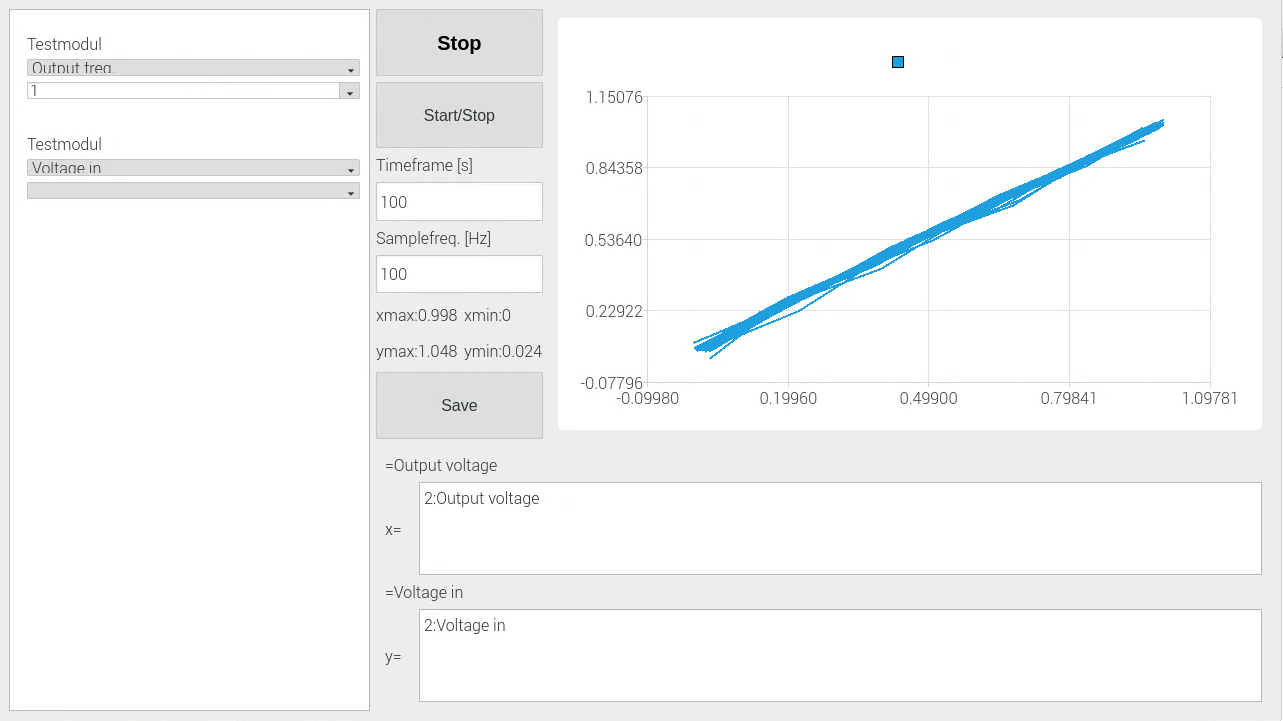
# Benutzeroberfläche

Um die Entwicklung und Bedienung so einfach wie möglich zu gestalten, finden sich alle Bedienelemente und der Graph auf einer Oberfläche, die bei Start des Systems gestartet wird.



Die einzelnen Elemente Skalieren mit unterschiedlichen Monitorauflösungen in gleichem Umfang, benötigen allerdings eine Mindestauflösung von 405 x 543 Pixel, wobei die Oberfläche dann nur noch sehr eingeschränkt nutzbar ist. Im linken Abschnitt werden die eingesteckten Karten angezeigt. Schwebt die Maus über einer Karte, wird der Slot angezeigt, den die Karte belegt. Die Zählweise beginnt bei 1 im Slot ganz links. Die RJ45 Buchsen erhalten die Nummern 6-10. Eine dargestellte Karte besteht aus dem Kartennamen, einer Auswahlbox, die die einzelnen Parameter der Karte beinhaltet und einer Werte-Auswahlbox. Wird ein Parameter ausgewählt, wird das Werte-Dropdown Menü angepasst. Je nachdem, ob der Parameter frei anpassbar ist, nur aus einer Liste ausgewählt werden kann, oder gar nicht geändert werden kann. Im obigen Fall besitzt das erste Testmodul einen Parameter “Output freq.“, der frei änderbar ist. Zusätzlich kann aber auch aus einer Liste ein vorgeschlagener Wert übernommen werden. Beim zweiten Testmodul ist der Parameter „Voltage in“ ausgewählt. Dieser Parameter kann nicht geändert werden, sondern dient lediglich als Datenlieferant. Die Karte übermittelt zu Beginn ihren Namen und alle Parameter an die Basisstation. Unter anderem auch die Schreib-Rechte und Wahlmöglichkeiten der einzelnen Parameter.

In der Mitte finden sich die Kontrollelemente. Der erste Button ist eine Kopie des physischen Buttons. Mit diesem kann eine Messung gestartet und gestoppt werden. Findet gerade eine Messung statt, wird „misst…“ angezeigt, ansonsten „Stop“.   
Mit dem zweiten Button wird der Messmodus gewechselt. Verfügbar sind „Start/Stop“, in dem eine feste Zeit lang gemessen wird, und „Cont.“, wohinter der Kontinuierliche Mess-Modus steckt. Wird der Modus während einer Messung gewechselt so werden alle bisherigen Werte verworfen und die Messung neu gestartet.  
Die Zeitdauer, die im Graph angezeigt wird, kann im Feld „Timeframe“ stufenlos eingestellt werden. Im Start/Stop-Modus entspricht diese der Messdauer. Auch die Sample-Frequenz kann stufenlos eingestellt werden. Hardwarelimitiert sind dabei Werte zwischen 0.1Hz bis 10kHz wählbar, wobei alles über 1kHz aufgrund fehlender Zuverlässigkeit vermieden werden sollte.   
Weil es zuweilen recht mühsam ist, Werte aus dem Graphen auszulesen, werden zumindest die aktuellen Minimal und Maximalwerte des sichtbaren Graphen für beide Achsen angezeigt.  
Mit dem „Save“-Button können alle erfassten Parameter des Messsystems in „.csv“-Dateien gespeichert werden. In diesem Prozess wird zunächst die aktuelle Messung gestoppt und dann alle Parameter aller Karten restlos ausgelesen. Abhängig von der Anzahl an Karten, Parameter und Messdauer kann das eine Weile in Anspruch nehmen. Im Anschluss wird der Benutzer aufgefordert einen Speicherort für die Dateien auszuwählen. Dies geschieht in einem Dateiexplorer, sodass auch angeschlossene Speichermedien gewählt werden können, wenn diese ein Linux-Kompatibles Dateisystem besitzen (z.B Fat32). In dem ausgewählten Verzeichnis wird eine Ordnerstruktur erstellet. Gekapselt unter einem Ordner mit dem aktuellen Datum und Uhrzeit wird für jede Karte ein eigener Ordner erstellt, der für die einzelnen Parameter eigene „.csv“-Dateien erstellt. So muss der Nutzer in der Auswertung nicht alle Parameter in Excel laden, sondern kann sich nur die genutzten auswählen. Gespeichert wird für jeden Parameter die Zeit in Sekunden und der Messwert.

Welche Messwerte, wie im Graphen dargestellt werden, wird mit den beiden Formeln dargestellt. Hier können für die x-Achse und y-Achse unabhängig mathematische Formeln angegeben werden. Im dargestellten Bespiel wurde der x-Achse der Spannungsausgang der Testkarte aus Slot 2 zugewiesen und der y-Achse der Spannungseingang derselben Karte. Im Testaufbau wurden lediglich die beiden Bananenbuchsen miteinander verbunden. Zuvor wurde der Spannungsausgang mithilfe der Parameter „Output min.“, „Output max.“ und „Output freq“ so konfiguriert, dass eine Dreieckspannung mit 1Hz zwischen 0V und 1V ausgegeben wird. Da viel Karten denselben Namen haben, in unserem Fall zweimal „Testkarte“, muss ein Parameter über seinen Kartenslot und Parameternamen angesprochen werden. „2:Voltage in“ referenziert also den Parameter „Voltage in“ der Karte in Slot 2. Sollte es diesen Parameter nicht geben, wird eine entsprechende Fehlermeldung ausgegeben. Das manuelle Eingeben der Parameter kann umgangen werden, indem die Karte mit ausgewähltem Parameter per Drag and Drop in den Formeleditor gezogen wird. Parameter können auch mit mathematischen Operatoren miteinander kombiniert werden. Zur Verfügung stehen + - \* / ( ) D S.  
„D“ bildet dabei die Ableitung einer nachstehenden Klammer und „S“ das Integral. Auch Gleitkommazahlen sind erlaubt. Soll auch die verstrichene Zeit in die Achse mit eingehen, so kann der Operator „T“ verwendet werden.

Die dadurch festgelegte Messreihe wird im Graphen dargestellt. Die Achsen skalieren dabei automatisch mit. Befindet sich die Messung im kontinuierlichen Modus, werden alle Messwerte aus dem festgelegten letzten Zeitraum angezeigt. Der Nutzer ist selbst verantwortlich, das System nicht mit der Menge der anzuzeigenden Messwerte zu überfordern. Ein guter Richtwert sind 1000 Messpunkte. Bei mehr als 10000 Messpunkte droht eine Überlastung der CPU und dem Freeze der Benutzeroberfläche. Im Start/Stop-Modus gibt es diese Limitierung nicht. Hier wird nur eine Auswahl der Messwerte angezeigt, sodass die maximale Anzahl in dieser Messung die 1000 nicht überschreitet. Gespeichert werden in beiden Modi aber alle Messwerte, ob sichtbar oder nicht.

# Aufbau der Raspberry Pi Software

## Threads

Die Software des Raspberry Pis ist darauf ausgelegt, den SPI Bus maximal auszulasten. Es soll vermieden werden, dass der Raspberry Pi keine Daten abfragen kann, und somit den Buffer der Steckkarten unnötig strapaziert, nur weil die Daten gerade ausgewertet werden. Es muss auch verhindert werden, dass das Lesen und Decodieren eines besonders langen Pakets Benutzereingaben verzögert. Die von uns gewählte Lösung besteht aus der Aufteilung in mehrere Threads. Das ermöglicht auch den Wechsel von einem Einkern-Systems, wie dem ursprünglich vorgesehenen Raspberry Pi Zero, auf ein Multicore System, wie den Raspberry Pi, der zusätzlich mit weitaus größerem Arbeitsspeicher ausgestattet ist, was die maximale Aufnahmedauer stark erweitert.

Die Threads laufen je eine Endlosschleife, wobei die Synchronisation durch verschieden Maßnahmen gewährleistet bleibt. Die Software besteht aus folgenden fünf Threads:

## Post-Thread:

Der Postthread ist für die Spi-Kommunikation mit den Steckkarten zuständig. Da diese dauerhaft ausgelastet sein soll, bleibt dieser auch immer aktiv. Seine Aufgabe besteht darin, Pakete aus einer queue zu lesen und entsprechend des Protokolls mit Daten zu füllen. Die eigentliche Low-Level Übertragung wird mittels der wiringPi – Bibliothek durchgeführt. Diese nutzt DMA Transfers, wodurch Rechenzeit für andere Tasks frei wird. Wurde ein Paket fertig gesendet und simultan empfangen, so wird es an den Decoder- Thread weitergeleitet, der dieses nun Zeitunkritisch bearbeiten kann. Um Race-Conditions ausschließen zu können, genügt es, lediglich die Queue und Chip-Select-GPIOs mittels Mutex-Semaphoren zu schützen. Um ein korrektes Abarbeiten der Pakete zu gewährleisten, reicht es, das Abfragen der Queue-Größe, sowie das Entnehmen eines Paketes zu schützen. Die Chip-Select-Leitungen werden zusätzlich vom Control-Thread zur Erkennung von neuen Karten benötigt. Diese müssen also auch mithilfe einer Mutex-Semaphore vor gleichzeitigem Zugriff geschützt werden. Die Suche nach neuen Karten wird lediglich selten und sehr schnell abgearbeitet und fällt daher nicht wirklich ins Gewicht. Manche Pakete, wie GET\_DATEN und GET\_PARAMETER besitzen keine feste Länge. Diese wird von der Steckkarte vorgegeben und wird in den ersten Bytes übermittelt. Für ein Paket sind in diesem Fall also zwei DMA Transfers notwendig. Der Thread ist in sich in einem Modul geschlossen, und bietet eine API zum Hinzufügen von neuen Paketen.

## Decoder-Thread

Nachdem das Paket empfangen wurde, können die Daten zeit-unkritisch ausgewertet werden. Weil die empfangenen Daten eine zu schützende Ressource darstellen, kann es im decode-Schritt zu Wartezeiten kommen, die die SPI-Kommunikation nicht aufhalten sollen. Aus diesem Grund benötigt auch der Decoder-Thread eine Warteschlange, die vom Postthread gefüllt wird. Hier kann es jedoch zu Zeiten kommen, in denen keine Pakete zum Auswerten bereit sind. In diesen Zeiten soll der Thread nicht ins busy-wait laufen, sondern Schlafen. Die Synchronisation wird hier durch eine condition-variable gelöst, die den Thread nur dann aufweckt, wenn er gerade schläft. Der Decoderer erkennt selbstständig die Art des Pakets und speist die erhaltenen Daten in den Rest des Systems ein.

## Control-Thread:

Gefüttert wird der Post-Thread durch den Control-Thread. Dieser ist in dem control-Modul beheimatet, das zentral den Zustand des Systems leitet. Die Aufgabe des Control-Threads ist es, neue Karten zu finden und nach Parameteren zu fragen, sowie die Daten zu besorgen, die gerade für den Graph benötigt werden. Die Bandbreite des SPI-Buses, limitiert durch das SPI-Modul des STM32, erlaubt es nicht, von allen Karten ständig alle Parameter-Daten abzufragen. Weil die Liste mit momentan gebrauchten Parametern jedoch vor der Grapherstellung geschützt werden muss, ist es sinnvoll, nur für das Befüllen der Post-queue diesen eigenen Thread zu verwenden.

## Graph-Thread:

Der Graphthread ist lediglich ein Relikt aus vergangenen Versuchen, der nun lediglich eine Funktion im Haupt-Thread aktviert, die neue Datenpunkte in den Graph lädt. Diese Funktion soll in einem weiteren Refaktoring-Schritt durch einen QTimer ersetzt werden.

## Haupt-Thread:

Im Hauptthread werden die Qt-Objekte verwaltet. Also die Gui-Elemente. Dazu zählen insbesondere die dynamische Anzeige der Karten, das Parsen der Achsen-Gleichungen und Anzeigen des Graphen. Weil ein Gui Element in Qt nur im Hauptthread erzeugt werden kann, die neue Karte jedoch im Decoder-Thread erzeugt wird, besteht eine Karte aus zwei Objekten. Das eine für den Zugriff auf die Karte mit Parameterliste und einem Gui-Teil. Durch Erzeugen der Karte im Decode-Thread wird ein Signal an den haupt-Thread gesandt, der dann den zugehörigen Gui-Teil erstellt.  
Als besonders Leistungshungrig hat sich das Erstellen des Graphen herausgestellt. Ein Graph in Qt besteht aus einem Graph, einem View-Objekt, das die Darstellung übernimmt und Serien, die die Datenpunkte besitzen. Zunächst wurde versucht, einzelne Datenpunkte aus der Serie zu löschen und neue Anzuhängen. Wie sich herausstellte, ist das eine sehr ineffiziente Möglichkeit. Besser ist es, die gesamte Serie mit neuen Datenpunkten zu befüllen (series->replace()). Dabei wird ein Zwischenspeicher benötigt, der gesamt in die Serie übernommen wird. Dazu wird zunächst festgestellt, zu welchem spät möglichsten Zeitpunkt alle Parameter über Daten verfügen. Befindet sich die Messbox im Kontinuierlichen Modus, so werden nun die nicht mehr benötigten Datenpunkte aus dem Zwischenspeicher gelöscht. Und dann neue Datenpunkte bis zum vorher bestimmten Zeitpunkt hinzugefügt. Nun können die Datenpunkte der Serie ausgetauscht werden. Was bleibt ist das Problem, dass die Achsen des Graphs, und damit auch der Anzeigebereich, eingestellt werden muss. Und dazu wird der minimale und maximale Wert des aktuell sichtbaren Bereichs benötigt. Ein stupides Durchgehen des Zwischenspeichers ist jedoch nur bei geringen Datenmengen durchführbar und wird daher auch nur im Kontinuierlichen Modus angewandt. Im Start-Stop-Modus wird beim Hinzufügen des Datenpunkts der minimale und maximale Wert aktualisiert. Weil bei langen Zeitdauern, wie sie spezifiziert sind, die Performance massiv beeinflusst würde, wenn mit jedem Zeitpunkt alle Datenpunkte neu gezeichnet werden würden, werden die angezeigten Datenpunkte in diesem Modus auf 1000 begrenzt. Es werden natürlich auch die dazwischen liegenden, nicht angezeigten Punkte mit abgespeichert.  
Festgelegt, welche Parameter auf welcher Achse angezeigt werden, wird durch Auswertung der Achsen-Formeln. Die sich daraus ergebende Struktur wird aktualisiert im Fall einer neuen Eingabe, oder auch bei Einfügen und Ausstecken einer Karte. Die Formeln werden zunächst in die einzelnen Komponenten geparst, um Parameter von Zahlen und Rechenoperationen unterscheiden zu können. In einem weiteren Schritt werden nun die Elemente in die richtige Topologie gebracht. Gelöst wird dies rekursiv durch Prioritäten der einzelnen Elemente. Um den neuesten Zeitpunkt zu erhalten, oder einen Wert für einen Zeitpunkt zu bekommen, muss lediglich die entsprechende Funktion in der Wurzel aufgerufen werden, die dann rekursiv weiterläuft und die Rechenoperationen anwendet. Obwohl die Daten nur in single precision float Qualität erhoben werden, erfolgt die Berechnung und Darstellung im double Zahlenbereich. Das sichert auch bei großen Zahlen die erforderliche Genauigkeit in den Berechnungen.

# Programmablauf

Nun soll beschrieben werden, wie das Programm funktioniert. Das Programm wird ständig versuchen, den Graph zu erstellen. Weil die Datenpunkte des Graphen ja nicht beständig neu gesucht werden, führen bestimmte Aktionen, wie das Eingeben eines neuen Zeitfenster oder Achsenformel zu einem Löschen des Graphen, wodurch dieser einmalig wieder komplett neu erstellt wird. Es ist ersichtlich, dass das Erstellen des Graphen völlig von den anderen Aktivitäten unabhängig ist. Das hat den Nachteil, dass die Daten und der Datenbehälter vor gleichzeitigem Zugriff geschützt werden müssen. Insbesondere muss das Programm davor geschützt werden, Daten von einem Parameter zu erfragen, den es schon gar nicht mehr gibt. Das kann vorkommen, wenn während des Erstellens des Graphen eine Karte ausgesteckt wird, oder sich die Achsenbelegung ändert. Gelöst wurde das Problem durch eine große Mutex-Semaphore, die sowohl die Graph Erstellung, als auch den Achsen-Formel-Parser und das Löschen einer Karte klammert. Da zwei der drei verwendeten Stellen nur sehr selten vom Benutzer ausgeführt werden, ist die Performanceeinbuße hinnehmbar.

Damit der Control-Thread weiß, welche Daten gerade benötigt werden, hält der Parser eine Liste an Parameter-Pointern bereit. Der Control-Thread wird nun beständig versuchen, die queue des Post-Thread mit 10 Paketen gefüllt zu halten. Dadurch können jederzeit Status, Parameter und Befehle gesendet werden und der Rest der SPI-Kapazität wird im Round Robin Verfahren mit Datenanfragen gefüllt. Auch das Abfragen nach Daten ist unabhängig vom Zustand des Systems. Das ermöglicht es, Auch nach abgeschlossener Messung, den Graph mit anderen Parametern zu belegen. Diese werden dann so schnell wie möglich aus dem Zwischenspeicher der Steckkarten ausgelesen. Nachdem der Decoder die Daten ausgelesen hat, stehen sie im System zur Verfügung, bis eine neue Messung gestartet wird. Dann werden die Daten sämtlicher Parameter gelöscht und die Steckkarten setzen ihren Zwischenspeicher zurück. Intern liegen die Datenpunkte in einem vector gespeichert in den Parametern. Das hat den Vorteil zu einer verketteten Liste, dass die Daten auch mit einem Index angesprochen werden können. So muss nicht jedes Mal die gesamte Liste bis zum gesuchten Zeitpunkt abgesucht werden. Stattdessen merkt sich der Parameter den zuletzt verwendeten Index. Es ist nun sehr wahrscheinlich, dass bei erneutem Aufruf derselbe(doppelte Verwendung des Parameters) oder nächste Datenpunkt gesucht wird. Auch wenn Datenpunkte in der Anzeige übersprungen werden, wie es im Start-Stop-Modus der Fall ist, lässt sich so die Graph Erstellung bedeutend beschleunigen.

# Protokoll

Um den Datenaustausch maximal effizient zu gestalten, setzen wir auf unser eigenes Daten und Kommando Protokoll. Die Verwendung einer Einheitlichen JSON – Form wurde zwar in Betracht gezogen, letztendlich wollten wir allerdings die Möglichkeit sowohl float-Daten binär, als auch ascii-Parameter, ohne viel overhead versenden zu können. Die übertragung erfolgt in Little-Endian, da sowohl STM32, als auch Raspberry Pi auf Little-Endian Arm Architekturen bestehen. Das erleichtert die Umwandlung von empfangenen Bytes in float-Variablen ungemein. Es sind bisher 8 Befehle definiert, wobei das Protokoll Platz für 256 bieten würde.

## GET\_PARAMETER: 0x01 : An einzelne Karte

Erkennt der Control-Thread eine neue Karte, sendet er ein GET\_PARAMETER Paket an den entsprechenden Slot. Mit diesem Paket wird der Name der Karte erfragt und welche Parameter die Karte bietet. Ein Parameter besteht dabei aus einer Nummer, die künftig zur Identifikation im SET\_PARAMETER und GET\_DATEN Paket dient, einem Parameternamen, mit dem der Benutzer interagiert, minimal und maximal Werten und einer optionalen Werteauswahl. Die einzelnen Angaben sind immer nur durch ein Zeichen getrennt. Die Paket-Länge bezieht sich auf die Byte-Zahl des Parameter-Teils

|  |  |  |  |
| --- | --- | --- | --- |
|  | Befehl | Paket-Länge | Parameter |
| MOSI | 0x01 |  |  |
| MISO |  | 0-20000 | Kartenname,Parameternummer,Name,Typ,Parametrierbar,Min,Max{Wahl1,Wahl2,…}Parameternummer,… |

uint8\_t uint32\_t ascii

|  |  |
| --- | --- |
| Befehl | 0x01 für GET\_PARAMETER |
| Paket-Länge | Byte-Zahl der Parameter |
| Kartenname | Wird dem Benutzer in der Kartenübersicht angezeigt. Intern verwendet wird jedoch ausschließlich der Index des Slots, in den die Karte gesteckt ist. Es können also auch zwei gleichnamige Karten eingesteckt sein. |
| Parameternummer | Nummer unter der der Parameter in weiteren Befehlen erreichbar ist |
| Parametername | Der Name des Parameters wird zusammen mit dem Kartenindex verwendet, um Parameter auf Achsen zuzuweisen |
| Typ | {s / f} : string oder float. Entscheidet, ob Daten vom Parameter abgefragt werden können und im Graphen dargestellt werden können. String Parameter können zum konfigurieren des Systems genutzt werden. |
| Parametrierbar | {l /f / n} : Liste, Frei, oder Nicht. Entscheidet, ob und wie der Benutzer den Parameter ändern kann. Liste: Der Benutzer kann keine freien Eingaben machen, aber aus den vorgegebenen Wahlmöglichkeiten wählen. Frei: Der Benutzer kann freie Eingaben machen, die zwischen min und max liegen müssen, sind Wahlmöglichkeiten vorhanden, so werden diese auch als drop down-Menü angezeigt. Nicht: Der Benutzer kann den Parameter nicht verändern. Er dient lediglich als Datenlieferant |
| Min | Der Minimalwert, der eingestellt werden kann. Muss immer angegeben werden; Wird als ASCII übertragen |
| Max | Der Maximalwert, der eingestellt werden kann. Muss immer angegeben werden.; Wird als ASCII übertragen |
| Wahl | Optional: Die Wahlmöglichkeiten, die dem Benutzer angezeigt werden. Werden im geschweiften Klammern angegeben. Falls keine Wahlmöglichkeiten vorhanden sind, folgt der nächste Parameter ohne Klammern mit einem Komma. Alle Wahlmöglichleiten werden als ASCII übertragen. |

Wenn die Karte noch nicht bereit ist und nicht antwortet, so wird das Paket im Decoder als Fehlerhaft erkannt und keine neue Karte erstellt. Der Control-Thread wird im nächsten Versuch die Karte wieder als neu erkennen und einen neuen Versuch starten.

## SET\_PARAMETER: 0x02 : An einzelne Karte

Ändert einen Parameter in der Karte. Der Parameter wird durch seine Nummer angesprochen. Es werden lediglich ASCII-Werte übertragen. Diese müssen eventuell in der Karte umgewandelt werden. Die Paket-Länge beinhaltet die 4 Byte der Parameternummer und die Länge des ASCII Parameterwert.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Befehl | Paket-Länge | Parameternummer | Parameterwert |
| MOSI | 0x02 | 4 - 2^32 | 0 – 2^32 |  |
| MISO |  |  |  | Parameterwert |

uint8\_t uint32\_t ascii

|  |  |
| --- | --- |
| Befehl | 0x02 für SET\_PARAMETER |
| Paket-Länge | Die Paket-Länge setzt sich zusammen aus 4 + Bytezahl des Parameterwerts |
| Parameternummer | Eindeutige Zuordnung des Parameters innerhalb der Karte |
| Parameterwert | Der Wert, auf den der Parameter gesetzt werden soll. Ist immer als ASCII codiert und wird bei Bedarf in der Karte umgewandelt. (atof()) |

## GET\_DATEN: 0x03 : An einzelne Karte

Fordert Daten von einem Parameter an. Um den Bus nicht zu verstopfen und fehlerhafte Pakete ansatzweise zu erkennen, ist die Paketgröße auf 5000 Bytes softwareseitig begrenzt. Es ist ratsam, nur einzelne Pages des Zwischenspeichers auf einmal zu übertragen, um genug Zeit zu haben, neue Daten in das RAM zu laden. Ein Paket besteht dabei aus einer Anfangszeit und fortlaufenden Werten. Die Zeit bezieht sich dabei auf die Samples und nicht die eigentlich vergangene Zeit. Die Umwandlung erfolgt erst bei der Grapherstellung.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Befehl | Parameternummer | Paket-Länge | Startzeit | Daten |
| MOSI | 0x03 | 0 - 2^32 |  |  |  |
| MISO |  |  | 0-5000 | 0-2^32 | Datum1,Datum2,… |

uint8\_t uint32\_t float

|  |  |
| --- | --- |
| Befehl | 0x03 für GET\_DATEN |
| Parameternummer | Damit die Karte angeben kann, wie viele Daten bereit sind, braucht sie den Parameter, um den es geht. |
| Paket-Länge | Die Paket-Länge entspricht nur der Länge des Daten-Teils. Also Anzahl an Datenpunkten \* 4 |
| Startzeit | Die Sample-Zeit des ersten Datums. Folgende Daten erhalten inkrementell entsprechende Sample-Zeiten |
| Daten | Die übertragenen Werte als float in Little Endian |

## START\_KONT: 0x04 : An alle Karten und ATMEGA88

Alle Daten löschen alle Zwischenspeicher und der ATMEGA88 beginnt den Sample-Takt. Nicht abgefragte Daten werden bei vollem Zwischenspeicher überschrieben.

|  |  |
| --- | --- |
|  | Befehl |
| MOSI | 0x04 |
| MISO |  |

## START\_STARTSTOP: 0x05 : An alle Karten und ATMEGA88

Alle Daten löschen alle Zwischenspeicher und der ATMEGA88 beginnt den Sample-Takt. Nicht abgefragte Daten werden bei vollem Zwischenspeicher nicht überschrieben, sondern neue Daten verworfen.

|  |  |
| --- | --- |
|  | Befehl |
| MOSI | 0x05 |
| MISO |  |

## STOP: 0x06 : An alle Karten und ATMEGA88

Der ATMEGA88 beendet den Sample-Takt.

|  |  |
| --- | --- |
|  | Befehl |
| MOSI | 0x06 |
| MISO |  |

## SET\_SAMPLE\_FREQ : 0x07 : An alle Karten und ATMEGA88

Setzt die Sample Frequenz des ATMEGA88 auf einen neuen Wert. Wird aber erst bei START\_KONT, oder START\_STARTSTOP gestartet.

|  |  |  |
| --- | --- | --- |
|  | Befehl | Frequenz |
| MOSI | 0x07 | 0.1-10000 |
| MISO |  |  |

uint8\_t float

|  |  |
| --- | --- |
| Befehl | 0x02 für SET\_PARAMETER |
| Frequenz | Die Frequenz, die der ATMEGA88 als Sample-Frequenz erzeugen soll |

## GET\_STATUS: 0x08 : Nur an ATMEGA88

Frägt die Betriebsspannungen der Basis ab. Der Raspberry Pi besitzt keinen ADC, also wird hierfür der ATMEGA88 genutzt.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Befehl | 5V | 3,3V | Ladestrom | Vbat | Vint | Vin |
| MOSI | 0x08 |  |  |  |  |  |  |
| MISO |  | 5.0 | 3.30 | 0.0 – 0.5 | 4.20 | 5.0 | 6.0 – 25.0 |

uint8\_t float

|  |  |
| --- | --- |
| Befehl | 0x08 für GET\_STATUS |
| 5V | Die 5V Spannungsversorgung, die entgültig den Raspberry Pi und die Steckkarten speist. |
| 3,3V | Der 3,3V Ausgang des Raspberry für die CPU des Raspberry Pi, den ATMEGA88 und die Pull-Ups der Chip Select Leitungen |
| Ladestrom | Der momentane Ladestrom des Lithium-Ionen Akkus |
| Vbat | Die aktuelle Spannung des Akkus |
| Vint | Die Basisplatine wandelt die Eingangsspannung in eine Zwischenspannung von 5V um, mit der der Akku geladen wird. |
| Vin | Der Eingangs Buck-Converter erlaubt eine Eingangsspannung zwischen 6 und 25V |

# Interessante Programmabschnitte

## Kartenerkennung

Um GPIO-Pins zu sparen, erfolgt die Erkennung neuer Karten durch die Chip-Select Leitungen. Der Raspberry aktiviert hierzu die 50k Pull-Downs auf den Chip-Select Leitungen. Auf den Karten hingegen befinden sich auf diesen Leitungen 10k Pull-Ups. Wenn der Raspberry nun auf neue Karten prüft, schaltet er die Chip-Select-Leitungen auf INPUT. Wenn keine Karte gesteckt ist, wird der Low-Pegel des internen Pull-Downs gelesen, wenn die Karte jedoch gesteckt ist, kann der Pull-Up der Streckkarte den Pegel auf 2,75V ziehen, was im High-bereich des Raspberry Pi liegt. Der SPI-Slave der Steckkarte sieht dabei die gesamte Zeit ein inaktives HIGH-Signal. Soll mit der Karte kommuniziert werden, muss lediglich der GPIO wieder auf OUTPUT und LOW geschaltet werden.

## Formel-Parser

Um die einzelnen Rechenelemente aus der Achsenformel zu separieren, wurde ein Mealy Zustandsautomat verwendet. Die folgenden Zustände können eingenommen werden: START, KLAMMERAUF,KLAMMERZU,TIME,INTEGRATE,DERIVATE,ZAHL,PUNKT,PARAMETER,MINUS. Wenn ein unerlaubter Übergang auftritt, wird das Parsen abgebrochen und der Benutzer erhält eine rudimentäre Fehlernachricht.

### START:

Der Startzustand wird nur am Anfang angesprungen und erlaubt nur rechtsseitige Operatoren

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| ( | Klammer auf | KLAMMERAUF |
| T/t | Zeit | TIME |
| S/s | Integral | INTEGRATE |
| D/d | Ableitung | DERIVATE |
| 0-9 |  | ZAHL |
| - |  | MINUS |
| . |  | PUNKT |

### MINUS:

Der MINUS Zustand kennzeichnet eine negierte Zahl/Parameter. Kein Minus Operand, der Zwei Zahlen subtrahiert. Bzw. Die erste Zahl ist in diesem Zustand immer 0

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| ( | Konstante 0, Minus, Klammer auf | KLAMMERAUF |
| T/t | Konstante 0, Minus, Zeit | TIME |
| S/s | Konstante 0, Minus, Integral | INTEGRATE |
| D/d | Konstante 0, Minus, Ableitung | DERIVATE |
| 0-9 |  | ZAHL |
| . |  | PUNKT |

### KLAMMERAUF:

In diesem Zustand befindet man ich nach einer geöffneten Klammer. Verhält sich wie START

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| ( | Klammer auf | KLAMMERAUF |
| T/t | Zeit | TIME |
| S/s | Integral | INTEGRATE |
| D/d | Ableitung | DERIVATE |
| 0-9 |  | ZAHL |
| - |  | MINUS |

### KLAMMERZU:

In diesem Zustand befindet man ich nach einer geschlossenen Klammer. Bei manchen Übergängen kann sich ein „Mal“ gespart werden

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| + | Plus | START |
| - | Minus | START |
| \* | Mal | START |
| / | Geteilt | START |
| ( | Mal, Klammer auf | KLAMMERAUF |
| ) | Klammer zu | KLAMMERZU |
| T/t | Mal, Zeit | TIME |
| S/s | Mal, Integral | INTEGRATE |
| D/d | Mal, Ableitung | DERIVATE |
| 0-9 | Mal | ZAHL |
| . | Mal | PUNKT |
| /0 |  | Fertig |

### TIME:

Nach einem t/T kann man sich ein „Mal“ sparen, da es wie eine Variable gelesen wird

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| + | Plus | START |
| - | Minus | START |
| \* | Mal | START |
| / | Geteilt | START |
| ( | Mal, Klammer auf | KLAMMERAUF |
| ) | Klammer zu | KLAMMERZU |
| T/t | Mal, Zeit | TIME |
| S/s | Mal, Integral | INTEGRATE |
| D/d | Mal, Ableitung | DERIVATE |
| 0-9 | Mal | ZAHL |
| . | Mal | PUNKT |
| /0 |  | Fertig |

### INTEGRATE:

Für INTEGRATE wird ein weiterer Zustand benötigt, da die Topologie keine zwei Integrierer hintereinander schalten kann. Diese müssen durch Klammern getrennt werden.

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| ( | Klammer auf | KLAMMERAUF |
| T/t | Zeit | TIME |
| . |  | PUNKT |
| 0-9 |  | ZAHL |
| - |  | MINUS |

### DERIVATE:

Für DERIVATE wird ein weiterer Zustand benötigt, da die Topologie keine zwei Ableitungen hintereinander schalten kann. Diese müssen durch Klammern getrennt werden.

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| ( | Klammer auf | KLAMMERAUF |
| T/t | Zeit | TIME |
| . |  | PUNKT |
| 0-9 |  | ZAHL |
| - |  | MINUS |

### ZAHL:

In diesem Zustand befindet sich der Parser mitten in einer ganzen Zahl. Das kann sich zu einer Gleitkommazahl, zu einem Parameter entwickeln, oder eine ganze Konstante bleiben

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| + | Konstante, Plus | START |
| - | Konstant, Minus | START |
| \* | Konstante, Mal | START |
| / | Konstante, Geteilt | START |
| ( | Konstante, Mal, Klammer auf | KLAMMERAUF |
| ) | Konstante, Klammer zu | KLAMMERZU |
| T/t | Konstante, Mal, Zeit | TIME |
| S/s | Konstante, Mal, Integral | INTEGRATE |
| D/d | Konstante, Mal, Ableitung | DERIVATE |
| 0-9 |  | ZAHL |
| . |  | PUNKT |
| : |  | PARAMETER |
| /0 | Konstante | Fertig |

### PUNKT:

Nach einem „.“ kann nur noch eine Gleitkommazahl gemeint sein. Ein zweiter „.“ wäre nicht sinnvoll.

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| + | Konstante, Plus | START |
| - | Konstante, Minus | START |
| \* | Konstante, Mal | START |
| / | Konstante, Geteilt | START |
| ( | Konstante, Mal, Klammer auf | KLAMMERAUF |
| ) | Konstante, Klammer zu | KLAMMERZU |
| T/t | Konstante, Mal, Zeit | TIME |
| S/s | Konstante, Mal, Integral | INTEGRATE |
| D/d | Konstante, Mal, Ableitung | DERIVATE |
| 0-9 |  | PUNKT |
| /0 |  | Fertig |

### Parameter:

Nach einem t/T kann man sich ein „Mal“ sparen, da es wie eine Variable gelesen wird

|  |  |  |
| --- | --- | --- |
| Zeichen | Neue erkanntes Element | Nächster Zustand |
| + | Parameter, Plus | START |
| - | Parameter, Minus | START |
| \* | Parameter, Mal | START |
| / | Parameter, Geteilt | START |
| ( | Parameter, Mal, Klammer auf | KLAMMERAUF |
| ) | Parameter, Klammer zu | KLAMMERZU |
| /0 |  | Fertig |
| else |  | PARAMETER |

Jetzt, nachdem die einzelnen Elemente der Formel getrennt wurden, müssen die Blöcke noch in die richtige Topologie gebracht werden. Zu diesem Zweck verfügt die Oberklasse „Rechenblock“ die rekursiv arbeitenden Funktionen „von\_oben()“ und „von\_unten()“. Eine Formel ist aufgebaut aus in sich geschachtelten Berechnungen. Es ist daher naheliegend, die Topologie der Formel als Baumstruktur nachzubilden. Und je nachdem, ob eine Berechnung gerade verlassen wurde, oder noch offen ist, muss der Baum verschieden erweitert werden. Gestartet wird der Baum zu Beginn an der Wurzel und ein Zeiger auf das zuletzt hinzugefügte Element wird zwischengespeichert. Bei diesem Rechenblock muss die rekursive Funktion fortgesetzt werden. Ob ein Rechenblock eine Berechnungsebene abschließt, oder eine neue startet, wird durch die Priorität der Blöcke festgelegt. Gestartet wird das Hinzufügen eines Rechenblocks durch aufrufen von „von\_unten()“ des zuletzt hinzugefügten Elements. Zunächst wird geprüft, ob es sich bei dem hinzuzufügenden Element um eine schließende Klammer handelt, die gerade eine geöffnete Klammer schließt. Eine schließende Klammer hat die Eigenschaft, kein echter Rechenblock zu sein, sondern lediglich die Priorität der öffnenden klammer von 1 auf 5 (max.) zu erhöhen. Somit ist dieser Bereich für alle weiteren Blöcke nicht mehr erreichbar. Ist dem nicht der Fall, entscheiden die Prioritäten, wie es weiter geht. Ist die Priorität des neuen Block größer als die des aktuellen, oder es handelt sich um eine Klammer, wird der neue Block ein Eingang des aktuellen. Es kann nicht passieren, dass der Block von unten an einen anderen Block weitergegeben werden muss, weil ein einmal verlassener Bereich nicht wieder von unten aufgemacht werden kann. Wenn hingegen die Priorität gleich oder niedriger der des aktuellen Rechenblocks ist, wird die Entscheidung an die Ebene darunter über „von\_oben()“ weitergegeben.

Dabei gibt die Funktion „von\_oben()“ den Block so weit nach unten, bis die aktuelle Ebene eine niedrigere Priorität besitzt als die des hinzuzufügenden Rechenblocks. Dann wird der Block ein neuer Eingang dieser Ebene. Auch hier muss jedoch zunächst festgestellt werden, ob eine Ebene durch eine Klammer abgeschlossen wird und entsprechend die Priorität der Klammer erhöht werden. Es ist ersichtlich, dass Klammern durch das ändern Ihrer Priorität die Möglichkeit haben, Zweige mit niedrigen Prioritäten oberhalb von höherwertigen Zweigen anzusiedeln.

Die Prioritäten sind folgendermaßen gestaffelt:

0 : Achse

1 : offene Klammer

2: Plus, Minus

3: Mal, Geteilt

4: Integrate, Derivate

5: Zeit, Konstante, Parameter, geschlossene Klammer

-1: schließende Klammer (Wird nicht eingebaut, sondern macht offene Klammer zu geschlossener)

Um neue Funktionalität hinzuzufügen, muss der Zustandsautomat um die entsprechenden Übergänge ergänzt werden und ein neuer von „Rechenblock“ abgeleiteter Block definiert werden, der die Funktionalität in „get\_data()“ implementiert und mit der richtigen Priorität ausgestattet ist. Möglicherweise müssen die Prioritäten dann neu geordnet werden.

# STM32 Software

Die Software des STM32 ist ausschließlich Interrupt basiert. Nach Initialisierung der Module, wird lediglich einmalig der SPI-Flash gelöscht und dann in der leeren main Schleife auf Interrupts gewartet. Die Software ist modular aufgebaut, sodass die Module einzeln entwickelt werden konnten und lediglich mit den richtigen Interrupt Prioritäten zusammenspielen müssen. Die Software besteht aus einem COM-Handler, Parametermodul, einem ADC Modul, einem DAC-Modul, einem Flash-HAL, einem GPIO-HAL und dem Hauptmodul, das noch einzelne Aufgaben erledigt, die zu klein für ein einzelnes Modul wären. Für viele der Aufgaben gäbe es auch von ST bereitgestellte HAL-Bibliotheken, jedoch sind diese enorm ineffizient und werden von uns dementsprechend ignoriert.

## GPIO-Modul

Um nicht jedes Mal, das Datenblatt des STM32 studieren zu müssen, wenn ein neuer Pin initialisiert werden soll, bietet unser Gpio-Modul eine init\_gpio-Funktion, die alle wichtigen Parameter des Pins einstellen. So kann der Pin als Eingang, Ausgang, Analogpin, oder Alternate-Function (direkt mit Peripherie verbunden) initialisiert werden. Darüberhinaus können auch Pull-Up/Down, Slew-Rate und open Kollektor / Push-Pull Betrieb mit einem Funktionsaufruf eingestellt werden. Weiterhin werden Funktionen zum Lesen, Schreiben und Toggeln eines Pins bereitgestellt. Die meisten Pins sind jedoch direkt mit der Peripherie des STM32 verbunden.

## COM-Handler

Das COM-Handler-Modul besteht aus einer einzigen großen State-Maschine, die durch den SPI-Interrupt des mit dem Raspberry Verbundenen Peripheriemoduls ausgelöst wird. Um zuverlässig den Anfang und das Ende eines Pakets zu erkennen, wird das Chip-Select-Signal nicht nur zur Aktivierung des SPI-Moduls genutzt, sondern auch zum reseten der State-Maschine. Aus diesem Grund ist es notwendig, den Chip-Select Eingang mit einem Pin-Change-Interrupt zu belegen. In diesem Interrupt wird dann auch das SPI-Modul mittels Software Slave-Select aktiviert und deaktiviert.  
Bei der Initialisierung dieses Moduls wird der Inhalt des GET\_PARAMETER-Datensatzes aus der zuvor im Parameter-Modul gebildeten parameterliste gebildet. Das erlaubt das einfache Hinzufügen neuer Parameter, ohne an dieser Stelle etwas ändern zu können. Werden größere Datenmengen an den Raspberry Pi übertragen, wie etwa bei GET\_PARAMETER oder GET\_DATEN Paketen, so geschieht dies mittels DMA Transfers. So wird die Rechenlast auf die CPU minimal gehalten.

## Flash-Modul

Das Flash-Modul stellt Funktionen bereit, um mit dem 32/64/128Mbit SPI-Flash daten auszutauschen. Das Protokoll des Flash-ICs erlaubt dabei verschiedene Datenübertragungen. Es können entweder 4KByte, 32KByte, 64KByte, oder der ganze Chip gelöscht werden. Weil das Löschen selbst eines 4KByte Blocks jedoch mindestens 100ms, bis zu 400ms dauert, ist es uns nicht möglich das während des Betriebs zu tun. Die Option besteht jedoch für niedrige Sample-Frequenzen. In jedem Fall wird der gesamte Chip zu Beginn gelöscht, was bis zu 20-40 Sekunden benötigt. Die LED der Steckkarte signalisiert, wann die Karte einsatzbereit ist. Neue Daten können lediglich in 256Byte Blöcken geschrieben werden. Da bei so einer geringen Block-Größe allerdings ständig ausgelagert werden müsste und jede Menge Metadaten entstehen würden, wurde eine Block-Größe von 2KByte gewählt. Gelesen kann hingegen von jeder beliebigen Adresse mit voller SPI-Geschwindigkeit, ein beliebig großer Datenblock. Um das Paging zu vereinfachen werden jedoch auch beim Lesen immer exakt 2KByte gelesen.  
Um nicht den gesamten Schreiben, oder Leseprozess im Programmablauf warten zu müssen, ist der gesamte Flash-HAL Interrupt basiert. Soll beispielsweise ein Block gelesen werden, um an den Raspberry geschickt zu werden, so wird das Modul in den Modus „Lesen“ versetzt und die Lesen-State-Maschine aufgerufen. Diese startet die Übertragung für das erste Byte and den Flash (In diesem Fall 0x0b, das „Fast Read“ Kommando) und gibt die Kontrolle bereits an das COM-Handler-Modul zurück, der die noch nicht eingetroffenen Daten an den Raspberry Pi leitet. Sobald das „Fast-Read“ Kommando gesendet wurde, wird der SPI-Interrupt ausgelöst. In der zugehörigen Interrupt Service Routine wird wieder die Lesen-State-Maschine aufgerufen, die die nächsten drei Male die Adresse an den SPI-Flash übertragt und dann ein Dummy Byte. Nun sind die Daten im Flash bereit an den STM32 übertragen zu werden. Um nicht bei jedem Byte extra einen Interrupt auslösen zu müssen und die State-Maschine aufzurufen, wird hierzu ein DMA-Transfer gestartet, der die Daten in einen Auslese-Buffer schreibt. Das macht es möglich, dass sich der Datenaustausch mit dem Raspberry Pi und dem Flash überlappen. In dem Moment, in dem der Raspberry bereits erste Daten erhält, wird noch der Rest der Daten aus dem Flash nachgeladen. Da der Flash mit seinen 10MHz weitaus schneller arbeitet als die Verbindung mit dem Raspberry Pi (1MHz), kann es nicht passieren, dass der Raspberry Daten liest, die noch nicht vom Flash befüllt wurden. Ist dieser DMA-Transfer abgeschlossen wird wieder ein Interrupt ausgelöst, die State-Maschine zurückgesetzt und das Flash-Modul für andere Operationen freigegeben. Auch die Lösch und Schreib-Operationen können völlig parallel zum Programmablauf und anderen Interrupts arbeiten.

## Parameter-Modul

Das parameter-Modul dient als Schnittstelle für die anderen Module. Ist ein neues Datum vorhanden, so kann dieses mit der „new\_data()“-Funktion Eingespeist werden. Benötigt das COM-Handler Modul Daten eines Parameters, muss lediglich die „get\_datenblock()“-Funktion genutzt werden, um bis zu einem Block an Daten zu erlangen. Das Modul übernimmt automatisch im Hintergrund anfallendes Paging und stellt ein Array aller Parameter bereit mit den zugehörigen Funktionen. Wenn ein neuer Parameter implementiert werden soll, muss lediglich der Name in eine entsprechende Enum-Variable eingetragen werden und der Parameter in der Parameterliste konfiguriert werden. Ein Parameter besteht dabei aus einem Strukt mit dem Typ „parameter\_t“. Um einen Parameter zu parametrieren, benötigt er einen Namen, es muss festgelegt werden, ob es sich um einen im Graph darstellbaren Float-Parameter handelt, oder um einen nicht parametrierbaren String, der natürlich ebenfalls als Float dargestellt werden kann, von der Basis jedoch nicht in einer Formel verwendet werden kann. Auch die Schreibrechte des Parameters müssen festgelegt werden, sowie minimal und maximal-Werte. Machen min/max Werte keinen Sinn, muss dennoch ein Wert angegeben werden. Soll der Parameter Auswahlmöglichkeiten bereitstellen, muss ein string array als „wahl“ eingestellt werden. Wenn der Parameter im Graph angezeigt können werden soll, muss ein Zeiger auf ein Buffer-Element angegeben werden. Der Buffer und die Wahlmöglichkeiten müssen als Pointer übergeben sein und dann selbst definiert werden. Dieser Weg wurde gewählt, um möglichst Arbeitsspeicher zu sparen, da eine Karte signifikant viele String Parameter bietet, deren Buffer lediglich Speicher kosten würde, ohne genutzt zu werden. Das Buffer-Element besteht dabei aus drei 1 Block großen Buffern. Zwei Eingangsbuffern und einem Ausgangsbuffer. Um die Datenerfassung möglichst ohne weiteren Programmieraufwand erweitern zu können, kann gleich hier auch ein ADC-Channel des ADC1 eingetragen werden, sowie ein auf die Eingangsspannung anwendbarer Faktor und Offset. Soll der Parameter beschreibbar sein, muss auch noch ein Funktionspointer angegeben sein, dessen Funktion den empfangen String auswertet.

Wird ein neues Datum einem Parameter hinzugefügt und der aktive Eingangsbuffer ist voll, so wird der Startzeitpunkt des Blocks gesichert und der Eingangsbuffer wechselt, damit weiterhin neue Daten empfangen werden können. In regelmäßigen Abständen muss nun die Funktion „page\_out\_next()“ aufgerufen werden, die in allen Parametern nach einem so übergelaufenen Buffer sucht und bei jedem Aufruf nur einen Buffer durch den Flash-HAL auslagert. Dieses Verfahren wurde gewählt, weil mit hoher Wahrscheinlichkeit nicht nur ein Buffer, sondern alle Buffer überlaufen werden. Ein sofortiges Auslagern würde so länger als ein Sample Zyklus dauern und Daten verloren gehen. So wird die Last des Auslagerns von Blöcken über mehrere Sample-Zyklen verteilt. Um zu wissen, welcher Block im Flash mit welchem Parameter belegt ist, führt das Parametermodul eine Liste aller Blöcke, in der festgehalten ist, welcher Parameter den Block besetzt und bei welcher Startzeit der Block beginnt. Im Flash selbst sind somit lediglich die reinen Parameter-Werte gespeichert. Abhängig von dem verwendeten Flash-Chip und der eingestellten Block-Größe, nehmen selbst diese Meta-Daten einen beträchtlichen Speicherbereich in Besitz. Um das Finden eines neuen Speicherbereich möglichst performant zu gestalten und die Speicherzellen gleichmäßig zu belegen, merkt sich das Modul den zuletzt genutzten Speicherblock und versucht von da an nach neuem freiem Speicher. So kann auch ohne Löschen des Speichers maximal lang zuverlässig gemessen werden. Benötigt der COM-Handler Daten eines Parameters, wird zunächst in den gesamten Flash-Metadaten nach dem ältesten von diesem Parameter belegten Block gesucht. Diese Zeit muss die Basis mindestens von sich aus warten, bis Sie mit der Kommunikation fortfahren darf. Das entspricht ca. 1ms. Wird kein Block in den Flash-Metadaten gefunden, wird geprüft, ob gerade ein Block ausgelagert werden soll. In diesem Fall wird die Auslagerung abgebrochen und der volle Eingangsbuffer direkt an den COM-Handler übergeben. Ansonsten wird der gerade halb gefüllte aktive Eingangsbuffer übergeben, wobei dann eben kein ganzer Block übertragen wird. In Sachen Performance ist es enorm wichtig, dass die Daten in der chronologisch richtigen Reihenfolge an den Raspberry übertragen werden.

## Main-Modul

Wie bereits gesagt, beinhaltet die Main-Routine lediglich die Initialisierung des Systems und das Löschen des Flash-Speichers. Mit dazu gehört auch das anheben des System-Takts von 4MHz auf 80MHz durch einstellen der integrierten PLL. Da der interne Programmspeicher-Flash so schnellen Betrieb nicht unterstützt, mussten 2 Warte-Zyklen eingebaut werden, die die Abarbeitung des Programm-Codes bedeutend verlangsamen. Zusätzlich ist das Modul allerdings auch für das Abarbeiten eines Sample-events zuständig, dass über einen Pin-Change Interrupt ausgelöst wird. Dazu wird zum einen die ADC-Sequenz gestartet und alle nicht-ADC-basierten Parameter-Werte an das Parameter-Modul übergeben.

## ADC-Modul

Der STM32L431CBT6 verfügt über ein hoch komplexes ADC-Modul, von dem jedoch nur ein kleiner Teil genutzt wird. Das Peripheriemodul besitzt die Möglichkeit mit nur einem Trigger eine zuvor festgelegt Sequenz an ADC-Messungen durchzuführen. Diese Sequenz wird in der Initialisierung anhand der parameterliste gefüllt. Für jeden Kanal kann einzeln die Sample-Zeit eingestellt werden. Es wird standardmäßig für alle Kanäle 10us eingestellt. Jede Messung löst einen End of conversion ADC Interrupt aus, in dem der ADC-Wert gelesen wird, mit dem eingestellten Faktor und Offset verrechnet wird und an das Parameter-Modul übergeben. Ist die Sequenz fertig durchgelaufen, wird das auslagern eines Buffers ermöglicht, indem „page\_out\_next()“ des Parameter-Moduls aufgerufen wird. Der ADC bietet die Möglichkeit, Channels überabzutasten. Das wird auch genutzt und somit eine 16bit-Auflösung erzielt.

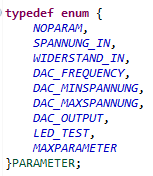
## DAC-Modul

Das Dac-Modul entspricht einem Beispiel einer Applikationssoftware. In diesem Fall wird ein Funktionsgenerator implementiert. Dieser hört auf die Parameter „Output max.“, „Output min“ und „Output freq.“. Es wird ein Dreiecksignal erzeugt, dass mit der gegebenen Frequenz zwischen den beiden Spannungen schwingt. Dazu wird ein extra Timer und das DAC-Modul des STM32 genutzt. Dieses bietet eine Auflösung von 12bit. Die momentan ideal ausgegebene Frequenz kann unter dem Parameter „Output voltage“ zur Grapherstellung genutzt werden.

# Hinzufügen eines Parameters

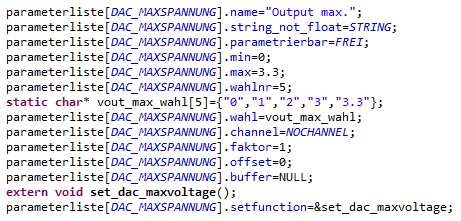
Einen neuen Parameter hinzuzufügen benötigt ein minimales Eingreifen ins System.

### „parameter.h“:

in parameter.h muss die intern verwendete Parameternummer in einer Enumerationsvariable definiert werden. Die somit eindeutig zugewiesene Nummer wird auch in der Basis zu Identifikation des Parameter verwendet. Der Eintrag muss dabei nach „NOPARAM“ und vor MAXPARAMETER erfolgen. Die Nummer NOPARAM=0 wird in den Metadaten des Flash-Speichers genutzt, um freie Speicherbereiche zu kennzeichnen. Mit MAXPARAMETER wird die Anzahl der Parameter ermittelt und dementsprechend die parameterliste dimensioniert.

### „parameter.c“:

Soll ein Parameter in der Grapherstellung genutzt werden, muss für ihn ein Buffer-Element bereitgestellt werden.

In der parameterliste muss der Parameter konfiguriert werden. Dazu gehören unter Umständen auch Wahlmöglichkeiten, Buffer und SET\_PARAMETER Funktion.

### „adc.c“:

Weil aus den Nummern der ADC-Kanäle nicht der Pin hervorgeht, an den der Kanal verbunden ist, muss der Pin hier initialisiert werden. Dazu eignet sich der GPIO-HAL.