Reaktionsspiel mit Bandgeräten

Softwareprojekt: Internetkommunikation

Steve Dierker, Patrick Hjort, Semjon Kerner

23. 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 Mitstudierenden zum eigenständigen entdecken der *PDP11* zu animieren. Daher haben wir die bestehenden Bandlaufwerke mit einem modernen Mikrocontroller ausgestattet, der von nun an die Ansteuerung der Hardware übernimmt. Dieser Mikrocontroller, die vorhandenen Bandlaufwerke, zusätzliche 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 Spielenden 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 zwischen den Knoten ein. Abschließend ziehen wir ein vergleichendes Résumé zwischen der Planung und dem Erfolg des Projekts und bieten einen Ausblick auf weiterführende 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 besitzen. 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* zu verwalten. Der *S74LS151* ist ein 8*bit* Multiplexer und ermöglicht es uns alle Taster mittels Polling abzufragen, allerdings kann immer nur ein Taster gleichzeitig mit einem Interrupt belegt werden.

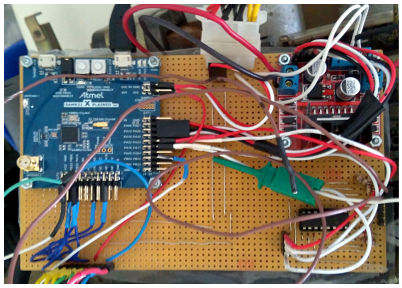

Abbildung 1: Mikrocontroller, Multiplexer und H-Brücke

Zur Beleuchtung der Bandlaufwerke haben wir RGB-LED-Streifen verbaut. Diese LED-Streifen sind kompatibel zu dem *WS2811*, der jeweils eine LED steuern und bis zu 1024 mal in Reihe geschaltet werden kann um komplette Animationen darzustellen. 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 kein zugehöriges Backlight dazu zukaufen, sondern jeweils zwei *WS2811* RGB-LEDs für die Hintergrundbeleuchtung zu verwenden.
Unsere verbaute Hardware braucht eine $3,3V$, $5V$ und $12V$ Spannungsversorgung und anstatt selbst 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* nutzt eine Schieberegisterarchitektur. Über einen digitalen Eingang zum Empfang und einen digitalen Ausgang zum Senden an den nächsten *WS2811* in Reihe werden je LED 24*bit* an Farbinformationen übertragen. Jeder *WS2811* steuert 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* kompatible 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,

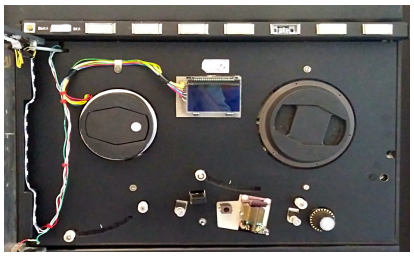

Abbildung 2: Frontansicht der Bandlaufwerke

aber die LEDs, die wir erhalten haben, boten eine 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 der Name der RIOT-Anwendung die auf dem *SAMR21 Xpro* läuft. Die Anwendung besteht aus drei Teilen. Als erstes ist in *Retro11* das Reaktionsspiel implementiert, das auch die Kontrolle über die gesamte Preipherie benötigt. Als zweites enthält jeder *Retro11* Knoten einen COAP-Server der Anfragen vom *Game-Master*-Knoten entgegennimmt. Als drittes und letztes läuft auf einem Knoten zusätzlich den *Game-Master*-Knoten, der die ablaufenden Spiele koordiniert und die Ergebnisse an den RasperryPi von *Team2* veröffentlicht.

3.1 Anwendungsdesign

Als erstes initialisiert *Retro11* die gesamte Peripherie und startet bis zu fünf Threads, die über eine Message-Queue sowie einen geteilten Status kommunizieren.
Der erste Thread ist der *MotorController*, der sich um die asynchrone Steurung der beiden Gleichstromotoren kümmert. Dieser Thread akzeptiert Nachrichten, die die Geschwindigkeit und die Laufdauer der Motoren enthält. Dieser Thread ist essentiel für das Reaktionsspiel, da die Motor sonst nicht asynchron zur Benutzereingabe angehalten werden könnten.
Der zweite Thread ist der *COAPServer*, der Anfragen des *Game-Masters* erwartet und dann durch eine geteilte Status Variable die Game-Loop des Reaktionsspielthreads kontrolliert.
Der dritte Thread ist das Reaktionsspiel, das in einer endlosen GameLoop läuft und durch die mit dme *CoapServer* geteilte Status Variable kontrolliert wird. Weiterhin kommunizert dieser Thread wie oben erwähnt mit dem *MotorController* und bildet das Herzstück von *Retro11*.
Der vierte Thread stellt eine Diagnose-Shell über den USB-Anschluss des *SAMR21 Xpro* bereit. In dieser Shell sind spezielle Hardware-Diagnose Befehle enthalten und außerdem werden Debuginformation über den derzeitigen Programmverlauf ausgegeben.
Der fünfte und letzte Thread ist nur auf einem der beiden Knoten vorhanden und stellt den *GameMaster* zur Verfügung. Dieser kommuniziert mit den beiden *CoapServern* und kontrolliert den gesamten Spielablauf zwischen den beiden Knoten. Der *GameMaster* ist kommuniziert außerdem mit dem RasperryPi von *Team2* und veröffentlich die Spielergebnisse.

3.2 Reaktionsspiel

Für dieses Spiele befinden sich zwei Spielende vor je einem Bandlaufwerk. Zuerst werden beide aufgefordert einen Nickname einzugeben, der dann an den *GameMaster* geschickt wird. Sobald beide Spielenden ihren Namen eingegeben haben, sendet der *GameMaster* den Start-Befehl und die Motoren beginnen sich zu drehen. Nach einer zufällig gewählten Zeit hören beide Motoren auf zu drehen und die Spielenden müssen möglichst schnell den Bestätigen-Knopf drücken. Die Zeit zwischen dem Anhalten der Motoren und dem Drücken des Knopfes ist die Reaktionszeit und wird von beiden Knoten an den *GameMaster* geschickt. Dieser entscheidet nun welcher der beiden Spielenden gewonnen hat und veröffentlich das Ergebniss in der

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-Application 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 ruft der Client die Nicknamen der Spielenden ab, fordert dann das Reaktionsspiel auf, im zweiten Zustand zu starten, erhält im dritten Zustand die Reaktionszeit und fordert anschließend jede Maschine auf, anzuzeigen, ob der Spielende gewonnen oder verloren hat. Zusätzlich verfügen beide Server über eine Ressource, die auf Wunsch von team2 Highscore-Informationen im SenML-Format bereitstellt.

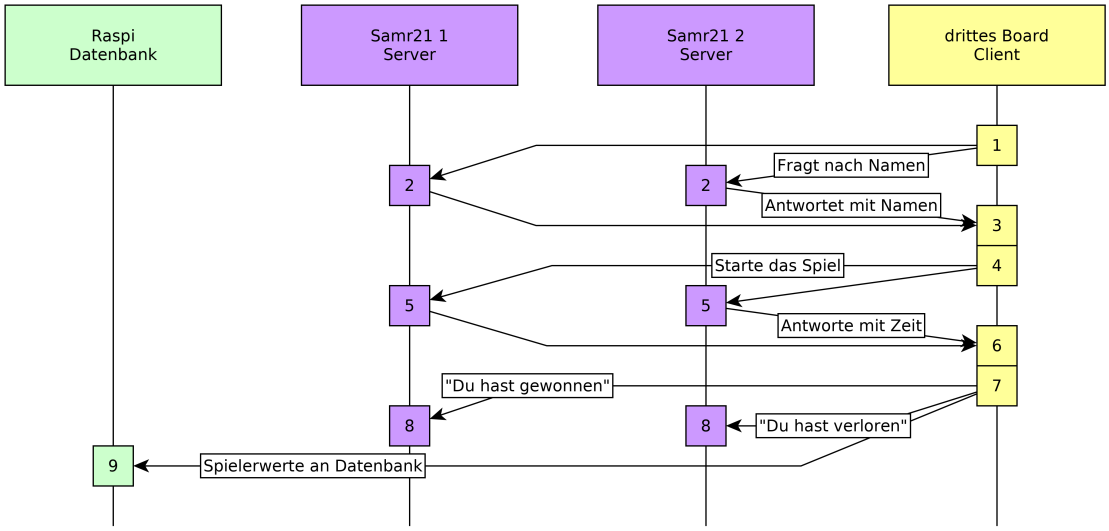

Abbildung 3: Sequenzdiagramm der Anwendung

4 Stand der Umsetzung

4.1 Probleme

- LEDs von Buttons fehlen
- Kommunikation Raspi
  - konsistente API zu netzwerkcode, keine trennung von businnes zu netzwerk
  - netwzwerk unreliable
  - integration in team2 ungenügend
- LCD Kontrast, Backlight, Wackelkontakt
- Kein dokumentiertes Board layout

4.2 Rückblick auf Zeitplan

- Zeitplan nicht eingehalten, warum?
- Hardware hat mehr Zeit gekostet als erwartet
- Probleme mit Netzwerkkommunikation
- Gar nicht nach Magnetbändern gesucht
- Spiel runterschreiben ging am Ende dann schnell, erahrungswerte haben gefehlt
- mangelhafte schnittstellen definition zwischen Application und Netzwerk stack
- trotz regelmaessiger persoenlicher treffen keine stand-up-meeting, somit mangelndes wissen über probleme und fortschritte der gruppenmitglieder
- 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 visuell hervorgehoben, sodass für Nutzer nicht offensichtlich ist welcher Taster welche Funktionsbelegung hat.
Zur weiteren Verschönerung der Bandlaufwerke sollten die fehlenden Tasterfronten 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. interessierte 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 Programmierneulinge bietet das einen niederschwelligen Einstieg in eingebettete Systeme.
Da regelmäßige manuelle Updates der vorhandenen Spiele auf der Plattform 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.