Softwareentwicklung eines Messsystems zur Spannungswinkelmessung auf Basis von Spannungsnulldurchgangsdetektion

Bachelorarbeit

Elektrotechnik – Elektrische Energietechnik,
Technische Hochschule Köln,
Cologne Institut for Renewable Energy

vorgelegt von: Martin Gut
Matrikel-Nr.: 11111564
Adresse: Am Bahnhof 88
51147 Köln

Erstgutachter: Prof. Dr. Eberhard Waffenschmidt (Technische Hochschule Köln)
Zweitgutachter: Prof. Dr. Ingo Stadler (Technische Hochschule Köln)

Köln, 28.09.2021

cire Cologne Institute for Renewable Energy

Technology Arts Sciences TH Köln

Erklärung

Erklärung zum eigenständigen Verfassen

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, 28.09.2021 — Martin Gut

Ort, Datum — Rechtsverbindliche Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meiner Bachelorarbeit unterstützt und motiviert haben. Zuerst gebührt mein großer Dank an *Herrn Prof. Dr. Waffenschmidt*, der es mir ermöglicht hat ein Teil des Progressus-Projekts zu sein und mir die Möglichkeit gegeben hat, diesen Themenschwerpunkt zu bearbeiten. Besonderer Dank gilt auch *Prof. Dr. Stadler* für die Übernahme des Koreferats. Selbstverständlich möchte ich mich an meine Betreuer *Herr Hotz* und *Herr Baum* meinen größten Dank aussprechen. Sie haben mich durch die gesamte Zeit über begleitet und standen mir jederzeit mit wertvollen Ratschlägen zur Seite. Ich konnte viel von ihnen lernen. Zusätzlich gilt mein Dank noch Herrn Baum, welcher sich die Zeit genommen hat, meine Arbeit auf Korrektur durchgelesen und Optimierungsvorschläge genannt hat. Des Weiteren einen großen Dank an meine Familie, als auch meine Freunde, die in den anstrengendsten Momenten an meiner Seite standen und mich durchgehend motiviert haben. An erster Stelle kommt meine Mutter *Natalia Gut*, die mich in jeder schweren Situation wieder aufgebaut hat und mich daran erinnert hat, den Fokus nicht zu verlieren. Ich bedanke mich bei meinen Freunden, Kommilitonen und meiner Lerngruppe *Ahmed Omer, Yassin Hemchi, Viktor Lucas, Sascha Jannsen* und *Pietro Ribecco,* die mich während des Studiums unterstützt und für eine gute Arbeitsatmosphäre gesorgt haben.

Zum Schluss bedanke ich mich bei allen Professoren und Mitarbeiter der technischen Hochschule, die mich bei meinem Weg durch das Studium begleitet haben.

Kurzfassung

Gegenstand der Arbeit ist die Softwareentwicklung eines Messsystems zur Bestimmung des Spannungswinkels auf Basis einer Sinusnulldurchgangsdetektion, welches in großflächigen Messstandorten eingesetzt werden soll. Die Abschlussarbeit ist eine Fortsetzung und Weiterentwicklung des von mir im März 2021 durchgeführtem Praxisprojekts zur *„Softwareentwicklung zu einem GPS-synchronisierten Phasor-Measurement-Unit auf Basis einer Spannungsdurchgangdetektion“*. In dem Praxisprojekt wurde eine Software entwickelt, die den Logikpegel Abfall eines Rechtecksignals erkennt und mit einem hochpräzisen Zeitstempel markiert. Um den Zeitstempel zu erhalten, wird die Entwicklungsplatine mit einem hochpräzisen GPS-Zeiterfassungsmodul synchronisiert, welcher alle wichtigen GNSS-Daten empfängt. Des Weiteren beinhaltet die entwickelte Software alle notwendigen Konfigurationseinstellungen für die Schnittstelle zwischen den beiden Hardwarebauteilen. Für die Entwicklung des Messsystems wird ein zweites Messgerät angefertigt, welches mit der gleichen entwickelten Software implementiert wird. Beide Messgeräte werden zeitgleich synchronisiert und gestartet. Alle aufgenommenen Messwerte werden gespeichert und über einen eingerichteten Samba-Server, der die Funktionalität eines „*Synchrophasor Communication System*“ hat, freigeben. Durch ein weiteres Programm, welches die Aufgabe eines „*Phasor Data Concentrators“* Systems hat, werden die aufgenommenen Messdaten aufgerufen, auswertet und der Spannungswinkel berechnet. Mit dem Spannungswinkel wird die Lage des elektrischen Energieversorgungsnetzes anhand von zwei Rechtecksignalen an jeweils zwei unterschiedlichen Einsatzorten untersucht.

Abstract

The subject of the thesis is the software development of a measurement system for the determination of the voltage angle based on a sine zero crossing detection, which is to be used in large-scale measurement sites. The thesis is a continuation and further development of the practical project on *"Software development for a GPS-synchronized phasor measurement unit based on voltage crossing detection",* which I carried out in March 2021. In the practical project, software was developed that detects the logic level drop of a square wave signal and marks it with a high-precision time stamp. To obtain the time stamp, the development board is synchronized with a high-precision GPS time acquisition module, which receives all important GNSS data. Furthermore, the developed software contains all necessary configuration settings for the interface between the two hardware components. For the development of the measuring system a second measuring device is manufactured, which is implemented with the same developed software. Both measuring devices are synchronized and started at the same time. All recorded measured values are stored and released via an established Samba server, which has the functionality of a "*Synchrophasor Communication System*". A further program, which has the task of a "*Phasor Data Concentrator"* system, calls up the recorded measurement data, evaluates it and calculates the voltage angle. The voltage angle is used to examine the position of the electrical power supply network based on two square-wave signals at two different locations in each case.

Inhaltsverzeichnis

Erklärung ..... **II**
Danksagung ..... **III**
Kurzfassung ..... **IV**
Abstract ..... **V**
Inhaltsverzeichnis ..... **VI**
1. Einleitung **1**
1.1 Methodisches Vorgehen 1
2. Stand der Technik **4**
2.1 Wide Area Monitoring System – (WAMS) 4
2.1.1 Phasor Measurement Unit – PMU 6
2.1.2 Phasor Data Concentrator – PDC 8
3. Theoretische Grundlagen der Hardware **9**
3.1 Auftretende Probleme an einem Niederspannungsnetz 9
3.2. WAMS-Lösungsansatz 12
3.3 Entwicklungsplatine – Raspberry Pi 13
3.4 Zeiterfassungsmodul – GNSS-5-Click 15
3.4.1 Zeitermittlung durch Pulse-Per-Second – PPS-Signal 16
3.4.2 UART – Serielle Kommunikationsschnittstelle 18
3.5 Komparatorschaltung 20
3.6 Samba-Server 21
3.6.1 Konfiguration des Samba-Servers 21
3.6.2 Samba-Freigabe für das Verzeichnis „share“ 22
3.7 Entwicklung des Messsystems 23
4. Softwareentwicklung **25**
4.1 Organigramm – PMU-Software 25
4.2 Software des PMU-Systems 26
4.2.1 Startpunkt der Zeitmessung 28
4.2.1.1 Zeitpunkt der Messdatenspeicherung 29
4.2.1.2 Wiederaufnahme einer Messreihe 30
4.2.2 Untersuchung der Messwerte 30
4.3 Organigramm - PDC-Software 31
4.4 Software des PDC-Systems 32
4.4.1 Einstellung der Systemkonfiguration 33
4.4.2 Startpunkt des Hauptprogramms 34
4.4.3 Überprüfung der PMU-Messdaten 36
4.4.4 Übersetzung des Datentyps 38
4.4.4.1 Funktion – reset() 39

4.4.5 Untersuchung der Zeitstempel ... 40
4.4.6 Mittelwertberechnung ... 41
4.4.7 Berechnung der Phasenverschiebung ... 43
4.4.8 Rasterverschiebung der Spannungswinkeln ... 44
5. Messwertaufnahme ... 46
5.1 Phasenverschiebung zwischen beiden PMU-Messgeräten ... 46
5.1.1 Messabweichung ... 49
5.1.2 Zusammenfassung ... 50
5.2 Phasenverschiebung unter einer Last ... 51
5.2.1 Simulationsumgebung ... 51
5.2.2 Simulationsmessung ... 53
6. Fazit ... 54
7. Ausblick und Diskussion ... 56
Abbildungsverzeichnis ... 58
Tabellenverzeichnis ... 60
Formelverzeichnis ... 61
Abkürzungsverzeichnis ... 62
Literaturverzeichnis ... 63
Anhang ... 67
Anhang 1: Einstellung und Freigabe des Samba-Servers ... 67
Anhang 2: Prototypentwicklung eines PMU-Messgerätes für das WAMS ... 69
Anhang 3: Anleitung zur Inbetriebnahme des WAMS ... 70

1. Einleitung

Um in der heutigen Welt den Ausstoß von schädlichen Klimagasen zu reduzieren, ist dafür besonders eine Wandelung des Energienetzes vom reinen Verteilernetz bis hin zum Smart Grid erforderlich [1]. Dabei hat Deutschland und Europa sich ehrgeizige Ziele mit dem Schwerpunkt des Klimaschutzes und Reduktion von CO_2 gesetzt [1]. Unter anderem arbeitet die Technische Hochschule Köln in Kooperation mit internationalen Partnern an dem Progressus-Projekt, welche das Ziel verfolgen die Spitzenbelastung durch Ladeinfrastruktur von Elektrofahrzeugen um 30 % zu reduzieren [1]. Dies soll vor allem durch die Entwicklung und Forschung von effizienter Energiewandlung, die Nutzung von lokalen Speicherbatterien und durch intelligentes Energiemanagement für die Mikrogrids erreicht werden [1]. Die Technische Hochschule Köln beteiligt sich bei der Entwicklung von intelligenten Algorithmen, mit denen Smart Meter Gateways, wie Ladeboxen für E-Mobilität oder ganze Hausanschlüsse analysiert, ans Netz angeschlossen und der elektrische Netzzustand, wie der aktuelle Spannungswinkel, elektrische Parameter oder der Leistungsfaktor untersucht und untereinander ausgetauscht werden soll [2]. Des Weiteren werden Algorithmen entwickelt, die den Netzbetrieb unter Berücksichtigung der maximalen Transformatorleistung, der Leistungsbelastung sowie die maximale Zulässigkeit des Spannungsabfalls an seine Grenzen untersuchen und bestimmen [2]. Zusätzlich forscht die Technische Hochschule Köln an dem Teilbereich der Optimierung von Smart Meter Gateways in Kombination mit einem gesamten Hausenergiesystem. Dazu gehören z.B. Ladeboxen für die E-Mobilität, Wärmepumpen und Photovoltaikanlagen [2]. Zu der Optimierung gehört das Hausenergiesystem und der Zustand des gesamten Niederspannungsnetzes. Ebenso wird die Kommunikation zwischen den unterschiedlichen Smart Meter Grid innerhalb eines Niederspannungsnetzes verbessert, um die Anschlusskapazität zu maximieren und zur selben Zeit die Netzfreundlichkeit zu gewährleisten [1]. Es werden verschiedene Szenarien auf Genauigkeit untersucht und ausgewertet. Die entwickelten Lösungsergebnisse sollen mit den bereits vorhandenen Installationen kompatibel und ausrüstbar sein.

1.1 Methodisches Vorgehen

Um für eine nachhaltige Stromversorgung zu sorgen, werden Generatoren, Lasten und die Übertragungsleitungen des elektrischen Energieversorgungsnetzes durchgehend überwacht und kontrolliert. Für das Energieversorgungsnetz werden bereits entwickelte Überwachungssysteme installiert. In den letzten Jahren kommen in der Netzüberwachung sogenannte „*Wide Area Monitoring System*“ (WAMS) zum Einsatz. Ein Nachteil eines solchen WAMS Überwachungssystems ist preislich bedingt, woraus sich das Forschungsthema dieser Abschlussarbeit herauskristallisiert hat. Das Ziel der Abschlussarbeit ist es, ein kostengünstiges WAM-System zu entwickeln, dass möglichst präzise den Phasenwinkel auf Basis von Nulldurchgangsdetektion einer angelegten Netzspannung erkennt und auswertet.

Um eine zeitliche Zustandsveränderung im Messsystem zu erfassen, werden jeweils zwei identische *„Phasor Measurement Unit“* (PMU) Messgeräte mit der gleichen Hardware und Software benötigt, welche dann in großflächigen Gebieten installiert werden sollen. Die beiden entwickelten PMU-Messgeräte werden jeweils an zwei unterschiedlichen Punkten des zu messenden Energieversorgungsnetzes implementiert, um mittels der erfassten zeitsynchronisierten Messwerte, die Phasenverschiebung bestimmen zu können.

Beide PMU-Messgeräte sollen die Eigenschaft besitzen jede eintreffende fallende Flanke einer $50\ \mathrm{Hz}$ Rechteckspannung zu erfassen und mit einem hochpräzisen Zeitstempel zu markieren. Wichtig dabei ist es, dass die erkannten fallenden Flanken mit einer Auslesegeschwindigkeit von $f_{min} = 50\ \mathrm{kHz}$ erfasst werden soll, um eine möglichst geringe Zeitverschiebung zwischen den beiden Rechtecksignalen zu messen. Das entwickelte Messsystem soll im Progressus-Projekt ausschließlich nur zur Spannungswinkelmessung eingesetzt werden.

Die Schwerpunkte der Abschlussarbeit lassen sich in vier Kapitel aufteilen:

i. Aktueller Forschungsstand der Technik
ii. Theoretische Grundlagen und Fertigstellung des Messsystems
iii. Software zur Spannungswinkelmessung
iv. Auswertung der Messdaten und Schlussfolgerung

Zunächst wird in dem ersten Kapitel (i.) auf den aktuellen Forschungsstand eines WAMS eingegangen. Zusätzlich wird die grundlegende Funktionalität eines WAMS genannt und der genaue Systemaufbau beschrieben, welcher aus Phasor Measurement Units, Synchrophasor Communication System und Phasor Data Concentrator besteht. Damit wird Verständlichkeit für die Entwicklung eines solchen Messsystem und der Verwendungszweck unter den einzelnen Bauteilen geschaffen.

Das darauffolgende Kapitel (ii.) beschreibt einen vereinfachten Netzstrahl eines Niederspannungsnetzes und mit welchen kritischen Problemen das Niederspannungsnetz belastet wird. Unter anderem wird dabei auf alle verwendeten Hardwarebauteile, die in der Abschlussarbeit für die Umsetzung eines solchen Messsystems notwendig sind, eingegangen und detailliert ihre Funktionalität beschrieben.

Im dritten Kapitel (iii.) der Abschlussarbeit wird die entwickelte Software des WAMS beschrieben. Dabei beinhaltet das Kapitel eine weiterentwickelte Version von dem im Praxisprojekt erstellten *„Zeitstempel.c“* Programm und die Software zur Spannungswinkelberechnung (*Spannungsphasenwinkel.c*). Es wurden Algorithmen entwickelt, die alle sämtliche Messungenauigkeiten identifizieren und aus der Berechnung entfernen. Die Softwareprogramme werden in mehrere Abschnitte unterteilt und zeilenweise detailliert beschrieben. Zudem werden alle Funktionen, genauso wie die dazugehörigen Bibliotheken, Variablen und Voreinstellungen im Programmcode erläutert.

Im letzten Kapitel „Auswertung der Messdaten und Schlussfolgerung“ (iv.) werden alle aufgezeichneten Messergebnisse analysiert und eine Schlussfolgerung aus ihnen

gezogen. Es werden weitere Schritte und Verbesserungsmöglichkeiten für Forschungen in diesem Themengebiet abgeleitet und vorgeschlagen.

2. Stand der Technik

In diesem Kapitel wird ein Überblick über das Wide Area Monitoring System (WAMS) mit dem das Energieversorgungsnetz überwacht wird, geschaffen. Dazu wird der aktuelle Entwicklungsstand eines WAMS beschrieben und weshalb WAMS in Smart Grid oder Niederspannungsnetzen installiert wird. Am Ende des Kapitels wird zusätzlich noch detailliert auf die einzelnen Eigenschaften der Baugeräte, aus denen ein Wide Area Monitoring System besteht, eingegangen.

2.1 Wide Area Monitoring System – (WAMS)

Das wirtschaftliche und politische Umfeld in der Elektrizitätswirtschaft führt in Europa als auch in den USA zu einer starken Auslastung der Übertragungsnetze [3]. Werden Energieversorgungssysteme zu nah an ihrer Belastungsgrenze betrieben, steigt die Gefahr von schwerwiegenden Stromausfällen oder Großstörungen [3]. Um eine Stabilität von wichtigen Netzparametern in Echtzeit gewährleisten zu können, werden elektrische Energieversorgungsnetze durchgängig mithilfe von installierten Überwachungssystemen kontrolliert. Durch den Einsatz von Wide Area Monitoring System werden die Energieversorgungsnetze überwacht, ausgewertet und durch eine komplexe Messsystemanalyse Stromausfälle verhindert [3]. Im Gegensatz zu bislang installierten konventionellen Schutzgeräten (SCADA), welche den lokalen Schutz vor einzelnen Komponenten (Transformatoren, Generatoren oder Leitungen) gewährleisten, bietet das WAM-System übergreifend Schutz für das gesamte Netz [4 S. 4]. Um einen Schutz für das globale Netz zu schaffen, wurden die Messgeräte der konventionellen Systeme mit Globalen Position System (GPS) Empfängern integriert [3 S. 1]. Das weiterentwickelte Messgerät wird auch als Phasor Measurement Unit (PMU) bezeichnet, welches in Kapitel 2.1.1 genauer beschrieben wird.

Das WAMS besteht aus drei unterschiedlichen Bauteilen, die alle eine eigene Funktionalität haben. Zum einen besteht das WAMS aus mehreren „*Phasor Measurement Units*“ (PMU), welche Messwerte über den Zustand des jeweiligen zugehörigen Messpunktes in Form von zeitgestempelten Spannung- und Stromamplitude aufzeichnen. Zum anderen wird für die Verarbeitung der aufgezeichneten Messwerte ein PDC-System eingesetzt. Das *„Phasor Data Concentrator*“ (PDC) dient dabei als eine Art Kommunikationszentrale, das die empfangenen PMU-Messdaten über des zu untersuchenden Energieversorgungssystems miteinander vergleicht und im Anschluss darauf das Energieversorgungssystem beurteilt. Als Kommunikationsschnittstelle zwischen den PMU-Messgeräten und dem PDC-System wird ein „*Synchrophasor Communication System*“ (SPCS) eingerichtet. Über die SPCS-Schnittstelle werden alle PMU-Messwerte gesammelt und weiter an das PDC-System geleitet. Abbildung 2 verdeutlich den beschriebenen Messaufbau eines installierten WAM-Systems und die genaue Kommunikation zwischen den einzelnen Bauteilen.

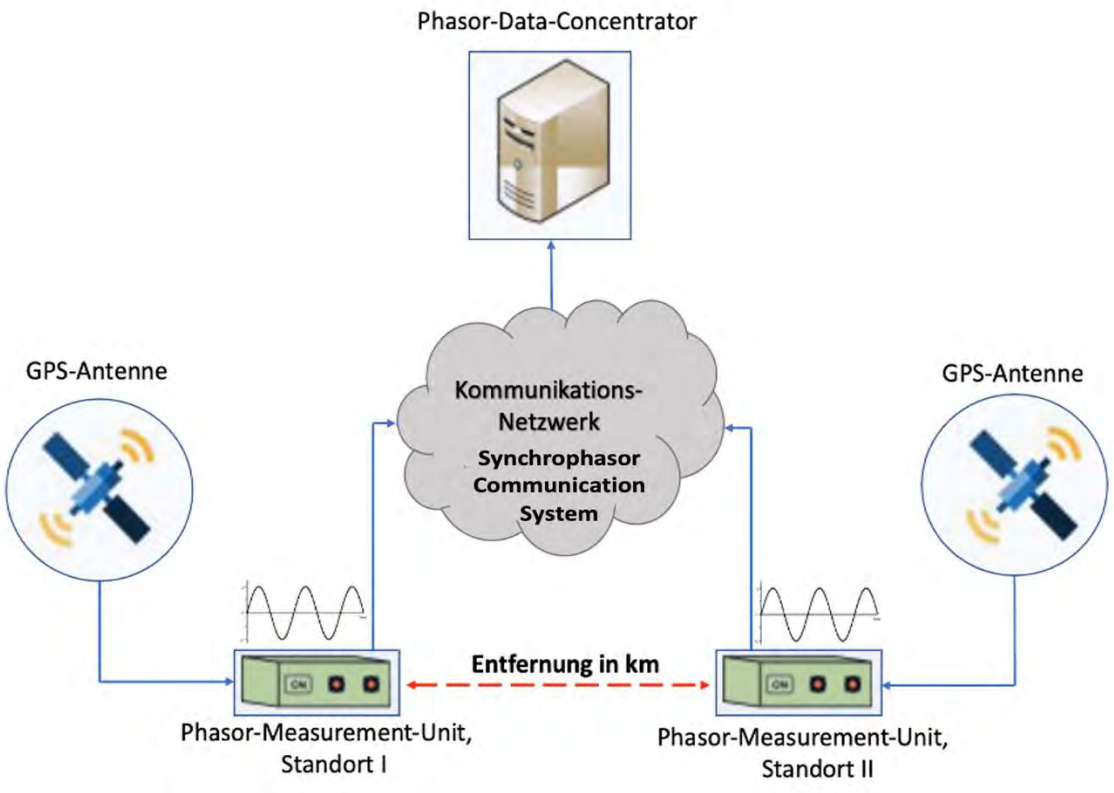

Abbildung 1: System- und Kommunikationsaufbau eines Wide Area Monitoring Systems - In Anlehnung an [38]

Heutzutage werden WAM-Systeme weltweit von verschiedenen Firmen entwickelt und als Schutzsystem gezielt eingesetzt. Positive Erfahrungen zeigen beispielsweise die Berichte der Firma ABB, die eigene WAMS als separate Systeme entwerfen und in Betrieb nehmen. Darunter hat die ABB das *PSGuard,* welches ein WAM-System ist, entwickelt. Das PSGuard entspricht dem neuesten Forschungsstand im Bereich der Überwachungssysteme und kommt u.a. aktuell in der Schweiz, Österreich, Griechenland, Kroatien und Finnland zum Einsatz [4]. Das PSGuard-System wird mit GPS-synchronisierten PMU-Messgeräten ausgerüstet und ermöglicht eine Messwertaufnahme mit einer Genauigkeit unter $1\ \mu s$ [6 S. 5].

Aber auch Firmen wie beispielsweise Tennet TSO GmbH oder Siemens forschen an der Entwicklung von WAM-Systemen. Dabei beschäftigt sich die Tennet TSO GmbH seit 2012 mit der Installation von PMU-Messgeräten, die Daten mit weiteren ÜNB des ENTSO-E-Verbundsystems austauschen [6 S. 13]. Somit wurden erste Erfahrungsberichte für ein europaweites Überwachungsnetzwerk gesammelt [6 S. 13]. Technische Informationen zu den installierten PMU oder PDC-System werden von der Tennet TSO GmbH nicht preisgegeben. Siemens hingegen hat die PMU-Messgeräte SIPROTEC 5, SIMEAS R-PMU und SIMEAS R entwickelt, die weltweit eingesetzt werden. Allein in Deutschland hat Siemens bereits sieben SIMEAS-R PMU-Messgeräte erfolgreich in das Versorgungsnetz installiert. Als zentrales Kontrollsystem wird das SIGUARD PDP verwendet, welches seinen Stand in Bayreuth (Bayern) hat [37 S. 22]. Mit den PMU-Messgeräten von Siemens werden Messwerte aufgezeichnet die ebenfalls $\leq 1\ \mu s$ genau sind

[32], [33 S. 9]. Weitere technische Unterschiede zwischen den genannten PMU-Messgeräten sind in der Tabelle 1 abgebildet.

	SIPROTEC 5	SIMEAS R-PMU	SIMEA R V3.0
Abtastfrequenz	320 Abtastungen pro Periode Bei 50 Hz = 16 kHz	192 Abtastungen pro Periode Bei 50 Hz = 11,52 kHz	256 Abtastungen pro Periode Bei 50 Hz = 12,8 kHz
Zeitsynchronisierung	GPS-Zeitsynchronisierung (< 1 µs)	GPS-Zeitsynchronisierung (< 5 µs)	GPS-Zeitsynchronisierung (< 5 µs)
Phasor Data Concentrator (Software)	SIGUARD PDP	OSCOP P, SIGUARD PDP	OSCOP P
Messgenauigkeit der Phasoren	≤ 1 µs	≤ 1 µs	≤ 1 µs
Reporting Rate	Kontinuierliches Update mit 10 Werten pro Sekunde	Kontinuierliches Update mit 10, 20, 50 Werten pro Sekunde	Kontinuierliches Update mit 10, 20, 50 Werten pro Sekunde
Entwicklungsjahr	2018	2009	2009
Preis	ca. 3000 €	unbekannt	unbekannt

Tabelle 1: Technische Unterschiede der PMU-Messgeräte der Firma Siemens [32], [33 S. 9-17]

Die größte Besorgnis stellen die Entwicklungskosten bei der Umsetzung einer solchen WAMS Technologie dar. Der durchschnittliche Gesamtpreis pro PMU-Messgerät liegt zwischen 40.000 und 180.000 $ (Kosten für Beschaffung, Installation und Inbetriebnahme) [7 S. 29]. Aufgrund des enormen Preises wurden in den letzten Jahren unterschiedliche Forschungen betrieben, um eine kostengünstige Variante eines WAMS herzustellen. Diese werden im Folgenden ansatzweise beschrieben.

Im OPENPMU-Projekt wurde eine WAMS-Technology entwickelt, welches zwei PMU-Messgeräte und ein PDC-System umfasst. Die Herausforderung des Projektes war es ein echtzeitfähiges PMU-Messgerät zu entwickeln, welches mit einem GPS-Zeiterfassungsgerät synchronisiert ist. Es werden jedoch keine genauen Details zur verwendeten Hardware-Implementierung oder Leistung der Messgeräte genannt. Für die gesamte Entwicklung des Messsystems wurde etwa 1000 $ investiert [34].

Auch das Institut für Informations-, Medien- und Elektrotechnik der Technischen Hochschule Köln hat sich mit dem Problemansatz auseinandergesetzt und Forschung in diesem Bereich betrieben. Dabei besteht das in [8] entwickelte PMU-Messgeräte jeweils aus einem Transformator und einem Mikrocontroller. Auch hier wurde für eine präzise Zeiterfassung jedes PMU-Messgerät mit einem GPS-Modul synchronisiert. In dieser Abschlussarbeit wurden zwei PMU-Messgeräte entwickelt, welche eine Phasenverschiebung von 0,1° bis 0,3° erzielen [8 S. 62 ff]. In [8] wird kein genauer Preis für die Umsetzung eines solchen Messsystem genannt.

2.1.1 Phasor Measurement Unit – PMU

Phasor Measurement Unit (PMU) oder auch Spannungswinkelmessgerät genannt, zeichnen Zeigerdaten (Phasor) von Spannung, Strom und Leistung eines Energieversorgungsnetzes zu einem festen Zeitpunkt auf. Zeiger sind eine vereinfachte Darstellung

von sinusförmigen Signalen, welche aus Betrag und Phase bestehen. Dabei wird jeder erfasste Zeigerwert mit einem hochgenauen Zeitstempel versehen. Um die Nulldurchgänge der Sinussignale mit hochgenauen Zeitstempel zu markieren, werden die PMU-Messgeräte mit hochpräzisen GPS-Zeiterfassungsmodul synchronisiert. Durch eine hochgenaue GPS-Zeitsynchronisierung werden alle erfassten Messdaten von den PMU-Messgeräten, die sich an verschiedenen geografischen Gebieten voneinander entfernten Messstationen positioniert sind, miteinander verglichen und die Phasenverschiebung bestimmt. Die Messgenauigkeit der Zeitstempel beträgt unter einer Mikrosekunde bei einer Systemfrequenz von $50\ \mathrm{Hz}$, welche eine Spannungswinkelgenauigkeit von unter 0,018° entspricht [3 S. 2].

Aus der Phasenverschiebung lässt sich eine Aussage über die Stabilität des elektrischen Energieversorgungsnetzes schließen. Die Abbildung 2 verdeutlicht die genaue Phasenverschiebung zweier unterschiedlicher Sinusspannungen, die an zwei Punkten in einem Niederspannungsnetz aufgenommen wurden.

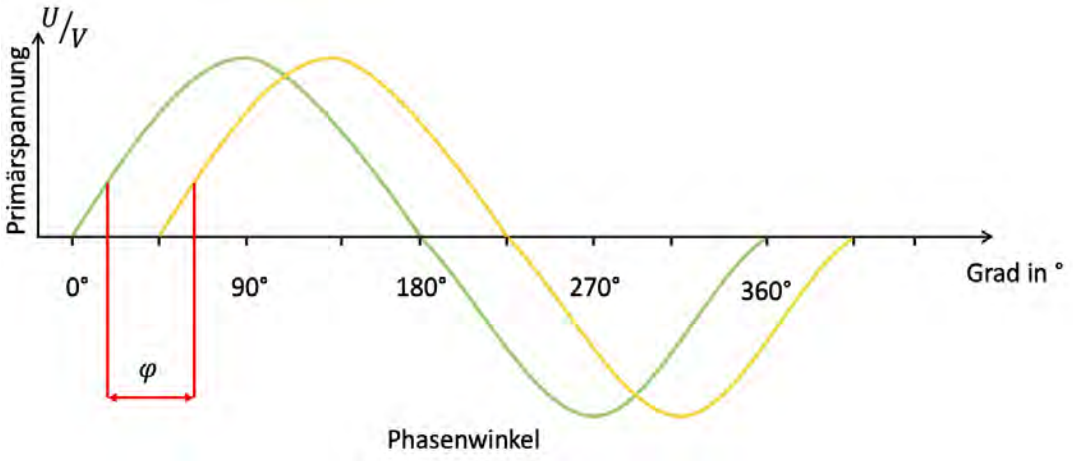

Abbildung 2: Phasenverschiebung zwischen zwei Sinussignalen [35]

Dabei stellt das grüne Sinussignal das Referenzsignale dar, wohingegen die gelbe Spannung, das verschobene Sinussignal ist. Aus der Darstellung lässt sich schließen, dass die gelbe Sinusspannung dabei um 45° der grünen Referenzspannung vor eilt. Da beide Sinussignale mit einer Frequenz von $50\ \mathrm{Hz}$ periodisch Schwingen lässt sich eine Aussage über die Lage des Spannungswinkels machen.

Eine Phasor Measurement Unit nimmt im Vergleich zu bislang verwendeten „*Supervisory Control and Data Acquisition*“ (SCADA) Überwachungssystem bis zu 60 Messaufnahmen pro Sekundenzyklus auf. Ein SCADA Überwachungssystem kann stattdessen nur zwei bis vier Messaufnahmen pro Sekundenzyklus erfassen [31], [5]. Somit liefert das PMU-Messgerät sekündlich Messdaten die um das 15 bis 30 fache höher sind. Bei einer solchen großen Messaufzeichnung pro Zeitzyklus ist die Datenmenge enorm und benötigt eine stabile Kommunikations- und Serverinfrastruktur. In der Tabelle 2 werden weitere Unterschiede zwischen den beiden bekanntesten Überwachungsmesssystemen genannt und miteinander verglichen [10 S. 12], [31 S. 4].

System	SCADA	PMU
Messwerte	Effektivwerte (RMS)	Phasoren
Zykluszeit	2 bis 30 s. (z. T. bis 5 min)	20 bis 100 ms. (reporting rate)
Zeit-Synchronisierung	lokale Zeitstempel	Globale Zeitstempel (GPS)
Phasenwinkelmessung	Nein	Ja
Überwachung	Lokale	große Gebiete

Tabelle 2: Unterschied zwischen beiden bekannten Überwachungsmesssystemen SCADA und PMU [10 S. 12]. [11], [12], [13], [31 S. 4]

2.1.2 Phasor Data Concentrator – PDC

Der Phasor Data Concentrator (PDC) ist das zentrale Kontroll- und Kommunikationszentrum des WAM-Systems. Alle von den PMU-Messgeräten aufgezeichneten Messwerte werden über die SPCS-Schnittstelle an das PDC-System weitergeleitet. Da alle aufgezeichneten Messungen mit einem GPS-Zeiterfassungsmodul erfasst und markiert werden, ist das PDC-System in der Lage die Messungen zeitlich miteinander abzugleichen und die Systemlage der aktuellen elektrischen Energieversorgung auszuwerten. Innerhalb einer Datenübertragung werden folgende Messgrößen dem PDC-System gesendet. Die Datenübertragung wird im IEEE-Standard C37.118.2 (2011) definiert [9] [10 S. 12].

- Synchronisationsfeld
- ID der Datenquelle (PMU und PDC)
- Zeitstempel
- Statuswörter, u.s. für Fehlerereignisse, Trigger und PMU-Zeitabweichung
- Änderungsrate der Frequenz
- Phasorenmesswerte
- Weitere analoge und digitale Statuswörter

Das PDC-System wird über das Internet-Netzwerk mit den installierten PMU-Messgeräten synchronisiert. In der Abschlussarbeit wird ein Samba-Server eingerichtet, welcher das Synchrophasor-Kommunikationssystem zwischen beiden entwickelten PMU-Messgeräten bildet und als Messdatenaustausch dient.

3. Theoretische Grundlagen der Hardware

In diesem Kapitel werden alle notwendigen theoretischen Grundlagen beschrieben und erklärt, welche für die Umsetzung eines WAMS wichtig sind. Zu Beginn werden die Grundlagen zur notwendigen Netzüberwachung und auf die Problematik, die an einem vereinfachten Netzstrahl im Niederspannungsnetz entstehen, genannt. Gleichzeitig wird noch die ausgewählte Hardware beschrieben, um eine WAMS-Software nach den gestellten Anforderungen entwickeln zu können. Für die Entwicklung eines WAMS wurden folgende Hardwarebauteile für die Abschlussarbeit gewählt. Für die Entwicklung der PMU-Messgeräte wird ein Raspberry Pi Model 3 B+ und 4 B der Firma „*Raspberry Foundation*“ verwendet, auf denen die entwickelten Programme kompiliert und gestartet werden. Für die Zeitmessung und Erstellung des daraus resultierenden hochgenauen Zeitstempels wurde ein GPS-zeitsynchronisiertes-Erfassungsmodul, der „*GNSS-5-Click*“ verwendet.

3.1 Auftretende Probleme an einem Niederspannungsnetz

Die Abbildung 3 beschreibt vereinfacht einen Netzstrahl, der aus mehreren Knotenpunkten besteht und über einen Transformator an ein Versorgungsnetz angeschlossen ist. An jedem Knotenpunkt der Versorgungsleitung werden Verbraucher angeschlossen, die als Lasten fungieren und beispielsweise Ladestationen sind, die zu jedem Zeitpunkt, elektrische Fahrzeuge versorgen sollen. Dabei kann es dazu kommen, dass an einer Versorgungsleitung des Niederspannungsnetzes es zu einer Überbelastung an einem Knotenpunkt kommt. Die Überbelastung führt dazu, dass kritische Netzparameter überschreiten und die betroffene Knotenstelle beschädigt wird [13 S. 44 ff]. Verursacht wird dies vor allem, wenn zur selben Zeit alle elektrischen Fahrzeuge aufgeladen werden und in Folge darauf eine maximale Belastung der gesamten Versorgungsleitung ausgelöst wird.

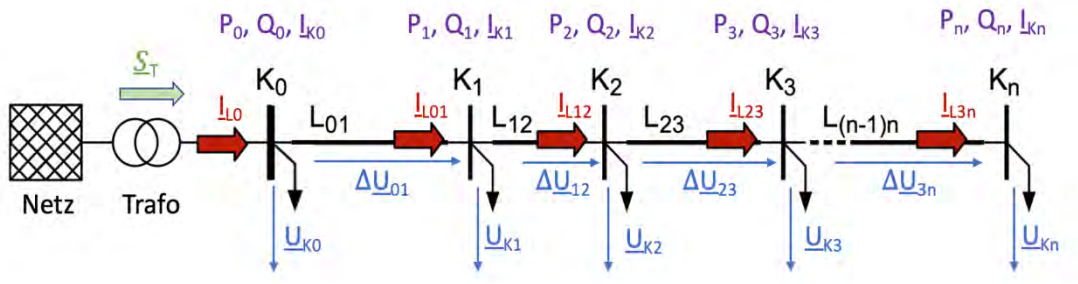

Abbildung 3: Vereinfachte Darstellung einer Versorgungsleitung - In Anlehnung an [29]

Zu den kritischen Netzparametern zählt, die Überschreitung des Netzstromes, der Nennscheinleistung und der Spannungsabfall zwischen zwei Knotenpunkten an einer Versorgungsleitung. Durch das Überschreiten des maximalen Spannungsabfalls $\Delta\underline{U}$ zwischen zwei Knotenpunkten an der Leitung, wird die Versorgungsleitung beschädigt. Verursacht

wird die Beschädigung schon durch eine Überschreitung des Spannungsabfalls $\Delta\underline{U}$ von $\pm$ 3 % [39]. Ein weiterer Punkt ist die Belastung durch zu hohe Netzströme. Dadurch wird die maximale Scheinleistung in den untergebrachten versorgenden Ortstransformators erhöht und überschritten [13 S. 44]. Zur Verdeutlichung der beschriebenen Netzparametern wird die Abbildung 3 um ein weiteres vereinfacht und beschreibt grundlegend das einphasige Ersatzschaltbild einer elektrischen Versorgungsleitung mit Längsimpedanz (siehe Abbildung 4).

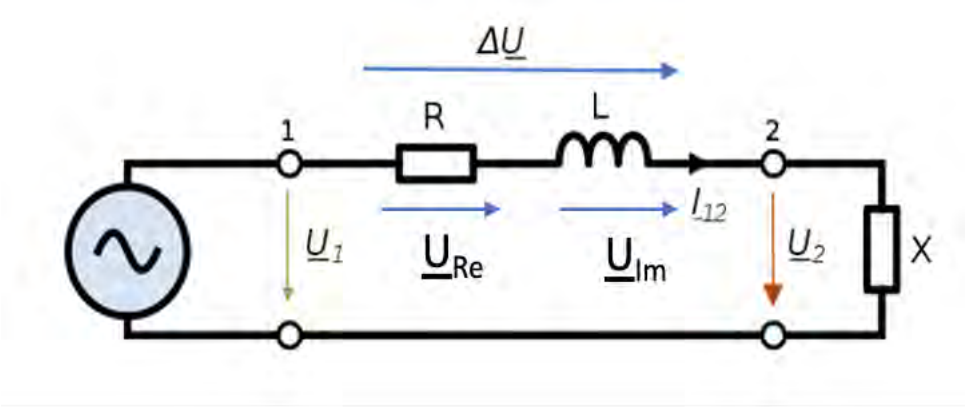

Abbildung 4: Vereinfachte Darstellung der Leitungsimpedanz - In Anlehnung an [30]

Dabei beschreibt die komplexe Längsimpedanz $\underline{Z}$, welche aus einer ohmsch-induktiven Last besteht, die zusammengefassten Knotenpunkte zwischen eins bis drei. Zur Vereinfachung der Abbildung 4 wurde der Knotenpunkt K_n der Reihe als der letzte Knotenpunkt Vier des Netzstrahls definiert. Die Versorgungsleitung wird, wie in Abbildung 3 von einem Transformator am Leitungsanfang mit der komplexen Spannung $\underline{U}_1$ gespeist. Knoten Vier beschreibt die letzte Last im Netzstrang. Am Anfang des Klemmenpunkts Eins fällt die komplexe Versorgungsspannung $\underline{U}_1$ ab. Wohingegen am Ende der Versorgungsleitung, am Klemmenpunkt Zwei die komplexe Lastspannung $\underline{U}_2$ abfällt, die sich nach Belastungsart der Impedanz X variieren lässt. Durch die Impedanz $\underline{Z}$, die sich zwischen beiden Messpunkten befindet, fällt die komplexe Spannung $\Delta\underline{U}$ ab. Die komplexe Spannung $\Delta\underline{U}$ bildet die Differenz zwischen der Eingangsspannung $\underline{U}_1$ und Lastspannung $\underline{U}_2$. Aus der Abbildung 4 lässt sich nach der Maschen-Regel ein Zeigerdiagramm ableiten, woraus die Phasenverschiebung des einphasigen Ersatzschaltbildes abgeleitet wird. Dabei eilt die Lastspannung $\underline{U}_2$ der Eingangsspannung $\underline{U}_1$ nach. In der Abbildung 5 wird der beschriebene Prozess des hergeleiteten Zeigerdiagramms verdeutlicht.

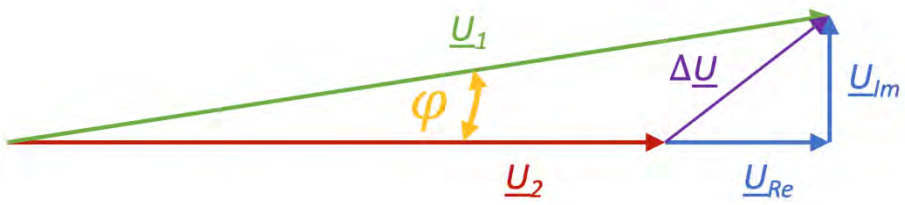

Abbildung 5: Zeigerdiagramm bei einer ohmsch-induktiven Belastung - In Anlehnung an [29]

Die Phasenverschiebung ergibt sich aus dem Spannungsabfall $\Delta\underline{U}$ an der Impedanz und setzt sich aus dem Parameter ΔT für die Zeitverschiebung (ms) sowie der Grundfrequenz

f (Hz) des gemessenen Signals zusammen. Um die Phasenverschiebung zwischen den beiden Signalnulldurchgängen in Gradmaß zu bestimmen, wird die Grundformel für den Phasenwinkel, welche in Bogenmaß angegeben wird

$$\varphi_{\mathrm{Rad}} = \omega \cdot \Delta t \tag{1}$$

mittels der folgenden Kreisfrequenz-Formel umgerechnet.

$$\omega = 2\,\pi\,f = 2\,\pi\,\frac{1}{T} \tag{2}$$

$$\varphi_{\mathrm{Rad}} = 2\,\pi \cdot \frac{\Delta t}{T} = 2\,\pi\,f \tag{3}$$

Dabei entspricht ein Vollwinkel mit 2π gleich 360° Grad. Daher gilt:

$$2\,\pi = 360° \tag{4}$$

Für die Phasenverschiebung in Gradmaß ergibt sich somit folgende Formel:

$$\varphi_{\mathrm{Deg}} = 360° \cdot \Delta T \cdot f \tag{5}$$

Um ein Aussagen über den Zustand des Niederspannungsnetzes machen zu können, wird mittels der ermittelten Phasenverschiebung (Gl. 5) die komplexe Scheinleistung des Netzstranges zwischen zwei Knotenpunkten berechnet.

$$\underline{S^*} = \underline{U_i^*} \cdot \underline{I} = P - jQ = \underline{U_i^*} \cdot \frac{(\underline{U_i} - \underline{U_j})}{\underline{Z}} \tag{6}$$

Dementsprechend lässt sich mit Hilfe der komplexen Spannung $\underline{U_i}$ und $\underline{U_j}$ (Polarkoordinatenform)

$$\underline{U_i} = \widehat{u_i} \cdot e^{-j\omega t} \text{ mit } |\underline{U_i}| = \frac{\widehat{u_i}}{\sqrt{2}} \tag{7}$$

$$\underline{U_j} = \hat{u_j} \cdot e^{-j\omega t} \text{ mit } |\underline{U_j}| = \frac{\hat{u_j}}{\sqrt{2}} \quad (8)$$

die resultierende komplexe Scheinleistung nach Gleichung 6 bestimmen. Dafür wird vorausgesetzt, dass der komplexe Parameter $\underline{Z}$, welcher für die Leitungsimpedanz steht, seitens des Netzanbieter vorgegeben wird. So können durch den Einsatz der Messgeräte den Phasenwinkel an den Nulldurchgänger, den Betrag der elektrischen Spannung ermittelt und weitere Netzinformation errechnet werden.

3.2. WAMS-Lösungsansatz

Bevor die Software für das WAM-System entwickelt werden kann, soll eine passende PMU-Schaltung ausgewählt werden, um die gestellten Anforderungen in Kapitel 1.1 am präzisesten zu erfüllen. Das zu entwickelnde WAM-System besteht aus einer Hardware und einer Softwarekomponente. Für die Abschlussarbeit werden zwei unterschiedliche PMU-Schaltungen gegenübergestellt und auf technische Vor- und Nachteile ausgewertet.

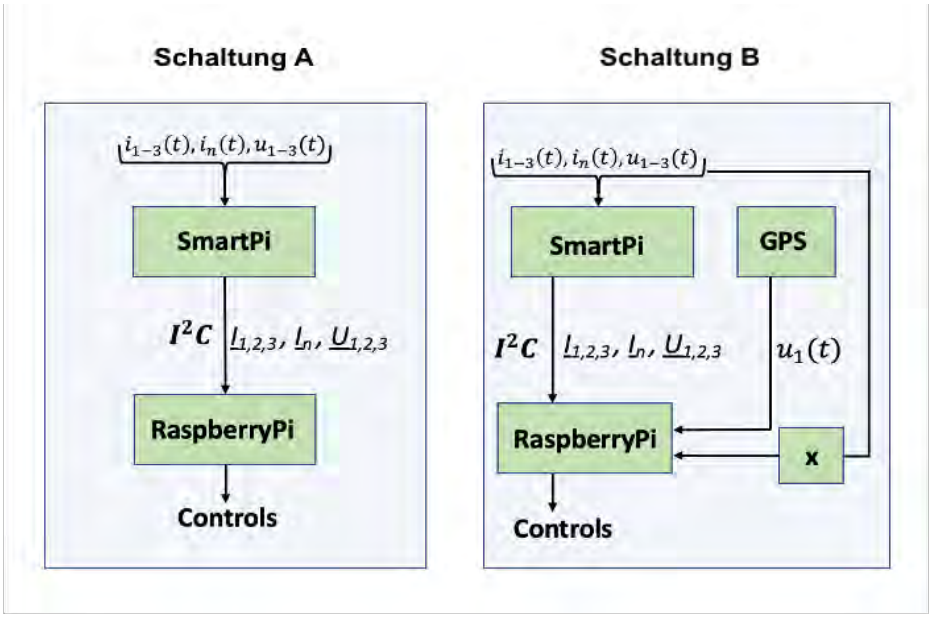

Abbildung 6: Unterschiedliche PMU-Schaltungen für das zu entwickelnde WAMS Messsystem [35]

Die Schaltung A besteht aus zwei Hardwarekomponenten, welche der Raspberry Pi und Smart Pi sind. Mit Hilfe des Smart Pis werden alle erforderlichen Spannungen und Ströme aufgezeichnet, welche über die serielle Kommunikationsschnittstelle dem I^2C-Bus auf den Raspberry Pi übertragen werden. Auf dem Raspberry Pi werden die Daten ausgewertet. Diese Schaltung ist aufgrund der kleinen Anzahl an Hardwarekomponenten die kostengünstigere und kompaktere Variante. Jedoch ist der größte technische Nachteil dieser PMU-Messschaltung, dass der Spannungswinkel nicht gemessen werden kann. Somit lassen sich keine Phasenverschiebungen innerhalb eines großflächigen

Messstandortes bestimmen. Die Umsetzung eines solchen Messgerätes liegt bei ca. 160 €.

Die B Schaltung ist eine Weiterentwicklung der ersten Schaltung A, welches ein funktionsfähiges PMU-Messgerät darstellt. Um das Problem mit der Aufnahme der Messungen innerhalb eines großflächigen Messstandortes zu beheben, wird der Raspberry Pi mit einem zeitsynchronisierten GPS-Modul ausgerüstet. Dadurch werden die am Nulldurchgang erzeugten Rechteckspannungen mit Zeitstempel softwaretechnisch markiert und ausgewertet. Für die Ermittlung der Phasenverschiebung werden zwei identisch entwickelte Messplatinen verwendet, die im Nulldurchgang einer $50\ \mathrm{Hz}$ Sinusschwingung ein Rechteckimpuls transformieren [28]. Die Umsetzung eines solchen Messgerätes liegt bei ca. 200 €.

Zusammenfassend lässt sich sagen, dass Schaltung A im Vergleich zu Schaltung B eine geringe Aufbaukomplexität aufweist und preislich deutlich günstiger ist. Jedoch kann das Versorgungsnetz mit dem Spannungswinkel nicht untersucht werden, weshalb Schaltung A für die Abschlussarbeit unbrauchbar ist. Die teurere Variante *„Schaltung B“* bietet hingegen den größten Mehrwert für die Netzwerkuntersuchung. Für die Entwicklung eines WAM-Systems werden zwei PMU-Messgeräte nach Schaltung B konzipiert.

3.3 Entwicklungsplatine – Raspberry Pi

Für die Entwicklung der Softwareprogramme wurde dafür der Raspberry Pi 3 B+ und 4 B ausgewählt. Die Softwareprogramme zur Ermittlung des Spannungswinkels werden vom Raspberry Pi gestartet und ausgewertet. In Abbildung 7 ist die verwendete Entwicklungsplatine, der Raspberry Pi abgebildet.

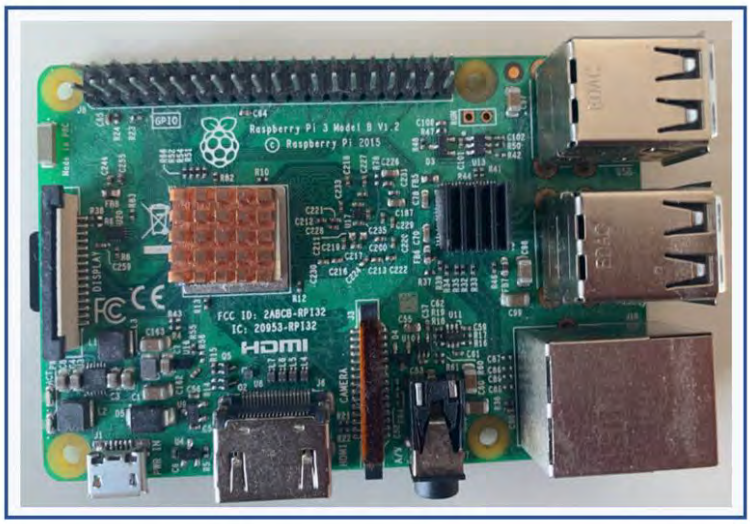

Abbildung 7: Entwicklungsplatine – Raspberry Pi 3 Model B+ [35]

Der verwendet Raspberry Pi 3 B+ besitzt ein ARM CORTEX-A53 Quad-Core Prozessor, welcher mit einer 4x1,2 GHz Taktung ausgerüstet ist. Der RaspberryPi 4 B hingegen ist mit einem ARM-Cortex A72 ausgerüstet, der mit 4x1,5 GHz taktet [14], [15]. Für die

Abschlussarbeit wurde der Raspberry Pi gewählt, da dieser eine nahe liegende hard- und softwaretechnische Übereinstimmung mit dem Mikrocontroller von Communication Gateway der Firma Devolo hat. Die einzigen Unterschiede zwischen den beiden Raspberry Pi Modellen liegt in der CPU-Taktung und den integrierten Anschlüssen. Der Raspberry Pi bietet eine Vielzahl von unterschiedlichen Anwendungsmöglichkeiten an. Darunter ist es möglich in verschiedenen Programmiersprachen, wie Python, C oder Java seinen Programmcode zu schreiben und kompilieren zu lassen. Auf der Platine der Raspberry Pis stehen 40 GPIO-Pins zu Verfügung. Einige der eingebauten Pins verfügen zusätzlich über speziell erweiterte Funktionen, wie der I^2C, SPI oder UART-Bus. Diese dienen als Kommunikationsschnittstelle zwischen zwei Peripheriegeräten. An einem GPIO-Pin lassen sich nur digitale Ein- und Ausgänge auslesen, weshalb der Raspberry Pi für das schnelle Auslesen von Rechteckflanken sehr geeignet ist. Nachdem ersten Start des Raspberry Pis werden alle GPIO-Schnittstellen standartmäßig von der Grundeinstellung deaktiviert und sollten über das Raspberry Pi-Konfigurationsverwaltungstool „*sudo raspi-config*“ für ihre Verwendung aktiviert werden [40]. Für die Kommunikation und Synchronisierung zwischen dem Raspberry Pi und dem GPS-Zeiterfassungsmodul wurde die UART „*Universelle Asynchrone-Empfänger-Überträger*“ als serielle Schnittstelle ausgewählt, welches in dem Kapitel 3.3.2 weiter vertieft wird. Zusätzlich werden noch zwei USB „*Universelle Serielle Bus*“ Schnittstellen zur Programmierung und Anwendung der Raspberry Pis angeschlossen.

Da das Betriebssystem *„Raspbian“* von der SD-Speicherkarte des Raspberry Pis gestartet wird, werden alle ermittelten Messdaten wie der Zeitstempel, die Frequenz und die PPS-Periode vom gemessenen Rechtecksignal auf der integrierten SD-Speicherkarte abgespeichert. Ein weiterer Vorteil des Raspberry Pis ist die eingebaute Schnittstelle zum WLAN-Netzwerk. Falls kein WLAN-Modul zu Verfügung steht, kann alternativ durch ein Netzwerkkabel dem Raspberry Pi der Internetzugriff ermöglicht werden. Mittels der integrierten Netzwerkschnittstelle wird eine Kommunikationszentrale zwischen den beiden Raspberry Pis ermöglicht. Die genaue Konfiguration der Kommunikationszentrale und der Messdatenaustausch zwischen den beiden verwendeten Raspberry Pis wird in den Kapiteln 3.6 & 4.4.3 der Abschlussarbeit erläutert. Um eine problemlose Netzwerkverbindung und einen Datenaustausch zwischen beiden Raspberry Pis herzustellen, muss für eine stabile Stromversorgung gesorgt werden [16]. Bei einer zu geringen Betriebsspannung von unter 5V reduziert der Raspberry Pi den Stromverbrauch. Dabei werden bestimmte Komponente, wie der Netzwerkanschluss deaktiviert [16]. Für die Kommunikation und Erstellung des Zeitstempels zwischen dem Raspberry Pi und GNSS-5-Click werden Pin12 (GPIO18), Pin16 (GPIO23) und Pin10 verwendet. Pin10 ist die UART-Schnittstelle auf dem Raspberry Pi und wird als Empfangsstelle der GNSS-Daten benötigt. Mit Pin12 und Pin16 werden die Rechteckflanken ausgelesen. Pin12 wird für das Auslesen der fallenden Flanken des zu messenden Rechtecksignals verwendet, wohingegen Pin16 für die steigenden PPS-Flanken (Pulse-Per-Second) zuständig ist. Das Zeiterfassungsmodul wird über Pin1 (3,3 V) versorgt und mit Pin6 (GND) des Raspberry Pi geerdet. Für die Programmierung (flashen) der Raspberry Pis ist es nicht

notwendig, einen Programmieradapter (UART-to-USB-Adapter) als Kommunikationsschnittstelle zwischen dem Computer und dem Raspberry Pi zu verwenden, da die geschriebenen Programme direkt von dem Betriebssystem „*Raspbian*“ gestartet werden.

3.4 Zeiterfassungsmodul – GNSS-5-Click

Das zeitsynchronisierte GPS-Modul, wie in der Abbildung 8 dargestellt, wird in der Abschlussarbeit für die Ermittlung der Koordinaten und für die Mikrosekunden genaue Zeitmessung benötigt. Das GPS-Modul wurde von der Firma „*MikroElektronik“* entwickelt und besitzt einen eingebauten Prozessor der Firma „*u-blox*“ [17]. Jedes entwickelte PMU-Messgerät wird mit einem Zeiterfassungsmodul synchronisiert, welches als zeitlicher Referenzpunkt dient. Als Alternativansatz ist die Installation einer Verkabelung für die Kommunikation zwischen beiden Messstandorten möglich, welches jedoch Aufgrund des enormen finanziellen Aspektes und des reinen zeitlichen Aufwandes zu umfangreich wäre, um es für das Progressus-Projekt umsetzen zu lassen.

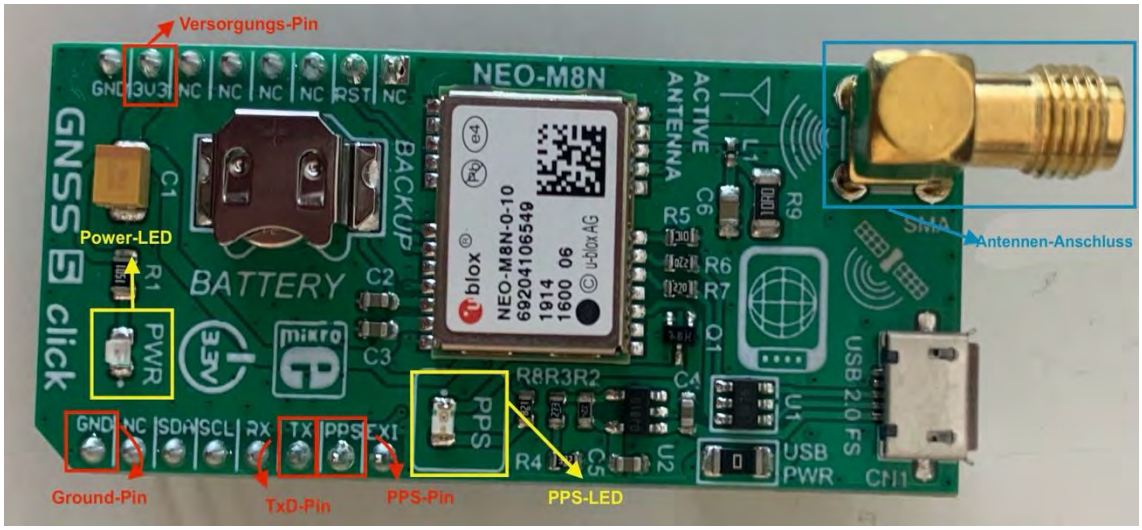

Abbildung 8: Zeitsynchronisiertes GPS Modul – GNSS-5-Click [35]

Das Zeiterfassungsmodul, der „*GNSS-5-Click*“ ist mit einem „*NEO-M8N*“ Empfänger Chip ausgerüstet, welcher die Koordinatenpakete des aktuellen Standortes mit dem globalen Satellitennavigationssystem empfängt. Das globale Satellitennavigationssystem oder auch als GNSS (Global Navigation Satelliten Systems) bekannt, lässt sich in folgende Systeme unterteilen: NAVSTAR GPS (Global Positioning System, USA), Galileo (Europa), GLONASS (Globales Satellitennavigationssystem, Russland) und BeiDou (China) [8 S. 8]. Die Satelliten senden sekündlich mehrere Datenpakete, welche vom GPS-Modul empfangen und an den Raspberry Pi weitergeleitet werden. Unter anderem werden vom GNSS-5-Click Datenpakete empfangen wie z.B. GPGGA, GPGLL, GPGSA, GPGSV, GPRMC, GPVTG und GPZDA. Aus den empfangenen Koordinaten werden vor allem die „*Coordinated Universal Time*“ (UTC) Uhrzeit im Format hhmmss.000 und das aktuelle Datum in ddmmyy benötigt, welche in dem Datensatz „*GPRMC*“ (Recommended Minimum Specific GPS/Transit Data) codiert sind [18].

3.4.1 Zeitermittlung durch Pulse-Per-Second – PPS-Signal

Neben den zu empfangenen Koordinaten, wie das Datum und die lokale Uhrzeit ist der GNSS-5-Click zusätzlich mit einer integrierten Pulse-Per-Second (PPS) Funktion ausgestattet, die als Startreferenzsignal für die Mikrosekunden genaue Zeitmessung dienen soll. Das PPS-Taktsignal ist ein positiver Rechteckimpuls, welcher sekündlich $100\ \mathrm{ms}$ andauert. Mit Hilfe des PPS-Signals ist es möglich, den Beginn jeder neuen eintreffenden Sekunde zu markieren. Da das PPS-Signal des GNSS-5-Clicks mit einer Genauigkeit unter $< 30\ \mathrm{ns}$ jede neu eintreffende Sekunde empfängt, wurde für die Abschlussarbeit die steigende Rechteckflanke des PPS-Signals als Startreferenzpunkt gewählt [19]. Zum Vergleich werden von Industrien GPS-Module verwendet, die mit einer PPS-Zeitgenauigkeit von $\sim 1\ \mu\mathrm{s}$ arbeiten [20]. Damit eignet sich der ausgewählte GNSS-5-Click mit seiner geringen PPS-Zeitverzögerung besonders gut für präzise Messwertaufnahmen. Mit dem PPS-Signal wird beiden Raspberry Pis signalisiert, wann der genaue Zeitpunkt der Messung anfängt. Die Darstellungen in Abbildung 9 zeigen zum einen, den genauen Prozessverlauf der Nulldurchgangstransformierung eines Sinussignals in einen positiven Rechteckimpuls. Zum anderen wird zusätzlich noch die Zeitmessung der fallenden Flanken mittels des PPS-Signals veranschaulicht. Wird der Nulldurchgang t_0 des grünen Referenzsinussignals in der Darstellung 1 erreicht, wird eine Rechteckflanke mit 3,3 V transformiert und bis zum nächsten Nulldurchgang t_2 des grünen Sinussignals konstant auf 3,3 V verlaufen. Zur selben Zeit wird das zeitverschobene gelbe Sinussignal zwischen den Nulldurchgängen t_1 und t_3 transformiert (siehe Darstellung 3). In der zweiten und dritten Darstellung sind die beiden am Sinusnulldurchgang transformierten Rechtecksignale abgebildet. In der letzten Darstellung 5 der Abbildung 9 wird der Periodenverlauf eines PPS-Signals dargestellt und an welchem genauen Augenblick die Zeitmessung angefangen wird. Der Parameter Δt_1 in der Darstellung 4 zeigt die gemessene Zeit zwischen der ausgelösten steigenden Flanke des PPS-Signals und der zu markierenden fallenden Flanke des grünen Referenzrechtecksignals. Währenddessen markiert der Parameter Δt_2 die Zeit zwischen der ausgelösten steigenden Flanke des PPS-Signals und der zu markierenden fallenden Flanke des verschobenen gelben Rechtecksignals. Um den genauen Zeitpunkt zu erhalten an dem die fallenden Flanken der Rechtecksignale ausgelöst werden, wird die vom GPS-Modul empfangene UTC-Uhrzeit mit den gemessenen Zeiten Δt_1 und Δt_2 zusammengerechnet und jeweils ein Zeitstempel für beide fallenden Flanken erstellt. Um die Zeitverschiebung zwischen den beiden Rechtecksignalen des Messsystems zu berechnen, wird aus den beiden ermittelten Zeitstempel $\Delta t_{2,Zst}$ und $\Delta t_{1,Zst}$ die Differenz (Gl. 9) gebildet.

$$\Delta \mathrm{T}_{2-1} = |\Delta \mathrm{t}_{2,\mathrm{Zst}} - \Delta \mathrm{t}_{1,\mathrm{Zst}}| \tag{9}$$

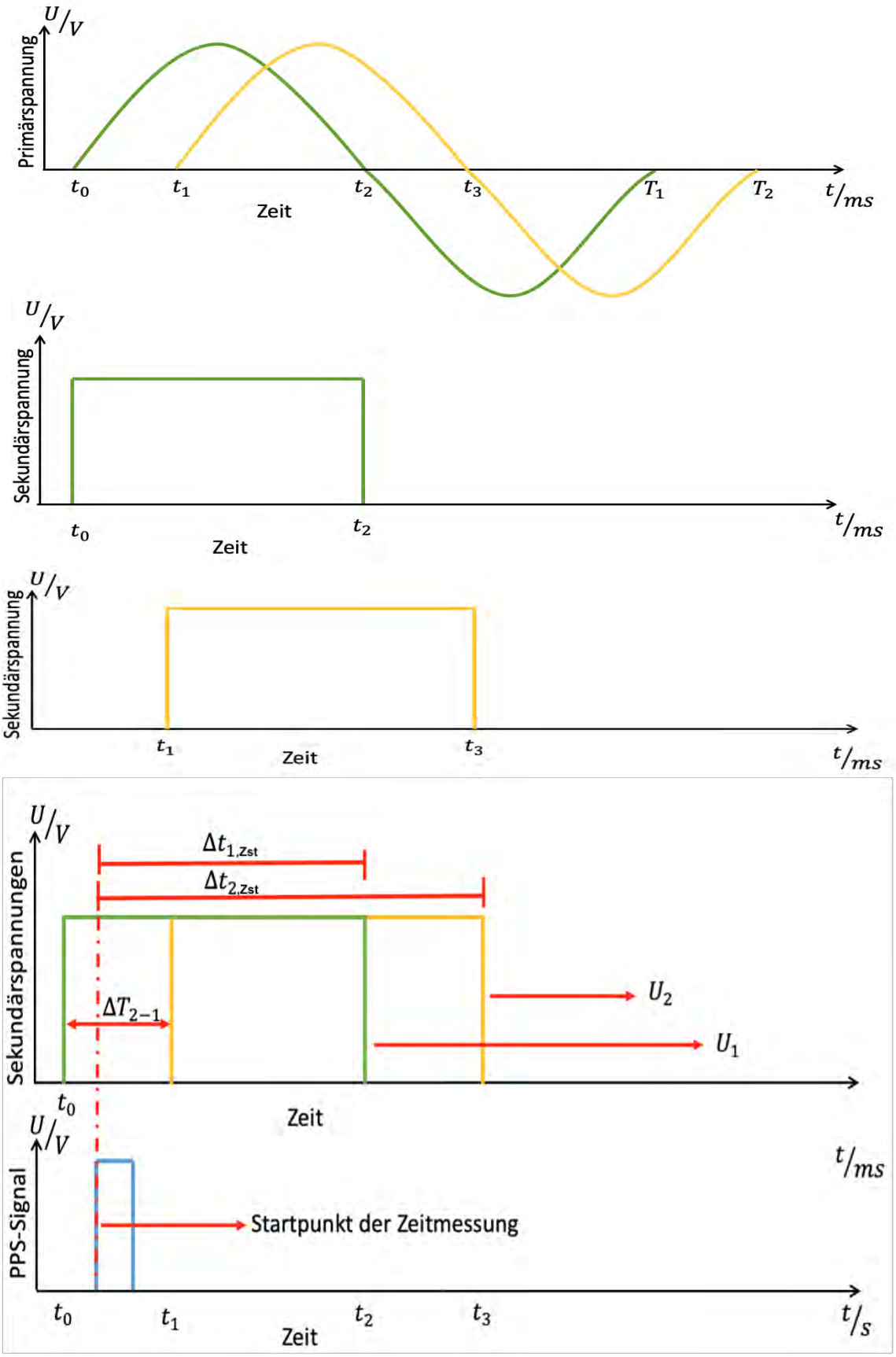

Abbildung 9: Startpunkt der Zeitmessung durch das PPS-Signal, Rechtecktransformierung am Sinusnull-durchgang [35]

3.4.2 UART – Serielle Kommunikationsschnittstelle

Als Datenübertragung zwischen dem GNSS-5-Click und dem Raspberry Pi wird die serielle UART-Kommunikationsschnittstelle gewählt sowie für beide Systeme passend konfiguriert. Die UART-Schnittstelle wird zur Kommunikation beider Prozessoren und zum Austausch von Dateninformationen zwischen Computer und Mikrocontroller verwendet. Die Datenübertragung basiert auf dem RS-232, welcher den Ablauf der Datenübertragung definiert. Für die Abschlussarbeit wurde die UART-Schnittstelle ausgewählt aufgrund der unterschiedlichen Betriebsmöglichkeiten und die Übertragungsart des Kommunikationsprotokolls. Die Datenübertragung zwischen beiden Peripheriegeräten kann auf drei unterschiedliche Betriebsarten konfiguriert werden.

Simplex	Datenübertragung jeweils in eine Richtung
Halbduplex	Datenübertragung in beide Richtungen, jedoch nicht zur selben Zeit
Vollduplex	Datenübertragung in beide Richtungen zur selben Zeit

Tabelle 3: Betriebskonfigurationsmöglichkeiten der UART-Schnittstelle [35]

Für die Abschlussarbeit wird die *Simplex* UART-Kommunikation als Betriebsart ausgewählt, da nur Daten in einer Richtung übertragen werden sollen. Für die Simplex Konfiguration wird vor der Inbetriebnahme ein Datenempfänger und ein Datensender vordefiniert. Für die Abschlussarbeit wird der GNSS-5-Click als Datensender gewählt, wohingegen der Raspberry Pi Daten empfangen soll, die vom GNSS-5-Click gesendet werden. Für eine fehlerfrei Kommunikationsherstellung zwischen beiden Hardwarekomponenten werden jeweils zwei unterschiedliche Schnittstellen benötigt, welche die Pins TxD und RxD sind. Der TxD-Pin (Transmit Data) fungiert als OUTPUT-Pin, welcher Datensätze an den Kommunikationspartner sendet. Der RxD-Pin (Receive Data) hingegen ist für das Empfangen von Datensätzen vom Kommunikationspartner zuständig und wird als INPUT-Pin definiert [21]. Die genaue serielle UART-Schnittstellenverbindung zwischen dem Datenempfänger (Raspberry Pi) und Datensender (GNSS-5-Click) wird in der Abbildung 10 veranschaulicht.

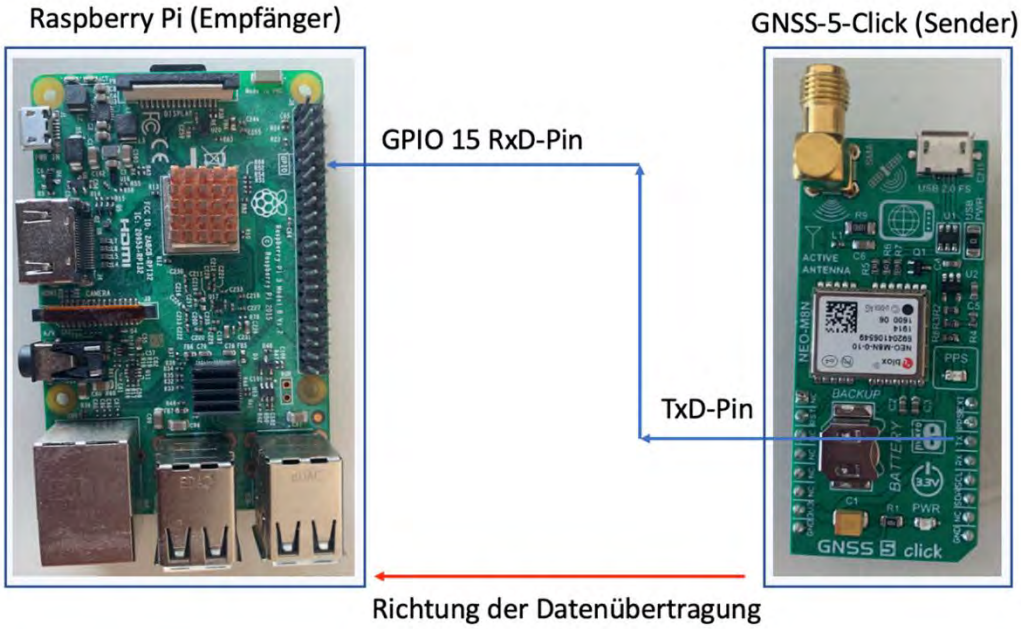

Abbildung 10: UART-Schnittstellenverbindung zwischen Raspberry Pi und GNSS-5-Click [35]

Sobald die serielle Verbindung zwischen beiden Kommunikationsgeräten synchronisiert und hergestellt ist, werden alle vom GNSS-5-Click empfangenen Koordinaten vom TxD-Pin zum RxD-GPIO-Pin des Raspberry Pis übertragen. Ein weiterer Vorteil der UART-Kommunikationsschnittstelle ist, dass eine Datenübertragung ohne Taktsignal funktioniert. Dadurch das die UART-Schnittstelle ohne Taktsignal Bits empfängt bzw. sendet, wird bei beiden Kommunikationspartnern dieselbe Baudrate Konfiguration eingestellt [21]. Wenn die Differenz der Baudrateeinstellung nicht dieselbe beträgt, ist das Timing der übertagenden Bits fehlerhaft. Die Übertragung der Datenpakete zwischen dem Raspberry Pi und GNSS-5-Click ist mit einer Übertragungsgeschwindigkeit (Baudrate) von 9600 $^{Bit}/_{s}$ vorkonfiguriert und kann besonders beim GNSS-5-Click nicht verändert werden. Außerdem wird jedes übertragende Byte mit einem Start- und Stoppbit gekennzeichnet. Des Weiteren wird kein Time-Pin für das Taktsignal benötigt, dass den Schaltungsaufbau und die Kommunikationsverbindung zwischen dem Raspberry Pi und GNSS-5-Click deutlich unkomplizierter gestaltet. Der Raspberry Pi hat zwei integrierte UART-Schnittstellen, die angesprochen werden können, der PL011 und Mini-UART [22]. Der Mini-UART ist standartmäßig als serielle Schnittstelle eingestellt. Um eine stabilere, leistungsstärkere, effektive und saubere Kommunikationsübertragung zu ermöglichen, sollte der PL011-UART vor Beginn der seriellen Kommunikation als Standart-UART eingestellt und passend konfiguriert werden[19]. Die effektive Kommunikation hängt damit zusammen, da die Kernfrequenz, die der PL011-UART taktet bei $f_{core} = 250\ \mathrm{MHz}$ betrieben wird und unabhängig von der Kernfrequenz der GPU ist. Der Mini-UART ist hingegen abhängig von der GPU-Kernfrequenz. Sobald die GPU-Kernfrequenz geändert wird, ändert sich die UART-Frequenz und dadurch auch die Baudrate des Raspberry Pis [19], [22].

3.5 Komparatorschaltung

In der Abschlussarbeit wird die Phasenlage von Rechteckspannungen untersucht und ausgewertet. Um am Nulldurchgang der 230 V Netzspannung eine 3,3 V Rechteckspannung zu erzeugen, wird eine bereits für das Progressus-Projekt entwickelte Komparatorschaltung verwendet [28]. Ein Bild der Komparatorschaltung ist in Abbildung 11 vorhanden.

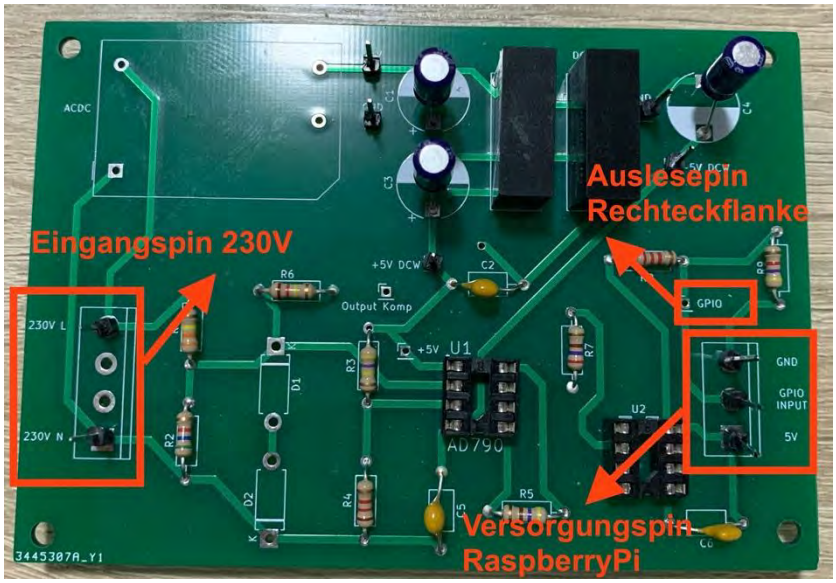

Abbildung 11: Prototyp der Komparatorschaltung [35]

Die Komparatorschaltung vergleicht einen analogen Spannungspegel V_{IN} mit einer voreingestellten Referenzspannung V_{REF} und erzeugt auf der Grundlage dieses Spannungsvergleichs ein Ausgangssignal V_{OUT} [23]. Dabei ist es wichtig, an welchem Eingang des Operationsverstärkers die Referenzspannungsquelle und Eingangsspannung angeschlossen wird. Da für die Abschlussarbeit nur die fallenden Flanken eines positiven Rechtecksignals von Bedeutung sind, wird für die Grundkonfiguration eine Referenzspannung am inventierten Eingang des Operationsverstärkers angeschlossen, während das Eingangssignal an den nicht inventierten Eingang angeschlossen wird [23]. Wenn das Eingangssignal V_{IN} größer als die definierte Referenzspannung V_{REF} ist, kommt es am Ausgang der Komparatorschaltung in Richtung der positiven Versorgungsschiene V_{CC} zu einer Sättigung, woraufhin eine steigende Flanke V_{OUT} erzeugt wird [23]. Für den Fall, dass die Versorgungsspannung V_{IN} kleiner als die Referenzspannung V_{REF} ist, wechselt der Ausgang des Operationsverstärkers den Zustand und sättigt sich an der negativen Versorgungsschiene auf 0 V [23]. Der genaue Prozess ist in Abbildung 9, Kapitel 3.4.1 dargestellt.

3.6 Samba-Server

Um eine Kommunikationsschnittstelle zwischen beiden PMU-Messgeräten herzustellen, sollten beide Messgeräte die Eigenschaft haben die Messdaten vom jeweiligen Messpartner öffnen und die Spannungswinkelberechnung starten zu können. Dafür wird ein netzwerkabhängiger Server eingerichtet, welcher auf beiden PMU-Messgeräten konfiguriert wird, um auf sämtliche Dateninhalte zugreifen zu können. Am besten eignet sich hierfür die Einrichtung eines Samba-Servers. Der Samba-Server hat dabei die Funktionalität eines „*Synchrophasor Communication System“* (SPCS). Über den Samba-Server lassen sich Daten oder Verzeichnisse einstellen und freigeben. Die Einrichtung des Samba-Servers muss auf beiden Raspberry Pis durchgeführt werden, da beide PMU-Messgeräte die Eigenschaft besitzen sollen, die Spannungswinkelberechnung starten zu können.

3.6.1 Konfiguration des Samba-Servers

Bevor der Samba-Server vom Raspberry Pi verwendet werden kann, werden passende Server-Pakete, wie „*samba“* und „*samba-common*“ installiert und die Samba-Grundkonfiguration durchgeführt (siehe Anhang 1, Abbildung 22). Außerdem wird vor der Einrichtung des Samba-Servers, das Betriebssystem und die Programm-Bibliotheken der verwendeten Raspberry Pis auf den aktuellen Stand gebracht, da es ansonsten zu Kommunikationsproblemen und zu einem fehlerhaften Datenaustausch zwischen beiden Geräten führt. Für die Samba-Grundkonfiguration existiert eine zentrale Vorkonfigurationsdatei auf dem Raspberry Pi [24]. In Abbildung 12 wird die zentral Vorkonfiguration des Raspberry Pi gezeigt, welche mit dem Funktionsbefehl *„/etc/samba/smb.conf“* aufgerufen werden kann (siehe Anhang 1, Abbildung 24).

```
[global]
workgroup = WORKGROUP
sercurity = user
encrypt passwords = no
client min protocol = SMB2
client max protocol = SMB3
```

Abbildung 12: Zentrale Vorkonfiguration des Raspberry Pis - In Anlehnung an [24]

Die Vorkonfiguration in Abbildung 12 signalisiert eine gültige Servereinstellung. Alle darin gemachten Einstellungen gelten für alle konfigurierten Freigaben [24]. Die global definierte Vorkonfiguration besteht aus den untenstehenden Angaben:

- **workgroup:**

  Der Parameter gibt die Bezeichnung der Netzwerkgruppe an. Damit wird ein Netzwerk-Rechner in die Gruppe mit strukturiert.

- **security:**

  Der Parameter steht für die Sicherheitsstufe, welcher bei dem Zugriff gelten soll. Der Parameter „*security*“ wird mit dem Wert „*user*“ initialisiert, um zu signalisieren, dass die User-Verwaltung des Servers verwendet wird. Damit wird die Samba-Freigabe auf dem Server aktiviert.

- **encrypt passwort:**

  Wird dem Parameter „*encrypt passwort“* eine Zeichenkette als Passwort zugewiesen, wird der Samba-Server verschlüsselt und bei einem Datenzugriff abgefragt. Für die Abschlussarbeit ist die Verschlüsselung von Daten nicht von höchster Priorität und wird mit *„no“* deaktiviert.

Die restlichen Parameter sind ebenfalls für die Abschlussarbeit nicht von Relevanz und werden nicht weiter erläutert.

3.6.2 Samba-Freigabe für das Verzeichnis „share“

Um ein Freigabeort zu definieren, auf denen die Messdaten vom *„Zeitstempel.c“* Programm abgespeichert werden sollen, wird ein Wurzelverzeichnis benötigt, dass alle Unterverzeichnisse beinhaltet, die vom Raspberry Pi freigegeben werden. Durch den Funktionsbefehl „*path = /home/pi/share“* wird das Verzeichnis „*pi“* angelegt. Im Verzeichnis pi wird zusätzlich ein Unterverzeichnis „*share*“ erstellt, indem alle Messdaten abgespeichert und öffentlich für beide PMU-Messgeräte zur Verfügung stehen. Damit beide PMU-Messgeräte die Möglichkeit haben die Messdaten im Verzeichnis *share* auszutauschen und aufzurufen, wird die Samba-Konfiguration, wie in Abbildung 13 angepasst (siehe Anhang 1, Abbildung 24).

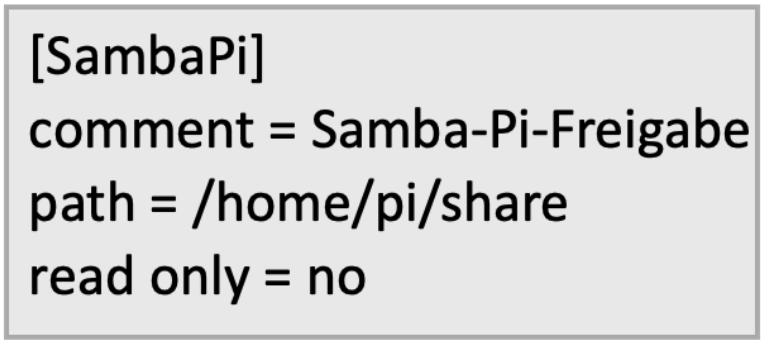

Abbildung 13: Einstellung der Freigabe des Samba-Servers auf dem Raspberry Pi - In Anlehnung an [24]

3.7 Entwicklung des Messsystems

Um die Kommunikationsschnittstelle zwischen Raspberry Pi, GNSS-5-Click und der Komparatorschaltung herzustellen, werden sieben unterschiedliche Verbindungsleitungen benötigt. Bei den verwendeten Verbindungskabeln handelt es sich um F2F-Connectoren Steckbrückenkabeln, welche für einen einfachen und zuverlässigen Kommunikationsaustausch zwischen den programmierten Pins auf den Breadboard der Platinen sorgen. Vier Leitungen stellen die Kommunikation zwischen dem Raspberry Pi und GNSS-5-Click her. Dabei wird eine Versorgungsleitung (grün), eine GND-Leitung (grau), eine UART-Leitung für den Kommunikationsaustausch von TxD- nach RxD-Pin (gelb) und eine Leitung für das Auslesen der steigenden Flanken der PPS-Signale (rot) benötigt. Über die Versorgungsspannung des Raspberry Pis wird der GNSS-5-Click über Pin1 mit 3,3 V versorgt und über Pin 6 geerdet. Über die serielle UART-Schnittstelle werden alle Datenpakete vom GNSS-5-Click (TxD) zum Raspberry Pi (RxD) gesendet. Das PPS-Signal wird mit dem Pin16 vom Raspberry Pi ausgelesen. Für die Verbindung zwischen dem Raspberry Pi und der Komparatorschaltung werden drei weitere F2F-Verbindungsleitungen benötigt. Eine Leitung für die 5 V Stromversorgung (grün) Pin2, ein GND (grau) Pin39 und eine Leitung (rot), welche die transformierten fallenden Rechteckflanken am GPIO-Pin12 des Raspberry Pis ausließt. Um eine transformierte 3,3 V Rechteckspannung zu erhalten, wird die Komparatorschaltung direkt an das elektrische Netz mit 230 V angeschlossen. Abbildung 14 zeigt den genauen Messaufbau jeder einzelnen Verbindung zwischen dem Raspberry Pi, GNSS-5-Click und der Komparatorschaltung.

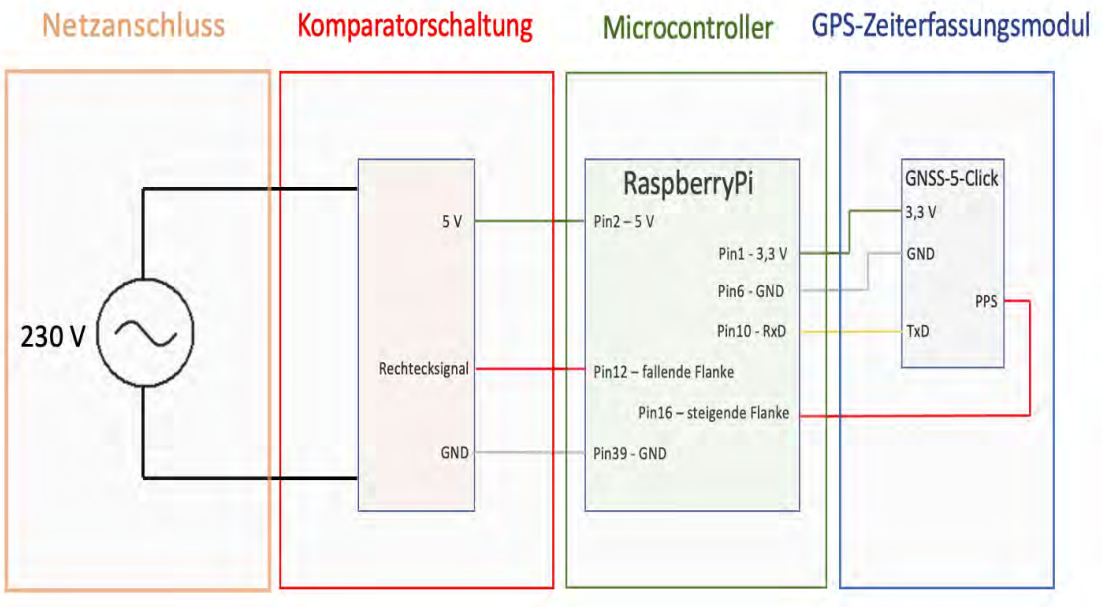

Abbildung 14: Schnittstellenschaltplan des entwickelten Spannungswinkelmessgeräts mit implementierter Komparatorschaltung [35]

Der GNSS-5-Click hält eine maximale Versorgungsspannung von 3,6 V aus. Um eine Zerstörung des GPS-Moduls zu verhindern, wird der Raspberry Pi an einem 5 V Pin angeschlossen, welcher von der Komparatorschaltung zur Verfügung gestellt wird. Dadurch kann der GNSS-5-Click mit der benötigten Betriebsspannung mit maximal 3,3

V (grün) versorgt werden. Als Alternative, kann die Versorgung des GNSS-5-Click über ein Micro-USB-Anschluss erfolgen. Für das GNSS-5-Click-Modul wird eine passende Signalleitung benötigt, die im Frequenzbereich von $1{,}575\ \mathrm{MHz}$ Nachrichten empfangen kann (siehe Anhang 2, Abbildung 26). Anderenfalls werden keine vollständigen Datenpakete vom NEO-M8N Prozessor empfangen. Zusätzlich kommt es beim PPS-Signal zum Kalibrierungsproblem, welches den Startpunkt der Zeitmessung zu ermitteln erschwert. Die Abbildung 27 im Anhang 2 zeigt den Prototypen eines funktionsfähigen entwickelten Spannungswinkelmessgeräts, welcher eine Zusammensetzung aus allen vorherigen Unterkapiteln 3.3 bis 3.6 beschrieben Hardwarebauteilen ist.

4. Softwareentwicklung

In diesem Kapitel wird auf die eigentliche Softwareentwicklung des Wide Area Monitoring System eingegangen. Dabei wird zu Beginn, um einen leichten Einstieg und Überblick in die Software zu erhalten, die Funktion und der Ablauf der Programme anhand von zwei unterschiedlichen Organigrammen beschrieben. Das PMU-Programm (Zeitstempel.c) wurde im Praxisprojekt detailliert und umfangreicht erklärt. Für die Abschlussarbeit wurde das Programm „*Zeitstempel.c*“ softwaretechnisch weiterentwickelt und verbessert. Dabei werden nur auf die wichtigsten Konfigurationseinstellungen sowie auf weiterentwickelte Funktionen zur Erstellung der Zeitstempel eingegangen. Das Programm zur Spannungswinkelmessung (*Spannungsphasenwinkel.c*), welches das PDC-System darstellt, wird in mehrere Abschnitte unterteilt, zeilenweise detailliert beschrieben und erklärt. Alle Funktionen, genauso wie die dazugehörigen Bibliotheken, Variablen und Voreinstellungen werden im Programmcode erläutert.

4.1 Organigramm – PMU-Software

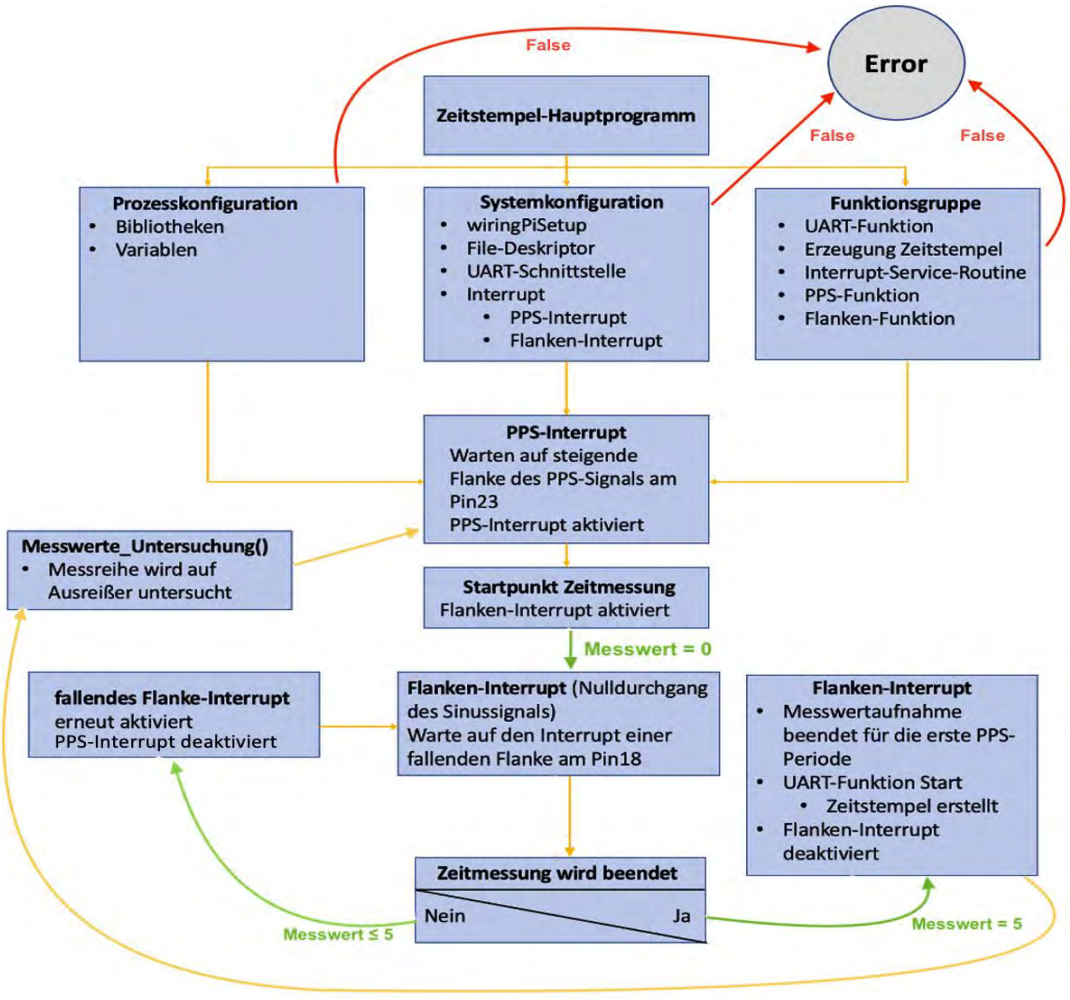

Abbildung 15: Organigramm – Ablauf und Programmstruktur der Zeitstempel.c Datei – In Anlehnung an [19]

Bevor der Programmcode (*Zeitstempel.c)* zur Flankenmessung und die daraus resultierenden Zeitstempel erläutert werden, soll mittels des Organigramms (siehe Abbildung 15) eine Orientierung des genauen Programmablaufes geschaffen werden. Um schrittweise folgen zu können, werden im Organigramm Pfeile farblich markiert. Wichtige Parameterwerte und Anweisungsblöcke werden durch Beschriftungen verdeutlicht. Das Organigramm beinhaltet grüne, gelbe und rote Pfeilsysteme und werden mit dem zugehörigen Parameterwert markiert. Wird der Wert des Parameters bei der Überprüfung in dem Anweisungsblock mit „*True*“ initialisiert, führt dies dazu, dass der darauffolgende Anweisungsblock ausgeführt wird. Andernfalls wird der Parameterwert im Anweisungsblock mit „*False*“ initialisiert und das Hauptprogramm zur Zeitmessung der fallenden Rechteckflanken mit der Anweisung *„Error“* sofort beendet. Der Zustand, der mit „*True“* abgeschlossen wird, markiert einen grünen Pfeil. Rote Pfeile signalisieren immer einen *„False“* Wert, welcher wiederrum für einen Fehler in der Funktion steht. Gelbe Pfeile zeigen hingegen immer auf die nächste aufzurufende Funktion. Bei den gelben Pfeilen werden keine Parameterwerte geprüft.

4.2 Software des PMU-Systems

Alle verwendeten Systemkonfigurationen, die zur Zeitmessung notwendig sind, werden zwischen Zeile 234 und 276 initialisiert. Wie im Organigramm (siehe Kapitel 4.1, Abbildung 15) abgebildet, werden alle deklarierten Funktionen, Konfigurationen, Messungen und die Initialisierung der Speichermethode in der *Zeitstempel.c* Datei programmiert und in der Hauptfunktion mit ausgeführt.

```
234  int main()
235  {
236      //main Variabelen-Definition
237      int serial_port = 0;              // eine ganzzahlige Variable, dient als Ruckgabewert für Initialieserung der UART-Schnittstelle
238      char Date ='D',Time ='T';         // character-Hilfs Variable: für die Ermittlung des Datenstrings des GPS-Signals
239      double t1 = 0;
240      int pps_sekunden = 0;
241      double mittelwert = 0;
242      double cnt_x =0;
243      int i = 0;
244
245
246      //Speichermodus festlegen
247      fp =fopen("PMU1-MW.csv", "w");
248      if(fp == NULL)
249      {
250          fprintf(fp, "Datei konnte nicht erstellt werden.");
251          return -1;
252      }
253      //Initialisierung der GPIO-Zugriffsbibliothek
254      if(wiringPiSetupGpio() == -1)
255      {
256          fprintf(fp, "Fehler bei der Initialisierung von WiringPiSetup");
257          return -1;
258      }
259      //Oeffnen der seriellen Schnittstelle fur den GPS-GNSS-5-Click Empfangers
260      if((serial_port = serialOpen("/dev/ttyAMA0", 9600)) < 0)
261      {
262          fprintf(fp, "Fehler bei der Initialisierung der UART-Schnittstelle mit dem GNSS ");
263          return -1;
264      }
265      //Output-Interrupt-Konfiguration fur die steigende PPS-Flanke
266      if (wiringPiISR (PPS, INT_EDGE_RISING, &PPS_Clock) < 0 )
267      {
268          fprintf(fp, "Fehler beim setup des Interrupts von low-to-high Pulse Pro Second Signal transitions");
269          return -1;
270      }
271      //Output-Interrupts-Konfiguration fur die fallende Rechtecksignal
272      if (wiringPiISR (EDGE_PIN, INT_EDGE_FALLING, &startCnt) < 0 )
273      {
274          fprintf(fp, "Fehler beim setup des Interrupts von low-to-high Voltage transitions");
275          return -1;
276      }
277
278      //Datensatz Aufrufen und uebertragende Daten werden gefiltet
279      UART_Rx_GPS(serial_port, Date);
280      fprintf(fp," Messwerte von 20 Schleifendurchgängen mit je 5 steigenden Spannungsmessungen am %s\n\n\n",UTC_Date);
```

Bevor die Zeitmessung anfängt, wird das System passend eingestellt und vorbereit. Dazu werden alle GPIO-Pins des Raspberry Pis durch die Initialisierung des *wiringPi-Setups* eingerichtet. Durch den Aufruf der Funktion in Zeile 254 wird das Standard GPIO-Pin Schema in das Nummerierungsschema von wiringPi codiert.

Zusätzlich wird die serielle UART-Kommunikationsschnittstelle (Zeile 260), das PPS-Interrupt (Zeile 266) und Flanken-Interrupt (Zeile 272) vor dem Start der eigentlichen Hauptschleife initialisiert. Dieses wurden im Bericht zum Praxisprojekt detailliert beschrieben. Im nächsten Programmschritt wird, bevor die Zeitmessung in der Hauptschleife gestartet wird die Speicherform in Zeile 247 festgelegt. Die Abspeicherung findet in Form einer CSV-Datei (Comma-Seperated Values) auf der SD-Karte statt. Die ausgewählte Speichermethode eignet sich besonders gut für Fehlerermittlung, da alle erfassten Messwerte strukturiert und tabellarisch in die Textdatei protokolliert werden. Die erstellten CSV-Messdaten werden daraufhin von dem zweiten Programm (*Spannungsphasenwinkel.c)* aufgerufen, untersucht und ausgewertet. Für die Ermittlung der Phasenverschiebung zwischen den beiden PMU-Messgeräten wird ein Referenzzeitpunkt, wie in Kapitel 3.4.1 beschrieben, benötigt. Das PPS-Signal (Pulse-Per-Second) wurde als ein solches Referenzzeitpunkt ausgewählt und passend in Zeile 272 definiert. Realisiert wird das ganze durch die Output-Interrupt-Service-Routine, das am GPIO-Pin23 eine steigende Flanke erkennt, den Messstartpunkt beider PMU-Messgeräte signalisiert und die Zeitmessung einleitet. Dadurch werden Abweichungen aufgrund der seriellen UART-Übertragung vermieden, was besonders wichtig für die Spannungswinkelmessung zwischen beiden Messpunkten ist. Das PPS-Signal wird am GNSS-5-Click durch eine sekündlich taktende rote LED angezeigt. Durch die Interrupt-Funktion werden die fallenden Rechteckflanken unter der wiringPi-Bibliothek mit einer $4{,}1\ \mathrm{MHz}$ Geschwindigkeit vom GPIO-Pin ausgelesen [25].

In Zeile 266 bis 276 werden alle Output-Interrupts im Konfigurationskopf passend initialisiert. In Zeile 266 wird das PPS-Interrupt aktiviert, dass nur bei einer eintreffenden steigenden Flanke des PPS-Signals auslöst. Zusätzlich wird noch in Zeile 272 zur fallenden Flankenerkennung ein weiteres Output-Interrupt (Flanken-Interrupt) konfiguriert und zusätzlich aktiviert. Damit es zu keinem Konflikt zwischen den beiden oben genannten Interrupts kommt, werden beide Interrupt-Zugrifffunktionen deutlich voneinander definiert. Zu diesem Zeitpunkt sind beide Output-Interrupts aktiviert und überprüfen die eingestellten GPIO-Pins auf logische Zustandsveränderung. Bevor die Hauptschleife mit der Zeitmessung und Erstellung des Zeitstempels anfängt, wird die *UART_Rx_GPS()* Funktion aufgerufen, um den Augenblick der Zeitmessung mit einem Datum zu markieren. Sobald das Datum aus dem GPRMC-Datensatz des GNSS-5-Click-Moduls über die serielle UART-Schnittstelle empfangen und gefiltert wurden ist, fängt die eigentliche Zeitmessung an.

4.2.1 Startpunkt der Zeitmessung

Nach der Initialisierung der Funktionen und Systemvorbereitung, wird die Hauptschleife in Zeile 283 so lange durchlaufen bis 20 Messreihen mit jeweils 5 Zeitstempel aufgenommen oder das Messgerät vom Stromnetz getrennt wurde. Der Code des Programmes *Zeitstempel.c* wird erst gestartet, wenn beide GNSS-5-Clicks, das PPS-Signal erfolgreich empfangen haben und die rote LED des GPS-Moduls den Startpunkt der Zeitmessung signalisiert (siehe Anhang 3, Abbildung 33).

```
282     //Startpunkt der Messwert aufnahme
283     while ( pps_sekunden != 20 ) //Für 20 Sekunden jeweils 5 Messungen pro Sekunde
284     {
285         //PPS-Interrupts ist aktiviert
286         if(pps_Cnt > 0)
287         {
288             //PPS-Interrupt ist deaktiviert
289
290             //Startpunkt der Zeitmessung
291             gettimeofday(&tim,(struct timezone*)0);
292             t1 = tim.tv_sec+(tim.tv_usec/1000000.0);
293
294             // fallende Flanke werden ermittlet
295             while ( i!=5 )
296             {
297                 //Flanken-Interrupts ist aktiviert
298                 if(EDGE_Cnt > 0)
299                 {
300                     //Flanken-Interrupt ist deaktiviert
301
302                     //Zeitpunkt der Zeitmessung wird beendet
303                     gettimeofday(&tim,(struct timezone*)0);
304                     valueX[i] = (tim.tv_sec+(tim.tv_usec/1000000.0)-t1);
305
306                     //Flanken-Interrupt wird aktiviert
307                     EDGE_Cnt = 0;
308                     //Aufnahme der nächsten Messwerte innerhlab der aktiven PPS-Periode
309                     i++;
310                 }
311             }
```

Der Programmabschnitt mit allen Funktionsaufrufen (Zeile 286 bis 310) ist das Herzstück der *Zeitstempel.c* Datei. In ihr werden alle definierten Interrupts aktiviert, deaktiviert und die Mikrosekunden aller auftretenden fallenden Flanken gemessen. Bevor die Zeitmessung startet, wird eine klare Prioritätengliederung der Interrupts festgelegt. Dabei hat das PPS-Interrupt die höchste Priorität gefolgt von dem Flanken-Interrupt. In Zeile 286 wird die Hauptschleife zur Zeitmessung gestartet. Wird das aktive geschaltete PPS-Interrupt durch eine steigende PPS-Flanke am GPIO-Pin23 ausgelöst, wird daraufhin die PPS-Interrupt-Funktion (PPS_CLOCK) aufgerufen und inkrementiert die für das PPS-Interrupt zuständige Variable *pps_Cnt.* Damit wird das PPS-Interrupt durch das Inkrementieren der Variable *pps_Cnt* deaktiviert und die darauffolgende Zeitmessung in Zeile 291 gestartet. Da die Priorität, der beiden Interrupts in der Systemkonfiguration klar definiert ist, kann das Flanken-Interrupt problemlos gestartet werden. Trifft eine fallende Flanke am GPIO-Pin18 ein, wird das aktive Flanken-Interrupt ausgelöst und der Anweisungsblock in der Zeile 295 gestartet. In diesem Anweisungsblock wird die Zeitmessung beendet und das Flanken-Interrupt deaktiviert. Die Anzahl der aufgenommenen PPS-Messreihen lässt sich mit der Variable *pps_sekunden* beliebig einstellen. Dazu wird jeder erfasste Messwert in das Array valueX abgelegt, welcher je nach Voreinstellung in der Prozesskonfiguration, die gewünschte Anzahl der PPS-Perioden aufnehmen kann.

4.2.1.1 Zeitpunkt der Messdatenspeicherung

Die Abspeicherung der Messdaten in das CSV-Dokument erfolgt ab der Zeile 323. Bevor die Messwerte auf die SD-Karte abgespeichert werden, findet eine Messwertüberprüfung der aktuell erfassten Messreihe statt, um eine höhere Spannungswinkelgenauigkeit zu erzielen. Durch den Funktionsaufruf *Flanken_Untersuchung()* in Zeile 320 werden alle identifizierten Ausreißer, die vom erwarteten Messwert abweichen aus dem Array *valueX* entfernt.

```
312             i=0;
313             //Messreihe wird abgebrochen
314             fprintf(fp, " Messung Nr.%i\n",pps_sekunden + 1);
315
316             // Zeit aus dem Datensatz wird gefiltet
317             UART_Rx_GPS(serial_port, Time);
318
319             //erfasste Messwerte werden analysiert
320             Flanken_Untersuchung();
321
322             //Messwerte der Messreihe wird abgespeichert
323             fprintf(fp, " Nr.1 %lf: Zeitstempel %lf \n Nr.2 %lf: Zeitstempel %lf Frequenz: %lf \n Nr.3 %lf: Zeitstempel %lf Frequenz: %lf \n",
324                     valueX[1], Zeitstempel2, valueX[2], Zeitstempel3, 1/(valueX[2]-valueX[1]), valueX[3], Zeitstempel4, 1/(valueX[3]-valueX[2]));
325             fprintf(fp, " Nr.4 %lf: Zeitstempel %lf Frequenz: %lf \n", valueX[4], Zeitstempel5, 1/(valueX[4]-valueX[3]));
326             fprintf(fp, " Nr.5 %lf: Zeitstempel %lf Frequenz: %lf \n", valueX[5], Zeitstempel6, 1/(valueX[5]-valueX[4]));
327
```

Damit die erfassten Messwerte übersichtlich und nachvollziehbar abgespeichert werden, wurde für die Abspeicherung der erstellten Zeitstempel die Funktion *fprintf()* gewählt. Dadurch werden alle erfassten Messwerte in dem CSV-Dokument *PMU1-MW.csv* abgesichert (siehe Anhang 3, Abbildung 35). Der fprintf()-Funktion werden die Parameter fp und FormatString übergeben. Der Parameter *fp* beschreibt den File Zeiger der CSV-Datei „*PMU1-MW.csv*“. FormatString ist ein Platzhalter für String-Funktionen und sichert alle Messwerte in dem CVS-Dokument ab. Die Bibliothek, welche den Zugriff auf die SD-Karte ermöglicht, speichert die Messwerte als den Datentyp *Char* ab. Die Vor- und Nachkommastellen der Zeitstempel werden durch einen Punkt voneinander getrennt und ebenfalls als Datentyp *Char* gespeichert. Die Aufnahme der berechneten Zeitstempel erfolgt durch die Variable *Zeitstempel* in Zeile 323. Für die Abspeicherung der unterschiedlichen Zeitstempel werden jeweils unterschiedliche Zeitstempel-Variablen verwendet. In der Funktion *UART_Rx_GPS()* wird mit Hilfe des Übergabeparameters „*Time*“ die aktuelle Uhrzeit aus dem empfangenen GPRMC-Datensatz gefiltert und mit der gemessenen Zeit, wie in Kapitel 4.2.1 beschrieben, addiert. Der genaue Funktionsablauf mit dem Empfangen der GPRMC-Datensätze, die Erstellung der Zeitstempel (hhmmss.ssssss) mittels der empfangen UTC (hhmmss.) und der gemessenen Zeit (.ssssss) zwischen der steigenden PPS-Flanke und der fallenden Rechteckflanke wurde im Praxisprojektbericht umfangreich beschrieben. Die Berechnung der Frequenzen der aufgenommen Rechtecksignale erfolgt durch die ermittelten Zwischenwerte. Die Zwischenwerte sind in dem *valueX*-Array abgespeichert. Dabei wird der aktuelle Messwert von dem vorherigen Messwert subtrahiert. Die Frequenz wird mit der folgenden Formel berechnet.

$$\Delta t = \text{Zwischenwert}_{i+1} - \text{Zwischenwert}_{i} \tag{10}$$

$$\text{Frequenz} = \frac{1}{\Delta t} \tag{11}$$

4.2.1.2 Wiederaufnahme einer Messreihe

Der Programmabschnitt zwischen Zeile 328 bis 333 dient zur erneuten Aufnahme einer weiteren Messreihe. Dafür wird das bereits deaktivierte PPS-Interrupt in Zeile 331 für eine erneute Messreihenaufnahme aktiviert, indem für die PPS-Interrupt zuständige Variable *pps_Cnt* mit Null initialisiert wird. Dadurch wird am GPIO-Pin23 ermöglicht, dass das PPS-Interrupt erneut bei einer steigenden PPS-Flanke ausgelöst werden kann. Um eine weitere Messreihe innerhalb des nächsten ausgelösten PPS-Signals signalisieren zu können, wird die Variable *pps_sekunden* in Zeile 333 mit einer Eins inkrementiert.

```
328                 Zeitstempel = 0;
329
330                 //PPS-Interrupts wird aktiviert
331                 pps_Cnt = 0;
332                 //Messreihe wird inkrementiert
333                 pps_sekunden++;
334
335             }
336
337         }
338         fclose(fp);
339         return 0;
340
```

4.2.2 Untersuchung der Messwerte

Die Funktion *Flanken_Untersuchung()* wird in der Endlosschleife (Zeile 163) des Hauptprogramms gestartet, nachdem die erste Messreihe mit fünf erfolgreichen Zeitstempel aufgenommen wurde.

```
163   void Flanken_Untersuchung()
164   {
165       //Überprüfung ob die Frequenz des ersten und zweiten Messwertes in diesem Messbereich liegt
166       if((1/valueX[1]-valueX[0]) < 49.9 || 1/(1/valueX[1]-valueX[0]) > 50.1)
167       {
168
169           //Falls False => erster Messwert ist ein Ausreißer und wird mit Null ueberschrieben
170           if(valueX[0] > 0.02)
171           {
172
173               valueX[0] = 0;
174
175           }
176           //Falls False => zweiter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
177           else if(valueX[1] < 0.02 || valueX[1] > 0.04)
178           {
179
180               valueX[1] = 0;
181
182           }
183       }
184
185       //Überprüfung ob die Frequenz des zweiten und dritten Messwertes in diesem Messbereich liegt
186       else if((1/valueX[2]-valueX[1]) < 49.9 || 1/(1/valueX[2]-valueX[1]) > 50.1)
187       {
188
189           //Falls False => zweiter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
190           if(valueX[1] < 0.02 || valueX[1] > 0.04)
191           {
192
193               valueX[1] = 0;
194
195           }
196           //Falls False => dritter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
197           else if(valueX[2] < 0.04 || valueX[2] > 0.06)
198           {
199
200               valueX[2] = 0;
201
202           }
203
204
205       }
```

Um eine höhere Messgenauigkeit des Spannungswinkels zu erzielen, werden alle Zeitstempel der aktuellen Messreihe auf Ausreißer überprüft. Dabei wird zunächst der Frequenzbereich zwischen dem Messwert *i* und *i+1* berechnet. Mit der ersten if-Verzweigung wird untersucht, ob die berechnete Frequenz in dem angegebenen Frequenzbereich zwischen $49{,}9\ \mathrm{Hz}$ und $50{,}1\ \mathrm{Hz}$ liegt. Ist die geprüfte Bedingung in Zeile 166 wahr, wird eine weitere Bedingung in einer zusätzlichen verschachtelten if-Verzweigung geprüft. In dieser Bedingung werden die Zeitstempel untersucht, ob die aufgezeichneten fallenden Flanken des Rechtecksignals in einem gültigen Messbereich liegen. Somit wird beispielsweise in Zeile 170 die erste aufgezeichnete fallende Flanke der ersten Messreihe kontrolliert, ob der Messwert größer als die transformierte Rechteckperiode mit $T = 20\ ms$ ist. Ist die Bedingung in Zeile 170 wahr, wird dieser gespeicherte Messwert als Ausreißer definiert und die identifizierte Arraystelle mit Null überschrieben (Zeile 173). Dieser Funktionsablauf wird so lange durchgeführt, bis alle Messwerte der Messreihe auf Ausreißer überprüft wurden. In dem unteren Programmabschnitt werden die restlichen Messwerte der Messreihe auf Ausreißer nach demselben Programmprinzip, wie im oberen Funktionsabschnitt kontrolliert.

```
207       //Überprüfung ob die Frequenz des dritten und vierten Messwertes in diesem Messbereich liegt
208       else if((1/valueX[3]-valueX[2]) < 49.9 || 1/(1/valueX[3]-valueX[2]) > 50.1)
209       {
210           //Falls False => dritter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
211           if(valueX[2] < 0.04 || valueX[2] > 0.06)
212           {
213
214               valueX[2] = 0;
215
216           }
217           //Falls False => vierter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
218           else if(valueX[3] < 0.06 || valueX[3] > 0.08)
219           {
220
221               valueX[3] = 0;
222
223           }
224
225
226       }
227
228       //Überprüfung ob die Frequenz des vierten und fuenften Messwertes in diesem Messbereich liegt
229       else if((1/valueX[4]-valueX[3]) < 49.9 || 1/(1/valueX[4]-valueX[3]) > 50.1)
230       {
231           //Falls False => vierter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
232           if(valueX[3] < 0.06 || valueX[3] > 0.08)
233           {
234
235               valueX[3] = 0;
236
237           }
238           //Falls False => fuenfter Messwert ist ein Ausreißer und wird mit Null ueberschrieben
239           else if(valueX[4] < 0.08 || valueX[4] > 0.1)
240           {
241
242               valueX[4] = 0;
243
244           }
245
246
247       }
248
249   }
```

4.3 Organigramm - PDC-Software

Um einen einfacheren Einblick in die Software zur Spannungswinkelberechnung zu erhalten, wird auch bei diesem Programm ein Organigramm erstellt. Damit wird der Programmablauf sowie die Kommunikation zwischen den einzelnen Funktionen veranschaulicht. Der Kopf des Organigramms ist die main-Funktion der Software, welche mit

„*Spannungswinkelmessung-Hauptprogramm*“ gekennzeichnet ist. In ihm werden alle wichtigen Konfigurationen wie die Bibliotheken oder Variablen passend für den weiteren Programmverlauf initialisiert. Zusätzlich werden alle Funktion in der main-Funktion definiert, welche für die Überprüfung der Zeitstempel und Spannungswinkelberechnung notwendig sind. Das Organigramm funktioniert nach demselben Prinzip, wie das Organigramm zur Zeitstempel.c Datei (siehe Kapitel 4.1), weshalb nicht weiter darauf eingegangen wird.

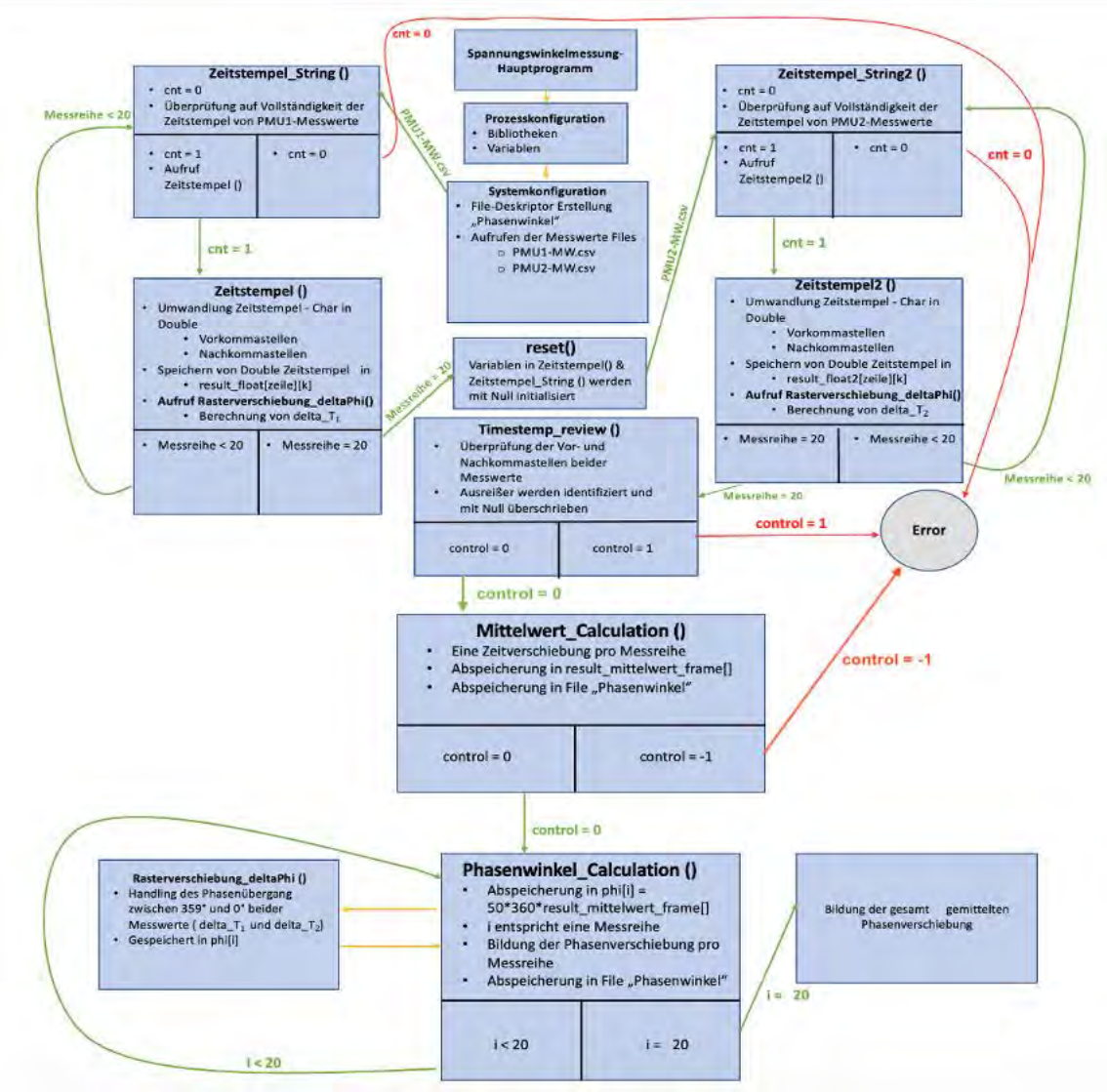

Abbildung 16: Organigramm - Ablauf und Struktur der Spannungsphasenwinkel.c Datei [35]

4.4 Software des PDC-Systems

In diesem Kapitel wird auf die Software zur Spannungswinkelberechnung des PDC-Systems eingegangen. Dabei wird das Programm in folgende Aufgabenbereiche eingeteilt: Systemkonfiguration (1), Vorbereitung, Überprüfung, Umwandlung der Zeitstempel (2) und die Berechnung der Zeitverschiebungen sowie die Phasenverschiebung (3). Das Programm wird gestartet, sobald beide PMU-Messgeräte ihre Messdatei über den

Samba-Server in den Ordner „*share*“ abgelegt haben (siehe Anhang 3, Abbildung 36). Dabei gehört die Messdatei „*PMU1-MW.csv*“ zu dem Referenzmessgerät und „*PMU2-MW.csv*“ zu dem zweiten PMU-Messgerät. Sind alle Vorbereitungen getroffen, wird das Programm „*Spannungsphasenwinkel.c*“ auf dem Raspberry Pi ausgeführt (siehe Anhang 3, Abbildung 39).

4.4.1 Einstellung der Systemkonfiguration

In dem Programmkopf werden alle Grundeinstellungen vorgenommen, die für die Überprüfung der Zeitstempel und Berechnung der Phasenverschiebung notwendig sind. Dabei wird besonders auf das Einbinden der Bibliotheken und die initialisierten Variablen eingegangen.

```
1   //Bibliotheken
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <math.h>
6
7   //Fur Schleifen, if-Verzweigungen oder Zwischenabspeicherung fur Arrays
8   int i=0, j=0, cnt=0, k=0, kk=0, MW_cnt=0, zeile=0, sum = 0, marker=0, PMU_check=1, sum2=1;
9   int zahl[120][12] = {0};
10  int zahl2[120][12] = {0};
11  double result_float[120][12] = {.0}, result_vorkomma=.0, result_nachkomma =.0;
12  double result_float2[120][12] = {.0}, result[120][12];
13
14  //Variablen fur die Rasterverschiebungsuntersuchung
15  double Raster_VS_PMU1[120] = {0}, Raster_VS_PMU2[120] = {0}, RV=0, delta_Tn[20] = {.0};
16  char zeichen[7451] = {0};
17
18  //Variablen fur beide PMU-Messdateien
19  FILE*csv_1;
20  FILE*csv_2;
21  FILE* Phasenwinkel;
22
23  //Variablen fur die Mittelwert- und Spannungswinkelberechnung
24  double  result_vorkomma_safeX[120]= {0}, result_vorkomma_safeX2[120]= {0};
25  double  result_nachkomma_safeX[120]= {0}, result_nachkomma_safeX2[120]= {0};
26  double result_mittelwert_frame[20] = {0}, phi[20] = {0};
```

Bei den verwendeten Bibliotheken in der Zeile 2 bis 5 handelt es sich dabei um implementierte Standardbibliotheken, die das Programmieren deutlich angenehmer gestalten. Ab der Zeile 8 bis zur Zeile 26 werden verschiedene globale Variableneinstellungen vorgenommen. In der Zeile 9 werden die Variablen als Zähler für diverse Schleifen sowie if-Verzweigungen deklariert. Die Variablen *i, k, zeile* und *j* haben die zusätzliche Funktion die Feldelemente der Arrays, wie *zeichen, zahl, zahl2, result_vorkomma_safe* und *result_float* mit Zwischenergebnissen zu füllen. Die Variablen *MW_cnt und cnt* werden stattdessen für die Feldelemente *result_vorkomma_safe2, result_nachkomma, result_float, result_float2, result* und *result_mittelwer* eingesetzt.

Die Variablen *zahl* und *zahl2* sind zweidimensionale Arrays, die für das Abspeichern der Zeitstempel innerhalb einer Messreihe zuständig sind. Die Variable z*ahl* wird für das Abspeichern der PMU1-MW.csv benötigt, wohingegen in *zahl2* die Messdaten von PMU2-MW.csv gespeichert werden. Da alle erstellten Zeitstempel aus Vor- und Nachkommastellen (Format: hhmmss.ssssss) bestehen, werden die Nachkommastellen von den Vorkommastellen getrennt und in unterschiedliche Variablen zwischengespeichert. Diesem Vorgang obliegt besonderer Gewichtung, da die Zeitstempel in dem CSV-

Dokument als Datentypen Char von der fprintf-Funktion abgespeichert werden. Die Vor- und Nachkommastellen werden von Char in Double umgewandelt, da ansonsten keine Phasenverschiebung berechnet werden kann. Die Vorkommastellen (hhmmss.) werden in *result_vorkomma_safeX* (Zeile 24) und die Nachkommastellen (.ssssss) in die Variable *result_nachkomma_safeX* (Zeile 25) abgelegt. Nachdem die Vor- und Nachkommastellen vollständig umgewandelt wurden, werden beide miteinander addiert und der daraus resultierende Zeitstempel in dem Array *result_float* in Zeile 11 gesichert. Die übersetzten Zeitstempel der PMU2-MW.csv Messdatei werden dagegen in das Arrays *result_float2* gespeichert. Beide PMU-Messwerte werden miteinander verrechnet und die daraus berechnete Zeitverschiebung zwischen den beiden gemessenen Punkten wird in das finale Array *result* (Zeile 12) gesichert. Die Variable *RV* in Zeile 15 ist für die Rasteruntersuchung der Spannungswinkeln zwischen 359° und 0° zuständig. In ihr werden alle Sekunden der Nachkommastelle einer Messreihe aufsummiert und abgespeichert. Die Variablen in derselben Zeile werden ebenfalls für die Ermittlung der Rasterverschiebung benötigt und in dem Kapitel 4.4.8 detaillierter erläutert. Für das Aufrufen beider PMU-Messdaten werden im Programm File Pointer benötigt, die auf die erstellte CSV-Messdatei zeigen. Mit File* csv_1 (Zeile 19) wird die CSV-Messdatei des Referenzmessgerätes und mit File* csv_2 (Zeile 20) des zweiten PMU-Messgerätes aufgerufen. In der Zeile 21 wird ein zusätzlicher File Pointer erstellt, der die ganzen Berechnungen in einer separat erstellten CVS-Datei protokolliert. Zum Schluss (Zeile 24-26) werden noch weitere Hilfsvariablen deklariert und mit Null initialisiert. Die genaue Funktionsaufgabe der Variablen wird in den späteren Kapiteln vertieft und anhand von Anweisungsblöcken erklärt.

4.4.2 Startpunkt des Hauptprogramms

In der main-Funktion der Spannungsphasenwinkel.c Datei werden alle wichtigen Grundkonfigurationen und Funktionsaufrufe initialisiert. Dabei lässt sich das Hauptprogramm in drei große Aufgabenblöcke unterteilen. Der erste Aufgabenblock ist die Überprüfung beider PMU-Messdaten. Der darauffolgende Block ist für die Datentypumwandlung der Zeitstempel zuständig. Nach der Vorbereitung der PMU-Messdaten wird der letzte Programmabschnitt aufgerufen. In diesem Aufgabenblock werden die übersetzten Zeitstempel ausgewertet und die gesamte Phasenverschiebung berechnet.

```
538  int main()
539  {
540      printf("\n\n * Alle Ergebnisse werden in das Dokument Phasenwinkel.csv abgespeichert!!!\n");
541
542      //CSV-Dokument wird erstellt in dem alle Ergebnisse und Fehler abgespeichert werden
543      Phasenwinkel =fopen("Phasenwinkel.csv", "w");
544
545      //Ueberpruefung ob die CSV-Datei "Phasenwinkel.csv" erstellt werden konnte
546      if(Phasenwinkel == NULL)
547      {
548          fprintf(Phasenwinkel,"Datei konnte nicht erstellt werden.");
549          return -1;
550      }
551
552      //Messwerte 1 vom RaspberryPi 1 Angaben
553      csv_1 = fopen("PMU1-MW.csv", "r");
```

Bevor beide PMU-Messdaten untersucht und bearbeitet werden können, wird zu Beginn ein Textdokument im CSV-Format mit dem Namen *„Phasenwinkel“* in Zeile 543 erstellt. In der Phasenwinkel.csv Datei werden alle Fehler und Berechnungen eingetragen, die während des gesamten Programmverlaufes entstehen. Der File-Funktion werden die Parameter *„PMU1-MW.csv“* und *„r“* übergeben. Dabei zeigt der erste Parameter den Pfad der Datei auf dem Raspberry Pi an und der zweite gibt die Art des Operators wieder, die ausgeführt werden soll. Mit dem Parameter *r* wird vornherein festgelegt, dass die PMU-Messdaten nur gelesen werden soll. Nach der Initialisierung der Phasenwinkel.csv Datei wird der erste Aufgabenblock in Zeile 552 ausgeführt.

```
552       //Messwerte 1 vom RaspberryPi 1 Angaben
553       csv_1 = fopen("PMU1-MW.csv", "r");
554
555       fprintf(Phasenwinkel,"\n\n * Messwerte 1 vom PMU-Messgereat 1 // Angaben in hhmmss.ssssss\n");
556
557       if(csv_1 == NULL)
558       {
559           fprintf(Phasenwinkel, "\n * Error");
560           return -1;
561       }
562       else
563       {
564
565           if(cnt==0)
566           {
567               //Erste Funktion fur die Zeitstempel Uebersetzung wird gestartet
568               Zeitstempel_String();
569
570               //delta_T fur jede Messreihe wird erstellt
571               Rasterverschiebung_deltaPhi(PMU_check);
572
573
574           }
575
576       }
```

In Zeile 553 wird die Datei *PMU1-MW.csv* mit der Funktion *Zeitstempel_String()* (siehe Kapitel 4.4.3) aufgerufen und untersucht. Es wird kontrolliert, ob es sich um das korrekte CSV-Dokument handelt, welches aufgerufen wurde. Anschließend wird nach der *Zeitstempel_String()* Funktion die nächste Funktion *Zeitstempel()* (siehe Kapitel 4.4.4) gestartet, welche für die Übersetzung des Datentyps der Zeitstempel zuständig ist. Nach der Überprüfung und Übersetzung der ersten PMU-Messdatei des Referenzmessgerätes wird die Funktion zur Rasterverschiebung in Zeile 571 gestartet, um die Sekunden der Nachkommastellen jeder Messreihe aufzusummieren. So kann für die Funktion *Rasterverschiebung_deltaPhi()* (siehe Kapitel 4.4.8) mit der Variable *PMU_check* der passende Anweisungsblock, welcher für die erste PMU-Messdatei zuständig ist, eingestellt werden. Bevor der zweite Aufgabenblock anfängt, werden die konkludierten Variablen, die in der Funktion *Zeitstempel_String()* und *Zeitstempel()* verwendet wurden mit der Funktion *reset()* in Zeile 581 (siehe Kapitel 4.4.4.1) zurückgesetzt.

```
578       fprintf(Phasenwinkel,"\n\n * Messwerte 2 vom PMU-Messgereat 2 // Angaben in hhmmss.ssssss\n");
579
580       //Alle verwendeten Variablen werden mit Null zurückgesetzt
581       reset();
582       k=0;
583       zeile =0;
584       MW_cnt=0;
585       marker =0;
586       RV=0;
587       PMU_check =2;
588       kk=0;
589
590           fprintf(Phasenwinkel,"\n");
591
```

```
592       //Messwerte 2 vom RaspberryPi 2
593       csv_2 = fopen("PMU2-MW.csv", "r");
594       if(csv_2 == NULL)
595       {
596           fprintf(Phasenwinkel, "\n * Error");
597           return -1;
598       }
599       else
600       {
601
602
603           if(cnt==0)
604           {
605               //Erste Funktion fur die Zeitstempel Uebersetzung wird gestartet
606               Zeitstempel_String2();
607
608               //delta_T fur jede Messreihe wird erstellt
609               Rasterverschiebung_deltaPhi(PMU_check);
610           }
611
612       }
```

In Zeile 592 wird der zweite Aufgabenblock gestartet, welcher für die Untersuchung und Übersetzung der zweiten PMU-Messdatei zuständig ist. Die Funktionen im zweiten Aufgabenblock (Zeile 593 bis 612) werden nicht weiter vertieft, da die Funktionen *Zeitstempel2()* und *Zeitstempel_String2()* die gleiche Funktionalität, wie die Funktionen *Zeitstempel()* und *Zeitstempel_String()* haben. Nachdem die Vorbereitung beider PMU-Messdaten erfolgreich durchgeführt wurde, wird der dritte Aufgabenbereich des Hauptprogrammes gestartet. In diesem Abschnitt findet die eigentliche Berechnung der Phasenverschiebung statt. Die Funktion in Zeile 614 besteht aus mehreren deklarierten Funktionen, die jeweils einzeln von innen nach außen ausgeführt werden. Zu Beginn wird die Funktion „*Timestemp_review()*“ gestartet.

```
613       //Überprüfung der Zeitstemepel + Berechnung des Spannungswinkels
614       Phasenwinkel_Calculation(Mittelwert_Calculation(Timestemp_review()));
615
616       //Alle aufgerufenen Dokumente werden geschlossen und das Programm beendet
617       fclose(csv_1);
618       fclose(csv_2);
619       fclose(Phasenwinkel);
620       return 0;
621   }
```

Daraufhin startet die zweite Funktion „*Mittelwert_Calculation()*“. Diese Funktion beinhaltet die Berechnung der Zeitverschiebungen innerhalb einer vom PPS-Signal ausgelösten Messreihe. Im Anschluss wird die Funktion „*Phasenwinkel_Calculation()*“ gestartet. Zum Schluss werden alle erstellten und geöffneten CSV-Dokumente vom Hauptprogramm in Zeile 617 bis 620 geschlossen.

4.4.3 Überprüfung der PMU-Messdaten

Wie in Kapitel 4.4.2 erwähnt, startet das Spannungsphasenwinkel.c Programm mit der Untersuchung der PMU-Messwerte. Dafür wird die Funktion *Zeitstempel_String()* aufgerufen, die alle Messreihen auf Vollständigkeit untersucht. Jeder Zeitstempel ist mit der Zeichenkette „*Zeitstempel*“ markiert. Bevor die eigentliche Codierung der Zeitstempel in eine Gleitkommazahl durchgeführt wird, startet zu Beginn die Suche nach der Zeichenkette „Zeitstempel“. Diese definierte Zeichenkette signalisiert den genauen Startpunkt der erfassten Zeitmessung. In Abbildung 17 wird die oben beschriebene Suchfunktion abgebildet.

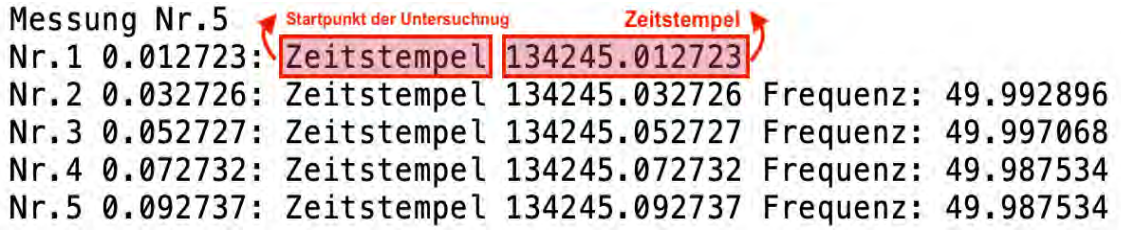

Abbildung 17: Aufgabeverdeutlichung der Funktion *Zeitstempel_String()* [35]

Abbildung 17 beinhaltet zwei rot markierte Blöcke. Der erste rote Block gibt den Startpunkt der Messung an, wohingegen die zweite rote Markierung die erstellten Zeitstempel einer fallenden Rechteckflanke abbildet.

```
213  void Zeitstempel_String(){
214
215      //Die ganze Zeichen der CSV-Messdatei werden durchlaufen
216      for(i=0; i<7451; i++)
217      {
218          //Jedes einzelne Zeichen wird untersucht => Es wird nach dem String "Zeitstempel" gesucht
219          zeichen[i] = fgetc(csv_1);
220          if(zeichen[i] == ' ' && cnt == 0)
221          {
222              //Wenn der String gefunden wurde, wird er Zwischengespeichert
223              if(zeichen[i-1]== 'l')
224              {
225                  if(zeichen[i-2] == 'e')
226                  {
227                      if(zeichen[i-3] == 'p')
228                      {
229                          if(zeichen[i-4] == 'm')
230                          {
231                              if(zeichen[i-5]== 'e')
232                              {
233                                  if(zeichen[i-6] == 't')
234                                  {
235                                      if(zeichen[i-7] == 's')
236                                      {
237                                          if(zeichen[i-8] == 't')
238                                          {
239                                              if(zeichen[i-9]== 'i')
240                                              {
241                                                  if(zeichen[i-10] == 'e')
242                                                  {
243                                                      if(zeichen[i-11] == 'Z')
244                                                      {
245                                                          cnt=1; //Ist die Zeichenkette vollstandig wird inktrementiert
246                                                                 // erster Zeitstempel wurde gefunden hhmmss.ssssss
247                                                      }
248                                                  }
```

Die Suche nach der Zeichenkette *Zeitstempel* in der PMU-Messdatei beginnt in Zeile 216. Dabei durchläuft die Schleife einzeln alle Zeichen der PMU-Messdatei, bis die erste Zeichenkette gefunden wurde. Während der Programmierung hat sich ergeben, dass eine PMU-Messdaten aus 7451 einzelnen Zeichen besteht. Mit Hilfe der fget()-Funktion (Zeile 219) werden alle gelesenen Zeichen einzeln von der aktuellen Position der Messdatei entnommen und in *zeichen* abgespeichert.

```
262              else if (cnt == 1)
263              {
264                  //Ist die Zeichenkette vollstandig wird die Zeitstempel-Codierung gestartet
265                  Zeitstempel();
266
267              }
268
269          }
270
271      }
```

Nachdem die erste Zeichenkette vollständig gefunden wurde, wird in Zeile 245 die Hilfsvariable *cnt* mit einer Eins initialisiert. Damit wird der Start der nächsten Funktion *Zeitstempel()* in Zeile 265 signalisiert.

4.4.4 Übersetzung des Datentyps

Mit der Funktion *Zeitstempel()* werden die Zeitstempel, welche als Datentypen Char codiert sind in eine Gleitkommazahl (Datentyp: Double) übersetzt. In Zeile 45 werden die zwölf Stellen des Zeitstempels (Format: *hhmmss.ssssss*) mit Hilfe der ASCII-Tabelle umgerechnet.

```
39  void Zeitstempel()
40  {
41
42      if(zeichen[i]!= '.' && j != 12)
43      {
44          //Die einzelnen Zeichen werden von Char in Double umgewandelt
45          zahl[k][j]=(zeichen[i]-48); //ASCI-CODE
46          j++;
47      }
48      else if(j == 12)
49      {
50
51          j=100000;
52
53          //Uebersetzung der Vorkommastellen des Zetstempels
54          for(i=0; i<6; i++)
55          {
56              result_vorkomma+= zahl[k][i]*j;
57
58              j=j/10;
59          }
60          //Fertige Vorkommastelle wird als Double-Wert abgespeichert
61          result_vorkomma_safeX[kk]=result_vorkomma;
```

Eine Null im ASCII-Code entspricht einer 48 als Dezimalzahl. Diese wird von den einzelnen Zeichen des Zeitstempels subtrahiert (Zeile 45). Die einzelnen übersetzten Werte des Zeitstempels werden an der Stelle i des zweidimensionalen Arrays *zahl* abgespeichert. In die erste Dimension k (Zeile) wird die Messreihennummer abgelegt, wohingegen die zweite Dimension j (Spalte) für die fünf übersetzten Zeitstempel zuständig ist. Sind die einzelnen Stellen des Zeitstempels vollständig übersetzt und abgespeichert, werden die einzelnen Werte des Zeitstempels in Zeile 61 zusammengerechnet. Dafür wurde ein Algorithmus entworfen, der aus den einzelnen Werten des Zeitstempels den Gesamtwert berechnet. Für die Berechnung des Gesamtwertes wird der Zeitstempel dafür zunächst in Vor- und Nachkommastellen aufgeteilt und einzeln umgerechnet. In Zeile 56 werden die einzelnen Werte der Vorkommastellen, die im Array *zahl* abgespeichert liegen, als ein Gesamtwert zusammengeführt und in *result_vorkomma_safeX* (Zeile 61) gesichert.

```
65          //Uebersetzung der Nachkommastelle des Zeitstempels beginnt
66          for(i=6; i<12; i++)
67          {
68              //Zwischenabspeicherung der Nachkommastellen für die Rasterverschiebungsberechnung
69              if(i>=6 && MW_cnt!=5) //Wenn die Nachkommastelle nicht der letzte Wert der Messreihe ist
70              {
71
72                  RV+= zahl[k][i]*j;
73
74              }
75              else if(i>=6 && MW_cnt == 5) //Wenn die Nachkommastelle der letzte der Wert der Messreihe ist
76              {
77
78                  if(i==6)
79                  {   //delta_T wird vorbereitet (fur jede Messreihe )
80                      Rasterverschiebung_deltaPhi(PMU_check);
81                      RV=0;
82                  }
83                  if(zahl[k][i]*j !=0)
84                  {
85                      sum++;
86                  }
87
88                  RV+= zahl[k][i]*j;
89
90              }
```

```
92              result_nachkomma+= zahl[k][i]*j;
93              j=j/10;
94          }
95          //Nachkommastelle wird als Double-Wert abgespeichert
96          result_nachkomma_safeX[kk]=result_nachkomma;
97
98          //Untersuchung der Zeitstempel: Es wird untersucht, ob der  Zeitstemple der letzte der Messreihe ist
99
```

Das Gleiche wird in der Schleife in Zeile 66 für die Nachkommastellen durchgeführt. Der Gesamtwert der Nachkommastelle wird stattdessen in *result_nachkomma_safeX* in Zeile 96 gesichert.

```
98          //Untersuchung der Zeitstempel: Es wird untersucht, ob der  Zeitstemple der letzte der Messreihe ist
99
100         if(MW_cnt!=5) //Wenn der Zeitstempel nicht der letzte der Messreihe ist
101         {
102             result_float[zeile][k] = result_vorkomma + (double)result_nachkomma/1000000;
103             // Zeitstempel wird verrechnet und in das CSV-Dokument "Phasenwinkel" gespeichert
104             fprintf(Phasenwinkel,"\n%lf", result_float[zeile][k]);
105             kk++;
106             MW_cnt++;
107             k++;
108
109
110
111         }
112         else //Wenn der Zeitstempel  der letzte der Messreihe ist
113         {
114             zeile++;
115             k=0;
116             MW_cnt=0;
117             result_float[zeile][k] = result_vorkomma + (double)result_nachkomma/1000000;
118             // Zeitstempel wird verrechnet und in das CSV-Dokument "Phasenwinkel" gespeichert
119             fprintf(Phasenwinkel,"\n%lf", result_float[zeile][k]);
120             k++;
121             MW_cnt++;
122             kk++;
123
124
125         }
126         //Variablen werden zuruckgesetzt und der naechste Zeitstempel wird uebersetzt
127         reset();
128
129     }
130 }
```

Wurde die Umrechnung erfolgreich umgesetzt, wird daraufhin von der if-Verzweigung (Zeile 100) geprüft, ob der übersetzte Zeitstempel auch zur richtigen Messreihe gehört. In der Zeile 102 werden die vorbereiteten Vor- und Nachkommastellen miteinander addiert und in dem Array *result_float* abgespeichert. Anschließend werden die Hilfsvariablen *MW_cnt* und *k* mit Eins inkrementiert, um die Übersetzung des nächsten Zeitstempels der Messreihe einzustellen. Ist die Bedingung in Zeile 100 falsch, wird stattdessen die else-Bedingung in Zeile 112 gestartet. Wenn der übersetzte Zeitstempel nicht zu der Messreihe gehört, wird die Variable *zeile* (Zeile 114) inkrementiert, um die nächste Zeile des Arrays *result_float* zu markieren. Mit der Nullinitialisierung der Variablen *k* und *MW_cnt* (Zeile 120 bis 121) wird das Array *result_float* auf die erste Stelle der nächsten Messreihe eingestellt, um die darauffolgenden übersetzten Zeitstempel in der richtigen Messreihe abspeichern zu können. Zum Schluss werden alle bisherigen, in der Funktion inkrementierten Variablen mit der *reset()*-Funktion (Zeile 127) mit Null initialisiert. Damit kann erneut der nächste Zeitstempel problemlos und fehlerfrei übersetzt werden. Dieser Vorgang wird so lange durchgeführt, bis alle Messreihen, die sich in der PMU-Messdatei befinden, vollständig überprüft und umgewandelt sind.

4.4.4.1 Funktion – reset()

Die reset() Funktion wird immer nur dann aufgerufen, wenn ein Zeitstempel erfolgreich aufgerufen und übersetzt wurde.

```
29  //Alle Variablen werden mit Null initialisiert
30  void reset()
31  {
32      cnt = 0;
33      j = 0;
34      result_vorkomma = 0;
35      result_nachkomma = 0;
36      sum = 0;
37  }
```

Daraufhin werden alle in der Funktion verwendeten Variablen, wie *result_vorkomma* und *result_nachkomma, j* und *cnt* mit Null initialisiert. Dieser Vorgang ist besonders wichtig, da die nächste PMU-Messdatei (PMU2-MW.csv) nach demselben Überprüf- und Übersetzungsverfahren, wie in Kapitel 4.4.3 und 4.4.4 beschrieben, durchgeführt werden.

4.4.5 Untersuchung der Zeitstempel

Nachdem die Datentypübersetzung der Zeitstempel erfolgreich ausgeführt wurde, wird die Funktion zur Berechnung der Phasenverschiebung gestartet. Die Berechnungsfunktion der Phasenverschiebung beinhaltet mehrere separat deklarierte Funktionen, welche einzeln aufgerufen werden und aufeinander aufbauen. Zuerst wird die Funktion *Timestemp_review()* gestartet. In der Funktion wird eine Kontrollvariable *„control*“ programmiert. Die Variable *control* wird bei einer erfolgreichen Funktionsausführung immer mit *true* initialisiert. Somit wird verhindert, dass auftretende Fehler mit in die nächste Funktion *Mittelwert_Calculation()* übergeben werden. Eine Null entspricht einem *true* und eine Eins einem *false* Wert. Wurde die Variable *control* mit dem booleschen Wert *false* initialisiert, wird das Hauptprogramm in der nächsten Funktion sofort beendet.

```
407 int Timestemp_review()
408 {
409     int control = 0;
410     for (i = 0; i < 100; i++)
411     {
412         //Vorkammastelle --Kontrolle ob beide PMU-Messwerte zum selben PPS-Zeitpunkt gestartet haben
413         if(result_vorkomma_safeX[i]!=result_vorkomma_safeX2[i] &&
414                 (result_vorkomma_safeX[i] != 0 && result_vorkomma_safeX2[i] != 0))
415         {
416
417             fprintf(Phasenwinkel, "\n * Zeitstempel an der Stelle %i: MW 1: %.3lf und MW2: %.3lf ungleich",
418                     i+1, result_vorkomma_safeX[i], result_vorkomma_safeX2[i]);
419             control = 1;
420
421
422         }//Ueberprüfung der Vorkommastellen auf Ausreißer
423         else if(result_vorkomma_safeX[i] == 0 && result_vorkomma_safeX2[i] == 0)
424         {
425
426             result_vorkomma_safeX[i] = 0;
427             result_vorkomma_safeX2[i] = 0;
428         }
```

Die Funktion untersucht alle übersetzten Zeitstempel auf Vollständigkeit und Messgenauigkeit. Dafür werden die erfassten Vor- und Nachkommastellen beider PMU-Messdaten miteinander verglichen. Zum einen wird kontrolliert, dass alle identifizierten Ausreißer, die innerhalb einer PMU-Messreihe erkannt, auch entfernt wurden. Zum anderen wird zusätzlich untersucht, ob beide PMU-Messgeräte zum selben PPS-Zeitpunkt dieselben Messwerte der Messreihe aufgezeichnet haben. Damit wird eine fehlerfrei Mittelwertberechnung und ein präziserer Spannungswinkel (Gleichung 5) innerhalb der Messreihe erzielt. Wenn die Vorkommastellen (Zeile 413-415) beider PMU-Messdaten unterschiedliche Zeitstempel aufweisen, wurden beide PMU-Messgeräte nicht zum selben PPS-Zeitpunkt gestartet. Für diesen Fall wird die Variable *control* in Zeile 419 mit einer

Eins initialisiert und der aktuelle Stand der Zeitverschiebungsberechnung auf der SD-Karte (Zeile 417) abgespeichert. Da es sich um einen *false* Wert handelt, wird die Funktion abgebrochen und das Hauptprogramm zur Spannungswinkelberechnung sofort beendet. In der if-Verzweigung (Zeile 423) werden erneut die Vorkommastellen untersucht, ob beide Feldelemente der Arrays mit einer Null initialisiert sind. Ist die geprüfte if-Verzweigung wahr, wird der zugehörige Anweisungsblock (Zeile 426 bis 427) gestartet. Dabei wird die andere Messstelle der Messreihe ebenfalls mit Null initialisiert, um keinen Berechnungsfehler bei der Zeitverschiebung zu erhalten.

```
429         //Ueberprüfung der Nachkommastellen auf Ausreißer
430         if(result_nachkomma_safeX[i] == 0 || result_nachkomma_safeX2[i] == 0)
431         {
432
433             fprintf(Phasenwinkel, "\n * XX Zeitstempel an der Stelle %i: MW 1: %.3lf und MW2: %.3lf ungleich",
434                     i+1, result_nachkomma_safeX[i], result_nachkomma_safeX2[i]);
435
436             result_nachkomma_safeX[i] = 0;
437             result_nachkomma_safeX2[i] = 0;
438         }
439
440
441     }
```

Für die Nachkommastellen werden die Messwerte an der Stelle *i* der Arrays *result_nachkomma_safeX* und *result_nackomma_safeX2* miteinander verglichen. Es wird ebenfalls untersucht, ob es sich bei den Messwerten, die mit Null initialisiert werden, um Ausreißerstellen handelt. Ist die geprüfte Bedingung in Zeile 430 wahr, wird der Aufgabenblock (Zeile 431-438) ausgeführt, welcher den Messwert an der Stelle *i* des Arrays mit Null initialisiert. Bei der Untersuchung der Vor- und Nachkommastellen werden alle berechneten Zwischenergebnisse und erkannten Fehler in dem Dokument *Phasenwinkel* protokolliert. Ist der Vorgang abgeschlossen wird der Wert der Variable *control* mit *return* der nächsten Funktion *Mittelwert_Calculation()* übergeben.

```
441     }
442     if(!control)
443     {   // Alles in Ordnung = TRUE mit control = 0
444         fprintf(Phasenwinkel, "\n\n * Alle Zeitstempel stimmen im Format hhmmss.000 ueberein \n");
445         return control;
446     }
447     else
448     {   //Fehler bei der Untersuchung FALSE = 1
449         return control;
450     }
451 }
```

4.4.6 Mittelwertberechnung

Nach der Überprüfung der beiden erfassten Zeitstempel wird die Funktion *Mittelwert_Calculation()* aufgerufen. In ihr wird die zweite Vorbereitung für die Berechnung der Phasenverschiebung durchgeführt. In der Funktion *Mittelwert_Calculation()* werden die Zeitverschiebungen aus allen gültigen Zeitstempel innerhalb einer Messreihe gebildet.

```
328 int Mittelwert_Calculation(int control)
329 {
330     sum = 0, k = 0;
331     if (!control)
332     {
333         fprintf(Phasenwinkel, "\n * Gesamte Zeitverschiebung jeder erfassten PPS-Periode ( delta T in [ms])\n");
334
```

In der Zeile 330 werden die Variablen *sum* und *k* deklariert und mit Null initialisiert. Bei diesen Variablen handelt es sich um Zählerindizes für Schleifen und Markierung der Arraystellen. In der Zeile 331 wird zu Beginn mit *control* geprüft, ob in der vorherigen

Funktion *Timestemp_review()* ein Fehler bei der Überprüfung der Vor- und Nachkommastellen der Zeitstempel aufgetreten ist.

```
335          //Zeitverschiebung wird erstellt (innerhalb einer Messreihe)
336          for(i=0; i<20; i++)
337          {
338              for(j=0; j<5; j++)
339              {
340                  //Zeitverschiebung: Berechnung + double casten + Abspeicherung
341                  result[i][j]= ((double)result_nachkomma_safeX[k]-
342                                 (double)result_nachkomma_safeX2[k])/1000;
343
344                  k++;
345
346                  //Betragbildung der berechneten Zeitverschiebung
347                  if(result[i][j] < 0)
348                  {
349                      result[i][j]*=-1;
350
351                  }
352              }
353
354              //Bildung der Zeitverscheibung einer Messreihe
355              result_mittelwert_frame[i] = (result[i][0]+result[i][1]+result[i][2]+result[i][3]+result[i][4]);
356              fprintf(Phasenwinkel, "\n %lf ms ", result_mittelwert_frame[i]*1000);
```

Beträgt die Variable *control* den Wert minus Eins, wird die Mittelwertberechnung in Zeile 404 beendet. Ansonsten startet die Berechnung der Zeitverschiebungen in Zeile 336. In Zeile 341 wird die Berechnung der Zeitverschiebung zwischen den beiden gemessenen fallenden Flanken erstellt (siehe Abbildung 9, Kapitel 3.4.1). Hierfür werden alle bearbeitenden Zeitstempel der ersten Messdatei mit der zweiten Messdatei subtrahiert und die daraus resultierende Zeitverschiebung in das finale Array *result* abgespeichert. Damit bei der Zeitverschiebungsberechnung keine fehlerhaften Ergebnisse entstehen, werden beide Nachkommastellen mit dem Datentyp *Double* gecastet (Zeile 341) und der Betrag gebildet (Zeile 347-351). Nachdem die Zeitverschiebungen der ersten Messreihe fertig berechnet wurde, werden diese anschließend miteinander addiert (Zeile 355).

```
360          //Untersuchung der gultigen Summe der Messwerte
361          for(i=0; i<20; i++)
362          {
363
364              if(result[i][0] !=0)
365              {
366                  sum++;
367
368              }
369              if(result[i][1] !=0)
370              {
371                  sum++;
372
373              }
374              if(result[i][2] !=0)
375              {
376                  sum++;
377
378              }
379              if(result[i][3] !=0)
380              {
381                  sum++;
382
383              }
384              if(result[i][4] !=0)
385              {
386                  sum++;
387
388              }
389
390
391          }
```

In der Schleife (Zeile 361 bis 391) werden die berechneten Zeitverschiebungen nach mit nicht Null initialisierten Stellen gezählt. Das Ganze wird durch das Inkrementieren der Variable *sum* realisiert.

```
392             // Alles in Ordnung = TRUE mit control = 0
393             control = 0;
394
395
396         }
397         else
398         {   //Fehler wurde entdeckt oder ist entstanden bei der Berechnung der Zeitverschiebung = FALSE mit control = -1
399             fprintf(Phasenwinkel,"\n * Fehler bei den Zeitstempeln keine Mittelwert berechnung moglich\n");
400             control = -1;
401         }
402
403
404         return control;
405     }
```

Um die Messreihe zu beenden, wird die Variable *sum* mit Null zurückgesetzt. Die Funktion wird beendet, wenn der Index *i* in der Berechnungsschleife (Zeile 320) den Endwert erreicht hat. Daraufhin wird *control* mit Null initialisiert und die nächsten Funktion *Phasenwinkel_Calculation()* übergeben.

4.4.7 Berechnung der Phasenverschiebung

Die Phasenverschiebung ist, wie in Kapitel 3.1 beschrieben, ein Produkt aus der Frequenz der Zeitverschiebung zwischen den beiden gemessenen Rechtecksignalen und der Gradkonstante von 360°. Zu Beginn wird der Wert der Variable *control* mittels der if-Verzweigung in Zeile 457 geprüft. Wird bei der Überprüfung eine Null ermittelt, wird der Anweisungsblock ab der Zeile 460 gestartet. Andernfalls wird die Zeile 489 ausgeführt, welche die Berechnung der Phasenverschiebung beendet. Zusätzlich wird die Frequenz des gemessenen Rechtecksignals mit der Variable *frequency* definiert (Zeile 455). Durch die Initialisierung der Variable *PMU_check* mit dem Wert Drei (Zeile 460), wird in der Funktion *Rasterverschiebung_deltaPhi()* (siehe Kapitel 4.4.9) der passende Anweisungsblock eingestellt. Nachdem alle Grundeinstellungen und Vorbereitungen ausgeführt wurden, startet die Schleife in Zeile 465 zur Berechnung der Phasenverschiebung.

```
453   void Phasenwinkel_Calculation(int control)
454   {
455       int frequency=50;
456
457       if (!control)
458       {
459           //Dritter Anweisungsblock wird fur die Rasterverscheibungsfunktion eingestellt
460           PMU_check = 3;
461
462           fprintf(Phasenwinkel, "\n\n * Phasenwinkel fur jedes Frame mit Rasteruntersuchung[Grad]\n");
463
464           //Jede Messreihe wird durchgegangen und gleichzeitig auf Rasteruebergang untersucht
465           for(i = 0; i < 20 ; i++)
466           {
467               if(Rasterverschiebung_deltaPhi(PMU_check))
468               {
469                   fprintf(Phasenwinkel, "\n Übergagng zwischen 359° und 0° entdeckt %lf° ", phi[i]);
470               }
471               else
472               {   //Phasenverschiebung jeder Messreihe
473                   phi[i]= frequency*result_mittelwert_frame[i]*360/1000;
474                   fprintf(Phasenwinkel, "\n Periode %i: %lf° ",i+1, phi[i]);
475               }
476           }
```

Dabei wird die Schleife in Zeile 465 20-mal durchlaufen. Ein Schleifendurchlauf entspricht einer Phasenverschiebung pro Messreihe. Während der Schleifenausführung wird die Rasterverschiebungsfunktion (Zeile 419) aufgerufen, um zu überprüfen, ob zwischen den beiden Spannungswinkeln eine Rasterverschiebung zwischen 359° und 0° aufgetreten ist. Für den Fall, dass beide Spannungswinkeln keine Rasterverschiebung aufweisen, wird der Anweisungsblock in der else-Bedingung (Zeile 471) ausgeführt. In Zeile 473 wird mit der Zeitverschiebung, welche im Array *result_mittelwert* gespeichert

wird, die Phasenverschiebung berechnet. Alle berechneten Phasenverschiebungen innerhalb einer Messaufnahme werden in dem Array *phi* gesichert.

```
477         //--Gesamte Phasenverschiebung des gemessene Systems
478         double Phasenverschiebung = 0;
479
480         for(i = 0; i < 20; i++)
481         {
482
483             Phasenverschiebung+=phi[i];
484         }
485         //Mittelwertbildung der Phasenverschiebung einer Messwertaufnahme
486         fprintf(Phasenwinkel, "\n\n\n Gesamte Phasenverschiebung des Messsytems %lf° ", Phasenverschiebung/sum);
487
488     }
489     else
490     {
491         fprintf(Phasenwinkel, "\n * Fehler bei Spannungswinkelberechnung");
492
493     }
```

In Zeile 480 wird die gesamte Phasenverschiebung aus allen einzelnen Phasenverschiebungen der Messreihen berechnet und gemittelt. Der Schleifenvorgang wird abgebrochen, wenn der Zählerindex *i* den Maximalwert von 20 erreicht hat. Daraufhin wird die Funktion beendet und die geöffneten Messdaten sowie das CSV-Dokument *Phasenwinkel* geschlossen.

4.4.8 Rasterverschiebung der Spannungswinkeln

In diesem Kapitel wird der Phasenwinkelübergang beider gemessenen Rechtecksignale behandelt. Dafür wurde in der Funktion *Rasterverschiebung_deltaPhi()* ein Algorithmus entwickelt, der beide Spannungswinkeln am Nulldurchgang untersucht. Es wird kontrolliert, ob zwischen den beiden fallenden Rechteckflanken ein Phasenübergang zwischen 359° und 0° gemessen wurde.

```
499 int Rasterverschiebung_deltaPhi(int PMU_check)
500 {
501     //Referenzknoten PMU-Messgeraet 1
502     if(PMU_check == 1)
503     {
504         Raster_VS_PMU1[marker] = (double)(RV/1000);
505         marker++;
506         sum=0;
507         return 0;
508     }
509     //zweiter Messpunkt PMU-Messgeraet 2
510     else if (PMU_check == 2 )
511     {
512         Raster_VS_PMU2[marker] = (double)(RV/1000);
513         marker++;
514         sum=0;
515         return 0;
516     }
```

Die Funktion wird immer nur dann gestartet, wenn eine Messreihe innerhalb der Messdatei erfolgreich überprüft und übersetzt wurde (siehe Kapitel 4.4.3 und 4.4.4). Der Funktion wird der Parameter *PMU_check* übergeben, welcher den auszuführenden Anweisungsblock in der Funktion einstellt. Bei einem *PMU_check* Wert von Eins wird die if-Verzweigung in Zeile 502 geprüft und ausgeführt. Bei einem *PMU_check* Wert von Zwei wird der Anweisungsblock in Zeile 510 eingestellt. Der erste Anweisungsblock (Zeile 502 bis 507) bezieht sich auf die erste PMU-Messdatei des Referenzmessgerätes. Der darauffolgende Anweisungsblock (Zeile 511 bis 515) ist für die PMU-Messdatei des zweiten PMU-Messgerätes zuständig. Die Variable *RV* (Zeile 504 & 512) beinhaltet den

Gesamtwert aller addierten Nachkommastellen, die innerhalb einer Messreihe verrechnet wurden. Die aufsummierten Nachkommastellen $t_{K_{RF,PMU1}}$ des ersten PMU-Messgerätes werden in dem Array *Raster_VS_PMU1* abgespeichert. Für die Nachkommastellen $t_{K_{PMU2}}$ des zweiten Messpunktes wird der Anweisungsblock in Zeile 512 ausgeführt und in der Variable *Raster_VS_PMU2* abgelegt, um somit das Überschreiben der berechneten Ergebnisse zu verhindern. Während der Berechnung der Phasenverschiebungen in der Funktion *Phasenwinkel_Calculation()* (siehe Kapitel 4.4.7) wird der dritte Funktionsblock von *Rasterverschiebung_deltaPhi()* aufgerufen. Dafür erhält *PMU_check* den Wert Drei, welcher in Zeile 518 geprüft wird. In Zeile 510 und 526 werden beide berechneten Werte innerhalb einer Messreihe ($t_{K_{RF,PMU1}}$ und $t_{K_{PMU2}}$) miteinander verrechnet, um die Art der Rasterverschiebung festzustellen.

```
517         //Auswertung der Phasenübergangslage zwischen 359° und 0°
518         else
519         {
520             if(Raster_VS_PMU1[i] - Raster_VS_PMU2[i] > 25){
521
522                 delta_Tn[i] = Raster_VS_PMU1[i] - Raster_VS_PMU2[i] -50;
523                 phi[i] = delta_Tn[i] *360*50;
524                 return 1;
525
526             }else if(Raster_VS_PMU1[i] - Raster_VS_PMU2[i] < -25){
527
528                 delta_Tn[i] = Raster_VS_PMU1[i] - Raster_VS_PMU2[i] + 50;
529                 phi[i] = delta_Tn[i] *360*50;
530                 return 1;
531             }
532
533         }return 0;
534
535
536     }
```

Zuerst wird in Zeile 529 geprüft, ob die Differenz zwischen $t_{K_{RF,PMU1}}$ und $t_{K_{PMU2}}$ größer als $25\ \mathrm{ms}$ ist. Ist das Ergebnis der Überprüfung *wahr*, wird der Differenz $50\ \mathrm{ms}$ abgezogen und in dem Array *delta_Tn* (Zeile 522) abgespeichert. Mit dem errechneten Wert wird die angepasste Phasenverschiebung berechnet und in der Variable *phi* gesichert. Ist die Abfrage der if-Verzweigung in Zeile 520 ungültig, wird die zweite if-Verzweigung in Zeile 526 untersucht. Dabei wird die Bedingung geprüft, ob die Differenz zwischen $t_{K_{RF,PMU1}}$ und $t_{K_{,PMU2}}$ kleiner als $-25\ \mathrm{ms}$ ist. Ist die gestellte Bedingung *wahr*, wird der Anweisungsblock in Zeile 528 gestartet. Dabei wird erneut eine Differenz zwischen den Werten gebildet und $50\ \mathrm{ms}$ werden dazu addiert. Die resultierende Zeitverschiebung wird in der Variable *delta_Tn* zwischengespeichert, woraus sich die angepasste Phasenverschiebung ergibt. Die Phasenverschiebung für den zweiten Fall wird ebenfalls in dem Array *phi* gespeichert.

5. Messwertaufnahme

Nachdem in den vorherigen Kapiteln detailliert auf die einzelnen Baukomponenten und die Software eines WAM-Systems eingegangen wurde, werden Messungen im elektrischen Energieversorgungsnetz aufgenommen. Bevor die Messung zur Phasenverschiebung in einem Energieversorgungsnetz realisiert werden kann, wird zu Beginn die Phasenverschiebung zwischen den beiden Komparatorschaltungen ermittelt. Eine gestartete Messung beinhaltet 20 Messreihen mit jeweils fünf hintereinander aufgenommenen Zeitstempel. Dabei werden Messungen zur steigenden und fallenden Rechteckflanke aufgezeichnet, um sicherzustellen, welche Flankendetektion messtechnisch eine höhere Genauigkeit liefert. Zur Veranschaulichung werden die minimale, mittlere und maximale Abweichung der Messaufnahme analysiert. Mit diesem erzielten Messergebnis wird daraufhin zwischen den beiden PMU-Messgeräten eine Last zugeschaltet und an das elektrische Energieversorgungsnetz angeschlossen.

5.1 Phasenverschiebung zwischen beiden PMU-Messgeräten

In diesem Kapitelabschnitt werden die Zeit- und Phasenverschiebungen zwischen den beiden entwickelten PMU-Messgeräten ermittelt. Dabei ist es wichtig, dass die gemessenen Zeitverschiebungen möglichst gering sind, um die darauf aufbauende Phasenverschiebung nicht zu verfälschen und somit die Abschlussarbeit brauchbar wird. Für die Untersuchung der Zeitverschiebungen wurden die steigenden und fallenden Rechteckflanken miteinander verglichen. Das Flanken-Interrupt wurde für beide Flankentypen passend im Programmcode *Zeitstempel.c* konfiguriert und kalibriert. Der genaue Versuchsaufbau zur Messung der steigenden und fallenden Rechteckflanke wurde nach Abbildung 18 umgesetzt.

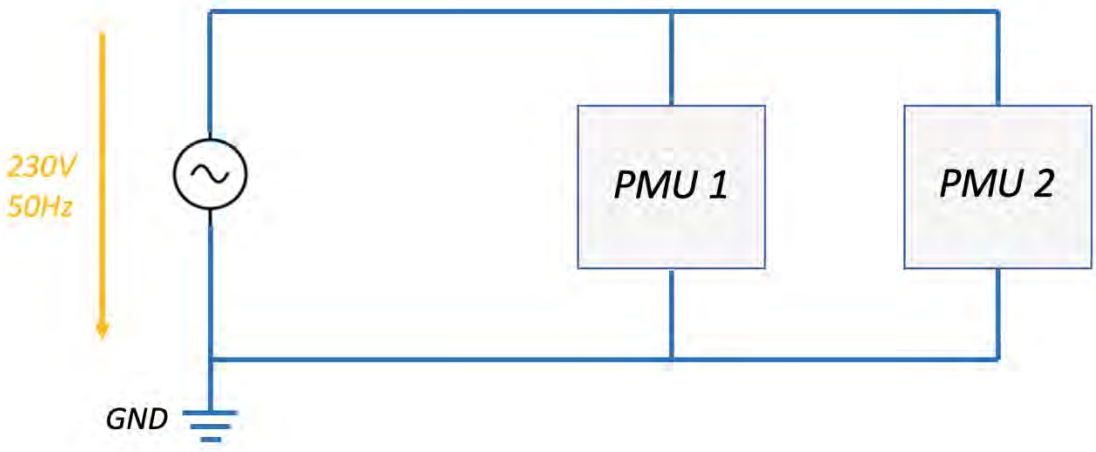

Abbildung 18: Schaltungsaufbau zur Zeitverschiebungsmessung zwischen beiden PMU-Messgeräten [35]

Die Tabelle 4 beinhaltet 20 gemessene Zeitverschiebungen zur steigenden und fallenden Rechteckflanke. Die Messergebnisse wurden softwaretechnisch ermittelt und

miteinander verglichen. Die grün markierten Zeitverschiebungen sind gültige Messergebnisse, die mit mindestens einer Abtastfrequenz von $50\ \mathrm{kHz}$ vom GPIO-Pin ausgelesen wurden. Die rot markierten Zeitverschiebungen zeigen eine zu große Messabweichung und stellen einen Messfehler dar.

Messreihe	Zeitverschiebung [μs]	
	steigende Flanke	fallende Flanke
1	24,4	15,0
2	19,2	1,0
3	15,6	1,0
4	24,8	5,0
5	5,4	3,0
6	13,6	4,0
7	40,6	5,4
8	15	2,4
9	17	3,8
10	8,6	3,8
11	8,8	4,6
12	3,8	11,4
13	11,2	5,8
14	10,4	7,2
15	4,4	8,6
16	36,2	14,4
17	6,6	25,6
18	5,8	32,4
19	31,4	24,6
20	27,2	13,6

Tabelle 4: Messwertaufnahme - gemittelte Zeitverschiebungen zwischen steigenden und fallenden Rechteckflanken [35]

Die Messergebnisse in Tabelle 4 zeigen, dass mit einer fallenden Flankendetektion eine deutlich präzisere Messgenauigkeit zwischen den beiden PMU-Messgeräten erreicht wird. Die steigenden Flanken sind im Vergleich zu den fallenden Flanken deutlich ungenauer und liefern Zeitverzögerungen, die teilweise über 30 Mikrosekunden lang sind. Um eine konkrete Aussage über die Messgenauigkeit treffen zu können, wurden die Messreihen aus der Tabelle 4 in ein Diagramm übertragen. Das Diagramm beinhaltet die Achsen für die Messreihe (x-Achse) und die berechneten Zeitverschiebungen in Mikrosekunden (y-Achse). Zusätzlich wurde eine zweite y-Achse eingeführt, um die Messwerte als minimale, mittlere oder maximale Abweichung definieren zu können. Die roten Punkte in der Abbildung 19 markieren die Messpunkte der Zeitverschiebungen. Die orangene Kurve zeigt den Verlauf der steigenden Flanke an, wohingegen die grüne Kurve für die fallende Flanke steht. Die blau gestrichelten Linien bilden die Grenze zwischen minimaler, mittlerer und maximaler Messabweichung.

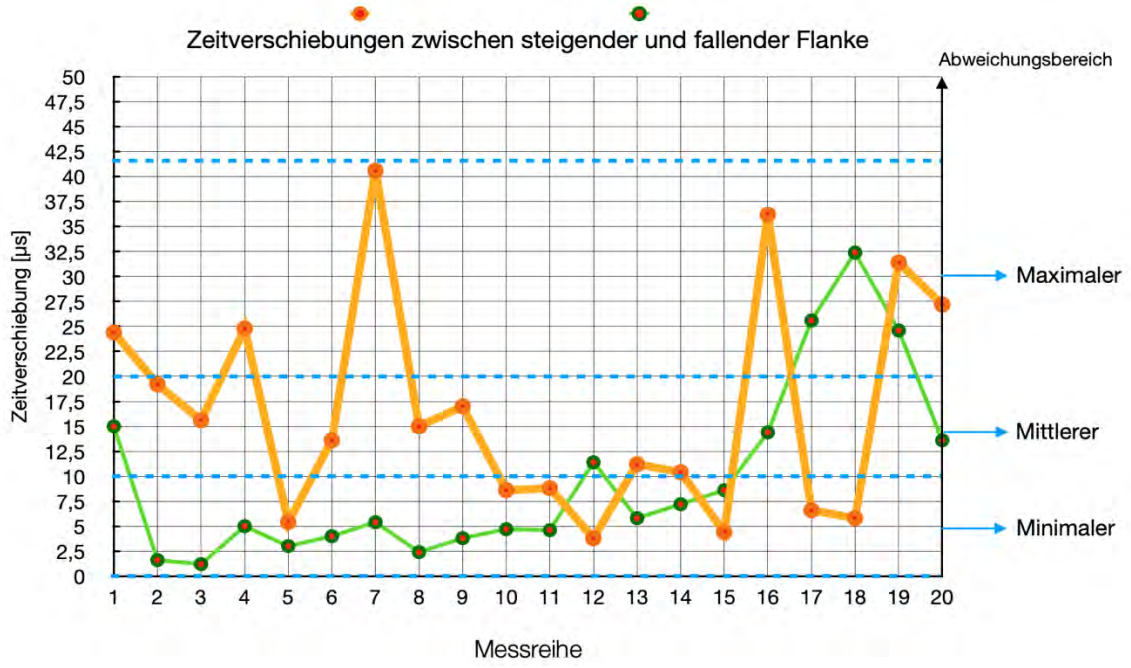

Abbildung 19: Zeitverschiebungen zwischen der steigenden und fallenden Rechteckflanke [35]

Aus dem grünen Kurvenverlauf geht hervor, dass die Zeitverschiebungen der fallenden Rechteckflanken viel dichter und enger aneinander liegen als bei der orangenen Kurve. Es wurden 13 von 20 Zeitverzögerungen mit der fallenden Flankendetektion erfasst, die in einem sehr guten Messbereich unter 10 µs liegen. Dieser Bereich wird als minimale Abweichung bezeichnet. Bei den steigenden Flanken wurden stattdessen nur sieben von 20 Messwerten aufgenommen, die im minimalen Messbereich liegen. Damit ist die Messwertaufnahme mit einer fallenden Rechteckflanke ca. um den Faktor zwei genauer. Mit einer steigenden Flankeneinstellung würde es zu einer enormen Messungenauigkeit der Phasenverschiebungen kommen und das Ergebnis dadurch unbrauchbar machen. Aus diesem Grund werden für die Berechnung der Phasenverschiebungen nur die fallenden Flanken verwendet, da diese eine höhere Messgenauigkeit liefert und eine Verschiebung von Nullsekunden zwischen den beiden PMU-Messgeräten sehr nahe kommt. Der Grund für eine solche enorm gemessene Zeitverzögerungen bei den steigenden Flanken, liegt an den entwickelten Komparatorschaltungen. Mögliche Ursachen werden in [28] nicht genannt, weshalb nicht weiter darauf eingegangen wird.

5.1.1 Messabweichung

Die minimale Abweichung ist der Messbereich, bei dem die Zeitverschiebungen zwischen 1 μs und 10 μs liegen. Aus der Tabelle 4 und Abbildung 19 kann entnommen werden, dass innerhalb einer Messwertaufnahme 13 Ergebnisse mit einer Abtastungsfrequenz von über $50\ \mathrm{kHz}$ erzielt wurden. Die minimale Messabweichung liegt bei ca. 65 %. Dabei liegt die beste Phasenverschiebung bei 0.018° (siehe Messreihe: 2, 3) und die schlechteste bei 0,1548° (siehe Messreihe: 15).

Messreihe	PMU1	Differenz zwischen PMU1 & PMU2	PMU2
	Zeit		Zeit
2	13:42:42		13:42:42
	delta t [ms]	delta T [ms]	delta t [ms]
	2,19	0,001	2,191
	Phasenwinkel [°]	delta Phasenwinkel [°]	Phasenwinkel [°]
	39,42	0,018	39,438
	Zeit		Zeit
15	13:42:55		13:42:55
	delta t [ms]	delta T [ms]	delta t [ms]
	4,117	0,0086	4,1256
	Phasenwinkel [°]	delta Phasenwinkel [°]	Phasenwinkel [°]
	65,412	0,1548	65,5668

Tabelle 5: Messreihenvergleich - minimaler Abweichungsbereich [35]

Die mittlere Messabweichung umfasst alle Zeitverschiebungen, die in dem Bereich zwischen 11 μs und 20 μs liegen. Dabei wurden bei 20 Messungen vier Messungen mit einer mittleren Abweichung erfasst. Diese Messwerte sind gelten, da sie ebenfalls mit einer Abtastungsfrequenz von $50\ \mathrm{kHz}$ vom GPIO-Pin ausgelesen wurden. Die beste Phasenverschiebung liegt bei 0,192° (siehe Messreihe: 12) und die schlechteste liegt bei 0,27° (siehe Messreihe 1). Somit weisen nur 20 % mittlere Messabweichungen auf.

Messreihe	PMU1	Differenz zwischen PMU1 & PMU2	PMU2
	Zeit		Zeit
12	13:42:52		13:42:52
	delta t [ms]	delta T [ms]	delta t [ms]
	3,634	0,011	3,653
	Phasenwinkel [°]	delta Phasenwinkel [°]	Phasenwinkel [°]
	65,412	0,192	65,754
	Zeit		Zeit
1	13:42:41		13:42:41
	delta t [ms]	delta T [ms]	delta t [ms]
	2,116	0,015	2,131
	Phasenwinkel [°]	delta Phasenwinkel [°]	Phasenwinkel [°]
	38,088	0,27	38,358

Tabelle 6: Messreihenvergleich - mittlerer Abweichungsbereich [35]

Die maximale Abweichung ist der Bereich, bei dem die Zeitverschiebungen einen Messwert aufweisen, der größer als 20 μs ist. Somit wurden die Zeitmessungen mit einer Abtastungsfrequenz von unter $50\ kHz$ erzielt. Bei 20 aufgenommen Messungen trat dreimal eine maximale Abweichung auf. Es ergibt sich eine maximale Abweichung von ca. 15 %. Die beste Phasenverschiebung entspricht dabei 0.37° (siehe Messreihe 19) und die schlechteste liegt bei ca. 0,6° (siehe Messreihe: 18).

Messreihe	PMU1	Differenz zwischen PMU1 & PMU2	PMU2
	Zeit		Zeit
19	13:42:59		13:42:59
	delta t [ms]	delta T [ms]	delta t [ms]
	4,698	0,0246	4,7226
	Phasenwinkel [°]	delta Phasenwinkel [°]	Phasenwinkel [°]
	84,564	0,378	85,006
	Zeit		Zeit
18	13:42:58		13:42:58
	delta t [ms]	delta T [ms]	delta t [ms]
	4,5132	0,0324	4,5456
	Phasenwinkel [°]	delta Phasenwinkel [°]	Phasenwinkel [°]
	81,2376	0,5832	81,8208

Tabelle 7: Messreihenvergleich - maximaler Abweichungsbereich [35]

5.1.2 Zusammenfassung

Um die resultierende Phasenverschiebung zwischen den beiden PMU-Messgeräten zu ermitteln, werden alle erfassten Zeitverschiebungen der fallenden Rechteckflanken aus der Tabelle 4 addiert und das Ergebnis gemittelt. Tabelle 9 zeigt die resultierende Phasenverschiebung zwischen den beiden PMU-Messgeräten. Es wurde eine gemittelte Zeitverschiebung von $10.0078\ ms$ berechnet, was einer Phasenverschiebung von 0,18° entspricht.

Gesamter Spannungswinkel		
PMU1	Differenz (PMU1 und PMU2)	PMU2
delta t - Mittelwert [ms]	delta T [ms]	delta t - Mittelwert [ms]
3,5078	0,0010078	3,5178
Spannungswinkel - Mittelwert [°]	delta Spannungswinkel [°]	Spannungswinkel - Mittelwert [°]
63,1413	0,180562	63,3218

Tabelle 8: Mittelwert des Spannungswinkels aus einer Messwertaufnahme [35]

5.2 Phasenverschiebung unter einer Last

Nachdem in dem vorherigen Kapitel die Phasenverschiebung zwischen den beiden entwickelten PMU-Messgeräten untersucht und das Ergebnis, dass mit einer fallenden Flankeneinstellung deutlich geringere Zeitverschiebungen erzielt wurde, werden im nächsten Schritt beide Messgeräte unter realen Netzbedingungen mit einer fallenden Flankeneinstellung getestet. Nach der Beschreibung des Messaufbaus werden mittels der Messaufnahmen die Phasenverschiebung ermittelt und untersucht.

5.2.1 Simulationsumgebung

Für die Untersuchung der Phasenverschiebung wurde als Simulationsumgebung eine Hochspannungsübertragungsleitung gewählt, die sich im Smart-City-Labor in der TH-Köln befindet. Die verwendete Hochspannungsübertragungsleitung kann mit dem folgenden Ersatzschaltbild (ESB) vereinfacht beschrieben werden.

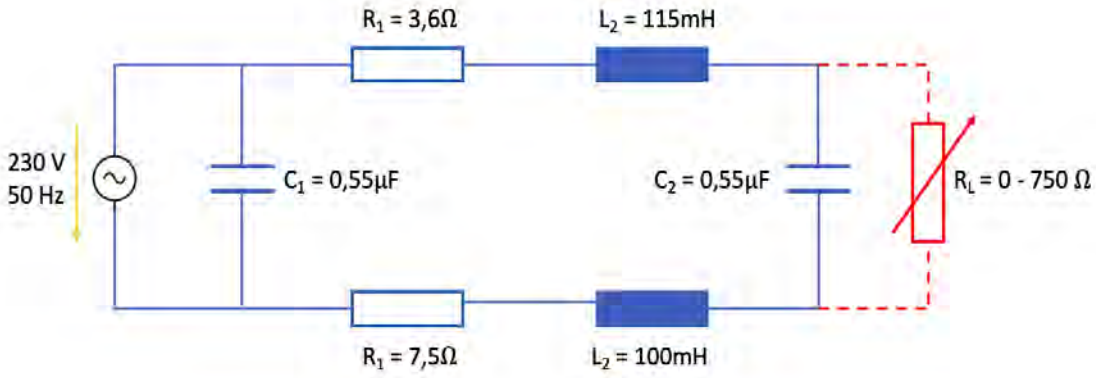

Abbildung 20:Ersatzschaltbild der Hochspannungsübertragungsleitung im Smart-City-Labor [35]

Die einphasige Übertragungsleitung in Abbildung 20 beinhaltet fest integrierte Bauteile, die nicht verändert oder ausgetauscht werden können. Durch das Zuschalten eines externen, einstellbaren und ohmschen Lastwiderstandes R_L lassen sich durch das verstellten von R_L unterschiedliche Phasenverschiebungen ermitteln. Für die Messung der Phasenverschiebung werden am Anfang der Klemme C_1 und am Ende der Klemme C_2 des ESBs jeweils ein PMU-Messgerät installiert. Bei einer Messwertaufnahme kommt es aufgrund der komplexe Leitungsimpedanz $\underline{Z}_1$, die sich zwischen den beiden Messpunkten befindet zu einer Phasenverschiebung. Um die entstehende Phasenverschiebung des Messsystems rechnerisch zu bestimmen, wird Abbildung 20 ein weiteres Mal vereinfacht zusammengefasst. Daraus folgt das ESB in Abbildung 21.

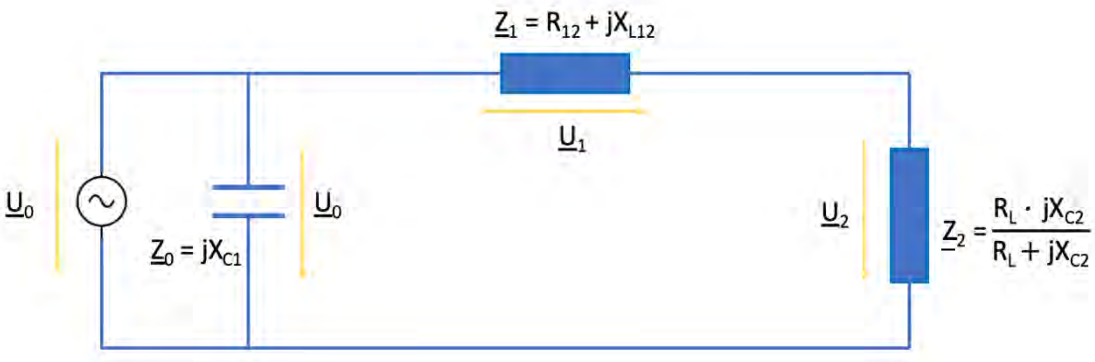

Abbildung 21: Vereinfachte Darstellung des ESB der Hochspannungsübertragungsleitung [35]

In der Abbildung 21 werden die Widerstände und Induktivitäten der Versorgungsleitung zusammengefasst, die die Leitungsimpedanz $\underline{Z}_1$ bilden. Die Kapazität C_2 und der ohmsche Widerstand R_L sind parallelgeschaltet und bilden zusammen die Impedanz $\underline{Z}_2$. Die Kapazität am Klemmenanfang wird als $\underline{Z}_0$ bezeichnet und liegt parallel zu den Impedanzen $\underline{Z}_1$ und $\underline{Z}_2$, welche wiederum in einer Reihe geschaltet sind. Mittels der komplexen Impedanzen $\underline{Z}_1$, $\underline{Z}_2$ und der komplexen Netzspannung $\underline{U}_0$ lässt sich ein Spannungsteiler bilden und die komplexe Spannung $\underline{U}_2$ berechnen. Die komplexe Spannung $\underline{U}_2$ setzt sich aus den folgenden Gleichungen zusammen:

$$\underline{Z}_1 = R_{12} + jX_{L12} = R_1 + R_2 + jX_{L1} + jX_{L2} \tag{13}$$

$$X_{L1} = 2\,\pi\,f\,L_1 \text{ und } X_{L2} = 2\,\pi\,f\,L_2 \tag{14}$$

$$\underline{Z}_2 = \frac{R_L \cdot jX_{C2}}{R_L + jX_{C2}} \tag{15}$$

$$X_{C2} = \frac{1}{2\,\pi\,f\,C_2} \tag{16}$$

$$\frac{\underline{U}_2}{\underline{U}_0} = \frac{\underline{Z}_2}{\underline{Z}_1 + \underline{Z}_2} \tag{17}$$

Um den Spannungswinkel des Messsystems zu bestimmen, wird die komplexe Spannung $\underline{U}_2$ von der kartesischen Koordinatenform (Gleichung 18) in die Polarkoordinaten (Gleichung 19) umgerechnet.

$$\underline{U}_2 = U_{Re} + jU_{im} \tag{18}$$

$$\varphi = \tan^{-1}\frac{U_{im}}{U_{Re}} \tag{19}$$

5.2.2 Simulationsmessung

Tabelle 9 zeigt die Phasenverschiebungen zwischen der Netzspannung und der Lastspannung R_L. Dabei wurde der Lastwiderstand auf zwei unterschiedliche Widerstandsgrößen eingestellt und Messergebnisse softwaretechnisch, sowie rechnerisch bestimmt.

Lastwiderstand RL [Ω]	Berechnete Phasenverschiebung [°]	Gemessene Phasenverschiebung [°]	Differenz zwischen Rechnung und Messung [°]
446	8,356	9,31932	0,963
715	5,484	6,36660	0,883

Tabelle 9: Messwertaufnahme der Phasenverschiebung mit unterschiedlichen R_L Einstellungen [35]

Die Abweichung der Phasenverschiebungen zwischen den berechneten und gemessenen Messwerten liegt bei 0,88° und 0,96°. Es gibt mehrere Ursachen, wie es zu einer solchen Abweichung zwischen der berechneten und gemessenen Phasenverschiebung kommen kann. Beispielsweise wird durch die Installation der PMU-Schaltung die Last am Ende der Klemme erhöht. Dies führt dazu, dass nicht nur ohmsch, sondern auch noch induktiv und kapazitiv belastet wird. Zusätzlich kommt es bei den Komparatorschaltungen zu minimalen Zeitverzögerungen, welche bei der Transformierung der Rechteckspannung am Sinusnulldurchgang entstehen. Für die Messaufnahme werden Verbindungsleitungen verwendet, die ebenfalls die Induktivität minimal erhöhen. Eine weitere Ursache könnten des Weiteren die eingebauten Bauteile der Hochspannungsüberragungsleitung sein. Die eingebauten Bauteile besitzen nämlich nicht immer die exakte Baugröße und weichen minimal vom angegebenen Wert ab. Zudem kann der verwendete Mikrocontroller ein zusätzlicher Grund für die Abweichung der Phasenverschiebung sein. Da es sich beim Raspberry Pi um ein komplettes System mit Prozessor, Grafikkarte und Speicher handelt, werden mehrere komplexe Programmaufgaben vom Raspberry Pi gleichzeitig ausgeführt [36]. Dadurch findet kein deterministischer Ablauf statt. In Kapitel 7 wird auf den Determinismus von Mikrocontrollern, besonders der vom Raspberry Pi, genauer darauf eingegangen.

6. Fazit

Das Ziel der Abschlussarbeit war eine kostengünstige Entwicklung eines WAM-Systems, das auf Basis einer Sinusnulldurchgangsdetektion die Phasenverschiebung ermittelt. Für die Umsetzung eines WAM-Systems werden zwei Phasor Measurement Unit Messgeräte konstruiert, die jeweils aus einem leistungsstarken Mikrocontroller, einem GPS-Zeiterfassungsmodul und einem Sinusnulldurchgangswandler bestehen. Beide PMU-Messgeräte werden mit einem hoch präzisen GPS-Zeiterfassungsmodul (GNSS-5-Click) synchronisiert. Die Besonderheit dieser GPS-Zeiterfassungsmodule ist die implementierte PPS-Funktion. Die Abweichung beider PPS-Signale liegt bei einer maximalen Zeitverzögerung von 30 Nanosekunden, weshalb das PPS-Signal als Startpunkt für die Zeitmessung der fallenden Flanke gewählt wurde. Um die Phasenverschiebung zwischen zwei positiv fallenden Rechteckflanken messen zu können, wurde für die Abschlussarbeit ein Mikrocontroller vom Progressus-Projektteam ausgewählt, der die Eigenschaft hat, mit einer hoch präzisen Abtastgeschwindigkeit, jedes eintreffende digitale Signal auszulesen. Es wurde ein leistungsstarker Raspberry Pi für die Entwicklung eines WAM-Systems ausgewählt, welcher mit seinen Eigenschaften kostengünstig und gleichzeitig flexibel ist [19].

Für die Abschlussarbeit wurden zwei Software-Programme auf dem Raspberry Pi entworfen, welche die Programme *Zeitstempel.c* und *Spannungsphasenwinkel.c* sind. Mit dem *Zeitstempel.c* Programm werden alle positiv fallenden Rechteckflanken ermittelt und mit einem hoch präzisen Zeitstempel markiert. Das Programm wird auf beiden entwickelten PMU-Messgeräten implementiert und beim Eintreffen des PPS-Signals die Messung gleichzeitig gestartet. Die serielle Kommunikation zwischen dem Raspberry Pi und GPS-Zeiterfassungsmodul funktioniert problemlos. Es werden Datenpakete vom GPS-Modul empfangen und über die serielle UART-Kommunikationsschnittstelle transferiert. Da der Raspberry Pi mit einer WLAN-Schnittstelle ausgerüstet ist, wurde für eine gemeinsame Kommunikation zwischen den beiden PMU-Messgeräten, ein Samba-Server eingerichtet, der die ähnliche Funktionalität eines SPC-Systems hat. Die Server-Kommunikation (Samba-Server) ist zuverlässig eingerichtet worden und ist problemlos von beiden Messgeräten nutzbar. Alle erfassten Messdaten werden auf der SD-Karte abgespeichert und auf dem Serverordner *„share“* für jedes PMU-Messgerät zugänglich gemacht. Zusätzlich wurde für die Ermittlung des Spannungswinkels das Programm *Spannungsphasenwinkel.c* entwickelt, welches die Funktionalität eines PDC-Systems hat. Mit dem *Spannungsphasenwinkel.c* Programm werden die aufgezeichneten Messdaten auf Ausreißer, Phasenübergänge zwischen 359° und 0° untersucht und die Phasenverschiebung des Versorgungsnetzes bestimmt. Beide PMU-Messgeräte sind mit dem Programm ausgerüstet und in der Lage die Spannungswinkelberechnung auszuführen

Die Ergebnisse bestätigen, dass die entwickelte Software zur digitalen Flankenmessung verwendet werden kann. Alle aufgezeichneten Messdaten vom Programm *Zeitstempel.c* weisen alle eine gute und stabile Kontinuität auf [19]. Mittels der implementierten

wiringPi-Bibliothek, konnte eine GPIO Auslesegeschwindigkeit von $50\ \mathrm{kHz}$ unter Verwendung von C mit $f_{wiringPi,C} = 4{,}1\ \mathrm{MHz}$ erzielt werden [25]. Unter anderem wurde das Programm *Zeitstempel.c* mit einer steigenden und fallenden Flankeneinstellung gestartet und getestet. Dabei wurde mit der fallenden Flankendetektion eine Phasenverschiebung von 0,18° erzielt. Von 20 Messreihen wurden 17 Zeitverzögerungen mit einer ausreichenden Messpräzision ermittelt, die unter einer Phasenverschiebung zwischen 0,018° bis 0,27° liegen. Jedoch wurden auch drei Messwerte aufgezeichnet, die deutlich über dem gewünschten Messergebnis sind. Die Phasenverschiebung liegt bei diesen Messwerten in dem Bereich von 0,4° bis 0,6°. Mögliche Gründe für die Messabweichung werden in Kapitel 7 beschrieben.

Abschließend lässt sich sagen, dass ein kostengünstiges WAM-System, welches aus zwei PMU-Messgeräten, einem PDC- und SPC-System besteht, mit weniger als 200 € umgesetzt werden konnte. Die entworfenen Software-Programme können für die Messung der Spannungswinkeln eingesetzt werden. Die im Anfang gestellte Zielsetzung konnte mit der Abschlussarbeit nachgewiesen und umgesetzt werden.

7. Ausblick und Diskussion

Im Laufe der WAM-Systementwicklung sind unterschiedliche Fehler entstanden. Bei der Untersuchung der Zeitverzögerungen von fallenden Flanken wurde messtechnisch eine Genauigkeit von 1 μs bis 8 μs zwischen den beiden Komparatorschaltungen aufgenommen [28]. Softwaretechnisch hingegen liegt die Messgenauigkeit zwischen 1 μs bis 32 μs. Die maximale Messungenauigkeit der Zeitverzögerung beträgt drei von 20 Messreihen, welche eine Phasenverschiebung zwischen 0,4° und 0,6° entspricht. Der Grund für eine solch große Messabweichung liegt nicht an den entwickelten Programmen, sondern am vorgegebenen Mikrocontroller.

Der Linux-basierte Raspberry Pi ist nicht als Echtzeit-Betriebssystem konzipiert worden. Das bedeutet, dass der Raspberry Pi die auszuführenden Echtzeit-Programme nicht als seine höchste Priorität ansetzt. Darunter fallen unter anderem das Codieren von Signalverarbeitungen. Durch das Abfragen der GPIO-Pins in der Endlosschleife, bleiben die CPU-Ressourcen blockiert, welches die Reaktionszeit zwischen den beiden ausgelösten Interrupts (PPS-Interrupt und Flanken-Interrupt) erhöht. Je mehr Interrupts in dem Programmcode konfiguriert werden, desto höher ist die Verzögerungszeit der konfigurierten Interrupts [27]. Dadurch nimmt der Determinismus ab und die Genauigkeit, der zeitlich zu messenden Flanken wird unpräziser. Um die Messgenauigkeit der fallenden Flanken der Rechtecksignale mit einer höheren Interrupt Präzision messen zu können, wird empfohlen ein deterministischen Mikrocontroller zu verwenden. Dieser soll die Echtzeitanforderung erfüllen, wenn mehrere Interrupts gleichzeitig ausgelöst werden [26]. Mit einem deterministischen Mikrocontroller werden zeitliche Grenzen gesetzt. Dabei werden Systemaufrufe und hohe Zeitverzögerungen der Interrupts deutlich verbessert.

Der Mikrocontroller sollte dabei mindestens zwei Auslesepins besitzen, um digitale Signale verarbeiten zu können. Ein Pin soll die steigende Flanke des PPS-Signals auslesen. Zusätzlich wird ein weiterer Pin benötigt, der die fallenden Flanken der Rechtecksignale ausließt, die von der Komparatorschaltung am Nulldurchgang transformiert werden. Ebenso soll der Mikrocontroller die Eigenschaft mitbringen über eine serielle UART-Kommunikationsschnittstelle die GPRMC-Datenpakete, welche vom GPS-Zeiterfassungsmodul empfangen werden, mit einer hohen Genauigkeit empfangen und bearbeiten zu können. Da alle Messwerte in ein CSV-Dokument auf die eingebaute SD-Karte des Raspberry Pis abgesichert werden, wird eine weitere Schnittstelle zwischen dem Mikrocontroller und dem Raspberry Pi benötigt. Die aufgenommenen Messwerte werden dann mittels der I^2C-Schnittstelle vom Mikrocontroller auf dem Raspberry Pi gesendet. Diese werden dann im Anschluss auf die SD-Karte des Raspberry Pis abgespeichert. Um zwischen diesen beiden Peripheriegeräten ein I^2C-Kommunikationsschnittstelle herzustellen, sollte die Softwaredatei *Zeitstempel.c* an der Codestelle nach der zeitkritischen Messung und Erstellung der Zeitstempel angepasst werden. Alternativ kann der deterministische Mikrocontroller über den USB-Anschluss mit dem Raspberry Pi verbunden werden. Der deterministische Mikrocontroller sollte zudem einen 3,3 V Stromanschluss haben der maximal einen Strom von $50\,\mathrm{mA}$ liefert, um den GNSS-5-Click

ansteuern zu können. Es wird empfohlen den Mikrocontroller „*Raspberry Pi Pico*“ zu wählen, da er zum einen eine nahe liegende hardware- und softwaretechnische Übereinstimmung mit dem Raspberry Pi hat. Zum anderen ist er auf Grund seiner deterministischen Eigenschaft ein echtzeitfähiger Mikrocontroller und kein Mini-Computer [36].

Ein weiterer Fehler, der nicht behoben werden konnte, ist die Absicherung der aufgezeichneten Messdaten bei einem Stromausfall. Dazu soll eine Batterie eingebaut werden, die das Messgerät weiterhin mit Strom versorgt. Ein Stromausfall würde zu einer lücken- und fehlerhaften erstellten CVS-Messdatei führen.

Abbildungsverzeichnis

Abbildung 1: System- und Kommunikationsaufbau eines Wide Area Monitoring Systems - In Anlehnung an [38] 5

Abbildung 2: Phasenverschiebung zwischen zwei Sinussignalen [35] 7

Abbildung 3: Vereinfachte Darstellung einer Versorgungsleitung - In Anlehnung an [29] 9

Abbildung 4: Vereinfachte Darstellung der Leitungsimpedanz - In Anlehnung an [30] 10

Abbildung 5: Zeigerdiagramm bei einer ohmsch-induktiven Belastung - In Anlehnung an [29] 10

Abbildung 6: Unterschiedliche PMU-Schaltungen für das zu entwickelnde WAMS Messsystem [35] 12

Abbildung 7: Entwicklungsplatine – Raspberry Pi 3 Model B+ [35] 13

Abbildung 8: Zeitsynchronisiertes GPS Modul – GNSS-5-Click [35] 15

Abbildung 9: Startpunkt der Zeitmessung durch das PPS-Signal, Rechtecktransformierung am Sinusnulldurchgang [35] 17

Abbildung 10: UART-Schnittstellenverbindung zwischen Raspberry Pi und GNSS-5-Click [35] 19

Abbildung 11: Prototyp der Komparatorschaltung [35] 20

Abbildung 12: Zentrale Vorkonfiguration des Raspberry Pis - In Anlehnung an [24]. 21

Abbildung 13: Einstellung der Freigabe des Samba-Servers auf dem Raspberry Pi - In Anlehnung an [24] 22

Abbildung 14: Schnittstellenschaltplan des entwickelten Spannungswinkelmessgeräts mit implementierter Komparatorschaltung [35] 23

Abbildung 15: Organigramm – Ablauf und Programmstruktur der Zeitstempel.c Datei – In Anlehnung an [19] 25

Abbildung 16: Organigramm - Ablauf und Struktur der Spannungsphasenwinkel.c Datei [35] 32

Abbildung 17: Aufgabeverdeutlichung der Funktion *Zeitstempel_String()* [35] 37

Abbildung 18: Schaltungsaufbau zur Zeitverschiebungsmessung zwischen beiden PMU-Messgeräten [35] 46

Abbildung 19: Zeitverschiebungen zwischen der steigenden und fallenden Rechteckflanke [35] 48

Abbildung 20:Ersatzschaltbild der Hochspannungsübertragungsleitung im Smart-City-Labor [35] 51

Abbildung 21: Vereinfachte Darstellung des ESB der Hochspannungsübertragungsleitung [35] 52

Abbildung 22: Einrichtung - Softwarepaket für Samba-Server [35] 67

Abbildung 23: Samba-Server Freigabe wird geöffnet [35] 67

Abbildung 24: Einrichtung des Freigabeordners "share" [35] 68

Abbildung 25: Systemneustart mit dem Funktionsbefehl „sudo service smbd restart“ [35] 68

Abbildung 26: GPS-Antenne [35] 69

Abbildung 27: Messaufbau des Spannungswinkelmessgerät – Prototypentwicklung des PMU-Messgerätes [35] 69

Abbildung 28: Einrichtung der Entwicklungsumgebung auf dem Raspberry Pi [35] .. 70

Abbildung 29: Programmieroberfläche - Softwareprogramm: Code::Block IDE Version. 17.12 [35] 70

Abbildung 30: Ordnerverzeichnis der Zeitmessungsdatei – Zeitstempel.c [35] 71

Abbildung 31: Zeitstempel.c Datei wird in die Entwicklungsumgebung *Codeblocks* implementiert [35] 71

Abbildung 32: Das GNSS-5-Click Modul empfängt kein PPS-Signal [35] 72

Abbildung 33: Das GNSS-5-Click hat das PPS-Signal empfangen (Markierung des Startpunktes der Zeitmessung) [35] 72

Abbildung 34: Ausführung der Zeitstempel.c Datei - Startpunkt der Zeitmessung mit dem PMU-Messgerät [35] 73

Abbildung 35: Die Zeitstempel.c erstellte CSV-Datei *„PMU1-MW.csv“* wird in den Ordern Zeitstempel.c abgespeichert [35] 73

Abbildung 36: Samba-Verzeichnis – Speicherort der Messdaten beider PMU-Messgeräte abgespeichert [35] 74

Abbildung 37: Spannungsphasenwinkel.c Datei im Filesystem Verzeichnis vorbereiten [35] 74

Abbildung 38: Die Spannungsphasenwinkel.c Datei wird in die Entwicklungsumgebung *Codeblock* implementiert [35] 75

Abbildung 39: Ausführung der Spannungsphasenwinkel.c Datei - Startpunkt des PDC-Systems [35] 75

Abbildung 40: Ergebnisse bei der beiden Messdateien gespeichert in "Phasenwinkel.csv" [35] 76

Abbildung 41: Dateienausgabe über das Terminal [35] 76

Tabellenverzeichnis

Tabelle 1: Technische Unterschiede der PMU-Messgeräte der Firma Siemens [32], [33 S. 9-17] 6

Tabelle 2: Unterschied zwischen beiden bekannten Überwachungsmesssystemen SCADA und PMU [10 S. 12]. [11], [12], [13], [31 S. 4] 8

Tabelle 3: Betriebskonfigurationsmöglichkeiten der UART-Schnittstelle [35] 18

Tabelle 4: Messwertaufnahme - gemittelte Zeitverschiebungen zwischen steigenden und fallenden Rechteckflanken [35] 47

Tabelle 5: Messreihenvergleich - minimaler Abweichungsbereich [35] 49

Tabelle 6: Messreihenvergleich - mittlerer Abweichungsbereich [35] 49

Tabelle 7: Messreihenvergleich - maximaler Abweichungsbereich [35] 50

Tabelle 8: Mittelwert des Spannungswinkels aus einer Messwertaufnahme [35] 50

Tabelle 9: Messwertaufnahme der Phasenverschiebung mit unterschiedlichen R_L Einstellungen [35] 53

Formelverzeichnis

Formelbezeichnung	Seite:
1. Grundformel für den Phasenwinkel (rad)	12
2. Kreisfrequenz-Funktion	12
3. Phasenwinkel in Abhängigkeit zur Frequenz oder Periode	12
4. Übersetzungsfaktor Rad in Grad	12
5. Resultierender Spannungswinkelformel (deg)	12
6. Komplexe Scheinleistung eines Netzstranges	12
7. Komplexe Spannung $\underline{U}_i$	12
8. Komplexe Spannung und $\underline{U}_j$	12
9. Zeitberechnung für den Zeitstempel	15
10. Grundformel für die Zeitverschiebungsberechnung	28
11. Grundformel für die Frequenzberechnung	28
12. Mittelwertberechnung der Zeitverschiebungen	42
13. Komplexe Impedanz $\underline{Z}_1$ der Hochspannungsübertragungsleitung	52
14. Blindwiderstand, X_{L1} und X_{L2}	52
15. Komplexe Impedanz $\underline{Z}_2$ der Hochspannungsübertragungsleitung	52
16. Blindwiderstand, X_{C2}	52
17. Komplexer Spannungsteiler	52
18. Binomialdarstellung der Spannung $\underline{U}_2$	52
19. Phasenwinkel	52

Abkürzungsverzeichnis

Abkürzung	Erklärung
GPS	Global Positioning System
GNSS	Global Navigation Satellit System
CSV	Character Seperated Values
WAMS	Wide Area Monitoring System
I^2C	Inter Integrated Circuit
UART	Universal Asynchronous Receiver Transmitter
RS-232	Recommended Standard 232
SPI	Serial Peripheral Interface
GPIO	General Purpose Input/Output
GND	Ground
USB	Universal Serial Bus
HDMI	High Definition Multimedia Interface
PPS	Pulse-per-Second
NMEA	National Marine Electronics Association
SD-Karte	Secure Digital Memory Card
GPGGA	Global Positioning System System Fix Data
GPGSA	GPS DOP and active satellites
GPRMC	Recommended minimum specific GPS/Transit data
GPVTG	Track Made Good and Ground Speed
GPZDA	Date and Time
UTC	Coordinated Universal Time
TxD	Transmit Data
RxD	Receiver Data
PMU	Phasor Measurement Unit
PDC	Phasor Data Concentrator
µs	Mikrosekunden
Ms	Millisekunden
Hz	Hertz

Literaturverzeichnis

[1] PROGRESSUS, Highly efficient and trustworthy electronics, components and systems for the next generation energy supply infrastructure, [Online]. Verfügbar unter: https://www.th-koeln.de/anlagen-energie-und-maschinensysteme/progressus_76058.php, 2020

[2] Eberhard Waffenschmidt 100% Erneuerbare Energie, Projekt – Netz für Erneuerbare Energie, PROGRESSUS: Elektroniksysteme für die Energieversorgungsinfrastruktur der nächsten Generation, [Online]. http://www.100pro-erneuerbare.com/projekte/2020-04_Progressus/progressus.htm, 2020

[3] Dr. Martin Heidl, Prof. Dr. Günther Brauner, „Störungsortung im Übertagungsnetz durch intelligentes Wide Area Monitoring“, [Online] https://publik.tuwien.ac.at/files/PubDat_183995.pdf, TU Wien, Institut für Elektrische Anlagen und Energiewirtschaft, 2009

[4] Dr. -Ing. Klaus von Sengbusch, „Wide Area Monitoring System – aktuelle Erfahrung und zukünftige Anwendungsbereiche“, [Online] https://docplayer.org/14031691-Dr-ing-klaus-von-sengbusch-wide-area-monitoring-systeme-aktuelle-erfahrungen-und-zukuenftige-anwendungsbereiche-abb-group-1-14-may-07.html, Mai 2007, (Zugriff am: 31. August. 2021)

[5] Bob Shively, Enerdynamic President and Lead Facilitator, The Energy Education Experts, “What Is a Phasor Measurement Unit and How Does it Make the Grid More Reliable?”, [Online] https://www.enerdynamics.com/Energy-Currents_Blog/What-Is-a-Phasor-Measurement-Unit-and-How-Does-it-Make-the-Grid-More-Reliable.aspx, (Zugriff am: 18. August. 2021)

[6] ABB Schweiz AG, Wide Area Measurement, Monitoring, Protection and Control, [Online] https://library.e.abb.com/public/57e2e097cfbb671ec1257b0c00552696/741_PSGuard_D_04.7.pdf, Schweiz, 2004, (Zugriff am: 31. August. 2021)

[7] American Recovery and Reinvestment Act of 2009, “Deployment of Synchrophasor: Decisions and Cost Impacts”, [Online] https://www.naspi.org/sites/default/files/reference_documents/34.pdf, September 2014, (Zugriff am: 31. August. 2021)

[8] Malte Seldschopf, „Entwicklung eines Messsystems zur Bestimmung des Phasenwinkels der Netzspannung“, Technische Hochschule Köln, 2019, (Zugriff am: 28. August. 2021)

[9] IEEE Power and Energy Society, “IEEE Std C37-118.2 2011 for Synchrophasor Data Transfer for Power Systems”, IEEE Standards Association, New York, 2011

[10] Marc Richter, “PMU-basierte Zustandsabschätzung in Smart Distribution”, [Online] https://opendata.uni-halle.de/bitstream/1981185920/12143/1/Dissertation_M.Richter_MAFO_2016.pdf, Magdeburg 2016, (Zugriff am: 31. August. 2021)

[11] K. Das, D. P. Seetharam, R. K. Reddi and K. A. Sinha, "Real-time Hybrid State Estimation Incorporating SCADA and PMU Measurements," in Proc. of 2012 IEEE ISGT Europe, Berlin, Germany, 2012.

[12] V. Murugesan, Y. Chakhchoukh, V. Vittal, G. T. Heydt, N. Logic and S. Sturgill, “PMU Data Buffering for Power System State Estimators”, *IEEE Power and Energy Technology Systems Journal*, pp.94-102, 2015

[13] Andreas Götz, Zukünftige Belastung von Niederspannungsnetzen unter besonderer Berücksichtigung der Elektromobilität, Dissertation, Chemnitz, 2016

[14] Raspberrypi.org, Raspberry Pi 3 Model B+, [Online], Verfügbar unter: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/, (Zugriff am: 7. August. 2021)

[15] Raspberrypi.org, Raspberry Pi 4 Model B, [Online], Verfügbar unter: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/, (Zugriff am: 7. August. 2021)

[16] Elektronik Kompedium, Raspberry Pi: WLAN-Probleme lösen, [Online], Verfügbar unter: https://www.elektronik-kompendium.de/sites/raspberry-pi/2202041.htm, (Zugriff am: 7. August. 2021)

[17] MIKROE Time-saving embedded tools, GNSS-5-Click, [Online], Verfügbar unter: https://www.mikroe.com/gnss-5-click, (Zugriff am: 7. August. 2021)

[18] NMEA.de Daten an Board, NMEA Datensatz RMC, [Online], Verfügbar unter: http://nmea.de/test/data/da-rmc.html, (Zugriff am: 7. August. 2021)

[19] Martin Gut, Praxisprojekt: Softwareentwicklung zu einem GPS-synchronisierten Phasor-Measurement-Unit auf Basis einer Spannungsnulldurchgangsdetektion, Köln, 2021

[20] Elsys AG, GPS Messungs-Synchronisierung, [Online], Verfügbar unter: https://www.elsys-instruments.com/de/produkte/gps_synchronisation.php, (Zugriff am: 7. August. 2021)

[21] Die UART Schnittstelle, Aufbau der UART Schnittstelle, [Online], Verfügbar unter: http://www.mathe-mit-methode.com/schlaufuchs_web/elektrotechnik/mikrocontroller_lernmaterial/microcontroller_allgemein/mikrocontroller_ext_hardware/mikrocontroller_uart.html, (Zugriff am: 7. August. 2021)

[22] ElectronicWings, Raspberry Pi UART Communication using Python and C, [Online], https://www.electronicwings.com/raspberry-pi/raspberry-pi-uart-communication-, (Zugriff am: 7. August. 2021)

[23] ElectronicTurtorial, OPV-Komparator, [Online], https://www.electronics-tutorials.ws/de/operationsverstarker/opamp-komparator.html, (Zugriff am: 7. August. 2021)

[24] ElektronikKompendium, Samba-Freigabe auf dem Raspberry Pi einrichten, [Online], https://www.elektronik-kompendium.de/sites/raspberry-pi/2007071.htm, (Zugriff am: 7. August. 2021)

[25] Code and Life, Brenchmarking Raspberry Pi GPIO Speed, [Online], Verfügbar unter: https://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/, (Zugriff am: 8. August. 2021)

[26] microcontroller.net, stationäre Drehmomentmessung, [Online], Verfügbar unter: https://www.mikrocontroller.net/topic/448579 (Zugriff am: 23. August. 2021)

[27] Elektronik Praxis, Leistungsgrenzen bei Microcontrollern überschreiten, [Online], Verfügbar unter: https://www.elektronikpraxis.vogel.de/leistungsgrenzen-bei-mikrocontrollern-ueberschreiten-a-263091/?p=2 (Zugriff am: 23. August. 2021)

[28] Tim Schäfer, Masterarbeit Elektrotechnik – Elektrische Energietechnik, Entwurf einer Schaltung zur GPS-synchronisierten Spannungsphasenwinkelmessung in großflächigen, elektrischen Netzen, April 2021

[29] Prof. Dr. Eberhard Waffenschmidt, Vorlesung zum Bachelor-Studium TH-Köln WS 2020/21, Energieverteilung: Netzwerke berechnen: Netzstrahl, [Online]. Verfügbar unter: https://ilias.th-koeln.de/goto.php?target=file_1728274_download&client_id=ILIAS_FH_Koeln, (Zugriff am: 23. August. 2021)

[30] Datei: Längs und Querspannungsabfall.svg, Quelle: Eigenes Werk, [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Datei:L%C3%A4ngs_und_Querspannungsabfall.svg, (Zugriff am: 20. August. 2021)

[31] Rohini Radip Haridas, Synchrophasor Measruement Technology in Electrical Power System, [Online], Verfügbar unter: https://www.ijert.org/research/synchrophasor-measurement-technology-in-electrical-power-system-IJERTV2IS60891.pdf (Zugriff am: 29. August. 2021), Department of Electrical Engineering, Amravati University, Maharashtra, India

[32] SIEMENS, Phasor Measuremtn Unit (PMU) mit SIPROTEC 5, [Online]. Verfügbar unter: [Online]. Verfügbar unter: https://ilias.th-koeln.de/goto.php?target=file_1728274_download&client_id=ILIAS_FH_Koeln, (Zugriff am: 10. September. 2021)

[33] Digitale Störschreiber SIMEAS R und SIMEAS R-PMU, Energieautomation Katalog SR 10.1.1 – 2019, [Online]. Verfügbar unter: https://www.siemens.com/download?DLA03_1212 (Zugriff am: 10. September. 2021)

[34] D. M. Laverty, R. J. Best, P. Brogan, I. Al Khatib, L. Vanfretti and D. J. Morrow, "The OpenPMU Platform for Open-Source Phasor Measurements," in IEEE Transactions on Instrumentation and Measurement, vol. 62, no. 4, pp. 701-709, April 2013

[35] Martin Gut, Eigene Darstellung – 2021

[36] Raspberry Pi Pico: Programmieren mit dem günstigen Mikrocontroller, [Online]. Verfügbar unter: https://tutorials-raspberrypi.de/raspberry-pi-pico-mikrocontroller-programmieren/, (Zugriff am: 14. September. 2021)

[37] Markus Wache, Eiq Ea, Wide area monitoring with Phasor Measurement Data, [Online]. Verfügbar unter: https://www.researchgate.net/publication/254012329_Wide_area_monitoring_with_Phasor_Measurement_Data, (Zugriff am: 16. September. 2021)

[38] Bhargav Appasani and Dusmanta Kumar Mohanta, A review on synchrophasor communication system: communication technologie, standards and applications, [Online]. Verfügbar unter: https://pcmp.springeropen.com/articles/10.1186/s41601-018-0110-4, (Zugriff am: 23. September. 2021)

[39] ET-Tutorials.de, Spannungsfall auf Leitungen, [Online]. Verfügbar unter: https://et-tutorials.de/15310/spannungsfall-auf-leitungen/, (Zugriff am: 24. September. 2021)

[40] Elektronik Kompendium, Raspberry Pi OS: Grundkonfiguration auf der Kommandozeile, [Online]. Verfügbar unter: https://www.elektronik-kompendium.de/sites/raspberry-pi/1906291.htm, (Zugriff am: 25. September. 2021)

Anhang

Anhang 1: Einstellung und Freigabe des Samba-Servers

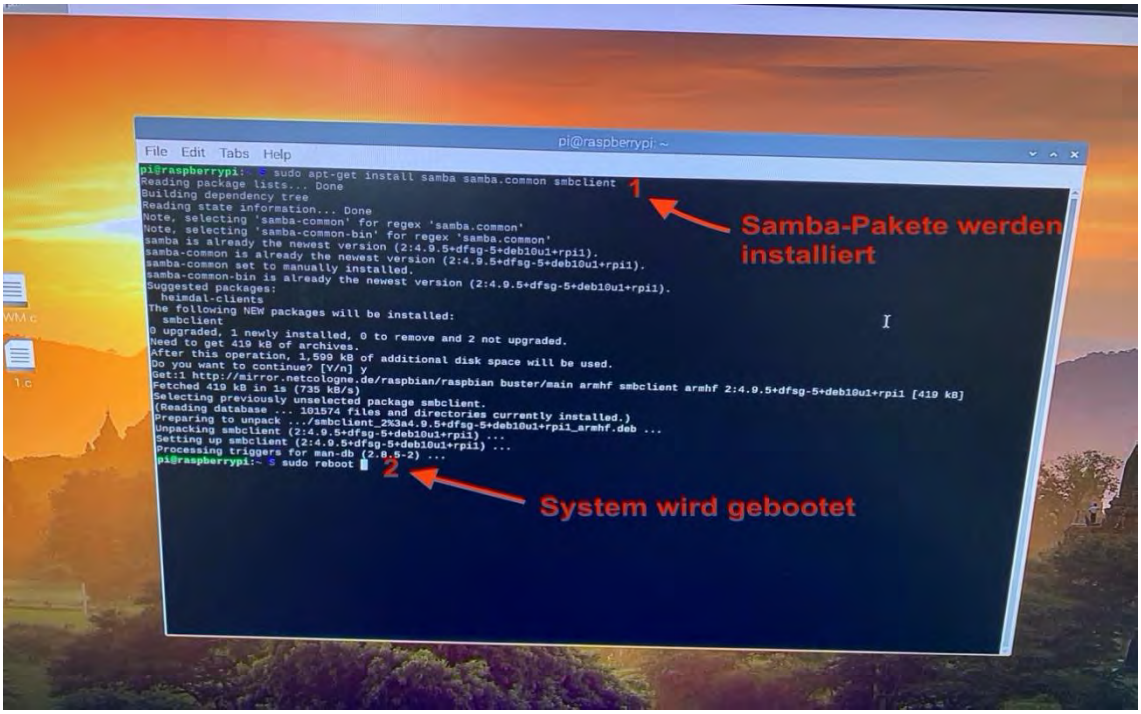

Abbildung 22: Einrichtung - Softwarepaket für Samba-Server [35]

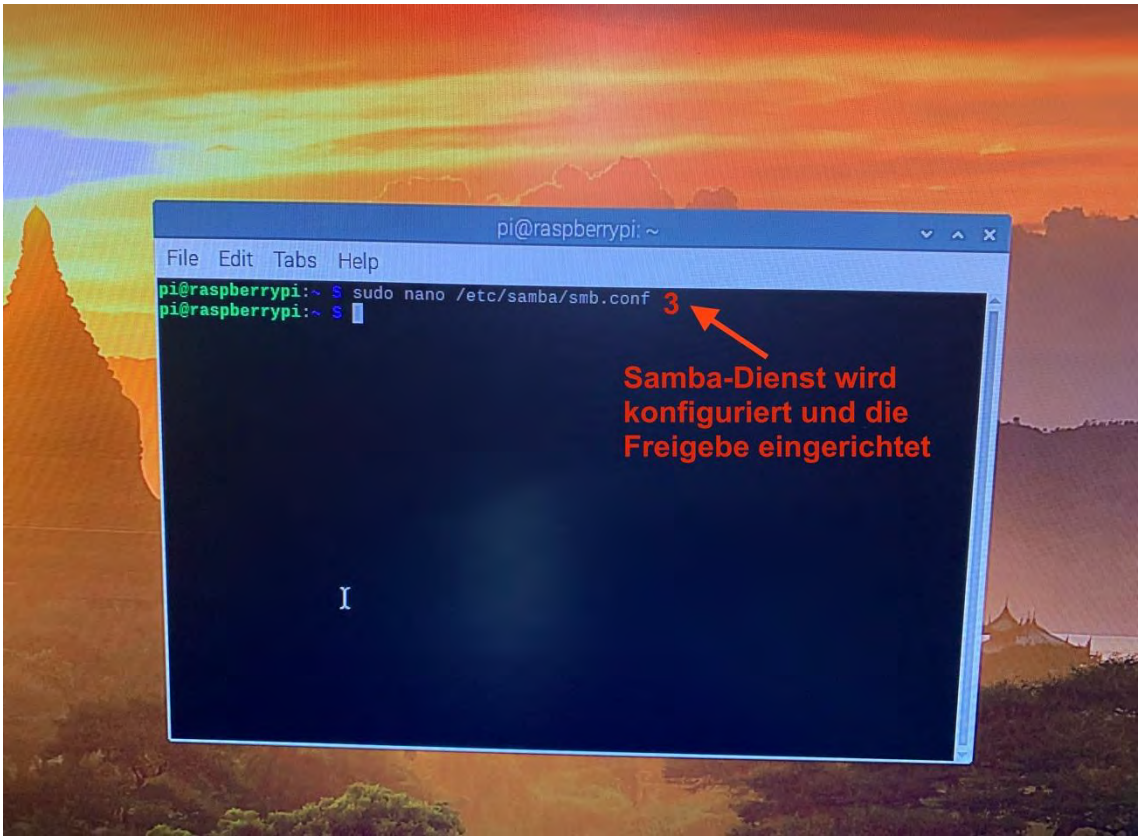

Abbildung 23: Samba-Server Freigabe wird geöffnet [35]

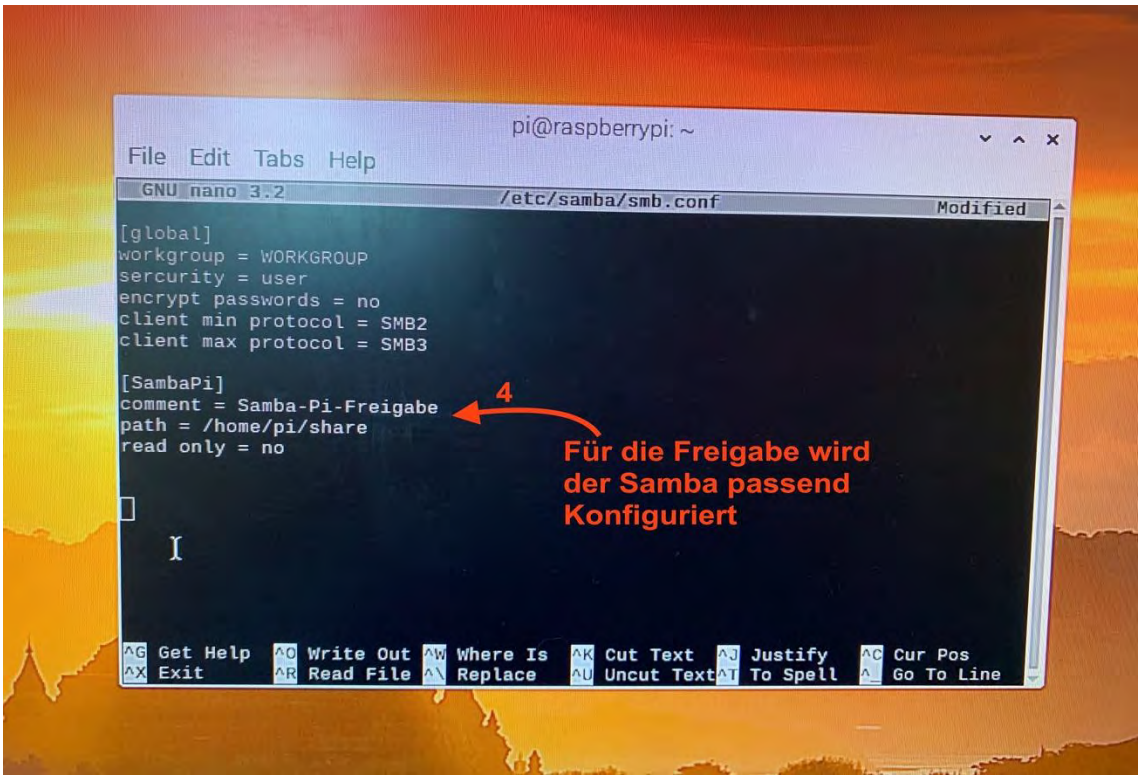

Abbildung 24: Einrichtung des Freigabeordners "share" [35]

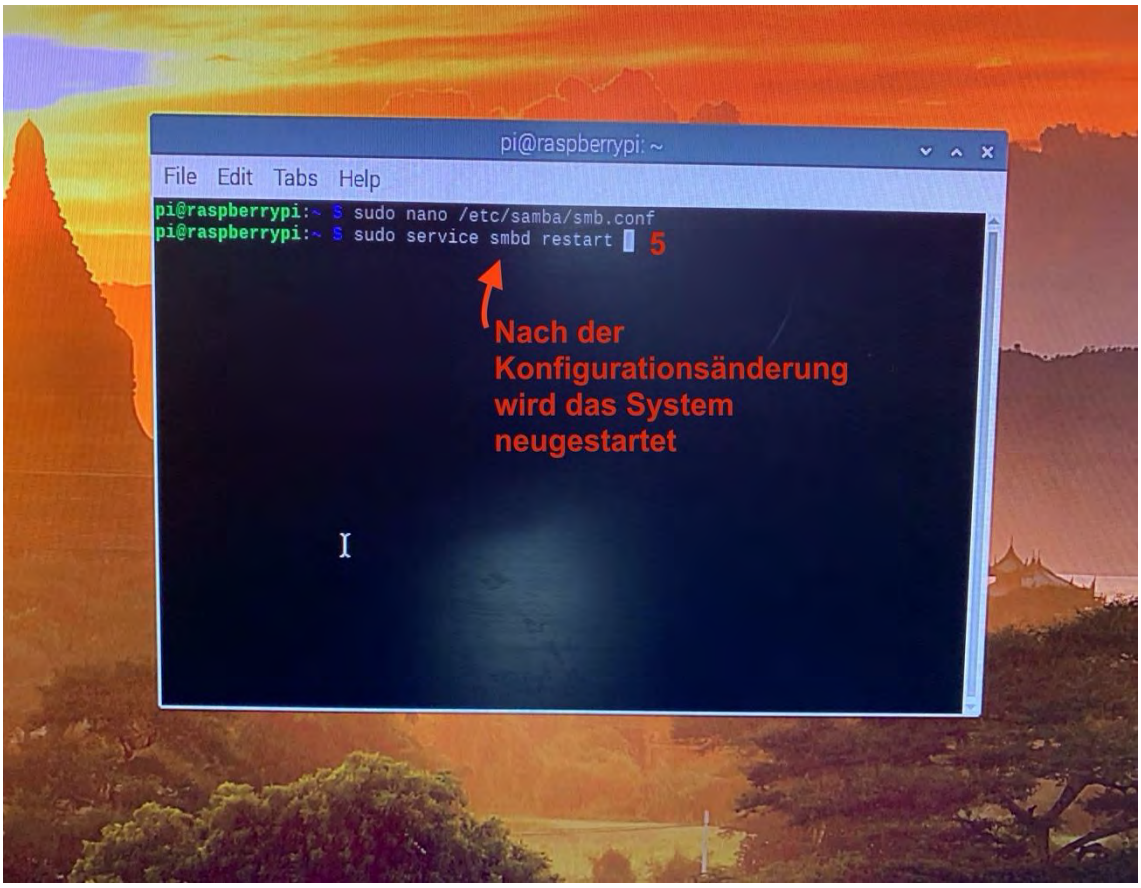

Abbildung 25: Systemneustart mit dem Funktionsbefehl „sudo service smbd restart“ [35]

Anhang 2: Prototypentwicklung eines PMU-Messgerätes für das WAMS

Abbildung 26: GPS-Antenne [35]

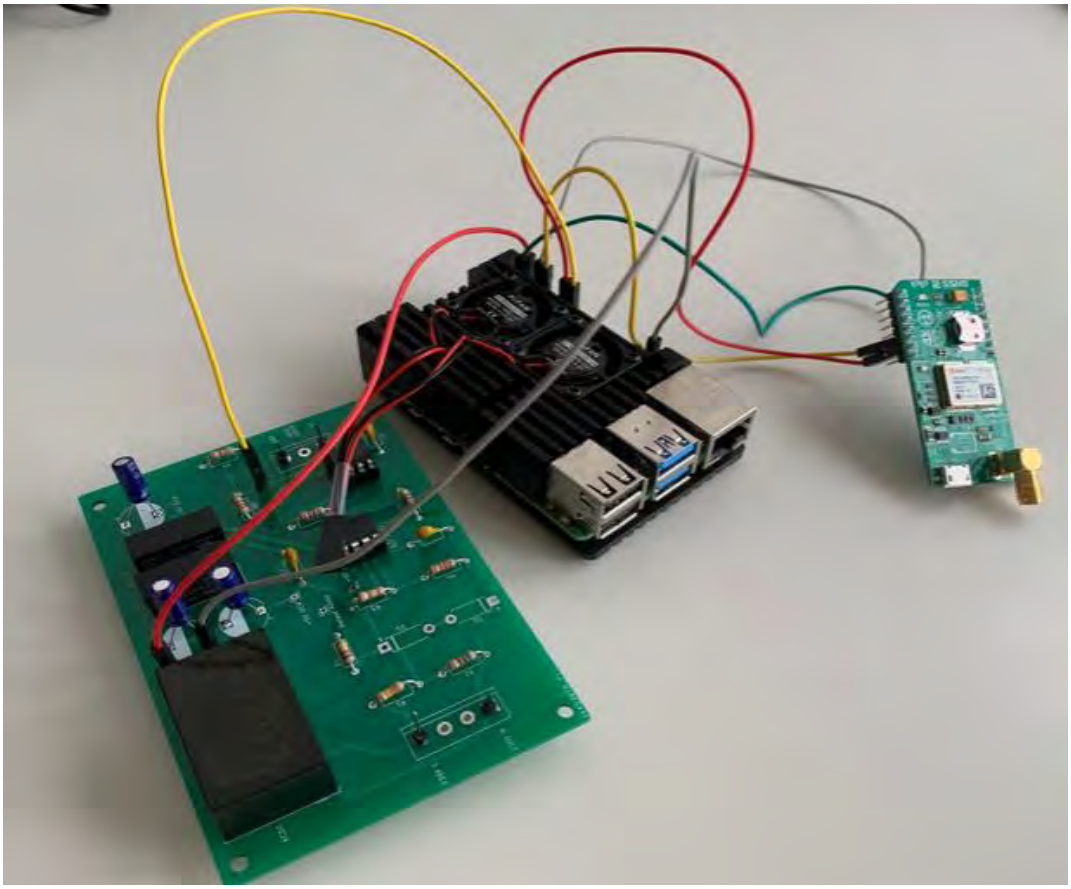

Abbildung 27: Messaufbau des Spannungswinkelmessgerät – Prototypentwicklung des PMU-Messgerätes [35]

Anhang 3: Anleitung zur Inbetriebnahme des WAMS

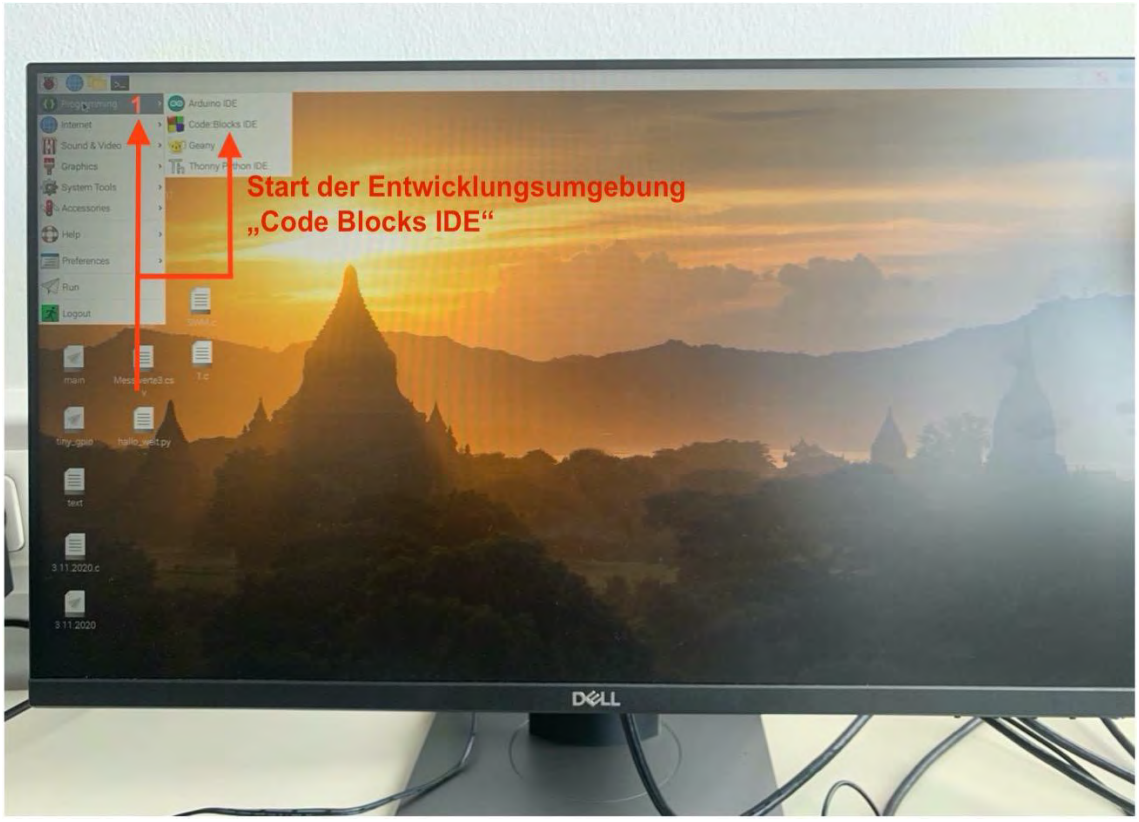

Abbildung 28: Einrichtung der Entwicklungsumgebung auf dem Raspberry Pi [35]

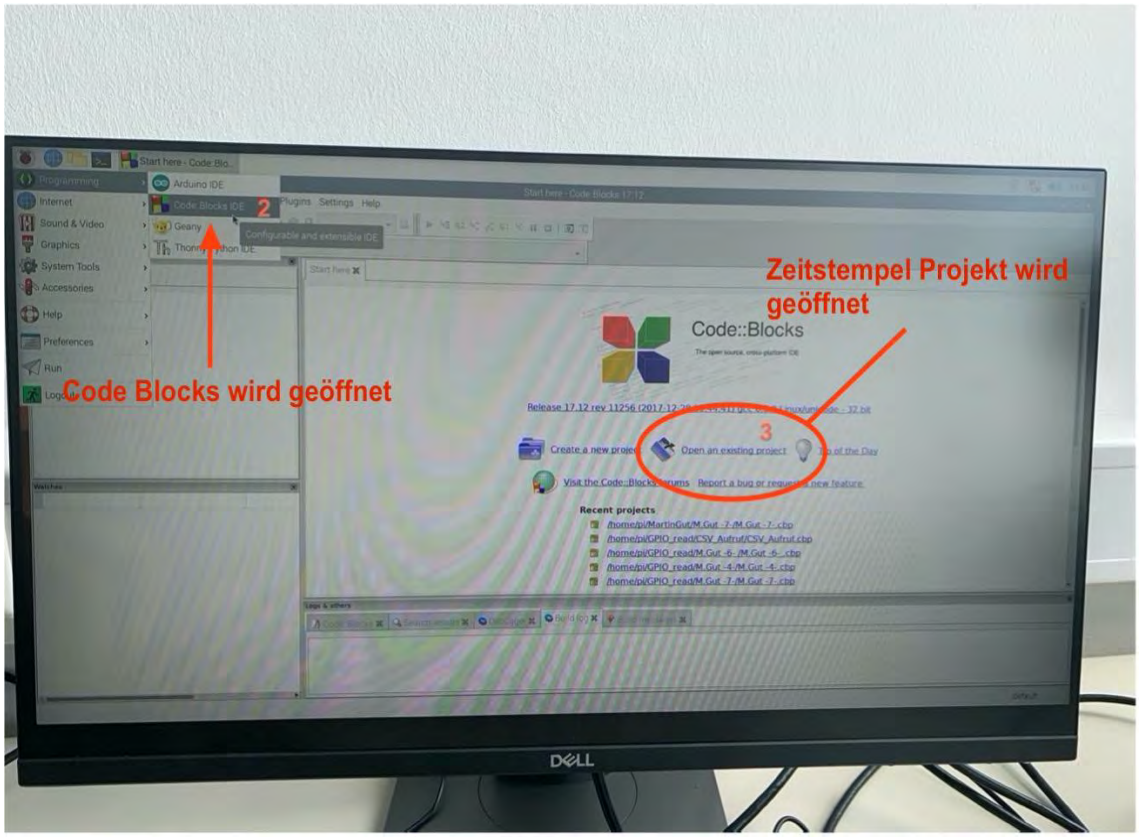

Abbildung 29: Programmieroberfläche - Softwareprogramm: Code::Block IDE Version. 17.12 [35]

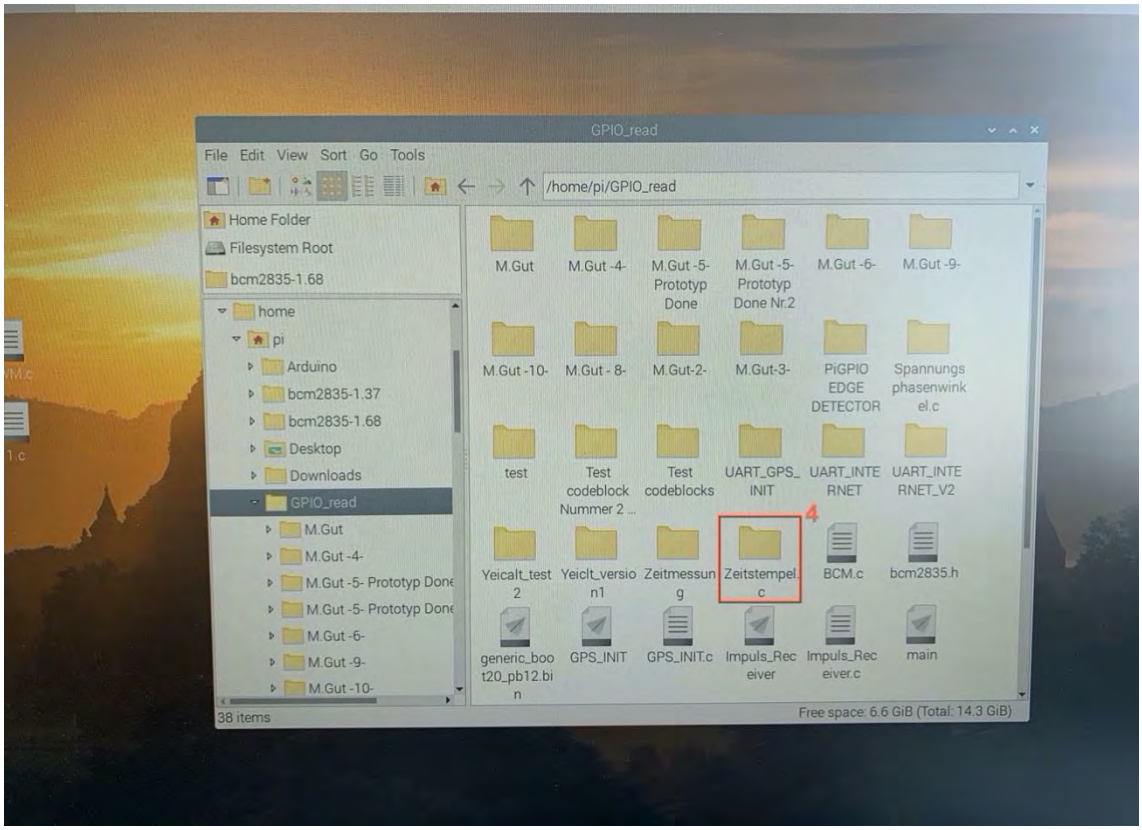

Abbildung 30: Ordnerverzeichnis der Zeitmessungsdatei – Zeitstempel.c [35]

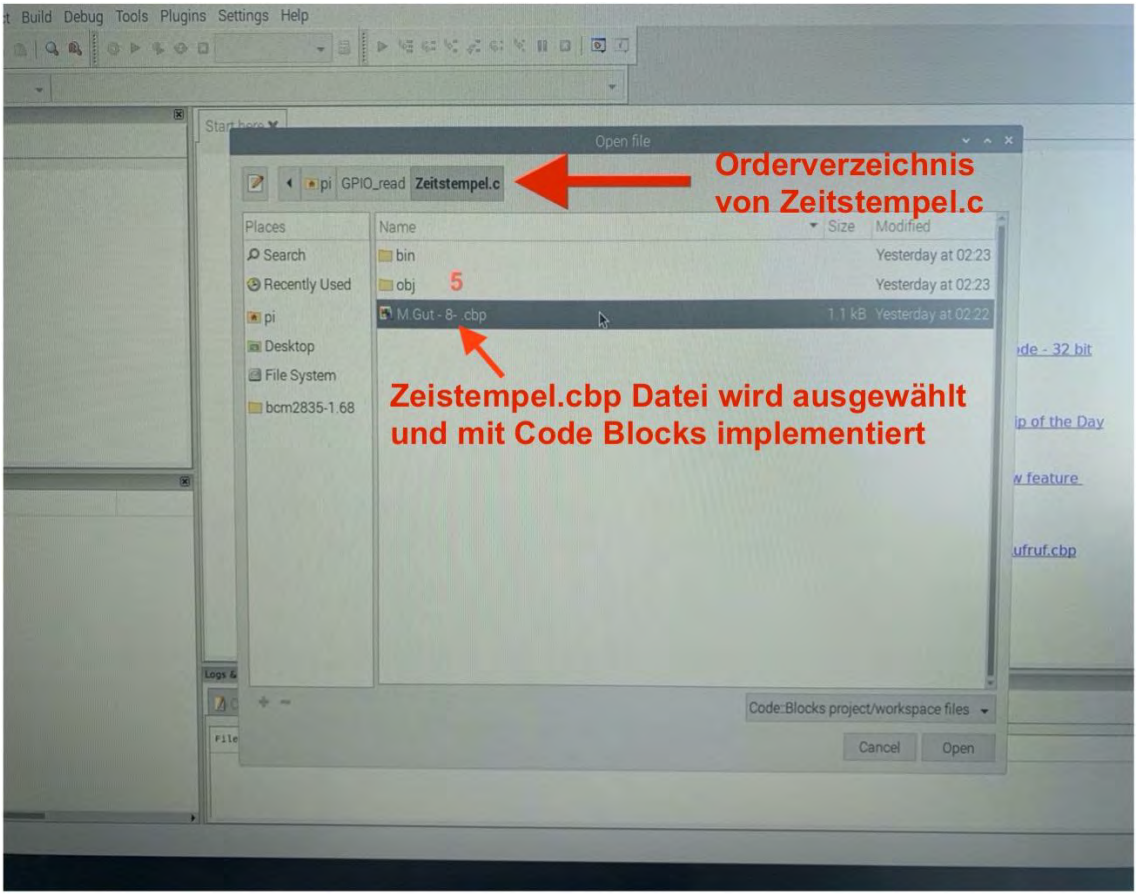

Abbildung 31: Zeitstempel.c Datei wird in die Entwicklungsumgebung *Codeblocks* implementiert [35]

Abbildung 32: Das GNSS-5-Click Modul empfängt kein PPS-Signal [35]

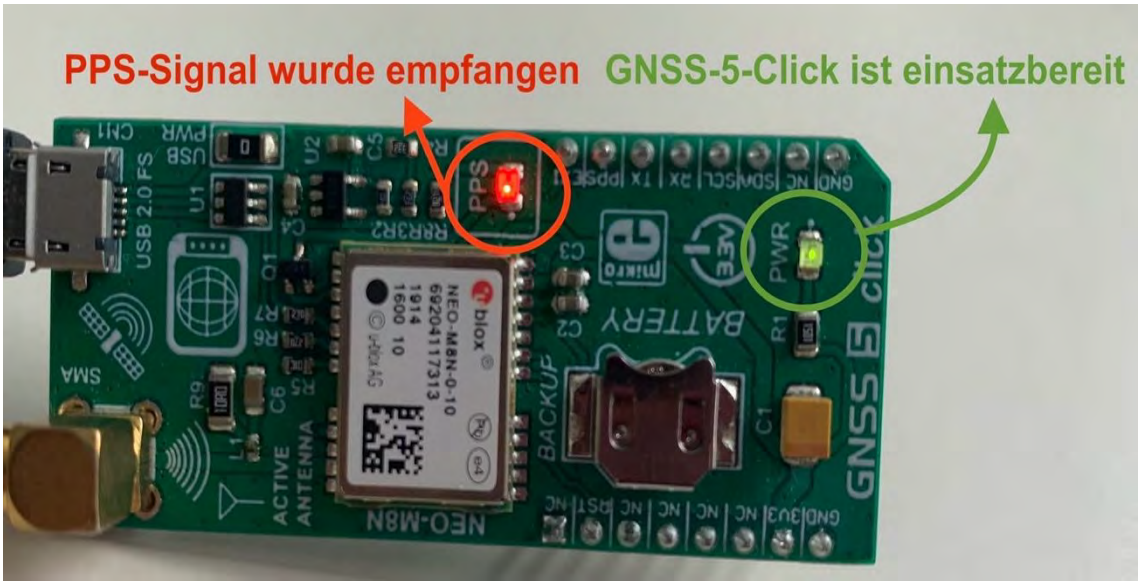

Abbildung 33: Das GNSS-5-Click hat das PPS-Signal empfangen (Markierung des Startpunktes der Zeitmessung) [35]

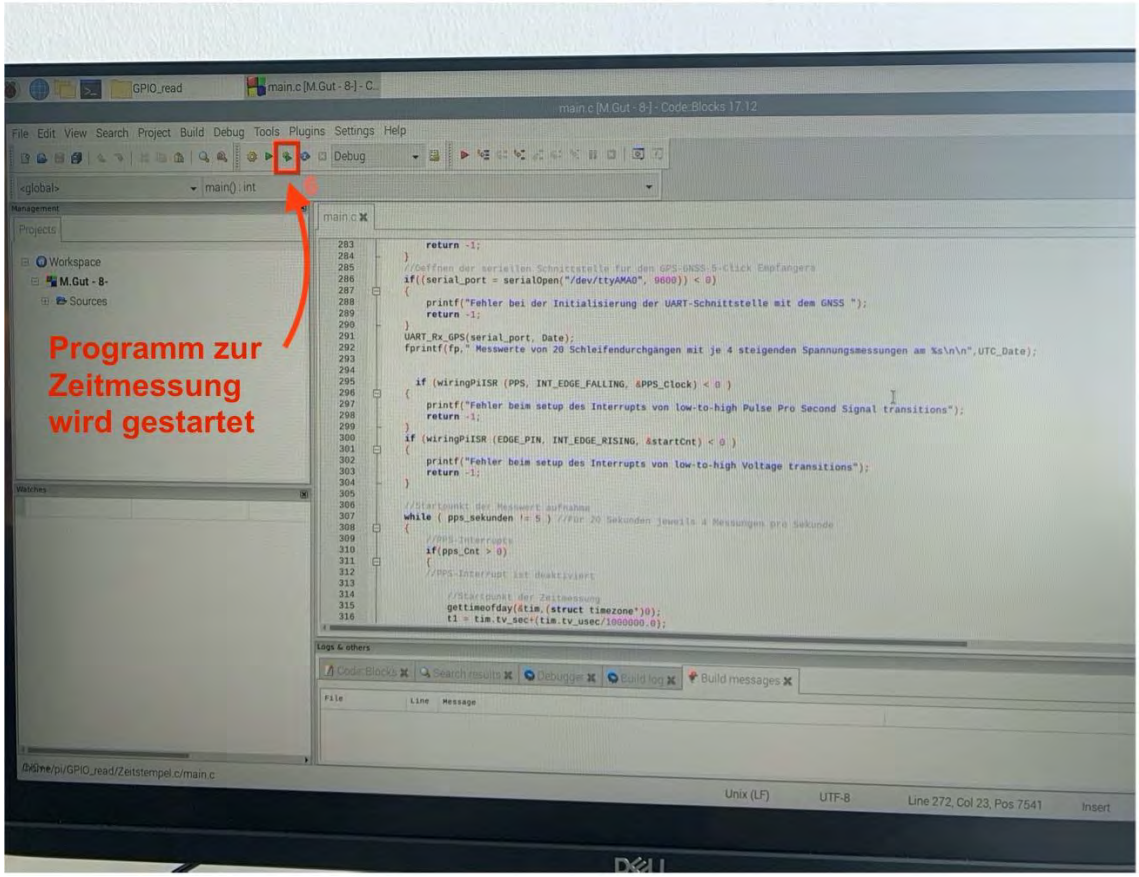

Abbildung 34: Ausführung der Zeitstempel.c Datei - Startpunkt der Zeitmessung mit dem PMU-Messgerät [35]

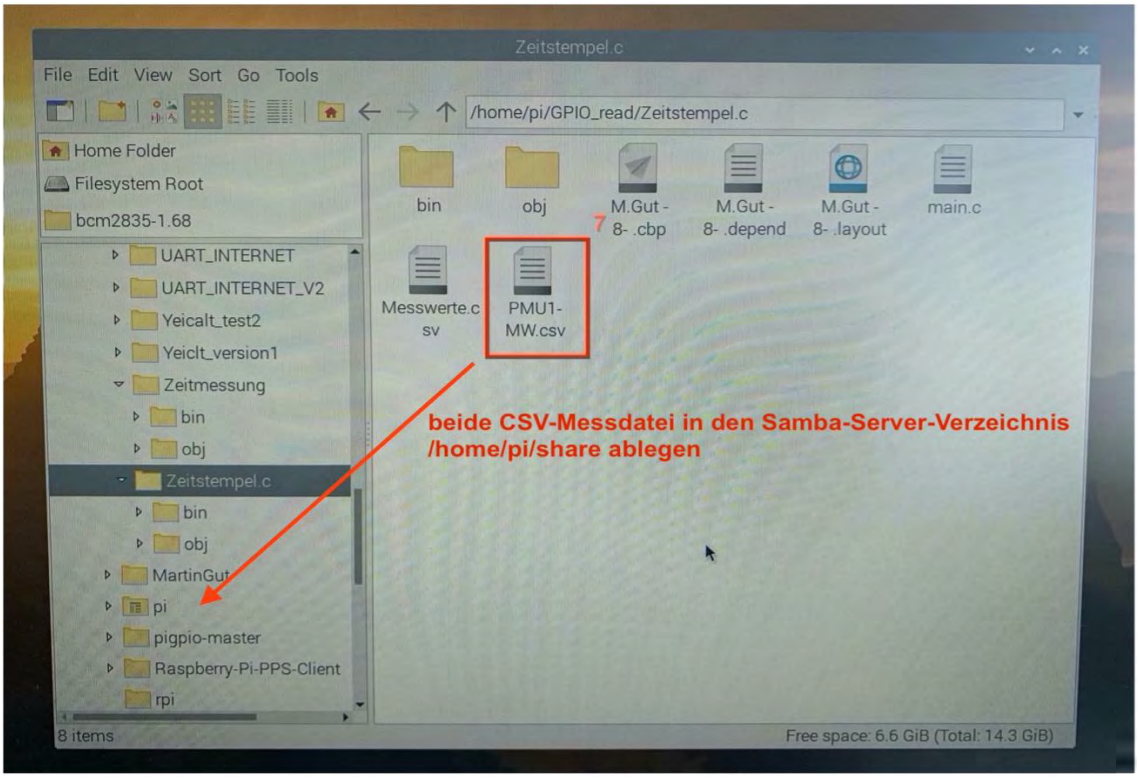

Abbildung 35: Die Zeitstempel.c erstellte CSV-Datei *„PMU1-MW.csv“* wird in den Ordern Zeitstempel.c abgespeichert [35]

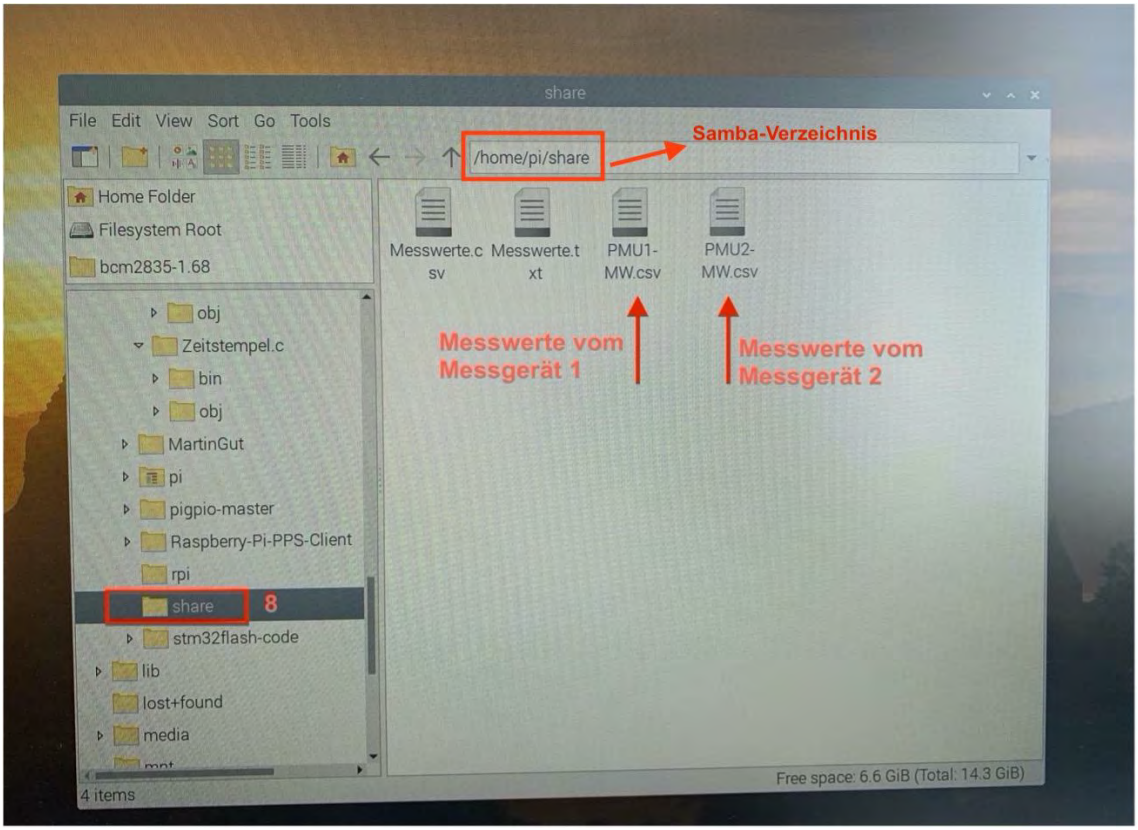

Abbildung 36: Samba-Verzeichnis – Speicherort der Messdaten beider PMU-Messgeräte abgespeichert [35]

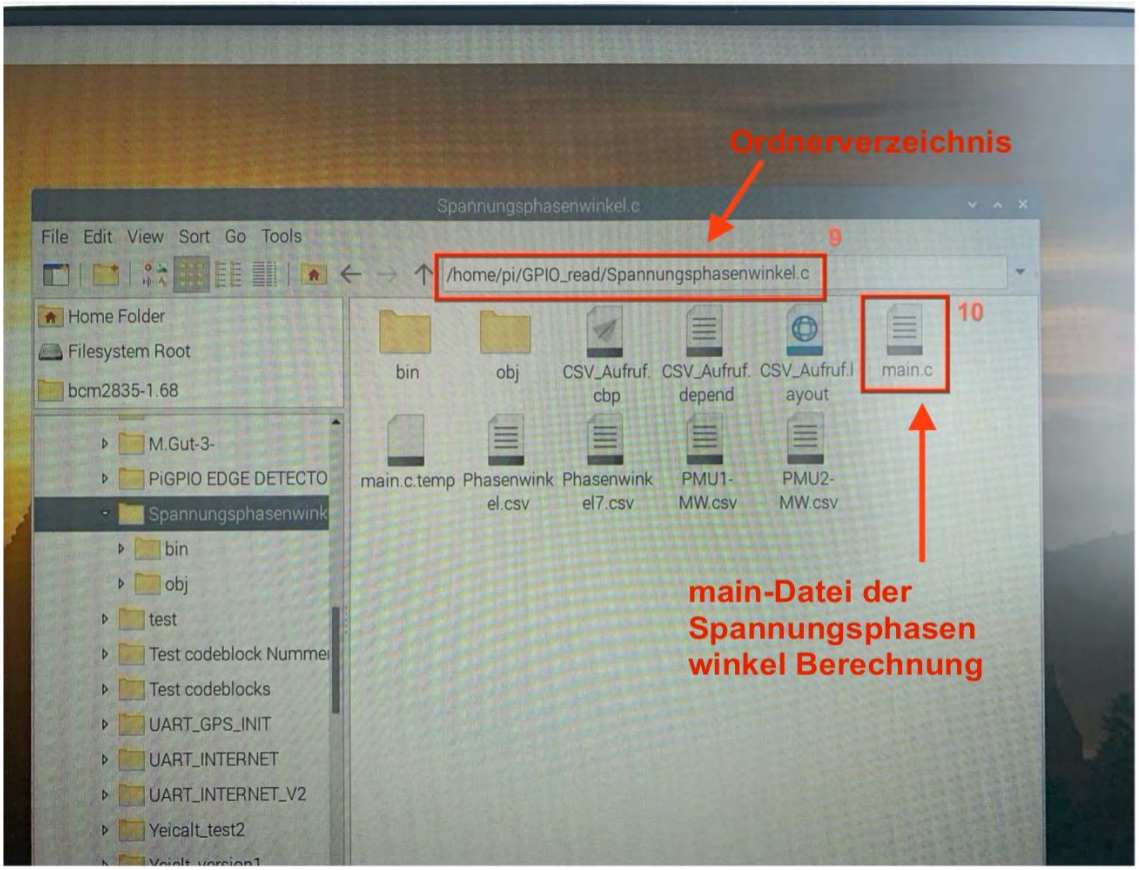

Abbildung 37: Spannungsphasenwinkel.c Datei im Filesystem Verzeichnis vorbereiten [35]

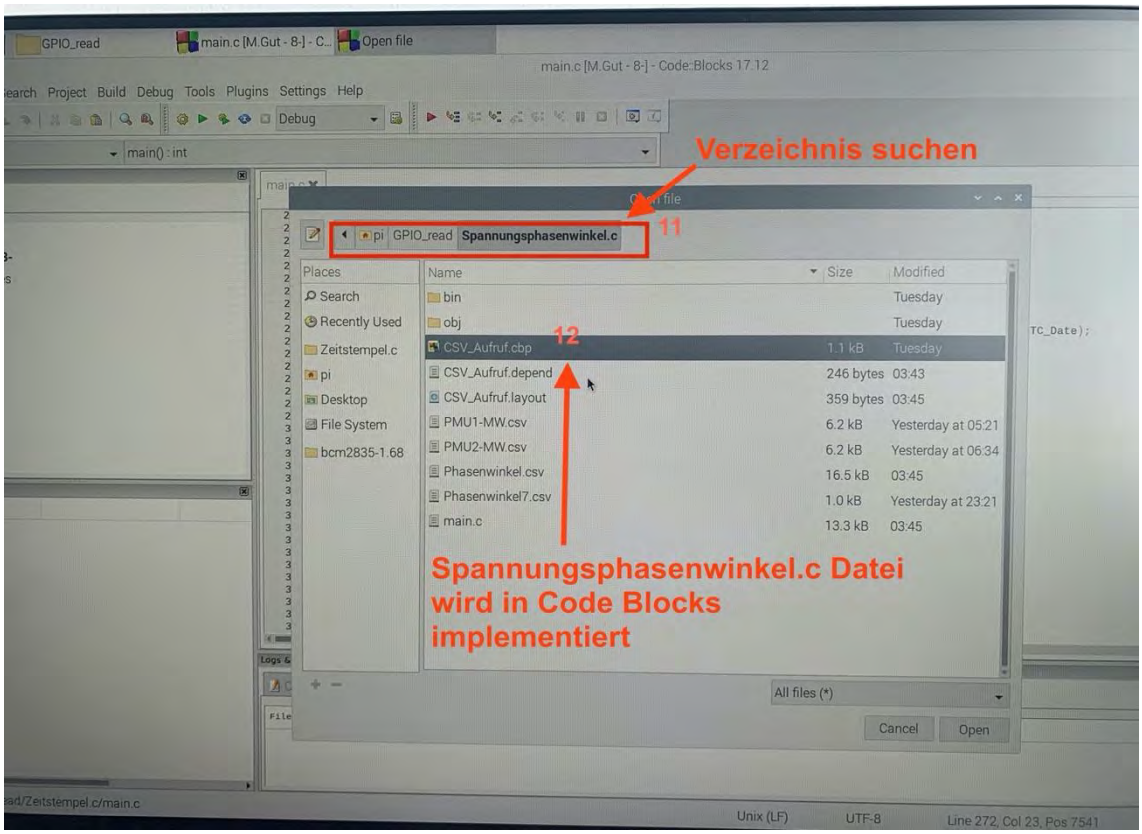

Abbildung 38: Die Spannungsphasenwinkel.c Datei wird in die Entwicklungsumgebung *Codeblock* implementiert [35]

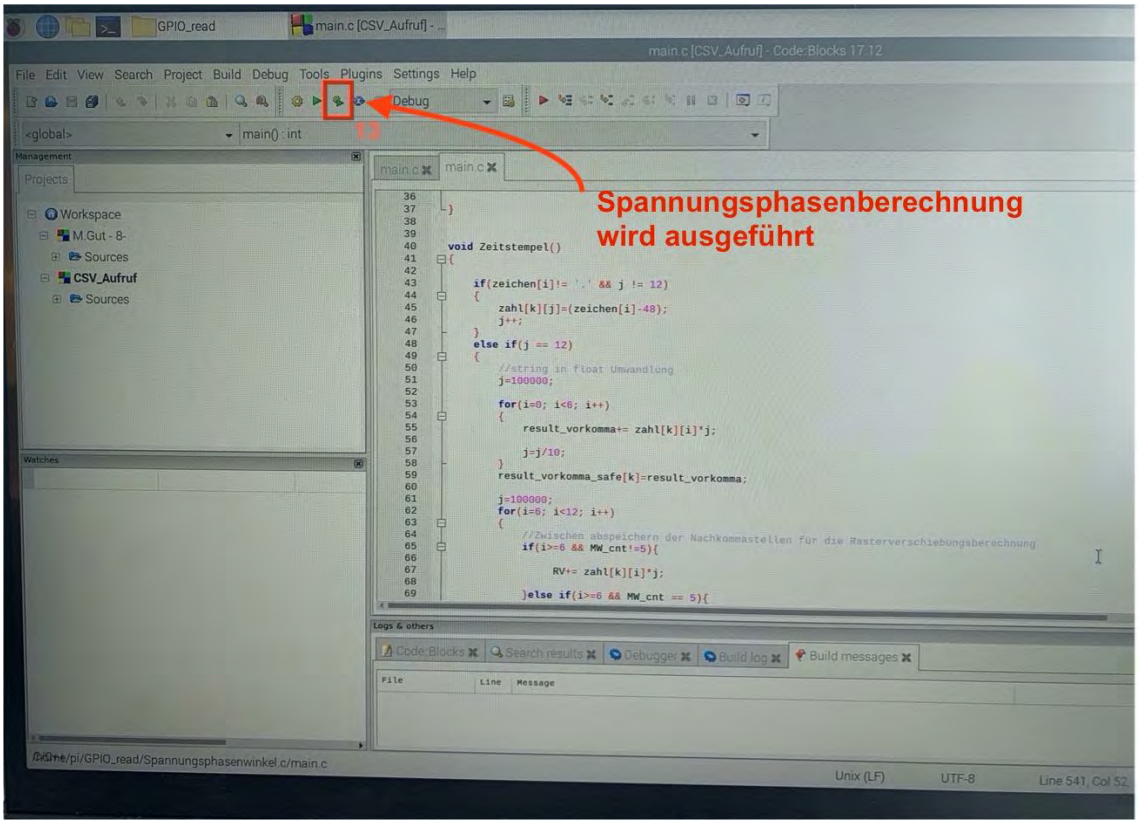

Abbildung 39: Ausführung der Spannungsphasenwinkel.c Datei - Startpunkt des PDC-Systems [35]

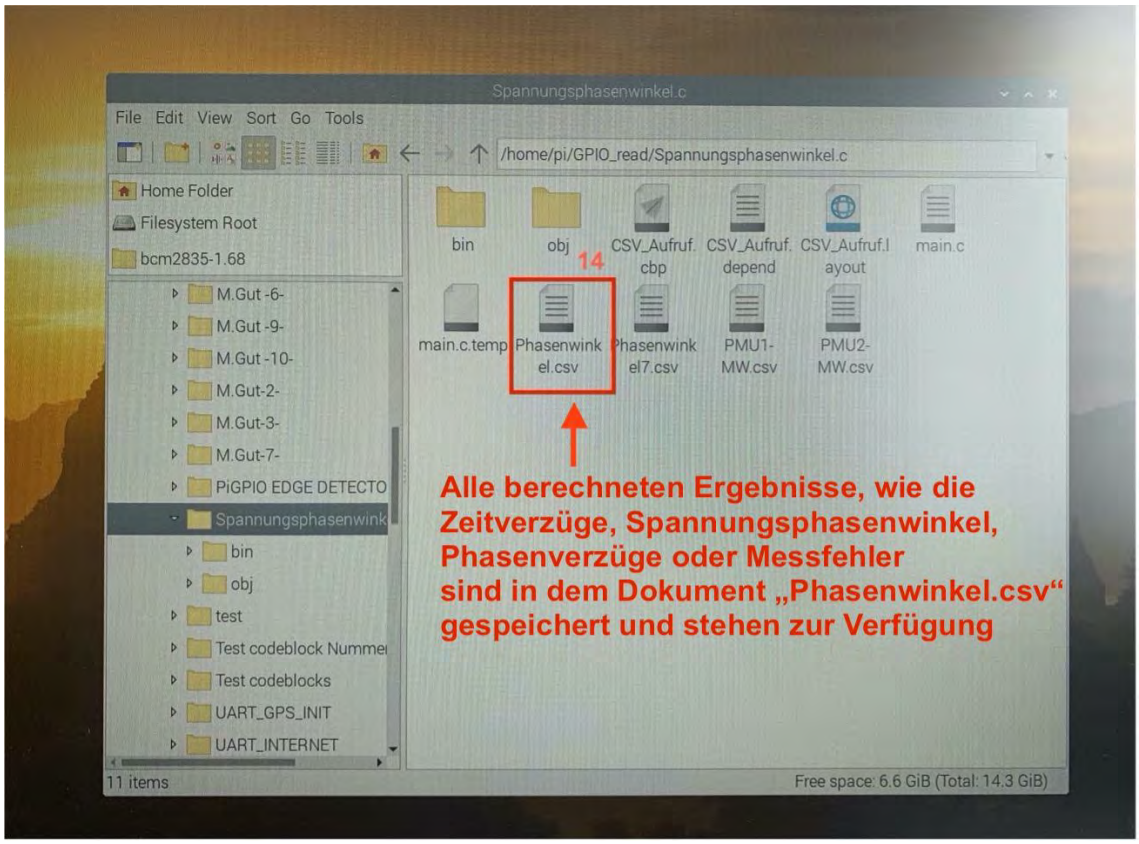

Abbildung 40: Ergebnisse bei der beiden Messdateien gespeichert in "Phasenwinkel.csv" [35]

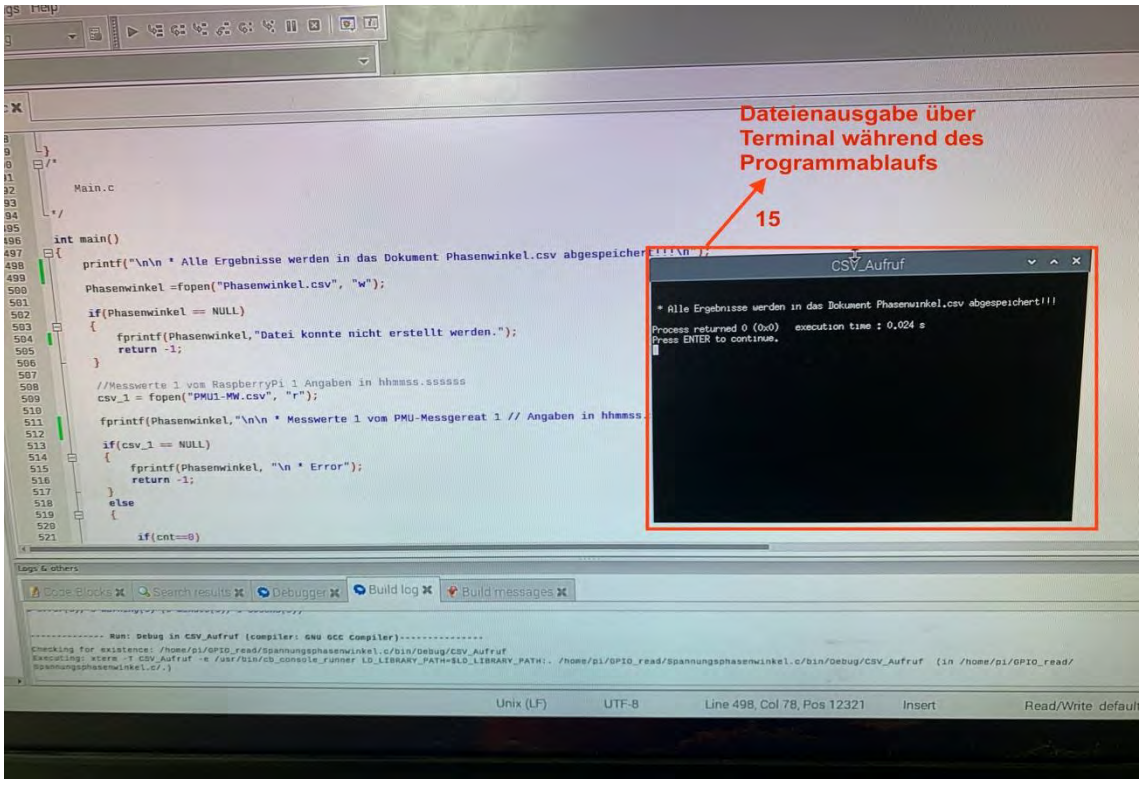

Abbildung 41: Dateienausgabe über das Terminal [35]