Reaktionsspiel mit Bandgeräten

Softwareprojekt: Internetkommunikation

Steve Dierker, Patrick Hjort, Semjon Kerner

18. Oktober 2017

1 Einführung

Das Ziel dieses Projektes ist es, die *PDP11* im Foyer des Informatikinstituts der FU Berlin wieder mit Leben zu füllen und sie nicht nur als leere Hülle zu präsentieren. Uns ist es dabei besonders wichtig unsere Mitstudierende zum eigenständigen entdecken der *PDP11* zu animieren. Daher haben wir die bestehende Bandlaufwerke mit einem modernen Mikrocontroller ausgestattet, der von nun an die Ansteuerung übernimmt. Dieser Mikrocontroller, die vorhandene Bandlaufwerke, ein paar weitere LEDs und ein LCD ermöglichen es uns die *PDP11* in Arcade ähnliche Spieleautomaten umzuwandeln.
Ziel dieses Projektes ist es, die Hardware fertigzustellen und ein Beispielspiel zu implementieren, das es zwei Spieler_innen ermöglicht sich in ihrer Reaktionszeit zu messen.
Als erstes werden wir einen Überblick über die verwendete Hardware geben und auf die Besonderheiten und Probleme bei der Ansteuerung dieser eingehen. Danach gehen wir auf die Software selbst und die Kommunikation unterhalb der Knoten ein. Abschliessend ziehen wir ein vergleichendes Resume zwischen der Planung und dem Erfolg des Projekts und bieten einen Ausblick auf weitereführende Ideen.

2 Hardware

2.1 vorhandene Hardware

Als Basis für unser Projekt dient die im Foyer stehende und leider nicht mehr funktionstüchtige *PDP11*. Im besonderen haben wir uns für die zwei Bandlaufwerke der *PDP11* interessiert, da sie jeweils neun Taster und zwei einzeln ansteuerbare Motoren bieten. Die Motoren können im und gegen den Uhrzeigersinn drehen und bieten durch die Geräuschentwicklung sowie ihre Bewegung einen guten Auslöser für unser Reaktionsspiel.
Von den neun Tastern sind zwar alle mit unserem Mikrocontroller verbunden und auch per Software ansteuerbar, allerdings sind derzeit nur die ersten drei Taster in Verwendung.

2.2 zusätzliche Hardware

Für dieses Projekt wurde uns ein *SAMR21 Xpro* zur Verfügung gestellt. Der *SAMR21 Xpro* ist ein Evaluationsboard für den *ATSAMR21G18A*. Das Board bietet integrierte Funkunterstützung zur Kommunikation und durch den *Cortex-M0+* auch ausreichend Leistung um unser Projekt umzusetzen. Im Zuge der Softwareentwicklung sind wir nicht an die Leistungsgrenzen des Chips gestoßen, allerdings bietet er nicht genügend I/O-Pins zur Ansteuerung der gesamten Peripherie.

In Ermangelung ausreichender I/O-Pins haben wir uns dazu entschieden die Taster über den *S74LS151* anzusteuern. Der *S74LS151* ist ein 8Bit Multiplexer und ermöglicht es uns alle Taster anzusteuern, allerdings kann nur ein Taster gleichzeitig mit einem Interrupt belegt werden.

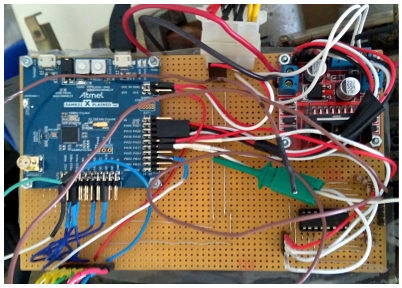

Abbildung 1: Mikrocontroller, Multiplexer und H-Brücke

Zur Beleuchtung der Bandlaufwerke haben wir RGB-LED-Streifen verbaut. Diese LED-Streifen basieren auf dem *WS2811*, der jeweils eine LED steuern und bis zu 1024mal in Reihe geschaltet werden kann um komplette Animationen zu steuern. Die existierenden Glühlampen der Taster wollten wir ebenfalls durch LED's ersetzen die auf dem *WS2811* basieren, allerdings gab es hier Lieferprobleme.

Zur Ansteuerung der Bandlaufwerkmotoren verwenden wir die doppel H-Brücke *L298-H*. Diese benötigt als einziges Modul eine 12V Versorgungsspannung.
Für zukünftige Menüführung, Eingabe der Nicknames und weiteres visuelles Feedback haben wir ein 128x64 Pixel LCD des Typs *DOGL128B-6* verbaut. Dieses Display ist ein Set aus Display und Controller und kann einfach über das SPI Protokoll verwendet werden. Um Kosten an der Hardware zu sparen haben wir uns dazu entschieden nicht vorgefertigte Backlight dazu zukaufen, sondern jeweils zwei *WS2811* RGB-LEDs für die Hintergrundbeleuchtung zu verwenden.
Unsere verbaute Hardware brauch eine $3,3V$, $5V$ und $12V$ Spannungsversorgung und anstatt selber ein Netzteil zu bauen oder zu kaufen haben wir ein ATX-Netzteil aus Restbeständen für beide Bandlaufwerke benutzt.

2.3 Treiber

Nach der Entwicklung der Hardwareplattform haben wir für jede Peripherie einen Teiber entwickelt, da diese noch nicht von RIOT selbst unterstützt wurden. Jeder Treiber ist entsprechend den von RIOT angebotenen Beispieltreibern implementiert. Die Treiberimplementierung war für alle Peripheriegeräte außer dem WS2811 unkompliziert. Der WS2811 bietet einen digitalen Eingang zum Empfang von Farbinformationen und einen digitalen Ausgang zum Senden von Farbinformationen an den nächsten WS2811 in Reihe. Zum Schluss steuert jede WS2811 eine LED und bis zu 1024 WS2811 können miteinander verkettet werden, um ganze LED-Arrays zu steuern. Da der WS2811 nur ein OneWire-Protokoll und keine zusätzliche Taktung anbietet, benötigt er für die Unterscheidung zwischen 0 und 1 in seinem Protokoll harte Timings. Für unser Projekt haben wir zwei verschiedene WS2811 LED-Sets gekauft. Das erste ist ein bereits montierter LED-Streifen, der mit unserem Treiber tadellos funktioniert und das zweite ist ein Set von separaten Chips und LEDs, die selbst zusammenzubauen sind. Das zweite Set weigerte sich, mit unserem Treiber zu arbeiten und nach mehrtägigem Debugging stellte sich heraus, dass der Controller eine gemeinsame Annode benötigt, aber die LEDs, die wir erhalten haben, boten eine

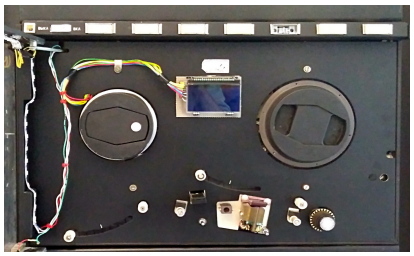

Abbildung 2: Frontansicht der Bandlaufwerke

gemeinsame Kathode. Leider war der Internetversand, bei dem wir die LEDs gekauft haben, nicht in der Lage, die richtigen LEDs rechtzeitig zur Verfügung zu stellen, um das Projekt komplett abzuschließen.

3 Retro11

Retro11 ist die C-Basierte RIOT-Anwendung, die auf dem *SAMR21 XPro* läuft. Neben der Kontrolle über die vorhandene Hardware wird diese Anwendung mit einem einfachen Spiel ausgeliefert, um in der Reaktionszeit zu konkurrieren, welche weiter unten erklärt wird. Um Spieler gegeneinander antreten zu lassen, ist es notwendig, zwischen den Spielplattformen zu kommunizieren. Dazu implementiert die Anwendung einen COAP-Client und -Server.

3.1 Anwendungsdesign

Was das Anwendungsdesign selbst betrifft, initialisieren wir zunächst alle benötigten Treiber und starten dann vier Threads, um die verschiedenen Aspekte unserer Anwendung zu steuern. Der erste Thread ist der MotorController, der auf Meldungen zum Starten oder Stoppen des Motors wartet. Da wir die Motoren für eine bestimmte Zeit benötigen, kann man auch ein Timeout übergeben, nach dem der Motor gestoppt werden soll. Der zweite Thread ist der CoapServer, der die Kommunikation zwischen den beiden Knoten handhabt und die Endergebnisse auch an das RasperryPi von team2 weiterleitet, um eine webfähige Highscore zu erstellen. Der dritte Thread ist das Spiel selbst, das die Kontrolle über alle Peripheriegeräte übernimmt und mit dem CoapServer über eine Shared-State-Variable kommuniziert. Das Spiel steuert die Benutzereingaben und -ausgaben und bietet ein Arcade-ähnliches Spielerlebnis. Der vierte und letzte Thread dient zu Debugging-Zwecken und führt einfach eine Shell mit benutzerdefinierten Befehlen für Hardware-Diagnose und andere Debuggingoperationen aus. Er ist nur zugänglich, wenn ein Computer an den USB-Port des SAMR21 Xpro angeschlossen wird.

3.2 Reaktionsspiel

Für dieses Spiel haben Sie mindestens zwei Spieler, die beide vor einem der PDP11 an der FU Berlin stehen. Zuerst werden sie aufgefordert, dort einen Nickname einzugeben, der an den zentralen Server geschickt wird, um später die Reaktionszeit zu speichern und einen Highscore anzuzeigen. Das Spiel startet die Bandlaufwerkmotoren der beiden Knoten und sobald sie gestoppt werden, wird der Spieler aufgefordert, so schnell wie möglich zu reagieren und einen Knopf zu drücken. Die Zeit zwischen dem Anhalten des Motors und der Reaktion des Spielers ist die Reaktionszeit, die dann zur Ermittlung des Siegers herangezogen wird. Den Highscore können Sie in der Webapplikation von team2 des diesjährigen Softwareprojekts nachschlagen.

3.3 Netzwerkkommunikation

Die Kommunikation zwischen den Plattformen erfolgt über das RESTful Constrained Application Protocol, CoAP, das Nachrichten über UDP und IPv6 sendet, wie in Abbildung 3 . Ein Mikrocontroller führt eine Client-Applictation aus, die das Spiel steuert. Die benötigten Informationen werden von einer Server-Anwendung bereitgestellt, die auf jedem Mikrocontroller ausgeführt wird.

Abhängig vom Spielstatus fragt der Client die Ressource mit den erforderlichen Informationen beider Server mit einer Get-Methode ab, bis die Server die Informationen bereitstellen, um zur nächsten Stufe überzugehen. Im ersten Zustand fordert der Client die Nicknamen des Spielers an, fordert dann das eigentliche Reaktionsspiel auf, im zweiten Zustand zu starten, fordert die Reaktionszeit im dritten Zustand an und fordert jede Maschine auf, sich anzeigen zu lassen, ob sie im letzten Zustand gewonnen oder verloren hat. Zusätzlich verfügen beide Server über eine Ressource, die auf Wunsch von team2 Highscore-Informationen im SenML-Format bereitstellt.

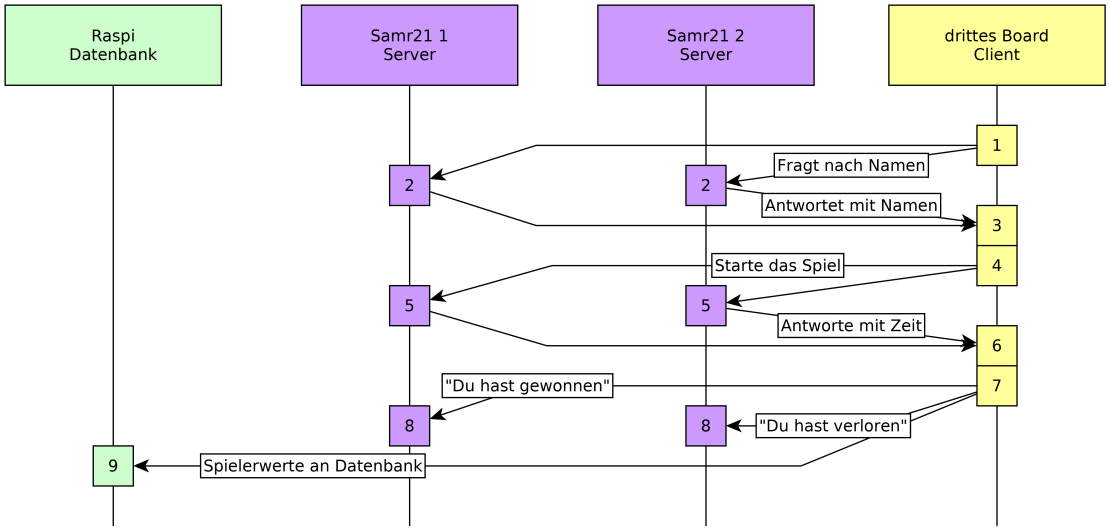

Abbildung 3: Sequenzdiagramm der Anwendung

4 Stand der Umsetzung

4.1 Stand

Kommunikation zwischen Knoten Hardware ($H-brcke$, $Motor$, $Multiplexer$, $Buttons$, LCD, $LED-Stripe$) Treiber Basisspiel

4.2 Probleme

LEDs von Buttons fehlen Kommunikation mit Raspi/Website/Coap-Gateway LCD Kontrast LCD Backlight

4.3 Rückblick auf Zeitplan

Zeitplan nicht eingehalten Hardware hat mehr Zeit gekostet als erwartet Probleme mit Netzwerkkommunikation Gar nicht nach Magnetbändern gesucht Spiel runterschreiben ging am Ende dann schnell Entwurf Poster war schnell aber nicht so früh wie geplant

5 Weiterführende Arbeit

Zuerst gilt es die bis jetzt nicht gelieferten RGB-LED's wie in Abschnitt 2.2 beschrieben in die Taster einzubauen. Bis jetzt werden die zu drückenden Taster nicht visuel hervorgehoben, sodass für Nutzer nicht offensichtlich ist welcher Taster welche Funktionsbelegung that.

Zur weiteren Verschönerung der Bandlaufwerke sollten die fehlenden Tasterfront gedruckt werden. Ein druckfertiges 3D-Modell ist schon im Repository vorhanden. Im Zuge dessen sollten auch die Motoren gereinigt werden, da sie sich zur Zeit erst bei einer verhältnismäßig hohen Anlaufspannung in Bewegung setzen. Dies würde Lautstärkeentwicklung und Stromverbrauch optimieren.
Im Bezug auf *Retro11* sollte als erstes ein Framework entwickelt werden in dem weitere Spiele implementiert werden können. Eine Möglichkeit besteht darin einen Lua-Interpreter in *Retro11* einzubetten. znteressierte Studierende sind dann in der Lage Spiele in einer Hochsprache gegen eine konsistente API zu entwickeln und müssen nicht notwendigerweise mit RIOT vertraut sein. Insbesondere für Programmieranfänger bietet das einen niederschwelligen Einstieg in eingebettete Systeme.
Da regelmäßige manuelle Updates der vorhandenen Spiele auf der Platform ein zeitintensives Unterfangen ist, sollte als nächstes über eine Over-the-Air Updatefunktion für die komplette Applikation oder zumindest für die Spielebibliothek nachgedacht werden. Für RIOT selbst ist gerade ein OTA-Update in Entwicklung das nach Fertigstellung genutzt werden kann.