Pflichtenheft Reaktionsspiel

Embedded Systems SMSB6300

Konzeption der Zustandsmaschine(n)

Autor:

Johannes Schumacher

Matrikelnr.:

16452

Datum: 03.02.2020

Version: 0.1

Inhalt

1 Allgemeines

1.1 Dokument

Dieses Pflichtenheft beschreibt die Umsetzung des Reaktionsspiels des Moduls Embedded Systems für den Studiengang SMSB nach dem Dokument „Anforderungen Reaktionsspiel“ von Prof. Creutzburg.

1.2 Projektbezug

In Bezug auf das Projekt und den Anforderungskatalog, wird hier speziell die Konzeption, Generierung und Umsetzung möglicher Zustandsmaschinen beschrieben, die zur strukturierten Umsetzung des Reaktionsspiels betrachtet werden können. Es ist als Teilprojekt des Reaktionsspiels zu sehen, wessen Grundlage die Liste der zu implementierenden Features aus dem Anforderungsdokument ist.

Im folgenden Verlauf werden die wesentlichen Kriterien des Anforderungsdokuments zur Umsetzung der Zustandsmaschinen herausgearbeitet.

1.3 Ziel und Idee

Dieses Dokument beschreibt die Konzeption, Generierung und Umsetzung von Zustandsmaschinen auf dem TI MSP430 Launch Pad. Es beschreibt eine mögliche Umsetzung des Reaktionsspiels im Rahmen des Moduls Embedded Systems und dient als Lernerfahrung des Studenten im praktischen Umsetzen von Zustandmaschinen nach einem strukurierten Entwurfsmuster. Hierzu gehört die Planung und Konzeption mit anschließender Generierung des C-Programmcodes mit dem FSM Generator von Texas Instruments. Zuletzt folgt eine Testweise Implementierung einer Zustandmaschine auf dem MSP430, wobei die Funktionsfähikeit durch das Verfolgen der Zustände nachvollzogen werden kann.

2 Features

2.1 Musskriterien

Zum Spielablauf:

- Die Spieler geben ihre Namen ein
- Die Spieler haben verschiedene Spielmodi zur Auswahl
- Die Spieler können ein ausgewähltes Reaktionsspiel spielen

Zum Programm:

- Das Programm arbeitet mit einem Parametersatz
- Der Parametersatz enthätt verschiedene Daten wie Spielernamen, gewählter Spielmodi, statistsiche Aussagen, Betriebsspannungen
- Der Parametersatz wird nach Beenden des Spiels(Spannungsabfall) persistent auf dem Flash abgelegt
- Zur Laufzeit wird mit einer Kopie des Parametersatzes im RAM gearbeitet

Verschiedenes:

- Einhaltung von Programmierstandards (SEI CERT C Coding Standard, MISRA-C Regeln)
- Quellcodedoku (Doxygen)
- Modularisierung (Header/C-Dateien)
- Verwendung des Excel-Code-Generators für State-Maschines

- Versionsverwaltung (Git)
- Git Remote Host mit Gitlab oder Github
- Abhängige Dateien wie lokale Buildabhängigkeiten werden nicht versioniert (gitignore)

2.2 Wunschkriterien

Zum Spielablauf:

- Die Spieler können sich wiederholt duellieren ohne auf redundante Eingabenaufforderung zu stoßen
- Es wird im wesentlichen auf eine intuitive Benutzbarkeit wertgelegt

2.3 Abgrenzungskriterien

- Spieleranzahl: 2
- Die aufgelisteten Anforderungen sind eine Abstraktion um relevante Kriterien für die Entwicklung der State-Maschines (als Teilprojekt) hervorzuheben
- Fokus ist das Entwickeln der State-Maschines
- Es findet keine Implementierung der vollständigen Spiellogik statt

3 Einsatz

3.1 Anwendungsbereiche

Zwei spiellustige Menschen können sich jederzeit in dem Reaktionspiel duellieren, wobei es um schnelles Erkennen und Reagiern geht.

3.2 Zielgruppe

Jeder kann das Spiel spielen.

3.3 Betriebsbedingungen

Zum Spielen benötigt wird ein MSP430 mit erweiterter Hardware-Plattform (Taster & LEDs). Des weiteren wird zur Kommunikation mit den Spielern ein Host-Rechner benötigt sowie ein Terminal-Programm.

4 Projektumgebung

Die Lösung ist auf dem MSP-EXP430G2 Lanuch Pad lauffähig. Das Programm ist über eine serielle Schnittstelle mit einem Host-Rechner installierbar und auf dem MSP430 ausführbar.

5 Spielfunktionen

5.1 /F10/ Setzen der Spieler und Wahl des Spielmodus

- /F11/ Eingabe Speiler 1: Benutzereingabe über UART-Schnittstelle und Speicherung im Parametersatz (RAM)
- /F12/ Eingabe Speiler 2: Benutzereingabe über UART-Schnittstelle und Speicherung im Parametersatz (RAM)
- /F13/ Spielmodus wählen über UART-Schnittstelle und Speicherung im Parametersatz (RAM)

5.2 /F20/ Spielablauf

- /F21/ Bereitschaftscheck: Beide Spieler bestätigen durch einen Ready-Check
- /F22/ Countdown: Das Spiel zählt einen Countdown herrunter (Darstellung mit LEDs)

- /F23/ Das Spiel: Die Spieler spielen eine gegebene Rundenzahl
- /F24/ Spielende: Speichere Statistiken in Parametersatz im RAM

5.3 /F30/ Beenden
Das Programm speichert den Parametersatz im Flash und erreicht einen Anfangszustand

6 Nicht-funktionale Anforderungen

6.1 Erweiterbarkeit
Die Zustandsmaschinen können zur vollständigen Implementierung des Reaktionsspiels herangezogen werden und spezialisiert werden. Neue entwicklete Spielmodi können hinzugefügt werden.

6.2 Programmiersprache
Es wird mit der Programmiersprache C und den MSP430 Libraries unter Einhaltung der Programmierstandards gearbeitet.

6.3 Programmierumgebung
Es wird mit Code Composer Studio als Entwicklungsumgebung programmiert.

6.4 Modifizierung
Das Projekt wird zur Modifizieren und Anpassung zur Verfügung gestellt und Remote gehostet.

7 Spieldaten

7.1 /D10/ Benutzerdaten
Gemäß /F11/ und /F12/ geben die Benutzer ihre Spielnamen ein:

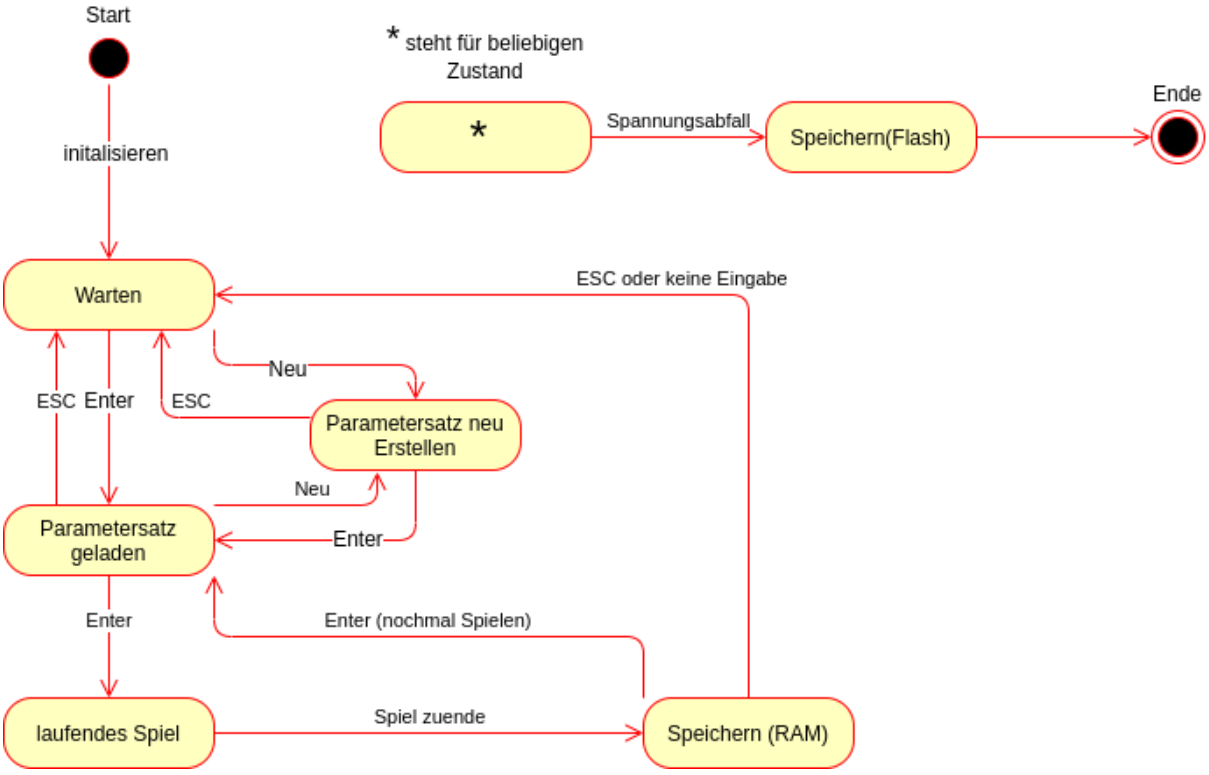
- /D11/ Spielname 1
- /D12/ Spielname 2

7.2 /D20/ Statistiken
Mit dem Spielende (Funktion /F24/) werden Statistiken gespeichert wie Spielstand, durschnittliche Reaktionszeit und Anzahl Gewinner links/rechts.

8 Zustandsmaschinen

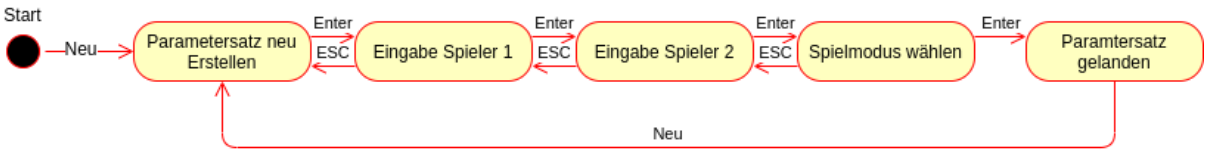
8.1 Controller-Hauptzustandsmaschine

Es folgt erste Konzeption der Hauptzustandsmaschine, die im laufe der Entwicklung vervollständigt und angepasst wird bzw. die Events mit Funktionsaufrufen verknüpft werden.



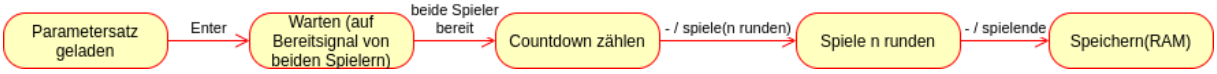
8.2 Datensatzauswahl und Spielereingaben

Ausgehend vom Event *Neu* und dem Zustand *Parametersatz neu Erstellen* (siehe Hauptzustandsmaschine) kann das Neuerstellen des Parametersatzen zustandmaschinenbasiert passieren.



8.3 Spielablauf

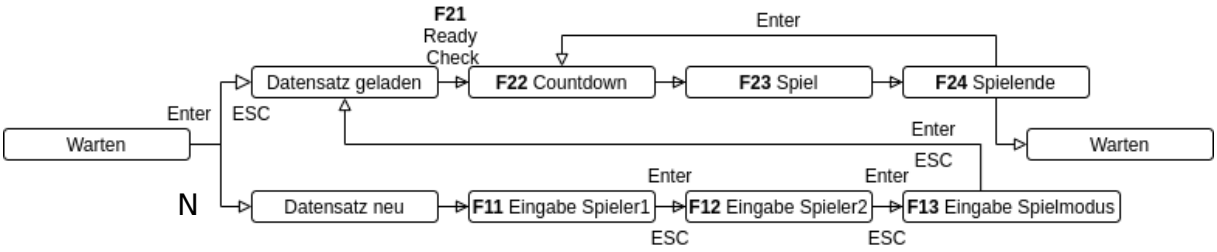
Auch der Spielfluss kann zustandsbasiert konkretisiert werden. Ausgangspunkt ist der Zustand *Parametersatz geladen* (siehe Hauptzustandsmaschine).



9 Benutzerinteraktion

9.1 Dialogstruktur

Im Folgenden wird die Dialogstruktur einer fehlerfreien bzw. konfliktfreien Benutzung des Spiels gezeigt. Fehlereingaben haben im Zweifel einen Rücksprung auf den Anfangszustand zur Folge. Mit **F??** wird auf die Spielfunktionen (aus Kapitel 5) referenziert



9.2 Benutzerinput

Die benutzer navigieren in der Regel mit **Enter** (Vor)und **ESC** (Zurück) durch die Dialogstruktur. Ausnahme ist das setzen eines neues Datensatzes, das mit **N** für Neu/New passiert.

10 Qualitätsanforderungen

Auf folgende Qualitätsstandards wird besonders Wert gelegt:

	sehr wichtig	*wichtig*	*weniger wichtig*	*unwichtig*
Zuverlässigkeit	×			
Geringe Komplexität	×			
Ressourceneffizienz		×		
Benutzerfreundlich			×	

11 Testszenarien

Grundsätzlich folgt auf jede funktionale Anforderung ***/F??/*** ein konkreter Testfall ***/T??/*** . Es folgt ein konkretes Testszenario des fehlerfreien Spielablaufs:

- /T11/ Eingabe Spieler 1: Der Testspieler 1 wählt mit N wie neu einen neuen Datensatz aus. Er gibt seinen Namen ein und bestätigt mit Enter.
- /T12/ Eingabe Spieler 2: Der Testspieler 2 gibt ebenso seinen Spielnamen an und bestätigt mit Enter.
- /T13/ Eingabe Spielmodi: Ein Testspieler wählt mit einer Zahl einen Spielmodus aus einer gegebenen Spielliste mit der korrespondierenden Listenzahl als Spiels. Die Eingabe wird geprüft.

- /T21/ Bereitschaftscheck: Die Testspieler bestätigen ihre Bereitschafft mit einem Knopfdruck auf jeweils einen ihrer Taster.
- /T22/ Countdown: Das Spiel zählt von selbst einen Countdown herunter, in Form von ausgehenden LEDs.
- /T23/ Das Spiel: Testspieler 1&2 spielen das Reaktionsspiel n Runden.

- /T24/ Spielende: Durch keine Eingabe oder ESC geht das Spiel in den Anfangszustand.

- /T30/ Beenden: Ein Testbenutzer löst einen Spannungsabbruch aus.