Future Technology Devices International Ltd.

Application Note AN_154

Vinculum-II Webcam Application for Windows

Document Reference No.: FT_000342

Version 1.0

Issue Date: 2010-10-14

This application note explains how to use a Vinculum-II webcam windows application to display images received from a webcam connected to Vinculum-II V2Eval evaluation board. It also provides and explains the source code used for this application.

Future Technology Devices International Limited (FTDI)

Unit 1,2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777   Fax: + 44 (0) 141 429 2758
E-Mail (Support): **support1@ftdichip.com** Web: http://www.ftdichip.com

Table of Contents

1 Introduction 2
2 How to Start the Webcam Application on a Windows PC 3
2.1 Running Webcam Application from Windows Command prompt 3
2.2 Running the Webcam Application from Windows Explorer 7
3 Operation of the Webcam Application 8
3.1 Flow of Control 8
3.2 Source code 9
4 Contact Information 10
Appendix A - References 12
Acronyms and Abbreviations 12
Appendix B – Code Listing 13
Appendix C - Revision History 25
Revision Record Sheet 26

1 Introduction

DisplayWebcam is an application that runs on a Windows PC. It can be used to display images captured by a webcam, using V2Eval board, on a Windows PC screen. This application note explains how to run the DisplayWebcam application and provides sample source code that demonstrates how it has been implemented. The source code is provided as an example and is neither guaranteed or supported by FTDI. All source code for the DisplayWebcam application can be downloaded from the following location on the FTDI website: http://www.ftdichip.com/Support/SoftwareExamples/VinculumIIProjects/WebcamWindowsAppProject.zip

The following are required to run the webcam demo application:

1. An FTDI V2Eval board loaded with webcam board side application. (WebCam.rom can be downloaded from FTDI website and can be flashed into the board using VinIDE).
2. "Logitech webcam pro 9000" webcam connected to USB (Port A) of V2Eval board.
3. A Windows PC connected to UART of V2Eval board.
4. The DisplayWebcam executable.

Figure 1 illustrates the connection between the webcam, the V2Eval board and the Windows PC.

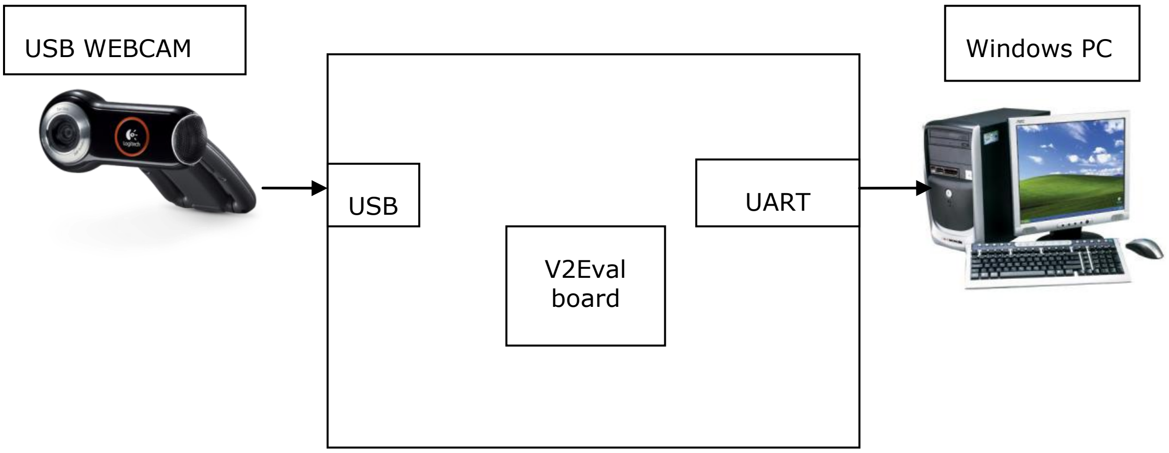


Figure 1: Connection diagram for webcam demo

The application running on the V2Eval board, WebCam.rom, configures the USB webcam. The V2Eval receives the webcam YUV data through the USB interface (in isochronous mode) and sends the data out through the V2EVAL UART interface to the PC. This Windows PC application configures the PC UART port to receive the data from the V2Eval board application, synchronizes the frames, converts the YUV data to RGB data, writes the RGB data into a BMP file and finally displays the BMP image on the PC screen.

2 How to Start the Webcam Application on a Windows PC

This webcam Windows application can be started on a PC in two ways.

1. From Windows command prompt.
2. From Windows Explorer.

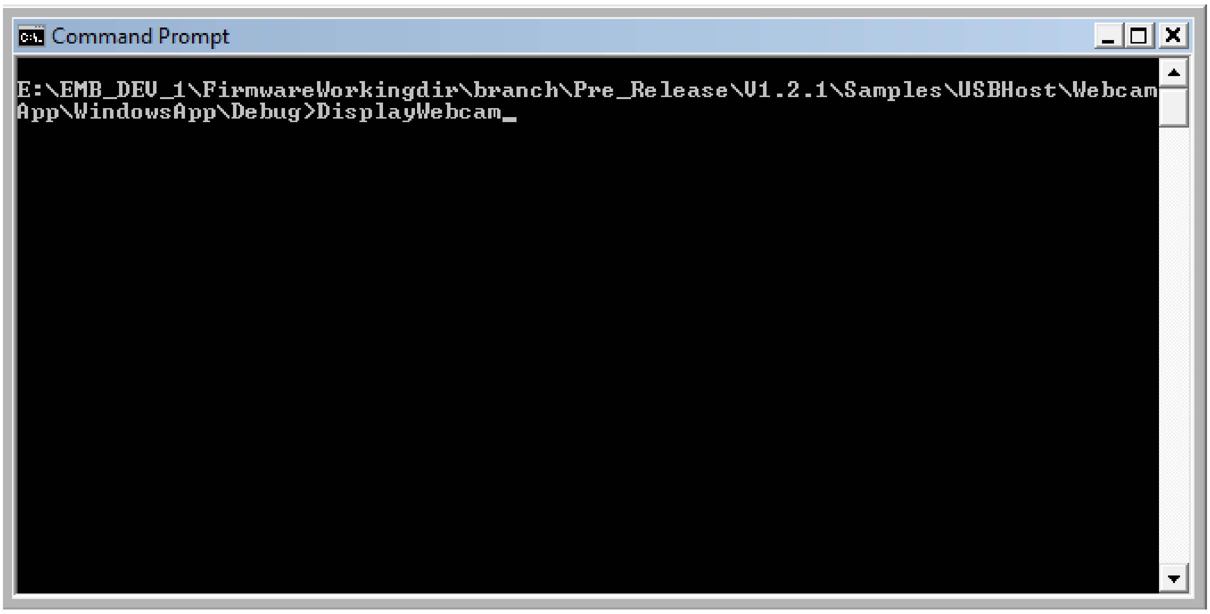
Before starting the application,
Connect the USB webcam to the USB port of V2Eval board (Port A).
Connect the debugger port to windows PC
Flash the Webcam.rom file to the V2Eval board.
Download the Windows webcam application from FTDI website and store it in C:

2.1 Running Webcam Application from Windows Command prompt

Open a command prompt on the PC.

Type cd c:\WindowsApp\Debug in the command prompt and press enter.

Type DisplayWebcam in the command prompt and press enter as in Figure 2.



```
E:\EMB_DEV_1\FirmwareWorkingdir\branch\Pre_Release\V1.2.1\Samples\USBHost\Webcam
App\WindowsApp\Debug>DisplayWebcam_
```

Figure 2: Starting webcam Windows application from command prompt

This results in the window shown in Figure 3

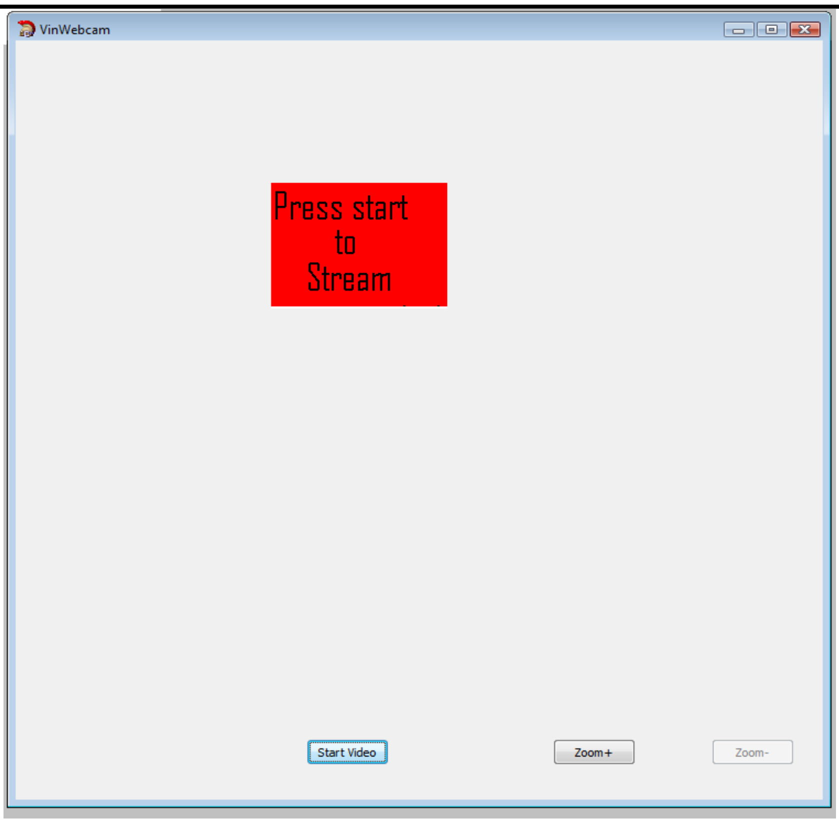


Figure 3: Initial screen of webcam Windows application

To start the application, Click on ‘Start Video’ button.

Once the streaming has started the webcam image is displayed on the PC in a window similar to that shown in Figure 4.

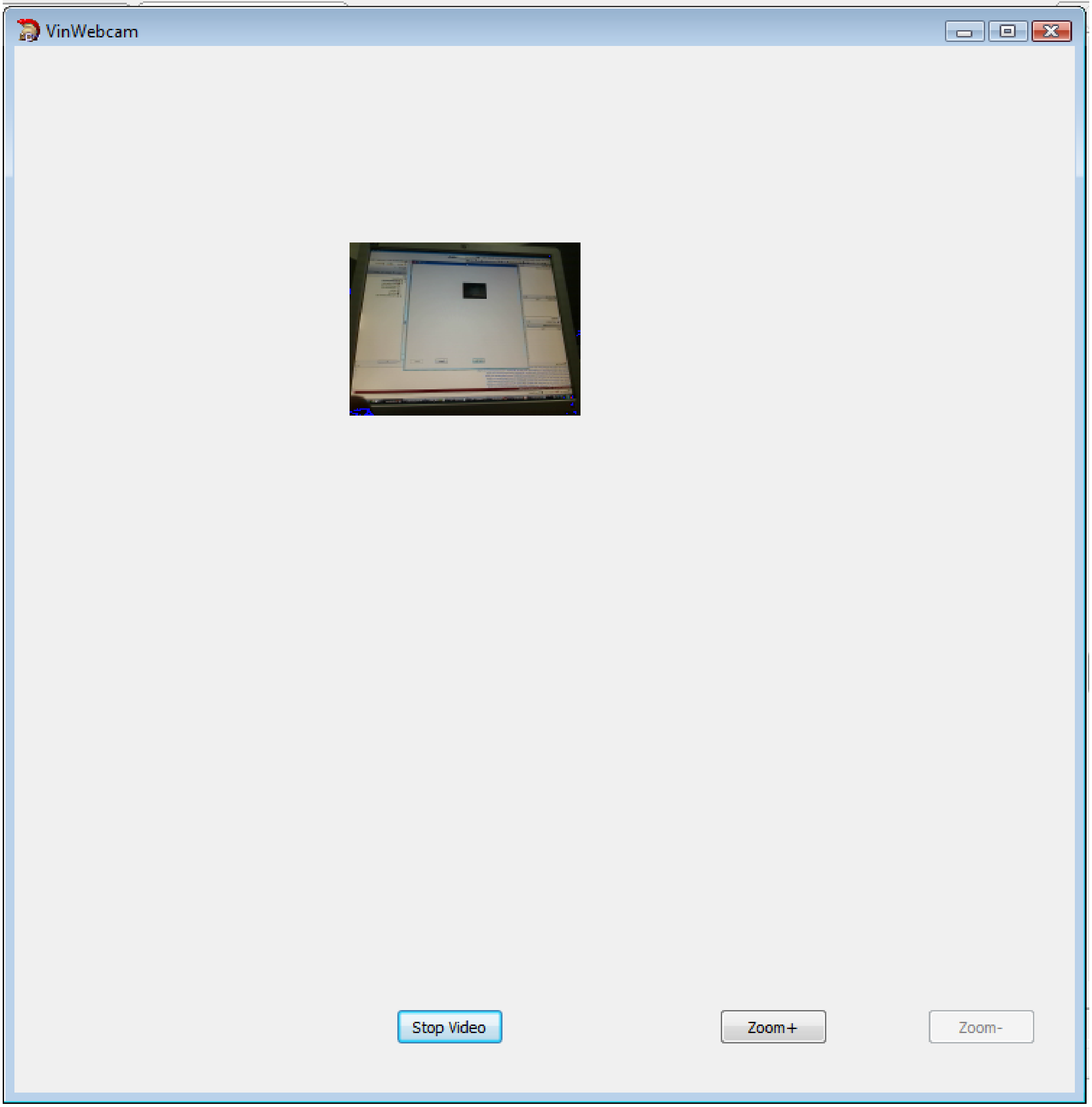


Figure 4: Displaying image streamed from webcam

There are 2 additional buttons available as shown in Figure 4 : 'Zoom +' and 'Zoom -'

'Zoom +' increases the size of the image (see Figure 5). The image can be magnified up to 3 times, after which the button will be disabled.

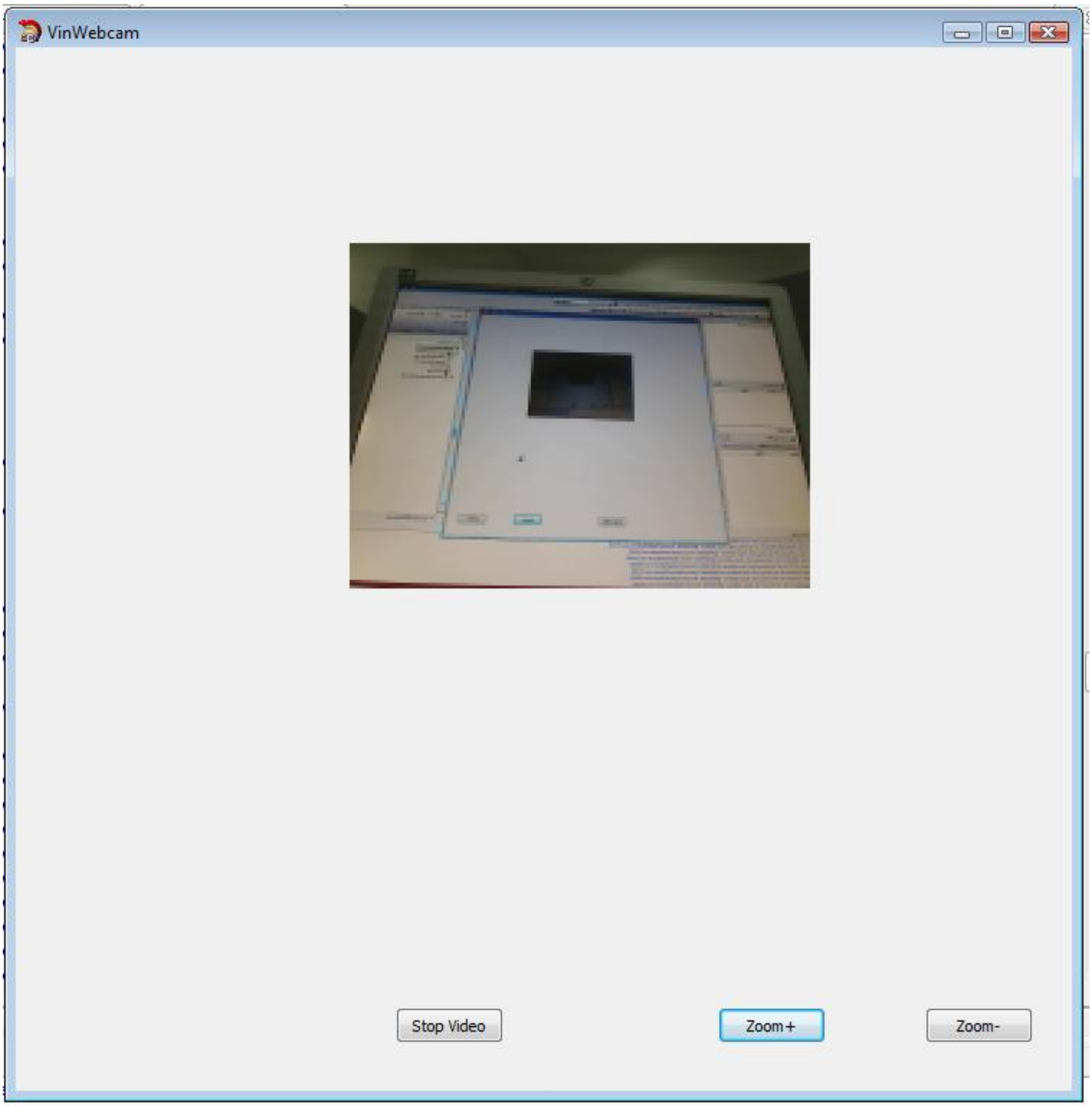
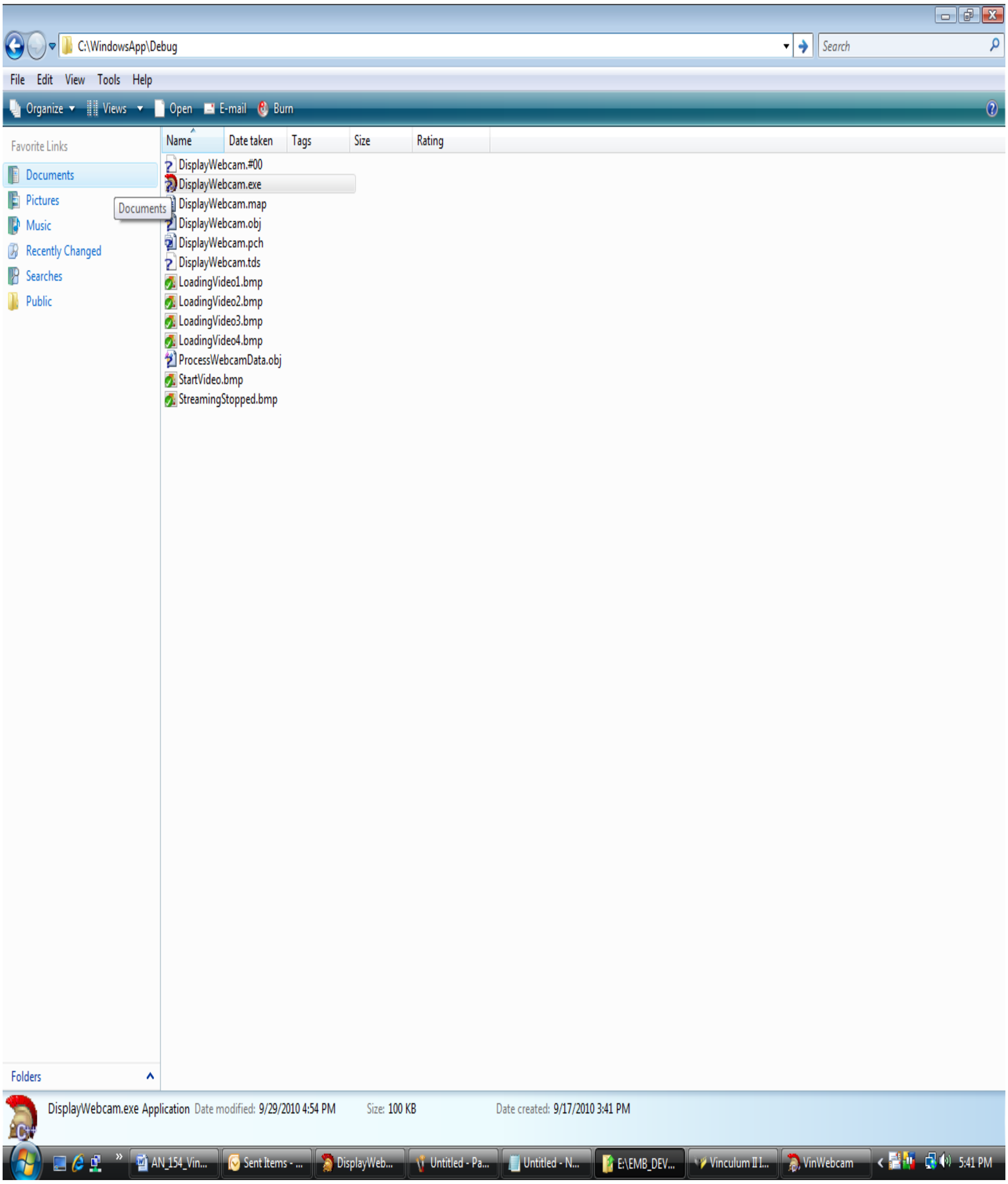


Figure 5:Zoomed image

'Zoom -' is disabled until the Zoom+ is used. When this button is pressed it reduces the size of the image. Once the image reaches its original size, the 'Zoom -' is disabled.

2.2 Running the Webcam Application from Windows Explorer

Running the PC Webcam application using Windows Explorer is similar to running from the command prompt. In this case, open Windows Explorer and navigate to the debug folder (i.e. C:\WindowsApp\Debug). Then select 'DisplayWebcam.exe' as shown in the Figure 6 and press enter.



C:\WindowsApp\Debug

Name	Date taken	Tags	Size	Rating
DisplayWebcam.#00				
DisplayWebcam.exe				
DisplayWebcam.map				
DisplayWebcam.obj				
DisplayWebcam.pch				
DisplayWebcam.tds				
LoadingVideo1.bmp				
LoadingVideo2.bmp				
LoadingVideo3.bmp				
LoadingVideo4.bmp				
ProcessWebcamData.obj				
StartVideo.bmp				
StreamingStopped.bmp				

DisplayWebcam.exe Application Date modified: 9/29/2010 4:54 PM Size: 100 KB Date created: 9/17/2010 3:41 PM

Figure 6: Running from Windows explorer

The remaining steps are the same as running from the command prompt.

3 Operation of the Webcam Application

This section describes the operation of the webcam application. It contains an overview of the flow of control in the application and provides code fragments that illustrate how it has been implemented.

3.1 Flow of Control

The main routine begins when the "Start Video" button is pressed on the resultant GUI (See Figure 4).

The following flow chart in Figure 7 explains the function of that main routine.

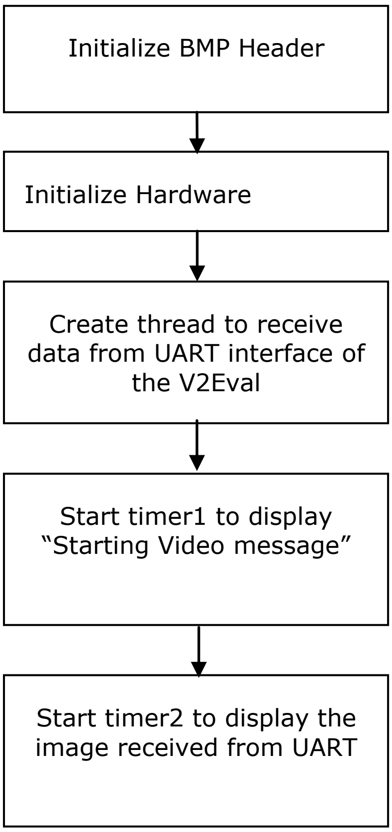

Initialize BMP Header
Initialize Hardware
Create thread to receive data from UART interface of the V2Eval
Start timer1 to display "Starting Video message"
Start timer2 to display the image received from UART

Figure 7: Flow diagram of Windows webcam application

The Initialize BMP Header routine initializes the BMP header which will be used to construct the BMP image. It includes the header for normal image and the zoomed image.

The initHardware routine initialises the UART to receive data from V2Eval board.

The ReceiveDataFromWebcam is a thread that continuously monitors for data in the UART and store it in the buffer. This is written in separate thread so that the other process will display the image in the screen as and when the data is received where as this thread keep receiving the data from UART. When all bytes of the frame have been received, this thread converts the received bytes into RGB format, writes the BMP header and writes the RGB data into a .bmp file. When the zoom feature is enabled, the appropriate header will be written in the .bmp file. The routine then sets a flag to notify the timer2 to display the image.

Timer1 displays the message "starting video" until the first frame is received from the UART. When the first image is available then timer1 is stopped.

Timer2 looks to see if the flag to display the image is set. When the flag is set it displays the image (which is received from the webcam and stored as a .bmp file).

3.2 Source code

The full source code listing is contained in Appendix B.

4 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales)	sales1@ftdichip.com
E-mail (Support)	support1@ftdichip.com
E-mail (General Enquiries)	admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com
Web Shop URL	http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales)	us.sales@ftdichip.com
E-Mail (Support)	us.support@ftdichip.com
E-Mail (General Enquiries)	us.admin@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
Shanghai, 200051
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales)	cn.sales@ftdichip.com
E-mail (Support)	cn.support@ftdichip.com
E-mail (General Enquiries)	cn.admin@ftdichip.com
Web Site URL	http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Vinculum is part of Future Technology Devices International Ltd. Neither the whole nor any part of the information contained in, or the product described in this manual, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied. Future Technology Devices International Ltd will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected. This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury. This document provides preliminary information that may be subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH United Kingdom. Scotland Registered Number: SC136640

Appendix A - References

1. www.fourcc.org/fccyvrgb.php
2. http://en.wikipedia.org/wiki/BMP_file_format
3. http://www.usb.org/developers/devclass_docs -- video class
4. Vinculum-II Embedded Dual USB Host Controller IC Data Sheet
5. Vinculum-II IO Mux Explained
6. Vinculum-II Tool Chain Getting Started Guide

Acronyms and Abbreviations

Terms	Description
IOMux	Input Output Multiplexer – Used to configure pin selection on different package types of the VNC2.
V2-Eval	Vinculum II Evaluation Board- Customer evaluation board for the VNC2 allowing prototype development.
UART	Universal Asynchronous Receiver/Transmitter
VinIDE	Vinculum Integrated Development Environment – Development environment for writing and building application code for the VNC2.
BMP	**Bitmap (file format)**
RGB	**RGB colour model** is an additive colour model in which red, green, and blue light are added together in various ways to reproduce a broad array of colours. The name of the model comes from the initials of the three additive primary colours, red, green, and blue.
YUV	The YUV model defines a colour space in terms of one luma (Y) and two chrominance (UV) components.
VNC2	Vinculum II – FTDI's second generation dual Host/Slave IC.
IDE	Integrated Development Environment
SD	Secure Digital

Table 1 Acronyms and Abbreviations

Appendix B – Code Listing

```
/*
This software is provided by Future Technology Devices International Limited "as is" and
any express or implied warranties, including, but not limited to, the implied warranties
of merchantability and fitness for a particular purpose are disclaimed. In no event shall
future technology devices international limited be liable for any direct, indirect,
incidental, special, exemplary, or consequential damages (including, but not limited to,
procurement of substitute goods or services; loss of use, data, or profits; or business
interruption) however caused and on any theory of liability, whether in contract, strict
liability, or tort (including negligence or otherwise) arising in any way out of the use
of this software, even if advised of the possibility of such damage.
*/

#include <vcl.h>
#include <Clipbrd.hpp>
#include <stdio.h>
#include <fstream>
#pragma hdrstop

#include "ProcessWebcamData.h"
#include "ftd2xx.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TVinWebcam *VinWebcam;
int testVar = 0;
int globalSync = 0;
int WebcamStarted = 0;
FT_HANDLE fthandle;
int testidx = 0;
int showImage = 0;
int imageReady = 0;
int len, actual, remain;
unsigned char YuYBuffer[YUV_IMAGE_SIZE+1];
unsigned char RGBData[RGB_IMAGE_SIZE + 1];
unsigned char ZoomedRGBData[ZOOM4_RGB_IMAGE_SIZE + 1];
unsigned char TempBuffer[192];

char idxExt[10];
char Filename[100] = "bmpData";
char FullFileName[100];
char *Ext = ".bmp";
int exitThread = 0;
int Enlarge = 0;
int Zoom = 1;
int HardwareInitFlag = 0;


HANDLE hThread;

/*
A typical BMP file usually contains the following blocks of data:

BMP File Header   Stores general information about the BMP file.

Bitmap Information
(DIB header)            Stores detailed information about the bitmap image.

Color Palette           Stores the definition of the colors being used for
                              indexed color bitmaps.
```

```
Bitmap Data             Stores the actual image, pixel by pixel.

*/

bmpfile_header HardcodeBMPFile_header;
bmpfile_header Zoom1HardcodeBMPFile_header;
bmpfile_header Zoom2HardcodeBMPFile_header;
bmpfile_header Zoom3HardcodeBMPFile_header;
bmpfile_header Zoom4HardcodeBMPFile_header;

bmp_dib_v3_header Hardcode_bmp_dib_v3_header;
bmp_dib_v3_header Zoom1Hardcode_bmp_dib_v3_header;
bmp_dib_v3_header Zoom2Hardcode_bmp_dib_v3_header;
bmp_dib_v3_header Zoom3Hardcode_bmp_dib_v3_header;
bmp_dib_v3_header Zoom4Hardcode_bmp_dib_v3_header;


int Resample(unsigned char *ResizedData,unsigned char *OriginalData,int
newWidth, int newHeight, int oriWidth,int oriHeight)
{
      double scaleWidth =  (double)newWidth / (double)oriWidth;
      double scaleHeight = (double)newHeight / (double)oriHeight;
      int cy,cx;
      for(cy = 0; cy < newHeight; cy++)
      {
            for(cx = 0; cx < newWidth; cx++)
            {
                  int pixel = (cy * (newWidth *3)) + (cx*3);
                  int nearestMatch =  (((int)(cy / scaleHeight) * (oriWidth *3))
+ ((int)(cx / scaleWidth) *3) );

                  ResizedData[pixel    ] =  OriginalData[nearestMatch    ];
                  ResizedData[pixel + 1] =  OriginalData[nearestMatch + 1];
                  ResizedData[pixel + 2] =  OriginalData[nearestMatch + 2];
            }
      }


      return 1;
}

void YUY2RGBConvert (unsigned char inputBuffer1[],
                                    unsigned char OutputBuffer2[],
                                    int size)
{
      double Blue, Green, Red;
      double Y0, Y1, U, V;
      int inIdx = 0, outIdx = 0;
      int pxcount = 0;

      while (pxcount < size)
      {
            Y0 = inputBuffer1[inIdx++];
            U = inputBuffer1[inIdx++];
            Y1 = inputBuffer1[inIdx++];
            V = inputBuffer1[inIdx++];
            pxcount += 2;

            Blue = 1.164 * (Y0 - 16) + 2.018 * (U - 128);
            Green = 1.164 * (Y0 - 16) - 0.813 * (V - 128) - 0.391 * (U - 128);
            Red = 1.164 * (Y0 - 16) + 1.596 * (V - 128);
            OutputBuffer2[outIdx++] = Red;
            OutputBuffer2[outIdx++] = Green;
            OutputBuffer2[outIdx++] = Blue;


            Blue = 1.164 * (Y1 - 16) + 2.018 * (U - 128);
            Green = 1.164 * (Y1 - 16) - 0.813 * (V - 128) - 0.391 * (U - 128);
            Red = 1.164 * (Y1 - 16) + 1.596 * (V - 128);
            OutputBuffer2[outIdx++] = Red;
            OutputBuffer2[outIdx++] = Green;
            OutputBuffer2[outIdx++] = Blue;
      }
      return;
}

__fastcall TVinWebcam::TVinWebcam(TComponent* Owner)
      : TForm(Owner)
{

}

void TVinWebcam::ShowStartImage1(void)
{
      Image1->Picture->LoadFromFile("LoadingVideo1.bmp");
      Sleep(30);

}


void TVinWebcam::ShowStartImage2(void)
{
      Image1->Picture->LoadFromFile("LoadingVideo2.bmp");
      Sleep(30);

}

void TVinWebcam::ShowStartImage3(void)
{
      Image1->Picture->LoadFromFile("LoadingVideo3.bmp");
      Sleep(30);

}
void TVinWebcam::ShowStartImage4(void)
{
      Image1->Picture->LoadFromFile("LoadingVideo4.bmp");
      Sleep(30);

}

void __fastcall TVinWebcam::ShowStarVideo(TObject *Sender)
{
      if (WebcamStarted == 0)
      {
            if (testVar == 0)
            {
                  ShowStartImage1();
            }
            else if (testVar == 1)
            {
                  ShowStartImage2();
            }
            else if (testVar == 2)
            {
                  ShowStartImage3();
            }
            else if (testVar == 3)
            {
                  ShowStartImage4();
            }
            testVar++;
            if(testVar == 4)
                  testVar = 0;
      }
}

/*
Hard code value for 160 * 120 BMP image

BMP header
Magic No    2 bytes
            0x42
            0x4d

filesz      4 bytes
            0x0000e136 // i.e (160 * 120 * 3) + 36 bytes header

creator1    2 bytes
            0x0000
creator2    2 bytes
            0x0000
bmp_offset  4 bytes
            0x00000036

Bitmap Information


Eh                4           the size of this header (40 bytes)
            0x00000028  // i.e 40 bytes
12h         4           the bitmap width in pixels (signed integer).
            0x000000a0  // 160
16h         4           the bitmap height in pixels (signed integer).
            0x00000078  //120
1Ah         2           the number of color planes being used. Must be set to 1.
            0x0001
1Ch         2           the number of bits per pixel, which is the color depth
of the image. Typical values are 1, 4, 8, 16, 24 and 32.
            0x0018     //24
1Eh         4           the compression method being used. See the next table
for a list of possible values.
            0x00000000
22h         4           the image size. This is the size of the raw bitmap data
(see below), and should not be confused with the file size.
            0x0000e100
26h         4           the horizontal resolution of the image. (pixel per
meter, signed integer)
            0x00000000
2Ah         4           the vertical resolution of the image. (pixel per meter,
signed integer)
            0x00000000
2Eh         4           the number of colors in the color palette, or 0 to
default to 2n.
            0x00000000
32h         4           the number of important colors used, or 0 when every
color is important; generally ignored.
            0x00000000
Followed by RGB data
```

```
*/

void initBMPHeaders()
{
      /*Hard coding BMP header*/
      HardcodeBMPFile_header.filesz = (RGB_IMAGE_SIZE)+BMP_HEADER_SIZE + 
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;
      HardcodeBMPFile_header.creator1 = 0;
      HardcodeBMPFile_header.creator2 = 0;
      HardcodeBMPFile_header.bmp_offset = BMP_HEADER_SIZE + 
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;

      /*Hard coding bitmap information*/
      Hardcode_bmp_dib_v3_header.header_sz = BITMAP_INFO_HEADER_SIZE;
      Hardcode_bmp_dib_v3_header.width = 160;
      Hardcode_bmp_dib_v3_header.height =120;
      Hardcode_bmp_dib_v3_header.nplanes = 1;
      Hardcode_bmp_dib_v3_header.bitspp = 24;
      Hardcode_bmp_dib_v3_header.compress_type = 0;
      Hardcode_bmp_dib_v3_header.bmp_bytesz = RGB_IMAGE_SIZE;
      Hardcode_bmp_dib_v3_header.hres = 0;
      Hardcode_bmp_dib_v3_header.vres = 0;
      Hardcode_bmp_dib_v3_header.ncolors = 0 ;
      Hardcode_bmp_dib_v3_header.nimpcolors = 0;
      #if 0
    /*Hard coding BMP header*/
      HardcodedEnlargedBMPFile_header.filesz = 
(ENLARGED_RGB_IMAGE_SIZE)+BMP_HEADER_SIZE + BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;
      HardcodedEnlargedBMPFile_header.creator1 = 0;
      HardcodedEnlargedBMPFile_header.creator2 = 0;
      HardcodedEnlargedBMPFile_header.bmp_offset = BMP_HEADER_SIZE + 
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;

      /*Hard coding bitmap information*/
      HardcodeEnlarged_bmp_dib_v3_header.header_sz = BITMAP_INFO_HEADER_SIZE;
      HardcodeEnlarged_bmp_dib_v3_header.width = ENLARGED_IMAGE_SIZE_X;
      HardcodeEnlarged_bmp_dib_v3_header.height = ENLARGED_IMAGE_SIZE_Y;
      HardcodeEnlarged_bmp_dib_v3_header.nplanes = 1;
      HardcodeEnlarged_bmp_dib_v3_header.bitspp = 24;
      HardcodeEnlarged_bmp_dib_v3_header.compress_type = 0;
      HardcodeEnlarged_bmp_dib_v3_header.bmp_bytesz = ENLARGED_RGB_IMAGE_SIZE;
      HardcodeEnlarged_bmp_dib_v3_header.hres = 0;
      HardcodeEnlarged_bmp_dib_v3_header.vres = 0;
      HardcodeEnlarged_bmp_dib_v3_header.ncolors = 0 ;
      HardcodeEnlarged_bmp_dib_v3_header.nimpcolors = 0;
    #endif

      /*zoom1*/
      Zoom1HardcodeBMPFile_header.filesz = 
(ZOOM1_RGB_IMAGE_SIZE)+BMP_HEADER_SIZE + BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;
      Zoom1HardcodeBMPFile_header.creator1 = 0;
      Zoom1HardcodeBMPFile_header.creator2 = 0;
      Zoom1HardcodeBMPFile_header.bmp_offset = BMP_HEADER_SIZE + 
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;

      /*Hard coding bitmap information*/
      Zoom1Hardcode_bmp_dib_v3_header.header_sz = BITMAP_INFO_HEADER_SIZE;
      Zoom1Hardcode_bmp_dib_v3_header.width = ZOOM1_IMAGE_SIZE_X;
      Zoom1Hardcode_bmp_dib_v3_header.height = ZOOM1_IMAGE_SIZE_Y;
      Zoom1Hardcode_bmp_dib_v3_header.nplanes = 1;
      Zoom1Hardcode_bmp_dib_v3_header.bitspp = 24;
      Zoom1Hardcode_bmp_dib_v3_header.compress_type = 0;
```

```
        Zoom1Hardcode_bmp_dib_v3_header.bmp_bytesz = ZOOM1_RGB_IMAGE_SIZE;
        Zoom1Hardcode_bmp_dib_v3_header.hres = 0;
        Zoom1Hardcode_bmp_dib_v3_header.vres = 0;
        Zoom1Hardcode_bmp_dib_v3_header.ncolors = 0 ;
        Zoom1Hardcode_bmp_dib_v3_header.nimpcolors = 0;



        /*zoom2*/
        Zoom2HardcodeBMPFile_header.filesz =
(ZOOM2_RGB_IMAGE_SIZE)+BMP_HEADER_SIZE + BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;
        Zoom2HardcodeBMPFile_header.creator1 = 0;
        Zoom2HardcodeBMPFile_header.creator2 = 0;
        Zoom2HardcodeBMPFile_header.bmp_offset = BMP_HEADER_SIZE +
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;

        /*Hard coding bitmap information*/
        Zoom2Hardcode_bmp_dib_v3_header.header_sz = BITMAP_INFO_HEADER_SIZE;
        Zoom2Hardcode_bmp_dib_v3_header.width = ZOOM2_IMAGE_SIZE_X;
        Zoom2Hardcode_bmp_dib_v3_header.height = ZOOM2_IMAGE_SIZE_Y;
        Zoom2Hardcode_bmp_dib_v3_header.nplanes = 1;
        Zoom2Hardcode_bmp_dib_v3_header.bitspp = 24;
        Zoom2Hardcode_bmp_dib_v3_header.compress_type = 0;
        Zoom2Hardcode_bmp_dib_v3_header.bmp_bytesz = ZOOM2_RGB_IMAGE_SIZE;
        Zoom2Hardcode_bmp_dib_v3_header.hres = 0;
        Zoom2Hardcode_bmp_dib_v3_header.vres = 0;
        Zoom2Hardcode_bmp_dib_v3_header.ncolors = 0 ;
        Zoom2Hardcode_bmp_dib_v3_header.nimpcolors = 0;


        /*zoom3*/
        Zoom3HardcodeBMPFile_header.filesz =
(ZOOM3_RGB_IMAGE_SIZE)+BMP_HEADER_SIZE + BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;
        Zoom3HardcodeBMPFile_header.creator1 = 0;
        Zoom3HardcodeBMPFile_header.creator2 = 0;
        Zoom3HardcodeBMPFile_header.bmp_offset = BMP_HEADER_SIZE +
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;

        /*Hard coding bitmap information*/
        Zoom3Hardcode_bmp_dib_v3_header.header_sz = BITMAP_INFO_HEADER_SIZE;
        Zoom3Hardcode_bmp_dib_v3_header.width = ZOOM3_IMAGE_SIZE_X;
        Zoom3Hardcode_bmp_dib_v3_header.height = ZOOM3_IMAGE_SIZE_Y;
        Zoom3Hardcode_bmp_dib_v3_header.nplanes = 1;
        Zoom3Hardcode_bmp_dib_v3_header.bitspp = 24;
        Zoom3Hardcode_bmp_dib_v3_header.compress_type = 0;
        Zoom3Hardcode_bmp_dib_v3_header.bmp_bytesz = ZOOM3_RGB_IMAGE_SIZE;
        Zoom3Hardcode_bmp_dib_v3_header.hres = 0;
        Zoom3Hardcode_bmp_dib_v3_header.vres = 0;
        Zoom3Hardcode_bmp_dib_v3_header.ncolors = 0 ;
        Zoom3Hardcode_bmp_dib_v3_header.nimpcolors = 0;

        /*zoom4*/
        Zoom4HardcodeBMPFile_header.filesz =
(ZOOM4_RGB_IMAGE_SIZE)+BMP_HEADER_SIZE + BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;
        Zoom4HardcodeBMPFile_header.creator1 = 0;
        Zoom4HardcodeBMPFile_header.creator2 = 0;
        Zoom4HardcodeBMPFile_header.bmp_offset = BMP_HEADER_SIZE +
BITMAP_INFO_HEADER_SIZE+PALLET_SIZE;

        /*Hard coding bitmap information*/
        Zoom4Hardcode_bmp_dib_v3_header.header_sz = BITMAP_INFO_HEADER_SIZE;
        Zoom4Hardcode_bmp_dib_v3_header.width = ZOOM4_IMAGE_SIZE_X;
        Zoom4Hardcode_bmp_dib_v3_header.height = ZOOM4_IMAGE_SIZE_Y;
```

```
      Zoom4Hardcode_bmp_dib_v3_header.nplanes = 1;
      Zoom4Hardcode_bmp_dib_v3_header.bitspp = 24;
      Zoom4Hardcode_bmp_dib_v3_header.compress_type = 0;
      Zoom4Hardcode_bmp_dib_v3_header.bmp_bytesz = ZOOM4_RGB_IMAGE_SIZE;
      Zoom4Hardcode_bmp_dib_v3_header.hres = 0;
      Zoom4Hardcode_bmp_dib_v3_header.vres = 0;
      Zoom4Hardcode_bmp_dib_v3_header.ncolors = 0 ;
      Zoom4Hardcode_bmp_dib_v3_header.nimpcolors = 0;

}

int initHardware()
{
      FT_DEVICE_LIST_INFO_NODE *devInfo = NULL;
      DWORD numDevs;

      FT_STATUS ftStatus;

      ftStatus = FT_CreateDeviceInfoList (&numDevs);
      if (ftStatus == FT_OK)
      {
            printf ("Number of devices is %d\n", numDevs);
      }
      else
      {
            printf ("fsStatus = %d\n", ftStatus);
            return -1;
      }


      ftStatus = FT_OpenEx((void *)"VII Eval Board
A",FT_OPEN_BY_DESCRIPTION,&fthandle);

      if (ftStatus == FT_OK)
      {
            printf ("FT_Open(), device  Success status = %d\n", ftStatus);
      }
      else
      {
        ftStatus = FT_Close(fthandle);
            free(fthandle);
            printf ("Error FT_Open(), device Status = %d\n", ftStatus);
            return -1;
      }

      FT_SetDataCharacteristics  (fthandle, FT_BITS_8, FT_STOP_BITS_1,
FT_PARITY_NONE);
      if ((ftStatus = FT_SetBaudRate (fthandle, 6000000)) != FT_OK)
      {
            printf ("Error FT_SetBaudRate(%d)\n", ftStatus);
      }
      else
      {
            printf ("FT_SetBaudRate(%d)\n", ftStatus);
      }

      if ((ftStatus = FT_SetRts     (fthandle) ) != FT_OK)
      {
            printf ("Error FT_SetRts(%d)\n", ftStatus);
            return -1;
      }
      else
      {
            printf ("FT_SetRts(%d)\n", ftStatus);
      }

      //use RTS/CTS flow control to avoid data loss
      FT_SetFlowControl(fthandle, FT_FLOW_RTS_CTS, 0, 0);
      return 1;
}



void syncToWebcamFrames(void)
{
      unsigned char syncd = 0;
      DWORD dwBytesRead;
      FT_STATUS ftStatus;

      // synchronize with the frame header
      while (syncd == 0)
      {

            ftStatus = FT_Read(fthandle, TempBuffer, 1 ,&dwBytesRead);
            if (TempBuffer[0] == 0x0C) {
                  // could be a start of a webcam frame
                  // burn the rest of the frame...
                  ftStatus = FT_Read(fthandle, TempBuffer, 191 ,&dwBytesRead);
                  // check for EOF bit
                  if (TempBuffer[0] & WEBCAM_EOF_BIT == WEBCAM_EOF_BIT) {
                        // Yup - it's synced to the webcam frames
                        return ;
                  }
            }

      }
      return;

}

DWORD ReceiveDataFromWebcam(void)
{
      FILE *fpWr;
      unsigned char oneByte;
      long int counter = 0;
      FT_STATUS ftStatus;
      DWORD dwBytesWritten, dwBytesRead;
      int RGBCounter = (ImageXSize *ImageYSize *3);

      while (HardwareInitFlag == -1)
      {
               HardwareInitFlag =  initHardware();
      }
      syncToWebcamFrames();
      while(1)
      {
            if(exitThread == 0)
            return 1;
            len = remain;
            remain = 0;
            do
            {
                  // read 1 frame of webcam data
                  ftStatus = FT_Read (fthandle, TempBuffer, 192 , &dwBytesRead);
                  if(ftStatus != FT_OK)
                  {
                        printf ("Error FT_Read(%d)\n", ftStatus );
                  }

                  if (TempBuffer[0] != 0x0C) {
                        printf("Not synchronised!  Attempting resync: ");
                        syncToWebcamFrames();
                        printf("done\n");
                        break;
                  }

                  if ((TempBuffer[1] & WEBCAM_EOF_BIT) == WEBCAM_EOF_BIT) {
                        break;
                  }
                  if ((TempBuffer[1] & WEBCAM_ERR_BIT) == 0) {
                        if ((len + 180) > YUV_IMAGE_SIZE)
                        {
                              remain = len + 180 - YUV_IMAGE_SIZE;
                              actual = 180 - remain;
                        }
                        else
                        {
                              remain = 0;
                              actual = 180;
                        }
                        memcpy(&YuYBuffer[len], &TempBuffer[12], actual);
                        len += actual;
                  }

            }
            while (len < YUV_IMAGE_SIZE);


            while(imageReady == 1)

            {

                  ;
            }

            itoa ( testidx, idxExt, 10 );
            testidx++;
            if(testidx == 100)
            {
                  testidx = 0;
            }
            strcpy(FullFileName,Filename);
            strcat(FullFileName,idxExt);
            strcat(FullFileName,Ext);
            YUY2RGBConvert(YuYBuffer, RGBData,IMAGE_SIZE);
            fpWr = fopen(FullFileName,"wb+");
            oneByte = 0x42;
            fwrite(&oneByte,1,1,fpWr);
            oneByte = 0x4d;
            fwrite(&oneByte,1,1,fpWr);
            switch (Zoom)
            {
            case ZOOM1_SIZE:

      Resample(ZoomedRGBData,RGBData,(int)ZOOM1_IMAGE_SIZE_X,(int)ZOOM1_IMAGE_SI
ZE_Y,(int)IMAGE_SIZE_X,(int)IMAGE_SIZE_Y);

      fwrite(&Zoom1HardcodeBMPFile_header,sizeof(bmpfile_header),1,fpWr);
      fwrite(&Zoom1Hardcode_bmp_dib_v3_header,sizeof(bmp_dib_v3_header),1,fpWr);

                  RGBCounter = ZOOM1_RGB_IMAGE_SIZE;
                  for (counter = 0;counter < ZOOM1_RGB_IMAGE_SIZE;counter++)
                  {
                        fwrite(&ZoomedRGBData[RGBCounter-1],1,1,fpWr);
                        RGBCounter--;
                  }

            break;
            case ZOOM2_SIZE:

      Resample(ZoomedRGBData,RGBData,(int)ZOOM2_IMAGE_SIZE_X,(int)ZOOM2_IMAGE_SI
ZE_Y,(int)IMAGE_SIZE_X,(int)IMAGE_SIZE_Y);

      fwrite(&Zoom2HardcodeBMPFile_header,sizeof(bmpfile_header),1,fpWr);

      fwrite(&Zoom2Hardcode_bmp_dib_v3_header,sizeof(bmp_dib_v3_header),1,fpWr);

                  RGBCounter = ZOOM2_RGB_IMAGE_SIZE;
                  for (counter = 0;counter < ZOOM2_RGB_IMAGE_SIZE;counter++)
                  {
                        fwrite(&ZoomedRGBData[RGBCounter-1],1,1,fpWr);
                        RGBCounter--;
                  }
            break;
            case ZOOM3_SIZE:

      Resample(ZoomedRGBData,RGBData,(int)ZOOM3_IMAGE_SIZE_X,(int)ZOOM3_IMAGE_SI
ZE_Y,(int)IMAGE_SIZE_X,(int)IMAGE_SIZE_Y);

      fwrite(&Zoom3HardcodeBMPFile_header,sizeof(bmpfile_header),1,fpWr);

      fwrite(&Zoom3Hardcode_bmp_dib_v3_header,sizeof(bmp_dib_v3_header),1,fpWr);

                  RGBCounter = ZOOM3_RGB_IMAGE_SIZE;
                  for (counter = 0;counter < ZOOM3_RGB_IMAGE_SIZE;counter++)
                  {
                        fwrite(&ZoomedRGBData[RGBCounter-1],1,1,fpWr);
                        RGBCounter--;
                  }
            break;

            default:
                  fwrite(&HardcodeBMPFile_header,sizeof(bmpfile_header),1,fpWr);

      fwrite(&Hardcode_bmp_dib_v3_header,sizeof(bmp_dib_v3_header),1,fpWr);

                  RGBCounter = RGB_IMAGE_SIZE;
                  for (counter = 0;counter < RGB_IMAGE_SIZE;counter++)
                  {
                        fwrite(&RGBData[RGBCounter-1],1,1,fpWr);
                        RGBCounter--;
                  };
            }

            fclose(fpWr);
            Sleep(1);
            imageReady = 1;
      }
      return 1;
}
```

```
void __fastcall TVinWebcam::Button1Click(TObject *Sender)
{
      FT_STATUS ftStatus;
      if (Button1->Caption == "Start Video")
      {
            Button1->Caption = "Stop Video";
            initBMPHeaders();
            //Timer1->Enabled = true;
            HardwareInitFlag = initHardware();
            Sleep(1);
            exitThread = 1;
            hThread =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)&ReceiveDataFromWebcam,0,0,NULL);
            Timer1->Enabled = true;
            Timer2->Enabled = true;

      }
      else
      {
            Timer1->Enabled = false;
            Timer2->Enabled = false;
            Image1->Picture->LoadFromFile("StreamingStopped.bmp");
            //destroy thread
            //need to stop the thread?????/????
            exitThread = 0;
            CloseHandle(hThread);
            //free hardware handle
            ftStatus = FT_Close(fthandle);
            free(fthandle);
            Button1->Caption = "Start Video";
      }

}

void __fastcall TVinWebcam::ShowWebcamData(TObject *Sender)
{
      if (imageReady == 1)
      {
        Timer1->Enabled = false;
            Image1->Picture->LoadFromFile(FullFileName);
            remove(FullFileName);
            imageReady = 0;
      }

}

void __fastcall TVinWebcam::Button2Click(TObject *Sender)
{
      Zoom++;
      if (Button3->Enabled != true)
            Button3->Enabled = true;
      if (Zoom == ZOOM3_SIZE)
      {
        Button2->Enabled = false;
      }
}

void __fastcall TVinWebcam::Button3Click(TObject *Sender)
{
      if (Zoom == ZOOM1_SIZE)
      {
         Button3->Enabled = false;
      }
      Zoom --;
      if (Button2->Enabled != true)
            Button2->Enabled = true;

}
```

Appendix C - Revision History

Revision History

Rev 1.0	Initial Release	14th September, 2010