CSharp USB-QUAD08 example

This example demonstrates how to program a USB-QUAD08. It configures one channel to read position from a quadrature encoder and another channel to count events using the modulus operator. To do this, it uses the CConfigScan to configure the counter inputs and the CLoad32 function to load the modulus value for the event counter. The program reads the input continuously. This is setup using the CInScan function with the background continuous scan options. The data is saved as an ASCII file compatible with the DASYLab Read Module.

The program also uses the discovery interface to locate and create the device object. Using the discovery interface, the InstaCal program must be installed however, the end user isn't required to run it.

This program is a 32-bit C# Console Application. If you start a new project, you must first add a Dot Net reference to the MccDaq object. The InstaCal installation our Dot Net UL object. Adding the reference is usually accomplished by right clicking the Project [under the Project Explorer] and selecting Add Reference.

The complete project is attached as a zip file at the end of this article.


```
using System;
using MccDaq;
using System.IO;
namespace Quad08
{
   class Program
   {

      public const int CHANCOUNT = 2;
      public const int FIRSTCHANNEL = 0;
      public const int LASTCHANNEL = CHANCOUNT-1;
      public const int FREQ = 100;
      public const int NUMOFPACKETS = 100;
      public const int PACKETSIZE = 100;
      public const int BUFFERSIZE = PACKETSIZE * CHANCOUNT;
      public const int HALFBUFFSIZE = BUFFERSIZE / 2;
      public const string DEVICE = "USB-QUAD08";
```

```
        public static StreamWriter fStream;

        public static MccBoard daq;

        public static MccDaq.CounterMode mode;

        public static MccDaq.ScanOptions opts;

        static void Main(string[] args)
        {
            MccDaq.ErrorInfo RetVal;

            int Rate = FREQ;
            bool ReadLower = true;

            daq = Discover(DEVICE);

            if (daq == null)
            {
                Console.WriteLine("No {0} detected!", DEVICE);
                WaitForKey();
                return;
            }

            IntPtr buffer = MccService.WinBufAlloc32Ex(BUFFERSIZE);

            if (buffer == IntPtr.Zero)
            {
                Console.WriteLine("Bad Handle");
                return;
            }

            //set counter 0 to encoder mode
            mode =
CounterMode.Encoder|CounterMode.EncoderModeBit32|CounterMode.EncoderModeX1;
            IsError( daq.CConfigScan(   0,
                                mode,
                                CounterDebounceTime.Debounce500ns,
                                CounterDebounceMode.TriggerAfterStable,
                                CounterEdgeDetection.RisingEdge,
                                CounterTickSize.Tick208pt3ns,
                                0));
```

```
        //set counter 1 to totalize mode with Modulus operation
        mode = CounterMode.Totalize|CounterMode.Bit32|CounterMode.ModuloNOn;
        IsError( daq.CConfigScan(    1,
                              mode,
                              CounterDebounceTime.Debounce500ns,
                              CounterDebounceMode.TriggerAfterStable,
                              CounterEdgeDetection.RisingEdge,
                              CounterTickSize.Tick20pt83ns,
                              1));

        //load register1 with modulus number
        IsError(daq.CLoad32(CounterRegister.MaxLimitReg1, 2000));

        //begin reading counter channels
        opts =  ScanOptions.Continuous|ScanOptions.Ctr32Bit|ScanOptions.Background;
        IsError(daq.CInScan(FIRSTCHANNEL, LASTCHANNEL, BUFFERSIZE, ref Rate, buffer,
opts));


        //Timer 0 output (DIO6)used as test signal for counter 1
        double tmrRate = 100;
        double tmrDuty = 0.5;
        double tmrDelay = 0.0;
        daq.PulseOutStart(  0,
                        ref tmrRate,
                        ref tmrDuty,
                        0,
                        ref tmrDelay,
                        0,
                        PulseOutOptions.Default);

        fStream = new StreamWriter(@"C:\Users\Public\Documents\DataFile.asc");
        CreateFileHeaders(); //writes basic info to the beginning of the file

        int Count = 0;
        int Index = 0;
        short daqStatus;

        int[] intArray = new int[BUFFERSIZE];

        //Loop until key press
        do
        {
           RetVal = daq.GetStatus(out daqStatus, out Count, out Index,
```

```
FunctionType.CtrFunction);
            if ((Index >= HALFBUFFSIZE) & ReadLower) //check for 50% more data
            {
               //get lower half of buffer
               IsError( MccService.WinBufToArray32(buffer, intArray, 0, HALFBUFFSIZE));

               DisplayData(intArray, HALFBUFFSIZE / CHANCOUNT);
               ReadLower = false; //flag that controls the next read
            }
            else if ((Index < HALFBUFFSIZE) & !ReadLower)
            {
               //get the upper half
               IsError(MccService.WinBufToArray32(buffer, intArray, HALFBUFFSIZE,
HALFBUFFSIZE));

               DisplayData(intArray, HALFBUFFSIZE / CHANCOUNT);
               ReadLower = true;//flag that controls the next read
            }

            if (Console.KeyAvailable == true) break;

         } while ((Count < (PACKETSIZE * NUMOFPACKETS)) || Console.KeyAvailable);


         //flush any buffered data out to disk
         fStream.Close();

         //stop the  acquisition
         RetVal = daq.StopBackground(FunctionType.AiFunction);

         //free up memory
         MccService.WinBufFreeEx(buffer);

         daq.PulseOutStop(0);

         WaitForKey();

      }
      public static int IsError(ErrorInfo e)
      {
         if (e.Value != 0)
         {
            Console.WriteLine(e.Message);
            WaitForKey();
            return 1;
```

```
        }
        return 0;
    }

    /*////////////////////////////////////////////////////////////////////////////*/
    public static MccDaq.MccBoard Discover(string dev)
    {

        //this ignores devices listed in InstaCal utility
        MccDaq.DaqDeviceManager.IgnoreInstaCal();

        //get inventory of MCC USB devices
        MccDaq.DaqDeviceDescriptor[] inventory =
MccDaq.DaqDeviceManager.GetDaqDeviceInventory(MccDaq.DaqDeviceInterface.Usb);

        int numDevDiscovered = inventory.Length;

        if (numDevDiscovered > 0)
        {   //loop through inventory looking for USB-1808
            for (int boardNum = 0; boardNum < numDevDiscovered; boardNum++)
            {

                try
                {

                    if (inventory[boardNum].DevString.Contains(dev))
                    {
                        MccDaq.MccBoard daqBoard =
MccDaq.DaqDeviceManager.CreateDaqDevice(boardNum, inventory[boardNum]);
                        return daqBoard;
                    }

                }
                catch (ULException ule)
                {
                    Console.WriteLine("Error occured: " + ule.Message);
                }
            }
        }
        return null;

    }

    /*////////////////////////////////////////////////////////////////////////////*/
```

```
        public static void WaitForKey()
        {
            Console.WriteLine("\nPress any key to continue...");
            do
            {
                System.Threading.Thread.Sleep(20);

            } while (!Console.KeyAvailable);
            ConsoleKeyInfo cki = Console.ReadKey();

        }

        public static void DisplayData(int[] datArray, int rows)
        {

            int i = 0;

            for (int row = 0; row < rows; row++)
            {

                for (int c = 0; c < CHANCOUNT; c++)
                {

                    Console.Write("{0}\t",
Convert.ToInt32(datArray[i]).ToString("D").PadLeft(10));
                    fStream.Write("{0}\t", Convert.ToInt32(datArray[i]).ToString("D"));
                    i++;
                }

                Console.Write("\r\n");
                fStream.Write("\r\n");
            }
            fStream.Flush();
        }
        /*////////////////////////////////////////////////////////////////////////////////*/

        public static void CreateFileHeaders()
        {
            Console.WriteLine("Reading channels {0} through {1}\n", FIRSTCHANNEL,
LASTCHANNEL);

            //''''''''''''''''''' create text file header strings ''''''''''''''
            string[] hdr = new string[7];
            hdr[0] = "Recording date     : {0}";
            hdr[1] = "Block length       : {0}";
```

```
            hdr[2] = "Number of Packets  : {0}";
            hdr[3] = "Delta              : {0} sec.";
            hdr[4] = "Runtime            : {0} sec.";
            hdr[5] = "Number of channels : {0}";

            //''''''''''''''''''''' build column headers string ''''''''''''''''''

            for (int j = FIRSTCHANNEL; j < CHANCOUNT; j++)
            {
               hdr[6] = hdr[6] + String.Format("In{0} [V]", j);
               hdr[6] = hdr[6] + "\t";

            }
            hdr[4].Trim('\t');

            //''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
            DateTime MyDate = DateTime.Now;

            fStream.WriteLine(hdr[0], MyDate);
            fStream.WriteLine(hdr[1], PACKETSIZE);
            fStream.WriteLine(hdr[2], NUMOFPACKETS);
            fStream.WriteLine(hdr[3], (1.0f / FREQ).ToString("0.000"));
            fStream.WriteLine(hdr[4], (NUMOFPACKETS * PACKETSIZE) / FREQ);
            fStream.WriteLine(hdr[5], CHANCOUNT);
            fStream.WriteLine(hdr[6]);

            Console.WriteLine(hdr[0], MyDate);
            Console.WriteLine(hdr[1], PACKETSIZE);
            Console.WriteLine(hdr[2], NUMOFPACKETS);
            Console.WriteLine(hdr[3], (1.0f / FREQ).ToString("0.000"));
            Console.WriteLine(hdr[4], (NUMOFPACKETS * PACKETSIZE) / FREQ);
            Console.WriteLine(hdr[5], CHANCOUNT);
            Console.WriteLine(hdr[6]);

        }

    }
}
```