The Kwan-Truc:

A Digital Wireless Video System for Remote Controlled Platforms

Todd Hummel

Travis Reed

Computer Engineering

Senior Design Project

Fall 2008

Advisor: Al Davis

University of Utah

Table of Contents

1. Introduction and Motivation

The basic idea of the Kwan-Truc is to display streaming video from an RC helicopter on any 802.11 enabled laptop. This allows the user to experience the flight as if in the cockpit and record a flight from the helicopter's perspective. This could be used for something as serious as a surveillance and reconnaissance system, or for something as simple as a hobbyist's coolest new toy.

There are systems readily available which transmit video wirelessly, but many of them are purely analog, and require a television for viewing the video feed. We thought that it would be more portable and user-friendly if the system were configurable to work with a laptop. It's also a whole lot easier to record the video from the flight in a digital format on your laptop.

The Kwan-Truc system consists of a camera (mounted forward-facing) on an RC helicopter, along with the other modules that manage the wireless transmission of the video from the helicopter to the laptop. These other modules are a microcontroller and wireless transceiver, and they are the only additional hardware necessary (aside from the helicopter, camera, and the laptop) to complete the Kwan-Truc system. This is because the RC helicopter itself still uses the normal RC radio for its control system, and the laptop uses its normal 802.11 interface to receive the video feed. The Kwan-Truc software on the laptop retrieves the video from the 802.11 interface card and display the live video. The video can also be recorded to a file to be played back later.

2. Hardware Design

The hardware design of the Kwan-Truc system is very straightforward. As seen in Figure 1, the microcontroller communicates with the webcam over a standard USB interface, and with the wireless transceiver over a standard RS-232 interface. The following sections will give more details about the specific hardware components and why they were chosen for the Kwan-Truc.

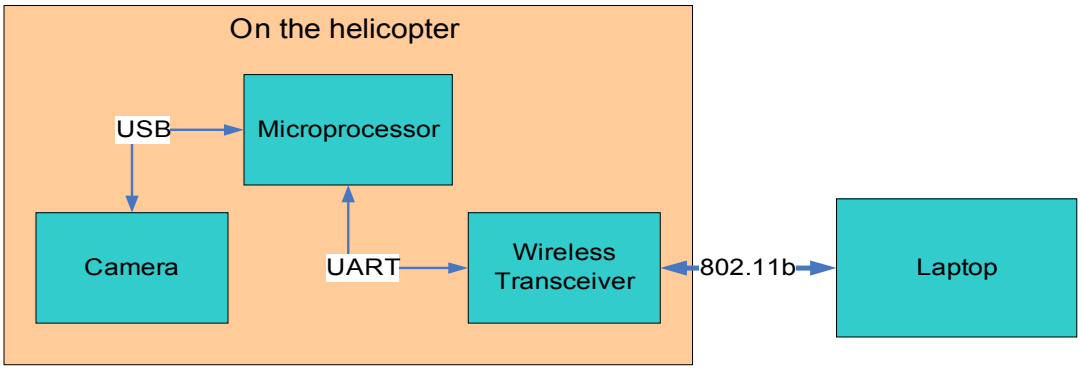

Figure 1: Block Diagram of the Kwan-Truc system

2.1. IBM Webcam

The camera we used is the IBM webcam (specifically, model number KSX-X9903), which is a very early model webcam. We chose this camera for two reasons. First, we knew by its age that it is only a USB1.1 compliant camera, meaning that it runs at a lower data rate than more modern cameras. As other areas of the Kwan-Truc have stricter bandwidth restrictions, this lower rate was actually easier to work with. The second reason that we chose this camera is because we thought that its interface would be easier to reverse engineer because the camera would have less options.

Figure 2: The IBM Webcam

As a nice side-effect of choosing an older camera, we were able to find an open-source Linux driver (www.linux-usb.org/ibmcam) that had some details about how to initialize IBM webcams and activate some of their features. Unfortunately, some of the details about the initialization commands were either unclear or missing, so we had to resort to using a USB sniffer (Source USB) to discover these missing pieces. In the process, we found that the initialization sequence in the Linux driver did not match up very well with what we observed from the Windows driver (through SourceUSB). Because of this, much of the reverse engineering work had to be by us.

We had to make one minor physical modification to the camera itself to get it to work a little better for our project. The brightness of the camera is normally controlled by software, but we were unable to identify or reverse engineer the necessary command sequence to control it. As the default brightness setting results in very whitewashed video, we installed a filter (made of a small piece of an anti-static bag) in front of the lens to manually darken the image. This worked very well for a well-lit laboratory situation, but was not enough for a sunny day outside. To resolve the light balancing problem, we needed to attach another layer or two of filtering on the outside of the camera.

2.2. Microcontroller

The microcontroller in the Kwan-Truc system only has a couple requirements. First, it must be able to communicate via USB with the IBM webcam. Second, it must be able to communicate via RS-232 with the wireless transceiver. In order to meet these requirements, we selected the MCF52221, a 32-bit Coldfire based processor from Freescale Semiconductor. This processor has a USB On-The-Go (OTG) port that is capable of acting as a USB host or a USB peripheral, although it is only used as a USB host in the Kwan-Truc system. It also has three RS-232 ports.

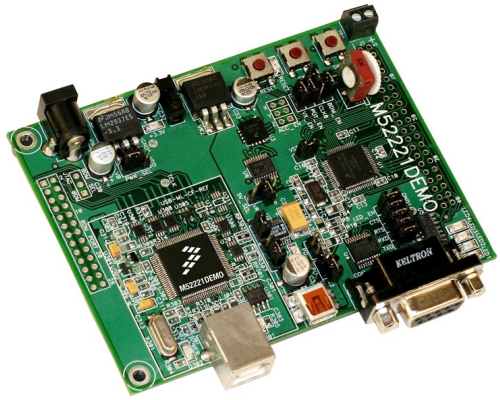

Figure 3: The Freescale M52221DEMO Board

The MCF52221 is available pre-mounted on Freescale's M52221DEMO evaluation board. This board is integrated into the CodeWarrior IDE, which allows for simple application development and debugging. Also, the M52221DEMO provides access to the USB and RS-232 ports via standard connectors, rather than having to wire directly to the pins. This made it very simple to connect the camera and the transceiver, because we could just use standard cables.

When received, the jumpers on the M52221DEMO are configured to receive power from a

USB cable instead of from an external power supply. In order to power the webcam from the board, it is necessary to use an external power supply. As such, the PWR_SEL jumper must be moved to the VDD side and the 5V_SEL jumper must be moved to the VR1 side.

There is a small quirk with using the M52221DEMO board. In order for a connected USB device (such as the webcam) to be recognized properly, the CTS line on the main RS-232 port must be held at low voltage. This can either be achieved by removing the CTS jumper and grounding the CTS pin that is closer to the edge of the board, or by connecting the board to another serial device which has flow control activated.

2.3. *Wireless Transceiver*

The requirement for our wireless transceiver are that it must be able to receive RS-232 data from the microcontroller and broadcast it to the laptop via 802.11. The Quatech wireless receiver (Part Number: WLNG-AN-DP101) that we used provides this interface between the microcontroller and the laptop. The data rate between the microcontroller and the wireless transceiver is 115200 bps, and hardware flow control (RTS, CTS) is enabled. The physical connection is made with a standard null modem cable (with appropriate flow control wires).

Figure 4: The WLNG-AN-DP101 Wireless Transceiver

When configured properly, the wireless transceiver acts as an automatic serial to wireless bridge, taking the data it receives from the RS-232 connection and sending it out wirelessly via 802.11. However, when the WLNG-AN-DP101 is first received, it is not set up to automatically transfer data in this manner. To properly configure it, you must connect it to the RS-232 port of another computer and use communication software such as HyperTerminal to talk to it. The sequence

of commands that is then necessary for configuring the transceiver is detailed below.

auth dpac dpac

This command allows you to access the settings of the wireless transceiver. The default user name is 'dpac', and the default password is 'dpac'. We never changed this user name or password, as we did not design the Kwan-Truc to be secure.

wl-ssid KwanTruc

This command sets the wireless SSID (Service Set identifier) to be KwanTruc. This was done to make the Kwan-Truc recognizable in the wireless networks list on the host laptop.

wl-type p

This command sets the networking style of the wireless transceiver to ad hoc. This eliminates the need for additional hardware, such as a wireless access point.

wl-chan 1

This command sets the wireless communications channel to 1. This was an arbitrarily chosen value, as any value from 1 to 11 should work equally well.

wl-dhcp 0

This command disables DHCP (Dynamic Host Configuration Protocol). If activated, it would interfere with the ad hoc nature of our wireless network.

wl-ip 192.168.10.150

This command sets the IP (Internet Protocol) address of the transceiver. A variety of values could work here, as long as the laptop software is configured to use this IP address when it attempts to connect to the Kwan-Truc.

wl-subnet 255.255.255.0

This command sets the subnet mask for the transceiver.

wl-tunnel 1

This command enables the wireless 'tunnel', which is the data bridge from the RS-232 input to the 802.11 output.

wl-tunnel-port 8023

This command selects the port to which the transceiver will output its data.

serial-default listen

This command sets the default startup behavior of the wireless transceiver to 'listen' mode. In listen mode, the transceiver waits for another device (in our case, the laptop) to initiate a connection to the port specified above. The alternative is pass mode, in which the transceiver is given an IP address and port number that it actively tries to access.

bit-rate 115200

This command sets the data transfer bit rate. The microcontroller and wireless transceiver must use the same bit-rate for the RS-232 link to work.

flow h

This command turns on hardware flow control.

commit

This command saves the above changes, so that they will remain after restarting the transceiver.

restart

This command restarts the transceiver, with the above changes in place.

Once this configuration is complete, the wireless transceiver will automatically perform its duties when it is powered up. This is especially nice, because it removes the burden of writing a

routine in the microcontroller to initialize the wireless transceiver.

2.4. Power

The input voltage range for our microcontroller evaluation board is 6-15 VDC, and the input voltage range for our transceiver board is 9-15 VDC. This made the option of using a 9 volt battery attractive, as this would have been lightweight and easily replaceable. Unfortunately, even a fully chared 9 volt battery can only run the full Kwan-Truc system for a few minutes. In order to accommodate the higher power requirements, we used 8 rechargeable NiMH (Nickel-metal hydride) AA batteries connected in series. This combination yielded a voltage of about 10.5 VDC, and was able to power our device for at least 2.5 hours.

To connect these batteries, we used a standard battery pack to hold all eight batteries and connect them in series. This pack contains a standard 9 volt battery lead to which we could easily connect a power cable. The cable itself was simply two power jacks that were soldered in parallel to the batteries. One of these jacks plugs into the wireless transceiver board and the other one plugs into the microcontroller board.

3. Software Design

There are only two pieces of software in the Kwan-Truc system which we had to write. The first is the laptop software, which is a Java GUI that displays the video. The second is the software on the microcontroller that handles the flow of data from the camera to the wireless transceiver.

3.1. Laptop GUI

The Java GUI is fairly simple. Conceptually, it just takes video information one line at a time from a VideoStream until there is a complete frame, then displays the frame. The video frames from the webcam are normally interlaced, but due to bandwidth restrictions we could only send one of

the two frames that are supposed to be interlaced. As such, the frame that is received by the GUI is only half as tall as it should be. Also, the resolution is quite low, making for a very small image. In an attempt to improve the video quality, the VideoPanel class expands the image by a factor of 4 in width and 8 in height, performing linear interpolation at the same time. This makes the image less pixelated, and greatly improves the quality of the video displayed on the screen.

The GUI has three options for its video source: the wireless interface, a direct serial connection, or a file. When used in conjunction with the Kwan-Truc system, the wireless interface is the source of video. In wireless mode, there is an option to record the live video stream. If you choose this option, you are asked to specify where to save the video. Later, this video can be played back by selecting "File" as your video source and choosing the saved file.

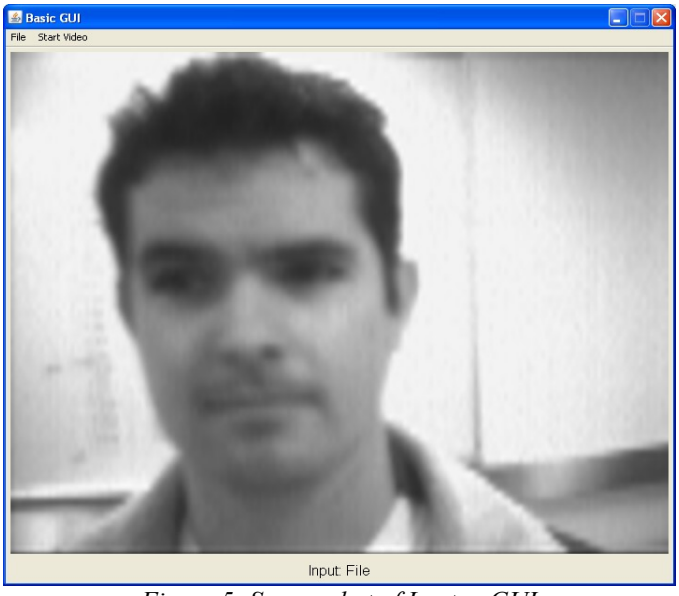

Figure 5: Screen shot of Laptop GUI

As the wireless transceiver may not always be available, there is a way to stream video directly from the microcontroller to the GUI. To do this, the microcontroller's RS-232 port is connected directly to a serial port on the computer with a standard serial cable. Then you can select "Serial" as your video source and test the functionality of the video system independent of the wireless transceiver.

3.2. Microcontroller

The logical flow of data through the microcontroller is conceptually very simple. The first state of the software is waiting for a USB device to connect. Once a device connects and sends its device

descriptor, the software determines if it is the IBM webcam and proceeds to the initialization state. After the camera is fully initialized, the video stream is started.

Processing the video stream is more difficult. Because USB is so much faster than RS-232, it is necessary to save an entire frame from the camera before sending it to the wireless transceiver. Fortunately, there was just barely enough RAM on the microcontroller to store half of a frame, but only in grayscale. Thus, there is some code to dump the extra color information and find the next frame header.

The microcontroller code was developed using the CodeWarrior IDE (version 7.0 for Coldfire architectures) downloadable from Freescale's website. To recreate the project, start with the default project for the M52221DEMO and make sure that it runs on the board. Then paste in the source code from Appendix C and build the INTERNAL_FLASH target. The software can then be loaded into the flash on the microcontroller using the flash programmer within the CodeWarrior IDE.

4. Lessons Learned

The biggest lesson we have taken from this project is that when choosing hardware, the presence of a particular interface does not always guarantee standard compliance. For example, the wireless transceiver advertises that it supports SPI communication, but when we had trouble getting this to work, we learned that they do not support it beyond the fact that they attempted to implemented it. Beyond that, we found out that the highest wireless data rate supported was far below what we had hoped for. We were hoping for a data rate of a few megabits (easily supported by 802.11), but the transceiver is not capable of even .5 Mbs.

Another important lesson we learned is that RC helicopters are much more difficult to fly when they are carrying a load. We should have started with small loads in order to get used to the extra weight before trying larger loads.

To improve the Kwan-Truc, we would need to have a better camera and a microcontroller with more RAM. A better camera would include features such as hardware white balancing, auto focusing, and hardware motion compensation (or a higher shutter speed). The motion compensation is probably the most important feature, but webcams are not generally designed with that goals in mind because they are not meant to be moving around. So, we would likely have to find a different type of camera.

5. Conclusions

Overall the Kwan-Truc works very well, although we had originally hoped for better video quality. The final video feed only had a frame rate of about .7 frames per second, was limited to grayscale, and was noticeably delayed (latency of about 2 seconds). However, the major goal of the project, which was to create an easily portable wireless datalink, was achieved. In fact, on the day of the final demonstration, we were able to get the whole system working on a fresh laptop in less than 5 minutes.

6. References and Acknowledgements

We would like to thank Freescale Semiconductor for donating two M52221DEMO evaluation boards to our project, as this greatly simplified the construction of the Kwan-Truc hardware. Also, the starter USB code which they have available on their website (www.freescale.com) was invaluable in understanding the proper usage of the USB OTG interface on the microcontroller.

We would also like to thank L-3 Communications for purchasing and lending us the RC helicopter for our project. Although the engineering work was technically complete without the helicopter, it was much more fun to test on the target platform.

We also acknowledge the open source USB IBM C-It Video Camera driver for Linux

(www.linux-usb.org/ibmcam) for giving some general directions to help out with the reverse engineering of the webcam interface. Without their previous work it would have been much harder to understand the video formatting and some of the features of the camera.

7. Appendix A: Bill of Materials

Device	Model	Manufacturer	Cost
Webcam	KSX-X9903	IBM	N/A
Microcontroller	M52221	Freescale	Donated by Freescale
Transceiver	WLNG-AN-DP101 Evaluation Board	Quatech	$300.00
Batteries	23-525	Radio Shack	$40 (2 sets of 4 ea.)
Helicopter	TREX-600	Align	Provided by L3

8. Appendix B: Part Schematics

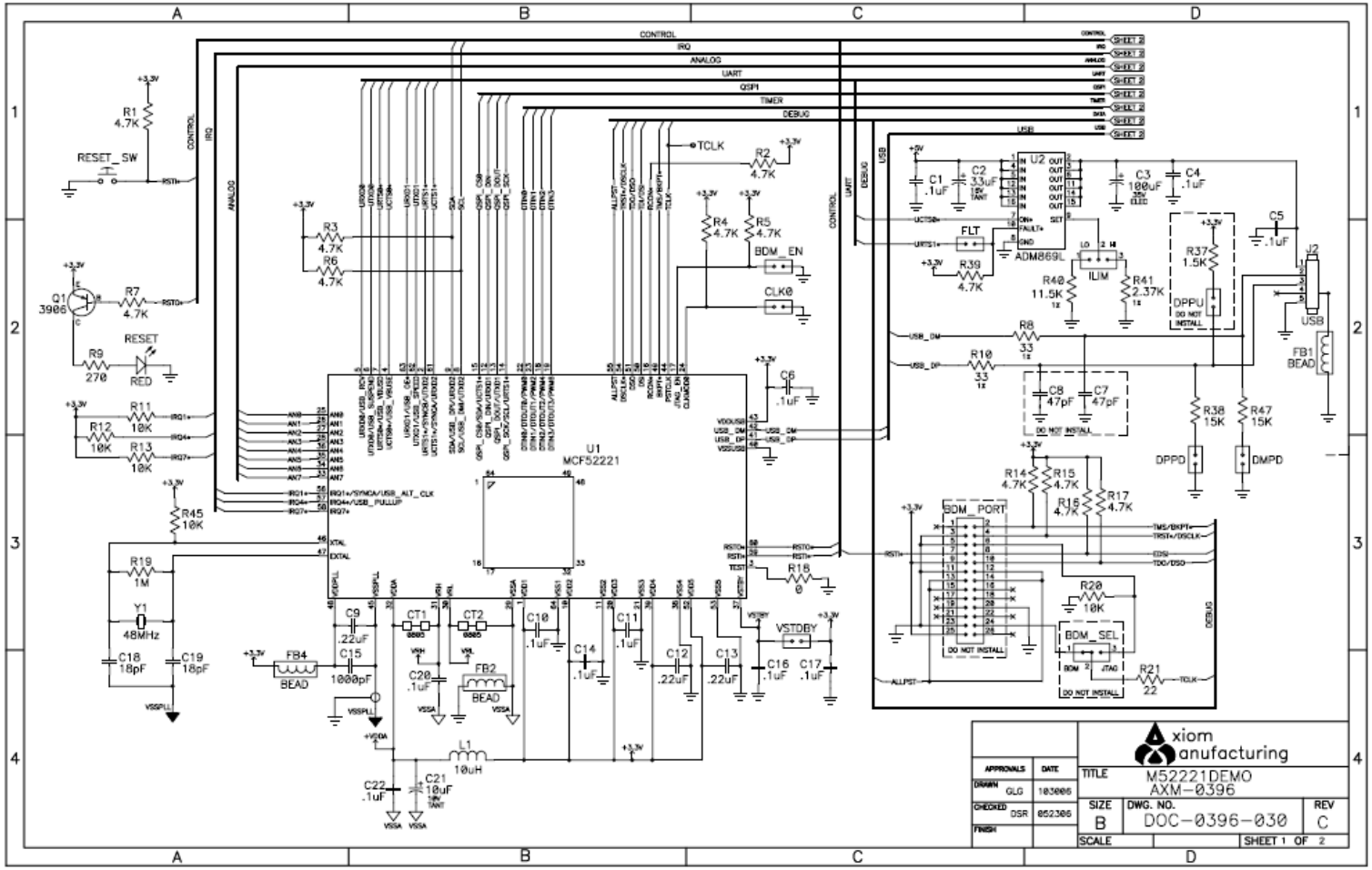

Figure 6: M52221DEMO Schematic- Sheet 1

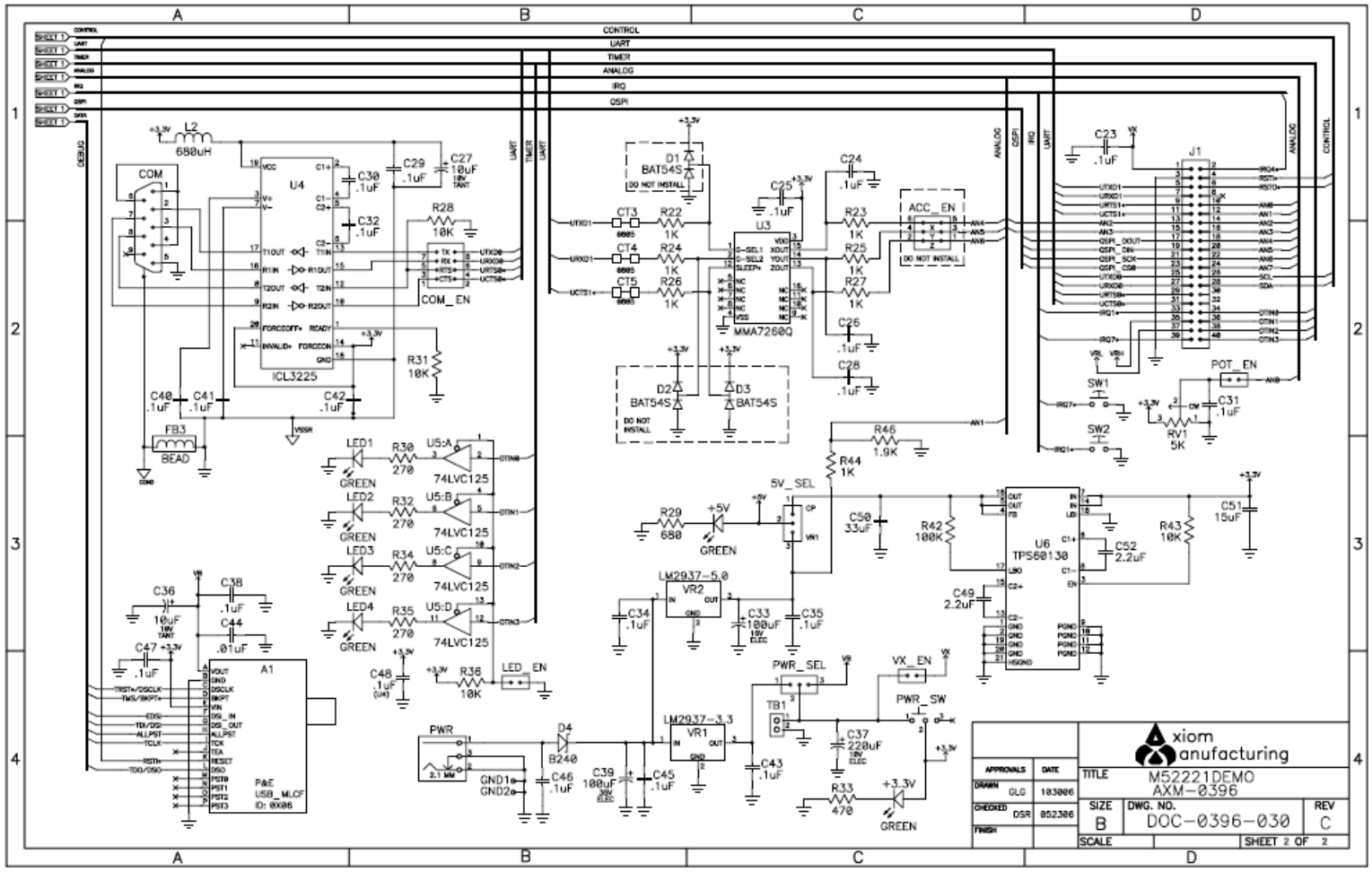

Figure 7: M52221DEMO Schematic - Sheet 2

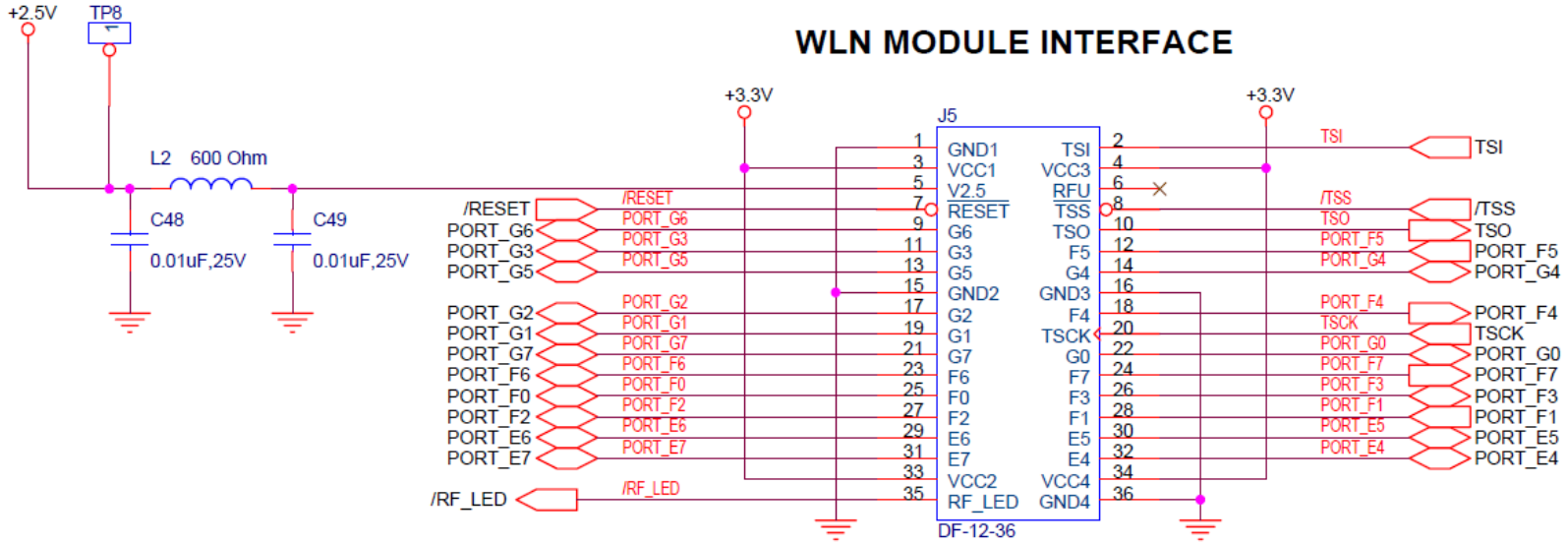

Figure 8: Wireless Transceiver Pin out

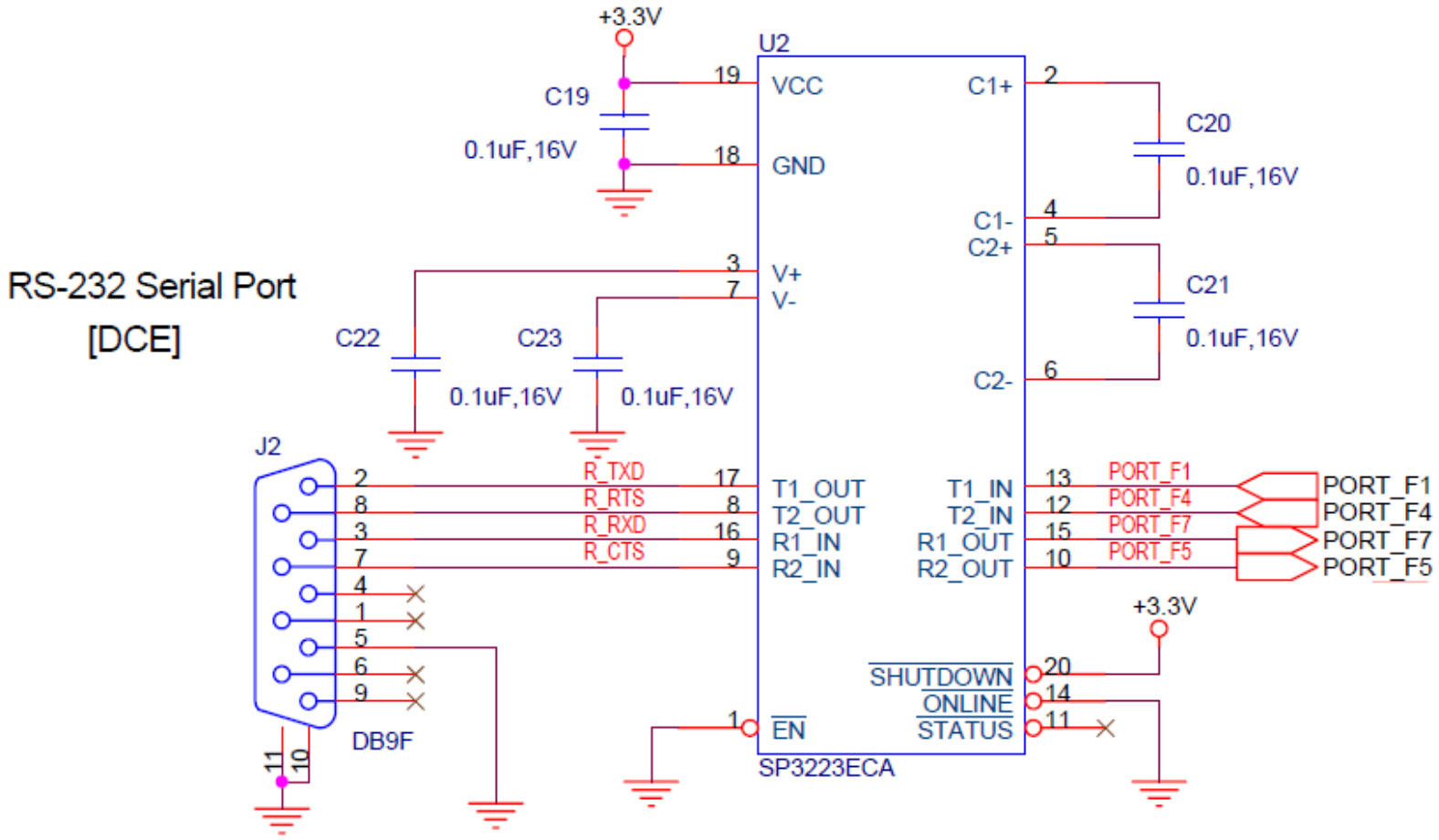

Figure 9: Wireless Transceiver Serial Port Connection

9. Appendix C: Source Code

9.1. Laptop GUI

9.1.1. GUI.java

```
package display;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.UIManager;
import javax.swing.filechooser.FileNameExtensionFilter;

import serial.FileToStream;
import serial.SerialToStream;
import serial.SocketToStream;
import serial.VideoStream;

public class GUI {
  static int width = 176;

  static int height = 144;

  // Initialize all swing objects.
  private JFrame f = new JFrame("Kwan-Truc GUI"); // create Frame

  private JPanel pnlNorth = new JPanel(); // video section

  private JPanel pnlSouth = new JPanel(); // label section

  // Label for information
  private JLabel label = new JLabel();

  private JButton recordButton = new JButton("Record");

  // Canvas to display video
  private VideoPanel video = new VideoPanel(width, height);

  // The thread that is grabbing the video
  private VideoStream videoThread;

  // Menus
  private JMenuBar mb = new JMenuBar(); // Menubar

  private JMenu mnuFile = new JMenu("File"); // File Entry on Menu bar
```

```
private JMenuItem mnuItemQuit = new JMenuItem("Quit"); // Quit sub item

private JMenu mnuVideo = new JMenu("Start Video"); // Video Menu

private JMenuItem mnuItemWireless = new JMenuItem("Wireless"); // Wireless
                                                                // sub item

private JMenuItem mnuItemSerialPort = new JMenuItem("Serial Port"); // Serial
                                                                    // Port
                                                                    // sub
                                                                    // item

private JMenuItem mnuItemFromFile = new JMenuItem("From File"); // From File
                                                                // subitem

// codec
public YUVtoRGB codec = new YUVtoRGB(width, height, video);

/** Constructor for the GUI */
public GUI() {
  // Needed to see menus when using VideoCanvas
  JPopupMenu.setDefaultLightWeightPopupEnabled(false);

  // Set menubar
  f.setJMenuBar(mb);

  // Build Menus
  mnuFile.add(mnuItemQuit); // Create Quit line
  mnuVideo.add(mnuItemWireless);
  mnuVideo.add(mnuItemSerialPort);
  mnuVideo.add(mnuItemFromFile);
  mb.add(mnuFile); // Add Menu items to form
  mb.add(mnuVideo);

  // Construct South Panel
  pnlSouth.add(label);
  label.setText("Choose a video input");
  label.setFont(new Font("Arial", Font.PLAIN, 16));
  pnlSouth.add(recordButton);
  recordButton.setVisible(false);
  try {
    Image im = ImageIO.read(new File("images\\recordOff.png"));
    im = im.getScaledInstance(16, 16, 0);
    recordButton.setIcon(new ImageIcon(im));
    im = ImageIO.read(new File("images\\record.png"));
    im = im.getScaledInstance(16, 16, 0);
    recordButton.setSelectedIcon(new ImageIcon(im));
    recordButton.setIconTextGap(8);
  } catch (IOException e) {
  }
  recordButton.setToolTipText("Click here to start recording");

  // Add video canvas
  pnlNorth.add(video);

  // Setup Main Frame
  f.getContentPane().setLayout(new BorderLayout());
  f.getContentPane().add(pnlNorth, BorderLayout.NORTH);
  f.getContentPane().add(pnlSouth, BorderLayout.SOUTH);
  f.setResizable(false);

  // Allows the Swing App to be closed
  f.addWindowListener(new ListenCloseWdw());

  // Add Menu and Button listener
  mnuItemQuit.addActionListener(new ListenMenuQuit());
  recordButton.addActionListener(new ListenRecordButton());
  ListenVideoMenu lvm = new ListenVideoMenu();
  mnuItemFromFile.addActionListener(lvm);
```

```
  mnuItemWireless.addActionListener(lvm);
  mnuItemSerialPort.addActionListener(lvm);
}

public class ListenMenuQuit implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.exit(0);
  }
}

private class ListenRecordButton implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    if (!recordButton.isSelected()) {
      JFileChooser chooser = new JFileChooser();
      chooser.setDialogTitle("Record As...");
      chooser.setDialogType(JFileChooser.SAVE_DIALOG);
      chooser.setMultiSelectionEnabled(false);
      chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
      chooser.setFileFilter(new FileNameExtensionFilter(
          "Kwan-Truc Video file (*.ktv)", "ktv"));
      if (JFileChooser.APPROVE_OPTION == chooser.showSaveDialog(f)) {
        String filename = chooser.getSelectedFile().getAbsolutePath();
        if (!filename.endsWith(".ktv")) {
          filename += ".ktv";
        }
        File file = new File(filename);
        if (videoThread != null) {
          videoThread.setOutputFileAndStartRecord(file);
          recordButton.setSelected(true);
        }
      }
    } else {
      recordButton.setSelected(false);
      videoThread.stopRecording();
    }
  }
}

private class ListenVideoMenu implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    VideoStream tempThread = null;
    Object o = e.getSource();
    recordButton.setSelected(false);
    if (o.equals(mnuItemFromFile)) {
      JFileChooser chooser = new JFileChooser();
      chooser.setDialogTitle("Choose a video file");
      chooser.setDialogType(JFileChooser.OPEN_DIALOG);
      chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
      chooser.setFileFilter(new FileNameExtensionFilter(
          "Kwan-Truc Video file (*.ktv)", "ktv"));
      if (JFileChooser.APPROVE_OPTION == chooser.showOpenDialog(f)) {
        File file = chooser.getSelectedFile();
        tempThread = new FileToStream(file, codec, width, height);
        label.setText("Input: File");
        recordButton.setVisible(false);
      }
    } else if (o.equals(mnuItemSerialPort)) {
      tempThread = new SerialToStream(codec, width, height);
      label.setText("Input: Serial Port");
      recordButton.setVisible(true);
    } else if (o.equals(mnuItemWireless)) {
      tempThread = new SocketToStream(codec, width, height);
      label.setText("Input: Socket");
      recordButton.setVisible(true);
    } else {
      System.out.println("What button did you hit?!");
      return;
    }

    // Don't change threads unless needed
```

```
      if (tempThread != null) {
        if (videoThread != null) {
          if (!videoThread.getClass().equals(tempThread.getClass())
              || videoThread instanceof FileToStream) {
            videoThread.killThread();
            codec.reset();
          }
        }

        videoThread = tempThread;
        videoThread.start();
      }
    }
  }

  public class ListenCloseWdw extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }
  }

  public void launchFrame() {
    // Display Frame
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.pack(); // Adjusts panel to components for display
    f.setVisible(true);
  }

  public static void main(String args[]) {
    // Set System L&F
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
      e.printStackTrace();
    }

    GUI gui = new GUI();

    gui.launchFrame();
  }
}
```

9.1.2. VideoPanel.java

```
package display;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;

import javax.swing.JPanel;

public class VideoPanel extends JPanel {

  static final long serialVersionUID = 0;

  private int scale;

  private int width;

  private int height;

  private BufferedImage i;

  public VideoPanel(int width, int height) {
```

```
    super();
    scale = 4;
    this.width = width;
    this.height = height;
    this.setSize(this.width * scale, this.height * scale);
    this
        .setPreferredSize(new Dimension(this.width * scale, this.height * scale));

    i = new BufferedImage(width * scale, height * scale,
        BufferedImage.TYPE_BYTE_GRAY);
  }

  public void paint(Graphics g) {
    if (i != null)
      g.drawImage(i, 0, 0, this);
    else {
      for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
          g.setColor(Color.BLACK);
          g.fillRect(x * scale, y * scale, scale, scale);
        }
      }
    }
  }

  public void paintFrame(Graphics g, byte[][] array, byte[][] gArray,
      byte[][] bArray) {
    int yEnd = array.length;
    int xEnd = array[0].length;
    for (int y = 0; y < yEnd; y++) {
      for (int x = 0; x < xEnd; x++) {
        int pixel = array[y][x] & 0xff;
        g.setColor(new Color(pixel, pixel, pixel));
        g.fillRect(x, y, 1, 1);
      }
    }
  }

  public void paintFrame(Graphics g, BufferedImage image) {
    Graphics2D g2d = null;
    try {
      g2d = i.createGraphics();
      g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
          RenderingHints.VALUE_INTERPOLATION_BICUBIC);
      // g2d.setRenderingHint(RenderingHints.KEY_DITHERING,
      // RenderingHints.VALUE_DITHER_ENABLE);
      // g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
      // RenderingHints.VALUE_ANTIALIAS_ON);
      AffineTransform at = AffineTransform.getScaleInstance(scale, scale * 2);
      g2d.drawRenderedImage(image, at);
    } finally {
      if (g2d != null)
        g2d.dispose();
    }
    g.drawImage(i, 0, 0, this);
  }
}
```

9.1.3. YUVtoRGB.java

```
package display;

import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;

public class YUVtoRGB {

  // stores the height and width of the frame
  private int height;
```

```
  private BufferedImage image;

  private WritableRaster raster;

  // private int width;

  // stores the current line being sent to our decoder, so that at a full frame,
  // the image will refresh

  private int currLine;

  // byte yr[][];// = new byte[144][176];
  private VideoPanel display;

  public YUVtoRGB(int width, int height, VideoPanel display) {
    this.currLine = 0;
    this.height = height;
    // this.width = width;
    this.display = display;
    image = new BufferedImage(width, height, BufferedImage.TYPE_BYTE_GRAY);
    raster = image.getRaster();
    // yr = new byte[height][width];
  }

  public void reset() {
    currLine = 0;
  }

  // input here should be one line of grayscale pixels, the lenght
  // of the line should be the width of the frame, and every (height/2)
  // lines recieved means one frame has been sent
  public void lineInputGrayscale(int[] line) {
    raster.setPixels(0, currLine++, line.length, 1, line);
    // raster.setPixels(0, currLine++, line.length, 1, line);
    // currLine+=2;

    if (currLine >= height / 2) {
      display.paintFrame(display.getGraphics(), image);
      currLine = 0;
    }
  }
}
```

9.1.4. FileToStream.java

```
package serial;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.SocketException;

import display.YUVtoRGB;

public class FileToStream extends Thread implements VideoStream {

  private int width;

  private int height;

  private YUVtoRGB decoder;

  private FileInputStream in;

  private File file;

  public FileToStream(File file, YUVtoRGB decoder, int width, int height) {
    this.file = file;
```

```
  this.width = width;
  this.height = height;
  this.decoder = decoder;
}

public void killThread() {
  this.interrupt();
  try {
    if (in != null) {
      in.close();
    }
  } catch (IOException e) {
  }
}

public void run() {
  try {
    in = new FileInputStream(file);

    byte[] readBuffer = new byte[width * height / 2 + 2];
    int[] line = new int[width];
    int offset = 0;
    while (true) {
      // Fill up a frame worth of data
      int numBytes = 0;
      int badBytes;
      do {
        badBytes = 0;
        while (numBytes < readBuffer.length) {
          if (this.isInterrupted()) {
            throw new InterruptedException("");
          }
          if (in.available() > 0)
            numBytes += in.read(readBuffer, numBytes, readBuffer.length
                - numBytes);
        }

        // Find the first 00FF in the buffer
        for (int i = 0; i < numBytes; i++) {
          if (readBuffer[i] == 0x00 && readBuffer[i + 1] == -1) {
            // If it wasn't at beginning, shift everything to front of buffer
            if (i != 0) {
              for (int j = 0; j < numBytes - i; j++) {
                readBuffer[j] = readBuffer[i + j];
              }
            }
            break;
          } else {
            badBytes++;
          }
        }
        numBytes -= badBytes;
      } while (badBytes > 0); // If anything was thrown out above
      // then go fill buffer again.

      // Now that you have a full frame and the first two bytes are
      // 0x00ff, put the rest of the data into the picture
      offset = 2;
      for (int y = 0; y < height / 2; y++) {
        for (int x = 0; x < width; x++) {
          // we still have data in our buffer, keep going
          line[x] = readBuffer[offset++] & 0xff;
        }
        // i have a line, send it to the decoder
        decoder.lineInputGrayscale(line);
      }

      sleep(1500);
    }
  } catch (SocketException e) {
```

```
      // Do nothing special
    } catch (IOException e) {
      // Do nothing special
    } catch (InterruptedException e) {
      // Do nothing special
    } finally {
      try {
        if (in != null) {
          in.close();
        }
      } catch (IOException e) {
        // Do nothing special
      }
    }
  }

  // The following functions are only included for interface compliance
  public void setOutputFileAndStartRecord(File file) {
  }

  public void stopRecording() {
  }

}
```

9.1.5. SerialToStream.java

```
package serial;

import gnu.io.CommPortIdentifier;
import gnu.io.NoSuchPortException;
import gnu.io.PortInUseException;
import gnu.io.SerialPort;
import gnu.io.UnsupportedCommOperationException;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

import display.YUVtoRGB;

public class SerialToStream extends Thread implements VideoStream {

  private int width;

  private int height;

  private YUVtoRGB decoder;

  private SerialPort serialPort;

  private FileOutputStream out;

  private InputStream in;

  public SerialToStream(YUVtoRGB decoder, int width, int height) {
    this.width = width;
    this.height = height;
    this.decoder = decoder;
  }

  public void killThread() {
    this.interrupt();
    try {
      if (serialPort != null) {
        serialPort.close();
      }
```

```
    if (in != null) {
      in.close();
    }
    stopRecording();
  } catch (Exception e) {
  }
}

public void run() {
  CommPortIdentifier portId;
  try {
    portId = CommPortIdentifier.getPortIdentifier("COM1");
    if (portId.isCurrentlyOwned()) {
      throw new PortInUseException();
    }
    serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
    in = serialPort.getInputStream();

    // activate the DATA_AVAILABLE notifier
    serialPort.notifyOnDataAvailable(true);
    serialPort.setSerialPortParams(115200, SerialPort.DATABITS_8,
        SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
    serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_RTSCTS_OUT);

    // we get here if data has been received
    byte[] readBuffer = new byte[width * height / 2 + 2];
    int[] line = new int[width];
    int offset = 0;
    while (true) {
      // Fill up a frame worth of data
      int numBytes = 0;
      int badBytes;
      do {
        badBytes = 0;
        while (numBytes < readBuffer.length) {
          if (this.isInterrupted()) {
            throw new InterruptedException("");
          }
          if (in.available() > 0)
            numBytes += in.read(readBuffer, numBytes, readBuffer.length
                - numBytes);
        }

        // Find the first 00FF in the buffer
        for (int i = 0; i < numBytes; i++) {
          if (readBuffer[i] == 0x00 && readBuffer[i + 1] == -1) {
            // If it wasn't at beginning, shift everything to front of buffer
            if (i != 0) {
              for (int j = 0; j < numBytes - i; j++) {
                readBuffer[j] = readBuffer[i + j];
              }
            }
            break;
          } else {
            badBytes++;
          }
        }
        numBytes -= badBytes;
      } while (badBytes > 0); // If anything was thrown out above
      // then go fill buffer again.

      // Now that you have a full frame and the first two bytes are
      // 0x00ff, put the rest of the data into the picture
      offset = 2;
      for (int y = 0; y < height / 2; y++) {
        for (int x = 0; x < width; x++) {
          // we still have data in our buffer, keep going
          line[x] = readBuffer[offset++] & 0xff;
        }
        // i have a line, send it to the decoder
```

```
          decoder.lineInputGrayscale(line);
        }

        // Write everything to a file too
        synchronized (this) {
          if (out != null)
            out.write(readBuffer, 0, numBytes);
        }
      }
    } catch (InterruptedException e) {
      // Do nothing special
    } catch (UnsupportedCommOperationException e) {
      e.printStackTrace();
    } catch (NoSuchPortException e) {
      System.out.println("COM1 doesn't exist");
    } catch (PortInUseException e) {
      System.out.println("Port busy");
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (null != serialPort)
        serialPort.close();
      if (in != null) {
        try {
          in.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      stopRecording();
    }
  }

  public void setOutputFileAndStartRecord(File file) {
    synchronized (this) {
      try {
        out = new FileOutputStream(file);
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      }
    }
  }

  public void stopRecording() {
    synchronized (this) {
      if (out != null)
        try {
          out.close();
          out = null;
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
  }
}
```

9.1.6. SocketToStream.java

```
package serial;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketException;
```

```
import display.YUVtoRGB;

public class SocketToStream extends Thread implements VideoStream {

  private int width;

  private int height;

  private YUVtoRGB decoder;

  private Socket socket;

  private InputStream in;

  private FileOutputStream out = null;

  public SocketToStream(YUVtoRGB decoder, int width, int height) {
    this.width = width;
    this.height = height;
    this.decoder = decoder;
    socket = new Socket();
  }

  public void killThread() {
    this.interrupt();
    try {
      if (socket != null) {
        socket.close();
      }
      if (in != null) {
        in.close();
      }
      stopRecording();
    } catch (IOException e) {
    }
  }

  public void run() {
    try {
      socket.connect(new InetSocketAddress("192.168.10.150", 8023));
      in = socket.getInputStream();

      // we get here if data has been received
      byte[] readBuffer = new byte[width * height / 2 + 2];
      int[] line = new int[width];
      int offset = 0;
      while (true) {
        // Fill up a frame worth of data
        int numBytes = 0;
        int badBytes;
        do {
          badBytes = 0;
          while (numBytes < readBuffer.length) {
            if (this.isInterrupted()) {
              throw new InterruptedException("");
            }
            if (in.available() > 0)
              numBytes += in.read(readBuffer, numBytes, readBuffer.length
                  - numBytes);
          }

          // Find the first 00FF in the buffer
          for (int i = 0; i < numBytes; i++) {
            if (readBuffer[i] == 0x00 && readBuffer[i + 1] == -1) {
              // If it wasn't at beginning, shift everything to front of buffer
              if (i != 0) {
                for (int j = 0; j < numBytes - i; j++) {
                  readBuffer[j] = readBuffer[i + j];
                }
              }
```

```
            break;
          } else {
            badBytes++;
          }
        }
        numBytes -= badBytes;
      } while (badBytes > 0); // If anything was thrown out above
      // then go fill buffer again.

      // Now that you have a full frame and the first two bytes are
      // 0x00ff, put the rest of the data into the picture
      offset = 2;
      for (int y = 0; y < height / 2; y++) {
        for (int x = 0; x < width; x++) {
          // we still have data in our buffer, keep going
          line[x] = readBuffer[offset++] & 0xff;
        }
        // i have a line, send it to the decoder
        decoder.lineInputGrayscale(line);
      }

      // Write everything to a file too
      synchronized (this) {
        if (out != null)
          out.write(readBuffer, 0, numBytes);
      }
    }
  } catch (SocketException e) {
    // Do nothing special
  } catch (InterruptedException e) {
    // Do nothing special
  } catch (IOException e) {
    e.printStackTrace();
  } finally {
    try {
      if (socket != null) {
        socket.close();
      }
      if (in != null) {
        in.close();
      }
      stopRecording();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}

public void setOutputFileAndStartRecord(File file) {
  synchronized (this) {
    try {
      out = new FileOutputStream(file);
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
  }
}

public void stopRecording() {
  synchronized (this) {
    if (out != null)
      try {
        out.close();
        out = null;
      } catch (IOException e) {
        e.printStackTrace();
      }
  }
}
}
```

9.1.7. VideoStream.java

```
package serial;

import java.io.File;


public interface VideoStream {
            public void killThread();
            public void start();
            public void setOutputFileAndStartRecord(File file);
            public void stopRecording();
}
```

9.2. Microcontroller Code

9.2.1. main_flash.c

```
/*
 * main implementation
 *
 */
#include "support_common.h" /* include peripheral declarations and more */
#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
/* Standard IO is only possible if Console or UART support is enabled. */
#include <stdio.h>
#define uart_init(...);
#else
/* Compile out all the printf and fflush in this file */
#include "uart_support.h"
#define printf(...);
#define fflush(arg);
#endif

#include "spi.h"
#include "usb_host.h"
#include "ibm_cam.h"

void enable_usb_pull_downs(void)
{
  MCF_USB_OTG_OTG_CTRL = MCF_USB_OTG_OTG_CTRL_DP_LOW | MCF_USB_OTG_OTG_CTRL_DM_LOW |
MCF_USB_OTG_OTG_CTRL_OTG_EN;
  MCF_GPIO_PQSPAR |= MCF_GPIO_PQSPAR_QSPI_CS2_USB_DM_PD | MCF_GPIO_PQSPAR_QSPI_CS3_USB_DP_PD;
}
#define busy_wait()

#define FRAME_WIDTH 176
#define FRAME_WIDTH_2X 352
#define FRAME_HEIGHT 72

unsigned char buffer[966];
unsigned char video_buffer[2 + FRAME_WIDTH * FRAME_HEIGHT];

int main(void)
{
  int i, offset, sent, skipped, rows, temp = 0;
  unsigned int received;
  device_info_t dev_inf;
  int dev_info_status;

            //Initialize the USB host
  Usb_Vbus_On();
  host_init();

  // initialize the UART MODULE
  //uart_init(0, SYSTEM_CLOCK_KHZ, 230400);
  uart_init(0, SYSTEM_CLOCK_KHZ, 115200);
```

```
//enable the attach interrupt
MCF_USB_OTG_INT_ENB |= MCF_USB_OTG_INT_STAT_ATTACH;

//printf("Host application started.\r\n");
fflush(stdout);

while(1)
{
  printf("Waiting for device...\r\n");
  fflush(stdout);

  while(!host_scan_for_device())
  {
    busy_wait();
  }

  printf("Device connected.\r\n");
  fflush(stdout);

  dev_info_status = get_device_info(&dev_inf);

  if(0 == dev_info_status)
  {
    if(is_ibm_cam(&dev_inf))
    {
      printf("IBM webcam detected. Initializing...\r\n");
      do_plugin_setup();
      do_program_start_setup();

      // This is supposedly the color balance, but it helped with brightness issues
      control_packet(0x22, 0x8);
      // This is supposedly the saturation.
      //control_packet(0x20, 0x7F);

      // This is another color balance thing that seemed to help
      control_packet_2(0x3, 0x4, 0xa);

      /* grab video and "print" it out here */
      while(host_has_device())
      {
        rows = 0;
        skipped = 0;
        sent = 0;

        while(2 != host_receive(buffer, 966u, 1u))
        {}
        //if(!buffer[0] == 0 || !buffer[1] == 255)
        //  break;

        video_buffer[0] = 0;
        video_buffer[1] = 255;
        offset = 2;

        while(rows < FRAME_HEIGHT)
        {
          received = host_receive(buffer, 966u, 1u);
          if( (received != ((hcc_u16)-1u)) && (received != 0) )
          {
            i = 0;
            while(buffer[i] == 0 && buffer[i+1] == 255)
                i+=2;
            for(; i < received; i++)
            {
              if(sent < FRAME_WIDTH)
              {
                      video_buffer[offset++] = buffer[i];
                sent++;
              }
              else
```

```
                {
                  skipped++;//=(received - i)%176
                  if(skipped == FRAME_WIDTH_2X)
                  {
                    skipped = 0;
                    sent = 0;
                    rows++;
                  }
                }
              } // end for(i = 0; i < received
            }// end host_receive
          } // end while(rows < FRAME_HEIGHT)

          for(i = 0; i < (FRAME_HEIGHT * FRAME_WIDTH + 2); i++)
          {
            uart_putchar(0, (char)video_buffer[i]);
          }

        } // end while(host_has_device())
      }
      else
      {
        printf("device is not a webcam\r\n");
        /*printf("VID: %d\r\n", dev_inf.vid);
        printf("PID: %d\r\n", dev_inf.pid);
        printf("REV: %d\r\n", dev_inf.rev);
        printf("CLAS: %d\r\n", dev_inf.clas);
        printf("SCLAS: %d\r\n", dev_inf.sclas);
        printf("PROTOCOL: %d\r\n", dev_inf.protocol);
        printf("NCFG: %d\r\n", dev_inf.ncfg);*/
      }
    }
    else
    {
      printf("Device enumeration failed. Replug it\r\n");
    }

    while(host_has_device())
    {
      busy_wait();
    }

            printf("Device Disconnected!!!\r\n");
            fflush(stdout);
  } // end while(1)
}
```

9.2.2. ibm_cam.h

```
/* This file contains all the driver code for the IBM webcam
 * Many thanks to the open source
 *    USB IBM C-It Video Camera driver
 *
 * This driver is based on earlier work of:
 *
 * (C) Copyright 1999 Johannes Erdfelt
 * (C) Copyright 1999 Randy Dunlap
 *
 * This version was written by Todd Hummel.
 */

#ifndef _IBM_CAM_H_
#define _IBM_CAM_H_

#include "types.h"
#include "usb_host.h"

#define IBMCAM_VENDOR_ID           0x0545
```

```
#define IBMCAM_PRODUCT_ID          0x8080


extern int is_ibm_cam(device_info_t* device);

extern int do_plugin_setup();

extern int do_program_start_setup();

extern void control_packet(hcc_u16 val1, hcc_u16 val2);

extern void control_packet_2(hcc_u16 val1, hcc_u16 val2, hcc_u16 val3);

#endif
```

9.2.3. ibm_cam.c

```
/* This file contains all the driver code for the IBM webcam
 * Many thanks to the open source
 *    USB IBM C-It Video Camera driver
 *
 * This driver is based on earlier work of:
 *
 * (C) Copyright 1999 Johannes Erdfelt
 * (C) Copyright 1999 Randy Dunlap
 *
 * This version was written by Todd Hummel.
 */

#include "ibm_cam.h"
#include "usb_host.h"
#include "usb_utils.h"

static unsigned short plugin_cmds[] =
{
            0x0, 0x100, /* led on */
            0xC0, 0x111,
            0xBC, 0x12C,
            0x1F, 0x12B,
            0x9B, 0x10F,
            0xBB, 0x10F,
            0xAA, 0x12D,
            0x0, 0x12F,
            0xD141, 0x124,
            0xAB, 0x12D,
            0xFF, 0x130,
            0xCD41, 0x124,
            0xFFFA, 0x124, //13

            0x20, 0x111,
            0xC0, 0x100  /* led off */  //15
};

static unsigned short program_start_cmds1[] =
{
  0x0, 0x100, //led on
  0xc0, 0x111, //set video size big?
  0xbc,     0x12c,
  0x1f, 0x12b,
  0x0, 0x108,
  0x1, 0x133,
  0x9b, 0x10f,
  0xbb, 0x10f,   //8

  0xaa, 0x12d,
  0x30, 0x12f,
  0xd141, 0x124,
  0x0, 0x127,
```

```
  0xfea8, 0x124, //13

  0xaa, 0x12d,
  0x4, 0x12f,
  0xd141, 0x124,
  0x0, 0x127,
  0xfb, 0x12e,
  0x0, 0x130,
  0x8a28, 0x124,
  0x5c, 0x12d,
  0x9, 0x12f,
  0xd145, 0x124,
  0xaa, 0x12e,
  0x12, 0x130,
  0x8a0a, 0x124,
  0xa, 0x12d,
  0x9545, 0x124,
  0xaa, 0x127,
  0x2a, 0x12e,
  0x0, 0x130,
  0x8a28, 0x124,
  0xffd5, 0x124, //33

  0xaa, 0x12d,
  0x2c, 0x12f,
  0xd141, 0x124,
  0x0, 0x127,
  0xfea8, 0x124, //38

  0x38, 0x119,
  0xc6, 0x107,
  0x3, 0x106 //41
};

static unsigned short program_start_cmds2[] =
{
  0xd0, 0x111, //set video size small?
  0xb9, 0x10a,
  0x1, 0x102,
  0x0, 0x104,
  0xe32c, 0x103,
  0xe324, 0x105, //6

  0xaa, 0x12d,
  0x16, 0x12f,
  0xd141, 0x124,
  0x2, 0x127,
  0xaa, 0x130,
  0x82a8, 0x124,
  0x14, 0x12d,
  0x2, 0x12f,
  0xd145, 0x124,
  0xaa, 0x12e,
  0x1a, 0x130,
  0x8a0a, 0x124,
  0x5a, 0x12d,
  0x9545, 0x124,

  0xaa, 0x127,
  0x18, 0x12e,
  0x4a, 0x130,
  0x8a28, 0x124,
  0xaa, 0x12f,
  0xd055, 0x124,
  0x1c, 0x127,
  0x6d, 0x12e,
  0xaa28, 0x124, //29

  0xaa, 0x12d,
  0x2e, 0x12f,
```

```
  0xd141, 0x124,
  0xc, 0x127,
  0xfea8, 0x124, //34

  0xaa, 0x12d,
  0x1e, 0x12f,
  0xd141, 0x124,
  0xc, 0x127,
  0x10, 0x12e,
  0x16, 0x130,
  0x8a28, 0x124,
  0xc, 0x12d,
  0x9545, 0x124,
  0xaa, 0x127,
  0x28, 0x12e,
  0x7, 0x130,
  0x8a28, 0x124,
  0xaa, 0x12f,
  0xd055, 0x124,
  0x26, 0x127,
  0x9a, 0x12e,
  0xaa28, 0x124,
  0xc0, 0x10c, // go camera go! //53

  0xaa, 0x12d,
  0x30, 0x12f,
  0xd141, 0x124,
  0x4, 0x127,
  0xfea8, 0x124  //58
};

/*****************************************************************************
 * send_control_veio
 * IN: val - the value
 *     idx - the index
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Sends a STP_TYPE_VENDOR packet.
 *****************************************************************************/
static int send_control_veio(hcc_u16 val, hcc_u16 idx)
{
  hcc_u8 setup[8];
  hcc_u8 retry=3;

  fill_setup_packet(setup, STP_DIR_OUT, STP_TYPE_VENDOR, STP_RECIPIENT_ENDPT, 0, val, idx, 0);

  return host_send_control(setup, setup, 0);
}

/*****************************************************************************
 * is_ibm_cam
 * IN: device - structure containing the USB device information
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Sends a STP_TYPE_VENDOR packet.
 *****************************************************************************/

int is_ibm_cam(device_info_t* device)
{
  return (IBMCAM_VENDOR_ID == device->vid && IBMCAM_PRODUCT_ID == device->pid)?1:0;
}

/*****************************************************************************
 * do_plugin_setup
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
```

```
 *   Sends a bunch of STP_TYPE_VENDOR packets to do initial setup.
 ****************************************************************************/
int do_plugin_setup()
{
  int i;
  set_config(1);
  /* if necessary, do a receive 0, 0x128 */
  for(i = 0; i < 26; i+=2)
              send_control_veio(plugin_cmds[i], plugin_cmds[i+1]);
            /* if necessary, do a receive 0 0x10e */
            for(;i < 30; i+=2)
              send_control_veio(plugin_cmds[i], plugin_cmds[i+1]);
}

/****************************************************************************
 * do_program_start_setup
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Sends more STP_TYPE_VENDOR packets to finish setup and start video.
 ****************************************************************************/
int do_program_start_setup()
{
  int i;
  for(i = 0; i < 82; i+=2)
              send_control_veio(program_start_cmds1[i], program_start_cmds1[i+1]);

            /* Select Interface 0, alt setting 1 (the isochronous pipe) */
            select_interface(0, 1);

            for(i = 0;i < 116; i+=2)
              send_control_veio(program_start_cmds2[i], program_start_cmds2[i+1]);

            /* Clear the halt on the isochronous pipe */
            clear_halt(0x1);

            /* Modify an endpoint in the host for use as the isochronous endpoint */
            host_modify_ep(1, EPTYPE_ISO, 0x1, 1, 0x3C6);
}

void control_packet(hcc_u16 val1, hcc_u16 val2)
{
  send_control_veio(0xaa, 0x12d);
  send_control_veio(val1, 0x12f);
  send_control_veio(0xd141, 0x124);
  send_control_veio(val2, 0x127);
  send_control_veio(0xfea8,         0x124);
}

void control_packet_2(hcc_u16 val1, hcc_u16 val2, hcc_u16 val3)
{
  send_control_veio(0xaa, 0x12d);
  send_control_veio(0x1e, 0x12f);
  send_control_veio(0xd141, 0x124);
  send_control_veio(val1, 0x127);
  send_control_veio(val2,           0x12e);
  send_control_veio(val3,           0x130);
  send_control_veio(0x8a28,         0x124);
  send_control_veio(val1,           0x12d);
  send_control_veio(0xf545,         0x124);
}
```

9.2.4. usb_utils.h

```
/****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
 *
```

```
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 ****************************************************************************/
#ifndef _USB_UTILS_H_
#define _USB_UTILS_H_

/* Standard descriptor types */
#define STDDTYPE_DEVICE        1u
#define STDDTYPE_CONFIGURATION 2u
#define STDDTYPE_STRING        3u
#define STDDTYPE_INTERFACE     4u
#define STDDTYPE_ENDPOINT      5u

/* Setup packet fields. */
#define STP_DIR_IN             (1u<<7)
#define STP_DIR_OUT            (0u<<7)
#define STP_TYPE_STD           (0u<<5)
#define STP_TYPE_CLASS         (1u<<5)
#define STP_TYPE_VENDOR        (2u<<5)
#define STP_RECIPIENT_DEVICE   (0<<0)
#define STP_RECIPIENT_IFC      (1<<0)
#define STP_RECIPIENT_ENDPT    (2<<0)
#define STP_RECIPIENT_OTHER    (3<<0)

/* Standard request values */
#define STDRQ_GET_STATUS         0u
#define STDRQ_CLEAR_FEATURE      1u
#define STDRQ_SET_FEATURE        3u
#define STDRQ_SET_ADDRESS        5u
#define STDRQ_GET_DESCRIPTOR     6u
#define STDRQ_SET_DESCRIPTOR     7u
#define STDRQ_GET_CONFIGURATION  8u
#define STDRQ_SET_CONFIGURATION  9u
#define STDRQ_GET_INTERFACE      10u
#define STDRQ_SET_INTERFACE      11u
#define STDRQ_SYNCH_FRAME        12u


#define DBUFFER_SIZE    128u

typedef enum {
  stderr_none=0,
  stderr_host,
  stderr_bad_desc
} std_error_t;

extern std_error_t std_error;

extern int get_dev_desc(void);
extern int get_cfg_desc(hcc_u8 ndx);
```

```
extern int set_address(hcc_u8 address);
extern int set_config(hcc_u8 cfg);
extern int select_interface(hcc_u8 intf, hcc_u8 altSetting);
extern int clear_halt(hcc_u8 ep);
extern int get_device_info(device_info_t *res);
extern int get_ifc_info(ifc_info_t *res, hcc_u16 offset);
extern hcc_u16 find_ifc_ndx(hcc_u8 ndx);
extern hcc_u16 find_ifc_csp(hcc_u8 class, hcc_u8 sclass, hcc_u8 protocol);
extern hcc_u16 find_descriptor(hcc_u8 type, hcc_u16 start, hcc_u8 next);
extern int get_ep_info(ep_info_t *res, hcc_u16 offset);
extern void fill_setup_packet(hcc_u8* dst, hcc_u8 dir, hcc_u8 type, hcc_u8 recipient,
                      hcc_u8 req, hcc_u16 val, hcc_u16 ndx, hcc_u16 len);
extern int get_cfg_info(cfg_info_t *res);
extern int set_ep0_psize(void);
extern int enumerate_device(void);
extern hcc_u8 dbuffer[DBUFFER_SIZE];
#endif
/***************************** END OF FILE *********************************/
```

9.2.5. usb_utils.c

```
/*****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 *****************************************************************************/
#include "types.h"
#include "usb_host.h"
#include "usb_utils.h"

/* Descriptors will be read to this buffer. */
hcc_u8 dbuffer[DBUFFER_SIZE];
std_error_t std_error;
/*****************************************************************************
 * fill_setup_packet
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Create data for a setup packet.
 *****************************************************************************/
void fill_setup_packet(hcc_u8* dst, hcc_u8 dir, hcc_u8 type, hcc_u8 recipient,
                      hcc_u8 req, hcc_u16 val, hcc_u16 ndx, hcc_u16 len)
{
  dst[0]=(hcc_u8)(dir | type | recipient);
  dst[1]=req;
  dst[2]=(hcc_u8)val;
```

```
  dst[3]=(hcc_u8)(val>>8);
  dst[4]=(hcc_u8)ndx;
  dst[5]=(hcc_u8)(ndx>>8);
  dst[6]=(hcc_u8)(len);
  dst[7]=(hcc_u8)(len>>8);
}

/*****************************************************************************
 * get_dev_desc
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Read device desriptor to dbuffer.
 *****************************************************************************/
int get_dev_desc(void)
{
   hcc_u8 setup[8];
   hcc_u8 retry=3;

   std_error=stderr_none;
   fill_setup_packet(setup, STP_DIR_IN, STP_TYPE_STD, STP_RECIPIENT_DEVICE,
   STDRQ_GET_DESCRIPTOR, (STDDTYPE_DEVICE<<8)|0, 0, 18);

   do {
     if (17 <= host_receive_control(setup, dbuffer, 0))
     {
       /* Check returned descriptor type and length (ignore extra bytes). */
       if ((USBDSC_TYPE(dbuffer) == STDDTYPE_DEVICE)
          && (USBDSC_LENGTH(dbuffer) <= 18))
       {
         return(0);
       }
     }
   } while(retry--);
   std_error=stderr_host;
   return(1);
}

/*****************************************************************************
 * get_cfg_desc
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Read device desriptor to dbuffer.
 *****************************************************************************/
int get_cfg_desc(hcc_u8 ndx)
{
   hcc_u8 setup[8];
   hcc_u16 length=5;
   hcc_u8 retry=3;

   std_error=stderr_none;
   do {
     fill_setup_packet(setup, STP_DIR_IN, STP_TYPE_STD, STP_RECIPIENT_DEVICE,
     STDRQ_GET_DESCRIPTOR, (hcc_u16)((STDDTYPE_CONFIGURATION<<8)|ndx), 0, length);
     if (length == host_receive_control(setup, dbuffer, 0))
     {
       /* Check returned descriptor type and length (ignore extra bytes) */
       if ((USBDSC_TYPE(dbuffer) == STDDTYPE_CONFIGURATION)
          && (USBDSC_LENGTH(dbuffer) <= 9))
       {
         length=RD_LE16(dbuffer+2);
         CMX_ASSERT(length < DBUFFER_SIZE);

         fill_setup_packet(setup, STP_DIR_IN, STP_TYPE_STD, STP_RECIPIENT_DEVICE,
         STDRQ_GET_DESCRIPTOR, (hcc_u16)((STDDTYPE_CONFIGURATION<<8)|ndx), 0, length);

         if (length == host_receive_control(setup, dbuffer, 0))
```

```
        {
          return(0);
        }
      }
    }
  }while(retry--);
  std_error=stderr_host;
  return(1);
}

/*****************************************************************************
 * set_ep0_psize
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Read device desriptor to dbuffer.
 *****************************************************************************/
int set_ep0_psize(void)
{
  hcc_u8 setup[8];

  std_error=stderr_none;
  fill_setup_packet(setup, STP_DIR_IN, STP_TYPE_STD, STP_RECIPIENT_DEVICE,
  STDRQ_GET_DESCRIPTOR, (STDDTYPE_DEVICE<<8) | 0, 0, 8);

  if (8 == host_receive_control(setup, dbuffer, 0))
  {
    if ((USBDSC_TYPE(dbuffer) == STDDTYPE_DEVICE)
       && (USBDSC_LENGTH(dbuffer) <= 18))
    {
      host_modify_ep(0, EPTYPE_CTRL, 0, 0, DEVDESC_PACKET_SIZE(dbuffer));
      return(0);
    }
  }
  std_error=stderr_host;
  return(1);
}

/*****************************************************************************
 * set_address
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Set device address.
 *****************************************************************************/
int set_address(hcc_u8 address)
{
  hcc_u8 setup[8];
  hcc_u8 retry=3;

  std_error=stderr_none;
  fill_setup_packet(setup, STP_DIR_OUT, STP_TYPE_STD, STP_RECIPIENT_DEVICE,
  STDRQ_SET_ADDRESS, address, 0, 0);

  do {
    if (0==host_send_control(setup, dbuffer, 0))
    {
      /* we need to wait maximum 50 mS to let the device change its address. */
      host_ms_delay(45);
      return(0);
    }
  }while(retry--);
  std_error=stderr_host;
  return(1);
}
/*****************************************************************************
 * set_config
 * IN:
```

```
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Set device configuration
 ******************************************************************************/
int set_config(hcc_u8 cfg)
{
  hcc_u8 setup[8];
  hcc_u8 retry=3;

  std_error=stderr_none;
  fill_setup_packet(setup, STP_DIR_OUT, STP_TYPE_STD, STP_RECIPIENT_DEVICE,
  STDRQ_SET_CONFIGURATION, cfg, 0, 0);
  do {
    if (0==host_send_control(setup, dbuffer, 0))
    {
      return(0);
    }
  } while(retry--);
  std_error=stderr_host;
  return(1);
}
/******************************************************************************
 * select_interface
 * IN: intf - The interface number to select
 *     altSetting - The alternate setting to use
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Set the interface to use given alternate setting number)
 ******************************************************************************/
int select_interface(hcc_u8 intf, hcc_u8 altSetting)
{
  hcc_u8 setup[8];
  hcc_u8 retry=3;

  std_error=stderr_none;
  fill_setup_packet(setup, STP_DIR_OUT, STP_TYPE_STD, STP_RECIPIENT_IFC,
  STDRQ_SET_INTERFACE, altSetting, intf, 0);
  do {
    if (0==host_send_control(setup, dbuffer, 0))
    {
      return(0);
    }
  } while(retry--);
  std_error=stderr_host;
  return(1);
}
/******************************************************************************
 * clear_halt
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Clears the halt on the given endpoint number (not endpoint handle)
 ******************************************************************************/
int clear_halt(hcc_u8 ep)
{
  hcc_u8 setup[8];
  hcc_u8 retry=3;

  std_error=stderr_none;
  /* STDRQ_CLEAR_FEATURE = request
   * 0 for value = ENDPOINT_HALT.
   * ep for index.
   * This clears endpoint halt on given pipe */
  fill_setup_packet(setup, STP_DIR_OUT, STP_TYPE_STD, STP_RECIPIENT_ENDPT,
  STDRQ_CLEAR_FEATURE, 0, ep, 0);
  do {
    if (0==host_send_control(setup, dbuffer, 0))
```

```
    {
      return(0);
    }
  } while(retry--);
  std_error=stderr_host;
  return(1);
}

/*****************************************************************************
 * get_device_info
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Get device identifycation info.
 *****************************************************************************/
int get_device_info(device_info_t *res)
{
   /* read descriptor from device. */
   std_error=stderr_none;
   if (get_dev_desc())
   {
     std_error=stderr_host;
     return(1);
   }

   /* give read values to caller */
   res->clas=DEVDESC_CLASS(dbuffer);
   res->sclas=DEVDESC_SCLASS(dbuffer);
   res->protocol=DEVDESC_PROTOCOL(dbuffer);
   res->rev=DEVDESC_REV(dbuffer);
   res->vid=DEVDESC_VID(dbuffer);
   res->pid=DEVDESC_PID(dbuffer);
   res->ncfg=DEVDESC_NCFG(dbuffer);

   return(0);
}

/*****************************************************************************
 * find_descriptor
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   find a descriptor in the loaded configuration info
 *****************************************************************************/
hcc_u16 find_descriptor(hcc_u8 type, hcc_u16 start, hcc_u8 next)
{
  hcc_u8 *p=dbuffer+start;

  CMX_ASSERT(USBDSC_TYPE(dbuffer) == STDDTYPE_CONFIGURATION);

  if (next)
  {
    p+=USBDSC_LENGTH(p);
  }

  while(p < dbuffer+CONFDESC_TOTLENGTH(dbuffer))
  {
    if(USBDSC_TYPE(p) == type)
    {
      return((hcc_u16)(p-dbuffer));
    }
    p+=USBDSC_LENGTH(p);
  }
  return((hcc_u16)-1u);
}
/*****************************************************************************
 * find_ifc_csp
 * IN:
```

```
 * OUT:
 * Assumptions: n/a
 * Description:
 *   find an interface in the loaded config descriptor
 ******************************************************************************/
/* Currently only able to find the first matching ifc. Add start. */
hcc_u16 find_ifc_csp(hcc_u8 class, hcc_u8 sclass, hcc_u8 protocol)
{
  int ifc;
  hcc_u8 *p=dbuffer;

  /* at least class must be specifyed */
  CMX_ASSERT(class != 0);
  /* config descriptor shall be in dbuffer */
  CMX_ASSERT(USBDSC_TYPE(dbuffer) == STDDTYPE_CONFIGURATION);
  /* pharse configuration descriptor */
  for(ifc=0; ifc < CONFDESC_INTRFACES(dbuffer); ifc++)
  {
    while(USBDSC_TYPE(p) != STDDTYPE_INTERFACE)
    {
      p+=USBDSC_LENGTH(p);
    }

    /* if interface class is ok */
    if (IFCDESC_CLASS(p) == class)
    {
      /* check subclass and protocol if needed */
      if (((sclass == 0) || (IFCDESC_SCLASS(p) == sclass))
         && ((protocol == 0) || (IFCDESC_PROTOCOL(p) == protocol)))
      { /* return start address of interface */
        return((hcc_u8)((hcc_u32)p-(hcc_u32)dbuffer));
      }
    }
    p+=USBDSC_LENGTH(p);
  }
  return(0);
}

/******************************************************************************
 * find_ifc_ndx
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   find an interface in the loaded config descriptor
 ******************************************************************************/
hcc_u16 find_ifc_ndx(hcc_u8 ndx)
{
  int ifc;
  hcc_u8 *p=dbuffer;

  /* config descriptor shall be in dbuffer */
  CMX_ASSERT(USBDSC_TYPE(dbuffer) == STDDTYPE_CONFIGURATION);

  /* pharse configuration descriptor */
  for(ifc=0; ifc < CONFDESC_INTRFACES(dbuffer); ifc++)
  {
    while(USBDSC_TYPE(p) != STDDTYPE_INTERFACE)
    {
      p+=USBDSC_LENGTH(p);
    }

    /* Nonboot hid mouse */
    if (IFCDESC_MY_NDX(p) == ndx)
    {
      return((hcc_u16)((hcc_u32)p-(hcc_u32)dbuffer));
    }
    p+=USBDSC_LENGTH(p);
  }
  return(0);
```

```
}

/*****************************************************************************
 * get_cfg_info
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Get configuration info
 *****************************************************************************/
int get_cfg_info(cfg_info_t *res)
{
  res->nifc=CONFDESC_INTRFACES(dbuffer);
  res->ndx=CONFDESC_MY_NDX(dbuffer);
  res->str=CONFDESC_MY_STR(dbuffer);
  res->attrib=CONFDESC_ATTRIB(dbuffer);
  res->max_power=CONFDESC_MAX_POW(dbuffer);
  return(0);
}

/*****************************************************************************
 * get_ifc_info
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Get interface info
 *****************************************************************************/
int get_ifc_info(ifc_info_t *res, hcc_u16 offset)
{
  res->clas=IFCDESC_CLASS(&dbuffer[offset]);
  res->sclas=IFCDESC_SCLASS(&dbuffer[offset]);
  res->protocol=IFCDESC_PROTOCOL(&dbuffer[offset]);
  res->ndx=IFCDESC_MY_NDX(&dbuffer[offset]);
  res->alt_set=IFCDESC_ALTERNATE(&dbuffer[offset]);
  res->str=IFCDESC_MY_STR(&dbuffer[offset]);
  res->nep=IFCDESC_ENDPONTS(&dbuffer[offset]);
  return(0);
}

/*****************************************************************************
 * get_ep_info
 * IN:
 * OUT:
 * Assumptions: n/a
 * Description:
 *   Get endpoint info
 *****************************************************************************/
int get_ep_info(ep_info_t *res, hcc_u16 offset)
{
  res->address=EPDESC_ADDRESS(&dbuffer[offset]);
  res->type=EPDESC_ATTRIB(&dbuffer[offset]);
  res->interval=EPDESC_INTERVAL(&dbuffer[offset]);
  res->psize=EPDESC_PSIZE(&dbuffer[offset]);
  return(0);
}


/****************************** END OF FILE **********************************/
```

9.2.6. usb_host.h

```
/*****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
```

```
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 ****************************************************************************/
#ifndef _USB_HOST_H_
#define _USB_HOST_H_

#include "types.h"

/* Transaction types. */
#define TRT_NONE  0u
#define TRT_IN    1u
#define TRT_OUT   2u
#define TRT_SETUP 3u

#define INVALID_ADDRESS  ((hcc_u8)-1u)
#define RETVAL_ERROR  ((hcc_u16) -1u)
#define DEVICE_IS_INVALID(dt, dev)  ((dt)[(dev)].address == INVALID_ADDRESS)

/* Access common properties of USB descriptors. */
#define USBDSC_LENGTH(p)  (((hcc_u8*)(p))[0])
#define USBDSC_TYPE(p)    (((hcc_u8*)(p))[1])

/* Read device descriptor members. */
#define DEVDESC_BCDUSB(p)      RD_LE16(((hcc_u8*)(p))+2)
#define DEVDESC_CLASS(p)       (((hcc_u8*)(p))[4])
#define DEVDESC_SCLASS(p)      (((hcc_u8*)(p))[5])
#define DEVDESC_PROTOCOL(p)    (((hcc_u8*)(p))[6])
#define DEVDESC_PACKET_SIZE(p) (((hcc_u8*)(p))[7])
#define DEVDESC_VID(p)         RD_LE16(((hcc_u8*)(p))+8)
#define DEVDESC_PID(p)         RD_LE16(((hcc_u8*)(p))+10)
#define DEVDESC_REV(p)         RD_LE16(((hcc_u8*)(p))+12)
#define DEVDESC_MANUFACT(p)   (((hcc_u8*)(p))[14])
#define DEVDESC_PRODUCT(p)    (((hcc_u8*)(p))[15])
#define DEVDESC_SERNO(p)      (((hcc_u8*)(p))[16])
#define DEVDESC_NCFG(p)       (((hcc_u8*)(p))[17])

/* Read configuration descriptor members. */
#define CONFDESC_TOTLENGTH(p) RD_LE16(((hcc_u8*)(p))+2)
#define CONFDESC_INTRFACES(p) (((hcc_u8*)(p))[4])
#define CONFDESC_MY_NDX(p)    (((hcc_u8*)(p))[5])
#define CONFDESC_MY_STR(p)    (((hcc_u8*)(p))[6])
#define CONFDESC_ATTRIB(p)    (((hcc_u8*)(p))[7])
#define CONFDESC_MAX_POW(p)   (((hcc_u8*)(p))[8])

/* Read interface descriptor members. */
#define IFCDESC_MY_NDX(p)     (((hcc_u8*)(p))[2])
#define IFCDESC_ALTERNATE(p)  (((hcc_u8*)(p))[3])
#define IFCDESC_ENDPONTS(p)   (((hcc_u8*)(p))[4])
#define IFCDESC_CLASS(p)      (((hcc_u8*)(p))[5])
#define IFCDESC_SCLASS(p)     (((hcc_u8*)(p))[6])
#define IFCDESC_PROTOCOL(p)   (((hcc_u8*)(p))[7])
#define IFCDESC_MY_STR(p)     (((hcc_u8*)(p))[8])
```

```
#define EPDESC_ADDRESS(p)      (((hcc_u8*)(p))[2])
#define EPDESC_ATTRIB(p)       (((hcc_u8*)(p))[3])
#define EPDESC_PSIZE(p)        RD_LE16(((hcc_u8*)(p))+4)
#define EPDESC_INTERVAL(p)     (((hcc_u8*)(p))[6])

#define EPTYPE_INVALID         ((hcc_u8)-1u)
#define EPTYPE_CTRL            ((hcc_u8)0u)
#define EPTYPE_ISO             ((hcc_u8)1u)
#define EPTYPE_BULK            ((hcc_u8)2u)
#define EPTYPE_INT             ((hcc_u8)3u)

/* Number of maximum endpoints supported per device. */
#define MAX_EP_PER_DEVICE  3u

/* Driver returns information extracted from device descriptors using this
   type. */
typedef struct {
  hcc_u16 vid;
  hcc_u16 pid;
  hcc_u16 rev;
  hcc_u8  clas;
  hcc_u8  sclas;
  hcc_u8  protocol;
  hcc_u8  ncfg;
} device_info_t;

/* Driver returns information extracted from interface descriptors using this
   type. */
typedef struct {
  hcc_u8 clas;
  hcc_u8 sclas;
  hcc_u8 protocol;
  hcc_u8 ndx;
  hcc_u8 alt_set;
  hcc_u8 str;
  hcc_u8 nep;
} ifc_info_t;

/* Driver returns information extracted from configuration descriptors using
   this type. */
typedef struct {
  hcc_u8 nifc;
  hcc_u8 ndx;
  hcc_u8 str;
  hcc_u8 attrib;
  hcc_u8 max_power;
} cfg_info_t;

/* Driver returns information extracted from endpoint descriptors using this
   type. */
typedef struct {
  hcc_u16 psize;
  hcc_u8 address;
  hcc_u8 type;
  hcc_u8 interval;
} ep_info_t;

 /* Information about endpoints of connected devices. */
 typedef struct {
  hcc_u16 last_due;
  hcc_u16 psize;
  hcc_u8 type;
  hcc_u8 address;
  hcc_u8 interval;
  hcc_u8 tgl_rx;
  hcc_u8 tgl_tx;
} device_ep_t;

/* Information about connected devices. */
typedef struct {
```

```
  hcc_u8 address;
  hcc_u8 low_speed;
  device_ep_t eps[MAX_EP_PER_DEVICE];
} dev_table_element_t;

/* Error codes. */
typedef enum {
  tre_none=0,
  tre_not_running,  /* host operation disabled */
  tre_no_device,    /* no device connected */
  tre_disconnected, /* device has been disconnected */
  tre_stall,        /* endpoint halted by device*/
  tre_data_error,   /* communicatin error (CRC, etc...)*/
  tre_silent,       /* device returned no handshake at all */
  tre_no_ep         /* endpoint not configured */
} tr_error_t;

extern tr_error_t tr_error;
/* Driver call this. Shall enable pull down resistors. */
extern void enable_usb_pull_downs(void);

extern void Usb_Vbus_On(void);
extern void Usb_Vbus_Off(void);

/* Shall be called before any other call. Initializes driver and hardware. */
extern void host_init(void);
/* Stop host functionality. */
extern void host_stop(void);
/* Send USB reset to connected device. */
extern void host_reset_bus(void);
/* Send data to the device on the control channel. */
extern hcc_u16 host_send_control(hcc_u8 *setup_data, hcc_u8* buffer, hcc_u8 ep);
/* Get data from device on the control channel. */
extern hcc_u16 host_receive_control(hcc_u8 *setup_data, hcc_u8* buffer, hcc_u8 ep);
/* Send data to the device on non control channels. */
extern hcc_u32 host_send(hcc_u8* buffer, hcc_u32 length, hcc_u8 ep);
/* Get data from device on non control channels. */
extern hcc_u32 host_receive(hcc_u8* buffer, hcc_u32 length, hcc_u8 ep);

/* Define an endpoint for the device. Driver above the host driver shall call
   it when selecting a configuration. */
extern hcc_u8 host_add_ep(hcc_u8 type, hcc_u8 address, hcc_u8 interval, hcc_u16 psize);
/* Remove an endpoint for the device. Driver above the host driver shall call
   it when selecting a configuration. */
extern void host_remove_ep(hcc_u8 ep_handle);
/* Modifi an endpoint of the device. Driver above the host driver may call
   it when selecting a configuration. */
extern void host_modify_ep(hcc_u8 ep_handle, hcc_u8 type, hcc_u8 address, hcc_u8 interval, hcc_u16
psize);
/* This function will wait the specifyed number of mS. */
extern void host_ms_delay(hcc_u32);
/* Check if a device is connected, and do basic configuration. */
extern int host_scan_for_device(void);
/* Return non0 if a device is connected. Use to see if a the device is still
   connected.*/
extern hcc_u8 host_has_device(void);
/* Prints the device information */
/*extern void print_device_info(void);*/

extern void host_sleep(void);
extern void host_wakeup(void);
#endif
/***************************** END OF FILE *********************************/
```

9.2.7. usb_host.c

```
/*****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
```

```
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 ****************************************************************************/
//#include "hcc_types.h"
#include "MCF52221.h"
#include "timer/timer.h"
#include "usb_host.h"
#include "usb_utils.h"

/* Standard IO is only possible if Console or UART support is enabled. */
#include <stdio.h>


#define DEBUG_TRACE 0u
#define DSC_BUG 0u
/*****************************************************************************
 * Macro definitions.
 ****************************************************************************/
#define MIN(a,b)  ((a) < (b) ? (a) : (b))

/* This macro shall evaluate to a uint32 pointer to the start address of the
   buffer descriptor table (BDT). The BDT has 32 bytes for each endpoint.
   The BDT shall be alignet to 512 byte boundary! */
extern hcc_u32 _BDT_BASE[];
#define BDT_BASE              ((hcc_u32*)(_BDT_BASE))
#define BDT_CTL_RX(ep, b)     (BDT_BASE[((ep)<<3)+((b)<<1)+0])
#define BDT_ADR_RX(ep, b)     (BDT_BASE[((ep)<<3)+((b)<<1)+1])
#define BDT_CTL_TX(ep, b)     (BDT_BASE[((ep)<<3)+((b)<<1)+4])
#define BDT_ADR_TX(ep, b)     (BDT_BASE[((ep)<<3)+((b)<<1)+5])

#define BDT_CTL_STALL BIT2
#define BDT_CTL_DTS   BIT3
#define BDT_CTL_DATA  BIT6
#define BDT_CTL_OWN   BIT7

/* Token values. */
#define TOKEN_SETUP   0xd
#define TOKEN_IN      0x9
#define TOKEN_OUT     0x1
#define TOKEN_NAK     0xa
#define TOKEN_STALL   0xe
#define TOKEN_ACK     0x2

/* Maximum possible packet size for endpoint 0. */
#define MAX_EP0_PSIZE   64u
#define MIN_EP0_PSIZE   8u

#if DEBUG_TRACE == 1
#define MKDBG_TRACE(evnt, epndx)   (evstk[evndx].ev=(evnt), evstk[evndx++].ep=(epndx))
```

```
#else
#define MKDBG_TRACE(evnt, epndx)
#endif


/*****************************************************************************
 * External references.
 *****************************************************************************/
/* none */

/*****************************************************************************
 * Local types.
 *****************************************************************************/

#if DEBUG_TRACE == 1
enum event
{
  ev_none = 0,
  ev_bus_reset,
  ev_host_init,
  ev_no_attach,
  ev_attach_ls_dev,
  ev_attach_fs_dev,
  ev_host_stop,
  ev_trt_setup,
  ev_trt_in,
  ev_trt_out,
  ev_got_ack,
  ev_got_nak,
  ev_got_stall,
  ev_got_unknown,
  ev_got_data_error,
  ev_disconnect,
  ev_send_control,
  ev_receive_control,
  ev_send,
  ev_receive,
  ev_no_device,
  ev_no_ep,
  ev_ep0_psize_failed,
  ev_set_address_failed
};
#endif
/*****************************************************************************
 * Module variables.
 *****************************************************************************/
#if DEBUG_TRACE == 1

volatile hcc_u8 evndx=0;

struct
{
  enum event ev;
  hcc_u8 ep;
} evstk[1<<(sizeof(evndx)<<3)];

#endif

/* This structure tells us which packet buffer was last used for TX and RX. */
struct {
  hcc_u8 next_rx;
  hcc_u8 next_tx;
} ep_info;

/* This table contains information about connected devices. Since we do not
   support external HUBS and this hardware has no ROOT HUB, only one device
   can be connected at a time. */
dev_table_element_t my_device;

/* Error code for the device. */
```

```
tr_error_t tr_error;
/*****************************************************************************
 * Function prototypes
 *****************************************************************************/
/*  */

/*****************************************************************************
 * Function declarations
 *****************************************************************************/

void Usb_Vbus_On(void)
{
  MCF_GPIO_PORTUA &= ~MCF_GPIO_PORTUA_PORTUA3; /* turn AMD (Vbus) switch on */
}

void Usb_Vbus_Off(void)
{
  MCF_GPIO_PORTUA |= MCF_GPIO_PORTUA_PORTUA3; /* turn AMD (Vbus) switch onff */
}

/*****************************************************************************
 * evt_disconnect
 * IN:  -
 * OUT: -
 * Assumptions:
 * Description:
 *   Put hardware and firmware to disconnected state.
 *****************************************************************************/
static void evt_disconnect(void)
{
  MKDBG_TRACE(ev_disconnect, 0);

#if DSC_BUG == 0
  while(MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY)
  {
    if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
    {
      MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY;
      break;
    }
  }
#endif

  my_device.address = INVALID_ADDRESS;
  my_device.low_speed = 0;
  /* disable SOF generation */
  MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_USB_RST | MCF_USB_OTG_INT_STAT_ATTACH;
  host_sleep();
}

static hcc_u8 evt_connect(void)
{
  hcc_u8 ep;
  volatile int x;

  /* debounce (100 mS) */
  host_ms_delay(100);

  /* clear attach event */
  MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_ATTACH;

  /* Some delay is needed between clearing the ATTACH flag, and checking
     it again. Unfortunately there is no information about the length of
     the delay. */
  for(x=0; x<10000; x++)
    ;

  /* Is a device connected? */
  if((MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_ATTACH) == 0)
  { /* no */
```

```
    return(0);
  }

  /* A newly connected device shall have address 0, */
  my_device.address=0;
  /* and only ep0 is working. We assume packet size of
     ep0 is the possible minimum. We will read the real
     value during enumeration. */
  host_modify_ep(0, EPTYPE_CTRL, 0, 0, MIN_EP0_PSIZE);

  /* remove all endpoints except 0 */
  for(ep=1;ep<MAX_EP_PER_DEVICE;ep++)
  {
    host_remove_ep(ep);
  }

  /* clear low speed bit to make JSTATE detection consistent. */
  MCF_USB_OTG_ADDR=0;

  /* let settle D+ and D- to the right state */
  host_ms_delay(1);

  /* Check if device is low or high speed. */
  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_JSTATE) == 0)
  {
    MKDBG_TRACE(ev_attach_ls_dev, 0);
    /* Low speed device. */
    my_device.low_speed=1;
    MCF_USB_OTG_ADDR = MCF_USB_OTG_ADDR_LS_EN;
  }
  else
  {
    MKDBG_TRACE(ev_attach_fs_dev, 0);
    my_device.low_speed=0;
  }
  return(1);
}

static hcc_u8 chk_dev(void)
{
  if (my_device.address != INVALID_ADDRESS)
  {
    /* If a reset was seen while we have not been issued a reset, then
       the device is disconnected (surprise removal). */
    if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
    {
      evt_disconnect();
    }
  }

  if (my_device.address == INVALID_ADDRESS)
  {
    volatile int x;
    /* If we can not clear the attach flag, then a device is connected. */
    MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_ATTACH;

    /* Some delay is needed between clearing the ATTACH flag, and checking
       it again. Unfortunately there is no information about the length of
       the delay. */
    for(x=0; x< 10000; x++)
      ;

    if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_ATTACH)
    {
      evt_connect();
    }
  }
  return((hcc_u8)(my_device.address != INVALID_ADDRESS));
}
/*****************************************************************************
```

```
 * host_has_device
 * IN:  -
 * OUT: -
 * Assumptions:
 * Description:
 *   Return true if a device is still connected.
 ******************************************************************************/
hcc_u8 host_has_device(void)
{
  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return(0);
  }
  tr_error=tre_none;
  return chk_dev();
}


/******************************************************************************
 * host_add_ep
 * IN:
 *     type:     type of endpoint
 *     address:  address of endpoint
 *     interval: poll interval for periodic endpoints
 *     psize:    maximum packet size
 * OUT:
 *     endpoint handle
 * Assumptions:
 * Description:
 *   Add a new endpoint to a device
 ******************************************************************************/
hcc_u8 host_add_ep(hcc_u8 type, hcc_u8 address, hcc_u8 interval, hcc_u16 psize)
{
  hcc_u8 x;

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return(INVALID_ADDRESS);
  }

  for(x=0; x < MAX_EP_PER_DEVICE; x++)
  {
    if (my_device.eps[x].address == INVALID_ADDRESS)
    {
      my_device.eps[x].type=type;
      my_device.eps[x].address=(hcc_u8)(address & 0x7f);
      my_device.eps[x].interval=interval;
      my_device.eps[x].psize=psize;
      my_device.eps[x].tgl_rx=0;
      my_device.eps[x].tgl_tx=0;
      my_device.eps[x].last_due=0;
      return(x);
    }
  }
  return(INVALID_ADDRESS);
}

/******************************************************************************
 * host_remove_ep
 * IN:  ep_handle: handle of endpoint
 * OUT: n/a
 * Assumptions:
 *   The parameter is a valid handle.
 * Description:
 *   Remove an dendpoint of a device.
 ******************************************************************************/
void host_remove_ep(hcc_u8 ep_handle)
{
```

```
  CMX_ASSERT(ep_handle < MAX_EP_PER_DEVICE);

  my_device.eps[ep_handle].address=INVALID_ADDRESS;
}

/*****************************************************************************
 * host_modify_ep
 * IN:
 *     ep_handle: handle of an endpoint
 *     type:      type of endpoint
 *     address:   address of endpoint
 *     interval:  poll interval for periodic endpoints
 *     psize:     maximum packet size
 * OUT:
 * Assumptions:
 * Description:
 *   Add a new endpoint to a device
 *****************************************************************************/
void host_modify_ep(hcc_u8 ep_handle, hcc_u8 type, hcc_u8 address, hcc_u8 interval, hcc_u16 psize)
{
  CMX_ASSERT(ep_handle < MAX_EP_PER_DEVICE);
//  printf("modifying ep%d: type=%d, address=%d, interval=%d, size=%d\r\n", ep_handle, type, address,
interval, psize);
  my_device.eps[ep_handle].type=type;
  my_device.eps[ep_handle].address=address;
  my_device.eps[ep_handle].interval=interval;
  my_device.eps[ep_handle].psize=psize;
}

/*****************************************************************************
 * host_reset_bus
 * IN: n/a
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Do reset signaling on the USB bus.
 *****************************************************************************/
void host_reset_bus(void)
{
  hcc_u8 ep=0;

  MKDBG_TRACE(ev_bus_reset, 0);

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return;
  }

  /********* Prepare for BUS reset. */
  /* Wait till pending tokens are processed. */
#if DSC_BUG == 0
  while(MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY)
  {
    if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
    {
      MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY;
      break;
    }
  }
#endif
  /********* reset */
  /* Start reset signaling. */
  MCF_USB_OTG_CTL |= MCF_USB_OTG_CTL_RESET;

  /* The minumum reset signal length is 10 mS. We use an 1 mS timer so it has
     at least 1mS error (1 period). USB specifies +-0.05% accuracy for frame
     interval. So we are good enough if we wait for 11 SOF due times. */
  host_ms_delay(11);
  /* stop reset signaling */
```

```
  MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_RESET;

  /* Clear reset event. */
  MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_USB_RST;

  /********* do firmware reset */
  /* A reset device shall have address 0, */
  if (my_device.address != INVALID_ADDRESS)
  {
    my_device.address=0;
  }
  /* and only ep0 is working. We assume packet size of
     ep0 is the possible minimum. We will read the real
     value during enumeration. */
  host_modify_ep(0, EPTYPE_CTRL, 0, 0, MIN_EP0_PSIZE);

  /* remove all endpoints except 0 */
  for(ep=1;ep<MAX_EP_PER_DEVICE;ep++)
  {
    host_remove_ep(ep);
  }

  /* check if we have a device connected */
  /* if a device is connected, it will answer to address 0 */
  if (chk_dev())
  {
    /* enable SOF generation */
    MCF_USB_OTG_CTL |= MCF_USB_OTG_CTL_USB_EN_SOF_EN;

    MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_SLEEP | MCF_USB_OTG_INT_STAT_RESUME;

    /* device may need max 10 mS reset recovery time */
    /*host_ms_delay(10);*/
    host_ms_delay(100);
    /* Read out endpoint 0 packet size. */
    if (set_ep0_psize())
    {
      MKDBG_TRACE(ev_ep0_psize_failed, (hcc_u8)-1u);
      return;
    }
    /* set device address */
    if (set_address(1))
    {
      MKDBG_TRACE(ev_set_address_failed, (hcc_u8)-1u);
      return;
    }
    my_device.address=1;
  }
}

/*****************************************************************************
 * host_init
 * IN: n/a
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Inicialize host.
 *****************************************************************************/
void host_init(void)
{
  hcc_u8 ep;
  MKDBG_TRACE(ev_host_init, 0);
  /****** inicialize module variables. */
  my_device.address=INVALID_ADDRESS;
  ep_info.next_tx=0;
  ep_info.next_rx=0;
  tr_error=tre_none;

  /* remove all endpoints */
  for(ep=0;ep<MAX_EP_PER_DEVICE;ep++)
```

```
  {
    host_remove_ep(ep);
  }
  /****** Configure hardware. */

  /* Clear all pending events. */
  MCF_USB_OTG_INT_STAT = 0xff;

  /* select USB clock source */
  MCF_USB_OTG_USB_CTRL = MCF_USB_OTG_USB_CTRL_CLK_SRC_OSCCLK;

  /* use first rx and tx buffer */
  MCF_USB_OTG_CTL |= MCF_USB_OTG_CTL_ODD_RST;
  /* set BDT address */
  MCF_USB_OTG_BDT_PAGE_01 = (hcc_u8)(((hcc_u32)BDT_BASE) >> 8);
  MCF_USB_OTG_BDT_PAGE_02 = (hcc_u8)(((hcc_u32)BDT_BASE) >> 16);
  MCF_USB_OTG_BDT_PAGE_03 = (hcc_u8)(((hcc_u32)BDT_BASE) >> 24);

  /* Set SOF threshold */
  MCF_USB_OTG_SOF_THLD = 74;

  /* Enable pull-down resistors. */
  enable_usb_pull_downs();

  /* Enable host operation mode. */
  MCF_USB_OTG_CTL = MCF_USB_OTG_CTL_HOST_MODE_EN;
}

/*****************************************************************************
 * host_scan_for_device
 * IN: n/a
 * OUT: 0: no device
 *      1: device detected
 * Assumptions: n/a
 * Description:
 *   Check if the device has been attached. If yes, set its basic parameters.
 *****************************************************************************/
int host_scan_for_device(void)
{
  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return(0);
  }

  tr_error=tre_none;

  if (chk_dev())
  {
    host_reset_bus();
    if (tr_error!= tre_none)
    {
      return(0);
    }
    return(1);
  }
  return(0);
}

/*****************************************************************************
 * host_stop
 * IN: n/a
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Stop host operation.
 *****************************************************************************/
void host_stop(void)
{
  MKDBG_TRACE(ev_host_stop, 0);
```

```
  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return;
  }

  /* Disable all interrupt sources. */
  MCF_USB_OTG_INT_ENB = 0;
  MCF_USB_OTG_ERR_ENB = 0;
  /* Clear all pending events. */
  MCF_USB_OTG_INT_STAT =0;
  MCF_USB_OTG_ERR_STAT =0;
  /* Disable SOF generation. */
  MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_USB_EN_SOF_EN;

  /* Stop USB.*/
  MCF_USB_OTG_CTL = 0;

  MCF_USB_OTG_USB_CTRL = 0;
}

/*****************************************************************************
 * usb_host_start_transaction
 * IN: type   - type of transaction
 *     buffer - data area for the transfer
 *     length - size of the buffer
 * OUT:
 *      -1: error
 *       x: number of processed bytes otherwise.
 * Assumptions: n/a
 * Description:
 *   Start the specified transatcion on the USB.
 *****************************************************************************/
static hcc_u16 usb_host_start_transaction(hcc_u8 type, hcc_u8 *buffer, hcc_u16 length, hcc_u8 ep)
{
  hcc_u8 token;
  hcc_u32* bdt_ctl;
  hcc_u8 retry=3;

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return((hcc_u16)-1u);
  }

  /* Check if a device is attached. */
  if (my_device.address == INVALID_ADDRESS)
  {
    tr_error=tre_no_device;
    MKDBG_TRACE(ev_no_device, 0);
    return((hcc_u16)-1u);
  }

  if (my_device.eps[ep].address == INVALID_ADDRESS)
  {
    tr_error=tre_no_ep;
    MKDBG_TRACE(ev_no_ep, ep);
    return((hcc_u16)-1u);
  }

  /* Set device address. */
  MCF_USB_OTG_ADDR = (hcc_u8)(my_device.low_speed ?
                        my_device.address | MCF_USB_OTG_ADDR_LS_EN :
                        my_device.address);
  /* We are not talking over a HUB. */
  MCF_USB_OTG_ENDPT0 = MCF_USB_OTG_ENDPT_HOST_WO_HUB;

  /* If this is an interrupt endpoint wait till the endpoint is due. */
  /* We aren't using interrupt endpoints, so save some code space */
```

```
  /*if (my_device.eps[ep].type == EPTYPE_INT)
  {
    hcc_u16 elapsed;
    // Disable hardware retry.
    MCF_USB_OTG_ENDPT0 |= MCF_USB_OTG_ENDPT_RETRY_DIS;
    // Disable software retry.
    retry=1;

    // wait till frame is due
    do {
      elapsed=(hcc_u16)(MCF_USB_OTG_FRM_NUM-my_device.eps[ep].last_due);
      elapsed &= ((1<<11)-1);
      if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
      {
        evt_disconnect();
        tr_error=tre_disconnected;
        return((hcc_u16)-1u);
      }
    } while(elapsed < my_device.eps[ep].interval);


    my_device.eps[ep].last_due += my_device.eps[ep].interval;
    my_device.eps[ep].last_due &= ((1<<11)-1);
  }*/

  do  {
    tr_error=tre_none;
    retry--;
    switch(type)
    {
      case TRT_SETUP:
        MKDBG_TRACE(ev_trt_setup, ep);
        /* Configure bi-directional communication. */
        /* Usb spec says setup packets shall be accepted
           even if the device was not able to process its packet
           buffer. Thus NAK handshake should not be given to a
           setup packet by the device. Anyway some devices will
           happily NAK setup packets :( */
        MCF_USB_OTG_ENDPT0 |= /*MCF_USB_ENDPT0_RETRY_DIS |*/ 0x0d;

        /* After the setup we shall send/receive DATA1 packets. */
        my_device.eps[ep].tgl_tx=1;
        my_device.eps[ep].tgl_rx=1;
        /* Set data buffer address. */
        WR_LE32(&BDT_ADR_TX(0, ep_info.next_tx), (hcc_u32)buffer);
        /* Set packet properties and give buffer to USB. */
        bdt_ctl=&BDT_CTL_TX(0, ep_info.next_tx);
        WR_LE32(bdt_ctl, (0x8<<16) | BDT_CTL_OWN | 0);
        /* Wait till pending tokens are processed. */
#if DSC_BUG == 0
        while(MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY)
        {
          if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
          {
            MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY;
            evt_disconnect();
            tr_error=tre_disconnected;
            return((hcc_u16)-1u);
          }
        }
#endif
        /* Start transaction. */
        MCF_USB_OTG_TOKEN=(hcc_u8)((TOKEN_SETUP<<4) | (my_device.eps[ep].address | (0<<7)));
        break;
      case TRT_IN:
        MKDBG_TRACE(ev_trt_in, ep);
        if(my_device.eps[ep].type == EPTYPE_ISO)
          MCF_USB_OTG_ENDPT0 |= 0x48;
        else
          /* Configure bi-directional communication + auto repeat. */
```

```
          MCF_USB_OTG_ENDPT0 |= 0x0d;
        /* Set RX buffer address. */
        WR_LE32(&BDT_ADR_RX(0, ep_info.next_rx), (hcc_u32)buffer);
        /* Set packet properies and give next buffer to USB. */
        bdt_ctl=&BDT_CTL_RX(0, ep_info.next_rx);
        WR_LE32(bdt_ctl, (length<<16) | BDT_CTL_OWN | my_device.eps[ep].tgl_rx);
        my_device.eps[ep].tgl_rx =(hcc_u8)(my_device.eps[ep].tgl_rx ? 0 : BDT_CTL_DATA);
        /* Wait till pending tokens are processed. */
#if DSC_BUG == 0
        while(MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY)
        {
          if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
          {
            MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY;
            evt_disconnect();
            tr_error=tre_disconnected;
            return((hcc_u16)-1u);
          }
        }
#endif
        /* Start transaction. */
        MCF_USB_OTG_TOKEN=(hcc_u8)((TOKEN_IN<<4) | (my_device.eps[ep].address | (1<<7)));
        break;
      case TRT_OUT:
        MKDBG_TRACE(ev_trt_out, ep);
        /* Configure bi-directional communication + auto repeat. */
        MCF_USB_OTG_ENDPT0 |= 0x0d;
        /* Set TX buffer address. */
        WR_LE32(&BDT_ADR_TX(0, ep_info.next_tx), (hcc_u32)buffer);
        /* Set packet properties ang give buffer to USB. */
        bdt_ctl=&BDT_CTL_TX(0, ep_info.next_tx);
        WR_LE32(bdt_ctl, (length<<16) | BDT_CTL_OWN | my_device.eps[ep].tgl_tx);
        my_device.eps[ep].tgl_tx = (hcc_u8)(my_device.eps[ep].tgl_tx ? 0 : BDT_CTL_DATA);
        /* Wait till pending tokens are processed. */
#if DSC_BUG == 0
        while(MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY)
        {
          if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
          {
            MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY;
            evt_disconnect();
            tr_error=tre_disconnected;
            return((hcc_u16)-1u);
          }
        }
#endif
        /* Start transaction. */
        MCF_USB_OTG_TOKEN=(hcc_u8)((TOKEN_OUT<<4) | (my_device.eps[ep].address | (0<<7)));
        break;
      default:
        CMX_ASSERT(0);
    }

            /*if(my_device.eps[ep].type == EPTYPE_ISO)
            {
                        while(RD_LE32(bdt_ctl) & BDT_CTL_OWN) {
                        }
            }
            else
            { */
      /* Wait for transaction end. */
      while((MCF_USB_OTG_INT_STAT & (MCF_USB_OTG_INT_STAT_TOK_DNE | MCF_USB_OTG_INT_STAT_STALL |
MCF_USB_OTG_INT_STAT_ERROR)) ==0)
      {
        if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_USB_RST)
        {
          evt_disconnect();
          tr_error=tre_disconnected;
          return((hcc_u16)-1u);
        }
```

```
      }

      if ((MCF_USB_OTG_ERR_STAT & ~MCF_USB_OTG_ERR_STAT_CRC5_EOF) != 0)
      {
        MCF_USB_OTG_ERR_STAT = 0xff;
        MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_ERROR;
        tr_error=tre_data_error;
        continue;
      }
    //       }

     /* device not disconnected while waiting for an aswer */
     /* clear transfer ok event */
     MCF_USB_OTG_INT_STAT=MCF_USB_OTG_INT_STAT_TOK_DNE;
     MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_TXSUSPEND_TOKENBUSY;

     /* switch to next buffer */
     if (type== TRT_SETUP || type == TRT_OUT)
     {
       ep_info.next_tx ^= 0x1u;
     }
     else
     {
       ep_info.next_rx ^= 0x1u;
     }

     if (MCF_USB_OTG_INT_STAT & MCF_USB_OTG_INT_STAT_STALL)
     {         /* device has an error */
       MCF_USB_OTG_INT_STAT=MCF_USB_OTG_INT_STAT_STALL;
       MKDBG_TRACE(ev_got_stall, ep);
       tr_error=tre_stall;
       return((hcc_u16)-1u);
     }

     /* Check transaction status.*/
     token=(hcc_u8)((RD_LE32(bdt_ctl) >> 2) & 0x0f);

     /* transaction accepted by device */
     switch (token)
     {
     default:
     case TOKEN_ACK:
       MKDBG_TRACE(ev_got_ack, ep);
       return((hcc_u16)((RD_LE32(bdt_ctl) >> 16u) & 0x3ffu));
     case TOKEN_NAK:
       /* device is not ready */
       MKDBG_TRACE(ev_got_nak, ep);
       if (my_device.eps[ep].type == EPTYPE_INT)
       {
         return(0);
       }
       /* retry */
       break;
     case TOKEN_STALL: /* endpoint stalled by device */
       /* we can not get here beacuse we already checked
          INT_STAT_STALL. */
       tr_error=tre_stall;
       return((hcc_u16)-1u);
     case 0xf:  /* data error, retry */
       MKDBG_TRACE(ev_got_data_error, ep);
       tr_error=tre_data_error;
       break;
     case 0:  /* no answer, retry. */
       MKDBG_TRACE(ev_got_unknown, ep);
       tr_error=tre_silent;
       break;
     }
   }while(retry);
   return((hcc_u16)-1u);
}
```

```
/*****************************************************************************
 * host_send_control
 * IN: buffer - data area for the transfer
 *     length - size of the buffer
 * OUT: != -1: Number of transmitted data bytes
 *      -1   : error
 * Assumptions: n/a
 * Description:
 *   Make an OUT transfer on a control endpoint.
 *****************************************************************************/
hcc_u16 host_send_control(hcc_u8 *setup_data, hcc_u8* buffer, hcc_u8 ep)
{
  hcc_u32 curr=0;
  hcc_u16 length=RD_LE16(setup_data+6);

  MKDBG_TRACE(ev_send_control, ep);

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return((hcc_u16)-1u);
  }

  /* setup transaction. */
  if (((hcc_u16)-1u)==usb_host_start_transaction(TRT_SETUP, setup_data, 8, ep))
  {
    return((hcc_u16)-1u);
  }

  /* data transactions */
  while(curr<length)
  {
    hcc_u16 psize=(hcc_u16)(MIN(my_device.eps[ep].psize, length));
    hcc_u8 r=(hcc_u8)usb_host_start_transaction(TRT_OUT, buffer+curr, psize, ep);
    if (r != psize)
    {
      CMX_ASSERT(r==((hcc_u8)-1u));
      return((hcc_u16)-1u);
    }
    curr += psize;
  }
  /* handshake transaction */
  my_device.eps[ep].tgl_rx = BDT_CTL_DATA;
  if (((hcc_u16)-1u)==usb_host_start_transaction(TRT_IN, (void *)0, 0, ep))
  {
    return((hcc_u16)-1u);
  }

  return((hcc_u16)curr);
}

/*****************************************************************************
 * host_send
 * IN: buffer - data area for the transfer
 *     length - size of the buffer
 * OUT: != -1: Number of transmitted data bytes
 *      -1   : error
 * Assumptions: n/a
 * Description:
 *   Make an OUT transfer on a non control endpoint.
 *****************************************************************************/
hcc_u32 host_send(hcc_u8* buffer, hcc_u32 length, hcc_u8 ep)
{
  hcc_u32 curr=0;

  MKDBG_TRACE(ev_send, ep);

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
```

```
    tr_error=tre_not_running;
    return(0);
  }

  /* data transactions */
  while(curr<length)
  {
    hcc_u16 psize=(hcc_u16)(MIN(my_device.eps[ep].psize, length));
    /* do transaction */
    hcc_u16 r=usb_host_start_transaction(TRT_OUT, buffer+curr, psize, ep);
    if (r!=psize)
    {
      CMX_ASSERT(r==((hcc_u16)-1u));
      break;
    }
    curr += psize;
  }
  return(curr);
}

/*****************************************************************************
 * host_receive
 * IN: buffer - data area for the transfer
 *     length - size of the buffer
 * OUT: != -1: Number of transmitted data bytes
 *      -1   : error
 * Assumptions: n/a
 * Description:
 *   Make an IN transfer on a non-control endpoint.
 *****************************************************************************/
hcc_u32 host_receive(hcc_u8* buffer, hcc_u32 length, hcc_u8 ep)
{
  hcc_u32 curr=0;

  MKDBG_TRACE(ev_receive, ep);

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return(0);
  }

  /* data transactions */
  while(curr<length)
  {
    hcc_u16 psize=(hcc_u16)(MIN(my_device.eps[ep].psize, length));
    return usb_host_start_transaction(TRT_IN, buffer+curr, psize, ep);
    /*got=usb_host_start_transaction(TRT_IN, buffer+curr, psize, ep);
    if (got == ((hcc_u16)-1u))
    {
      break;
    }
    curr += got;
    // short packet means end of transfer
    if (got != my_device.eps[ep].psize)
    {
      break;
    }*/
  }
  return(curr);
}


/*****************************************************************************
 * host_receive_control
 * IN: buffer - data area for the transfer
 *     length - size of the buffer
 * OUT: != -1: Number of transmitted data bytes
 *      -1   : error
 * Assumptions: n/a
```

```
 * Description:
 *   Make an IN transfer on a control endpoint.
 ****************************************************************************/
hcc_u16 host_receive_control(hcc_u8 *setup_data, hcc_u8* buffer, hcc_u8 ep)
{
  hcc_u32 curr=0;
  hcc_u16 length = RD_LE16(setup_data+6);

  MKDBG_TRACE(ev_receive_control, ep);
  /* setup transaction. */

  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return((hcc_u16) -1u);
  }

  if (((hcc_u16)-1u)==usb_host_start_transaction(TRT_SETUP, setup_data, 8, ep))
  {
    return((hcc_u16)-1u);
  }

  /* data transactions */
  while(curr<length)
  {
    hcc_u16 got,
            psize=(hcc_u16)(MIN(my_device.eps[ep].psize, length-curr));
    /*printf("receive_control data transaction requested %d bytes\r\n", psize);*/
    got=usb_host_start_transaction(TRT_IN, buffer+curr, length /*psize*/, ep);
    /*printf("receive_control data transaction got %d bytes\r\n", got);*/
    curr += got;
    if (got == ((hcc_u16)-1u))
    {
      return((hcc_u16)-1u);
    }
    /* short packet means end of transfer */
    if (got != my_device.eps[ep].psize)
    {
      break;
    }
  }
  /* handshake transaction */
  my_device.eps[0].tgl_tx = BDT_CTL_DATA;
  if (((hcc_u16)-1u)==usb_host_start_transaction(TRT_OUT, (void *)0, 0, ep))
  {
    return((hcc_u16)-1u);
  }

  /*printf("host_receive_control returning %d\r\n", curr);*/
  return((hcc_u16)curr);
}

void host_ms_delay(hcc_u32 delay)
{
  start_mS_timer((hcc_u16)delay);
  do {
  } while(!check_mS_timer());
}

/****************************************************************************
 * host_drop_device
 * IN: n/a
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Remove device fro host.
 ****************************************************************************/
static void host_drop_device(void)
{
```

```
  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return;
  }

  /* if we don't have any device */
  if (my_device.address == INVALID_ADDRESS)
  {
    return;
  }
  host_sleep();
}


/*****************************************************************************
 * host_sleep
 * IN: n/a
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Put the BUS into low-power mode.
 *****************************************************************************/
void host_sleep(void)
{
  /* Sleep does not seems to work when SOF_TOK is off. Thus we treat
     resume=0 while suspended as sleep. */
  MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_RESUME
                       | MCF_USB_OTG_INT_STAT_SOF_TOK;
  /* disable SOF generation */
  MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_USB_EN_SOF_EN;
  /* Wait 3 mS to let BUS enter suspended state. */
  host_ms_delay(3);
}

/*****************************************************************************
 * host_wakeup
 * IN: n/a
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Wake-up.
 *****************************************************************************/
void host_wakeup(void)
{
  if ((MCF_USB_OTG_CTL & MCF_USB_OTG_CTL_HOST_MODE_EN) == 0)
  {
    tr_error=tre_not_running;
    return;
  }

  /* if we don't have any device */
  if (my_device.address == INVALID_ADDRESS)
  {
    return;
  }

  /* Check if device has been disconnected. */
  if (!chk_dev())
  {
    return;
  }

  /* Start wakeup signaling. */
  MCF_USB_OTG_CTL |= MCF_USB_OTG_CTL_RESUME;
  host_ms_delay(20);
  MCF_USB_OTG_CTL &= ~MCF_USB_OTG_CTL_RESUME;

  /* enable SOF generation */
  MCF_USB_OTG_CTL |= MCF_USB_OTG_CTL_USB_EN_SOF_EN;
```

```
  MCF_USB_OTG_INT_STAT = MCF_USB_OTG_INT_STAT_SLEEP | MCF_USB_OTG_INT_STAT_RESUME;
  /* Wakeup recovery. */
  host_ms_delay(10);
}

/*****************************************************************************
 * print_device_info
 * IN: dev_table_element_t (device information structure)
 * OUT: n/a
 * Assumptions: n/a
 * Description:
 *   Prints the device information.
 *****************************************************************************/
/* Prints the device information */
/*void print_device_info(void)
{
  printf("address: %d\r\n", my_device.address);
  printf("low speed: %d\r\n", my_device.low_speed);
  fflush(stdout);
}*/

/****************************** END OF FILE **********************************/
```

9.2.8. timer.h

```
/*****************************************************************************
 *
 *            Copyright (c) 2007 by CMX Systems, Inc.
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 *****************************************************************************/
#ifndef _TIMER_H_
#define _TIMER_H_

#include "types.h"

void start_mS_timer(hcc_u16 delay);
hcc_u8 check_mS_timer(void);

#endif
/****************************** END OF FILE **********************************/
```

9.2.9. timer.c

```
/*****************************************************************************
 *
 *            Copyright (c) 2007 by CMX Systems, Inc.
 *
```

```
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 ****************************************************************************/
#include "timer.h"
#include "MCF52221.h"
#include "types.h"

/* The OTG and USB host driver needs a mS timer. This is implemented using
   PIT0 */
#define P0CLK    40000000ul
#define P0PRES_VAL       12u

void start_mS_timer(hcc_u16 delay)
{
  hcc_u32 val=delay*((P0CLK/1000)/(1<<P0PRES_VAL));

  CMX_ASSERT(val == (hcc_u16)val);

  MCF_PIT0_PCSR = MCF_PIT_PCSR_OVW | MCF_PIT_PCSR_PIF;

  MCF_PIT0_PMR = (hcc_u16)val;

  MCF_PIT0_PCSR = MCF_PIT_PCSR_PRE(P0PRES_VAL) | MCF_PIT_PCSR_EN | MCF_PIT_PCSR_OVW | MCF_PIT_PCSR_PIF;
}

hcc_u8 check_mS_timer(void)
{
  if (!(MCF_PIT0_PCSR & MCF_PIT_PCSR_PIF))
  {
    return(0);
  }

  MCF_PIT0_PCSR = MCF_PIT_PCSR_PIF;
  return(1);
}
/***************************** END OF FILE ***********************************/
```

9.2.10. MCF52221_INTERNAL_FLASH.lcf

```
# Sample Linker Command File for CodeWarrior for ColdFire


# Memory ranges

MEMORY {
   vectorrom   (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000400
   cfmprotrom  (RX)  : ORIGIN = 0x00000400, LENGTH = 0x00000020
   code        (RX)  : ORIGIN = 0x00000500, LENGTH = 0x0001FB00
   vectorram   (RWX) : ORIGIN = 0x20000000, LENGTH = 0x00000000 #0x00000400
```

```
   userram     (RWX) : ORIGIN = 0x20000000, LENGTH = 0x00004000
}

SECTIONS {

# Heap and Stack sizes definition
            ___heap_size              = 0x0000;
            ___stack_size     = 0x400;



# MCF52221 Derivative Memory map definitions from linker command files:
# __IPSBAR, __RAMBAR, __RAMBAR_SIZE, __FLASHBAR, __FLASHBAR_SIZE linker
# symbols must be defined in the linker command file.

# Memory Mapped Registers (IPSBAR= 0x40000000)
   ___IPSBAR         = 0x40000000;

# 16 Kbytes Internal SRAM
   ___RAMBAR         = 0x20000000;
   ___RAMBAR_SIZE    = 0x00004000;

# 128 KByte Internal Flash Memory
   ___FLASHBAR       = 0x00000000;
   ___FLASHBAR_SIZE  = 0x00020000;

   ___SP_AFTER_RESET = ___RAMBAR + ___RAMBAR_SIZE - 4;

  .userram    : {} > userram
  .code         : {} > code
  .vectorram            : {} > vectorram

            .vectors :
            {
                        exceptions.c(.vectortable)
                        . = ALIGN (0x4);
            } > vectorrom

            .cfmprotect :
            {
                        *(.cfmconfig)
                        . = ALIGN (0x4);
            } > cfmprotrom

            .text :
            {
                        *(.text)
                        . = ALIGN (0x4);
                        *(.rodata)
                        . = ALIGN (0x4);
                        ___ROM_AT = .;
                        ___DATA_ROM = .;
            } >> code

            .data : AT(___ROM_AT)
            {
                        ___DATA_RAM = .;
                        . = ALIGN(0x4);
                        *(.exception)
                        . = ALIGN(0x4);
                        __exception_table_start__ = .;
                        EXCEPTION
                        __exception_table_end__ = .;

                        ___sinit__ = .;
                STATICINIT
                        __START_DATA = .;

                        *(.data)
                        . = ALIGN (0x4);
```

```
                __END_DATA = .;

                __START_SDATA = .;
                *(.sdata)
                . = ALIGN (0x4);
                __END_SDATA = .;

                ___DATA_END = .;
                __SDA_BASE = .;
                . = ALIGN (0x4);
        .                                       = ALIGN(512);
            } >> userram

            .bss :
            {
                ___BSS_START = .;
                __START_SBSS = .;
                *(.sbss)
                . = ALIGN (0x4);
                *(SCOMMON)
                __END_SBSS = .;

                __START_BSS = .;
                *(.bss)
                . = ALIGN (0x4);
                *(COMMON)
                __END_BSS = .;
                ___BSS_END = .;

        /* Buffer descriptor base address
           shall be aligned to 512 byte boundary.
           Size shall be 512 bytes. */
        .                                       = ALIGN(512);
        __BDT_BASE                  = .;
        .               = . + 512;
        __BDT_END       = .;

                . = ALIGN(0x4);
            } >> userram

            .custom :
            {
                ___HEAP_START           = .;
                ___heap_addr            = ___HEAP_START;
                ___HEAP_END             = ___HEAP_START + ___heap_size;
                ___SP_END               = ___HEAP_END;
                ___SP_INIT              = ___SP_END + ___stack_size;

                . = ALIGN (0x4);
            } >> userram

            ___VECTOR_RAM                       = ADDR(.vectorram);

            __SP_INIT               = ___SP_INIT;

            _romp_at = ___ROM_AT + SIZEOF(.data);
            .romp : AT(_romp_at)
            {
                __S_romp = _romp_at;
                WRITEW(___ROM_AT);
                WRITEW(ADDR(.data));
                WRITEW(SIZEOF(.data));
                WRITEW(0);
                WRITEW(0);
                WRITEW(0);
            }

}
```