

Mini-DM™ Deformable Mirror System



User Manual, V.3.2 Rev C

January 2011

This document describes the following products:

- **Mini-DM – Continuous Deformable Mirror (CDM)**
1.5 μm , 3.5 μm , & 5.5 μm Stroke
- **Mini-DM – Segmented Deformable Modulator (SDM)**
1.5 μm Stroke
- **Mini Driver**
Control Driver Electronics DRV0060-02
- **Mini-DM Software**
Demo Control LinkUI Software v2.0.5.2
Mini-DM Sample Program
MATLAB Script Program

Table of Contents

1.	Introduction	3
1.1	Features.....	3
1.2	General Specifications.....	3
2.	System Components Overview.....	4
2.1	Mini-DM Deformable Mirror	4
2.2	Mirror Interface and Accessories	4
2.3	Mini Driver Electronics	4
2.5	Demonstration Software	6
3.	Getting Started.....	6
3.1	Connecting the DM: Hardware Setup for DRV0060-02.....	6
3.2	Mini Driver and LinkUI v2.0.5.2 Software Setup.....	6
3.3	BMC Demo Software	6
4.	Settings	7
4.1	Operation Overview	7
4.2	Data Line Toggle Option.....	7
4.3	Framing.....	8
4.4	Data Modes	8
4.5	Pattern Selector	8
4.6	Frame Rate	8
4.7	Data Rate.....	8
4.8	Status and Data Count	9
5.	Supplemental Information	9
5.1	Mirror Mapping for DRV0060-02	9
5.2	Mini Driver Output DB-37 Details	11
5.3	Mini Driver Front-panel AUX Connector	11
6.	Demo Software for Developers.....	12
6.1	Framing Loop Example.....	12
6.2	CI USB Library.....	14
	Object Registration and Instance Creation	14
	Definitions	14
	CIUsb_GetAvailableDevices	15
	CIUsb_SetNotify.....	15
	CIUsb_GetStatus	15
	CIUsb_SetControl	15
	CIUsb_SendFrameData.....	16
	CIUsb_StreamFrameData	16
	CIUsb_StepFrameData.....	16
	CIUsb_FlushStream.....	16
7.	Version Information	16
8.	Appendix A: BMC Software Installation Instructions	17
8.1	Installation of LinkUI Software for 32-bit Operating Systems.....	17
8.2	Installation of LinkUI Software for 64-bit Operating Systems.....	23
9.	Appendix B: Cambridge Innovations USB Software Installation Instructions for Windows XP....	27
	BMC Driver Mapping Information.....	30
10.	Other Programs	31
10.1	Mini-DM Sample Program	31
10.2	MatLab Script Program.....	35
10.3	DM Flat Map for CDM-Type Devices Only	34

Manufacturer Declarations

About this document

This manual is designed to help the reader install and operate the Mini-DM and Mini Driver. It assumes that the reader has a fundamental understanding of electronic components, electronic instruments, optics, and applicable safety procedures.

The manual describes the physical specifications of the Mini-DM and Mini Driver as well as the installation procedures that are required to use the system.

This document is available as a PDF file. Updated releases may be available by contacting moreinfo@bostonmicromachines.com.

Certification

Boston Micromachines Corporation certifies that the Mini-DM, Mini Driver, and all related components are fully functional at time of shipment.

The Mini-DM included in this system has been tested and certified to function beyond the hardware limits set on the Mini Driver. Therefore, a "safety factor" need not be applied to the commands which send voltage to the mirror during normal operation.

Warranty

Any software or hardware failures due to manufacturing or inherent defects will be repaired or replaced (at Boston Micromachines Corporation's discretion) for a period of ninety days after delivery. After ninety days, any software or hardware failures due to manufacturing or inherent defects will be repaired or replaced (at Boston Micromachines Corporation's discretion) for one year after delivery, at a cost to include parts and labor.

Limitations of Warranty

Warranty does not include any damage incurred from mishandling or misuse of the DM or driver including failure due to Electric Shock Discharge (ESD), excessive optical intensity, or by not following the instructions included in this document.

WARNINGS	
Shock Hazard	Voltages up to 300 V can be present on the deformable mirror (DM), packaging, electrodes, cable, and electronics driver.
DM Damage	The DM is highly sensitive to electrostatic discharge. Always handle the DM in an electrostatically sensitive environment while wearing a Grounding Wrist Strap. Avoid touching the electrodes on the back of the DM.
Mini Driver Damage	The Driver is also highly sensitive to electrostatic discharge. Always handle the Driver in an electrostatically sensitive environment while wearing a Grounding Wrist Strap. Avoid touching any electrical interconnect.

Symbols

The following symbols designate risk to the system operator and risk to the Mini-DM or Mini Driver.

Operator Safety Alert



System Damage Alert



1. Introduction

1.1 Features

- The DM is capable of achieving high spatial resolution shapes due to high actuator count and low inter-actuator coupling
- Highly stable operation with zero hysteresis
- Compact, high resolution DM driver electronics suitable for bench-top or OEM integration
- Real-time, high precision wavefront measurement and correction.
- LinkUI control software.

1.2 General Specifications

Deformable Mirror Specifications

- Actuator Count: 32
- Actuator Pitch: 300 – 450 μm
- Max. Stroke (surface): 1.5 – 5.5 μm **
- Aperture: 3.3 - 4.9 mm
- Inter-actuator coupling, SDM: 0%
- Inter-actuator coupling, CDM: 20%- 40%
- Mirror Coating: Aluminum or Gold
- Surface Quality: < 40 nm RMS

** The Deformable mirror can be safely actuated to the maximum stroke specified. No factor of safety is necessary to safeguard proper function of the mirror.

Mini Driver Specifications

- Computer Interface: USB 2.0
- Max Frame Rate: 34 kHz
- Resolution: 14 Bit
- Dimensions (W x D x H): 9in x 7in x 2.5 in
- Power Requirements: 24V, 2A Max
- Voltage limit in hardware to ensure safe mirror operation

Mini-DM 1.5 μm CDM

- Max Stroke: 1.5 μm
- Continuous membrane surface
- Actuator Pitch: 300 μm
- Clear Aperture: 1.50 mm x 1.50 mm
- Avg. Step Size: < 1nm

Mini-DM 1.5 μm SDM

- Max Stroke: 1.5 μm
- Segmented mirror surface
- Actuator Pitch: 300 μm
- Clear Aperture: 1.80 mm x 1.80 mm
- Avg. Step Size: < 1nm

Mini-DM 3.5 μm CDM

- Max Stroke: 3.5 μm
- Continuous membrane surface
- Actuator Pitch: 400 μm
- Clear Aperture: 2.00 mm x 2.00 mm
- Avg. Step Size: < 1nm

Mini-DM 5.5 μm CDM

- Max Stroke: 5.5 μm
- Continuous membrane surface
- Actuator Pitch: 450 μm
- Clear Aperture: 2.25 mm x 2.25 mm
- Avg. Step Size: < 3nm

2. System Components Overview

2.1 Mini-DM Deformable Mirror

The Mini-DM is fabricated using polysilicon surface micromachining fabrication methods. The microelectromechanical systems (MEMS) device is packaged and wire bonded to a ceramic chip carrier and sealed using a window enclosure. A pin grid array on the back of the package interfaces with the ZIF socket inside the mirror Interface Box (see Figure 1). This entire mirror package, including ceramic chip carrier, window enclosure, and ZIF socket are enclosed in the DM Interface Box (see Figure 3).

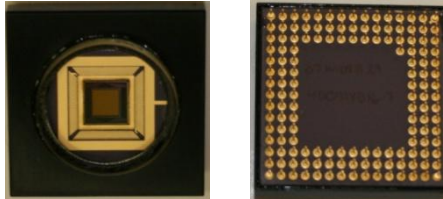


Figure 1: Packaged DM Front with protective window (left) and Back with electrodes (right)

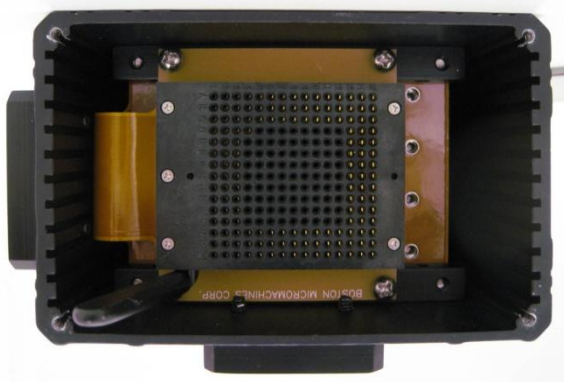


Figure 2: ZIF Socket



Figure 3: DM Interface Box

2.2 Mirror Interface and Accessories

2.2.1 DB-37 cables (4)

Four 37-channel ribbon cables extending from the rear of the DM Interface Box connect the DM to the Mini Driver electronics.

2.2.2 ESD Wrist Strap

An ESD wrist strap should be worn at all times while handling the DM or Mini Driver electronics.



2.3 Mini Driver Electronics

2.3.1 Description

The Mini-DM driver electronics provide 14-bit, digital control of 32 high-voltage output channels. A host PC, connected with a USB 2.0 interface, controls each output channel independently.

2.4 Front Panel Connections

USB	Standard USB 2.0 Mini-B type. The Mini-32 Driver draws up to the maximum power (2.5W) from the USB connector, and generates all voltages internally.
AUX	The AUX is a female, 6-pin auxiliary connector that provides timing, low voltage output, and status signals. See Section 0 for more information. Reserved for future applications such as frame synchronization to external devices, or triggering.

Operation LEDs

<i>PWR</i>	Indicates that the power cable is plugged in and the system is on.
<i>SNC</i>	Indicates frame synchronization and that data is being sent from the computer.
<i>ERR</i>	Red Error (“ERR”) will illuminate if the USB FIFO goes empty before all 32 channels have been received (likely due to the host software, if it has sent a packet which is not 64 bytes in size). The error condition will be cleared if the FIFO subsequently goes empty on a 64 byte boundary.



Figure 4: Front Panel of Mini Driver

2.4.1 Rear Panel

Connections

DB-37	Output Connectors: The output connector provides the 32 high voltage outputs, and is the interface to the external EEPROM, via a 37-pin male connector. See Section 4.2 for details.
-------	--



Figure 5: Rear Panel of Mini Driver

2.5 Demonstration Software

The Mini-DM System comes with Demonstration Software called LinkUI. It is designed to provide an introduction to the system and has basic functionality that allows the user to address the DM actuators, voltage patterns, and user-defined voltage maps.

3. Getting Started

3.1 Connecting the DM: Hardware Setup for DRV0060-02

Plug in the DB-37 Ribbon cable extending from the rear of the DM Interface Box into the connector J10 on the back of the Mini Driver box (see Figure 5).

For more information on the Output DB-37 Connector, see Section 5.2.

3.2 Mini Driver and LinkUI v2.0.5.2 Software Setup

Software Installation: See Appendix # for Software installation

1. Install BMC LinkUI Software disk in CD driver.
2. Open "My Computer" on the Desktop or Click "Start" then "My Computer."
3. Click "LinkUI Software".
4. Click "LinkUI Universal" (**See Section 8 Appendix A: LinkUI Universal Software installation for details**)
5. After installation plug the USB cable into your computer and plug the mini-USB port onto the Front Panel of the Mini Driver (see Figure 4).
6. Install Driver USB Software (**See Section 9 Appendix B: Cambridge Innovations USB Software Installation for details**)
7. The software can now be started by running the LinkUI application.

3.3 BMC Demo Software

BMC Demo Software once you have launched the BMC LinkUI software, refer to Sections 3.4 through 3.10 for an explanation of operation.

NOTE: Data entered into the BMC LinkUI software corresponds to the 14 bit DACs that drive each mirror channel. Each data number corresponds to ~18mV applied to the mirror.
The nominal driver output voltage is calculated by:

$$V_{out} = \frac{300 \cdot D}{65536},$$

Where D is the commanded 16-bit data word.

Commanded Input (Decimal)	Commanded Input (HEX)	Nominal Output Voltage (V)
10923	0x2AAB	50
16384	0x4000	75
32768	0x8000	150
32769	0x8001	150
32770	0x8002	150
32771	0x8003	150
32772	0x8004	150.018
43691	0xAAAB	200
65535	0xFFFF	300

Note that although the DACs have 14-bit precision, the data words sent to the driver are 16-bit (the two least significant bits are dropped in the DM control logic). Therefore, the driver output voltage increments by 18mV every 4 counts.

All data should be entered in the software as a 4 character, hexadecimal number, as shown in the chart above. The hexadecimal numbers are denoted by the software with a "0x" in the two left-most characters, indicating that the subsequent four characters entered by the user should be in Hexadecimal format.



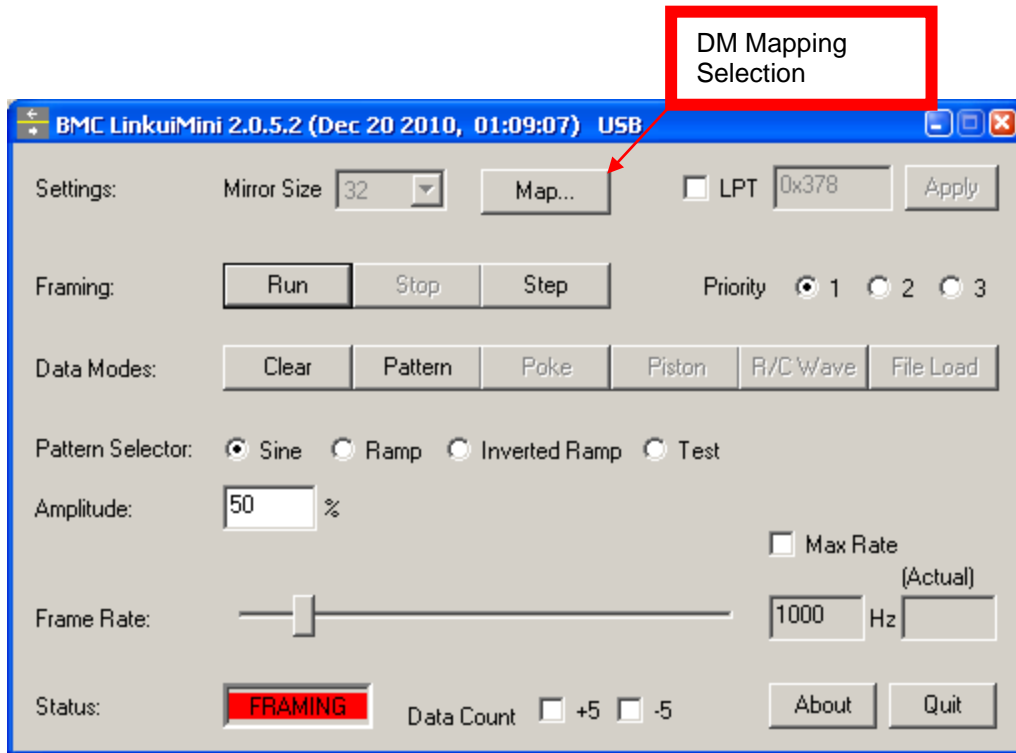


Figure 6: LinkUI Software User Interface

For Sections 4.4 to 4.8, refer to Figure 6.

4. Settings

4.1 Operation Overview

LinkUI software operates with two interfaces: the VMETRO *DPIO2* and a standard *USB* interface. The *USB* interface supports 32 actuators (and a mirror size of 32). The software system for the Mini Driver only uses the *USB* interface. The number of actuators controls the frame size in the hardware interface. In this mirror's case, the number of words sent per frame is actually 32, which corresponds to the number of physical digital-to-analog converters (DACs), (1x32).

4.2 Data Line Toggle Option

On each frame sent, LinkUI gives you the option to toggle data lines on the printer port (these signals can be used as a frame-sync for diagnostic purposes). To enable this data line toggling option, specify a printer port address by clicking the checkbox next to *LPT* and typing the printer port address in the field provided. Note that the printer port address must begin with an "0x" and must be followed by a four character hexadecimal number (see Table 1: Voltage, Decimal, Hexadecimal Conversion). Click the *Apply* button to save these changes.

4.3 Framing

There are two framing modes used by LinkUI: continuous and single step.

Button	Notes
<i>Run</i>	Starts continuous framing mode.
<i>Stop</i>	Ends continuous framing mode. <i>Stop</i> , will return all the actuator values to NULL.
<i>Step</i>	Starts single step framing mode. Single step framing mode will stop automatically after one frame.

The *Priority* of the framing thread can be set to three different levels: *1*=above normal, *2*=highest, and *3*=time critical. Depending on the performance capability of the PC and the number of active processes, this setting can be used to tune the response of the LinkUI framing thread. The option for changing the framing thread priority is only available on PCs with more than one logical processor.

4.4 Data Modes

There are six data modes supported by LinkUI, listed below. To enable data modes other than *Clear* or *Pattern*, click the *Run* button first (discussed in Section 3.6).

Data Mode	Notes
<i>Clear</i>	Sets all actuator values to NULL.
<i>Pattern</i>	Sets the actuator values to a pre-selected pattern (discussed in Section 0).
<i>Poke</i>	Allows one or more actuators, selected from a list, to be set to a user-specified constant voltage, entered as a hexadecimal number (see Chart 1: Decimal, Voltage, and Hexadecimal Conversion). When using the Poke mode, the <i>Next</i> and <i>Prev</i> buttons in the pop-up screen allow the user to poke adjacent actuators in sequence. To stop and exit the Poke mode, click the <i>OK</i> button.
<i>Piston</i>	Sets all actuators to a user-specified constant value, entered as a hexadecimal number (see Chart 1: Decimal, Voltage, and Hexadecimal Conversion). In the pop-up screen, click <i>Go</i> to run the voltage and click <i>Done</i> to stop and exit the Piston Option.
<i>R/C Wave</i>	Creates a step-through row/column pattern useful for checking larger mirrors.
<i>File Load</i>	Allows a set of actuator values to be read from a file. The file format must be a 1x160 Hexadecimal array. By default, piston and file load data modes are immediate.

4.5 Pattern Selector

The Pattern Selector chooses which voltage pattern is produced on the DM when the *Pattern* data mode is selected (discussed in Section 3.7). The peak-to-valley *Amplitude* (in percent full scale) can be specified. The pattern contains 1000 data points, independent of the operating frequency of the data interface.

The *Sine* option produces a sinusoidal pattern on the DM. The *Ramp* option produces a slanted rising then falling pattern on the DM. The *Inverted Ramp* option produces a slanted falling then rising pattern on the DM. The *Test* option produces a random pattern of voltages on the DM.

4.6 Frame Rate

The *Frame Rate* selector controls the frame rate on the interface. The maximum frame rate is dependent on a number of factors: the performance level of the host PC, the number of actuators selected, and the *Max Rate* setting, which uses overlapped I/O for peak performance. The *Actual* frame rate achieved is displayed and is determined by counting iterations of the framing loop and using the high resolution timer.

4.7 Data Rate

The *Data Rate* selector controls the actuator data rate on the parallel interface (DPIO2 only). This is not used with the Mini-DM system.

4.8 Status and Data Count

The *Status* indicator shows the framing status. It turns green when frames are being sent and turns red when no frames are being sent. The *Data Count* selector allows the number of actuators in the frame to be varied by +/- 5 actuators. This is used for diagnostic purposes to check for DM driver underflow/overflow error reporting.

5. Supplemental Information

5.1 Mirror Mapping for DRV0060-02

- This Product uses **MiniDM-00** Mapping. This mapping is the default mapping and is automatically selected by LinkUI Software. Any other mapping will result in inaccurate data.
- Orient your Deformable Mirror so that the Boston Micromachines Logo on the Interface Box is in the top left corner, as shown in Figure 3.
- The actuator numbers on your Deformable Mirror are represented by the DM Actuator map in Figure 7.
- Refer to Table 1 to see the mapping relationship between the DM, Mini Driver, and LinkUI Software.

Figure 7: DM Actuator Map

	1	2	3	4	
5	6	7	8	9	10
11	12	13	14	15	16
17	18	19	20	21	22
23	24	25	26	27	28
	29	30	31	32	

Table 1: DM, Mini-DM-Driver pin outs, and LinkUI Software Mapping

LinkUI Actuator #	PGA Pin #	Driver Output Pin #
A1	A3	20
A2	B7	1
A3	B8	21
A4	B10	2
A5	B2	22
A6	B4	3
A7	B6	23
A8	A8	4
A9	A11	24
A10	C11	5
A11	A1	25
A12	C5	6
A13	C6	26
A14	C8	7
A15	B9	27
A16	A13	8
A17	N4	12
A18	P5	30
A19	N7	13
A20	N9	31
A21	Q11	14
A22	P13	32
A23	P3	15
A24	Q4	33
A25	Q6	16
A26	Q9	34
A27	P11	17
A28	Q14	35
A29	N6	18
A30	Q7	36
A31	Q10	19
A32	Q12	37

Table 2: DB-37 Connector EEPROM, GND, and +3.3V pin Location

Driver Output Pin#	Functional Description
9	EEPROM Signal
10	GND Presence detect
11	EEPROM Signal
28	+3.3V
29	Ground

5.2 Mini Driver Output DB-37 Details

The output connector on the Mini Driver’s rear-panel provides a total of 32 high voltage outputs via a 37-pin female connector. The pins are numbered as shown in Figure 8.

All pins carry HV output signals except for pins 10 (ground presence-detect), pin 29 (ground), Pin 28 (+3.3v), and pins 9 and 11 (used for EEPROM functions). The presence detect (pin 10) must be connected to ground at the load to activate the high voltage power supply.

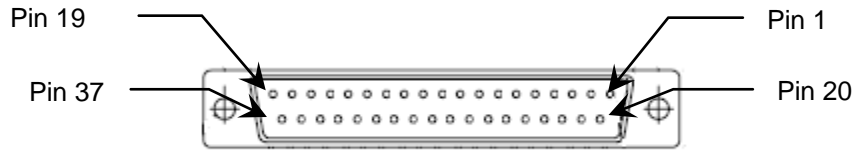


Figure 8: Mini Driver Rear-panel DB-37 Female Output Connector

5.3 Mini Driver Front-panel AUX Connector

An auxiliary (AUX) connector is provided on the front panel of the Mini Driver, reserved for future applications such as frame synchronization to external devices, or triggering. The connector is a standard circular Mini-DIN 6-pin receptacle, with power, ground and 4 unassigned low voltage logic-level I/O signals assigned as shown in Figure 9 and Table 2 below. The shield is also tied to ground.



Figure 9: Mini Driver Front-panel AUX Connector

Pin	Description
1	LVIO_0
2	LVIO_1
3	LVIO_2
4	LVIO_3
5	+3.3V
6	Ground

Table 3: Mini Driver Front-panel Aux Connector Outputs

5.3.1 Dataflow

Data sent across the USB interface is bundled into packets. For the Mini-32 Driver, these packets are organized into groups of 64 bytes (32 words, 16-bits wide), which comprise a frame of data. The digital-to-analog converters (DACs) used in the Mini-32 Driver are 14-bit, and this data field is left-justified within the 16-bit word such that the 2 least-significant bits (LSBs) are ignored.

The USB protocol ensures that every packet is properly received at the destination by means of checksum. Once a complete packet has been received and verified by the dedicated USB Controller within the Mini-32 Driver, it is held in a FIFO and ready for output to the DAC.

Each DAC requires 1 μ s. per write, so it takes 32 μ s. to write the frame once the first packet is received. The maximum frame rate is 30 KHz, approaching the reciprocal of the frame write time.

6. Demo Software for Developers

6.1 Framing Loop Example

Figure 10 shows how the framing loop is structured for the USB case. The illustration is for example purposes only and other application loops may function differently.

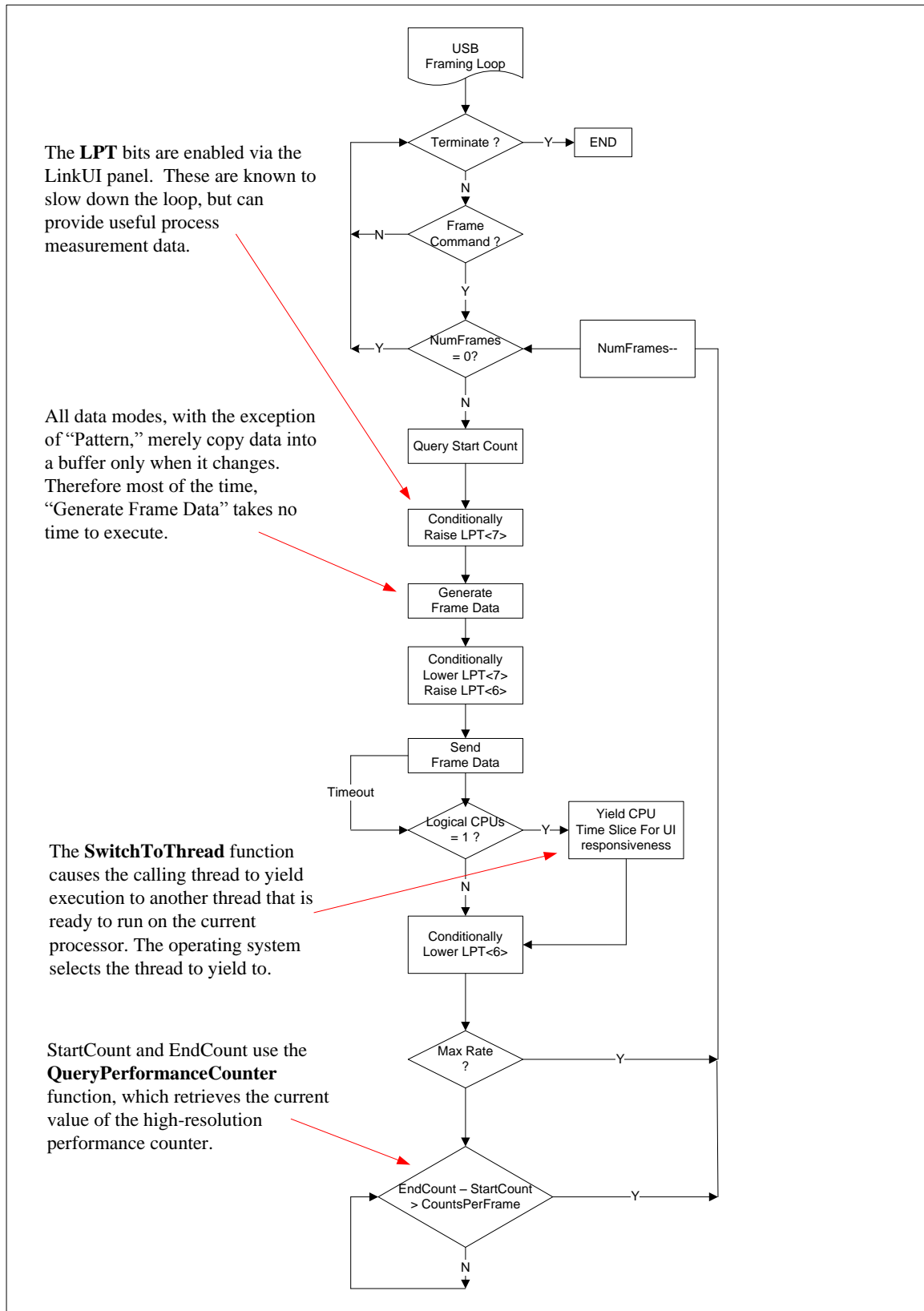


Figure 10: USB Framing Loop Example

6.2 CI USB Library

The CI USB Library is implemented as an object that conforms to Microsoft's Component Object Model (COM). A COM object is an instance of a COM class. Clients interact with a COM object only through its interfaces.

Object Registration and Instance Creation

You can use the Regsvr32 tool (Regsvr32.exe) to register and unregister COM objects such as CIUsbLib.dll that are self-registerable.

Regsvr32.exe is included with Microsoft Internet Explorer 3.0 and later versions, Windows 95 OEM Service Release 2 (OSR2) or later versions, and Windows NT 4.0 Service Pack 5 (SP5) and later versions. Regsvr32.exe is installed in the System (Windows Me/Windows 98/Windows 95) or System32 (Windows NT/Windows XP/Windows Vista) folder.

To create an instance of the COM object in a C++ program, include the following lines of code in your program:

```
#include "..\CIUsbLib\CIUsbLib.h"

#import "..\CIUsbLib\_CIUsbLib.tlb" no_namespace
using namespace std;

CComPtr<IHostDrv> m_pIHostDrv;

HRESULT hr = CoCreateInstance( __uuidof(CHostDrv), NULL, CLSCTX_INPROC,
                               __uuidof(IHostDrv),
                               (LPVOID *) &m_pIHostDrv);
```

Definitions

```
#define MAX_USB_DEVICES 32 // This is the maximum number of
devices supported
#define USB_NUM_ACTUATORS_MULTI 160 // Number of words sent in either 128
or 140 actuator modes (MULTI)
#define USB_BYTES_PER_FRAME_MULTI (USB_NUM_ACTUATORS_MULTI*2)
#define USB_NUM_ACTUATORS_MINI 32 // Number of words sent in either 32
actuator mode (MINI)
#define USB_BYTES_PER_FRAME_MINI (USB_NUM_ACTUATORS_MINI*2)
#define EXT_EEPROM_BYTES 128

// Error codes from CIUSBLib interface methods
#define H_DEVICE_STATUS_OK 0 // No Errors
#define H_DEVICE_NOT_FOUND -1 // USB Device was not found
#define H_DEVICE_NO_COMM -2 // USB communication was lost
#define H_DEVICE_CMD_ERR -3 // Unknown command
#define H_DEVICE_TIMEOUT -4 // USB transfer time out

// USB Message callback definitions
#define CIUsb_MESSAGE_USBDEVICE 0 // USB PNP Message
#define CIUsb_MSG_USBDEV_REMOVAL 0 // USB PNP Message: removal
#define CIUsb_MSG_USBDEV_ARRIVAL 1 // USB PNP Message: arrival

// Used in CIUSBLib CIUsb_GetStatus()
#define CIUsb_STATUS_DEVICENAME 0 // Get Device Name
#define CIUsb_STATUS_VID 1 // Get Device Vendor ID
#define CIUsb_STATUS_PID 2 // Get Device Product ID
#define CIUsb_STATUS_FRAME_ERROR 3 // Get value of FRAME_ERROR bit
#define CIUsb_STATUS_PD_HVA_E 4 // Get value of PD_HVA_E bit
#define CIUsb_STATUS_CABLE_OK 5 // Get value of CABLE_OK bit
#define CIUsb_STATUS_EXT_EEPROM 6 // Get value of EEPROM probe: N=size,
0=absent
#define CIUsb_STATUS_EXT_EEPROM_READ 7 // Get EEPROM data
```

```
// Used in CIUSBLib CIUsb_SetControl()
#define CIUsb_CONTROL_ASSERT_FRAME_SYNC 0 // Set FRAME_SYNC = 1
#define CIUsb_CONTROL_DEASSERT_FRAME_SYNC 1 // Set FRAME_SYNC = 0
#define CIUsb_CONTROL_ASSERT_FRESET 2 // Set FRESET = 1
#define CIUsb_CONTROL_DEASSERT_FRESET 3 // Set FRESET = 0
#define CIUsb_CONTROL_ASSERT_HV_ENAB 4 // Set HV_ENAB = 1
#define CIUsb_CONTROL_DEASSERT_HV_ENAB 5 // Set HV_ENAB = 0
#define CIUsb_CONTROL_ASSERT_LV_SHDN 6 // Set LV_SHDN = 1
#define CIUsb_CONTROL_DEASSERT_LV_SHDN 7 // Set LV_SHDN = 0
#define CIUsb_CONTROL_ASSERT_EXT_I2C 8 // Set EXT_I2C = 1
#define CIUsb_CONTROL_DEASSERT_EXT_I2C 9 // Set EXT_I2C = 0
#define CIUsb_CONTROL_MINI_MODE 10 // Set MINI Mode (frame size =
64 bytes)
```

CIUsb_GetAvailableDevices

```
HRESULT CIUsb_GetAvailableDevices
([out] LONG *pDeviceIds, [in] LONG nSizeBuff, LONG * p_nStatus);
```

CIUsb_GetAvailableDevices() returns an array of device IDs that can be used in later calls as the nDevId parameter. nSizeBuff is the size in LONG words of the array. p_nStatus points to the return status word. This method should be called after each notification of USB device removal or USB device arrival.

CIUsb_SetNotify

```
HRESULT CIUsb_SetNotify
([in] HWND hWnd, [in] UINT uMessageId);
```

CIUsb_SetNotify() specifies a window handle and message for notification of USB device removal and arrival events. When a device is inserted or removed, the message uMessageId is sent to hWnd. Upon receiving the message, the handler can check lParam for CIUsb_MESSAGE_USBDEVICE and wParam for either CIUsb_MSG_USBDEV_REMOVAL or CIUsb_MSG_USBDEV_ARRIVAL. The method CIUsb_GetAvailableDevices() should be called in either case to update the application's view of available devices.

CIUsb_GetStatus

```
HRESULT CIUsb_GetStatus
([in] LONG nDevId, [in] LONG nStatId, [out] LONG* p_nStatus);
```

CIUsb_GetStatus() returns requested status from device index nDevId. The status requested is specified by nStatId. The status is returned using the p_nStatus pointer. The status requests that are currently supported are:

CIUsb_STATUS_DEVICENAME	(char *)	Device "FriendlyName"
CIUsb_STATUS_VID	(long *)	Device Unique Vendor ID
CIUsb_STATUS_PID	(long *)	Device Unique Product ID
CIUsb_STATUS_FRAME_ERROR	(long *)	Device Framing Error Status
CIUsb_STATUS_PD_HVA_E	(long *)	Device High Voltage Status
CIUsb_STATUS_CABLE_OK	(long *)	Device Cable Status

CIUsb_SetControl

```
HRESULT CIUsb_SetControl
([in] LONG nDevId, [in] LONG nCntlId, [out] LONG* p_nStatus);
```

CIUsb_SetControl() commands control bits on device index nDevId. The control function is specified by nCntlId. The control functions that are currently supported are:


```

CIUsb_CONTROL_ASSERT_FRAME_SYNC      Set FRAME_SYNC = 1
CIUsb_CONTROL_DEASSERT_FRAME_SYNC    Set FRAME_SYNC = 0
CIUsb_CONTROL_ASSERT_FRESET          Set FRESET      = 1
CIUsb_CONTROL_DEASSERT_FRESET        Set FRESET      = 0
CIUsb_CONTROL_ASSERT_HV_ENAB         Set HV_ENAB     = 1
CIUsb_CONTROL_DEASSERT_HV_ENAB       Set HV_ENAB     = 0

```

CIUsb_SendFrameData

```

HRESULT CIUsb_SendFrameData
([in] LONG nDevId, [in] BYTE* pFrameData, [in] LONG nBuffSize, [out] LONG* p_nStatus);

```

The CIUsb_SendFrameData() method is identical to and has been replaced by CIUsb_StepFrameData(). See section 4.9.

CIUsb_StreamFrameData

```

HRESULT CIUsb_StreamFrameData
([in] LONG nDevId, [in] BYTE* pFrameData, [in] LONG nBuffSize, [out] LONG* p_nStatus);

```

CIUsb_StreamFrameData() sends a frame of actuator data to the device index nDevId. The pFrameData parameter points to the actuator data and the nBuffSize parameter specifies the size of the data buffer in bytes. When using this method, frame data is buffered to maximize throughput at the expense of data latency. There will be some delay between calling this method and the frame data arriving at the target.

CIUsb_StepFrameData

```

HRESULT CIUsb_StepFrameData
([in] LONG nDevId, [in] BYTE* pFrameData, [in] LONG nBuffSize, [out] LONG* p_nStatus);

```

CIUsb_StepFrameData() sends a frame of actuator data to the device index nDevId. The pFrameData parameter points to the actuator data and the nBuffSize parameter specifies the size of the data buffer in bytes. When using this method, frame data is not buffered to minimize latency at the expense of throughput. Each call to this method will result in data arriving at the target with minimal latency. The overall frame rate will be limited as compared to the CIUsb_StreamFrameData() method.

CIUsb_FlushStream

```

HRESULT CIUsb_FlushStream
([in] LONG nDevId, [out] LONG* p_nStatus);

```

CIUsb_FlushStream() terminates a frame stream to the device index nDevId. When using the CIUsb_StreamFrameData() method, CIUsb_FlushStream() should be called at the end of streaming operations to properly terminate. Some frames may be lost (and not sent to the target) as the stream buffers are cleared.

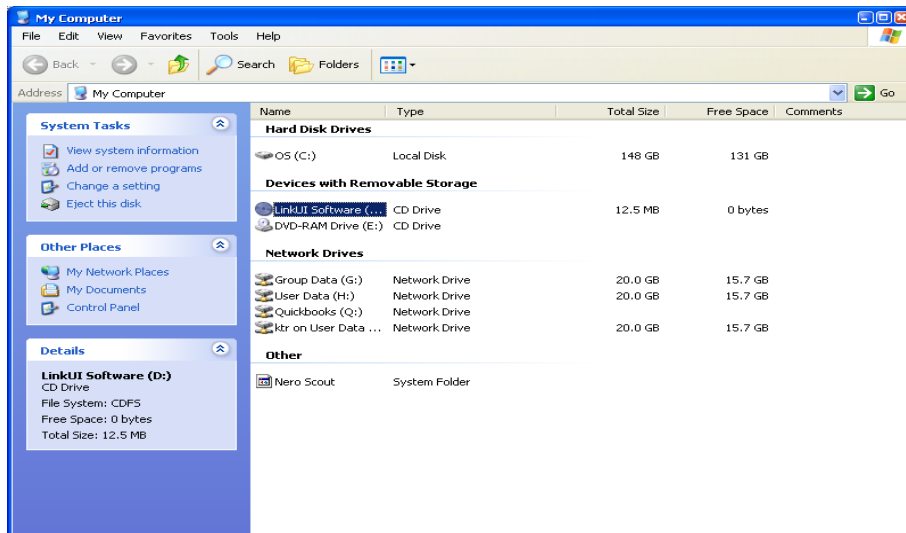
7. Version Information

v. 2.0	11/21/2008	Added Interface box details	M.S.
v. 2.1	05/11/2009	Removed EEPROM Information	K.R.
v. 3.0	07/22/2009	Added picture with new connector	
v. 3.0	07/22/2009	Added new output map for mini	K.R.
v. 3.1	01/08/2010	Update procedure for LinkUI Universal	K.R.
v. 3.2	03/23/2010	Added reference to Matlab, Sample Program and LinkUI Version Update.	K.R.
v. 3.2 rev b	07/07/2010	Add DM Flat Map, CIUsbLib.dll regsvr	K.R.
v. 3.2 rev c	01/11/2010	Added Software version update and 64-bit OS Software instructions, removed CIUsbLib.dll registration	KR

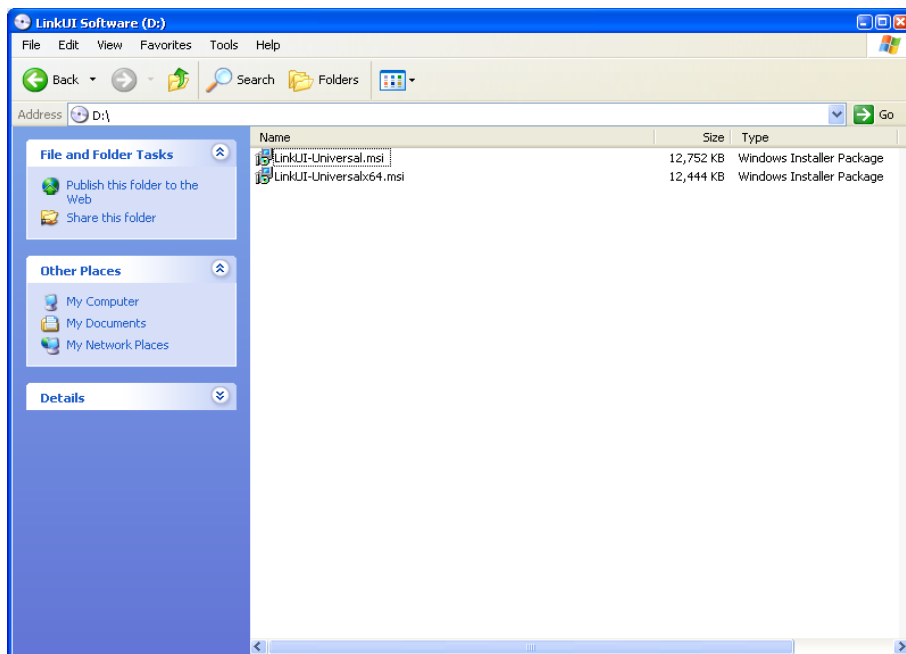
8. Appendix A: BMC Software Installation Instructions

8.1 Installation of LinkUI Software for 32-bit Operating Systems.

8.1.1 Open “My Computer” on Desktop or Click “Start” then “My Computer” and double click on “LinkUI Software”



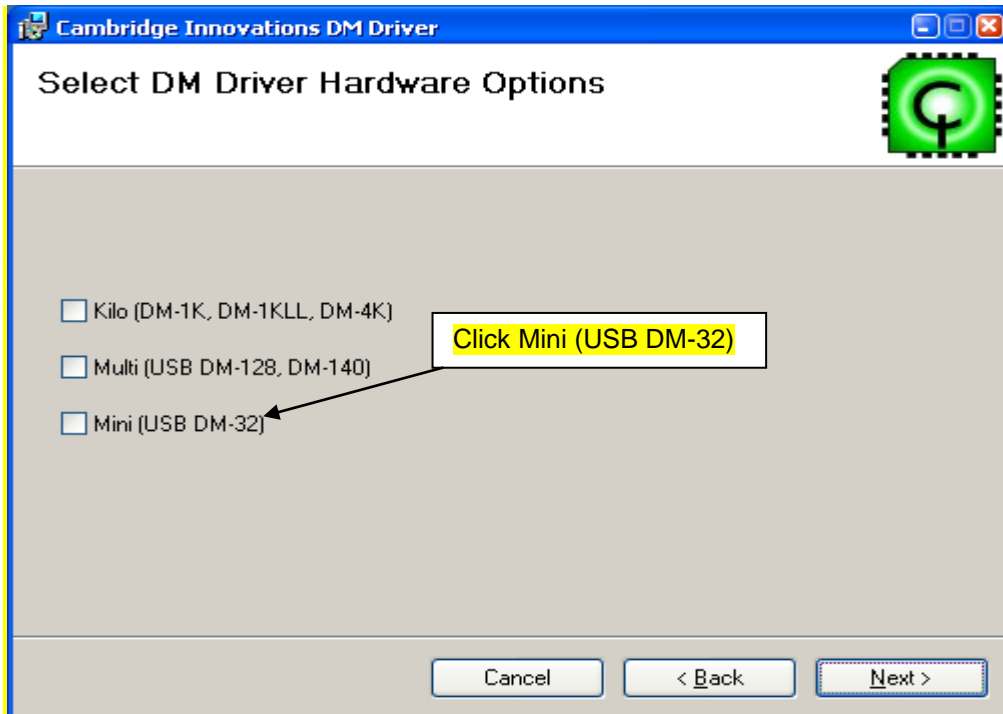
8.1.2 Double click “LinkUI-Universal” and the Setup Wizard will execute.



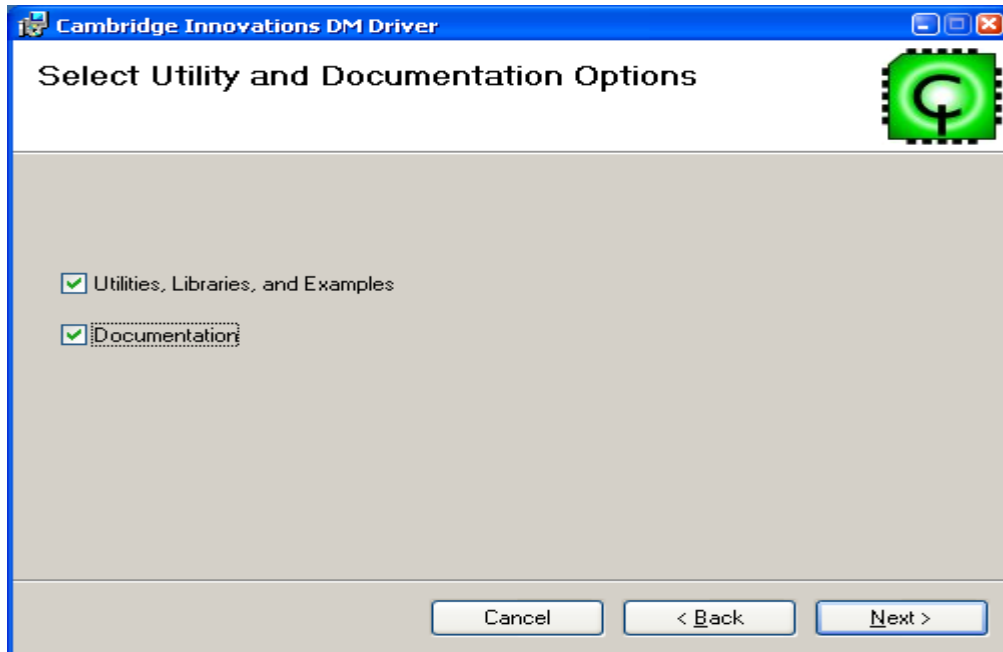
8.1.3 Click "Next" to continue.



8.1.4 Choose "Mini (USB DM-32)" and click "Next".




- 8.1.5 Check “Utilities, Libraries, and Examples” and “Documentation.” Click “Next” when finished.

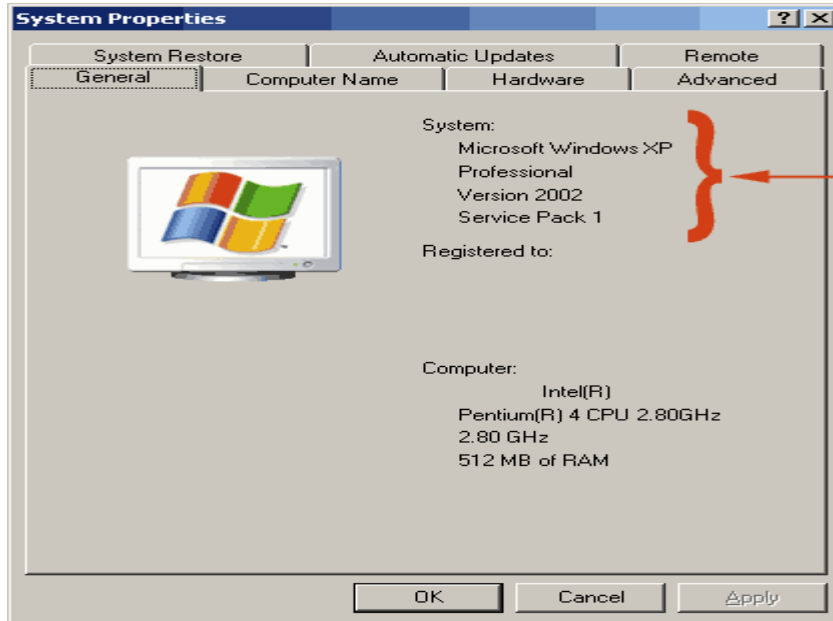


- 8.1.6 Choose Operating System.

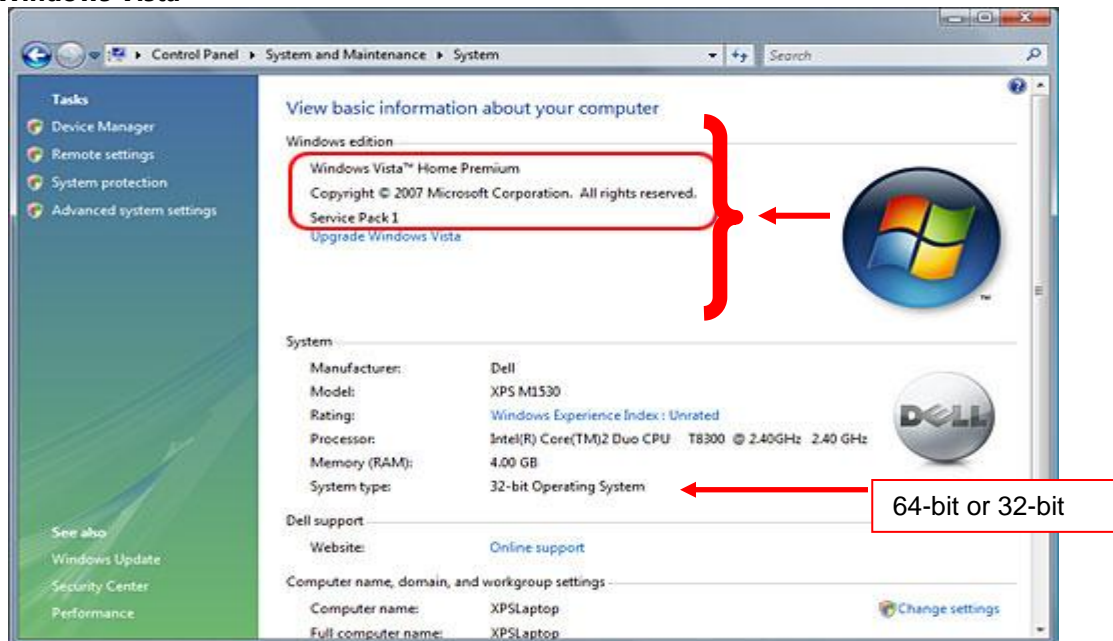


- **** 1) Open System by clicking the **Start** button , clicking **Control Panel**, clicking **System and Maintenance** in Vista and Windows 7, **Performance and Maintenance** in Windows XP and then clicking **System**.
- 2) Under **System**, you can view the operating system type XP, Vista or Windows 7 and if it is a 64-bit or 32-Bit Platform.

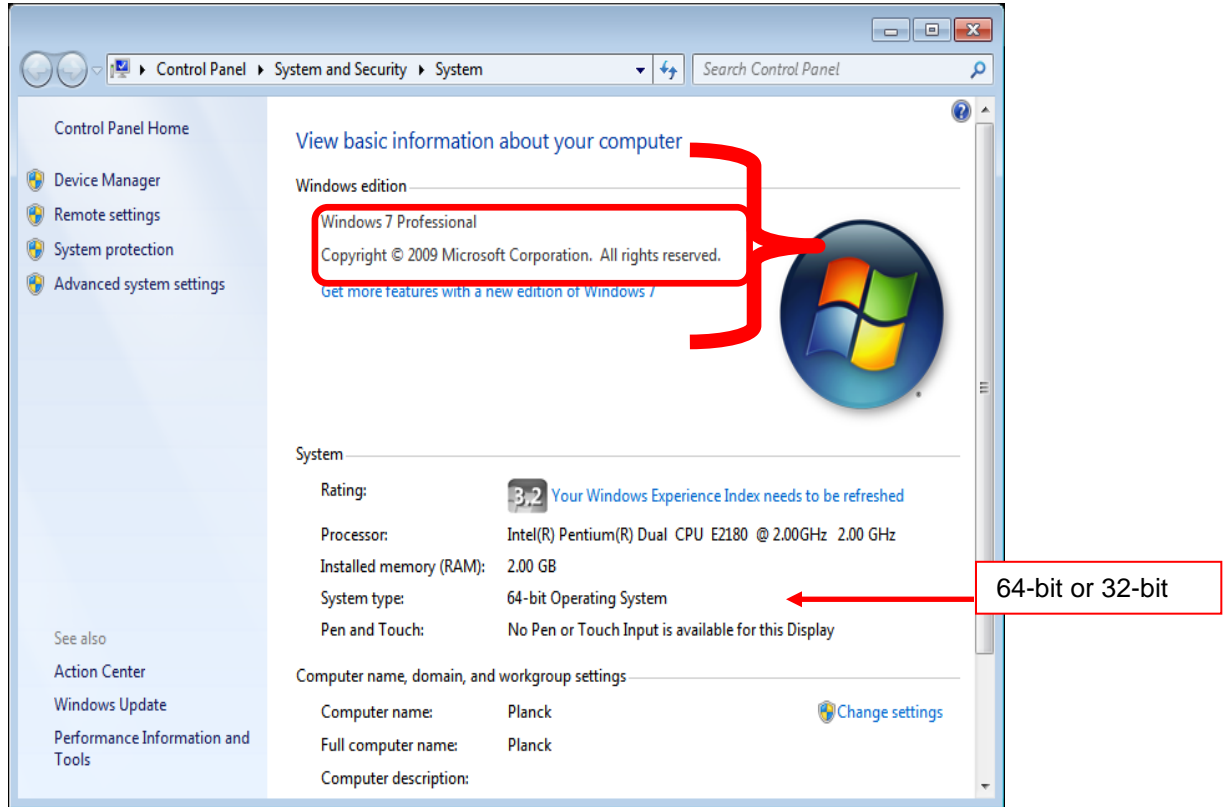
Window 32-bit edition



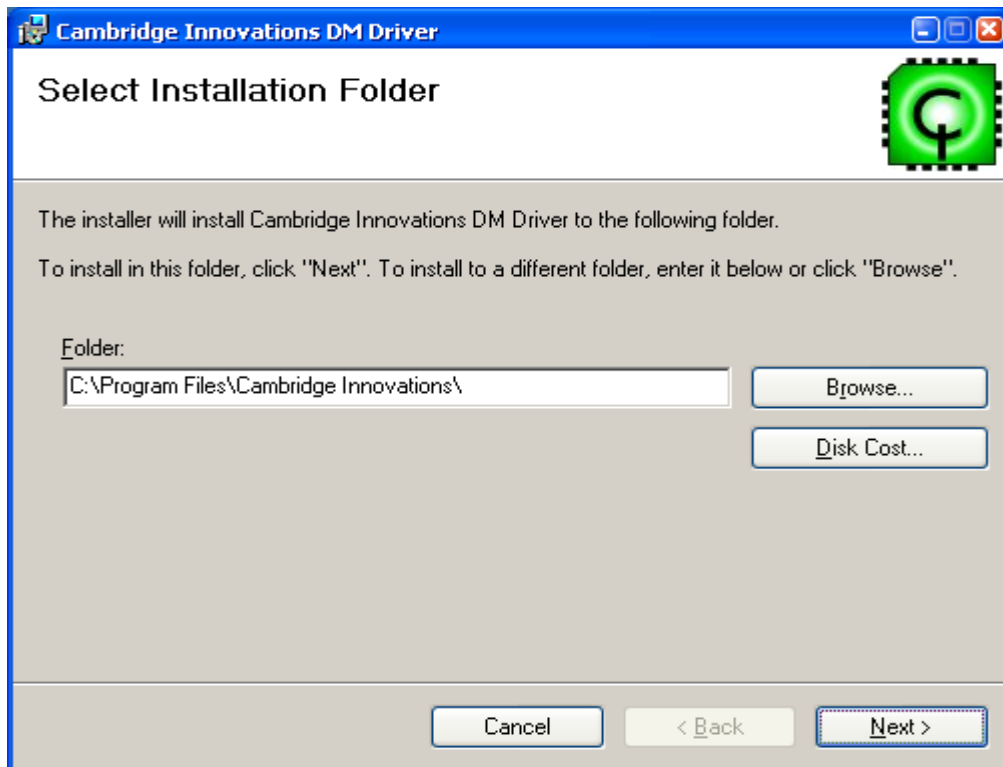
Windows Vista



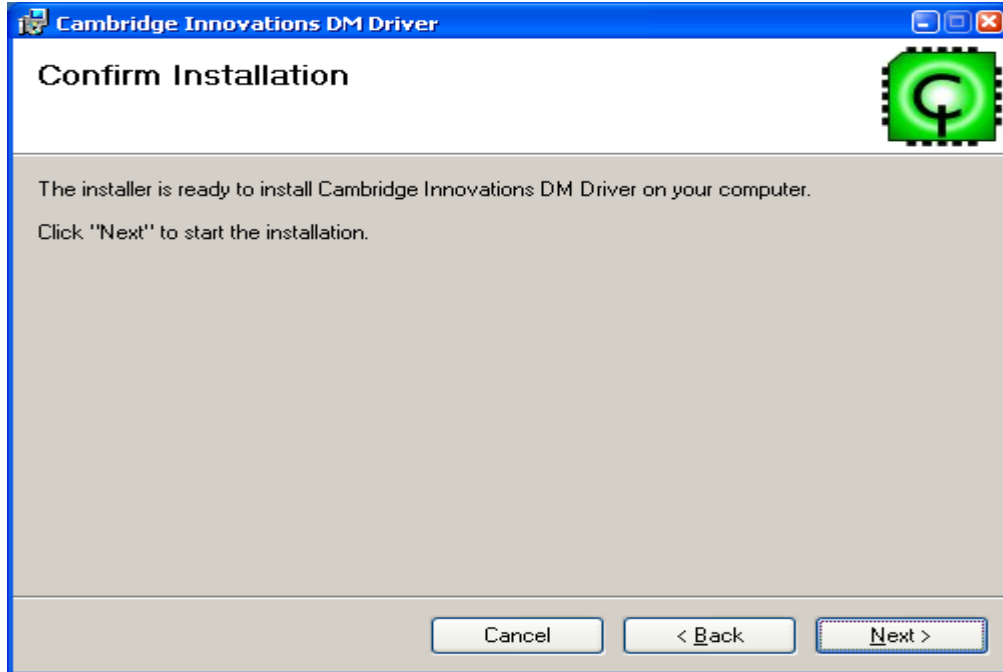
Windows 7



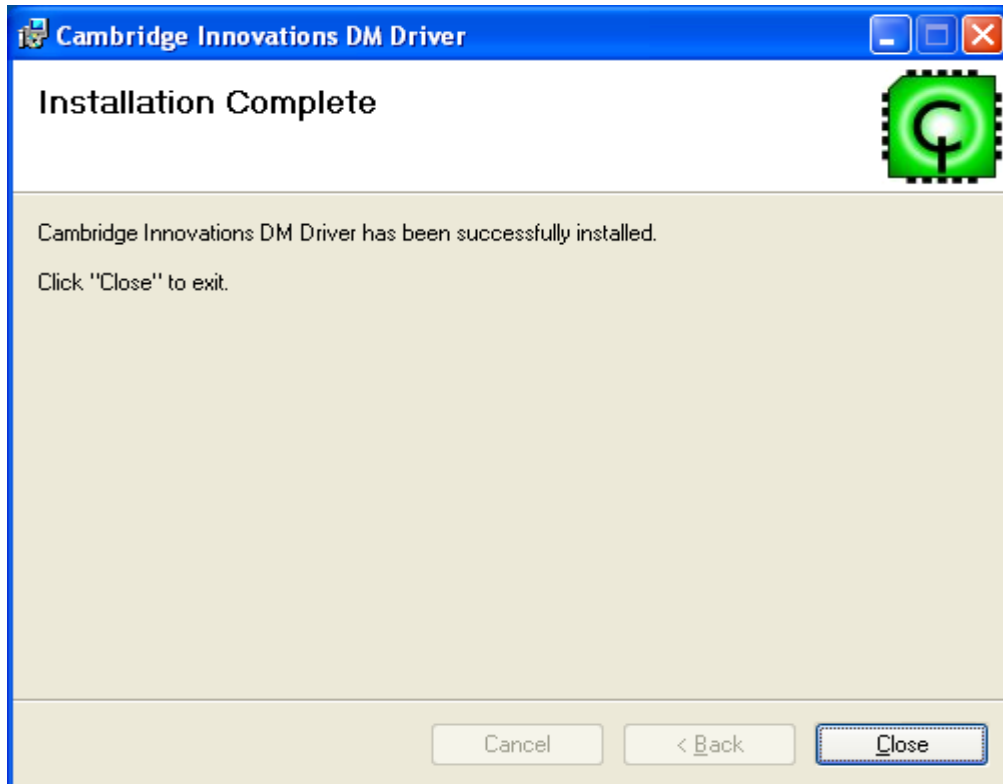
8.1.7 Choose Installation folder and click "Next." (Default folder is C:\ProgramFiles\Cambridge Innovations\)



8.1.8 Click "Next" to start installation.

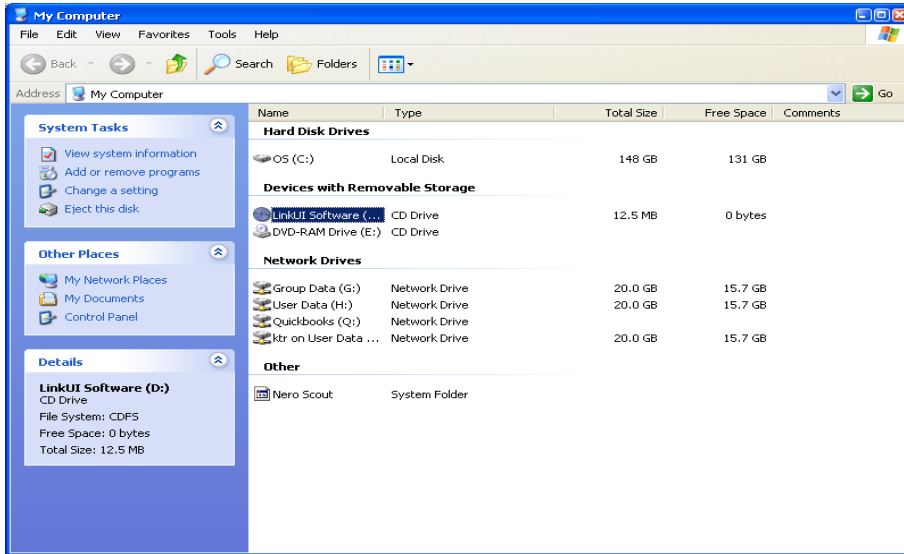


8.1.9 Click "Close" once installation is complete.

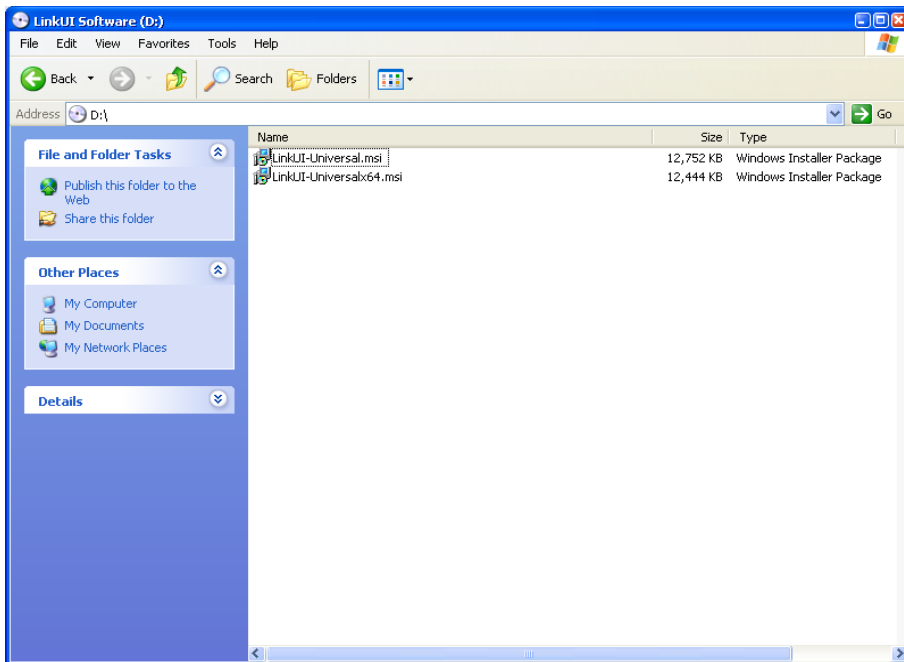


8.2 Installation of LinkUI Software for 64-bit Operating Systems.

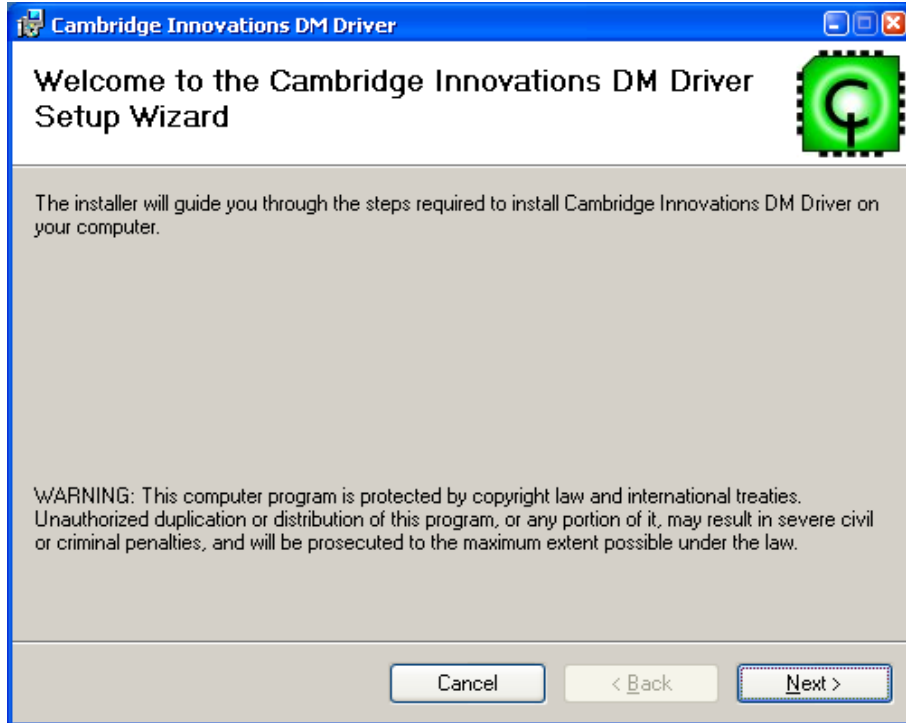
8.2.1 Open “My Computer” on Desktop or Click “Start” then “My Computer” and double click on “LinkUI Software”



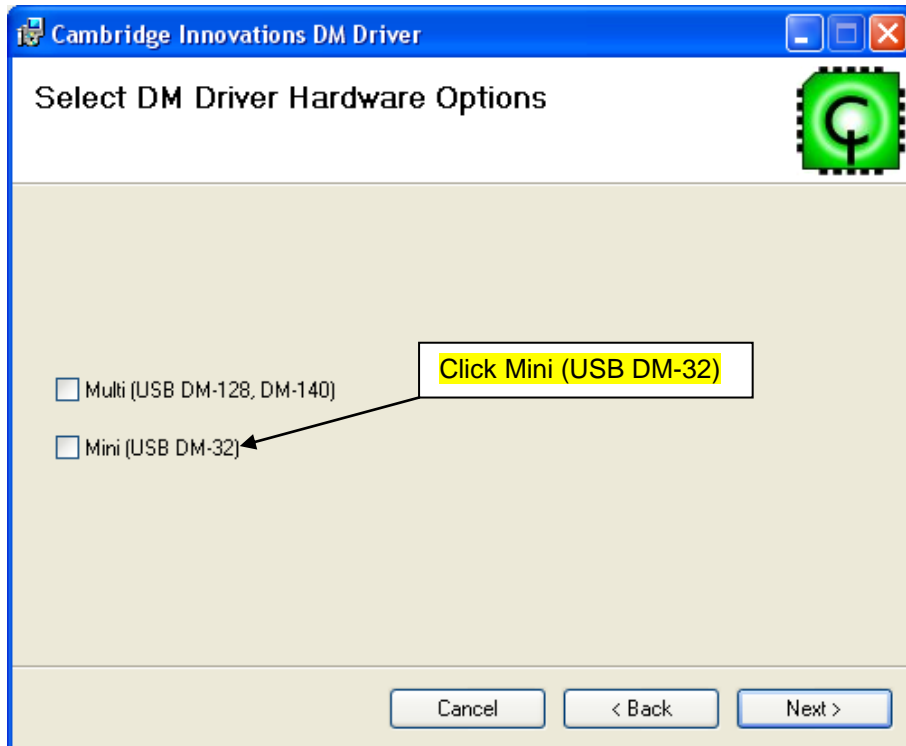
8.2.2 Double click “LinkUI-Universalx64” and the Setup Wizard will execute.



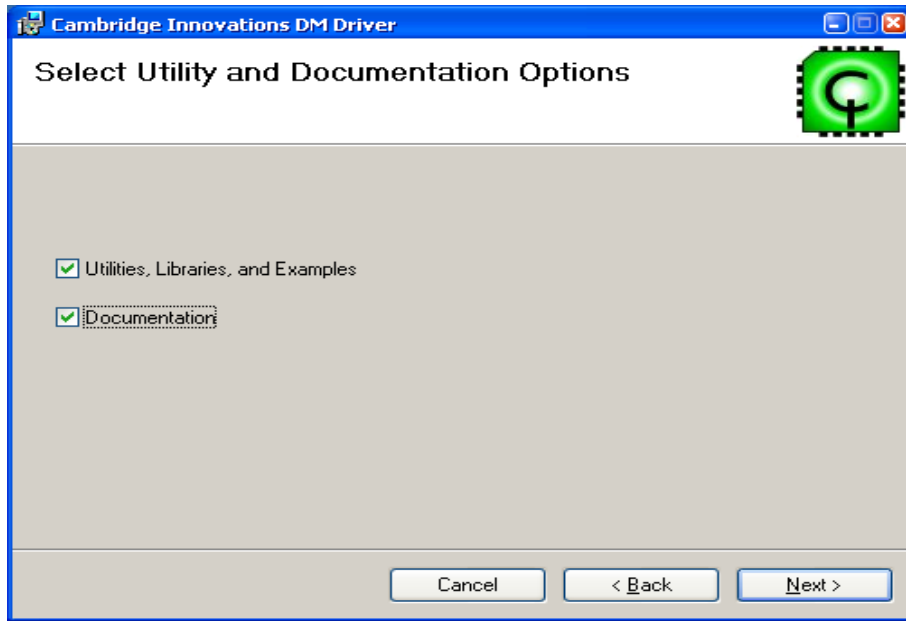
8.2.3 Click “Next” to continue.



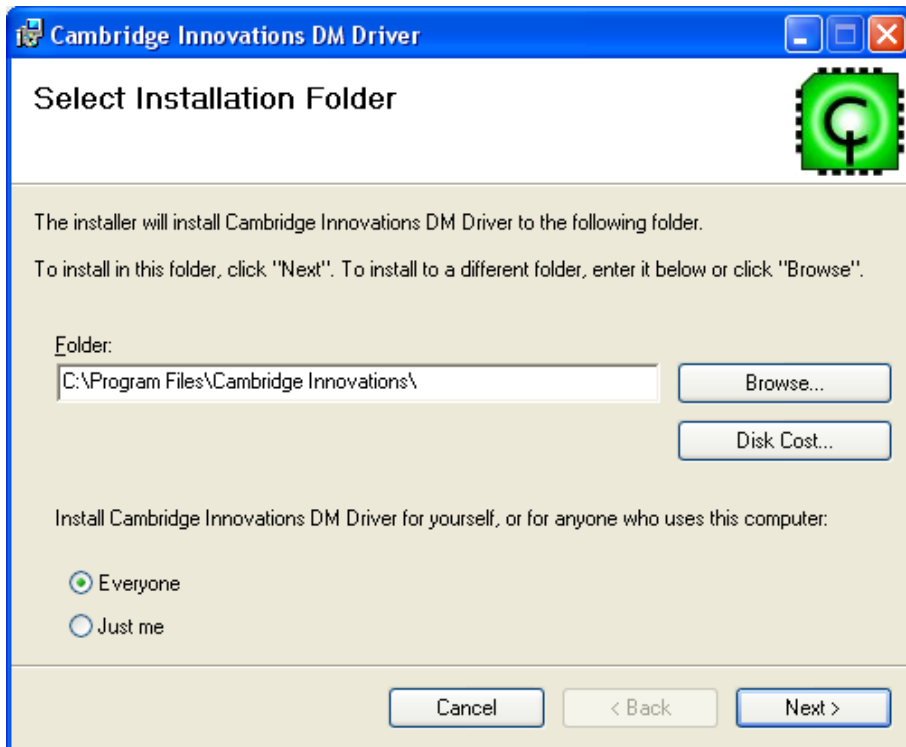
8.2.4 Choose “Mini (USB DM-32)” and click “Next”.



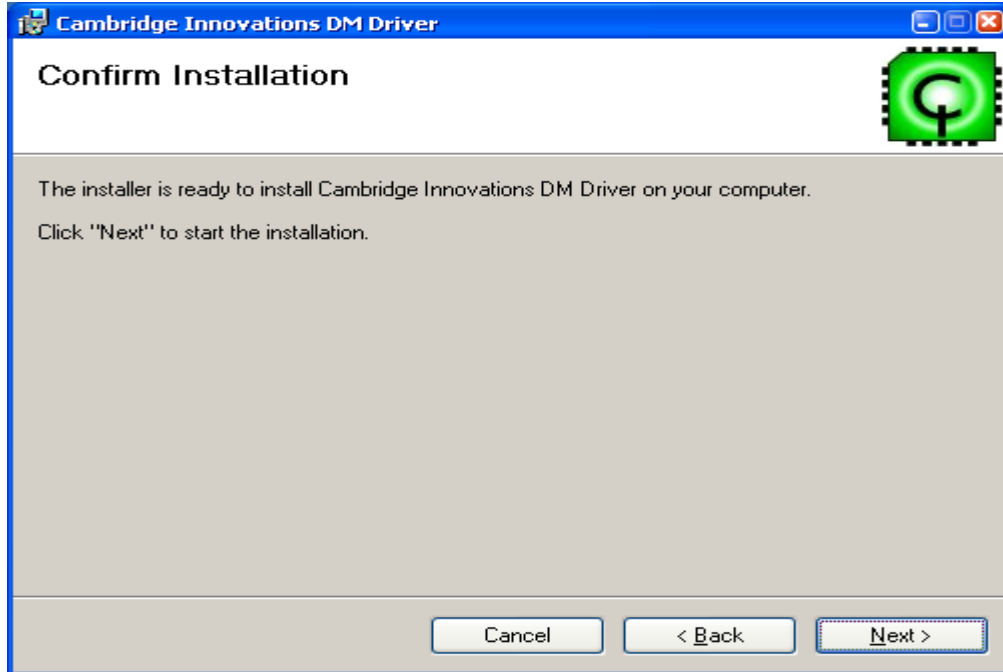
- 8.2.5 Check "Utilities, Libraries, and Examples" and "Documentation." Click "Next" when finished.



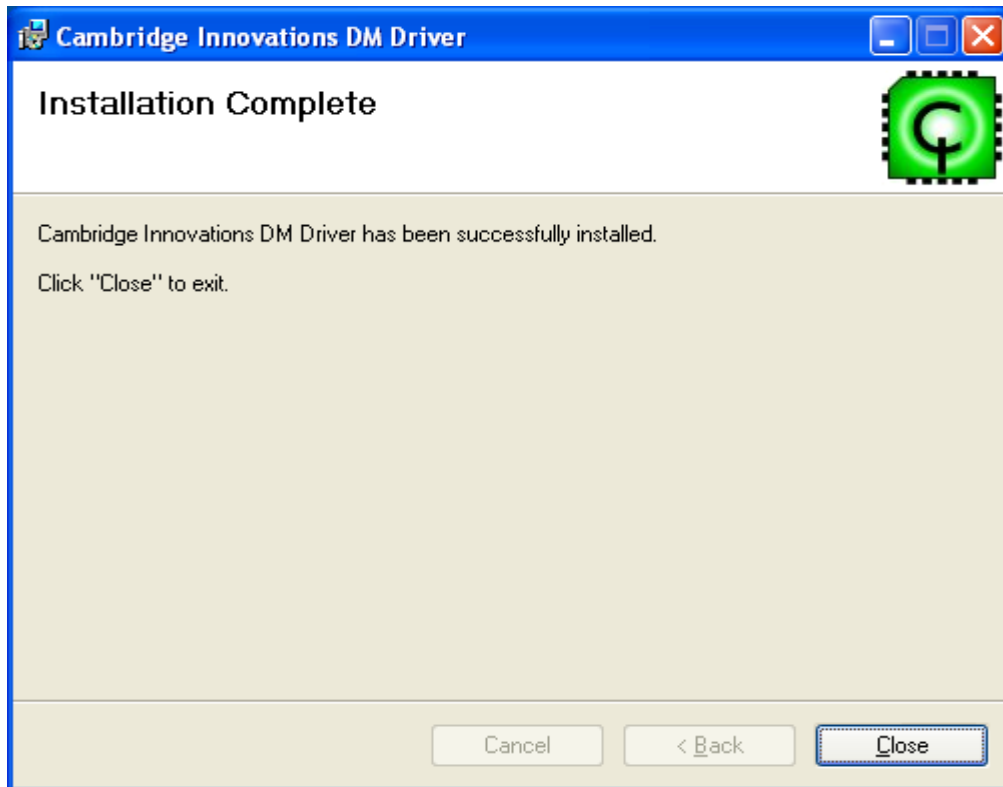
- 8.2.6 Choose Installation folder and click "Next." (Default folder is C:\ProgramFiles\Cambridge Innovations\)



8.2.7 Click "Next" to start installation.



8.2.8 Click "Close" once installation is complete.



9. Appendix B: Cambridge Innovations USB Software Installation Instructions for Windows XP.

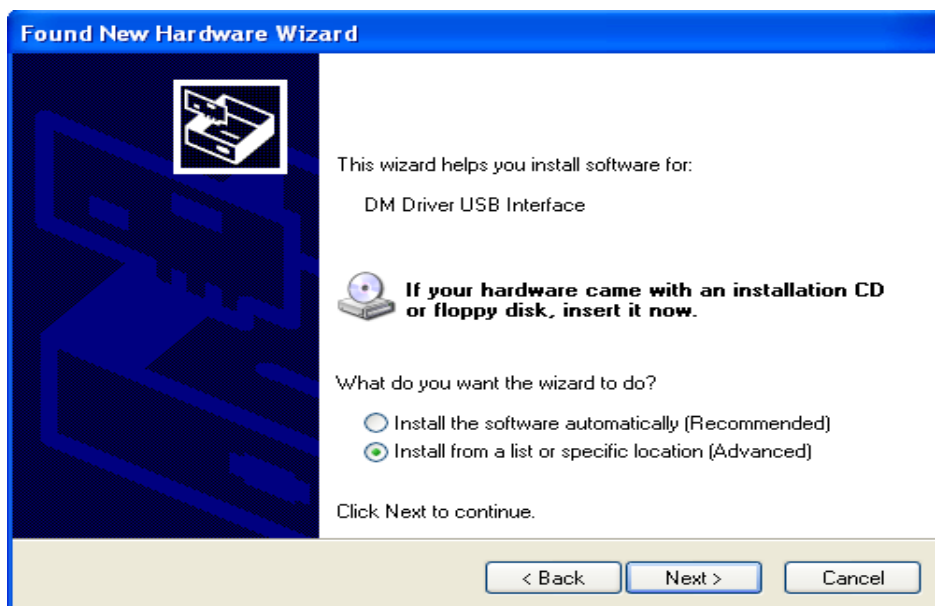
Note: Windows 7 OS will install driver automatically

9.1 After following the instructions in Section 3.2 through step 7 and Appendix 8 carry out the following steps to complete installation process.

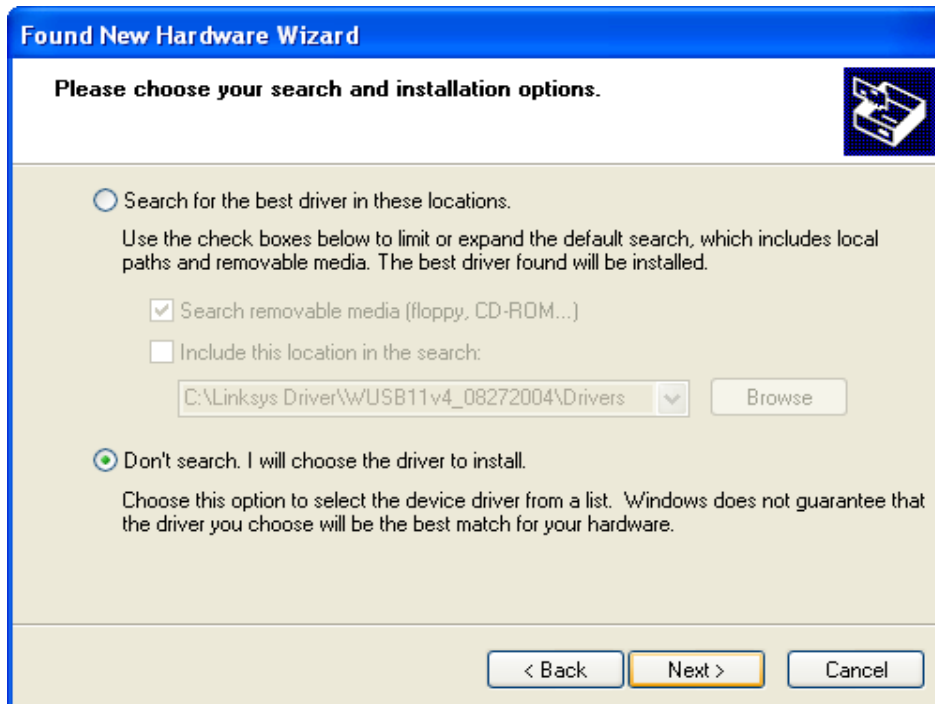
9.1.1 Select the “No, not at this time” radio button and click “Next.”



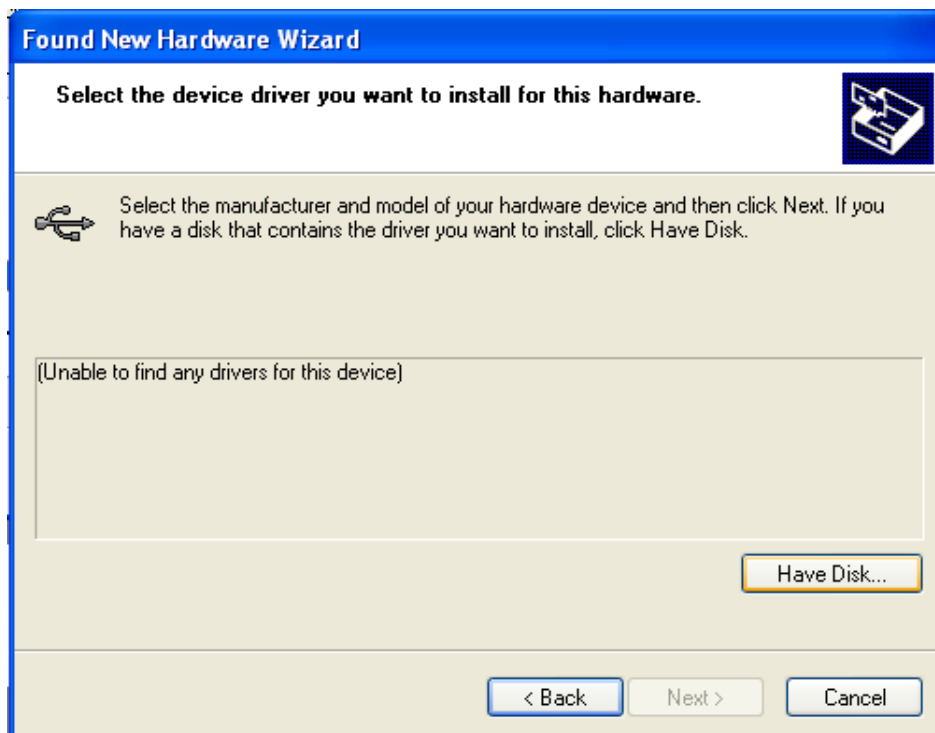
9.1.2 Choose to install from a list or specific location and click “Next.”



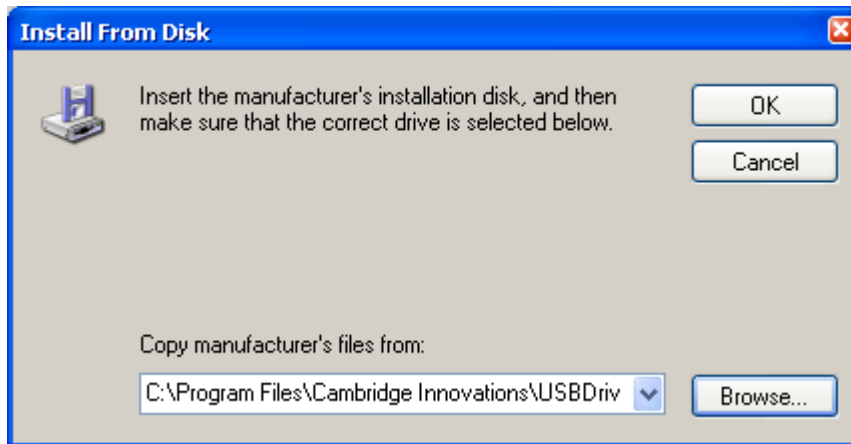
9.1.3 Choose “Don’t to search. I will choose the driver to install”. Click “Next.”



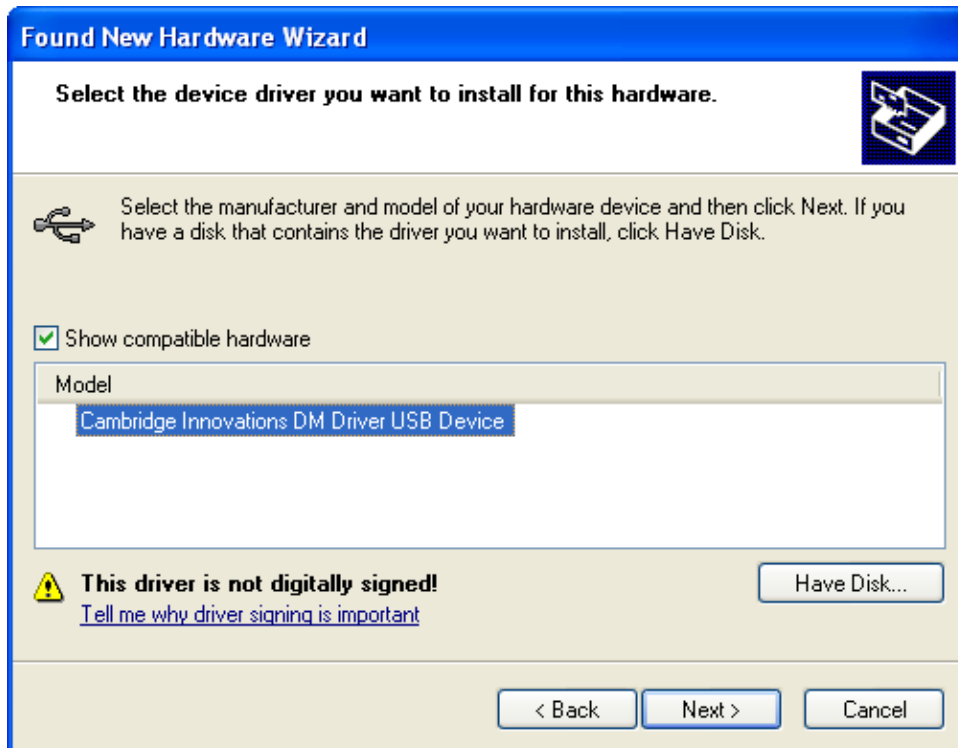
9.1.4 Click “Have Disk...”



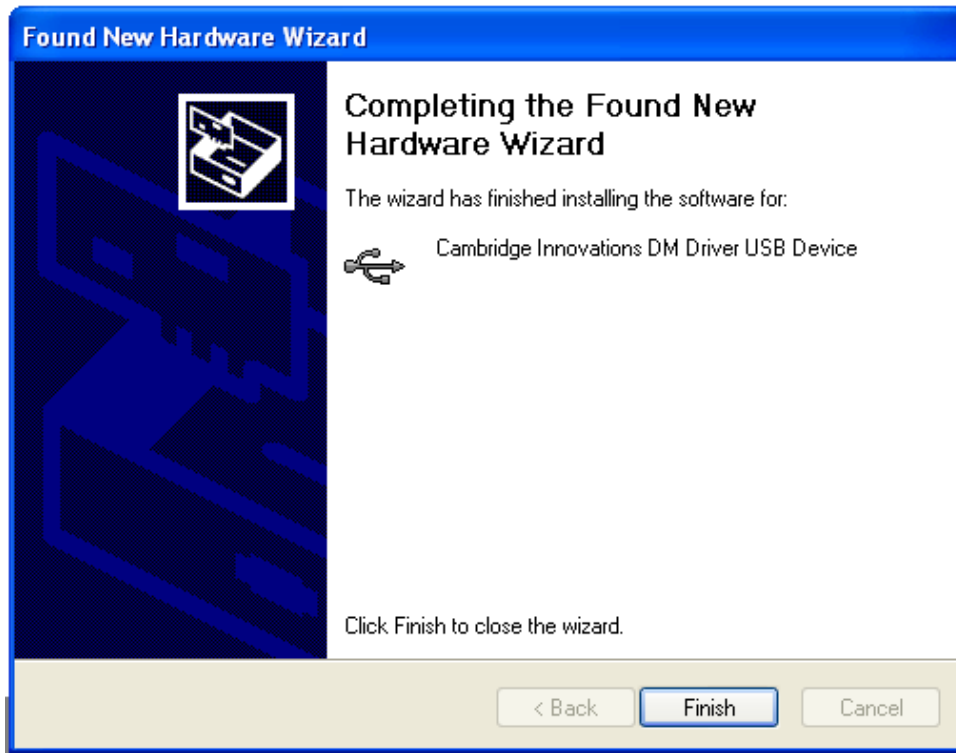
9.1.5 Enter "C:\Program Files\Cambridge Innovations\USBDrivers"



9.1.6 Ensure that "Cambridge Innovations DM Driver USB Device" is displayed in the Model list. Click "Next."



9.1.7 Click "Finish."



BMC Driver Mapping Information

KiloDM-00
KiloDM-01
KiloDM-02
KiloDM-03
4kDM-00
4xKiloDM-00
MiniDM-00
MultiDM-00
MultiDM-01

Please refer to list for correct mapping, the highlighted row is the mapping for your new deformable mirror. Any other mapping may cause failures or inaccurate results.

10. Other Programs

10.1 Mini-DM Sample Program

Sample Program Location:

For 32-bit OS: C:\Program Files\Cambridge Innovations\Examples\UsbExMini\UsbExMini\Win32\Release
 For Windows 7 64-bit OS:
 C:\Program Files\Cambridge Innovations\Examples\UsbExMini\UsbExMini\x64\Release

Sample Code:

Sample Source Code that shows how to implement the USB DLL Libraries in C++.

```
// UsbExMini.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

// define a macro for short-hand error processing
#define CheckHr(s) if (FAILED(hr)){printf(s);return 0;}

// main entry point of console program
int _tmain(int argc, _TCHAR* argv[])
{
    // Initializes the COM library on the current thread and
    // identifies the concurrency model as single-thread apartment.
    // Applications must initialize the COM library before they
    // can call COM library functions
    CoInitialize(NULL);

    long lCurDev = -1; // current USB device index: -1 means no devices
    long lStatus = 0; // CIUsbLib return status
    CComPtr<IHostDrv> pIHostDrv; // CIUsbLib COM Pointer

    // Creates a single uninitialized object of the class associated with
    // a specified CLSID=__uuidof(CHostDrv)
    // This object will be found on the system if CIUsbLib.dll is
    // registered via regsvr32.exe
    HRESULT hr = CoCreateInstance(__uuidof(CHostDrv), NULL,
    CLSCTX_INPROC, __uuidof(IHostDrv), (LPVOID *) &pIHostDrv);
    if (FAILED(hr) && pIHostDrv == NULL)
    {
        printf("Unknown error creating CHostDrv object (CIUsbLib.dll)\n");
        return 0;
    }
    else if(hr == REGDB_E_CLASSNOTREG)
    {
        printf("The CHostDrv class is not registered.\nUse regsvr32.exe
        to register CIUsbLib.dll\n");
        return 0;
    }

    // Check for USB devices supported by the CIUsbLib
```



```

// The array lDevices is set by CIUsb_GetAvailableDevices to indicate
// which devices are present in the system
// In order to recognize MINI DM devices, {CiGenUSB.sys,
// CiGenUSB.inf} need to be installed properly
long lDevices[MAX_USB_DEVICES] = {-1};
hr = pIHostDrv->CIUsb_GetAvailableDevices(lDevices,
sizeof(lDevices)/sizeof(long), &lStatus);
CheckHr("Failure to get available USB devices.\n");

// loop through devices found
for (int i=0; i<MAX_USB_DEVICES; i++)
{
    // check for device indices not equal to -1
    if (lDevices[i] != -1)
    {
        // if we have any present, check specifically for MINI
        // via CIUsb_STATUS_DEVICENAME
        char cDevName[4096] = {0};
        hr = pIHostDrv->CIUsb_GetStatus(lCurDev,
CIUsb_STATUS_DEVICENAME, (long *) cDevName);
        CheckHr("Failure to get available USB device name.\n");

        // check the device for MINI signature
        bool fFoundMini = (strstr(cDevName, USB_DEVNAME)!=NULL);
        if (fFoundMini)
        {
            // record device index
            lCurDev = i;
            // report devices present
            printf("Found: %s\n", cDevName);
            // bail after finding first device (simplest
            // method)
            break;
        }
    }
}

// if lCurDev is still -1, we found none
if (lCurDev == -1)
{
    printf("No Mini DM devices were found.\n");
    return 0;
}

// CIUsb_CONTROL_MINI_MODE must be first to setup CIUsbLib properly
hr = pIHostDrv->CIUsb_SetControl(lCurDev, CIUsb_CONTROL_MINI_MODE,
&lStatus);
CheckHr("Failure to set device MINI mode.\n");

// reset the hardware: control signal FRESET is active low
hr = pIHostDrv->CIUsb_SetControl(lCurDev,
CIUsb_CONTROL_DEASSERT_FRESET, &lStatus);
CheckHr("Failure to deassert MINI hardware reset control.\n");
hr = pIHostDrv->CIUsb_SetControl(lCurDev,
CIUsb_CONTROL_ASSERT_FRESET, &lStatus);
CheckHr("Failure to assert MINI hardware reset control.\n");

// assert high voltage enable

```

```

// CIUsb_CONTROL_ASSERT_HV_ENAB also sequences
// CIUsb_CONTROL_DEASSERT_IV_SHDN for MINI
hr = pIHostDrv->CIUsb_SetControl(lCurDev,
CIUsb_CONTROL_ASSERT_HV_ENAB, &lStatus);
CheckHr("Failure to enable MINI hardware high voltage enable.\n");

////////////////////////////////////
// The following is the start of an example application sequence.
// Five actuators will be poked using the index and value arrays.

USHORT sActData [NUM_ACTUATORS] = {0x0000};
// unmapped actuator data for sending to the DM
USHORT sMapData [NUM_ACTUATORS] = {0x0000};
// mapped actuator data for sending to the DM

int i32TestMap[NUM_ACTUATORS] = // example actuator map
{ 0, 1, 2, 3, 4, 5, 6, 7,
  8, 9,10,11,12,13,14,15,
  16,17,18,19,20,21,22,23,
  24,25,26,27,28,29,30,31};

// iActIndex: actuator index (raster=unmapped)
// sActValues: values to set each actuator
#define NUM_TEST_POKES 5
int iActIndex [NUM_TEST_POKES] = {2, 5, 19, 25, 30};
USHORT sActValues [NUM_TEST_POKES] = {0x8000, 0x8000, 0x8000,
0x8000, 0x8000};

for (int i=0; i<NUM_TEST_POKES; i++)
{
    if (iActIndex[i] < 0 || iActIndex[i] >= NUM_ACTUATORS)
    {
        printf("Actuator index %d is out of range... skipping to
next poke.\n", iActIndex[i]);
        continue;
    }

    // modify the actuator data at index iActIndex[i] with value
    // sActValues[i]
    sActData[iActIndex[i]] = sActValues[i];

    // copy our actuator data into the mapped buffer;
    // use the actuator map to re-order the data into the proper
    // sequence
    for( int j=0; j<NUM_ACTUATORS; j++ )
        sMapData[j] = sActData[i32TestMap[j]];

    // send the actuator data to the DM synchronously
    hr = pIHostDrv->CIUsb_StepFrameData(lCurDev, (UCHAR *)
sMapData, NUM_ACTUATORS*sizeof(short), &lStatus);
    CheckHr("Failure to send MINI frame data.\n");

    // check for framing errors
    if (lStatus == H_DEVICE_NOT_FOUND)
    {
        printf("Framing error: device not found");
        return 0;
    }
}

```

January 7, 2011

```
else
if (lStatus == H_DEVICE_TIMEOUT)
{
    printf("Framing error: device timeout");
    return 0;
}

// print the loop status
printf("Output frame %d: Actuator Index [%04d] = 0x%04x\n", i,
iActIndex[i], sActValues[i]);
}

return 0;
}
```

10.2 MatLab Script Program (Version 2.0)

Found on the Software CD that came with system

Introduction:

The functions described below provide control of the BMC Mini-DM driver technology using Matlab. They are known as “mex” functions (file extension *.mexw32) and are used to interface functionality developed in the C/C++ language with the Matlab environment. The functions were created using Matlab version 7.4.0 (R2007a). Older releases may not be compatible, as Matlab has updated this functionality in recent years. The functions should be used in the same manner (syntax) as functions defined in Matlab: [return_variables] = function_name(input_variables). They require that the Mini-DM driver software is installed on the controlling system, and that the CIUsbLib.dll is registered in the OS. The “install.bat” file supplied can be used for DLL registration.

Function descriptions:

OPEN_miniDM.mexw32

Calling syntax: [error_code driver_info] = OPEN_miniDM(mapping_ID);

Purpose: Used to open and initialize the Mini-DM driver electronics. It should be used at the beginning of Matlab scripts that use the driver functions. It enables the driver’s high voltage amplifier and initializes the USB connection. Handles to the USB connection are returned in the structure “driver_info”, which is required for the function calls below.

Input variables:

“mapping_ID” – Scalar value identifying the Mini-DM system in use, such that the correct DM actuators are mapped to the correct “actuator_amplitude” entries in the “UPDATE_miniDM” function call. Please contact BMC if you would like to verify the version of your driver hardware. Currently supported mappings:

- 0 – No mapping used; Indices of “actuator_amplitudes” vector correspond to driver channels
- 1 – Generation 1 MiniDM mirror mount board “MiniDM -00”.

Return variables:

“error_code” – Scalar value used for error handling

- 0 = no error
- 1 = DLL not registered, or TLB file not matching DLL, or can't instance COM object
- 2 = Unknown error
- 4 = Data send to driver failed
- 5 = Driver not connected

“driver_info” – Structure containing 3 fields {"USB_ID","USB_pointer","mapping_ID"} that are used for storing USB handles for successive DM update function calls (“UPDATE_miniDM”, described below)

UPDATE_miniDM.mexw32

Calling syntax: [error_code] = UPDATE_miniDM (driver_info, actuator_amplitudes);

Purpose: Updates voltages output to DM actuators as specified by the “actuator_amplitudes” array. All actuators voltages are refreshed in each function call. DM frame rates are on the order of 3kHz.

Input variables:

“actuator_amplitudes” – 36 element vector of actuator voltages in units of percent max driver voltage. Valid amplitudes values must be between 0 and 100, corresponding to 0 volts and max voltage, respectively, which varies from system to system. Input values greater than 100 are set to 100 percent, and values less than 0 are set to 0 percent. Precision exceeding 2¹⁴ bits is disregarded. The “actuator_amplitudes” vector index corresponds to the actuator for which it controls, as illustrated in Figure 1 below. Index 1 corresponds to actuator #1, index 2 to actuator #2, etc. The corner actuators of the DM (#s 1, 6, 31 and 36) are inactive, but included in the “actuator_amplitudes” vector for simplified Matlab reshaping and data visualization of the array.

“driver_info” – Structure containing 3 fields {"USB_ID","USB_pointer","mapping_ID"}, which is returned by the "OPEN_miniDM " function call. The values of this structure should not be changed. Any changes will result in the loss of the USB communication settings.

Return variables:

“error_code” – scalar value used for error handling
0 = no error
-4 = Data send to driver failed
-6 = "USB_ID" and "USB_pointer" values unrecognized

CLOSE_miniDM.mexw32

Calling syntax: [error_code] = CLOSE_miniDM (driver_info);

Purpose: Used to close the Mini-DM driver electronics. It should be used at the end of Matlab scripts that call the above driver functions. It disables the driver’s high voltage amplifier and frees the USB connection. Handles to the USB connection are changed in the “driver_info” structure.

Input variables:

“driver_info” – Structure containing 3 fields {"USB_ID","USB_pointer","mapping_ID"} that are used for storing USB handles for successive DM update function calls ("UPDATE_miniDM ", described above)

Return variables:

“error_code” – Scalar value used for error handling
0 = no error
-6 = "USB_ID" and "USB_pointer" values unrecognized

Example Matlab script:

```
% Define two amplitude control arrays
num_actuators = 36;
amplitudes1 = zeros(num_actuators,1);
amplitudes2 = amplitudes1 + 50;

% Open and initialize MiniDM driver USB connection
mapping_ID = 1;
[error_code, driver_info] = OPEN_miniDM(mapping_ID);
```

```

% Alternate driver output between 0 and 50 at ~3kHz
for m = 1:2000
    UPDATE_miniDM(driver_info, amplitudes2);
    UPDATE_miniDM(driver_info, amplitudes1);
end

% Disable and close MiniDM driver USB connection
error_code = CLOSE_miniDM(driver_info);

```

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

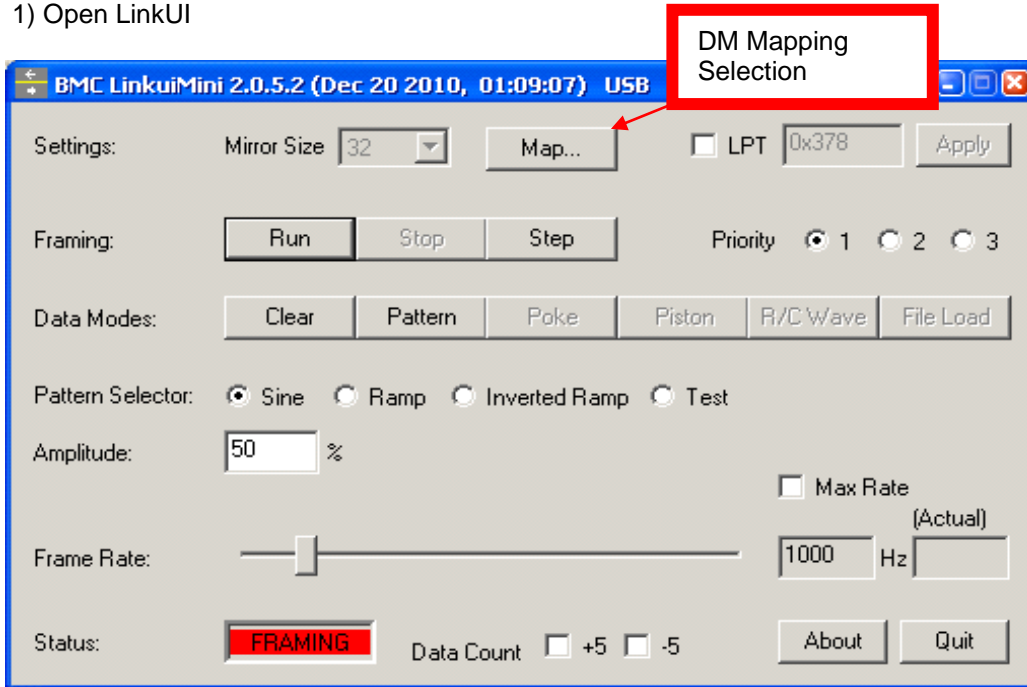
Figure 11: Mini-DM actuator numbering, as used by Matlab control software. Actuators 1, 6, 31 and 36 are inactive, yet included in control functions for easy array manipulation in Matlab.

10.3 DM Flat Map: Loading Voltage Map to Flatten DM

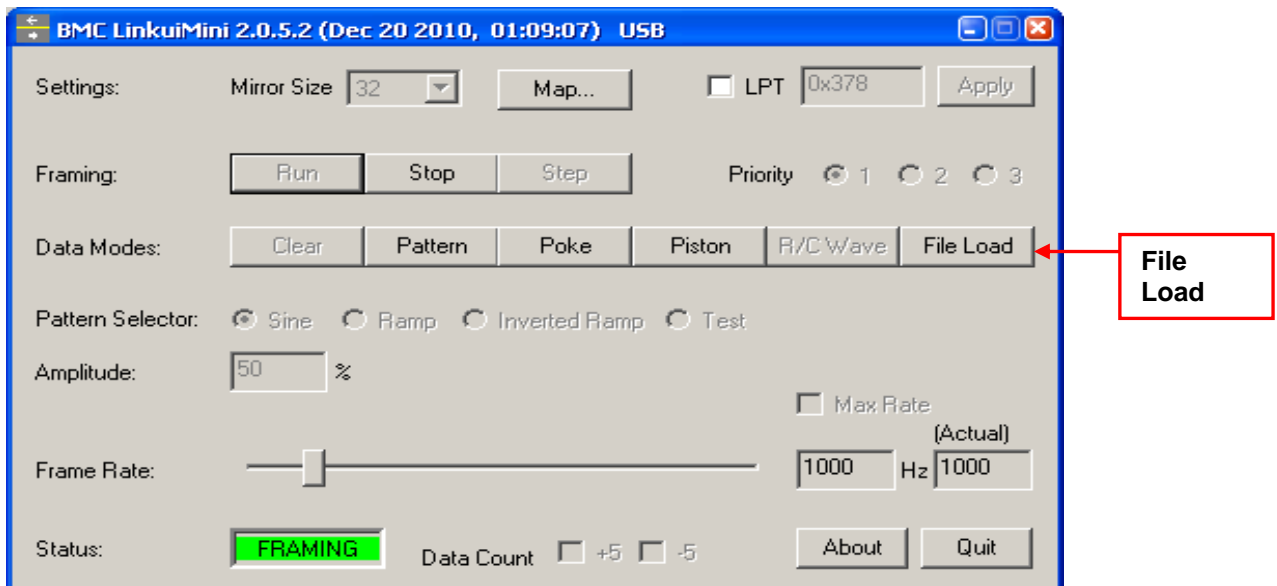
*****Note: For CDM-Type Devices Only******

The unpowered surface figure of the DM has low order curvature (See Figure 12 on page 37). A voltage map capable of flattening and pistoning the DM to half actuator stroke (50% bias) has been supplied. This voltage map should be used during optical alignment. It is located on the LinkUI Software CD that was provided with the DM in folder “<DM serial number> DM Flat Map”. It consists of an array of 16 bit hexadecimal values. Please follow the directions below to load this file.

1) Open LinkUI

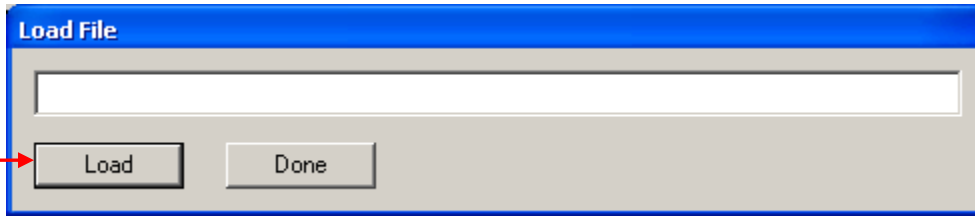


2) Click “Run” and then Select “File Load”.

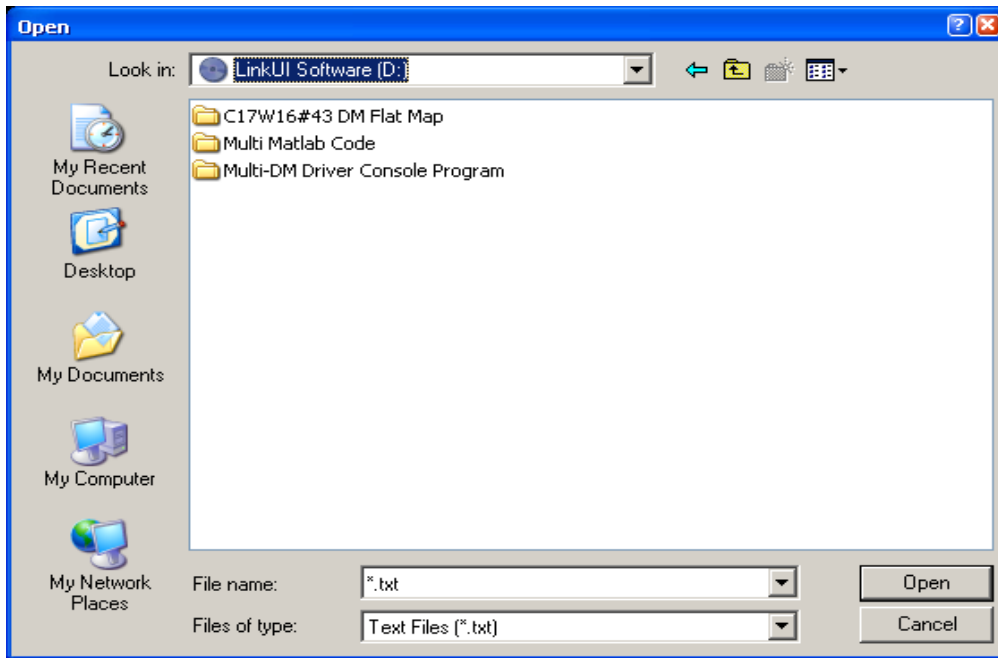


3) Install LinkUI CD into CD-Rom or if folders have been saved to your computer make note of where these files are located.

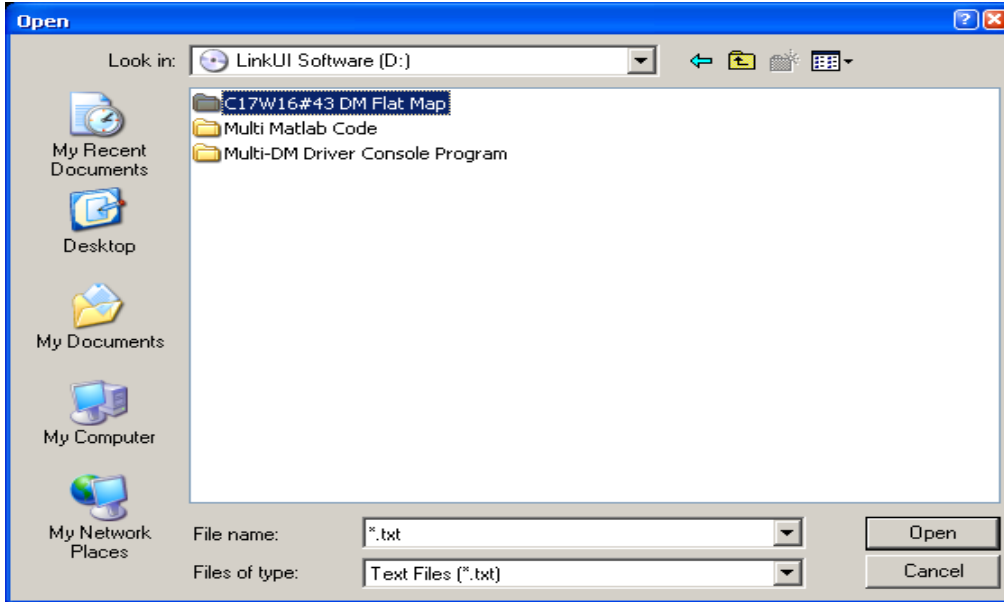
4) Click "Load" button in Load File window



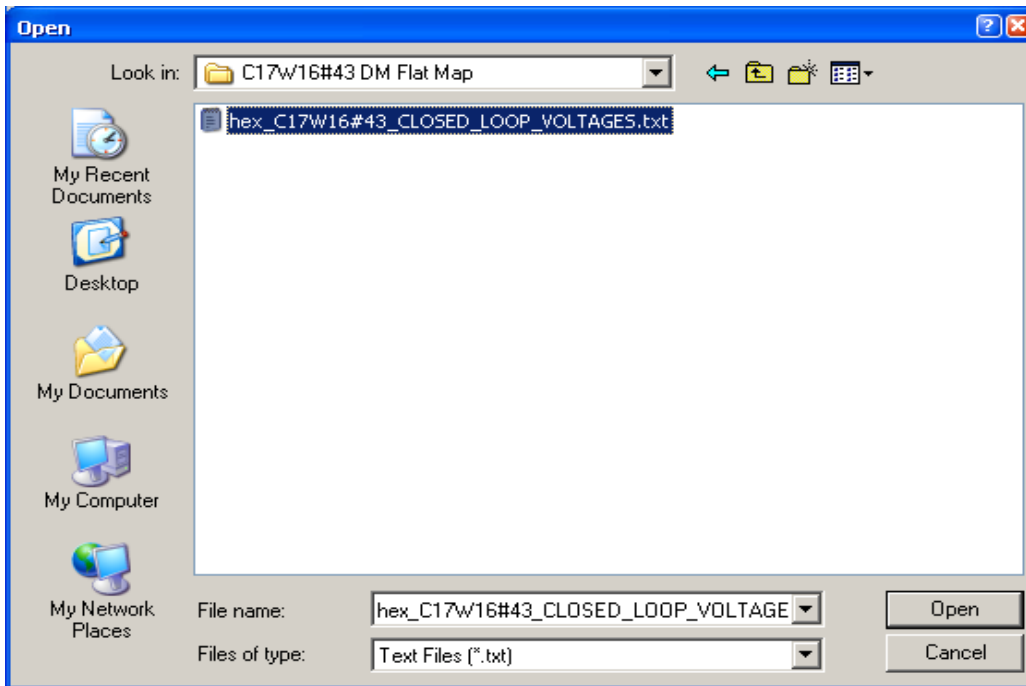
5) In Load window navigate to CD-Rom Drive or where LinkUI Software CD Files are located.



6) Click on Folder titled "<DM serial number> DM Flat Map".



7) Select txt file "Hex_<DM Serial Number>_CLOSED_LOOP_VOLTAGES.txt" as shown in example below. Click "Open".



8) A new window will pop up stating Operation Successful.



9) The DM is now flat and pistoned to 50% of the actuator stroke, as shown in Figure 13.

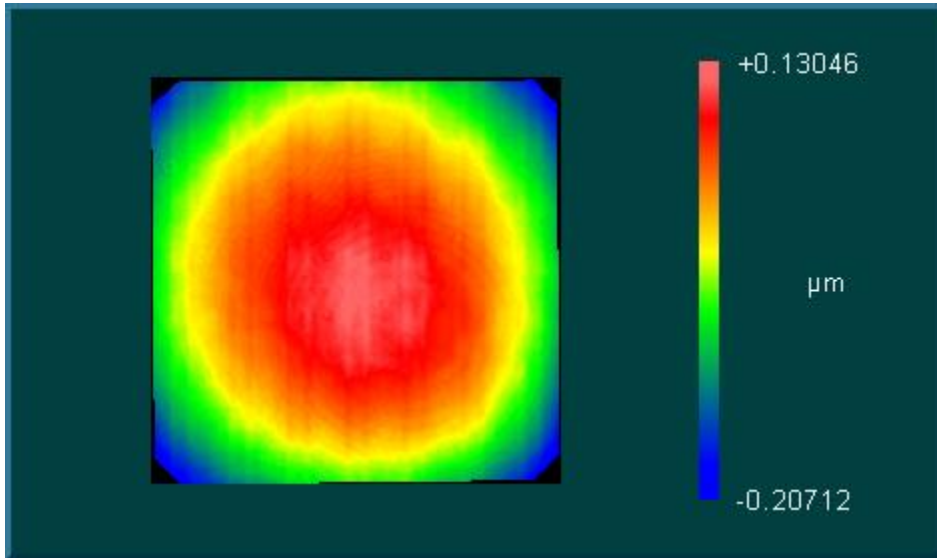


Figure 12: Example of a DM in unpowered stroke

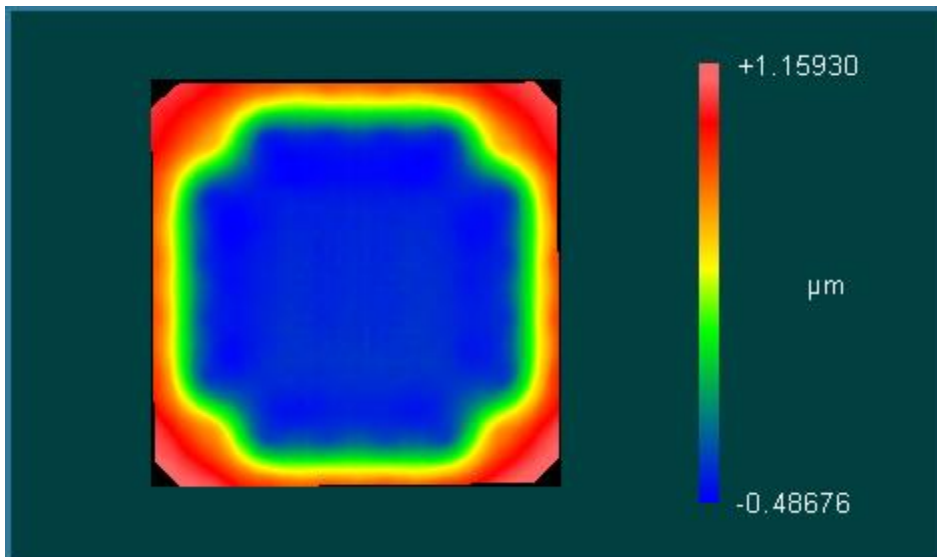


Figure 13: Example of a DM at half actuator stroke (50% bias)

10) Once finished click “Done” and Quit LinkUI

