Einfache Integration USB-Softwarebibliothek für Mikrocontroller

Embedded USB leicht gemacht!

Die Entwicklung eines Embedded Systems mit USB-Schnittstelle wird durch die Verwendung der USB-Softwarebibliothek für die 16- und 32-bit Mikrocontroller von Fujitsu erheblich vereinfacht.

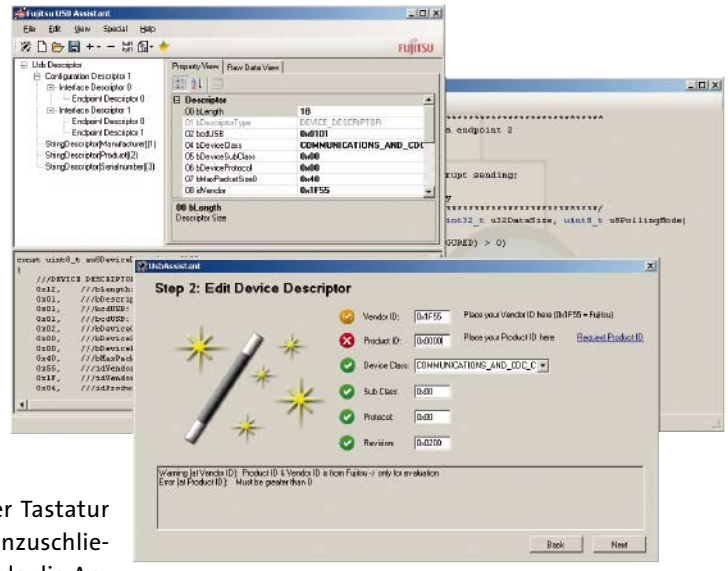
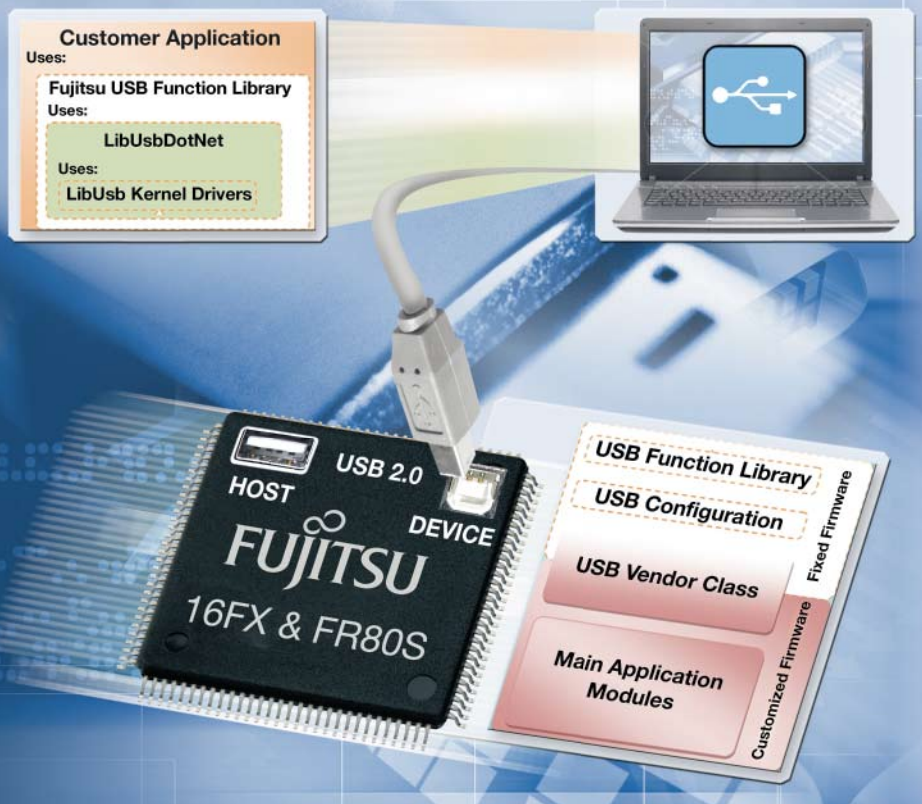


Bild 1: Konfigurationsprogramm des USB Assistant (alle Bilder und Grafiken Fujitsu Semiconductor Europe)

Der Universal Serial Bus (USB), wird immer mehr zu einem Standardinterface für Embedded Systeme aller Art. Neben der Möglichkeit, portable Speichermedien wie USB-Sticks oder Bediengeräte wie eine Maus oder Tastatur an das Gerät anzuschließen, stellt gerade die Anbindung an einen PC eine der häufigsten Anwendungsfälle dar. Genutzt wird diese Verbindung zu unterschiedlichen Zwecken: zum Update der Gerätefirmware, zum Austausch von Daten aller Art oder auch einfach nur zum Setzen von gewissen Betriebs- und Kontrollparametern. Auch als Diagnoseschnittstelle im Servicefall bietet sich USB an. USB verdrängt hierbei die in der Vergangenheit etablierten seriellen Schnittstellen wie RS232, PS/2 oder auch das parallele Centronics/IEEE-1284. Dies basiert zum einen auf der Tatsache, dass immer größere Datenmengen übertragen werden müssen, zum anderen aber schlichtweg darauf, dass moderne PC-Systeme und Notebooks nicht mehr mit den entsprechenden Schnittstellen ausgerüstet sind, aber meist eine große Zahl

AUTOR

Manuel Schreiner ist als Applikationsingenieur und Markus Vogel im Marketing der Business Unit Embedded Solutions von Fujitsu Semiconductor Europe beschäftigt.



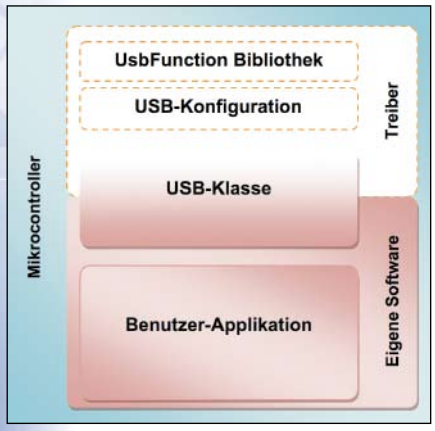


Bild 2: Schichtenmodell auf dem Mikrocontroller.

```
void main(void)
{
  uint8_t* pu8Buffer;
  uint32_t u32DataSize;
  UsbClass_Init();          //Init buffers and event handlers
  UsbFunction_Initialize(TRUE); //TRUE: Autoconnect, FALSE: Manual connect
  for(;;)
  {
    /* returns 0, if no data was received. Otherwise the received data length */
    u32DataSize = UsbClass_GetReceivedDataEndpoint1(&pu8Buffer);
    if (u32DataSize > 0)
    {
      UsbClass_SendDataVia2(pu8Buffer, u32DataSize, USB_SENDING_MODE_POLLED);
    }
  }
}
```

Bild 3: Software-Beispiel Loopback-Gerät.

an USB-Ports zur Verfügung stellen. Weitere Schlagworte zum Vormarsch des USBs sind die hohe Fehlerkorrektur, Plug & Play-Fähigkeit sowie die Möglichkeit, Geräte über den USB mit einer stabilisierten Spannung zu versorgen.

USB-relevanten Teile der Firmware

Die Entwicklung der USB-relevanten Teile der Firmware erfordert vom Programmierer üblicherweise ein tiefgreifendes Verständnis der USB-Spezifikation und des auf dem Chip implementierten Hardware-Makros. Werden insbesondere bei der Anwendung des Embedded Systems als USB Function, also der Slave-Anbindung an einen PC, keine im Betriebssystem hinterlegten USB-Klassen verwendet, ist auch die Entwicklung eines eigenen Treibers notwendig. Dies setzt natürlich auch entsprechende Kenntnisse in der Programmierung des PC-Betriebssystems voraus. Da dieser Aufwand von den meisten Entwicklern gescheut wird, greifen sie meist auf bereits implementierte Standardklassen wie Human Interface Devices (HID) – Mäuse, Tastaturen und ähnliche Geräte – oder Mass Storage Devices (MSD) – Massenspeicher aller Art wie USB-Sticks und externe Festplatten – zurück.

Des Weiteren haben sich auf dem Markt verschiedene RS232-USB-Konverterbausteine durchgesetzt, die letztendlich zwar eine USB-Verbindung zum PC ermöglichen, aber auf der Embedded-Seite lediglich die Daten in das bekannte RS232-Format umsetzen. Auf PC-Seite wird mit Hilfe der Communication Device Class (CDC) ein COM-Port virtuell nachgebildet. Der benötigte Treiber ist in den Windows und Linux-basierten Betriebssystemen implementiert. Diese Lösung ist aber nur für einige Fälle anwendbar und natürlich mit der Performance des seriellen RS232-Protokolls limitiert.

Um aber die volle Datenbandbreite und die verschiedenen Übertragungsmodi des USB wie Interrupt-, Bulk- oder Isochroner Transfer passend für die jeweilige Applikation voll nutzen zu können, kommt man an der Implementierung einer vollwertigen USB-Klasse nicht vorbei. Fujitsu stellt hierbei für seine 16-bit-Mikrocontroller (Serie MB96330) sowie die 32-bit-Mikrocontroller (Serien MB91660/665) Projekttemplates zur Firmware-Entwicklung zur Verfügung. Diese Templates beinhalten eine USB-Funktionsbibliothek, welche die Grundfunktionalitäten der USB-Function wie Anmelde- und Abmeldeprozeduren am Bus (auch Enumeration genannt) und das Aufsetzen von sogenannten Endpoints für die Datenübertragung übernimmt. Aufgesetzt auf diesen Funktionen ist die beispielhafte Realisierung einer kundenspezifischen USB Vendor Class beinhaltet. Die Realisierung von typischen Standardklassen wie HID oder MSD wird anhand von Demo-Beispielen gezeigt. Für einen einfachen Start mit USB sind entsprechende Konfigurationswerkzeuge nötig, die helfen, ein USB-Gerät zu erstellen. Dazu gehört die eigentliche USB-Konfiguration (USB-Deskriptoren) und das Kommunikationsmodul, welches die Schnittstelle zwischen Applikation und Treiber herstellt (USB-Klasse). Mit dem ‚Fujitsu USB Assistant', welcher alle für USB notwendigen Dateien als C-Code erzeugt, stellt Fujitsu solch ein Werkzeug zur Verfügung. ▶

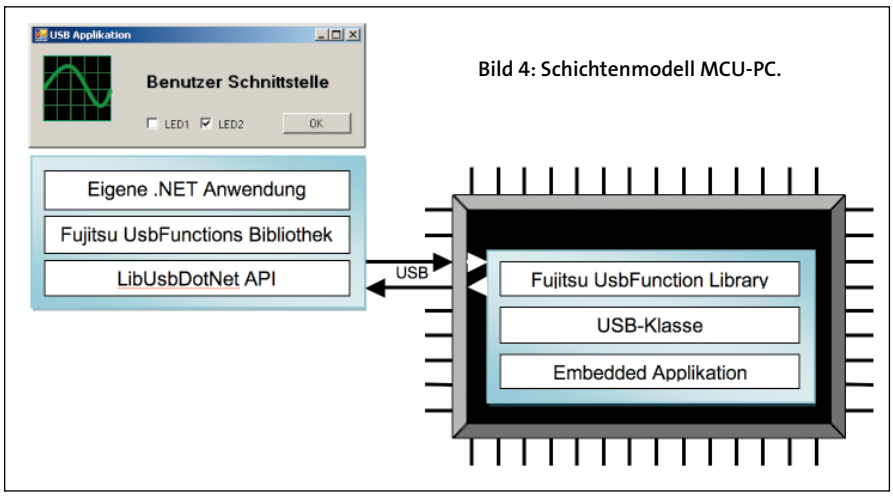


Bild 4: Schichtenmodell MCU-PC.

Um die USB-Klasse zwischen Treiber und Benutzerapplikation transparenter zu gestalten, erzeugt der Konfigurator als USB-Klasse eine Vorlage für Kommunikationsroutinen, die auf die entsprechende Konfiguration angepasst wird. Die USB-Klasse behandelt vor allem Datentransfers, die als interruptgesteuerte Blocktransfers ein- und ausgehen und stellt die Daten der Benutzerapplikation zur Verfügung. Letztendlich wird in der USB-Klasse das Übertragungsprotokoll umgesetzt, das per API von der Benutzerapplikation gesteuert werden kann. Die Embedded-Applikation kann dadurch einfach auf USB zugreifen.

Beispiel: Einfaches Loopback-Gerät

Das Programmbeispiel in **Bild 3** demonstriert ein einfaches Loopback-Gerät, das per Endpoint 1 Daten empfängt und per Endpoint 2 die Daten wieder versendet. Nach dem Aufsetzen des Pointers zum Sende-/Empfangspuffer sowie einer Integer-Variable für die Datenlänge wird die USB-Klasse initialisiert. Hierzu werden die entsprechenden Informationen aus den zuvor mit Hilfe des Konfigurators erzeugten USB-Deskriptoren verwendet und die notwendigen Event Handler, Datenpuffer und Request Handler aufgesetzt. Endpoint 1 wird in diesem Fall als Empfänger (HOST OUT) und Endpoint 2 als Sender (HOST IN) initialisiert. Anschließend wird die USB Function Bibliothek gestartet. Der Parameter TRUE erlaubt hierbei die Verwendung des Auto Connect Features der Library. In der nun folgenden Endlosschleife werden die empfangenen Daten auf Endpoint 1 in den Empfangspuffer geschrieben. Die Auslesefunktion von Endpoint 1 gibt hierbei auch die Länge der empfangenen Daten als Rückgabewert zurück, welcher in der Variablen u32DataSize gespeichert wird. Dieser Wert wird beim Aufrufen der Sendefunktion auf Endpoint 2, neben der Angabe des Sendepuffers, als Parameter übergeben. Ebenso wird als weiterer Parameter der Übertragungsmode übergeben. In diesem Fall soll das Senden per Polling überwacht werden. Weitere Möglichkeiten wäre das Verwenden von MCU-internen Interrupts oder die Verwendung des internen DMAs, um die Daten an das USB-Makro zu liefern.

Und auf der PC-Seite?

Die meisten Probleme haben Entwickler jedoch oft computerseitig, da dort natürlich auch passende Treiber notwendig sind. Hier wird deshalb oft auf Standardklassen wie ein virtueller COM-Port oder Human Interface Devices zurückgegriffen, um eine Eigenentwicklung entsprechender Treiber zu umgehen. Dabei ist man allerdings auf gewisse Geschwindigkeiten und vorgefertigte Übertragungsprotokolle begrenzt. Schwieriger wird es bei eigenen Protokollen und Klassen, jedoch werden damit höhere Geschwindigkeiten gewährleistet. Hierfür hat Fujitsu eine Lösung mit dem Open-Source Projekt LibUsbDotNet gefunden. Dieses Paket beinhaltet die LibUsb Treiber sowie ein Konfigurationswerkzeug zum Anpassen eben dieser. Darauf aufsetzend bietet Fujitsu eine .NET Wrapper Bibliothek an, mit der auch PC-seitig auf das eigene USB-Gerät zugegriffen werden kann. Die Wrapper Bibliothek sorgt vor allem für eine stabile Implementierung von LibUsbDotNet in eigene Projekte und dient des Weiteren für eine strikte Trennung zwischen GPL-geschütztem Code und eigenem Code. Besonders viel Wert wurde auch auf Hot-Plug gelegt, da USB nicht mehr als statische Verbindung angesehen werden kann. Außerdem unterstützt die Wrapper Bibliothek auch diverse .NET Programmierumgebungen wie Visual Studio, Borland und LabVIEW. Neue Ansätze von LibUsbDotNet öffnen USB .NET Anwendungen wiederum der Linux und Mac OS Welt. Somit ist keine Portierung zwischen verschiedenen Systemen mehr notwendig.

Fujitsu USB Assistant

Das Konfigurationswerkzeug ‚Fujitsu USB Assistant' unterstützt neben der in diesem Artikel besprochenen USB Function Bibliothek auch die Anwendung des Mikrocontrollers als USB-Host, also als Master des Systems, an den z. B. eine Maus oder ein Speicherstick angeschlossen werden kann. Auch hier generiert das Werkzeug die entsprechend benötigten Konfigurationsdateien in C-Code, welche die ebenfalls in den USB-Templates vorhandene USB-Host-Softwarebibliothek steuern. Darauf aufbauend gibt es mit dem ‚USB Supervisor' auch Unterstützung zur Implementierung verschiedener Standardklassen.

Fujitsu bietet weiterhin auf seiner Webseite viele weitere Demo-Beispiele und Application Notes zum Thema USB an. Mit dem von der Firma Thesycon entwickelten USB-Treiber für Host und Function steht für die Fujitsu Mikrocontroller weitere, teilweise auch kostenfreie, Third-Party Software zur Verfügung. Um interessierten Kunden einen einfachen Einstieg in das Thema Embedded USB zu ermöglichen, bietet Fujitsu auch zwei Tagesseminare zum Thema USB Host bzw. Function an. (jj)

infoDIRECT 598ei1010

▶ Link zu **Fujitsu Semiconductor Europe**

www.elektronik-industrie.de