Reaktionsspiel mit Bandgeräten

Softwareprojekt: Internetkommunikation

Steve Dierker, Patrick Hjort, Semjon Kerner

16. Oktober 2017

1 Einführung

Ziel dieses Projektes ist es, die PDP11 im Foyer des Informatikinstituts der FU Berlin zu präsentieren. Dazu verschmelzen wir es mit einem modernen Mikrocontroller und geben ihm die Kontrolle über das vorhandene Bandlaufwerk im PDP11. Mit ein paar zusätzlichen LEDs und LCD Zucker darauf sind wir in der Lage, eine einfache Gaming-Plattform für speziell entwickelte Spiele zu bauen. Ziel dieses Projektes ist es, die Hardware fertig zu stellen und ein Beispielspiel zu implementieren, bei dem zwei Spieler in der Reaktionszeit gegeneinander antreten können.

2 Retro11

Dies ist die C-basierte RIOT-Anwendung zur Steuerung des vorhandenen PDP11-Bandlaufwerks, einer zusätzlichen LCD-Anzeige und LEDs. Neben der Kontrolle über die vorhandene Hardware wird diese Anwendung mit einem einfachen Spiel ausgeliefert, um in der Reaktionszeit zu konkurrieren, was 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.

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.

Was das Anwendungsdesign selbst betrifft, initialisieren wir zunächst alle unsere 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 passieren, nach dem der Motor gestoppt werden soll. Der zweite Thread ist der CoapServer, der die Kommunikation zwischen den einzelnen 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 Hardware

Ein paar einleitende Sätze

1. samr21 Xpro
2. Multiplexer S74LS151 und Buttons
3. LED Stripe WS2811 und WS2812B
4. LCD DOGL128B-6
5. L298H-Modul
6. ATX Netzteil
    (a) Spannungen

Retro11 ist eine C-basierte RIOT-Anwendung, die auf dem Mikrocontroller SAMR21 XPro läuft. Zunächst haben wir mit der Entwicklung der Hardwareplattform und dem Schreiben von Treibern für jedes Peripheriegerät begonnen, 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 Einleiter-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 aus der Box heraus gearbeitet hat und das zweite ist ein Set von separaten Chips und LEDs, die Sie selbst zusammenbauen können. Der zweite Kit weigerte sich, mit unserem Treiber zu arbeiten und nach mehrtägigen Debugging-Tests stellte sich heraus, dass der Controller eine gemeinsame Annode benötigt, aber die LEDs, die wir erhielten, boten eine gemeinsame Kathode. Leider war der Shop, bei dem wir die LEDs gekauft haben, nicht in der Lage, die richtigen LEDs rechtzeitig zur Verfügung zu stellen, um das Projekt komplett abzuschließen.

3.1 Treiber

4 Netzwerkkommunikation

Die Kommunikation zwischen den Plattformen erfolgt über das RESTful Constrained Application Protocol, CoAP, das Nachrichten über UDP und IPv6 sendet, wie in Abbildung ?? . 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.

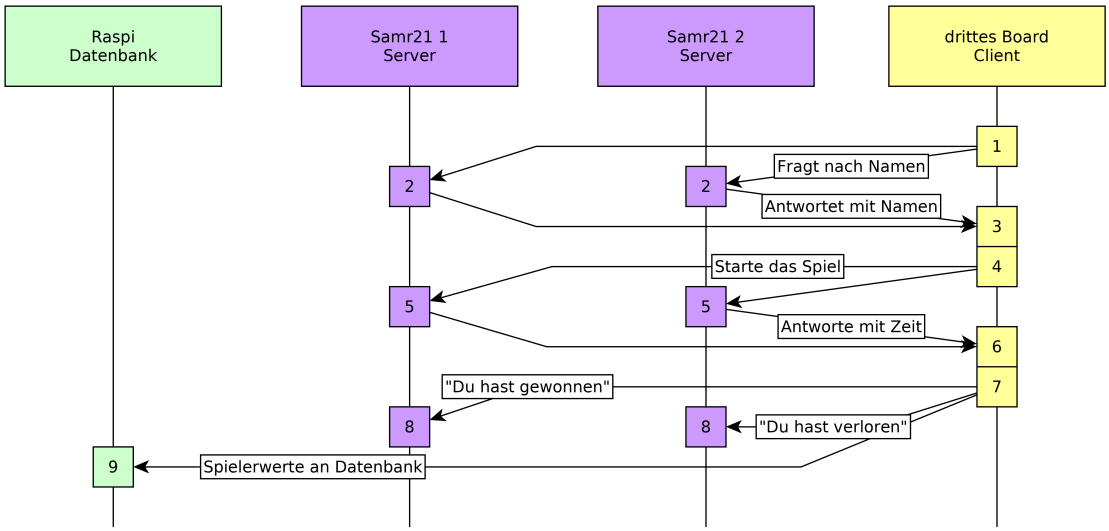

Abbildung 1: Sequenzdiagramm der Anwendung

5 Stand der Umsetzung

- Probleme
- Rückblick auf Zeitplan

6 Weiterführende Arbeit

- Spiele
  1. LUA-API
  2. mehr Spiele
  3. OTA
- Buttons
  1. LEDs
  2. 3D-Druck Gehäuse
  3. IRQ statt Polling
- Motoren reinigen