13 Fallstudie

Nachdem die einzelnen Komponenten zur Softwareerstellung – Compiler, Linker, Library, Make, Quellcode-Tools, Editoren und Simulatoren – besprochen wurden, soll nun eine Fallstudie folgen, in der die geschilderten Methoden und Werkzeuge Anwendung finden. Dieser Praxisteil soll die Kenntnisse ordnen helfen und die Theorie auflockern. In den vorangegangenen Kapiteln wurden alle Beispiele allgemein gehalten, da es nahezu unendlich viele verschiedene Prozessortypen mit den dazugehörigen Tools gibt. Dem Leser nutzt es wenig, wenn ein Tool oder ein Prozessor eines Herstellers bis in das Detail erklärt wird, er aber aus verschiedenen Gründen eine ganz andere Tool- und Prozessorauswahl getroffen hat. Bei der hier geplanten Fallstudie kann man natürlich eine solche Vorgehensweise nicht durchhalten. Es muss alles konkretisiert werden. Bei der Auswahl der Komponenten habe ich mich von folgenden Kriterien leiten lassen:

Die Komponenten sollen möglichst weit verbreitet, einfach zu beschaffen, möglichst preiswert oder kostenlos und trotzdem leistungsfähig sein. Außerdem ist die hier vorgestellte Fallstudie einfach gehalten, da nur die Prinzipien des Systemdesigns und der Softwareentwicklung und keine seitenlangen Codelistings gezeigt werden sollen. Das einfache Beispiel hat, im Gegensatz zum in der Literatur viel zitierten Geldautomaten, den Vorteil, dass man dieses Beispiel komplett beschreiben und konkretisieren kann. Es handelt es sich um ein vollständiges Projekt, das in sich abgeschlossen ist und auch funktioniert. Aus reinen Platzgründen wurden ab und zu Kleinigkeiten, die für das Verständnis nicht unbedingt nötig sind, weggelassen. Der vollständige lauffähige Sourcecode befindet sich aber auf der beigefügten DVD. Die Fallstudie kann auf mehrere Arten zur Vertiefung des Stoffs beitragen:

– Sie kann als reines „Leseobjekt“ dienen und so die bisher beschriebenen Kenntnisse vertiefen.
– Das Projekt kann mit zwei verschiedenen Simulatoren am PC durchgespielt werden.
– Es ist aber zusätzlich möglich, die einfache Hardware aufzubauen und so praxisorientiert ein kleines Projekt zu entwickeln.

13.1 Anforderungen an das System

Zuerst muss eine genaue Beschreibung der Anforderungen an ein System erstellt werden. Idealerweise liegt diese in Form eines *Pflichtenhefts* bereits vor. Zur Thematik der Pflichtenhefterstellung finden sich ausführliche Darstellungen im „Lehrbuch der Softwaretechnik“ [11]. Meistens ist es jedoch Aufgabe der Entwicklung, dieses oder zumindest eine Systembeschreibung mit einer Beschreibung der erforderlichen Funktionen zu erstellen. Danach muss man sich Gedanken über die *Benutzeroberfläche* des Systems machen. Unter der Benutzeroberfläche versteht man eine Beschreibung, wie dem Benutzer die einzelnen Funktionen des Systems zugänglich gemacht werden sollen. Nun kann das eigentliche Systemdesign erfolgen, indem verschiedene Funktionsgruppen in der Software gebildet werden.

Um das Prinzip zu verdeutlichen, soll hier eine einfache Torsteuerung entworfen werden. Alle mechanischen und elektromechanischen Komponenten werden als vorhanden vorausgesetzt. Es geht nur darum, Prozessorausgänge entsprechend der Motorsteuerung zu betätigen und für dieses System die Software zu erstellen. Es handelt sich hier um ein reines Demonstrationssystem. Dies schließt auch jegliche Haftung für Fehlfunktionen (z. B. zerquetschte Autos oder Haustiere) aus!

Für die Fallstudie sei folgende Beschreibung der gewünschten Funktionalität gegeben:

Das Rolltor wird über einen Elektromotor angetrieben, der für jede Laufrichtung eine Steuerleitung besitzt. Die Torendposition wird dem System über zwei Endschalter am oberen und unteren Ende des Torbereichs gemeldet. Der Tormotor wird über zwei Ausgänge des Mikrocontrollers gesteuert. Ein Ausgang betätigt den Motor in Aufwärtsrichtung, der andere Ausgang in Abwärtsrichtung. Zur Bedienung durch den Benutzer ist ein Taster vorhanden. Nach Betätigung des Tasters wird das Tor für 5 s geöffnet, nach Ablauf der Zeit wieder geschlossen. Beim Herunterfahren wird das Tor durch eine Lichtschranke gesichert. Spätestens 100 ms nach der Unterbrechung der Lichtschranke muss der Motor gestoppt und das Tor wieder für 5 s geöffnet werden. Außerdem ist es möglich, das Herunterfahren durch die Betätigung des Schalters zu unterbrechen, das Tor wieder zu öffnen und es für eine erneute Zeit von 5 s wieder offen zu halten. Wird während der Öffnungszeit der Schalter betätigt, so wird die Öffnungszeit erneut auf 5 s gesetzt. Das Tor benötigt ca. 8 s zum Öffnen und Schließen. Wenn nach dem Start des Antriebs in die Zu- oder Auf-Richtung nicht nach 10 s ein Endschalter erreicht wird, wird der Antrieb abgeschaltet und das System geht in einen Fehlerzustand über, in dem keine externen Ereignisse mehr angenommen werden sollen. Es reagiert jetzt so lange nicht auf die Tasterbetätigung, bis das System aus- und wieder eingeschaltet wurde (Reset). Dieser Fehlerzustand wird durch das Dauerleuchten der Kontroll-LED angezeigt. In allen normalen Betriebszuständen blinkt die LED.

Das Tor und die angebrachten Einrichtungen sind im ***Bild 13.1*** skizziert.

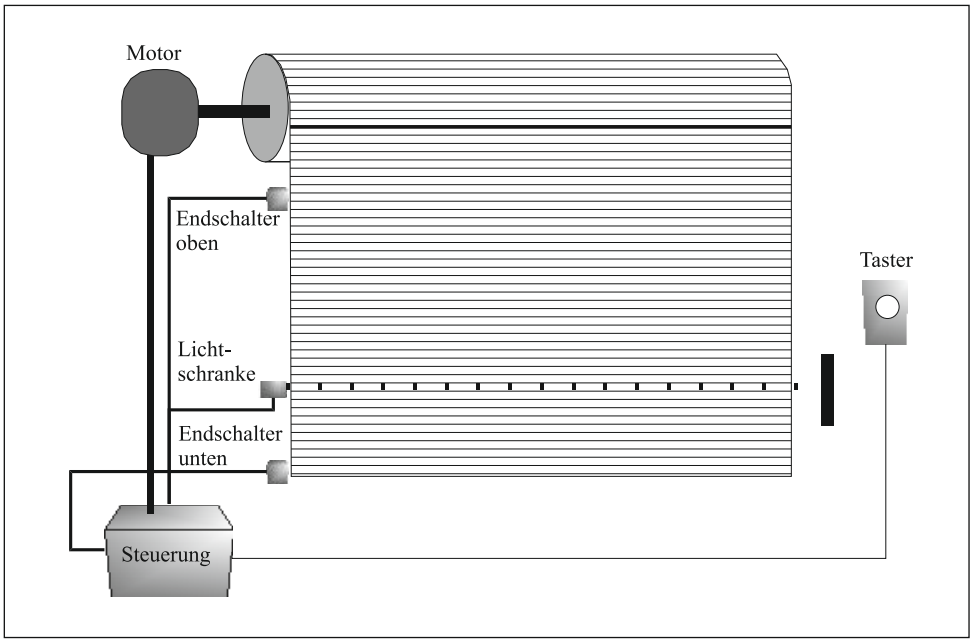

Bild 13.1: *Skizze des Rolltors*

Damit der Leser das Projekt leicht nachvollziehen kann, sollen die Ausgaben für die Entwicklungsumgebung möglichst gering gehalten werden. Für die Entwicklungsumgebungen anderer Prozessoren gehen übrigens leicht einige tausend Euro über den Tisch.

Zuerst werden jetzt die einzelnen Komponenten des Softwaresystems erfasst. Es werden benötigt:

1. Betriebssystem,
2. Motortreiber,
3. Treiberkomponenten für Endschalter, Taster und Lichtschranke,
4. Steuerungslogik.

Neben dem Betriebssystem besteht das Softwaresystem aus Treibern für die verschiedenen Eingangs- und Ausgangssignale und der eigentlichen Steuerungslogik. Als *Treiber* werden alle Softwarekomponenten bezeichnet, die sich entweder mit dem Erkennen von Eingangssignalen befassen oder für die angeschlossene Hardware Ausgangssignale erzeugen. Die *Steuerungslogik* bekommt von den Treibern entsprechende Signale und verarbeitet diese weiter. Die Steuerungslogik erzeugt aufgrund ihrer internen Berechnungen Ausgangssignale, die vom Motortreiber in die entsprechenden Signale für den Motor (bzw. die vorgeschalteten Relais) umgesetzt werden. Im vorliegenden Beispiel ist der Motortreiber ein einfaches Programm, welches den entsprechenden Port setzt, der wiederum über eine Verstärkerstufe die Relais zur Motorbetätigung ansteuert.

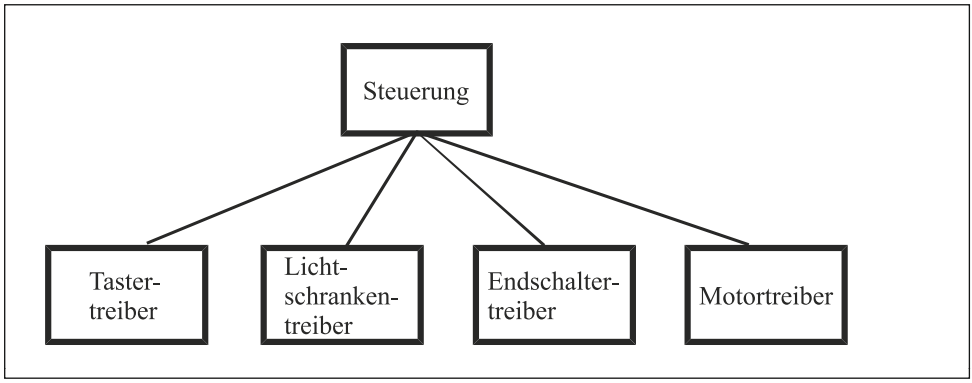

Bild 13.2: *Top-down-Design der Steuerungssoftware*

Um aus den aufgezählten Komponenten ein System zu designen, wird das Top-down-Design angewendet. Die Steuerungslogik befindet sich an der Spitze der Systemhierarchie. Die Steuerungslogik erhält die Eingangssignale von den Modulen Endschalter, Taster- und Lichtschrankentreiber. Die Befehle von der Steuerungslogik werden über den Motortreiber in elektrische Signale zur Ansteuerung des Motors umgewandelt. Die Steuerungslogik besteht hauptsächlich aus einem einfachen endlichen Zustandsautomaten. Als Top-down-Design dargestellt, ergibt sich ***Bild 13.2***. Obwohl in der Aufzählung der Systemkomponenten das Betriebssystem vorkam, findet es sich nicht in der grafischen Darstellung des Top-down-Design wieder. Die Dienstleistungen des Betriebssystems, wie Timerverwaltung, Kommunikation und Verteilung der Rechenzeit auf die einzelnen Systemkomponenten, finden sich üblicherweise nicht im Systemdesign wieder, da sie keinen Einfluss auf das Design des Systems haben. Die Anforderungen an das Betriebssystem ergeben sich aus dem Systemdesign, sind aber keine Komponente in der grafischen Darstellung. Anforderungen, die sich aus dem Systemdesign ergeben können, sind z. B., dass das Betriebssystem hochgenaue Timer zu Verfügung stellen muss, dass bei der Kommunikation zwischen den Tasks große Datenmengen bewegt werden müssen oder dass ein preemptives Scheduling unterstützt werden muss.

Da die Aufgabe in einem kleinen Rahmen liegt und die Kosten für das System gering gehalten werden sollen, wurde als Mikrocontroller der Single-Chip-Prozessor ATmega88 von Atmel gewählt. Der Prozessor enthält alle benötigten Systemkomponenten wie Flash-ROM, RAM, Timer und Interruptlogik. Der Mikrocontroller selbst ist sehr preiswert (ca. 6 €) und die benötigte Entwicklungsumgebung ist als GNU-Software kostenlos zu beziehen. Um schnell mit der Fallstudie arbeiten zu können empfehle ich das *myAVR Board MK2 USB*. Es besteht aus einer flexiblen Experimentierplatine und einer Programmiereinrichtung für den Mikrocontroller über die USB-Schnittstelle. Weitere Komponenten wie Summer, Potentiometer, Leuchtdioden und Schalter runden das Board ab. Mit dem optional erhältlichen Kabelsatz *USB* bietet die Experimentierplatine die Möglichkeit, die Hardware für das Fallbeispiel in 10 Minuten ohne irgendwelche Lötarbeiten aufzubauen. Die USB-Schnittstelle ist auf jedem Desktoprechner und Laptop vorhanden und wird auch so

schnell nicht wieder aus den Systemen verschwinden. Die Stromversorgung kann in der Entwicklungsphase direkt über den USB-Anschluss erfolgen oder wahlweise über ein Netzteil oder eine Batterie. Eine nähere Beschreibung, der hier verwendeten Komponenten mit einer Bestellempfehlung findet sich Kapitel 15. Das Board ist standardmäßig mit dem Mikrocontroller AtMega8 bestückt. Dieser würde zwar ohne weiteres den hier gestellten Anforderungen genügen, aber es handelt sich dabei um einen älteren Prozessor, der einigen Einschränkungen bezüglich des Debuggens unterliegt und vielleicht irgendwann nicht mehr verfügbar sein wird. Der Nachfolger des Prozessors ist der hier verwendete und beschriebene ATmega88. Dieser modernere Prozessor muss im Moment noch zusätzlich zum Board separat bestellt werden.

Der ATmega88 stellt 2 Interrupteingänge zur Verfügung, die für die Eingänge Taster und Lichtschranke verwendet werden. Die Auswertung der Endschaltereingänge erfolgt über Scannen (oder Polling) durch Software (siehe Abschnitt 9.4). Die Ausgänge des Systems werden über die Prozessorports realisiert. Jeweils ein Prozessorport wird

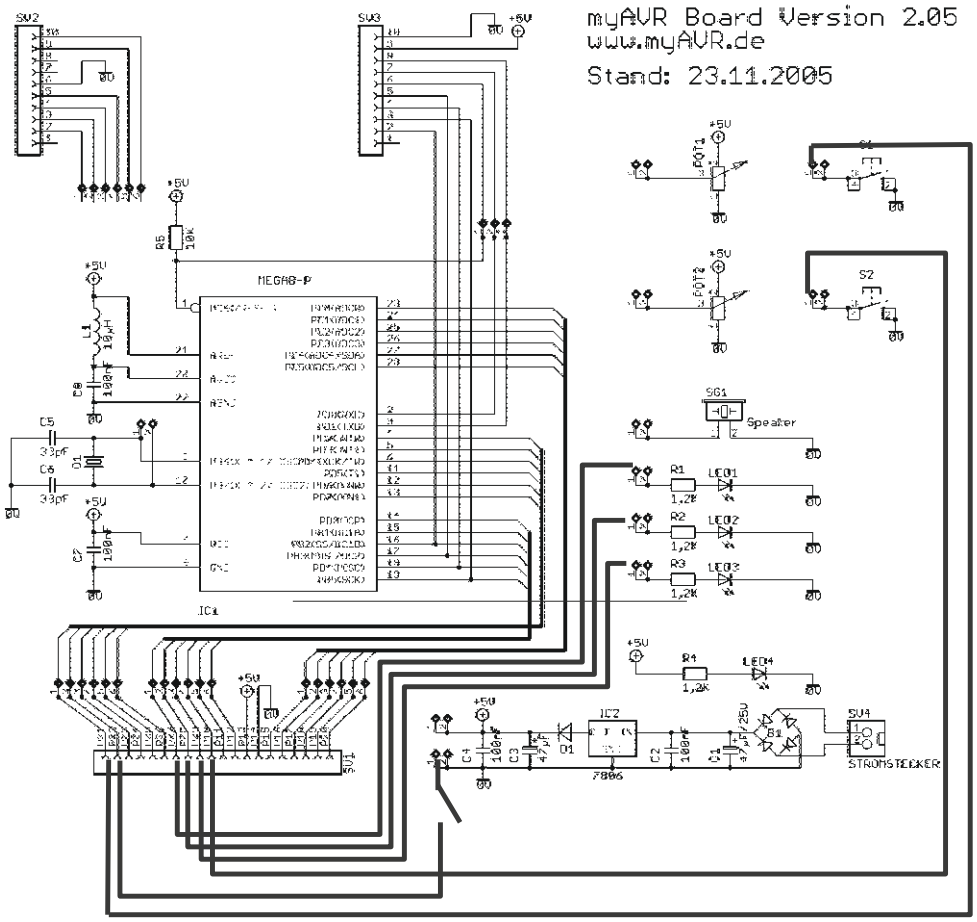

Bild 13.3: *Schaltplan der Fallstudie mit zusätzlichen Drahtbrücken*

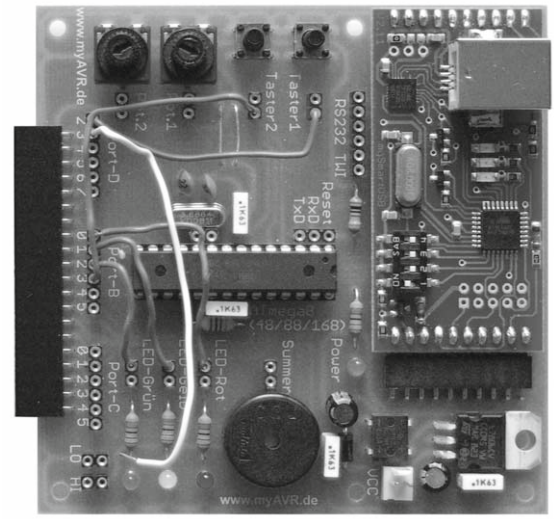

Bild 13.4: *Platine mit Drahtbrücken*

für die Signale Auf, Ab und die Fehleranzeige über die LED benötigt. Der Schaltplan des Prozessorsystems ist im ***Bild 13.3*** dargestellt. Die zusätzlich notwendige Verdrahtung ist im Schaltplan durch die dicken Linien verdeutlicht. Auf dem Foto (***Bild 13.4***) kann man die Verdrahtung sehen, die in ungefähr 10 Minuten erstellt wurde. Auf die „Starkstromtechnik“ und die Umsetzung der Lichtschranke wurde verzichtet, da es ja hier nur um die Softwaretechnik des Systems gehen soll. Die Lichtschranke liefert einen Low-aktiven Ausgang, der direkt an den Interrupteingang des Prozessors angeschlossen wird und der hier, mangels Schalter, durch eine Drahtbrücke simuliert wird. Die Demonstrationsschaltung wird mit 5 Volt betrieben und kann über den USB-Anschluss des Programmers mit Strom versorgt werden. An den beiden Prozessorausgängen für die Motorsteuerung sind die LEDs über Vorwiderstände angeschlossen, um die Funktion des Systems optisch zu kontrollieren. Die beiden auf der Platine vorhandenen Taster bilden den Betätigungseingang und den Eingang für die Endschalter und die Lichtschranke nach. Als Steckmodul sitzt auf der Platine ein USB-Programmer, der dazu verwendet wird, die übersetzte und gelinkte Software von der Computerfestplatte auf den Mikrocontroller zu übertragen. Der USB-Programmer stellt eine virtuelle serielle Schnittstelle zur Verfügung, die dann von der Programmiersoftware auf dem Rechner angesprochen wird.

13.2 Kurze Einführung in den Mikrocontroller AVR ATmega88

Um die hier gestellten Anforderungen zu erfüllen, passt der Mikrocontroller ATmega88 ideal. Die gesamte Peripherie des Mikrocontrollers (Blockschaltbild in ***Bild 13.5***; das Bild stammt aus dem AVR-Prozessorhandbuch [12]) einschließlich RAM, Flash-ROM, Timer und EEPROM befindet sich auf einem Chip. Der RAM-Speicher hat