AN197: CP210x 用シリアル通信ガイド

このアプリケーション・ノートは、次のデバイスに適用されます。CP2101、CP2102、CP2102N、CP2103、CP2104、CP2105、および CP2108。

本書は、CP210x USB - UART ブリッジ・コントローラをベースにした製品の開発者を対象としています。シリアル通信に関する情報や特定の CP210x デバイスのポート番号を取得する方法について説明し、Windows 10 以降で COM ポートのオープン、クローズ、設定、読み取り、および書き込みを行うためのコード・サンプルを提供しています。これらの API を活用するには、複数の Windows ヘッダーを含める必要があります。

要点

- COM ポートのオープンについて
- データ転送用のオープン COM ポートを準備する方法について
- COM ポートのクローズについて
- シリアル機能を示すためのサンプル・プログラム
- CP210x COM ポートを検出する方法
- アプリケーション設計ノート

第 1 章 COM ポートのオープン

COM ポートを構成してデータの送受信に使用するには、まず COM ポートをオープンする必要があります。COM ポートをオープンすると、CreateFile() 関数によってハンドルが返され、今後このハンドルはすべての通信において使用されます。COM3 をオープンするコード例を以下に示します。

```
HANDLE hMasterCOM = CreateFile("¥¥¥¥.¥¥COM3",
      GENERIC_READ | GENERIC_WRITE,
      0,
      0,
      OPEN_EXISTING,
      FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
      0);
```

CreateFile() 関数の最初のパラメータは、使用する COM ポートの番号を含む文字列です。この文字列の形式は常に ¥¥¥¥.¥¥COMX となり、「X」は使用する COM ポートの番号を表します。2 番目のパラメータにはアクセスを記述するフラグが含まれます。上記の例では GENERIC_READ と GENERIC_WRITE で示され、読み取りと書き込みの両方のアクセスを許可します。COM アプリケーションで CreateFile() を使用する場合、3 番目と 4 番目のパラメータは常に 0 にし、5 番目のパラメータは常に OPEN_EXISTING にする必要があります。6 番目のパラメータには常に FILE_ATTRIBUTE_NORMAL フラグを含める必要があります。また、FILE_FLAG_OVERLAPPED は、非同期の転送を行う場合に使用されるオプションのフラグです（本書では例としてこのオプションを使用します）。オーバーラップ・モードを使用する場合、COM ポートで読み取りと書き込みを行う関数によって、ファイル・ポインタを識別する OVERLAPPED 構造体を指定する必要があります。詳しくは、2.1 COM ポートのパージと 2.2 COM ポートの元の状態の保存で説明します（オーバーラップ I/O の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/ms686358(v=vs.85).aspx を参照してください）。7 番目の最後のパラメータは常に 0 にする必要があります。

この関数が正常に実行されると、COM ポートへのハンドルが HANDLE 変数に割り当てられます。この関数が失敗すると、INVALID_HANDLE_VALUE が返されます。返されたときにハンドルを確認し、ハンドルが有効な場合は、データ転送用の COM ポートを準備します。

第 2 章 データ転送用のオープン COM ポートの準備

ハンドルが COM ポートに正常に割り当てられたら、いくつかの手順に従ってこれを設定する必要があります。まず、COM ポートをパージしてから、COM ポートの初期状態を取得する必要があります。その後、デバイス制御ブロック（DCB）構造体によって COM ポートの新しい設定の割り当てと設定を行うことができます（DCB 構造体の詳細については、「2.3 新しい COM 状態を設定する DCB 構造体の設定」および https://msdn.microsoft.com/en-us/library/windows/desktop/aa363214(v=vs.85).aspx を参照してください）。

2.1 COM ポートのパージ

まず、PurgeComm() 関数を使用して COM ポートをパージし、COM ポートから送信される、または COM ポートが受信する既存のデータを消去します。

```
PurgeComm(hMasterCOM, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR);
```

PurgeComm() 関数の最初のパラメータは、パージされるオープン COM ポートへのハンドルです。2 番目のパラメータには、次の実行すべき操作を表すフラグが含まれます。4 つすべてのフラグ（PURGE_TXABORT、PURGE_RXABORT、PURGE_TXCLEAR、および PURGE_RXCLEAR）を必ず使用する必要があります。最初の 2 つのフラグは、オーバーラップしている読み込みと書き込み操作を終了し、最後の 2 つのフラグは出力と入力バッファを消去します。

この関数が正常に実行されると、0 以外の値が返されます。この関数が失敗すると、0 が返されます。返されたときに、戻り値を確認します。0 以外の場合、COM ポートの設定を続行します（PurgeComm() 関数の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/aa363428(v=vs.85).aspx を参照してください）。

2.2 COM ポートの元の状態の保存

COM ポート設定はさまざまなニーズに合わせて変更できるため、COM ポートがクローズしたときに COM ポートを元の状態に戻せるように、COM ポートの現在の状態を取得して保存するようにしてください。これを行うには、GetCommState() 関数を使用します。

```
DCB dcbMasterInitState;
GetCommState(hMasterCOM, &dcbMasterInitState);
```

GetCommState() 関数の最初のパラメータは、設定を取得するオープン COM ポートへのハンドルです。2 番目のパラメータは、COM ポートの設定を保存する DCB 構造体のアドレスです。また、この DCB 構造体は、COM ポートに新しい設定を指定するときに、初期状態として使用する必要があります（2.3 新しい COM 状態を設定する DCB 構造体の設定を参照）。

この関数が正常に実行されると、0 以外の値が返されます。この関数が失敗すると、0 が返されます。返されたときに、戻り値を確認します。0 以外の場合、COM ポートの設定を続行します（GetCommState() 関数の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/aa363260(v=vs.85).aspx を参照してください）。

2.3 新しい COM 状態を設定する DCB 構造体の設定

すべての COM ポートの設定は DCB 構造体に保存されます。「2.2 COM ポートの元の状態の保存」セクションでは、GetCommState() 関数を使用して COM ポートの初期設定が含まれる DCB 構造体を取得しました。COM ポートの設定を変更するには、DCB 構造体を作成し、これに目的の設定を指定する必要があります。次に、SetCommState() 関数を使用して、これらの設定をアクティブにします。

```
DCB dcbMaster = dcbMasterInitState;

dcbMaster.BaudRate = 57600;
dcbMaster.Parity = NOPARITY;
dcbMaster.ByteSize = 8;
dcbMaster.StopBits = ONESTOPBIT;

SetCommState(hMasterCOM, &dcbMaster);

Sleep(60);
```

ここでは、新しい DCB 構造体 dcbMaster は、COM ポートの現在の設定である dcbMasterInitState に初期化されています。現在の設定に初期化されたら、新しい設定を割り当てることができます。

2.3.1 ボーレート

ボーレートのプロパティは 57600 bps に設定されますが、CP210x でサポートされているボーレートのいずれかに設定することもできます （CP210x でサポートされているボーレートの一覧については、最新のデータ・シートを参照してください）。

2.3.2 パリティ

パリティは NOPARITY に設定されますが、CP210x でサポートされている場合、ODDPARITY、EVENPARITY、SPACEPARITY、および MARKPARITY に設定することもできます （CP210x でサポートされているパリティの一覧については、最新のデータ・シートを参照してください）。

2.3.3 バイト・サイズ

バイト・サイズは 8 に設定されます。そのため、送信されるデータのすべてのバイトに 8 データ・ビットが存在します。CP210x でサポートされている場合、この値は 5、6、または 7 に設定することもできます （CP210x でサポートされているバイト・サイズの一覧については、データ・シートを参照してください）。

2.3.4 ストップ・ビット

ストップ・ビットは ONESTOPBIT に設定されますが、TWOSTOPBITS または ONE5STOPBITS（1.5）に設定することもできます （CP210x でサポートされているストップ・ビットの一覧については、最新のデータ・シートを参照してください）。データ・ビットとストップ・ビットの組み合わせは、5 データ・ビットと 2 ストップ・ビットの組み合わせ、および 6、7、または 8 データ・ビットと 1.5 ストップ・ビットの組み合わせを除き、すべてを使用することができます。

これらの各設定を目的の値に設定したら、SetCommState() 関数を呼び出して COM ポートを設定することができます。SetCommState() 関数の最初のパラメータは、設定を変更するオープン COM ポートへのハンドルです。2 番目のパラメータは COM ポートの新しい設定を含む DCB 構造体のアドレスです（DCB 構造体を使用したシリアル設定の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/aa363214(v=vs.85).aspx を参照してください）。

この関数が正常に実行されると、0 以外の値が返されます。この関数が失敗すると、0 が返されます。返されたときに、戻り値を確認します。0 以外の場合、設定を変更するための遅延時間を 60 ミリ秒とってから、COM ポートの設定を続行します。この遅延は必須ではありませんが、他の操作を行う前に設定が確実に変更されるよう、念のため 60 ミリ秒の遅延時間をとるようにしてください。

第 3 章 COM ポート間のデータの送信

COM ポートのオープンと設定が正常に完了すると、データの書き込みまたは読み取りが可能になります。

3.1 データの書き込み

書き込み操作を行うには、いくつかのステップが必要なため、書き込みが発生するたびに呼び出される write 関数を作成することをお勧めします。write 関数の例を以下に示します。

```
bool WriteData(HANDLE handle, BYTE* data, DWORD length, DWORD* dwWritten)
{
   bool success = false;
   OVERLAPPED o = {0};

   o.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

   if (!WriteFile(handle, (LPCVOID)data, length, dwWritten, &o))
   {
      if (GetLastError() == ERROR_IO_PENDING)
         if (WaitForSingleObject(o.hEvent, INFINITE) == WAIT_OBJECT_0)
            if (GetOverlappedResult(handle, &o, dwWritten, FALSE))
            success = true;
   }
   else
      success = true;

   if (*dwWritten != length)
      success = false;

   CloseHandle(o.hEvent);

   return success;
}
```

この関数に渡されるパラメータは、オープン COM ポートへのハンドル、書き込むバイト配列へのポインタ、配列内のバイト数、および書き込むバイト数を格納して返す変数へのポインタです。

関数の先頭で、2 つのローカル変数が宣言されています。書き込みの成功を格納する success という名前の bool（false に初期化され、書き込みの成功時にのみ true に設定されます）と、WriteFile() 関数に渡され、転送が完了したかどうかをアラートする overlapped オブジェクト o（hEvent が割り当てられる前に 0 に必ず初期化されます）です。CreateEvent（NULL, FALSE, FALSE, NULL）関数を使ってイベントを作成し、o の hEvent プロパティを設定し、WriteFile() 関数に渡す準備をします（CreateEvent() の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/ms682396(v=vs.85).aspx を参照してください）。

次に、書き込まれたデータ量を格納するため、ハンドル、データ、データの長さ、および変数を使って WriteFile() 関数を呼び出します（WriteFile() の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/aa365747(v=vs.85).aspx を参照してください）。この関数が正常に実行されると、0 以外の値が返されます。この関数が失敗すると、0 が返されます。if 文は、書き込みが成功したかどうかを確認し、失敗した場合は、最後のエラーを取得して、実際にエラーが発生したのか、または書き込み処理が完了しなかっただけなのかを確認します。ERROR_IO_PENDING が返された場合、オブジェクト o は書き込みが終了または失敗するまで待機します（ERROR_IO_PENDING 以外が GetLastError() 関数によって返された場合、突然の削除（surprise removal）が発生した可能性があります。突然の削除に関するコメントについては、「第 6 章 アプリケーション設計ノート」を参照してください）。待機が終わると、結果が取得され、書き込まれたバイト数が更新されます。次に、success 変数に適切な値が割り当てられ、o.hEvent のハンドルがクローズされます。書き込まれたバイト数が確認され、最後に、書き込みが正常に完了した場合、関数は書き込みの成功（true）を返します。

3.2 データの読み取り

読み取り操作を行うには、いくつかのステップが必要なため、読み取りが発生するたびに呼び出される read 関数を作成することをお勧めします。read 関数の例を以下に示します。

```
bool ReadData(HANDLE handle, BYTE* data, DWORD length, DWORD* dwRead, UINT timeout)
{
   bool success = false;
   OVERLAPPED o = {0};

   o.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

   if (!ReadFile(handle, data, length, dwRead, &o))
   {
      if (GetLastError() == ERROR_IO_PENDING)
         if (WaitForSingleObject(o.hEvent, timeout) == WAIT_OBJECT_0)
            success = true;
         GetOverlappedResult(handle, &o, dwRead, FALSE);
   }
   else
      success = true;

   CloseHandle(o.hEvent);

   return success;
}
```

この関数に渡されるパラメータは、オープン COM ポートへのハンドル、読み取るバイト配列へのポインタ、配列内のバイト数、読み取るバイト数を格納して返す変数へのポインタ、およびタイムアウト値です。

関数の先頭で、2 つのローカル変数が宣言されています。読み取りの成功を格納する success という名前の bool（false に初期化され、読み取りの成功時にのみ true に設定されます）と、ReadFile() 関数に渡され、転送が完了したかどうかをアラートする overlapped オブジェクト o（hEvent が割り当てられる前に 0 に必ず初期化されます）です。CreateEvent(NULL, FALSE, FALSE, NULL) 関数を使ってイベントを作成し、o の hEvent プロパティを設定し、ReadFile() 関数に渡す準備をします（CreateEvent() の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/ms682396(v=vs.85).aspx を参照してください）。

次に、書き込まれたデータ量を格納するために、ハンドル、データ、データの長さ、および変数を使って ReadFile() 関数を呼び出します（ReadFile() の詳細については、https://msdn.microsoft.com/en-us/library/windows/desktop/aa365467(v=vs.85).aspx を参照してください）。この関数が正常に実行されると、0 以外の値が返されます。この関数が失敗すると、0 が返されます。if 文は、書き込みが成功したかどうかを確認し、失敗した場合は、最後のエラーを取得して、実際にエラーが発生したのか、または書き込み処理が完了しなかっただけなのかを確認します。ERROR_IO_PENDING が返された場合、オブジェクト o は書き込みが終了または失敗するまで待機します（ERROR_IO_PENDING 以外が GetLastError() 関数によって返された場合、突然の削除（surprise removal）が発生した可能性があります。突然の削除に関するコメントについては、「第 6 章 アプリケーション設計ノート」を参照してください）。待機が終わると、結果が取得され、読み取られたバイト数が更新されます。次に、success 変数に適切な値が割り当てられ、o.hEvent のハンドルがクローズされます。最後に、読み取りが正常に完了した場合、関数は読み取りの成功（true）を返します。

第 4 章 COM ポートのクローズ

すべての通信が完了したら、COM ポートをクローズする必要があります。まず、COM ポートを初期状態の設定に戻してから、COM ポートへのハンドルをクローズし、無効なハンドルに設定します。コード例を以下に示します。

```
SetCommState(hMasterCOM, &dcbMasterInitState);

Sleep(60);

CloseHandle(hMasterCOM);
hMasterCOM = INVALID_HANDLE_VALUE;
```

SetCommState() 関数は「2.3 新しい COM 状態を設定する DCB 構造体の設定」で説明されている動作と同じように動作します。確実に設定できるように、60 ミリ秒の遅延が使用されています。最後に、CloseHandle() 関数を使用してデバイスをクローズします。この関数は、COM ポートのハンドルを引き受けるだけです。この関数を呼び出した後、変数を INVALID_HANDLE_VALUE に設定することが重要です。

第 5 章 シリアル通信をデモするサンプル・プログラム

USBXpressSDK for Windows には、CP210xSerialTest という名前の例が含まれています。この例には、様々なシリアル通信用関数（「第 2 章 データ転送用のオープン COM ポートの準備」、「第 3 章 COM ポート間のデータの送信」、「第 4 章 COM ポートのクローズ」の各セクションを参照してください）を使用する Visual Studio プロジェクト用のソース・コードと実行可能ファイルが含まれています。このプログラムは、2 つの COM ポート番号を受け入れ、COM ポート間で 64 バイトのテスト用データ配列を送受信する基本的なダイアログ・ベースのアプリケーションです。SDK のインストール後、例は CP210x\Examples\CP210xSerialTest に配置されます。

第 6 章 アプリケーション設計ノート

「第 2 章 データ転送用のオープン COM ポートの準備」、「第 3 章 COM ポート間のデータの送信」、および「第 4 章 COM ポートのクローズ」で使用されている関数は、Windows COMM API 関数です。示されている例は、シリアル通信を処理するうえで推奨される方法のサンプルに過ぎません。これらの関数の詳細については、https://msdn.microsoft.com/en-us/library/ff802693.aspx を参照してください。

また、SetCommState() 関数では、COM ポートのオープンとクローズ間の設定が保存されないことに注意してください。前述のように、COM ポートをオープンした後の現在の設定を取得して、クローズする前にこれらの設定を復元するようにしてください。

ここに記載されているすべての関数はエラー・コードを返します。エラーが発生した場合に GetLastError() 関数を使用してエラーを取得するために、これらの関数をネストすることをお勧めします。これにより、無効なハンドルの発生を検出および処理できるため、突然の削除の問題も解決されます。アプリケーション（CP210xSerialTest）の例には、突然の削除を検出する複数のケースが含まれています。この例では、各関数にチェックがあり、リターン・コードが true であることを確認できます。true 以外の場合、エラーが発生した場所が出力ウィンドウに表示されます。サポートされる正しい設定が関数に渡されている場合は、関数は正常に実行されます。ほとんどの失敗は、INVALID_HANDLE_VALUE が原因で発生しますが、突然の削除が発生した後はハンドルをこの値に設定する必要があります。

通常の COM ポートは常に表示されるため、データを読み取る方法がない場合でも、データを COM ポートに正常に書き込むことができます。しかし、CP210x は仮想 COM ポートであるため、デバイスを取り外した場合に、これに書き込みを試みると、デバイスが使用していたハンドルは無効になります。何らかの理由で、CP210x デバイスを取り外した場合、書き込みは失敗し、GetLastError() によって ERROR_OPERATION_ABORTED が返されます。これが発生した場合、ハンドルをクローズしてから、INVALID_HANDLE_VALUE に設定する必要があります。また、通常の COM ポートからはデータを常に読み取ることができますが、データが存在しない場合はタイムアウトになります。仮想 COM ポートとして CP210x を使用し、読み取りを行う前にこれを取り外した場合、読み取りは失敗し、GetLastError() によって ERROR_ACCESS_DENIED が返されます。同様に、これが発生した場合、ハンドルをクローズしてから、INVALID_HANDLE_VALUE に設定する必要があります。

第 7 章 リファレンス

MSDN - 以下のサイトから、特定の Windows API 関数を検索できます。

http://msdn.microsoft.com/

シリアル通信のリファレンス

https://msdn.microsoft.com/en-us/library/ff802693.aspx

通信リソースのリファレンス

https://msdn.microsoft.com/en-us/library/windows/desktop/aa363196(v=vs.85).aspx

第 8 章 ドキュメント変更リスト

改訂 1.1

2023 年 2 月
- 概要の GetPortNum の説明を削除。
- セクション第 5 章 シリアル通信をデモするサンプル・プログラムの CP210xSerialTest の例に関する情報を更新
- セクション 2.3 およびセクション 4 の Delay()を Sleep()に変更。

改訂 1.0

2017 年 2 月
- 形式を最新のスタイル・ガイドに更新。
- すべてのリファレンスを AN144 から AN721（AN144 を置換）に更新。
- MSDN 記事へのリンクを更新。
- 例が最新ではないため、COM ポートの検索に関するセクション 6 を削除。

改訂 0.9

2012 年 10 月
- 関連デバイス一覧に CP2108 を追加。

改訂 0.8

2010 年 10 月
- 関連デバイスに CP2104/5 を追加。
- セクション 7 で GetPortNum 関数を更新し、Windows 7 のサポートを追加。
- Windows 98 のサポートを削除。

改訂 0.7

2008 年 8 月
- のレジストリ・パスを修正。

改訂 0.6

2007 年 10 月
- Server 2003/Vista を含むように XP/2000 のリファレンスを更新。
- セクション の WinXP/2000 のキー一覧を更新。
- セクション のコードを更新。

改訂 0.5

2005 年 2 月
- ページ 1 の関連デバイスに CP2103 を追加。

改訂 0.4

2004 年 12 月
- を更新。

改訂 0.3

2004 年 10 月
- 最初のリリース。

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com