# Das LED-Info-Display selbst programmieren – Demoanwendung ID100

Produktvideo Infos unter Webcode #2001

**Für das im ELVjournal 4 und 5/2012 vorgestellte LED-Info-Display gibt es eine einfach konfigurierbare PC-Software für die Ausgabe von Anzeigen und Lauftexten. Wer über dieses Angebot hinaus das ID100 individuell nutzen und fortgeschritten programmieren möchte, erhält mit den hier vorgestellten Demoanwendungen neben dem eigentlichen Anwendungsprogramm ein Werkzeug sowie eine Anleitung dazu.**

## Information zur gewählten Entwicklungsumgebung

Bei der Erstellung der Demoanwendung kam die Programmiersprache C# zum Einsatz. Als Entwicklungsumgebung kann das kostenlose Visual Studio 2010 Express [1] oder eine der Nachfolgeversionen verwendet werden. Angesprochen wird das ID100 mittels des Silabs-USB-Xpress-Treibers und der SiUSBXp-Kommunikationsbibliothek (SiUSBXp.dll).

Mit dem der Demosoftware [2] beiliegenden CP2102-ID-Changer wird das Info-Display auf die Erkennung als virtueller COM-Port (VCP) umprogrammiert. Die Installation des Treibers ist für die erfolgreiche Erkennung des ID100 notwendig. Der Treiber liegt der Demosoftware [2] bei oder kann alternativ auf der Webseite von Silabs [3] für das jeweilige Betriebssystem bezogen werden. Aufgrund der Nutzung des ID100 über den virtuellen COM-Port ist es universell einsetzbar, da dieser bereits von vielen Programmiersprachen ohne zusätzliche Kommunikationsbibliotheken unterstützt wird. Sofern das jeweilige System die gewählte Programmiersprache unterstützt, besteht auch die Möglichkeit, das ID100 nach der Umprogrammierung an einem Linux- oder Mac-System mit den jeweiligen Treibern von Silabs [3] zu verwenden.

### Kommunikation über USB

Für die Kommunikation mit dem ID100 mittels des virtuellen COM-Ports kann die SerialPort-Klasse aus dem .NET-Framework verwendet werden. Dieser Klasse werden der Name und die Kommunikationsparameter übergeben, welche in Tabelle 1 „VCP Kommunikationsparameter“ einzusehen sind. Anschließend wird die Verbindung über die Open-Methode der SerialPort-Klasse initialisiert und es können im Anschluss die Read- und Write-Methoden zur Kommunikation genutzt werden.

Tabelle 1

| VCP Kommunikationsparameter | |
|---|---|
| Baudrate | 38.400 |
| Datenbits | 8 |
| Parität | none/keine |
| Stoppbits | 1 |

Die Datenübertragung erfolgt in einem festen Kommunikationsrahmen (siehe Tabelle 2). Bei der Übertragung gibt es zwei Steuerzeichen, die in den zu übertragenden Daten nicht vorkommen dürfen: Das Start-Zeichen (0x02) und das Escape-Zeichen (0x10). Wenn eines dieser Steuerzeichen in der Länge, den Nutzdaten oder der Prüfsumme enthalten ist, wird diesem das Escape-Zeichen vorangestellt und das höchstwertige Bit (0x80) gesetzt. Hieraus ergeben sich folgende Umsetzungen: 0x02 wird zu 0x1082 und 0x10 wird zu 0x1090.

In der Demoanwendung findet die Kommunikation in der ComPortDevice-Klasse statt. Die einzelnen Befehle und die dazugehörigen Antworten können den Tabellen 3 und 4 entnommen werden. Bei allen 16-Bit (2-Byte)-Zahlen wird das jeweils höchstwertige Byte zuerst übertragen (Big-Endian).

**Prüfsummenberechnung**

Zur Sicherung der Übertragung wird aus den zu übertragenden Daten eine Prüfsumme (CRC) mit einer Länge von 16 Bit (2 Byte) berechnet. Die Parameter für die Berechnung sind der Initialwert (0xFFFF) und das CRC-Polynom (0x8005). Details zur Berechnung sind auf Wikipedia [4] zu finden.

In der Demosoftware verwenden wir für die Berechnung eine eigene „CRC16"-Klasse, die die CRC-Berechnung letztlich durchführt. Nach der Initialisierung erfolgt die Berechnung der Prüfsumme durch Übergabe der Daten an die Shift-Methoden. Hierbei ist zu beachten, dass zuvor die Steuerzeichen aus der Länge und den Nutzdaten zu ersetzen sind, bevor diese an die Shift-Methoden übergeben werden. Anschließend werden die Steuerzeichen aus dem errechneten CRC entfernt und zusammen mit den übrigen Daten an das Gerät übertragen.

Tabelle 2

**Kommunikationsrahmen**

| Inhalt | Byteanzahl | Beschreibung |
|---|---|---|
| STX (0x02) | 1 | konstantes Startzeichen |
| Nutzdatenlänge | 2 | Anzahl der Nutzdaten (Befehl und Parameter) im Rahmen |
| Befehl | 1 | Befehl der Übertragung siehe Tabelle 3 und 4 |
| Parameter | 0 bis 258 | Parameter des Befehls siehe Tabelle 3 und 4 |
| Prüfsumme | 2 | CRC16 von allen vorherigen Bytes, siehe Abschnitt Prüfsummenberechnung |

**Bild übertragen**

Die Übertragung eines kompletten Bildes kann mit dem Befehl ‚D' erfolgen. Für die gewünschte Darstellung des Bildes auf dem ID100 ist die richtige Reihenfolge der Daten wichtig. Betrachtet man das Info-Display von vorn, so kann es als Koordinatensystem interpretiert werden, allerdings ist der Ursprung, also der Punkt, an dem die X- und die Y-Koordinate den Wert „0" annehmen, oben links. Die X-Achse geht von dort aus nach rechts und umfasst somit einen Bereich von „0" bis „16". Die Besonderheit ist hierbei der Verlauf der Y-Achse, dieser erstreckt sich von oben nach unten und umfasst einen Bereich von „0" bis „11". Für ein ganzes Bild sind 26 Byte zum ID100 zu übertragen. Eine einzelne Matrix wird somit durch eine Aneinanderreihung dieser Bytes repräsentiert. Eine genauere Aufschlüsselung kann man der Tabelle 5 entnehmen.

**Uhrkonfiguration übertragen**

Die Konfiguration der Uhr umfasst je ein Bild pro Sekunde des Tages und somit werden 86.400 Bilder (60 Sekunden * 60 Minuten * 24 Stunden) für einen ganzen Tag benötigt. Es werden immer 6 Bilder in einer einzelnen Flashpage gespeichert, wodurch beispielsweise eine Flashpage für 6 Sekunden, 10 Flashpages für eine Minute und 600 Flashpages für eine Stunde benötigt werden. Es entsteht also ein Tagesgesamtbedarf von 14.400 Flashpages, deren Index den Bereich von 0 bis 14.399 umfasst. Jedes Bild benötigt 26 Byte, wie im Abschnitt „Bild übertragen" erläutert. Die Übertragung der Daten erfolgt jeweils mit den Befehlen ‚F' zum Schreiben und ‚f' zum Lesen.

**Termine übertragen**

Für die Termine stehen ab dem Flashpage-Index 14.400 weitere 20 Flashpages zur Verfügung, wobei eine Flashpage für je einen Termin belegt wird. Die Termindaten umfassen insgesamt 34 Byte pro Termin, wobei hiervon erneut 26 Byte für die LED-Matrix benötigt werden. Zusätzlich werden zwei Byte benötigt, um zu überprüfen, ob der Termin („Active Flag") und das Overlay („Overlay Flag"), welches die normale Anzeige der Uhrzeit mit dem Terminbild überlagert, aktiv (0x01) oder inaktiv (0x00) sind. Weiterhin werden 6 Byte für die eigentlichen Termindaten benötigt. Diese Termindaten setzten sich zusammen aus dem Monat, dem Tag des Monats, dem Wochentag und der Uhrzeit, bestehend aus Stunde, Minute und Sekunde. Eine genauere Aufschlüsselung der jeweiligen Wertebereiche kann man Tabelle 6 entnehmen. Weiterhin ist zu beachten, dass die Sekunden bei den Termindaten immer auf „0" gesetzt sind und zu dem aktuellen Zeitpunkt keine weitere Verwendung finden, allerdings für zukünftige Entwicklungen weiterhin im Kontext bestehen bleiben.

## Genutzte Klassen beider Demoanwendungen

**ComPortDevice**

Die Kommunikation mit dem ID100 ist in der ComPortDevice-Klasse gekapselt, welche Methoden zum Senden von Befehlen anhand des erläuterten Datenrahmens und zum Lesen von Antworten zur Verfügung stellt. Auch werden Methoden zur Behandlung von Steuerzeichen angeboten, damit mögliche Steuerzeichen in den Daten ersetzt bzw. in einer Antwort wiederhergestellt werden.

**CRC16**

In der CRC16-Klasse wird die Prüfsumme für die Datenübertragung berechnet. Hier stehen neben der CRC16_init-Methode zum Initialisieren die Shift-Methoden für die Verarbeitung der Prüfsumme zur Verfügung (siehe gesonderten Abschnitt zur Prüfsummenberechnung).

**ID100LedMatrix**

Diese Klasse repräsentiert die LED-Matrix des ID100. Sie stellt ein zweidimensionales Array aus booleschen Werten zur Verfügung, welches mit der Anzahl der LEDs in der Breite und in der Höhe des Geräts initialisiert wurde. Ein einzelner Wert zeigt den Status der jeweiligen LED gemäß ihrer Koordinate. Die GetBytes()-Methode errechnet die benötigten 26 Byte aus der aktuellen LED-Matrix, die für die Anzeige eines einzelnen Bildes auf dem ID100 benötigt werden. Der Aufbau der Berechnung orientiert sich an dem in der Tabelle 5 angegebenen Schema.

**Nutzung des ID100 als Anzeige für Spiele**

Aufgrund der Matrix von 17 x 12 LEDs lässt sich das ID100 auch als Anzeige für Spiele mit einer geringen

Tabelle 3

## Befehlsübersicht

| Befehl | Parameter | Parameter-länge | Beschreibung |
|---|---|---|---|
| v (0x76) | | 0 | liest die Firmwareversion aus |
| T (0x54) | Zeitinformation je 1 Byte in der Reihenfolge: Tag, Monat, Jahr (zweistellig), Wochentag (Sonntag = 0), Stunde, Minute, Sekunde, Sommer- (0x01) bzw. Winterzeit (0x00) | 8 | setzt die Systemzeit |
| t (0x74) | | 0 | liest die Systemzeit aus |
| A (0x41) | | 0 | aktiviert den Normalmodus |
| a (0x61) | | 0 | aktiviert den Vorschaumodus |
| B (0x42) | Helligkeitswert (0x08, 0x0C, 0x13, 0x1D, 0x2D, 0x46, 0x6B, 0xA6, 0xFF) | 1 | setzt die Standardhelligkeit |
| b (0x62) | | 0 | liest die Standardhelligkeit |
| C (0x43) | Kalibrierwert | 4 | setzt den RTC-Kalibrierwert (Real-Time Clock) |
| c (0x63) | | 0 | liest den Zeitpunkt der letzten Kalibrierung und die aktuelle Zeit des ID100 aus |
| D (0x44) | LED-Matrix, Aufbau siehe Tabelle 5 | 26 | sendet das Vorschaubild |
| E (0x45) | 2 Byte Nummer des ersten Flash-Speichers | 2 | löscht 16 Flashpages (1 Sektor) |
| F (0x46) | 2 Byte Nummer des Flash-Speichers, 156 Byte (6 LED-Matrizen) oder 34 Byte (LED-Matrix und Termindaten, Aufbau siehe Abschnitt „Termine übertragen“) | 158 oder 36 | schreibt die Flashpage mit der Konfiguration der Uhr oder der Termine |
| f (0x66) | 2 Byte Nummer des Flash-Speichers | 2 | liest den Inhalt des Flash-Speichers aus |
| R (0x52) | 20 Termine zu je 8 Byte Termindaten, Aufbau siehe Abschnitt „Termine übertragen“ | 160 | setzte die Termindaten im EEPROM |
| r (0x72) | | 0 | liest die Termindaten aus dem EEPROM |
| S (0x53) | je 1 Byte Einschaltstunde, Einschaltminute, Ausschalt-stunde, Ausschaltminute, Aktivierungsstatus (0x01 aktiv, 0x00 inaktiv) | 5 | setzt die Stand-by-Zeit |
| s (0x73) | | 0 | liest die Stand by-Zeit |
| X (0x58) | | 0 | Werksreset starten |
| ! (0x21) | | 0 | Bootloader starten |

Tabelle 4

## Antwortübersicht

| Befehl | Parameter | Parameter-länge | Beschreibung |
|---|---|---|---|
| v (0x76) | je 2 Byte pro Versionsstelle | 6 | liefert die Firmwareversion zurück |
| T (0x54) | | 0 | Systemzeit wurde gesetzt |
| t (0x74) | Zeitinformation je 1 Byte in der Reihenfolge: Tag, Monat, Jahr (zweistellig), Wochentag (Sonntag = 0), Stunde, Minute, Sekunde, Sommer- (0x01) bzw. Winterzeit (0x00) | 8 | Systemzeit des ID100 |
| A (0x41) | | 0 | Normalmodus wurde gestartet |
| a (0x61) | | 0 | Vorschaumodus wurde gestartet |
| B (0x42) | | 0 | Standardhelligkeit wurde gesetzt |
| b (0x62) | Helligkeitswert (0x08, 0x0C, 0x13, 0x1D, 0x2D, 0x46, 0x6B, 0xA6, 0xFF) | 1 | aktuelle Standardhelligkeit |
| C (0x43) | | 0 | RTC-Kalibrierwert (Real-Time Clock) wurde gesetzt |
| c (0x63) | je 1 Byte Tag, Monat, Jahr (zweistellig), Stunde, Minute, Sekunde der aktuellen Gerätezeit<br>je 1 Byte Tag, Monat, Jahr (zweistellig), Stunde, Minute, Sekunde der letzten Synchronisierungszeit | 12 | Zeitpunkt der letzten Kalibrierung und die aktuelle Zeit des ID100 |
| D (0x44) | | 0 | Vorschaubild wurde gesendet |
| E (0x45) | 2 Byte Nummer des ersten Flash-Speichers | 2 | Flashpages wurden gelöscht |
| F (0x46) | 2 Byte Nummer des Flash-Speichers | 2 | Flashpage wurde geschrieben |
| f (0x66) | 2 Byte Nummer des Flash-Speichers, 256 Byte Inhalt der Flashpage, davon 156 Byte (6 LED-Matrizen) oder 34 Byte (LED-Matrix und Termindaten, Aufbau siehe Abschnitt „Termine übertragen“) | 258 | Nummer und Inhalt einer Flashpage |
| R (0x52) | | 0 | Termindaten wurden im EEPROM gesetzt |
| r (0x72) | 20 Termine zu je 8 Byte Termindaten, Aufbau siehe Abschnitt „Termine übertragen“ | 160 | Termindaten aus dem EEPROM |
| S (0x53) | | 0 | Stand-by-Zeit wurde gesetzt |
| s (0x73) | je 1 Byte Einschaltstunde, Einschaltminute, Ausschalt-stunde, Ausschaltminute, Aktivierungsstatus (0x01 aktiv, 0x00 inaktiv) | 5 | Stand-by-Zeit des ID100 |
| X (0x58) | | 0 | Werksreset wurde gestartet |
| ! (0x21) | | 0 | Bootloader wurde gestartet |

Auflösung nutzen. Denkbare Spiele wären beispielsweise Pong oder Space Invaders, für dieses Beispiel wurde Snake realisiert. In diesem muss sich eine Schlange durch ein Spielfeld bewegen und nach Futter suchen. Wenn sie fündig geworden ist, verspeist sie das Futter und wird letztlich eine Einheit länger. Danach wird ein neues Futterstück auf das Spielfeld gesetzt und der Ablauf beginnt von Neuem. Das Spiel endet, wenn die Schlange sich selbst berührt, was mit fortlaufender Spieldauer aufgrund der stetig wachsenden Schlange immer schwieriger zu vermeiden ist.

Für die Oberfläche (Bild 1) wurde Windows Forms genutzt. Bevor man eine Partie starten kann, müssen ein COM-Port, das gewünschte Spiel selbst und der bevorzugte Schwierigkeitsgrad vom Spieler ausgewählt werden. Der COM-Port muss dem des angeschlossenen ID100 entsprechen, damit das Gerät entsprechend als Anzeige genutzt werden kann. Als Spieltyp steht aktuell nur Snake zur Verfügung. Unter einer Schaltfläche zum Starten oder Beenden des Spiels werden die Steuerung und der aktuelle Spielstand dargestellt. Die Steuerung erfolgt mittels der Tasten W, A, S und D, die entsprechend ihrer Anordnung auf der Tastatur für die Bewegung der Schlange nach oben, links, unten und rechts stehen. Der Spielstand (Bild 2) errechnet sich aus der Differenz der Länge der Schlange und 3 (Länge der Schlange zu Beginn des Spiels), welche mit 10 multipliziert wird (beispielsweise bei einer Länge von 8: (8 – 3) * 10 = 50 Punkte).


*Bild 1: Die Oberfläche der Demoanwendung „Spiele“*

Tabelle 5

**Aufschlüsselung der Bytes als Matrix**

| X-Koordinate | Y-Koordinate | Byteindex | Bitindex |
|---|---|---|---|
| 0 | 0 | 0 | 7 |
| 0 | 1 | 0 | 6 |
| 0 | 2 | 0 | 5 |
| 0 | 3 | 0 | 4 |
| 0 | 4 | 0 | 3 |
| 0 | 5 | 0 | 2 |
| 0 | 6 | 0 | 1 |
| 0 | 7 | 0 | 0 |
| 0 | 8 | 1 | 7 |
| 0 | 9 | 1 | 6 |
| 0 | 10 | 1 | 5 |
| 0 | 11 | 1 | 4 |
| 1 | 0 | 1 | 3 |
| 1 | 1 | 1 | 2 |
| 1 | 2 | 1 | 1 |
| 1 | 3 | 1 | 0 |
| … | … | … | … |
| 16 | 8 | 25 | 7 |
| 16 | 9 | 25 | 6 |
| 16 | 10 | 25 | 5 |
| 16 | 11 | 25 | 4 |
| leer | | 25 | 3 |
| leer | | 25 | 2 |
| leer | | 25 | 1 |
| leer | | 25 | 0 |

Zu Beginn des Spiels befindet sich die Schlange in der oberen Hälfte der LED-Matrix und läuft nach unten, ein Futterstück ist bereits automatisch auf dem Spielfeld platziert. Die Schlange wird durch eine Liste von Point-Objekten, die jeweils primär eine X- und Y-Koordinate enthalten, repräsentiert. Zu Beginn des Spiels sind dies die Punkte (8|0), (8|1) und (8|2). Ebenso wird das Futter durch ein Point-Objekt dargestellt, wobei hier beim Setzen der Koordinate darauf geachtet werden muss, dass ein entsprechender Abstand zur Schlange eingehalten wird.

Das Spiel an sich ist in der Klasse „SnakeGame“ implementiert. Sie enthält eine Methode Tick(), die bei einem Aufruf das Spiel dazu veranlasst, den nächsten Schritt zu errechnen und die Oberfläche entsprechend zu aktualisieren. Dies bedeutet beispielsweise für den ersten Aufruf nach dem Start des Spiels, dass sich die Schlange um eine LED nach unten bewegt hat, was durch eine Inkrementierung aller Y-Werte der einzelnen Point-Objekte der Schlange erreicht wird. Daraufhin muss geprüft werden, ob die Koordinaten des Kopfes der Schlange mit denen des Futters übereinstimmen. Wenn dies zutrifft, wird die Schlange um eine Einheit erweitert bzw. der Liste von Point-Objekten ein weiterer Point hinzugefügt und ein neues Futterstück wird zufällig auf dem Spielfeld verteilt. Auch muss überprüft werden, ob eine Koordinate innerhalb der Schlange mehrfach vorkommt. Wenn dies zutrifft, hat sich die Schlange in ihren eigenen Schwanz gebissen und das Spiel ist für den Spieler beendet. Dies äußert sich in einem Setzen aller LEDs des ID100. Die Schlange kann sich über die Grenzen der LED-Matrix bewegen, wobei sie auf der jeweils gegenüberliegenden Seite wieder herauskommt. Wenn sie beispielsweise das Spielfeld nach rechts verlässt, bewegt sie sich von links aus wieder hinein.

Die Oberfläche ruft die Tick-Methode anhand eines Intervalls, das sich aus dem zuvor gewählten Schwierigkeitsgrad bestimmt, zyklisch auf. Je geringer das Intervall für den Aufruf ist, desto schwieriger ist die Schlange aufgrund ihrer steigenden Geschwindigkeit zu steuern. Nach jedem berechneten Tick werden mithilfe der ID100LedMatrix-Klasse die aktuellen Koordinaten der Schlange und des Futters auf dem ID100 dargestellt.

Die Klasse „SnakeGame“ ist eine abgeleitete Klasse von der abstrakten Klasse „Game“. Dies sorgt dafür, dass sich die Anwendung leicht um weitere Spiele erweitern lässt, da ein neues Spiel in Form einer weiteren abgeleiteten Klasse von „Game“ nur eine Tick-Methode implementieren muss, damit es zyklisch berechnet und auf dem ID100 dargestellt werden kann.

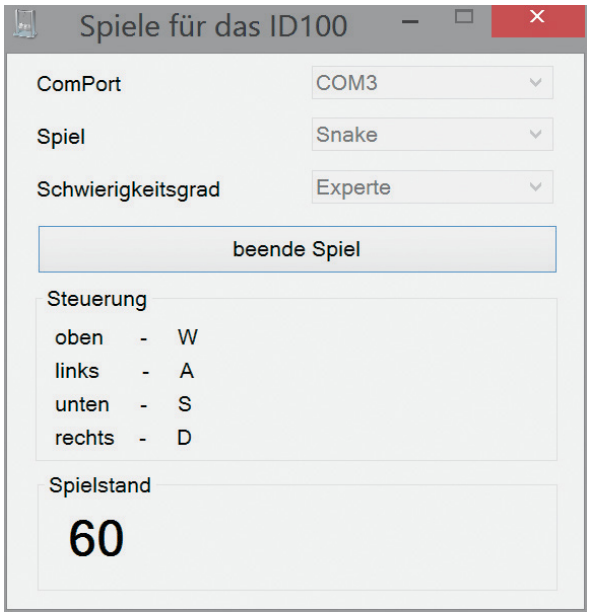


*Bild 2: Die Spielstandsanzeige in der Demoanwendung „Spiele“*

## Spektrumanalysator zur Visualisierung von Musik

Für diese Demoanwendung wurde die freie Audio-Bibliothek „NAudio“ [5] genutzt. Diese Bibliothek liegt allerdings nur für das .NET-Framework 4.0 oder höher vor. Unter Verwendung der Oberflächentechnologie WPF (Windows Presentation Foundation) wurde die Oberfläche des Spektrumanalysators erstellt.

### Aufbau der Oberfläche

In der Oberfläche (Bild 3) wird zunächst der COM-Port ausgewählt, über den das ID100 angeschlossen ist, und mit der entsprechenden Schaltfläche eine Verbindung zu diesem hergestellt. Über weitere Schaltflächen lassen sich Dateien zu der Wiedergabeliste hinzufügen bzw. wieder entfernen. Diese Tracklist befindet sich am unteren Rand der Oberfläche, welche sich automatisch mit der Menge des Inhalts anpasst. Direkt über der Wiedergabeliste wird der Pfad zu der aktuell wiedergegebenen Datei angezeigt. Zusätzlich gibt es noch Schaltflächen zur Steuerung der Wiedergabe mit den Funktionen Start/Pause, Stopp, vorheriger Song und nächster Song. Als ergänzende Information werden noch die aktuelle Laufzeit sowie die Gesamtlaufzeit der aktuell wiedergegebenen Datei angezeigt.

### Abspielen von Musik und Anzeige des Spektrums

Das Abspielen erfolgt in der AudioPlayback-Klasse, welche die Funktionen der Audio-Bibliothek nutzt, in der das eigentliche Abspielen stattfindet. In dieser wird auch die SampleAggregator-Klasse verwendet, welche das Spektrum berechnet. Immer wenn die im aktuell abgetasteten Signal der wiedergegebenen Audio-Datei vorkommenden Frequenzen berechnet wurden, erfolgt das Auslösen eines entsprechenden Events. Der dafür benötigte Algorithmus wird von der Audio-Bibliothek bereitgestellt. Auch wird die Wiedergabesteuerung mittels Events gesteuert, beispielsweise eine Anweisung zur Unterbrechung der Wiedergabe.

Da das ID100 mit seinen 17 LEDs in der Breite und 12 LEDs in der Höhe weniger Informationen zum Spektrum darstellen kann, als das Programm errechnet, muss eine entsprechende Umrechnung auf die Maße erfolgen. In der Höhe äußert sich dies in einer Umrechnung der Stärke des jeweiligen Frequenzbereichs in einen Prozentwert, wobei die Höhe des ID100 von 12 LEDs 100 % entspricht. Die verfügbaren Frequenzbereiche werden gleichmäßig auf die 17 LEDs in der Breite verteilt. Somit ergeben sich 17 Säulen zur Darstellung des Spektrums mit ihren jeweiligen Ausprägungen in Abhängigkeit zur aktuellen Wiedergabe. Die Umrechnung selbst erfolgt in der TriggerID100WithSpectrum-Klasse. Auch werden die errechneten Daten auf dem ID100 dargestellt. Hierfür nutzen wir wie bei der vorigen Demoanwendung die ID100LedMatrix-Klasse, um die gewünschten LEDs zu setzen und die 26 Byte zu errechnen, die für eine Darstellung der Matrix auf dem ID100 vonnöten sind. ELV

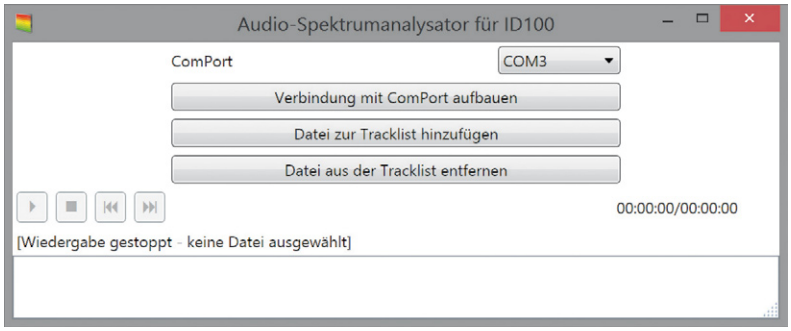


*Bild 3: Die Oberfläche für die Demoanwendung „Audio-Spektrumanalysator“*

### Weitere Infos:


[1] Entwicklungsumgebung Download: www.microsoft.com/visualstudio/eng/downloads#d-2010-express

[2] ID100 Demosoftware: Webcode #1231

[3] Silabs VCP Treiber: www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

[4] Wikipedia-Artikel zur Prüfsumme: http://de.wikipedia.org/wiki/Zyklische_Redundanzprüfung

[5] NAudio Klassenbibliothek: http://naudio.codeplex.com


Tabelle 6

**Aufschlüsselung der Terminparameter**

| Beschreibung | Parameter | | Parameterlänge |
|---|---|---|---|
| Active Flag | aktiv 0x01 | inaktiv 0x00 | 1 |
| Overlay Flag | aktiv 0x01 | inaktiv 0x00 | 1 |
| Monat | 0 für jeden Monat des Jahres | für spezifischen Monat von 1 für Januar bis 12 für Dezember | 1 |
| Tag | 0 für jeden Tag des Monats | für spezifischen Tag 1 bis 31 | 1 |
| Wochentag | 7 für jeden Tag der Woche | 0 für Sonntag | 1 |
| | | 1 für Montag | |
| | | 2 für Dienstag | |
| | | 3 für Mittwoch | |
| | | 4 für Donnerstag | |
| | | 5 für Freitag | |
| | | 6 für Samstag | |
| Stunde | 0 bis 23 | | 1 |
| Minute | 0 bis 59 | | 1 |
| Sekunde | 0 bis 59, 0 als Standardwert | | 1 |