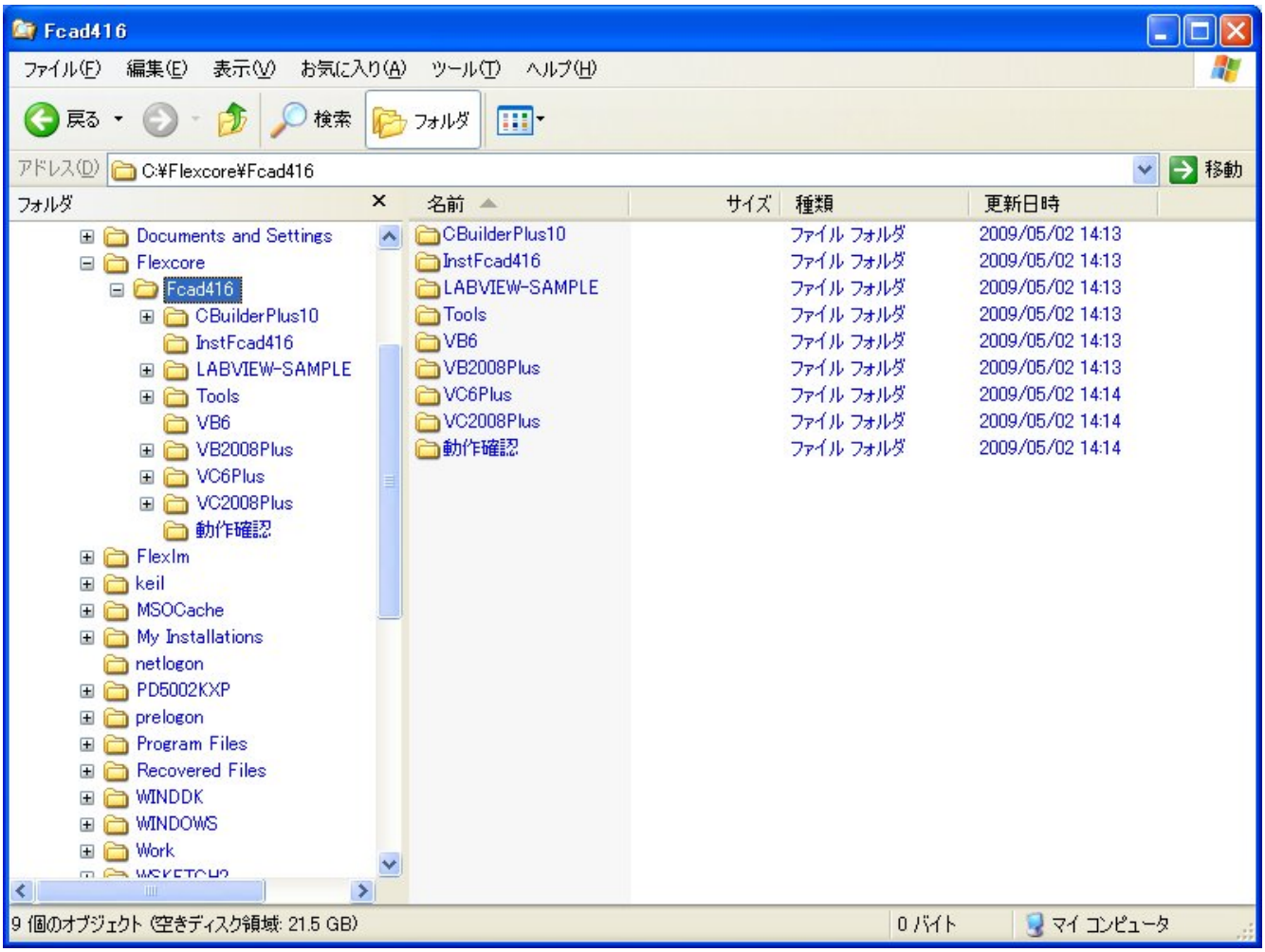
そして、“次へ“ボタンをクリックすると解凍が始まります。



解凍が終了すると次のウィンドーがあらわれます。（次頁）



この時点で“完了”ボタンをクリックすると、実際に解凍されたファイル全体を見る事ができます。



この処理が終了した後、ダウンロードファイル（Fcad416.zip）は削除して頂いても問題ありません。

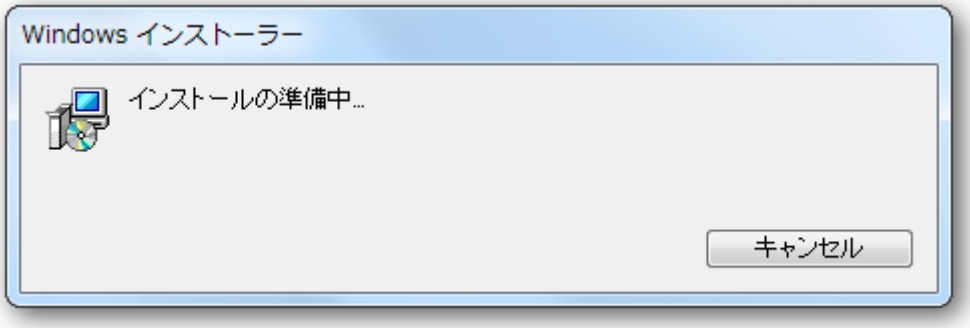
１－５．ユニットのインストール

１－５－１　Windows7 から WindowsVISTA までの操作

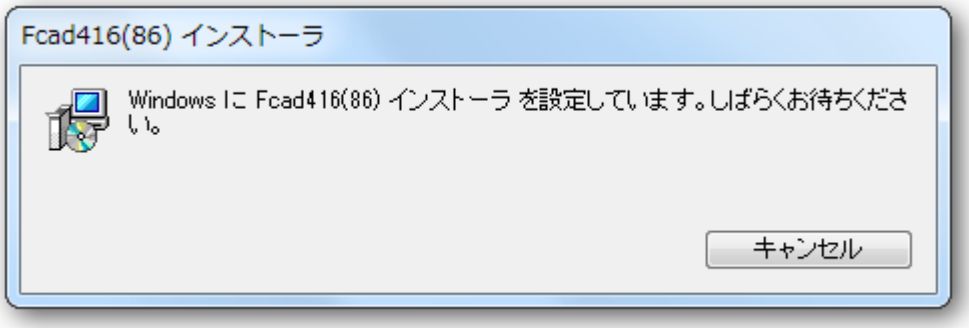
１－５－１－１　ドライバーのインストール

ＦＣＡＤ４１６はプラグアンドプレイに対応したＵＳＢ接続アナログ入力変換ユニットです。
御使用に先立ち、パソコンにインストール（認識・リソース割り当て）する必要があります。 この作業はパソコンに本ユニットを始めて接続した時に自動的に実行されます。或いは、当初とは別のＵＳＢポートに、初めて接続した際にも行われます。

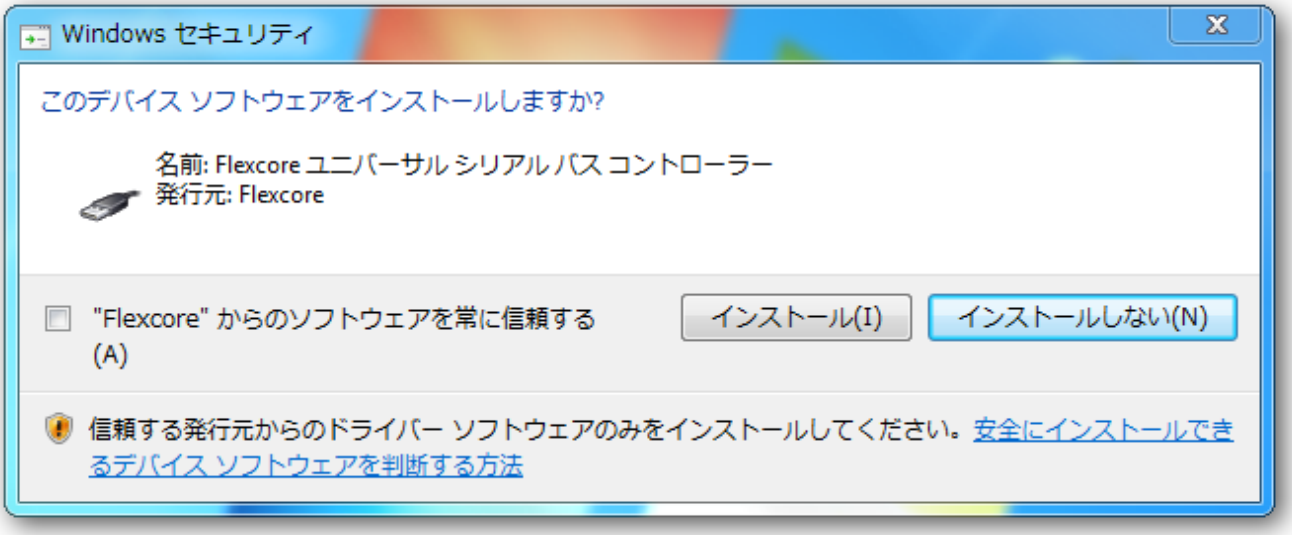
- 本ユニットをパソコンに接続する前に、弊社ホームページよりＦＣＡＤ４１６用ソフトウェアセットをダウンロードして頂き、パソコンの中に解凍します。解凍場所は、どこでもかまいませんが、後の作業を考えると、Ｃドライブ等が無難でしょう。（本ソフトウェアセットには、デバイスハンドラー、サンプルソフトが含まれています。）詳細については、１－１－１項、１－１－２項および１－４項を参照して下さい。
- 解凍したソフトウェアパッケージの中に、ドライバーパッケージが含まれています。この中から、本ユニットをインストールしようとしているパソコンに導入されているＯＳに対応したパッケージを選択し、マウスでダブルクリックして下さい。ＯＳとドライバーパッケージとの関係については、１－１－２項に詳細な記述があります。
  また、この時点で、本ユニットは、まだパソコンとは接続しないで下さい。この順序が守られない場合、正常なインストールは保証できません。ドライバーパッケージが解凍され、次のようなウィンドーが表示されます。



しばらく待っているとユーザーアカウント制御ウィンドーが表われます。ここでは、左側の“はい”というボタンをクリックして下さい。ウィンドーが次のように変わります。

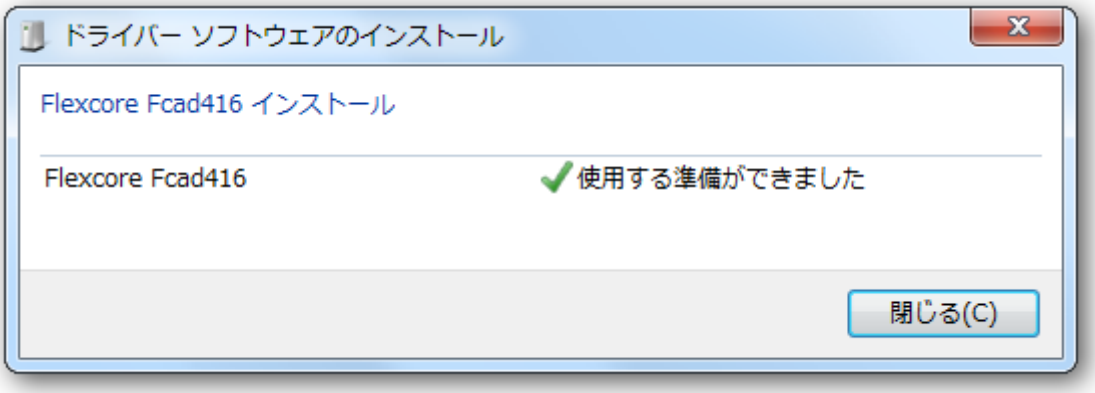


更に待っていると、次に下記のようなウィンドーが表われます。 （次ページ）



ここでは、まず“Flexcore”からのソフトウェアを常に信頼する　チェックボックスにチェックを入れ、“インストール”ボタンをクリックします。そして、しばらくすると、インストールウィンドーが終了します。

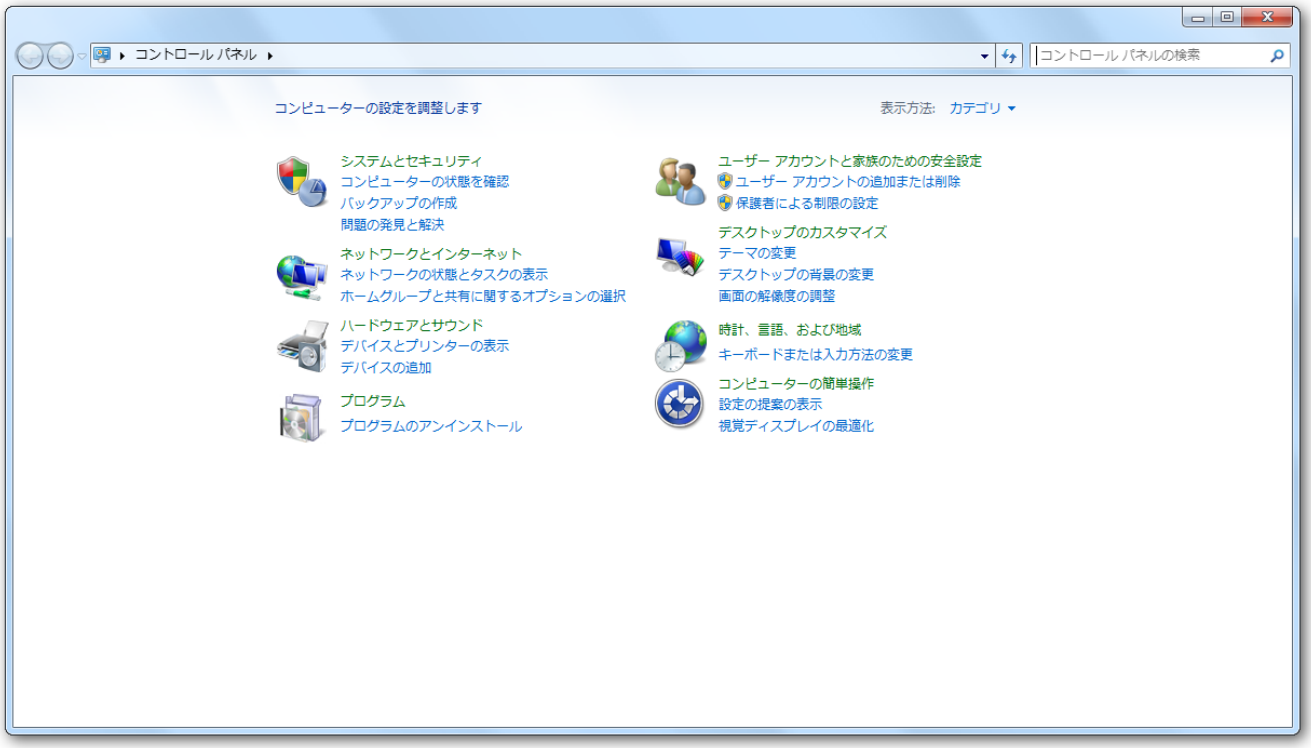
・　この後、本ユニットのＣＮ２（ＵＳＢコネクタ）とパソコンのＵＳＢコネクタとをＵＳＢ２．０対応ケーブルで接続します。少し待っていると画面下部のタスクバーの所に、デバイスドライバーが正常にインストールされたというメッセージが表示されます。



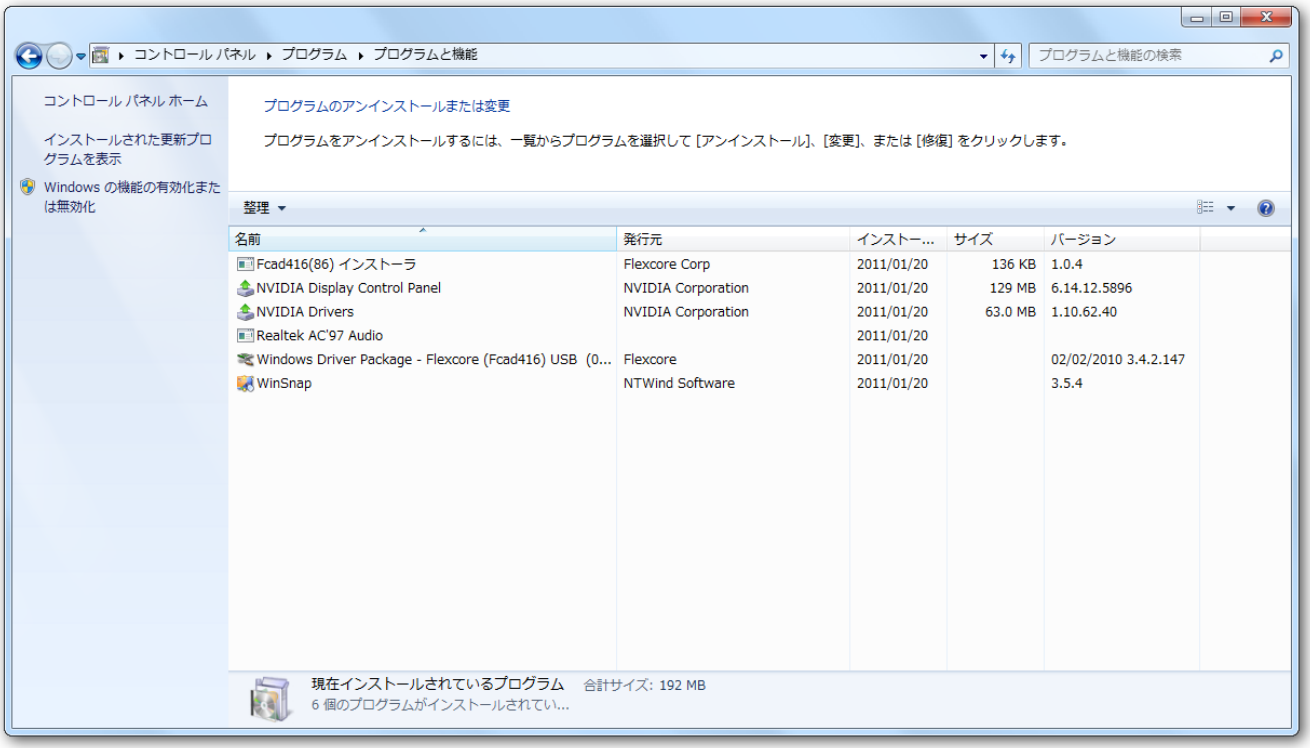
これで本ユニットのインストールは完了しました。

１－５－１－２　ドライバーのアンインストール

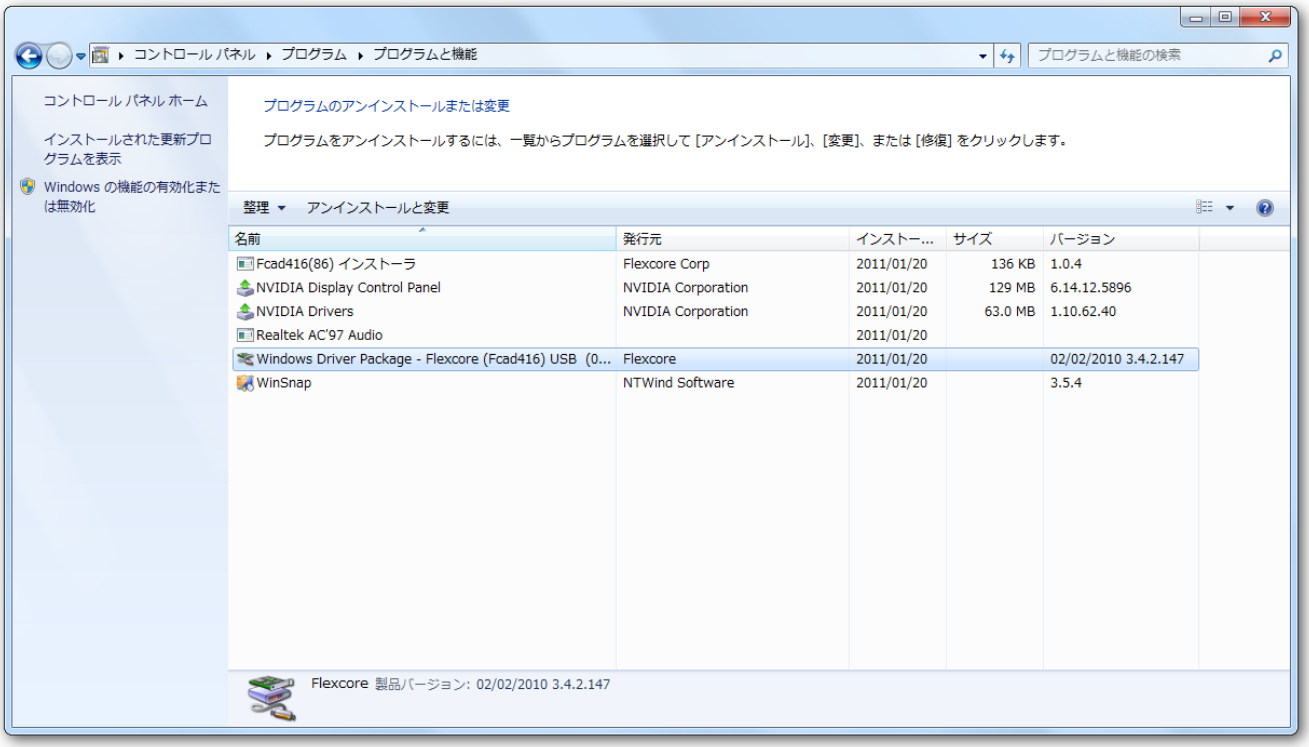
この方式でインストールしたドライバーはドライバー更新時などにドライバーをアンインストールする際にも簡単に行う事ができます。まず、
スタート＞コントロールパネルを開きます。(次ページ)



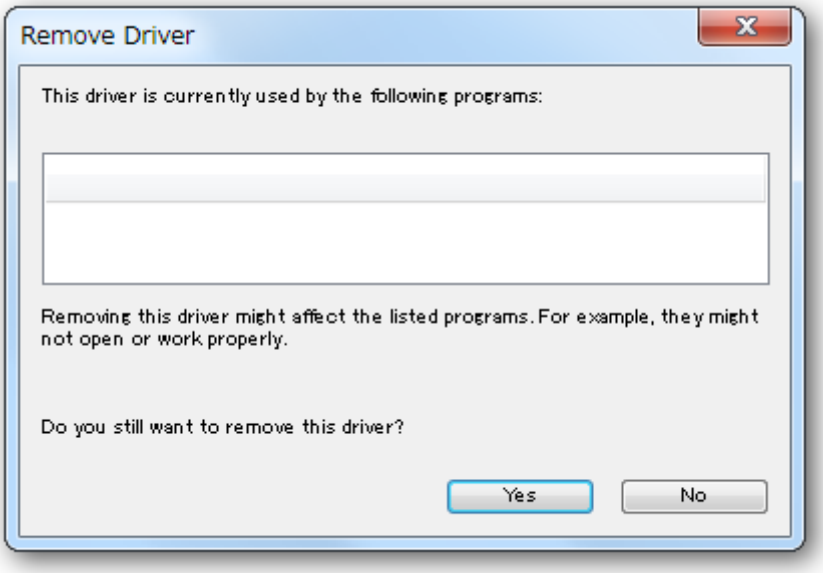
この中からプログラムのアンインストールを選びクリックします。



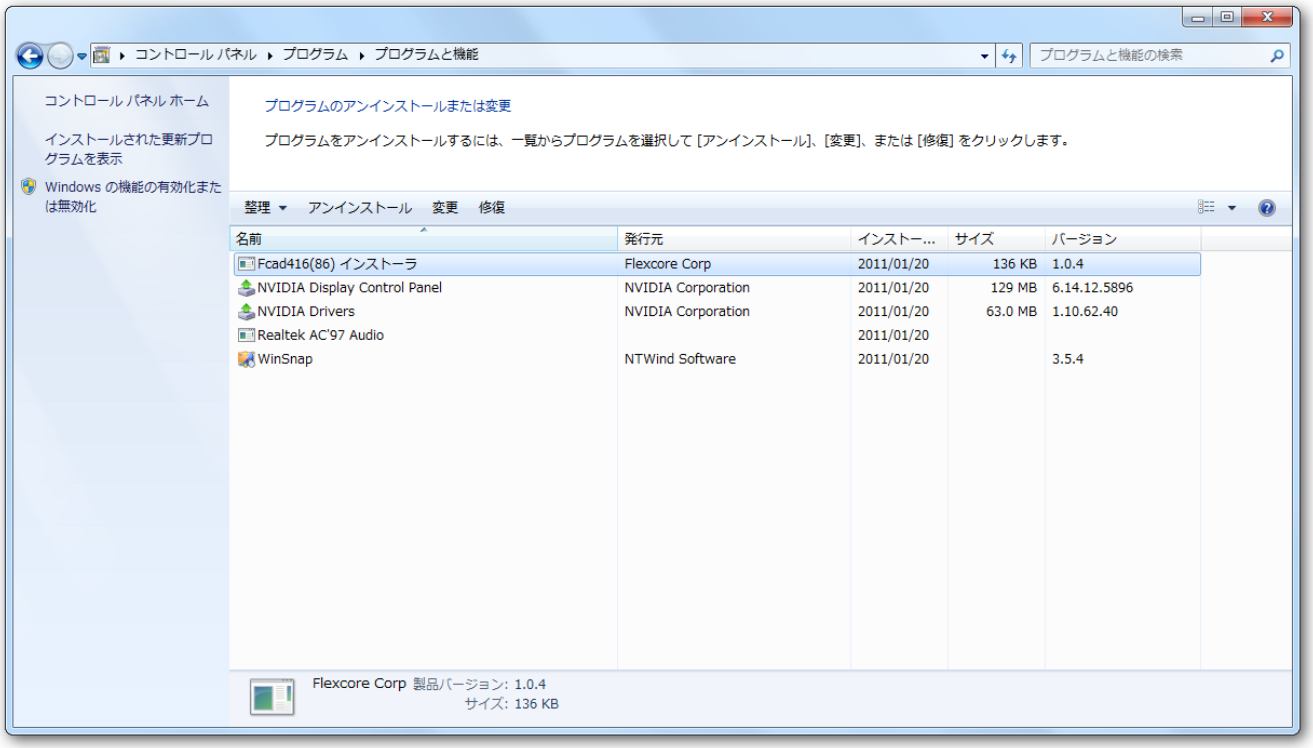
すると、インストールされたドライバーが表示されますから（上の図で発行元がFlexcoreとなっているラインです。）まず、WindowsDriverPackageと表示された項目を右クリックし、現れたメニューからアンインストールを選び、クリックします。



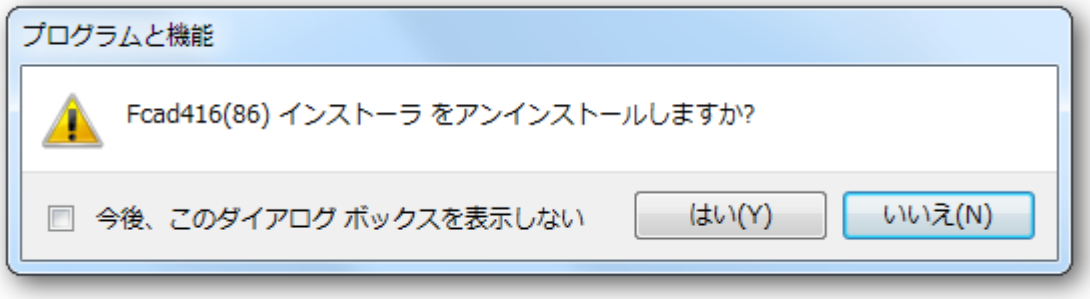
すると、アンインストール確認のウィンドーが開きますので（下の図）



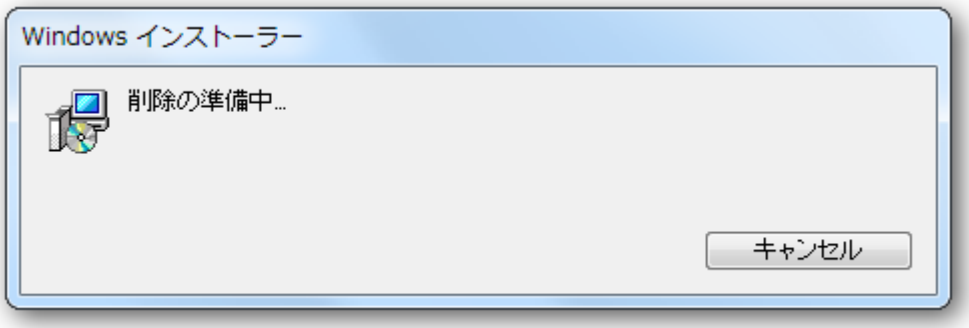
“Ｙｅｓ”ボタンをクリックします。これに続いて、Fcad416(86)インストーラも同様にしてアンインストールします。
(次ページ)



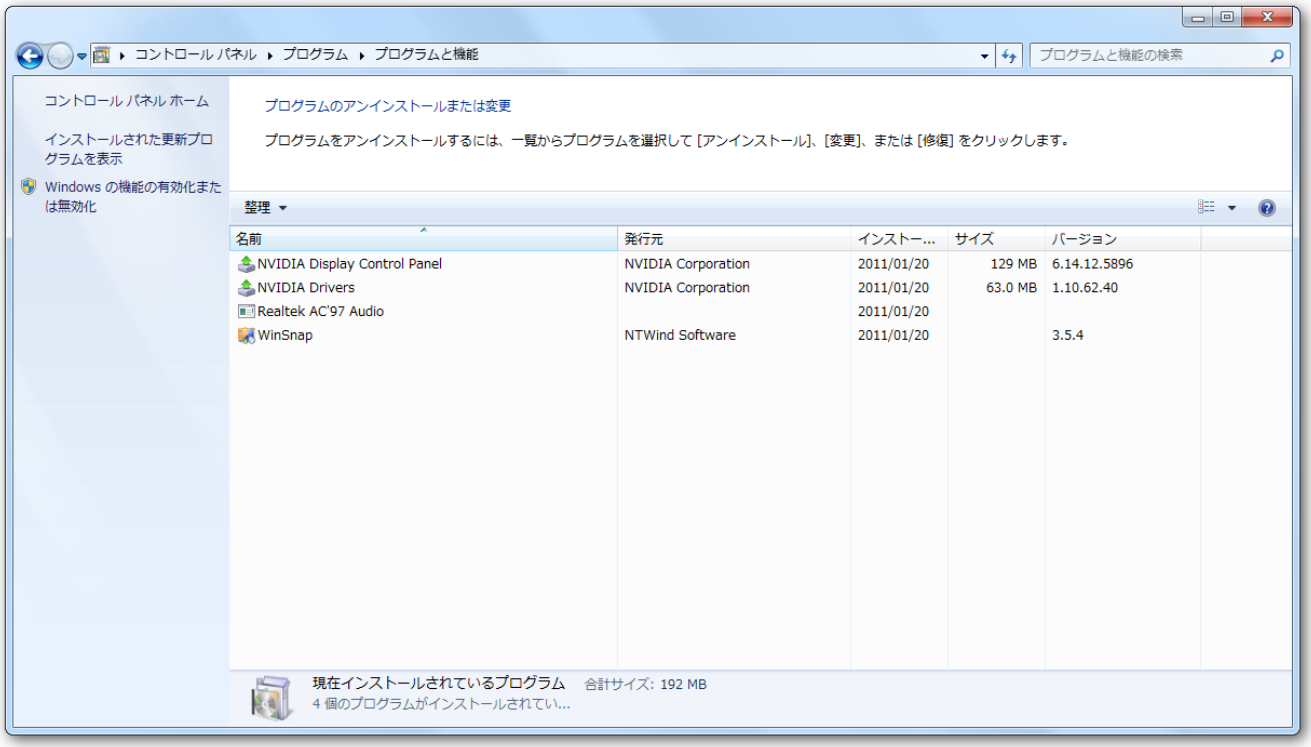
すると、アンインストール確認ウィンドーが開きますので（下の図）



“はい“をクリックします。すると暫く内部処理が行われた後、次のウィンドーが表われます。



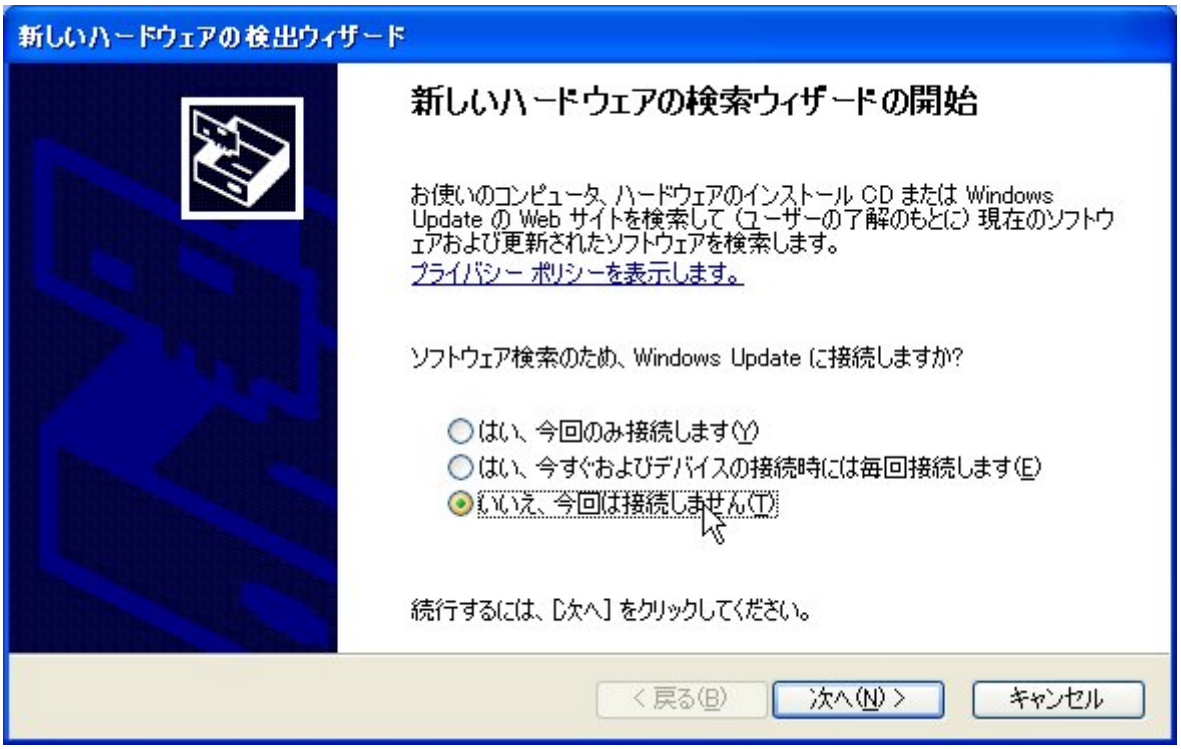
そして最終的に次のウィンドーが表示され、アンインストールは完了します。


コントロール パネル ▸ プログラム ▸ プログラムと機能
プログラムと機能の検索
コントロール パネル ホーム
インストールされた更新プログラムを表示
Windows の機能の有効化または無効化
プログラムのアンインストールまたは変更
プログラムをアンインストールするには、一覧からプログラムを選択して [アンインストール]、[変更]、または [修復] をクリックします。
整理
名前
発行元
インストー...
サイズ
バージョン
NVIDIA Display Control Panel
NVIDIA Corporation
2011/01/20
129 MB
6.14.12.5896
NVIDIA Drivers
NVIDIA Corporation
2011/01/20
63.0 MB
1.10.62.40
Realtek AC'97 Audio
2011/01/20
WinSnap
NTWind Software
2011/01/20
3.5.4
現在インストールされているプログラム
合計サイズ: 192 MB
4 個のプログラムがインストールされてい...

１－５－２　WindowsXP(86)などハードウェアウィザード使用ＯＳでの操作

ＦＣＡＤ４１６はプラグアンドプレイに対応したＵＳＢ接続アナログ入力変換ユニットです。
御使用に先立ち、パソコンにインストール（認識・リソース割り当て）する必要があります。 この作業はパソコンに本ユニットを始めて接続した時に自動的に実行されます。或いは、当初とは別のＵＳＢポートに、初めて接続した際にも行われます。

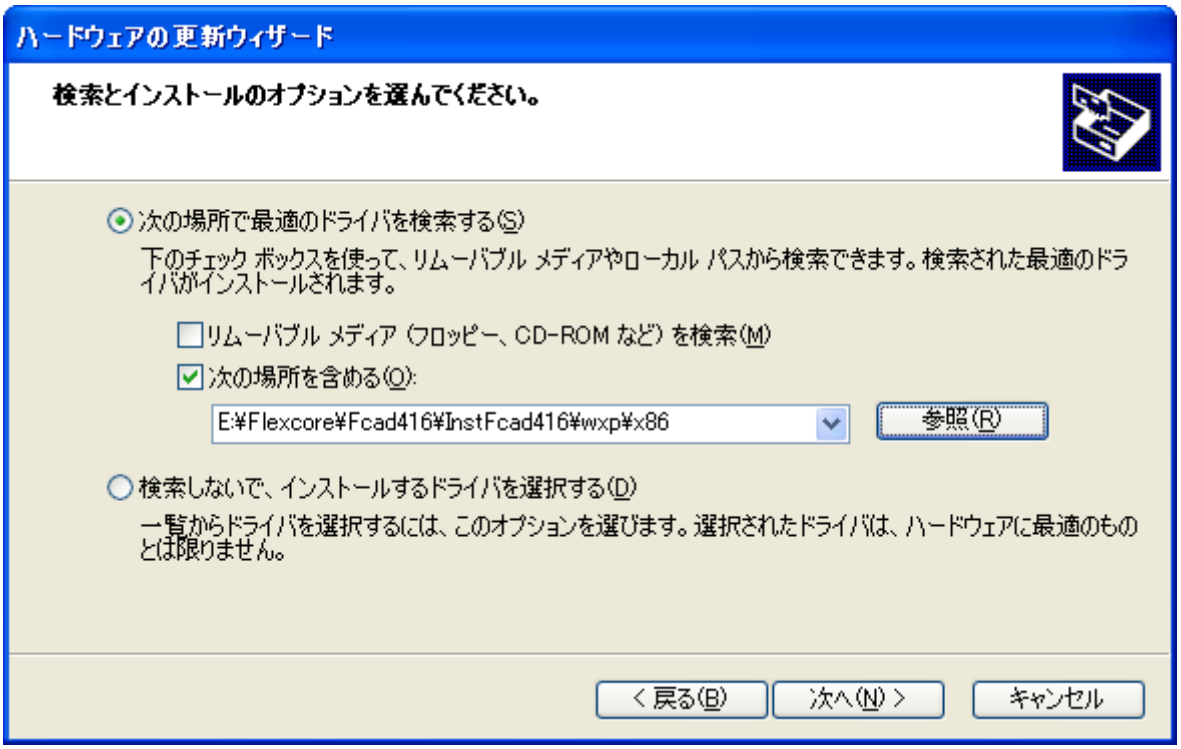
- 本ユニットをパソコンに接続する前に、弊社ホームページよりＦＣＡＤ４１６用ソフトウェアセットをダウンロードして頂き、パソコンの中に解凍します。解凍場所は、どこでもかまいませんが、後の作業を考えると、Ｃドライブ等が無難でしょう。（本ソフトウェアセットには、デバイスハンドラー、サンプルソフトが含まれています。）詳細については１－４項を参照して下さい。
- 本ユニットのＣＮ２（ＵＳＢコネクタ）とパソコンのＵＳＢコネクタとをＵＳＢ２．０対応ケーブルで接続します。少し待っていると、新しいハードウェアの検索ウィザードが表示されます。



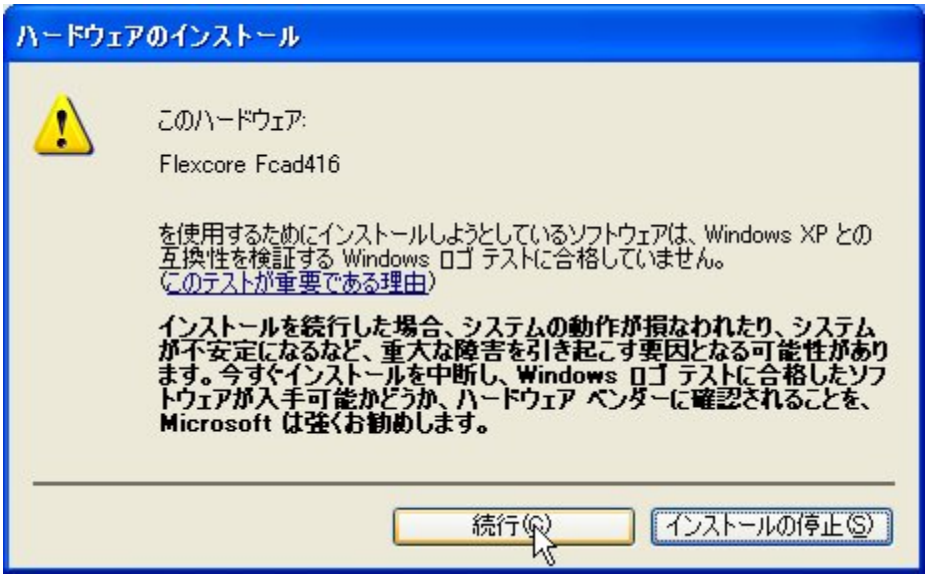
ここでは、３番目の“いいえ、今回は接続しません”というオプションを選び“次へ“ボタンをクリックします。
(次頁)

このウィンドーでは、下側の“一覧または特定の場所からインストールする”オプションを選び“次へ”ボタンをクリックします。

すると、“参照”ボタンを持ったウィンドーに切り替わりますので、そのボタンをクリックし、あらかじめソフトウェアを解凍しておいた場所を指示します。（上の図）そして“OK”ボタンをクリックすると、次の画面に切り替わります。また、この時“リムーバブルメディアを検索“にチェックが入っていたら外しておいたほうが無難です。



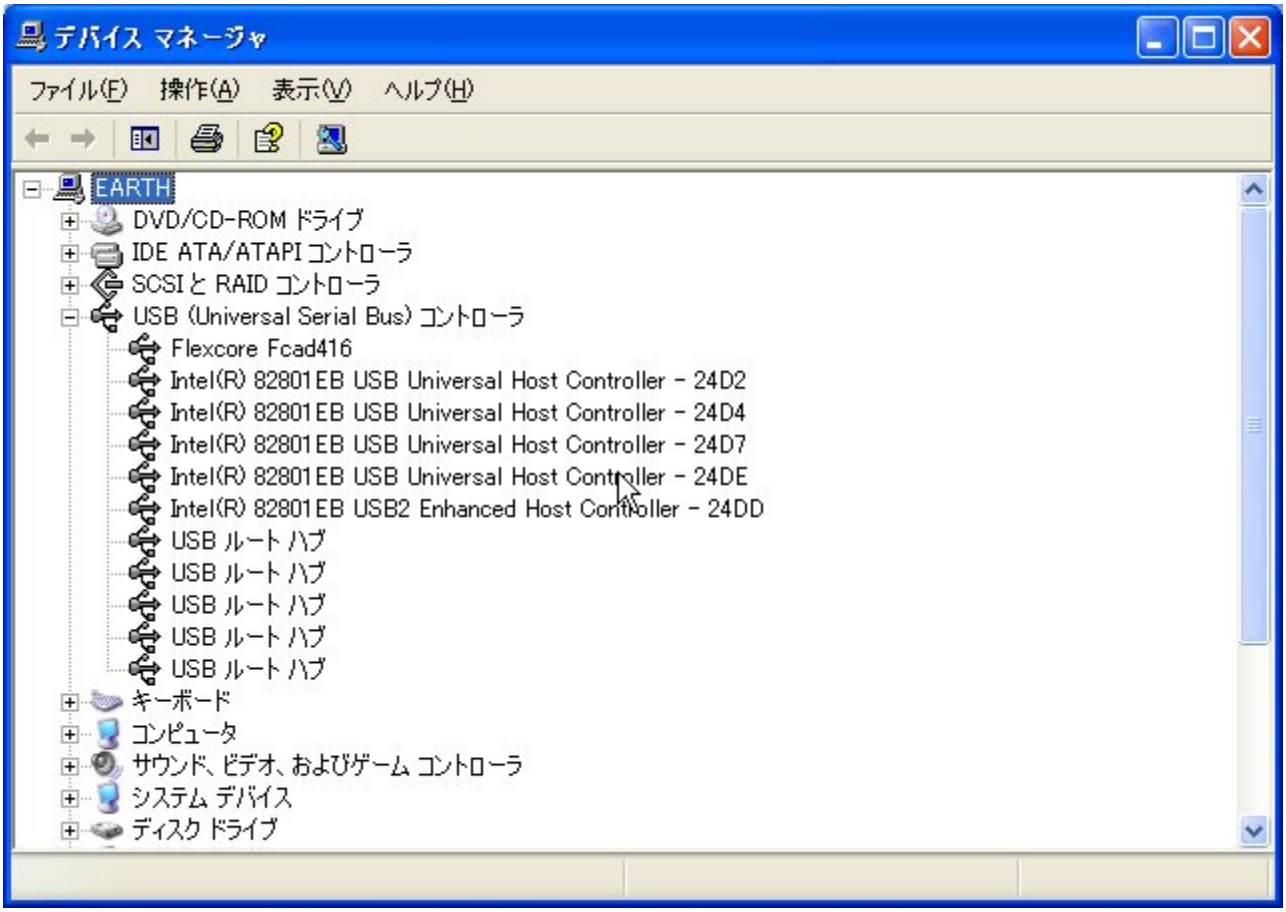
ここで、インストールに必要な情報が確定しましたので再び“次へ”ボタンをクリックします。すると



しばらく作業が行なわれた後、上のようなウィンドーがあらわれますが、ここでは“続行”ボタンをクリックして下さい。

そして少し待っていると、上のウィンドーがあらわれ、デバイスドライバーが正常にインストールされた事が報告されます。

実際にインストールが終了した後のデバイスマネージャの画面を例として示します。



１－５－２－１　ドライバーのアンインストール

WindowsXP でのドライバーアンインストールは多少の経験が必要です。全体の流れとしては

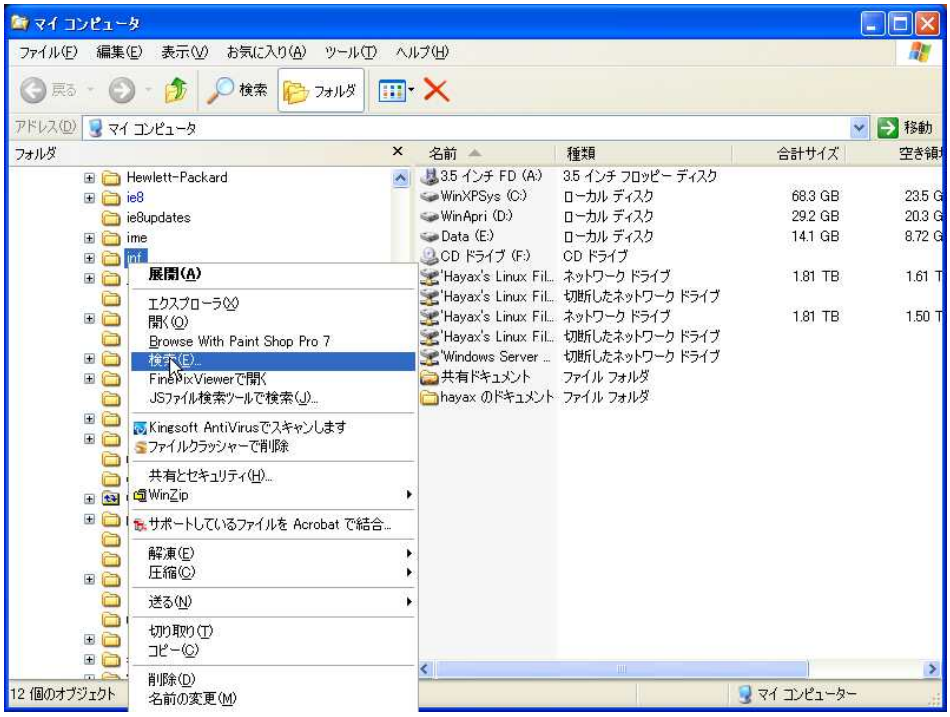
１　ハードウェア記述情報ファイル（inf ファイル）を探し、そのファイルと共に pnf ファイルをパソコンから削除

２　ドライバーファイル（Fcad416.Sys）及びドライバーインターフェースファイル（Fcad416.dll）を探し削除

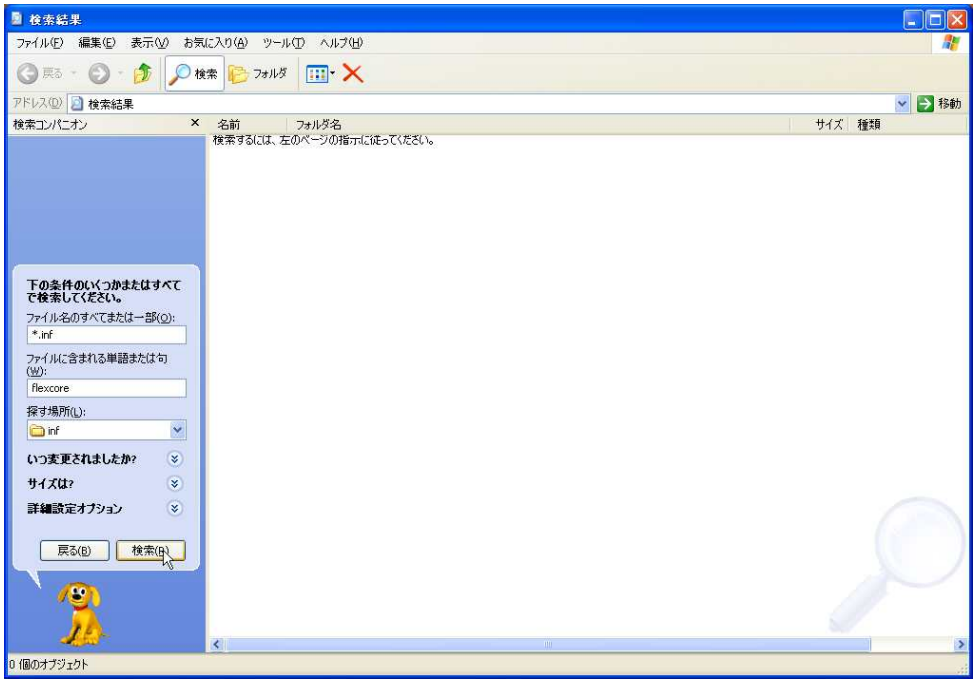
の 2 段階からなります。尚、以下の処理を開始する前に Fcad416USB をパソコンから外しておく事が必要です。

１－５－２－１－１　ハードウェア記述情報ファイルの探索

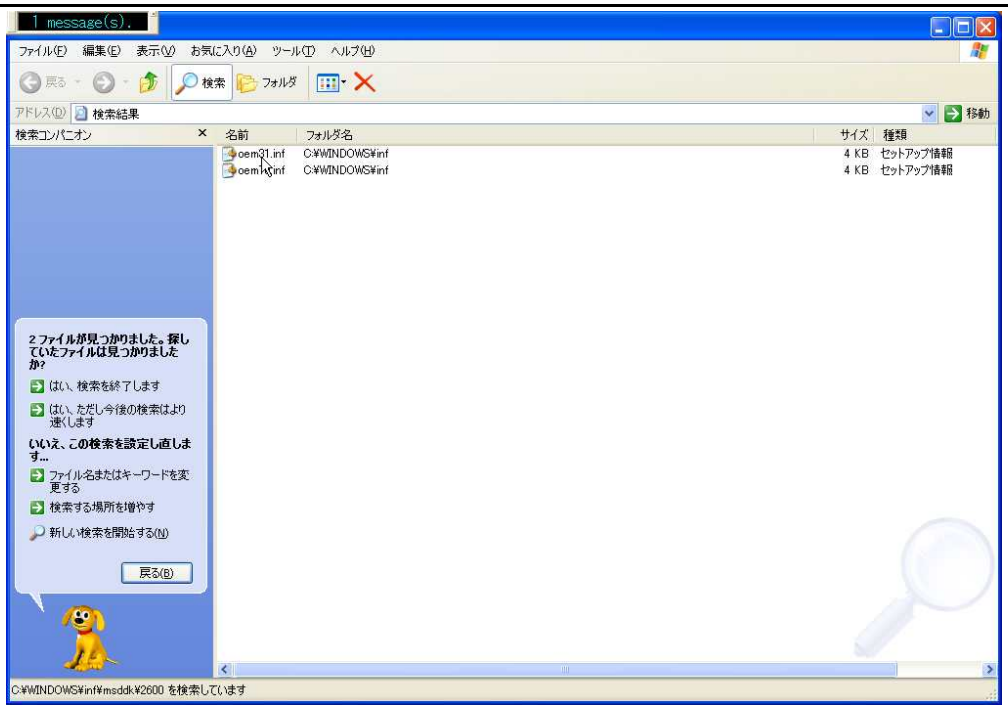
エクスプローラを開き、C ドライブの Windows フォルダ更にその中の Inf フォルダを右クリックし、ファイル検索メニューを呼び出します。下図を参照して下さい。



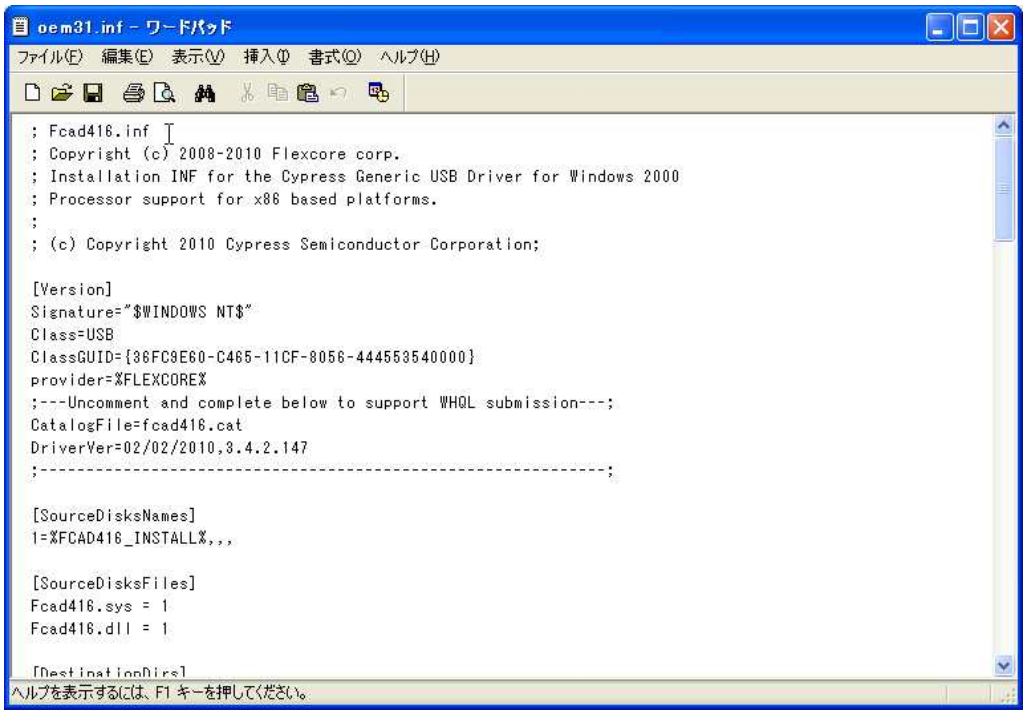
検索ウィンドーには次の図に示す条件を指定します。つまり、検索ファイル名は“*.inf”、ファイルに含まれる言葉として“flexcore“を指定することになります。



そして検索を実行させると（暫く時間が必要ですが）次のような画面が現れます。（下図参照）



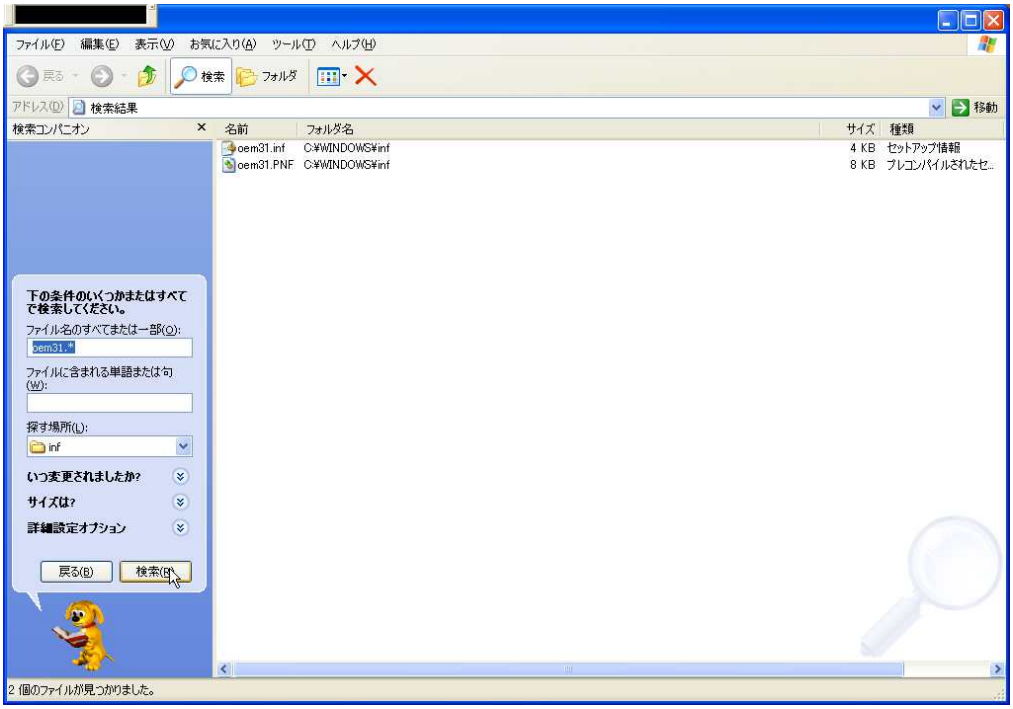
本検索例では、二つのファイルがヒットしていますが、一方は現在開発中の別ユニットのものですので通常は一つのファイルのみヒットする筈です。また、この時点で念のためテキストエディター（メモ帳など）で検索したファイルを開き、その内容を確認しておく事が重要です。下図にその表示例を示します。



正当な inf ファイルであれば、上の例のようにファイルの先頭に“Fcad416.inf”という形でハードウェア情報が記載されていますので、これを確認してから次のステップへ移行しなければなりません。ここで間違ったファイルを削除してしまうと、そのファイルがサポートしていたハードウェアが正常に動作しなくなる事も考えられ、このような場合には、対応するハードウェアのドライバーをもう一度インストールするか、場合によってはＯＳの再インストールまで戻らなければならない事もあり得るので充分な注意が必要です。

そこで今度は、ヒットした inf ファイルと同名の pnf ファイルも同時に消去するため検索条件を若干変えて再度検索します。その条件は、ファイル名として oemxx.*（ここで xx は前のステップでヒットした inf ファイルのファイル名です）を指定し、ファイルに含まれる単語または句の項目を削除します。

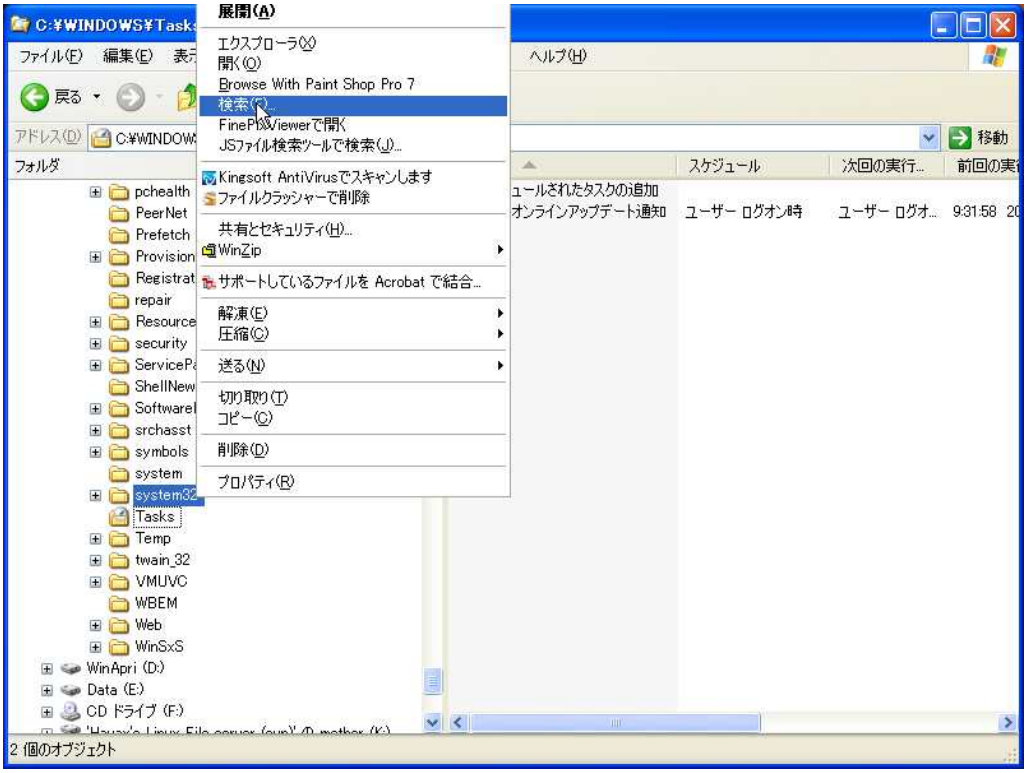
そして検索を実行すると次のような画面が現れます。



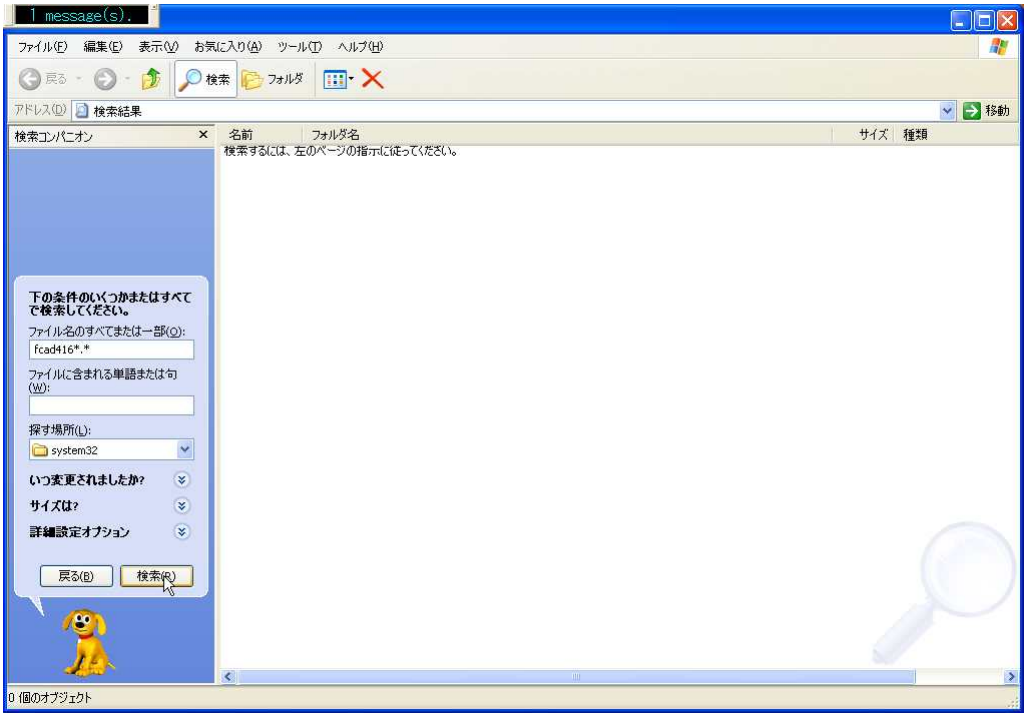
そこでこれら二つのファイルを選択しツールバーメニューのファイル削除“×”を使用して削除します。
またはこれらのファイルを選択した状態で“Del”キーを押す事で削除します。

１－５－２－１－２　ドライバーファイル及びドライバーインターフェースファイルの削除

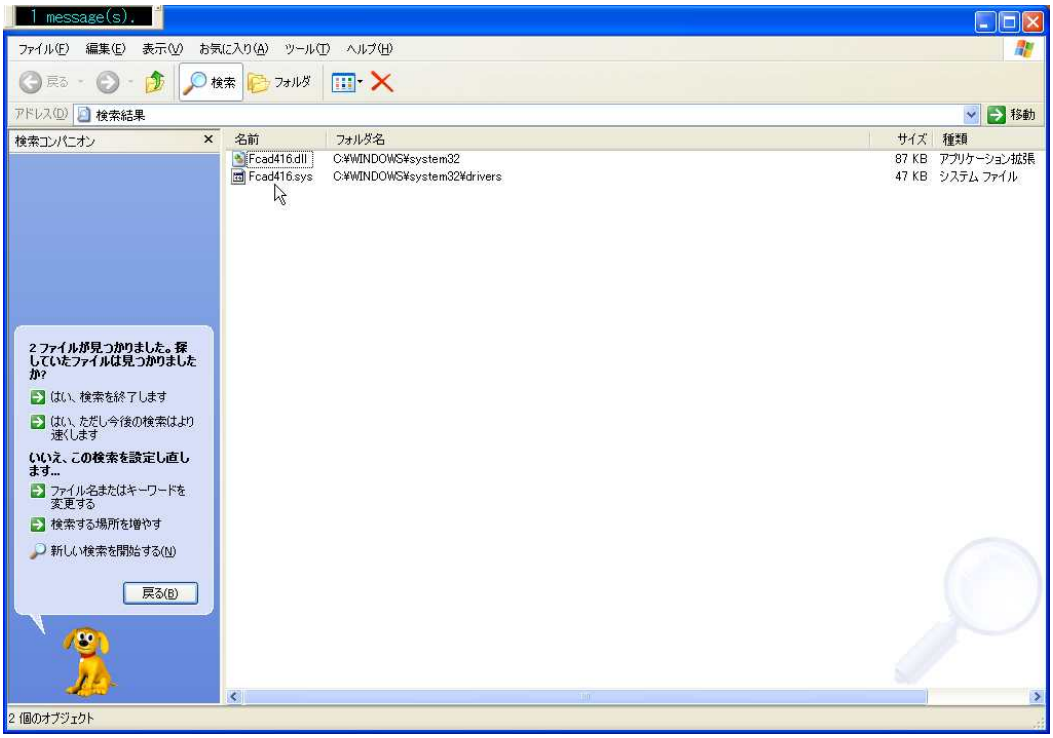
今度は、同じエクスプローラ画面で¥Windows¥System32 フォルダに対し同じように検索を実施します。その方法は下図を参照してください。



検索条件は、ファイル名称として“fcad416.*”と指定するだけです。実際の画面を次ページに示します。



この条件で検索を行うと、暫く経った後検索結果が現れます。



この場合もこれら二つのファイルを選択しツールメニューの削除ボタン“×”を使用して削除します。
この後、必要であれば新しいドライバーをパソコン内に展開し再びFcad416のインストールが可能となります。

注意点

複数のユニットを使用される場合でもインストールは１台ずつ追加しながら行って下さい。この時点では、ユニットＩＤは全て出荷時のままでも問題ありません。（しかし、ユニットＩＤの重複がないかどうかをハンドラーＤｌｌで検出しているため、サンプリングプログラムを動作させる際には適切な設定が必要です。仮に、全て“０”のままで動作させようとすると、ハンドラーの初期化関数がエラー終了してしまい、アプリケーションの終了以外何もできません。）
また、パソコンとの接続については、ＵＳＢ２．０対応のセルフパワーハブを使用する事もできます。（ポート当たりの供給電流が５００ｍＡあるものに限ります。）
この時点で、本ユニット制御用デバイスドライバー（Fcad416.SYS）とハンドラー関数ライブラリＤｌｌ（Fcad416.DLL）が所定のフォルダにコピーされます。

１－６．動作確認・試運転

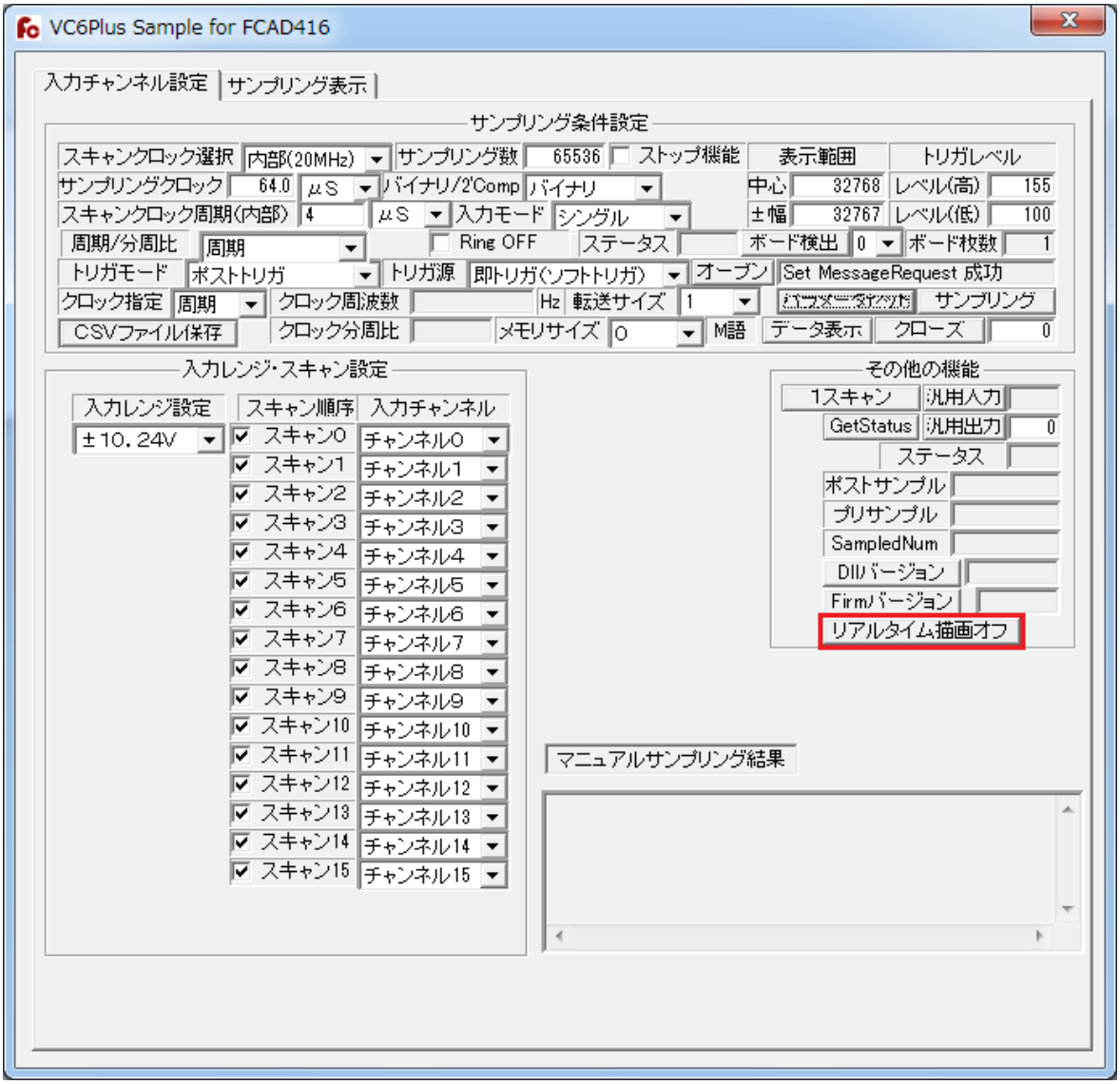
あらかじめダウンロード、解凍していただいたソフトウェアセットには実行ファイルが用意されています。
この実行ファイルを使用して頂くと動作確認を行うことができます。

操作手順

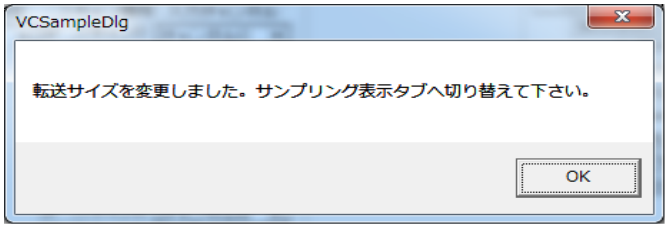
複数ユニットを使用頂く場合であっても、最初は１ユニットだけで動作確認をされる事をお勧めします。
ドライバーのインストールが完了したパソコンに、本ユニットを１台接続し、ＬＥＤ１が点燈状態になるのを待ちます。最低限の動作確認はこの状態でも可能ですが、実際のアナログ変換の様子を確認するには、最低でも定電圧源、できれば発振器があると尚良いと思います。これらの信号源をＣＮ１のチャンネル０（１ピン）及びグランド（２０ピン）に接続します。

試運転

展開したソフトウェアセットの中から¥Flexcore¥Fcad416¥動作確認¥x86フォルダ(32ビットOSの場合)若しくは、¥Flexcore¥Fcad416¥動作確認¥x64フォルダ(64ビットOSの場合)をエクスプローラで開き、その中にあるVC6Plus.exe又はVC6Plus_64.exeを開きます（またはダブルクリックします）。下図は32ビット版の例です。



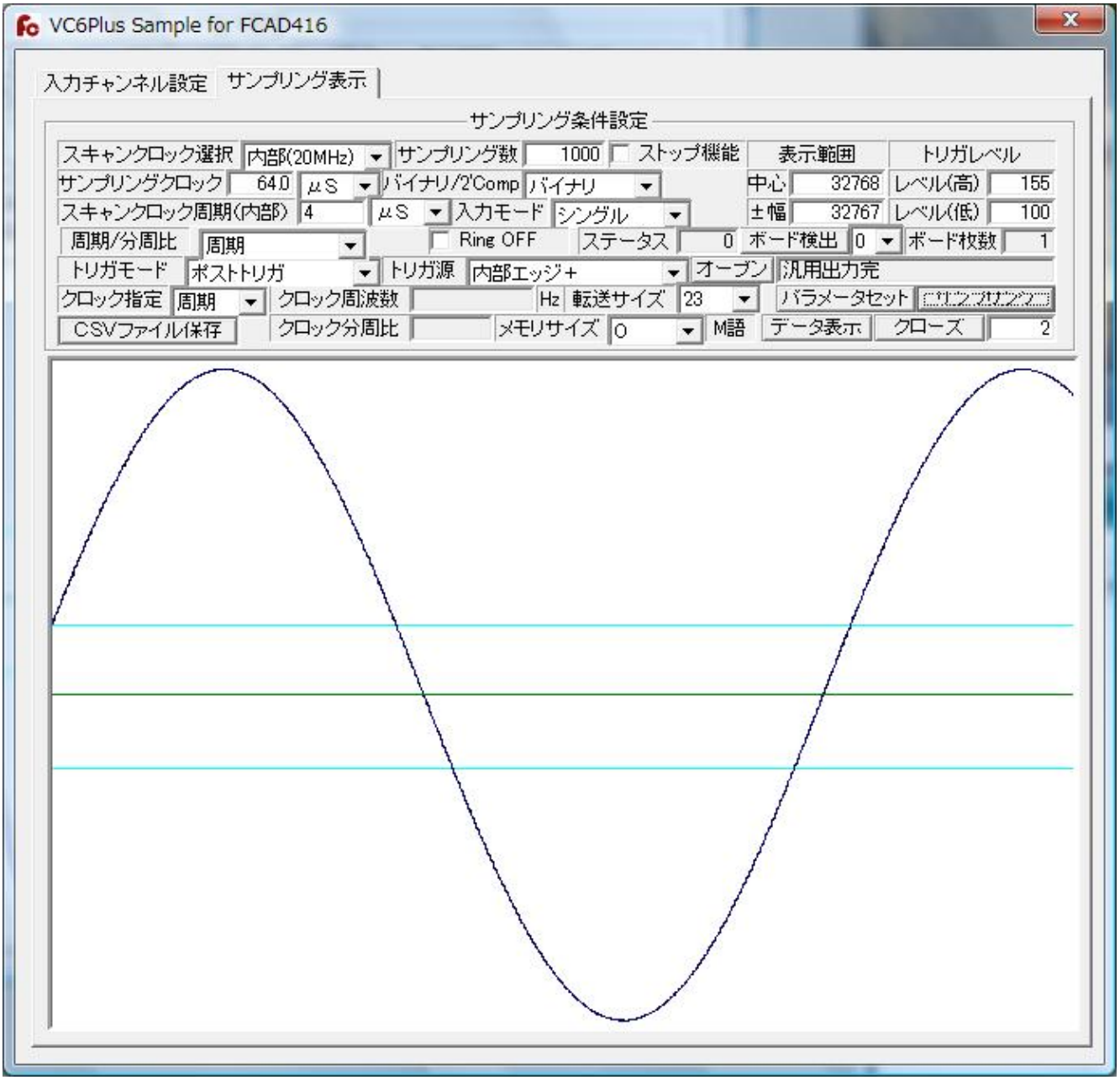
まず、画面右側にある“ボード検出”ボタンをクリックし、ボード検出枚数が１となって正常終了する事を確認します。また“ボード検出”ボタンの右側にあるドロップダウンリストには、検出されたユニットのIDが全て表示されている筈なので、ここに“０”という項目が一つだけあることも確認して下さい。（もしＩＤ＝１のユニットも存在していれば、“１”という項目も存在します。そしてボード検出枚数が２になります。）
次に、その下にある“オープン”ボタンをクリックし正常終了する事を確認した後、更に下にある“パラメータセット”ボタンをクリックします。ここで、（ポストトリガモードであれば）リアルタイム描画機能を利用する事も可能です。その為には、上図で赤枠で囲まれたボタン“リアルタイム描画オフ”ボタンをクリックして下さい。すると、次のメッセージが出現します。



メッセージの内容を確認したら、ここまでエラーがない事を確認し、上のタブボタンをサンプリング表示に切り替え、“サンプリング”ボタンをクリックします。すると時間の経過と共にサンプリングが進行し最終的にサンプリングが終了し画面下部のウィンドーにアナログ入力の変換結果が全て表示されるはずです。また、この後“ＣＳＶファイル保存”ボタンをクリックすると、ファイル保存画面が開き、名称を指定してサンプリングした結果をＣＳＶファイルとして保存することもできます。

ここまで、エラーがなく実行できれば、基本的な動作の確認は終了しています。その後は、いろいろなサンプリングモードや、マスタースレーブ運転など応用問題に進む事ができます。下図に実際に上記のプログラムを実行した結果を示します。

実行例

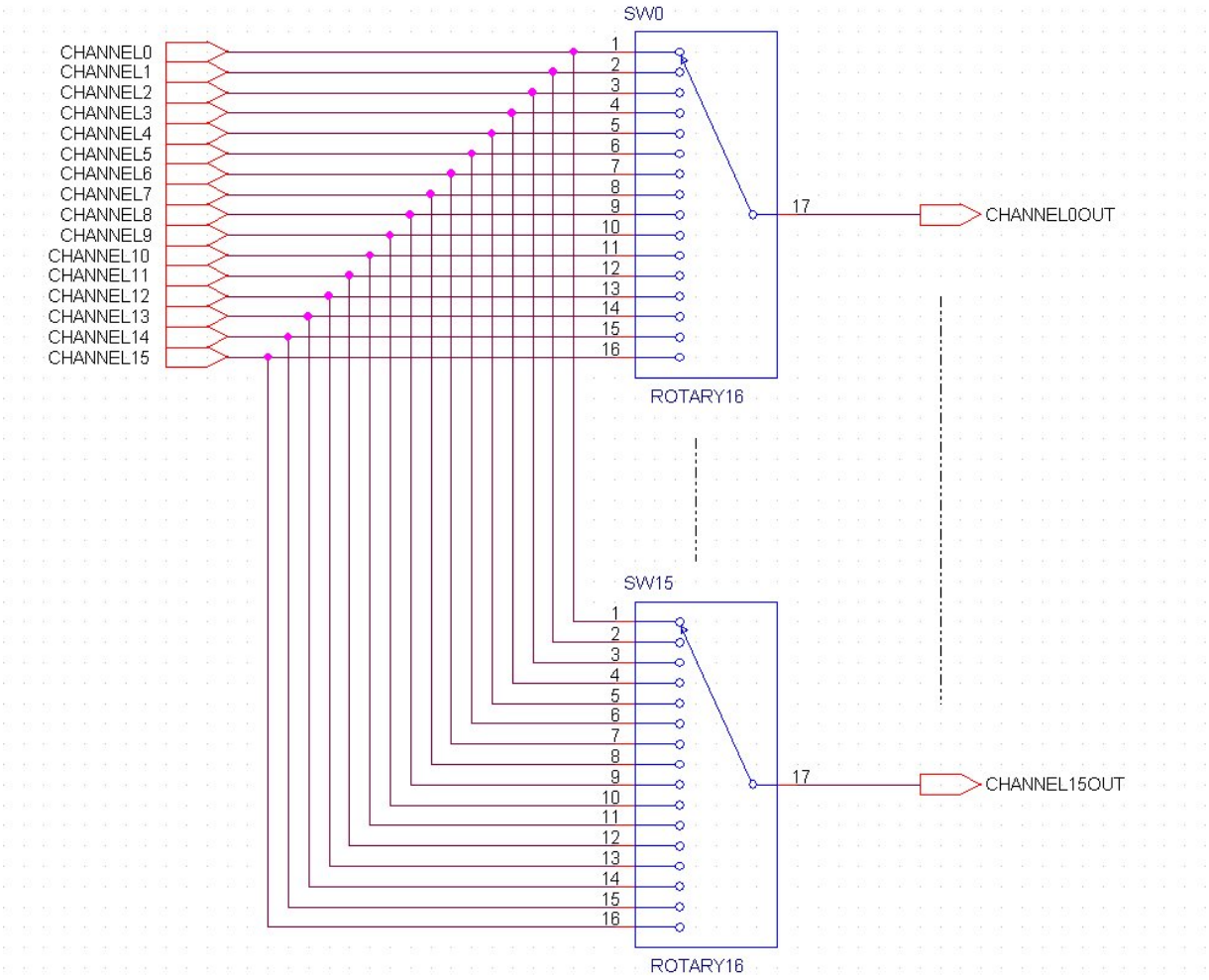


※　本動作例では、アプリケーション起動時の設定を一部デモンストレーションのため変更しています。
　　変更箇所は、
　　　１　サンプリング数（６５５３６点から１０００点へ）
　　　２　トリガ源（即トリガから内部エッジ＋へ）
　　です。

第２章. 信号入出力

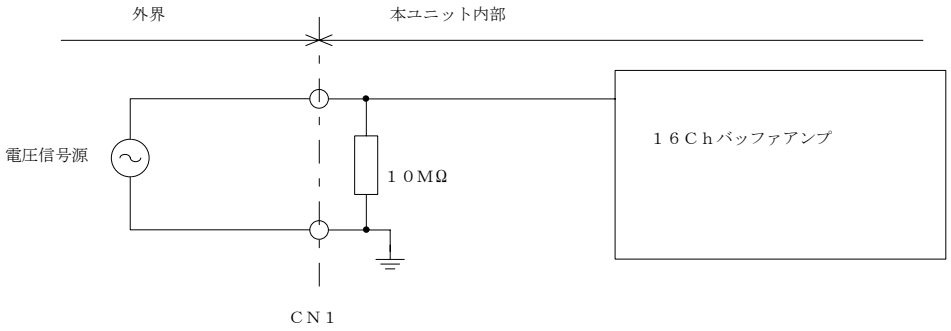
２－１. アナログ入力回路

本ユニットのアナログ入力～ＡＤ変換回路は各チャンネルを任意の順序で走査しＡＤ変換を行う逐次変換方式です。

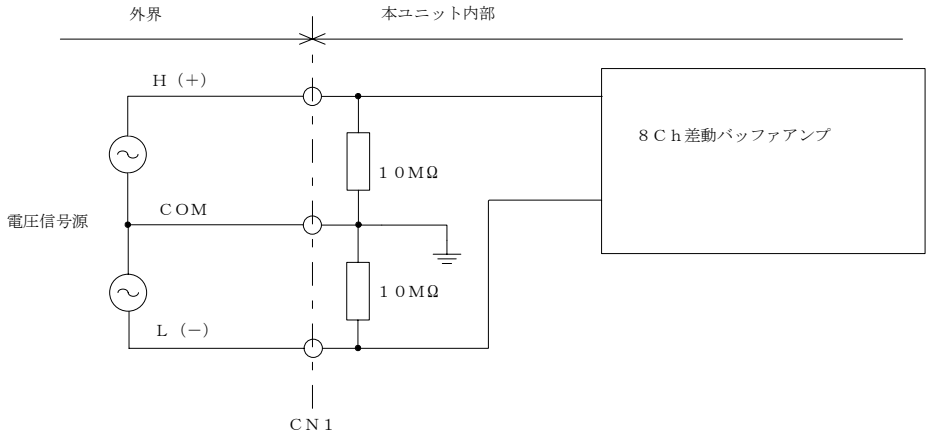


本ユニットでは、実際のアナログ入力部は上図のように、入力チャンネル毎に独立してソフトウェア設定によって実際の入力端子と接続されます。これにより、入力結線を変更する事無くトリガ入力を変更する事ができます。
アナログ入力範囲の最大レンジは±１０．２４Ｖ、絶対最大定格は－３５Ｖ～＋３５Ｖです。 これ以上の電圧が印加される恐れがある場合は保護対策が必要です。（アナログ入力特性の項を参照して下さい。）
なお、各チャンネル入力端には入力インピーダンスを下げるために１０ＭΩの終端抵抗が実装されています。
また、上の図では省略されていますが、実際には各入力にはバッファアンプが実装されており、高インピーダンス信号源に対応できるようになっています。次ページに、実際のシングルエンド入力と差動入力夫々の結線方法を示します。

シングルエンド電圧入力の接続（１チャンネル分）



差動電圧入力の接続（１チャンネル分）



２－２．アナログ入力範囲と伝達関数

本ユニットの入力範囲・モードは、ソフトウェア設定になっています。
選択方法は、制御方法の説明項およびサンプルプログラムを御参照下さい。
なお、シングルエンド入力と差動入力の混在はできません。仮に混在させようとすると、最後の指定が有効になります。
また、本ユニットは全ての入力組み合わせに対し調整されているため、入力範囲を切り替えても再調整の必要はありません。

アナログ入力範囲

本ユニットは16ビットの分解能を持っています。従ってその分解能は“２の１６乗分の１”ですから、変換データとアナログ入力電圧の関係は以下のようになります。

分解能　　Ｒｅｓ＝Vspan÷65536　［V／digit］
変換データ　Dad＝（Vio÷Res）＋32768　［digit］
入力電圧 Ｖio＝（Dad－32768）×Res　［v］

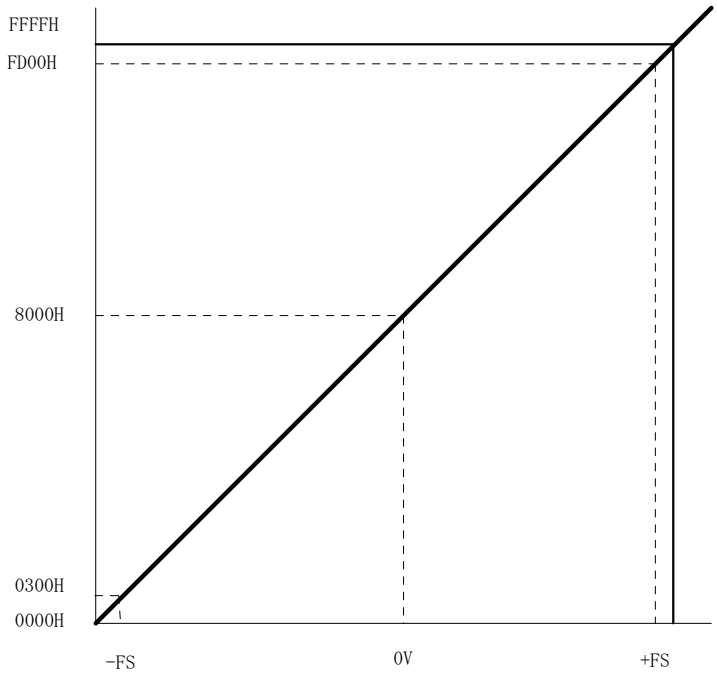
【注14】　Vspan は入力範囲の絶対幅です。 具体的には表２－２の実際の入力範囲（計算値）に1digit 分の電圧値を加算した値です。

公称入力範囲 (Vspan)	実際の入力範囲（計算値）	入力電圧（有効範囲）注１	分解能 (Res)	正確度
±１０．２４Ｖ	-10.24V～+10.23969V	-10.000V～+10.000V	0.3125mV	±０．０２％ＦＳ
±５．１２Ｖ	-5.12V～+5.11984V	-5.000V～+5.000V	0.15625mV	
±２．５６Ｖ	-2.56V～+2.55992V	-2.500V～+2.500V	0.078125mV	
±１．２８Ｖ	-1.28V～+1.27996V	-1.250V～+1.250V	0.0390625mV	

注１　有効範囲とは、回路内使用素子の特性により入力電圧の精度が保証される範囲を指します。

例えば、入力レンジが±１０．２４Ｖの場合に１０Ｖの入力電圧が印加されると夫々の値は次のようになります。
Ｒｅｓ＝２０．４８／６５５３６＝０．３１２５ｍＶ
Ｄａｄ＝１０．０Ｖ／０．３１２５ｍＶ＋３２７６８＝６４７６８（＝ＦＤ００Ｈ）
或いは
Ｄａｄ＝６４７６８ならば
Ｖｉｏ＝（６４７６８－３２７６８）×０．３１２５ｍＶ＝３２０００×０．３１２５ｍＶ＝１０Ｖ
となります。

伝達関数（バイナリフォーマット）



ここで－ＦＳ、＋ＦＳとは上表で示す有効な入力電圧の上下限を示すものです。また、アナログ変換値の中で００００Ｈから０２ＦＦＨ、及びＦＤ０１ＨからＦＦＦＦＨの範囲に対応する入力電圧については、本ユニットで使用しているアナログ回路素子の精度定義域を超えているため、精度を保証することができません。
本ユニットで精度が保証できる入力電圧範囲は、±ＦＳの間となります。

２－３．アナログ入力特性

ＡＤ変換誤差

本ユニットのAD入力は±１０．２４Ｖ、±５．１２Ｖ、±２．５６Ｖ及び±１．２８Ｖの範囲で調整されています。その為、入力感度を切り替えても再調整の必要はありません。

温度ドリフト

本ユニットの周囲温度が変化したとき、同一アナログ入力に対するＡＤ変換データが変化する度合いを示します。本ユニットでは、１０ｐｐｍ／℃(typ)です。

経年変化

経年変化のデータはありません。充分な精度の維持が必要な用途では、念に１～２回程度、標準電圧源及び充分な精度を持った電圧測定器を使用して校正し、必要に応じて再調整する事が望まれます。

内部雑音

本ユニットの内部雑音は、各チャンネルの入力端をアナログ・グランドに接続してみれば見当が付きます。（実使用状態に応じて変動する要素があります）
本ユニットでは±１０．２４Ｖレンジで±１ＬＳＢ(typ)
±１．２８Ｖレンジで±４ＬＳＢ（typ）となっています。

入力耐圧

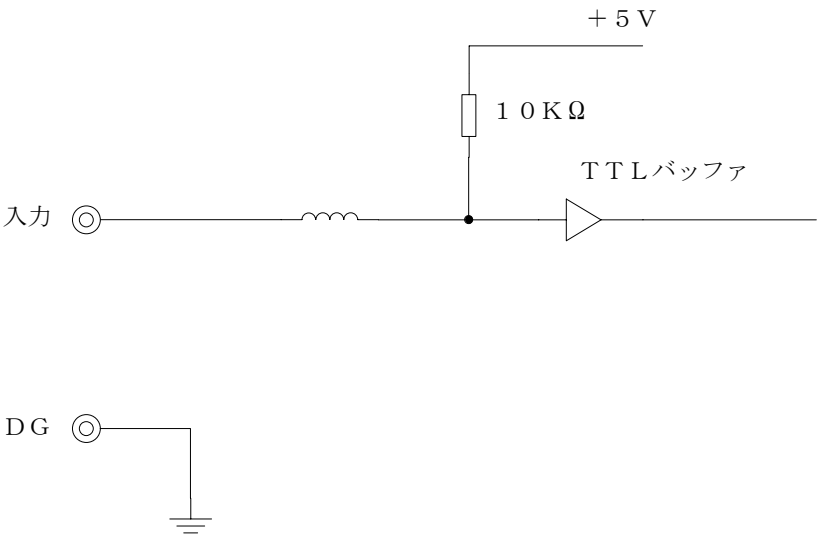
本ユニットのアナログ入力回路は±３５Ｖまでの過電圧に対して保護機能を有していますが、入力電圧がこの値を超えると、入力保護回路に使用しているＩＣが破壊されてしまいます。そのため入力回路に±３５Ｖを超える電圧が印加される可能性がある場合には、本ユニット外部に過電圧対策回路が必要です。
回路方式は色々考えられますが、代表的なものは抵抗（電流制限用）と直列ツェナーダイオードを組み合わせたものだと思います。印可される最大電圧とツェナー電圧とから抵抗値とサイズ（消費電力）を、保護したい回路電圧からツェナー電圧を決定します。
ツェナー電圧は、３５Vを下回る必要があるため、そのばらつきを考慮して通常１２Ｖ～１５Ｖ程度とするのが一般的です。

また、抵抗値をＲiとすると
Ri≦（Vmax－Vzmax）/Izとなります。
ここで
Vmax：最大印可電圧
Vzmax：ツェナーダイオードの最大ツェナー電圧
Iz：ツェナー電圧を発生させるために必要なツェナー電流、ツェナーダイオードのデータシートから
　決定するが通常２～３mA程度
この回路の注意点としては、仮にVmaxが１００Ｖだとすると
１　ツェナーダイオードで消費される最大電力（≒50mW程度->この場合実際には250mW乃至500mWクラスのダイオードが必要）
２　抵抗で消費される最大電力（≒300mW程度：最大電圧が100Vとして->この場合実際には１Ｗクラスの抵抗が必要）
３　保護機能が働かない状態でもツェナーダイオードに数μＡ程度の漏れ電流が流れRiでの電圧降下が入力電圧値に影響する
　等が考えられますので、さまざまな面からの考察が必要です。

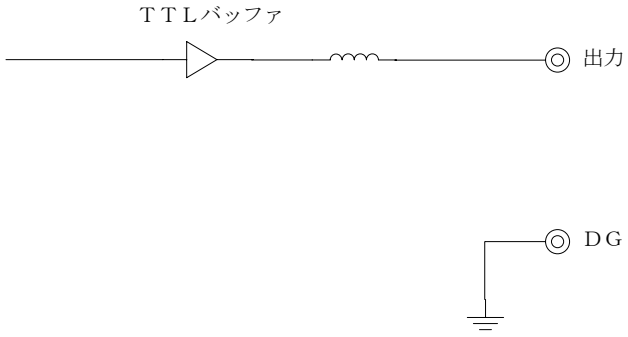
２－４．デジタル入出力回路

外部クロック入力、外部トリガ入力、サービスリクエスト入力、外部ストップ入力、汎用４ＢＩＴデジタル入力、汎用４ＢＩＴデジタル（ラッチ）出力は全てＴＴＬレベルです。 入力は全て１０ＫΩでプルアップされています。なお電源投入直後のデジタル出力は“１”となります。（出力極性設定＝負論理の場合）
また、全てのデジタル入出力ラインにはノイズ混入を防ぐため、フィルターが挿入されています。

全てのデジタル入力



ＴＴＬ入力の絶対最大定格は負側－０．５Ｖ、正側＋７Ｖです。この値を一瞬でもオーバーすると素子破壊につながるので注意が必要です。



デジタル出力の論理はユニット内のショートプラグＪＰ３によって設定します。出荷時は“N”（負論理）になっています。

２－４－１　汎用デジタル入力

汎用デジタル入力Ｉ０～Ｉ３はＴＴＬレベルです。その使用方法は任意ですが、本ユニット内部には全く影響を与えません。パソコンのアプリケーション側に直接（厳密にはＵＳＢバスの通信時間経過後）読み取られます。

２－４－２　汎用デジタル出力

汎用デジタル出力素子Ｑ０～Ｑ３はＴＴＬレベルです。なお出力論理はボード上のジャンパーＪＰ３（出荷時：負論理）で選択できます。この信号についても、本ユニット内部には全く影響を与えません。パソコンのアプリケーションから直接（厳密にはＵＳＢバスの通信時間経過後）制御されます。

２－４－３　制御信号入力

制御入力はサンプリング制御用３点と、サービスリクエスト入力１点から構成されています。サンプリング制御用３点の入力のうち、外部クロック入力と外部トリガ入力の２点については、文字通りの意味（外部クロック、及びトリガ外部トリガ）と、マスタースレーブ接続時のスレーブユニット入力という意味の二重の意味合いがあります。

マスタースレーブ接続時には、マスターユニットの同期クロック出力が、外部クロック入力に接続される一方マスターユニットの同期トリガ出力が外部トリガ入力に接続されることでマスタースレーブ間の同期を取ることができます。詳細については、マスタースレーブ動作の説明を参照して下さい。

外部ストップ入力は、自動サンプリングを中断する機能を受け持っています。これは、無限サンプリングであっても、有限サンプリングのサンプリング中であっても有効ですが、特に無限サンプリング動作時停止条件として指定する事でサンプリング終了をメッセージ処理によりアプリケーションに通知する事ができるため有効であると思われます。しかしこの機能は、サンプリング開始以前に有効・無効の設定を行っておかなければなりません。

また、サービスリクエスト入力は、本ユニットの外部からサービスを要求する機能を実現しています。

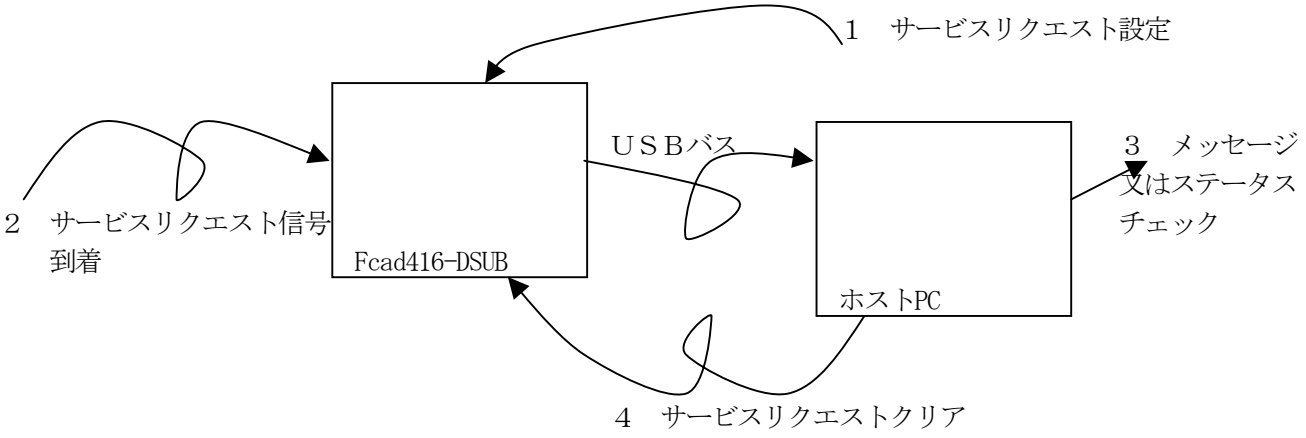
この機能は、ＵＳＢのバス通信機能を利用して（さほど遅滞ない時間差で）この入力の変化をアプリケーションに（メッセージ通信等の形で）通知する事を目的として実装されています。

また、サービスリクエスト設定以前に印可されたサービスリクエスト及び一旦ＰＣ側へ報告がなされた後新たにサービスリクエスト設定がなされる前のサービスリクエスト再入力は無視されます。

更にサンプリング中のメッセージ応答は、ＵＳＢバス転送の合間に行われるため、４－５－【16】項で説明したパラメータをどの値にするかによって応答レスポンスが変わってくるので注意が必要です。

実際のＵＳＢバス転送時間は、理論的に求める事は困難ですが、実動作時の測定結果によると概ね２０ｍＳ程度となっています。（ポストトリガサンプリング、１台動作時：尚ＵＳＢバス転送のパケット長が８Ｋ語の場合。この場合ＵＳＢバス転送速度は概ね８Ｋ語／２０ｍＳ≒４００Ｋ語／秒となります）

また、複数台を同時に動作させる場合はラウンドロビン方式の制御になっているため、概ね上記時間×台数分の時間がかかります。



サービスリクエスト処理の流れ

２－４－４　制御信号出力

制御信号出力には、同期クロック出力と同期トリガ出力の２系統が準備されています。これらの信号は、マスタースレーブ接続動作を行なう際、マスターユニットからスレーブユニットの制御用に出力されるものです。
詳細な動作、機能については、３－14　各種サンプリングにおけるタイミング説明　を参照して下さい。

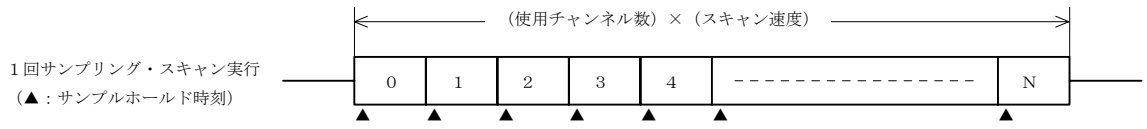
第３章. 制御・操作

３－１. ＡＤサンプリング動作・トリガ動作の様子

本ユニットには大別して３種類のサンプリングモードがあります。
何れの場合も、サンプリングされた結果のＡＤデータ（２バイト構成）は順番にＦＩＦＯバッファ、或いはリングバッファ構成となっているプリトリガバッファに書き込まれていきます。パソコン側からは、本ユニットのステータスを常時監視し、適宜ＡＤデータを読み込んでゆきます。
本ユニットには、ＦＩＦＯバッファ、或いはプリトリガバッファとして、合計で８Ｍデータ分が準備されておりＵＳＢバスの転送速度も、２Ｍワード／秒程度は確保されているため、通常は本ユニットの性能を充分に発揮することができます。（２Ｍワード／秒という制限は、ＵＳＢバスそのものからくるものではなく、本ユニット内部のアーキテクチャからの制限になります。）
但し、表示やその他の演算等を含む応用ではマルチタスク動作となるため、それらの処理時間次第で、実現可能な最高速度が決まります。

マニュアルサンプリング

指定したアナログ入力チャンネル群に対して１回だけＡＤサンプリングを実行するものです。チャンネル０を先頭に指定し順番に最終指定チャンネルまで自動実行します。アナログ入力を順次切り替えてＡＤ変換するため、各チャンネルのサンプリング実行時刻に一定の差（最小４ｕＳ、最大３．２６ｍＳ）が生じます。
また、本ユニットでは、論理入力チャンネルに任意の実チャンネルを対応させてサンプリングを行うことができるため、例えばチャンネル０を３回サンプリングして次にチャンネル４を２回、といった変則サンプリングも可能です。（但し最大で合計１６回までという制限が付きます。）



スキャン速度＝最短４ｕＳ、最長３．２６８ｍＳ（プログラマブル）

操作手順
1　制御部リセット
2　動作条件設定　ＡＤデータコード、スキャン速度
使用チャンネル数
トリガモード
先頭チャンネル、最終チャンネル
3　動作開始　マニュアルサンプリング開始
4　ステータス監視　ステータス評価
5　ＡＤデータ取得　ＡＤデータを取り込み

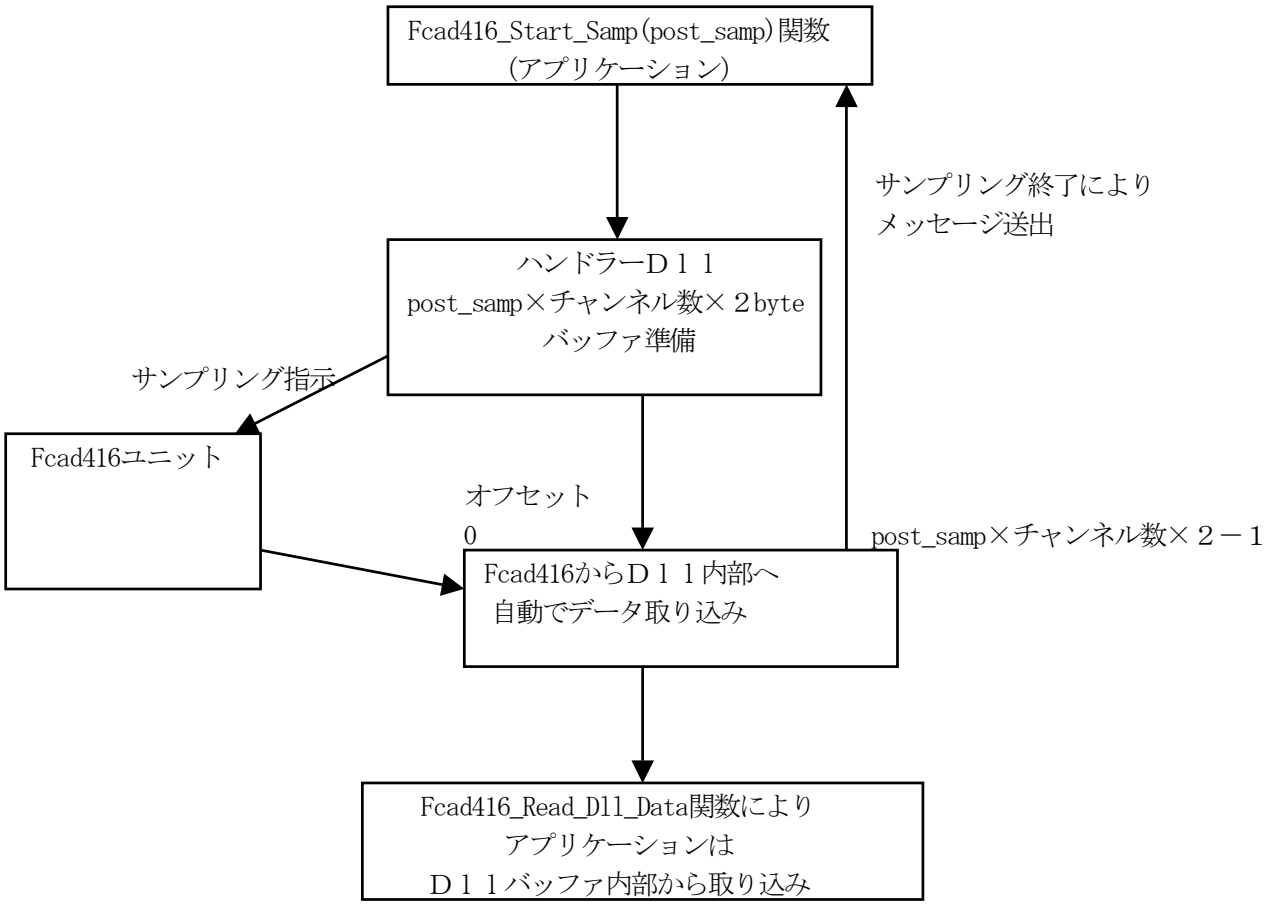
連続・自動サンプリング（ポストトリガ動作とプリトリガ動作）

本ユニットには２種類の連続サンプリングモードがあります。
(a) ポストトリガ・モードでは（ソフトによる即トリガを含む）トリガ発生により連続サンプリングを開始します。また (b)プリトリガモードでは連続サンプリングが開始された後、トリガ発生を待ちながらサンプリングを続け、トリガ発生以後は事前指定のサンプリング回数だけ追加実行して停止します。
いずれの場合もサンプリングされた結果のＡＤデータ（１語＝２byte 構成）は順番にバッファメモリに書き込まれて行きます。パソコン側からは本ハンドラーがバッファメモリの充満状態を参照しながらＡＤデータをＤ１１専用バッファに読み込みます。ユーザーアプリケーションからはReadDll関数を使用して必要な部分をＤ１１専用バッファから読み込みます。

ポストトリガ連続・自動サンプリング

指定したアナログ入力チャンネル群に対して指定したトリガ発生以後、指定のクロックでADサンプリングしＦＩＦＯバッファメモリに転送を連続・自動的に実行するものです。

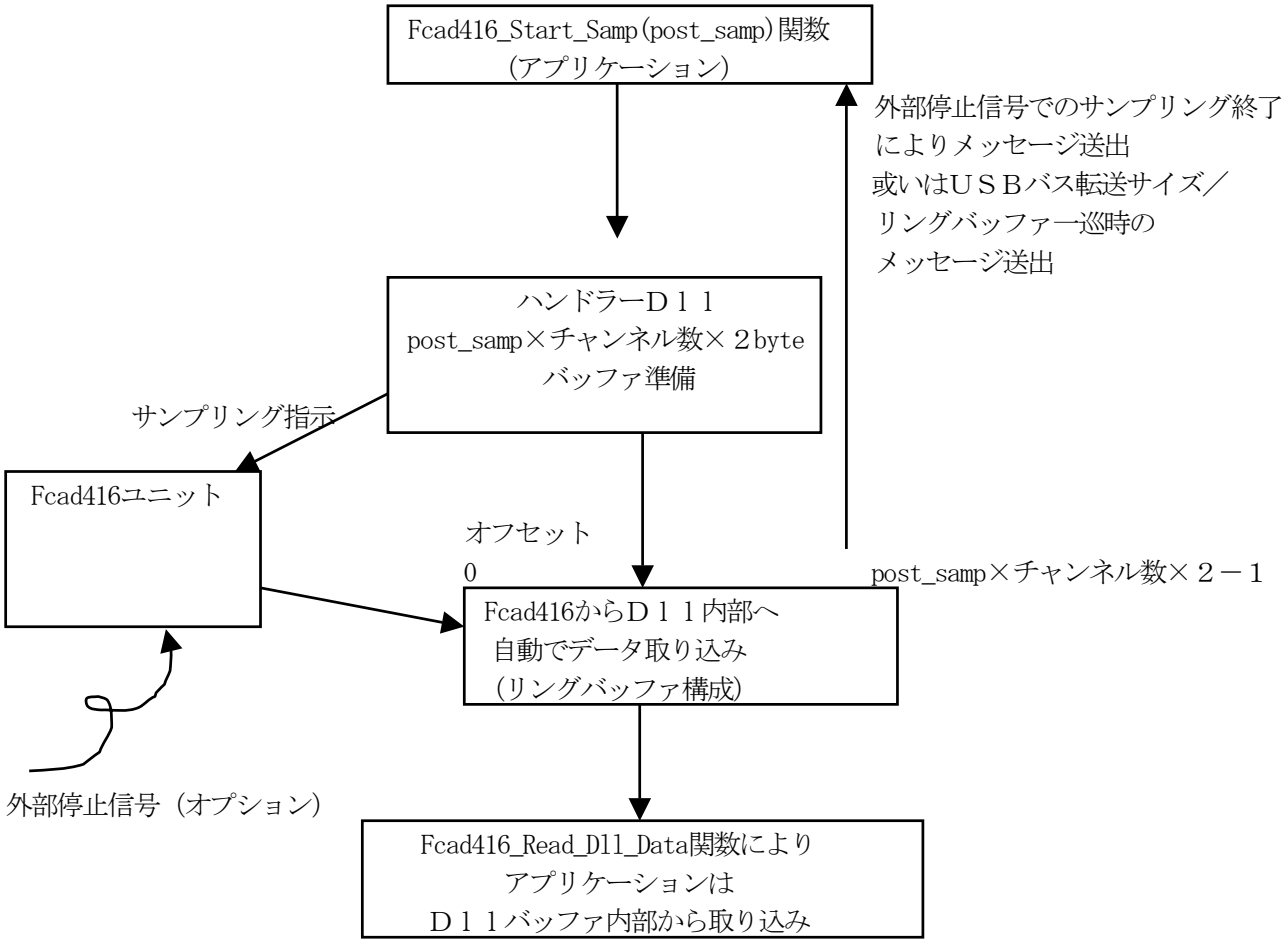
有限サンプリング動作時のハンドラーＤＬＬとアプリケーションの動き



アプリケーションからは、Fcad416_Start_Samp(post_samp)関数によりサンプリング開始が指示されます。これにより、ハンドラーＤＬＬは、ユニット毎にpost_samp×チャンネル数×２（バイト）のエリアをポストトリガサンプリングデータ保持エリアとして確保し、Ｆｃａｄ４１６ユニットにサンプリング開始を指示します。これ以降ハンドラーＤＬＬはＦｃａｄ４１６ユニットと適宜交信し、ユニットの状態を判断しながらサンプリング済みデータを取り込み、先に確保したバッファ内に書き込んでゆきます。そして、指定したサンプリング数に達すると、ハンドラーＤＬＬから、メッセージがユーザーアプリケーションにポストされ、アプリケーション側はサンプリング終了を検出します。その後、アプリケーションは、上記バッファの内部、任意のオフセット位置から任意の数のサンプリングデータを（上記バッファのエリア内という条件を満たしている限り）Fcad416_Read_Dll_Data関数により取り込む事ができます。

或いは、（サンプリング速度にもよりますが）トリガ検出でメッセージをポストさせ、それ以降Fcad416_Get_Status関数によりサンプリング済みデータ数を確認しつつ読み込むという方法でも対応する事ができます。

無限サンプリング動作時のハンドラーＤＬＬとアプリケーションの動き



無限サンプリングの場合は、サンプリングを開始する前にFcad416_Set_SampLoop関数を使用してＤＬＬバッファをリングバッファとして使用する宣言をしておきます。これによりＤＬＬ内部に用意されるバッファはリングバッファとなり無限サンプリングに対応するバッファ処理が可能になります。本ユニットは、３－１３－１で示すように、パソコンを選択することで最大１２台程度の同時最高速無限サンプリング（４μＳ／チャンネル）が可能ですので、この時外部停止信号を併用するようにサンプリング条件を設定することができれば、この停止信号によるサンプリング終了をアプリケーション側で検出しデータ処理を開始することができます。或いは（低速サンプリングの場合特に有効ですが）一回のサンプリングスキャンで得られるデータ数を、ＵＳＢバス転送サイズ（４－５－【16】参照）と合わせておくことができれば、サンプリングスキャン周期毎に、アプリケーションへのメッセージが送られてくるため、データ処理のタイミングを確定することも可能です。

実際に、アプリケーションからの指示でサンプリングを開始すると、ハンドラーＤＬＬは、Ｆｃａｄ４１６ユニットと適宜交信し、サンプリングデータを取り込んでゆきます。そしてＵＳＢバス転送サイズ分のデータを取り込んだ時点またはＤＬＬバッファラップラウンド時、或いは、外部停止信号によりサンプリングが完了した時点で、アプリケーション側へメッセージがポストされてくるので、アプリケーション側は、このタイミングでFcad416_Get_Status関数によって転送済みポストトリガ後サンプリングデータ数を確認し、Fcad416_Read_Dll_Data関数を使用してサンプリングデータを読み込みます。（ここで、ＵＳＢバス転送サイズ分のデータ取り込みメッセージでは、サンプリングデータ数の確認は不要です。）

サンプリングスキャン毎のメッセージ処理のように、継続してサンプリングデータを取得するアプリケーションはFcad416_Get_Status関数を使用して現在の取得データのＤＬＬバッファ内での位置を判断します。この関数で返されるpost_sampled変数の値は読み込まれたデータの数ではなく、最後にＤＬＬバッファに取り込まれたデータのバッファ先頭からのオフセットを示しているため、実際に読み込まれたデータ数は前回のFcad416_Get_Status関数で返されたpost_sampled変数の値との差分になります。また、このバッファはリングバッファ形式となっているため、書き込

まれたポインタが一巡すると前回のポインタ位置よりも手前の位置を示している事になります。この場合は、

１　まず、ステータスを確認して無限ループラップラウンドフラグがセットされていることを確認し

２　次に、前回のポインタ位置からバッファ末尾までのデータを取り込み

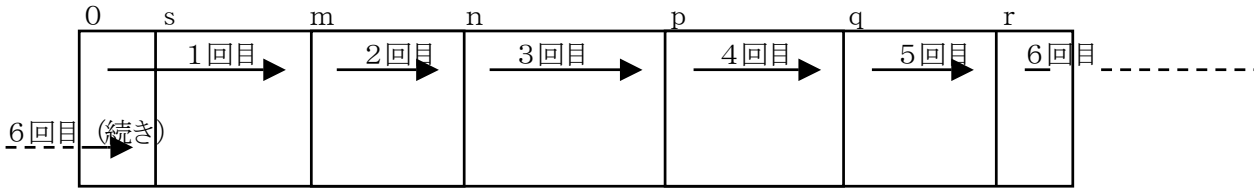
３　それから、バッファ先頭から今回のポインタ位置までのデータを取り込み

４　最後に無限ループラップラウンドフラグをクリアする

という処理を行う必要が生じます。

下図は、実際にＤｌｌバッファからデータを取得する際の方法について模式的に示したものです。

無限サンプリング動作時のＤｌｌ内部バッファの動き



上の図において、ｍ＜ｎ＜ｐ＜ｑ＜ｒの順にFcad416_Get_Status関数のpost_sampled変数の値が返され、その次の関数実行の結果返される値がｓと仮定すると

１回目のデータは、オフセット０からオフセットｍ－１まで

２回目のデータはオフセットｍからオフセットｎ－１まで

というように処理を進め、６回目のデータは、まずオフセットｒからバッファ末尾までを取り込み、更にオフセット０からオフセットｓ－１までを取り込む、という処理になります。

ハンドラーＤｌｌとアプリケーションの動きそのものについては、バッファの管理がリングバッファ形式になる事以外有限サンプリング処理と本質的に変わるところはありません。しかし、サンプリングデータがＤｌｌバッファを上回った場合（所謂ロールアップ状態）次に述べるプリトリガバッファの場合とは異なりＤｌｌバッファからの読み出しには注意を払う必要があります。読み込む順序としては

サンプリング終了時に一括して読み込む場合

１　最後のFcad416_Get_Status関数で得られたpost_sampled変数の位置（ｎ）が最新のデータ位置なので、この変数の値＋１のところからＤｌｌバッファの末尾までをまず読み出し

２　Ｄｌｌバッファの先頭から１で得られたpost_sampled変数の位置までのデータを引き続いて読み込む

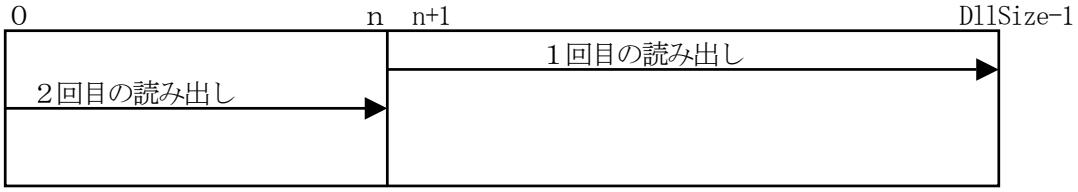
という処理になります。

また、例えば、スキャン周期毎にデータ処理を行なっているような場合には、データを読み込む毎に、このフラグもチェックしクリアもしておく必要があります。

尚、この時必要とされるロールアップフラグは拡張ステータスレジスタのビット３に反映されています。

詳細については４－５－【11】ステータス取得を参照して下さい。

ロールアップフラグセット時のＤｌｌ内部バッファの状態と読み出し順序



プリトリガ連続・自動サンプリング

指定したアナログ入力チャンネル群に対して指定クロックでADサンプリングし、ＦＩＦＯバッファメモリに転送を連続・自動的に実行することは前記のポストトリガ動作と同様ですが、プリトリガ動作の場合はスタート操作で即連続サンプリングが開始され、トリガが発生すると以後、事前設定のサンプリング回数だけ実行して終了します。なおＲAMバッファはリング状・無限ループ構造ですから、トリガ前データ点数の最大はＲAMバッファのサイズと同じになります。
こちらに関しても、ユニットからの読み出しはハンドラー内部で行い、ユーザーアプリケーションからはReadDirectRam関数を使用してハンドラー内部のバッファから必要な部分を読み込みます。
このサンプリング方式では、トリガ検出までが専用のリングバッファによる無限サンプリング、トリガ検出後が有限サンプリングの組み合わせとなっています。しかしリングバッファによる無限サンプリングの動作については、トリガ検出以前の最新データを取得したいという設計理念により、前出の無限サンプリングとは若干動作方法が異なります。
プリトリガ動作のトリガ前サンプリングでは、チャンネルあたり４μＳという最高速度で無限にサンプリングを行なう事ができます。その一方で、Ｆｃａｄ４１６ユニット内部のリングバッファはＦＩＦＯメモリとは別のプリトリガバッファメモリとなっているため、そのバッファからの読み出しはトリガ後サンプリングデータをＦＩＦＯメモリから読み出し終わるまで行なう事ができません。更にこのバッファはリングバッファ形式であるためバッファサイズを上回るデータを保持する事はできず、常に最新のトリガ前サンプリングデータを保持しているという特性が与えられています。プリトリガバッファ内部のデータ数は、Fcad416_Get_Status関数が返すpre_sampled変数によって与えられますが、実際問題としてはＦＩＦＯメモリと同様、ハンドラーＤｌｌ内部にサンプリング開始時に専用バッファがユニット毎に確保され、ハンドラーＤｌｌがＦｃａｄ４１６ユニットと適宜交信を行なう事によって自動的に取り込んでいます。また、プリトリガバッファが一巡（或いはそれ以上の繰り返し）を行なった場合も、ハンドラーＤｌｌ内部のバッファには、連続したデータとして取り込まれ、アプリケーション側からは、そのまま参照する事ができます。この部分は3-12節で詳細に説明を行なっています。

■ 最高サンプリング速度

１回サンプリングスキャン実行時間の逆数が本ADボード自体の最高サンプリング周波数(サンプリングクロック)となります。

■　トリガ機能

◇ソフトトリガはプログラム上・任意のプロセスで実行できる即トリガ機能。
◇外部トリガは外部ＴＴＬ入力信号の指定エッジで機能します。
◇内部（アナログ）トリガは指定条件とチャンネル０入力をボード上で比較して機能します。
（アナログトリガはチャンネル０固定です。但しチャンネル０に実際のアナログ入力ｎを設定する事ができるため、実際は任意のチャンネル入力をトリガ入力として使用できます。）

■ トリガ動作遅れ

本ボード上で自動的に行われるトリガ検出・動作の遅れ時間はトリガの種類によって少しだけ異なります。

◇内部（アナログ）トリガ：　約４μｓ（ポストトリガ）
◇内部（アナログ）トリガ：　最短でサンプリング周期と同じ時間
　　最長でサンプリング周期の２倍の時間（プリトリガ）
◇内部ソフトトリガ　　：　　５０ｎｓ
◇外デジタル入力トリガ：　２００ｎｓ以下（内部クロック動作時）
　　印加クロック周期＋２００ｎｓ以下（外部クロック動作時）

尚、この時間は本ユニット内部で消費される時間を示しており、ＵＳＢバスの遅延時間は含まれていません。

■ 内部クロックと外部クロックの違いについて

本ユニットでは、サンプリングの基準クロックとして、内部クロックと、外部クロックの２種類を選択して使用することができます。ここでは、この機能について厳密に説明を行います。

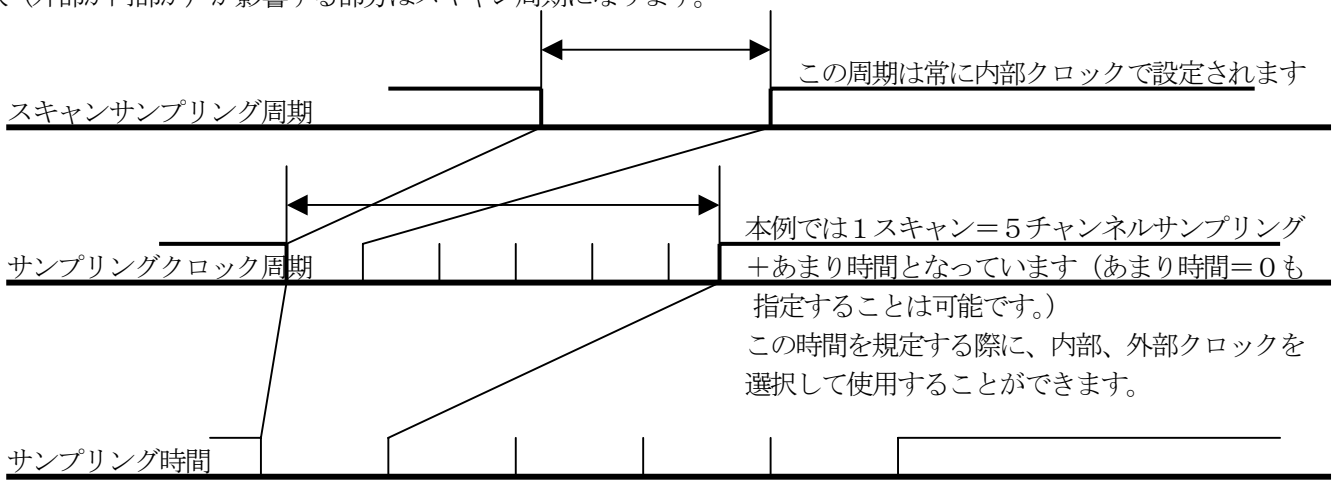
サンプリングのタイミングは、あるチャンネルのサンプリングを行なうための、スキャンサンプリング周期、これをチャンネル数分組み合わせて実現されるサンプリングクロック周期と、更にこのサンプリングクロック周期を必要回数繰り返すことで形成されるサンプリング時間から成り立っています。

これらの関係を式で表すと

サンプリング時間＝サンプリングクロック周期×サンプリング繰り返し数

≧スキャンサンプリング周期×チャンネル数×サンプリング繰り返し数

という関係が必要になります。ここで≧記号が使用される訳を下記の図で説明しますが、本ユニットでクロックの選択（外部か内部か）が影響する部分はスキャン周期になります。



ここでは、繰り返しサンプリング数を５回とした場合を想定してタイミングチャートを作成しています。実際には４－５【９】スキャン速度指定でチャンネルサンプリング周期を４－５【８】クロック源、クロック周期、外部クロック等設定でスキャン周期を設定します。また、外部クロック使用時にあまり時間を０に指定する場合、本ユニット内部のクロックとの周波数誤差を考慮する必要があり、実際の外部クロックとの関係であまり時間がマイナスにならないよう数値を選択する必要があります。

３－２．アナログ入力モードについて

本ユニットでは、アナログ入力モードとして、夫々の入力ラインを単独で処理するシングルエンド入力モードと隣接した二つの入力ラインを組み合わせて使用する差動入力モードをサポートしています。また変換したアナログデータの表現方法としてバイナリモードと、２の補数モードをサポートしています。ここでは、これらのモードについて簡単な説明を行います。

シングルエンド入力モードとは、グランドを基準とするアナログ信号をそのまま測定する方式です。その為、所謂コモンモードノイズに対し弱いのが弱点になります。

一方、差動入力モードでは、（隣接する）二つの入力ラインを組み合わせ、被測定信号の差分を測定します。例えば、電流検出用抵抗の両端電圧を測定し間接的に電流値を求めるなどの応用があります。但し、本ユニットの入力回路は非絶縁入力なので、夫々の入力電圧の絶対値が±１０Ｖ以内[注1]でなければならないため、この点に注意が必要です。また、差動入力モードでは、シングルエンド入力モードの２チャンネル分の信号ラインを使用して、１チャンネルの信号を測定するため、測定できるチャンネル数が１／２になってしまうというデメリットもあります。

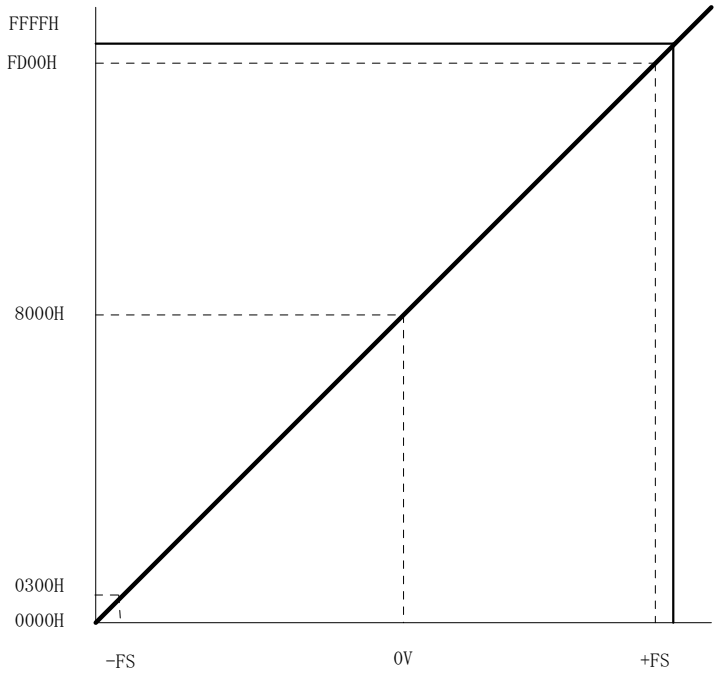
変換されたアナログデータの表現方法については、バイナリフォーマットと２の補数フォーマットの二種類をサポートしています。バイナリフォーマットは、－側最大入力電圧を０００００Ｈ、＋側最大入力電圧をＦＦＦＦＨとする表現方法で、本仕様書の中で変換値の表現として使用しているフォーマットです。また２の補数フォーマットは、最上位ビットを符号ビットとする表現方法で－側最大入力電圧を８０００Ｈ、＋側最大入力電圧を７ＦＦＦＨと表現する方法です。

この場合、０Ｖは００００Ｈと表現され、他方バイナリフォーマットでは０Ｖは８０００Ｈと表現されます。（次頁を参照して下さい。）

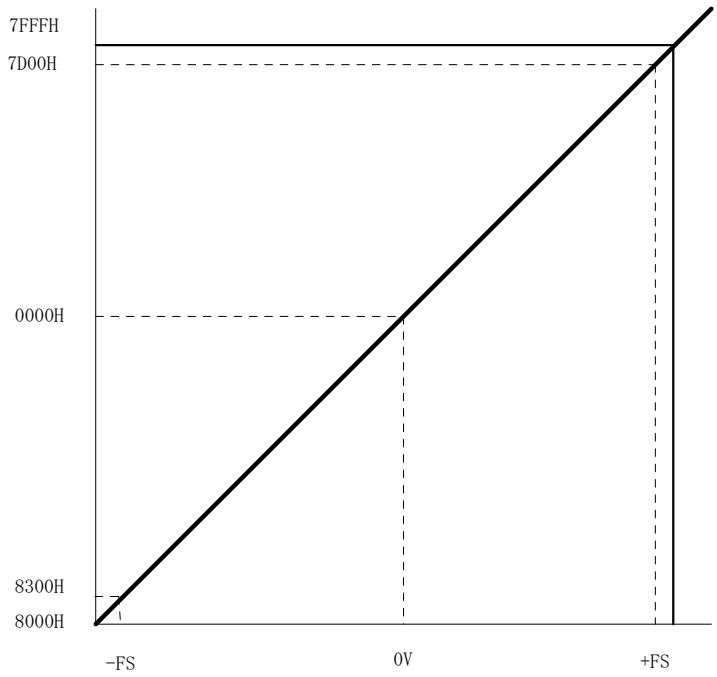
注１）　本ユニットの入力レンジは最大で±１０．２４Ｖとなっていますが、これは±１０Ｖの電圧に対して、その変換値が００００Ｈより大きく（－１０Ｖの場合）またはＦＦＦＨより小さく（＋１０Ｖの場合）なるように、回路全体のゲインを設定する事により±１０Ｖを確実に判断するための方便で、使用されているアナログ回路素子の特性はあくまで±１０Ｖ以下の入力電圧に対してのみ保証されているためです。この関係は他の電圧レンジについても同様です。（±５．１２Ｖでは±５Ｖ、±２．５６Ｖでは±２．５Ｖ、±１．２８Ｖでは±１．２５Ｖとなります。）

３－２－１　フォーマットの違いによるアナログ変換値の違い

伝達関数（バイナリフォーマット）



伝達関数（２の補数フォーマット）



３－３．入力チャンネルの可変割り当てについて

本ユニットでは、マルチプレクサを使用した逐次変換方式によるアナログデータのサンプリングを行いますが、その際あらかじめ定義されたシーケンスではなく、ユーザーが定義したシーケンスに従って入力チャンネルを切り替える機能を実現しています。この事により

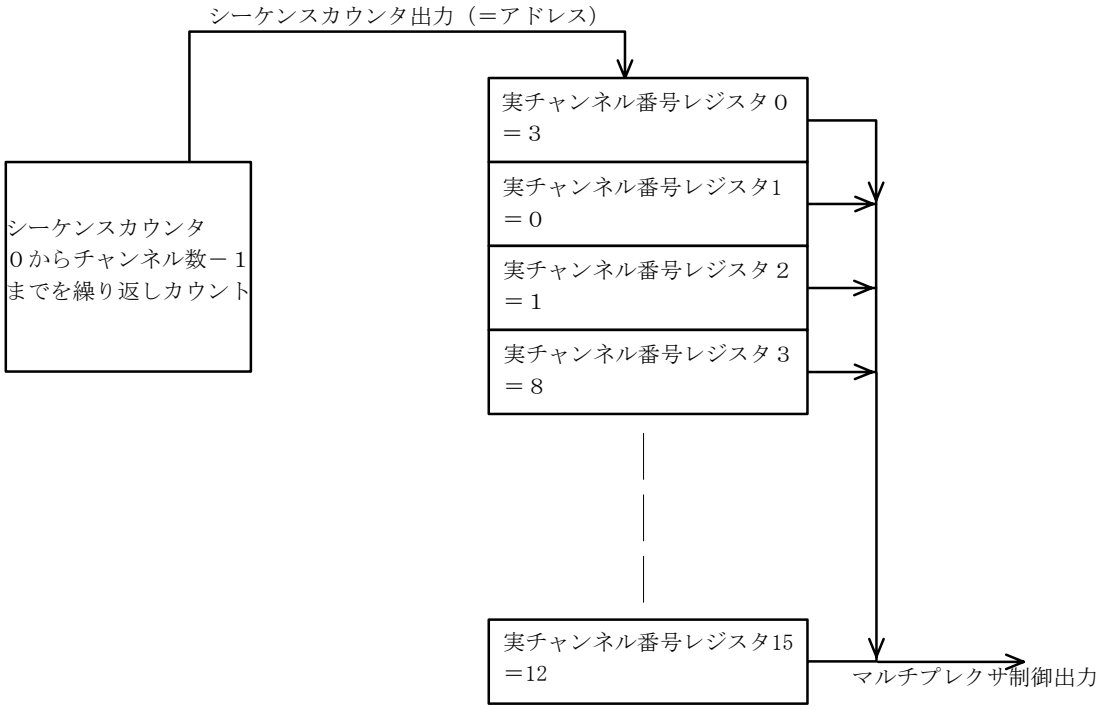
a　アナログトリガに使用する観測チャンネルを入力結線には手をつけずに切り替える事ができる。

b　入力結線を変更することなく、測定チャンネルの組み換えが可能となる。

等のメリットが生じます。

３－４．アナログ入力切り替えレジスタ

実際にアナログ入力シーケンスを変更するためには、論理チャンネル番号０から１５に対応して用意されている実チャンネル番号レジスタに、実際に測定を行うチャンネル番号をあらかじめ設定しておく必要があります。この為に用意されている関数がFcad416_Set_Sampch関数です。尚、電源投入時には、このレジスタには０から順番に１５までの数値が設定されています。



この例では、先頭チャンネルは物理チャンネル３、続いて物理チャンネル０、物理チャンネル１と続き、最後には物理チャンネル１２が接続されて一回のスキャンが終了するように設定されています。

アプリケーションレベルでは、次の関数Ｄ１１を使用することで対応します。

Fcad416_Set_SampCh(WORD board_no , int scan_ch, int scan_order[], int range);

ここで、board_no は設定を実行しようとしているユニットの ID、

scan_ch はスキャンサイクルを構成するチャンネル数、

scan_order[]は実チャンネル番号レジスタ０から（最大）１５までのレジスタにセットすべきサンプリングチャンネル番号配列（上の例では、３，０，１，８，・・・・１２という配列になります。）

また、この場合アナログトリガの検出はチャンネル０ではなくチャンネル３に対して行われます。

range は、使用する入力レンジを示す数値（次項を参照して下さい。）

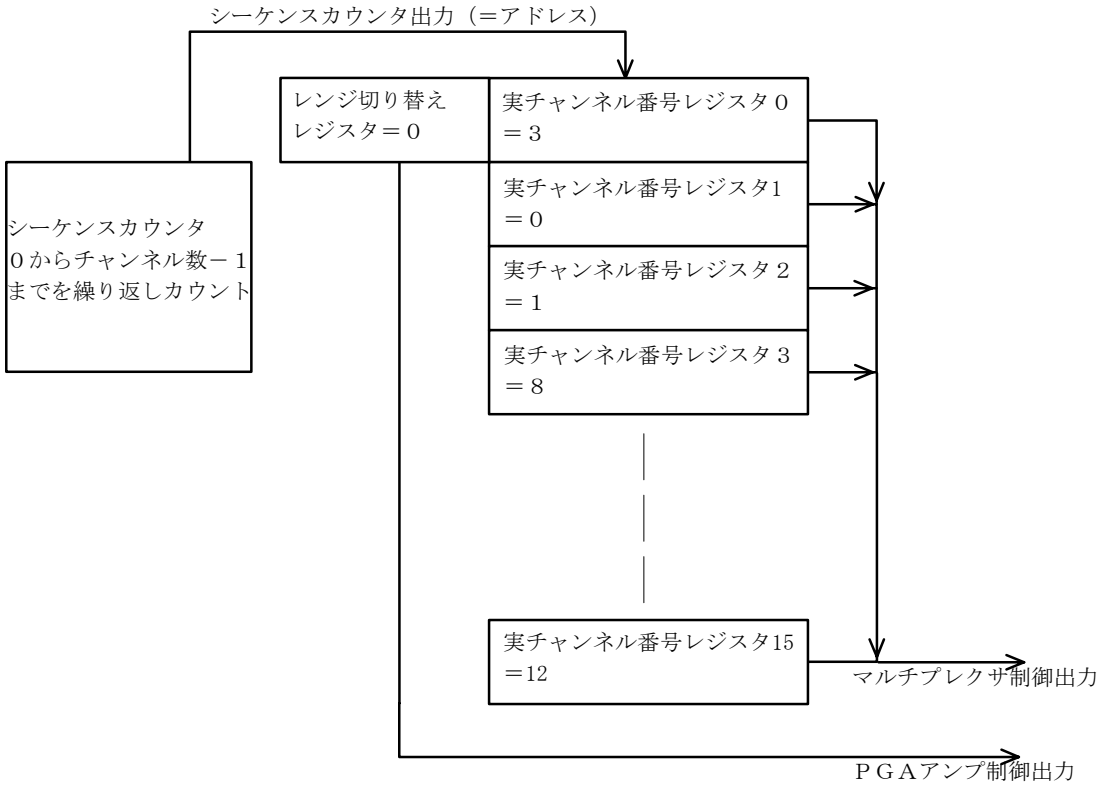
となります。

また、マスタースレーブ接続の場合、サンプリング順序は任意ですが、サンプリングチャンネル数は全てのユニットで同じ値になっていなければなりません。

他方、差動入力形式の場合は若干様子が異なります。この場合は、項１－３で示したように、シングル入力モードで１６チャンネルあったアナログ入力チャンネルが８チャンネルに減少します。また、この際、差動入力のプラス側とマイナス側とは、ある規則性に従った形で組み合わされています。例えば差動入力チャンネル０は、シングル入力チャンネル０とチャンネル１との組み合わせで構成されており、この組み合わせを組み替えることはできません。
そのため、差動入力形式の場合には前項で述べた実チャンネル番号レジスタは０番から７番までが使用可能で８番以降のレジスタについては使用出来ません。またレジスタに設定できる数値も０から７までで（０がＣＨ０に、１がＣＨ１に、７がＣＨ７に対応します）仮に８以上の数値を設定しても８で割った余りが実際のレジスタ設定に使用されるため注意が必要です。

３－５．アナログ入力レンジ切り替えレジスタ

３－４で説明した実チャンネル番号レジスタには、一括して設定できる入力レンジ切り替えレジスタが付属しています。このレジスタに必要な設定を行っておく事により、任意の電圧レンジによる測定を行う事が可能です。また下記のブロック図からも想像されるとおり、このレンジ切り替え機能は一連のサンプリング動作中には切り替える事はできません。



ここでは、入力レンジは±１０．２４Ｖに設定されています。
また設定値と実際の入力レンジとの関係は次のようになっています。

設定値	入力レンジ
０	±１０．２４Ｖ
１	±５．１２Ｖ
２	±２．５６Ｖ
３	±１．２８Ｖ

実際の設定方法については前項（Range変数）を参照して下さい。

３－６．トリガモードについて

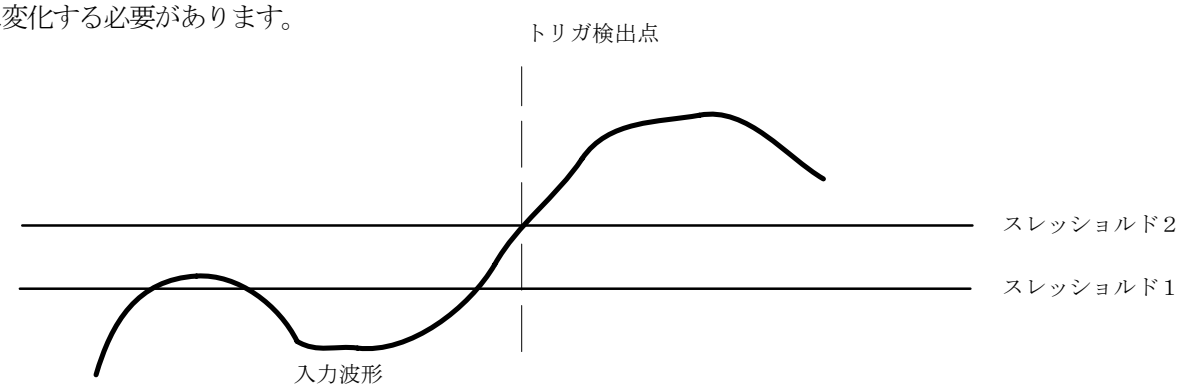
本ユニットでは、トリガの種類として、アナログトリガ、デジタルトリガ、及びソフトトリガ（即トリガとも呼称）の３種類を実現しています。また、これらの内アナログトリガについては、エッジトリガ（±）、レベルトリガ（±）、アウトレンジ、インレンジ、デュアルスロープ（±）を、デジタルトリガについてはエッジトリガ（±）を、夫々サポートしています。また、アナログトリガには入力に重畳するノイズによる誤動作を防ぐため、コンパレータ、及びトリガレベルレジスタが二組準備されています。

ａ．　エッジトリガ（±、アナログ）
ｂ．　レベルトリガ（±）
ｃ．　レンジトリガ（イン、アウト）
ｄ．　デュアルスロープトリガ（±）
ｅ．　エッジトリガ（±、デジタル）

以下に、これらのトリガモードの詳細を示します。

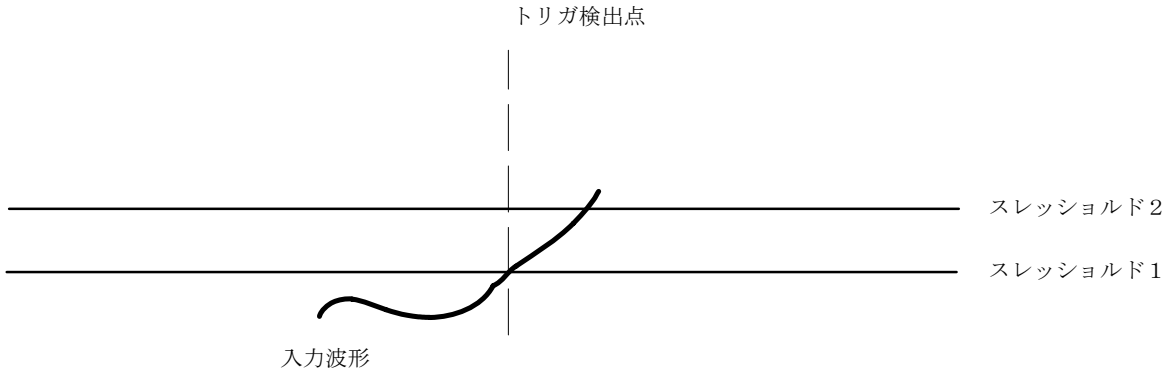
エッジトリガ（±、アナログ）

チャンネル０入力があらかじめ設定されたスレッショルドを立ち上がり、或いは立下りでよぎった事を検出してトリガとするものです。具体例を以下に示します。（例は立ち上がりエッジの例です）この条件でトリガが成立するためには、下の例に見るように一旦スレッショルド１よりも低い値に入力が低下した後に、スレッショルド２よりも大きな値に変化する必要があります。



レベルトリガ（±）

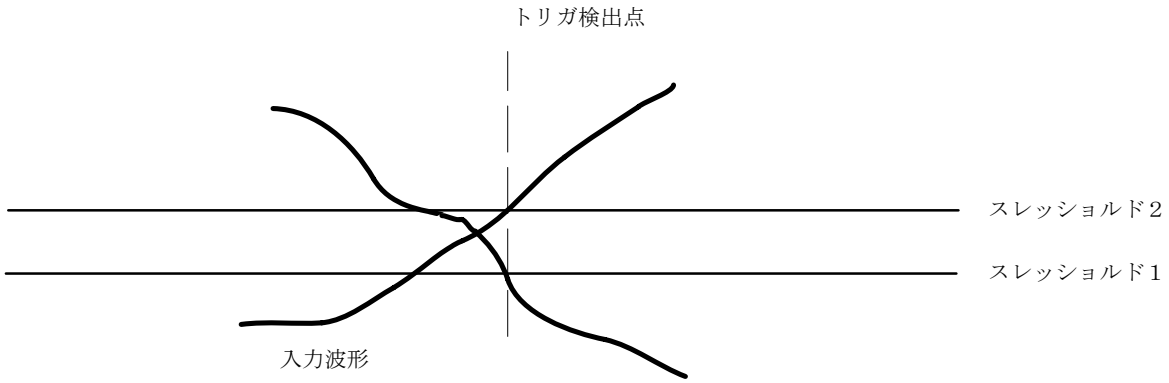
チャンネル０入力があらかじめ設定されたスレッショルド１以上（＋の場合）或いは以下（－の場合）であれば、トリガ成立とするものです。その為、サンプリング開始と同時にトリガ成立となる場合もあります。具体例を以下に示します。（例はレベル＋の場合です）



この例で分かるとおり、レベルトリガでは、信号の履歴は関係なく、単にスレッショルド１との比較によりトリガ条件が決定されます。そのため、サンプリング開始と同時にトリガ成立という事も起こりえます。

レンジトリガ

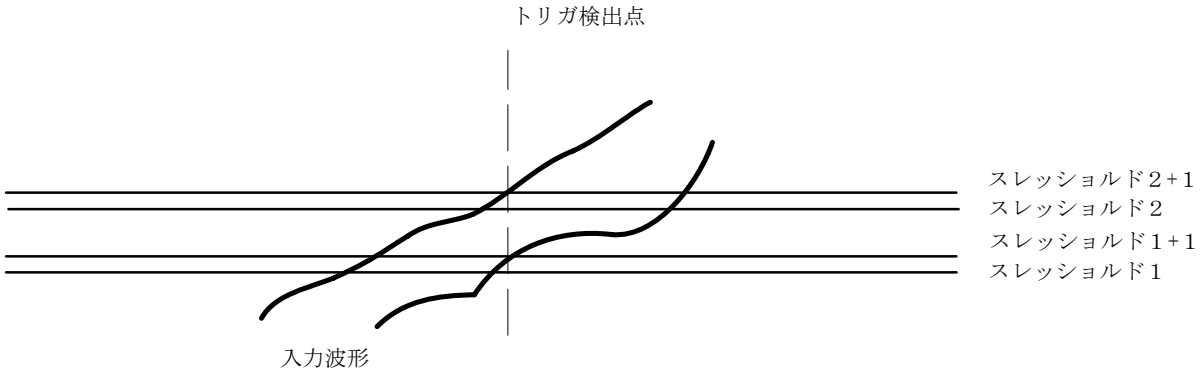
二つのスレッショルドによるウィンドーをレンジとみなし、そのレンジに入ってくるか(インレンジ)そのレンジから出てゆくか（アウトレンジ）という判断基準でトリガ成立とする手法です。
具体例を次に示します。



この場合は、アウトレンジトリガの例になっています。実際の入力レベルがスレッショルド２以上から低下してスレッショルド１を下回るか、またはスレッショルド１を下回っている入力が上昇してスレッショルド２を上回った時点でトリガ成立となるものです。
注）このトリガ方式のみ、スレッショルド２≧スレッショルド１＋２という関係がトリガ成立判断のために必要です。

デュアルスロープトリガ

アナログトリガの最後は、デュアルスロープトリガです。これまで説明してきたトリガ方式は、二つのスレッショルドによるウィンドーを使用していましたが、デュアルスロープトリガでは、二つのスレッショルドに夫々+１のスレッショルドを追加し、これら二組のスレッショルドウィンドーでトリガ条件を判断します。具体例を次に示します。



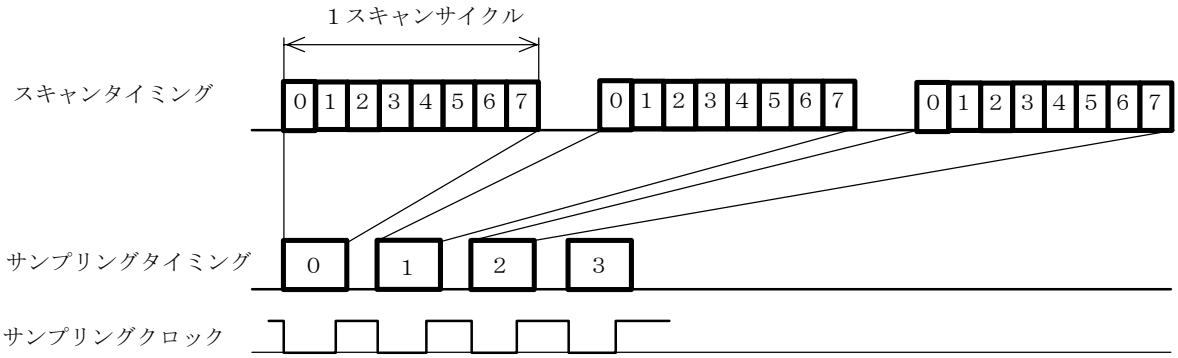
この例は、デュアルスロープトリガ (+) の例です。どちらの入力波形も図中のトリガ検出点でトリガ発生となります。

デジタルトリガ

このトリガについては、文字通りデジタルトリガ入力の指定エッジでトリガ成立となる動作です。厳密には、本ユニット内のＦＰＧＡ内部で、ユニット内部のクロックと同期をとる必要上、若干のディレイが存在します。（２００ｎＳ以下）

３－７．サンプリングクロックとは

本ユニットでは、サンプリングクロック源として、２０MHｚ（内部クロック）、１６．３８４MHｚ（内部クロック）及び外部クロック（想定最高応答周波数１MHｚ）という３種類のクロックを使用することが可能です。
アプリケーションからは、Fcad416_Set_Clock関数によって、サンプリング周期等を指定します。
サンプリングクロックとは、上記３種類のクロック源を使用して自動サンプリングの繰返し周期を決定するタイミング信号です。その最小値はスキャン周期×スキャンチャンネル数で最大値については特に規定はありません。（現実的にはクロック生成用分周カウンターに設定できる最大値がその上限（≒２１４Ｓｅｃ）を決定する事になります。）
以下に、サンプリングクロックとスキャンタイミングとの関係を示します。



３－８．スキャンタイミングとは

自動サンプリングにおいて、逐次変換方式ではＡＤ変換チップの仕様に基づくサンプリング周期で入力を順次切り替えサンプリングを行います。この際の切り替えタイミングのスピードをスキャンタイミングと呼称しています。
本ユニットでは最短で４μＳ／チャンネル、最長で３．２７６ｍＳ／チャンネル（内部クロック２０MHｚの時）或いは３．９９９９ｍＳ／チャンネル（内部クロック１６．３８４MHｚの時）の間の値を任意に設定して使用することができます。設定単位としては、周期或いは分周比が使用可能です。また内部クロックとして１６．３８４MHｚクロックを使用した際には、基本クロックとしてこのクロックがスキャンタイミングの設定にも使用されます。一方外部クロックを選択した際にも、スキャンタイミングの基本クロックは内部クロックになりますが、本ユニットではこの場合であっても内部２０MHｚクロック或いは内部１６．３８４MHｚクロックを選択使用できる設計になっています。（但し、一回のサンプリングサイクルの中で複数のタイミングを切り換えて使用することはできません）

３－９．オフセット調整レジスタ

本ユニットでは、オフセットを調整するために専用のＤＡＣが使用されており、入力チャンネル・入力レンジ・入力モードにより夫々別個のレジスタを使用して、サンプリング実行時に適切なレジスタの値をＤＡＣに設定する事でオフセット・ゲインを調整しています。オフセット調整レジスタは、入力１６チャンネル夫々に入力レンジ４パターン・入力モード２パターン（計１２８パターン）に対応するレジスタが準備されており、このレジスタ群を特定するために７ビットのアドレスが使用されています。

３－10. ゲイン調整レジスタ

本ユニットでは、ゲイン調整も専用のＤＡＣを使用しています。オフセット調整レジスタとは異なり、入力チャンネルには無関係に、入力レンジ・入力モードの組み合わせによって３ビット（８パターン）のアドレスにより選択されたレジスタが実際の調整に使用されます。またこれらのレジスタの内容は電源投入時、専用のＥＥＰＲＯＭから読み出され、ＦＰＧＡに設定されます。実際のアドレスに対するビット割付を下表に示します。ここで８ビット目（最上位ビット）は、通常モードと調整モードを切り替えるビットで、オフセット及びゲインの調整を行う際にはセットしておく必要があります。

アドレス割付	ビット割付	ビット番号
調整モード／通常モード	１＝調整／０＝通常	Ｂ７
差動／シングルエンド	１＝差動／０＝シングルエンド	Ｂ６
入力レンジ	００＝±１０．２４Ｖ ０１＝±５．１２Ｖ １０＝±２．５６Ｖ １１＝±１．２８Ｖ	Ｂ５
		Ｂ４
対象チャンネル番号	０ＨからＦＨ （ゲイン調整レジスタでは 無意味）	Ｂ３
		Ｂ２
		Ｂ１
		Ｂ０

オフセット・ゲイン調整レジスタのメモリアドレッシング

３－11.　ＦＩＦＯバッファメモリの構造・動作

指定されたトリガ条件が成立した後、ＡＤ変換されたデータは本ユニット内のＦＩＦＯメモリに順番に書き込まれ、一方パソコン側からハンドラーＤ１１を経由して適宜読み出されていきます。（ファーストインファーストアウト）ＵＳＢバスには、バスタイムアウトという制限事項も存在するため、実際には、サンプリング速度や、スキャン速度、本ユニット内でのデータ蓄積度等を考慮し、適切なブロックサイズによるバルク転送を行っています。このときのデータ転送速度は最大でほぼ５００Ｋバイト／秒となっています。（４μＳサンプリング実行時）

３－12. プリトリガバッファメモリの構造・動作

３－12－１　ハードウェアの構造

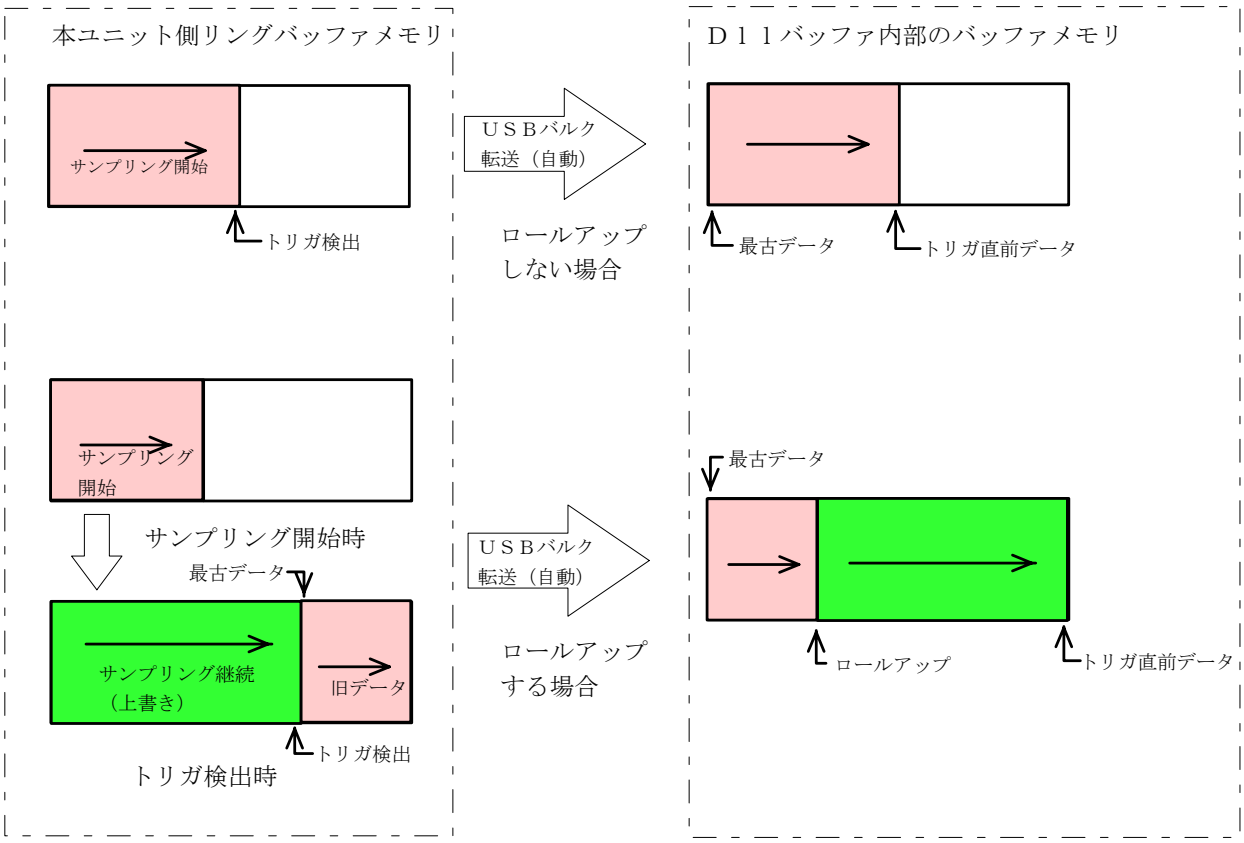
プリトリガバッファＲＡＭはハードウェア上FIFOと同じDRAMの中に構成され、FCAD416内部のソフトウェアからはリング状バッファとして制御が行われ、プリトリガサンプリングのトリガ前データが書き込まれます。サンプリング開始と共にサンプリングチャンネル分の変換データが書き込まれてゆき、バッファの最後まで書き込みが終了すると、書き込みアドレスは先頭に戻り以降上書き状態を繰り返します。そのため、プリトリガサンプリングのトリガ前データは最小１M語から最大で７M語までの間でホストPCからプログラム可能な形で構成されています。但し、総容量８M語のDRAMを分割使用している関係上、FIFOの容量はそれぞれ最大で７M語から最低で１M語に制限されます。

３－12－２　プリトリガデータの読み出し

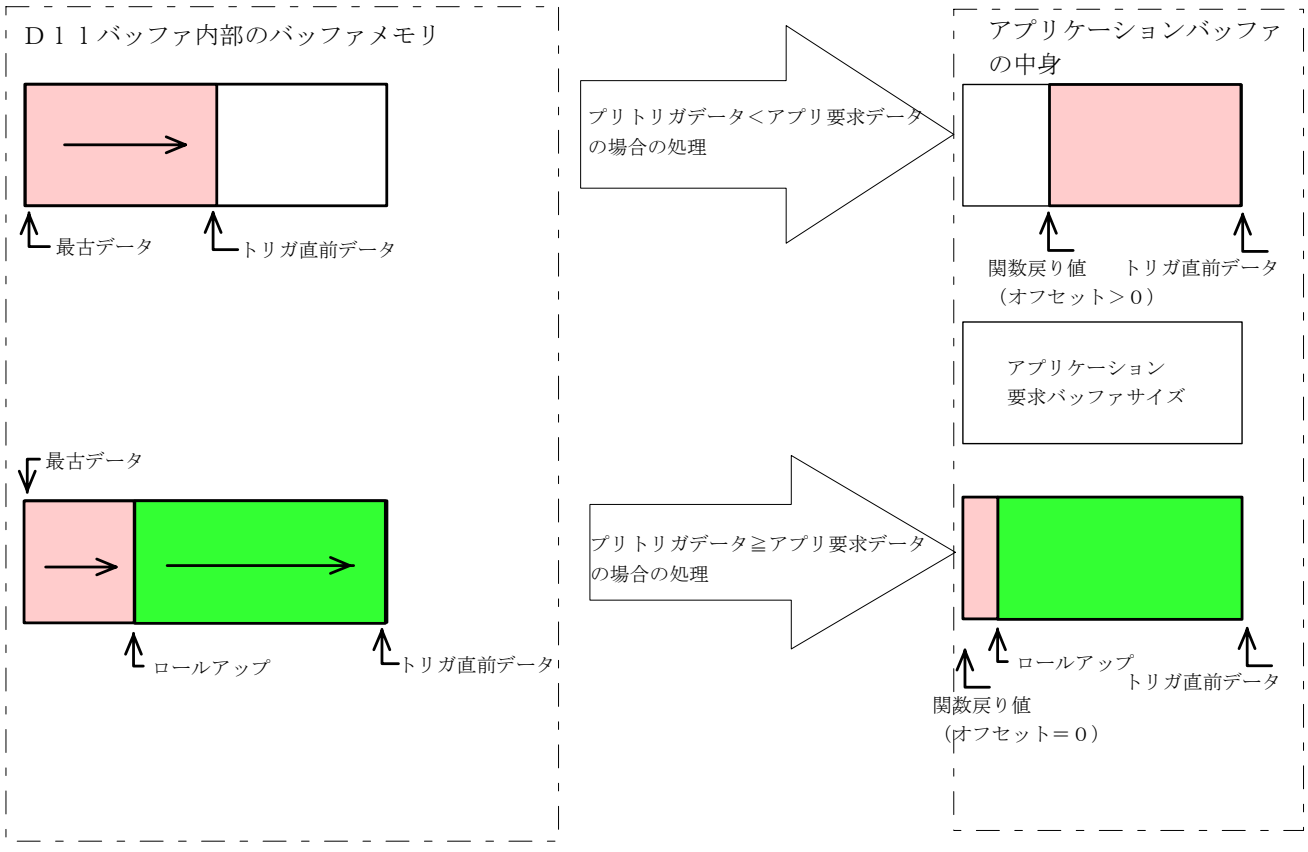
サンプリング中はＦＩＦＯメモリのデータ読み出しが最優先されるため、プリトリガバッファメモリからの読み出しはサンプリング動作終了後に限られます。 １回サンプリング分のＡＤデータはFIFO（前３－11項参照）と同様でチャンネル０～ｎの各ＡＤデータが若いアドレス順に格納されています。ハンドラーＤ１１では、このバッファに対し、先頭アドレス及びデータ数を指定して予め用意していた専用のD11バッファへ全てのデータを読み出しています。データ読み出しタイミングは、FIFOデータ読み込み後で、尚且つサンプリング終了メッセージ発行前のタイミングになります。また、このプリトリガバッファメモリは無限サンプリング動作を行っているため、その動作はロールアップする場合としない場合（即ち、書き込まれたデータがバッファを一巡するかどうか）があります。また、ロールアップが発生したかどうかを記録するロールアップフラグも用意されています。但し、ロールアップフラグには、ロールアップ回数をカウントする機能はありません。そして、それぞれの場合におけるトリガ位置、およびトリガ前サンプリング回数はハンドラーＤ１１内部で処理され、何れの場合であってもユーザーから見ると最古のデータから最新のデータへと連続してアクセスできるように再配置されています。このあたりの状況は次ページの図に表されており、ユーザーアプリケーション（及びハンドラーＤ１１）は、同図の右側の部分に対する制御を行う事でプリトリガバッファメモリを扱う事ができます。

言い換えると、プリトリガバッファメモリは、プリトリガサンプリングモードでは常に全てのデータがハンドラーＤ１１へ読み込まれている為、サンプリング終了後アプリケーション側で必要な部分だけを取り出して使用する事ができます。

実際の応用例として、次のような例題を考えて見ます。
プリトリガバッファメモリとして１M語を指定し、１６チャンネルのプリトリガサンプリングを指定しました。
チャンネル当たりのプリトリガバッファサイズは６４Ｋ語となります。プリトリガサンプリングを開始し、サンプリング終了後、Get_Status関数により、プリトリガデータサイズ１００００語を得ました。アプリケーション側の要請はトリガ前１Ｋ語が必要だったため、所要データ数１０２４語として、Fcad416_Read_DirectRam関数を呼び出し、必要なデータをアプリケーション側に引き渡しました。また、同条件で再度サンプリングを行ったところ、プリトリガデータサイズは２５６語となりましたが、アプリケーション側の要請も同じく１Ｋ語だったため、同じ条件でFcad416_Read_DirectRam関数を呼び出しましたが、戻り値として０ではなく７６８（有効プリトリガバッファ先頭オフセットの値）が返されたため、オフセット７６８からオフセット１０２３までの２５６語をアプリケーション側に引き渡しました。この場合、未使用となるバッファ領域は、ハンドラーＤ１１によって０フィルされています。
ここで、オフセットの値は、他の関数と同様チャンネル当りの数値で表現されているため、実際のバッファ内のオフセットは、ここで得られた値とチャンネル数の積になります。



ＦＣＡＤ４１６内部メモリ　　－＞　　ハンドラーＤ１１内部メモリ



ハンドラーＤ１１内部メモリ　　－＞　　アプリケーションバッファメモリ

３－13. マスタースレーブ動作

複数の本ユニット（最大16台）を使用し、マスターユニットが出力するクロックで同期運転することができます。ここでは、トリガを検出し、同期クロックを出力するユニットをマスター、他のユニットをスレーブと呼称します。スレーブユニットはマスターユニットからのクロック、及び同期トリガ出力を受けて同期運転を行いますが、３－６デジタルトリガの項で説明した同期損失が存在するため、若干のディレイが存在します。（２００ｎｓ以下）

ユニットの設定

ユニット番号設定ＳＷ（ＳＷ１）を、マスターユニットは“０”、スレーブユニットは“０以外”の任意の重ならない値とします。（値の範囲は１から１５までの１５種類が最大となります）

ユニット間の接続

マスターユニットの同期トリガ出力（ＳＹＮＣ－ＴＲＧ）をスレーブユニットのトリガ入力（ＴＲＧ－ＩＮ）へ、同期クロック出力（ＣＬＫ－ＯＵＴ）をスレーブユニットのクロック入力（ＣＬＫ－ＩＮ）へ接続します。
スレーブユニットが複数台存在するときは、全てのスレーブユニットに対して同じ接続を行って下さい。デジタル入出力コネクタ（ＣＮ３）には、この敗戦を実現しやすくするためトリガ入力、クロック入力共に２端子ずつが割り当てられています。（１－３の項を参照して下さい）

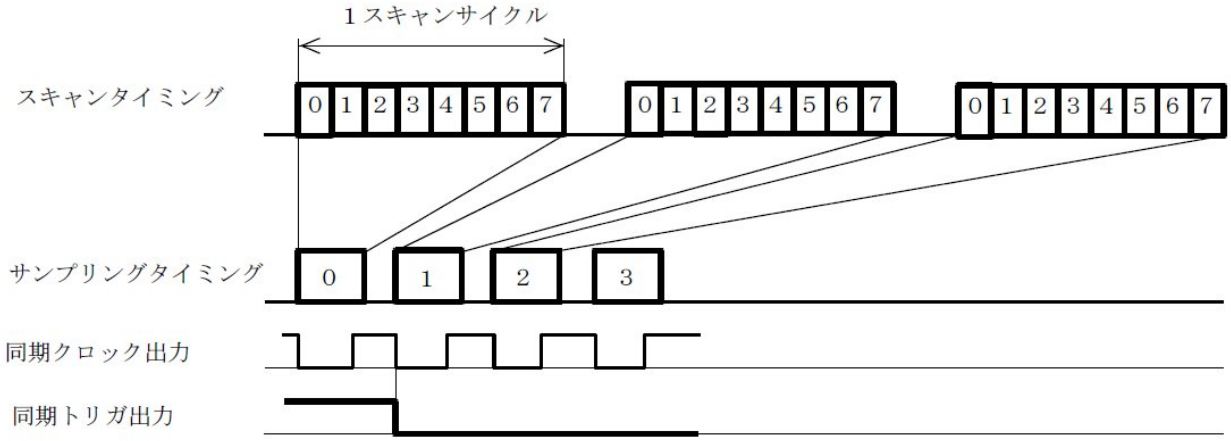
ソフトウェア

関数の引数としてユニット番号が必要とされているものについては、全てのユニットに対して関数を発行して下さい。またユニット番号が不要な形式をとっている関数については、マスターユニット用の関数として必要な処理を定義しておいて下さい。ユニット番号を必要としない関数では、スレーブユニットに対しては、ハンドラーＤ１１の内部で必要な設定を行っています。
（スレーブユニットに対し暗黙的に設定される内容）

- トリガ条件は外部トリガ（立下り：但しソフトトリガの場合のみスレーブユニットもソフトトリガとなります。）
- クロック入力は外部クロック
- クロックの仕様は立ち下がりエッジ動作で1分周クロック
- スキャン速度はマスターユニットと同じ速度

マスタースレーブ方式の運転は、以下に示すようにして行われます。

まず全てのスレーブユニットのサンプリングを開始し、外部トリガ、クロック入力待ち状態にしておきます。この時点で、パラメータが不足しているスレーブユニットが発見されるとエラーを返し、サンプリングは開始されません。全てのスレーブユニットがサンプリングを開始した後、マスターユニットのサンプリングを開始します。



サンプリングを開始したマスターユニットはサンプリングタイミングをＣＮ３の同期クロック出力端子（ＣＬＫ－ＯＵＴ）に同期クロック出力として出力を開始します。スレーブユニットはこのクロックを基本パルスとしてサンプリングを開始します。マスターユニットは、自身のトリガ検出タイミングもＣＮ３の同期トリガ出力端子（ＳＹＮＣ－ＴＲＧ）に同期トリガ出力として出力します。スレーブユニットは、この信号と同期クロック出力及びサンプリン

グモード（ポストトリガ/プリトリガ）とによって、次のように動作を行います。但しソフトトリガの場合はスレーブユニットもソフトトリガの設定とし、マスターユニットからの同期クロック出力のみで連係動作を行ないます。

スレーブユニット動作対応表

同期トリガ出力	トリガ検出前	トリガ検出後
ポストトリガ	サンプリングは行いません	同期クロック出力のタイミングによりサンプリングを行い、結果をＦＩＦＯメモリにストアしてゆきます
プリトリガ	同期クロック出力のタイミングによりサンプリングを行い、結果をプリトリガメモリにストアしてゆきます	同期クロック出力のタイミングによりサンプリングを行い、結果をＦＩＦＯメモリにストアしてゆきます

３－13－１ 本ユニットの性能

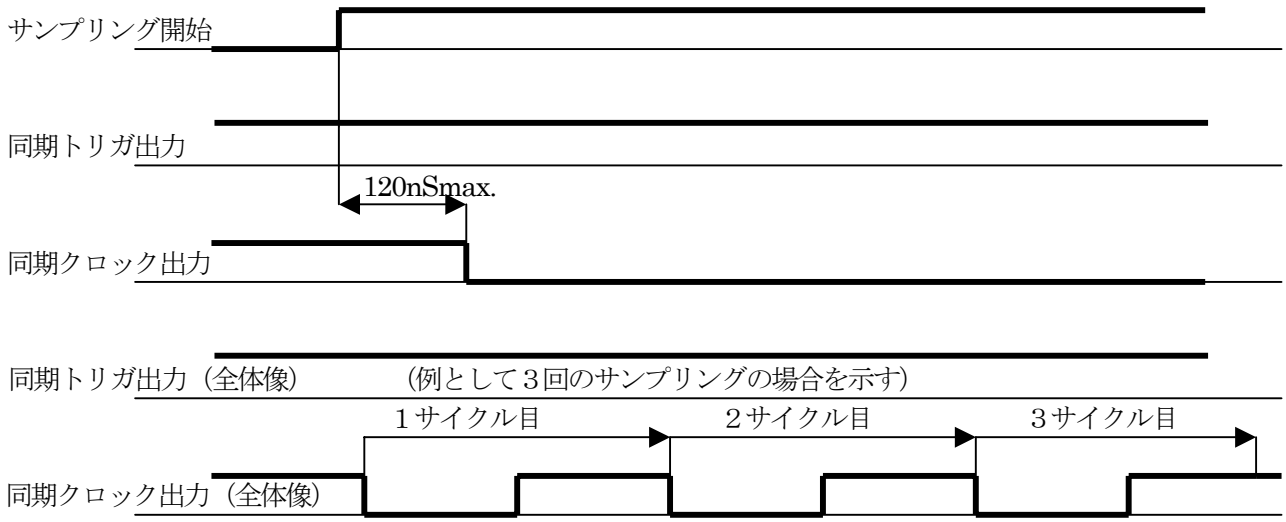
本ユニットをマスター･スレーブ接続で１２ユニット同時運転を行いその挙動を調べたところ、この程度の台数では動作確認プログラムを実行する程度のものであれば、メモリ２５６Mバイトを装備したセレロンプロセッサクラス（動作周波数１．３ＧＨｚ）で充分な性能を発揮できる事が判明しています。（１２ユニット同時にチャンネルあたり６４K語で４μＳサンプリングを行わせ、夫々のユニットで５００Kバイト／秒≒２５０K語／秒程度のデータ転送レートが実現できた。総データ転送量は１２M語≒２４Mバイトとなる。）
より高機能なアプリケーションでは、例えばデュアルコアＣＰＵ等の導入が必要となるケースも考えられますが、本ユニット内部にも最低でも１M語のＦＩＦＯが搭載されている事になるため、全体的な性能はそれ程急激には低下しないのではないかと思われます。

一方、無限サンプリングを行なう場合は、若干状況が変わります。１．３ＧＨｚセレロンクラスのパソコンでは同時動作できる台数こそ概ね１２台程度ですが、サンプリング実行中のＣＰＵ占有率がほぼ５０％程度となりかなり限界に近い値です。（サンプリング速度はチャンネルあたり４μＳ、１６チャンネル／ユニットサンプリング）

同じ構成で、パソコンを３ＧＨｚのペンティアムDとした場合でも、サンプリング実行中のＣＰＵ占有率は４０％程度となるため、このあたりが実用的な上限ではないかと思われます。

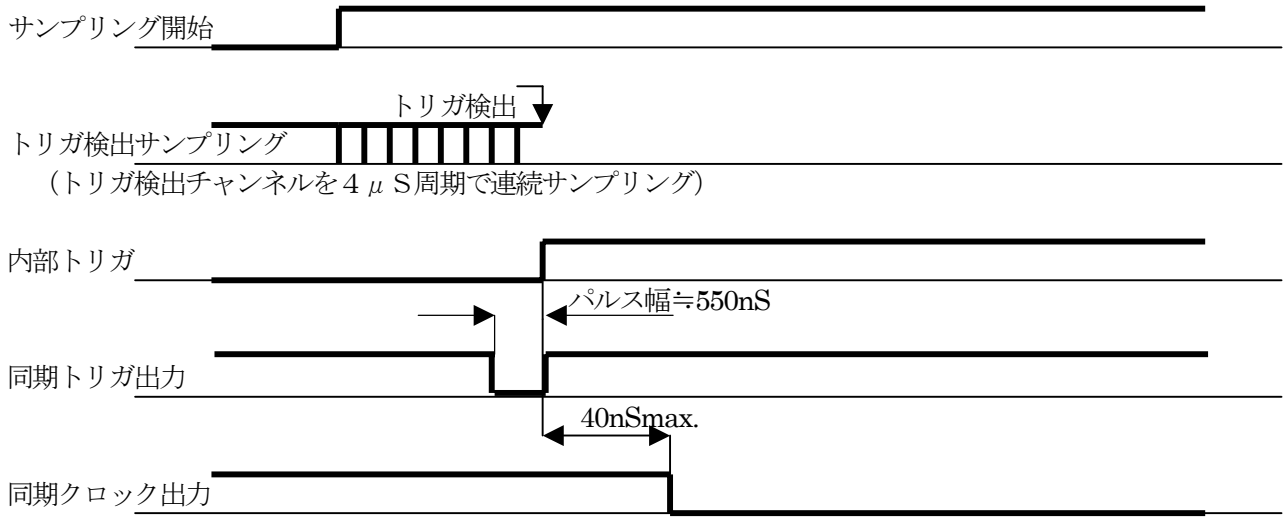
３－14. 各種サンプリングにおけるタイミング説明

３－14－１　ソフトトリガ

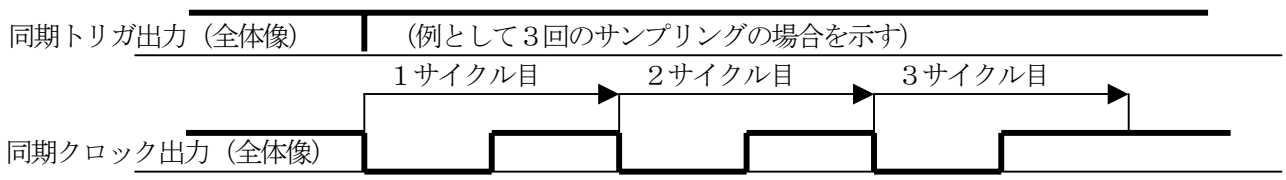


※　ソフトトリガでは同期トリガ出力は出力されません。

３－14－２　アナログトリガ（ポストトリガ）

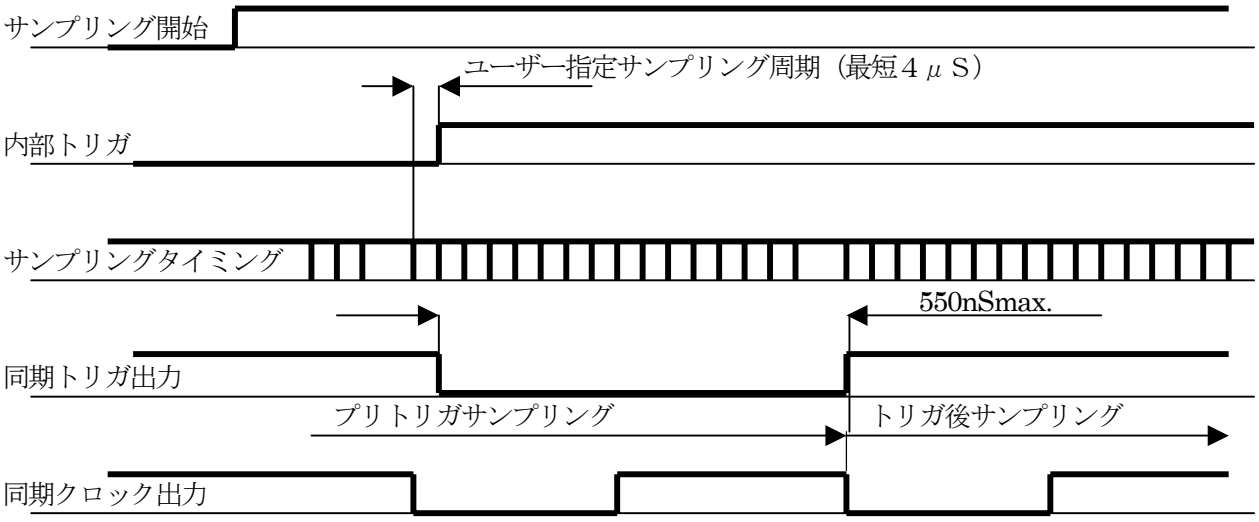


概要タイミング

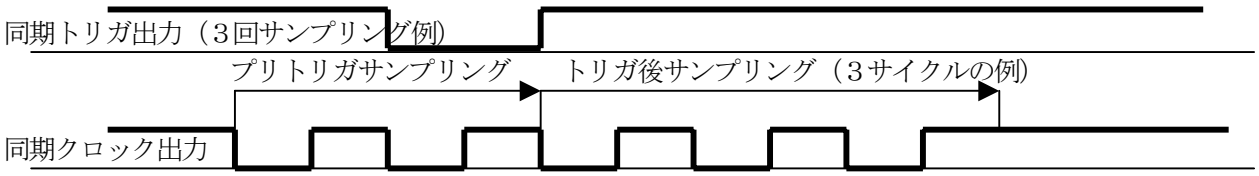


※　同期トリガ出力と同期クロック出力は、マスタースレーブ接続による同期運転時、マスターユニットからタイミング決定用に出力され、スレーブユニットではこれらの信号を元にしてサンプリングが行なわれます。スレーブユニットでは、クロックモードを外部クロック立下りエッジ１分周、トリガモードを外部トリガとしてサンプリング動作を行ないます。

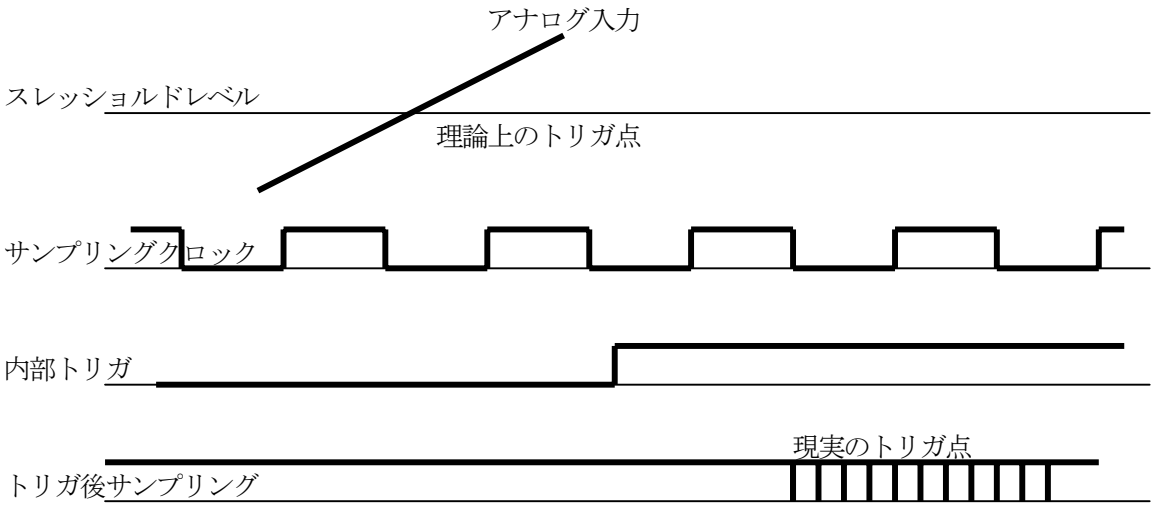
３－14－３　アナログトリガ（プリトリガ）



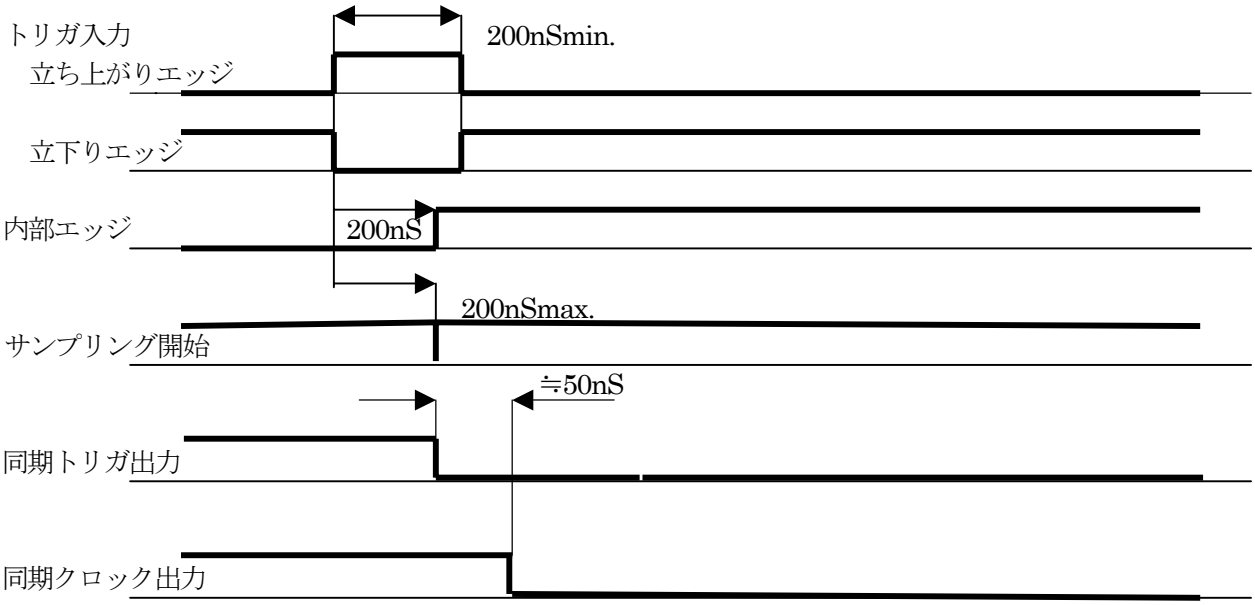
概要タイミング



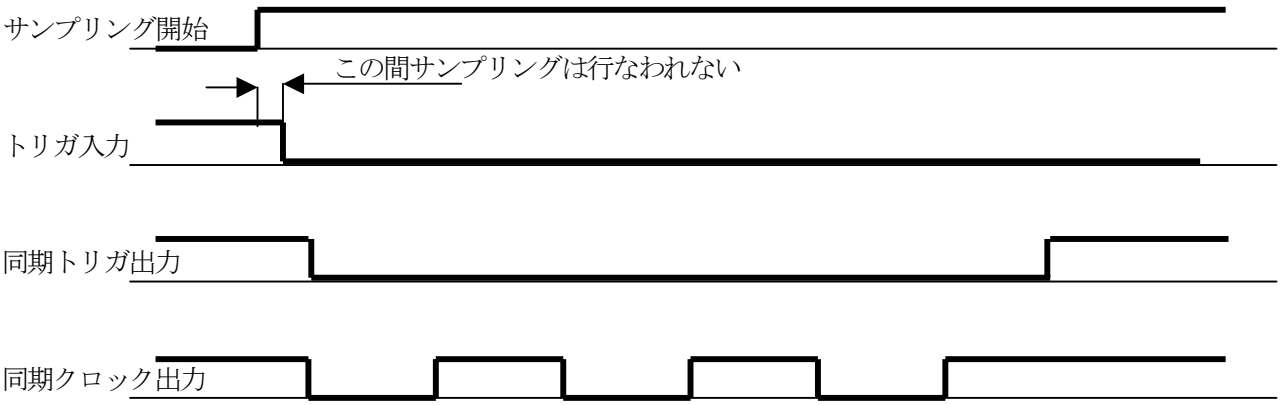
※　プリトリガモードでは、サンプリング開始と同時に指定されたサンプリング条件でプリトリガサンプリングが開始され、サンプリング毎の先頭チャンネルサンプリングについて指定されたアナログレベルとの比較が行なわれます。（トリガ検出）そしてトリガ条件が成立すると、その次のサンプリングサイクルから、トリガ後サンプリングが開始されます。従って最悪条件では、検出しようとするアナログレベルから最大でおよそ２サイクル遅れたところから（先頭チャンネルサンプリング直後に指定レベルに到達した為１サイクル遅れた次のサイクルでレベル検出が行なわれ、更にそのサイクルが最後のプリトリガサンプリングサイクルになるため更に１サイクル遅れる）トリガ後サンプリングが開始する事になります。またスレーブユニットに対する同期トリガ出力は内部トリガ成立時点からトリガ後サンプリング開始までの間出力されます。



３－14－４　デジタルトリガ（ポストトリガ）

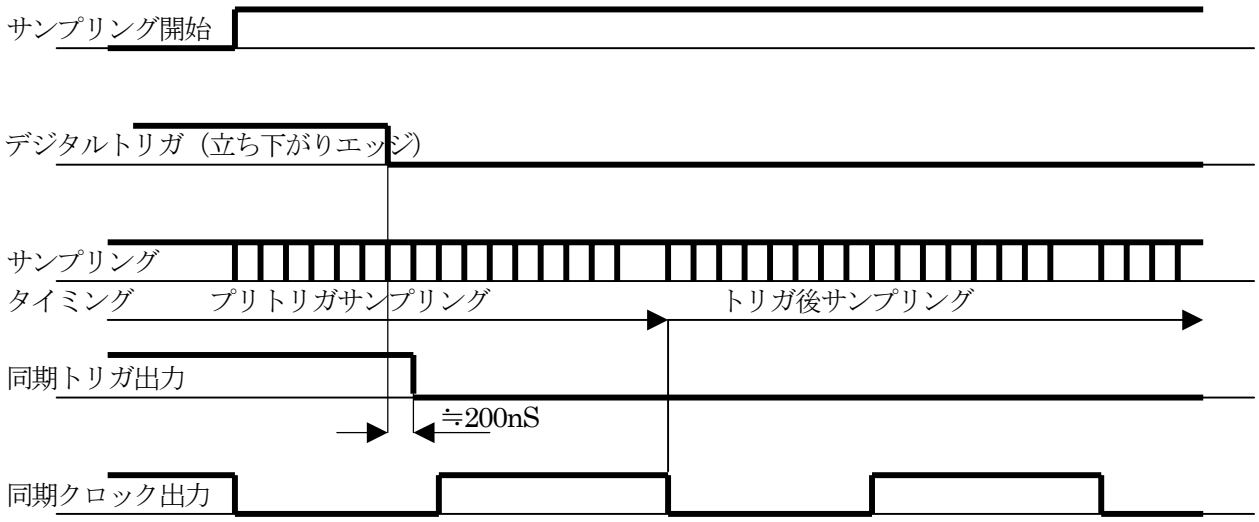


概要タイミング



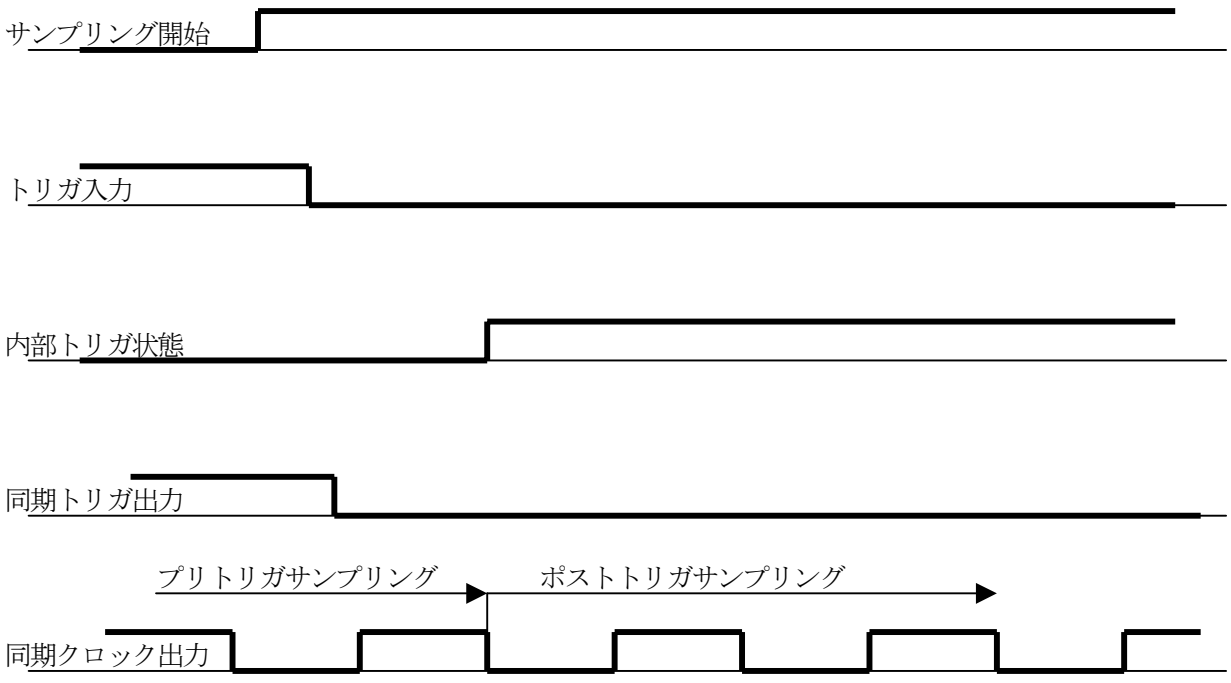
※　デジタルトリガ（ポストトリガ）は３－14－１で説明したソフトトリガ（サンプリング開始と同時にトリガ後サンプリングを開始する）と大変よく似たサンプリングモードです。ソフトトリガサンプリングとの相違点は、前者がサンプリング開始と同時にトリガ後サンプリングを開始するのに対し、サンプリングが開始されてもデジタルトリガ入力が検出されるまではトリガ後サンプリングを行なわないという点にあります。また、デジタルトリガの場合は、同期トリガ出力も同時に出力されます。

３－14－５　デジタルトリガ（プリトリガ）

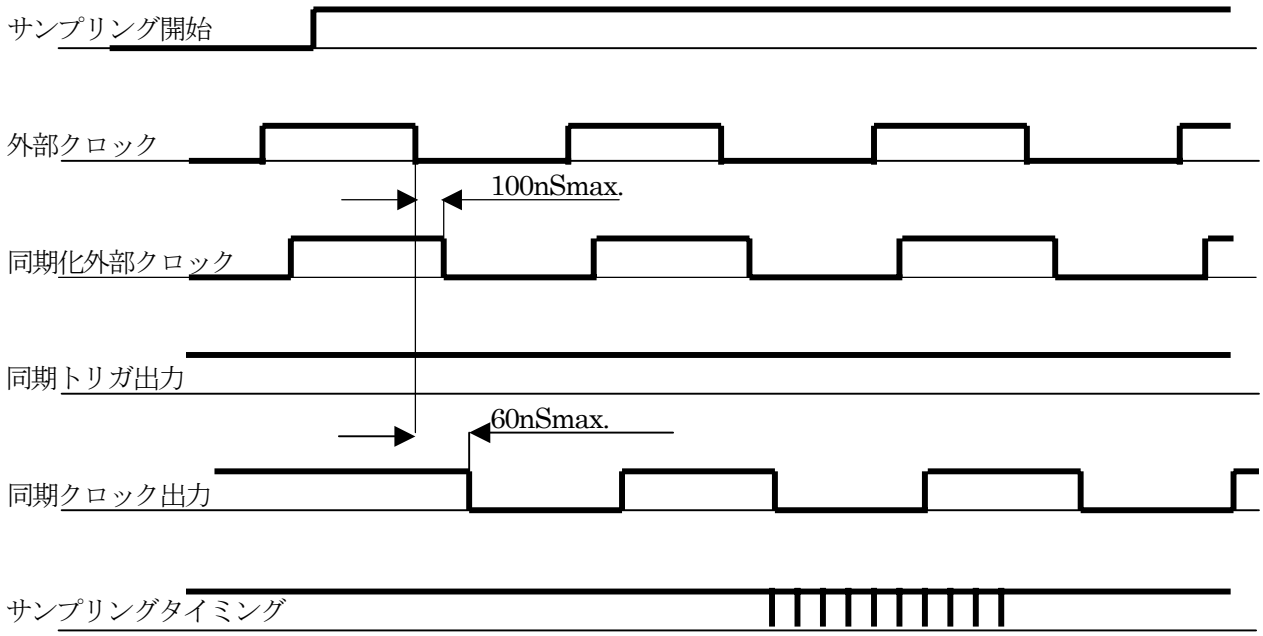


※　デジタルトリガにおけるプリトリガモードでは、サンプリング開始と同時に指定されたチャンネル数でプリトリガサンプリングが開始され、同時にデジタルトリガ入力のモニターが開始されます。そして、デジタル入力の指定エッジ変化が検出されると、次のサンプリングサイクルからトリガ後サンプリングが開始されます。従って、最悪条件ではデジタルトリガ入力から最大でおよそ１サイクル遅れたところからトリガ後サンプリングを開始する事になります。

概要タイミング



３－14－６　外部クロック、ソフトトリガ



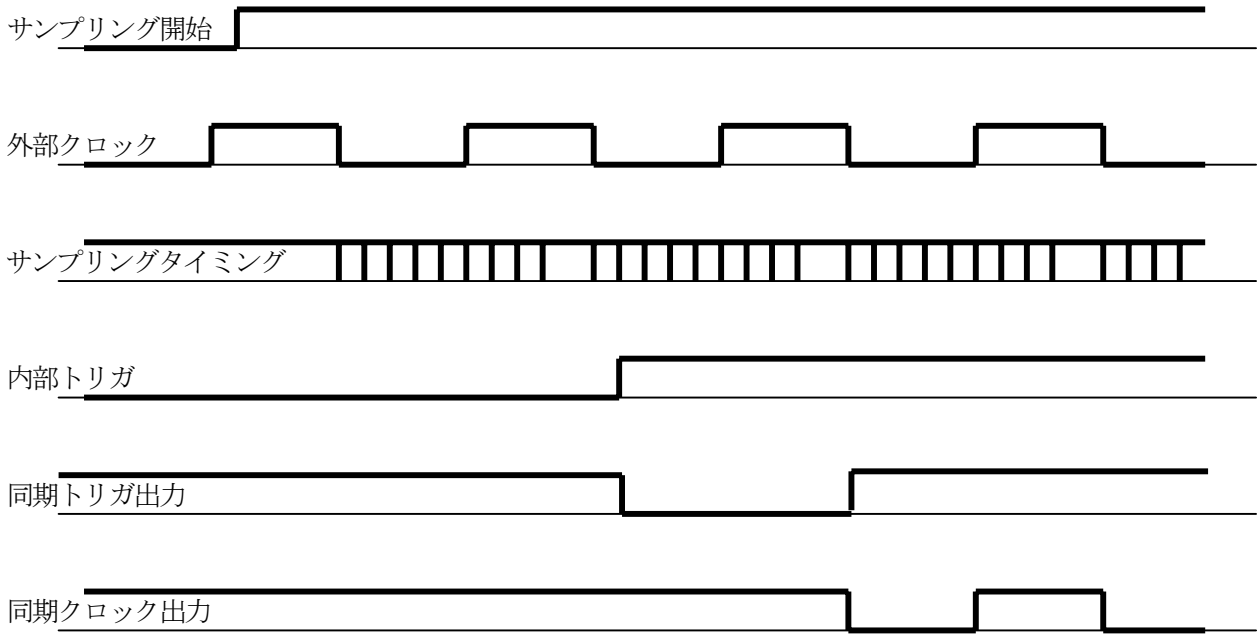
※　外部クロックは、FPGA内部で内部クロックと同期化される関係上、最大で100nSディレイが発生します。サンプリング開始は、この同期化された外部クロックによってスタート信号が検出されるため、最大で外部クロックの2サイクル分遅れることになります。

※　外部クロックモード（ここでは立下りエッジ有効、１分周モードを設定）の場合、ソフトトリガであっても外部クロックの有効エッジ以降に実際のサンプリングが開始されます。また、同期トリガ出力は出力されません。

３－14－７　外部クロック、アナログトリガ（ポストトリガ）

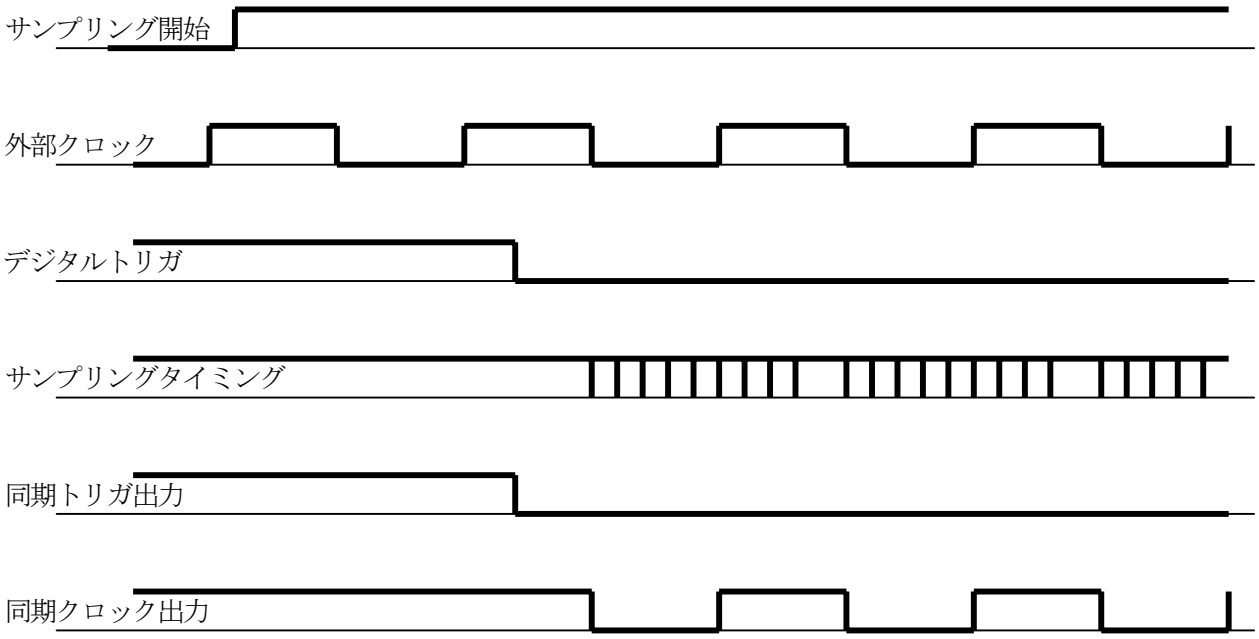
※　外部クロック（ここでは立下りエッジ有効、１分周モードを設定）、アナログトリガ（ポストトリガ）モードでは、サンプリングを開始すると､外部クロックの有効エッジ毎にトリガ検出用アナログ入力のサンプリングを開始します。そして指定トリガ状態を検出すると次の外部クロック有効エッジからトリガ後サンプリングを開始します。内部クロック使用時と異なり、トリガ検出サンプリングは各有効エッジ毎にしか行ないません。これは、外部クロックと本ユニットの最高速サンプリング（４μＳ周期サンプリング）との間に同期関係が成立するとは限らないためです。同期トリガ出力は、これまでと同様、マスターユニット内部のトリガ検出エッジと同期して出力されマスターユニットがポストトリガサンプリングを開始するタイミングで解除されます。

３－14－８　外部クロック、アナログトリガ（プリトリガ）



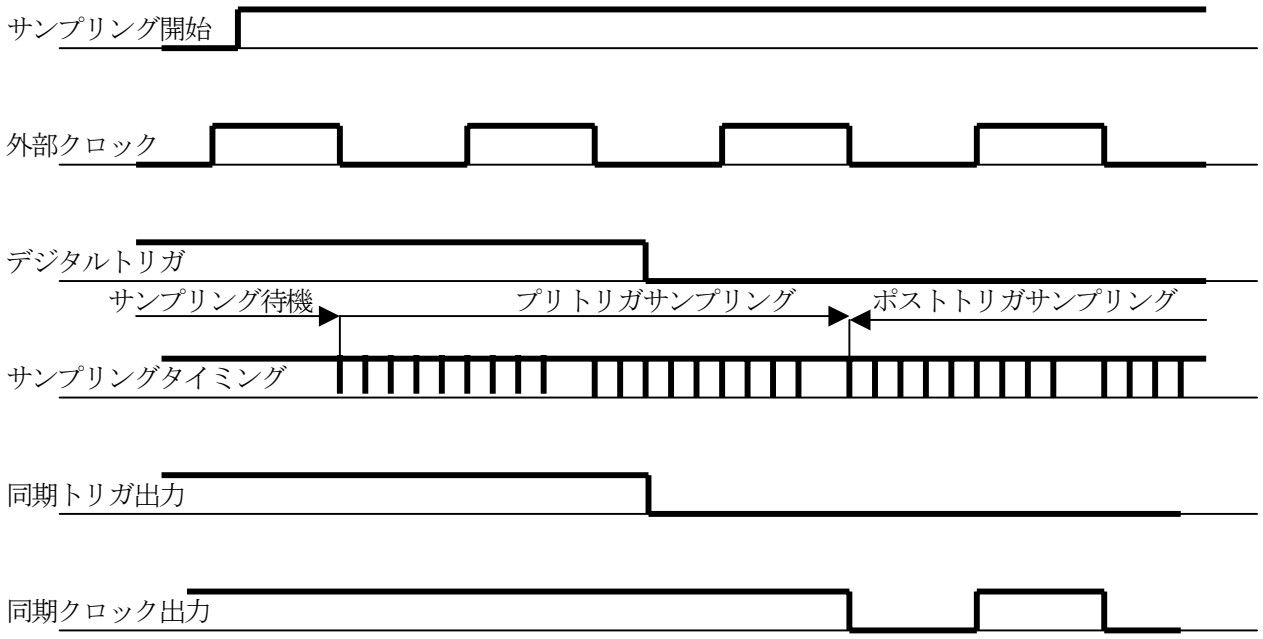
※　外部クロック（ここでは立下りエッジ有効、１分周モードを設定）、アナログトリガ（プリトリガ）モードでは、サンプリングを開始すると、外部クロックの有効エッジ毎に指定されたチャンネル数でプリトリガサンプリングが開始され、各先頭チャンネルのアナログ変換値によってトリガ検出が行なわれます。そしてトリガ条件が成立すると、その次のサンプリングサイクルからトリガ後サンプリングが開始されます。そのため、トリガ成立の次のサイクルからトリガ後サンプリングが開始される事になります。

３－14－９　外部クロック、デジタルトリガ（ポストトリガ）



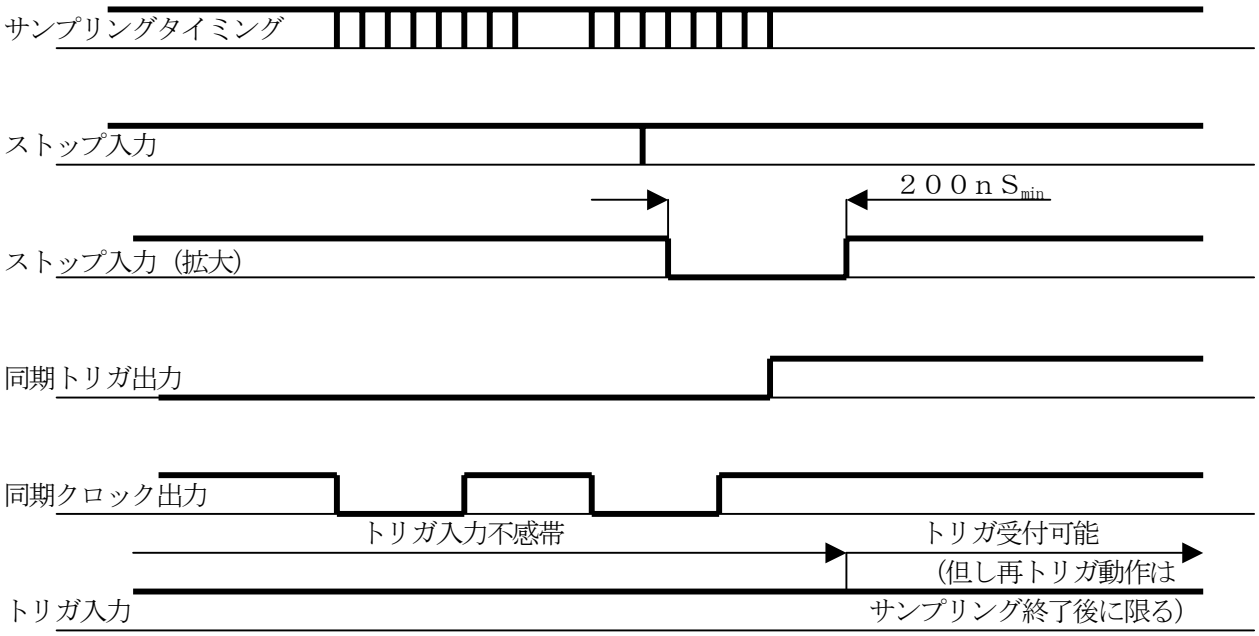
※　外部クロック（ここでは立下りエッジ有効、１分周モードを設定）、デジタルトリガ（ポストトリガ）モードでは、サンプリングを開始しても、デジタルトリガ入力が検出されるまではトリガ後サンプリングを開始しません。実際のサンプリングは、デジタルトリガ検出後の外部クロック有効エッジに同期して行なわれます。

３－14－10　外部クロック、デジタルトリガ（プリトリガ）



※　外部クロック（ここでは立下りエッジ有効、１分周モードを設定）、デジタルトリガ（プリトリガ）モードでは、サンプリングを開始しても、その後の外部クロックの有効エッジからしかサンプリングを開始しません。（プリトリガサンプリング）そして、デジタルトリガが検出されると、その次の外部クロックの有効エッジからポストトリガサンプリングが開始されます。そのため、デジタルトリガ入力から最大で１スキャン分遅れてポストトリガサンプリングが開始される事になります。

３－14－11　ストップ入力



ストップ入力は、任意のトリガと組み合わせてサンプリングを中断するために用意されたトリガ入力です。その使用方法は、アナログ、或いはデジタルトリガと組み合わせて、任意のタイミングで、サンプリングの中断を行なわせるというものです。ストップ入力は任意のタイミングで与える事が可能ですが、本ユニット内部でサンプリングスキャン周期と同期を取ることによって、この入力が加えられた後に来る最初のサンプリングサイクル終了時にサンプリングを中断するように設計されています。そのため、必然的に、サンプリング中のトリガ入力及びサンプリング中断中のストップ入力、更にはストップ処理中のトリガ入力も内部的に無視されます。また、この信号と組み合わせたデジタルトリガ入力において、２回目以降のトリガ入力は、単なるサンプリングスタート信号として処理され、新たにトリガ検出を開始する事はありません。新しくトリガ検出を行うためには、一旦サンプリングを終了させる必要があります。（ポストトリガ、プリトリガ共に同じ条件となります。）

第４章．ソフトウェアとＷｉｎｄｏｗｓハンドラー

４－１．サンプルプログラム及び使用上の注意

本ユニット用に用意したサンプルプログラムは
- ＶＣ２００８．Ｎｅｔ（ＶＣ６＋）
- ＶＢ６
- ＶＢ２００８．Ｎｅｔ
- （ＶＣ２００８．Ｎｅｔ）
- ＬａｂＶＩＥＷ
- ~~ＣＢｕｉｌｄｅｒＰｌｕｓ１０~~
- ~~Ｃ＋＋Ｂｕｉｌｄｅｒ２００９~~
- ~~Ｄｅｌｐｈｉ２００９~~　　ＤｅｌｐｈｉＸＥ３

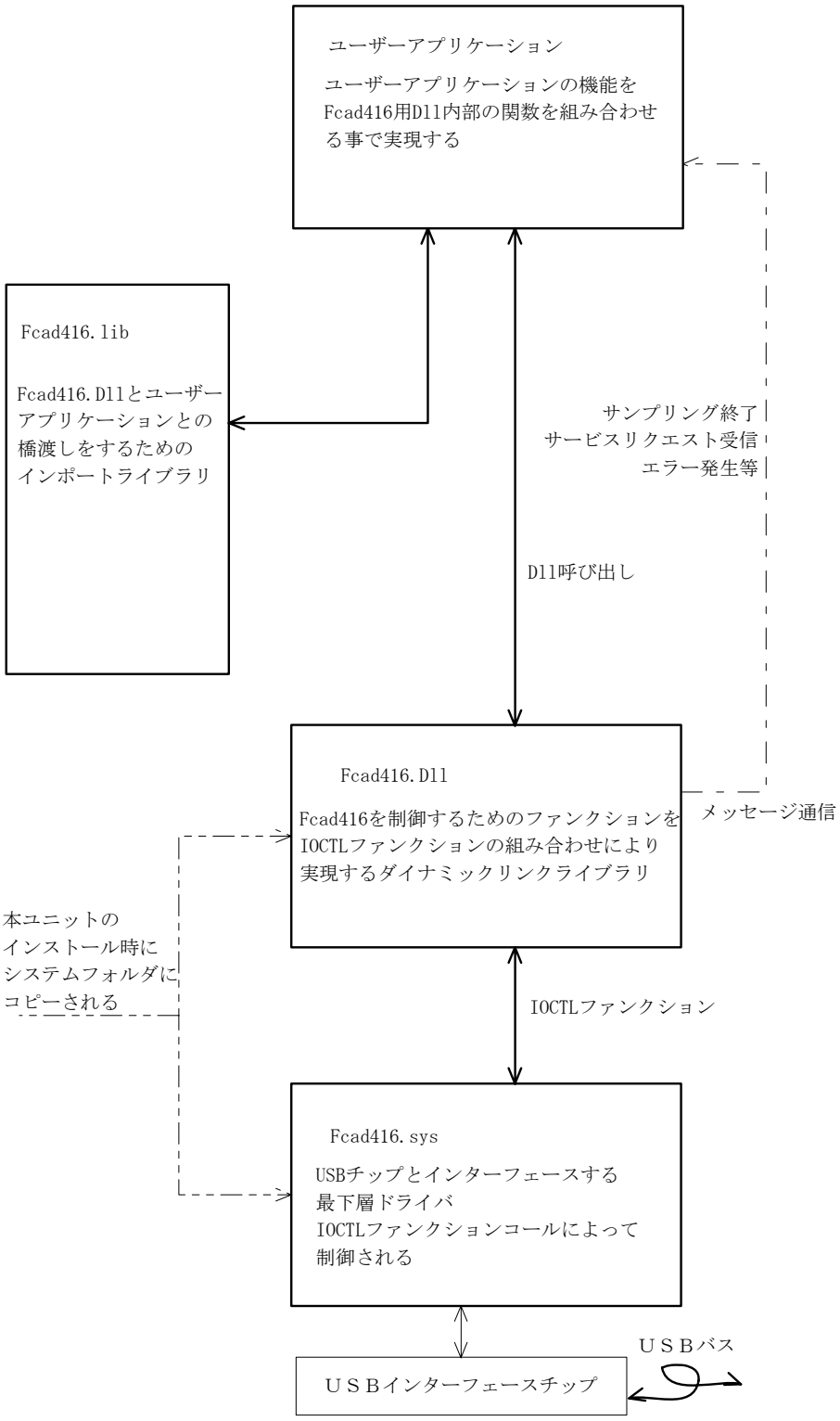
の５種類となりました。
ＶＣ６＋は、開発環境が手に入らなくなっていますが、開発環境をインストールする事無しに動作するプログラムを得るために用意してありました。しかしながら、Windows64ビットバージョンへの動作確認プログラム対応の為、開発環境をＶＳ６からＶＳ２００８に変更しＶＣ２００８．Ｎｅｔ環境で再構築しました。これに伴い、これまでのＶＣ２００８．Ｎｅｔサンプルプログラムは廃止しました。
Ｄｅｌｐｈｉ２００９は、ＤｅｌｐｈｉＸＥ３にバージョンを変更しています。
　使用上の注意事項としては、単体で動作させる場合、ユニットＩＤが０以外の値に設定してあると（つまりＩＤ＝０のユニットがない状態だと）ユニット検索でエラーが発生してしまう事が挙げられます。
また、複数ユニットが接続されている状態で、マスタースレーブ接続が（ＣＮ３によって）行われていないと、ユニット検索、オープンは正常に終了しますが、サンプリングを行わせようとするとアプリケーションがハングアップする事になります。これは、ユニットＩＤ！＝０のユニットがあるという事実からスレーブユニットの存在が認識されているにも関わらず、クロック待ち状態となっているスレーブユニットに同期クロック出力や同期トリガ出力が与えられないため、このユニットのサンプリング終了を無限に待ち続ける事態になってしまうためです。このような場合、一旦アプリケーションを終了し、ユニットＩＤやケーブル接続等を再確認してみて下さい。

２０１３年２月１１日　サンプルプログラムの種類を見直し
２００９年８月５日　サンプルプログラムの種類を追加

４－２．システム構成・ソフトウェア構造

本ユニットの運転・操作は、ＵＳＢバスを経由したコマンド・パラメータをパソコン本体とやり取りをする事によって行います。パソコン側では、その機能はＤｌｌ関数セットによって実現しています。また、本ユニット側ではＵＳＢ通信ＩＣによってコマンド・パラメータが解析され、ユニット上に搭載されたＦＰＧＡ、アナログ素子及びデジタル素子を制御することにより各種動作を実現しています。（第１章参照）

Ｗｉｎｄｏｗｓ環境での制御構造



４－３．サンプリングの様子とステータスデータ通信

ハンドラーの使用方法はとても簡単です。 具体的には４－５項以下で詳述しますが、要は使用するチャンネル数／サンプリング点数／クロック値／トリガ（スタート）条件等のパラメータをセットして各関数を呼ぶだけで、ＡＤデータはバッファに格納されて戻ってきます。

本ハンドラーでは大別して以下に記す３形態のサンプリング動作が可能です。

(1) マニュアルサンプリング

サンプリングスタート指示と同時に、１サイクル分のサンプリングを行い即終了します。

(2) ポストトリガ連続サンプリング

予め設定されたトリガ条件が検出されるまでサンプリングを開始せず、トリガ検出後指定されたサンプリング数（有限サンプリング）或いはサンプリング停止まで無限にサンプリングを行います(無限サンプリング)。
アナログトリガの場合には、論理先頭チャンネルのアナログ変換のみを最高速（４μＳ）で行いトリガ条件を探します。

(3) プリトリガ連続サンプリング

サンプリングスタート指示と同時に、与えられた条件に従いサンプリングを開始し、その結果を専用のメモリ（プリトリガバッファ）へ書き込みます。このメモリはリングバッファ管理されておりデータ書き込みがメモリの最後まで到達すると、再びメモリバッファの先頭から書き込みを行います。そしてトリガ条件が成立すると、次のサンプリングサイクルから、サンプリング結果のＦＩＦＯメモリ書き込みを開始します。そして、トリガ検出後のサンプリングサイクル数が指定回数に達すると自動的にサンプリングを終了します。ＦＩＦＯメモリからのデータ読み出しが終了すると、その後、プリトリガバッファからのデータ読み出しができる様になります。
この動作方式から分かるように、トリガ方式として即トリガには対応していません。また、トリガ検出タイミングはアナログトリガであってもサンプリング周期毎の検出となり、単チャンネルサンプリングの場合を除き最高速（４μＳ）での検出は行いません。この方式では有限サンプリングのみに対応しており、無限サンプリングには対応しておりません。

ステータスデータ通信

本ユニットでは、ステータス（サンプリング状態を示す）を常にユーザーに開示する必要があり、これをサンプリング中にも実現するため、データ取得ハンドラー内部でデータブロック取得サイクルの隙間を利用して間断なく行っています。

そのため、サンプリング中のステータス取得については若干の時間ずれが（通常のＵＳＢ通信処理時間の他に）生じます。
また汎用入力のサンプリング、汎用出力の更新についても同じタイミングで行っています。

４－４．使用準備

各ユニットにアナログ入力信号をコネクタＣＮ１によって接続します。この時、ユニットＩＤはマスターとして使用するユニットについては“０”を指定し、スレーブを使用する場合は、各々“０”以外の夫々異なる数値を指定します。また、スレーブを使用する場合は、コネクタＣＮ３同士によるマスタースレーブ接続も行う必要があります。ソフトウェア設定順序は以下の通りです。具体的には、サンプリングを開始する際には全てのパラメータを設定する必要があるという事になります。

(1)　ユニット初期化
(2)　ユニットオープン
(3)　トリガモード・レベル設定
(4)　チャンネル数、スキャン順、レンジ設定
(5)　入力モード（ＳＥ／差動）、アナログ変換コード指定
(6)　サンプリングクロック設定
(7)　スキャン速度決定
(8)　サンプリング開始
(9)　ステータス確認／メッセージ待ち
(10) サンプリング終了処理
(11)　(3)からの操作に戻るか、
　　或いは
(12) ハンドラー動作停止
(13) ハンドラー終了

※　ＦＣＡＤ４１６－ＤＳＵＢでは、仕様上の制限から、【7】トリガモード設定をその他の設定に先立って行っておく必要があります。

４－５．関数仕様・エラーコード

⑴初期化　　【1】
⑵ユニットオープン　　【2】
⑶トリガモード・レベル設定【7】
⑶サンプリング条件設定　　【5】～【9】,【13】
⑷スタート又はトリガ待ち　【10】
⑸ステータス評価　　【11】
⑹以下・途中は任意
⑺サンプリング停止　　【12】
⑻ハンドラー終了　　【3】,【4】

制御関数一覧

関数名	機能・内容	引数(パラメータ)等
【1】Fcad416_Open_Driver	ハンドラー初期化	
【2】Fcad416_Start_Driver	ハンドラー動作開始	
【3】Fcad416_Stop_Driver	ハンドラー動作停止	
【4】Fcad416_Close_Driver	ハンドラー終了	
【5】Fcad416_Set_SampCh	チャンネル数、スキャン順、レンジ設定	
【6】Fcad416_Set_InpMode	SE/差動入力、コード設定	
【7】Fcad416_Set_Trigger	トリガモード、レベル関係設定	
【8】Fcad416_Set_Clock	サンプリングクロック設定	
【9】Fcad416_Set_ScanSpeed	スキャン速度設定	
【10】Fcad416_Start_Samp	サンプリング開始	
【11】Fcad416_Get_Status	ステータス取得	
【12】Fcad416_Stop_Samp	サンプリング停止	
【13】Fcad416_Stop_Samp_Loop	サンプリング中の強制停止	
【14】Fcad416_Clear_Flags	ステータスフラグクリア	
【15】Fcad416_Set_SampLoop	Dll バッファの使用モード設定	
【16】Fcad416_Set_TransferSize	USB バス転送サイズ設定	
【17】Fcad416_Read_DllData	Dll 内バッファからデータ読み出し	
【18】Fcad416_Get_OneScan	マニュアルサンプリング	
【19】Fcad416_Set_ServiceReq_Pol	サービスリクエスト信号の極性を指定する	
【20】Fcad416_Get_ServiceReq	サービスリクエスト信号の状態を確認する	
【21】Fcad416_Clear_ServiceReq	サービスリクエスト信号をクリアする	
【22】Fcad416_Out_Aux	汎用デジタル出力更新（４ビット）	
【23】Fcad416_In_Aux	汎用デジタル現在値入力（４ビット）	
【24】Fcad416_Get_LibVer	Dll バージョン取得	
【25】Fcad416_Get_FirmVer	ファームウエアバージョン取得	
【26】Fcad416_Set_Memory	プリトリガバッファのサイズを設定する関数	
【27】Fcad416_Read_DirectRam	プリトリガバッファのデータを読み出す関数	
【28】Fcad416_Plus_Message	ユーザーへのメッセージ送信を指定する関数	
【29】Fcad416_Set_Calibration	キャリブレーションデータを書き込む関数	
【30】Fcad416_Get_Calibration	キャリブレーションデータを取得する関数	
【31】Fcad416_Handring_Eeprom	EEPROM とのデータ通信を行う関数	
【32】Fcad416_Get_SerialNumber	シリアル番号を取得する関数	
【33】Fcad416_Read_PriTrig_Buff	プリトリガバッファの内容を読み出す関数	

以下に各関数の詳細を示します。

【1】ハンドラー初期化

書式	int __stdcall Fcad416_Open_Driver(WORD *num_board , WORD *board_no);
引数	num_board:検出ボード枚数をセットする変数（ユーザー側にて確保） board_no:各ユニットのロータリースイッチ番号をセットする配列（ユーザー側にて確保:最大 16 個）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ハンドラー内部の初期化を行い、システム内で検出されたユニット台数及びそれぞれのユニットのロータリースイッチ番号を報告する。

使用例（Ｃ＋の場合）

```
if( (retc = Fcad416_Open_Driver( &num_board , &board_no[0])) != 0 )
        m_messagebox.SetWindowText("Fcad416_Open_Driver 失敗");
else if (num_board == 0)
        m_messagebox.SetWindowText("No Fcad416 board presence");
else
        m_messagebox.SetWindowText("Fcad416_Open_Driver 成功");
```

ここで、board_no[]配列（アプリケーション側で準備）には、USB バスの中でユニットが検索された順番にロータリースイッチ番号がセットされてきます。例えば、

```
num_board=3;
board_no[0]=5;
board_no[1]=3;
board_no[2]=0;
```

等のようになります。（この例でもわかるように、本ユニットを複数使用する際に注意することは、

１　ロータリースイッチ番号が０であるユニットが存在すること

２　すべてのロータリースイッチ番号がそれぞれユニークであること：連続した番号でなくてもよい

の 2 点です。）

この配列を使用して、ロータリースイッチ番号を指定して呼び出す関数は、例えば

```
for (loop = 0 ; loop < num_board ; loop++){
        if((retc = Fcad416_Set_InpMode( board_no[loop], inp_mode, ad_code)) != 0){
                m_messagebox.SetWindowText("SetInpMode fail");
                return;
        }
        else
                m_messagebox.SetWindowText("SetInpMode success");
}
```

のように使用します。このようなコーディング法を採ることで、ロータリースイッチ番号の連続性を不要とすることができます。

【2】ハンドラー動作開始

書式	int __stdcall Fcad416_Start_Driver(HWND Owner);
引数	Owner:アプリケーションのウインドーハンドル
戻り値	正常終了時:0 エラー発生時:負の値（複数ユニットの際は最初に検出されたエラーが返される。その為このエラーを解消しても次のユニットでさらにエラー報告がされる可能性がある）
機能・動作	オープンドライバで検出されたユニット群と通信を開始する。

使用例（Ｃ＋の場合）

```
if( (retc = Fcad416_Start_Driver( m_hWnd )) != 0 ){
        m_messagebox.SetWindowText("Fcad416_Start_Driver 失敗");
        return;
}
else
        m_messagebox.SetWindowText("Fcad416_Start_Driver 成功");
```

【3】ハンドラー動作停止

書式	int __stdcall Fcad416_Stop_Driver(void);
引数	なし
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	全てのユニットで使用していたメモリリソースを開放し通信を終了する。

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Stop_Driver( )) != 0 ){
        m_messagebox.SetWindowText("Fcad416_Stop_Driver 失敗");
        return;
}
else
        m_messagebox.SetWindowText("Fcad416_Stop_Driver 成功");
```

【4】ハンドラー終了

書式	int __stdcall Fcad416_Close_Driver(void);
引数	なし
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	全てのユニットについてハンドルを開放する。

使用例（Ｃ＋の場合）

```
if( (retc = Fcad416_Close_Driver()) != 0)
        m_messagebox.SetWindowText("Fcad416_Close_Driver 失敗");
else
        m_messagebox.SetWindowText("Fcad416_Close_Driver 成功");
```

【5】入力チャンネル数、スキャン順序、入力レンジ設定

書式	int __stdcall Fcad416_Set_SampCh(WORD board_no , int no_ch, int scan_order[], int range);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） no_ch:サンプリングチャンネル数（１～１６） scan_order[]:スキャン順序を指定する配列（０から最大１５までの配列により入力スキャン順序 ０番目から１５番目を指定する:no_ch 個の要素が必要） range:入力電圧範囲を指定する変数（０:±10.24V、１：±5.12V、２：±2.56V、３：±1.28V）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに、サンプリングチャンネル数、スキャン順序及び入力電圧範囲を通信する。またスキャン先頭チャンネル及びスキャン最終チャンネルの設定もこの関数で処理されるが、この処理はトリガモード設定後に行う必要がある為、トリガモード設定後でなければ、内部変数にいったん保存され、後に改めて出力される。また、この場合エラーが報告される。マスタースレーブ環境で、ユニット毎のチャンネル数設定に違いがあった場合もエラーとして報告される。

使用例（Ｃ＋の場合）

```
for (loop = 0 ; loop < num_board ; loop++){
        if((retc = Fcad416_Set_SampCh( board_no[loop], no_ch, scan_order, range[board_no[loop]])) != 0){
                m_messagebox.SetWindowText("Set sampch fail");
                return;
        }
        else
                m_messagebox.SetWindowText("Set sampch success");
}
```

この例では、no_ch（すべてのユニットで同じ値である必要があります）と scan_order（こちらのほうは、ユニット単位で組み合わせを変えても問題ありません）を共に、全ユニットで共通の形にしていますが、これは、サンプルプログラムとして簡略化を図ったための処置で他に意味はありません。

　また、機能・動作の項でも述べたように、トリガ設定関数実行後に、この関数を実行する必要があります。この関係を守らなかった場合、アプリケーション側にエラーが報告されます。

【6】シングルエンド/差動入力、データコード、レンジ設定

書式	int __stdcall Fcad416_Set_InpMode(WORD board_no , int inp_mode, int ad_code);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） inp_mode:シングルエンド/差動入力切り替え（０：シングルエンド、１：差動入力） ad_code:バイナリ/２の補数切り替え（０：バイナリ、１：２の補数）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに、シングル/差動、AD コードを通信する。 マスタースレーブモードでユニット毎にシングルエンド/差動入力設定が異なるとサンプリング開始時にエラーとして報告されサンプリングは行われない。

使用例（Ｃ＋の場合）

```
        if((retc = Fcad416_Set_InpMode( board_no[loop], inp_mode, ad_code)) != 0){
                m_messagebox.SetWindowText("SetInpMode fail");
                return;
        }
        else
                m_messagebox.SetWindowText("SetInpMode success");
```

【7】トリガモード、レベル関係設定

書式	int __stdcall Fcad416_Set_Trigger(int trg_mode, int trg_source,int trg_type, int trig_level1 ,int trig_level2 , int Force_Stop);
引数	trg_mode:サンプリングモード（０：ポストトリガ、１：プリトリガ、２：クロックオンリー） trg_source:トリガーソース指定（０：即トリガ、１：アナログトリガ、２：デジタルトリガ） trg_type:トリガタイプ指定（０：立ち下がりエッジ、１：立ち上がりエッジ、２：負レベル、３：正レベル、４：アウトレンジ、５：インレンジ、６：デュアルスロープマイナス、７：デュアルスローププラス） trg_level1:アナログトリガレベル１（０～２５５） trg_level2:アナログトリガレベル２（０～２５５） Force_Stop:間歇動作制御入力を使用するかどうかを指定するフラグ、１なら使用可能を０なら使用不可能を指示する。
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号０のユニットに、トリガ条件を通信する。アナログトリガレベルについてはレベル２＞レベル１という設定にしないとエラーとして処理される。更に、インレンジ・アウトレンジではレベル２≧レベル１＋２という関係が成立しないと、エラーとして処理される。またスレーブユニットに対してはトリガーソースとタイプ指定について固定の条件が設定される。（デジタルトリガ、立ち下がりエッジ）それ以外の条件は指定されたものを使用する。間歇動作制御入力については、マスタースレーブ接続時には、マスターユニットに接続されたもののみが有効となる。スレーブ側は、マスターの動作に追随するため制御入力としては不要である。また間歇動作制御入力は外部トリガ（デジタルトリガ）使用時のみ有効である。

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Set_Trigger( trg_mode, trg_source, trg_type, trig_level1, trig_level2, StopFunc)) != 0){
        m_messagebox.SetWindowText("Trigger Fail");
        return;
}
else
        m_messagebox.SetWindowText("Trigger set success");
```

トリガモード、トリガレベルの設定は、ロータリースイッチ番号０のユニットに対してのみ行ないます。スレーブユニット（もし存在していれば）については、サンプリングモードによる固定した値をセットすればよいため、ユーザープログラムから指定する必要は何もありません。

【8】クロック源、クロック周期、外部クロック等関係設定

書式	int __stdcall Fcad416_Set_Clock(int clk_source, int set_mode, int time_unit, int clk_period, int exclk_freq);
引数	clk_source:クロック源指定（０：内部クロック１：20MHz、１：内部クロック２：16.384MHｚ、２：外部クロック立下りエッジ20MHz、３：外部クロック立ち上がりエッジ20MHz、４：外部クロック立下りエッジ16.384MHz、５：外部クロック立ち上がりエッジ16.384MHz） set_mode:クロック値の設定方法（０：クロック周期、１：分周比） time_unit:クロック周期の値または分周比 clk_period:クロック周期の値で指定する場合の単位（０：Ｓ、１：ｍＳ、２：ｕＳ） exlck_freq:外部クロック源を使用し、クロック周期の値で動作を指定する場合の周波数値（Ｈｚ）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号０のユニットに、クロック設定条件を通信する。またスレーブユニットには固定の条件を設定する。（外部クロック立ち下がりエッジ、分周比設定、分周比＝１）

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Set_Clock( clk_source, set_mode, time_unit, clk_period, exclk_freq)) != 0){
        m_messagebox.SetWindowText("Set clock fail");
        return;
}
else
        m_messagebox.SetWindowText("Set clock success");
```

クロック（サンプリングクロック）の設定は、ハンドラー内部では、常に基本クロック（内部クロック又は外部クロック）に対する分周比によって決定されます。そのため、この関数を使用してクロック周期を設定する際、クロックの種類（内部クロックか外部クロックか）及び分周比指定か周期指定かによって、ハンドラー内部の動作が微妙に異なってきます。

分周比指定でクロック周期を指定する場合、内部クロックを使用するのであれば、手計算によって決定した値を使用すれば問題ありません。この場合、最大値としては、２１４．７秒（20MHz時）／２６８．４秒（16MHz時）となります。

例題：内部クロック１（２０MHｚ）を使用して６４μＳのサンプリング周期を分周比指定で設定する場合

分周比＝６４×１０$^{-6}$÷（１÷２０×１０^{6}）＝６４×１０$^{-6}$÷５０×１０$^{-9}$＝１２８０　となります。

また、外部クロックを使用して分周比指定によるサンプリング周期指定の場合も、同様の考え方で分周比を計算することができます。但し、この場合後述するスキャンクロックの周期計算のため**使用する内部クロックの選択を同時に行っておく**必要があります。（スキャンクロック周期は内部クロックを常に使用するため）

例題：外部クロックとして１６MHｚのクロックを使用し、６４μＳのサンプリング周期を得ようとする場合

分周比＝６４×１０$^{-6}$÷（１÷１６×１０^{6}）＝６４×１０$^{-6}$÷６２．５×１０$^{-9}$＝１０２４　となります。

一方、周期指定の場合は、指定された周期と使用するクロックの周波数を基にして、ハンドラー内部で上記と同様の計算を行い、求めた分周比をユニットへ送信します。そのため、外部クロックを使用するときには、そのクロック周波数をパラメータとして指定する必要がある訳です。また、この方式（周期指定方式）では、計算の結果、分周比が整数にならない可能性も高く、その場合、最も指定した周期に近い計算値を上回った分周比になります。

例題：内部クロック２（１６．３８４MHｚ）を使用して６４μＳのサンプリング周期を周期指定で設定する場合

分周比＝６４×１０$^{-6}$÷（１÷１６．３８４×１０^{6}）＝６４×１０$^{-6}$÷６１．０３５×１０$^{-9}$≒１０４８．５７６となります。このような場合、分周比としては１０４９となるわけです。

例題：外部クロックとして１０．２４MHｚを使用し、６４μＳのサンプリング周期を得ようとする場合
分周比＝６４×１０$^{-6}$÷（１÷１０．２４×１０^{6}）＝６４×１０$^{-6}$÷９７．６５６２５×１０$^{-9}$＝６５５．３６
となります。このような場合、分周比としては６５６が選ばれる事になります。

【9】スキャン速度設定

<table>
<tr><td>書式</td><td colspan="4">int __stdcall Fcad416_Set_ScanSpeed(int ScanClkMode,int ScanSpeed);</td></tr>
<tr><td>引数</td><td colspan="4">ScanClockMode:スキャンクロック指定を周期で指定するか分周比で指定するかを切り分ける変数
０の場合：周期指定
１の場合：分周比指定
ScanSpeed:チャンネルあたりのサンプリング速度または分周比である。実際の値は下記による。
周期指定の場合　４から３２７６：２０MHｚ、４から３９９９：１６．３８４MHｚ（単位μＳ）
分周比指定の場合　２０から６５５３５：２０MHｚ、６６から６５５３５：１６．３８４MHｚ</td></tr>
<tr><td></td><td colspan="2">基本クロック　２０MHｚの場合</td><td colspan="2">基本クロック　１６．３８４MHｚの場合</td></tr>
<tr><td></td><td>分周比</td><td>８０</td><td>６５５３５</td><td>６６</td><td>６５５３５</td></tr>
<tr><td></td><td>設定周期</td><td>４μＳ</td><td>３２７６．７５μＳ</td><td>４．０２８３２μＳ</td><td>３９９９．９４μＳ</td></tr>
<tr><td>戻り値</td><td colspan="4">正常終了時:0
エラー発生時:負の値</td></tr>
<tr><td>機能・動作</td><td colspan="4">全てのユニットに、スキャンクロックの値を通信する。基本クロックとしては、Fcad416_Set_Clock関数で予め設定されている内部クロックの値を使用する。また外部クロックを設定した場合はその設定値によって、基本クロックとして２０MHｚクロック或いは１６．３８４MHｚを使用する。また周期指定で設定不可能な値の場合、直近のより長い周期の設定となる。例えば１６．３８４MHｚクロック使用時、４μＳ指定の実際の周期は４．０２８３２μＳとなる。(これは分周比６６に相当する。)</td></tr>
</table>

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Set_ScanSpeed( ScanClockMode , Scanclockvalue)) != 0){
        m_messagebox.SetWindowText("ScanSpeed set fail");
        return;
}
else
        m_messagebox.SetWindowText("ScanClock set success");
```

スキャンクロックの設定は、【８】クロック源、クロック周期、外部クロック等関係設定で決定した内部クロックの値と、この関数の引数（ScanClockMode）とから決定されます。しかし何れにせよ、最終的にはユニットに対して分周比を設定することになるため、【８】項で述べたように周期指定で設定を行っても分周比として値が切り上げになってしまうことがあります。

例題：【８】項の設定で内部クロック２（１６．３８４MHｚ）を使用することになっている前提でスキャンクロック周期を４μＳと指定した場合

分周比＝$4\times10^{-6}\div(1\div16.384\times10^{6})=4\times10^{-6}\div61.035\times10^{-9}\fallingdotseq65.536$となりますが、実際に設定できる分周比は整数に限られるため、６６という分周比が使用されることになります。この場合のスキャンクロック周期は上で述べているように４．０２８３２μＳとなります。(近似値)

例題：【８】項の設定で内部クロック２（１６．３８４MHｚ）を使用することになっている前提でスキャンクロック周期を４μＳとすべく、分周比指定で６５を選択した場合、この設定ではスキャン速度最低４μＳという条件が守られないため（実際の周期は３．９６７３μＳ程になる）ハンドラーでエラーと判断されてしまい設定はできません。

【10】サンプリング開始

<table>
<tr><td>書式</td><td colspan="3">int __stdcall Fcad416_Start_Samp(int post_samp);</td></tr>
<tr><td>引数</td><td colspan="3">post_samp:トリガ成立後のチャンネル当たりサンプリングデータ数</td></tr>
<tr><td>戻り値</td><td colspan="3">正常終了時:0
エラー発生時:負の値、または不足パラメータを示す正の値</td></tr>
<tr><td>機能・動作</td><td colspan="3">全てのユニットに、サンプリング開始を通信する。この際、まず整合性を確認する必要があるパラメータについて確認を行い、次にスレーブユニットを起動し、最後にマスターユニットを起動する。
確認を行う内容
【５】Fcad416_Set_SampCh：ユニットによる入力チャンネル数の差異
【６】Fcad416_Set_InpMode：シングル/差動の組み合わせはないか
【７】Fcad416_Set_Trigger：トリガ指定は行なわれているか
【８】Fcad416_Set_Clock：クロック指定は正常に行なわれているか
【９】Fcad416_Set_ScanSpeed：スキャン速度設定は正常に行われているか
が確認され、異常がある場合サンプリングは開始されない。この場合アプリケーション側へ返されるエラーコードは以下の通りとなる。
<table>
<tr><td>ビット番号</td><td>エラー発生条件</td><td>特記事項</td></tr>
<tr><td>Ｄ７＝１</td><td>Fcad416_Set_SampCh 関数が未実行又は異常終了</td><td></td></tr>
<tr><td>Ｄ６＝１</td><td>Fcad416_Set_Trigger 関数が未実行又は異常終了</td><td></td></tr>
<tr><td>Ｄ５＝１</td><td>Fcad416_Set_Clock 関数が未実行又は異常終了</td><td>Ｄ４もセットされる</td></tr>
<tr><td>Ｄ４＝１</td><td>Fcad416_Set_Clock 関数が未実行又は異常終了</td><td>Ｄ５もセットされる</td></tr>
<tr><td>Ｄ３＝１</td><td>Fcad416_Set_ScanSpeed 関数が未実行又は異常終了</td><td></td></tr>
<tr><td>Ｄ２＝１</td><td>Fcad416_Set_InpMode 関数が未実行又は異常終了</td><td></td></tr>
<tr><td>Ｄ１＝１</td><td>Fcad416_Set_SampCh 関数が未実行又は異常終了</td><td>トリガ条件未設定</td></tr>
<tr><td>Ｄ０＝１</td><td>Fcad416_Set_SampCh 関数が未実行又は異常終了</td><td>トリガ条件未設定</td></tr>
</table>
</td></tr>
</table>

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Start_Samp( post_samp)) != 0){
        m_messagebox.SetWindowText("Sampling start fail");
        return;
}
else
        m_messagebox.SetWindowText("Sampling start success");
```

この関数は、これまでに設定された諸パラメータを使用して実際にサンプリングを開始する指示をユニットへ送信する処理です。まず各パラメータの整合性を確認し、問題があれば正の数を戻り値としてユーザー側へエラー報告が行われ、サンプリングは開始されません。実際の値としては１から２５５までの間の値を取ります。（上表のエラーコードとビット番号の関係を参照して下さい）

設定したパラメータの関係に問題がなければ(マスタースレーブ接続になっている場合)最初にスレーブユニット各々にサンプリング開始が指示され、すべてのスレーブユニットはマスターユニットからのサンプリング開始信号待ち状態に移行します。そして次にマスターユニットに対してサンプリング開始が指示されサンプリング動作が開始されます。この時、ハンドラー内部では、post_samp 引数とサンプリングチャンネル数とから計算された容量のＤ１１バッファがユニット毎に確保されています。

また後述する【１５】Ｄ１１バッファ使用モード切替関数によりＤ１１バッファがリングバッファーモードとなっている場合も、Ｄ１１バッファのサイズはまったく同じように計算されます。そのため、リングバッファモード（別称無限サンプリングモード）の場合も post_samp 引数にはある程度の大きさを指定しておいたほうが安定した動作を行うことができます。実際の値がどの程度になるのかは、アプリケーションの内容によって千差万別ですので、ある程度の試行錯誤によって値を決定されることが望ましいと考えられます。

【11】ステータス取得

<table>
<tr><td>書式</td><td>int __stdcall Fcad416_Get_Status(WORD board_no , DWORD *post_sampled, DWORD *pre_sampled, int status[]);</td></tr>
<tr><td>引数</td><td>board_no:通信相手のロータリースイッチ番号（０～１５）
post_sampled:チャンネル当たりの転送済みトリガ後データ点数
pre_sampled:チャンネル当たりのプリトリガ済みデータ点数
status[]:ロータリースイッチ番号で指定したユニットからのステータス戻り値（２要素）
<table>
<tr><td colspan="3">Ｓｔａｔｕｓ［1］</td><td colspan="3">Ｓｔａｔｕｓ［0］</td></tr>
<tr><td>ビット</td><td>Ｄ３１からＤ８：予約ビット</td><td>通常</td><td>ビット</td><td>Ｄ３１からＤ８：予約ビット</td><td>通常</td></tr>
<tr><td>Ｄ７</td><td></td><td>０</td><td>Ｄ７</td><td>連続サンプリング終了</td><td>０</td></tr>
<tr><td>Ｄ６</td><td></td><td>０</td><td>Ｄ６</td><td>各回サンプリング終了</td><td>０</td></tr>
<tr><td>Ｄ５</td><td>強制停止完了フラグ</td><td>０</td><td>Ｄ５</td><td></td><td>０</td></tr>
<tr><td>Ｄ４</td><td>ロールアップフラグ</td><td>０</td><td>Ｄ４</td><td></td><td>０</td></tr>
<tr><td>Ｄ３</td><td>無限ループラップラウンド</td><td>０</td><td>Ｄ３</td><td>データロストエラーフラグ</td><td>０</td></tr>
<tr><td>Ｄ２</td><td>サービスリクエスト要求発生</td><td>０</td><td>Ｄ２</td><td>ＦＩＦＯフル</td><td>１</td></tr>
<tr><td>Ｄ１</td><td>サンプリングクロック先端</td><td>０</td><td>Ｄ１</td><td>ＦＩＦＯハーフフル</td><td>１</td></tr>
<tr><td>Ｄ０</td><td>トリガ発生認識</td><td>０</td><td>Ｄ０</td><td>ＦＩＦＯノットエンプティ</td><td>０</td></tr>
</table>
強制停止完了フラグ：あらかじめ許可しておいた本ユニットへの強制停止入力が実際に加えられ、それによるサンプリングの停止が行なわれた事を示す状態フラグである。
ロールアップフラグ：プリトリガバッファに書き込まれるデータが一巡したことを示すフラグである。但し巡回回数のカウントは行わない。
無限ループラップラウンド：無限サンプリング時に、用意されたＤ１１バッファを一巡した事を示すフラグである。取り込まれたデータがＤ１１バッファを一巡する度にセットされるため、サンプリング終了までに複数回セットされる可能性があるが、バッファ長が有限であるためセットされる回数についてはカウントしていない。
サービスリクエスト要求発生：本ユニット外部からサービス要求が到着したことを示すフラグである。
サンプリングクロック先端：サンプリングサイクルの開始を検出し設定されるフラグである。
トリガ発生認識：設定されたトリガ条件が検出されたことを示すフラグである。
連続サンプリング終了：有限サンプリングにおいて指定回数のサンプリングが終了した事を示すフラグである。
各回サンプリング終了：一回のサンプリングサイクルが終了した事を示すフラグである。
データロストエラーフラグ：内部ＦＩＦＯからの読み出しが書き込みに追いつかずＦＩＦＯがあふれた際にセットされるフラグである。
ＦＩＦＯフル：内部ＦＩＦＯが満杯状態である場合セットされる状態フラグである。
ＦＩＦＯハーフフル：内部ＦＩＦＯが指定データ数以上のデータ書き込みされたことを示す状態フラグである。
ＦＩＦＯノットエンプティ：内部ＦＩＦＯに最低でも１データが書き込まれていることを示す状態フラグである。
強制停止完了フラグ、ロールアップフラグ及びＦＩＦＯ関連の３ビットについては状態フラグなので特にクリアなどを行う必要はないが、これ以外のフラグについては、ラッチフラグなので再度検出するためには該当するフラグをクリアする必要がある。</td></tr>
<tr><td>戻り値</td><td>正常終了時:0（サンプリング停止中）、１（サンプリング動作中）
エラー発生時:負の値</td></tr>
<tr><td>機能・動作</td><td>ロータリースイッチ番号で指定したユニットから、ステータスを取得し同ユニットからの読み込み済みデータ点数を返す。但しサンプリング中には、別スレッドでのステータス確認結果がユーザーに返却され直接ユニットとの通信は行われない。また、この場合、又はサンプリングスレッドが動作中には、戻り値として“1”が返される。</td></tr>
</table>

使用例（Ｃ＋の場合）
使用例　１　（サンプリングが終了するまで待つ）

```
do{
        retctotal = 0;
        for (loop = 0 ; loop < num_board ; loop++){
                retc[loop] = Fcad416_Get_Status( board_no[loop], &Post_Sampled, &Pre_Sampled, status);
                // board_no[loop]番目の ID を持つユニットのステータスを確認する
                if (retc[loop] < 0)
                        goto Err_Exit;                  // エラーが発生したら後の処理は飛ばす
                retctotal = retctotal + retc[loop];     // do ループ処理を抜ける条件を構築する
        }
}while(retctotal != 0);         // Get_Status 関数の戻り値は、サンプリングスレッドが停止するまで０にならないので
```

この場合、Fcad416_Get_Status 関数が、サンプリングスレッド動作中には戻り値として“１”を返すことを利用しそのスレッド終了を待つ仕組みを実現しています。また、エラーが返ってきた場合には、後の処理を飛ばす工夫もしてあります。

使用例　２　（サンプリングデータ数を取得する）

```
for (loop = 0 ; loop < num_board ; loop++){
        if ((retc = Fcad416_Get_Status( board_no[loop], &post_sampled[board_no[loop]],¥
                &pre_sampled[board_no[loop]], status[board_no[loop]])) != 0) {
                m_messagebox.SetWindowText("GetStatus  失敗");
                retc_total += retc;
        }
        else{
                itoa( status[board_no[loop]][1] * 256 + status[board_no[loop]][0] , str , 16);
                m_GetStatus.SetWindowText( str );               // ステータス値を表示
                itoa( post_sampled[board_no[loop]] , str , 16);
                m_postsample.SetWindowText( str );              // ポストサンプリングデータ数を表示
                itoa( pre_sampled[board_no[loop]] , str , 16);
                m_presample.SetWindowText( str );               // 実際のプリトリガデータサイズを表示
        }
}
```

この例では、エラーの判断も行いながら、現在までにサンプリング済みとなっているデータ数を取得しています。
ここではプリトリガサンプリングデータ数も取得していますが、実際のプリトリガデータを取得できるのは、
サンプリング後データを取得した後になります。

【12】サンプリング停止

書式	int __stdcall Fcad416_Stop_Samp(void);
引数	なし
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	全てのユニットに、サンプリングストップを通信する。

使用例（Ｃ＋の場合）

```
Fcad416_Stop_Samp( );                    // スタートビットのみを抑止する
```

この関数を発行すると、接続中のすべてのユニットに対し、サンプリング停止を指示します。

【13】サンプリング中の強制停止

書式	int __stdcall Fcad416_Stop_Samp_Loop(void);
引数	なし
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ユニットを強制的にストップするためのフラグをセットする関数。実際のサンプリング停止は、ＵＳＢ通信の切れ目に送信される。ユーザープログラムには実際の停止とは無関係に直ちに制御が戻る。また、この関数を発行する事でデータ取得スレッドも動作を停止する。

使用例（Ｃ＋の場合）

```
// 無限サンプリング時の処理
if((retc = Fcad416_Stop_Samp_Loop( )) != 0){
        m_messagebox.SetWindowText( "Stop samp fail" );
        return;
}
else
        m_messagebox.SetWindowText( "Stop samp success" );
```

この関数は、無限サンプリング実行時に何らかの要因によってサンプリングを中断したい場合に発行するものです。この関数を発行することにより各ユニットのサンプリングは停止します。ただ、実際のサンプリング停止は、サンプリングデータを取得するデータ通信の狭間に送信されるコマンドによって行われるため、この関数発行との間にはある程度の時間的なずれが存在します。

【14】ステータスフラグクリア

書式	int __stdcall Fcad416_Clear_Flags(WORD board_no, int clear_bits);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） clear_bits:クリアすべきステータスフラグのビット指定（セットするとクリアされる） ビット０：データロストビットクリア ビット１：予約ビット ビット２：予約ビット ビット３：各回サンプリング終了フラグクリア ビット４：サンプリング終了フラグクリア ビット５：トリガビットクリア ビット６：サンプリングクロックフロントエッジ検出フラグクリア ビット７：サービスリクエスト要求ビットクリア ビット８：無限ラップラウンドクリア ビット９～ビット３１：予備（値は任意）
戻り値	常に正常終了
機能・動作	ステータスフラグの内指定したビットをクリアする。

この関数によって（必要であれば一部の）フラグをクリアすることができます。個別にクリアする要因としては、無限サンプリング実行時のＤ１１バッファラップラウンドフラグ、或いは各回サンプリング終了フラグ（このフラグは一回のスキャンが終了するとセットされるため特に低速度でサンプリングを行う場合のデータ取り込みタイミングとして有用です。）等が優位性のあるフラグではないかと思われます。
また、サービスリクエスト要求ビットも外部装置とのそれほど時間的にシビアではないタイミングあわせの手段としては有効だと考えられます。

【15】Dll バッファ使用モード切替

書式	int __stdcall Fcad416_Set_SampLoop (int ring);
引数	ring:Dll バッファを通常モード・リングバッファモードのどちらにするかを設定する変数 (０：通常モード、１：リングバッファモード)
戻り値	常に正常終了
機能・動作	Dll バッファを通常モードで使用するか、リングバッファモードで使用するかを設定する。 リングバッファモードに設定すると、同時にサンプリングモードは有限サンプリングから 無限サンプリングに変更される。

使用例（Ｃ＋の場合）

```
if (retc = Fcad416_Set_SampLoop( ringmode )){
        m_messagebox.SetWindowText( "Fcad416_Set_SampLoop 失敗" );
        return;
}
else{
if (ringmode == 0)
        m_messagebox.SetWindowText( "通常サンプリングモード設定" );
else
        m_messagebox.SetWindowText("リングバッファーモード設定");
}
```

この例では、あらかじめ数値が設定されている ringmode 変数を使用してサンプリングモードを通常モードまたはリングバッファモードに切り替える様子が表現されています。

【16】USB バス転送サイズ設定

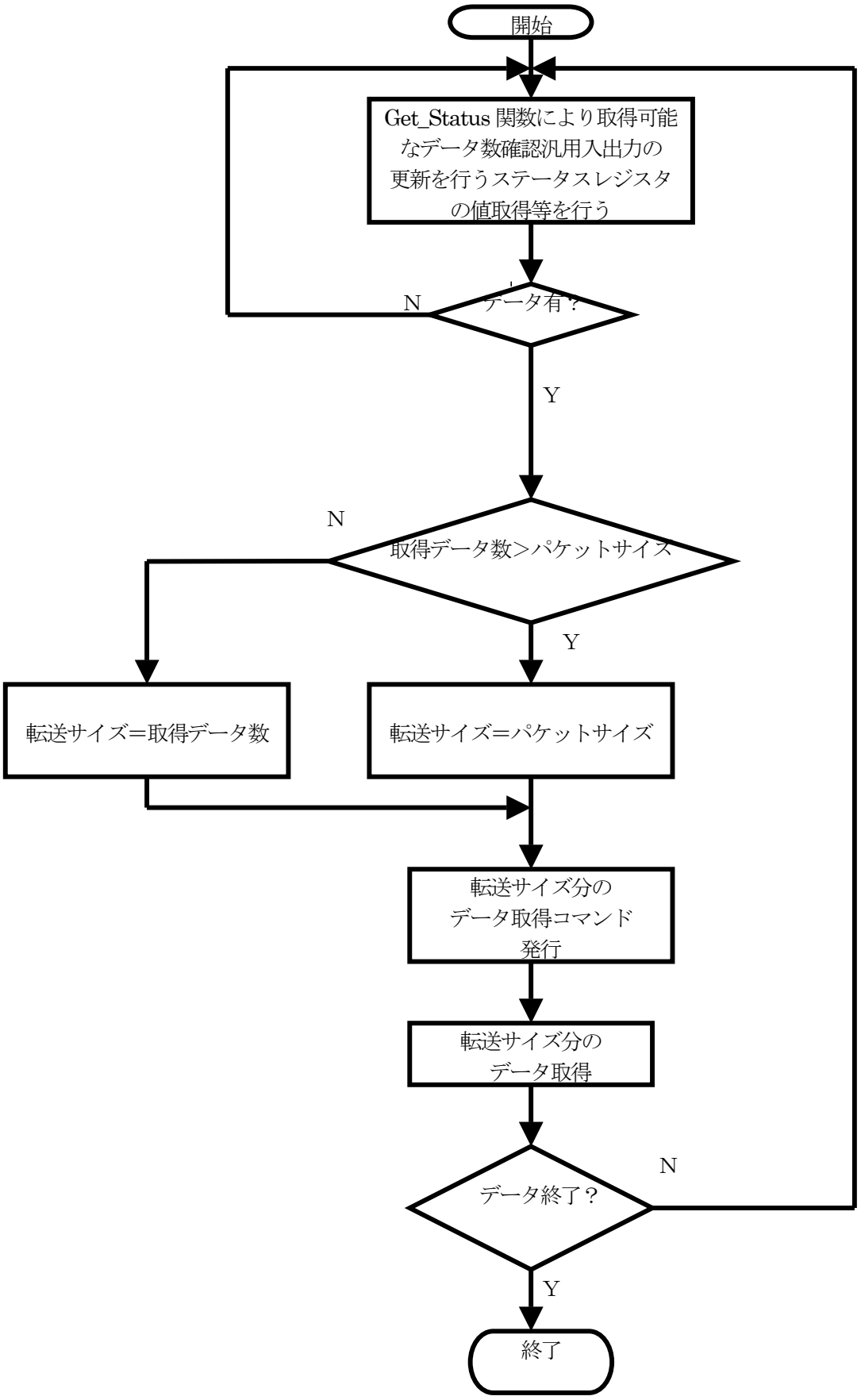
書式	int __stdcall Fcad416_Set_TransferSize(int size);
引数	size:通信相手のユニットとＰＣ間のデータ転送パケットサイズを指定する変数であり、単位はチャンネル当たりの転送データ数を示す（０：１語から１３：８K 語まで可能）size 変数と実際のパケット値の関係は以下のようになっている。 size：$2^{(size)}$×（サンプリングチャンネル数）語
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ユニットとの、ＡＤデータ転送のパケットサイズを指定する。 サンプリング途中での GetStatus 関数に対する読み込み済みデータ数は、このパケット単位になる。 又、このパケット読み込み単位で汎用入出力等の更新も行うので高速転送を狙ってパケットサイズを大きくするか、汎用入出力等の更新を頻繁に行うためにパケットサイズを小さくするかは アプリケーション側での選択にゆだねられる。（４－３項参照）尚、アプリケーション起動時は８K語の設定になっている。

使用例（Ｃ＋の場合）

```
transfersize = m_transfersize.GetCurSel();                    // 転送バッファサイズを得る
if((retc = Fcad416_Set_TransferSize( transfersize )) != 0){
        m_messagebox.SetWindowText("Transfersize set fail.");
        return;
}
else
        m_messagebox.SetWindowText("Set transfersize success.");
```

ここでは、ドロップダウンリストのインデックスを使用して、パケット転送サイズを設定しています。
本ハンドラー内部でのデータ取得処理について大まかな流れを次ページに示します。

本ハンドラーでのデータ読み込み処理は次のようになっています。
実際には、複数ユニット間の同期を取るなど、内部ではより複雑な処理を行っています。
また、下記に示すパケットサイズは、本関数で指定した値がハンドラー内部で使用されます。



【17】Dll バッファ内部に読み込まれたデータをユーザーへ渡す

書式	int __stdcall Fcad416_Read_DllData(WORD board_no , DWORD no_data, DWORD data_pos, WORD *bufptr, DWORD buf_size);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） no_data:チャンネル当たりの読み出しデータ点数 data_pos:読み出すデータの Dll バッファ先頭からのオフセット（チャンネル当たり） bufptr:ユーザーバッファ先頭アドレス buf_size:ユーザーバッファサイズ（バイト単位）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットから読み込まれている Dll バッファ内のデータをユーザーバッファに転送する。

使用例（Ｃ＋の場合）

```
if ((retc = Fcad416_Read_DllData( SelectedBoard, memcount / no_ch, 0, buffer, memcount * 2)) != 0){
        m_messagebox.SetWindowText("ReadDllData fail");
        return;
}
else
        m_messagebox.SetWindowText("ReadDllData success");
```

この例では、“SelectedBoard”変数で指定されたユニットから読み込まれたデータを、総データ数/チャンネル数(=memcount/no_ch)データ、Ｄｌｌバッファの先頭から、予め確保している buffer の領域へ取り込むという処理を行っています。

【18】１サイクルサンプリング（マニュアルサンプリング）

書式	int __stdcall Fcad416_Get_OneScan(WORD board_no , WORD *bufptr, int buf_size);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） bufptr:ユーザーバッファ先頭アドレス buf_size:ユーザーバッファサイズ（バイト単位）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに、マニュアルサンプリング指示を送信しスキャンデータを得る。buf_size が不足している場合はサンプリングが行なわれず、エラーが返される。

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Get_OneScan(SelectedBoard, bufptr , no_ch * 2)) != 0)
        m_messagebox.SetWindowText("Fcad416_Get_OneScan 失敗");
else
        m_messagebox.SetWindowText("Fcad416_Get_OneScan 成功");
```

この例では、SelectedBoard 変数で指定されたユニットに対して bufptr 変数で指定したバッファエリアを用意した上で no_ch チャンネル数でのマニュアルサンプリング指示を行っています。サンプリングの結果は、bufptr 変数で示されるバッファエリアに格納されて制御が返ってきますので、その後必要な処理（例えば数字データを ascii コードに変換する等）を行ないます。

【19】サービスリクエスト入力の極性を指定する

書式	int __stdcall Fcad416_Set_ServiceReq_Pol(WORD board_no , WORD sig_pol);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） sig_pol:サービスリクエスト入力の極性指定（０：立下り、１：立ち上がり/０がデフォルト）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに、サービスリクエスト信号の極性指定を送信する。 この操作で、サービスリクエスト要求そのものも有効化され Fcad416_Clear_ServiceReq 関数発行でサービスリクエスト要求が終了する。また、指定しようとしているユニットがオープン状態ではない場合、或いはすでにサンプリング中の場案、要求は受け付けられない。

使用例（Ｃ＋の場合）

```
if (board_no[loop] == 0){
// マスターユニットのみメッセージを受ける
        if(MessageRequest[board_no[loop]] & 0x08)
                Fcad416_Set_ServiceReq_Pol( board_no[loop] , 0x01 );
}
```

この例では、マスターユニットについて、後述する Fcad416_Plus_Message 関数でサービスリクエストからのメッセージ処理が追加されていた場合、サービスリクエスト入力の立ち上がりエッジを検出してメッセージ送信を行うよう指定を行っています。(本来サービスリクエスト要求はスレーブユニットであっても処理することが可能ですが、本例題では簡略化するため処理要求をマスターユニットに限定しています。)

【20】サービスリクエスト信号の状態を確認する

書式	int __stdcall Fcad416_Get_ServiceReq(WORD board_no , BYTE *service_req);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） service_req:サービスリクエスト信号の状態（０：リクエスト無し、１：リクエストあり）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	【19】で指定したユニットに対し、指定した極性のサービスリクエスト信号の有無を確認する。 この関数も、サンプリング中にはサンプリングデータ読み込みスレッドでのステータス確認結果がユーザーに渡され、直接ユニットと通信は行わない。

使用例（Ｃ＋の場合）

```
Fcad416_Get_ServiceReq( SelectedBoard , &service_req);
if (service_req == 1){
        m_ServiceRes.SetWindowText( "Service Request recieved" );
        Fcad416_Clear_ServiceReq( SelectedBoard );
}
else
        m_ServiceRes.SetWindowText( "No Service Request" );
```

ここでは、後述する Fcad416_Clear_ServiceReq 関数と組み合わせ、SelectedBoard 変数で指定されたユニットに対し、サービスリクエスト信号の確認作業を行うコードを示しています。

【21】サービスリクエスト信号をクリアする

書式	int __stdcall Fcad416_Clear_ServiceReq(WORD board_no);
引数	board_no:通信相手のロータリースイッチ番号（０～１５）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	【20】で確認したサービスリクエスト信号をクリアする。同時にサービスリクエスト要求そのものも終了する。この関数は、サンプリング中には実行できず、エラーコードが返される。

使用例（Ｃ＋の場合）

```
case 6:                                    // サービスリクエスト受信メッセージ
        sprintf(buf, "Service Request Received from ID%d",wParam);
        m_ServiceRes.SetWindowText( buf );
        Fcad416_Clear_ServiceReq( SelectedBoard );
        break;
```

ここでは、メッセージハンドラーの一部として、サービスリクエスト信号のクリアコマンドを使用しています。

【22】汎用デジタル出力更新（４ビット）

書式	int __stdcall Fcad416_Out_Aux(WORD board_no , BYTE outdata);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） outdata:汎用デジタル出力端子への出力値（４ビット）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに、汎用デジタル出力更新を指示する。但し サンプリング中には、サンプリングデータ読み込みスレッドでのステータス確認コマンドによって出力データが通信されるため、この関数発行のタイミングではユニットとの通信は行わない。

使用例（Ｃ＋の場合）

```
if (( retc = Fcad416_Out_Aux( SelectedBoard , OutputData)) != 0){
        m_messagebox.SetWindowText("汎用出力失敗");
        return;
}
else
        m_messagebox.SetWindowText("汎用出力完");
```

ここでは、SelectedBoard 変数で指定されるユニットに対し、汎用出力を行うコードが記述されています。但し上の機能・動作の項でも説明されているように、サンプリング中にはデータ読み込みスレッドの中でパケットデータ処理の狭間を縫って当該ユニットに出力データが送信されるため、データ出力の同時性は著しく損なわれる事があります。

【23】汎用デジタル現在値入力（４ビット）

書式	int __stdcall Fcad416_In_Aux(WORD board_no , BYTE *indata);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） indata:汎用デジタル現在値入力（４ビット）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットから、汎用デジタル現在値を取得する。但し サンプリング中には、サンプリングデータ読み込みスレッドでのステータス確認結果と共に送られてくる入力データが返されるため、この関数発行のタイミングではユニットとの通信は行わない。

使用例（Ｃ＋の場合）

```
if(( retc = Fcad416_In_Aux( SelectedBoard , &InputData)) != 0){
        m_messagebox.SetWindowText("汎用入力失敗");
        return;
}
else{
        itoa(InputData , buf , 16 );
        for(retc = 0 ; retc < (int)strlen(buf) ; retc++){
                OneChar = buf[retc];
                buf[retc] = toupper(OneChar);
        }
        m_generalinputvalue.SetWindowText( buf );
        m_messagebox.SetWindowText("汎用入力取得");
}
```

この例では、SelectedBoard 変数で指定されるユニットから汎用入力を読み込むコードが記述されています。汎用入力の場合も、サンプリング中にはデータ取得スレッドの狭間でユニットからの入力データを取り込むためデータの同時性は著しく損なわれることがあります。

【24】Dll バージョン取得

書式	int __stdcall Fcad416_Get_LibVer(int *ver);
引数	ver:Dll バージョン番号を返すための変数
戻り値	常に正常終了(0)
機能・動作	Dll のバージョン番号を取得する。

使用例（Ｃ＋の場合）

```
if ((retc = Fcad416_Get_LibVer( &libver)) != 0)
        m_messagebox.SetWindowText("GetLibVer 失敗");
else{
        itoa(libver, libstr, 16);
        m_messagebox.SetWindowText("GetLibVer 成功");
        m_LibVer.SetWindowText( libstr );
}
```

ここでは、Ｄ１１のバージョンを取得しテキストボックスへ表示するというコーディング例を示しています。Ｄ１１バージョンは１６ビットの整数で、現状０ｘ０１０５となっています。

【25】ファームウエアバージョン取得

書式	int __stdcall Fcad416_Get_FirmVer(WORD board_no , int *ver);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） ver:ファームウエアのバージョン番号を返すための変数
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットのファームウエアバージョンを取得する。

使用例（Ｃ＋の場合）

```
if ((retc = Fcad416_Get_FirmVer( SelectedBoard , &firmver)) != 0)
        m_messagebox.SetWindowText("GetFirmVer 失敗");
else{
        itoa(firmver, firmstr, 16);
        m_messagebox.SetWindowText("GetFirmVer 成功");
        m_FirmVer.SetWindowText( firmstr );
}
```

ここでは、SelectedBoard 変数で指定されるユニットからファームウェアバージョンを取得するコードが記述されています。ファームウェアバージョンは１６ビットの整数で、現状０ｘ０１００となっています。

【26】プリトリガバッファのサイズを設定する関数

書式	int __stdcall Fcad416_Set_Memory(WORD board_no , int mem);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） mem:指定ユニットのプリトリガバッファサイズを指定する変数。総メモリサイズの１／８単位で０／８から７／８まで指定可能。値はそれぞれ0から７とし、メモリ容量はそれぞれ0M 語（プリトリガバッファ無し）から1M 語単位で7M 語まで指定可能
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	全てのユニットのプリトリガバッファサイズを設定する

使用例（Ｃ＋の場合）

```
if((retc = Fcad416_Set_Memory( board_no[loop], Memory_size[board_no[loop]])) != 0){
        m_messagebox.SetWindowText("PritriggermemorySize set fail");
        return;
}
else
        m_messagebox.SetWindowText("PritriggerMemorySize set success");
```

この使用例では、board_no[loop]変数で示されるユニットに対して、プリトリガバッファメモリを割り当てるということをしています。しかしこの段階では、Ｆｃａｄ４１６側にメモリの割り当てを指示するだけに留まり、パソコン側のメモリ確保は、この関数の引数を記録しておき、ハンドラー内部でサンプリング開始時に行われます。

【27】プリトリガバッファのデータを読み出す関数

書式	int __stdcall Fcad416_Read_DirectRam(WORD board_no , DWORD no_data, DWORD data_pos, WORD *bufptr, DWORD buf_size);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） no_data:チャンネル当たりの読み出しデータ点数 data_pos:読み出しデータ先頭からのオフセット bufptr:ユーザーバッファ先頭アドレス buf_size:ユーザーバッファサイズ（バイト単位）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットのプリトリガデータを読み出す。読み出すことのできる最大データ数は Fcad416_Get_Status 関数によって得ることができる。また、実際のデータは、サンプリング終了時Ｄ１１バッファに全て取り込まれている。

使用例（Ｃ＋の場合）

```
if ((retc = Fcad416_Read_DirectRam( SelectedBoard, pre_sampled, disp_buffer, precount * 2)) < 0){
        m_messagebox.SetWindowText("Read pritrigger buffer failed");
        free( disp_buffer );
        disp_buffer = NULL;
        return;
}
else
        m_messagebox.SetWindowText( "Read pritrigger buffer success" );
```

この例では、SelectedBoard 変数で指定されるユニットからのプリトリガバッファメモリの内容を disp_buffer エリアに読み込むというコーディングになっています。実際のデータはサンプリング終了メッセージが発行される以前にハンドラー内部のメモリに読み込まれています。また、プリトリガバッファメモリは無限ループとなっているため、プリトリガサンプリングデータ数が準備したバッファエリアを越えると所謂リングバッファとなりますが、この場合であっても、パソコン内部に確保したバッファの内容はアドレス先頭に最古参のデータ、アドレス末尾に最新のデータというならびに変換されて保存されています。

【28】ユーザーへのメッセージ送信を指定する関数

書式	int __stdcall Fcad416_Plus_Message（WORD board_no ,DWORD Bit_Of_Message);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） Bit_Of_Message:　該当ビットをセットすることでメッセージ送信を要求する。 ビット０：最初のトリガ検出タイミングでメッセージを送信 ビット１：Fcad416_Set_TransferSize 関数で指定したデータ数を読み込んだ時メッセージを送信（リングバッファ一巡時には、ビット２で定義されるメッセージも発信されるが、上記指定データ数の切れ目と一致するかどうかはユーザーアプリケーションの設定によるため、他の場合と同様データ数の切れ目でメッセージが発信される。また、リングバッファ一巡時には Get_Status 関数のステータスに該当ビットがセットされているため、処理を場合わけする必要がある） ビット２：無限サンプリング時、リングバッファを一巡する毎にメッセージを送信 ビット３：サービスリクエスト受信時にメッセージを送信 ビット４：強制停止完了通知メッセージを送信 ビット５～ビット３１：予約ビット 尚、サンプリング終了とデータロストエラー発生及びケーブル脱落エラーについては、本関数の指定とは無関係にメッセージ送信が行われる。尚、返されるメッセージ番号（メッセージの第二パラメータ）は次の通りである。 サンプリング終了：０ データロストエラー発生：１ ケーブル脱落エラー：２　　以上３パターンはこの関数での指定有無に関係なく送信される トリガ検出：３ パケット転送終了：４ リングバッファ一巡：５ サービスリクエスト受信：６ 強制停止完了通知：７ また、ユニットＩＤはメッセージの第一パラメータとしてアプリケーション側に返される。この情報を利用して複数存在するユニットへのサービスリクエストを判別する事ができる。
戻り値	正常終了時：０ エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットとの間で指定された動作が行われたときにメッセージを送信するかどうかを設定する。該当ビットをセットして本関数を呼び出すことで対応するメッセージ送信を許可する事ができる。ビット０とビット１、ビット３での要求については、有限サンプリング、無限サンプリングのいずれであっても要求があればメッセージを送信するものとする。但し、スレーブユニットにも対応するのはケーブル脱落エラー及びサービスリクエスト受信メッセージのみである。尚、パケット転送終了メッセージ、リングバッファ一巡メッセージ、サービスリクエスト受信メッセージについては、マスターユニット及び各スレーブユニットからもメッセージが送信されるが、その他のメッセージについてはマスターユニットからのみ送信される。また、全てのメッセージはユーザーアプリケーションウィンドーへ送信される。その際、アプリケーション側からはメッセージの第一パラメータがユニット番号になっている。

※　リングバッファ一巡メッセージについて

無限サンプリング動作時のデータ転送に関係するメッセージにはパケット転送終了メッセージと本メッセージとがある。パケット転送終了メッセージが、ユーザーから指定されたパケット転送データの転送が終了した時点でユーザーアプリケーションに渡されるのに対し、リングバッファ一巡メッセージは、無限サンプリング動作時、ユーザーから設定されたＤ１１バッファの最終オフセットまでデータが保存された後、同バッファの先頭にデータ保存アドレスが更新された事を示すメッセージである。即ち、パケット転送終了メッセージが常にサンプリング動作と同期しているのに較べ、リングバッファ一巡メッセージについては、そのような同期関係が成立するかどうかは定かではない。そのため、本ドライバー及びサンプルプログラムでは、無限サンプリング動作を行わせるにあたり、常にパケット転送終了メッセージを使用し、そのメッセージ受信時に、ステータスフラグを確認する事で、ラップラウンドとの関係

を調べデータを処理する方式を採用している。その結果、パケット転送データ数、Ｄ１１バッファ長の組み合わせについての制限がかなり緩和されている。

使用例（Ｃ＋の場合）

```
        if (retc = Fcad416_Plus_Message( board_no[loop], MessageRequest[board_no[loop]])){
                m_messagebox.SetWindowText( "Set MessageRequest  失敗" );
                return;
        }
        else
                m_messagebox.SetWindowText( "Set MessageRequest  成功" );
```

この例では、board_no[loop]変数で指定されるユニットの動作に伴うメッセージ送信サービスを設定しています。これに対するメッセージハンドラーの例は以下のようになります。（詳細は省略）

```
LRESULT CVC6PlusDlg::OnDllMsg( WPARAM wParam, LPARAM lParam )
{       // wParam はユニット番号（ロータリースイッチ番号）なので複数ユニットの場合は以下の処理の前に
        // まずこの変数で場合わけをする必要がある
    switch( lParam ){                           // メッセージ番号で分ける
    case 0:                                     // MSG_SAMP_END
        do{
                retc = Fcad416_Get_Status( SelectedBoard, &Post_Sampled, &Pre_Sampled, status);
                if (retc < 0)
                        goto Err_Exit;                      // エラーが発生したら後の処理は飛ばす
        }while(retc != 0);
        // Get_Status 関数の戻り値は、サンプリングスレッドが停止するまで０にならないので
        // ここで時間合わせをする
        Fcad416_Clear_Flags(SelectedBoard, 0xff );
        m_messagebox.SetWindowText( "指定サンプリング終了" );
Err_Exit:
        Fcad416_Stop_Samp( );                       // スタートビットのみを抑止する
        break;
case 1:                                 // MSG_LOST_ERROR
        Fcad416_Stop_Samp( );
        OnButton10();                               // エラーが発生したらステータスをチェックする
        sprintf( buf, "%d", data_count);
        m_debug.SetWindowText( buf );
        m_messagebox.SetWindowText( "エラーフラグ" );
        break;
case 2:                                             // ケーブル脱落をレポートする
        Fcad416_Stop_Samp( );
        sprintf( buf, "ID%d のケーブル脱落",wParam);
        m_messagebox.SetWindowText( buf );
        break;
case 3:                                 // MSG_TRIGGER
        m_messagebox.SetWindowText( "Trigger Detected" );
        break;
case 4:                                 // 転送バッファサイズデータ取得メッセージ
        OnButton10();                               // ステータスチェック
        sprintf( buf, "%d", data_count);
```

```
            m_debug.SetWindowText( buf );
            break;
    case 5:                                 // リングバッファ一巡メッセージ
            m_messagebox.SetWindowText( "RingBuffer wrap round" );
            break;
    case 6:                                 // サービスリクエスト受信メッセージ
            m_messagebox.SetWindowText( "Service Request Received" );
            Fcad416_Clear_ServiceReq( SelectedBoard );
            break;
    default:
            break;
    }
}
```

これはあくまでもサンプルとして単純化した例ですので、実際にはかなり詳細なコーディングが必要となります。

【29】キャリブレーションデータを書き込む関数

書式	int __stdcall Fcad416_Set_Calibration(WORD board_no , BYTE memory_address , BYTE calibrate_data , int Mode);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） memory_address:　キャリブレーションデータを書き込むアドレス、内容は、電圧レンジ、入力形式、チャンネル番号及び調整モード指定ビットから生成される８ビットの数値である。（３－10 項参照） calibrate_data:　調整用 DAC に出力されるデータ。００H（0V）から FFH(＋5V)までの間の値をとる（３－10 項の詳細説明を参照の事） Mode:　与えられた情報をオフセットのものとして扱うかゲインのものとして扱うかを指定する数値 ０：　オフセットとして処理する １：　ゲインとして処理する
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに対して memory_address と Mode で指定されたアドレスにキャリブレーションデータを書き込む。

使用例（Ｃ＋の場合）

```
OffsetChangeVariables.OffsetTags.Offset00 = 0;
calibrate_data = m_spinofs00.GetPos();                    // 新しいオフセット設定値を取得して
memory_address = 0x80 + Dif_Single + AnalogSpan;
Fcad416_Set_Calibration( SelectedBoard , memory_address , calibrate_data , OFFSET);
```

この例は、実際の調整プログラムから引用したもので、SelectedBoard 変数で指定されたユニットの指定チャンネルに新しいオフセットデータを設定する部分のコーディングです。
メモリアドレスについては以下のように定義されています。（３－10 項より）

アドレス割付	ビット割付	ビット番号
調整モード／通常モード	１＝調整／０＝通常	Ｂ７
差動／シングルエンド	１＝差動／０＝シングルエンド	Ｂ６
入力レンジ	００＝±１０．２４Ｖ ０１＝±５．１２Ｖ １０＝±２．５６Ｖ １１＝±１．２８Ｖ	Ｂ５ Ｂ４
対象チャンネル番号	０ＨからＦＨ （ゲイン調整レジスタでは 無意味）	Ｂ３ Ｂ２ Ｂ１ Ｂ０

オフセット・ゲイン調整レジスタのメモリアドレッシング

【30】キャリブレーションデータを取得する関数

書式	int __stdcall Fcad416_Get_Calibration(WORD board_no , BYTE memory_address , BYTE *calibrate_data , int Mode);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） memory_address:　キャリブレーションデータを取得するアドレス、内容は、電圧レンジ、入力形式、チャンネル番号及び調整モード指定ビットから生成される８ビットの数値である。（３－10 項参照） calibrate_data:　調整用 DAC に出力されるデータが返される変数 Mode:　与えられた情報をオフセットのものとして扱うかゲインのものとして扱うかを指定する数値 ０：　オフセットとして処理する １：　ゲインとして処理する
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットから memory_address と Mode で指定されたアドレスにセットされたキャリブレーションデータを読み取る。

使用例（Ｃ＋の場合）

```
memory_address = 0x80 + Dif_Single + AnalogSpan;      // ビット７をハイにするのは、調整モード
// Dif_Single は、シングルエンド入力と差動入力を切り分ける数値００Ｈならシングル４０Ｈなら差動入力
// AnalogSpan は、入力レンジを切り替える数値、００Ｈから３０Ｈまで
// これらにチャンネル番号を下位４ビットとして加算してアドレスが決定される
Fcad416_Get_Calibration( board_no[0] , memory_address , &calibrate_data , OFFSET);
m_spinofs00.SetPos(calibrate_data);
```

こちらは、オフセット・ゲイン定数をユニットへ設定するための関数使用方法です。メモリアドレスなどについては３－10 項、前頁等を参照して下さい。

【31】EEPROM とのデータ通信を行う関数

書式	int __stdcall Fcad416_Handring_Eeprom (WORD board_no , int Mode);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） Mode:　EEPROM からのデータ読み出しか EEPROM へのデータ書き込みかを指定する数値 ０：　データ書き込みとして処理する １：　データ読み出しとして処理する
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに対してキャリブレーションデータを保持している EEPROM の内容を処理する関数。ユニットオープン時には EEPROM の内容を FPGA 内部に読み出しているが、この関数を Mode=1 として実行する事により、任意のタイミングでデータ読み出しができる。 また Mode=0 として実行する事で、任意のタイミングで FPGA 内部のデータ（キャリブレーション）を EEPROM に書き込む事ができる。

使用例（Ｃ＋の場合）

```
Fcad416_Handring_Eeprom( SelectedBoard , Mode);          // EEPROM 内容を読み込む
```

この例では、SelectedBoard 変数で指定されるユニットに対して EEPROM（オフセット定数やゲイン定数を格納している読み出しメモリ）からのデータ読み出しを Mode 変数を１とすることで指示をしています。通常の使用方法ではこの関数を使用する機会は恐らくありませんが、ユニット再調整などの局面では重要な関数です。

【32】シリアル番号を取得する関数

書式	Int __stdcall Fcad416_Get_SerialNumber(WORD board_no , BYTE *SerialNumber , int BufSize);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） SerialNumber: シリアル番号を格納するバイト配列の先頭を示すポインタ、シリアル番号そのものは１３バイトの文字列である BufSize：SerialNumber 配列のサイズ（最低１４バイトは必要、機能・動作の項参照）
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットに対して、そのシリアル番号を要求する関数。ユニットからは１３バイトのアスキーコードが送られてくる。ユーザーバッファへコピーする際、文字列のデリミタであるヌルキャラクタが追加されるので、アプリケーションサイドとしては１４バイトのエリアを用意しておく必要がある。

使用法（Ｃ＋の場合）

```
unsigned char SerialNumber[14];
int          loop;
CString str;

Fcad416_Get_SerialNumber( board_no[0] , SerialNumber , 14);
str = "";
for(loop = 0 ; loop < 14 ; loop++)
        str += SerialNumber[loop];
m_GetSerialNumber.SetWindowText( str );
```

これは、ユニットそれぞれに対して与えられているシリアル番号を読み込むための関数です。現状この変数に対しては何も機能を割り当てておりませんが、将来的には、ユニットそれぞれを判別することも可能なように予め埋め込まれている機能です。

【33】プリトリガバッファの内容を読み出す関数

書式	int __stdcall Fcad416_Read_PriTrig_Buff (WORD board_no , DWORD no_data, WORD *bufptr, DWORD buf_size);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） no_data:チャンネル当たりの読み出しデータ点数 bufptr:ユーザーバッファ先頭アドレス buf_size:ユーザーバッファサイズ（バイト単位）
戻り値	正常終了時: 正の値（チャンネル当り有効データ数を示す値：非０） エラー発生時:負の値
機能・動作	ロータリースイッチ番号で指定したユニットのトリガ直前のプリトリガデータを指定したデータ数読み出す。正常な場合の戻り値は正の数値であり、アプリケーションバッファ内でのチャンネルあたり有効データ数を示す値となる。また、バッファ内での有効データは、先頭から書き込まれており、プリトリガバッファ内に要求データ数未満のデータしかなかった場合、アプリケーションバッファの残りエリアには０が書き込まれる。

使用例（Ｃ＋の場合）

```
if ((retc = Fcad416_Read_PriTrig_Buff ( SelectedBoard, pre_sampled, disp_buffer, precount * 2)) < 0){
        m_messagebox.SetWindowText("Read pritrigger buffer failed");
        free( disp_buffer );
        disp_buffer = NULL;
        return;
}
else{
        m_messagebox.SetWindowText( "Read pritrigger buffer success" );
        data_size = no_ch * retc;       // アプリケーションバッファ内の有効データ数を求める
}
```

この例では、SelectedBoard 変数で指定されるユニットからのプリトリガバッファメモリの内容を disp_buffer エリアに読み込むというコーディングになっています。実際のデータはサンプリング終了メッセージが発行される以前にハンドラー内部のメモリに読み込まれています。戻り値は、正常終了時には正の値となっていますが、その数値は、アプリケーションバッファに格納されたプリトリガデータの(チャンネル当たり)データ数を示しています。更に、戻り値が指定データ数に満たなかった場合、アプリケーションバッファの後半未使用領域は０フィルされています。USB ユニット内のプリトリガバッファメモリは無限ループとなっているため､プリトリガサンプリングデータ数が準備したバッファエリアを越えると所謂リングバッファとなりますが、この場合であっても、パソコン内部に確保したバッファの内容はアドレス先頭に最古参のデータ、アドレス末尾に最新のデータというならびに変換されて保存されています。

４－６．エラーコード一覧

戻り値	エラーの意味	備考
－１	オープンしていないユニットを使用しようとした	Fcad416_Close_Driver 以外の各関数
－２	コマンドパケットのサイズが不正	Fcad416_Get_MemoryAddress
－３	リストデバイス関数でエラーが発生	Fcad416_Open_Driver
－４	指定した番号のボードがない	各関数
－５	シングルモードと差動モードが混在している	Fcad416_Start_Samp
－６	まだスタートしていないボードを使用	各関数
－７	サンプリング中なのでスタートできない	Fcad416_Start_Samp
－８	検出可能なボード枚数を超えたボードが存在する	Fcad416_Open_Driver
－９	ロータリースイッチ番号が範囲外	各関数
－１０	ＦＣＡＤ４１６－ＤＳＵＢが存在しない	Fcad416_Open_Driver
－１１	ロータリースイッチの値が重複している	同上
－１２	サンプリングチャンネル数がユニットによって異なる	Fcad416_Start_Samp
－１３	アナログ入力チャンネル数が不正	Fcad416_Set_SampCh
－１４	入力レンジ設定異常	同上
－１５	データコード不正	Fcad416_Set_InpMode
－１６	入力モード不正	同上
－１７	ID=0 のユニットがない	Fcad416_Open_Driver
－１８	リングバッファサイズが０でプリトリガサンプリングを指定	Fcad416_Start_Samp
－１９	トリガモード指定が異常	Fcad416_Set_Trigger
－２０	トリガーソース指定が異常	同上
－２１	トリガタイプが異常	同上
－２２	トリガ１とトリガ２の関係が逆転（トリガ１＞トリガ２）	同上
－２３	クロックソース指定が異常	Fcad416_Set_Clock
－２４	クロックセット指定が異常	同上
－２５	クロック周期指定で指定値が異常	同上
－２６	スレーブユニットに対して周期指定を行った	同上
－２７	スキャン周期指定が仕様範囲外	Fcad416_Set_ScanSpeed
－２８	ストップトリガをデジタルトリガ以外のトリガと組み合わせた	Fcad416_Set_Trigger
－２９	サンプリングパラメータ不足	Fcad416_Start_Samp
－３０	ポストサンプリング数が０と指示された	同上
－３１	非リングバッファモードで Stop_Samp_Loop 関数を指定	Fcad416_Stop_Samp_Loop
－３２	データ転送サイズが指定範囲外	Fcad416_Set_TransferSize
－３３	Ｄｌｌバッファからのリード開始位置指定が誤り	Fcad416_Read_DllData
－３４	Ｄｌｌバッファからのリードでバッファエリア過少	同上
－３５	ReadDirectRam 関数でバッファエリアが過少	Fcad416_Read_DirectRam
－３６	プリトリガバッファサイズ指定が異常	Fcad416_Set_Memory
－３７	ボード ID 不良	Fcad416_Start_Driver
－３８	メモリ開放に失敗した	Fcad416_Stop_Driver
－３９	サービスリクエスト信号をクリアできない（サンプリング中）	Fcad416_Clear_ServiceReq
－４０	サービスリクエストの極性指定（立下りか立ち上がり）が異常	Fcad416_Set_ServiceReq_Pol
－４１	スキャン周期設定用の分周比指定が使用可能範囲外	Fcad416_Set_ScanSpeed
－４２	トリガ条件設定前にサンプリングチャンネル情報をセットした	Fcad416_Set_SampCh
－４３	マルチメディアタイマーが使用できなかった	Fcad416_Start_Driver
－４４	既に停止している状態で、更に停止コマンドが発行された	Fcad416_Stop_Driver
－４５	メモリ確保ができなかった	各関数

ベンダーリクエスト内容説明書の使用・適用についての注意事項

（1）　本説明書は、ＦＣＡＤ４１６及びＦＣＡＤ４１６－ＤＳＵＢに対して適用されるベンダーリクエストコードの解説を行い、併せてlinux等での使用方法について手引きとする事を目的として作成されているものです。

（2）　本説明書を元としてＦＣＡＤ４１６或いはＦＣＡＤ４１６－ＤＳＵＢが組み込まれたシステムが運用対象・方法・場所・環境等によって、故障・誤動作等が生じた場合に起こり得る、身体・生命・財産等に対する損害の回避処置は同システムの設計・製作に別途付加・反映させて下さい。同製品及び本説明書にかかるベンダーリクエストコード自体には、前述の機能はなく、従って弊社では本製品が組み込まれたシステムの運用により発生した故障・誤動作・事故に起因する身体・生命・財産等の損害に対する責任は負えません。これは本製品の故障・誤動作が原因となった場合も含み、理由の如何を問いません。

（3）　本説明書は本製品利用の方法を示す例であり、現在未発見のバグ存在の可能性も含めて、運用結果についての責任は一切負えません。これらのソフトウェアには自身が組み込まれたシステムに故障・誤動作・事故等が生じた場合に起こり得る身体・生命・財産等に対する損害の回避機能はありません。御利用の場合は同システムの設計・製作で配慮・付加・反映させて下さい。

（4）　本説明書では、インターフェースの例としてlinux上で動作するlibusbというライブラリを想定しています。このライブラリを使用してアプリケーションを作成するのであれば、当然戻り値としてエラーコード等も必要ですが、その情報については本説明書の守備範囲を超えるものであるため一部を除き記載していません。

４－７．ベンダーリクエスト詳細説明

４－７－１　オープンユニット　（リクエストコード＝0x21）

本ユニットを使用する際には最初にこのリクエストコードを発行しなければなりません。このリクエストを発行することによって、ユニット内部ではさまざまなリクエストに対応する準備が完了します。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x21	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Open_Unit, 0, 0, ctrl_buf, 0, 0);
if (r < 0)
        return r;                    // Return with error
```

４－７－２　リセットユニット　（リクエストコード＝0x22）

本ユニットをあらかじめ定められた初期状態に設定するリクエストコードです。このリクエストを発行することによってユニットの初期化が行われます。（但し汎用出力信号の状態は変化しません）またこのリクエストを発行するとユニットが正常な状態であれば 0x2F というユニット ID が返されてきます。また、何らかの条件により準備未完であれば 0x00 という値が返されてきます。例えば、USB 接続後内部電源が正常に立ち上がるまでには最大２０秒程度の時間が必要なため、その間このリクエストに対する戻り値は 0x00 になります。そこで、この条件を使用してアプリケーションとのインターロックをとることができます。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x22	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0001	リターン文字列長

例題（libusb 環境）

```
elasp_time = 0;
do{                  // １秒毎にリセット関数を発行して起動完了を待つ
        sleep( 1 );
        elasp_time++;
        r = libusb_control_transfer(devh, CTRL_IN, Reset_Unit, 0, 0, ad_status, 1, 0);
}while((ad_status[0] != 0x2f) && (elasp_time < 30));
if (ad_status[0] == 0x2f)          // 但し３０秒待ってもだめならキャンセルする
        return 0;
else
        return -1;
```

４－７－３　クローズユニット　（リクエストコード＝0x23）

本ユニットをクローズするリクエストコードです。同時に内部メモリの初期化なども行われます。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x23	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Close_Unit, 0, 0, ctrl_buf, 0, 0);
```

４－７－４　セットチャンネル　（リクエストコード＝0x24）

16 チャンネル存在するアナログチャンネルそれぞれについて、スキャン順序、及び入力レンジ（入力レンジは全てのアナログチャンネルで共通です）を設定するリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x24	リクエストコード
2	wValue(L)	1	--	下位バイト：スキャン順序：ｎ
3	wValue(H)	1	--	上位バイト：スキャン順序番目の入力ライン：scan_ch_order[i]
4	wIndex(L)	1	--	下位バイト：入力レンジ：range
5	wIndex(H)	1	0x00	上位バイト：未使用
6	wLength	2	0x0000	未使用

このパラメータの意味は、ｎ番目のスキャン順序に対して入力ライン scan_ch_order[n]と入力レンジ range を設定するという事になります。
scan_ch_order[i]の値の範囲はアナログ入力チャンネル番号の範囲となるため０から 15 までとなります。
また range 変数の値は

range	入力レンジ
0	±１０．２４Ｖ
1	±５．１２Ｖ
2	±２．５６Ｖ
3	±１．２８Ｖ

となります。

例えば、設定値を

設定順	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
スキャン順	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
入力ライン	12	11	10	9	8	7	6	5	4	3	2	1	0	1	2	3
入力レンジ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

とした場合、入力レンジは±１０．２４Ｖで最初の入力ラインはチャンネル１２となり以下チャンネル１１、・・・と続き１５番目の入力ラインはチャンネル３となることになります。その結果、この設定例では、アナログトリガ検出はチャンネル１２に対して行われることになります。

例題（libusb 環境）

```
for (i = 0; i < scan_channel; i++){
    r = libusb_control_transfer(devh, CTRL_OUT,Set_Channel , ¥
    (i + (scan_ch_order[i] << 8)), inputrange, (unsigned char *)ctrl_buf, 0, 0);
}
```

この例で、scan_channel=5、scan_ch_order[]={3,8,12,0,9};とすると

i	scan_ch_order[i]	range
0	3	0
1	8	0
2	12	0
3	0	0
4	9	0

というような設定を行うことができます。この設定によるサンプリングでは、チャンネル３->８->12->０->９の順番で入力チャンネルがスキャンされます。またこの場合アナログトリガとして使用されるチャンネルはチャンネル番号３となります。また入力レンジは±１０．２４Ｖとなっています。

４－７－５　セットオーダー　（リクエストコード＝0x25）

サンプリング開始に先立ち、スキャン先頭スキャン順番号（０固定）及びスキャン最終スキャン順番号を設定するリクエストコードです。本ユニットではほとんど全ての設定リクエストコードについて発行順序を問いませんが、唯一本リクエストコードと、７項で説明するトリガーセットリクエストコードの間のみトリガーセット->本リクエスト（つまり項目７->項目５の順番である事が必須）という順序関係が要求されます。これは、本ユニットの特徴であるサンプリング入力任意切替を実現するための仕様です。
しかし、ユニット内部ではこの順序関係を確認していないため、ドライバー側での注意が必要です。
この順序を守らずにサンプリングを開始した場合、その動作は不定となります。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x25	リクエストコード
2	wValue(L)	1	--	下位バイト：スキャン先頭スキャン順番号
3	wValue(H)	1	--	上位バイト：スキャン最終スキャン順番号
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
#define  SCAN_0   0                    // 先頭スキャン順番号は０固定である
channels[0] = 16;            // １６スキャンを行う場合
r = libusb_control_transfer(devh, CTRL_OUT,Set_Order , (SCAN_0 + (unsigned short)((channels[0] - 1) << 8)),
0, ctrl_buf, 0, 0);
```

４－７－６　セットインプットモード　（リクエストコード＝0x26）

アナログ入力モードを指定するリクエストコードです。本ユニットでは、サンプリングモードとして、シングルエンド入力と差動入力モードをサポートしており、このモード指定及び出力アナログコードの指定を本リクエストコードで行います。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x26	リクエストコード
2	wValue(L)	1	--	下位バイト：インプット指定コード
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

インプット指定コードのビット定義は下の表のようになっています。そこで値が 0x00 であれば、シングルエンド入力でバイナリコード指定、値が 0x42 であれば差動入力で２の補数コード指定というようになります。

ビット	各ビットの機能・意味	１の時	０の時
B7	未使用	未使用	未使用
B6	入力形態	差動	シングルエンド
B5	未使用	未使用	未使用
B4			
B3			
B2			
B1	ＡＤデータ・コード	２の補数	バイナリ
B0	未使用	未使用	未使用

ここで差動入力の場合、若干の制限事項があります。シングルエンド入力の場合、最大１６回のスキャンに対して、１６チャンネルある入力のうちどれを使用しても問題はありませんでしたが、差動入力の場合は、シングルエンド入力を組み合わせて差動入力とする関係上組み合わせることのできる入力ラインについて、以下のような関係が必要です。
入力チャンネルｎと組み合わせることができる入力チャンネルはｎ＋１である。つまり

差動入力チャンネル	第１シングルエンドチャンネル	第２シングルエンドチャンネル
チャンネル０	チャンネル０	チャンネル１
チャンネル１	チャンネル２	チャンネル３
チャンネル２	チャンネル４	チャンネル５
チャンネル３	チャンネル６	チャンネル７
チャンネル４	チャンネル８	チャンネル９
チャンネル５	チャンネル１０	チャンネル１１
チャンネル６	チャンネル１２	チャンネル１３
チャンネル７	チャンネル１４	チャンネル１５

という組み合わせしか使用できないことになります。(この表の差動入力チャンネル０から７について順番を入れ替える事、例えば差動入力チャンネル０に対してチャンネル１２とチャンネル１３を使用する等は勿論可能です)
　また、ＡＤデータ・コードについては最上位ビットを符号ビットとする２の補数つまり－最大値を８０００Ｈ、＋最大値を７ＦＦＦＨとする数値表記、又は－最大値を０、＋最大値を６５５３５とするバイナリ何れかを選択することが可能です。

例題（libusb 環境）

```
InpMode[0] = 0x40;                    // 差動入力、データコードはバイナリ
r = libusb_control_transfer(devh, CTRL_OUT,Set_Inpmode , (InpMode[0]), 0, ctrl_buf, 0, 0);
```

４－７－７　セットトリガー　（リクエストコード＝0x27）

トリガ条件を設定するリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x27	リクエストコード
2	wValue(L)	1	--	下位バイト：トリガモード
3	wValue(H)	1	--	上位バイト：アナログトリガスレッショルド（下）
4	wIndex(L)	1	--	下位バイト：アナログトリガスレッショルド（上）
5	wIndex(H)	1	0x00	上位バイト：未使用
6	wLength	2	0x0000	未使用

トリガモード変数の内容は以下のように定義されています。この表のビットパターンの組み合わせにより所要のトリガモードを指定することになります。例えば0x38 の場合、アナログ立ち上がりエッジトリガということになります。

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7	即トリガ制御（ソフトトリガ）	許可	禁止
B6	外部トリガ入力信号制御	許可	禁止
B5	内部（アナログ）トリガ制御	許可	禁止
B4	トリガ極性選択	＋（↑）	－（↓）
B3	トリガ認識モード選択２	エッジ	レベル
B2	トリガ認識モード選択１	レンジ	レンジ以外
B1 /B0	未使用		

また、アナログトリガスレッショルドは、入力範囲を１／２５６に分割し、スレッショルド値として指定します。例えば、±１０．２４Ｖレンジでスレッショルド値＝１２８とするとスレッショルド電圧は０Ｖ、１２９とすると同様にして＋０．０４Ｖということになります。更に、レンジトリガ（Ｂ２＝１の場合）は両スレッショルド値の差異が２以上無いと（例えばスレッショルド値が１２８と１３０等）レンジ内（この例では１２９がレンジ内と判定される）という判定が行えないため、サンプリング開始を指示してもトリガ検出が行われず、見かけ上デッドロック状態のようになってしまうため注意が必要です。さまざまなトリガ入力とその機能説明については“３－６．トリガモードについて“を参照して下さい。

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT,Set_Trigger , (TrigValue[0] + (trig_level1 << 8)), trig_level2, ctrl_buf, 0, 0);
```

４―７－８　セットクロック　（リクエストコード＝0x28）

サンプリングクロック周期を指定するリクエストコードです。このリクエストコードは情報量が大きく標準の８バイトコマンドでは対処できないため、EP0 バッファをクロック分周比パラメータの受け渡しのために４バイト使用しています。ユニット内部では、分周比の正当性等については感知していないため、ドライバー側での管理が必要になります。(最低分周比を下回っていないか等)

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x28	リクエストコード
2	wValue(L)	1	--	下位バイト：クロックバリュー
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0004	パラメータ長

また、クロックバリュー変数の内容は以下のように定義されています。例えば、0x20 というコードを指定するとこの表より、内部 16.384MHz 使用という条件になります。また 0x10 というコードを指定すると外部立下りクロックを使用し、内部タイミングは 20MHz を使用するという条件になり、0x30 というコードでは外部立下りクロックを使用して内部タイミングは 16.384MHz を使用するという条件になります。外部クロックの場合も内部クロックの指定が可能ですが、これは、スキャンタイミングクロックは必ず内部クロックを使用するため、ここで指定する事でどちらの内部クロックでもスキャンタイミングクロックとして使用できるようにするためです。

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7	外部クロック源使用時の有効極性指定	↑（＋）	↓（－）
B6	未使用		
B5	内部クロック選択	16.384MHz	20MHz
B4	クロック源選択（外部／内部）	外部	内部
B3	未使用		
B2			
B1			
B0			

例題（libusb 環境）
この例題では、クロック分周比が PacerClock[0]変数にセットされ、クロックバリューが ClockValue[0]変数に設定されているという前提でコーディングされています。

```
for (loop = 0 ; loop < 4 ; loop++)
        ctrl_buf[loop] = (unsigned char)(PacerClock[0] >> (loop * 8));
r = libusb_control_transfer(devh, CTRL_OUT,Set_Clock , (ClockValue[0]), 0, ctrl_buf, 4, 0);
```

４－７－９　セットスキャンスピード　（リクエストコード＝0x29）

スキャン周期を指定するためのリクエストコードです。本ユニットは基本クロックとして、内部クロック（20MHz 及び16.384MHz の2種類）及び外部クロックを使用できますが、スキャンスピードの周期指定は常に指定された内部クロックが使用されます。そのため、内部クロックとして 16.384MHz を使用する際にはスキャンスピードの指定も61.035nS 単位で規定されることになります。また、外部クロックを使用する際にも、スキャンクロック周期の設定はクロック設定の項目で指定された内部クロックによって行われます。そのためこの際にもスキャンクロック周期として2種類ある内部クロックを選択使用できるように設計されています。
また使用しているＡＤ変換チップの最高速度は４ｕＳですので、基本クロックが20MHz の場合、分周比は８０から６５５３５までの範囲となり、基本クロックが16.384MHz の場合は６６から６５５３５までの範囲となります。
この数値の範囲については､ユニット内部でチェックを行っていないため､ハンドラー側での対応が必要になります。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x29	リクエストコード
2	wValue	2	--	スキャンスピード分周比
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT,Set_ScanSpeed , (ScanClockValue[0]), 0, ctrl_buf, 0, 0);
```

４－７－10　セットメモリー　（リクエストコード＝0x2a）

ユニットに搭載している８M語のメモリをＦＩＦＯメモリとプリトリガサンプリング時に使用するリングバッファメモリとに分割するリクエストコードです。プリトリガメモリのサイズを指定すると、８M語からの残りをＦＩＦＯメモリとするような働きを持っています。またＦＩＦＯメモリとしてのハーフフルサイズは５１２語にハードコードされています。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x2a	リクエストコード
2	wValue(L)	1	--	下位バイト：メモリコード
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

ここでメモリコードについては次のようなエンコード処理がなされています。

ビット	各ビットの機能・意味	定義
B7～B3	未使用	未使用
B2	メモリ分割指定	全体で８M語搭載しているメモリをＦＩＦＯメモリとプリトリガバッファメモリに分割する機能を実現している
B1		
B0		

ここで、B2 から B0 の値をｎと設定するとプリトリガバッファメモリのサイズがｎM語、FIFO メモリのサイズが（８－ｎ）M語となるようにユニット内部で設定が行われます。
例題（libusb 環境）

```
mem＝3;
```

```
r = libusb_control_transfer(devh, CTRL_OUT,Set_Memory , mem, 0, ctrl_buf, 0, 0);
```

このリクエストによりリングバッファメモリ３M語、FIFO メモリ５M語という設定が行われます。

４－７－11　サンプリングスタート　（リクエストコード = 0x2b）

サンプリングを開始或いは停止するリクエストコードです。このリクエストコードも８バイトのコマンドパケットではパラメータ（サンプリング数）を渡すことができないため、EP0 バッファを４バイト使用しています。また、ここで渡されているサンプリング数パラメータは全チャンネルではなくチャンネルあたりのサンプリング数になります。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x2b	リクエストコード
2	wValue(L)	1	--	下位バイト：サンプリングモード
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0004	パラメータ長

ここでサンプリングモードは次のようにコーディングされています。

ビット	各ビットの機能・意味	“1”の時	“0”の時
B7	サンプリング／クロックのみ	クロックのみ出力	サンプリング
B6	未使用		
B5			
B4	プリトリガモード選択	プリトリガ許可	プリトリガ禁止
B3	外部ストップ入力極性選択	＋（↑）	－（↓）
B2	外部ストップ入力信号制御	許可	禁止
B1	有限／無限モード選択	有限回数動作	無限回数動作
B0	スタート／ストップ制御	スタート（トリガ待ち）	強制ストップ

ですから、例えば 0x83 というコードを指定すると、サンプリングクロックのみ出力で有限回数動作のスタート（トリガ待ち状態を含む）ということになります。また、この場合、プリトリガモードは禁止、外部ストップ機能も禁止状態になっていることになります。

例題（libusb 環境）
この例題では、サンプリング数が local_post_samp 変数にセットされサンプリングモード変数が Startmode[0]変数にセットされている前提でコーディングがなされています。

```
for (loop = 0 ; loop < 4 ; loop++)
        ctrl_buf[loop] = (unsigned char)(local_post_samp >> (8 * loop));
r = libusb_control_transfer(devh, CTRL_OUT, Start_Samp, Startmode[0], 0, ctrl_buf, 4, 10);
```

４－７－12　ステータス取得　（リクエストコード＝0x2c）

本ユニットのステータス問い合わせを行うリクエストコードです。このリクエストコードにより、２バイトのステータス情報が得られます。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x2c	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0002	リターンデータ数

また得られる２バイトの情報は以下に示すものです。

基本ステータスコード

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7	END:連続サンプリング終了	終了済み	実行中
B6	EOS:各回サンプリングスキャン終了	終了済み	実行中
B5	未使用	未使用	未使用
B4	ORE:オーバーランエラーフラグ	発生済み	未発生
B3	LST:データロストエラーフラグ	発生済み	未発生
B2	FUL:Not Full	未満杯	満杯
B1	HLF:Not Half-Full	１／２未満	１／２以上
B0	EMP:Not Empty	データ有り	データ無し

それぞれのビットの意味は
END：連続サンプリングが終了した時にセットされるビットです。FIFO メモリ容量に対して十分余裕があるサンプリングであれば、このビットがセットされるまで、アプリケーション側ではデータの引取りを行う必要はありません。またこのビットは明示的にクリアする必要があります。
EOS：一回のサンプリングスキャンが終了するとセットされるビットです。低速サンプリングの場合、このビットがセットされた段階で一スキャン分のデータを引き取るタイミングとして使用すると有効ではないかと思われます。このビットについても明示的にクリアする必要があります。
ORE：オーバーランエラー（最高サンプリング速度：４μＳ/チャンネルを上回る速度でのサンプリング）が発生した場合セットされるフラグです。標準の Windows ハンドラーでは、ハンドラー内部でスキャン速度を管理しているためこのビットが発生する事はありませんが、ベンダーリクエストレベルではそのような管理が行われていないため、発生する可能性があります。そのような場合には、明示的にクリアする必要があります。
LST：FIFO メモリからのデータ引取りがサンプリングに対し間に合わず、FIFO メモリがオーバーフローした場合、このフラグがセットされます。その場合、サンプリングはその時点で停止してしまいますが、その時点までのサンプリングデータはメモリ内部に保存されています。このフラグも明示的なクリアが必要です。

FUL：FIFO メモリが満杯になるとクリアされるフラグです。このフラグは状態フラグですから、特にクリアなどを行う必要はありません。
HLF：FIFO メモリの中に５１２語（＝５１２データ）のデータが保存されている場合クリアされる状態フラグです。このフラグがクリアされている場合、無条件に５１２データの引取りが可能です。このフラグについても明示的なクリアは必要ありません。
EMP：FIFO メモリの中にデータが保存されていない場合クリアされ、１バイトでもデータが保存されていればセットされる状態フラグです。本ユニットはＵＳＢ経由で使用するため、現実的には殆ど問題はありませんが、このビットがセットされる最低条件はデータが１バイト存在するという事になるので、このフラグを使用してデータを引き取るという処理は理論上薦められる処理ではありません。このフラグも明示的なクリアは必要ありません。

拡張ステータスコード

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7	未使用		
B6			
B5	STP:強制停止モード完了	強制停止完了	強制停止未完
B4	RUP:ロールアップフラグ	ロールアップあり	ロールアップ無し
B3	BSY:入力範囲設定中フラグ	設定中	設定済み
B2	SVC:サービスリクエスト要求発生	発生	未発生
B1	TIM:各回サンプリングクロック先端	発生	未発生
B0	TGD:トリガ発生認識	発生	未発生

STP：サンプリングモード設定時（項目１１　サンプリングスタート参照）に、強制停止機能許可の条件を指定し、実際に外部から強制停止信号が印加された事を示す状態フラグです。このビットについても明示的なクリアは必要ありません。
RUP：プリトリガサンプリングの際、プリトリガバッファが一巡したかどうかを判定する状態ビットです。プリトリガバッファからデータを引き取る際に、このビットがセットされているかどうかで取り込めるデータの量、範囲が異なってくるため注意が必要です。このフラグについても明示的なクリアは必要ありません。
BSY：項目５のセットオーダーリクエストで説明したシーケンスに関係する状態フラグです。本リクエストレベルでは検知される可能性は皆無に近いものですが、このビットがセットされている状態は、トリガーセット後のスキャンオーダー設定後サンプリング開始が不可能である時間帯になります。（およそ４μＳ）実際には、この時間帯はFCAD416側のファームウェアで処理されているため、アプリケーション側で対応する必要はありません。
SVC：サービスリクエスト要求処理が完了した事を示すステータスフラグです。このフラグはラッチフラグですので、サービスリクエスト完了処理によって、要因をクリアすると共にこのビットも明示的にクリアする必要があります。要因検出レジスタはエッジトリガですので、この際クリアの順序についてはどちらが先であっても構いません。
TIM：毎回のスキャン開始タイミングでセットされるフラグです。主に低速サンプリングと同期を取るために使用するフラグです。
TGD：トリガ条件を検出した時点でセットされるフラグです。
これらフラグの内、基本ステータスコードのB3からB7、拡張ステータスコードのB0からB2については
使用するためには後述するフラグクリアリクエストにより使用後クリアする必要があります。

例題（libusb環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_Status, 0, 0, ctrl_buf, 2, 0);
```

４－７－13　サンプリング中のステータス取得　（リクエストコード＝0x2d）

本ユニットのステータス問い合わせ並びにサンプリング済データ数情報等を取得するリクエストコードです。
具体的には、ホスト側から汎用出力データを受信し、返信データとして
基本ステータスコード（１バイト）
拡張ステータスコード（１バイト）
汎用入力データ（１バイト）
サービスリクエスト入力（１バイト）
送付済みデータ数（４バイト：最下位バイトから最上位バイトの順）
残りデータ数（３バイト：最下位バイトから最上位バイトの順）
の１１バイトをこの順番で送信してきます。
コマンドパケットは以下のようになっています。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x2d	リクエストコード
2	wValue(L)	1	--	下位バイト：汎用出力データ
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	WLength	2	0x000b	リターンデータ数

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_Status_InSamp, (unsigned char)Aux_Out, 0, ctrl_buf, 11, 0);
Image_Status[localmatchnum][0] = ctrl_buf[0];        // 基本ステータス
Image_Status[localmatchnum][1] = ctrl_buf[1];        // 拡張ステータス
Image_Aux_in[localmatchnum] = ctrl_buf[2];  // 汎用入力データ
ServiceRequestStatus[localmatchnum] = ctrl_buf[3];   // サービスリクエスト入力をコピー
getdata_number = 0L;
for (loop = 0 ; loop < 4 ; loop++)
    getdata_number |= (int)((ctrl_buf[loop + 4]) << (loop * 8));
fifodata_number = 0L;
for (loop = 0 ; loop < 3 ; loop++)
    fifodata_number |= (int)((ctrl_buf[loop + 8]) << (loop * 8));
fifodata_number++;
fifodata_number /= 2;                                    // 読み出し可能な総データ数を取得する
```

４－７－14　ＦＩＦＯメモリからのデータ取得　（リクエストコード＝0x2e）

サンプリング中のステータス取得リクエストコード等により決定したアナログ変換データをバースト転送によりホスト側に取得するリクエストコードです。このリクエストコードも８バイトのコマンドパケットではパラメータ（取得データ数）を渡すことができないため、EP0 バッファを４バイト使用しています。ここで取得データ数は、総データ数を表しています。（チャンネルあたり１K語で16チャンネルであれば１６K語）

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x2e	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0004	パラメータ長

このコマンドパケットによってデータ送信を要求し、引き続いてバースト転送コマンドを発行します。

例題（libusb 環境）

```
        for (loop = 0 ; loop < 4 ; loop++)
                ctrl_buf[loop] = ((length >> (loop * 8)) && 0xff);
        r = libusb_control_transfer(devh, CTRL_OUT, Read_Fifo_Data, 0, 0, ctrl_buf, 4, 0);
        // このリクエストを発行することでユニット側にデータ転送の準備を行わせる。

        if ( r < 0)
                return r;
        r = libusb_bulk_transfer(devh, 0x86, AD_ReadBuf, length, &actual_length, 0);
        // このリクエストによって実際のデータ転送を行う
        // バースト転送ポートアドレスは0x86
        if (r < 0)
                return r;
        for (rr = 0 ; rr < sampnum ; rr++){
                for (r = 0 ; r < channels[localmatchnum] ; r++){
                        length = AD_ReadBuf[(rr * channels[localmatchnum] + r) * 2]¥
 + (AD_ReadBuf[(rr * channels[localmatchnum] + r) * 2 + 1] << 8);
                        printf("%05d,",length);    // エクセルで処理するためカンマ付十進数値として出力
                }
                printf("¥n");
        }
```

４－７－15　リングバッファメモリからのデータ取得　（リクエストコード＝0x2f）

プリトリガモード時有効となるリングバッファからのデータを取り込むリクエストコードです。基本的な動作は前項のＦＩＦＯメモリからのデータ取得と変わりません。ただ、ハードウェア構成上、ＦＩＦＯメモリからのデータ取得後でなければデータの取得ができません。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x2f	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0004	パラメータ長

このコマンドパケットを送信してバースト転送を開始させておき、引き続いてバースト転送コマンドを発行してデータを引き取ります。

例題（libusb 環境）

```
        for (loop = 0 ; loop < 4 ; loop++)
                ctrl_buf[loop] = ((length >> (loop * 8)) && 0xff);
        r = libusb_control_transfer(devh, CTRL_OUT, Read_Ram_Data, 0, 0, ctrl_buf, 4, 0);
        if ( r < 0)
                return r;
        r = libusb_bulk_transfer(devh, 0x86, AD_ReadBuf, length, &actual_length, 0);
        // バースト転送ポートアドレスは0x86
        if (r < 0)
                return r;
        for (rr = 0 ; rr < sampnum ; rr++){
                for (r = 0 ; r < channels[localmatchnum] ; r++){
                        length = AD_ReadBuf[(rr * channels[localmatchnum] + r) * 2]¥
+ (AD_ReadBuf[(rr * channels[localmatchnum] + r) * 2 + 1] << 8);
                        printf("%05d,",length);    // エクセルで処理するためカンマ付十進数値として出力
                }
                printf("¥n");
        }
```

４－７－16　マニュアルサンプリング開始　（リクエストコード＝0x30）

マニュアルサンプリングを実施させるリクエストコードです。何もパラメータを設定せずにこのリクエストコードを発行するとチャンネル０に対し１回のサンプリングを行います。またサンプリングチャンネル数を指定しておくとそのチャンネルグループに対して１回のサンプリングを行います。また、サンプリング結果のデータ引取りは、通常のサンプリングと同様、バースト転送によって行います。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x30	リクエストコード
2	wValue(L)	1	--	下位バイト：先頭チャンネル番号（＝０）
3	wValue(H)	1	--	上位バイト：最終チャンネル番号
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Get_One_Scan, (unsigned short)((last_channel << 8) | 0), ¥
0, ad_status, 0, 0);
        if (r < 0)
                return r;
        length = (last_channel + 1) * 2;
        r = libusb_bulk_transfer(devh, 0x86, ad_data, length, &actual_length, 0);
```

４－７－17　リングバッファ（プリトリガバッファ）メモリデータ数確認　（リクエストコード＝0x31）

プリトリガサンプリング時に使用するプリトリガバッファが保持しているデータ数を問い合わせるリクエストコードです。問い合わせに対する回答はEP0 バッファ経由で返されてきます。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x31	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0003	パラメータ長

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_Pre_Count, 0, 0, ad_status, 3, 0);
if (r < 0)
        return r;
pritriggerbufcount = 0L;
        // 初期値を０に！
for(loop = 0 ; loop < 3 ; loop++)
        pritriggerbufcount |= ctrl_buf[loop] << (loop * 8);
pritriggerbufcount /= 2;
        // ワード単位の数値に変換する
        // 全チャンネル分のデータ
```

４－７－18　サービスリクエスト極性設定　（リクエストコード ＝0x32）

本ユニットへのサービスリクエスト入力の極性を指定するリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x32	リクエストコード
2	wValue(L)	1	--	下位バイト：サービスリクエスト極性
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

サービスリクエスト極性は、次のように指定します。

ビット	各ビットの機能・意味	“1”の時	“0”の時
B7～B1	未使用		
B0	サービスリクエスト信号の有効極性指定	↑（+）	↓（－）

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Set_ServiceReq_Pol, ServiceReqPol, 0, ad_status, 0, 0);
```

４－７－19　サービスリクエスト確認　（リクエストコード ＝0x33）

設定したサービスリクエストが受信されたかどうかを確認するリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x33	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0001	リターンデータ数

サービスリクエストを受け付けている場合はリターンデータは“1”となります。またサービスリクエストが受け付けられていない場合、リターンデータは“0”となります。

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_Service_Req, 0, 0, ad_status, 1, 0);
```

４－７－20　サービスリクエストクリア　（リクエストコード＝0x34）

本体取扱説明書にも記載されているとおり、サービスリクエストは一旦受け付けられると、その処理は完了してしまい、新しいサービスリクエスト要求は、一旦終了したサービスリクエストフラグをクリアし再び要求を設定しなければ開始できません。
そこで、本リクエストコードを発行し終了したサービスリクエストフラグのクリアを行います。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x34	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Clear_Service_Req, 0, 0, ad_status, 0, 0);
```

４－７－21　汎用出力への出力　（リクエストコード＝0x35）

汎用出力ポートへアプリケーションからデータを出力するリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x35	リクエストコード
2	wValue(L)	1	--	下位バイト：汎用出力ポート出力値
3	wValue(H)	1	0x00	上位バイト：未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Out_Aux, (unsigned short)Out_Data, 0, ad_status, 0, 0);
```

４－７－22　汎用入力からの入力　（リクエストコード＝0x36）

汎用入力ポートからアプリケーションへデータを取り込むリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x36	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0001	リターンデータ数

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, In_Aux, 0, 0, ad_status, 1, 0);
in_data = ad_status[0];
return r;
```

４－７－23　ファームウェアバージョンの取得　（リクエストコード＝0x37）

本ユニットのファームウェアバージョン番号を取得するリクエストコードです。戻り値は２バイトの文字列で値は０ｘ０１００が返されます。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x37	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0002	リターンデータ数

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_Firm_Ver, 0, 0, ad_status, 2, 0);
```

４－７－24　ステータスクリア　（リクエストコード＝0x38）

本ユニット内部のステータスレジスタをクリアするリクエストコードです。ステータスレジスタは２バイト存在するためクリアコードも２バイト渡す必要があります。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x38	リクエストコード
2	wValue	2	--	上位バイト：拡張ステータスクリアビット 下位バイト：基本ステータスクリアビット
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

パラメータとして渡すクリアビットパターンは、以下に示すように構成されている必要があります。この中でセットされているビットに対応するレジスタのフラグビットがこのリクエストコードを発行することでクリアされます。

基本ステータスレジスタクリアビット

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7	END:連続サンプリング終了	クリア	無変化
B6	EOS:各回サンプリングスキャン終了	クリア	無変化
B5	IRE:割り込みオーバーランエラーフラグ	クリア	無変化
B4	ORE:オーバーランエラーフラグ	クリア	無変化
B3	LST:データロストエラーフラグ	クリア	無変化
B2	未使用	未使用	未使用
B1			
B0			

拡張ステータスレジスタクリアビット

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7～B3	未使用	未使用	未使用
B2	SVC:割り込み要求発生	クリア	無変化
B1	TIM:各回サンプリングクロック先端	クリア	無変化
B0	TGD:トリガ発生認識	クリア	無変化

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_OUT, Clear_Flags, (Ext_Flags << 8) |Base_Flags, 0, ad_status, 0, 0);
```

４－７－25　シリアル番号取得　（リクエストコード＝0x39）

ユニット固有のシリアル番号を取得するリクエストコードです。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x39	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x000d	リターンデータ長

１３バイト長のシリアル番号を返してくるリクエストコードです。注意点としては、文字列の最後にデリミタが付属していないため、アプリケーション側の要求に従ったデリミタを付加して制御を返す必要があります。

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_Serial_Code,0 , 0, ad_status, 13, 0);
```

※　ad_status 変数は最低でも１３バイト長である必要があります。（デリミタ分を含まず）

※※
※※　以下に示すリクエストコードは、調整プログラム等から必要に応じて呼び出されるものです。※※
※※

４－７－26　ロータリースイッチの値を取得　（リクエストコード＝0x40）

ユニットごとのロータリースイッチ番号を取得するリクエストコードです。アプリケーションレベルからユニットを識別するために、このロータリースイッチが準備されていますが、ハンドラー側から見るとこのスイッチの値ではなくＵＳＢバス上のアドレスがユニットを認識するために必要になります。このリクエストコードはこのアドレスとユニットとの関係をハンドラー内部で把握するために用意されているコードです。そのためこのコードがアプリケーションから呼び出されることは通常ありません。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x40	リクエストコード
2	wValue	2	0x0000	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0001	リターンデータ数

例題（libusb 環境）

```
r = libusb_control_transfer(devh, CTRL_IN, Get_RswValue, 0, 0, ad_status, 1, 0);
```

４－７－27　ゲイン・オフセットレジスタからの値取得　（リクエストコード＝0x41）

このリクエストコードは、通常のアプリケーションから呼び出されることはまずありません。このコードは、本ユニットの調整時に調整用のアプリケーションから指定したチャンネルのゲインレジスタ或いはオフセットレジスタの値を取得するために使用します。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0xc0	Host<-Device コマンド
1	bRequest	1	0x41	リクエストコード
2	wValue(L)	1	―	下位バイト：レジスタアドレス
3	wValue(H)	1	―	上位バイト：オフセットレジスタ／ゲインレジスタ指定
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0001	リターンデータ長

オフセットレジスタ／ゲインレジスタ指定バイトは、値が０であればオフセットレジスタを、０以外であればゲインレジスタを指定することになります。レジスタアドレスについては、以下の表に従います。

オフセット・ゲインレジスタアドレッシングレジスタ

ビット	各ビットの機能・意味	“１”の時	“０”の時
B7	調整／通常動作	調整モード	通常動作モード
B6	差動／シングルエンド	差動	シングルエンド
B5	入力レンジ指定	０：±１０．２４Ｖ、１：±５．１２Ｖ、２：±２．５６Ｖ、３：±１．２８Ｖ	無効
B4			
B3	入力チャンネル指定（０から１５までを０からＦにより指定）	調整対象チャンネル番号	無効
B2			
B1			
B0			

オフセット、ゲインレジスタからの値を読み込む際には、アドレッシングレジスタのB6からB0が制御対象レジスタを示す値になるように設定し、本リクエストコードを呼び出します。但し、ゲインレジスタは入力チャンネルによらず入力レンジ毎に一種類なのでB6からB4を指定するだけでかまいません。また、このリクエストではレジスタの値を読み出すだけなのでB7についてはセットしてはなりません。

例題（libusb環境）

```
RegAddress = 0x00;                    // シングルエンド、±１０．２４Ｖレンジ、チャンネル０レジスタ
RegSelect = 0x0100;                   // ゲインレジスタを指定
r = libusb_control_transfer(devh, CTRL_IN, Get_OfsGain, (RegSelect | RegAddress), 0, ad_status, 1, 0);
// シングルエンド、±１０．２４Ｖレンジのゲインレジスタの設定値を読み出す（値は ad_status[0]変数に格納されている）
```

４－７－28　ゲイン・オフセットレジスタへの値設定　（リクエストコード＝0x42）

このリクエストコードも、通常のアプリケーションから呼び出されることはまずありません。このコードは、本ユニットの調整時に調整用のアプリケーションから指定したチャンネルのゲインレジスタ或いはオフセットレジスタの値を設定するために使用します。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x42	リクエストコード
2	wValue(L)	1	--	下位バイト：レジスタアドレス
3	wValue(H)	1	--	上位バイト：オフセットレジスタ／ゲインレジスタ指定
4	wIndex(L)	1	--	下位バイト：レジスタ設定値
5	wIndex(H)	1	0x00	上位バイト：未使用
6	wLength	2	0x0000	未使用

オフセットレジスタ／ゲインレジスタ指定バイトは、値が０であればオフセットレジスタを、０以外であればゲインレジスタを指定することになります。レジスタアドレスについては、以下の表に従います。

オフセット・ゲインレジスタアドレッシングレジスタ

ビット	各ビットの機能・意味	“1”の時	“0”の時
B7	調整／通常動作	調整モード	通常動作モード
B6	差動／シングルエンド	差動	シングルエンド
B5 B4	入力レンジ指定	０：±１０．２４Ｖ、１：±５．１２Ｖ、２：±２．５６Ｖ、３：±１．２８Ｖ	無効
B3 B2 B1 B0	入力チャンネル指定（０から１５までを０からＦにより指定）	調整対象チャンネル番号（オフセットレジスタのみ適用、ゲインレジスタはチャンネルによらず共通）	無効

オフセット、ゲインレジスタへ値を書き込む際には、アドレッシングレジスタのB6からB0が制御対象レジスタを示す値になるように設定し、本リクエストコードを呼び出します。但し、ゲインレジスタは入力チャンネルによらず入力レンジ毎に一種類なのでB6からB4を指定するだけでかまいません。また、このリクエストではB7ビットをセットしても構いませんが、ファームウェア側で制御を行っているため、その必要はありません。

例題（libusb 環境）
```
RegAddress = 0x00;                  // シングルエンド、±１０．２４Ｖレンジ、チャンネル０レジスタ
RegSelect = 0x0000;                 // オフセットレジスタを指定
RegValue = 0x56;                    // レジスタ設定値は 56H
r = libusb_control_transfer(devh, CTRL_OUT, Set_OfsGain, (RegSelect | RegAddress), RegValue, ad_status, 0, 0);
// シングルエンド、±１０．２４Ｖレンジ、チャンネル０のオフセットレジスタに 0x56 という値をセットする。
```

４－７－29　ゲイン・オフセットレジスタのデータストア・リロード　（リクエストコード = 0x43）

オフセットレジスタとゲインレジスタの値をユニット内部の EEPROM との間でやり取りするリクエストコードです。通常､本ユニットでは電源投入時に EEPROM からオフセットレジスタとゲインレジスタの値を読み込み起動しますのでアプリケーションからこのリクエストコードを発行する必要性はありません。しかしながら調整プログラムにおいては、このようなリクエストコードが必要です。

Offset	Field	Size	Value	Description
0	bmRequestType	1	0x40	Host->Device コマンド
1	bRequest	1	0x43	リクエストコード
2	wValue(L)	1	--	モード
3	wValue(H)	1	0x00	未使用
4	wIndex	2	0x0000	未使用
6	wLength	2	0x0000	未使用

モード変数が０であれば、EEPROM へのデータストア、１であれば EEPROM からのデータリードを行います。
データハンドリングは常に一括処理ですので、一部変更であっても、すべてのデータが同じように扱われます。

例題（libusb 環境）
```
Mode = 1;                           // 変更されたオフセット・ゲインデータを EEPROM へ書き込む
r = libusb_control_transfer(devh, CTRL_OUT, ReadWriteEeprom, Mode, 0, ad_status, 0, 0);
```

４－８　改版履歴

Ｄｌｌバージョン履歴

改版日付	バージョン情報	改版履歴
初版	０１０１	初版リリース
2010/07/10	０１０２	外部クロック使用時のスキャン周期用内部クロック設定ミス（常に 20MHz を使用するようにコーディングされていた）
2010/09/14	０１０３	bool 変数処理の誤りなどを修正
2011/10/31	０１０５	サンプル機能追加及びハンドラー関数追加その他改良

第５章. 保守・その他

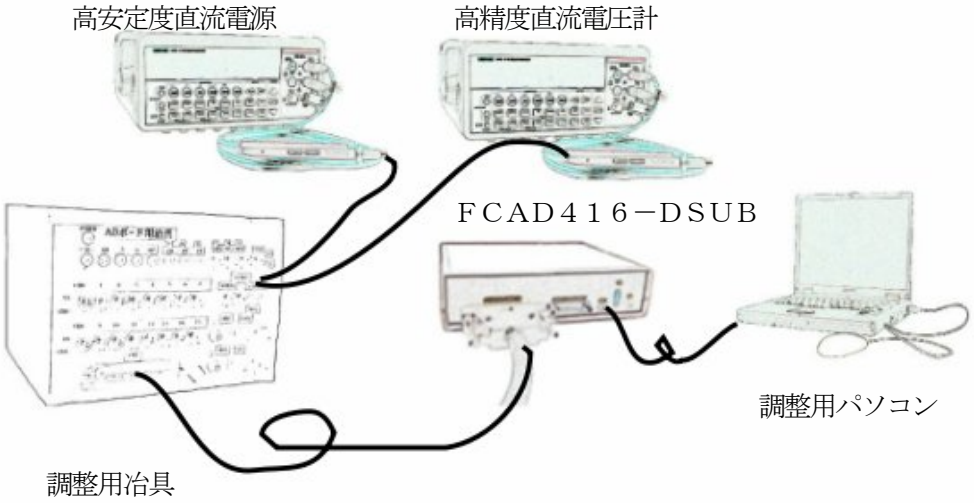
５－１. アナログ入力範囲の再調整

１－６項に示すような形で動作確認を行い、入力範囲やオフセットの変化が認められるときは再調整が必要です。アナログ回路は経年環境変化に対する保守を定期的に行うことが望ましく、夏冬の使用環境・周囲温度に差がある場合は季節単位、通年安定した使用環境の場合は１～２年に１度は校正することが理想的です。

再調整の方法・手順を以下に記しますが、被調整対象のＦＣＡＤ４１６－ＤＳＵＢの他に、高確度の基準電源と高精度の電圧計、各入力を個別に切り替える事ができるスイッチボックス及びWindow XP／Vistaがインストールされたパソコンを準備する必要があります。

５－１－１　機器間の接続

全体的な機器間の接続を」以下に示します。



オフセット・ゲイン調整ブロック図

５－１－２　暖機運転

一般的に言って、これらの測定器は使用前数時間に渡って通電しておかないと、所期の性能を発揮できません。ＦＣＡＤ４１６－ＤＳＵＢ自体も、それ程ではありませんがある程度のウォーミングアップが必要です。実際に必要なウォーミングアップ時間については御使用頂く各測定器の取扱説明書により適切な時間を設定するようにして下さい。（ＦＣＡＤ４１６－ＤＳＵＢ自身については一時間程度のウォーミングアップが必要です。）

５－１－３　調整作業の実際

ＦＣＡＤ４１６－ＤＳＵＢ用の調整プログラムは、１－４節で展開していただいたソフトウェアセットの中にあります。

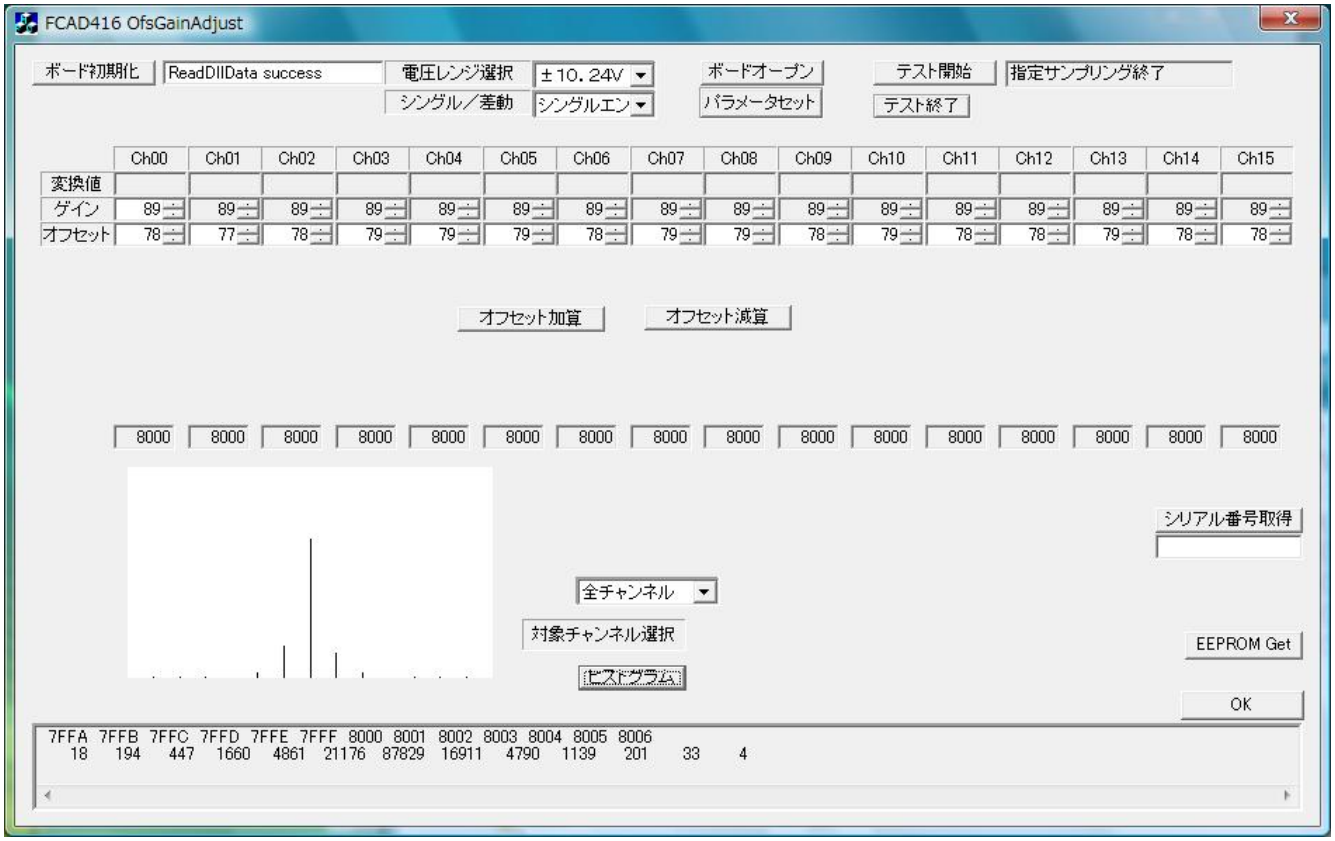
その場所は、“Flexcore¥Fcad416¥Tools¥OfsGainAdjust¥Release”フォルダです。エクスプローラでこのフォルダを開き、OfsGainAdjust.exeプログラムを実行します。（ダブルクリック或いは“右クリック－>開く”）

このプログラムは、入力１６チャンネル夫々についてサンプリングを行い、アナログ変換値の出現頻度をヒストグラムの形で出力するユーティリティです。これを調整用冶具及び直流電源と組み合わせ、入力がグランドの場合の値（オフセット）と入力がある値の場合の値（ゲイン）を調整するものです。またドロップダウンリストを操作することで、４種類ある入力レンジ並びにシングルエンド入力及び差動入力について逐次調整を行なう事ができます。

５－１－４　調整プログラム

弊社で実際に使用している調整プログラムの動作を以下に示します。操作ボタンは上の左から
“ボード初期化”
“ボードオープン”
“パラメータセット”
“テスト開始”
“テスト終了”
“オフセット加算”
“オフセット減算”
“ヒストグラム”
“シリアル番号取得”
“EEPROM　Get”
“OK”となっています。

プログラム操作の流れは次のようになります。
まずボード初期化ボタンでＦＣＡＤ４１６－ＤＳＵＢが存在する事を確認し、次に電圧レンジ選択ドロップダウンリストとシングル／差動入力選択ドロップダウンリストにより適切な入力モードを選定します。
そしてボードオープンボタンですぐ下のゲイン・オフセットパラメータボックスに実際の設定値を読み込み、パラメータセットボタンでサンプリング条件を設定します。
　ここで、調整用治具の入力切替スイッチをグランド側（オフセット調整時）或いは入力電圧側（ゲイン調整時）に切り替えます。また差動入力ゲイン調整時には偶数チャンネルを入力電圧側に、奇数チャンネル側をグランド側にします。これ以外の組み合わせを行なってサンプリングを開始すると場合によってはプログラムが破綻します。
ここでテスト開始ボタンを押すと、ボタン表示が“In Sampling”に変わり、すぐに“テスト開始”表示に戻ります。この時点で画面下部のヒストグラムボタンを押すと中央部の小窓に１６チャンネル夫々のヒストグラム中心値が表示され左下部のウィンドーには全チャンネル合計でのヒストグラムがグラフ化されて表示されます。また、ヒストグラムボタンのすぐ上の対象チャンネル選択ドロップダウンリストから適切なチャンネルを選択して再度ヒストグラムボタンを押すと該当チャンネルのヒストグラム表示が左下部のウィンドーに再描画されます。



オフセット調整は、チャンネル毎に微調整ができます。画面上部のオフセット変換値をチャンネル毎に増減するか又は画面中央部のオフセット加算、オフセット減算ボタンで１６チャンネル一括でオフセット変換値を増減すると該当チャンネルのオフセット状態におけるサンプリング値が増減します。その方向はオフセット変換値が増加すると、アナログ変換値も増加するような関係になっています。

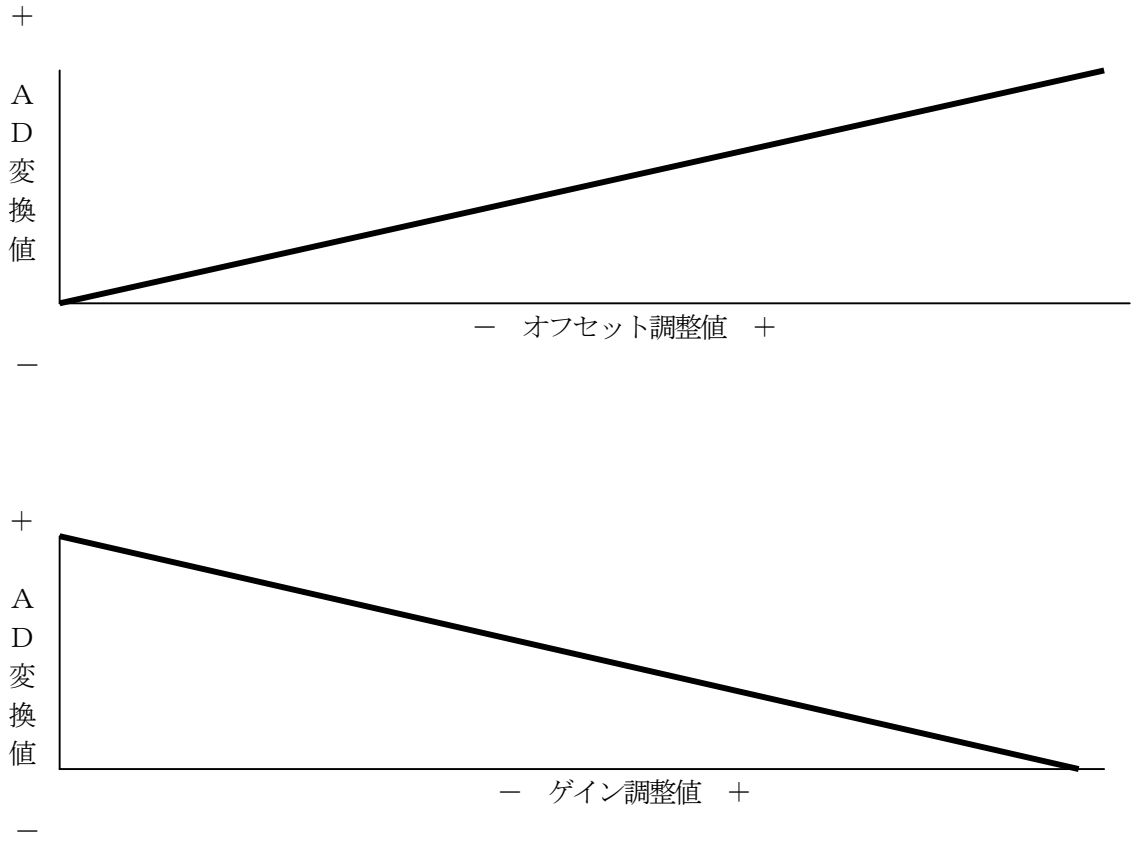
一方ゲイン調整は画面上部のゲイン変換値（チャンネル０分）を増減する事で行ないます。この時便宜的にチャンネル１から１５のゲイン変換値も同時に変化します。（表示上このようになっているだけで実際には、３－１０項ゲイン調整レジスタで説明したように１６チャンネル全てが一つのレジスタによってゲイン調整されます。）

またゲイン調整の方向は、オフセット調整の場合と異なり、ゲイン変換値が増加すると実際のゲインは減少し、入力電圧に対するアナログ変換値はより小さな値になってゆきます。

実際の調整では、予め弊社にて調整が行なわれているのであまり大きな変動は発生しないと思いますが、厳密にはオフセットが変化するとゲインもそれに合わせて再調整する必要があります。

調整全体の流れとしては、まずオフセット調整を行い、オフセットが合わせ込まれたところで、入力電圧によるゲイン調整を行い、再度オフセット調整を行ないます。このループを数回まわす事で調整を完了させる事ができます。そしてある入力レンジ・入力モード（シングル入力又は差動入力）の組み合わせについて調整が完了したと判断した時点で、右上にあるテスト終了ボタンを押します。すると数秒ウィンドー全体が動かせなくなり（所謂ハングアップ状態のような感じ）その後マウスなどで自由に動かす事ができるようになります。この数秒の時間の中でＦＣＡＤ４１６－ＤＳＵＢ内部では、調整を行なった結果のレジスタ値をユニット内部のEEPROMに書き込み次のユニット初期化からその値が使われてアナログ回路が動作するようになります。言い方を替えるとテスト終了ボタンを押さなければＦＣＡＤ４１６－ＤＳＵＢ内部のオフセット・ゲイン調整レジスタの値は揮発性メモリの中に保存されないため、そのままの状態でユニットを再起動すれば、同じ条件で再び動作する事になります。

また画面右下のEEPROM Getボタンは規定のフォーマットに従いＦＣＡＤ４１６－ＤＳＵＢのオフセット･ゲインレジスタの値をCSVファイルとして取り出す機能を持っています。



オフセット調整、ゲイン調整とＡＤ変換値との相互関係

５－２．故障・トラブル等の原因と対処

本ユニットは、ＤＯＳ／Ｖパソコン＋パワードハブのシステム構成で全数検査の上出荷されています。
お手元での動作確認方法は１－６項に記されています。動作に不具合がある時は、以下の諸点を再点検して下さい。
それでも不明なときは、システム構成等をご記入の上、弊社品質管理部まで御連絡下さい。情報伝達手段としてfax及びEmailを準備しておりますが、ご使用しやすい方の手段をお選びの上、御連絡願います。
一般的に言って、応答速度を求められるときはEmailが、複雑な情報を含む場合はfaxがお勧めだと思います。

再点検、確認ポイント

ユニット ID

単体で動作させる時、この番号が０以外の設定になっているとハンドラーがユニットを認識しません。
実際に起こり得る局面としては、マスタースレーブ接続で使用されていたスレーブユニットを、単独で使用する等の局面が考えられます。

トリガ方法

アナログレベルトリガは、アナログ変換値の値で、アナログエッジトリガは、アナログ変換値の値の変化で、夫々トリガを認識します。そのため、アナログレベルトリガでは、サンプリング開始と同時にトリガ検出が行われることがある一方で、アナログエッジトリガでは、アナログ電圧値がスレッショルドをよぎる変化を起こさないとトリガとして認識しません。詳細については、３－６項を参照して下さい。

デジタル入出力

本ユニットのデジタル入力に接続できる信号源はTTL(LS,CMOS等の５Ｖ電源動作素子)に限ります。
それ以外の電源電圧を用いている信号源を接続すると、殆ど一瞬のうちに入力素子が破壊します。
出力についてもTTL信号源以外の信号源に接続することは、即出力素子の破壊を招きます。
本ユニットで使用しているデジタル入出力素子の絶対最大定格は－０．６Ｖ～＋７．０Ｖとなっています。

アナログ入力

本ユニット単体での過電圧入力保護は±３５Vが最大です。そのため一瞬であってもこの電圧範囲を超える電圧が印加されるとアナログ入力回路素子の破壊につながります。また、アナログ回路は、デジタル回路と異なり異常電圧が伝播していく可能性が高いため、異常電圧による被害はデジタル回路に比べて大きくなりがちです。
複数チャンネル使用時の各信号源のグランド電位差にも注意が必要です。特に様々な機器からの入力を接続する機会が多いこともあいまって、この原因による回路破壊も時折見られます。予防策としては、夫々の回路を接続する前に、機器間のグランド電位差をテスターなどで確認しておく事があげられます。一般的には交流的な電位差がある場合が多いと考えられますが、直流的な電位差についても確認が必要です。
これと似た現象に、差動入力接続を行った機器との間でグランドを接続せず使用し、何かの折にこれらの機器間でグランド同士が接触することで回路が破壊する事も、時折見受けられます。本ユニットの差動入力は入力回路が非絶縁形式であるため、必ず被測定機器との間でグランド接続を行う必要があります。
更に、本ユニットのグランド電位は接続されているパソコンのグランド電位と共通であるという点にも注意が必要です。

動作確認方法

弊社では、ユーザー作成のソフトウェアについては評価しません。動作確認は本ユニット添付の弊社製プログラム（１－６項）の実行結果について推測、適否、判定を行ないます。質問の際にはこのプログラム実行結果も添付して頂くと原因の特定に繋がる可能性が高くなる場合があります。

５－３．修理のときは

本製品使用上の質問・トラブル対応・故障修理などは入手経路の如何にかかわらず、弊社宛に直接御相談下さい。商社等を経由されても弊社として問題はありませんが、情報交換の密度、修理完了までの時間などの面で多少不利になるのではないかと思われます。
また、導入当初からの不具合については、誤解や情報不足による事が多いので、事前にご相談下さい。
無償修理と有償修理の場合分けにつきましては以下の通りとなります。
納入後１年間は自然故障、及び弊社製造上の問題に起因した事が明らかな故障製品に対して無償修理を行います。但し保証期間中であっても、次の場合は有償修理となります。

ア　取扱上の不注意、誤用による故障および損傷
イ　弊社以外での修理、改造、分解掃除等による故障および損傷
ウ　泥、砂、水などのかぶり、落下、衝撃等が原因で発生した故障および損傷
エ　火災、地震、水害、落雷その他の天変地異、公害や異常電圧による故障および損傷
オ　保管上の不備（高温多湿の場所等）や手入れの不備による故障
カ　接続している他の機器に起因して故障が生じた場合
キ　その他使用者側の責に帰する故障

また、不良要因が再現されなかった場合につきましては、着払いにより製品を返却させて頂く場合もありますのでご了承下さい。

修理は宅配便によるセンドバックで行います。尚運賃は互いに発送する側が負担するものとします。
出張修理は行っておりません。簡単な故障であれば一週間程度で修理可能ですが、故障状況によっては更に日数を要する事もあります。
また、有償修理の場合の修理料金については、簡単な故障で基本料金￥８，０００円＋部品代とお考え下さい（事務手数料を含む）。修理費用限度額がある場合は、お申し出頂ければ超過する事が明確になった時点で御連絡いたします。

修理品送付先
〒301-0853
茨城県龍ヶ崎市松ヶ丘３丁目１８番地３
フレックスコア
品質管理部　林
E-mail: support@flexcore.jp
Fax:050-3488-3354

設計変更通知（ECO）

ECO20130110

ECO 番号	ECO20130110
概略内容	ECO20121126 によって Fcad416_Set_TransferSize 関数を不要と判定したが、その事により、（特に無限サンプリングにおいて）ハンドラー内部のＤ１１バッファからユーザーアプリケーションへのデータ転送タイミングが固定化し、アプリケーション作成に当たって大きな問題を与える事が判明した。そのため、上記関数を再度有効化すると共に、その内容について見直しを行った。 この事により、ハンドラー内部のＤ１１バッファからユーザーアプリケーションへのデータ移動についてより細かい制御を行う事が可能となった。ハンドラーからは、上記パケットサイズ転送毎にメッセージを発信することによって固定量のデータ転送を指示する事ができ、更にＤ１１バッファのラップラウンド条件と組み合わせる事で、無限サンプリングを行う際にもユーザーアプリケーション側で必要な量のアナログデータを必要なタイミングで取り込む事が可能となった。
変更前	

~~書式~~	~~int __stdcall Fcad416_Set_TransferSize(int size);~~
~~引数~~	~~size:通信相手のユニットとＰＣ間のデータ転送パケットサイズ（１：２語から２３：８Ｍ語まで可能）~~
~~戻り値~~	~~正常終了時:0~~ ~~エラー発生時:負の値~~
~~機能・動作~~	~~ユニットとの、ＡＤデータ転送のパケットサイズを指定する。~~ ~~サンプリング途中でのGetStatus関数に対する読み込み済みデータ数は、このパケット単位になる。又、このパケット読み込み単位で汎用入出力等の更新も行うので高速転送を狙ってパケットサイズを~~ ~~大きくするか、汎用入出力等の更新を頻繁に行うためにパケットサイズを小さくするかは~~ ~~アプリケーション側での選択にゆだねられる。（４－３項参照）尚、アプリケーション起動時は８Ｋ語の設定になっている。~~

本ハンドラーでのデータ読み込み処理は次のようになっています。
実際には、複数ユニット間の同期を取るなど、内部ではより複雑な処理を行っています。
また、下記に示すパケットサイズは、ハンドラー内部で８Ｋ語の固定サイズに設定されています。

更に、メッセージ送信関数にも本 ECO に関連した修正がある（不要メッセージ番号の削除）

書式	int __stdcall Fcad416_Plus_Message（WORD board_no ,DWORD Bit_Of_Message);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） Bit_Of_Message:　該当ビットをセットすることでメッセージ送信を要求する。 ビット０：最初のトリガ検出タイミングでメッセージを送信 ~~ビット１：Fcad416_Set_TransferSize関数で指定したデータ数を読み込んだ時メッセージを送信~~ ビット２：無限サンプリング時、リングバッファを一巡する毎にメッセージを送信 以下省略

２－４－３項の内容（追加内容）
実際のＵＳＢバス転送時間は、理論的に求める事は困難ですが、実動作時の測定結果によると概ね２０ｍＳ程度となっています。（ポストトリガサンプリング、１台動作時：尚ＵＳＢバス転送のパケット長は８Ｋ語となっています。）
また、複数台を同時に動作させる場合はラウンドロビン方式の制御になっているため、概ね上記時間×台数分の時間がかかります。

変更後

書式	int __stdcall Fcad416_Set_TransferSize(int size);
引数	size:通信相手のユニットとＰＣ間のデータ転送パケットサイズを指定する変数であり、単位はチャンネル当たりの転送データ数を示す（０：１語から１３：８K 語まで可能） size 変数と実際のパケット値の関係は以下のようになっている。 size：$2^{(size)}$×（サンプリングチャンネル数）語
戻り値	正常終了時:0 エラー発生時:負の値
機能・動作	ユニットとの、ADデータ転送のパケットサイズを指定する。 サンプリング途中でパケットサイズ受信メッセージ受付時に発行する GetStatus 関数に対する読み込み済みデータ数は、このパケット単位以上の値になる。又、このパケット読み込み単位で汎用入出力等の更新も行うので高速転送を狙ってパケットサイズを大きくするか、汎用入出力等の更新を頻繁に行うためにパケットサイズを小さくするかはアプリケーション側での選択にゆだねられる。（４－３項参照）尚、アプリケーション起動時は８K語の設定になっている。

本ハンドラーでのデータ読み込み処理は次のようになっています。
実際には、複数ユニット間の同期を取るなど、内部ではより複雑な処理を行っています。
また、下記に示すパケットサイズは、本関数で指定した値がハンドラー内部で使用されます。

更に、メッセージ送信関数の修正も元に戻し、（ビット１定義の復活）更に追加修正がある

書式	int __stdcall Fcad416_Plus_Message (WORD board_no ,DWORD Bit_Of_Message);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） Bit_Of_Message:　該当ビットをセットすることでメッセージ送信を要求する。 　ビット０：最初のトリガ検出タイミングでメッセージを送信 　ビット１：Fcad416_Set_TransferSize 関数で指定したデータ数を読み込んだ時メッセージを送信（リングバッファ一巡時には、ビット２で定義されるメッセージも発信されるが、上記指定データ数の切れ目と一致するかどうかはユーザーアプリケーションの設定によるため、他の場合と同様データ数の切れ目でメッセージが発信される。また、リングバッファ一巡時には Get_Status 関数のステータスに該当ビットがセットされているため、処理を場合わけする必要がある） 　ビット２：無限サンプリング時、リングバッファを一巡する毎にメッセージを送信 以下省略

２－４－３項の内容（追加内容）
実際のＵＳＢバス転送時間は、理論的に求める事は困難ですが、実動作時の測定結果によると概ね２０ｍＳ程度となっています。（ポストトリガサンプリング、１台動作時：ＵＳＢバス転送のパケット長が８K語の場合。この場合のＵＳＢバス速度は概ね８K語／２０mS≒４００K語／秒となります）
また、複数台を同時に動作させる場合はラウンドロビン方式の制御になっているため、概ね上記時間×台数分の時間がかかります。
ドライバーバージョンを１．０．６に変更

ECO20121126

<table>
<tr><td>ECO 番号</td><td>ECO20121126</td></tr>
<tr><td>概略内容</td><td>Fcad416_Set_TransferSize 関数について、実機動作に全く関係ない事が判明した。当初の設計方針では、同関数によって設定されるパラメータ（size:通信相手のユニットとＰＣ間のデータ転送パケットサイズ）によってターゲットとの通信量を制御しようとしていたが、実機デバッグを行った際に複数機同時制御を行わせると、パケットサイズを可変とする事によって、特にパケットサイズが大きくなった場合システム全体の協調動作が行われにくくなる事が判明したため、パケットサイズを固定化し、全体のデータ転送を多重分割化する事により、システム全体の協調動作を図る事になった。しかし、その際取説について方針変更が周知徹底されていなかったため、取説の内容が実際のシステムと異なってしまった。また、上記機能削除に係り２－４－３項の説明についても修正が発生した。</td></tr>
<tr><td>変更前</td><td>
<table>
<tr><td>書式</td><td>int __stdcall Fcad416_Set_TransferSize(int size);</td></tr>
<tr><td>引数</td><td>size:通信相手のユニットとＰＣ間のデータ転送パケットサイズ（１：２語から２３：８Ｍ語まで可能）</td></tr>
<tr><td>戻り値</td><td>正常終了時:0
エラー発生時:負の値</td></tr>
<tr><td>機能・動作</td><td>ユニットとの、ＡＤデータ転送のパケットサイズを指定する。
サンプリング途中での GetStatus 関数に対する読み込み済みデータ数は、このパケット単位になる。又、このパケット読み込み単位で汎用入出力等の更新も行うので高速転送を狙ってパケットサイズを
大きくするか、汎用入出力等の更新を頻繁に行うためにパケットサイズを小さくするかは
アプリケーション側での選択にゆだねられる。（４－３項参照）尚、アプリケーション起動時は８Ｋ語の設定になっている。</td></tr>
</table>
一方本ハンドラーでのデータ読み込み処理は次のようになっています。
実際には、複数ユニット間の同期を取るなど、内部ではより複雑な処理を行っています。

２－４－３項の内容
更にサンプリング中のメッセージ応答は、ＵＳＢバス転送の合間に行われるため、４－５－【16】項で説明したパラメータをどの値にするかによって応答レスポンスが変わってくるので注意が必要です。</td></tr>
<tr><td>変更後</td><td>
<table>
<tr><td>~~書式~~</td><td>~~int __stdcall Fcad416_Set_TransferSize(int size);~~</td></tr>
<tr><td>~~引数~~</td><td>~~size:通信相手のユニットとＰＣ間のデータ転送パケットサイズ（１：２語から２３：８Ｍ語まで可能）~~</td></tr>
<tr><td>~~戻り値~~</td><td>~~正常終了時:0~~
~~エラー発生時:負の値~~</td></tr>
<tr><td>~~機能・動作~~</td><td>~~ユニットとの、ＡＤデータ転送のパケットサイズを指定する。~~
~~サンプリング途中でのGetStatus関数に対する読み込み済みデータ数は、このパケット単位になる。又、このパケット読み込み単位で汎用入出力等の更新も行うので高速転送を狙ってパケットサイズを~~
~~大きくするか、汎用入出力等の更新を頻繁に行うためにパケットサイズを小さくするかは~~
~~アプリケーション側での選択にゆだねられる。（４－３項参照）尚、アプリケーション起動時は８Ｋ語の設定になっている。~~</td></tr>
</table>
本ハンドラーでのデータ読み込み処理は次のようになっています。
実際には、複数ユニット間の同期を取るなど、内部ではより複雑な処理を行っています。
また、下記に示すパケットサイズは、ハンドラー内部で８Ｋ語の固定サイズに設定されています。

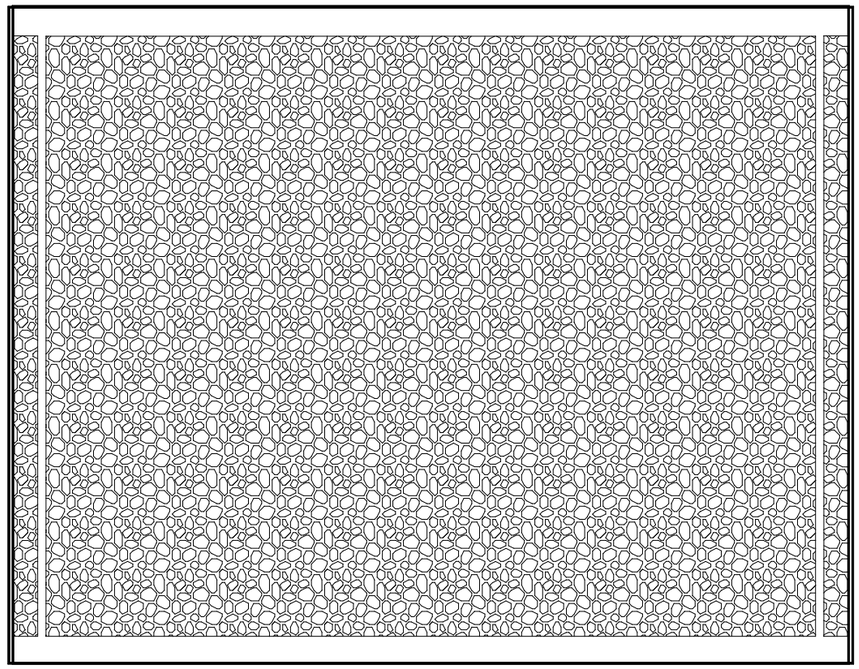
更に、メッセージ送信関数にも本 ECO に関連した修正がある（不要メッセージ番号の削除）</td></tr>
</table>

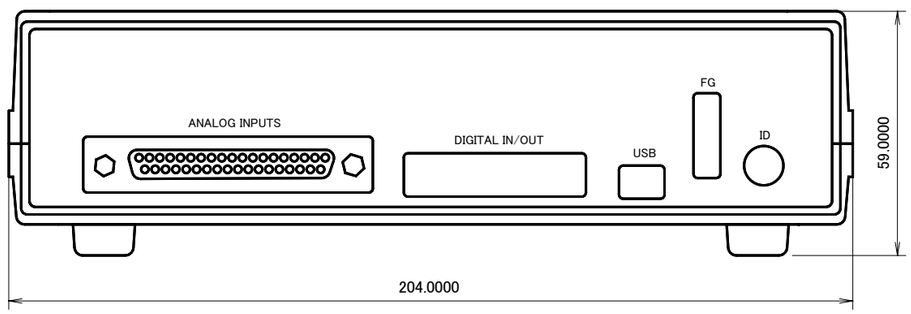
書式	int __stdcall Fcad416_Plus_Message（WORD board_no ,DWORD Bit_Of_Message);
引数	board_no:通信相手のロータリースイッチ番号（０～１５） Bit_Of_Message:　該当ビットをセットすることでメッセージ送信を要求する。 ビット０：最初のトリガ検出タイミングでメッセージを送信 ~~ビット１：Fcad416_Set_TransferSize関数で指定したデータ数を読み込んだ時メッセージを送信~~ ビット２：無限サンプリング時、リングバッファを一巡する毎にメッセージを送信 以下省略

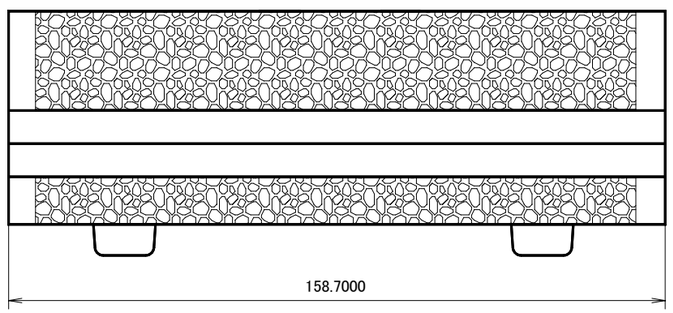
２－４－３項の内容（追加内容）
実際のＵＳＢバス転送時間は、理論的に求める事は困難ですが、実動作時の測定結果によると概ね２０ｍＳ程度となっています。（ポストトリガサンプリング、１台動作時：尚ＵＳＢバス転送のパケット長は８Ｋ語となっています。）
また、複数台を同時に動作させる場合はラウンドロビン方式の制御になっているため、概ね上記時間×台数分の時間がかかります。

《メモ》

索引







2				数量	
1				処理	
No	年月日	署名	記事	材質	
承認		製図	設計	単位 mm	尺度 1／1 三角法
			K.HAYASHI	フレックスコア TOKYO JAPAN	

名称: FCAD416−DSUB 外形図

図番: FC830002