Abschlussprojekt

42. Laboratory

Autor: Roland Lezuo, Thomas Gadner
Last change: 5. Juni 2025

Battleship (game)

Das Projekt besteht aus mehreren Teilaufgaben. Ziel des Projektes ist es, ein Programm zu schreiben, das *Schiffe versenken* gegen einen PC implementiert. Dazu muss ein serielles Protokoll (UART) implementiert werden. Dieses Protokoll ist in diesem Dokument beschrieben. Für den PC gibt es eine Referenzimplementierung in Python. Dein Programm spielt gegen dieses Python-Programm. Das Spielfeld ist wie in Abb. 1 ist 10 mal 10 Felder groß. Die Tabelle 1 beschreibt, welche Schiffe auf dem Spielfeld stehen:

Anzhal der Schiffe	Klasse	Größe
1	Schlachtschiff	5
2	Kreuzer	4
3	Zerstörer	3
4	U-Bote	2
Summe		30

Tabelle 1 – Schiffe

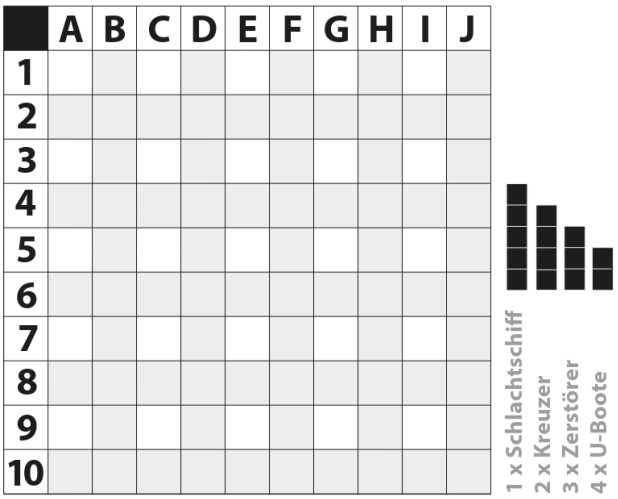

Abb. 1 – Spielfeld eines traditionellen Schiffsversenkspiels

Die Schiffe dürfen sich nicht berühren, auch nicht diagonal.

Kommunikation

Für den Spielablauf werden Nachrichten zwischen dem Host (dem PC, auf dem `schiff2.py` läuft) und dem Device (deinem STM32-Board) ausgetauscht. Der prinzipielle Ablauf der Kommunikation ist im Anhang dargestellt. Die ausgetauschten Nachrichten werden im ASCII-Format mit 115200 Baud, 8N1 übertragen. Jede Nachricht endet mit dem ASCII-Zeichen *newline* (`\n` in C), *carriage-return* wird ignoriert. Die Nachrichten haben das Präfix `HD_`, wenn sie vom Host zum Device gesendet werden, und `DH_`, wenn sie vom Device zum Host gesendet werden. Die Kommunikation erfolgt in 3 Phasen:

- Initialisierung
- Beschuss
- Gewinn oder Verlustmeldung

Device und Host tauschen danach Prüfsummen (Checksum) über das Spielfeld aus. Dies soll verhindern dass ein Programm schummelt. Die Prüfsumme berechnet sich aus den Summe der Schiffteile in jeder Zeile des Spielfelds. Die nullte Ziffer ist die nullte Zeile, usw. Während der Initialisierungsphase sucht der Host (PC) nach einem Device, gegen das er spielen kann. Dazu sendet er regelmäßig die Nachricht `DH_START` und wartet, bis ein Device mit `DH_START_{name}` antwortet. Für `{name}` ist der Name im ASCII-Format anzugeben, z.B. `DH_START_ROLAND`.Device und Host tauschen dann Prüfsummen (Checksum) über das Spielfeld aus. Dies soll verhindern, dass ein Programm betrügt. Die Prüfsumme errechnet sich aus der Summe der Schiffsteile in jeder Zeile des Spielfeldes. Die nullte Zahl ist die nullte Zeile usw.

Dann beginnt der Host zu schießen. Dazu sendet er die Nachricht `HD_BOOM_x_y`, wobei x und y für die Zeile (von oben) und y für die Spalte (von links) stehen. Die Werte für x und y liegen zwischen 0 und 9. Das Device antwortet nun mit `DH_BOOM_H` (für hit) bzw. `DH_BOOM_M` (für miss), ob es sich um einen Treffer oder ein Wasserfeld handelt. Unmittelbar danach schießt das Device (mittels `DH_BOOM_x_y`) und der Host antwortet mittels `HD_BOOM_H` bzw. `HD_BOOM_M`. Diese Schritte werden wiederholt, bis alle Schiffsteile von einem Spieler getroffen wurden. Wird ein Feld mehrmals beschossen, so wird beim zweiten Mal nicht getroffen (das Schiffsteil ist bereits versenkt). Die Gewinn/Verlustphase beginnt, wenn ein Spieler keine Schiffsteile mehr hat.

Wird das letzte Teil getroffen, antwortet man nicht mit einem `XX_BOOM_[HM]`, sondern gibt das eigene Spielfeld ab. Das Format ist `XX_SF{Zeile}D{10 Ziffern}`, wobei {Zeile} für die Zeilennummer des Spielbretts (0-9) steht und die 10 Ziffern von links nach rechts die Spalten der Zeile beschreiben. Die Ziffer 0 steht für ein Wasserfeld, die Ziffern 2-5 für ein Schiffsteil, wobei die Größe des Schiffes angegeben wird. Anschließend übergibt der Mitspieler sein Spielfeld. Jeder Spieler prüft nun das Spielfeld gegen die Prüfsumme und das eigene Schussprotokoll, ob der Gegner geschummelt hat. Tut man dies nicht oder prüft nicht gründlich, kann der Gegner mit Schummeln durchkommen, so ist das Leben. Das Spiel ist nun beendet und entweder Sieg, Niederlage oder nicht gewertet, weil geschummelt wurde und der Host beginnt wieder mit der Initialisierungsphase.

Das Spiel kann jederzeit vorzeitig abgebrochen werden:

- Ein Spieler antwortet nicht schnell genug (1 Sekunde), wenn er eine Nachricht senden muss.
- Ein Spieler schickt eine falsche Nachricht
- Ein Spieler sendet eine völlig ungültige Nachricht (z.B. `HD_ALLES_IST_FOOBAR`)

Das Spiel wird als abgebrochen gewertet, der Spieler, der den Fehler gemacht hat, ist der Verlierer.

In der folgenden Abb. 2 ist der Kommunikationsablauf noch einmal grafisch dargestellt.

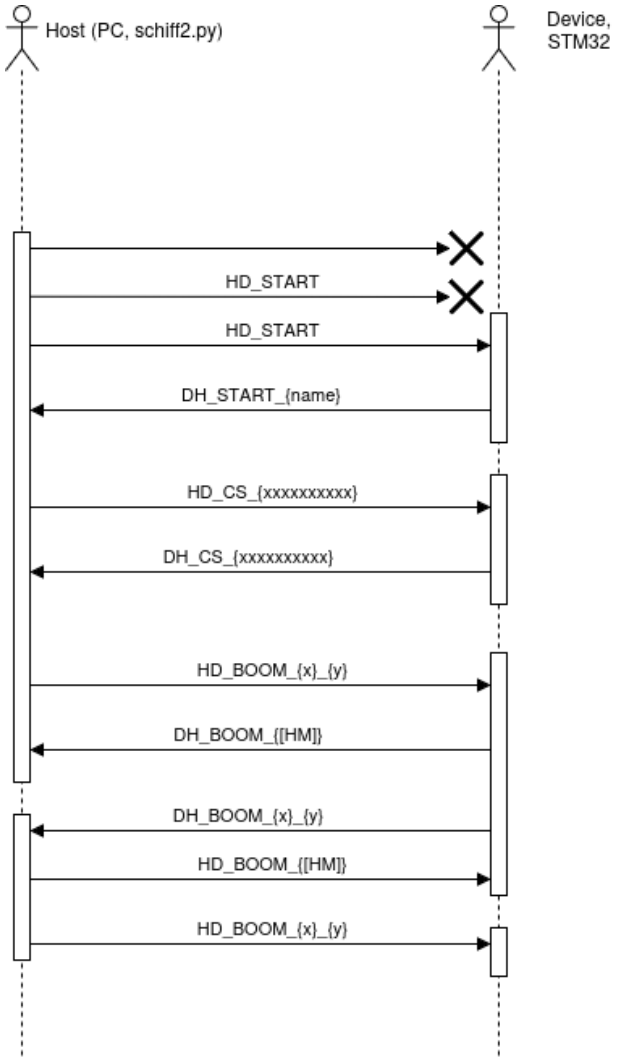

Abb. 2 – Ablauf der Kommunikation

Referenzimplementierung

Achtung: schiff2.py ist neu und kann noch Bugs enthalten, unklares Verhalten bitte mit den Betreuern besprechen! Eventuell könnt ihr einen issue auf github eröffnen. Die Referenzimplementierung findet ihr im folgenden Repository: MECH-B-4-ILV-Embedded-Systems-Project auf GitHub

Teilaufgaben

Task 1 Staying above the Water (60 Punkte)

Implementiere ein Programm, das im Wesentlichen funktioniert. Es sollte in der Lage sein, ein (1) Spiel gegen die Referenzimplementierung zu spielen. In diesem Schritt kann dein Programm noch ein hart codiertes Spielfeld haben und muss keine intelligente Schussstrategie implementieren. Implementiere nacheinander die folgenden Funktionen:

Task: 5/100 Punkte

erkennen einer HD_START Nachricht

Task: 5/100 Punkte

senden einer DH_START_ Nachricht

Task: 10/100 Punkte

HD_CS Nachricht empfangen und prüfen

Task: 10/100 Punkte

DH_CS Nachricht für dein Spielfeld verschicken

Task: 10/100 Punkte

HD_BOOM_x_y wird verstanden und mit DH_BOOM_[HM] beantwortet

Task: 10/100 Punkte

DH_BOOM_x_y wird erzeugt und HD_BOOM_[HM] wird verstanden

Task: 10/100 Punkte

Gewinn / Niederlage und HD_SF bzw. DH_SF Nachrichten implementiert

Task 2 Sailing before the Wind (29 Punkte)

In diesem Schritt erweiterst du dein Programm, bis es voll funktionsfähig ist. Erstelle vor jedem Spiel ein zufälliges Spielfeld. Platziere deine Schiffe automatisch entsprechend den Regeln. Schieße klug auf deine Gegner, z.B. wenn du ein Schiff getroffen hast, versenke es (hunter&kill), schieße nicht auf die Felder neben den Schiffen (das ist Wasser). Mache dein Programm robust.

Task: 10/100 Punkte

Zufällig erzeugtes Spielfeld

Task: 10/100 Punkte

Kluge Schussstrategie, je klüger und komplexer desto mehr Punkte

Task: 9/100 Punkte

Spiele 100 schnelle Spiele gegen (*schiff2.py -t*) und gewinne mindestens ein Spiel

Task 3 Whatever floats your Boat (6-11 Punkte)

Mache aus deinem Programm ein sehr gutes Programm, indem du die Module verwendest, z.B. den Lautsprecher für die Schüsse, die LED für die Datenkommunikation, das 10x10 LED Matrix Modul für das 10x10 Spielfeld... Die Details sind dir überlassen.

Task: 5/100 Punkte

Lautsprecher oder LED

Task: 6/100 Punkte

LED Matrix Modul

Report

Es wird ein Abschlussgespräch geben, bei dem dein Programm gegen den PC gespielt wird und ein kurzes Interview über deinen Code. Das Verständnis wird getestet.