Heimautomatisierung für Hardwarebastler

von Hans Müller

Wer gerne und viel eigene Hardware entwickelt, hat in der Regel das Problem, dass er nicht auch noch die zusätzlich benötigte Software entwickeln kann oder möchte. Da es ohne diese aber oftmals nicht geht, sind intelligente und einfach einzusetzende Lösungen gefragt. Eine dieser Lösungen ist die OpenAPC-Software [1], welche hier vorgestellt werden soll.

Auf einschlägigen Messen werden in schöner Regelmäßigkeit immer wieder faszinierende Heimautomatisierungslösungen vorgestellt, welche gar Sagenhaftes zu leisten vermögen: Sie verwalten teilweise das gesamte Haus inklusive Beleuchtungs-, Lüftungs- und Heizungssteuerung sowie Kühlschrank und Speisekammer inklusive automatisiertem Einkauf. All diese Lösungen haben zumeist aber auch das gleiche Problem: Sie sind unglaublich teuer, damit für den Normalanwender nicht erschwinglich und aufgrund dessen auch kaum mehr als Studien, die zudem niemals wirklich den Markt erreichen. Darüber hinaus sind all diese umfassenden Lösungen zumeist auch nur auf Hausbesitzer zugeschnitten. Wohnungsmieter, welche nicht nach Belieben die Wände aufreißen und Kabel verlegen können, bleiben mehrheitlich außen vor.

Einen Lichtblick gab es vor einiger Zeit mit der Ankündigung der digitalSTROM-Technologie. Eine Schweizer Firma wollte kleine Schalteinheiten in der Größe einer Lüsterklemme verkaufen, welche sich wie eine solche in bestehende Elektroinstallationen einbauen lassen, um dann über ein Signal von einem Steuergerät im Sicherungskasten geschaltet zu werden. Was sich in der Ankündigung wie die ultimative Lösung für alle Heimautomatisierer angehört hat, war in der Realität leider eine Enttäuschung. Schlussendlich wurde verkündet, dass eben diese „Schalt-Lüsterklemmen“ nicht einzeln, sondern nur zusammen mit fertigen Geräten verkauft werden. Ein weiteres K.O.-Argument war der angekündigte Preis für die verbliebenen Möglichkeiten mit nicht unter 1000 Euro pro Raum [2].

Selbst gebaut und maßgeschneidert

So bleibt dem ambitionierten Bastler also nach wie vor nichts weiter übrig, als selber Lösungen zu finden und diese in eigener, selbst entwickelter Hardware zu realisieren. Während diese Hardwarekomponenten immer sehr individuell und speziell auf den gewünschten Einsatzzweck zugeschnitten sein werden, ist es für die Ansteuerungsseite wünschenswert, eine fertige Software einzusetzen, welche die Kontrolle übernimmt und auch eine schöne, leicht verständliche Benutzeroberfläche darstellt. Da dieser Teil eines Projektes einen durchaus nennenswerten Teil des Gesamtaufwandes ausmachen kann, sind einfach zu handhabende Lösungen gefragt, die es im Optimalfall noch nicht mal erfordern, selbst zum Compiler zu greifen und eigenen Code für die Steuerungsplattform zu schreiben.

Aber auch hier wird es wieder eng, wenn die Vorgaben „möglichst preiswert“ und „Linux-Unterstützung“ lauten. Anders als in anderen Fällen ist hier aber der Preis bereits das Argument, welches die Suche kompliziert macht. Bemüht man die eigene Lieblingssuchmaschine, um nach Steuerungs- bzw. Visualisierungssoftware zu suchen, so wird man von Treffern geradezu überrollt. Leider sind die gefundenen Lösungen aber allesamt teuer und / oder an eine (ebenso teure) herstellerspezifische Hardware gebunden und / oder nur für Windows zu bekommen und / oder für den eigentlichen Einsatzzweck viel zu komplex und verbrauchen damit viel zu viele Ressourcen. Die Suche auf Plattformen für Freie Linux-Software verbessert die Situation ein wenig: Freshmeat [3] und SourceForge [4] wissen von vielen spannenden Projekten aus den Bereichen Steuerung und Automatisierung. Viele davon sind wiederum Lösungen, welche in andere Software eingebunden werden möchten (da sie nur Teilaspekte oder bestimmte Kommunikationsstandards beherrschen) oder sie erfordern explizite Programmierkenntnisse bzw. kommen gar ganz ohne Oberfläche daher, da sie im Hintergrund laufen sollen.

Klickibunti mit Anspruch

OpenAPC steht für „Open Advanced Process Control“ und bedeutet so viel wie „offene, fortgeschrittene Prozesskontrolle“. Die Software selbst ist scheinbar für den industriellen Einsatz ge-

dacht, kann dank der direkten Unterstützung von z. B. einfachen Parallelports oder der LCDproc-Displayansteuerungssoftware [5] aber auch problemlos für Heimautomatisierungsprojekte verwendet werden. Die hier vorgestellte Version 1.0 (Codename „Cocoa Beach“) besitzt deswegen auch viele Funktionen, wie beispielsweise eine Benutzer- und Rechteverwaltung, welche für heimische Steuerungsaufgaben eher irrelevant sind. Was letztendlich den Ausschlag gegeben hat, eben diese Softwarelösung zu wählen, sind vier Argumente:

- Die Software ist frei und kostenlos.
- Es ist ein GUI-Editor enthalten, mit dem sich Oberflächen schnell und ohne Programmieraufwand erstellen lassen.
- Benutzer- und Logikelemente lassen sich mittels einer Schaltplansymbol-ähnlichen Syntax miteinander verknüpfen, sodass auch hier nichts programmiert werden muss.
- Die Software ist als Binärpaket für alle wichtigen Linux-Systeme verfügbar (und sogar für ARM-Plattformen zu haben).

Bei weiterer Vertiefung in das Paket finden sich dann doch – optional – noch Möglichkeiten, selbst geschriebene Software zu integrieren, um z. B. eigene Hardware anzubinden oder komplexere Steuerungsaufgaben zu übernehmen. Beispielprogramme und Dokumentationen dazu hält ein eigenes SDK bereit.

Doch der Reihe nach: Auf der Downloadseite des Projektes [6] finden sich die verschiedenen Softwarepakete. Neben einer Variante für Windows gibt es dort auch fertige Pakete im RPM-Format für Fedora, Red Hat und Co. sowie im DEB-Format für Ubuntu, Debian und kompatible. Dies gilt jedoch nur für das Gesamtpaket, welches quasi die Entwicklungsumgebung darstellt.

Darüber hinaus sind auf der Downloadseite auch noch als „Runtime“ titulierte Pakete als ZIP- bzw. TAR.BZ2-Archiv zu haben. Diese sind für das Zielsystem gedacht und müssen dort manuell installiert werden. Das ist umständlich, hat aber den Vorteil, dass nur die Komponenten auf das eigentlichen Zielsystem kopiert werden müssen, die wirklich benötigt werden. Das spart Platz, was bei knappen Ressourcen interessant sein kann. Insbesondere bei typischer Embedded-Hardware sind diese ja meist sehr knapp bemessen. Um diesen doch etwas mühsamen Kopiervorgang etwas zu erleichtern, enthalten diese Runtime-Pakete die Daten aber bereits in einer Struktur, welche das Dateisystem und die Lage der einzelnen Teile in selbigem reflektiert.

Doch zurück zum Entwicklungspaket: Nach der Installation des zur Distribution und zur Plattform passenden Paketes (es werden gesonderte Varianten für x86_64, x86 und ARM angeboten) findet sich im Startmenü ein neuer Eintrag „*OpenAPC Editor*“. Dieser versteckt sich entweder unter „*Sonstiges*“ oder ist unter „*Elektronik/Automation*“ zu finden. Nach dem Start dieses Editors präsentiert sich eine sehr aufgeräumte Oberfläche mit nur wenigen Menüs und Toolbar-Buttons. Die eigentlich interessanten Funktionen sind allesamt in Kontextmenüs versteckt. So ergibt sich der erste Aha-Effekt, wenn man im HMI-Editor die rechte Maustaste betätigt: Es öffnet sich ein Popup-Menü, welches Zugriff auf verschiedenste GUI-Elemente bietet, die wiederum in logischen Untergruppen wie z. B. „*Steuerung*“, „*Statisch*“ oder „*Anzeige*“ angeordnet sind.

Dieser HMI-Editor selbst ist sozusagen der Oberflächendesigner, was der Name auch schon selbst aussagt: „HMI“ steht für „Human Machine Interface“, also Mensch-Maschine-Schnittstelle und stellt eine Verbindung dar, welche auf der einen Seite „Mensch“ spricht (also leicht verständliche Bedienelemente anbietet) und diese in „Maschine“ übersetzt (also Kommandos und Datenflüsse generiert, welche vom Computer verstanden und verarbeitet werden können).

Oberflächendesign

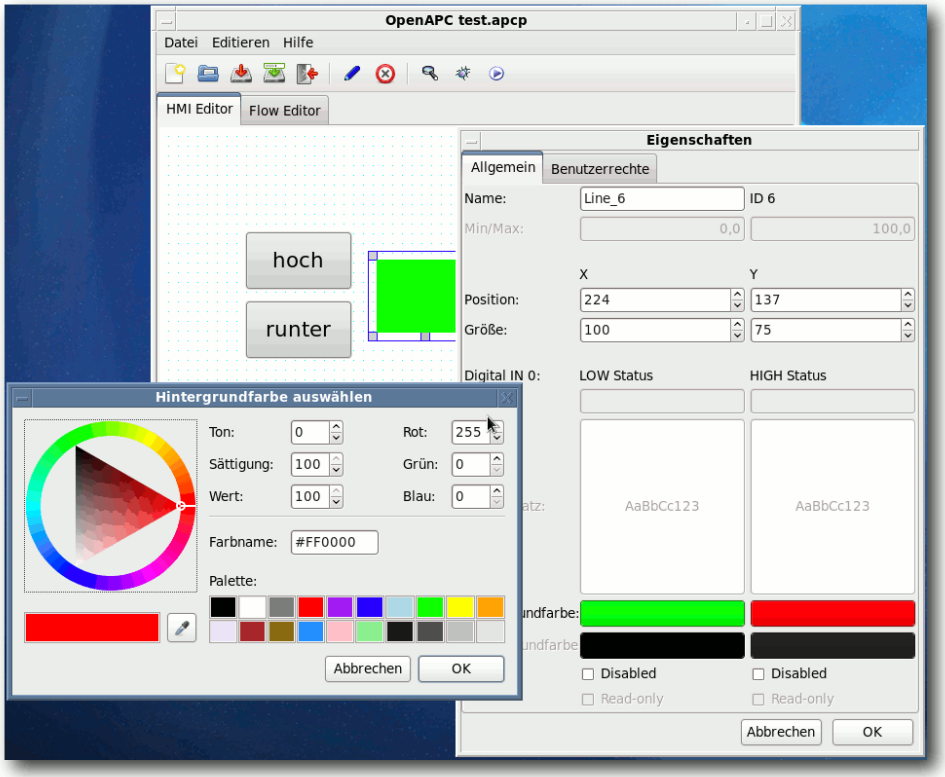
Im Folgenden soll nun die Benutzung der Software für eigene Steuerungsaufgaben anhand einer Rollladensteuerung exemplarisch beschrieben werden. Die Steuerung für dieses Beispiel ist recht simpel aufgebaut: es gibt einen Button für „*hoch*“, einen für „*runter*“ sowie eine kleine Anzeige, wenn die Rollladen gerade gefahren werden. Die Verbindung zur Hardware soll in diesem Beispiel über die Parallelschnittstelle geschehen, da es für diese bereits fertige Relaiskarten gibt, welche zum Schalten der Hardware benutzt werden können.

In einem ersten Schritt soll das Layout der Benutzerschnittstelle erstellt werden. Dazu wird mit Hil-

fe des bereits erwähnten Kontextmenüs im HMI-Editor mittels des Menüpunktes „*Steuerung → Umschalter*“ zuerst ein einzelner Togglebutton hinzugefügt. Ist dieser Button für die weitere Bearbeitung ausgewählt, so wird das mittels eines blauen Rahmens signalisiert. Dieser besitzt an den Ecken und an den Seiten hervorgehobene Greifpunkte, welche sich dazu benutzen lassen, die Größe des Elements zu verändern. Der Greifpunkt rechts unten behält bei diesem Skaliervorgang übrigens das Seitenverhältnis bei. Greift man das Element innerhalb dieser blauen Umrandung aber außerhalb der markierten Greifpunkte, so kann die Position verändert werden. So können die beiden Buttons schon mal grob angeordnet werden.

Eine exaktere Positionierung und weiterführende Konfigurationsmöglichkeiten finden sich in den Objekteigenschaften. Den entsprechenden Dialog erreicht man durch einen Doppelklick auf den entsprechenden Button. Leider funktioniert dieser Doppelklick nicht immer, insbesondere bei GUI-Elementen, die als externes Plug-in implementiert sind, scheint dies der Fall zu sein. Hier kann aber entweder der Editier-Button in der Toolbar (gekennzeichnet durch ein Stift-Symbol) oder aber wieder das Kontextmenü auf dem Pfad „*Ausgewähltes Steuerungselement → Editieren*“ bemüht werden.

In diesem Dialog finden sich verschiedene Karteikartenreiter, mit denen sich diverse Eigenschaften verändern lassen. Im Ersten können beispielsweise die exakte Größe und Position angegeben sowie dem Element ein eindeutiger Name zugewiesen werden. Dieser Name wird später noch benötigt; der erste Button soll hier den Namen **RL1Hoch** erhalten. Die untere Hälfte dieses



Der HMI-Editor mit geöffnetem Objektdialog und dem GNOME Color Chooser.

Karteikartenreiters ist zweigeteilt und lässt unterschiedliche Definitionen für die Zustände HIGH und LOW zu. Jedes GUI-Element kann extern angesteuert zwei unterschiedliche Zustände mit unterschiedlichen Layouts haben. Diese Layouts werden hier festgelegt – für dieses Beispiel wird das gleich noch bei der Anzeige des Betriebszustandes verwendet. Für den aktuell bearbeiteten Button genügt es jedoch, diesem eine Größe zuzuweisen (damit auch eine Bedienung per Touchscreen möglich ist, wird die Höhe auf 60 Punkte festgelegt) und eine größere Schrift für eine gefälligere Darstellung auszuwählen. Sowohl die Schrift als auch der neue Text „hoch“ des Buttons sind in diesem Fall in der Spalte LOW vorzunehmen.

Die beiden Karteikartenreiter „*Datenerfassung*“ und „*Benutzerrechte*“ bleiben unverändert, diese sind für Loggingaufgaben und die Rechteverwaltung vorgesehen, welche in diesem Beispiel nicht benötigt werden. Interessanter wird es allerdings ganz auf der rechten Seite im Karteikartenreiter „*Gegenseitiger Ausschluss*“: Hier befindet sich eine Combobox mit gleichem Namen, in dieser wird der einzige Eintrag „*Neuen Ausschluss anlegen*“ ausgewählt. Dieser Ausschluss wird im Zusammenhang mit dem Button zum Herunterfahren der Rollladen benötigt. Die Funktion stellt sicher, dass immer

nur einer der beiden Buttons betätigt ist und deswegen keine widersprüchlichen Steuerungssignale abgegeben werden. Der neue Anschluss erhält den Namen **RL1**. Damit sind die Einstellungen an diesem Button beendet.

Für den zweiten Button wird es schon einfacher: Das Kontextmenü hält eine Funktion „*Ausgewähltes Element → Duplizieren*“ bereit, welches eine Kopie dieses Buttons anlegt. In dieser Kopie müssen lediglich noch der eindeutige Name neu gesetzt (**RL1Runter**), der Text von „hoch“ auf „runter“ geändert und der gegenseitige Ausschluss auf das jetzt bereits vorhandene Element **RL1** gesetzt werden. Damit sind alle nötigen Steuerungselemente bereits vorhanden, es fehlt nur noch ein Anzeigeelement.

Hier gibt es viele verschiedene Möglichkeiten: Man kann mit Symbolen arbeiten, Bilder anzeigen und anderes mehr. Der künstlerischen Gestaltungsfreiheit sind also kaum Grenzen gesetzt. Da ich persönlich allerdings wenig künstlerische Begabung habe, soll in diesem Beispiel ein statisches Element für diesen Zweck „missbraucht“ werden: Eine Linie, die als farbige Box eingesetzt wird. Das Vorgehen ist bei der Nutzung der anderen gestalterischen Möglichkeiten prinzipiell ähnlich, sodass die Verwendung der Linie zur Erklärung der Funktion keine Einschränkung darstellen sollte. Das gesuchte GUI-Element selbst findet sich im Kontextmenü unter „*Statisch → Linie*“ und soll zukünftig zwei Zustände signalisieren: Rot, wenn der Rollladenantrieb angesteuert wird, und grün, wenn er deaktiviert ist und deswegen nicht läuft. Die entsprechenden Einstellungen werden wieder im ersten Karteikartenreiter der Objekteinstellungen vorgenommen. Zuerst wird wieder ein eindeutiger Name vergeben, hier **RL1Display**. Dann wird die Größe auf 100x75 Pixel gesetzt, sodass auch wirklich ein Rechteck dargestellt wird. Die beiden Farben werden über die Hintergrundfarbe festgelegt, ein Klick auf den entsprechenden Button öffnet einen Auswahldialog. Hier nun soll in der Spalte LOW grün gewählt werden, in HIGH rot. Damit ist das GUI-Design bereits beendet; der nächste Schritt ist die Implementierung der Ansteuerungslogik.

Form und Funktion

Dafür kommt nun der „*Flow Editor*“ zum Einsatz, welcher sich hinter dem zweiten Karteikartenreiter im Hauptfenster verbirgt. Während der „*HMI Editor*“ rein für die Erstellung des visuellen Layouts zuständig ist, wird hier festgelegt, was im Hintergrund geschehen soll und wie die einzelnen Elemente voneinander abhängig sind. Das passiert mittels einer Art Blockschaltbild.

Ein Rechtsklick in diesem Editor öffnet wiederum ein umfangreiches Kontextmenü, bei dem vorerst der erste Eintrag „*Steuerungselement anlegen*“ wichtig ist. Dieser öffnet seinerseits ein Fenster, in dem die zuvor angelegten GUI-Elemente mit ihren eindeutigen Namen aufgelistet sind. Ein Doppelklick auf **RL1Hoch** fügt nun dem Blockschaltbild ein erstes Element hinzu.

Dieses Symbol kann man mit der linken Maustaste verschieben. Es besitzt diverse Ein- und Ausgänge. Die Eingänge eines solchen Elementes sind dabei immer an der Oberseite angeordnet und zeigen in das Element hinein, die Ausgänge sind an der Unterseite angeordnet und zeigen heraus.

An dieser Stelle vielleicht ein wenig Theorie. Es gibt ein paar einfache Regeln für ein solches Blockschaltbild:

- Es existieren vier Datentypen DIGI (digital, LOW/HIGH), NUM (numerisch, jeder mögliche Zahlenwert), CHAR (Texte) und BIN (Binärdaten) die im Elementsymbol farblich unterschieden werden.
- Verbindungen zwischen Aus- und Eingängen können immer nur bei gleichen Datentypen hergestellt werden.
- Verbindungen sind keine elektrischen Leitungen, die ständig irgendein Signal übertragen bzw. irgendeinen Pegel haben, sondern logische Verknüpfungen, welche ereignisorientiert ein Datenpaket übertragen.

Mit diesen Regeln und dem Wissen um die Funktion der Ein- und Ausgänge von Elementen lässt sich nun schon alles Mögliche recht einfach realisieren.

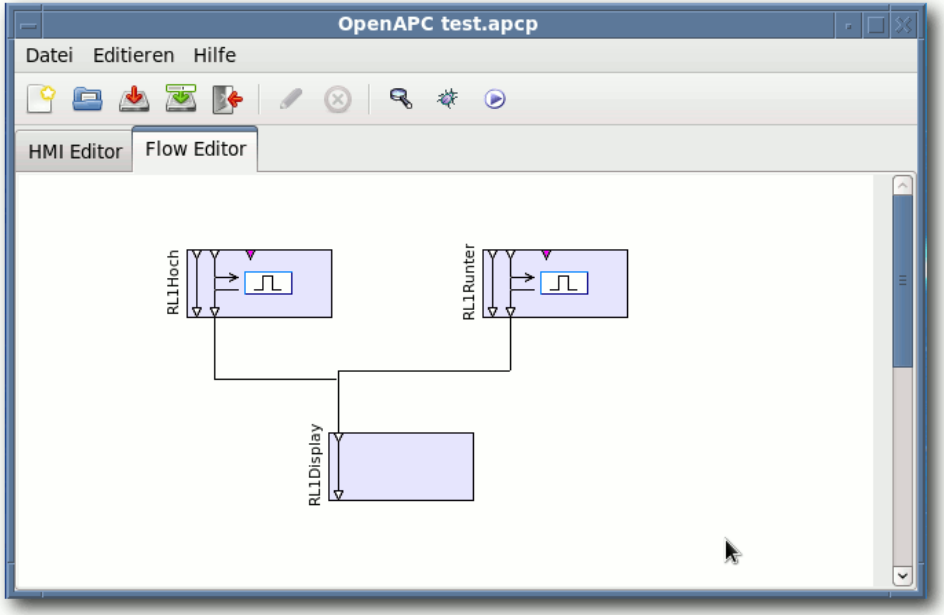
Mit einigen weiteren Klicks können auch die anderen beiden Elemente aus der „*Steuerungselemente*“-Liste (deren Inhalt ja den GUI-Elementen im HMI-Editor entspricht) in den Flow-Editor eingefügt werden. Hier fällt nun eines auf: Alle Elemente besitzen ganz links einen Digitaleingang

D0 (Symbolfarbe weiß), der direkt auf den Ausgang D0 durchgeschleift wird. Dieser Eingang bestimmt den Status des GUI-Elements, Änderungen des Eingangssignals bewirken direkt eine Änderung des Layouts entsprechend den Voreinstellungen für LOW und HIGH in den Objekteigenschaften. Die Buttons besitzen noch einige Ein- und Ausgänge mehr; an dieser Stelle ist aber vorerst nur der Ausgang D1 interessant: Hier wird der Status des jeweiligen Buttons übertragen, wenn er vom Nutzer betätigt wird. Damit steht fest, wie das erste Blockschaltbild aussehen muss: Die beiden Ausgänge D1 der Buttons **RL1Hoch** und **RL1Runter** müssen mit dem Eingang D0 des **RL1Display** verbunden werden.

Dazu ist die linke Maustaste direkt in dem kleinen Dreieck des Ausganges festzuhalten, anschließend zieht man die Maus auf das kleine Dreieck des gewünschten Zieleinganges und lässt sie los. Die Verbindung zwischen den Elementen wird automatisch eingezeichnet. Fertig ist das erste Blockschaltbild, welches jetzt im Debugger getestet werden soll.

Dazu sollte das aktuelle Projekt an dieser Stelle gespeichert werden, damit nicht noch irgend etwas Wichtiges verloren geht. Im Beispiel wird der Name `test.apcp` gewählt. Anschließend kann der Debugger (welcher ebenso wie der Editor im Runtime-Paket nicht mit enthalten ist) durch einen Klick auf das Käfersymbol in der Werkzeugleiste gestartet werden.

Nun öffnet sich ein rahmenloses Fenster, in dem die Schaltflächen und die Statusanzeige zu sehen sind. Dieses Fenster entspricht der Darstellung auf dem Zielsystem. Abweichend zum Player öffnet der Debugger rechts daneben aber noch ein zusätzliches Statusfenster, welches diverse Meldungen anzeigt und im Fall von Problemen während der Laufzeit auch darüber informieren würde. Die weiteren Debuggerfunktionen wie Singlestepping, Variablenüberwachung sowie Programmflussunterbrechung bei bestimmten Ereignissen werden an dieser Stelle nicht weiter benötigt.



Erste logische Verknüpfungen im Flow-Editor.

Im Gegensatz zum OpenPlayer, welcher OpenAPC-Projekte in der Runtime „wiedergibt“, startet ein Projekt im Debugger nicht sofort und automatisch; dies geschieht erst durch einen Klick auf das Käfersymbol in der Werkzeugleiste. Mit demselben Button kann ein Programmfluss auch wieder unterbrochen werden. Wenn das Beispielprojekt nun gestartet wird, „erwachen“ die Buttons und zeigen schon erstes Leben: Werden die Buttons wechselseitig betätigt, so wird der jeweils andere Button deaktiviert (was durch den gegenseitigen Ausschluss **RL1** realisiert wird, der für beide Buttons gesetzt wurde). Solange einer der beiden Buttons betätigt ist, ist die Anzeige rot (da der Eingang D0 dieses Elements durch den Button ja auf HIGH gesetzt wurde). Nur wenn beide Buttons deaktiviert sind, zeigt auch die Anzeige wieder grün (entsprechend eines LOW-Signals von beiden Buttons am Eingang D0).

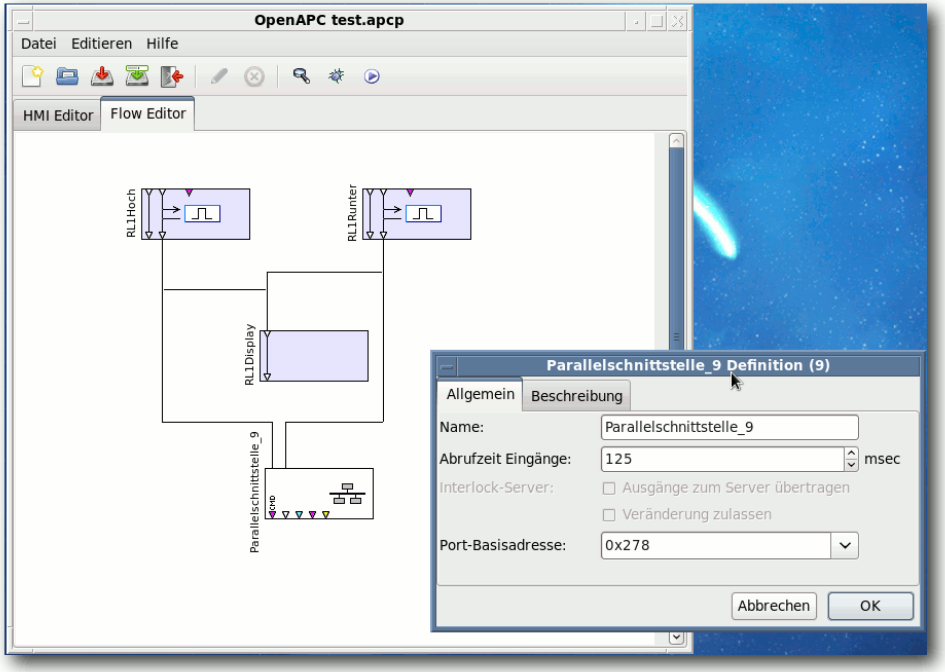
Die Hardwareanbindung

Was nun noch fehlt, ist die Verbindung zur Hardware, welche in diesem Fall per Parallelport realisiert werden soll. Dazu ist der Debugger zu schließen, da Änderungen am Blockschaltbild vorgenommen werden müssen. Hier soll erstmalig ein Element zum Einsatz kommen, welches in der eigentlichen Oberfläche nicht sichtbar ist, sondern vollständig im Hintergrund arbeitet.

Mit einem Rechtsklick im Flow-Editor öffnet sich wieder das Kontextmenü, welches unter „*Element hinzufügen → Ein/Ausgang*“ eine ganze Liste an unterschiedlichsten IO-Elementen anbietet. Für dieses Beispiel ist „*Parallelschnittstelle*“ die richtige Wahl. Es wird dem Blockschaltbild ein neues Element hinzugefügt, welches per Doppelklick ein Konfigurationsfenster öffnet. Die Möglichkeiten hier sind recht übersichtlich: Dem Element kann ein anderer Name gegeben werden, die Abrufhäufigkeit der Parallelporteingänge kann festgelegt werden (welche bei 125 ms belassen werden kann) und die Basisadresse der Schnittstelle muss angegeben werden (diese ist rechnerabhängig und lässt sich aus den BIOS-Einstellungen ermitteln).

Als nächstes sind noch Verbindungen von den Buttonausgängen zum Parallelportelement herzustellen. Leider ist das Symbol für das Parallelschnittstellenelement zumindest in der Version 1.0 der Software fehlerhaft: Das Element besitzt eigentlich acht digitale Eingänge D0 . . . D7 und fünf digitale Ausgänge D0 . . . D4, welche so aber leider nicht im Flow-Editor angezeigt werden. Die Verbindungen müssen also blind auf die Eingänge D0 und D1 gesetzt werden, funktionieren anschließend aber problemlos.

Der Versuch, dieses Projekt jetzt im Debugger zu starten, führt zu einer Überraschung: Dieser meldet, dass nicht auf die Parallelschnittstelle zugegriffen werden kann. Des Rätsels Lösung: Direkte Portzugriffe sind unter Linux nur dem Benutzer „root“ erlaubt, kein anderer Nutzer darf an echten Hardwareadressen herumpoken. Deshalb ist es an dieser Stelle erforderlich, das Projekt manuell und mit Root-Rechten zu starten. Ursache ist hier das Parallelschnittstellen-Plug-in, welches auf die Basisadresse des Parallelports zugreifen muss. Wird dieses Plug-in nicht verwendet, sind keine Root-Rechte erforderlich.



Der Flow-Editor: Leitungen führen hier ins Nichts, funktionieren aber dennoch.

Zum manuellen Starten ist in der Konsole entweder der Debugger mit

```
# OpenDebugger test.apcp
```

oder aber der Player mittels

```
# OpenPlayer test.apcp
```

aufzurufen.

Sofern jetzt tatsächlich irgendetwas am Parallelport angeschlossen ist, mit dem sich die Schaltvorgänge überwachen lassen, kann man sehen, wie die einzelnen Bits entsprechend den Zuständen der beiden Togglebuttons ein- und ausgeschaltet werden.

Damit ist nun bereits das erste Projekt realisiert – und das komplett ohne auch nur eine Zeile Code zu schreiben oder sich um das zu verwendende GUI-Toolkits zu kümmern.

Wie geht es weiter?

Die grundsätzliche Verwendung der OpenAPC-Software sollte anhand dieses kleinen Beispiels verdeutlicht worden sein. Es lohnt sich aber auf alle Fälle, noch ein wenig in der Software zu graben, zu testen, mit ihr zu spielen, zu experimentieren und Möglichkeiten auszuprobieren. Allein die Fülle an HMI- und Flow-Elementen, welche sich noch in den Kontextmenüs verstecken, lassen erahnen, was hier so alles machbar ist.

So finden sich für das GUI-Design die unterschiedlichsten Elemente: Buttons, Eingabefelder, Dreh- und Schieberegler, Checkboxen, Radiobuttons, Rohre, unterschiedlichste schaltbare Symbole, 2-D-Datenplotter, Textlabels, verschachtelbare Karteikartenreiter und anderes mehr. Mit dieser recht vollständigen Sammlung an GUI-Elementen sollte einer ansprechenden Oberfläche eigentlich nichts mehr im Wege stehen.

Ähnlich sieht es bei den Flowelementen aus, hier gibt es z. B. verschiedenste konfigurierbare Logikgatter, Konverterfunktionalitäten, um die verschiedenen Datentypen ineinander umwandeln zu können, Rechenoperationen, Elemente zur Steuerung und Beeinflussung des Programmablaufes und diverse Hardwaretreiber. Insbesondere letztere sind eine Fundgrube: Während Robotercontroller und Treiber zur Ansteuerung von Lasermaschinen für den Heimanwender eher uninteressant sind, gibt es hier auch die Möglichkeit, Stellmotoren einzubinden, auf MySQL- [7] und PostgreSQL-Datenbanken [8] zuzugreifen, mit einem LCDproc-Server zu kommunizieren, TCP/IP-Datenverbindungen herzustellen, Text als Sprache auszugeben und vieles mehr. Nicht unerwähnt bleiben sollen die ebenfalls bereits vorhandenen Treiber für das AVR Net-IO-Board [9], welches ob seines günstigen Preises gerade in Bastlerkreisen sehr beliebt ist.

Weiterführende Informationen bietet auch das Handbuch [10], welches ebenfalls auf der Downloadseite [6] des Projektes heruntergeladen werden kann, aber leider nur in Englisch vorliegt. Dort finden sich auch andere interessante Ressourcen wie einige Beispielprojekte sowie das Software Development Kit, welches für all jene interessant wird, die eigene Plug-ins schreiben wollen, um damit selbst gebaute Hardware ansteuern zu können. Auch hier ist die Beschreibung der Programmierschnittstellen allerdings wieder komplett in Englisch gehalten.

Fazit

Die OpenAPC-Software hat durchaus noch Mängel und offensichtliche Bugs, auch ist die deutsche Übersetzung teilweise lückenhaft. Allerdings kann sie mit nur wenig Tüftel- und Bastelarbeit durchaus produktiv genutzt werden. Allein die Tatsache, dass es sowohl unter Windows als auch unter Linux kein vergleichbares freies Softwarepaket gibt, macht OpenAPC aus meiner persönlichen Sicht ganz klar zu einer Empfehlung.

Hier lassen insbesondere die offenen und gut dokumentierten Programmierschnittstellen hoffen: Wenn die Open-Source-Community diese einmal für sich entdeckt, kann innerhalb kürzester Zeit ein unschlagbar großer und hilfreicher Pool aus Plug-ins entstehen.

Ein anderes Highlight, das nicht unerwähnt bleiben soll, ist die Portabilität: So verspricht die OpenAPC-Software, dass es egal ist, auf welchem Betriebssystem und auf welcher Plattform ein Projekt erstellt wurde, es wird in jedem Fall auch in einer Runtime eines anderen Systems laufen. Zumindest für x86 und ARM kann ein kleiner Test das bestätigen: Ein OpenAPC-Projekt, welches auf einem x86-Fedora erstellt wurde, funktioniert im OpenPlayer problemlos unter ARM/Debian.

LINKS

[1] http://www.openapc.com/

[2] http://www.heise.de/newsticker/meldung/Guenstige-Heimautomatisierung-dank-digitalStrom-979944.html

[3] http://www.frehmeat.net/

[4] http://www.sourceforge.net/

[5] http://lcdproc.org/

[6] http://www.openapc.com/download.php

[7] http://www.mysql.de/

[8] http://www.postgresql.org/

[9] http://www.pollin.de/shop/dt/NjI5OTgxOTk-/Bausaetze_Module/Bausaetze/AVR_NET_IO_Fertigmodul.html

[10] http://www.openapc.com/download/manual.pdf

Autoreninformation

Hans Müller ist als Elektroniker beim Thema Automatisierung mehr der Hardwareimplementierung zugeneigt als dem Softwarepart und hat demzufolge auch schon das ein oder andere Gerät in der privaten Wohnung verkabelt.

Diesen Artikel kommentieren