Description

CambridgeIC's CTU Adapter enables a PC to communicate with CambridgeIC's Central Tracking Unit (CTU) chips over a USB interface.

CambridgeIC.dll is a .NET class library providing access to CambridgeIC's CTU Adapter from a Windows PC. The class library is written in C# and can be accessed from any .NET programming language.

The CTU Adapter uses a USB communications IC supplied by FTDI. It requires FTDI's DLL and the Windows CTU Adapter Driver in order to function.

Features

- For PC software code developers
- VisualBasic.NET code examples
- Labview VI code examples
- For Windows XP or Vista operating systems

Performance

- 200 SPI transactions per second for a typical PC

Applications

- PC software for testing CambridgeIC products
- PC demonstration software
- PC test instruments needing position measurement

Related Products	
Part no.	**Description**
013-7001	CTU Adapter
021-0001	Windows CTU Adapter Driver
021-0002	CambridgeIC.dll
023-0001	Visual Basic examples

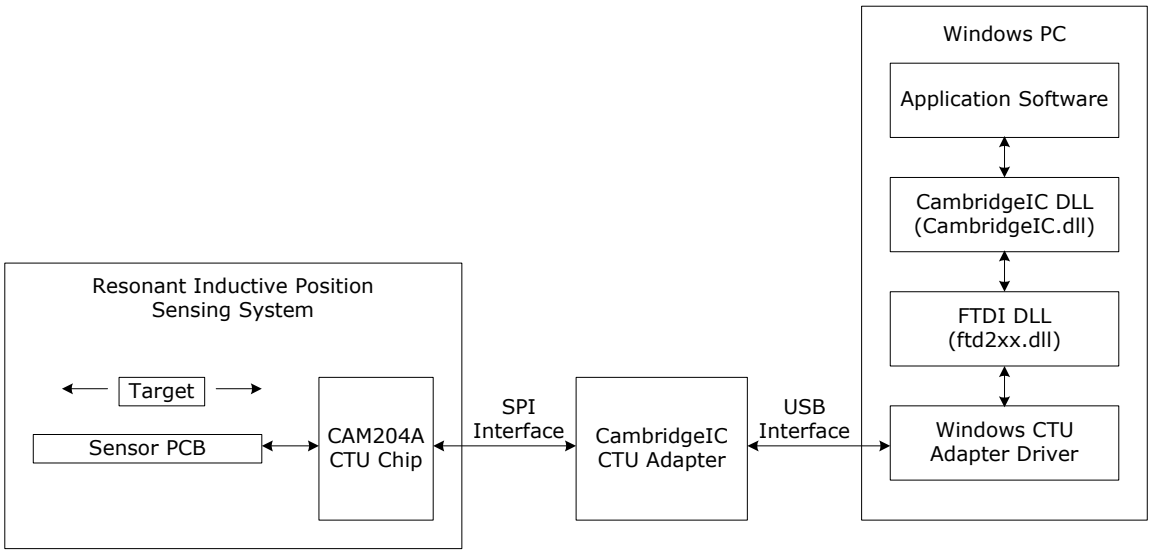

Figure 1 System block diagram

1 Introduction

1.1 Background

CambridgeIC's CTU chips such as the CAM204A are for resonant inductive position sensing. They track the position of a contactless target relative to a linear or rotary sensor built from a conventional PCB.

CambridgeIC provides a CTU Adapter which enables a Windows PC to communicate with CTU chips using a USB interface. This may be used with CambridgeIC's development and test applications.

The class library provides a variety of functions for communication with the CTU using a CTU Adapter.

The functions allow low-level access to the CTU. For higher level programming, where National Instruments' LabVIEW software is available, it is recommended to use CTU LabVIEW VIs instead, see the CTU LabVIEW VI User Guide.

1.2 Audience

This document is for software developers wishing to write demonstration or test software for CambridgeIC's CTU chips.

1.3 Overview of Functions

All classes and structures are contained in the **CambridgeIC** namespace. Functions that communicate with the Adapter are in the **Adapter** class.

The first function called should always be **Open**. This function will open the driver and allow the other functions to be called. Calling any other function before **Open** has succeeded will fail.

Once a call to **Open** has succeeded the Adapter can be configured by calling **Configure**. **Configure** sets up the Adapter for use and needs to be called once after **Open**. It is not necessary to call it again after the Adapter has been closed (**Close**) unless an Adapter reset is performed (**InternalReset**). In addition, it is not necessary to call Close directly as it will be called automatically when the **Adapter** class is disposed of.

Information on the Adapter connected to the PC can be accessed using **GetInfo**.

The CTU must be powered externally or by the Adapter for the functions below to work. The **PowerControl** function is used to turn power to the CTU from the Adapter on and off. The SPI bus also requires pull-ups. If these are not already provided in hardware, the Adapter can activate pull-ups. These are controlled with **PullUpControl**.

The main communication functions are split into three levels. At the lowest level are **WriteCharacters**, **ReadCharacters** and **WriteReadCharacters**. These functions send and receive characters directly to/from the Adapter. No formatting of the data takes place. These functions are used by the higher level functions to send and receive data and use of the higher level functions is preferred where possible.

WriteReadSpi is a middle level function. It transmits and receives data across the SPI bus between the Adapter and CTU.

The high level functions are **WriteReadCTURegisters**, **WriteReadCTURegister**, **ReadCTURegisters** and **ReadCTURegister**. These functions perform formatting on the data necessary to write to and read from the CTU registers across the SPI bus. These functions should be used for communication where possible.

The high level functions above return the state of the IO pins on the CTU. If the IO state is required without an SPI transaction, it can be read using the function **ReadIoState**.

It is possible to reset the CTU using the function **ToggleResetPin** and reset the Adapter using the function **InternalReset**. Note that after resetting the Adapter it will be necessary to call **Configure** if further communication between the application and the Adapter is necessary. **InternalReset** can be used to reset the baud rate to the default after the application is finished to allow communication programs such as HyperTerminal to be used.

All functions return a status code to indicate the whether they succeeded.

1.4 Format Conventions

Prototype functions are written in C#. Code examples are in Visual Basic .NET. Their formatting is similar, except that when code is split across more than one line, Visual Basic requires "_" between lines.

2 Description of Functions

2.1 Status Codes

All functions return a status code to indicate the whether they succeeded. The possible status codes are defined by the enumerated type **CommandStatus** in the **Adapter** class. The status code OK is used to signify that the function completed successfully. All other status codes represent an error.

```
public enum CommandStatus
{
        OK,
        DEVICE_NOT_FOUND,
        DEVICE_NOT_OPEN,
        INVALID_PARAMETER,
        READ_TIMEOUT,
        USB_INSUFFICIENT_RESOURCES,
        USB_ERROR,
        UNEXPECTED_RESPONSE,
        ERR
}
```

2.2 Open

This function opens the CTU Adapter. It should be the first function called. Calling other functions before this function has completed successfully will cause them to return DEVICE_NOT_OPEN status. The function will return OK when completed successfully. If the function returns DEVICE_NOT_FOUND, check that the CTU Adapter is connected to the PC.

Returns

Status code giving the result of the function.

Prototype

```
public CommandStatus Open()
```

Example

```
Dim AdapterInstance As New CambridgeIC.Adapter
Dim Status As CambridgeIC.Adapter.CommandStatus

Status = AdapterInstance.Open()
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
   MessageBox.Show("Can not open Adapter.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

2.3 Close

This function closes the Adapter. It may be called when all operations are complete to close internal handles held to the FTDI drivers. Calling this function is optional. The class library will call Close when the class is disposed of.

Calling Close does not alter any of the parameters set internally by the Configure function (section 2.4). It is therefore possible to call Open once Close has been called and continue working with the Adapter in the same state.

Returns

Status code giving the result of the function.

Prototype

```
public CommandStatus Close()
```

Example

```
Dim Status As CambridgeIC.Adapter.CommandStatus

Status = AdapterInstance.Close()
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
   MessageBox.Show("Can not close Adapter.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

2.4 Configure

This function configures the CTU Adapter for use. It should be called after Open is called for the first time. It is not necessary to call the function after subsequent calls to Open. Configure will perform an internal reset of the Adapter before setting up the necessary parameters.

The UseFastBaud parameter determines the baud rate used inside the Adapter. This should normally be set to true to achieve the fastest data transfer possible. The exception is when loading new Adapter firmware, which requires a slow data transfer and hence the false setting.

Note that the Adapter's baud rate is not the same as the SPI data transfer rate, which is always 1Mbit/s.

Parameters

UseFastBaud

Specifies whether to use the fast baud rate for communication

Returns

Status code giving the result of the function.

Prototype

```
public CommandStatus Configure(bool UseFastBaud)
```

Example

```
Dim FastBaud As Boolean = True
Status = AdapterInstance.Configure(FastBaud)
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not configure Adapter")
End If
```

2.5 GetInfo

This function returns a table of strings providing information about the Adapter connected to the PC. The table consists of two columns and six rows. The first column contains a description of the data. The second column contains the corresponding data. Table 1 shows the data retuned by the function.

Table 1

Description	Data
Software Version	Adapter Firmware Version Number
Vendor ID	The USB vendor ID (four digit hexadecimal).
Product ID	The USB product ID (four digit hexadecimal).
Serial Number	The serial number returned by the Adapter
Description	The description returned by the Adapter

Parameters

Info

Table of strings giving the information returned by the device. The table [rows, columns] consists of two columns: description and data. The rows are organised aaccording to Table 1.

Returns

Status code giving the result of the function.

Prototype

```
public CommandStatus GetInfo(out string[,] Info)
```

Example

```
Dim Info(5, 2) As String
Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.GetInfo(Info)
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not get info from Adapter")
End If

SoftwareVersionTextBox.Text = Info(0, 1)
VendorIdTextBox.Text = Info(1, 1)
```

```
ProductIdTextBox.Text = Info(2, 1)
SerialNumberTextBox.Text = Info(3, 1)
DescriptionTextBox.Text = Info(4, 1)
```

2.6 WriteCharacters

This function is for CambridgeIC internal use only. It performs low-level communication by sending strings of characters to the Adapter. It is the lowest level of communication function and all other communication functions (such as WriteReadCharacters, WriteReadSpi and WriteReadCtuRegister[s]) use it to transmit data.

Parameters

TxCharacters

The characters to be written to the Adapter.

Returns

Status code specifying whether the write succeeded.

Prototype

```
public CommandStatus WriteCharacters(string TxCharacters)
```

2.7 ReadCharacters

This function is for CambridgeIC internal use only. It reads a string of characters from the Adapter. It is the lowest level of communication function and all other communication functions (such as WriteReadCharacters, WriteReadSpi and WriteReadCtuRegister[s]) use it to receive data.

ReadCharacters will always return a number of complete lines unless the read times out. A read will time out if a line is not complete and no characters are read from the Adapter within 1000 ms. If a read times out, the function will return READ_TIMEOUT. A successful read will return OK. If the read failed without timing out (if USB is disconnected), USB_ERROR will be returned.

Parameters

RxCharacters

The characters read back from the Adapter.

Returns

Status code specifying whether the read succeeded.

Prototype

```
public CommandStatus ReadCharacters(out string RxCharacters)
```

2.8 WriteReadCharacters

This function is for CambridgeIC internal use only. It writes a string of characters to the Adapter and reads the response back. It uses ReadCharacters and WriteCharactes to do the sensing and receiving. Higher level communication functions (such as WriteReadSpi and WriteReadCtuRegister[s]) use it to transmit data.

As with ReadCharacters, WriteReadCharacters will always return a number of complete lines unless a read times out. A read will time out if a line is not complete and no characters are read from the Adapter within 1000 ms. If a read times out, the function will return READ_TIMEOUT. A successful read will return OK. If the read failed without timing out (if USB is disconnected), USB_ERROR will be returned.

Parameters

TxCharacters

The characters to be written.

RxCharacters

The characters read back from the Adapter.

Returns

Status code specifying whether the write/read succeeded.

Prototype

```
public CommandStatus WriteReadCharacters(string TxCharacters, out string RxCharacters)
```

Example

```
Dim Status As CambridgeIC.Adapter.CommandStatus
Dim VersionCommand As String = "! ver" + vbNewLine
Dim Version As  As String
Status = AdapterInstance.WriteReadCharacters(VersionCommand, Version)

If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not write/read characters")
End If

VersionTextBox.Text = Version
```

2.9 WriteReadSpi

This function sends and receives data across the SPI bus between the Adapter and the CTU. It uses the WriteReadCharacters function to communicate with the Adapter. When communicating with CambridgeIC's CTU Adapter, SPI transactions are usually to write (or write and read) one or more of the CTU's registers. In this case, a higher level register access function is preferred, see the next sections.

If the length of TxData is greater than the maximum permissible (or no data is supplied) the function will return INVALID_PARAMETER. If data is successfully transmitted the function will return OK.

Parameters

TxData
: Data to transmit. The maximum length of TxData is 162 words.

RxData
: Data received. The length is equal to TxData.

IoState
: The state of the IOs returned by the CTU, see section 2.16.

Returns

Status code specifying whether the write/read succeeded.

Prototype

```
public CommandStatus WriteReadSpi(UInt16[] TxData,
                                  out UInt16[] RxData,
                                  out UInt16 IoState)
```

2.10 WriteReadCtuRegister

This function writes to a single CTU register, and reads back its contents before the change. It is a high level communication function and should be used where possible in preference to lower level functions, such as WriteReadCharacters or WriteReadSpi.

The CTU register functions do not perform any checking on the address provided. The application must ensure that valid addresses are used.

Parameters

Address
: The address of the register to write to.

TxData
: Data to write.

RxData
: Data received back from the CTU.

IoState
: The state of the IOs returned by the CTU, see section 2.16.

SysId
: The SysID returned by the CTU.

Returns

Status code specifying whether the write/read succeeded.

Prototype

```
public CommandStatus WriteReadCtuRegister(UInt16 Address,
                                          UInt16 TxData,
                                          out UInt16 RxData,
                                          out UInt16 IoState,
                                          out UInt16 SysId)
```

Example

This example will write 0 to the SCW register for first sensor and will clear the SIE, SIF, NEW and GO bits.

```
Dim ScwAddress As UInt16 = &H100
Dim TxData As UInt16 = 0
Dim RxData As UInt16
Dim IoState As UInt16
Dim SysId As UInt16

Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.WriteReadCtuRegister(ScwAddress, _
                                              TxData, _
                                              RxData, _
                                              IoState, _
                                              SysId)

If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not write/read CTU register")
End If
```

2.11 WriteReadCtuRegisters

This function writes to a series of CTU registers, and reads back their contents before the change. It is a high level communication function and should be used where possible in preference to lower level functions, such as WriteReadCharacters and WriteReadSpi.

The CTU register functions do not perform any checking on the address provided. The application must ensure that valid addresses are used.

Parameters

Address
: The starting address of the register to write to.

TxData
: Data to write.

RxData
: Data received back from the CTU.

IoState
: The state of the IOs returned by the CTU, see section 2.16.

SysId
: The SysID returned by the CTU.

Returns

Status code specifying whether the write/read succeeded.

Prototype

```
public CommandStatus WriteReadCtuRegisters(UInt16 Address,
                                           UInt16[] TxData,
                                           out UInt16[] RxData,
                                           out UInt16 IoState,
                                           out UInt16 SysId)
```

Example

```
Dim Scw As UInt16 = &H81
Dim TxDataArray() As UInt16 = {Scw, 0, 0, 0, 0, 0, 0, 0}
Dim RxDataArray(TxDataArray.Length) As UInt16

Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.WriteReadCtuRegisters(ScwAddress, _
                                               TxDataArray, _
                                               RxDataArray, _
                                               IoState, _
                                               SysId)
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not write/read CTU registers")
End If
```

2.12 ReadCtuRegister

This function reads a single CTU register. It is a high level communication function and should be used where possible in preference to lower level functions, such as ReadCharacters.

The CTU register functions do not perform any checking on the address provided. The application must ensure that valid addresses are used.

Parameters

Address
The address of the register to read.

RxData
Data received back from the CTU.

IoState
The state of the IOs returned by the CTU.

SysId
The SysID returned by the CTU, see section 2.16.

Returns

Status code specifying whether the read succeeded.

Prototype

```
public CommandStatus ReadCtuRegister(UInt16 Address,
                                     out UInt16 RxData,
                                     out UInt16 IoState,
                                     out UInt16 SysId)
```

Example

```
Dim SysIdAddress As Uint16 = &HF
Dim RxData As UInt16
Dim IoState As UInt16
Dim SysId As UInt16
Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.ReadCtuRegister(SysIdAddress, _
                                         RxData, _
                                         IoState, _
                                         SysId)
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not get SysID from Adapter")
End If

SysIdTextBox.Text = RxData.ToString("X4")
```

2.13 ReadCtuRegisters

This function reads a series of CTU registers. It is a high level communication function and should be used where possible in preference to lower level functions, such as ReadCharacters.

The CTU register functions do not perform any checking on the address provided. The application must ensure that valid addresses are used.

Parameters

Address
The starting address of the register to write to.

RxData
Data received back from the CTU.

Length
The number of registers to read.

IoState
The state of the IOs returned by the CTU, see section 2.16.

SysId
The SysID returned by the CTU.

Returns

Status code specifying whether the reads succeeded.

Prototype

```
public CommandStatus ReadCtuRegisters(UInt16 Address,
                                      out UInt16[] RxData,
                                      int Length,
                                      out UInt16 IoState,
                                      out UInt16 SysId)
```

Example

```
Dim RegistersToRead As Integer = 3
Dim ResAAddress As UInt16 = &H102
Dim RxDataArray(RegistersToRead) As Uint16
Dim Status As CambridgeIC.Adapter.CommandStatus
Status  = AdapterInstance.ReadCtuRegisters(ResAAddress, _
                                           RxDataArray, _
                                           RegistersToRead, _
                                           IoState, _
                                           SysId)

If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not get results from Adapter")
End If
```

2.14 PowerControl

This function is used to turn the power to the CTU from the Adapter on or off. Unless the CTU is powered externally, this function must be called to switch CTU power on before any other functions that communicate with the CTU.

After turning the power on to the CTU the application should wait 20 ms before issuing further commands to the CTU (such as WriteReadCtuRegisters). This is to allow the CTU sufficient time to power on and self test.

Parameters

PowerOn

true if the power is to be turned on, otherwise false.

Returns

Status code specifying whether the command succeeded.

Prototype

```
public CommandStatus PowerControl(bool PowerOn)
```

Example

This example turns the power on the CTU on.

```
Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.PowerControl(True)
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not set power on CTU")
End If
```

2.15 PullUpControl

This function is used to turn the pull-ups on the SPI bus between the Adapter and the CTU on or off. They should normally be turned on for correct SPI interface function, unless pull-ups are already wired to the CTU.

Parameters

PullUpOn

true if pull-ups are to be turned on, otherwise false.

Returns

Status code specifying whether the command succeeded.

Prototype

```
public CommandStatus PullUpControl(bool PullUpOn)
```

Example

This example turns the pull-ups on the SPI bus on.

```
Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.PullUpControl(True)
If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not set pull-ups on CTU")
End If
```

2.16 ReadIoState

This function reads the state of the CTU's user configurable IO lines. Unlike the other functions above that return IO state, it does not perform an SPI transaction. It is therefore a good choice for polling the CTU's IO lines without interrupting measurements in progress.

IO state is a represented as 16-bit number. IO1 is mapped to the least significant bit, IO2 to the next most significant bit and so on. If an IO is high, the corresponding bit will be 1.

As an example, if IO1 is high and the remaining IOs are low, then IoState will be set to 0x0001. Similarly, if IO1 and IO4 are high and the remaining IOs are low, then IoState will be set to 0x0009.

Parameters

IoState

Returns the IO state.

Returns

Status code specifying whether the command succeeded.

Prototype

```
public CommandStatus ReadIoState(out UInt16 IoState)
```

Example

This example loops reading the IO state until one of the IOs goes high.

```
Dim IoState As UInt16
Dim Status As CambridgeIC.Adapter.CommandStatus
Do
    Status = AdapterInstance.ReadIoState(IoState)
    If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
        Throw New System.Exception("Could not read IO state")
    End If
Loop Until IoState <> 0
```

2.17 ToggleResetPin

This function instructs the Adapter to force the CTU's RESET pin low for a short period, forcing a *pin reset* of the CTU.

An overloaded version of this function without the Delay parameter is provided, which yields the shortest possible reset pulse of 1µs. This is the preferred setting.

Parameters

Delay [optional]

The delay in µs to wait after the reset line is asserted before bringing the CTU out of reset.

Returns

Status code specifying whether the command succeeded.

Prototype

```
public CommandStatus ToggleResetPin(UInt16 Delay)
public CommandStatus ToggleResetPin()
```

Example

```
Dim Delay As Uint16 = 100
Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.ToggleResetPin(Delay)

If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not reset CTU")
End If
```

2.18 InternalReset

This function resets the Adapter. It clears all previous configuration, resets the baud rate to the slow setting and clears the buffers.

This function can be used to place the Adapter into a known state after it has been used by an application so it can be used with communication software, such as HyperTerminal or Tera Term.

Returns

Status code specifying whether the command succeeded.

Prototype

```
public CommandStatus InternalReset()
```

Example

```
Dim Status As CambridgeIC.Adapter.CommandStatus
Status = AdapterInstance.InternalReset()

If Status <> CambridgeIC.Adapter.CommandStatus.OK Then
    Throw New System.Exception("Could not reset adapter")
End If
```

3 Document History

Revision	Date	Description
0001	20 May 2009	First Draft
0002	20 January 2010	Updated logo and style

4 Contact Information

Cambridge Integrated Circuits Ltd
21 Sedley Taylor Road
Cambridge
CB2 8PW
UK

Tel: +44 (0) 1223 413500

info@cambridgeic.com

5 Legal