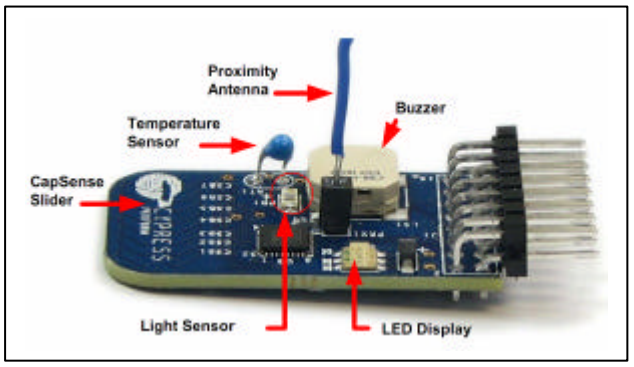
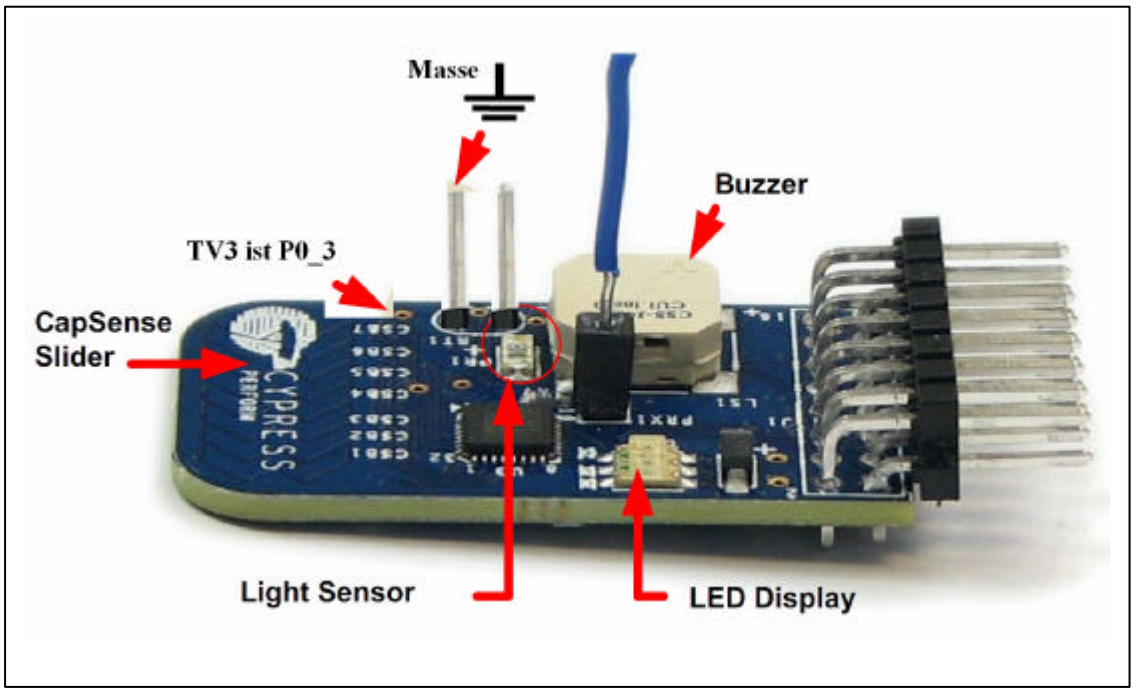
Ziel: Es wird gezeigt, wie man mit dem PSoC FirstTouch Starter Kit über USB_I2C Messwerte auf dem Computer graphisch darstellen kann.

Bei z.B. http://www.msc-ge.com/ kauft man das PSoC FirstTouch Starter Kit für 19.99Euro + ... und über http://www.cypress.com/firsttouch/ und hier auf der Registerkarte Design Examples kann man zu den auf dem Expansion Board verbauten Sensoren Beispiel-Anwendungen und Anleitungen herunterladen.

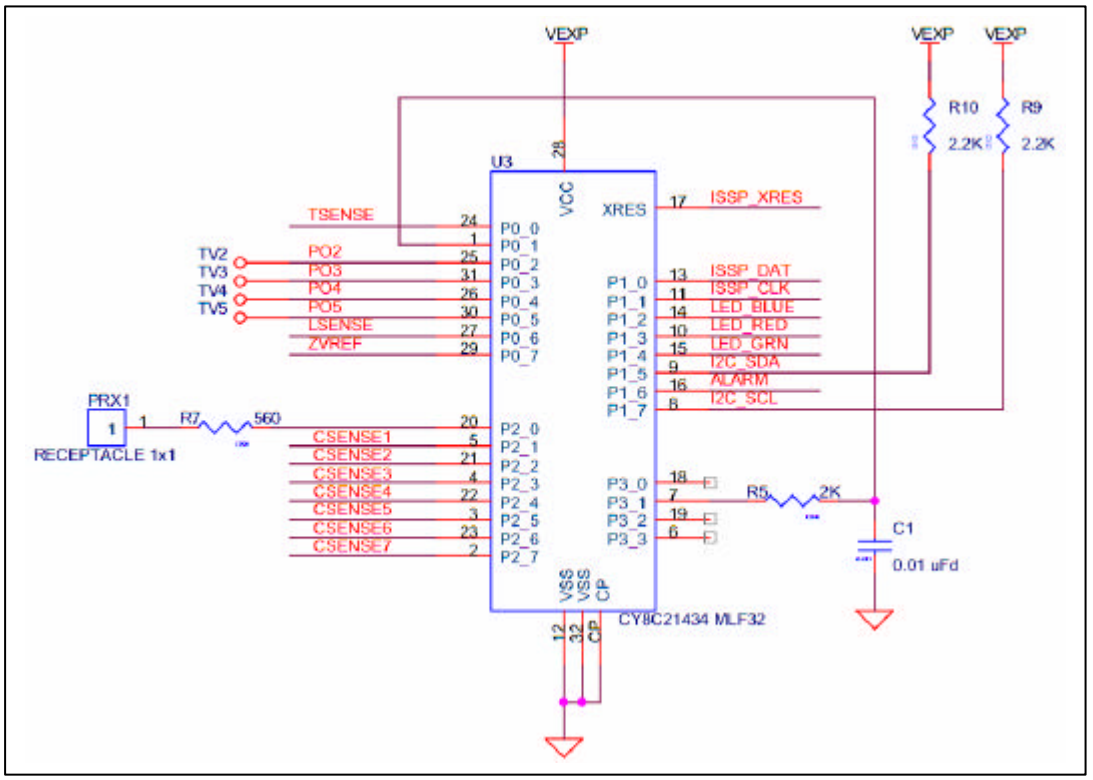
Auf der abgebildeten Expansion Card sind die unbenutzten Ports P0_2 bis P0_5 ansatzweise herausgeführt und so leichter zu erreichen.

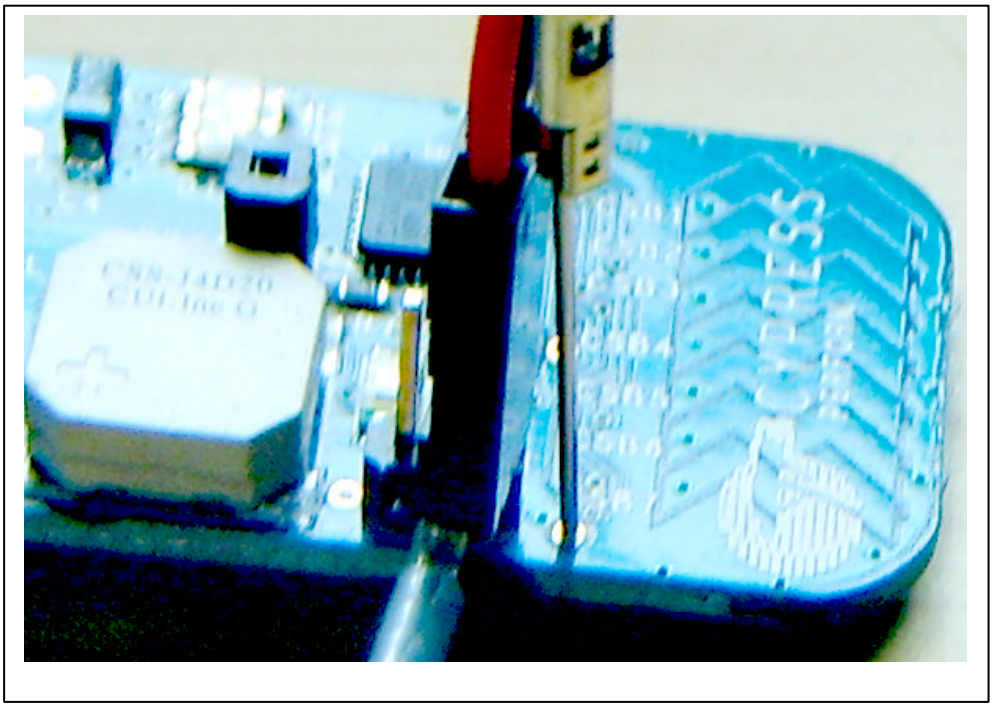


Wir löten den Temperatursensor heraus und 2 Pfostenkontakte hinein.



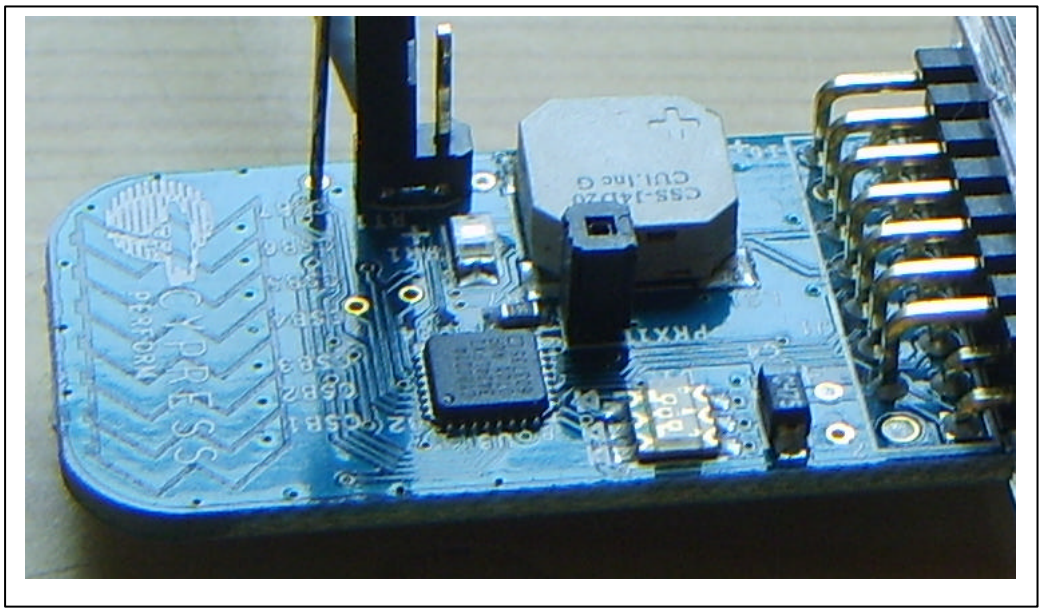
Die Bilder und der Schaltplan sind aus: First Touch Guide.pdf





Wir benutzen eine Nadel.

Festgelötet wird nichts; so ist das Board immer anders verwendbar.

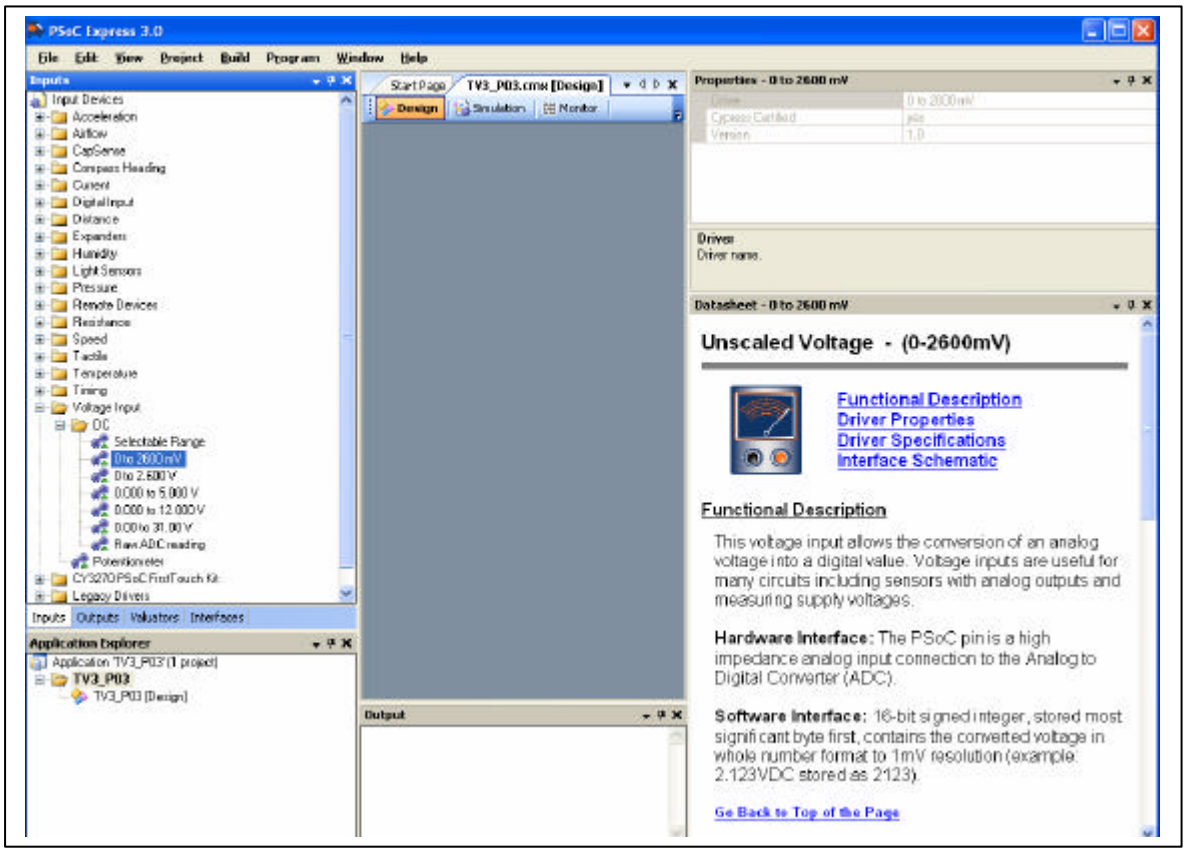


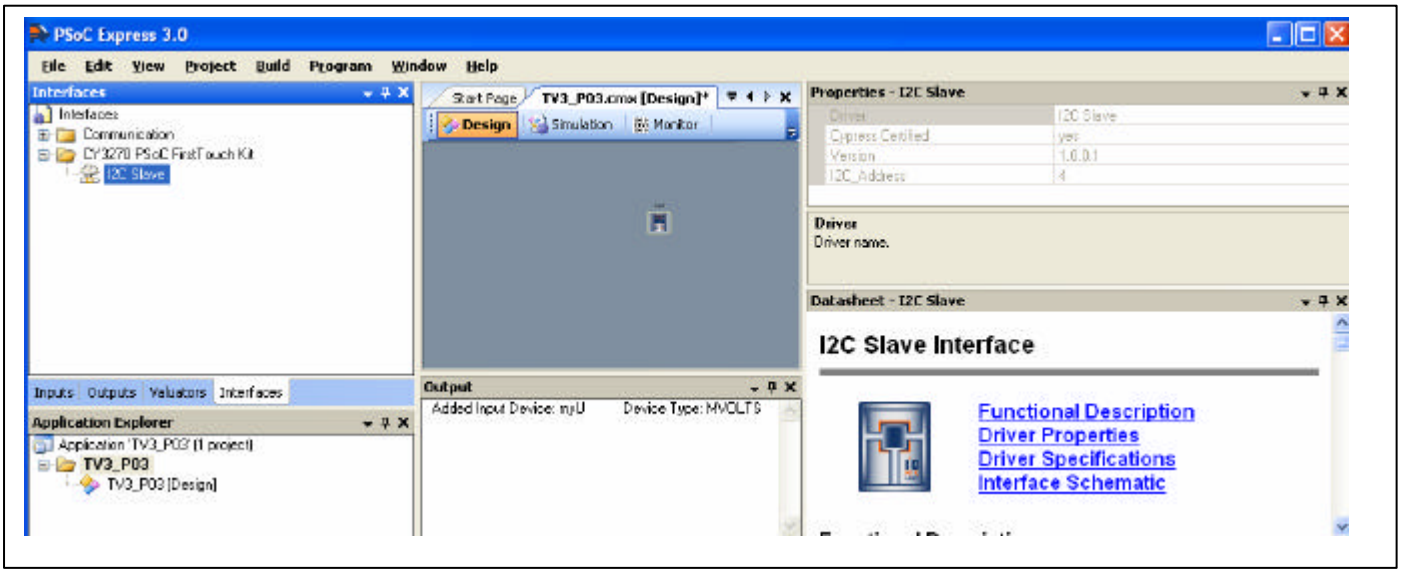
Visual Embedded Design Tool: PSoC Express 3.0

Über File/New Project… erzeugt man ein Projekt namens TV3_P03. Browsen Sie im New Project Dialog an eine für Sie geeignete HD-Stelle und setzen Sie das Häkchen „Create directory for application“. Das Verzeichnis „TV3_P03“ wird auf der Festplatte erzeugt.

Da wir Spannungen messen und diese Signale graphisch darstellen wollen, wählen wir im Input-Katalog Voltage Input/DC/0 to 2600mV per Doppelklick. Im darauf folgenden Add Input Driver Dialog schreiben wir für den Namen der Spannungsvariablen: myU, und klicken auf OK. Ein Icon mit dem Namen myU erscheint im Designer und über Project/Assign Register Map for TV3_P03 Project… kann man sich von der Existenz der 16-bit Registervariablen überzeugen. Da wir deren Wert über USB_I2C in WinXP verarbeiten möchten benötigen wir noch das I2C-Interface. Per Doppelklick auf I2C erscheint der Add Interface Dialog. Wir vergeben den Namen I2Ccom und akzeptieren die I2C-Adresse mit der Nummer 4. Sie wird später in der WinXP-Application benötigt und über USB_I2C für read/write-Anforderungen gesendet.

Im nächsten Schritt sagen wir PSoC-Express welche IC-Pins wir mit welcher Aufgabe belegen möchten.

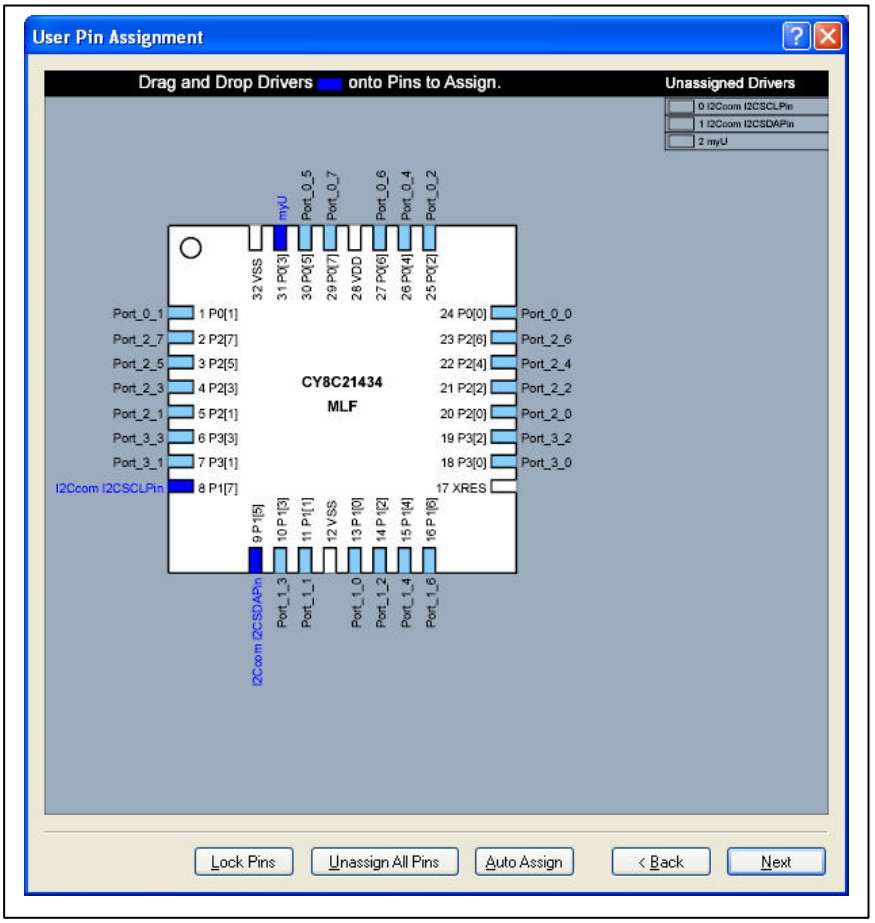




Dazu klicken wir auf Build/Generate/Build TV3_P03 Project. Es erscheint der PSoC Device Configuration Selection Dialog in dem wir sehr sorgfältig das auf dem Expansion Board verbaute IC auswählen müssen. Wir wählen CY8C21030

und hier das CY8C21434, 32 Pin Modell aus. Über den Button Next – und das dauert 30 Sekunden – gelangen wir zum User Pin Assignment Dialog.

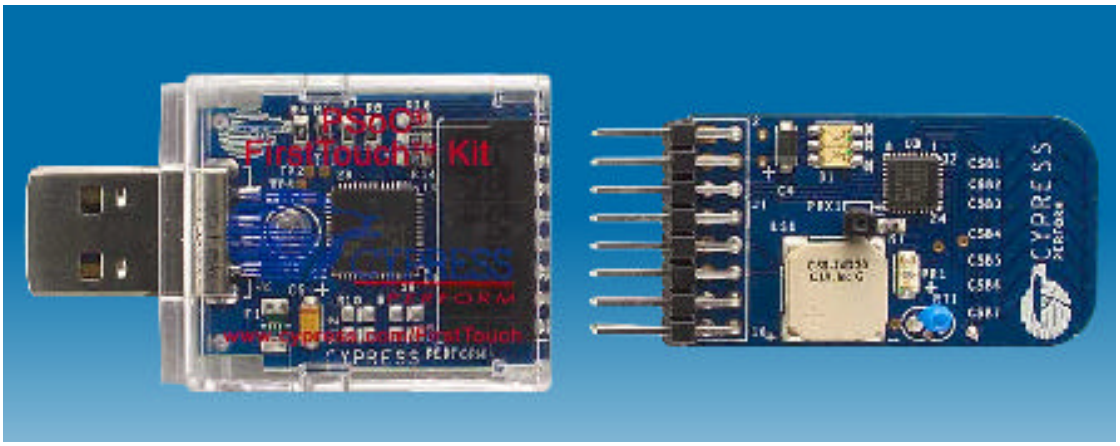
Wir klicken auf Unassign All Pins und belegen dann manuell die richtigen drei. Als Unassigned Drivers werden von PSoC-Express I2CSCL, I2CSDA und myU aufgeführt. Port_1_7 wird per Drag-and-Drop zum I2CSCL-Pin, Port_1_5 zum I2CSDA-Pin und Port_0_3 zum myU-Pin. Mit einem Klick auf Next generiert PSoC-Express das Projekt: es schreibt für uns den C-Code.



Über Program/PSoC Programmer wird nun im letzten Schritt das IC programmiert. Der Programmer sollte immer die neuesten Web-Updates enthalten. Wir müssen auf Port, Device Family, Device und den Programming Mode achten. Den Port stellen wir auf FirstTouch/…, die Device Family auf 21x34 und das Device ist unser IC vom Typ CY8C21434… .
Den Programming Mode stellen wir auf Reset. Als letztes klicken wir in der Zeile: File Load, Program, Checksum, Read auf den Button Program und das war’s.

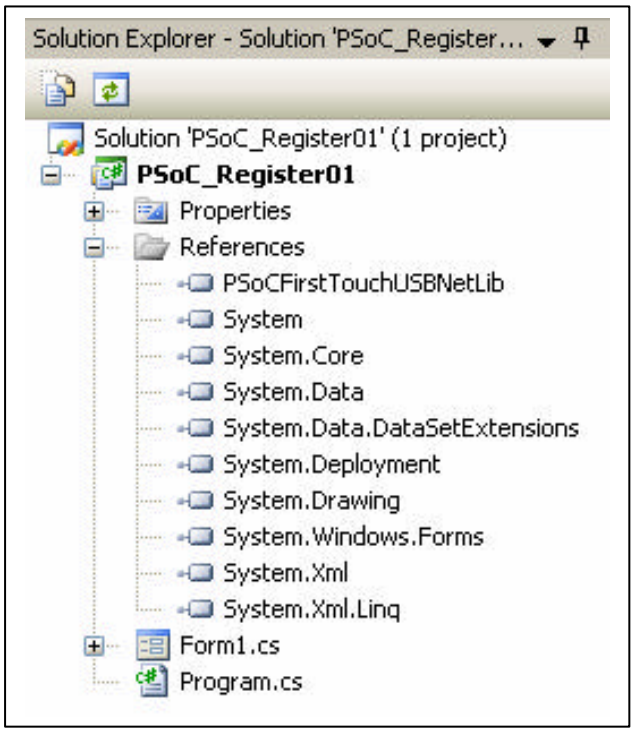
Visual Studio 2008 Visual C# GUI-Applikation zu PSoC-USB-I2C

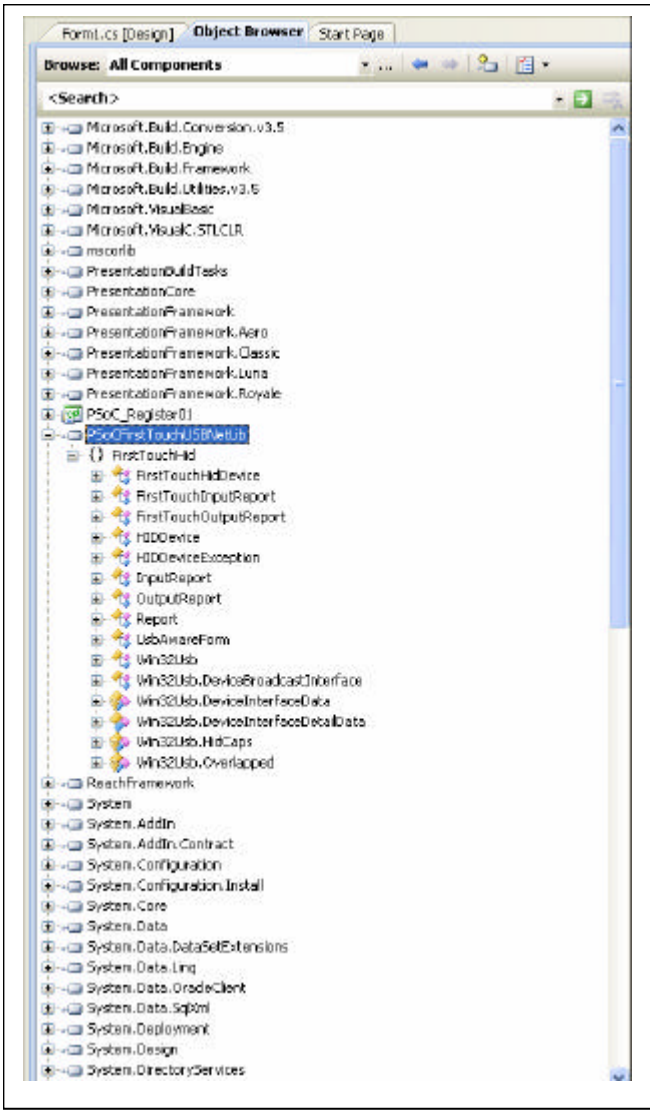
Wir benötigen das von http://www.microsoft.com/express/vcsharp/ das frei erhältliche Visual Studio 2008 Visual C#.



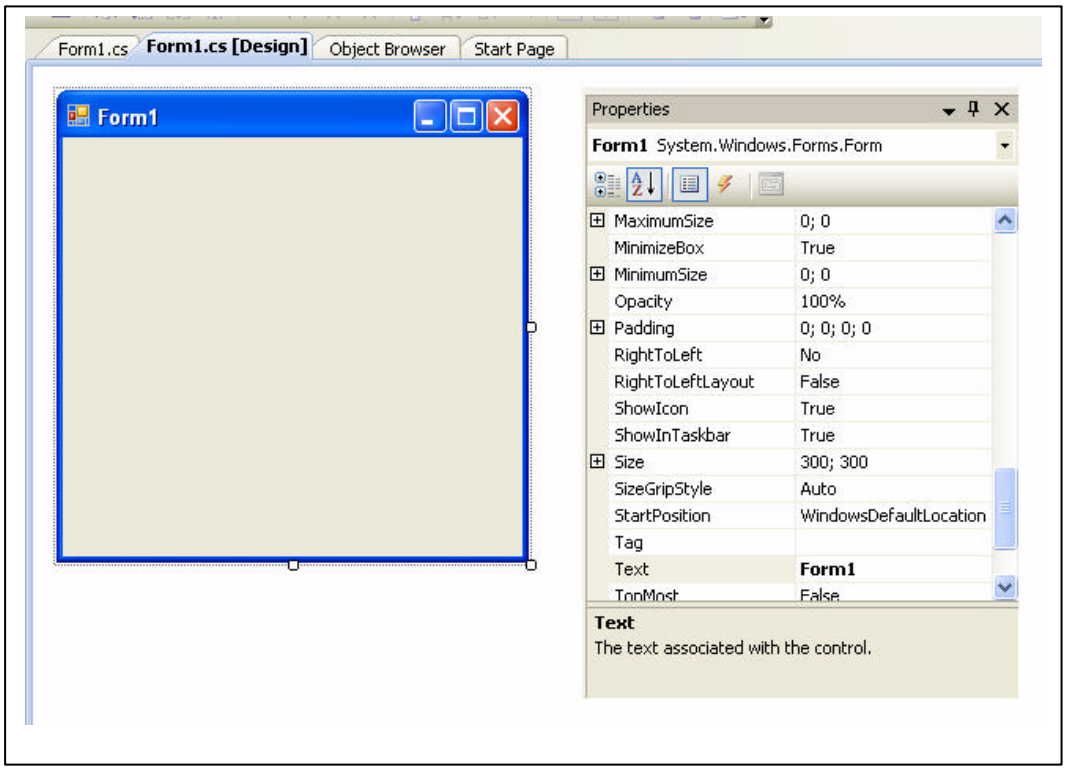
Microsoft Visual C# 2008 Express Edition präsentiert sich mit der Start Page. Über File/New Project… erzeugt man eine Windows Forms Application und gibt ihr den Namen USB_I2C_App_01. Das Project wird erzeugt. Ist C# damit fertig, speichert man das Project über File/Save All auf der Festplatte. Browsen Sie im Save Project Dialog an eine für Sie geeignete HD-Stelle und setzen Sie das Häkchen „Create directory for solution“. Das Verzeichnis „USB_I2C_App_01“ wird auf der Festplatte erzeugt.
In dieses Verzeichnis kopieren Sie jetzt die PSoCFirstTouchUSBNetLib.dll. Um nun in ihrem Project diese DLL nutzen zu können müssen Sie Microsoft Visual C# 2008 dies mitteilen. Dazu gehen Sie mit der rechten Maustaste (RMB) auf Solution Explorer/References und Add Reference…, wählen und über die Registerkarte „Browse“ die PSoCFirstTouchUSBNetLib.dll aus und binden sie ein.

Doppelklickt man jetzt unter References auf PSoCFirstTouchUSBNetLib, so öffnet sich im Object-Browser der Inhalt der DLL.





Mit RMB auf Form1.cs im Solution Explorer lässt man sich den momentanen Quelltext anzeigen: „View Code“ und man wählt zusätzlich noch „View Designer“.
Klickt man nun einmal auf das Fenster Form1, werden einem rechts die Properties dieser Form angezeigt. Dort gelangt man über den orangenen Blitz zu den Events des Fensters Form1. Wir doppelklicken mit LMB auf das Ereignis „Paint“.



Die Prozedur

```
private void Form1_Paint(object sender, PaintEventArgs e)
{

}
```

wird erzeugt.

Nun fügen wir noch einen Timer ein, der alle paar Millisekunden über USB mit unserem IC Kontakt aufnimmt. Dazu aktivieren wir die Seite Form1.cs[Design]*. Aus der dann links geöffneten ToolBox ziehen wir per Drag-and-Drop den Timer in das Form1-Fenster. Mit den gleichen Schritten wie man zur Form1_Paint-Methode kommt, gelangt man nun zur timer1_Tick-Methode:

```
private void timer1_Tick(object sender, EventArgs e)
{

}
```

Damit sieht Form1.cs* wie folgt aus:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace USB_I2C_App_01
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs
e)
        {

        }

        private void timer1_Tick(object sender, EventArgs e)
        {

        }
}
```

Diesen Quelltext erweitern wir nun zu

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using FirstTouchHid;
```

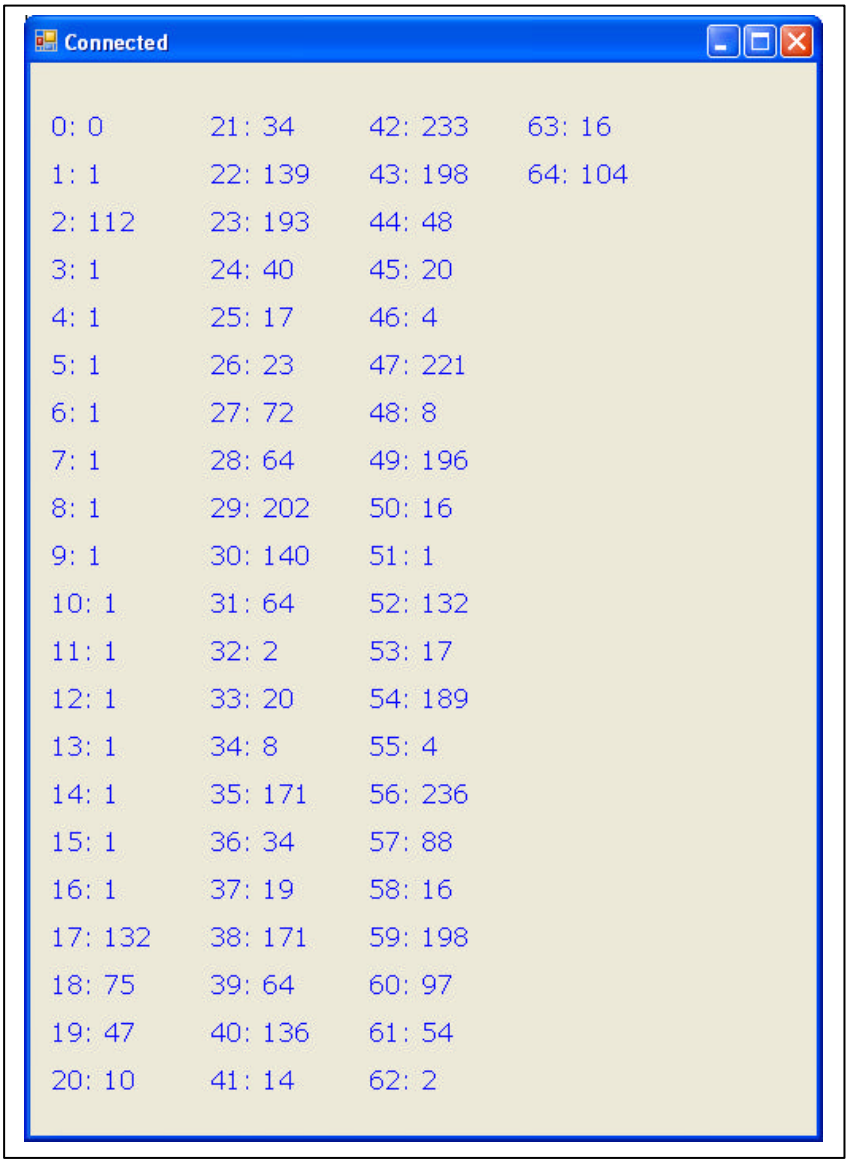
```
namespace USB_I2C_App_01
{
   public partial class Form1 : Form
   {

      //Gadget is in this case PSoC FirstTouch Starter Kit
      private FirstTouchHidDevice pGadget = null;
      const int n_data = 65;
      private byte[] cmd_bytes = new byte[n_data];
      private byte[] data_bytes = new byte[n_data];
      IntPtr ip_USB_EventHandle;
      private Guid g_DeviceClass;
      private int NumProc;
      const int cmd_read = 0;
      const int cmd_write = 1;
      const int I2Ccom_address = 4; //See PSoC Express and the I2C-properties

      public Form1()
      {
         //Is the application already running?
         Process[] processName = Process.GetProcessesByName("USB_I2C_App_01");
         if (processName != null)
         {
            NumProc = processName.Length;
            if (NumProc > 1)
            {
               MessageBox.Show("PSoC FirstTouch Starter Kit already running.", "ERROR",
                     MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
               Environment.Exit(1);
            }
            else
            {
               //MessageBox.Show("USB_I2C_App_01", "ok",
               //           MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
         }
         InitializeComponent();
         timer1.Interval = 1;  //minimal 1; 1000=1000ms

         /*Is the gadget connected?*/
         if (GetHidDevice())
         {
            this.Text = "Connected";
            this.timer1.Enabled = true;
         }
         else
         {
            this.Text = "Disconnected";
            this.timer1.Enabled = false;
            MessageBox.Show("GetHidDevice()", "Disconnected",
                     MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
        }

    }

    //Support for Human Interface devices (HID)
    private bool GetHidDevice()
    {
        bool Status = false;
        //idVendor:  0x04B4 = Cypress Semiconductor
        //idProduct: 0xF110 with UVCView.x86.exe
        //http://msdn2.microsoft.com/en-us/library/aa906848.aspx
        pGadget = FirstTouchHidDevice.FindFirstTouchDevice(0x04b4, 0xF110);
        if (pGadget != null){Status = true;}
        //RegisterForUsbEvents: attach/removal
        g_DeviceClass = Win32Usb.HIDGuid;
        //Class View then RMB at FirstTouchHid then Browse Definition
        ip_USB_EventHandle = Win32Usb.RegisterForUsbEvents(this.Handle,
g_DeviceClass);
        return (Status);
    }

    private void write_to_IC(Byte IC_address, Byte n_bytes, Byte read_or_write_byte)
    {
        if (read_or_write_byte == cmd_read)cmd_bytes[1] = 0x00;
        else if (read_or_write_byte == cmd_write)cmd_bytes[1] = 0x01;

        cmd_bytes[2] = n_bytes;
        cmd_bytes[3] = IC_address;

        pGadget.WriteToPSoC(cmd_bytes);
    }

    private bool read_from_IC(Byte IC_address, Byte n_bytes)
    {
        int sleep_time;
        bool Status = true;
        //calculate the sleep_time
        sleep_time = 1 + (n_bytes / 8);
        //inform IC what you want: read
        if (pGadget != null && data_bytes != null)
        {
            try{write_to_IC(IC_address, n_bytes, cmd_read);}
            catch{}
        }
        //Since USB and I2C needs time
        System.Threading.Thread.Sleep(sleep_time);
        //read what you get from the IC via USB_I2C
        if (pGadget != null)
        {
            data_bytes = pGadget.ReadFromPSoC();
        }
```

```
            //is there anything?
            if (data_bytes == null)
            {
                Status = false;
            }
            return (Status);
        }

        //An Event is called when a new device is detected
        protected override void WndProc(ref Message m)
        {
            if (m.Msg == Win32Usb.WM_DEVICECHANGE)  //WM: An USB Device has
changed
            {
                switch (m.WParam.ToInt32())
                {
                    case Win32Usb.DEVICE_ARRIVAL:   //USB-Gadget inserted
                        if (GetHidDevice())
                        {
                            this.Text = "Connected";
                            timer1.Enabled = true;
                        }
                        break;
                    case Win32Usb.DEVICE_REMOVECOMPLETE:    //USB-Gadget inserted
                        if (!GetHidDevice())
                        {
                            this.Text = "Disconnected";
                            timer1.Enabled = false;
                        }
                        break;
                }
            }
            //Debug.WriteLine(m); //View/Output RMB Clear all
            base.WndProc(ref m);
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            //Unter Form1.cs[Design]* unten timer1 Properties kann man die Zeit Einstellen
            timer1.Enabled = false;
            read_from_IC(I2Ccom_address, n_data - 2);
            this.Refresh();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Font fn = new Font("Verdana", 12);
            Brush br = new SolidBrush(Color.Blue);
            int n;
            for (n = 0; n < n_data; n++)
```

```
            g.DrawString(n.ToString() + ": " + data_bytes[n].ToString(), fn, br,
                10 + (n / 21) * 100, ((n % 21) * 30) + 30);
        timer1.Enabled = true;
    }

  }
}
```

Diese Anwendung ist wichtig, zeigt sie einem doch, welche Inhalte die einzelnen gesendeten Bytes enthalten.
Man erhält 65 übertragene Bytes angezeigt. Byte Nr.0 ist 0, Byte Nr.1 ist 1, Byte Nr.2 ist 112 usw. Legt man an das Expansion Board eine variierbare Spannung zwischen 0 und 2600mV an, und variiert diese ständig – variables Netzteil mit Spannungsteiler – dann erkennt man, dass sich nur Byte 1 und Byte 2 verändern. Genau sie enthalten den 16-bit-Code von TV3 = P03 = IC-Pin 31 = myU.



Für die graphische Darstellung fügen wir die beiden Bytes wieder zu einem 16-bit-Wert zusammen:

```
y = (data_bytes[1]<<8) + data_bytes[2];
```

und wir erhalten mit ein paar Zeilen zusätzlichem Code folgende graphische Darstellung:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using FirstTouchHid; //Solution Explorer RMB auf References Add Reference Browse
PSoCFirstTouchUSBNet.lib


namespace USB_I2C_App_01
{
   public partial class Form1 : Form
   {
       private double xmin, xmax, ymin, ymax;
       private double xStreckfaktor, yStreckfaktor;
       double yA=0;
       //Gadget is in this case PSoC FirstTouch Starter Kit
       private FirstTouchHidDevice pGadget = null;
       const int n_data = 65;
       private byte[] cmd_bytes = new byte[n_data];
       private byte[] data_bytes = new byte[n_data];
       IntPtr ip_USB_EventHandle;
       private Guid g_DeviceClass;
       private int NumProc;
       const int cmd_read = 0;
       const int cmd_write = 1;
       const int I2Ccom_address = 4; //See PSoC Express and the I2C-properties
       int n_w, n_x = 0;

       public Form1()
       {
           //Is the application already running?
           Process[] processName = Process.GetProcessesByName("USB_I2C_App_01");
           if (processName != null)
```

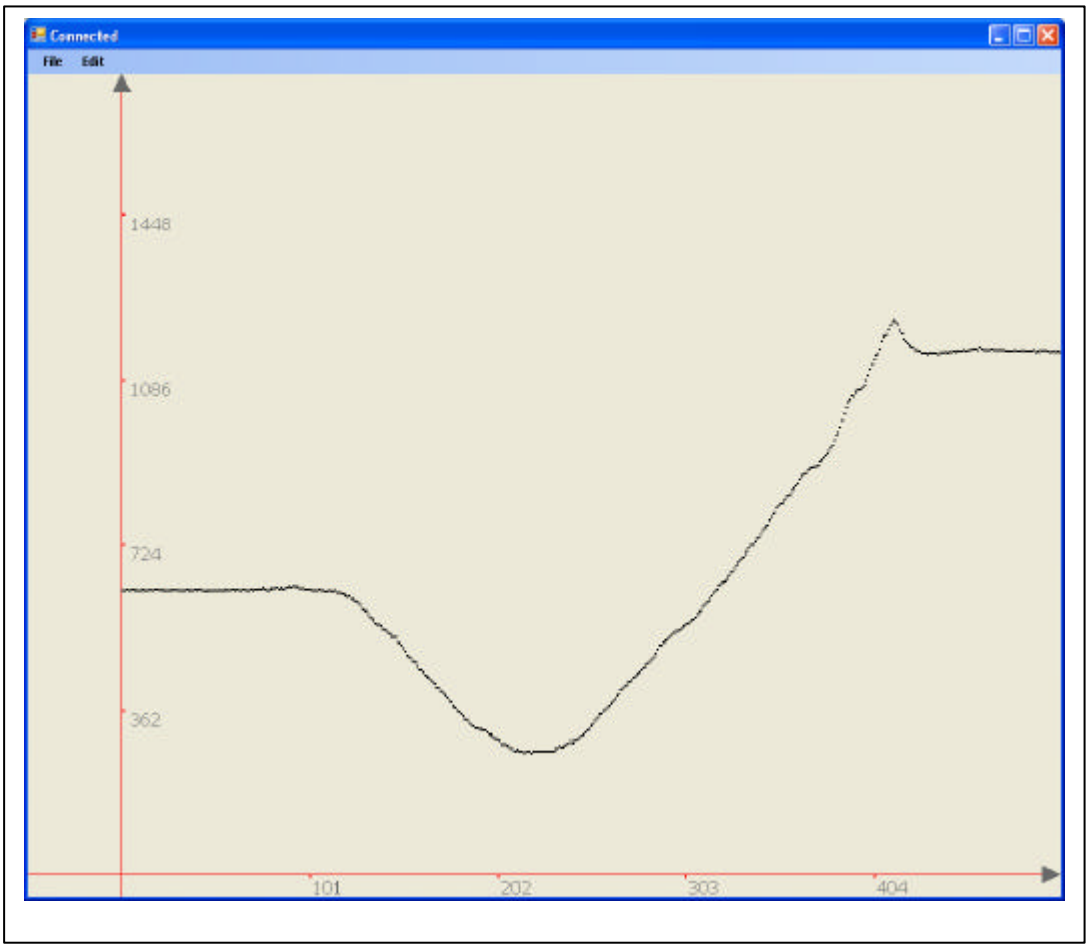
```
    {
        NumProc = processName.Length;
        if (NumProc > 1)
        {
            MessageBox.Show("PSoC FirstTouch Starter Kit already running.", "ERROR",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            Environment.Exit(1);
        }
        else
        {
            //MessageBox.Show("USB_I2C_App_01", "ok",
            //          MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
    InitializeComponent();
    timer1.Interval = 1;  //minimal 1; 1000=1000ms
    n_w = this.ClientRectangle.Width;
    xmax = n_w;
    xmin = -50;
    ymax = 1810;
    ymin = -50;
    xStreckfaktor = this.ClientRectangle.Width / (xmax - xmin);
    yStreckfaktor = this.ClientRectangle.Height / (ymax - ymin);


    /*Is the gadget connected?*/
    if (GetHidDevice())
    {
        this.Text = "Connected";
        this.timer1.Enabled = true;
    }
    else
    {
        this.Text = "Disconnected";
        this.timer1.Enabled = false;
        MessageBox.Show("GetHidDevice()", "Disconnected",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

}
public int xX(double x)
{
    return (int)Math.Round(xStreckfaktor * (x - xmin));
}

public int yY(double y)
{
    return (int)Math.Round(yStreckfaktor * (ymax - y));
}

public double Xx(int x)
```

```
{
    return x / xStreckfaktor + xmin;
}

public double Yy(int y)
{
   return ymax - y / yStreckfaktor;
}

public void Koordinatensystem(Graphics g, double dx, double dy, double hier_yAchse)
{
   Font fn = new Font("Verdana", 12);
   Brush br = new SolidBrush(Color.DimGray);
   Pen pen = new Pen(Color.Red, 1/*Strichdicke*/);

   Point[] pxA =
   {
      new Point(xX(xmax),yY(0)),
      new Point(xX(xmax) - 18,yY(0) - 9),
      new Point(xX(xmax) - 18,yY(0) + 9),
      new Point(xX(xmax),yY(0))
   };
   int h = menuStrip1.Height;
   Point[] pyA =
   {
      new Point(xX(hier_yAchse) + 9, yY(ymax) + 18 + h),
      new Point(xX(hier_yAchse), yY(ymax)-2 + h),
      new Point(xX(hier_yAchse) - 9, yY(ymax) + 18 + h),
      new Point(xX(hier_yAchse) + 9, yY(ymax) + 18 + h)
   };

   g.DrawLine(pen, xX(xmin), yY(0), xX(xmax), yY(0));//x-Achse
   g.FillPolygon(br, pxA);//Pfeil der x-Achse

   g.DrawLine(pen, xX(hier_yAchse), yY(ymin), xX(hier_yAchse), yY(ymax));//y-
Achse
   g.FillPolygon(br, pyA);//Pfeil der y-Achse


   double x = dx;
   while (x < xmax)
   {
      g.DrawString(x.ToString("#.#"), fn, br, xX(x), yY(0) + 3);
      g.DrawRectangle(pen, xX(x), yY(0), 1, 3);
      x += dx;
   }
   x = -dx;
   while (x > xmin)
   {
      g.DrawString(x.ToString(), fn, br, xX(x), yY(0) + 3);
      g.DrawRectangle(pen, xX(x), yY(0), 1, 3);
```

```
            x -= dx;
        }
        x = dy;
        while (x < ymax)
        {
            g.DrawString(x.ToString("0.#####"), fn, br, xX(hier_yAchse) + 5, yY(x));
            g.DrawRectangle(pen, xX(hier_yAchse), yY(x), 3, 1);
            x += dy;
        }
        x = -dy;
        while (x > ymin)
        {
            g.DrawString(x.ToString("0.#####"), fn, br, xX(hier_yAchse) + 5, yY(x));
            g.DrawRectangle(pen, xX(hier_yAchse), yY(x), 3, 1);
            x -= dy;
        }
    }

    //Support for Human Interface devices (HID)
    private bool GetHidDevice()
    {
        bool Status = false;
        //idVendor:  0x04B4 = Cypress Semiconductor
        //idProduct: 0xF110 with UVCView.x86.exe
        //http://msdn2.microsoft.com/en-us/library/aa906848.aspx
        pGadget = FirstTouchHidDevice.FindFirstTouchDevice(0x04b4, 0xF110);
        if (pGadget != null){Status = true;}
        //RegisterForUsbEvents: attach/removal
        g_DeviceClass = Win32Usb.HIDGuid;
        //Class View then RMB at FirstTouchHid then Browse Definition
        ip_USB_EventHandle = Win32Usb.RegisterForUsbEvents(this.Handle,
g_DeviceClass);
        return (Status);
    }

    private void write_to_IC(Byte IC_address, Byte n_bytes, Byte read_or_write_byte)
    {
        if (read_or_write_byte == cmd_read)cmd_bytes[1] = 0x00;
        else if (read_or_write_byte == cmd_write)cmd_bytes[1] = 0x01;

        cmd_bytes[2] = n_bytes;
        cmd_bytes[3] = IC_address;

        pGadget.WriteToPSoC(cmd_bytes);
    }

    private bool read_from_IC(Byte IC_address, Byte n_bytes)
    {
        int sleep_time;
        bool Status = true;
        //calculate the sleep_time
```

```
            sleep_time = 1 + (n_bytes / 8);
            //inform IC what you want: read
            if (pGadget != null && data_bytes != null)
            {
               try{write_to_IC(IC_address, n_bytes, cmd_read);}
               catch{}
            }
            //Since USB and I2C needs time
            System.Threading.Thread.Sleep(sleep_time);
            //read what you get from the IC via USB_I2C
            if (pGadget != null)
            {
               data_bytes = pGadget.ReadFromPSoC();
            }
            //is there anything?
            if (data_bytes == null)
            {
               Status = false;
            }
            return (Status);
         }

         //An Event is called when a new device is detected
         protected override void WndProc(ref Message m)
         {
            if (m.Msg == Win32Usb.WM_DEVICECHANGE)  //WM: An USB Device has
changed
            {
               switch (m.WParam.ToInt32())
               {
                  case Win32Usb.DEVICE_ARRIVAL:   //USB-Gadget inserted
                     if (GetHidDevice())
                     {
                        this.Text = "Connected";
                        timer1.Enabled = true;
                     }
                     break;
                  case Win32Usb.DEVICE_REMOVECOMPLETE:    //USB-Gadget inserted
                     if (!GetHidDevice())
                     {
                        this.Text = "Disconnected";
                        timer1.Enabled = false;
                     }
                     break;
               }
            }
            //Debug.WriteLine(m); //View/Output RMB Clear all
            base.WndProc(ref m);
         }

         private void exitToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            this.Close();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            //Unter Form1.cs[Design]* unten timer1 Properties kann man die Zeit Einstellen
            timer1.Enabled = false;
            Graphics g = Graphics.FromHwnd(Handle);
            Pen pn = new Pen(Brushes.Black);
            pn.Width = 2;
            int y=0;
            if (read_from_IC(I2Ccom_address, n_data - 2))
            {
                y = (data_bytes[1]<<8) + data_bytes[2];
                if (n_x <= n_w)
                {
                    g.DrawLine(pn, xX(n_x), yY(y), xX(n_x) + 2, yY(y));
                    n_x++;
                }
            }
            timer1.Enabled = true;
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Font fn = new Font("Verdana", 12);
            Brush br = new SolidBrush(Color.Blue);
             Koordinatensystem(g, xmax / 5, ymax / 5, yA);
            timer1.Enabled = true;
        }

        private void Form1_SizeChanged(object sender, EventArgs e)
        {
            xStreckfaktor = this.ClientRectangle.Width / (xmax - xmin);
            yStreckfaktor = this.ClientRectangle.Height / (ymax - ymin);
            n_w = this.ClientRectangle.Width;
            n_x = 0;
            this.Refresh();
        }

    }
}
```



Danke an Cypress und viel Spaß beim weitern Experimentieren,

Edgar Marx
edgarmarx@t-online.de