

Projektdokumentation

Gyro-Game

Gruppe 6:
Felix Neumann (215892),
Lukas Both (215828)

Hochschule Heilbronn
Fakultät: TE
Studiengang: Mechatronik und Robotik
Vorlesung: Elektronische Systeme
Sommer Semesters 2024
Betreuer: Prof. Dr. rer. nat. Tim Fischer,
Ralf Ziegler
Datum der Auswertung: 23.07.2024

1 Inhalt

2 Einleitung.....	3
3 Grundlagen Hardware	4
3.1 TFT-LCDs.....	4
3.2 MEMS-Beschleunigungssensoren.....	6
4 Grundlagen Software.....	8
4.1 Callback-Funktionen.....	8
4.2 Game Loop.....	10
4.3 Microchip Harmony.....	11
5 Methodik.....	12
6 Komponenten Auswahl.....	13
6.1 SAMD21	13
6.2 Beschleunigungssensor BMA400	15
6.3 Display Adafruit 1596	16
6.4 Display Treiber-Board RA8875	17
7 Umsetzung	18
7.1 Umsetzung Software	18
7.1.1 Allgemeines zur Programmierung.....	18
7.1.2 Beschleunigungssensor BMA400	18
7.1.3 Display	18
7.1.4 Game Loop	19
7.1.5 Kollisionserkennung	20
7.2 Umsetzung Hardware.....	22
7.2.1 Steckbrett Aufbau	22
7.2.2 Platine für Handheld Konsole.....	23
7.2.3 Power Supply	24
7.2.4 Power Supply SAMD21	25
7.2.5 Debug Schnittstelle.....	25
7.2.6 Reset-Taster	26
7.2.7 General Purpose LEDs.....	26
7.2.8 Beschleunigungssensor	27
7.2.9 Routing	28
8 Diskussion der Ergebnisse	29
9 Danksagung	30
10 Quellen	31

2 Einleitung

Im Rahmen der Vorlesung Elektronische-Systeme an der Hochschule Heilbronn soll ein frei wählbares Elektronik-Projekt umgesetzt werden. Das Projekt setzt sich aus der Entwicklung von Hardware sowie der dazugehörigen Software zusammen. Als Projekttidee haben wir uns für ein digitales Kugellabyrinth entschieden.

Die Kugel, sowie das Kugellabyrinth wird auf einem LCD dargestellt. Der Spieler hat die Aufgabe die digitale Kugel durch Neigen des Displays in das kariert markierte Ziel zu bewegen. Die Neigung des Displays soll mit einem 3-achsigen Beschleunigungssensor gemessen werden. Der Spielablauf, die Auswertung der Beschleunigungsdaten und die Darstellung des Displays sollen zentral mit einem 32-Bit Mikrocontroller umgesetzt werden.

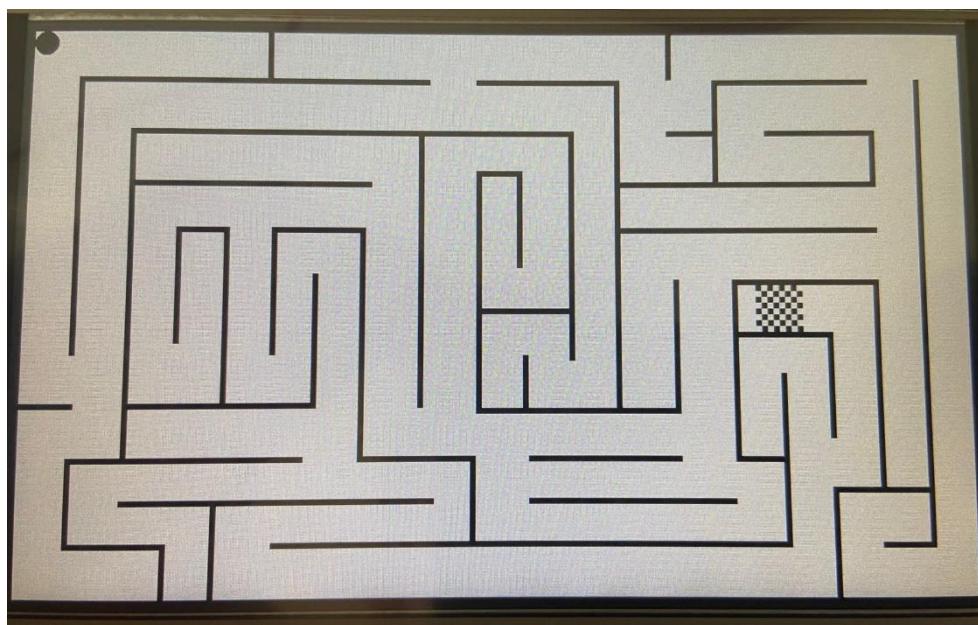


Abbildung 1 Digitales Kugellabyrinth

3 Grundlagen Hardware

3.1 TFT-LCDs

TFT-LCDs sind Flüssigkeitskristallbildschirme die über Dünnschichttransistoren, kurz TFT, angesteuert werden. Flüssigkeitskristalle haben die Eigenschaft das sie ihre Durchlässigkeit für polarisiertes Licht ändern, wenn eine elektrische Spannung angelegt wird.

TFT-Displays haben eine durchgängige weiße Hintergrundbeleuchtung. Ein Pixel eines TFT-Displays besteht aus drei Flüssigkeitskristallen. Durch die angelegte Spannung an den Flüssigkeitskristallen wird gesteuert wie viel der Hintergrundbeleuchtung beim Betrachter ankommt. Das Licht, das durch den Flüssigkeitskristall tritt, wird je nach Filter in rotes, grünes oder blaues Licht gefiltert. Beim Betrachter ergibt sich für den Pixel die Mischfarbe aus dem Grün-, Rot- und Blauanteil.

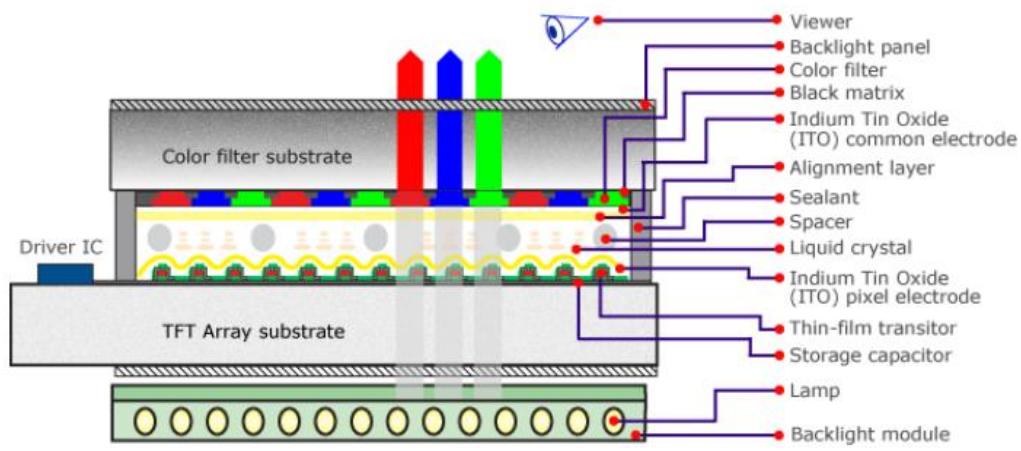


Abbildung 2 Aufbau TFT-Display

Die Spannung an den Flüssigkeitskristallen wird über Dünnschichttransistoren eingestellt, in Abbildung 3 sieht man das Ersatzschaltbild für einen Flüssigkeitskristall. CS ist ein Speicher kondensator, um das elektrische Feld am Flüssigkeitskristall aufrecht zu erhalten, wenn der Transistor nicht durchsteuert. Da sich der Speicher kondensator durch Kriechströme langsam entlädt müssen auch Pixel dich sich nicht regelmäßig neu angesteuert werden.

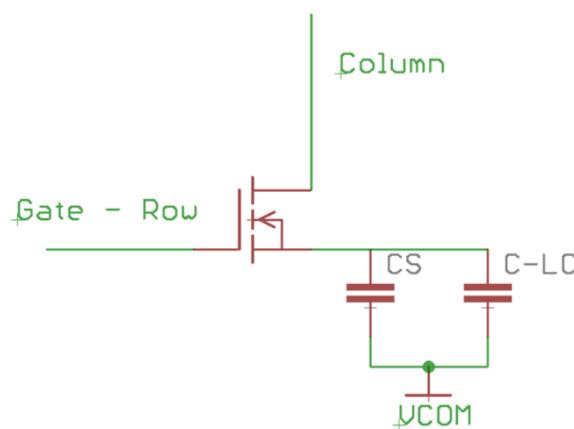


Abbildung 3 Ersatzschaltbild Flüssigkeitskristall

Elektronische Systeme

Jeder Pixel besteh aus drei Transistoren für die jeweils die Source- und Drain Spannung eingestellt werden muss. Dafür werden die Pixel im Bildschirm, wie in Abbildung 4 in einer Matrix angeordnet. In jeder Reihe sind die Gate-Anschlüsse miteinander verbunden und in jeder Spalte sind die Source-Anschlüsse miteinander verbunden. Um ein Transistor anzusteuern, muss zur selben Zeit die Reihe und die Spalte des Transistors angesteuert werden.

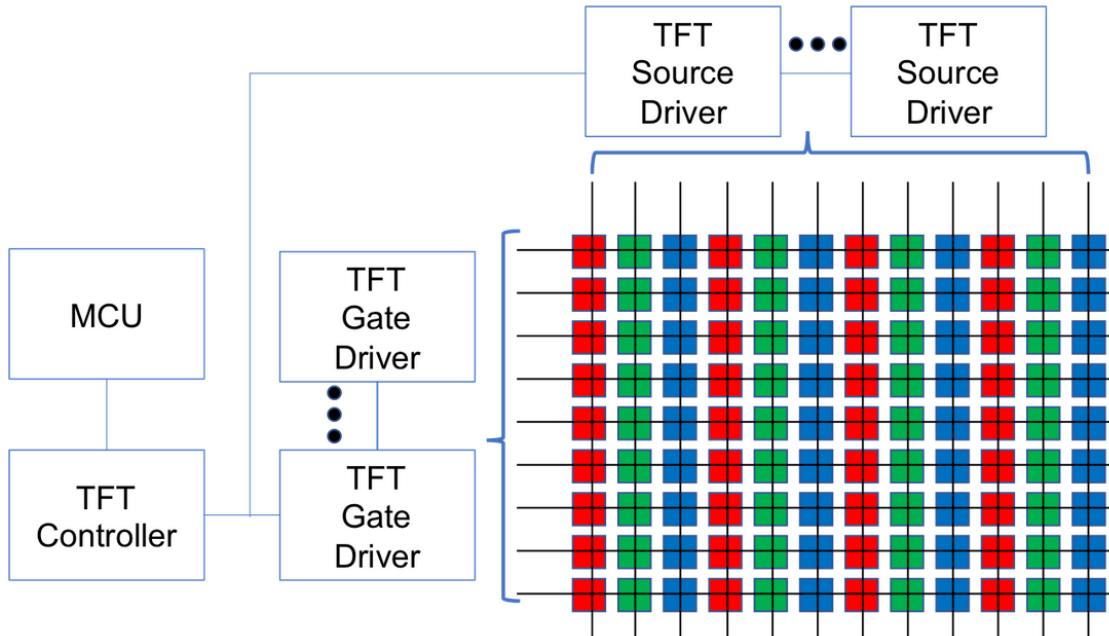


Abbildung 4 Pixel Anordnung eines TFT-Displays

Um ein fünf Zoll RGB Display mit einer Auflösung von 800x480 Pixeln anzusteuern, benötigt man also mindestens 2240 Pins. Für kleinerer Bildschirme gibt es Treiber, die genug Pins bereitstellen. Um Displays größer als 3,5 Zoll anzusteuern werden mehrere Treiber benutzt und in Gate und Source Treiber unterteilt. In Abbildung 5 ist die Treiberaufteilung von einem 800x480 Pixel TFT-Display zu sehen.

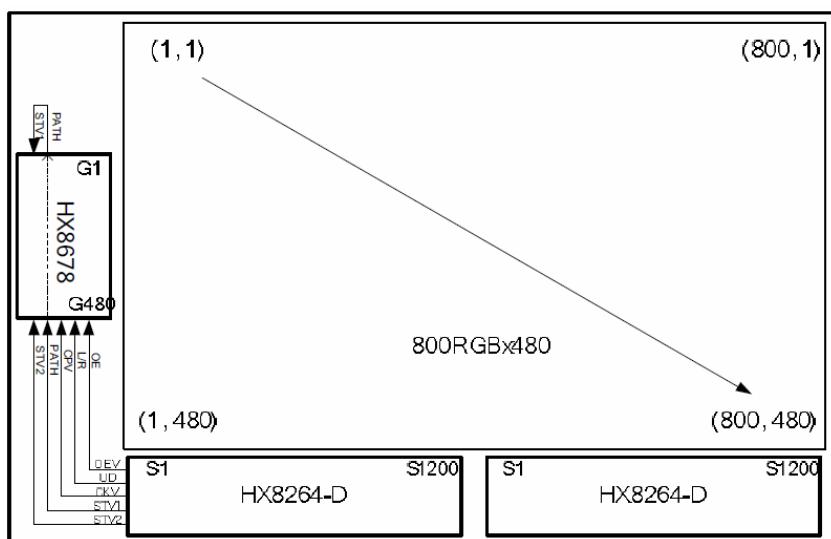


Abbildung 5 Aufteilung Gate- & Source-Treiber, LCD Adafruit 1596

3.2 MEMS-Beschleunigungssensoren

MEMS ist die Abkürzung für mikro-elektromechanische Systeme. MEMS-Beschleunigungssensoren sind Feder Masse Swinger aus Silicium, die durch Fotolithografie hergestellt werden. Sie bestehen aus einer seismischen Masse die federnd zwischen zwei Elektroden gelagert ist. Die seismische Masse und die Elektroden bilden eine Kamm Struktur von wenigen Mikrometern Breite. Zwischen der seismischen Masse und den Elektroden entsteht ein Plattenkondensator.

Erfährt der Sensor eine translatorische Beschleunigung wird die seismische Masse ausgelenkt. Die Auslenkung kann durch eine Kapazitätsänderung gemessen werden. Aus dem Verhältnis der Kapazitäten der beiden Elektroden kann die Richtung der Beschleunigung ermittelt werden. Das Messergebnis wird digital als vielfaches der Erdbeschleunigung angegeben. Um alle drei Raumrichtungen zu erfassen sind immer drei zueinander orthogonal angeordnete Sensoren verbaut.

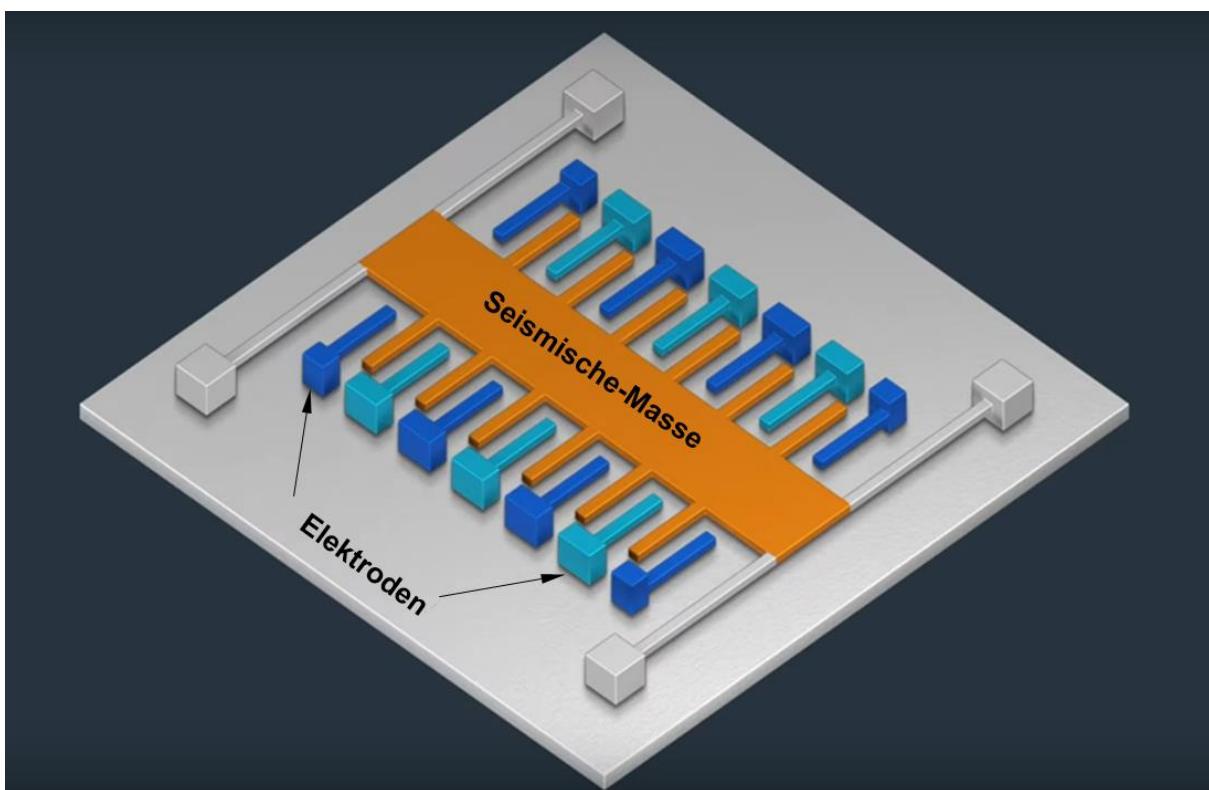


Abbildung 6 MEMS-Beschleunigungssensor

In Z-Richtung wirkt auf den Beschleunigungssensor immer die Erdbeschleunigung. Diesen Effekt kann man sich zu Nutze machen, um die Neigungswinkel des Sensors im Raum zu ermitteln. Wirkt auf den Beschleunigungssensor nur die Erdbeschleunigung und keine translatorische Beschleunigung teilt sich die Erdbeschleunigung je nach Neigungswinkel auf die drei Raumachsen des Sensors auf.

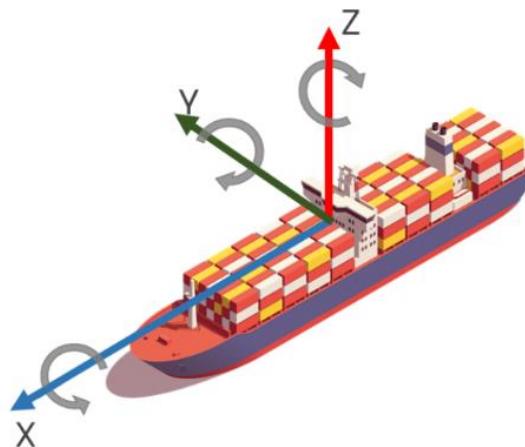


Abbildung 7 Definition der Neigungswinkel im Raum

Kennt man nun die Teilbeschleunigungen der drei Raumachsen und die Erdbeschleunigung kann über trigonometrische Beziehungen auf die Neigungswinkel im Raum zurück gerechnet werden. Alternativ kann der Neigungswinkel auch über eine zweifache Integration der Winkelbeschleunigung ermittelt werden. Die beiden Verfahren zur Bestimmung des Neigungswinkels unterscheiden sich hauptsächlich in ihrer Fehlercharakteristik. Die Bestimmung durch doppelte Integration weist ein geringeres Rauschen der Messwerte auf, hat allerdings auch eine deutlich höhere Langzeitdrift.

$$\text{Neigung X-Achse: } \Phi = \tan^{-1} \frac{y}{z}$$

$$\text{Neigung Y-Achse: } \theta = \tan^{-1} \frac{z}{x}$$

$$\text{Neigung Z-Achse: } \phi = \tan^{-1} \frac{x}{y}$$

4 Grundlagen Software

4.1 Callback-Funktionen

Callback-Funktionen sind Funktionen, die andere Funktionen als Parameter übergeben. Diese Funktion wird unter definierten Bedingungen der Argumente aufgerufen. Der Callback wird über Referenzen aufgerufen, beispielsweise in der Programmiersprache C via Funktions-Pointer.

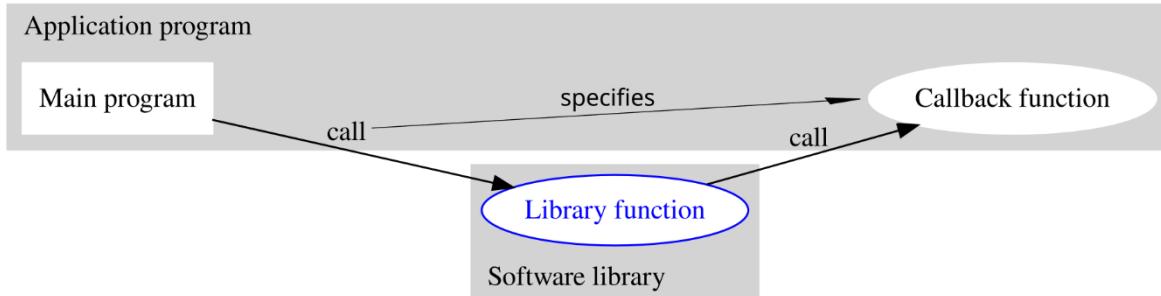


Abbildung 8 Callback Aufruf

In der Abbildung 8 wird dargestellt, wie ein Callback-Aufruf abläuft. Im Main Programm wird eine Callback-Funktion über seine Referenz aufgerufen, welche dann innerhalb der Software Library gesucht wird. Folgend wird die Funktion aufgerufen, hierbei kann aber über Spezifikationen beim Aufruf unterschiedliche Callbacks aufgerufen werden. Beispielsweise kann eine Callback-Funktion aufgerufenen werden, wenn ein Interrupt auftritt und dabei auch unterschiedliche Funktionalitäten für verschiedene Arten von Interrupts erfüllen. Callback Funktionen eignen sich, um Event-Handling für verschiedene Anwendungen einzurichten.

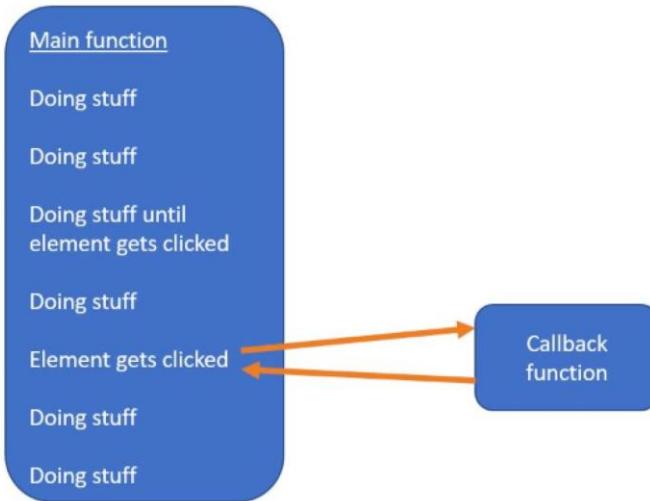


Abbildung 9 Event-Handling

In Abbildung 9 wird das Event-Handling für eine graphische Anwendung dargestellt. Hierbei wird eine Callback-Funktion aufgerufen, falls eines der Elemente der Anwendung angeklickt wird. Abhängig von zusätzlichen Parametern könnte ebenfalls unterschiedliche Funktionalitäten erfüllt werden, beispielsweise ob der Nutzer Administrator oder normaler Nutzer ist. Abschließend ist bei Callbacks noch zwischen Blocking und Non-Blocking zu unterscheiden. Bei Blocking wird nach dem Funktionsaufruf erst in das Main Programm zurückgegangen, nachdem die Funktion vollständig abgearbeitet wurde. Für Non-Blocking ist das nicht der Fall, hier wird der Aufruf parallel zum Main Programm abgearbeitet. Diese geschieht, indem der Callback auf einen anderen Thread auslagert, wird.

4.2 Game Loop

Als Game Loop bezeichnet man die Schleife der Prozesse, die für ein klassisches Computerspiel notwendig sind. Diese Schleife läuft endlos, außer das Spiel wird beendet.

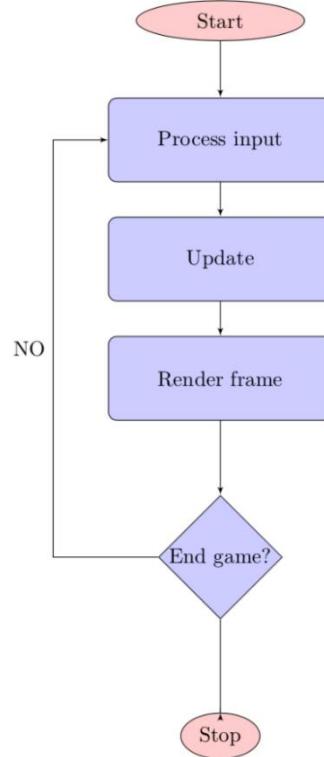


Abbildung 10 Essenzielle Schritte Game Loop

In Abbildung 10 sind die essenziellen Schritte eines Game Loops dargestellt. Zuerst werden die Eingaben verarbeitet, danach die Parameter des Spiels aktualisiert und abschließend wird der neue Spielstatus gerendert.

Dieser Prozess muss in jedem Frame durchlaufen werden, um ein flüssiges und stockfreies Spiel zu erreichen. Folglich hat die Anzahl der Frames pro Sekunde einen großen Einfluss auf die notwendige Rechenleistung des Spiels. Aus diesem Grund ist die Bildwiederholrate bei den meisten kommerziellen Computerspielen variable, um das Spiel auf verschiedenen Endgeräten spielbar zu machen.

Die unterschiedlichen Phasen enthalten üblicherweise folgende Prozesse für 3D-Spiele (z.B. Minecraft). Bei der Eingabe Verarbeitung werden die Eingaben durch Tastatur, Maus oder andere Steuerungen bzw. Sensoren erkannt und temporär gespeichert. Anschließend werden bei der Aktualisierung die Parameter der Objekte anhand der Eingaben oder unabhängig von Eingaben angepasst. Des Weiteren wird in dieser Phase auch Kollisionserkennung oder andere Spielalgorithmen abgearbeitet. Abschließend wird eine neue Szene mit bewegter Kamera, bewegtem Spieler und geänderten Umgebung bzw. geänderten Objekten gerendert. Final wird der Frame ausgegeben und es werden erneut Eingaben eingelesen.

4.3 Microchip Harmony

Die Software MPLAPX IDE ist eine der Integrierten Entwicklungsumgebungen des Herstellers Microchip. Diese beinhaltet das Plugin Microchip Harmony/Content Creator, welches ermöglicht auf Software Librarys zur Konfiguration sowie Nutzung der Grundfunktionen des Mikrocontrollers zuzugreifen. (z.B. Pins-Konfigurieren sowie Setzen)

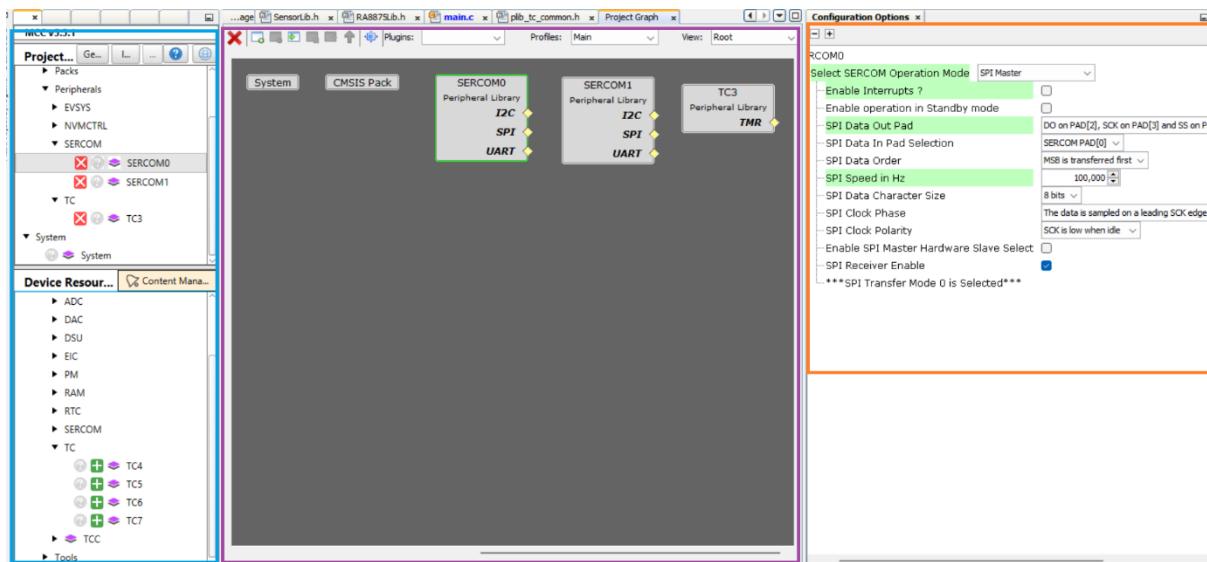


Abbildung 11 IDE MPLAPX

Die Abbildung 11 zeigt die wichtigsten Elemente des Plugins. Auf der linken Seite (blaue Box) werden die Ressourcen des Projekts und des Mikrocontrollers angezeigt. Die Mikrocontroller Ressourcen können dem Projekt hinzugefügt werden, wodurch in den Dateien eine Library hinzugefügt wird diese beinhaltet Funktionen zur Nutzung der Elemente. Beispielsweise Funktionen für Starten und Stoppen eines Timers oder Prototypenfunktionen für Callbacks zum Interrupt des Timers.

Die Projektressourcen sind in der Mitte (lila Box) dargestellt. Auf der rechten Seite (orangene Box) werden die Konfigurations-Optionen für die ausgewählte Projektressource angezeigt. In der Abbildung 11 sind die Konfigurationsmöglichkeiten für eine serieller Schnittstelle zu sehen. Bei Änderung dieser Einstellungen wird automatisch die Library somit auch die Funktionen des Elements aktualisiert. Weitere Funktionalitäten des Plugins sind PIN-Zuweisung, System-Clock Konfiguration, Priorität der Interrupts festlegen sowie Event Konfiguration.

5 Methodik

Für die Umsetzung der Projektidee wurde folgenden Methodik gewählt. Nachdem die Projektidee ausgearbeitet und mit dem betreuenden Professor abgeklärt wurde begann die Bauteilauswahl. Bei der Auswahl der Bauteile lag der Fokus darauf die Projektidee mit möglichst geringen Entwicklungskosten umzusetzen.

Die Komponenten wurden für einen Steckbrettaufbau ausgewählt, um flexibel zu bleiben und möglichst schnell in die Softwareentwicklung einzusteigen. Nachdem bei der Steckbrettentwicklung die grundlegenden Funktionen des Spiels erfolgreich implementiert wurden, wurde das Projekt in eine separate Hardware und Software-Entwicklung aufgeteilt. Die Hardwareentwicklung hatte dabei das Ziel den Steckbrettaufbau in eine Handheldkonsole zu übersetzen. Die Softwareentwicklung hatte das Ziel, das Spiel weiter auszubauen und bestehende Bugs zu fixen. Final wurden die separate Hardware- und Softwareentwicklung im Endprodukt zusammengeführt.

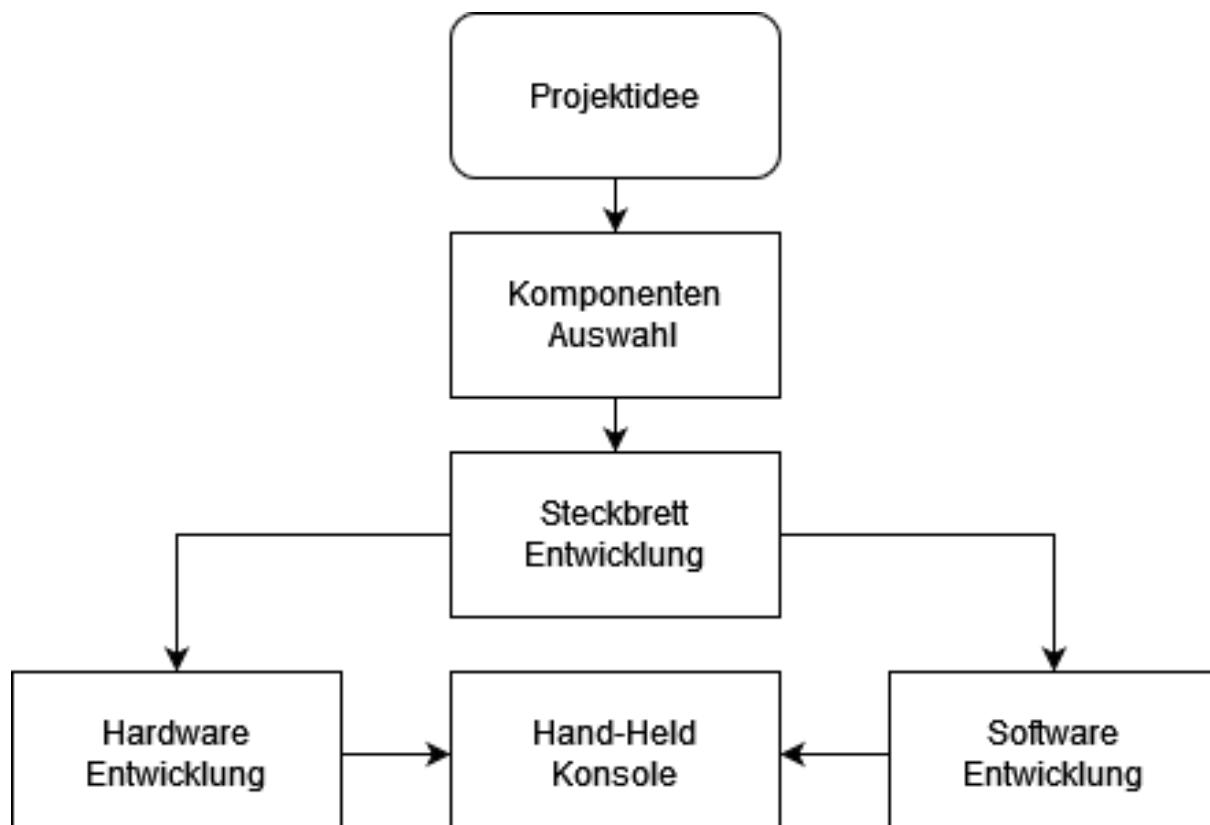


Abbildung 12 Methodik Projektumsetzung

6 Komponenten Auswahl

6.1 SAMD21

Zu Beginn des Projekts war nicht absehbar, wie viel Rechenleistung notwendig ist, um das Spiel mit einer ausreichenden Bildwiederholrate zu betreiben. Da im Elektrolabor bereits ein Entwicklungsboard für den SAMD21 vorhanden war fiel die Auswahl auf den SAMD21. Der 32-Bit-Mikrocontroller zeichnet sich durch die ARM-Cortex-M0+ CPU aus, die mit 48Mhz taktet und 256 kB Speicher und 32 kB Arbeitsspeicher besitzt.

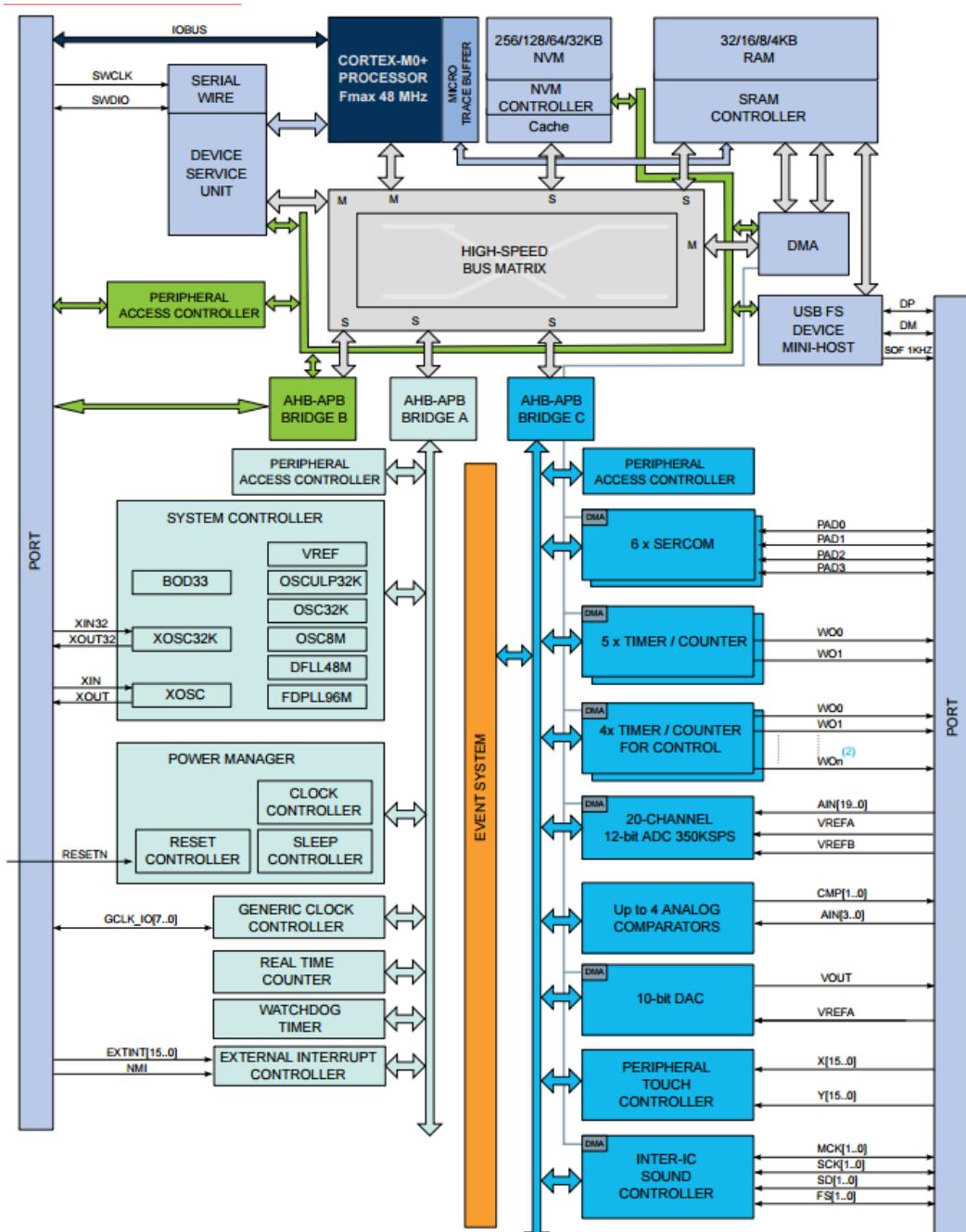


Abbildung 13 Block Diagramm SAMD21

Elektronische-Systeme

In Abbildung 13 sind die Baugruppen des SAMD21 zu sehen. Er besitzt 64 programmierbare Pins sowie 6 frei einstellbare serielle Schnittstellen, um mit externen Teilnehmern zu kommunizieren. Ebenfalls sind 5 Timer/Counter verbaut, welche für einen periodischen Game Loop notwendig sind.

6.2 Beschleunigungssensor BMA400

Anforderung an den Sensor war die Beschleunigung in drei Raumachsen mit möglichst hoher Auflösung zu messen. Außerdem war für den Steckbrettaufbau ein Entwicklungsboard vom Sensor notwendig.

Es wurde der BMA400 ausgewählt. Aufgrund seiner geringen Baugröße von 2x2mm und seinem niedrigen Strombedarf von maximal 14,5 μ A, ist der Sensor gut für Handheld Anwendungen geeignet. Der Sensor gibt die Beschleunigung als vorzeichenbehafteten 12-Bit-Integer aus, diese Auflösung ist für unsere Anwendung ausreichend.

Der Sensor misst die Beschleunigung in allen drei Raumachsen. Des Weiteren kann der Sensor auf einen Messbereich von 2, 4, 8 oder 16 g konfiguriert werden. Zusätzliche Eigenschaften des Sensors beinhalten konfigurierbare Output-Datenraten, integrierte Tiefpassfilter sowie eine programmierbare Interrupt-Engine. Die Kommunikation mit dem Mikrokontroller erfolgt über eine SPI-Schnittstelle.

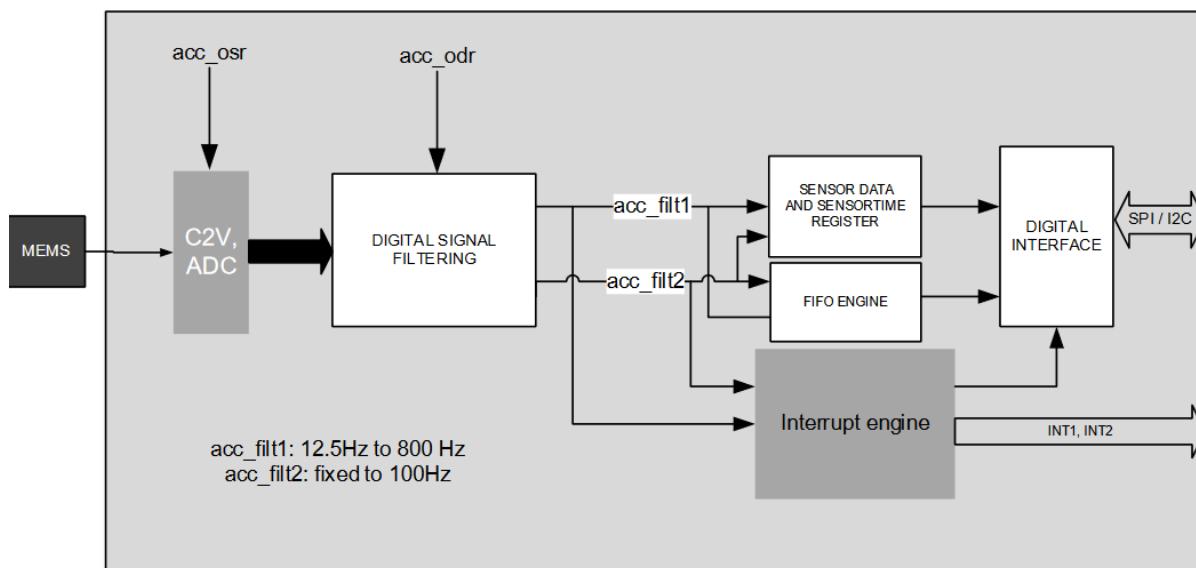


Abbildung 14 Blockschaltbild BMA400

6.3 Display Adafruit 1596

Die Auswahlkriterien beim Display waren die Bildschirmdiagonale, Bildschirmauflösung und Bildwiederholrate. Die Bildschirmdiagonale muss mindestens 5 Zoll betragen um eine ausreichend große Spielkarte darstellen zu können. Um das Spielerlebnis für den Nutzer angenehm zu gestalten, sollten die Bildwiederholrate und Bildschirmauflösung möglichst hoch sein.

Das TFT-Display 1596 von Adafruit erfüllt die Anforderungen mit einer Bildschirmdiagonale von fünf Zoll, einer Auflösung von 800x480 Pixeln und einer Bildwiederholrate von 60 Hz. Das Display wird über eine 40 Pin FPC Stecker angeschlossen. Das Display benötigt eine digitale Versorgungsspannung von drei Volt und eine separate zwölf Volt Versorgung für die Hintergrundbeleuchtung. Angesteuert wird das Display mit dem Gate-Treiber HX8664-B und dem Source-Treiber HX8264-D.

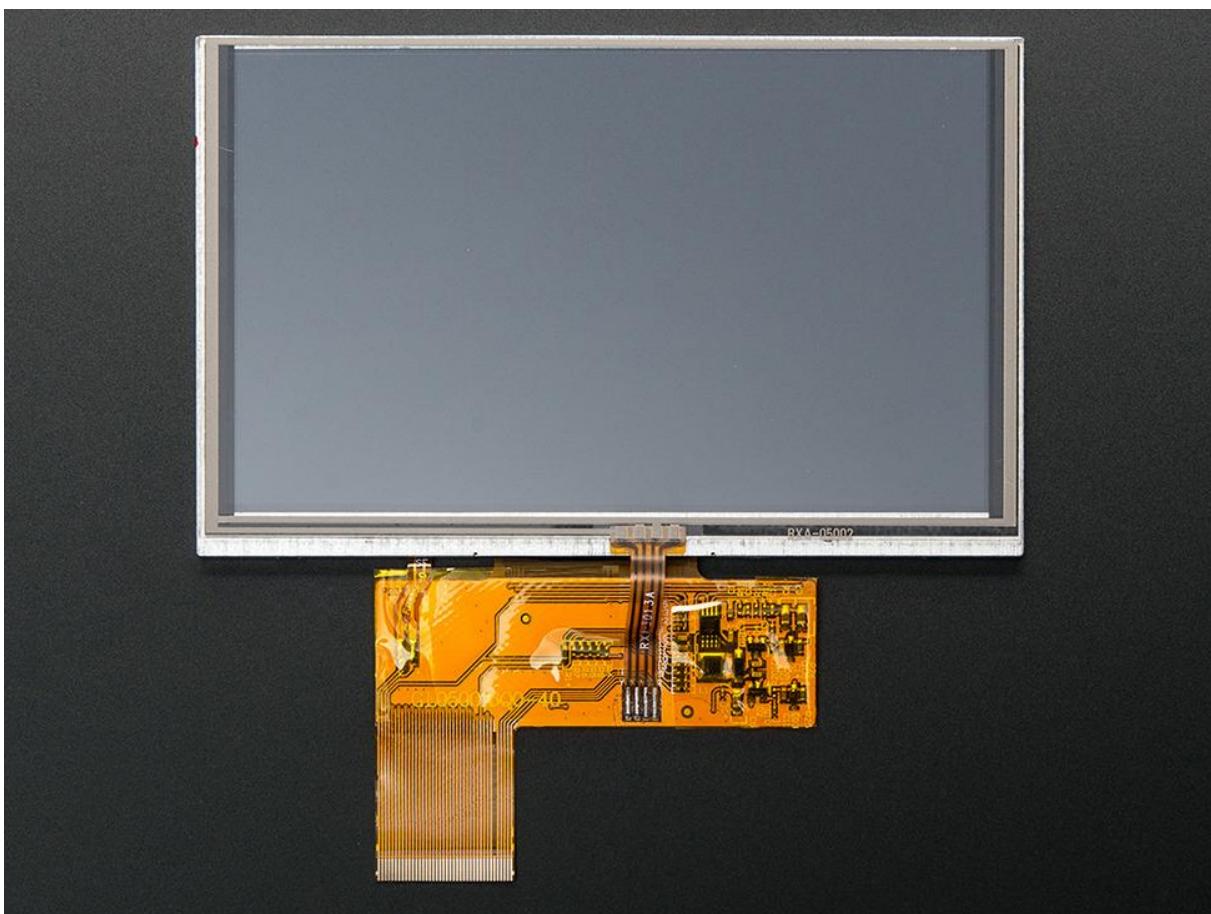


Abbildung 15 LCD Adafruit 1596

6.4 Display Treiber-Board RA8875

Bei größeren LCDs, wie dem Adafruit 1596, lassen sich die Daten der Pixel nicht mehr einfach über eine serielle Schnittstelle des Mikrocontrollers an den Displaytreiber übertragen. Einerseits reicht die Übertragungsgeschwindigkeit der seriellen Schnittstelle meist nicht mehr aus um eine ausreichen Bildwiederholrate zu erreichen. Andererseits müssen die Pixel in einem Pufferspeicher zwischen gespeichert werden, da die Speicher kondensatoren der Dünnschichttransistoren sich entladen.

Der Arbeitsspeicher des SAMD21 reicht nicht aus, um die Pixel des Adafruit 1596 zu puffern. Mit dem Treiberboard RA8875 von Adafruit lassen sich auch größere TFT-Displays ansteuern. Das Board basiert auf dem LCD Controller Chip RA8875 von RAiO. Der Chip bietet eine ausreichend schnelle Schnittstelle für das Display und 768KB Arbeitsspeicher, um das Display zu puffern.

Der SAMD21 kommuniziert mit dem RA8875 über eine SPI-Schnittstelle. Für die Kommunikation per SPI bietet Adafruit eine Open-Source-Library die bereits Grafik- und Textfunktionen bereitstellt. Zusätzlich zu dem LCD-Controller Chip ist auf dem Treiber-Board ein Boost Converter, der die Versorgungsspannung für die Hintergrundbeleuchtung des LCDs liefert.

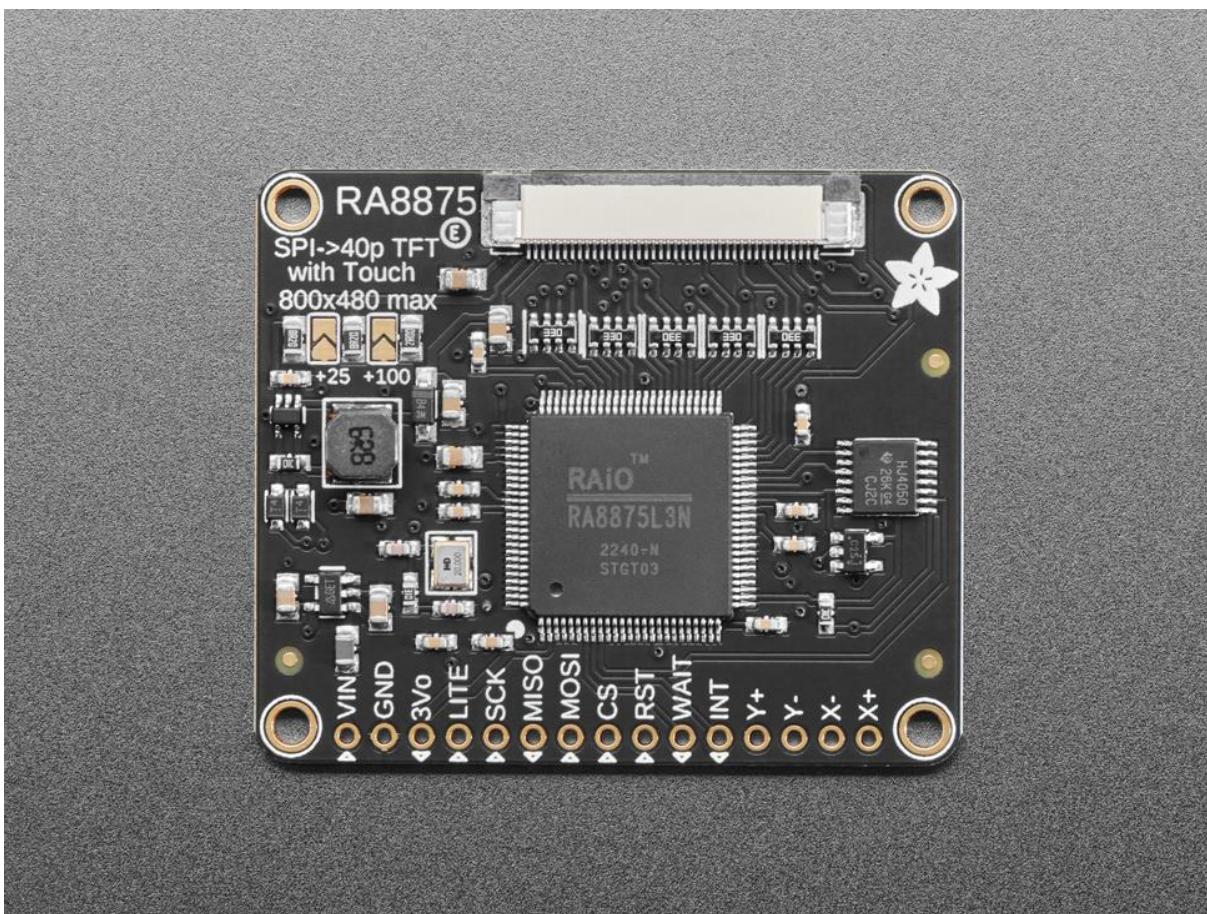


Abbildung 16 Treiber Board RA8875

7 Umsetzung

7.1 Umsetzung Software

7.1.1 Allgemeines zur Programmierung

Die Grundfunktionen des SAMD21J18A wurden mittels Harmony programmiert. Es wurden zwei SPI-Schnittstellen konfiguriert, ein Timer und die SysTick-Funktionalität der Cortex M0.

7.1.2 Beschleunigungssensor BMA400

Der BMA400 wird mittels SPI angesteuert, die Taktfrequenz wurde auf 100 kHz konfiguriert. Zu Beginn der Kommunikation werden die Konfigurationsregister des Sensors so beschrieben, dass die Output-Datenrate 100 Hz beträgt und der Tiefpassfilter genutzt wird. Der Messbereich wird auf 2g gesetzt.

Die Beschleunigungsdaten werden, jeden Game-Loop, für alle Achsen mittels SPI-Multiread aus den Registern ausgelesen und in 16-Bit-Integer gewandelt. Die Beschleunigungsdaten der X- sowie Y-Achse werden direkt in eine Bewegung auf der Anzeige übertragen.

7.1.3 Display

Der Treiber des Displays wird über eine der SPI-Schnittstellen des SAMD21J18A angesteuert. Die Schnittstelle wird mit 2 MHz getaktet und im SPI-Modus 0 betrieben. Hierfür wurde die Library von Adafruit von C++ in C umgeschrieben. Außerdem wurde der Low-Level-Access der Library auf den SAMD21 angepasst, ursprünglich wurde die Library für einen Arduino Mikrocontroller geschrieben.

7.1.4 Game Loop

In Abbildung 17 wird der Game-Loop dargestellt. Dieser dient zur Aktualisierung der Anzeige, sowie den Positions- und Bewegungsdaten der Kugel. Der Timer wurde darauf programmiert, alle 25 Millisekunden periodischen einen Interrupt auszulösen. Der Interrupt ruft dabei einen Callback auf, welcher als Game Loop für das Spiel dient. Bei der Errechnung der nächsten Position werden die Daten des Sensors skaliert und limitiert. Die Limitierungen dienen dazu, das das Zittern der Hände sowie starke Stöße kein Einfluss auf die Bewegung der Kugel haben. Die Bewegung in Pixel ist somit proportional zu der Beschleunigung.

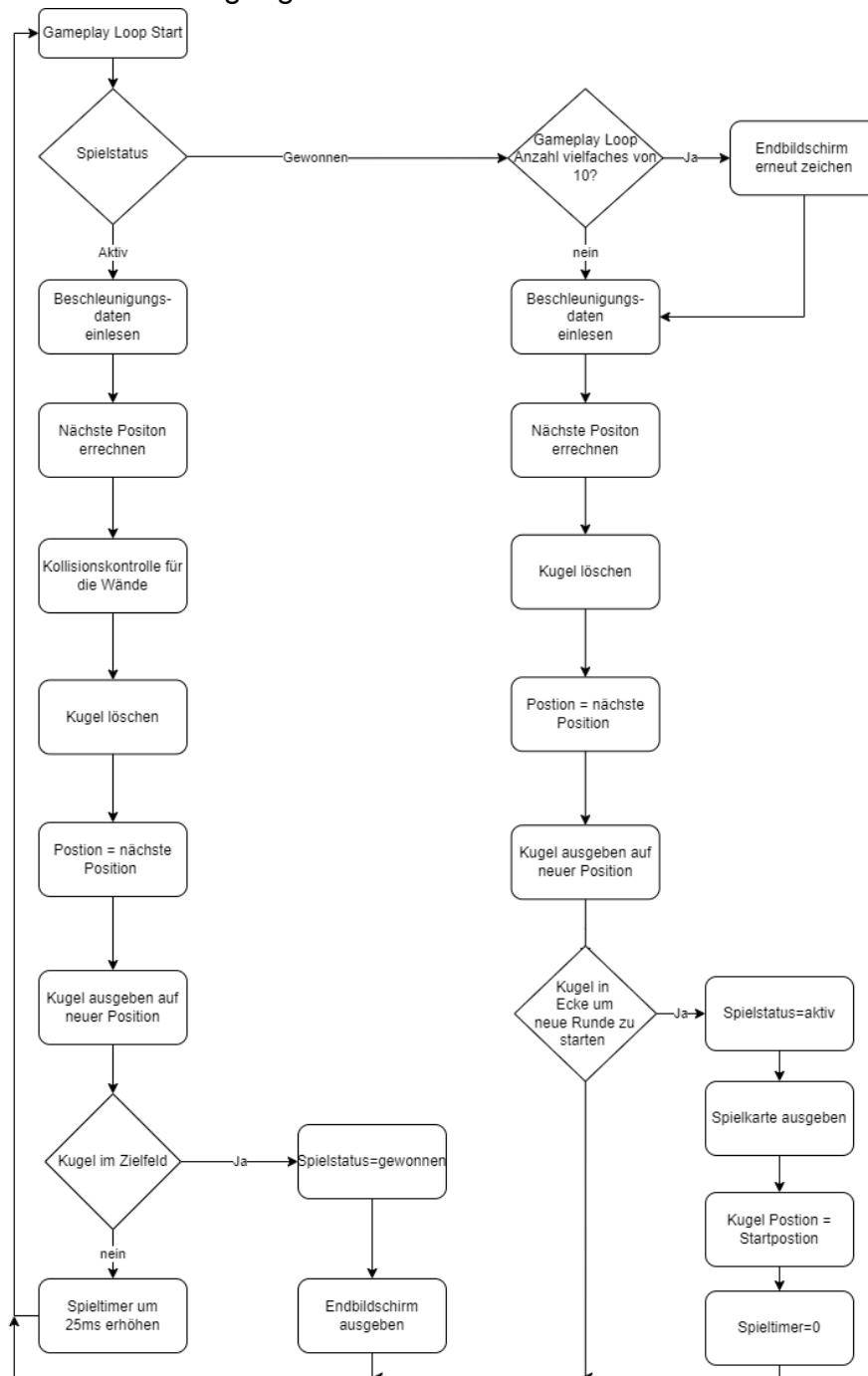


Abbildung 17 Game Loop Gyro-Game

7.1.5 Kollisionserkennung

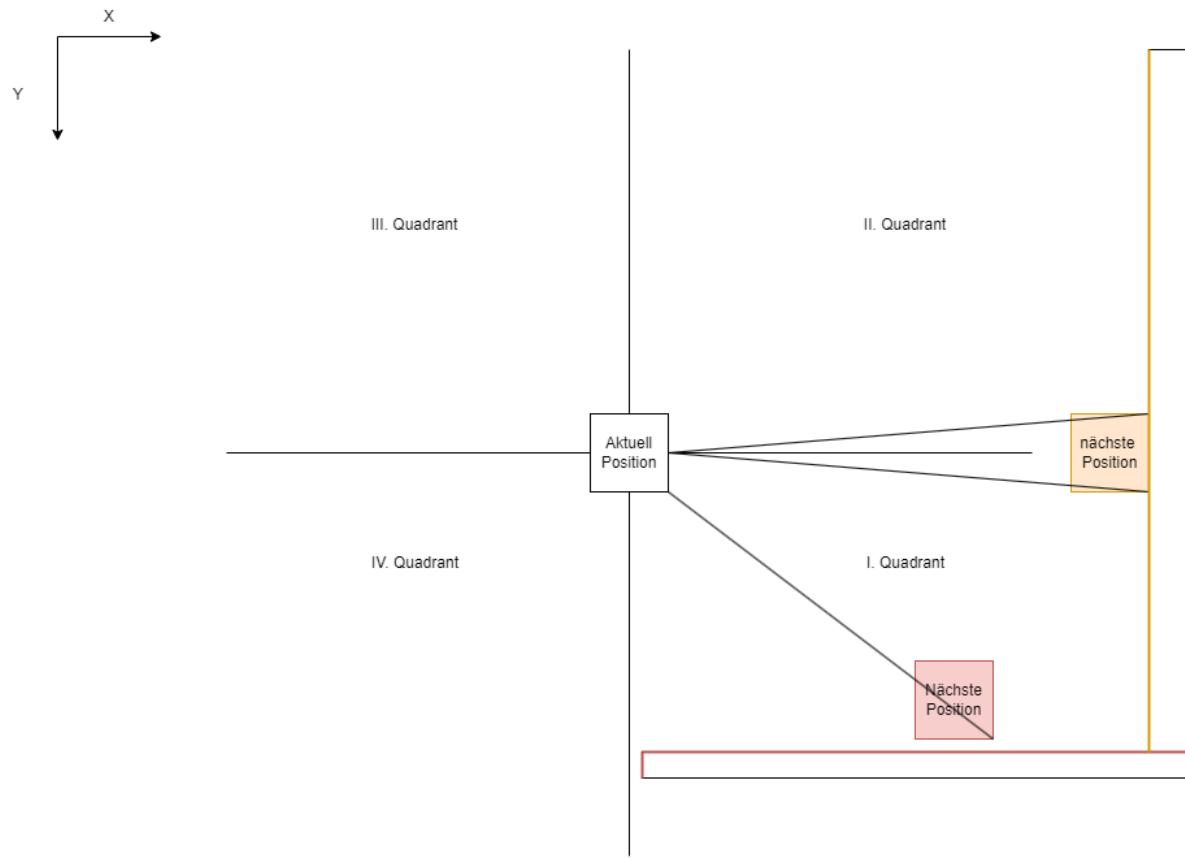


Abbildung 18 Konzept Kollisionserkennung

In der Abbildung 18 wird das Konzept der Kollisionserkennung dargestellt. Die Kugel wird als Quadrat angenommen, um die Berechnungen zu vereinfachen. Zur Kollisionserkennung wird zwischen der aktuellen und nächsten Position eine Linie berechnet. Die Line verläuft vom Mittelpunkt des Quadrats zu einem oder mehreren Eckpunkten der nächsten Position, abhängig davon in welche Richtung die Bewegung verläuft und ob die Bewegung eine Kombination von Änderungen in x sowie y Richtung sind oder rein auf eine der Achsen basiert. Anschließend wird überprüft, ob die Linie bzw. Linien Schnittpunkte mit den in Abbildung 18 farblich markierten Rändern der Wände haben. Hierbei werden ebenfalls abhängig von Bewegungsquadranten sowie diagonale oder gerade Bewegung nur die Ränder überprüft, welche Möglichkeiten für eine Kollision darstellen. Der Schnittpunkt wird nachfolgenden Konzept berechnet.

Zunächst werden alle Punkte der Linie durch die Start- sowie Endpunkte und den Parameter t und u dargestellt.

(Bezier Kurve 1. Grades)

$$L_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + t \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix}, L_2 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} + u \begin{bmatrix} x_4 - x_3 \\ y_4 - y_3 \end{bmatrix}$$

Danach wird nun ein Schnittpunkt berechnet, dieser ist auf den Linien, wenn $0 \leq t \leq 1$ und $0 \leq u \leq 1$ für folgende Berechnung gilt:

$$t = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

$$u = \frac{(x_1 - x_2)(y_1 - y_3) - (y_1 - y_2)(x_1 - x_3)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

Wenn ein Schnittpunkt erkannt wird, kann über das Einsetzen von t oder u in L_1 oder L_2 der Schnittpunkt errechnet werden. Da mehrere Schnittpunkte pro Frame berechnet werden können wird ebenfalls der Abstand der Kugelposition zum Schnittpunkt errechnen. Folglich wird die Kollision mit dem kleinsten Abstand zur Position übernommen. Dieser Schnittpunkt wird nach Addition oder Subtraktion des Kugelradius, abhängig von der Bewegungsrichtung, als nächste Position übernommen.

Bei der Kollisionserkennung wurde sich gegen die Nutzung von üblichen Algorithmen, wie AABB (axis-aligned bounding boxes) oder SAT (separating axis theorem), entschieden. Da die Algorithmen annehmen, dass die Position bzw. nächste Position zu einer Überlappung der Körper führt. In dem Kugellabyrinth ist das nicht der Fall, da die Bewegung pro Frame groß genug ist, um keine Überlappung in der nächsten Position zu verursachen.

Abschließend ist zu erwähnen, dass in der aktuellen Version des Spiels die Kollisionserkennung stark fehlerbehaftet ist. Diese Fehler treten durch Rundung bei der Berechnung des Schnittpunkts auf, welcher Sub Pixel genau ist, aber durch eine Integer-Konversion abgerundet wird, was zu Fehlern der kalkulierten nächsten Position führt. Ebenfalls sollte bei der rein geraden Bewegung noch eine weitere Linie berechnet werden, da der Abstand der Eckpunkte des Quadrats größer als die Wanddicke ist, dadurch können Kollisionen unerkannt bleiben. Des Weiteren sollte beachtet werden, dass durch Fehler der vorherigen Kollision, wenn die Position in der Wand ist, die folgende ebenfalls unerkannt bleibt.

7.2 Umsetzung Hardware

7.2.1 Steckbrett Aufbau

Bei der Recherche der Bauteil gab es keine vergleichbare Projektarbeiten. Um bei der Entwicklung flexibel auf Fehlentscheidungen bei der Bauteilauswahl reagieren zu können und schnell in die Softwareentwicklung einzusteigen wurde mit einem Steckbrettaufbau begonnen.

Der Steckbrettaufbau besteht aus dem LCD Adafruit 1596, dem LCD Treiberboard RA8875 von Adafruit, einem SAMD21 Xplained Pro Board von Microchip Technology und einem Entwicklungsboard der Firma MikroElektronika für den Beschleunigungssensor BMA 400. Die Spannungsversorgung wurde mit einem Labornetzteil realisiert. Am Labornetzteil kann eine Variable Versorgungsspannung für die Bauteil eingestellt werden. Außerdem wurde der Versorgungsstrom begrenzt, um eine Beschädigung der Bauteile auszuschließen. Die seriellen Schnittstellen der Bauteile wurden mit Jumper Kabeln verbunden. Jumper Kabel ermöglichen eine flexible Verbindung sind allerdings nicht für serielle Kommunikation geeignet, je nach Lage der Kabel kam es öfters zu Fehlern bei der Datenübertragung.

Die Kommunikation zwischen Treiberboard und Mikrocontroller funktionierte zu Beginn aufgrund eines Software-Fehlers gar nicht. Durch den Steckbrettaufbau konnte die serielle Schnittstelle mit dem Oszilloskop ausgemessen werden und der Fehler dadurch behoben werden.

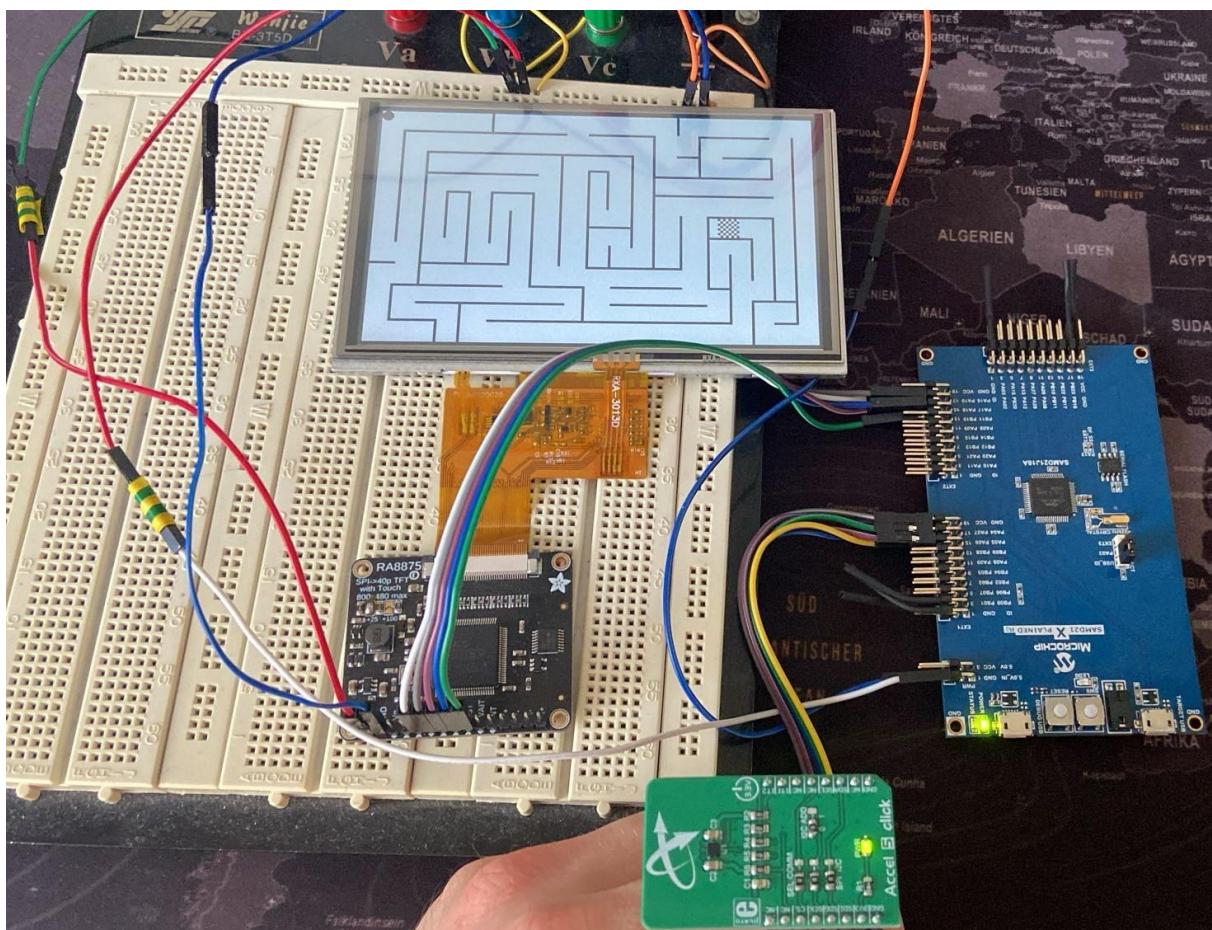


Abbildung 19 Steckbrettaufbau

7.2.2 Platine für Handheld Konsole

Der Steckbrettaufbau mit den Entwicklungsboards und dem Labornetzteil ist zu groß für eine Handheld-Konsole. Um die Baugröße deutlich zu reduzieren, wurden die Entwicklungsboards vom SAMD21, Bosch BMA400 und das Labornetzteil durch eine kompakte Platine ersetzt. Das Displaytreiberboard wurde nicht ersetzt, da es nur wenig Bauraum einnimmt und die Einzelbauteile einen hohen Kaufpreis haben. Bei der Entwicklung der Platine wurde sich am Datenblatt des SAMD21 orientiert. In Kapitel 44 des Datenblatts, Schematic-Checklist, wird der Leser schrittweise durch die Entwicklung einer Platine für den SAMD21 geführt.

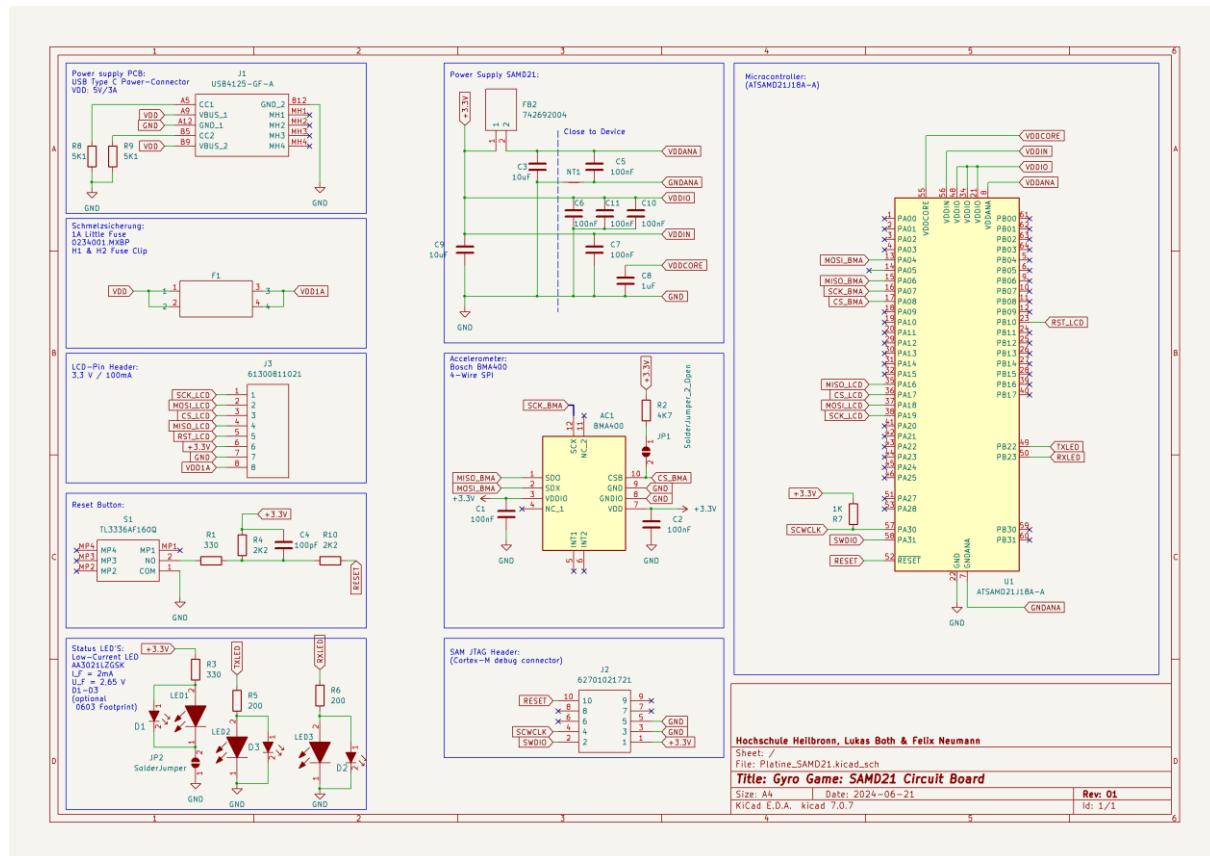


Abbildung 20 Schaltplan Platine

7.2.3 Power Supply

Die Platine wird über einen USB-C Stecker versorgt, da heutzutage jeder Haushalt mindestens ein Ladegerät mit USB-C Anschluss besitzt. Über die CC-Pins vereinbaren Quelle und Senke wie hoch die Versorgungsspannung und Leistung ist. Für die Leistungsverhandlungen zwischen Quelle und Senke gibt es zwei Möglichkeiten. Erstens ein 32-Bit codierten Bus, dafür benötigen Quelle und Senke einen Port-Controller, die Industrie bietet dafür fertige ICs. Die zweite Möglichkeit ist ein Spannungsteiler zwischen Quelle und Senke. In der Senke kann über einen Pulldown-Widerstand das gewünschte Leistungsniveau eingestellt werden.

Für die Spannungsversorgung der Platine ist der Spannungsteiler die bessere Lösung, da die Busverbindung zwar flexibler ist, aber auch weitere Kosten mit sich bringt. Für die Platine wurde mit zwei 5,1 Kilo Ohm Widerständen eine Versorgung von 5V/3A eingestellt.

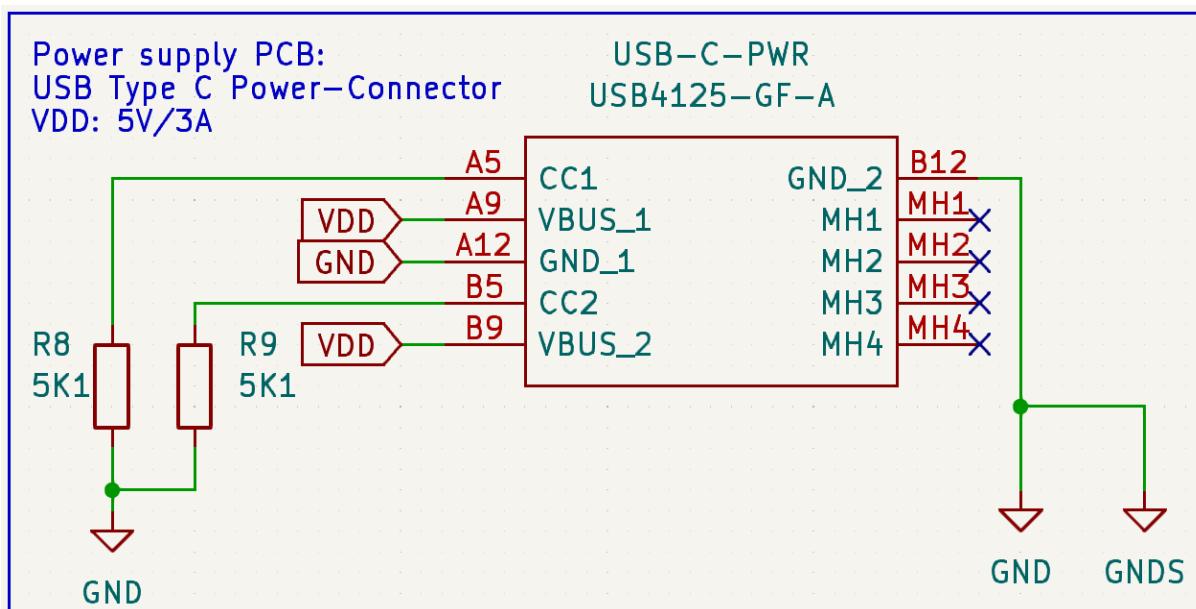


Abbildung 21 USB-C Connector

Die Versorgung wird über eine 1 Ampere Schmelzsicherung an die Versorgungsspins des Displaytreiber-Boards geleitet. Die Platine selbst braucht eine Versorgung von 3,3V/100mA für den Mikrocontroller und den Beschleunigungssensor. Die Platine selbst wird über den 3V0 Pin des Treiber-Boards versorgt, welcher regulierte 3,3 Volt liefert.

7.2.4 Power Supply SAMD21

Der SAMD21 benötigt drei separate Spannungsversorgungen. An den Pins für die Spannungsversorgung befinden sich Stützkondensatoren, siehe Abbildung 22. Die analogen Baugruppen werden über die Spannung VDDANA versorgt. Um die analogen Baugruppen vom Rauschen der Spannungsversorgung zu entstören ist ein SMD-Ferritkern vorgeschalten und der analoge Ground ist durch ein Net Tie entstört. Die digitalen Ein- und Ausgänge werden über die Spannung VDDIO versorgt. VDDIN ist die interne Spannungsversorgung des SAMD21, die regulierte Versorgungsspannung wird am Pin VDDCORE mit einem Kondensator gestützt.

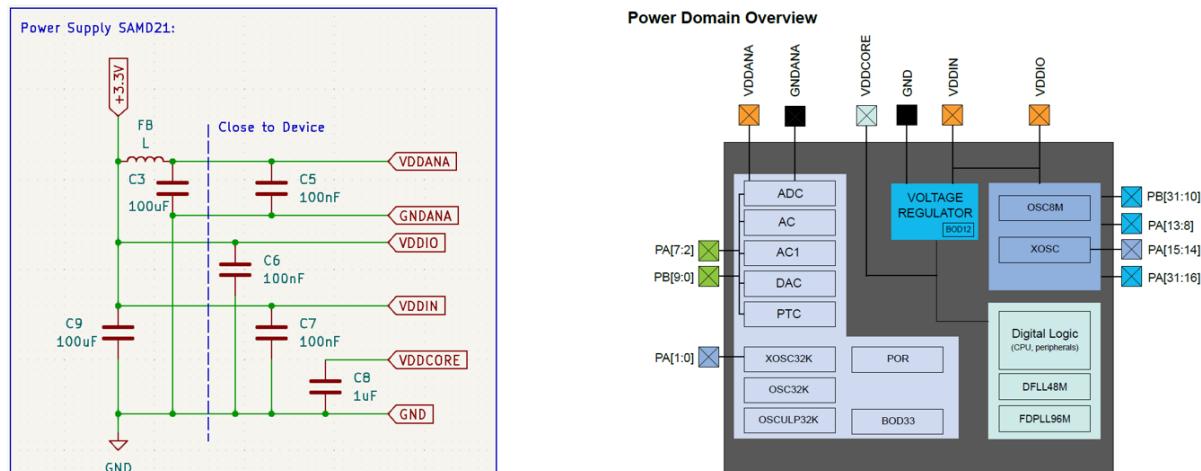


Abbildung 22 Power Supply SAMD21

7.2.5 Debug Schnittstelle

Der SAMD21 unterstützt Serial Wire Debug, im Elektronik Labor stand bereits ein JTAGICE3 Programmiergerät zur Verfügung. Die Debug Schnittstelle ist ein 10 Pin 50 mil. Cortex Debug Connector. Die Schnittstelle verbindet das Programmiergerät mit der Clock (SWCLK) und der bidirektionalen Datenleitung (SWDIO) des SAMD21. Außerdem braucht das Programmiergerät eine Verbindung zum Reset-Pin des SAMD21 und eine Referenz der Versorgungsspannung. Die Clock sowie der Reset-Pin haben jeweils einen Pullup-Widerstand.

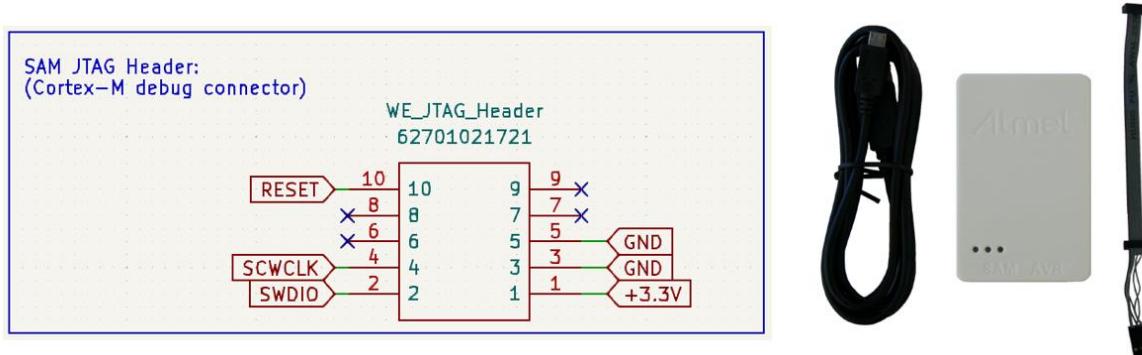


Abbildung 23 Debug Schnittstelle SAMD21

7.2.6 Reset-Taster

Auf der Platine befindet sich ein Reset-Taster, um den Mikrocontroller bei Softwarefehlern manuell zurücksetzen zu können. Der Taster ist im 90° Winkel zur Platine angebracht, dass er später an der Seitenwand des Gehäuses angebracht werden kann. Der Reset des SAMD21 ist aktiv bei Low-Pegel, Widerstand R4 und Kondensator C4 verhindern, dass die Reset-Leitung ungewollt durch EMV-Störungen auf Low-Pegel gezogen wird. Der Reset-Taster liegt parallel zur Reset-Leitung der Debug Schnittstelle. Der Widerstand R10 verhindert, dass der Kondensator C4 die Reset-Leitung auf High-Pegel hält, wenn das Programmiergerät die Reset-Leitung auf Low-Pegel zieht.

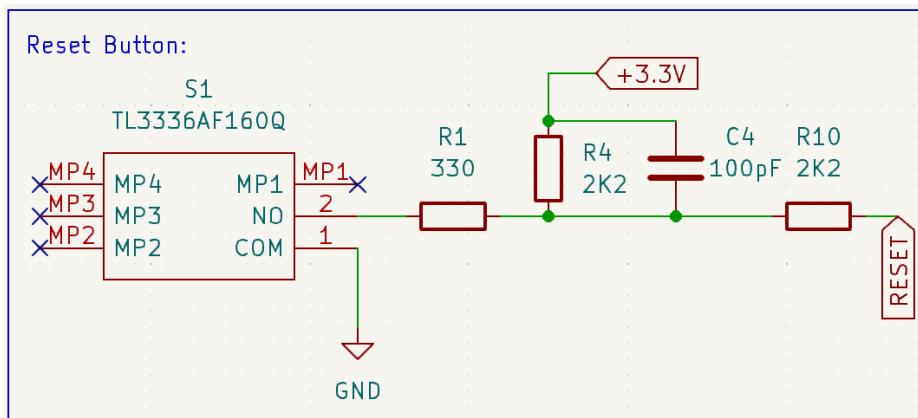


Abbildung 24 Reset Button

7.2.7 General Purpose LEDs

Auf der Platine wurden drei LEDs angebracht, um eine mögliche Fehlersuche zu vereinfachen. Bei den LEDs handelt es sich um low current LEDs, da die 3,3V Spannungsversorgung auf 100mA begrenzt ist. Die PWR LED leuchtet, wenn das Treiberboard 3,3V liefert, sie kann später über einen Solder-Jumper aus dem Versorgungsstromkreis ausgelötet werden. Die TX- und RX-LEDs sind an zwei digitalen Pins des SAMD21 angeschlossen. Sie können beim Debuggen an einer gewünschten Stelle im Code auf High-Pegel gesetzt werden.

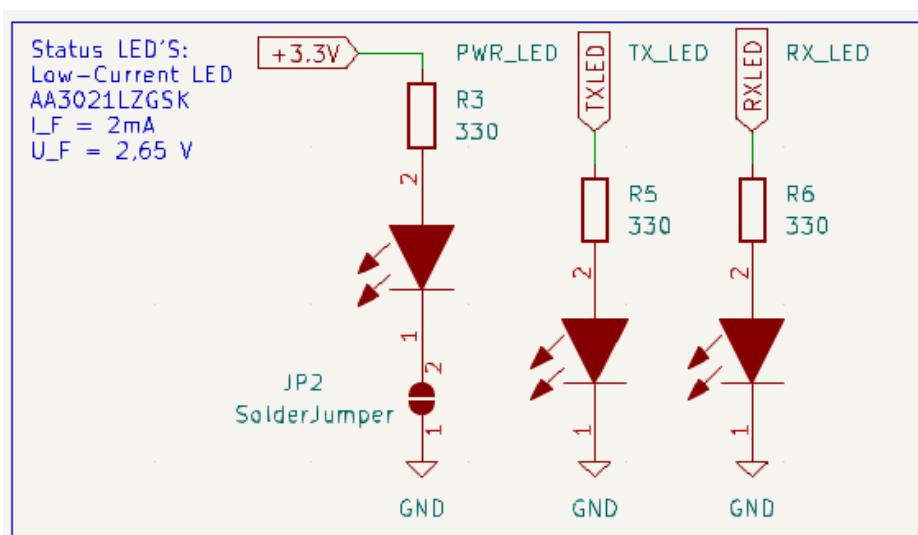


Abbildung 25 General Purpose LEDs

7.2.8 Beschleunigungssensor

Der BMA400 kommuniziert über 4-Wire-SPI mit den SAMD21. Die Chipselect-Leitung des Sensors wird mit dem Widerstand R2 auf High-Pegel gezogen. Der Beschleunigungssensor benötigt 3,3V Versorgungsspannung. Die Versorgungsspannung wird an den Pins 3 und 7 mit zwei Kondensatoren gestützt.

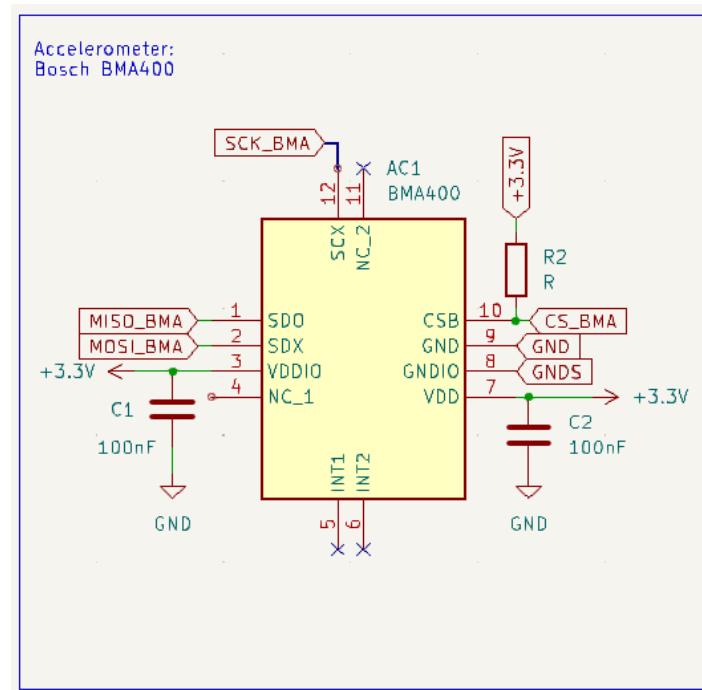


Abbildung 26 Beschleunigungssensor BMA400

7.2.9 Routing

Beim Routing der Platine wurde sich an den folgenden Punkten orientiert. Die Bauteile wurden so gut wie möglich in funktionalen Baugruppen platziert. Das Hauptaugenmerk lag hierbei auf einem möglichst geringen Abstand zwischen den Stützkondensatoren und ihren zugehörigen Pins. Außerdem wurde darauf geachtet die seriellen Datenleitungen räumlich voneinander zu trennen. Leitungen wurden möglichst kurz und ohne spitze Winkel gelegt, um Leitungsreflektionen zu vermeiden. Serielle Datenleitungen wurden möglichst ohne Vias verlegt. Leitungen in unterschiedlichen Schichten wurden immer quer zueinander verlegt. Für eine gleichmäßige Masseverteilung wurden zusätzliche Vias auf Masse gelegt.

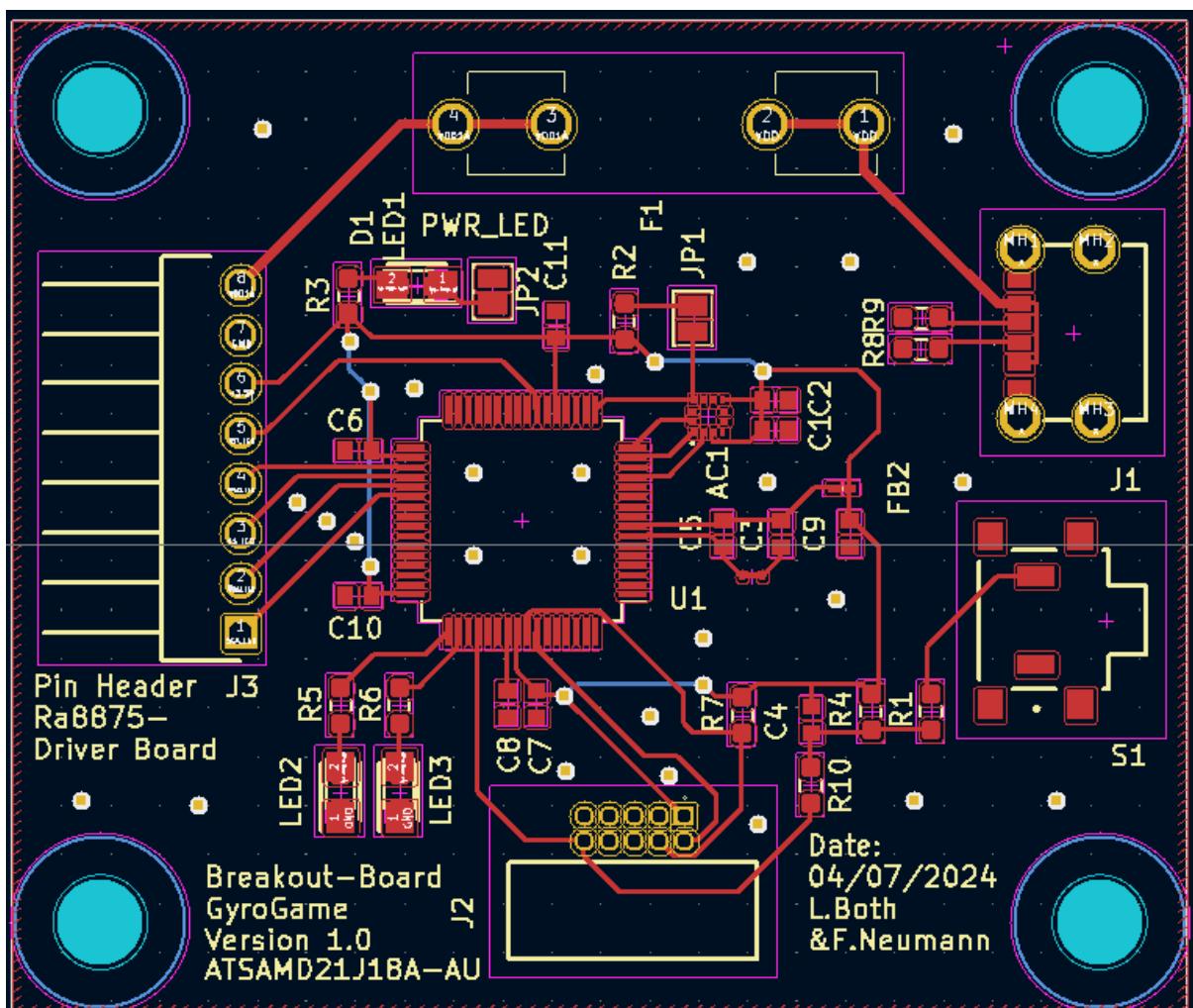


Abbildung 27 Routing Platine

8 Diskussion der Ergebnisse

Die ursprüngliche Projektidee, eines digitalen Kugellabyrinths, wurde erfolgreich umgesetzt. In den folgenden Punkten könnte das Projekt noch verbessert werden.

Die aktuelle Kollisionserkennung ist mit einem hohen Rechenaufwand verbunden und weist immer noch Fehler auf. Der Algorithmus könnte mit Filtern für irrelevante Wände schneller gemacht werden. Es könnte auch der Algorithmus durch eine schnellere Alternative ersetzt werden. Beispielsweise könnte ein AABB-Algorithmus (axis aligned bounding boxes) verwendet werden, indem die Wände einen virtuellen Schatten in Bewegungsrichtung werfen. Die Bewegung der Kugel könnte mit einem physikalischen Modell, das die Massenträgheit berücksichtigt noch realistischer gemacht werden. Bei der Hardware fehlt noch ein Gehäuse, um wie anfangs geplant das Spiel als Handheld Konsole umzusetzen.

In Zukunft könnte man die ursprüngliche Projektidee noch in den folgenden Punkten erweitern. Die aktuelle Version des Spiels ist nicht besonders herausfordernd für den Spieler. Das Spiel könnte mit Löchern und Projektilen zum Ausweichen deutlich erschwert werden. Außerdem wäre ein Spielkartengenerator gut, um den Spielspaß auf Dauer zu erhalten. Bei der Hardware könnten das Display-Treiberboard ersetzt werden, um die Kosten deutlich zu reduzieren.

9 Danksagung

An dieser Stelle möchten wir uns bei unseren Betreuern Herr Professor Doktor Tim Fischer und Herr Ralf Ziegler bedanken, durch die das Projekt erst möglich wurde.

Durch die Unterstützung in den wöchentlichen Meetings konnten wir uns auch an Themengebiete heranwagen, die außerhalb unserer bisher im Studium erlernten Kompetenzen lagen.

Das viele Feedback hat uns sehr geholfen uns im Laufe des Projekts fachlich weiterzuentwickeln.

Lukas Both und Felix Neumann

Heilbronn, 23.07.2024

10 Quellen

Grundlagen TFT Displays:

<https://iotexpert.com/embedded-graphics-tft-displays-drivers/>

<https://www.adafruit.com/product/1590>

MEMS-Beschleunigungssensoren:

https://de.wikipedia.org/wiki/Inertiale_Messeinheit

<https://www.youtube.com/watch?v=KuekQ-m9xpw&t=39s>

Callback Funktionen:

<https://de.wikipedia.org/wiki/R%C3%BCckruffunktion>

https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

<https://www.beningo.com/using-callbacks-with-interrupts/#>

<https://dev.to/flippedcoding/what-is-a-callback-function-3g5j>

Gameloop:

https://de.wikipedia.org/wiki/Game_Loop

<https://www.gameludere.com/2019/11/01/game-loop/>

SamD21J18a:

<https://www.microchip.com/en-us/product/atsamd21j18>

BMA400:

<https://www.bosch-sensortec.com/products/motion-sensors/accelerometers/bma400/>

Microchip Harmony:

<https://www.microchip.com/en-us/tools-resources/configure/mplab-harmony>

Adafruit RA8875 Library:

https://github.com/adafruit/Adafruit_RA8875/tree/master

Kollisionserkennung:

https://en.wikipedia.org/wiki/Line-line_intersection

https://en.wikipedia.org/wiki/B%C3%A9zier_curve

