
Der Flipper

Gruppenmitglieder: Luca Perri, Luis Schäfer
Fachlehrer: Herr Reiter
Fach: NWT

Inhaltsverzeichnis

I. Themenfindung und Hypothese

1. Themenfindung	Seite 1
2. Hypothese	Seite 1

II. Die Theorie hinter dem Flipper

1. Multitasking bei einem Computer	Seite 2
2. Wire.h Kommunikation	Seite 2/3
2.1 Blocking Functions	Seite 3
2.2 Das Design eines Flippers	Seite 4
2.3 Die Ballschleuder	Seite 4
2.4 Der Flipper-Finger	Seite 4
2.5 Der Münzzähler	Seite 5
3. Technik und Handwerk vereint; Wie es funktionieren sollte	Seite 5/6
3.1 Testen der Elektrik und Software	Seite 7

III. Fazit und Bezug auf die Hypothese

1. Fazit: Ist Multithreading/Multitasking mit mehreren Arduinos möglich?	Seite 8
---	---------

III. Quellen und Anhang

1. Quellen	Seite 9
2. Eigenständigkeitserklärung	Seite 9

I. Themenfindung und Hypothese

1. Themenfindung:

Anfangs hatten wir eine ganz andere Idee im Kopf, denn eigentlich wollten wir im Rahmen des Energiehauses etwas machen. Unsere ersten Ideen waren, aus dem Energiehaus ein Smart Home zu machen. Jedoch bemerkten wir sehr schnell, dass diese Idee im Rahmen einer Facharbeit wohl kaum umzusetzen war, da der handwerkliche Teil kaum vorhanden gewesen wäre. Daher verwarfen wir unsere erste Idee und entschieden uns dazu, einen Flipper zu bauen. Hier wäre der handwerkliche Teil und der technische Teil deutlich ausgewogener, sodass beide Teammitglieder ihre Stärken voll ausschöpfen können.

2. Hypothese:

Unsere Hypothese, die wir über das ganze Projekt verfolgten, bezog sich auf eine Eigenschaft des Arduinos: Der Arduino kann nämlich nur eine Sache gleichzeitig machen.

Daher stellten wir die Hypothese auf, dass es doch möglich sein müsste, durch das Verbinden mehrerer Arduinos miteinander, ein System zu entwickeln, dass mehrere Sachen gleichzeitig machen kann. Dies ist nämlich ein unabdingbarer Punkt wenn man einen Spielautomaten wie einen Flipper bauen möchte.

Somit lautete auch unsere Leitfrage:

Können zwei oder mehr Arduinos eine Multitaskingaufgabe bewältigen?

II. Die Theorie hinter dem Flipper

1. Multitasking bei einem Computer:

Um zu verstehen wie der Flipper gebaut werden muss, mussten wir zuerst verstehen wie Computer überhaupt Multitasking, fachlich ausgedrückt auch Multithreading, betreiben.

Ein Artikel von [javatpoint](#) verdeutlichte dabei die Wichtigkeit der Synchronisation. Das bedeutet, wenn wir wollen, dass die Arduinos miteinander arbeiten, müssen wir in irgendeiner Art eine Synchronisation zwischen beiden herstellen.

Das bewältigten wir damit, dass einige Variablen und Abläufe von beiden Arduinos verwendet werden. Im Detail sind damit die Prozesse im *void loop()* und die *GAMESTATUS* Variable gemeint. Dabei besitzen beide Arduinos eine Phase vor dem Spielbeginn und eine Phase nach dem Spielbeginn, welche durch den *GAMESTATUS* gesteuert werden.

2. Wire.h Kommunikation:

Natürlich reichen die obigen Maßnahmen nicht aus, um die Arduinos zu synchronisieren. Die Arduinos müssen miteinander Informationen austauschen, zum Beispiel ob ein Spiel starten muss, oder ob der Ball bereits im Gameover-Bereich ist.

Um dies zu erreichen, verwenden wir Arduino-Standardbibliothek¹ *Wire.h*. Dabei werden die Arduinos miteinander Verbunden, indem man ihren I2C-Bus verbindet. Damit die Arduinos miteinander kommunizieren können, braucht einer von ihnen eine Adresse, die in der *begin()* Funktion angegeben wird. Der andere Arduino erhält keine Adresse. Dieser Arduino ist der sogenannte **Master**, der Arduino mit Adresse der **Slave**.

Der Master sendet dabei Befehle in Form von Bytes oder Byte-Arrays². Der Slave liest diese dann aus und führt den Befehl aus.

Unsere Idee hierbei war, dass der Master kontrolliert, wie viel Geld eingeworfen wurde und ob der Startknopf gedrückt worden ist.³ Dann sendet er einen Befehl an den Slave, der dafür sorgt, dass das LCD-Display korrekt neu geladen wird. Zu dem sendet der Master weitere Informationen, wie die Menge an gezahlten Spielen. Dafür sendeten wir in sogenannte Datenpakete⁴. Diese enthalten eine Reihe an Daten, die von unserem Slave ausgelesen und zugeordnet werden.

2.1 Blocking Functions:

Das Konzept von Blocking Functions haben wir bei unserem ersten Test kennengelernt. Blocking Functions sind Funktionen (*voids*), die verhindern, dass der Arduino andere Signale, wie z.B. ein I2C-Paket⁵, verwertet.

Als Blocking Functions gelten *delay()*, aber auch *while*- und *for*-Loops. Dies machte sich bemerkbar, da das Spiel vom Slave nie gestartet wurde, da viele unserer Abläufe für das LCD-Display sowohl Delays, als auch Loops verwendeten.

Das bedeutete, dass wir fast immer unser gesendetes Start/Stop-Paket verpassen würden. Um dieses Problem zu beheben, entfernten wir alle Delays aus unserem Code und ersetzen sie, wenn sie unabdingbar waren, durch *while*-Loops, welche die Zeit vergangene Zeit in ihrer Bedingung prüften. In den *while*-Loops überprüften wir konstant, ob der Pin 13 auf *HIGH* ist. Dies war unser (erster) Sicherheitspin, der vom Master je nach Spielphase aktualisiert wird.

Ist der Pin *HIGH*, ist das Spiel zu Ende. Das bedeutet, wenn der Slave nun diese Veränderung misst, löst er sofort sein Gameover-Protokoll aus und verlässt den Loop.

2.2 Das Design eines Flippers:

Natürlich mussten wir uns auch damit auseinandersetzen, wie denn ein Flipper den Aussehen hat. Luis sein Onkel besitzt einige Flipper bei sich zu Hause, wodurch wir das generelle Design einfach ausmessen und uns ansehen konnten. Wir maßen dabei die Steigung der Spielplatte und stellten fest, dass sie eine Steigung von $4,7^\circ$ besitzt. Danach wählten legten wir fest, wie Groß der Flipper ungefähr sein musste.

2.3 Die Ballschleuder ([Plunger](#) 19.06.2023):

Nun, da wir ein Spielfeld haben, mussten wir herausfinden wie wir den Ball ins Spiel bekommen. Die Flipper von Luis' Onkel benutzen alle Magneten um den Ball in das Spielfeld zu schleudern. Das stellte für uns ein Problem, da wir nicht so einfach an starke Magneten kommen würden, mit denen man solch eine Kugel in unser Spielfeld befördern kann.

Als Lösung fanden wir die Idee, dass der Spieler den Ball selber mit einer Schleuder (die eine Feder als Energiegeber nutzt) ins Spielfeld schleudern darf, eine gute Ergänzung.

2.4 Der Flipper-Finger ([Elektormechanisch](#) 19.06.2023):

Auch die Flipper-Finger bereiteten uns Probleme, da die meisten modernen Flipper starke und schnelle Motoren verwenden, um diese zu bewegen. Diese Motoren sind bei weitem nicht ersetzbar durch die DC-Motoren, Servos oder Schrittmotoren an unserer Schule. Ein Motor dieser Klasse würde uns eine [menge Geld](#) kosten.

Daher entschieden wir uns für die altmodische Weise des Flipperbauens und benutzten einen mechanischen Weg. Dieser hat sogar den Vorteil, dass der Spieler selber entscheiden kann, wie stark der Ball fliegen soll.

2.5 Der Münzzähler:

Für den Münzzähler haben wir uns auch etwas einfaches überlegt und einen Zähler wiederverwendet, denn wir bereits in einem früheren Lego-Mindstorm Projekt verbaut haben. Es ist ein mechanischer Münzsortierer, der sich das Attribut der Münzgröße zum Vorteil macht und die Münzen nach Größe sortiert. Des weiteren benötigt die Münze ein gewisses Gewicht um genug beschleunigt zu werden und nicht hängen zu bleiben. Zwei IR-Sensoren erkennen dann, ob eine Münze einfällt oder nicht.

3. Technik und Handwerk vereint; Wie es funktionieren sollte:

Diese Sektion bezieht sich stark auf Codefragmente und den Schaltplan. Zu finden ist alles [hier](#)
Stand der Beschreibung: 19.06.2023 (Letzte Änderung)

Die Funktion des Flippers ist auf zwei Phasen aufgebaut:

Phase 1 ist die Phase, wo der Flipper darauf wartet, dass ein Spiel beginnt. Im Master ist diese Phase in der Funktion *check_start_init()* gegliedert, der Slave gliedert diese Phase in der *while*-Schleife im *void loop*.

Eigentlich war auch zu diesem Zeitpunkt geplant, dass ein weiterer Slave-Arduino die Musik und eine Lichtshow steuert, jedoch konnte leider kein Lautsprecher der eine angemessene Preis-Leistung besitzt, geliefert werden.

Der Master wartet darauf, dass ein Spieler Geld einwirft, indem er alle 10 Millisekunden die zwei im Münzzähler eingebauten IR-Sensoren abprüft. Währenddessen steuert der Slave-Arduino das LCD-Display, dass dem Spieler sagt was er zu tun hat um ein Spiel zu starten. Gleichzeitig überprüft er den *PIN 3* um sicherzustellen, dass kein Spiel gestartet wurde und das *Packet* verpasst wurde. Wird nun erkannt, dass eine Münze eingeworfen wurde, oder dass

noch ein Spiel aus der vorherigen Zahlung aussteht, wartet der Master 2 Sekunden, bis er den Servo, der den Ball in die richtige Position zu schiebt, ansteuert. Danach führt er die Funktion *transmit_aryInst()* aus, um dem Slave über den Spielstart zu informieren und ihm ggf. die übrigen Spiele für die Anzeige auf dem LCD-Display zu geben. Zugleich wird der *PIN 13*, der mit dem *PIN3* des Slaves verbunden ist, auf *LOW* gestellt, um zu signalisieren, dass ein Spiel gestartet ist. Dieser Pin bleibt *LOW*, bis der Spieler Game Over geht.

Der Slave, der nun entweder das *Packet* erhalten hat, oder den Wert des *PIN 3* ausgelesen hat, verlässt seinen momentanen *LCD-Status* und startet nun seine Spielsequenz.

Nun gehen beide Arduinos zu Phase 2 über, der Spielsequenz.

Der Spieler hat nun die Möglichkeit, den Ball aus der Startposition herauszuschleudern und mit den Flipperfingern den Ball im Spielfeld zu halten.

Währenddessen wechselt der Slave-Arduino nun zwischen 3 verschiedenen Anzeigen auf dem LCD-Display; Wie lange das Spiel bereits andauert, das wievielte Spiel das bisher ist und (falls vorhanden) wie viele Spiele bezahlt wurden und noch ausstehen. Zuerst jedoch wird sichergestellt, dass alle Daten korrekt erhalten worden sind, indem der Slave den *PIN 6* auf *HIGH* stellt. Das signalisiert dem Master, dass er ein Datenpaket mit 3 *BYTES* senden soll:

[Spielstatus, Spielnummer, Übrige_Spiele]

Der Master liest in seiner Spielphase in der Funktion *check_game_over()* ständig den *PIN 6* aus und überprüft ob der Ball auf dem Game-Over-Feld landet.

Er sendet, falls dann nötig, das nötige *Packet* um das Spiel zu beenden, oder den Slave mit den richtigen Werten zu versorgen.

All dies geschieht ohne delay, sodass es keine Verzögerungen oder Unterbrechungen gibt.

Erkennt der Master nun, dass der Ball in den Game-Over-Bereich rollt, sendet dieser ein *Stop-Packet* und setzt den *PIN 13* auf *HIGH* um das Spiel zu beenden. Der Slave folgt dem Master kurz darauf und schreibt alle Daten des Spieles in eine SD-Karte und aktualisiert das LCD-Display.

3.1 Testen der Elektrik und Software:

Als nun endlich der Flipper und seine Technik fertiggestellt waren, wurde beides separat getestet. Die mechanischen Funktionen des Flippers funktionierten einwandfrei und waren somit fertiggestellt. Die Technik des Flippers benötigte an einigen Stellen jedoch ein kleines Finetuning, da z.B. eine Münze die eingeworfen wurde so schnell gezählt wurde, dass sie mehrere Male ausgelesen wurde. Auch die Anzeige für das LCD-Display hatte einen Fehler wodurch es flackerte, jedoch war auch dieser Fehler recht einfach zu beheben. Damit funktionierte der Flipper auch so, wie das geplant war. Nun mussten nur beide Teile, der Flipper an sich und die Technik verbunden werden und der Flipper wäre Einsatzbereit.

Geplante Fertigstellung des Flippers war angesetzt auf den 9. Juni, jedoch wurden wir durch den Diebstahl und das Umstecken unserer Technik massiv in den Rückstand geworfen worden. Zu dem Zeitpunkt wo dieser Absatz entsteht ist immer noch nicht sicher ob es zeitlich möglich sein wird, den Flipper bis zum Abgabetermin wieder auf Vordermann zu bringen.

III. Fazit und Bezug auf die Hypothese

1. Fazit: Ist Multithreading/Multitasking mit mehreren Arduinos möglich?:

Im Laufe des Projektes wurde immer deutlicher, dass unsere ursprüngliche Hypothese sich als korrekt herausstellen wird. Schon während Testläufen in der Entwicklungsphase, wo die beide Arduinos miteinander kommunizierten, stellte sich heraus, dass zwei oder mehr Arduinos definitiv dazu in der Lage sind Multitasking zu betreiben.

Während der Master sich darum kümmert, dass das Geld gezählt wird und der Ball im Auge behalten wird, kümmert sich der Slave um die Zeit des Spieles und die Anzeige auf dem LCD-Display. Auch der Musik-Arduino hat, als er noch eingeplant und eingebaut war, eine Lichtshow ausgeführt. Alle drei Aufgaben können nicht von einem Arduino selber ausgeführt werden, da das Timing unmöglich zu bewältigen sei.

Als Fazit lässt sich also letztendlich sagen, dass mehrere Arduinos unter korrekter Verwendung auf jeden Fall zu Multitasking Fähig sind.

III. Quellen und Anhang

1. Quellen:

<https://de.wikipedia.org/wiki/Flipperautomat>
(19.06.2023)

<https://www.javatpoint.com/benefits-of-multithreading> (19.06.23)

<https://www.pinball.center/de/shop/flipperteile/spielfeld-teile/motoren/?p=1>
(19.06.2023)

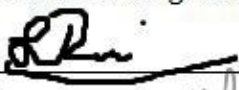
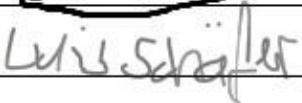
<https://www.techopedia.com/definition/24297/multithreading-computer-architecture>
(19.06.2023)

<https://www.digikey.de/en/maker/projects/multi-tasking-the-arduino-part-1/b23d9e65c4d342389d20cbd542c46a28#:~:text=The%20Arduino%20is%20a%20very,load%20and%20run%20multiple%20programs.>
(19.06.2023)

2. Eigenständigkeitserklärung:

EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit versichere ich, dass ich diese Arbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen sind als solche gekennzeichnet.

<u>19.06.2023</u>	<u></u>	Ort, Datum Unterschrift
<u>19.06.2023</u>	<u></u>	Ort, Datum Unterschrift