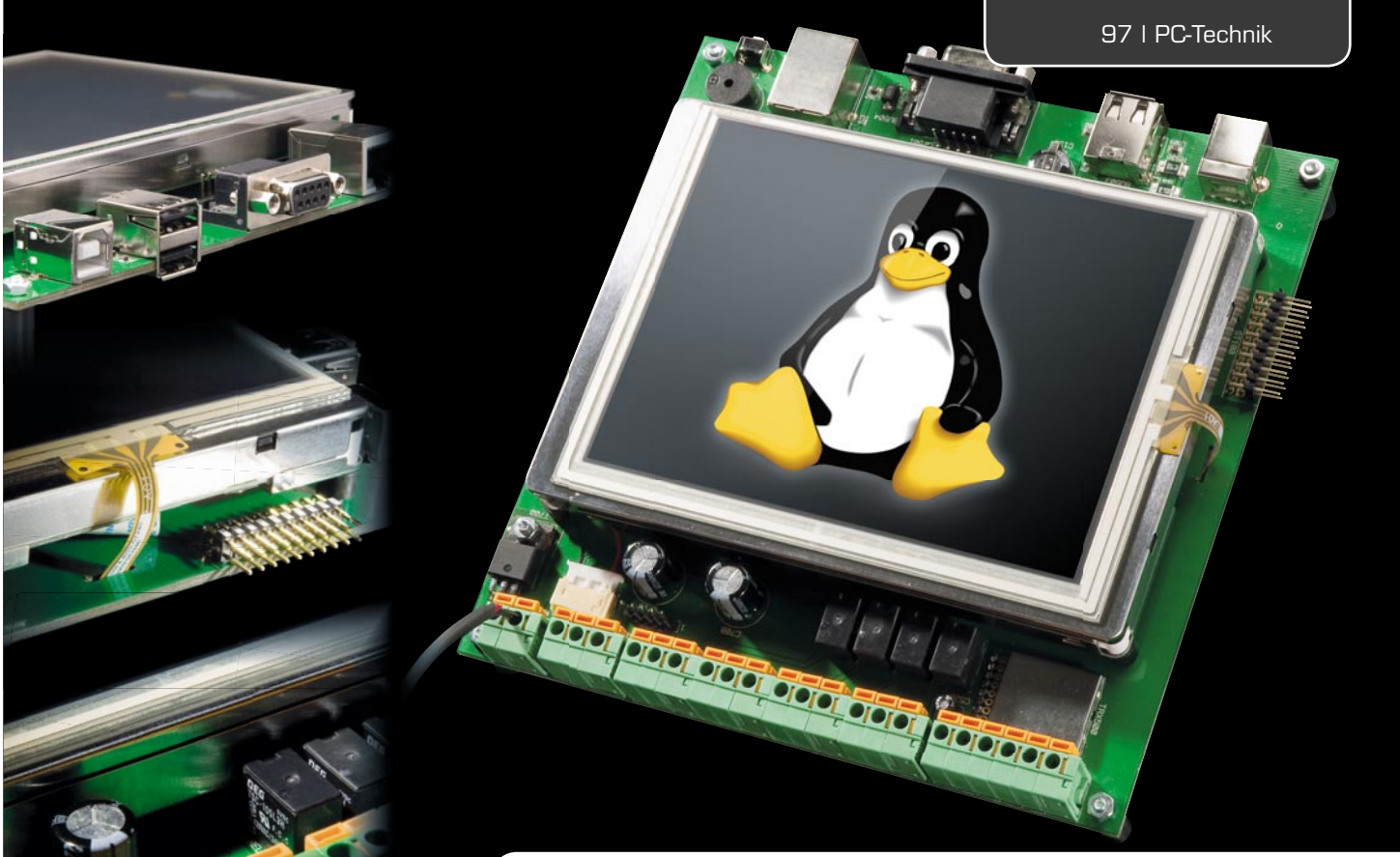

Universal-embedded Linux-Plattform

Linux-Control-Unit LCU 1, Teil 3

Mit der Linux-Control-Unit stellen wir eine komplette Hard- und Software-Plattform mit zahlreichen unterschiedlichen Schnittstellen, Eingängen, Ausgängen und Speichermöglichkeiten vor. Derjenige, der sich mit dem Thema Embedded Linux-Systeme beschäftigen möchte, wird damit in die Lage versetzt, schnell und preiswert zu einer eigenen, kompletten Lösung, vornehmlich im Steuerungsbereich, zu kommen. Im dritten und letzten Teil zeigen wir neben der Installation des Linux-Kernels die der Entwicklungsumgebung und stellen kurz die mit dem kompletten Software-Paket gelieferte Demo-Applikation vor.

Die Entwicklungsumgebung

Gleich zu Beginn sei gesagt, dass die Kompilierung und Nutzung des Quellcodes der Entwicklungsumgebung Linux-Kenntnisse erfordert. Für Linux-Einsteiger seien die zahlreichen Einführungen und die zu den jeweiligen Distributionen vorhandenen Einsteigerforen, die auf den Seiten der Distributionen verlinkt sind, empfohlen.

Die Nutzung der mitgelieferten, im Quellcode vorliegenden Software und deren Komponenten erfordert zunächst die Installation eines Linux-Systems, wie z. B. Ubuntu oder Debian, auf dem Entwicklungsrechner. Zusätzlich sind über die Programmpaketverwaltung des jeweiligen Systems noch der C|C++-Compiler und die Bibliothek zur Entwicklung grafischer Anwendungen „Qt“ zu installieren. Ein frisch installiertes Linux fordert beim Kompilieren des Quellcodes häufig noch die Installation weiterer Paketdateien, die in den meisten Fällen ebenfalls über die Programmpaketverwaltung installierbar sind.

Danach kopiert man den Quellcode (lcu1-oss-src-0.5.tar.gz, Bezeichnung kann je nach aktuell ausgelieferter Version abweichen) der LCU-Software in ein per „mkdir“-Befehl zu erzeugendes Verzeichnis im Linuxsystem, z. B. /home/user/lcu1.

Nach dem Wechsel in dieses Verzeichnis:

```
cd /home/user/lcu1
```

wird der Quellcode hiermit:

```
tar xzf lcu1-oss-src-0.5.tar.gz
```

entpackt. Bild 6 zeigt die Reihe der entpackten Dateien bzw. Ordner.

Danach wird die Software mit den folgenden Schritten kompiliert und erstellt:

```
make -C buildroot
```

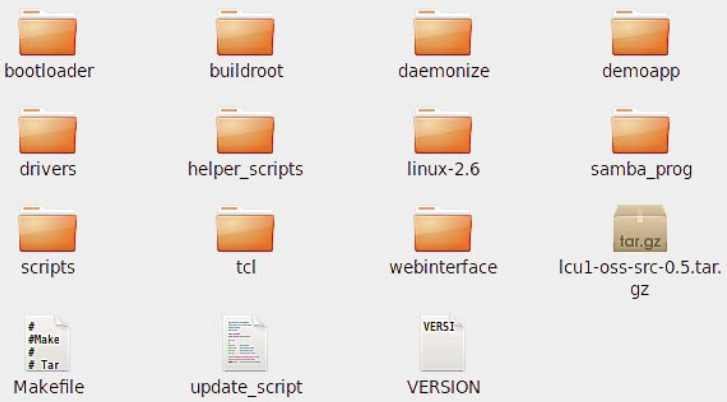

Bild 6: Die nach dem Entpacken sichtbaren Dateien und Ordner des Quellcode-Files

Damit wird „buildroot“, ein Basissystem mit Toolchain zur Cross-Kompilierung, erstellt. Dieser Vorgang kann mehrere Stunden dauern, ist jedoch nur einmalig nötig. Die mit buildroot erstellte Toolchain wird dann zum Erzeugen der eigentlichen Firmware verwendet. Dieser Erstellungsprozess wird gestartet mit:

```
make
```

Schließlich erzeugt man die installierbare Firmware-Version mit:

```
make install
```

Dieser Befehl bringt die durch „make“ erstellten kompilierten Dateien in eine Form, die zum Installieren auf der Hardware geeignet ist. Es werden folgende Dateien erzeugt:

- *bootloader\at91bootstrap\binaries\at91sam9261ek-dataflashboot-2.10.bin*

Dies ist der First-Level-Bootloader, er wird ins Dataflash der LCU 1 installiert. Buildroot enthält zwar auch bereits eine Version von „at91bootstrap“, unsere Version wurde jedoch so modifiziert, dass der Watchdog des AT91SAM9261 weiter nutzbar ist.

- *bootloader\u-boot\u-boot.bin*

Dies ist der bereits vorgestellte Bootloader „U-Boot“, auch er wird ins Dataflash der LCU 1 installiert. Er wurde hier um den Support für die LCU erweitert.

- *uImage*

Die (komprimierte) Imagedatei des Linux-Kernels. Sie wird ebenfalls ins Dataflash installiert und später von „U-Boot“ geladen und gestartet. Der Linux-Kernel (2.6) wurde um den Support für die LCU 1 erweitert. Er enthält gegenüber dem Standard-Kernel noch Bugfixes für: Treiber, Touch-Controller ADS7843, Atmel-SPI-Controller und USB-Gadget-Ethernet.

- *jffs2.image*

Unter dieser Image-Datei verbirgt sich das Root-Dateisystem für den NAND-Flash, sie wird in das NAND-Flash installiert.

- *lcu1-firmware-0.5-img*

Dieses Firmware-Image enthält die Dateien uImage und jffs2.image sowie ein Shell-Skript, welches ein Firmware-Update durchführt. Dieses Image ist für die Installation der Firmware über das Web-Interface geeignet.

Für die Erstinstallation ist die von Atmel bereitgestellte Software „SAM-BA“ nötig. Diese gibt es für Windows und für Linux. Da die Linux-Version aufgrund des verwendeten Gerätetreibers für USB-zu-RS232-Umsetzer schwierig zu verwenden ist, insbesondere, wenn sich noch weitere USB-zu-RS232-Umsetzer im System befinden, empfehlen wir die Verwendung der Windows-Version. Bei Verwendung der Linux-Version ist die Vorgehensweise aber prinzipiell dieselbe.

Nach der Installation von SAM-BA, enthalten im Paket „AT91-ISP v1.13“, muss die Software durch Entpacken von „sam-ba 2.9_patch_lcu.zip“ (im Ordner „samba-prog“) nach C:\ATMEL Corporation\AT91-ISP v1.13 um den Support für die LCU 1 erweitert werden.

SAM-BA verwendet den im ROM des AT91SAM9261 enthaltenen Bootloader. Damit dieser gestartet wird, darf keine andere Firmware (z. B. im Dataflash) vorhanden sein. Daher muss für den Updatevorgang über SAM-BA beim Einschalten der LCU 1 der Jumper J 500 gesetzt sein. Das Dataflash wird hierdurch deaktiviert. Vor dem eigentlichen Zugriff mit SAM-BA muss der Jumper wieder entfernt werden.

Die LCU 1 wird per USB an den PC angeschlossen. Nach dem ersten Anschließen der LCU 1 an den PC erfolgt automatisch die Windows-Treiberinstallation. Danach ist im Ordner „samba-prog“ die Batchdatei „program-all.bat“ zu starten. In diesem Ordner befinden sich noch weitere Batchdateien, die nur einzelne Teile des Speichers programmieren.

Die Firmware lässt sich bei bereits installierter Firmware auch über den Web-Browser updaten mit:

```
http://<lcu-IP>/fwupdate.cgi
```

Weitere Programmbestandteile

Die oben mit Buildroot erstellte Toolchain kann auch zum Erstellen eigener Programme verwendet werden. Dazu ist das Compiler-|Toolchain-Verzeichnis (buildroot/output/staging/usr/bin) in den Suchpfad aufzunehmen. Es handelt sich dabei um eine „arm-linux“-Toolchain. Der Compiler „gcc“ wird daher als „arm-linux-gcc“ aufgerufen.

Die Konfiguration des Linux-Kernels kann über eine Qt-basierte grafische Oberfläche angepasst werden. Dazu kann der folgende Befehl verwendet werden:

- `make kernel_xconfig`

Ebenso lässt sich „buildroot“ per Qt konfigurieren:

- `make -C buildroot xconfig`

Die Demo-Applikation

Die mitgelieferte Demo-Applikation ermöglicht ein erstes Kennenlernen der grafischen Schnittstelle und der Hardwareausstattung der LCU 1. Gestartet wird

diese an der Kommandozeile der LCU (siehe 2. Teil) durch Eingabe von

```
· demoapp
```

Auf dem Touchscreen wird ein komplettes Anzeige- und Bedienfeld dargestellt (Bild 7).

In der oberen Reihe „Inputs“ erfolgt die Anzeige des Zustands der Digital-Eingänge. Ist ein Eingang aktiv, wechselt er die Farbe auf Grün.

In der mittleren Reihe „Outputs“ lassen sich die Relais und der Signalgeber über berührungsempfindliche Bedienfelder schalten. Auch hier wird der aktivierte Zustand grün angezeigt.

In der Reihe „ADCs“ schließlich zeigen zwei Felder den Spannungswert der beiden ADC-Eingänge sowohl numerisch als auch durch einen mit dem Wert wachsenden Farbbalken an.

In den Auszügen aus dem Demoprogramm in Bild 8 sind als Beispiel die Programmteile dargestellt, die für die beschriebenen Teile der Demo-Applikation federführend sind. Da das Programm offenliegt, kann man sich sofort an eigene Experimente machen bzw. Teile des Programms in eigene Applikationen übernehmen.

Der Quellcode findet sich in der Entwicklungsumgebung unter „demoapp“. Er wird beim Kompilieren des Gesamt-Quellcodes standardmäßig mit kompiliert und installiert, aber nicht automatisch gestartet.

Soll dies erfolgen, kann man in der Entwicklungsumgebung im Makefile „scripts/Makefile“ das Doppelkreuz in der letzten Zeile:

```
#$(ROMFSINST) -p755 /etc/init.d/S99demoapp
```

löschen. Dann wird die Startdatei „S99demoapp“ nach „/etc/init.d/S99demoapp“ auf dem Zielsystem installiert und beim Systemstart ausgeführt.

Wer als erfahrener Linux-Nutzer nur allein die Demo-Applikation kompilieren möchte, kann dies mit:

```
# make subdir_demoapp
```

tun.

Die eigene (modifizierte) Version des Programms lässt sich z. B. mit Hilfe des kostenlosen Open-Source-FTP-Programms „Filezilla“ oder unter Linux mit dem scp-Befehl auf das Zielsystem übertragen:

```
scp demoapp/demoapp root@10.101.81.51/
usr/local/demoapp
```

Diese eigene Version wird dann unter Angabe des kompletten Pfades mit

```
# /usr/local/demoapp
```

gestartet.

Soweit die Einführung in die Installation der Software und die Konfiguration der Hardware der LCU 1. Unter Nutzung der mitgelieferten Quellcode-Software ist es dem Linux-Programmierer somit möglich, die LCU 1 für das eigene Projekt zu konfigurieren, zu programmieren und fernzusteuern. ELV

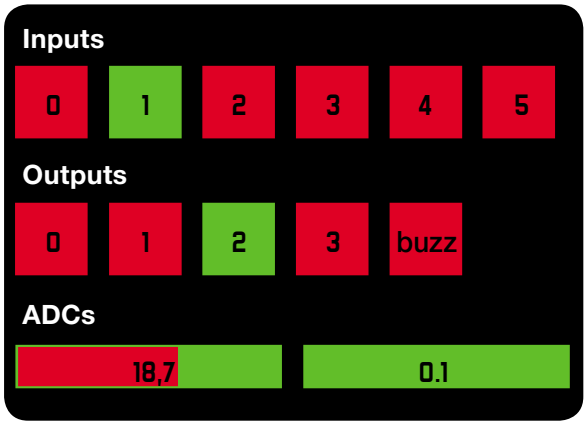

Bild 7: Das Demoprogramm in Aktion: Digital-Eingang 1 ist aktiviert, ebenso der Relais-Ausgang 2, und am Analog-Eingang 1 werden 18,7 V gemessen.

```
/* Anzahl der Digitaleingänge */
#define NUM_INPUTS 6

/* Anzahl der Digitalausgänge */
#define NUM_OUTPUTS 5

/* Anzahl der Analogeingänge */
#define NUM_ADCS 2

/* Größe der Quadrate für die Digitaleingänge und -ausgänge in Pixeln */
#define RECT_SIZE 48
/* Rand um die Quadrate für die Digitaleingänge und -ausgänge in Pixeln */
#define RECT_MARGIN 6
/* Y-Position der Oberkante der Quadrate für die Digitalausgänge in Pixeln */
#define OUTPUT_RECTS_TOP 120
/* Y-Position der Oberkante der Quadrate für die Digitaleingänge in Pixeln */
#define INPUT_RECTS_TOP 30

/* Y-Position der Oberkante der Rechtecke für die Analogeingänge in Pixeln */
#define ADC_TOP 210
/* Breite der Rechtecke für die Analogeingänge in Pixeln */
#define ADC_RECT_WIDTH 150
/* Höhe der Rechtecke für die Analogeingänge in Pixeln */
#define ADC_RECT_HEIGHT 25
/* Rand um die Rechtecke für die Analogeingänge in Pixeln */
#define ADC_RECT_MARGIN 6
/* Innerer Rand um den Wertebalken für die Analogeingänge in Pixeln */
#define ADC_RECT_INTERNAL_MARGIN 2
/* Maximaler ADC-Wert der Analogeingänge. Einheit mV */
#define ADC_MAX_VALUE 38360

/* Bitflags für die Verarbeitung von Touchscreen-Ereignissen */
/* Ein "Klick" wird erkannt, wenn nach einem Tastendruckereignis noch eine X-Koordinate und eine Y-Koordinate empfangen
wurden */
#define EVT_FLAG_X_RECEIVED 1 /* X-Koordinate empfangen */
#define EVT_FLAG_Y_RECEIVED 2 /* Y-Koordinate empfangen */
#define EVT_FLAG_PENDOWN_WAIT 4 /* Tastendruckereignis empfangen */
#define EVT_FLAG_PENDOWN 8 /* Tastendruckereignis und Koordinaten empfangen */
#define EVT_FLAG_COMPLETE (EVT_FLAG_X_RECEIVED|EVT_FLAG_Y_RECEIVED|EVT_FLAG_PENDOWN_WAIT)

/* Dateideskriptoren für die Gerätedateien der Digitaleingänge */
static int _inputFds[NUM_INPUTS];
/* Dateideskriptoren für die Gerätedateien der Digitalausgänge */
static int _outputFds[NUM_OUTPUTS];
/* Dateideskriptoren für die Gerätedateien der Analogeingänge */
static int _adcFds[NUM_ADCS];

/* Aktuelle Zustände der Digitaleingänge */
static bool _inputStates[NUM_INPUTS];
/* Aktuelle Zustände der Digitalausgänge */
static bool _outputStates[NUM_OUTPUTS];
/* Aktuelle Werte der Analogeingänge */
static int _adcValues[NUM_ADCS];

/*
 * Ausgabe der Grafikdarstellung
 */
static void renderScreen()
{
        _primarySurface->SetFont( _primarySurface, _font );

    /* Schriftfarbe setzen */
        _primarySurface->SetColor( _primarySurface, 0xff, 0xff, 0xff, 0xff );

    /* Überschriftstexte ausgeben */
        _primarySurface->DrawString( _primarySurface, "Inputs", -1, 10, INPUT_RECTS_TOP - TEXT_HEIGHT - RECT_MARGIN,
DSTF_TOPLEFT );
        _primarySurface->DrawString( _primarySurface, "Outputs", -1, 10, OUTPUT_RECTS_TOP - TEXT_HEIGHT - RECT_MARGIN,
DSTF_TOPLEFT );
        _primarySurface->DrawString( _primarySurface, "ADCs", -1, 10, ADC_TOP - TEXT_HEIGHT - ADC_RECT_MARGIN,
DSTF_TOPLEFT );

    /* Rechtecke zeichen für die Digitaleingänge */
    for( int i=0; i<NUM_INPUTS; i++ )
    {
        /* Farbauswahl, grün für aktiv, rot für nicht aktiv */
        if( _inputStates[i] )
        {
                _primarySurface->SetColor( _primarySurface, 0x00, 0xff, 0x00, 0xff );
        }else{
                _primarySurface->SetColor( _primarySurface, 0xff, 0x00, 0x00, 0xff );
        }
        /* Zeichnen eines gefüllten Rechtecks */
        _primarySurface->FillRectangle( _primarySurface, RECT_MARGIN + i*(RECT_SIZE + 2*RECT_MARGIN), INPUT_RECTS_TOP +
RECT_MARGIN, RECT_SIZE, RECT_SIZE );


    /* Aktuelle Werte der Analogeingänge lesen */
    readAdcValues();

    /* Balken zeichen für die Analogeingänge */
    for( int i=0; i<NUM_ADCS; i++ )
    {
        /* Grünes Rechteck als Hintergrund */
        _primarySurface->SetColor( _primarySurface, 0x00, 0xff, 0x00, 0xff );
        _primarySurface->FillRectangle( _primarySurface, ADC_RECT_MARGIN + i*(ADC_RECT_WIDTH + 2*ADC_RECT_MARGIN), ADC_TOP
+ ADC_RECT_MARGIN, ADC_RECT_WIDTH, ADC_RECT_HEIGHT );

        /* Darin rotes Rechteck mit Breite proportional zum Spannungswert */
        _primarySurface->SetColor( _primarySurface, 0xff, 0x00, 0x00, 0xff );
        int width = (ADC_RECT_WIDTH - 2*ADC_RECT_INTERNAL_MARGIN) * _adcValues[i] / ADC_MAX_VALUE;
        _primarySurface->FillRectangle( _primarySurface, ADC_RECT_MARGIN + ADC_RECT_INTERNAL_MARGIN + i*(ADC_RECT_WIDTH +
2*ADC_RECT_MARGIN), ADC_TOP + ADC_RECT_MARGIN + ADC_RECT_INTERNAL_MARGIN, width, ADC_RECT_HEIGHT-
2*ADC_RECT_INTERNAL_MARGIN );

        /* Gemessene Spannung als Text mittig im Balkendiagramm ausgeben */
        char buffer[8];
        sprintf(buffer, "%.1f", double(_adcValues[i])/1000.);
        _primarySurface->SetColor( _primarySurface, 0x00, 0x00, 0x00, 0xff );
        _primarySurface->DrawString( _primarySurface, buffer, -1, ADC_RECT_MARGIN + ADC_RECT_WIDTH/2 + i*(ADC_RECT_WIDTH +
2*ADC_RECT_MARGIN), ADC_TOP + ADC_RECT_MARGIN + ADC_RECT_HEIGHT/2 - TEXT_HEIGHT/2, DSTF_TOPCENTER );
    }
```

Bild 8: Auszüge aus dem Demoprogramm „demoapp“