Bachelorarbeit

Ansteuerung eines LCD-Displays mit Matlab Simulink

von

Konstantin Lutz

Datum 29.01.2021

Betreuung: Prof. Dr. Heinz Rebholz
Zweitprüfer: Dipl.-Ing. Klemens Graf

Fakultät für Elektro- und Informationstechnik

HTWG Hochschule Konstanz Technik, Wirtschaft und Gestaltung

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

LCD Liquid Crystal Display

I²C Inter-Integrated Circuit

IIC Inter-Integrated Circuit

SDA Serial Data Line

SCL Serial Clock Line

µC Mikrocontroller

TWI Two-Wire Serial Interface

ACK Acknowledgement

I/O Input and Output

USB Universal Serial Bus

MSB Most Significant Bit

LSB Least Significant Bit

MCU Microcontroller Unit

DDRAM Display Data RAM

CGRaM Character Generator ROM

RAM Read Access Memory

ROM Read Only Memory

TLC Target Language Compiler

1 Einleitung

1.1 Motivation

In der Vorlesung elektrische Antriebe und Leistungselektronik wird aktuell alles auf eine einheitliche µC Platform umgestellt, dazu wird das TI Launchpad TI28069 als zentrales Element verwendet. Dieses bietet viele flexiblen Möglichkeiten zur Implementierung verschiedenster Projekte, auch ist der Software Support sehr umfangreich, dazu gehört auch die Unterstützung dieses Controllers direkt in MATLAB SIMULINK.

Wichtiges Element, das aktuell nicht zur Verfügung steht, ist eine geeignete Ansteuerung eines Displays zur Ausgabe von verschiedenen Parametern, welches direkt am Launchpad angeschlossen und einfach verwendet werden kann.

1.2 Zielsetzung

Die Implementierung einer LCD Ansteuerung in MATLAB SIMULINK und deren Test ist der Bestandteil dieser Arbeit, dabei sollten auch folgende Eigenschaften erreicht werden:

- einfache Einbindung
- zeilenweise Zuweisung
- automatische und manuelle Aktualisierung
- Kommunikation über I^2C
- Unterstürzung von Standard Displays
- Betrieb in Standalone und Monitor & Tune Modus

Zur Umsetzung der Ziele ist es nötig sich mit den Grundlagen der I^2C Kommunikation vertraut zu machen. Auch die nötigen Hardwaregegebenheiten müssen beachten werden, damit auch später ein zuverlässiger Betrieb gegeben ist. Damit kann dann auch begonnen werden die Software in MATLAB SIMULINK umzusetzen, dabei sind kleine Implementierungsschritte einzuhalten, da beim verändern zu großer Programmteile schnell der Überblick und die Funktionalität verloren gehen kann. Dazu muss immer wieder ausgiebig getestet werden, damit die Implementierung robust und zuverlässig bleibt.

2 Grundlagen

In diesem Kapitel werden alle Grundlagen aufgeführt, die zur Implementierung eines Displaytreibers in MATLAB SIMULINK notwendig sind. Zudem auch, wie ein solcher Display an einem Mikrocontroller angeschlossen und beschalten werden muss.

2.1 Das I²C Protokoll

Das I²C Protokoll ist ein serieller Kommunikationsbus mit einer Master/Slave Struktur, dabei können mehre Slaves mit einem oder auch mehren Master über SDA und SCL miteinander kommunizieren und Daten austauschen. Die Abkürzung I²C, auch geschrieben IIC stammt von Begriff Inter-Integrated Circuit. Dieser wurde von Philips Semiconductors, heute NXP Semiconductor entwickelt, als einfacher bidirektionaler 2-Draht Bus zur Kommunikation zwischen integrierten Schaltkreisen. Dieser 8 Bit orientierter Bus besitzt eine Standardgeschwindigkeit von 100kbit/s, kann aber bis auf 5 Mbit/s in unidirektionalen Modus erhört werden, wenn dies auch unterstützt wird. Viele Hersteller bezeichnen I²C auch mit TWI, dies ist essenziell dasselbe System nur gehen die Hersteller so einen Konflikt mit NXP aus dem Weg. Das liegt aber nur am Logo, das von NXP für I²C verwendet wird.

2.1.1 I²C Hardwareebene

Wie schon erwähnt wurde, verwendet der der I²C-Bus zwei Leitungen, einmal SDA (Serial Data) und SCL(Serial Clock). Alle Teilnehmer sind parallel an diesen Bus angeschlossen und jeder dieser Teilnehmer besitzt eine eindeutige Adresse, mit der er angesprochen werden kann. Hierbei wird häufig Peripherie an einen Mikrocontroller angeschlossen, dass kann wie hier ein Display sein, aber auch sonstige Sensoren und Bausteine. Dabei wird für den Master meist ein Mikrocontroller verwendet, da dieser die Kommunikation koordiniert und den Teilnehmern meist die nötige Intelligenz dazu fehlt, dennoch verfügt I²C über die Multi-Master-Eigenschaft.

Durch die Verwendung einer Taktleitung vereinfacht sich die Implementierung auf der physikalischen Ebene sehr, da keine präzisen Taktgeber benötigt werden, die auf die Daten synchronisiert werden müssen. Dieser Takt wird vom Master erzeugt und bestimmt auch so die Datenrichtung. Mit der eindeutig Adresse können die Teilnehmer angesprochen und Daten ausgetauscht werden.

Auf der physikalischen Ebene werden die beiden Signalleitungen SDA und SCL mittels Pull-Up-Widerständen gegen die Versorgungsspannung, in diesem Fall, 3,3V gezogen. Somit sind diese im Ruhezustand auf einem HIGH-Pegel. Die einzelnen Teilnehmer besitzen Open-Drain bzw. Open-Collector Anschlüsse, die zusammen mit den Widerständen eine Wired-AND-Verknüpfung bilden. So ist es möglich das mehrere Teilnehmer eine Leitung zur bidirektionalen Kommunikation nutzen können. Wie ein solcher Aufbau schematisch aussieht, ist in Abbildung 1 dargestellt, hier wurde jeweils nur ein Master(Controller) und Slave(Display) angeschlossen, natürlich lässt sich das fast beliebig erweitern.

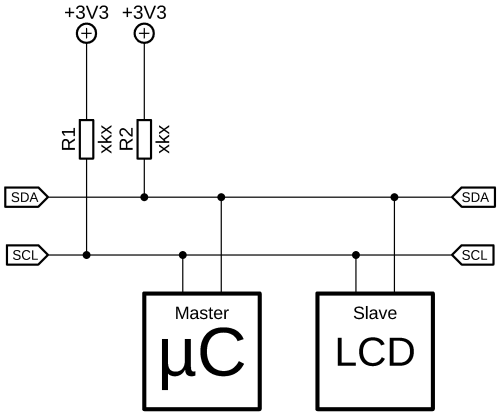

Abbildung 1: I²C-Bus Schaltplan

Zur Dimensionieren der Widerstände gibt es einiges zu beachten, nicht nur ist der Pegel, sondern auch die Kapazität der Signalleitungen und Eingangskapazitäten entscheidend, die zusammen mit den Widerständen die Steigzeit beeinflusst. Denn sonst kann es zu Fehlern in der Kommunikation führen, dies ist natürlich noch abhängig von der gewünschten Datenrate. Das System verhält sich grundlegend wie ein RC-Tiefpass, wobei die Widerstände das R bilden und der Kapazitätsbelag der Signalleitungen mit den Eingangskapazitäten zusammen das C.

Um die Widerstandswerte zu ermitteln können diese aus einem Diagramm oder genauer die obere und untere Grenzen berechnet werden. Texas Instruments haben hierzu auch einen Applikationsbericht veröffentlicht aus dem die folgenden Formel 2.1 und 2.2 entnommen wurden.[1] Als Datenrate wurde 100kbit/s gewählt, für die Versorgungsspannung sind 3,3V angesetzt. Die Kapazität lässt sich nur abschätzen, aber 100pF sind hier auch angemessen für diesen Testaufbau.

[1] Ins15, S. 2.

$$R_p(max) = \frac{t_r}{(0,8473 \times C)} = \frac{1\mu s}{(0,8473 \times 100pF)} = 11,8k\Omega \tag{2.1}$$

$$R_p(min) = \frac{V_{CC} - V_{OL}(max)}{(I_{OL})} = \frac{3,3V - 0,4V}{3mA} = 967\Omega \tag{2.2}$$

Nach Berechnung kommen Pull-Up-Widerstände im Bereich von 1kΩ bis 10kΩ in Frage, das deckt sich auch mit typischen Werte, die für einen I²C-Bus verwendet werden. Der $R_p(min)$ wird hauptsächlich durch den gewünschten Stromverbrauch und den maximal zulässigen Eingangsströmen bestimmt. Dabei ist darauf zu achten, dass I²C nicht für eine längere Kabelstrecke geeignet ist, durch die hochohmigen Abschlüsse kann es leicht zu Fehlanpassungen kommen, die dadurch schlechte Signalintegrität kann bis zum Totalausfall führen.

Moderne Mikrocontroller besitzen in der Regel die Möglichkeit interne Widerstände zu aktivieren, in vielen Fällen sind diese vollkommen hinreichend für eine solche Kommunikationsschnittelle und erspart das Beschalten von zusätzlich Komponenten, auch haben viele I²C Displays eigene Pull-Up-Widerstände verbaut. Das muss natürlich alles bei der Auslegung berücksichtigt werden. Hier sind die Pull-Up-Widerstände ausreichend klein um mit einer Taktfrequenz von 100kHz arbeiten zu können. In Abbildung 2 sind die Zeitleichenverläufe der Signale SDA und SCL dargestellt um die Steigzeiten zu prüfen.

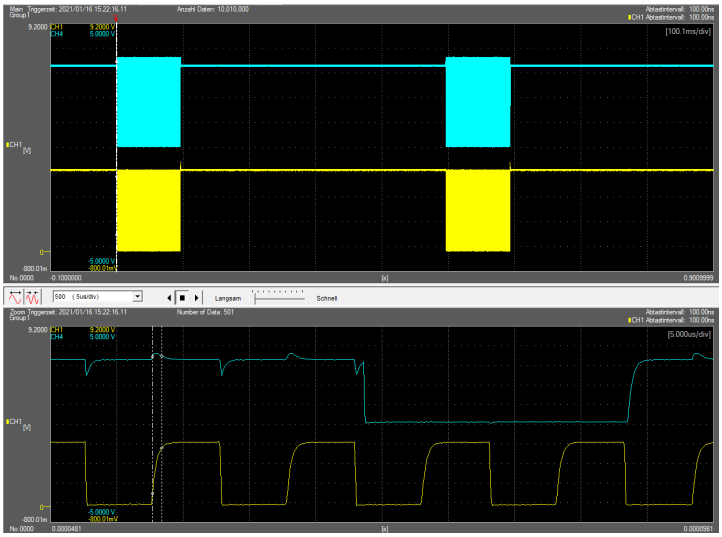

Abbildung 2: Anstiegszeit(700ns) und Übersprechen der SDA(blau) und SCL(gelb) Datenleitungen

Was im Signalverlauf, vor allem bei der SDA Leitung auffällt, sind kleine transiente Erscheinungen korrelierend mit den Übergängen des Taktsignales(SCL). Dies ist eine Folge der hohen Impedanz der Datenleitungen in Kombination mit der Steigzeit und der kapazitive Kopplung zwischen den nah miteinander verlegten Leitungen. Für den I²C-Bus ist das aber kein Problem, da die Daten beim Übergang von 0 auf 1 des Tanksignales erst und nur dann eingelesen werden, auch einflankentaktgesteuert genannt. Das könnte nur bei einem LOW-Datenbit einen Fehler verursachen, wie aber in der Abbildung 2 zu sehen ist, sind dort keine Transienten vorhanden nur bei HIGH-Bits, wo es zu keinem versehentlichen kippen des Bits kommen kann. Dennoch sollte diese Eigenschaft nicht überstrapaziert werden, um einen reibungslosen Betrieb zu gewährleisten.

2.1.2 I²C Softwareebene

Auf der Softwareebene gestaltet sich die Verwendung von I²C nicht weiter kompliziert. Viele Mikrocontroller besitzen bereits Hardwareunterstützung für I²C, so müssen dort nur einige wenige Register initialisiert werden, was meist schon durch die Verwendung von geeigneten Bibliotheken erledigt wird, auch stellen diese Funktonen zur Verfügung, die Daten senden und empfangen können. Es ist auch möglich I²C ohne Hardware-unterstützung zu verwenden, falls diese nicht vorhanden sind oder die Ports unbedingt anderweitig benötigt werden. Dazu ist es meist auch empfehlenswert Hardwareinterrupts zu verwenden, doch wird eine Software Realisierung deutlich mehr Programmlaufzeit in Anspruch nehmen.

2.2 I²C Display

Auf dem Markt sind viele Varianten solcher Displays verfügbar, nicht nur verschiedene Schnittstellen, sondern auch Größen und Display Technologien. Dabei gibt es viele Faktoren, die abhängig von der späteren Anwendung beachtet werden müssen. Kriterien hierfür waren eine gut lesbare alphanumerische Ausgabe, I²C Schnittstelle und ein leicht zu beschaffendes Model. Das in Abbildung 3 gezeigte Display ist der hierfür ausgewählte Typ. Es ist nicht nur ein sehr häufig verwendetes und verfügbares Modell, sondern bieten auch einiges an Display Fläche. Diese traditionellen LCDs werden meist mit der Angabe versehen wie vielen Zeilen mal Spalten sie anzeigen können und welches Ansteuerungs-IC für das Panel selbst verwendet wird, in diesem Fall ist es ein 4x20 mit einem HD44780.

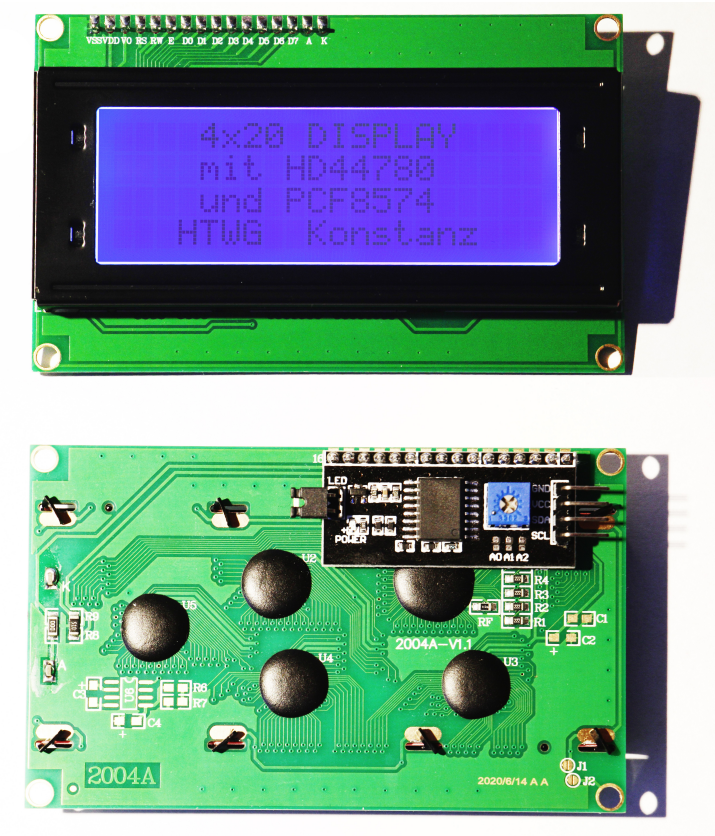

Abbildung 3: 4x20 HD44780 Display mit PCF8574 Adapter Platine

Displays mit einem HD44780 haben in der Regel ein parallel Schnittstelle, diese lässt sich entweder in einem 8-Bit oder 4-Bit Modus verwenden, je nach verfügbarer Peripherie. Dabei gibt es noch ein paar weitere relevanten Steuersignale, die in der Tabelle 1 aufgeführt und beschrieben sind.

Name	Typ	Beschreibung
Vcc	V+	Versorgungsspannung
Vss	GND	Masse
RS	I	0 = Befehl, 1 = Daten
R/$\overline{W}$	I	0 = schreiben, 1 = lesen
E(Enable)	I	Starts data read/write
D0	I/O	Daten-Bit 0 LSB
D1	I/O	Daten-Bit 1
D2	I/O	Daten-Bit 2
D3	I/O	Daten-Bit 3
D4	I/O	Daten-Bit 4
D5	I/O	Daten-Bit 5
D6	I/O	Daten-Bit 6
D7	I/O	Daten-Bit 7 MSB

Tabelle 1: HD44780 Pinbelegung

Vorteile eines integrierten Controllers ist auch, dass die intern gespeicherten ASCII-Zeichenspeicher(CGRAM) leicht abgerufen werden können, ohne das diese Pixel für Pixel auf das Display über den I²C-Bus geschrieben werden müssen. Auch können die ersten acht ASCII Adressen mit eigen Zeichen befüllt werden, die dann im CGRAM des HD44780 gespeichert sind.[2]

2.3 PCF8574 I²C Wandler

Nun muss aber diese parallele Schnittstelle in eine serielle umgewandelt werden, dazu wird meist ein PCF8574 verwendet. Dieser IC ist ein I²C 8-Bit Port Expander, da mit 8-Bit keine Anschlüsse mehr frei wären für die Steuersignale, muss der HD44780 im 4-Bit Modus betrieben werden. Dazu muss später das Daten-Byte in ein HIGH- und LOW-Nibble geteilt und gesendet werden. Dafür werden die drei Steuerleitungen und die oberen 4-Bit an den PCF8574 angeschlossen, die weiteren Datenbits werden auf Masse gelegt. Beim Erwerb eines HD44780 als I²C Display ist häufig eine Adapterplatine mit PCF8574 angelötet und kann so gleich mit passender Software über I²C angesteuert werden.

[2] HIT99, S. 13.

Die Anschlüsse des HD44780 sind wie in Tabelle 2 an den Pins des PCF8574 angeschlossen, auch die jeweils dazugehörigen Datenbits des PCF8574, die später angesprochen werden müssen, um sie dementsprechend LOW oder HIGH zu setzen.

PCF8574	**HD44780**	**Beschreibung**
Bit-0	RS	0 = Befehl, 1 = Daten
Bit-1	R/$\overline{W}$	0 = schreiben, 1 = lesen
Bit-2	E(Enable)	starte lesen/schreiben
Bit-3	Hintergrundbeleuchtung	0 = aus, 1 = ein
Bit-4	D4	Daten-Bit 0
Bit-5	D5	Daten-Bit 1
Bit-6	D6	Daten-Bit 2
Bit-7	D7	Daten-Bit 3

Tabelle 2: PCF8574 Anschlussbelegung für 4-Bit Modus

Die Ansteuerung des PCF8574 ist durchaus trivial, dabei wird bei korrekter Ansprache der Adresse des ICs über I^2C das Datenbyte in ein Schieberegister geschrieben, danach dementsprechend auf die Ausgänge gelegt. Daher auch die Bezeichnung I^2C Port-Expander, es können auch Daten ausgelesen werden, was in dieser Anwendung aber nicht notwendig ist. Des weiteren ist es möglich bis zu acht verschiedene Adressen am PCF8574 direkt einzustellen, so wäre es möglich acht verschiedene Displays am selben I^2C-Bus zu betreiben, was diese Implementierung hier auch so zulässt.

2.4 Matlab

MATLAB ist eines der bekanntesten Programme, gerade in den Bereichen numerische Berechnungen, Regelungstechnik und wissenschaftliche Datenverarbeitung. Zu den einfachen Berechnungsmöglichkeiten in der Konsole bietet MATLAB auch die Programmierung von eigenen Skripten. Auch lassen sich leicht graphische Ausgaben generieren, welche sich gerade zur Visualisierung von Daten und Messungen gut eignen. MATLAB selbst leitet sich aus den Namen MATrix LABoratory ab. Zudem bietet die Firma The Math Works Inc. aus den USA zahlreiche Erweiterungen und Pakete zur Hardware und Software Unterstürzung an. Auch stellen Nutzer eigens erstelle Bibliotheken und Pakete zu Verfügung, die von jedem verwendet und auch erweitert werden können. Somit besitzt MATLAB viele Möglichkeiten das Arbeiten erheblich zu erleichtern oder gar erst möglich zu machen.

2.4.1 Simulink

SIMULINK ist eine Erweiterung der Software MATLAB, es ist ein grafikorientiertes Softwaretool zur Modellierung und Simulation von linearen und nichtlinearen mathematischen Systemen, SIMULINK kann auf eine Vielzahl von Zusatz Bibliotheken und Paketen zurück greifen.

2.4.2 Code Generation mit Simulink Coder und Embedded Coder

Mit der Toolbox für SIMULINK Simulink Coder und Embedded Coder können MATLAB SIMULINK Modelle in andere Programmiersprachen übersetzt werden. Dabei können verschiedene so genannte Targets ausgewählt werden, hierfür müssen System-Target Dateien angelegt werden, diese Dateien dienen als Makefile zur Generierung des Codes, dort werden Dateipfade, Abhängigkeiten und Compiler definiert. Es gibt auch speziell zugeschnittene Konfigurationsdateien, die für die C2000 Mikrocontroller von MATLAB und TI zu Verfügung gestellt werden, diese sind meist im XML Format. So ist es möglich ohne umfangreiches Einarbeiten schnell funktionierenden Code mit MATLAB SIMULINK zu erzeugen.

2.5 TI Launchpad F28069M

Das TI Lunchpad F28069M ist eine Entwicklungsplattform von Texas Instruments, als Mikrocontroller ist ein C2000 Piccolo MCU F28069M verbaut, der auf dieser Platine mit Peripherie wie USB, Debugger, LEDs, Steckverbinder der Ports und viele weiter Schnittstellen, wie auch an den vielen Stift- und Buchsenleisten in Abbildung 4 zu erkennen ist, ausgestattet wurde.[3] So wird ermöglicht, dass ein Testen und Aufbauen von Prototypen ohne aufwendige Platinenherstellung möglich ist, auch kann Software so schnell evaluiert und geprüft werden. Auch bietet es gerade für studentische Labore den Vorteil, dass diese universellen Platinen sich häufig wiederverwenden lassen und auch leicht zu beschaffen sind.

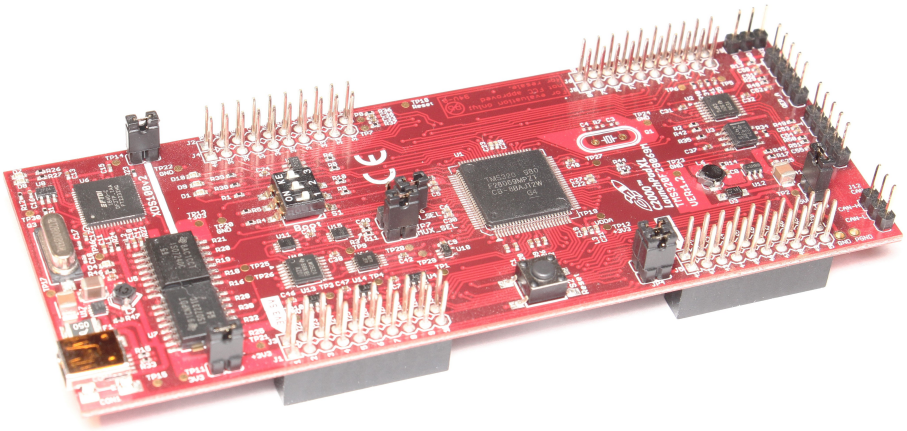

Abbildung 4: TI Launchpad F28069M Leiterplatte

Auch existiert eine sehr gute Softwareunterstützung innerhalb von MATLAB SIMULINK, hierdurch können digitale Regler mit direktem ansteuern von Leistungstufen realisiert werden, z. B. über PWM Kanäle, dies ist gerade für den Bereich der Leistungselektronik sehr interessant. Durch die hohe Taktfrequenz von 90 Mhz und der 32-Bit Architektur lassen sich auch komplexe Programme in Echtzeit ausführen, zudem kommt noch die Hardware Beschleunigung, gerade für kritische Bereiche wie Kommunikation, PWM, ADC und Komparatoren hinzu.

[3] Ins19, S. 2.

3 Implementierung

In diesem Kapitel werden die Aspekte der Implementierung und Entwicklung behandelt und erklärt.

3.1 Ansatz

Zuerst wird evaluiert welche Implementierungen der I^2C Kommunikation in MATLAB SIMULINK möglich sind. Es gibt zwei Methoden die dies umzusetzen können, einmal ist es möglich mit der C2000 Bibliothek in SIMULINK die Kommunikation mit Blöcken aufzubauen oder durch schreiben einer eigenen Bibliothek oder zusätzlich das hinzufügen dieser in die von TI bereitgestellten Bibliothek. Letzteres hat sich als deutlich komplizierter herausgestellt, vor allem durch die kaum verfügbare Dokumentation hätte dies zeitlich nicht realisiert werden können. Hierfür müssen unter andrem TLC Dateien angelegt werden, die dann mit Hilfe des Code Generator den dementsprechenden C Code aus SIMULINK für den Mikrocontroller generieren. Daher wurde die Display Ansteuerung mithilfe der verfügbaren Blöcke aus der Bibliothek in SIMULINK umgesetzt.

3.1.1 Toolkette

Die in der Tabelle 3 aufgeführten Programme und Erweiterungen wurden in dieser Arbeit verwendet, dabei waren die Pakete MATLAB SIMULINK von MathWorks und der Code Composer Studio der Firma Texas Instruments essenziell zur Umsetzung. In MATLAB werden diese Pakete auch Add-Ons genannt. In MATLAB ist es möglich durch die Eingabe von "ver -support" in die Kommandozeile die Installierten Add-Ons und deren Versionen sich anzeigen zu lassen. Die Funktion der hier erarbeiteten Implementierung sollte nicht all zu stark von den verwendeten Versionen abhängig sein.

Hersteller	**Name**	**Version**
Texas Instruments	Code Composer Studio	10.2.0.00009
	controlSUITE	v3.4.9
MATLAB	MATLAB R2020b	9.9
	SIMULINK	10.2
	Embedded Coder	7.5
	MATLAB Coder	5.1
	Simulink Coder	9.4
	Symbolic Math Toolbox	8.6

Tabelle 3: Software Toolkette

3.2 Initialisierung

Zur Initialisierung ist es sinnvoll eigens hierfür Blöcke anzulegen, die zur Übertragung der notwendigen Informationen dienen, die im Folgenden weiter erläutert werden.

3.2.1 I²C 4-Bit Datenübertragung

Durch die Verwendung des I²C Blockes aus der Bibliothek, entfällt dessen separate Implementierung in SIMULINK, so können die Rohdaten direkt damit versendet werden. In Abbildung 5 ist dieser I2C Transmit Block abgebildet, zusammen mit den notwendigen Parametern. Hierbei ist anzumerken das der 7-Bit Adressraum verwendet wird und acht Datenbits, der Statusausgang wird terminiert, dieser könnte aber auch zur Fehlerabfrage verwendet werden. Dem Block werden die zu übertragenden Daten aus den darüberliegenden Strukturen übergeben und auch die dazugehörige Adresse, so ist es möglich Displays mit beliebigen Adressen anzusprechen. Somit muss nun die richtige Sequenz der Befehle zur Initialisierung des Displays ermittelt werden. Hierzu liefert das Datenblatt des HD44780 Aufschluss darüber, welche Befehle dafür notwendig sind.

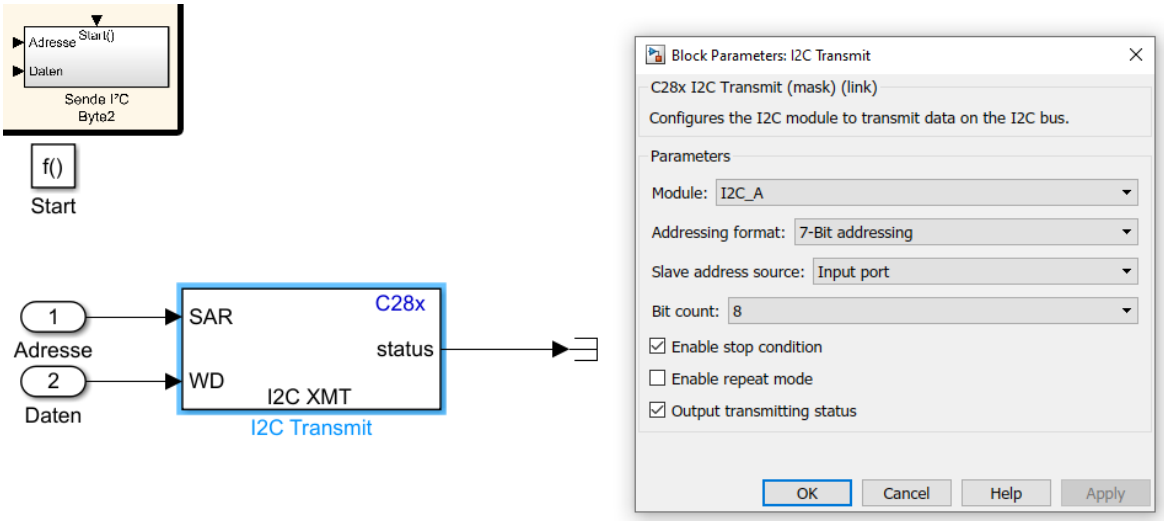

Abbildung 5: I²C Block mit Parametern

Da zur Konvertierung der seriellen I²C Daten ein Konverter(PCF8574) eingesetzt wird, muss dies in der Art und Weise wie die Datenübertagung stattfindet berücksichtigt werden. So sind für jedes zu übertragende Nibble drei I²C Pakete notwendig. Da der HD44780 die Daten erst dann übernimmt, wenn der Enable Pin von LOW auf HIGH

wechselt, müssen die Daten zuerst angelegt werden. Danach erneut mit dem zusätzlichen HIGH-Enable. Zuletzt muss der Enable wieder zurück genommen werden, bei gleichbleibenden Daten, was dem letzten, der drei Pakete entspricht. Diese Struktur wurde wie in Abbildung 6 zu sehen umgesetzt, hier wird zu den Datenbits im zweiten Schritt, mit einem Bitweisen ODER das Enable-Bit gesetzt. Die unteren vier Bits sind den Steuerleitungen zugeordnet und die oberen vier der Daten den Datenleitungen des HD44780.

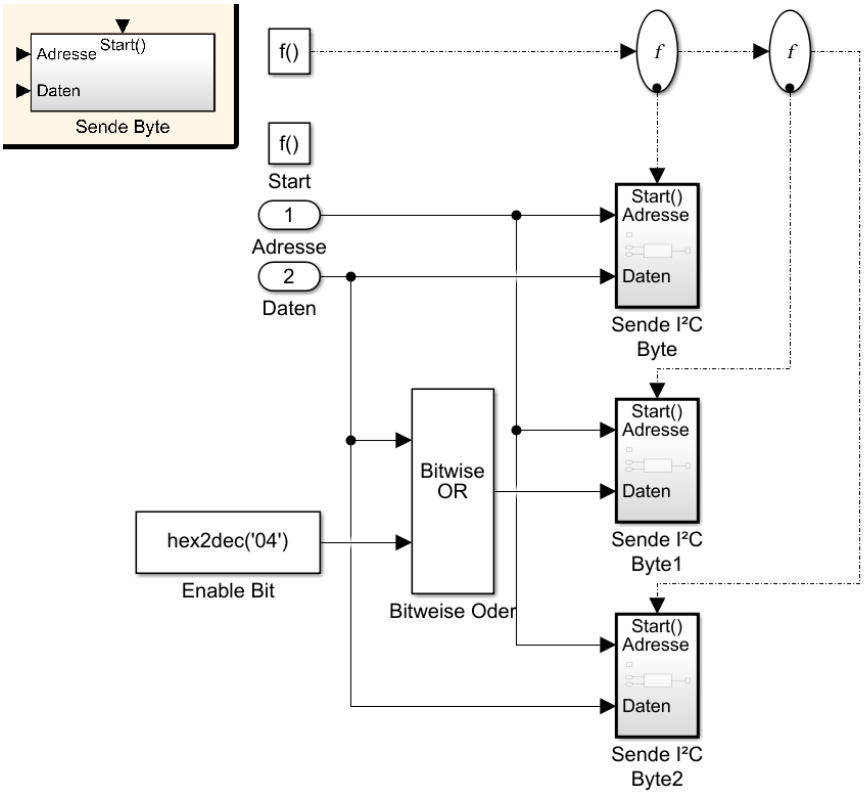

Abbildung 6: I²C Block zur Übertragung von Daten mit Enable Übergang

Zu sehen sind auch die ovalen mit Serifen f dargestellten "Funktion Call Split" Blöcke, diese werden hier verwendet, um den zeitlichen Ablauf des Programmes zu koordinieren, die ausgehend vom "Funktion Call Generator" ausgelöst werden. Es ist auch möglich einen Multiplexer zu verwenden, um den Funktionsaufruf zu verteilen, doch ist der Split Block deutlich übersichtlicher, auch werden eventuelle Fehler beim Aufrufen konkreter aufgelistet. Auch finden in der Übertragungsstruktur in Abbildung 6 der Funktionsblock "Sende I²C Byte" aus Abbildung 5 Anwendung zur Übertragung der I²C Bytes.

In Abbildung 7 ist die Übertragung eines Befehl Nibbels dargestellt, zusätzlich auch die durch den PCF8574 erzeugten Datenbits am HD44780 Display. Dabei gibt es einige wichtige Beobachtungen, die diese dreifach Struktur der Übertragung illustriert. Zu Beginn sind die Datenbits noch vom vorherigen Nibbel auf 0x00 gesetzt. Das erste Paket erhält die Daten 0x10, was am Ende der Übertragung dafür sorgt, dass das Datenbit D4 am HD44780 auf HIGH gesetzt wird.

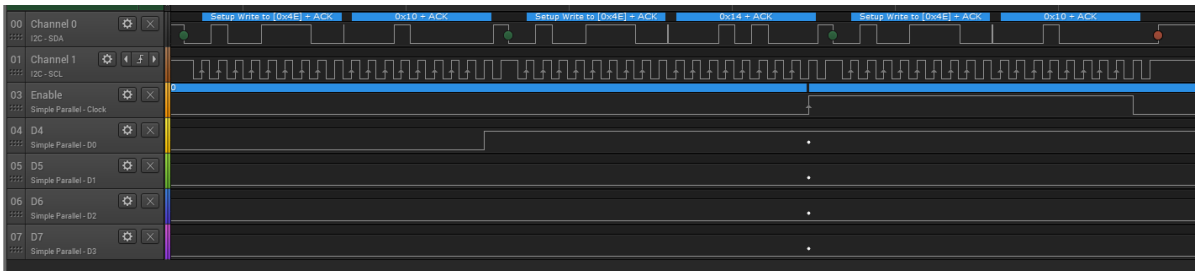

Abbildung 7: Aufzeichnung eines Nibbels des I²C-Bus und prallen Bits am HD44780

Damit nun auch der HD44780 diesen Befehl einlesen kann, muss der Enable Pin von LOW auf HIGH gesetzt werden, das erfolgt mit den nächsten Daten Nibble 0x14. Wie zuvor bleibt D4 auf HIGH, doch kommt nun der Enable Pin hinzu. Zuletzt wird das Enable wieder zurückgenommen, mit dem schon anfänglich gesendeten Daten Nibble 0x10. Da das einlesen mit Enable nicht zwangsläufig flankengesteuert ist, ist der letzte Schritt mit zurücksetzen von Enable notwendig, damit beim Anlegen der nächsten Daten keine Fehleinlesungen stattfinden.

3.2.2 Initialisierungssequenz des HD44780

Nachdem nun die Grundfunktionen zur Übertragung von Rohdaten, als auch die von Daten mit dem hierfür notwendigen Enable Übergang implementiert worden sind, können nun die eigentlichen Befehle ermittelt werden. Hierzu werden aus dem Datenblatt des HD44780 die Befehle und deren Sequenz entnommen. Das Display initialisiert sich zwar selbst mit Standardeinstellungen, diese sind aber hier nicht hinreichend, auch ist das nur der Fall, wenn die Spannungsversorgung neu angelegt wurde, so kann auch nicht davon ausgegangen werden, dass dieser richtig konfiguriert ist. Daher ist es notwendig alle nötigen Einstellungen vorzunehmen. Aus dem Datenblatt wurde die in der Abbildung 8 dargestellt Initialisierungssequenz zusammengestellt, deren Ablauf wird im folgenden erklärt.[4] Anzumerken ist, dass die Befehle generell ein 8-Bit Format haben, werden aber durch das Verwenden des 4-Bit Schnittstelle in zwei geteilt, daher besteht jede Einstellung aus zwei hintereinander folgenden 4-Bit Befehlen für das HIGH und LOW Nibble. Sie enthalten auch jeweils die zusätzlichen Steuersignale des HD44780.

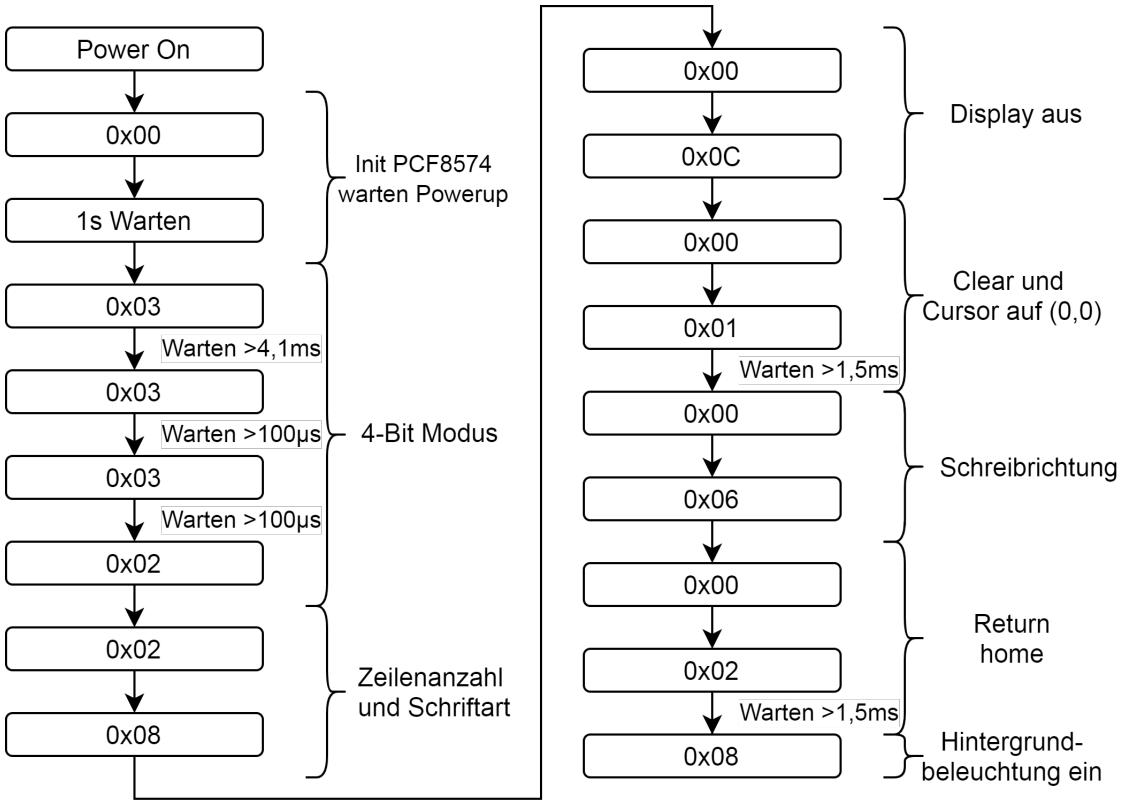

Abbildung 8: 4x20 Display Initialisierungsablauf

[4] HIT99, S. 39-46.

Die Initialisierungssequenz kann grob in 8 Teilsequenzen unterteilt werden, die unterschiedlichste Aufgaben übernehmen. Im Normalfall wird das Display mit dem F28069M zusammen eingeschaltet, da das Display einen Moment benötigt um sich selbst zu initialisieren und auch dem Controller etwas Zeit gegeben wird, um eigene Routinen im Start-up abzuarbeiten, wird zum Beginn der Kommunikation eine Warteperiode von einer Sekunde veranschlagt. Auch werden die Pins des PCF8574 durch das Byte 0x00 auf LOW gesetzt.

Da zur Ansteuerung nur eine 4-Bit Schnittstelle zur Verfügung steht muss der HD44780 zuerst in den 4-Bit Modus gebracht werden. Hierzu wird aufeinanderfolgen dreimal der Befehlt 0x03 gesendet, hinzu kommt dass die Wartezeiten eingehalten werden müssen, diese wurden auch sequentiell in SIMULINK so umgesetzt.(s. Abbildung 9)

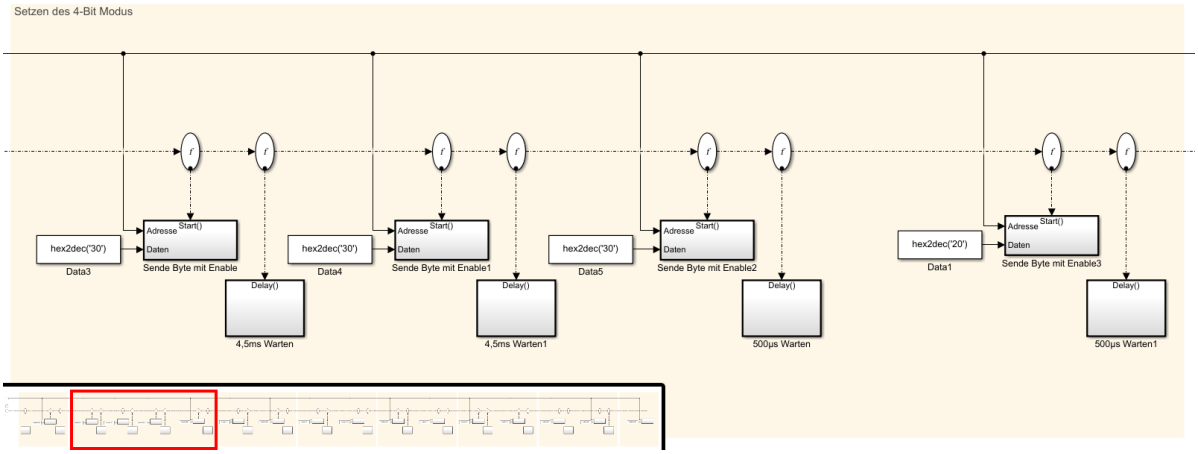

Abbildung 9: 4-Bit Modus Initialisierungssequenz

Die Wartezeiten sind z.B. 500µs, diese reduzieren sich da die Übertragung asynchron zum Programmablauf stattfindet, somit muss die Zeit der Übertragung von der Wartezeit abgezogen werden, so erhält man mindestens 200µs, was die Bedingung von >100µs aus dem Datenblatt erfüllt. Dies ist auch in Abbildung 10 zu sehen, dort wurde der Abstand zwischen den zwei letzten 0x03 und 0x02 Befehlen der aufgezeichneten 4-Bit Initialisierung gemessen. Zu erkennen ist auch die Dreifachstruktur aus 0x30, 0x34 und 0x30 der Übertragung zum eintakten der Daten über den Enable Pin, wie sie auch in Abbildung 6 zuvor definiert wurde.

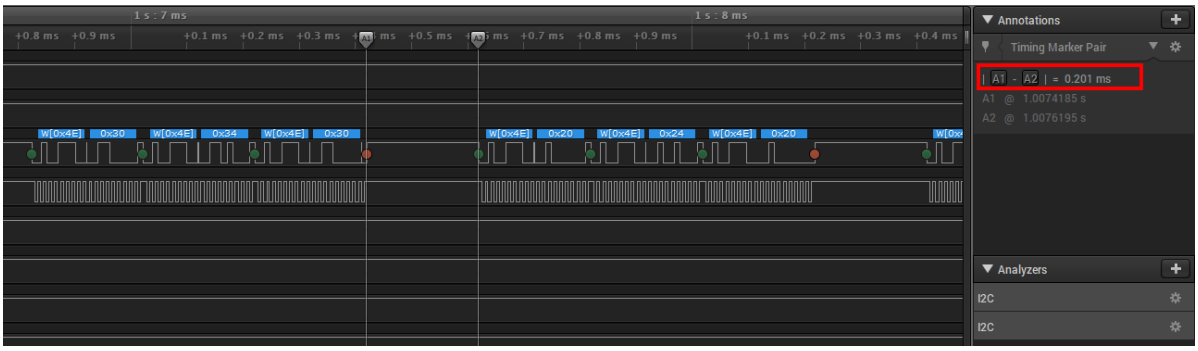

Abbildung 10: 200µs Wartezeit zwischen zwei Befehlen

Nach dem Setzen des 4-Bit Modus können nun auch die weiten Befehle im 4-Bit Format übertragen werden. Für den nächsten Befehl zur Einstellung der Schrift und Zeilenanzahl sind dabei nur die oberen zwei Bits ausschlaggebend. Die unteren Bits werden für diesen Befehl ignoriert. Durch das Senden von 0x28 erhalten wir den binären Befehl 00101000, das entspricht einer Zeilenanzahl von zwei und einer Schrift mit der Matrixgröße von 5x8 Pixel.[5] Nun scheinen zwei Zeilen für einen 4x20 Display nicht korrekt zu sein, dies ist aber nicht der Fall, da sich eine Zeile über zwei Zeilen des eigentlichen Display erstrecken. Eine typische Zeilenbreite von 40 war früher üblich und so wurde der Treiber auch darauf ausgelegt. Das Übergreifen in eine der nächsten Zeile, muss bei der Ansteuerung später berücksichtigt werden, darauf wird dann noch genauer eingegangen. Bei Verwendung eines anderen Display Formats, muss gegebenenfalls dieser Befehl angepasst werden. Das setzen der Zeilenanzahl und Schriftart kann nur jetzt gemacht werden, zu einem später Zeitpunkt ist das nicht mehr möglich.[6]

Mit dem Befehl 0x0C wird die Display-Steuerung deaktiviert, das beinhaltetet den Cursor und dessen Blinkfunktion, falls diese Funktionen benötigt werden, können sie durch das Setzen der unteren 3 Bits aktiviert werden. Danach soll ein Display clear ausgelöst werden, was durch 0x01 erreicht wird, dabei wird auch die DDRAM Adresse auf 0 gesetzt. Der DDRAM ist der Speicher, der die Zeichen speichern die auch zeitgleich auf dem Display zu sehen sind, auch Frame buffer genannt. Zu beachten ist, dass dieser Befehl deutlich länger benötigt um ausgeführt zu werden, daher die Mindestwartezeit von 1,5ms. Nun ist es noch wichtig die Schreibrichtung zu setzen, durch 0x06, dabei sind die unten zwei Bits wieder entscheidend. Das LSB entscheidet, ob das gesamte Displayinhalt verschoben wird, wenn Zeichen darauf geschrieben werden. Mit dem zweiten

[5] HIT99, S. 27ff.
[6] HIT99, S. 46.

Bit wird festgelegt, ob diese dann nach links(0) oder rechts(1) propagieren. Auch wird dadurch definiert ob die DDRAM Adresse dekrementiert(0) oder inkrementiert(1) wird. Der "Retrun home" Befehl 0x02 ist nicht zwangsläufig notwendig, dennoch ist es hilfreich, dabei werden eventuelle Shifts und auch final die DDRAM Adresse auf 0 zurück gesetzt. Hier ist auch wieder die längere Wartezeit zu berücksichtigen. Zuletzt setzen wir noch das Bit-3, das die Hintergrundbeleuchtung einschaltet, damit ist die Initialisierung des Displays abgeschlossen und es kann damit begonnen werden Zeichen zu senden.

Bisher wurden nur die Funktion Call Split Blöcke erwähnt und wie diese ermöglichen Unterblöcke und Funktionen sequenziell aufzurufen. Nun müssen diese aber noch mit dem korrekten Triggersignal versorgt werden. Das hängt davon ab, ob diese einmal oder mehrmals ausgelöst werden sollen. Bei der Initialisierung ist es wichtig, dass diese nur einmal und zu Beginn ausgeführt wird. In Abbildung 11 ist die Übergabe der Trigger an die Initialisierungssequenz zu sehen. Der Parameter Block "Adresse" ist nicht weiter besonders, es handelt sich um eine ordinäre Weitergabe eines Parameters in ein Subsystem, hier ist es die Adresse des I²C Slaves.

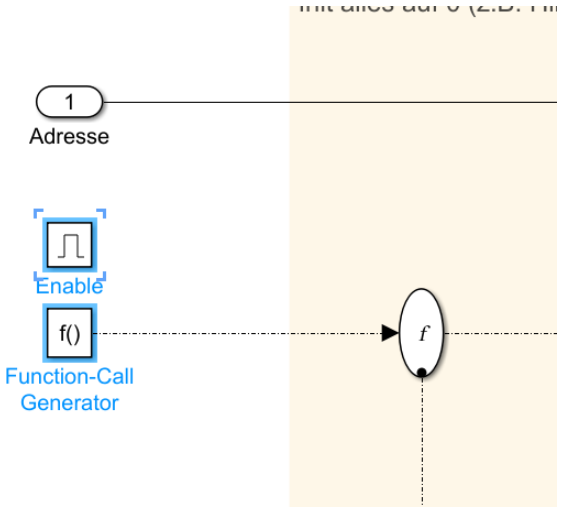

Abbildung 11: Funktionsaufruf und Weiterleitung mit Enable

Durch den Enable Block wird aus dem Subsystem ein "enabeld subsystem", ein solches System hat die Eigenschaft, dass es bedingt vom Zustand des Enable-Eingangs, zu jedem größeren Zeitschritt ausgeführt wird.[7] Es können noch einige weitere Einstellungen vorgenommen werden, z. B. ob die Werte der letzten Ausführung beibehalten oder

[7] TheoJ.

zurück auf deren initialisierten Werte gesetzt werden. Zusammen mit dem "Function-Call Generator" können so, wie der Name schon verrät, die benötigen Funktionsaufrufe erzeugt und gesteuert werden, um den Ablauf der Subroutinen auszulösen. Ohne den Enable könnte man diesen nicht so ohne weiteres von außen auslösen und steuern. Er besitzt auch zwei wichtige Parameter, zum einen wie viele Iterationen durchlaufen werden sollen, hier wird nur eine benötigt und auch mit was für einer Abtastrate die Aufrufe generiert werden sollen. Mit "-1" für die Abtastrate wird diese aus der darüber liegenden Struktur vererbt. Auch sorgt es dafür, dass die gesamte Struktur durchlaufen wird und nicht mehrere unterschiedliche Abtastraten verwendet werden, was zu inkonsistenten Daten führen kann.

Nun muss ein geeignetes Signal generiert werden, um den Enable-Eingang korrekt anzusteuern. Im Falle der Initialisierung soll dies nur einmal zu Beginn des Programmes erfolgen. Durch die in Abbildung 12 dargestellte Verschaltung wird dies erreicht.

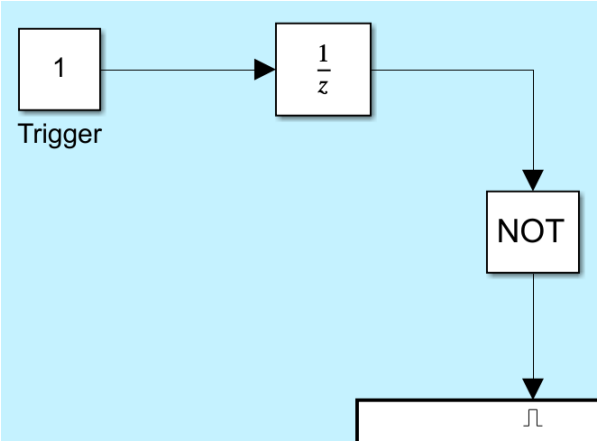

Abbildung 12: Initialisierung Triggerquelle

Dabei liefert der Constant Block mit dem Namen "Trigger" das dafür nötige Signal, dieser ist mit einer Abtastfrequenz von "-1" konfiguriert, damit wird diese wieder vererbt, es wäre auch möglich sie auf "inf" zu setzten, das ist aber nicht nötig, auch führen andere Werte zu Dateninkonsistenzen. Wie diese Abfolge von Blöcken genau das gewünschte Signal erzeugt, wird in ein Minimalbeispiel demonstriert. (s. Abbildung 13)

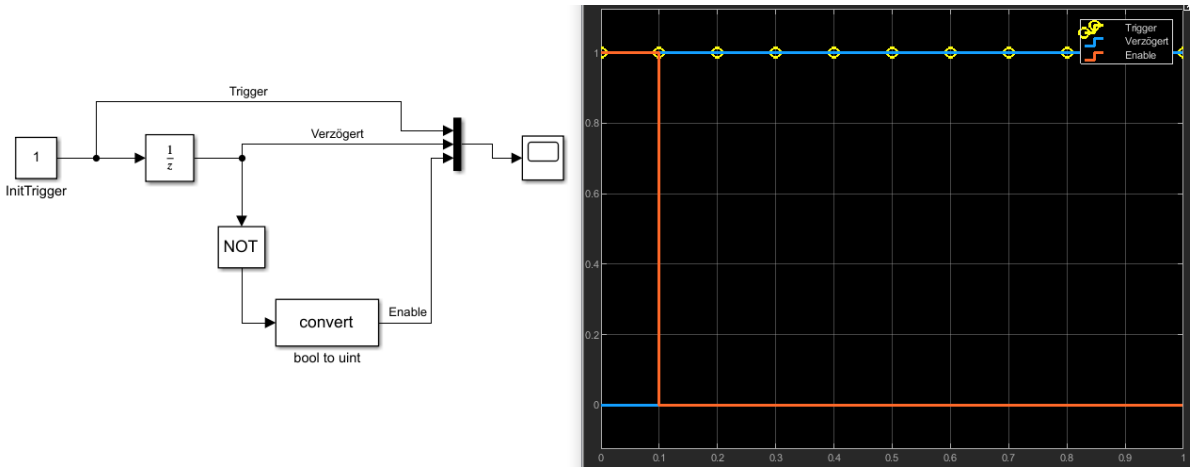

Abbildung 13: Demonstration der Auslöseschaltung mit Zeitlicherdarstellung vom Trigger(gelb) dem verzögertem(blau) und dem resultierenden Enable Signal(rot)

Zur Veranschaulichung wurde eine Abtastrate von 0,1 gewählt und für eine Sekunde lang simuliert. Der Constant Block liefert zu jedem Abtastzeitpunkt einen Wert von 1, da zu Beginn jeder Simulation oder Programmablauf dieser Wert 0 beträgt, wird er durch den Verzögerungsblock $\frac{1}{Z}$ um einen Abtastzeitschritt verzögert. Somit ist der Ausgang für den ersten Abtastzeitraum 0, zu erkennen in Abbildung 13 am verzögertem Signal. Damit nun der Initialisierungsblock auch richtig aktiviert wird, musst das Signal noch mithilfe des NOT Bausteines invertiert werden. Dabei wird auch der uint16 in einen booleschen Wert umgewandelt, weshalb auch im Beispiel das Signal für das Oszilloskop noch zuvor konvertiert werden muss. Die gesamte Initialisierung ist in einem SIMULINK Subsystemblock untergebracht, mit der daran angeschlossenen Triggerschaltung.(s. Abbildung 14)

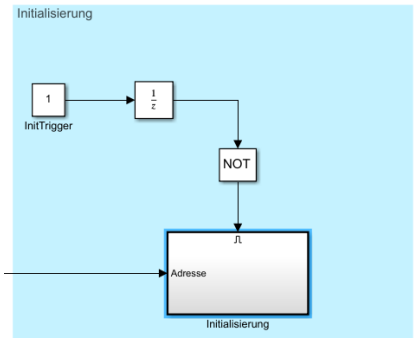

Abbildung 14: Initialisierungsblock in SIMULINK

3.3 LCD Schreibroutine

Nachdem das Display erfolgreich die Initialisierung der Displayeinheit durchlaufen hat, können nun auch Werte angezeigt und übertragen werden. Hierfür muss eine geeignete Implementierung gefunden werden, welche im folgenden genauer behandelt wird.

3.3.1 Senden eines Datenbytes

Wie auch schon in der Initialisierung müssen alle Daten oder Befehl Bytes in zwei 4-Bit Nibble aufgeteilt werden. Um die Verarbeitung zu erleichtern wurde die Übertragung eines kompletten Datenbytes umgesetzt. Das in Abbildung 15 dargestellt Subsystem realisiert diese Übertragung.

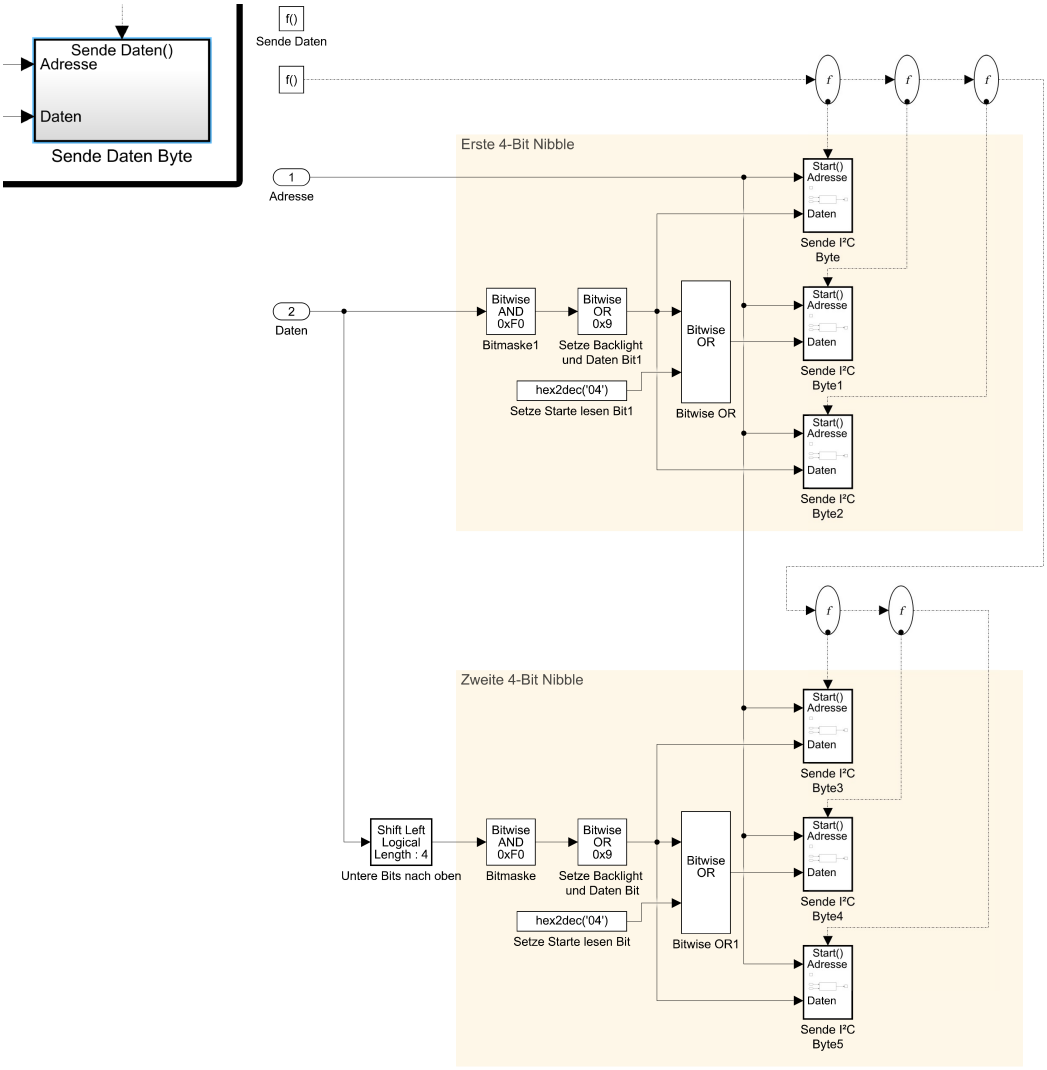

Abbildung 15: Struktur zur Übertragung eines Datenbytes

Die schon bekannte Dreifachstruktur zur Übertragung eines Nibbles aus der Initialisierung wurde auch hier wieder aufgegriffen, jedoch mit einigen kleinen Modifikationen zur Verarbeitung der Daten. Die durch den Port 2 weitergeleiteten Bytes müssen zuerst in die zwei Nibble aufgeteilt werden, dies wird dadurch erreicht indem die Daten mit einem Bitweisen UND maskiert werden, jeweils mit 0xF0. Für das zweite Nibble werden die unteren 4-Bits nach oben verschoben, um dann auf gleiche Weise maskiert zu werden. Die Maske ist hier nicht zwangsläufig notwendig, aber zu empfehlen da Bytes auch zyklisch verschoben werden können, somit ist eine zusätzliche Filterung sinnvoll. Danach unterschieden sich die beiden Strukturen nicht voneinander. Damit der HD44780 auch Befehle von Daten differenzieren kann, muss entsprechend der Tabelle 2 das Bit-0 gesetzt werden. Auch sollte das Bit für die Hintergrundbeleuchtung HIGH bleiben, da sonst die Beleuchtung ausgeschaltet wird, dann müsste man diese separat wieder einschalten was zu einem unerwünschten Flackern führen würde. Durch das verodern mit 0x9 wird genau das erreicht, binär betrachtet werden Bit-0 und Bit-4 immer auf HIGH gesetzt.

3.3.2 Konvertieren von Strings zur Übertragung der Zeilen

Jede Zeile des LCDs soll später separat in SIMULINK einem String-Vektor zugeordnet sein. Für jedes ASCII-Zeichen muss so ein Datenbyte übertragen werden, das Datenbyte einspricht im Wert dem des dazugehörigen ASCII-Zeichen. Da der interne Zeichenspeicher des HD44780 verwendet wird, muss die entsprechende Speicheradresse dafür übermittelt werden. Im Datenblatt des HD44780 ist eine Tabelle des internen Speichers abgebildet.(s. Abbildung 16)

Abbildung 16: Zeichen ROM und RAM Tabelle des HD44780[8]

Was die Umsetzung erleichtert ist, dass die Speicheradressen der Zeichen gleich des ASCII Werts ist. Somit müssen die Strings nur in ASCII umgewandelt werden. SIMULINK liefert einen passenden Block zur Umwandelung von Strings in einen ASCII-Vektor, da jede Zeile bis zu 20 Zeichen beinhalten kann, können wir den Ausgang des Konverters auf

[8]HIT99, S. 17.

20 festsetzten. Die fixe Länge des ASCII-Vektors erleichtert die Verarbeitung, da sonst das String Array mit leeren Zeichen aufgefüllt werden müsste. So wird automatisch der ASCII-Vektor mit Nullen aufgefüllt. In Abbildung 17 ist die Implementierung der Umwandlung in ASCII und das auftrennen der Zeile in einzelne Zeichen zu sehen.

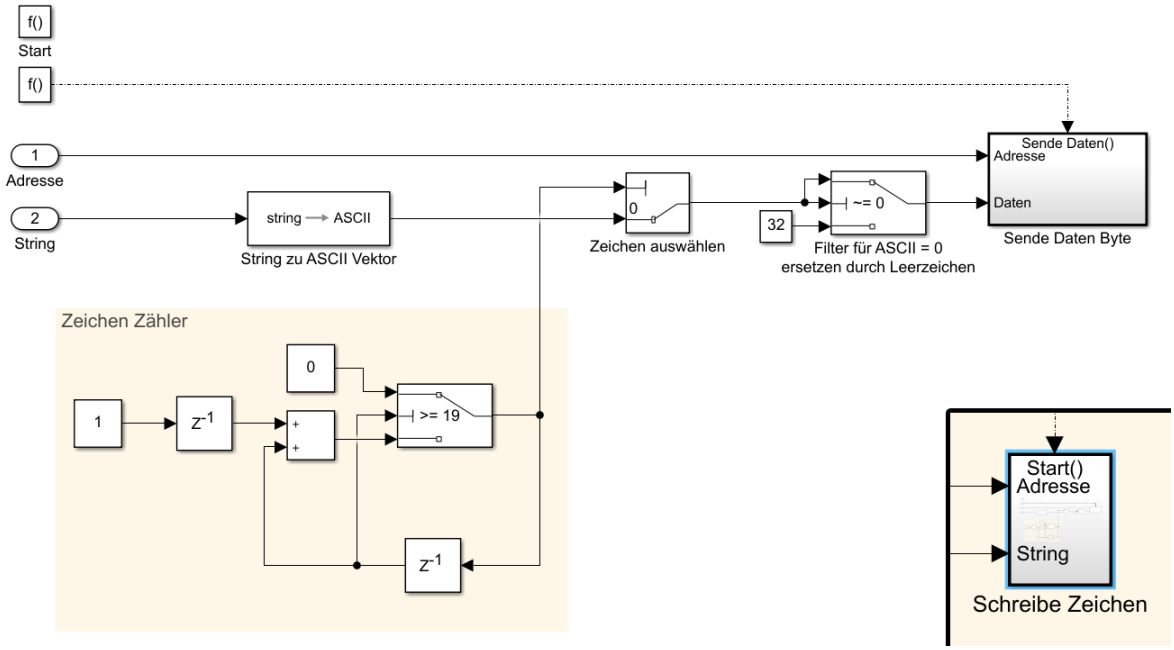

Abbildung 17: Umwandlung eines String Array in einen ASCII-Vektor und dessen Übertragung an das Display

Der zu anzeigende String wird über den Port 2 in das Subsystem übergeben, zusammen mit der Adresse und dem Funktionsaufruf. Zuerst muss dieser, wie zuvor beschrieben, in einen ASCII-Vektor der Länge 20 konvertiert werden. Da jedes Zeichen einzeln übertragen werden muss, wird für jedes Element eine Übertragung ausgelöst. Um die Elemente auszuwählen wird ein Multi-Port Switch verwendet, dieser erhält von einem Zeichen Zähler den jeweiligen Index des zu übertragenden Symbol. Da der String nicht immer 20 Zeichen lang ist, füllt der ASCII Konverter diese mit Nullen auf, doch wenn diese Null übertragen wird, wird nach der Tabelle in Abbildung 16 das erste Zeichen des CGRAMs angezeigt. Das ist erstmal kein Problem, da diese hier nicht verwendet und beschrieben wurden. Doch werden diese nicht initialisiert, somit ist dort Speicherrauschen abgespeichert, das zufällig beim einschalten entsteht. Somit würden bis zum Ende jeder Zeile zufällige Pixelmatrizen angezeigt werden. Nun könnte man diese auch bei der Initialisierung mit Nullen beschreiben, doch ist es sinnvoller dies aus dem ASCII-Vektor

zu filtern. Was mit dem Switch Block erreicht wird, damit wird jede Null mit dem Wert 32 ersetzt, welcher einer Leerzeile einspricht. Diese Struktur muss nun für jedes Zeichen einer Zeile wiederholt werden, für dieses Display sind das 20 Mal. Durch den in Abbildung 18 dargestellten Aufbau wird das Subsystem I Fach aufgerufen. Durch das eingeben von 20 für die Iterationsanzahl im Function-Call Generator wird dieser Wert für I gesetzt.

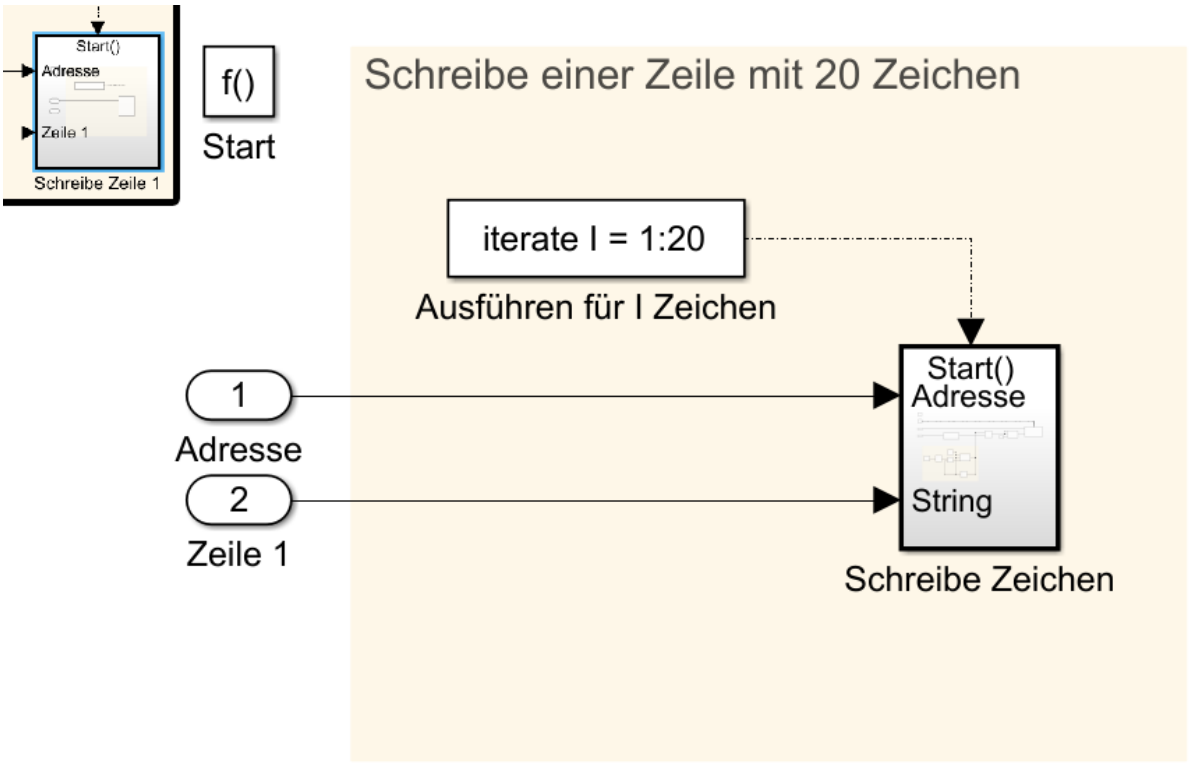

Abbildung 18: Ausführen der Zeichenroutine für jedes Element der Zeile

Diesem Subsystem kann nun eine arbiträre Zeile übergeben werden, diese wird dann sequenziell auf das Display übertragen.

3.3.3 Ablauf der Display Aktualisierung

Nun können einzelne Zeilen an das Display gesendet werden, jetzt müssen diese Einzelfunktionen zusammengeführt werden, um das gesamte Display aktualisieren zu können. Da es gewünscht ist das jede Zeile seinen eigenen Stringeingang besitzt, wurde hier das Schreiben der Zeilen mehrfach angewandt.(s. Abbildung 19)

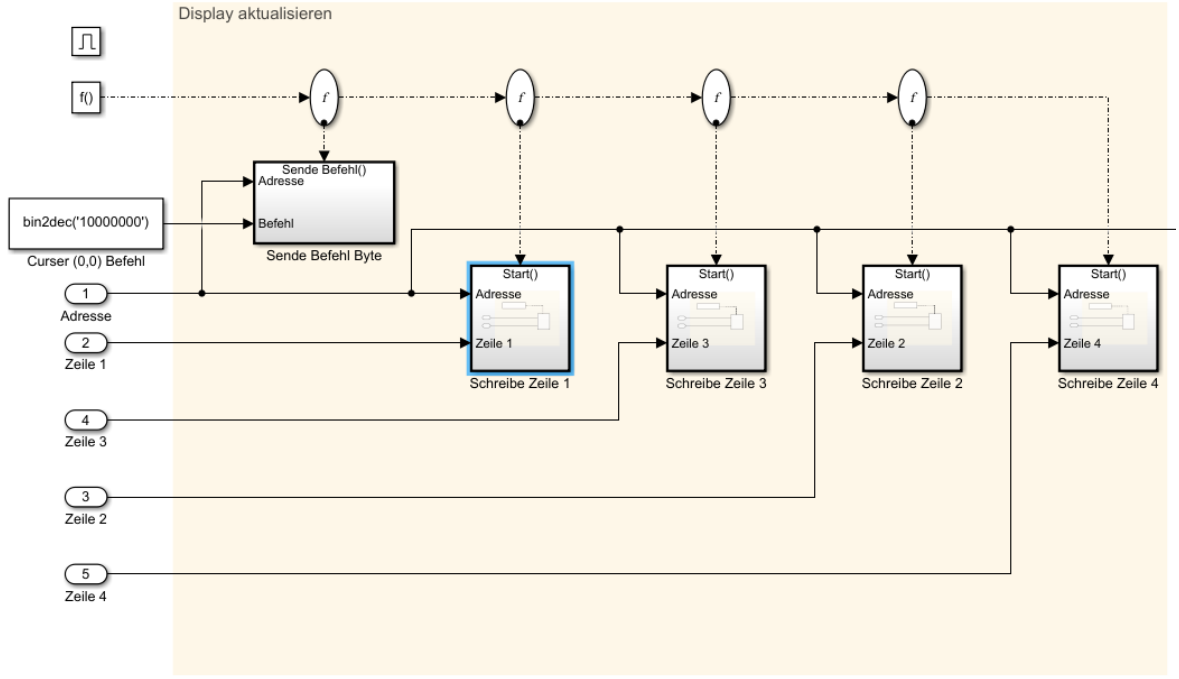

Abbildung 19: Struktur zur Aktualisierung aller Zeilen mit Cursor Initialisierung

An dieser Stelle wird wieder ein Enable Subsystem generiert, was an dem Enable Block in der linken oberen Ecke zu erkennen ist. Damit kann eine beliebige Aktualisierungsrate extern angeschlossen werden. Zuerst muss noch ein Befehl Byte an das Display gesendet werden, der Block zur Übertragung von Befehlen ist bis auf einen Parameter identisch zu der in Abbildung 15 dargestellten Struktur, dort wird anstelle von 0x9 0x8 verodert was dem HD44780 signalisiert, dass es sich um einen Befehl handelt. Dieser Befehl setzt den Cursor an die erste Stelle des Displays und das bei jedem Update. Somit landen alle Zeichen an der richtigen Position. Danach können sequenziell die Zeilen auf das Display geschrieben werden, dabei ist zu beachten, dass durch das Multiplexing des HD44780 die Zeilenreihenfolge im Speicher 1, 3, 2 und dann 4 ist. Der komplette Block zur Ansteuerung des I^2C Displays aus SIMULINK heraus ist in Abbildung 20 abgebildet.

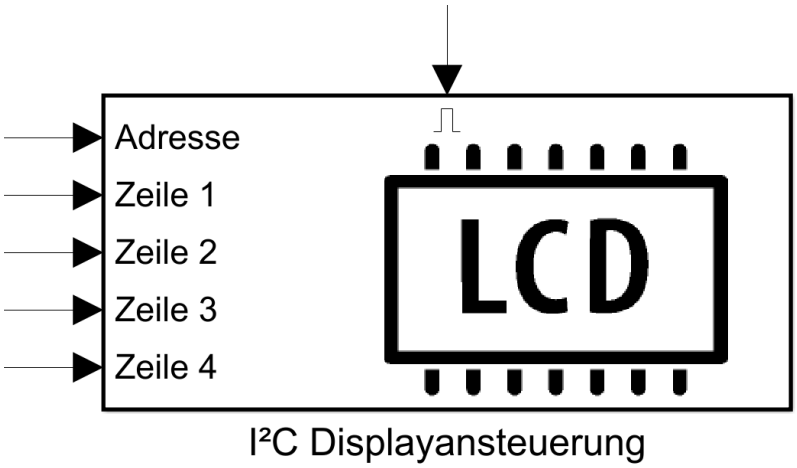

Abbildung 20: Fertige Displayansteuerungsblock in SIMULINK

Dieser Block lässt sich so sehr leicht in jedes Programm einfügen, um Daten auf einem Display auszugeben. In Abbildung 21 sind zwei Aktualisierungspakete abgebildet, bei einer Wiederholrate von 2Hz. Durch die Messung von ca. 100ms für die Dauer eines kompletten Schreibvorgangs, kann eine maximale Aktualisierungsrate von 10Hz erreicht werden. Eine höhere Frequenz könnte nur durch eine größere I²C Geschwindigkeit wie z. b. 400khz oder einer selektiven Aktualisierung erreicht werden. Doch ist eine Rate von 1-10Hz völlig ausreichend für die vorgesehenen Anwendungen.

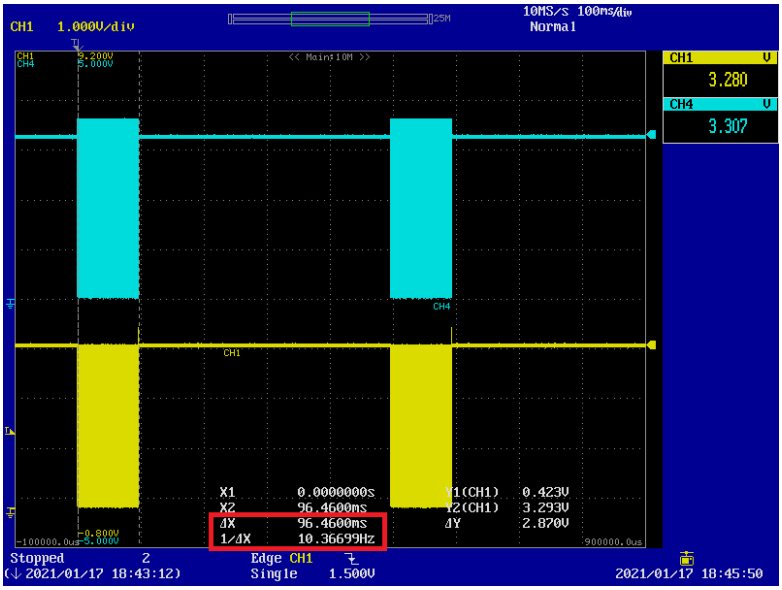

Abbildung 21: Aufzeichnung von SDA(blau) und SCL(gelb) während eines kompletten Aktualisierungszyklus

3.4 Demonstration der Implementierung

Es kann der erzeugte SIMULINK Subsystemblock direkt verwendet werden, um die Grundzüge seiner Funktionen anschaulich zu demonstrieren wird im Folgenden an einem Beispiel erklärt.

3.4.1 Hardwarekonfiguration in Simulink

Die einzig wichtige Einstellung, die in SIMULINK vorgenommen werden muss, ist das anpassen des Vorteilers(Prescaler) für den I²C Taktgenerator im F28069M. Die Einstellungen befindet sich unter "Hardware Settings"–>"Hardware Implementation" und dann im den "Target hardware resources" Menü. Im dem Bereich I²C müssen der "Master clock Low-time divider" und "Master clock Low-time divider" beide auf 40 gestellt werden, es wird auch automatisch die I²C Taktung ausgerechnet, in diesem Fall sind es dann die gewünschten 100kHz.(s. Abbildung 22)

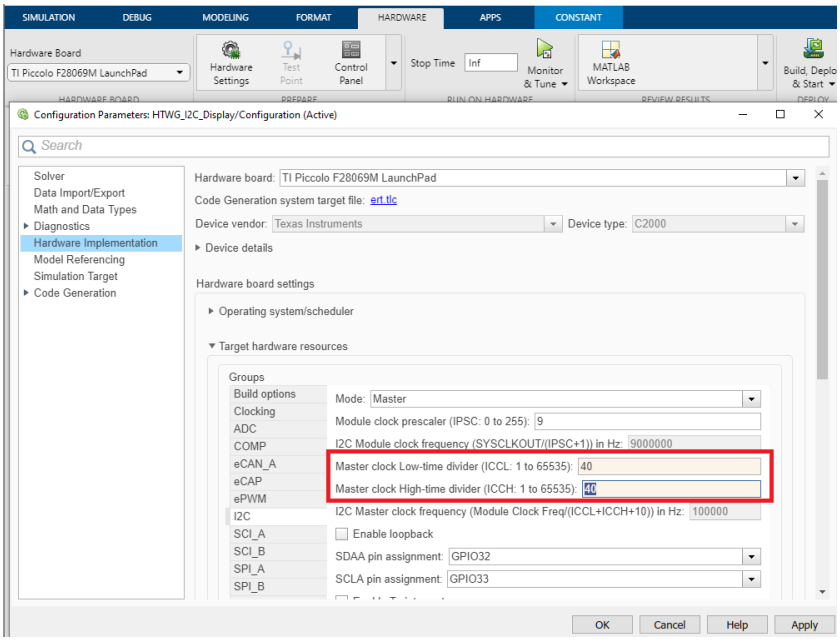

Abbildung 22: I²C Taktgenerator Einstellung in SIMULINK

3.4.2 Demoprogramm

Um die Grundfunktionen des SIMULINK Blocks zu demonstrieren wurde ein kleines Demoprogramm erstellt, das grundlegend zeigt, wie dieser verwendet werden muss. Das in Abbildung 23 dargestellte Programm kann direkt auf dem F28069M online oder offline ausgeführt werden.

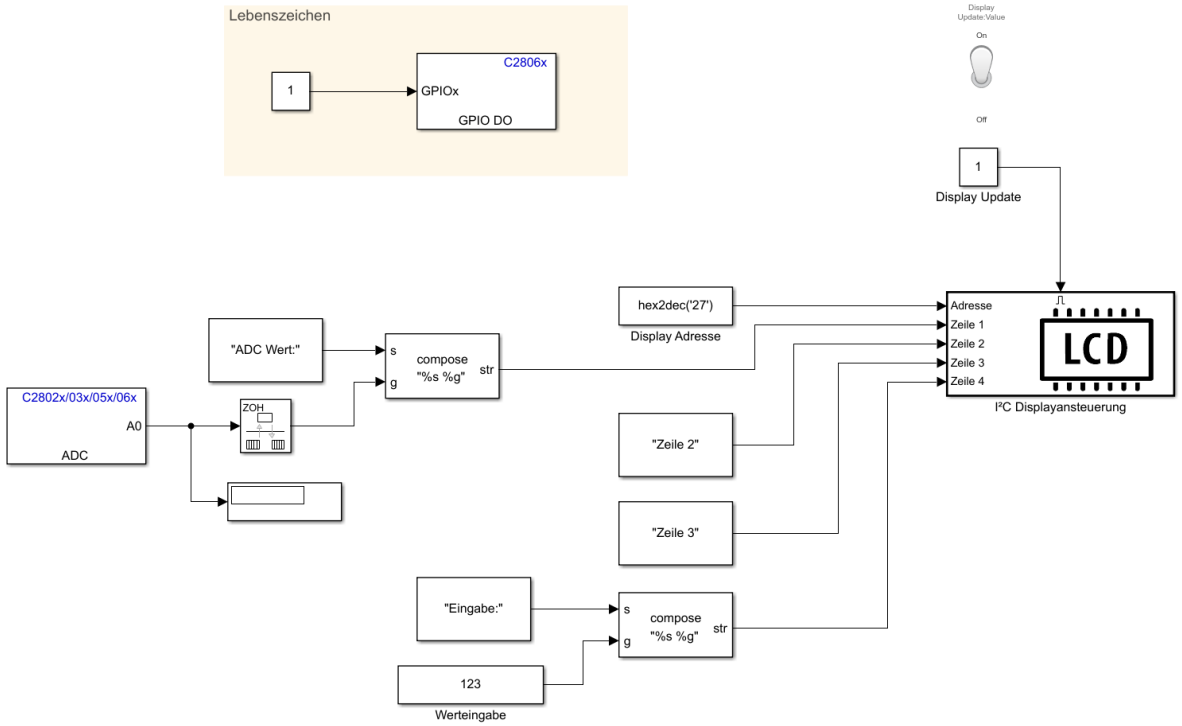

Abbildung 23: Übersicht über das Demoprogramm zur Displayansteuerung

Dem Block besitzt mehrere Eingänge, einmal für die vier Zeilen des Displays und dessen Adresse, hier ist die Adresse Standardmäßig 27. Den Zeileneingängen können einfach Strings zugewiesen werden, z. B. mit dem String Constant Block, so werden hier Zeile 2 und 3 befüllt. Mit dem Compose String können Strings aus verschiedenen Parametern zusammengestellt werden. Für die Zeile 1 wird der ADC des F28069M ausgelesen, dieser Wert sollte durch das Verwenden eines Rate Transition Blockes konvertiert werden, so ist die Aktualisierungsrate des Displays nicht an den ADC gebunden und umgekehrt. Als letztes wird noch eine Eingabe für Zeile 4 erzeugt. Falls das Programm im Monitor & Tune Modus verwendet wird, können live Werte in die Eingabe eingegeben werden, diese werden dann über USB zum F28069M übertragen und auf dem Display dargestellt. Über den Schalter lässt sich die Aktualisierung des Displays ein und ausschalten. Die Display

Aktualisierungsrate über den Display Update block frei einstellbar. Das “Lebenszeichen” steuert die blaue LED periodisch an, um zu signalisieren, dass das Programm ausgeführt wird. Das komplette Modell ist durchweg mit hilfreichen Beschreibungen erstellt worden, die auch das spätere modifizieren oder anpassen leicht möglich machen, auch kann so ein tieferes Verständnis der Implementierung erfolgen.

In Abbildung 24 ist der Hardwareaufbau des Testprogramms abgebildet. Standardmäßig wird die I^2C Schnittstelle an GPIO P32(SDA) und P33(SCl) verwendet, das kann aber auch wenn benötigt verändert werden.

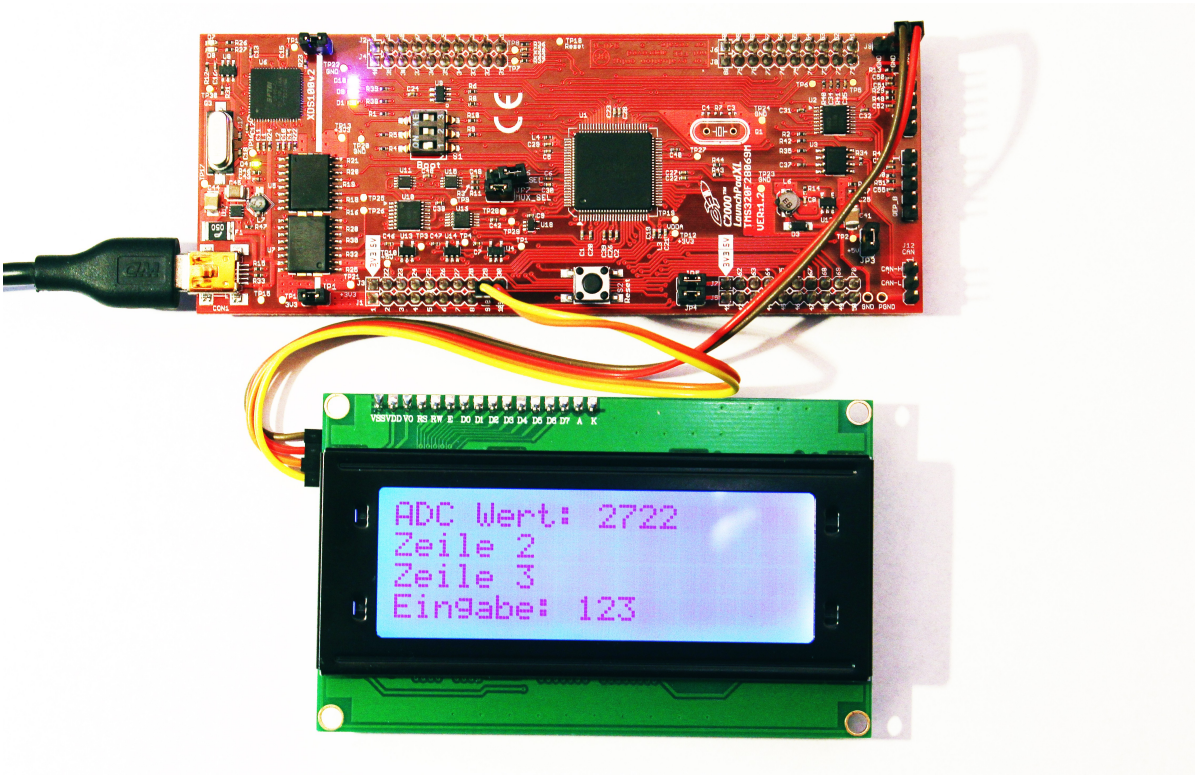

Abbildung 24: Versuchsaufbau zur Displayansteuerung

Es ist möglich, wie schon anfangs erwähnt, das eine vielzahl an I^2C Displays angeschlossen werden können. Hierfür müssen nur verschiedene Adressen gewählt und der Block in SIMULINK dementsprechend vervielfacht werden.

4 Fazit

Die Umsetzung einer in MATLAB SIMULINK integrierten Display Ansteuerung über I^2C mit dem TI Launchpad F28069M konnte erfolgreich erzielt werden. Durch die Verwendung von diskreten SIMULINK Blöcken lässt sich die Implementierung auch sehr schnell selbst nachvollziehen. Die im Rahmen dieser Arbeit entstandene Implementierung kann nicht nur fortlaufend weiter eingesetzt, sondern auch jederzeit erweitert werden. Eventuell ist es auch interessant, den Ansatz der Integration dieser Implementierung in eine Bibliothek zu migrieren, auch können noch weitere Optimierungen mit Zuhilfenahme der TLC Programmierung erreicht werden. Zusammenfassend lässt sich sagen, dass vieles trivial wirkt, nachdem es wie gewünscht umgesetzt ist. Nichtsdestotrotz gibt es Aspekte, die besonders bei der Implementierung einiges an Nachtforschungsarbeit und Versuchen benötigen, gerade bei den ersten Schritten kann es dauern, bis die ersten funktionieren Grundbausteine erarbeitet sind. Auch die langen Übersetzungszeiten und Codegenerierungen sorgen für ein abbremsen des Entwicklungsvorschritts.

Abschließend kann gesagt werden, dass mit der hier vorgestellten Methodik viele weitere Projekte entwickelt und umgesetzt werden können. Auch kann es als Grundlage dienen um verschiedenste weitere I^2C Peripherie zu integrieren.

Literatur

[HIT99] HITACHI. *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)*. 1999. URL: http://academy.cba.mit.edu/classes/output_devices/44780.pdf.

[Ins15] Texas Instruments. *I2C Bus Pullup Resistor Calculation*. 2015. URL: https://www.ti.com/lit/an/slva689/slva689.pdf.

[Ins19] Texas Instruments. *LAUNCHXL-F28069M Overview*. 2019. URL: https://www.ti.com/lit/ug/sprui11b/sprui11b.pdf.

[TheoJ] Inc. The MathWorks. *Using Enabled Subsystems*. o.J. URL: https://www.mathworks.com/help/simulink/ug/enabled-subsystems.html.

Danksagung

Möchte mich persönlich für die kompetente Betreuung bedanken, insbesondere für die klare Aufgabenstellung und der zu erwartende Ziele. Die Findung eines Themas, welches mir auch in der aktuellen Pandemie ermöglicht hat, dieses abseits der Hochschule bearbeiten zu können, rechne ich sehr hoch an.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbstständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht.
Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstantin
28.07.2021 Friedrichshafen

Ort, Datum

Anhang

Für die Erstellung der Arbeit wurden die folgenden Messgeräte verwendet.

YOKOGAWA DL750 ScopeCorder
Dabei handelt sich um einen Hybrid aus Oszilloskop und Datenschreiber. Je nach Erfassungskarte können Signale mit bis zu 10MS/s abgetastet werden. Auch durch den großen integrierten Speicher können lange Zeiträume detailliert aufgezeichnet werden.

AZDelivery Logic Analyzer
Dieser handlicher USB Logikanalysator besitzt 8 Eingänge und eine maximale Abtastrate von 25MHz. Er ist auch mit der Software von Saleae kompatibel, ermöglicht so das Analysieren von digitalen Signalen. Auch können verschiedenste Protokolle wie I^2C decodiert werden.