

MCT U.I. Driver Reference Manual

Motor Control Technologies; LLC

www.mocontech.com

1. The MCTUI Driver	2
2. MCT Hardware Methods.....	2
2.1.1. BuildDataPacket()	2
3. Third Party Hardware Methods.....	5
3.1. LabJack Data Acquisition Hardware	5
3.1.1. LJAnalogOut().....	5
3.1.2. LJComConfig().....	5
3.1.3. LJConnect()	6
3.1.4. LJCurrent().....	7
3.1.5. LJDigLine()	8
3.1.6. LJPWMConfig()	9
3.1.7. LJPWMUpdate()	10
3.1.8. LJSendData().....	10
4. Utility Methods.....	11
4.1.1. Hex2Float()	11
4.1.2. Float2Hex()	12
5. Revision History	12

1. The MCTUI Driver

The MCTUI Driver is high-level interface built on Microsoft's .NET framework. It is designed to ease software development by providing a standardized set of tools for interfacing with MCT hardware. The MCTUI handles all low-level function calls and formatting, exposing a simple set of methods for controlling our motor drives. Figure 1 depicts a block diagram for the MCTUI driver.

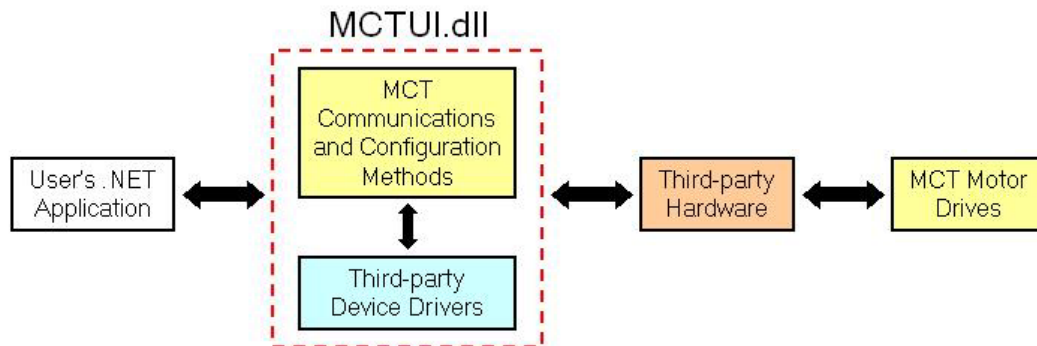


Figure 1. MCTUI.dll block diagram.

2. MCT Hardware Methods

2.1.1. BuildDataPacket()

Constructs data packets used to communicate with MCT hardware. BuildDataPacket converts double precision user data into a properly formatted multi-byte data packet.

Method Definition:

```
void BuildDataPacket(byte MCTDevice,  
                    ref byte ArraySize,  
                    double[] UserData,  
                    byte[] XmitData)
```

Inputs:

MCTDevice – MCT Hardware device descriptor.
0 = DCA-10
1 = IDCA-10

ArraySize – Number of populated elements in the UserData array.

UserData – Double precision data array containing data for transmission. See below for various method formats. User data packets are further defined in section 2.1.1.1.

XmitData – Data array to store transmission packet. XmitData must be at least 50 elements to store all returned data.

Outputs:

- ArraySize – Number of populated elements in the XmitData array.
- XmitData – n-element data array formatted for MCT hardware. The XmitData array is automatically formatted for the device specified in the MCTDevice variable. The data packet CRC value is automatically appended to the last two data elements.

2.1.1.1. User Data Packet Definitions

Access Flash (3 Elements)

- Element 0 – Device address (0-127)
- Element 1 – Software reset function code
- Element 3 – Flash Option (see product documentation)

Configure IDCA-10 (15 Elements)

- Element 0 – Device address
- Element 1 – Software reset function code
- Element 2 – Mode value
- Element 3 – Bridge configuration value
- Element 4 – Duty cycle
- Element 5 – Frequency scale factor
- Element 6 – PID period
- Element 7 – Encoder resolution (counts / revolution)
- Element 8 – Kp
- Element 9 – Ki
- Element 10 –Kd
- Element 11 – Dead-band
- Element 12 – Approach speed
- Element 13 – Maximum speed
- Element 14 – Encoder scale value

Retrieve Log Data (3 Elements)

- Element 0 – Device address (0-127)
- Element 1 – Software reset function code
- Element 2 – Log data subset (see product documentation)

Read Memory Registers(8)

- Element 0 – Device address
- Element 1 – Software reset function code
- Element 2 – Register Count (1-5)
- Element 3 – Register 1 Address
- Element 4 – Register 2 Address
- Element 5 – Register 3 Address
- Element 6 – Register 4 Address
- Element 7 – Register 5 Address

Note: Set unused register addresses to zero.

Software Reset – (3 Elements)

- Element 0 – Device address
- Element 1 – Software reset function code
- Element 2 – Reset option (see product documentation)

Write Memory Registers (13 Elements)

- Element 0 – Device address
- Element 1 – Software reset function code
- Element 2 – Register Count (1-5)
- Element 3 – Register 1 address
- Element 4 – Register 1 value
- Element 5 – Register 2 address
- Element 6 – Register 2 value
- Element 7 – Register 3 address
- Element 8 – Register 3 value
- Element 9 – Register 4 address
- Element 10 – Register 4 value
- Element 11 – Register 5 address
- Element 12 – Register 5 value

Note: Set unused register addresses and values to zero.

Write to Instruction Buffer (7 Elements)

- Element 0 – Device address
- Element 1 – Software reset function code
- Element 2 – MCT Op-code
- Element 3 – Data
- Element 4 – Data
- Element 5 – Data
- Element 6 – Data

3. Third Party Hardware Methods

3.1. LabJack Data Acquisition Hardware

3.1.1. LJAnalogOut()

Sets an LJ DAC channel to the specified voltage.

Method Definition:

```
void LJAnalogOut(int LJHandle,  
                 byte LJDevice,  
                 byte Chan,  
                 double Volt)
```

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the HOST PC when first opened.

LJDevice – LJ Device descriptor

0 = U12

1 = U3

2 = U6

3 = UE9

Chan – LJ DAC channel number.

Volt – Output voltage measured in volts.

Outputs:

NA

3.1.2. LJComConfig()

Configures the LJ SPI and I²C buses.

Method Definition:

```
void LJComConfig(int LJHandle,  
                 byte LJDevice,  
                 bool SPI,  
                 byte[] ConArray)
```

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the HOST PC when first opened.

LJDevice – LJ Device descriptor

- 0 = U12
- 1 = U3
- 2 = U6
- 3 = UE9

Chan – Bus configuration

- True = SPI
- False = I2C

ConArray – Configuration array used to specify bus speed and associated output channels. Elements 1 through 4 are used to map serial lines to specific digital output channels.

Element	SPI	I2C
0	Clock Factor	Clock Factor
1	MISO	SCL
2	MOSI	SDA
3	SCK	Not used
4	NSS	Not used
5	Mode	Not used

The U12 only supports SPI communications. The Clock Factor, MISO, MOSI, and SCK values in ConArray are all ignored when configuring the U12 for SPI. The U12 internally sets the bus speed to 10 kHz and outputs the MOSI, MISO, and SCK lines on D13, D14, and D15, respectively. The NSS line may be configured on any one of the other available digital lines (D0-D12).

Note: All MCT devices use Mode B for SPI communications. Always pass a 1 for Mode when communicating with MCT hardware.

Outputs:

NA

3.1.3. LJConnect()

Opens a connection to a specified LabJack device.

Method Definition

```
static void LJConnect(ref int LJHandle,  
                    byte LJDevice,  
                    bool USB,  
                    bool FirstFound,  
                    String Address)
```

Inputs:

LJHandle – Variable used to store the LJ hardware handle assigned by the host PC.

LJDevice – LJDevice Descriptor

0 = U12

1 = U3

2 = U6

3 = UE9

ComMeth – Communication method. Only used with the UE9. Set to true for all other devices.

True = USB

False = Ethernet

Address – Ethernet address if connecting via Ethernet (UE9). Device I.D. number if connecting with a specific device. The FirstFound option must be set to false if connecting by means of the device I.D. value.

Outputs:

LJHandle - Hardware handle assigned to the LJ device by the HOST PC when first opened.

3.1.4. LJCurrent()

Reads an analog input channel and converts the voltage to the equivalent current conducting through the bridge. LJCurrent is only used to read bridge on the DCA-10.

Method Definition:

```
void LJCurrent(int LJHandle,  
               byte LJDevice,  
               byte Chan,  
               byte ChanN,  
               bool Diff,  
               ref double Current)
```

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the host PC when first opened.

LJDevice – LJ Device descriptor

0 = U12

1 = U3

2 = U6

3 = UE9

Chan – Analog input channel to read. Chan is the positive input if a differential measurement is made.

ChanN – Negative input channel for differential measurements. Not used in single-ended mode.

Diff – Sets singled-ended or differential mode when reading an analog input.

True = Differential

False = Single-ended

Current – Reference to a variable used to store measured current.

Outputs:

Current – DCA-10 H-bridge current measured in amps.

3.1.5. LJDigLine()

Reads or writes to a digital input/output channel. LJDigLine will only read/write to channels D0-D15 on the LabJack U12.

Method Definition:

```
void LJDigLine(int LJHandle,  
               byte LJDevice,  
               byte Chan,  
               byte Dir,  
               ref byte State)
```

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the host PC when first opened.

LJDevice – LJ Device descriptor

0 = U12

1 = U3

2 = U6

3 = UE9

Chan – Digital channel

Dir – Channel direction
0 = Digital input
nonzero = Digital output

State – Reference to a variable used to store channel state.

Outputs:

State – Digital line state for specified digital line.

3.1.6. LJPWMConfig()

Configures PWM output channel on a specified channel. The digital channel is configured as 8-bit PWM (1000 Hz - 10,000 Hz). LJPWMConfig is only used with the DCA-10.

Method Definition:

```
void LJPWMConfig(int LJHandle,  
                 byte LJDevice,  
                 int Freq,  
                 byte IN2Chan,  
                 ref int ActFreq)
```

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the host PC when first opened.

LJDevice – LJ Device descriptor
0 = U12
1 = U3
2 = U6
3 = UE9

Freq – Command frequency for PWM output.

IN2Chan – Channel connected to the DCA-10's IN2 pin.

ActFreq – Reference to a variable used to store PWM frequency set by hardware.

Outputs:

ActFreq – Variable used to store PWM frequency set by hardware. The actual PWM frequency may vary depending on how

the commanded frequency divides into the LabJack's internal clock.

3.1.7. LJPWMUpdate()

Updates the duty cycle on the PWM output signal. Call LJPWMUpdate after using LJPWMConfig() to configure a PWM output channel. LJPWMConfig is only used with the DCA-10.

Method Definition:

```
void LJPWMUpdate(int LJHandle,  
                 byte LJDevice,  
                 ref byte DC)
```

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the host PC when first opened.

LJDevice – LJ Device descriptor

0 = U12

1 = U3

2 = U6

3 = UE9

DC – Command duty cycle for the PWM output.

Outputs:

DC – The actual duty cycle of the PWM output. Internal clock frequency and rounding may result in a PWM duty cycle different from the commanded value. The actual value used by hardware is returned in the DC variable.

3.1.8. LJSendData()

Transmits a data packet to the MCT hardware using the SPI or I²C bus. LJSendData automatically retrieves the response packet from hardware. Use in conjunction with the BuildDataPacket() method to properly construct and format transmitted data packets.

Method Definition:

```
void LJSendData(int LJHandle,  
               byte LJDevice,  
               bool SPI,
```

byte[] DataArray,
ref byte Bytes)

Inputs:

LJHandle – Hardware handle assigned to the LJ device by the host PC when first opened.

LJDevice – LJ Device descriptor

0 = U12

1 = U3

2 = U6

3 = UE9

ComType – Serial bus protocol used to transmit data packet

True = SPI

False = I²C

DataArray – Data array to transmit to MCT hardware

Bytes – Number of bytes data array being transmitted.

Outputs:

DataArray – Data packet returned by the MCT hardware

Bytes – Number of bytes in the returned data packet.

4. Utility Methods

4.1.1. Hex2Float()

Converts a 4-byte data array representing a floating-point value in IEEE-754 floating-point storage standard (32-bits).

Method Definition:

```
void Hex2Float(ref float FVal,  
               byte[] ByteArray)
```

inputs:

FVal – Reference to variable that will store the floating-point value.

ByteArray = 4-byte IEEE-754 representation of the floating point number.

Element 0 = S E E E E E E E

Element 1 = E M M M M M M M

Element 2 = M M M M M M M M

Element 3 = MMMM MMMM

Note: Each bit in the 4-byte array corresponds to numbers sign, exponent, or mantissa as defined in the IEEE-754 floating-point storage standard.

S = sign, E = Exponent, M = Mantissa

Outputs:

Fval – Converted floating-point value.

4.1.2. Float2Hex()

Converts a floating-point number to a 4-byte representation, according to the IEEE-754 floating point storage standard (32-bits).

Method Definition:

```
void Float2Hex(float FVal,  
               byte[] ByteArray)
```

inputs:

FVal - Floating point number to convert

Outputs:

ByteArray = 4-byte IEEE-754 representation of the floating point number.

Element 0 = SEEE EEEE

Element 1 = EMMM MMMM

Element 2 = MMMM MMMM

Element 3 = MMMM MMMM

Note: Each bit in the 4-byte array corresponds to numbers sign, exponent, or mantissa as defined in the IEEE-754 floating point storage standard.

S = sign, E = Exponent, M = Mantissa

5. Revision History

- Revision 1.0 – Initial Release