Crystal Technology, Inc.

AOTF Controller Communication

Revision 0.3

2010/10/13

Document # 60-00112-01

Crystal Technology, Inc.
A Group Company of TDK-EPC Corporation

Reproduction of the contents of this document without the permission of Crystal Technology, Inc. is expressly prohibited.

Crystal Technology, Inc. reserves the right to make changes to its products or to discontinue any product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current and complete.

Crystal Technology, Inc. warrants the performance of its products and related software to the specifications applicable at the time of sale in accordance with Crystal Technology's standard warranty. Testing and other quality control techniques are utilized to the extent Crystal Technology, Inc. deems necessary to support this warranty. Specific testing of all parameters is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

Crystal Technology, Inc. products are not designed, intended, authorized, or warranted to be suitable for use in life support applications, devices, systems, or other critical applications.

Inclusion of Crystal Technology, Inc. products in such applications is understood to be fully at the risk of the customer. Use of Crystal Technology, Inc. products in such applications requires the written approval of an appropriate Crystal Technology, Inc. officer. Questions concerning potential risk applications should be directed to Crystal Technology, Inc. at the address listed below.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

Crystal Technology, Inc. assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does Crystal Technology, Inc. warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of Crystal Technology, Inc. covering or relating to any combination, machine, or process in which such products or services might be or are used.

This document has been prepared with the greatest care regarding its contents. However, in the event that it contains omissions, errors or any other misinformation, please feel free to direct comments to Crystal Technology, Inc. at the address listed below.

You are also welcome to call Crystal Technology, Inc. at (650) 856-7911, or you may FAX inquiries to (650) 354-0173.

Crystal Technology, Inc.
1040 East Meadow Circle
Palo Alto, CA 94303-4230

Revision History			
Revision	**Date**	**Who**	**Comments**
0.1	2008/09/26	Dale Gifford	Genesis.
0.2	2008/10/23	Dale Gifford	Added Plug and Play (PnP) descriptions.
0.3	2010/10/13	Dale Gifford	Updated logo, fixed typos. Release 2010-08.

Table 1: Revision History

Table of Contents

List of Tables

List of Figures

This Page Intentionally Left Blank

1. Introduction

1.1. Purpose

This document describes the techniques available to software developers for communicating with Crystal Technology's *Acousto-Optic Tunable Filter (AOTF) Controllers*. The *AOTF Controller* communicates with a desktop or notebook PC or PDA environment via a USB interface, a Serial interface, or a Bluetooth interface.

1.2. Scope

This document describes the software techniques that can be used to communicate with an *AOTF Controller*. The software techniques include USB communication, Serial communication, and Bluetooth communication, associated device drivers, and utilities.

1.3. Compatibility

The *AOTF Controller* is compatible with the following standards:

- Electronic Industries Association RS232 communication standard (EIA232). www.eia.org.

- Universal Serial Bus standard (USB). www.usb.org.

- Bluetooth Specification, Bluetooth Special Interest Group (SIG), www.bluetooth.org.

1.4. Related Documents

The following references may be useful in fully understanding and utilizing the *AOTF Controller*:

- Eight Channel Acousto-Optic Tunable Filter Controller (AOTF Controller), Revision 1.0, www.CrystalTechnology.com, Crystal Technology, Inc. 1040 East Meadow Circle, Palo Alto, CA 94303-4230.

- EZ-USB FX Technical Reference Manual, Version 1.2, www.cypress.com, Cypress Semiconductor Corporation, Interface Products Division, 15050 Avenue of Science, Suite 200, San Diego, CA 92128.

- CY7C64613 EZ-USB FX USB Microcontroller, Document # 38-08005 Rev. A, December 17, 2002, www.cypress.com, Cypress Semiconductor Corporation, 3901 North First Street, San Jose, CA 95134.

- AD9954 Data Sheet, Revision PrC, no date, www.analog.com, Analog Devices, Inc., One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106.

- Cordless Serial Adapter User's Guide, Document # 6410-00207 B, June 2004, www.socketcom.com, Socket Communications, Inc., 37400 Central Court, Newark, CA 94560.

- Socket Cordless Serial Command User's Guide, Document # 6410-00201, Revision 0.86, August 2, 2004, www.socketcom.com, Socket Communications, Inc., 37400 Central Court, Newark, CA 94560.

- TCP/IP Lean, Second Edition, Jeremy Bentham, www.cmpbooks.com, CMP Books, 1601 West 23rd Street, Suite 200, Lawrence, Kansas, 66046.

1.5. Notation

- Numbers with an "h" suffix or "0x" prefix are hexadecimal. All other numbers are decimal.
- Register and bit names ending in "**[#]**" and "**[#:#]**" signify selection of a subset of the register (e.g. **I2CS[0]** represents bit 0 of the **I2CS** register, and **I2CS[5:3]** represents bit 5 through 3 of the **I2CS** register).
- Signal names ending with '#' (e.g. **INT0#**) indicates an active low signal.
- N/A is an abbreviation for Not Applicable.
- Register bits are either set (1) or cleared (0).

2. *AOTF Controller* Communication Overview

The *AOTF Controller* is capable of communicating with a host platform (desktop, notebook, or PDA) via any of these three communication interfaces:

- USB
- Serial
- Bluetooth

The tools and techniques used for communicating with an *AOTF Controller* using each of these interfaces will be explained in the sections that follow. A typical system that utilizes the *AOTF Controller* is shown in *Figure 1*, on *page 4*.

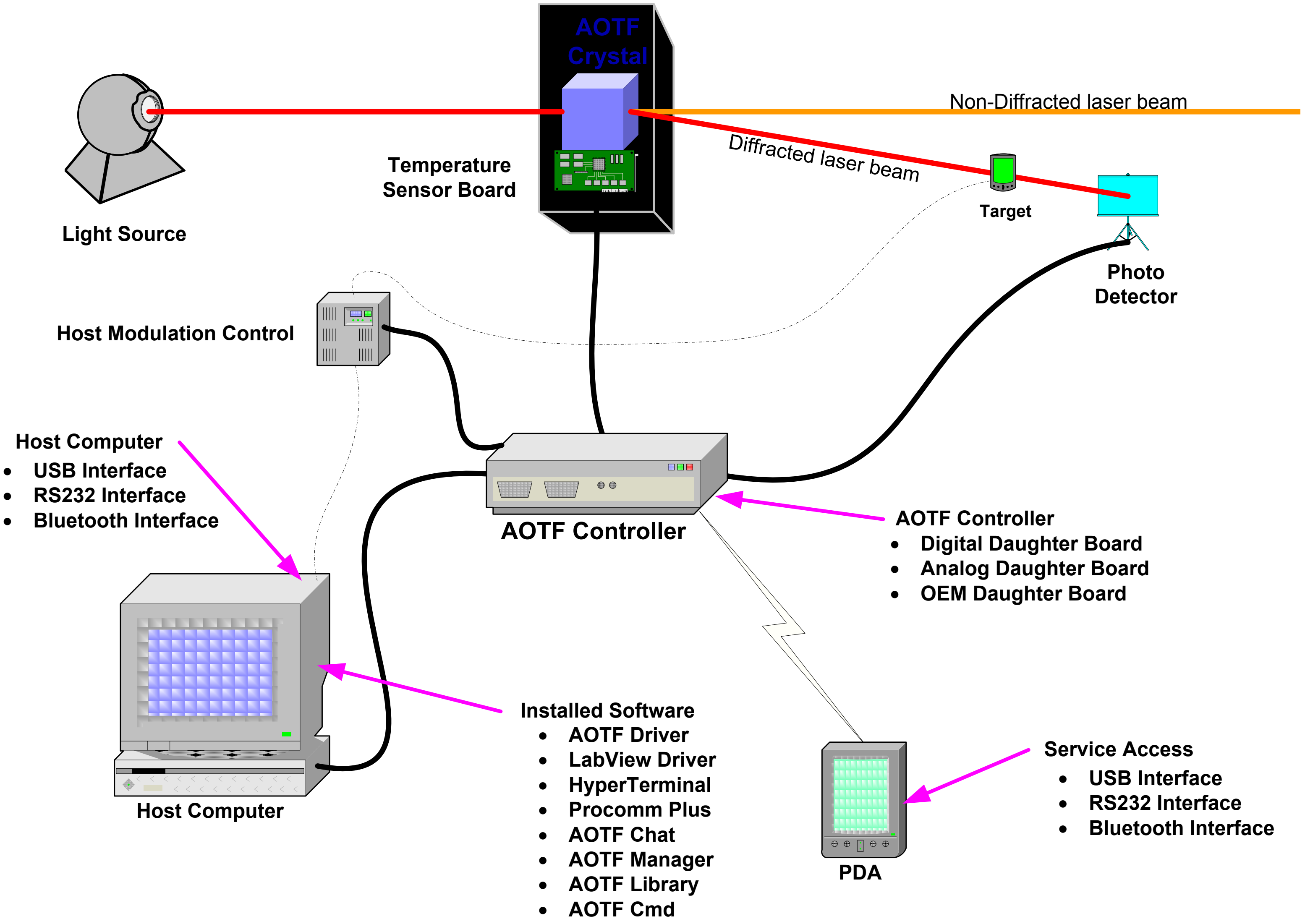

Figure 1: Typical System Utilizing the AOTF Controller

3. Communication via USB

Using the USB interface to communicate with the host platform is the most common form of *AOTF Controller* communication. Via the USB interface commands are issued from the host to the *AOTF Controller* and responses returned back to the host. There are two common scenarios for communicating with the AOTF Controller via the USB interface:

- **Using the AotfLibrary**
  This is the easiest way. The *AotfLibrary* is a wrapper around the *AotfDriver* that provides an easier interface for integration into applications. The *AotfLibrary* is provided as a DLL and can be used by any programming environment such as Visual C, C++, C#, Visual Basic, LabVIEW, etc. The AotfChat, AotfDeviceMonitor, AotfLibraryTest, and LabVIEW GUI applications use the *AotfLibrary* for interfacing with the *AOTF Controller*.

- **Using the AotfLibraryWrapperClass**
  This is the easiest way for C# applications. The *AotfLibrary* is a wrapper around the *AotfDriver* that provides an easier interface for integration into applications. The *AotfLibrary* is provided as a DLL and can be used by any programming environment such as Visual C, C++, C#, Visual Basic, LabVIEW, etc. The AotfChat and LabVIEW GUI applications use the *AotfLibrary* for interfacing with the *AOTF Controller*.

- **Using the AotfDriver**
  This is the most direct way. The *AotfDriver* exposes all of the USB interface capabilities to applications via I/O Control functions (IOCTLS – pronounced "eye octals'"). IOCTLS form the basis of communication between applications and device drivers. Using the *AotfDriver* directly can provide the best performance and access to every capability, but it usually requires more software development effort.

The two techniques for communicating via USB are shown graphically in *Figure 2,* on *page 5,* and *Figure 3,* on *page 6*.

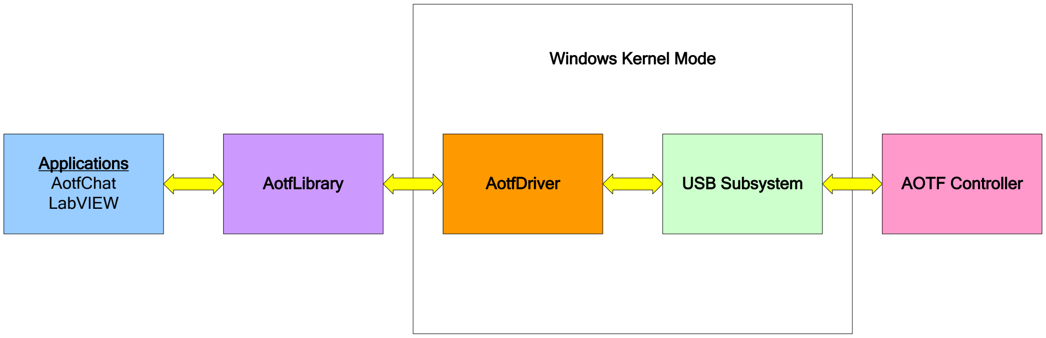

Figure 2: Communication via AotfLibrary

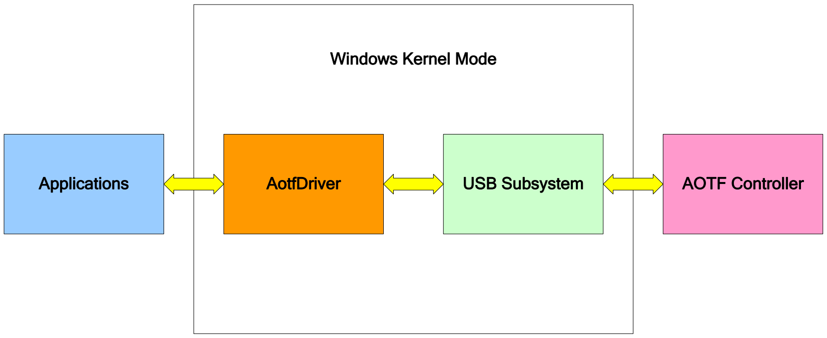

Figure 3: Communication via AotfDriver

3.1. Functions available with AotfLibrary

AotfLibrary is a standard *DLL*, and it therefore supports the standard entry point for *DLL*, *DllMain*. In addition to *DllMain*, these additional functions are available for applications to use:

- AotfOpen
- AotfGetInstance
- AotfClose
- AotfWrite
- AotfRead
- AotfIsReadDataAvailable

The associated AotfLibrary.h file is available for inclusion into application projects, and the associated AotfLibrary.lib is available for linking with application projects.

3.2. AotfOpen

`AotfOpen` must be the first function called by an application. The returned `HANDLE` is used to identify the *AOTF Controller* during subsequent calls to any routine in the *AotfLibrary*.

Usage:

```
AOTFLIBRARY_API HANDLE AotfOpen (int iInstance);
```

The returned `HANDLE` is non-zero for success, otherwise 0 for failure.

The `iInstance` parameter is a zero based index to the *AOTF Controller* to open. Each *AOTF Controller* connected to the USB bus is assigned an index by the operating system. The first *AOTF Controller* connected is assigned an `iInstance` of 0. The `iInstance` parameter is usually 0, unless there is more than one *AOTF Controller* connected to the system.

Example:

```
 int iInstance = 0;
```

```
HANDLE hAotfController = AotfOpen (iInstance);
if (hAotfController)
    {
    printf ("AotfController-%d successfully opened.", iInstance);
    AotfClose (hAotfController);
    }
else
    printf ("AotfController-%d could not be opened.", iInstance);
```

3.3. AotfGetInstance

`AotfGetInstance` returns the `iInstance` that was used to open the `HANDLE`.

Usage:

```
AOTFLIBRARY_API INT AotfGetInstance (HANDLE hAotfController);
```

The returned `iInstance` is a zero based index to the *AOTF Controller* that is referenced by the `HANDLE` parameter. Each *AOTF Controller* connected to the USB bus is assigned an index by the operating system. The first *AOTF Controller* connected is assigned an `iInstance` of 0. The `iInstance` is usually 0, unless there is more than one *AOTF Controller* connected to the system.

The `hAotfController` parameter must be a `HANDLE` returned by a successful call to *AotfOpen*.

Example:

```
int iInstance = 0;
HANDLE hAotfController = AotfOpen (iInstance);
if (hAotfController)
    {
    printf ("AotfController-%d successfully opened.\n", iInstance);
    int iReturnediInstance = AotfGetInstance (hAotfController);
    printf ("AotfGetInstance returned %d\n", iReturnediInstance);
    AotfClose (hAotfController);
    }
else
    printf ("AotfController-%d could not be opened.\n", iInstance);
```

3.4. AotfClose

`AotfClose` should be the last function called, the `HANDLE` will become invalid after it is closed.

Usage:

```
AOTFLIBRARY_API BOOL AotfClose (HANDLE hAotfController);
```

The returned value is non-zero for success, otherwise 0.

The `hAotfController` parameter must be a `HANDLE` returned by a successful call to *AotfOpen*.

Example:

```
int iInstance = 0;
HANDLE hAotfController = AotfOpen (iInstance);
if (hAotfController)
    {
    printf ("AotfController-%d successfully opened.\n", iInstance);
    int iReturnediInstance = AotfGetInstance (hAotfController);
    printf ("AotfGetInstance returned %d\n", iReturnediInstance);
    if (!AotfClose (hAotfController))
        printf ("AotfClose failed.\n");
    }
else
    printf ("AotfController-%d could not be opened.\n", iInstance);
```

3.5. AotfWrite

`AotfWrite` will send data to the *AOTF Controller*. The data should be in the form of ASCII commands that the *AOTF Controller* will interpret and execute.

Usage:

```
AOTFLIBRARY_API BOOL AotfWrite (HANDLE hAotfController,
                                 UINT uiLength,
                                 PVOID pData);
```

The returned value is non-zero for success, otherwise zero for failure.

The `hAotfController` parameter must be a `HANDLE` returned by a successful call to *AotfOpen*.

The `uiLength` parameter is the length in bytes of the data to be written.

The `pData` parameter is a pointer to the data to be written. `pData` must point to a buffer that is at least `uiLength` in size in bytes.

Example:

```
int iInstance = 0;
HANDLE hAotfController = AotfOpen (iInstance);
if (hAotfController)
    {
    printf ("AotfController-%d successfully opened.\n", iInstance);
    int iReturnediInstance = AotfGetInstance (hAotfController);
    printf ("AotfGetInstance returned %d\n", iReturnediInstance);
    char cCommand[] = "BoardId Serial\r";
    if (!AotfWrite (hAotfController, strlen (cCommand), cCommand))
        printf ("AotfWrite failed.\n");
    if (!AotfClose (hAotfController))
        printf ("AotfClose failed.\n");
    }
else
    printf ("AotfController-%d could not be opened.\n", iInstance);
```

3.6. AotfRead

`AotfRead` will get data from the *AOTF Controller*. The data will be in the form of ASCII bytes that are the responses to commands that the *AOTF Controller* has executed.

Usage:

```
AOTFLIBRARY_API BOOL AotfRead (HANDLE hAotfController,
                                UINT uiLength,
                                PVOID pData,
                                PUINT puiBytesRead);
```

The returned value is non-zero for success, otherwise zero for failure.

The `hAotfController` parameter must be a `HANDLE` returned by a successful call to *AotfOpen*.

The `uiLength` parameter is the length in buffer pointed to by `pData`.

The `pData` parameter is a pointer to a buffer where the read data will be placed. `pData` must point to a buffer that is at least `uiLength` in size in bytes.

The `puiBytesRead` parameter is the actual number of bytes read.

Example:

```
int iInstance = 0;
HANDLE hAotfController = AotfOpen (iInstance);
if (hAotfController)
    {
    printf ("AotfController-%d successfully opened.\n", iInstance);
    int iReturnediInstance = AotfGetInstance (hAotfController);
    printf ("AotfGetInstance returned %d\n", iReturnediInstance);
    char cCommand[] = "BoardId Serial\r";
    if (!AotfWrite (hAotfController, strlen (cCommand), cCommand))
        printf ("AotfWrite failed.\n");
    for (UINT uiDelay = 1000000; uiDelay; uiDelay--)
        {
        if (AotfIsReadDataAvailable (hAotfController))
            {
            uiDelay = 1000000;
            UINT uiBytesRead = 0;
            char cDataRead;
            if (!AotfRead (hAotfController, sizeof (cDataRead),
&cDataRead, &uiBytesRead))
                {
                printf ("AotfRead failed.\n");
                break;
                }
            else
                printf ("%c", cDataRead);
            }
        }
    if (!AotfClose (hAotfController))
        printf ("AotfClose failed.\n");
    }
else
    printf ("AotfController-%d could not be opened.\n", iInstance);
```

3.7. *AotfIsReadDataAvailable*

`AotfRead` will get data from the *AOTF Controller*. The data will be in the form of ASCII bytes that are the responses to commands that the *AOTF Controller* has executed.

Usage:

```
AOTFLIBRARY_API BOOL AotfIsReadDataAvailable (
                              HANDLE hAotfController);
```

The returned value is non-zero if there is data available for reading, otherwise zero if no data is available.

The `hAotfController` parameter must be a `HANDLE` returned by a successful call to *AotfOpen*.

Example:

```
int iInstance = 0;
HANDLE hAotfController = AotfOpen (iInstance);
if (hAotfController)
    {
    printf ("AotfController-%d successfully opened.\n", iInstance);
    int iReturnediInstance = AotfGetInstance (hAotfController);
    printf ("AotfGetInstance returned %d\n", iReturnediInstance);
    char cCommand[] = "BoardId Serial\r";
    if (!AotfWrite (hAotfController, strlen (cCommand), cCommand))
        printf ("AotfWrite failed.\n");
    for (UINT uiDelay = 1000000; uiDelay; uiDelay--)
        {
        if (AotfIsReadDataAvailable (hAotfController))
            {
            uiDelay = 1000000;
            UINT uiBytesRead = 0;
            char cDataRead;
            if (!AotfRead (hAotfController, sizeof (cDataRead),
&cDataRead, &uiBytesRead))
                {
                printf ("AotfRead failed.\n");
                break;
                }
            else
                printf ("%c", cDataRead);
            }
        }
    if (!AotfClose (hAotfController))
        printf ("AotfClose failed.\n");
    }
else
    printf ("AotfController-%d could not be opened.\n", iInstance);
```

3.8. *Functions and IOCTLS available with AotfDriver*

TBD (To Be Documented)….

4. Communication via Serial

TBD (To Be Documented)….

5. Communication via Bluetooth

TBD (To Be Documented)….