Sebastian Farrenkopf

Anbindung von Echtzeitsimulationsmodellen auf embedded Systems an die Leitwarte eines virtuellen Kraftwerks durch ein Kommunikationsgateway von Modbus/TCP zu IEC61850

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Wolfgang Renz
Zweitgutachter : Dipl.-Ing. (FH) Hans Schäfers
Abgegeben am 10. Oktober 2014

Sebastian Farrenkopf

Thema der Bachelorthesis

Anbindung von Echtzeitsimulationsmodellen auf embedded Systems an die Leitwarte eines virtuellen Kraftwerks durch ein Kommunikationsgateway von Modbus/TCP zu IEC61850.

Stichworte

Echtzeitsimulationsmodell, Matlab-Simulink, virtuelle Kraftwerke, Gateway, Modbus/TCP, IEC 61850, Python, Raspberry Pi

Kurzzusammenfassung

Diese Bachelorarbeit beschreibt den Aufbau und das Testen von Kommunikationsprotokollen zwischen einer Matlab-Simulink Simulation und einer Leitwarte. Die Arbeit beginnt mit dem Export eines echtzeitfähigen BHKW-Models auf den „ein-Platinen-Linux-Rechner“ Raspberry Pi. Mittels eines weiteren Raspberry Pis wird ein Gateway erstellt, als Kommunikationsschnittstelle zwischen der Simulation und einer Leitwarte. Das Gateways sorgt für eine Verbindung der BHKW-Simulation durch Modbus-TCP und einer Leitwarte durch die Norm IEC 61850, außerdem soll das Gateway die Simulationswerte und die Stellgrößen in eine Datenbank speichern.

Sebastian Farrenkopf

Title of the paper

Integration of real-time simulation models of embedded systems to the control room of a virtual power plant through a communication gateway Modbus/TCP to IEC61850.

Keywords

Real-time simulation model, Matlab-Simulink, virtual power plants, Gateway, Modbus-TCP, IEC 61850, Python, Raspberry Pi

Abstract

This thesis describes the design and testing of communication protocols between a Matlab-Simulink simulation and a control room. The work begins with the export of a real-time CHP Models to the "one-board Linux computer" Raspberry Pi. Assistance with another Raspberry Pis is a gateway created as a communication interface between the simulation and a control room. The gateway provides a connection of the CHP simulation through Modbus-TCP and a control room by the standard IEC 61850, also the gateway store the simulation values and the variables in a database.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Nomenklatur

Verwendete Indizes

Symbol	Bedeutung
BHKW	Blockheizkraftwerk
KWK	Kraft-Wärme-Kopplung
CHP	Combined Heat and Power (englisch für KWK)
DEA	dezentrale Energieversorgungsanlage
TCP	Transmission Control Protocol
IEC	International Electrotechnical Commission
SLK	Spitzenlastkessel
HS	Heizstab
MST	Verbleibenden minimalen Ausschaltzeit
MRT	Verbleibenden minimalen Einschaltzeit
P_el	Elektrische Last
P_th	Thermische Last
kW	Kilowatt
VPN	Virtuelle private Netzwerke
UDP	User Datagram Protocol
SCADA	Supervisory Control and Data Acquisition
HTML	HyperText Markup Language
CSV	Comma-separated values
VNC	Virtual Network Computing

1 Motivation

1.1 Motivation für das Thema

In einem Stromnetz müssen Strombedarf und Stromerzeugung stets ausgeglichen sein. Weichen die Größen voneinander ab, müssen entweder Lasten verschoben oder die Stromerzeugung angepasst werden.

Doch die wetterabhängige Stromerzeugung von Solar- und Windenergieanlagen sind begrenzt voraussagbar, und selbst bei Stromüberschuss soll es den regenerativen Erzeugern möglich sein, ins Stromnetz einzuspeisen.

Deshalb bedarf es zum Ausgleich eine weitere Leistung, eine Regelleistung oder Reserveleistung. Mit Hilfe dieser Reserve kann die Stromversorgung gewährleistet werden, die Regelleistung sorgt für den Ausgleich zwischen dem Strombedarf und der Stromerzeugung.

Dezentrale Blockheizkraftwerke können zu dieser Reserve beitragen, durch die gekoppelte Erzeugung von Wärme und Strom können sie Strom produzieren, wenn Sonne und Wind nicht ausreichen.

Derzeit werden aber BHKWs fast ausschließlich in Volllast betrieben und speisen die elektrische Energie gleichmäßig und unabhängig vom Bedarf ins Stromnetz ein. Um diese Grundlast zu flexibilisieren und das BHKW zeitweise mit hoher Einspeiseleistung zu betreiben, aber auch in Zeiten von Stromüberschuss stillzustehen, braucht das BHKW und seine Komponenten eine Informations- und Kommunikationsschnittstelle.

Doch nicht nur Blockheizkraftwerke können zu einer weiterhin zuverlässigen Stromversorgung beitragen, auch der Ausbau von Speichersystemen kann zur Reserve beitragen. Diese Speicherprozesse müssen jedoch nicht zwingend mit einer Rückverstromung in Verbindung stehen, die Erzeugung von Wasserstoff oder Methan oder der Einsatz von thermischen Energiespeichern kann überschüssigen Strom speichern und zu einem späteren Zeitpunkt bedarfsgerecht zur Verfügung stellen[Fra13].

Doch dazu muss eine einheitliche Kommunikation möglich sein, die Netze müssen intelligenter gemacht werden und es bedarf mehr Verständigung, jeder Erzeuger, egal wie unterschiedlich, sollte einheitlich angesprochen werden können. Damit befasst sich IEC 61850.

1.2 Problemstellungen

Das Ersetzen von hauptsächlich großen Kraftwerken durch kleinere, dezentrale Erzeugungsanlagen zieht auch das Ersetzen eines zentralen und informationsarmen Versorgungsnetzes durch ein dynamisches und intelligentes Netz mit sich. Doch für dieses intelligente Netz besteht ein großer Entwicklungsbedarf. Um das Verhalten und die Dynamik von Erzeugern in diesem Netz zu testen benötigen Energiedienstleister zum einen Simulationsmodelle, die das reale Anlagenverhalten simulieren können und zum anderen auch die Kommunikationsschnittstellen zur Anbindung an die Leitwarte. Diese Simulationsmodelle helfen bei der Entwicklung und Erprobung von Algorithmen zur Anlagensteuerung in einem virtuellen Kraftwerk.

Das Center for Demand Side Integration (C4DSI) beschäftigt sich in dem Forschungsprojekt Smart Power Hamburg mit genau diesen Themen[SPH11].

2 Grundlagen

Dieses Kapitel soll zur grundlegenden Verständlichkeit der Arbeit dienen.

Es soll die Grundlagen der einzelnen Einheiten erklären, zum einen die BHKW-Echtzeitsimulation, das Gateway und die Leitwarte, zum anderen auch die zur Kommunikation benutzten Protokolle Modbus-TCP und IEC 61850 vorstellen.

2.1 BHKW

Ein Blockheizkraftwerk (BHKW) ist eine Anlage zur Gewinnung von Wärme und elektrischer Energie; eine stromerzeugende Heizung. Die Stromerzeugung wird meist durch einen Verbrennungsmotor mit flüssigen oder gasförmigen Kraftstoffen (Diesel, Pflanzenöl oder Gas) betrieben. Die Verbrennung ist im Vorgehen mit einem PKW- oder LKW-Motor gleichzusetzen, nur das es sich in einem BHKW um einen stationären Motor handelt. D.h. die Drehzahl ist konstant und die Drehzahlschwankungen relativ klein. Die Kurbelwelle des BHKWs ist mit einem Generator verbunden, der die elektrische Energie erzeugt. Die Wärme wird aus dem Abgas und aus dem Kühlwasserkreislauf gewonnen [Sch10].

Damit der Motor im wirtschaftlichen Bereich betrieben wird, sollte eine hohe Betriebsstundenzahl erreicht werden. Dieser Bereich liegt je nach Auslegung bei ungefähr 4.500 Volllastbetriebsstunden pro Jahr. Ein modernes BHKW erreicht einen elektrischen Wirkungsgrad von über 45 %. Mit der Nutzung der Abwärme beträgt der Gesamtwirkungsgrad über 90% [Sut08].

Die gleichzeitige Gewinnung von mechanischer Energie und nutzbarer Wärme wird auch Kraft-Wärme-Kopplung genannt, eine Stromerzeugung, die seit April 2002 in Deutschland unter dem Kraft-Wärme-Kopplungsgesetz gefördert und auch nach dem Erneuerbare-Energien-Gesetz vergütet wird [BSU11].

Kleinere BHKWs richten sich meist nach dem Wärmebedarf. Sie sind in Leistungsklassen < 50kW vertreten. Da ein BHKW am wirtschaftlichsten arbeitet, wenn es an 365 Tagen im Jahr 24 Stunden läuft, eignet es sich besonders, wenn eine hohe Grundlast an Wärme gebraucht wird. Häufige Standorte sind Gewerbe- und Handwerksbetriebe mit einem hohen Warmwasserbedarf sowie Hotels, Krankenhäuser und Häuser mit beheiztem Schwimmbädern. Mit Hilfe einer Absorptionskühlanlage ist es möglich, die Wärme auch zum Kühlen zu benutzen.

Größere BHKWs sind meist in Gebieten zu finden, wo schnell und im großen

Mengen auf den Brennstoff zugegriffen werden kann. Meist werden diese BHKWs mit Rohstoffen aus der Landwirtschaft oder Holzbetrieben befeuert. Durch den Bonus für Strom aus nachwachsenden Rohstoffen (Nawaro Bonus) ist der Betrieb aus Biogas oder Energiepflanzen wirtschaftlich gesehen sehr interessant, ökologisch aber nur, wenn die Wärmenutzung auch Verwendung findet, und so die Effizienz von Wärmeerzeugung und Wärmenutzung vorhanden ist [Sut09].

Doch können größere, stromgeführtes BHKW ihre Arbeit am Strombedarf anpassen, was vor allem für Energieversorger interessant ist, da so Stromlastspitzen nachgefahren werden können. Auch können so Flauten bei der Energiegewinnung durch Windkraft überbrückt werden. Die Wärme wird von den Versorgern meist in das Fernwärmenetz eingespeist, sodass die Erzeugung von Strom und Wärme vollen Nutzen hat [Zah10].

BHKWs eignen sich also nicht nur für die Wärmegewinnung, sondern können durch ihre Stromgewinnung zur Flexibilisierung am Strommarkt betragen. Doch dazu müssen sie im Verbund arbeiten und zentral geschalten werden. Sie müssen also Netzgeführt werden. Dazu muss es aber eine entkoppelte Wärmenutzung und eine einheitliche Kommunikation zwischen BHKW und Leitwarte geben. Mit dieser flexibilisierten Steuerung beschäftigt sich diese Arbeit.

Abbildung 2.1: ZuhauseKraftwerk (Pel:20kW) und Wärmespeicher des Hamburger Energieversorgungsunternehmen Lichtblick (Quelle: www.lichtblick.de)

2.2 Kommunikationsprotokolle

In der Welt der Automatisierung von dezentralen Energieversorgungsanlagen (DEA) werden viele Sprachen gesprochen. Die Kommunikation zwischen einer Leitwarte und einer Anlage erfolgt zwar meist über das Internet, doch gibt es diverse Kommunikationsprotokolle, die zur Fernwirklösung und zum Datenaustausch zur Verfügung stehen. Dieses führt bei zunehmender Internationalisierung der Energiemärkte und der Vernetzung dezentraler Stromerzeugungsanlagen zu Schnittstellenproblemen. [Scn06]

Doch gerade die Vernetzung auch von kleineren Stromproduzenten im kommunalen und regionalen Energieverbund gewinnt zunehmend an Bedeutung. Die Kommunikation zur Steuerung, zur Lastenverteilung und zur Erzeugung und Speicherung von elektrischer Energie in intelligenten Netzen, den sogenannten Smart Grids, ist eine der Herausforderungen der Energiewende. Denn gerade der Ausbau der regenerativen Energie fordert die Einbindung dezentraler Energiequellen und die Vernetzung von Stromerzeugern. Dazu reicht die bisherige Überwachung und Steuerung des Mittel- und Hochspannungsnetzes nicht aus, die Integration vieler neuer und kleinerer regenerativer Energieerzeugungseinheiten in das Netz setzt eine genaue Kenntnis auch der unteren Netzebene und dem Verteilnetz voraus. Nur so können dezentrale und

wetterabhängige Energieanlagen in das Netz einspeisen und koordiniert werden.

2.2.1 Modbus-TCP

Die Art der internen und externen Datenübertragung einer BHKW Steuerung ist zwar vom Hersteller abhängig, doch viele benutzen das offene Nachrichtenprotokoll Modbus-TCP.

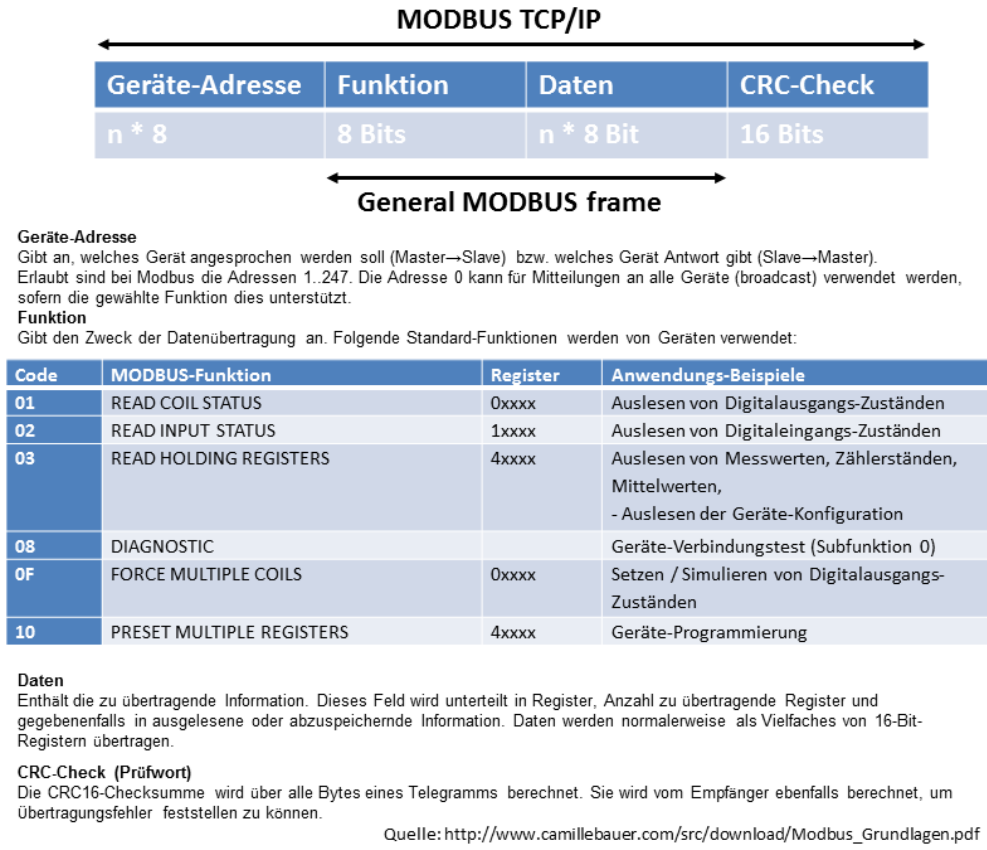
Dieses offene Protokoll wurde 1979 für die Kommunikation von Speicherprogrammierbare Steuerungen entwickelt und hat sich in der Industrie als De-facto-Standard entwickelt.

Es basiert auf einer Master/Slave-Architektur, was bedeutet, dass es eine hierarchische Verwaltung in der Datenübertragung gibt. Meist gibt es einen PC als Master, der mit mehreren, meist unterschiedlichen Teilnehmer kommunizieren kann. Da es nur einen Datenkanal gibt, in diesem Fall das Ethernet, ist es nötig, die Rechte auf den Zugriff der bereitgestellten Ressourcen klar zu definieren. Im Master/Slave verfahren ist es nur der Master, der unaufgefordert Abfragen machen kann. Das heißt aber auch, dass die einzelnen Devices, die als Slave fungieren, nicht untereinander kommunizieren können. Die Kommandos (Anfragen) kommen also vom Modbus-Master, die restlichen Busteilnehmer Reagieren (Antworten) entweder mit dem Senden der gewünschten Daten oder es gibt eine Ausführungsbestätigung der gewünschten Betriebsfunktion. Somit wechseln sich bei der Übertragung Anfrage und Antwort ständig ab [IDA06].

Bevor Nutzerdaten über Modbus-TCP übertragen werden können, wird eine TCP/IP-Verbindung zwischen Client und Server aufgebaut. Sobald diese hergestellt ist, können Server und Client beliebig oft und beliebige viele Nutzerdaten austauschen. Diese Nutzerdaten werden in den Nutzerdatencontainer des TCP-Telegramms, in der Anwendungsschicht, eingebettet. Diese TCP-IP Verbindung bleibt meist bestehen, solange kommuniziert wird, was bei zyklischen Übertragungen die Protokolleffizienz erhöht. Aber auch so ist die Effizienz des Modbusses sehr hoch, da es durch das mapping auf TCP-IP zu dem gewöhnlichen TCP-IP Overhead nur wenige Bytes zusätzlichen Protokoll-Overhead hinzugefügt wird, was sich auf die Performance auswirkt [Cam06].

Modbus besitzt einen Standardport, den „well known Port 502“. Dieser öffentliche Anschluss wurde von der Internet Engineering Taskforce (IETF) definiert.

Das Modbus Protokoll kann auch als Frage/Antwort-Protokoll verstanden werden, es stehen verschiedene benutzerdefinierte Funktionscodes zur Verfügung, die für den Zugriff auf Daten, zum Lesen und Schreiben, oder für die Diagnose verwendet werden können.



Geräte-Adresse
Gibt an, welches Gerät angesprochen werden soll (Master→Slave) bzw. welches Gerät Antwort gibt (Slave→Master).
Erlaubt sind bei Modbus die Adressen 1..247. Die Adresse 0 kann für Mitteilungen an alle Geräte (broadcast) verwendet werden, sofern die gewählte Funktion dies unterstützt.

Funktion
Gibt den Zweck der Datenübertragung an. Folgende Standard-Funktionen werden von Geräten verwendet:

Code	MODBUS-Funktion	Register	Anwendungs-Beispiele
01	READ COIL STATUS	0xxxx	Auslesen von Digitalausgangs-Zuständen
02	READ INPUT STATUS	1xxxx	Auslesen von Digitaleingangs-Zuständen
03	READ HOLDING REGISTERS	4xxxx	Auslesen von Messwerten, Zählerständen, Mittelwerten, - Auslesen der Geräte-Konfiguration
08	DIAGNOSTIC		Geräte-Verbindungstest (Subfunktion 0)
0F	FORCE MULTIPLE COILS	0xxxx	Setzen / Simulieren von Digitalausgangs-Zuständen
10	PRESET MULTIPLE REGISTERS	4xxxx	Geräte-Programmierung

Daten
Enthält die zu übertragende Information. Dieses Feld wird unterteilt in Register, Anzahl zu übertragende Register und gegebenenfalls in ausgelesene oder abzuspeichernde Information. Daten werden normalerweise als Vielfaches von 16-Bit-Registern übertragen.

CRC-Check (Prüfwort)
Die CRC16-Checksumme wird über alle Bytes eines Telegramms berechnet. Sie wird vom Empfänger ebenfalls berechnet, um Übertragungsfehler feststellen zu können.

Quelle: http://www.camillebauer.com/src/download/Modbus_Grundlagen.pdf

Abbildung 2.2: Protokollstruktur des Modbuses

Jedes Gerät, das über Modbus kommuniziert, besitzt eine eigene Datenorganisation, weshalb der Benutzer vor der Benutzung des Modbus einen Blick in die Dokumentation des Gerätes werfen muss.

2.2.2 IEC 61850

Das Übertragungsprotokoll für die Schutz- und Leittechnik in elektrischen Schaltanlagen der Mittel- und Hochspannungstechnik IEC 61850 ist ein objektorientiertes Ethernet-Protokoll.

Es ist besonders für die Prozessübertragung zwischen entfernten Stationen und der Netzleittechnik geeignet, da die Norm einerseits die Bedürfnisse der Mittel- und Hochspannungsautomatisierung, aber es auch die Prozessdatenübertragung hin

zur Leittechnik lückenlos und durchgängig deckt.

Es eignet sich aber auch für virtuelle Kraftwerke und Erzeugerverbunde, da es eine sehr vereinfachte Architektur besitzt, das durch die Nutzung eines Prozessbus von den Aktoren und Sensoren im Feld hin zu den SCADA- und Kontrollsystemen der Leittechnik reicht. Die Normreihe IEC 61850 beschreibt so zum einen die allgemeinen Festlegungen für Schaltanlagen, wichtige Informationen für Funktionen und Anlagen und den Informationsaustausch für die Überwachung, Steuerung und Messung dieser [Cra09].

Das Protokoll verwendet ebenfalls TCP/IP als Basisübertragungsprotokoll, kann also auch über das Internet kommunizieren.

Die Grundlage der IEC 61850 sind die modularen und skalierbaren Konfigurationen, die logische Knoten (LN) genannt werden. Diese Knoten können völlig frei auf logische und physikalische Geräte aufgeteilt werden.

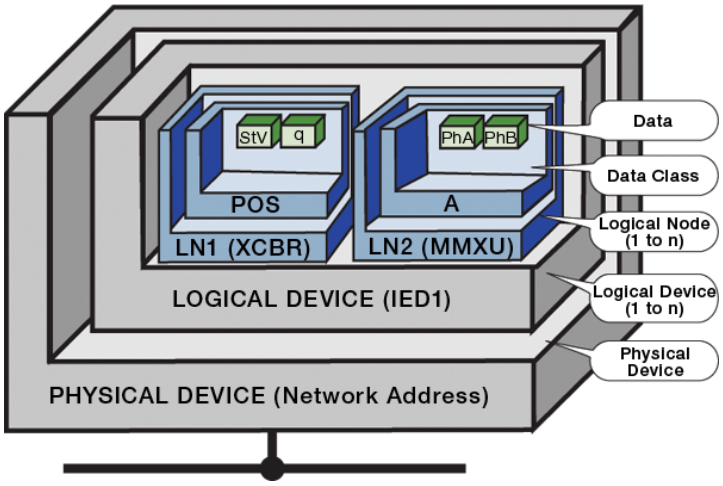


Abbildung 2.3: Schaubild eines Gerätes als Container mit zwei logischen Knoten [Ger11]

Jeder Knoten besteht aus verschiedenen Datenobjekten, ähnlich den Klassen in der Objektorientierung, nur das die Datenobjekte den Attributen der Klassen entsprechen. Die Datenobjekte der Knoten sind in Gruppen organisiert, jedes Datenobjekt ist einer Bedeutung und einer Referenz zugeordnet. Diese Datenmodelle lassen sich auf alle wesentlichen Anwendungen in der Stationsautomatisierung modellieren, in der aktuellen Norm werden nahezu 100 verschieden Logische Knoten mit zusammen etwa 2000 Datenobjekten definiert, welche weitere Attribute besitzen [Ger11].

2.3 Gateway

Die Aufgabe eines Gateways ist das Verbinden von Netzen, die zwar physisch zusammenhängen, aber aufgrund ihrer Netz- oder ihren Kommunikationsprotokollen nicht miteinander korrespondieren können. Das Gateway nimmt dazu eine Protokollumsetzung vor, und kann, je nachdem wie weit sich die Protokolle unterscheiden, zur Konvertierung auch Daten und Informationen weglassen.

Durch ein Gateway können verschiedene Protokolle gekoppelt werden, es wird an dem Netzübergang installiert und sorgt damit für eine logische Verbindung.

Abbildung 2.4: Verschiedene Modbus TCP/IP zu IEC61850 Gateways,
der Preis liegt zwischen 700€ und 2500€

2.4 Virtuelles Kraftwerk

Ein Virtuelles Kraftwerk ist der Zusammenschluss dezentraler Stromerzeuger. Es ist eine Art Cluster aus mehreren Anlagen, die nicht nebeneinander stehen,

sondern weit entfernt von einander Strom erzeugen. Am besten funktionieren virtuelle Kraftwerke mit möglichst unterschiedlichen Energiequellen, sodass bei einer Flaute dennoch Solaranlagen Strom produzieren. Und wenn weder Sonne scheint noch Wind weht das virtuelle Kraftwerk den Verbund aus BHKW und/oder Biogasanlage einspringen können. Denn durch eine zentrale Leitstelle, die Zugriff und Steuerungsmöglichkeiten auf die Anlagen hat, können diese so zusammengeschaltet werden, dass ein Virtuelles Kraftwerk konstanten und vorhersehbaren Strom liefern kann.

Gerade Betreiber kleiner Anlagen können so mit der Hilfe des Verbundes gezielt dann Strom einspeisen, wenn an der Strombörse in Leipzig oder am Regelenergiemarkt die Strompreise hoch sind. Durch kalkulierte Wetterprognosen, aktuelle Strompreise und Schätzungen des Energiebedarfs kann Strom so bestmöglich vermarktet werden, was für einen einzelnen Erzeuger schon wegen der zu geringen Erzeugung verwehrt bleiben würde.

Doch nicht nur Stromerzeuger können in Virtuellen Kraftwerken gebündelt werden, auch Wärme kann mittels Virtuellen Kraftwerken gebündelt vermarktet werden [Nie20].

3 Anforderungen

Nachdem die Grundlagen beschrieben wurden kommen jetzt die nötigen Anforderungen.

Dazu wird zuerst die Anforderung des Gesamtsystems betrachtet, um dann auf die Anforderungen der einzelnen Komponenten eingehen zu können. Zuletzt wird in diesem Kapitel die Entwicklungsumgebung mit den Programmen zur Umsetzung des Projekts vorgestellt.

3.1 Aufgabenstellung

Das Projekt soll zum Ziel haben, einer existierenden Leitwarte den Zugriff auf ein Simulationsmodell zu ermöglichen.

Dazu soll ein Simulationsmodell, bestehend aus einem BHKW, einem Spitzenlastkessel und einem Warmwasserspeicher mit Heizstäben, welches die Arbeitsgruppe C4DSI in Matlab/Simulink entwickelt hat und hierfür zur Verfügung stellt, aus der Simulink-Umgebung gelöst und über das Modbus Protokoll kommunizieret werden.

Des Weiteren soll ein kostengünstiges Gateway entwickelt werden, dass die Kommunikation zwischen Leitwarte und Modell steuert. Zudem soll dieses Gateway die Simulationsdaten loggen und eine Schnittstelle zum Monitoring bereitstellen.

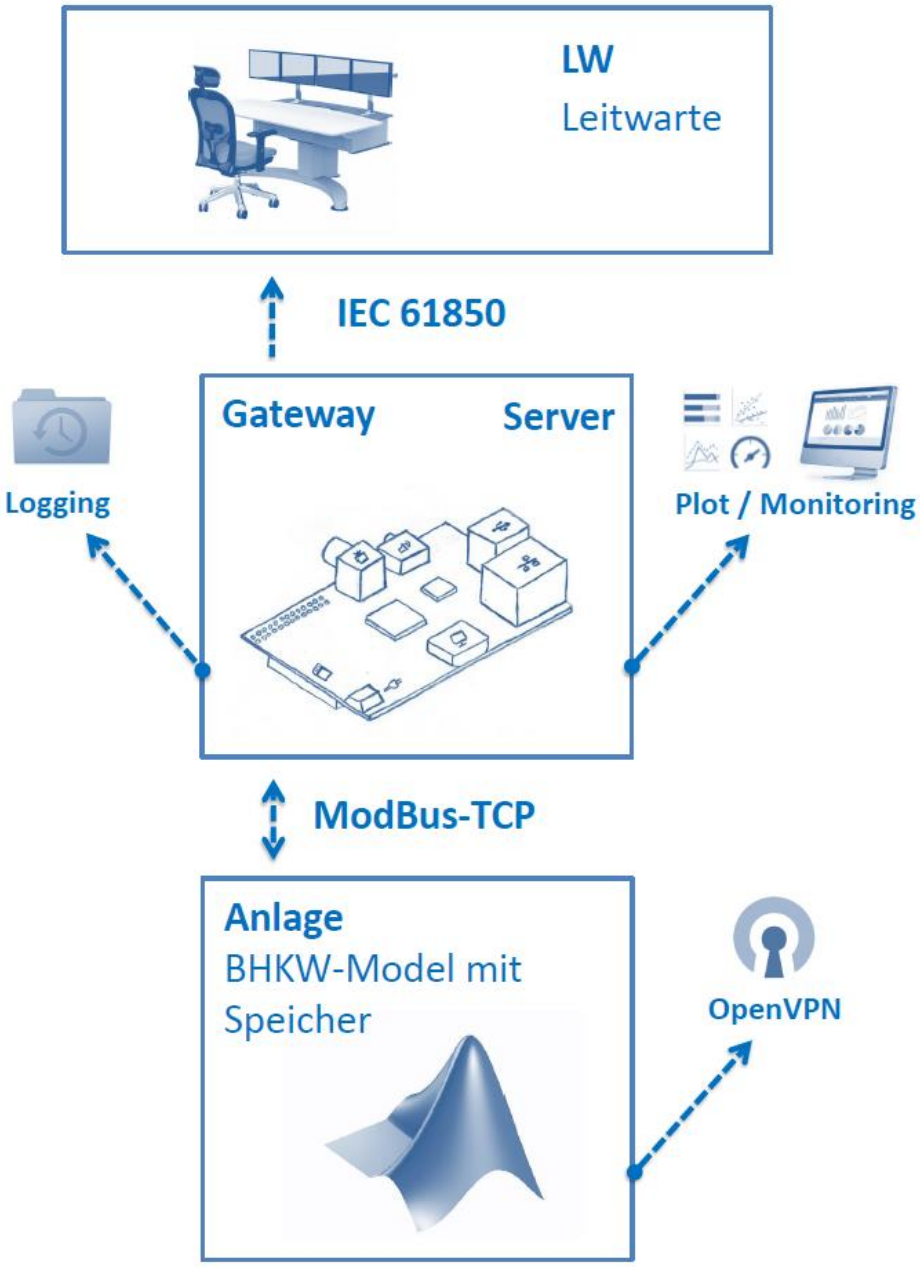


Abbildung 3.1: Schematische Architektur des Projektes

Die Eingabe- und Ausgabeparameter des Anlagenmodells werden über ModBus TCP durch ein Gateway kommuniziert, das einen Raspberry Pi als Hardwareumgebung nutzt. Das Gateway soll mit einer Leitwarte über den Kommunikationsstandard IEC 61850 kommunizieren. Dies soll hier durch einen Stack realisiert werden, den die RWTH Aachen unter Python entwickelt hat.

Eine weitere Aufgabe an diese Arbeit ist es, für das Forschungsprojekt Smart Power Hamburg die Matlab/Simulink Modelle von der Hochschule für

Angewandte Wissenschaften Hamburg echtzeitlauffähig an die Leitwarte von Hamburg Energie anzubinden. Hierfür werden die Modelle ohne ein Gateway versehen. Die Anbindung erfolgt direkt über Modbus-TCP. Die Modelle sollen zum Testen der entwickelten Leitwartensoftware genutzt werden.

3.2 Anforderungsanalysen

Das System soll es einer Leitwarte ermöglichen, auf Simulationsmodelle zuzugreifen als ob es sich um reale Anlagen handeln würde. Zum einen kann dadurch die softwaretechnische Steuerung von virtuellen Kraftwerken betrachtet werden, zum anderen können systematische Untersuchungen geeigneter Betriebs- und Regelungskonzepte abgefahren und nachgeahmt werden.

Mit Hilfe der Simulationsmodelle können so leittechnische Steuerungs- und Regelungskonzepte für ein virtuelles Kraftwerk entwickelt und getestet werden. Doch ist eine Simulation auch immer nur ein beschränktes Abbild der Wirklichkeit, sie läuft nicht kontinuierlich, sondern sequentiell. Durch den zeitlich diskreten Systemzustand gibt es zwischen den Intervallen unbekannte Größen. Das bedeutet, dass sich erst nach einem nächsten Schritt die Werte ändern.

Da sich die Simulation aber möglichst real verhalten soll, ist es wichtig, diese Schritte möglichst klein zu halten, so dass die gehaltenen Werte zwischen den Schritten als gültig angenommen werden können, und es sich dadurch keine großen Sprünge in der Stetigkeit ergeben.

3.2.1 Simulationsmodell der Anlagen

Das Modell vom C4DSI „BHKW + Speicher“ besteht aus mehreren Teilmodellen. Erst einmal aus dem Blockheizkraftwerk (BHKW) und dem Schichtspeicher (Speicher), und darüber hinaus noch einem Spitzenlastkessel (SLK) und einem Heizstab (HS). Das Module SLK soll das BHKW bei hohem Wärmebedarf unterstützen. Die HS werden ausschließlich für die Regelenergievermarktung genutzt, hierbei ist der Begriff Power to heat sehr bekannt.

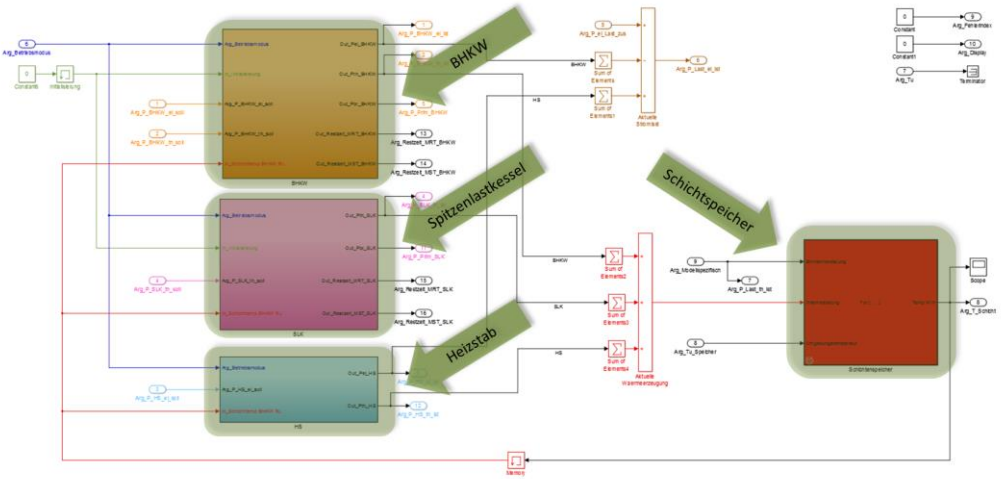


Abbildung 3.2: Das Modell BHKW + Speicher

Ummantelt werden diese Module von einem Anlagenzugang, wo alle Ein- und Ausgänge verwaltet und geroutet werden.

Sinn des Anlagenzugangs ist es, dem System eine Art Maske vorzuschalten. Das System erwartet zwar genau dieselben Eingänge wie der Anlagenzugang, jedoch werden die Dimensionen der Eingänge in dem Anlagenzugang bearbeitet. Bekommt der Anlagenzugang eines BHKWs einen 2-Spaltenvektor an elektrische Stellgrößen, weiß das Modell dass 2 BHKWs simuliert werden sollen.

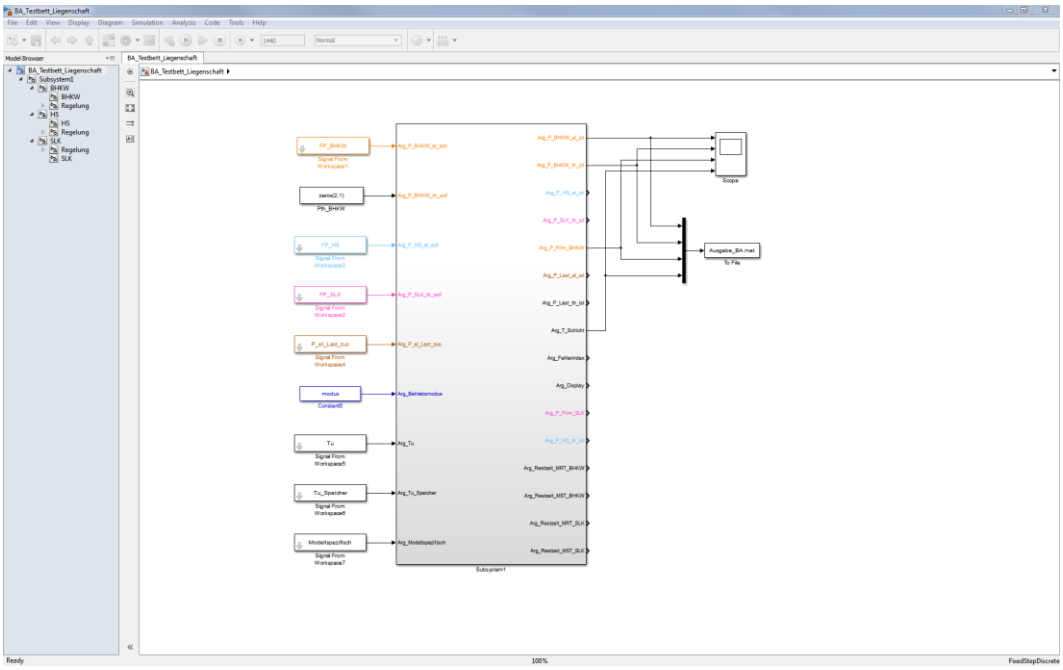


Abbildung 3.3: Das Testbett vom Modell BHKW + Speicher

Das in Abbildung 3.3 dargestellte Testbett bekommt die Eingänge aus dem Matlab Workspace, die benötigten Eingabegrößen liegen als Vektoren in diesem vor. Dieses Testbett dient gleichzeitig als Referenzmodell, hier werden verschiedene Betriebsarten des BHKWs durchfahren und sollen später mit der Simulation auf dem Raspberry verglichen werden.

Die Simulation erwartet aus dem Workspace folgende Werte als Eingänge:

Tabelle 3.1: Erwartete Eingänge der Simulation

Eingang	Bedeutung
FP_BHKW	Sollwerte der elektrischen Leistungen für die BHKWs in kW
Pth_BHKW	Sollwerte der thermischen Leistungen für die BHKWs in kW
FP_HS	Sollwert der elektrischen Leistung für den Heizstab in kW
FP_SLK	Sollwert der thermischen Leistung für den Spitzenlastkessel in kW
P_el_Last_zus	Stromverbrauch der Liegenschaft
modus	Der Betriebsmodus, wie die Anlage gefahren werden soll
Tu	Außentemperatur in °Celsius
Tu_Speicher	Außentemperatur des Speichers in °Celsius
Modellspezifisch	Thermische Last der Liegenschaft

Von der Simulation werden folgende Größen ausgegeben:

Tabelle 3.2: Ausgegebene Größen der Simulation

Ausgang	Bedeutung
P_BHKW_el_ist	Istwerte der elektrischen Leistungen von den BHKWs in kW
P_BHKW_th_ist	Istwerte der thermischen Leistungen von den BHKWs in kW
P_HS_el_ist	Istwert der elektrischen Leistung von dem Heizstab in kW
P_SLK_th_ist	Istwert der thermischen Leistung von dem Spitzenlastkessel in kW
P_Prim_BHKW	Gasverbrauch / Ölverbrauch
P_Last_el_ist	Aktuelle Last (Rechnung zwischen P_el_Last_zus - P_BHKW_el_ist + P_HS_el_ist)
P_Last_th_ist	Modellspezifisch (Thermische Last der Liegenschaft)
T_Schicht	Temperaturen der Speicherschichten in °Celsius
Fehlerindex	Deaktiviert, Fehlercode zum Debuggen (Matlabintern)
Display	Deaktiviert, Funktionsüberwachung (Matlabintern)
P_Prim_SLK	Gasverbrauch/Ölverbrauch SLK
P_HS_th_ist	Istwert der thermischen Leistung von dem Heizstab in kW
Restzeit_MRT_BHKW	Verbleibenden minimalen Einschaltzeiten der BHKWs in Minuten
Restzeit_MST_BHKW	Verbleibenden minimalen Ausschaltzeiten der BHKWs in Minuten
Restzeit_MRT_SLK	Verbleibende minimale Einschaltzeit des SLKs in Minuten
Restzeit_MST_SLK	Verbleibende minimale Ausschaltzeit des SLKs in Minuten

Reale BHKWs bekommen im stromgeführten Modus von der Leitwarte jede viertel Stunde einen Fahrplanwert, der der Anlage vorschreibt, wieviel Leistung sie in den nächsten 15min erbringen soll.

Doch für eine Simulation wären 15 Minutenschritte zu klein, so ändern sich in einem BHKW viele interne Größen innerhalb einer viertel Stunde extrem, so ist zum Beispiel die minimale Laufzeit eines BHKWs auf eine viertel Stunde gesetzt. Auch kann eine Speicherschicht im Speicher, abhängig der Speichergröße und des Speicherfüllstands, innerhalb einer viertel Stunde zweistellige Gradwerte an Temperatur verlieren. Deshalb sollte das Modell im Minutentakt simulieren.

In der Regel hat ein BHKW ein Bedienfeld, die Steuereinheit. Diese Besteht meist aus einem Tastenfeld, bei neueren Modellen ist diese auch als Touchpad realisiert.

Über diese BHKW-Steuerung kann man die wichtigsten Einstellungen vornehmen, die Betriebsart auswählen oder die mithilfe von Analysedaten und –Auswertungen die Anlage überwachen.

Bei diesem Modell geht man weiterhin davon aus, dass diese Steuereinheit mit dem Internet verbunden ist. So kann man das BHKW nicht nur vom Rechner oder Smartphone, sondern auch von einer Leitwarte außerhalb, steuern und

überwachen.

Die Art der internen und externen Datenübertragung der BHKW Steuerung ist zwar vom Hersteller abhängig, doch weit verbreitet ist das Bus-Protokoll Modbus-TCP, das in dieser Arbeit auch Verwendung finden soll.

Eine Besonderheit dieses BHKWs ist die vorgeschaltete Intelligenz, die das Modell im Gegensatz zu einer realen Anlage besitzt. So sind die minimalen Einschaltzeiten und Ausschaltzeiten zwar Empfehlungen des Herstellers, doch werden diese Zeiten bei einer realen Anlage nicht gemessen, sondern sind bei der Simulation durch die Machbarkeit in das Modell mit eingebaut worden.

Somit sind die Anforderungen an das Anlagenmodell die Transformation des Matlab/Simulink Modells in C-Code und die Kompilierung und das Ausführen auf einem Linux-System.

Die Echtzeitfähigkeit des Systems muss ebenfalls hergestellt werden, und die Kommunikationsanbindung muss implementiert werden.

Diese Anbindung sieht die Kommunikation mittels Modbus vor, aber auch einen verschlüsselten Zugang, um der Leitwarte von Hamburg Energie eine gesicherte Verbindung bereit zu stellen.

3.2.2 Gateway

Das Gateway soll als Schnittstelle zwischen dem Fernüberwachungs- und –Steuerungssystem der Leitwarte sowie eines Energieerzeugers, wie hier einem Blockheizkraftwerks, fungieren.

Eine Schnittstelle ist daher von Nöten, da es verschiedene Arten von Kommunikationsprotokollen gibt, und hier zwei sehr unterschiedliche Kommunikationsebenen vorhanden sind. Zum einen ist es der eher einfach gehaltene Modbus, der aus der Industrie kommt, zum anderen das IEC 61850 Protokoll, das die Bedürfnisse der Leitwartenautomatisierung erfüllt.

Das Gateway soll aber nicht nur die Kommunikationsdaten mappen und weiterleiten, sondern auch vorselektieren und komprimieren.
Deshalb sind die Anforderungen an das Gateway neben dem senden und empfangen der Kommunikation mittels Modbus und IEC 61850 auch das Übersetzen der jeweiligen Protokolle. Dazu gehört auch das Speichern der Kommunikation direkt in dem Gateway als CSV-Logdatei, aber auch über das

Internet auf einer externen Datenbank.

3.2.3 Leitwarte

Die Leitwarte, als Teil eines Leitsystems, soll den Menschen bei der Leitung eines Prozesses unterstützen. Sie soll Maßnahmen treffen, um Sollwerte und Sollzustände zu erreichen. Dadurch wird das komplette Verhalten, das heißt die vollständige Steuerung des BHKWs, von der Leitwarte aus geführt.

Es ist also eine zentrale Sammelstelle sämtlicher relevanter Anlagendaten.

Deshalb ist eine direkte Verbindung mit dem Erzeuger wichtig, um zu messen, um zu regeln und zu steuern, um den Betrieb anzuzeigen und um gegebenen falls zu alarmieren. Die Anlage bekommt mit den Leistungsvorgaben der Leitwarte einen Fahrplan, den sie ablaufen wird. Im Gegenzug bekommt die Leitwarte Ist-Werte der Anlage zurück.

Die Anforderungen an diese Leitwarte sind neben dem Empfangen von Simulationsdaten auch das Senden von Stellgrößen.

4 Technische Analyse

Nachdem die Komponenten des Systems und die Aufgabenstellung erläutert wurde, soll in diesem Kapitel eine technische Analyse betrieben und beschrieben werden.

4.1 Kommunikationsstack für IEC 61850

Für die Kommunikationstrecke zwischen Modell und Leitwarte ist ein Gateway vorgesehen, welches die Kommunikationsprotokolle von Modbus-TCP zu IEC 61850 übersetzt. Die RWTH Aachen hat für das Projekt Smart Power Hamburg einen Stack entwickelt, welcher die Kommunikation mit IEC 61850 beherrscht. Da der Stack in Python realisiert ist und in dieser Arbeit benutzt wird, bietet es sich an, Python als Programmiersprache für die komplette Steuerung der Kommunikationsstrecke und zum Aufrufen der Simulation einzusetzen. So können Variablen innerhalb der Python-Umgebung direkt gehandelt werden, und Werte müssen nicht zwischen verschiedenen Skriptsprachen befördert werden.

Vorteile ergeben sich durch die Wahl der Programmiersprache auch auf die Hardwareauswahl, die hier vorgenommen wurde. So ist der Raspberry Pi, der schon durch vorherige Projekte des C4DSIs Verwendung fand, ideal für das Programmieren unter Python.

Der Stack arbeitet mit einer MySQL-Datenbank, um die Werte der IEC 61850-Kommunikation zu verwalten. Solch eine Datenbank wird auch vom C4DSI betrieben und kann im Rahmen dieser Arbeit genutzt werden.

4.2 Analyse des Simulationsmodells im Testbett

Aus der Analyse des vorhandenen Simulationsmodells soll ermittelt werden, ob das Modell echtzeitfähig gemacht werden kann. Die Simulation kann in verschiedenen Modi gefahren werden, wird das Modell auf den „Modus 0“ gestellt, so läuft das BHKW im wärmegeführten Modus, die interne Reglung sorgt dabei für eine stetige Temperaturversorgung des Schichtspeichers mit Hilfe der implementierten Hysterese Regelung. Die Inputdaten von außen werden in diesem Modus nicht beachtet.

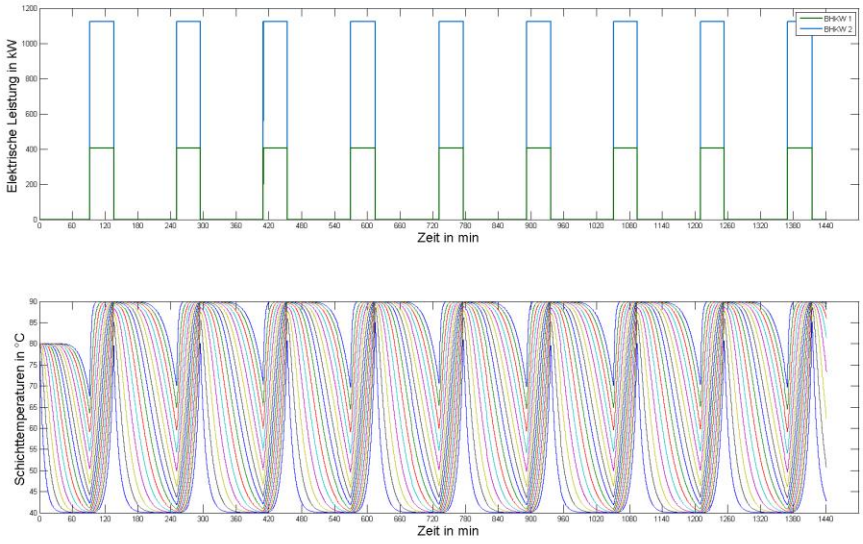


Abbildung 4.1: Ergebnisse einer Simulation im Modus 0 (Wärmegeführt)

In Abbildung 4.1 ist in der oberen Grafik die elektrische Leistung der beiden BHKWs zu sehen, in der unteren Grafik die Sichttemperaturen des Speichers.

Der Regler im Modell betrachtet dabei die Temperatur im Speicher, genauer gesagt die Temperatur der 3. Schicht von unten. Fällt diese Temperatur unter 60°C, werden alle BHKWs mit voller Last angeschaltet, erreicht die 3. Schicht eine Temperatur über 85°C, werden diese wieder ausgeschaltet. In dieser Simulation hat das BHKW 1 eine maximale elektrische Leistung von 405 kW, das BHKW 2 eine maximale elektrische Leistung von 1125 kW.

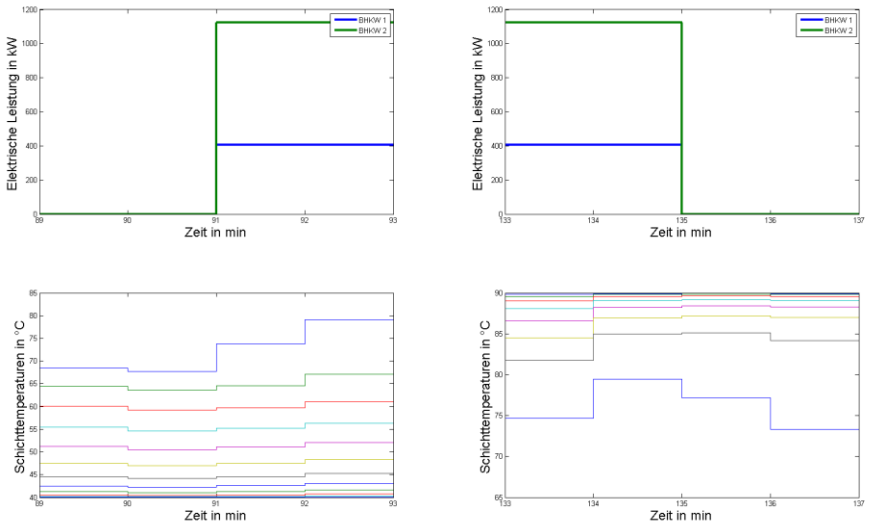


Abbildung 4.2: Geregelte Ein- und Ausschaltung der BHKWs im wärmegeführten Modus

In Abbildung 3.3 sind die Ein- und Ausschaltvorgänge in Bezug auf die Speicherschicht genauer zu sehen, fällt die 3. Schicht unter 60° C, werden die BHKWs eingeschalten, erreicht die Schicht 85°C, werden sie ausgeschalten.

Wird das Modell in dem Modus 2 gefahren, so erwartet es Stellgrößen für die zwei Blockheizkraftwerke. Diese Stellgrößen werden in dem Testbett als Fahrplan hinterlegt. Dieser Fahrplan ist ein Vektor mit minütlichen Sollwerten für die zu vollbringende elektrische Last der BHKWs sowie dem Spitzenlastkessel und für den Heizstab, welche dann später direkt von der Leitwarte aus gesetzt werden sollen. In Abbildung 4.3 ist eine Tagessimulation in diesem Modus 2 zu sehen, die obere Grafik zeigt den Fahrplan der BHKWs als vorgegebene Stellgrößen der elektrischen Leistung, die untere Grafik wieder die Schichttemperaturen des Speichers.

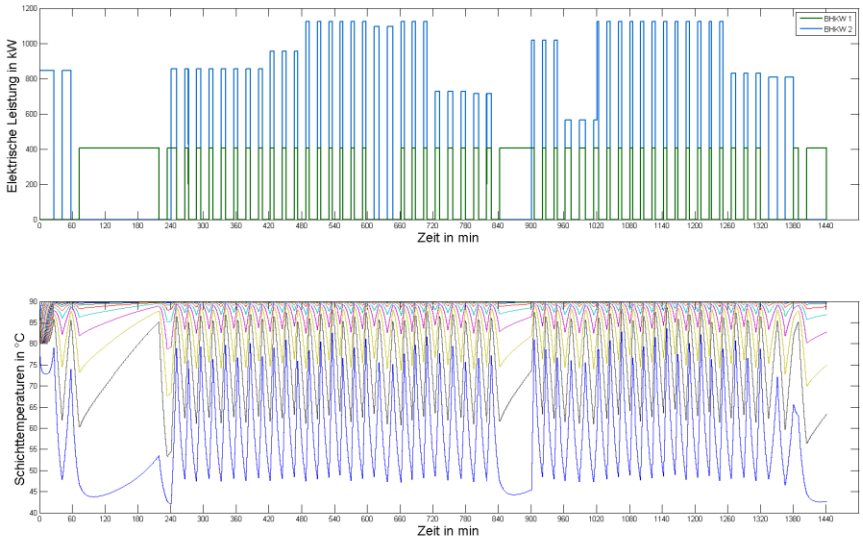


Abbildung 4.3: Ergebnisse einer Simulation im Modus 2 (Fahrplangesteuert)

Das BHKW-Modell braucht auf einem Windows 7 64 Bit Desktop Rechner knappe 3 Sekunden, um eine Tagessimulation von 1440 Schritten zu berechnen.

```
Command Window
>> load('BA_WORKSPACE_TESTBETT.mat')
>> ModelName = 'BA_Testbett_Liegenschaft.slx';
>> tic

sim('BA_Testbett_Liegenschaft.slx')

toc
Elapsed time is 2.892695 seconds.
fx >>
```

Abbildung 4.4: Zeitmessung einer Tagessimulation unter Matlab mit dem Befehl tic-toc

Auf dem Respberry Pi hat das Modell für die Berechnung einer Tagessimulation in Echtzeit einen ganzen Tag Zeit, deshalb sind die 3 Sekunden Simulationszeit von einem PC eine gute Ausgangbasis, um die Simulation echtzeitfähig zu machen.

4.3 Programmiersprache Python

Python wird häufiger als High-Level-Allzweck-Programmiersprache bezeichnet. Die Rechte an Python werden von der Python Software Fundation gehalten, doch ist die Sprache offen und die Entwicklung gemeinschaftlich. Es ist eine

universelle, höhere Programmiersprache die objektorientiert, aber auch funktional, ähnlich der C Programmierung, geschrieben werden kann. Die Skript-Sprache Python hat eine riesige Bibliothek an Modulen, und das Ziel der Sprache ist die Einfachheit und die Übersichtlichkeit der Syntax. Zu sehen ist das an der Programmstruktur, anstatt Klammern oder Schlüsselwörter wird der Code durch Leerzeichen oder Tabulatorzeichen eingerückt.

Von der Raspberry Pi Stiftung ist Python als die offizielle Lehrsprache ausgewählt worden, da es sich um eine moderne, verbreitete, vielfältig einsetzbare und für alle gängigen Plattformen verfügbare Sprache handelt [Bra14].

Unter der Raspbian Distribution, das hier benutzte Betriebssystem der Raspberry Pis, ist Python 2.7.3 schon vorinstalliert.

4.4 Raspberry Pi

Der Minicomputer Raspberry Pi, eine kleine, checkkartengroße Platine, bestückt mit einer Handvoll Bauteilen, wurde im Februar 2012 von der britischen Raspberry Pi Fundation entwickelt. Die 30 Euro teure Platine enthält ein Ein-Chip-System von Broadcom mit einem 700-Mhz-ARM11-Prozessor sowie 512 MB Arbeitsspeicher.

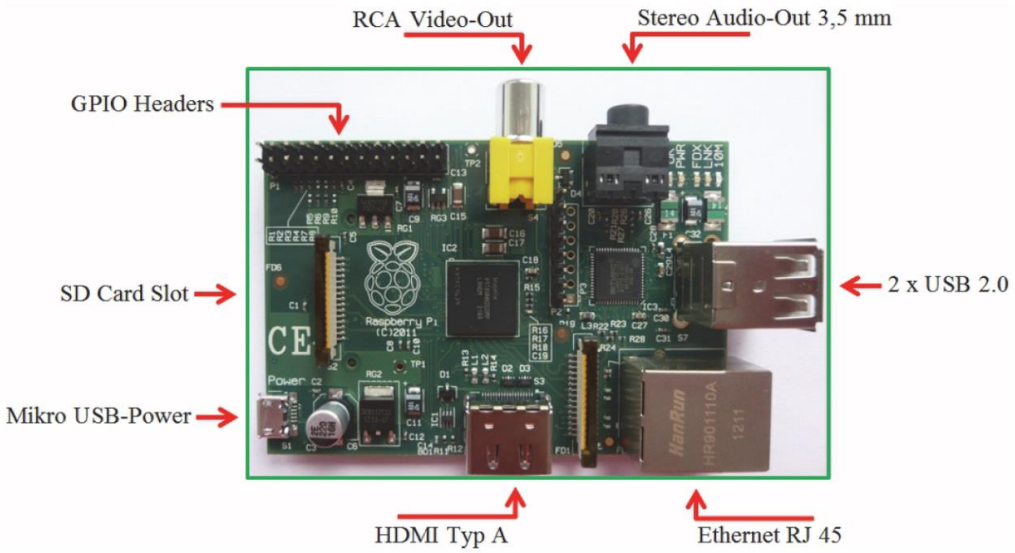


Abbildung 4.5: Die Platine des Raspberry Pi [Bar12]

Das Herzstück des Raspberry Pi-Boards ist der Broadcom Prozessor BCM2835 SoC (System-on-Chip). In diesem Prozessor befinden sich der überwiegende Teil

von Systemkomponenten: die CPU und der Grafikprozessor, aber auch die Audio- und Kommunikationshardware sind in diesem Chip integriert. Obwohl die schon in den 1980er Jahren vom britischen Unternehmen Acorn Computers entwickelte ARM-Architektur (Advanced RISC Machines), ist diese in der Desktop-Welt relativ selten vertreten. Doch seit dem Boom im Embedded-Bereich finden sich ARM-Prozessoren in nahezu allen Smartphones und Tablet-Computern. Durch den einfachen und effizienten Befehlssatz des RISC (Reduced Instruction Set Computer) Designs und seines geringen Stromverbrauchs ist er im Vergleich zu den geradezu stromfressenden, auf CISC (Complex Instruction Set Computer)-Befehlssätzen aufbauenden Desktop-Prozessoren die beste Wahl. So kommt der Raspberry Pi mit einer Stromstärke von 1 Ampere bei 5 Volt aus. Dadurch kommt es, selbst bei komplexen Rechenprozessen, zu einer geringen Hitzeentwicklung, sodass es keinerlei Kühlkörper oder Belüftungen bedarf, und der Stromverbrauch kann über einen Mikro USB-Port gedeckt werden [Bar12].

Die ARM-Architektur unterstützt, im Gegensatz zu den x86-Mikroprozessoren von AMD oder Intel, kein Microsoft Windows. Aber im Gegensatz zu den Closed Source Betriebssystemen wir Apples OS X oder Microsofts Windows baut der Raspberry Pi auf das frei GNU/Linux Betriebssystem, das komplett Open Source ist. Das heißt man kann sich den Quelltext des gesamten Betriebssystems kostenlos herunterladen und ihn beliebig ändern und weiterverbreiten. Es gibt für den Raspberry Pi verschiedene Versionen, sogenannte Distributionen, darunter Debian, Fedora oder Arch Linux.

Der zweite große Baustein ist der LAN-Controller, der für den Netzwerkbetrieb verantwortlich ist. An das Netzwerk wird der Raspberry Pi dann mit dem RJ-45-Anschluss verbunden.

Weitere Anschlüssen gibt es für die Bildausgabe, einen Composite Video- und einen HDMI (High Definition Multimedia Interface) Anschluss. Da der Composite Video Anschluss die Farbsignale Rot, Grün und Blau mischt und über eine einzelne Leitung schickt, die an der Chinchbuchse abgegriffen werden können, ist das Signal sehr anfällig für Rauschen und andere Interferenzen. Es ist nur eine geringe Auflösung mit dieser eher älteren Anschlussart möglich, weshalb das digitale HDMI, wenn man nicht mit einem Röhrenmonitor arbeitet, die beste Art ist, das Anzeigegerät mit dem Raspberry Pi zu verbinden.

Die Stromversorgung wird, wie schon erwähnt, über Mikro-USB angeschlossen.

Zum Betrieb wird 700mA benötigt, um Tastaturaussätze oder hakende Mauszeiger auszuschließen, sollte man aber ein 5 V / 1000mA-Netzteil benutzen. Als Alternative eignen sich auch 4 x AA Batterien.

Für die Peripheriegeräte wie die Eingabegeräte Maus und Tastatur gibt es zwei Universal Serial Bus (USB) Ports. Wenn noch weitere Geräte angeschlossen werden sollten, zum Beispiel ein WLAN Stick, sollte ein Hub mit eigener Stromversorgung mit angeschlossen werden, da der Raspberry Pi sonst an seine Grenzen der Stromversorgung tritt.

Ein Festplattenlaufwerk gibt es bei dem Raspberry Pi nicht, zur Datenspeicherung wird eine SD (Secure Digital) Memory Card verwendet. Auf dieser Karte befindet sich auch das Betriebssystem, deshalb sollte die Größe der SD-Karte mindestens 2 GB (Gigabyte) groß sein.

4.5 Entwicklungsumgebung

Der zur Verfügung gestellte Arbeitsplatz in den Räumen des C4DSI hat einen Desktop-PC. Auf diesem läuft ein aktuelles Windows 7 SP1 mit der Mathwork Studentenversion Matlab 2013a. Auf den Raspberry Pis laufen aktuelle Debian 7 Distributionen. Um eine komfortable Verbindung zwischen dem PC und dem Raspberrys zu gewährleisten sind alle Systeme im selben Netzwerk (mit dem Netzadressbereich 192.168/16), verbunden durch ein Switch. Dieses private Netzwerk kann von außen nicht geroutet werden, der IP-Bereich ist nicht vom Internet aus erreichbar.

Zur Editierung von Quellcodes eignet sich das Tool Notepad++, da eine direkte SFTP-Verbindung zum Raspberry hergestellt werden kann. Ebenfalls praktisch an diesem freien Texteditor ist die bunte Syntax und Strukturhervorhebung von vielen Programmiersprachen.

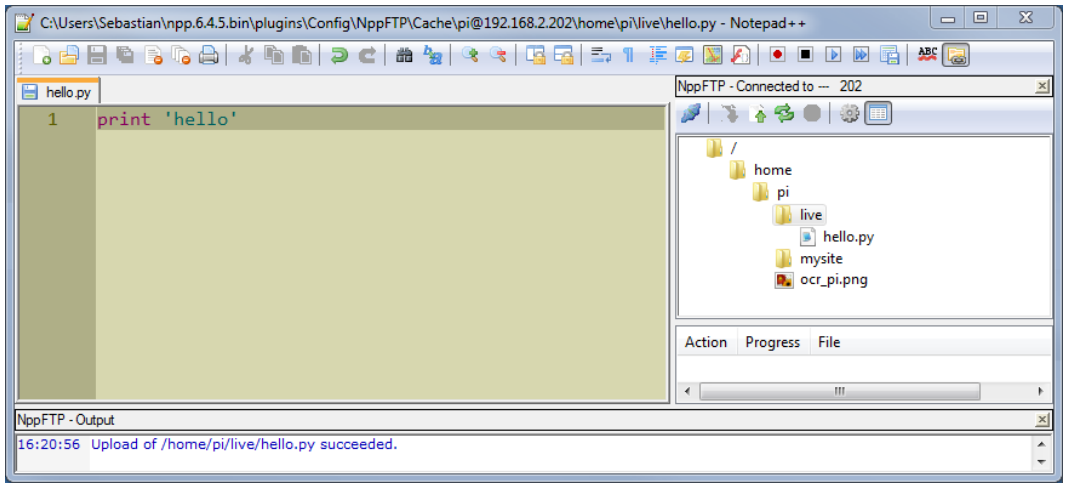


Abbildung 4.6: „hello.py“-Datei geöffnet in Notepad++

In Abbildung 4.6 ist eine Verbindung zu dem Raspberry Pi aufgebaut, die Ordnerstruktur ist im rechten Teil zu sehen. Es kann somit direkt vom PC in die Datei hello.py geschrieben werden, obwohl die Datei auf der SD-Karte des Raspberry Pis liegt.

Um Dateien zu Verwalten und für den Daten- und Dateitransfer hat sich WinSCP als bestes Werkzeug dieser Art bewährt.

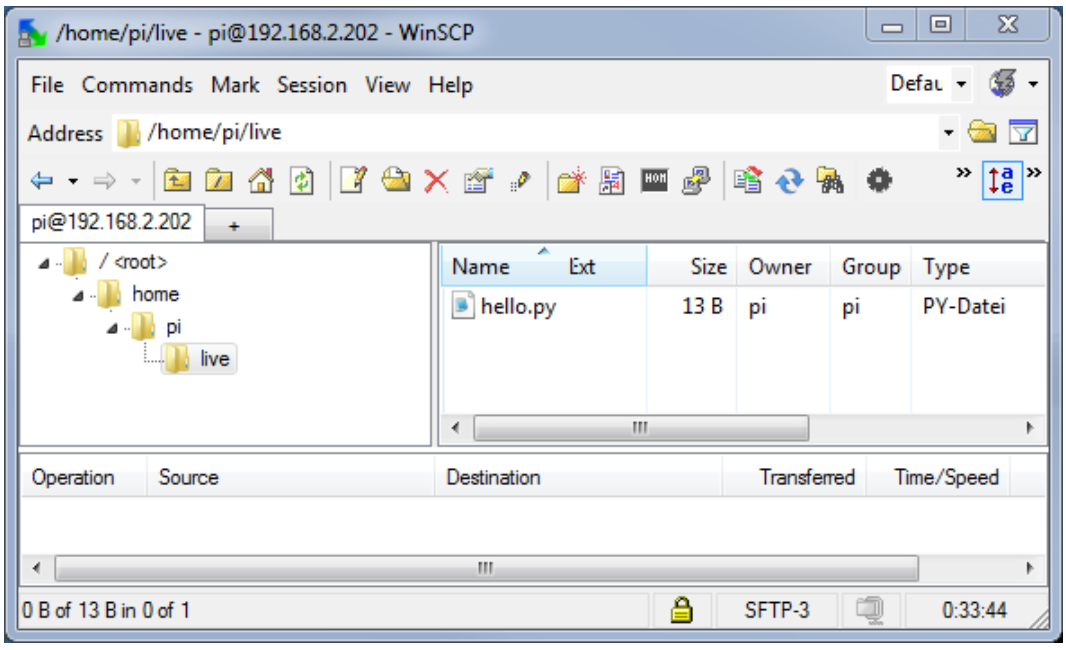


Abbildung 4.7: WinSCP zeigt den Inhalt des Ordners „live“ an

Mithilfe von WinSCP können die typischen Explorer-Befehle ausgeführt werden, wie das kopieren, das umbenennen oder das neu erstellen.

Mit Hilfe des Tools Putty kann von dem PC eine SSH-Verbindung zum Raspberry

hergestellt werden um in einer textorientierten Terminalsitzung direkt Befehle abzusetzen.

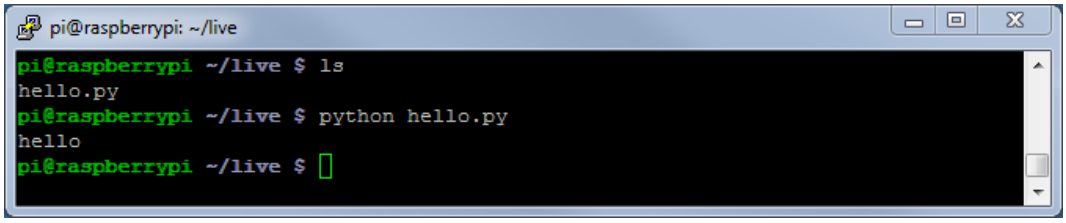


Abbildung 4.8: Putty listet den Ordnerinhalt „live“ auf, danach wird hello.py mit Python gestartet

In Abbildung 4.8 ist eine Terminalsitzung zu sehen, wo der Befehl ls gesendet wird, welcher eine Liste der Dateien im Arbeitsverzeichnis ausgibt. Er entspricht dem DOS-Befehl DIR. Danach wird die Datei hello.py mittels Python gestartet, deren Funktion das ausgeben des Wortes „hello“ auf dem Monitor ist.

Alle drei Tools sind sogenannte frei Software.

5 Implementierung

5.1 Modellexport Matlab/Simulink zu Raspberry Pi

Zum Generieren von C-Code aus einem Simulink-Modell braucht man den Mathwork Simulink Coder, den die Studentenversion von Matlab 2013a beinhaltet. Zwar gibt es auch noch die Möglichkeit mit Hilfe des „Support Package for Raspberry Pi Hardware“ das Modell direkt auf einen Raspberry Pi zu senden, und eine UDP-Verbindung, ein minimales, verbindungsloses Netzprotokoll, zwischen Matlab/Simulink und dem Modell auf dem Raspberry aufzubauen. Doch gibt es bei diesem Package weder die Möglichkeit die Übertragungsstrecke durch ein anderes Protokolle zu ersetzen, noch die Möglichkeit auch ohne Matlab und die damit verbundenen Lizenzen das Modell zu starten oder zu stoppen.

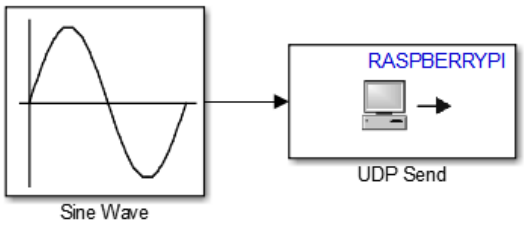


Abbildung 5.1: Matlab Target Hardware hat sich als nicht tauglich erwiesen

Deshalb wird mithilfe des Matlab Simulink Coders das Simulink Modell in C-Code generiert. Der produzierte C-Code ist zwar Plattform unabhängig, dennoch gibt es verschiedene Hardwareeigenschaften, die man vor der Umwandlung in Simulink einstellen muss. Dies sind zum Beispiel die Wortbreiten der Datentypen oder die Byte-Reihenfolge, also die Organisation von Werten im Arbeitsspeicher.

Diese können in Simulink durch den Aufruf der Modell Configuration Parameters eingestellt werden.

Der Raspberry Pi arbeitet mit einem 700-MHz-ARM11-Prozessor, als Betriebssystem eignet sich das offene Raspbian, eine Debiant 7 Distribution.

Dafür sind folgende Einstellungen vorzunehmen:

Anhand eines festgelegten Zeitraums werden in den Solver-Einstellungen dynamische Systemzustände für aufeinanderfolgende Zeitschritte eingerichtet.

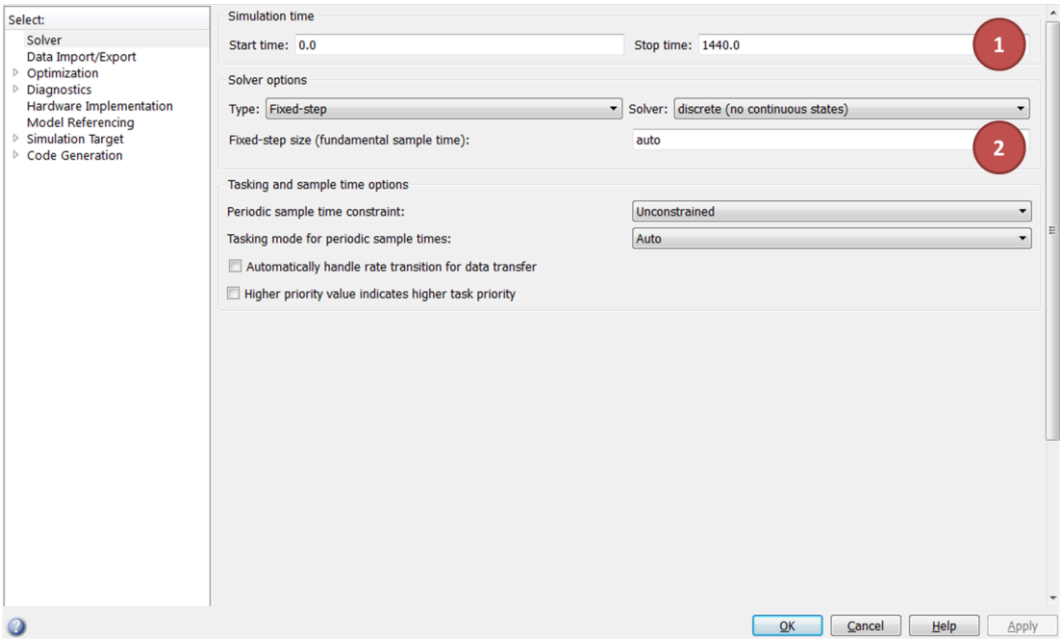


Abbildung 5.2: Die Einstellungen des Solvers unter dem Model Configuration Parameters in Simulink

1 Im Solver-Fenster können die Start und Endzeiten eingetragen werden. Da das Modell als Echtzeitmodell transformiert wird, entspricht die Simulationszeit gleichzeitig der Taktzeit. Eine Simulationszeit von 1440 Schritten entspricht also genau einen Tag. [inf]

2 Auch der Solver-Typ kann ausgewählt werden. Solver sind Komponenten von Simulink, die durch verschiedene numerische Methoden die Differentialgleichungen, die das Modell darstellt, lösen. Das Modell soll später durch Python Schritt für Schritt aufgerufen bzw. simuliert werden, deshalb wird hier eine feste Schrittweite eingestellt. Damit bleibt die Schrittweite die ganze Zeit konstant. Auch soll der nächste Zeitpunkt der Simulation durch das addieren der festen Schrittgröße zur aktuellen Zeit berechnet werden. Deshalb läuft der Solver diskret, ohne Dauerzustände.

In der Hardware-Implementierung werden die Eigenschaften des Zielsystems beschrieben.

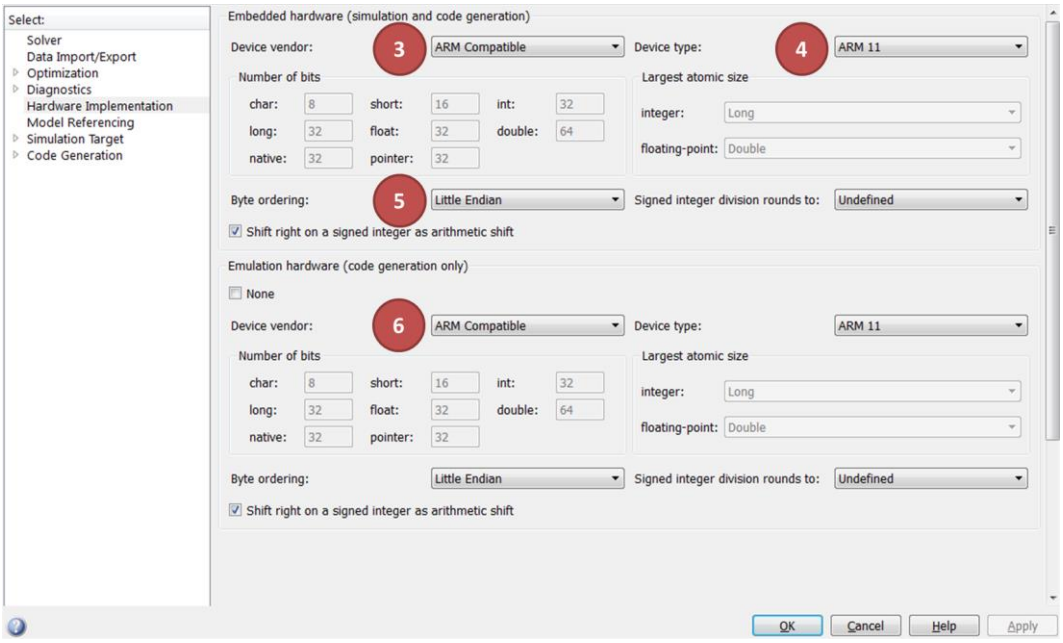


Abbildung 5.3: Die Einstellungen der Hardware Implementierung

3 Da das Modell auf einer anderen Hardware laufen wird als das Matlab-Simulink, ist es nötig, Simulink zu beschreiben, was für Hardware- und Compiler-Eigenschaften für die Implementierung verwendet werden sollen, um den Code so effizient wie möglich zu gestalten. Da es sich bei dem Raspberry Pi um einen ARM11 Device handelt,
wird hier ARM Compatible,

4 Device type: ARM11 ausgewählt.

5 Die Byte-Reihenfolge ist little Endian, die Speicheradressen der Zahlen beginnen mit dem kleinsten Wert, ähnlich der Schreibweise des Datums (Tag.Monat.Jahr), dies ist ein Merkmal der ARM-Speicherorganisation.

6 Diese vorgenommenen Einstellungen gelten sowohl für die Simulation als auch für die Code-Generierung.

Um aus dem Modell Code zu erzeugen, muss in der Code Generation die entsprechende Sprache und der Compiler festgelegt werden.

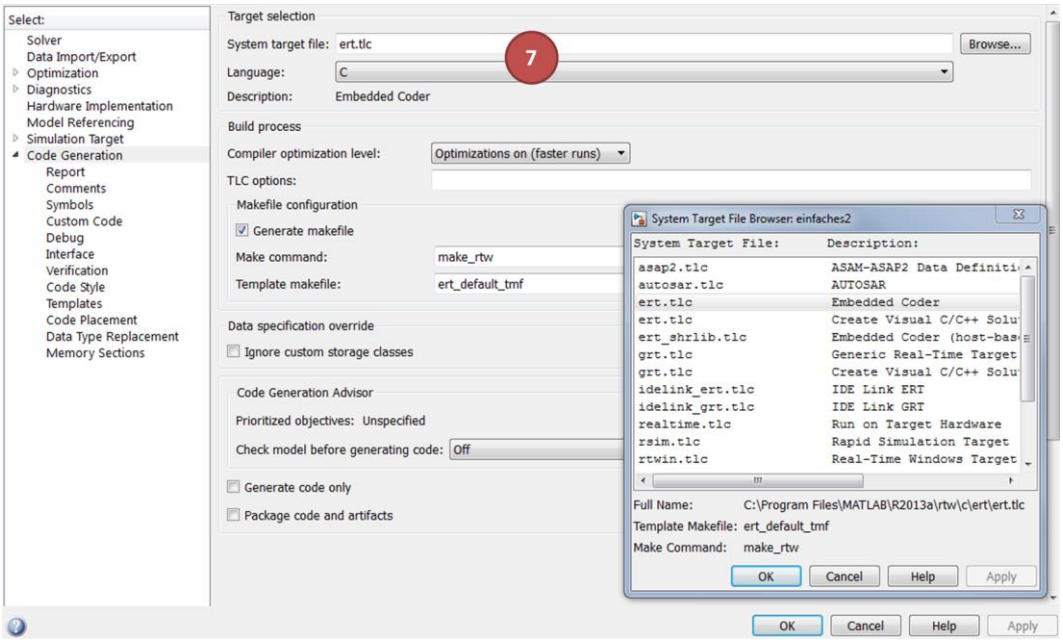


Abbildung 5.4: Die Einstellungen der Code Generation

7 Bei der Code Generierung wird die Zieldatei definiert, die verwendet werden soll. Sie bestimmt die Art des Ziels, für das der Code erzeugt werden soll. Im Fall des Raspberry Pis wird die ert-tlc benötigt, dieses steht für embedded real-time target. Durch diese Einstellung werden drei Funktionen generiert. Diese sind MODEL_INITIALIZE, MODEL_STEP und MODEL_TERMINATE. Die Initialize-Funktion wird einmal am Anfang der Simulation ausgeführt und initialisiert das Simulationsmodel. Die Step-Funktion ist die Funktion, die pro Zeitschritt einmal ausgeführt wird, die Simulationswerte berechnet und bis zum nächsten Zeitschritt behält, und die Terminate-Funktion ist für das richtige Beenden beim Abbrechen der Simulation zuständig.

Zusätzlich wird eine effiziente Echtzeit-Ausführung mit ANSI/ISO C-Code mit Fließkomma- und Festkommadaten erzeugt. Durch das auswählen der Makefile werden alle weiteren Bibliotheken zum kompilieren mit hinzu genommen, die das Modell noch braucht.

Im Untermenü Interface werden weitere Bibliotheken mit eingebunden.

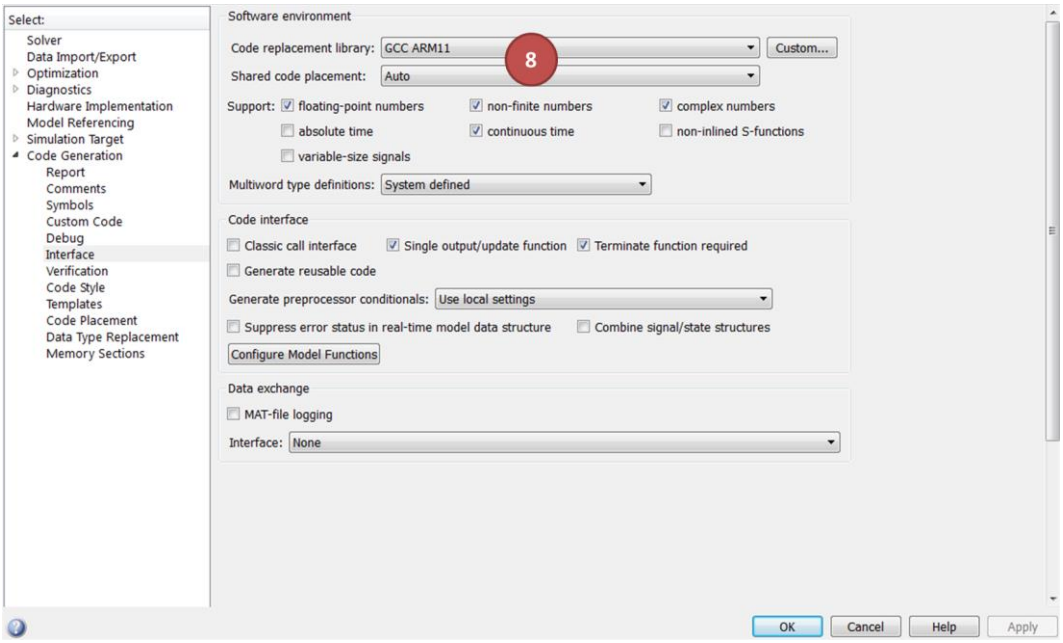


Abbildung 5.5: Die Einstellungen des Interfaces

8

Diese weiteren Bibliotheken sind zum einen die Fließkomma-Bibliothek oder die Bibliothek für Komplexe Zahlen. Dazu gehören aber auch die Bibliotheken mit den einzelnen Parameterwerten, die der genutzte Compiler verwendet. Auf dem Raspberry Pi wird der C-Code mit Hilfe von GCC kompiliert, weshalb hier auch GCC-ARM ausgewählt wird. Der Support von weiteren Zahlenmengen sollte sparsam gewählt werden, besonders die variablen-size Signals können zu Problemen und Kompilierfehlern führen. Die Erzeugung einer Terminate-Funktion ist für das richtige beenden des Skripts erforderlich.

Kommentare im generierten Code helfen beim Verständnis, doch können sie auch die Übersicht erschweren.

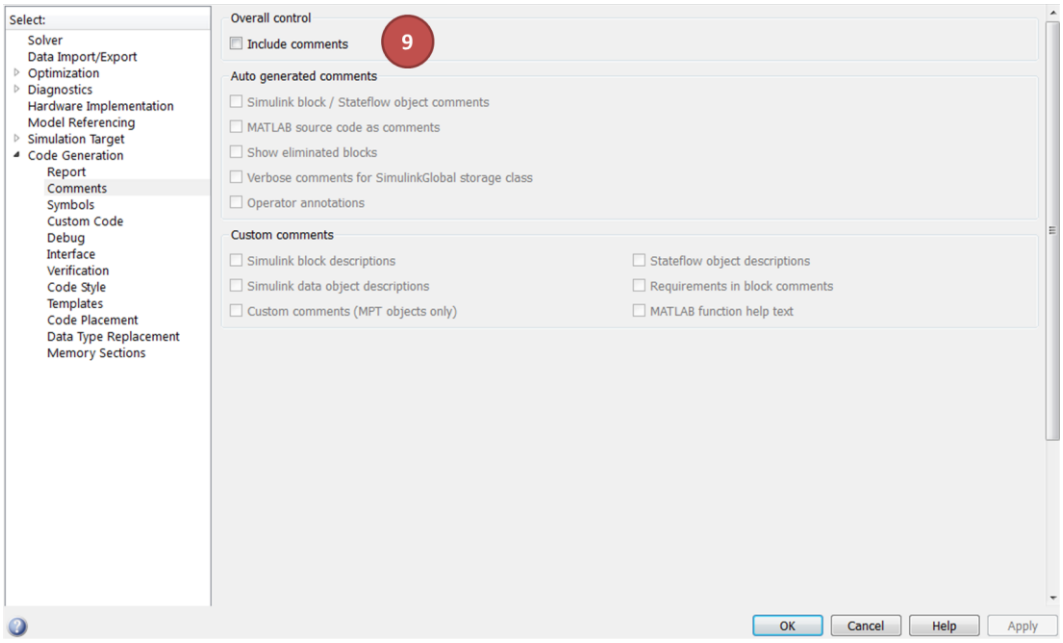


Abbildung 5.6: Die Kommentareinstellung unter dem Modell Configuration Parameters in Simulink

9 Das Miteinbeziehen von Kommentaren bei der Code-Generierung ist eine aufschlussreiche Hilfe, die sie viele Abschnitte im Code erklären, doch werden dadurch viele weitere Codezeilen erzeugt.

Durch diese Einstellungen ist es nun möglich, den von Matlab gebildeten C-Code auf dem Raspberry Pi zu kompilieren.

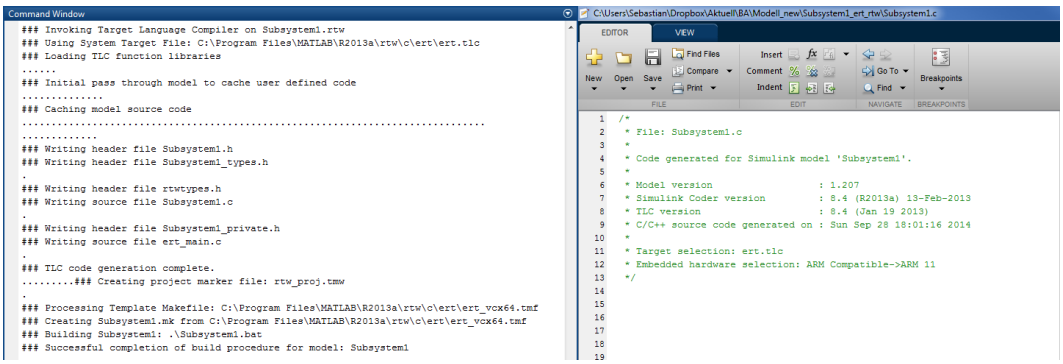


Abbildung 5.7: Successful Completion of build procedure for model

In Abbildung 5.7 ist auf der linken Seite die Command-Window Ausgabe einer durchgeführten Compilierung von Matlab zu sehen, auf der rechten Seite die

Einleitungskommentare der Systemdatei.c

Der C-Code besteht aus mehreren Dateien. Wichtig für die Kompilierung ist die main, in diesem Fall die ert_main.c, sowie die Systemdatei.c und die Systemheaderdatei.h. Wie fast alle Linux-Betriebssystem beinhaltet Raspbian auch die GNU-Pakete der Freien Software Foundation, welche ein offenes Betriebssystem sicherstellen sollen. Unter den GNU-Paketen ist auch die GNU Compiler Collection zu finden, eine Sammlung von Compilern für diverse Programmiersprachen.

Durch das Aufrufen von gcc mit den C-Dateien, die Matlab aus dem Modell generiert hat, stellt der Compiler die Programmiersprache fest, und der entsprechende Sprach-Compilier wird ausgeführt. Die Ausgabe wird dann direkt an den Assembler übergeben, der zum Schluss einen Linker aufruft. Das Ergebnis ist dann ein lauffähiges Programm.

Da gcc über die Kommandozeile aufgerufen wird, ist es recht mühsam Compilierungsfehler ausfindig zu machen, geschweige denn zu debuggen. Einfacher ist es, mit der mitgelieferten OS des Raspbian zu arbeiten.

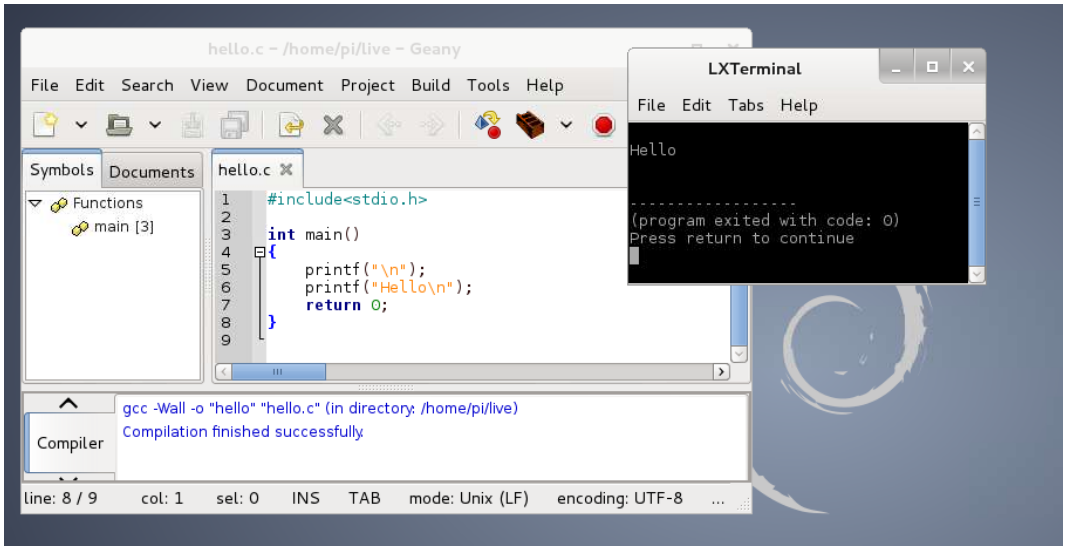


Abbildung 5.8: Die Programmierumgebung Geany

Die von Matlab Simulink erzeugten C-Code Files können nun mit WinSCP auf den Raspberry kopiert werden.

Mit Hilfe einer VNC-Verbindung, eine Verbindung zum Betrachten des Bildschirminhalts eines anderen Rechners, kann nun der Code auf dem Raspberry Pi in der Entwicklungsumgebung Geany geöffnet, und von dort debugged und

kompiliert werden.

In Abbildung 5.8 ist die Programmierumgebung Geany zu sehen, eine für die Bearbeitung von C-Code erstellte Entwicklungsoberfläche, die das editieren und erweitern der C-Dateien erleichtert.

Um nun aus den Quelldateien *.c und den Headerdateien *.h eine Bibliotheksdatei zu erstellen muss zuerst eine Objektdatei erzeugt werden

```
gcc -c -fPIC Subsystem.c rtwtypes.h Subsystem.h
Subsystem_data.c Subsystem_types.h
```

Die Verwendung von –fPIC sorgt dafür, dass ein positionsunabhängiger Code erzeugt wird. Dies verhindert, dass im Maschinencode eine feste Adresse dafür vergeben wird und somit die dynamische Bibliothek an (fast) jeder beliebigen Stelle des Adressraums vom Programm eingebunden werden kann.

Aus der Objektdatei *.o kann nun ein Archiv erstellt werden. Standardmäßig wird eine Archivdatei von statischen Bibliotheken mit einer lib am Anfang des Namens gekennzeichnet.

Dazu gibt es den Befehl ar.

```
ar crs libSubsystem.a Subsystem.o
```

Die Option r sorgt dafür, dass im Falle einer bereits vorhandenen Bibliothek diese von der neuen ersetzt wird. Die Option c erzwingt das Anlegen des Archivs, falls es diese noch nicht geben sollte, s fügt der Bibliothek noch einen Index hinzu. Mithilfe des Indexeintrags werden die Zugriffe auf Funktionen im Archiv optimiert. Dies kann aber auch nochmal separat durch den Aufruf ranlib erledigt werden.

```
ranlib libSubsystem.a
gcc -shared -Wl,-soname,libfak.so.1 –o
libfak.so.1.0 fak.o -lc
```

Mit -shared wird angegeben, dass eine dynamische Bibliothek erzeugt werden soll. -Wl leitet die Optionen an den Linker weiter.

Für -soname wird der vollständige Bibliotheksname mit den Versionsnummern angegeben. Mit libfak.o wird die Objektdatei angegeben, die in dieser dynamischen Bibliothek aufzunehmen ist.

Hierbei können selbstverständlich mehrere Objektdateien aufgelistet werden.

Mit -lc können noch weiter C-Bibliothek hinzugefügt werden, was man öfter

machen muss. Natürlich können auch hierbei noch mehr Bibliotheken hinzugelinkt werden.

```
lib(name).so.(Hauptversionsnr.)(Unterversionsnr.)
(Releasenr.)
```

5.2 Python-Skripte für den Simulationsserver und das Gateway

Jeder Raspberry Pi besitzt ein Python Skript, das zu Beginn gestartet wird.

Diese Skripts werden anhand Programmablaufpläne in den Unterkapiteln 5.21 und 5.22 beschrieben.

5.2.1 Server-Skript

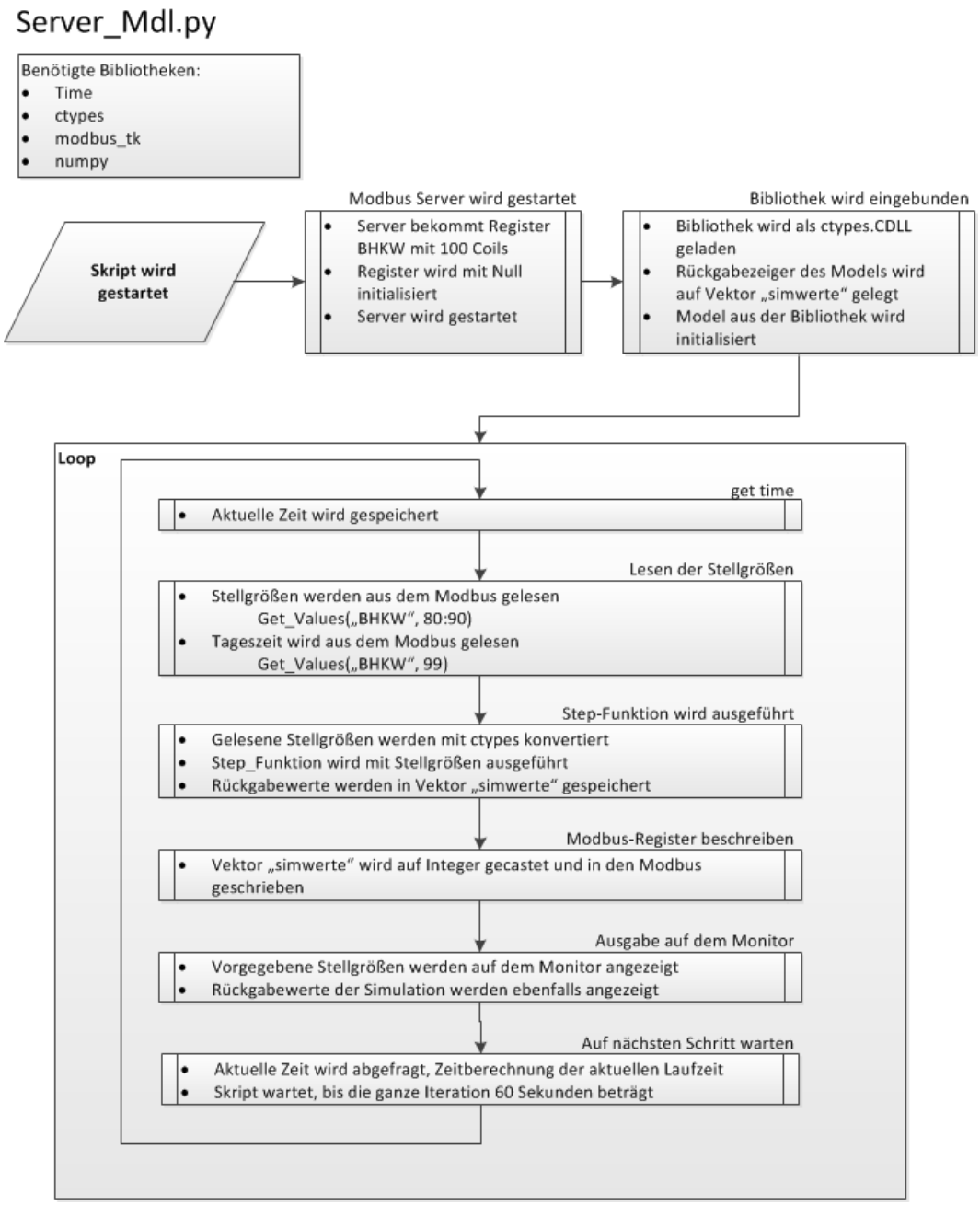


Abbildung 5.9: Programmablaufplan des Raspberry-Simulationsservers

Wichtig für die Echtzeitfähigkeit ist die Bibliothek Time, welche Befehle wie now = time.time() und time.sleep(0.001) beinhaltet. So kann die aktuelle Zeit exakt abgefragt werden und wie in den Skripts durchgeführt, das System kurz ruhen zu lassen, bis eine bestimmte Zeit erreicht ist. So ist es möglich, in der Schleife, die die Step Funktion aufruft, mit der Bedingung

```
while ((60. - (time.time() - now)) > 0):time.sleep(0.001)
```

eine Puffer zu erzeugen, der dafür sorgt, dass die Schrittweite genau eine Minute lang ist, bzw. mit einer Abweichung bis zu maximal einer Millisekunde.

Um C-Code und Bibliotheken, die in C geschrieben sind, in anderen Programmiersprachen zu nutzen, gibt es diverse Sprachanbindungen (language binding). Beispielsweise bietet ctypes die Möglichkeit, die in C geschriebene Bibliothek in Python zu benutzen. Somit ist es möglich mit Python auf dem Raspberry die Simulation mit den Stellgrößen des Gateways aufzurufen, auszuführen und abzufahren. Die aufgerufene Bibliothek der Simulation kann auch Rückgabewerte ausgeben, diese werden dann mittels dem restype Befehl von Ctype als Vektor entgegengenommen.

```
lib = ctypes.CDLL("/home/pi
     /Liegenschaft ert rtw/libLiegenschaft.so")
lib.Liegenschaft_step.restype = ndpointer
     (dtype=ctypes.c_double, shape=(39,)
```

Wichtig ist hier allerdingst das Wissen über die Größe und den Datentyps, welches die aufgerufene Bibliothek zurückgibt. Alle Werte mit der die Bibliothek arbeitet, werden dabei auf spezielle, an ctypes angepasste Datentypen gecastet, diese Datentypen werden mit dem Modul zusammen geladen. So sieht der Cast einer Double Variable P_el_Last_zus aus:

```
Ct_P_el_Last_zus = ctypes.c_double(P_el_Last_zus)
```

5.2.2 Gateway-Skript

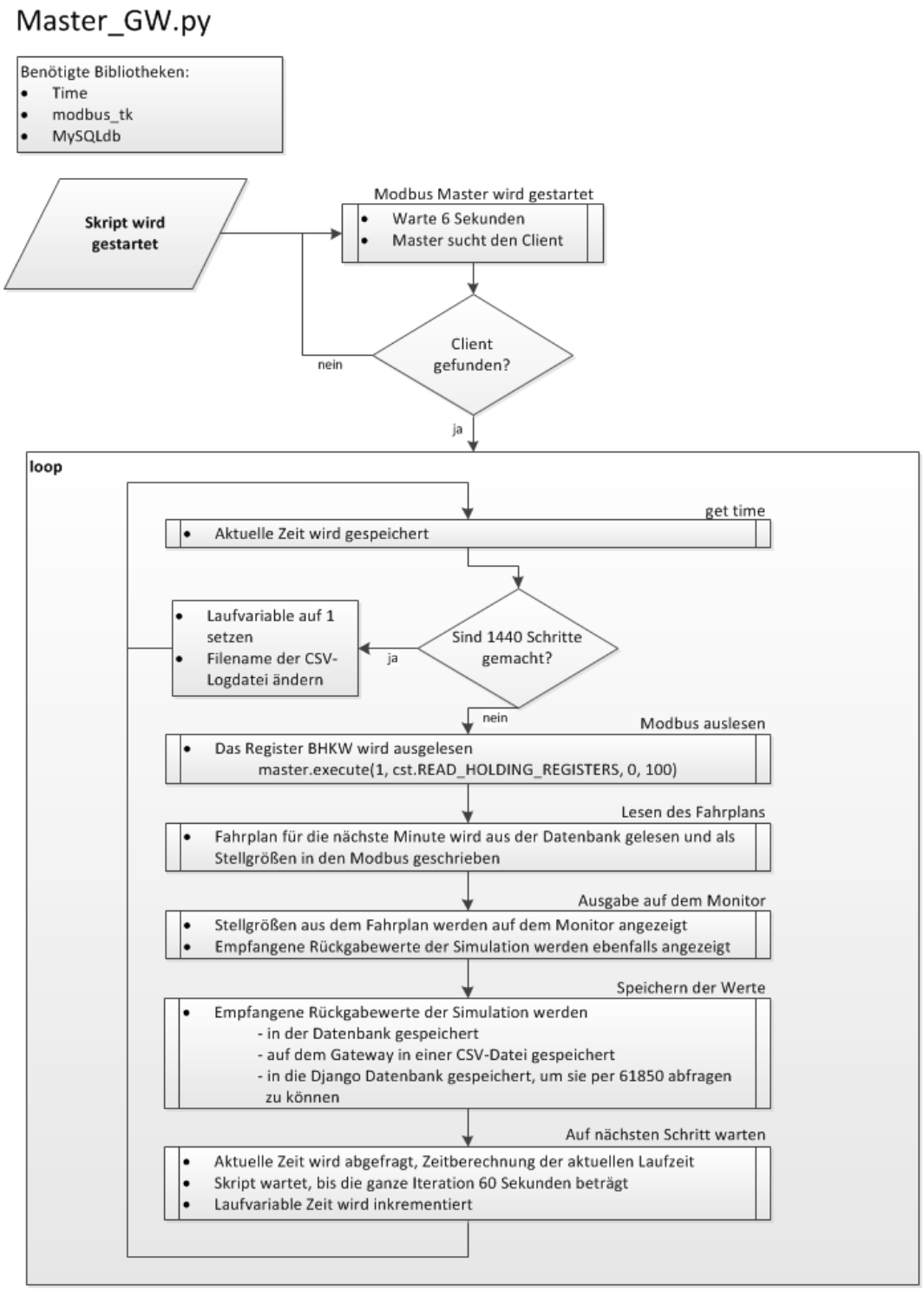


Abbildung 5.10: Programmablaufplan des Raspberry-Gateways

Dieses Skript liegt auf dem Gateway und stellt die Kommunikationsstrecke zwischen Simulation und Leitwarte. Das Skript braucht neben der Time-

Bibliothek und der modbus-Bibliothek auch noch die Bibliothek mySQLdb zum Einlesen des Fahrplans aus der Datenbank, sowie das Speichern der Simulationswerte in diese.

Um Pythoncode beim Systemstart automatisch zu starten, kann die rc.local-Datei im etc-Ordner angepasst werden (/etc/rc.local)

```
/home/pi/Master_Gwy&
exit 0
```

Das &-Zeichen sorgt für das Ausführen des Skripts im Hintergrund, da rc.local eigentlich sequenziell abgearbeitet wird. Durch das abschließende &-Zeichen wird nicht auf das beenden des Befehls gewartet, sondern direkt der nächste ausgeführt. So kommt es zu keiner Blockade weiterer Befehle.

5.3 Kommunikation mittles Modbus

Für eine Kommunikation mittels Modbus wurde das Modul modbus.tk implementiert. Dieser freie Modbus Interpreter steht unter der GNU-Lizenz und wurde von Luc Jean aus Frankreich geschrieben. Mit diesem Modul ist es möglich einen Server aufzubauen, aber auch einen Master zu konfigurieren, der den Modbus auslesen und beschreiben kann.

Der Server wird über die Konsole mittels

```
server = modbus_tcp.TcpServer(address="141.22.122.202")
```

gestartet, der Client greift mit dem Befehl

```
master = modbus_tcp.TcpMaster(host="141.22.122.202")
```

auf dem Server zu.

In diesem Projekt wird vom Simulationsserver ein Modbus-Server aufgebaut, welche ein Register von 100 Coils besitzt. Ein Coil entspricht dabei einem Datum von 2 Byte. Dazu baut der Server einen slave auf

```
slave1 = server.add_slave(1)
```

welcher den Namen BHKW bekommt, 100 Coils lang ist und bei Adresse 0 startet

```
slave1.add_block("BHKW", mdef.HOLDING_REGISTERS,0, 100)
```

Modbuswerte können von jedem Teilnehmer im Netzwerk ausgelesen werden, eine Authentifizierung findet nicht statt. So kann auch von dem Desktoprechner mittels Modbus-Software das Register ausgelesen werden.

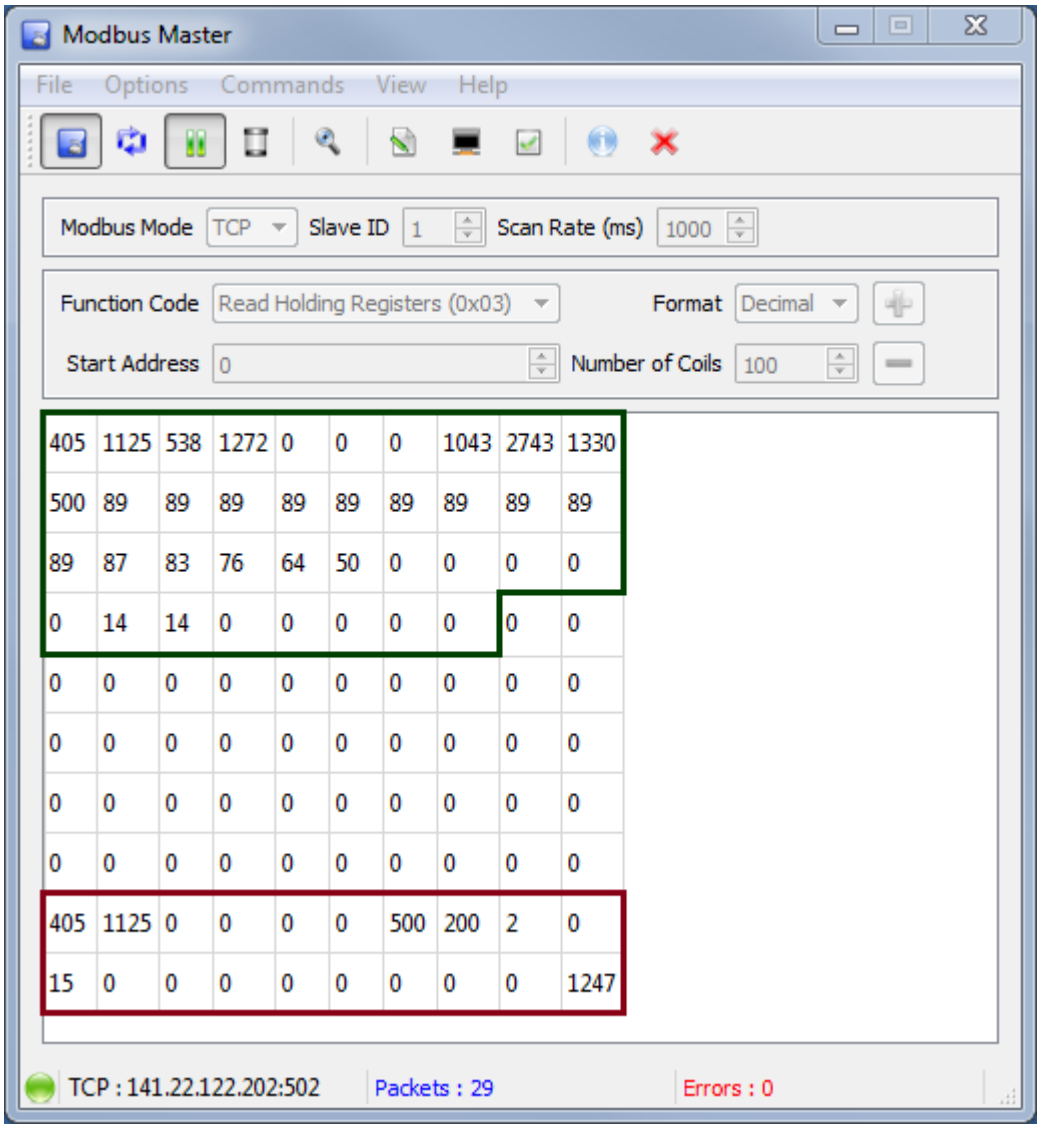


Abbildung 5.11: Datenanordnung des Modbus TCP/IP

Wie in Abbildung 5.11 zu sehen werden innerhalb des Registers BHKW die Simulationswerte vom Server hineingeschrieben (grün umrandet, vgl.: Tabelle 3.2: Ausgegebene Größen der Simulation), und vom Gateway ausgelesen.

Auch werden die Stellgrößen vom Gateway in das Register geschrieben (rot umrandet, vgl.: Tabelle 3.1: Erwartete Eingänge der Simulation), die wiederum der Server ausließt und damit den nächsten Simulationsschritt ausführt.

5.4 Kommunikation mittels IEC 61850

Die Kommunikation über IEC 61850 wird mit Hilfe eines Django Stacks durchgeführt, welcher von der RWTH Aachen in Zusammenarbeit mit der FGH Mannheim entwickelt wurde.

Das quelloffene Django Framework kommt eigentlich aus der online Berichterstattung und ist auf schnell wechselnde und große Mengen an Content spezialisiert. Installiert wird es nicht mittels des Advanced Packaging Tools (apt-get), sondern mit dem python package manager, welcher zuerst installiert werden muss:

```
sudo apt-get install python-pip
sudo pip install django
sudo apt-get install python-django-piston
```

Das Modul Piston wird für die REST-Kommunikation benötigt, da keine dynamisch erstellten Seitenlinks abgerufen werden, sondern die Simulationswerte über eine statische, unveränderbare URL abgefragt werden. Dabei hilft Piston.

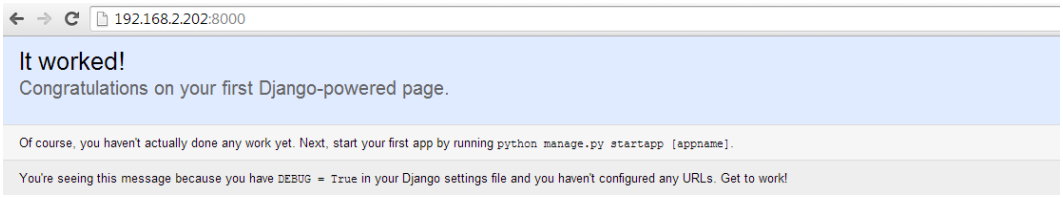


Abbildung 5.12: Die Willkommensseite eines frisch erstellten Django Projekts

Es wird mittles

```
django-admin.py startproject IEC61850SPH
```

ein neues Djangoprojekt erstellt (Abbildung 5.12), und entsprechende Konfigurationsdateien durch die Konfigurationsdateien der RWTH ersetzt.

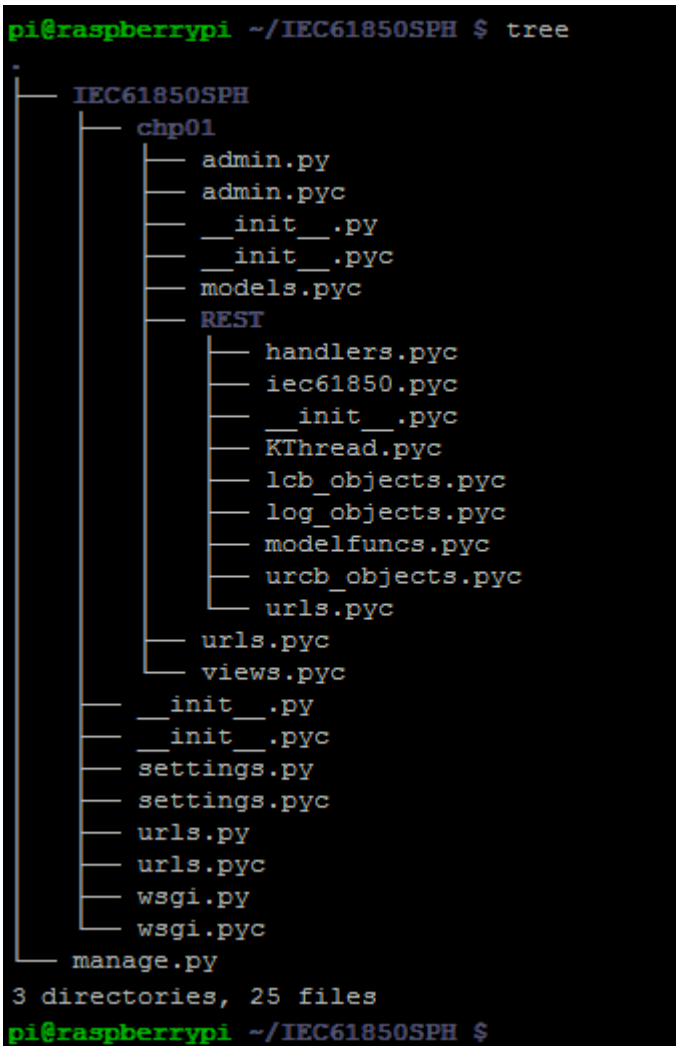


Abbildung 5.13: Die Projektdateien des Django Projekts IEC61850SPH

Django wird über einen Server betrieben, was in diesem Fall das Gateway ist. Von da aus werden die Session-Handlings und die URL-Mappings durchgeführt.

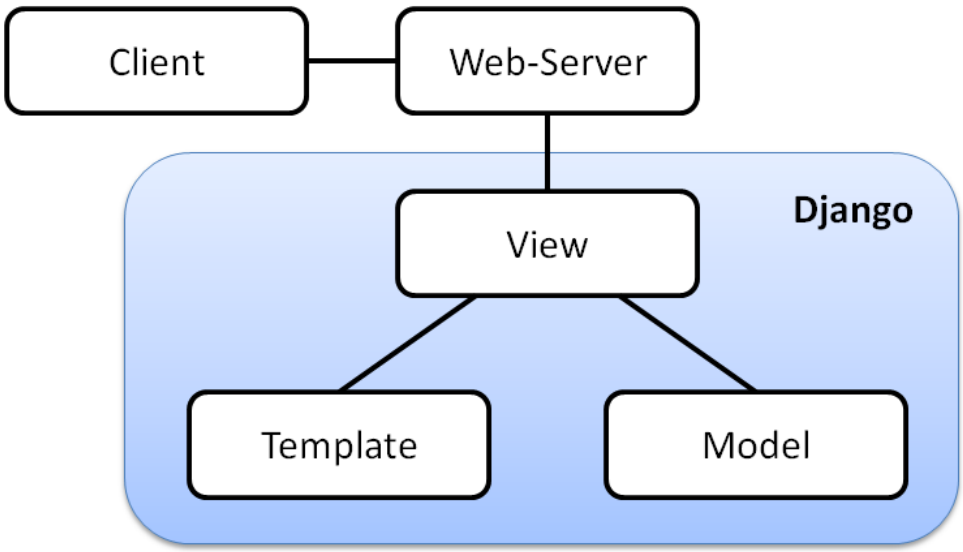


Abbildung 5.14: Aufbau von Django

Der Controller ist bei Django der View, dieser dient als Schnittstelle zwischen Django und dem Client. Es ist der sichtbare Zugriffspunkt für den Benutzer.

Die Aufgabe des Views ist das entgegennehmen von Anfrage und entsprechende Antworten zu generieren.

Die Anfragen werden vom Client als URL gestellt, und als Antwort wird ein "HTTPRequest"-Objekt erzeugt. Dies passiert mittels einer Template, eine Vorlage der zu generierenden Antwort, die mit den gefragten Werten gefüllt wird.

Diese Werte kommen aus dem Model, einer Art Tabelle, die aus Klassen und Attributen besteht. Das Modell in diesem Projekt ist eine SQL-Datei, die aus folgenden 61850-Knoten (Tabelle 5.1) und Datenobjekte samt deren Attributen besteht.

Tabelle 5.1: Ausgewählte Rückgabewerte für IEC 61850

Datenbezeichnung	Knoten	Attribut
Elektrische IST_Leistung BHKW	MMXU	TotW
Thermische IST_Leistung BHKW	DCTS	ThrmOut
Thermische IST_Leistung SLK	DCTS	ThrmIn
Verbleibenden minimalen Einschaltzeiten der BHKW	DRCT	StrDITms
Verbleibenden minimalen Ausschaltzeiten der BHKW	DRCT	StopDITms

5.5 Datenbankanbindung

Zur Ausstattung des C4DSI gehört ein Datenbankserver, der im Netz der HAW Hamburg erreichbar ist. Auf diesem Datenbankserver läuft eine MySQL Datenbank, wo ein Tabel „Fahrplan“ eingerichtet wurde. Dort befinden sich Minutenwerte für zwei BHKWs, für einen Spitzelastkessel und einen Heizstab, die vom Gateway abgefragt werden können und an das Modell als Stellgrößen weiter geleitet werden. Als Primary-Key dient ein Integer Wert, welcher die Tageszeit darstellen soll.

Abgefragt wird der Fahrplanwert durch das Gateway mit dem SQL-Statement:

```
cursor.execute("SELECT * from Fahrplan
     where Tageszeit = %s", zeit)
```

Auf dieser Datenbank befindet sich auch das Table Simulationsdaten, unter dem die vom Simulationsserver erhaltenen Simulationswerte mittels des SQL-Befehls „Insert“ gespeichert werden.

Auch die SQL-Datei, die für den Django-Stack benutzt wird, befindet sich auf der Datenbank.

5.6 Direktanbindung an die Leitwarte Hamburg Energie

Unter dem Projekt Smart Power Hamburg soll es Hamburg Energie möglich sein, errechnete Simulationsdaten abzufragen und Stellgrößen zu setzen, damit eigene entwickelte Algorithmen getestet werden können. Da aber eine Leitwarte ständige Verfügbarkeit sowie Zuverlässigkeit gewährleisten muss, hat die Sicherheit einen hohen Stellenwert. Die Leitwarte, als Mensch-Maschinen-Schnittstelle, hat direkten Einfluss auf die Prozess- und Anlagensteuerung. Um jeglichen Missbrauch von außen zu verhindern ist die IT-Infrastruktur netztechnisch größtmöglich isoliert, jedoch nicht komplett abgeschottet.

Deshalb erfordert die Fernwartung von Anlagen sowie die Steuerung von Smart Grids eine abgesicherte Netzanbindung. Abgesichert gegen unbefugten Durchgriff und gegen gezielte Angriffe. Hamburg Energie nutz deshalb ein eigenes Netz, welches von außen nicht erreichbar ist. Mittels eines Virtuellen privaten Netzwerkes (VPN) ist es aber möglich, diesem Netz auch von Außerhalb beizutreten. Es wird eine verschlüsselte Verbindung aufgebaut, die einem dann als Netzteilnehmer den vollen Zugriff gewähren und es erlauben, eine stabile und

sichere Kommunikation über das Internet zu führen. Grund dafür ist das Nutzen von virtuellen Verbindungen, die das Abhören oder das Verändern von Daten unmöglich machen. Gesichert werden diese VPN-Verbindungen durch ein Tunneling-Verfahren, bestehend aus Zertifikaten, Schlüsseldateien, Passwörtern und kryptographischen Hashfunktionen. Genutzt wird hier die Internet Protokoll Security (IPsec), ein Protokoll-Stack, der es unter Linux erlaubt, verschlüsselte Verbindungen aufzubauen und Punkt-zu-Punkt bzw. Tunnelverbindungen zu verwalten.

Die IPsec Verbindung sendet alle Daten, die an die Ethernet-Schnittstelle gesendet wird, direkt an einen weiteren Teilnehmer, an den Teilnehmer am Ende des aufgebauten Tunnels. Will der Teilnehmer nun mit dem Server kommunizieren, der den Modbus bereitstellt, müssen beide ohne Umwege verbunden sein. Ansonsten würde der Teilnehmer nur die Kommunikation zwischen dem Gateway und dem Server sehen, könnte aber nicht direkt an der Kommunikation teilnehmen.

Auszug aus der Security Policy Database (SPD) des ipsec-tools.conf Datei:

```
spdadd 10.1.1.0/24 10.1.3.0/24 any -P in
ipsec esp/tunnel/80.152.166.90-41.22.122.140/require
```

Bei der obigen Sicherheitsmethode werden alle Datenpakete, die von Hamburg Energie (IP: 80.152.166.90) kommen, als Adresse 10.1.1.0/24 gehandelt, und alle Datenpakete, die als Absender 141.22.122.140 (Simulations-Raspberry) haben, sollen die Adresse 10.1.3.0/24 bekommen. Somit liegen beide, der Simulations-Raspberry sowie der Teilnehmer von Hamburg Energie, in derselben Range und können durch einen Tunnel kommunizieren.

Dieselbe Regel gilt auch für die andere Richtung:

```
spdadd 10.1.3.0/24 10.1.1.0/24 any -P out
ipsec esp/tunnel/141.22.122.140-80.152.166.90/require;
```

6 Test und Bewertung

Das Verhalten einer exportierten Matlab/Simulink Simulation auf einem Linux-Rechner läuft problemlos. Die Voreinstellungen für die Kompilierung in Simulink passend zu dem ARM-basierten Raspberry Pi sind zwar sehr komplex (Siehe Kapitel 5.1), doch wenn sie richtig eingestellt sind, verhalten sie sich problemlos.

Das einzig nicht lösbare Problem war die automatische Erstellung der Parameterübergabe bei der Stepfunktion. Matlab bietet eine „Get Default Configuration“ an, um den Aufruf der Funktion, die immer einen Schritt geht, in diesem Fall die Funktion, die mit den vorgegebenen Stellgrößen eine Minute lang simuliert, zu erstellen.

Diese Automatisierung klappt so nicht, was wohl mit einer Verdrehung der Datentypen im Modell zusammenhängt. So sind manche Ausgangsvektoren Spaltenvektoren, die auf höheren Ebenen Zeilenvektoren sind. Die Vermutung ist, dass diese Transformation von Simulink automatisch korrigiert wird, das exportierte Modell dieses aber nicht schafft.

Abhilfe schafft nur das manuelle Anpassen der Step-Funktion im C-Code, vor der Kompilierung.

So wurde aus

```
void Liegenschaft_step(void)
```

dann

```
real_T * Liegenschaft_step (real_T arg_P_BHKW_el_soll[2],
     real T arg P BHKW th soll[2], real T
     arg_Modellspezifisch, real_T arg_P_el_Last_zus,
     real_T arg_Betriebsmodus, real_T arg_P_SLK_th_soll,
     real_T arg_Tu, real_T, arg_Tu_Speicher,
     real_T arg_P_HS_el_soll)
```

Am Ende der Funktion werden die Simulationsergebnisse in einem Vektor gelegt, der dann mit „return“ als Rückgabewert bzw. als Funktionsantwort zurückgegeben wird.

Somit ist es leider nicht ganz möglich, die Kompilierung komplett zu automatisieren, und es sind grundlegende C-Kenntnisse erforderlich, um die Ergänzungen an der richtigen Stelle einzufügen.

Da der Fehler aber in der Modellarchitektur vermutet wird, kann es durchaus sein, dass dieser Fehler noch korrigiert wird.

Um die Simulation aus dem Testbett und die Simulation, die auf dem Raspberry Pi läuft, gut zu vergleichen, eignet sich die Betrachtung der Temperaturen der Speicherschichten. Der Speicher sitz, wie in Abbildung 3.2: Das Modell BHKW + Speicher gut zu erkennen, hinter dem BHKWs und hinter den Spitzenlastkesseln mit seinem Heizstab. Der Speicher ist somit ein Indikator des Verhaltens der Erzeuger und zeigt daher die Stellgrößen, welche die Simulation bekommt, gut an.

Der Speicher besteht aus 15 Schichten, somit können ziemlich genau Vergleiche zwischen den Simulationen gezogen werden.

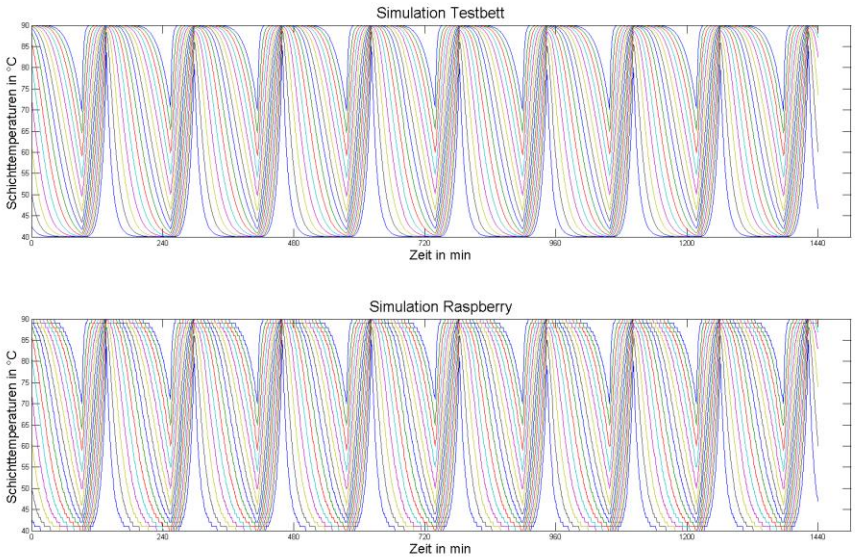


Abbildung 6.1: Vergleich von Tagessimulationen im geregelten, wärmegeführten Modus

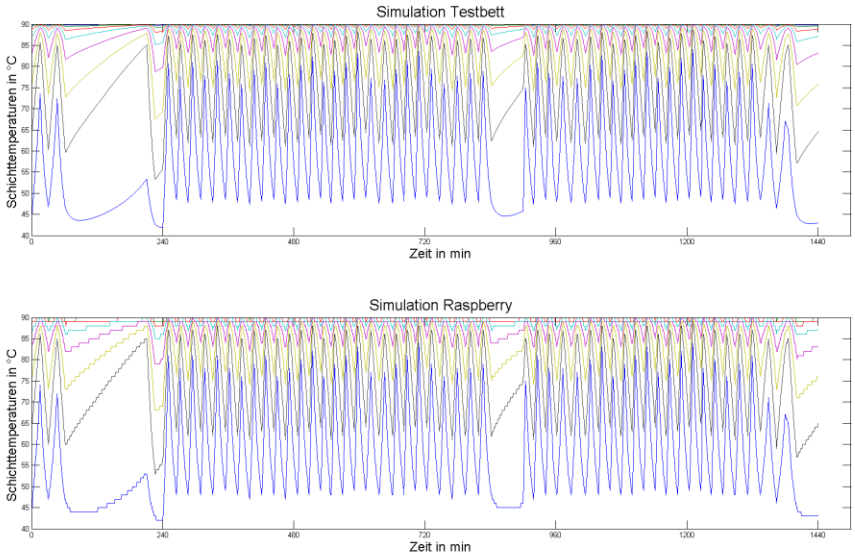


Abbildung 6.2: Vergleich von Tagessimulationen bei einer konstanten thermischen Last von 500 kW

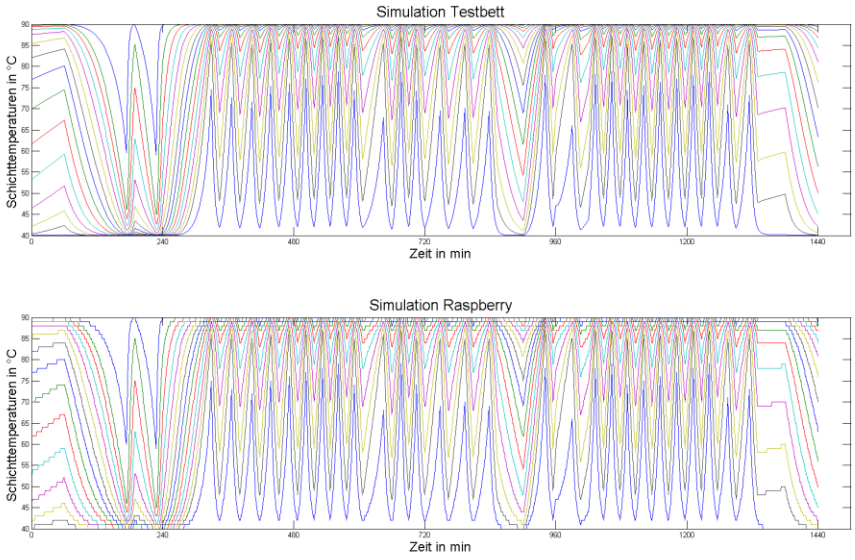


Abbildung 6.3: Vergleich von Tagessimulationen bei einer konstanten thermischen Last von 900 kW

Einzig die Rundungsfehler, die beim Senden über den Modbus auftauchen, da dieser nur ganzzahlige Werte verarbeitet, müssen berücksichtigt werden.

6.1 Aufgabenstellung Anbindung Hamburg Energie

Die Anbindung an die Leitwarte soll nicht über das Protokoll 61850 erfolgen, da bei Hamburg Energie dafür nicht die passenden Ressourcen zur Verfügung

stehen. Vielmehr sollen die Simulationswerte mittels einer verschlüsselten Verbindung direkt aus dem Modbus gelesen werden.

Da hierfür die Security-Suite IPsec benutzt wird, ist eine direkte Verbindung zwischen Hamburg Energie und dem Modbus Master notwendig. Nur so kann direkt aus dem Modbus gelesen werden.

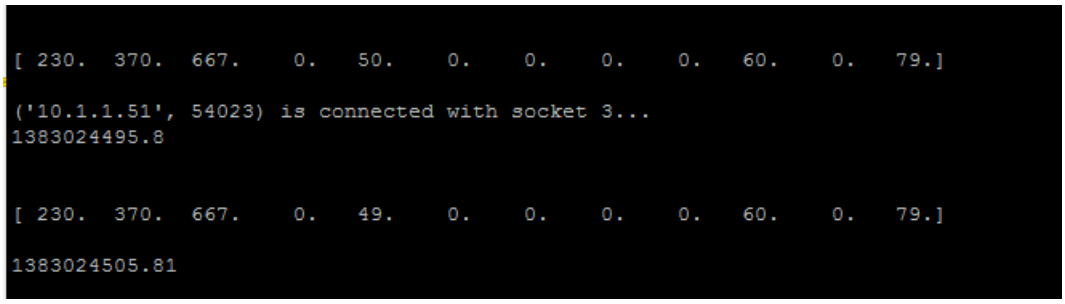


Abbildung 6.4: Hamburg Energie verbindet sich aus dem Tunnel mit dem Simulationsserver

In Abbildung 6.4 ist die Verbindung der Leitwarte mit dem Modbus-Server der Simulation zu sehen. Da dies hinter dem Tunnel geschieht, wird die interne IP-Adresse der Leitwarte genutzt.

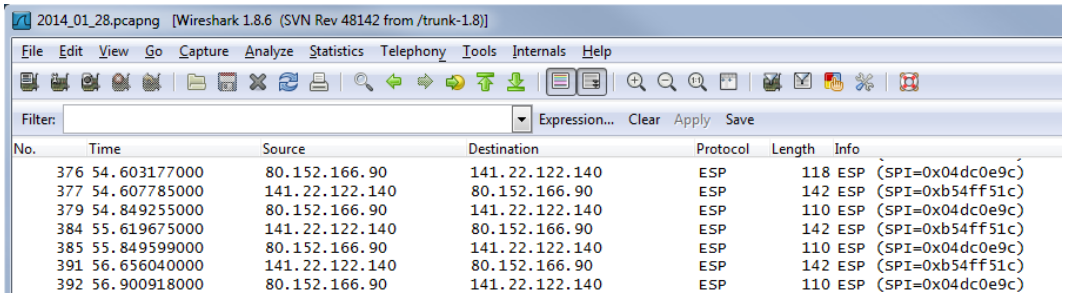


Abbildung 6.5: Die Tunnelverbindung von Hamburg Energie zum Simulationsmodell

Anders ist es in Abbildung 6.4, die eine mitgeschnittene Aufzeichnung der Netzwerkkommunikation mithilfe von Wireshark zeigt. Hier ist die Adresse von Hamburg Energie die öffentliche IP-Adresse 80.152.166.90, die sich mit der öffentlichen Adresse der Simulation verbindet. Außerhalb des Tunnels ist somit nicht zu erkennen, wer sich wirklich mit wem verbindet.

6.2 Aufgabenstellung Modbus

Der Modbus kann mit Hilfe des Netzwerkanalysetools Wireshark überprüft werden. Wireshark erlaubt das Mitschneiden und analysieren der

Kommunikationspakete, die über die Ethnernet-Netzwerkkarte gesendet und empfangen werden.

Wireshark steht unter der GPL Lizenz und ist somit ebenfalls eine freie Software.

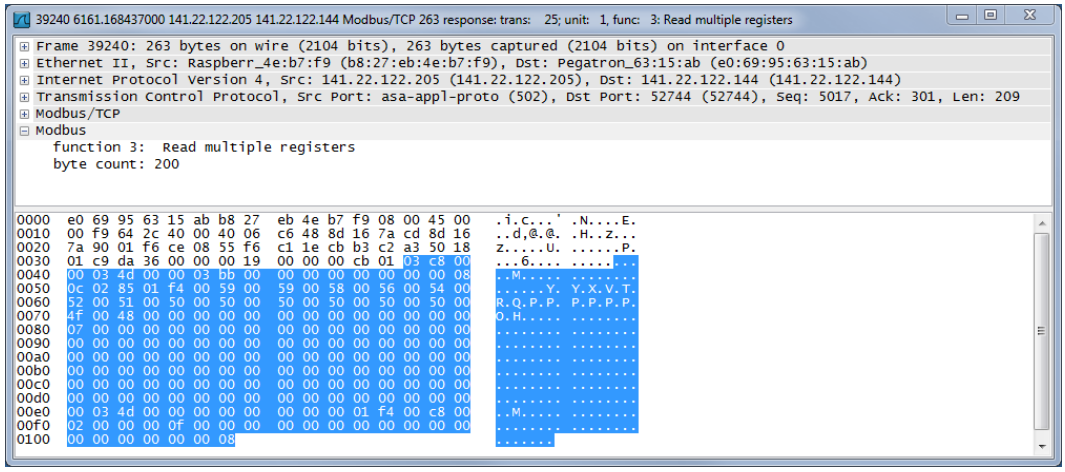


Abbildung 6.6: Einzelnes Datenpaket einer Modbus Kommunikation

In Abbildung 6.6 sieht man die Kommunikation zwischen dem Raspberry und einem PC, der mithilfe des Shareware-Tools ModbusMaster Werte aus dem Modbus liest.

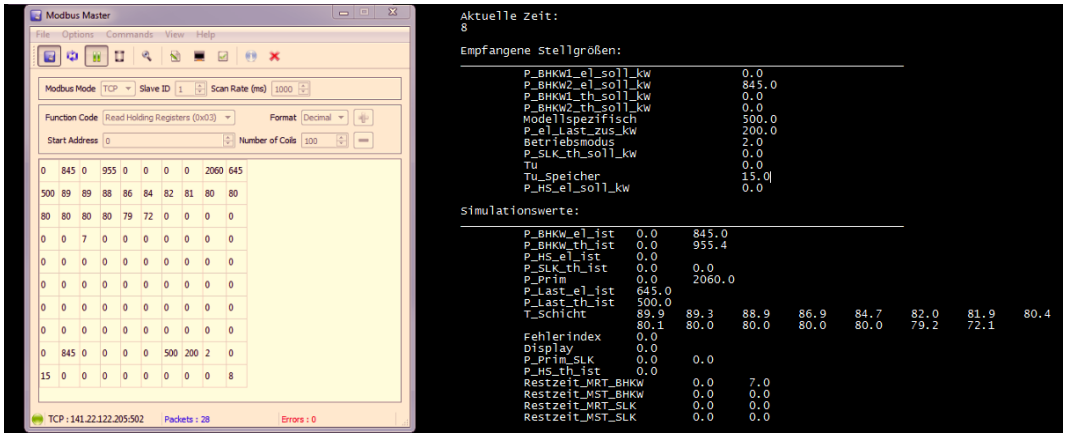


Abbildung 6.7: Anzeige der Modbuswerte auf dem Windows Rechner

Der Modbus wird vom Raspberry aufgebaut, der auch die Simulation laufen lässt. Dieser Modbus ist 100 Coils groß, im oberen Teil des Registers werden die Istwerte der Stellgrößen geschrieben, gefolgt von den Simulationsergebnissen. Im unteren Teil des Registers schreibt das Gateway die Stellgrößen hinein.

Bei der Übersetzung der Hexadezimalzahlen aus dem mitgeschnittenen Modbus Packet, zu sehen in Abbildung 6.8, erkennt man die Einteilung des Registers. Es

werden Funktion und Größe des Registers (blau umrandet) sowie der TCP-IP Header wie z.B IP-Adresse des Absenders (rot umrandet) Byteweise übermittelt, doch haben die Daten im Register eine Größe von 2 Byte. So wird die Simulationsgröße P_BHKW_el_ist_kW als $3 \cdot 2^8 + 77$ für 845 und P_Last_th_ist als $1 \cdot 2^8 + 244$ für 500 übertragen (grün umrandet). Deshalb liegt der zu sendende Wertebereich bei 0-65.535 pro Coil.

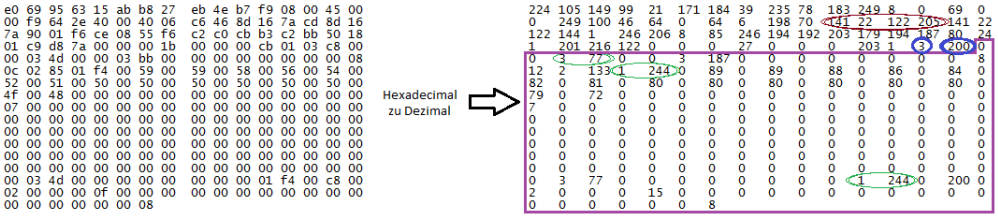


Abbildung 6.8: Mitschnitt des Modbus-Pakets in Dezimalwerten

6.3 Aufgabenstellung IEC 61850

Da der IEC-61850 Stack der RWTH mit REST-Befehlen arbeitet, kann die Abfrage der Werte unter 61850 direkt im Webbrowser erfolgen. Es kann die HTTP-Methode GET verwendet werden, da sich die URLs der abzufragenden Werte nicht ändern. Die Methode GET wird von allen HTTP-fähigen Clients unterstützt.

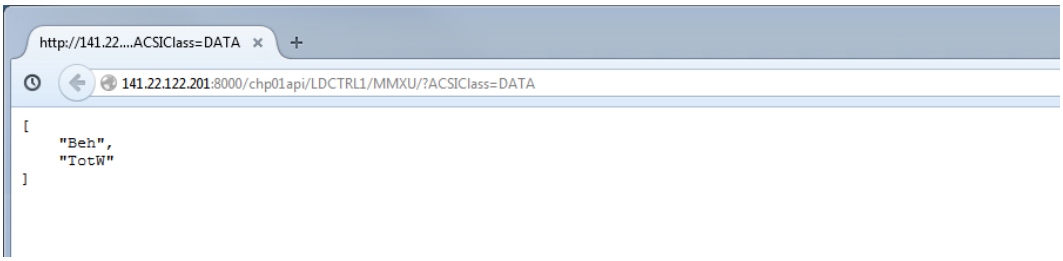


Abbildung 6.9: REST-Abfrage der Attribute des Knoten MMXU

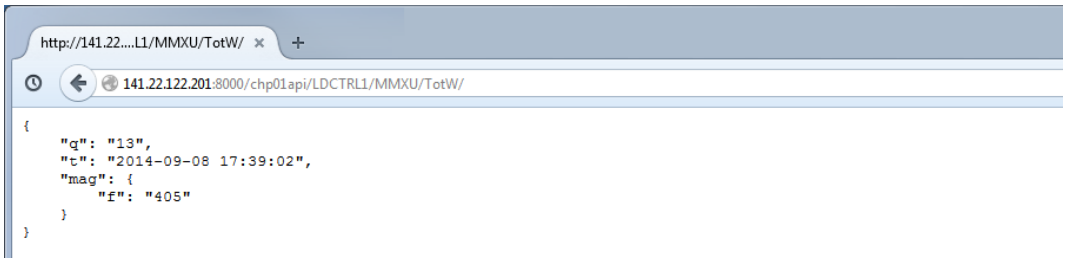


Abbildung 6.10: REST-Abfrage des Attributs TotW des Knoten MMXU

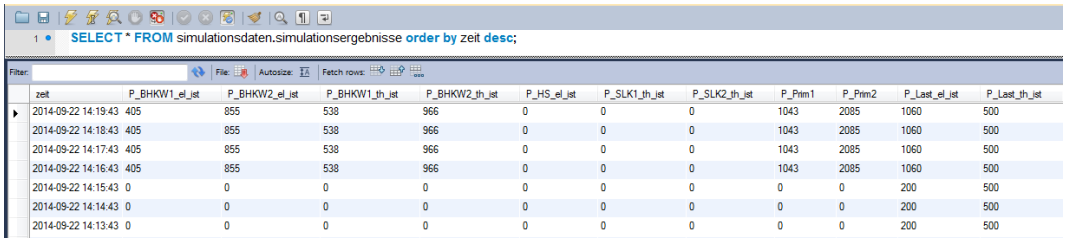
Der benutzte Stack der RWTH Aachen wurde leider nicht Quelloffen als Python-Dateien zur Verfügung gestellt, sondern als bytecode generierte pyc-Datei.

Somit ist es nicht möglich, die Python-Dateien urls und view zu ergänzen oder zu erweitern. Die SQL-Datei, welche das Modell des BHKWs abbilden soll, kann zwar erweitert werden, jedoch ist die URLs-Datei für das korrekte mappen einer Anfrage und die View-Datei zum generieren des Request nötig.

Deshalb können nur Werte verwendet werden, die von der RWTH zur Verfügung gestellt werden. Das Hinzufügen weiterer Knoten und deren Attribute sowie weiterer Geräte ist nicht möglich.

6.4 Aufgabenstellung Datenbank

Die MySQL Datenbank liefert die Stellgrößen und speichert die Simulationswerte. Für die Simulationswerte gibt es zwei Table, in denen die Werte geschrieben werden. Einmal den Table chp_test, der immer die aktuellen Werte für die Kommunikation über IEC 61850 vorhält, und den Table Simulationsergebnisse, der zur Archivierung dient.



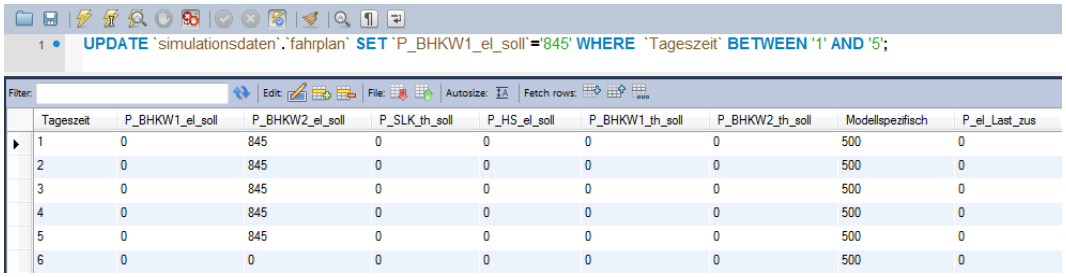
```
SELECT * FROM simulationsdaten.simulationsergebnisse order by zeit desc;
```

zeit	P_BHKW1_el_ist	P_BHKW2_el_ist	P_BHKW1_th_ist	P_BHKW2_th_ist	P_HS_el_ist	P_SLK1_th_ist	P_SLK2_th_ist	P_Prim1	P_Prim2	P_Last_el_ist	P_Last_th_ist
2014-09-22 14:19:43	405	855	538	966	0	0	0	1043	2085	1060	500
2014-09-22 14:18:43	405	855	538	966	0	0	0	1043	2085	1060	500
2014-09-22 14:17:43	405	855	538	966	0	0	0	1043	2085	1060	500
2014-09-22 14:16:43	405	855	538	966	0	0	0	1043	2085	1060	500
2014-09-22 14:15:43	0	0	0	0	0	0	0	0	0	200	500
2014-09-22 14:14:43	0	0	0	0	0	0	0	0	0	200	500
2014-09-22 14:13:43	0	0	0	0	0	0	0	0	0	200	500

Abbildung 6.11: Der Table Simulationsergebnisse in der MySQL Datenbank

Um einen Verlauf der Simulationsergebnisse anzuzeigen und diese auch im Vergleich der Stellgrößen zu sehen ist der Table Simulationsergebnisse hilfreich. Alle Stellgrößen aus Tabelle 3.1: Erwartete Eingänge der Simulation und Tabelle 3.2: Ausgegebene Größen der Simulation, mit Ausnahme der Speicherschichttemperaturen, werden in diesem Table gespeichert.

Zum Ändern der Stellgrößen kann der Table Fahrplan editiert werden. Dadurch können durch SQL-Statements direkte Änderungen in den angeforderten Leistungen oder Lasten vorgenommen werden. Zu sehen in Abbildung 6.12.



```
UPDATE `simulationsdaten`.`fahrplan` SET `P_BHKW1_el_soll`='845' WHERE `Tageszeit` BETWEEN '1' AND '5';
```

Tageszeit	P_BHKW1_el_soll	P_BHKW2_el_soll	P_SLK_th_soll	P_HS_el_soll	P_BHKW1_th_soll	P_BHKW2_th_soll	Modellspezifisch	P_el_Last_zus
1	0	845	0	0	0	0	500	0
2	0	845	0	0	0	0	500	0
3	0	845	0	0	0	0	500	0
4	0	845	0	0	0	0	500	0
5	0	845	0	0	0	0	500	0
6	0	0	0	0	0	0	500	0

Abbildung 6.12: Der Table Fahrplan, hier befinden sich die Stellgrößen der Simulation

6.5 Aufgabenstellung Leitwarte

Um den Zugriff einer Leitwarte nachzustellen wurde mittels Python und den Bibliotheken MySQLdb und cgitb ein Skript geschrieben, welches eine dynamische HTML-Seite aufbaut. Dazu werden die Simulationsergebnisse und die Stellgrößen aus der Datenbank zu einer CSV-Datei zusammengefügt und gespeichert. Mit Hilfe von Google Chart kann aus dieser CSV-Datei eine graphische Abbildung erstellt werden, die mit Python zu einer Webseite geformt wird.

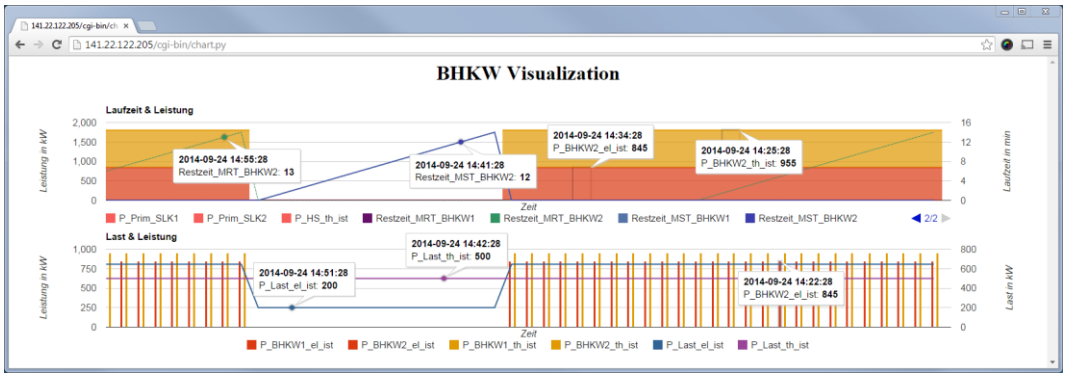


Abbildung 6.13: Graphische Anzeige der Simulation anhand der Datenbankwerte

Um diese generierten Graphen anzeigen zu lassen wurde die Website auf einen Server gestellt, der diese zur Verfügung stellt.

7 Zusammenfassung und Ausblick

Das in Matlab-Simulink gebaute Simulationsmodell konnte als C-Code auf ein open Source System portiert und dort kompiliert werden.

Durch Python wurde das schrittweise Aufrufen der generierten Simulation umgesetzt und ein Echtzeitverhalten geschaffen, ebenfalls unter Python wurde die praxisgerechte Kommunikation mittels Modbus-TCP verwirklicht. Somit konnte eine realitätsnahe Simulation eines Blockheizkraftwerks und eines Spitzenlastkessels, das in Kombination mit einem Wärmespeicher und einem Heizstab auf einem ein-Platinen-Computern (Raspberry Pi) als „Hardware in the Loop“ günstig und effektiv gebaut werden.

Diese Simulationsmodelle wurde durch eine verschlüsselte Verbindung an die Leitwarte eines Energieversorgers (Hamburg Energie) angeschlossen und konnte dort direkt in die Erzeugerliste aufgenommen werden.

Mit dem Ziel mehrere Simulationen an eine Leitwarte anzuschließen und durch eine intelligentere Kommunikation anzusteuern wurde ein zusätzliches Kommunikationsgateways entwickelt. Dafür wurde der Python Stack der RWTH Aachen eingesetzt, der die Kommunikation über IEC 61850 beherrscht. Mit diesem Stack wurde ein Schnittstellensystem gebaut, welches Simulationswerte aus dem Modbus-TCP-Protokoll auf IEC 61850 übersetzt und Stellgrößen aus einem Energiefahrplan für die Simulationsmodelle in das Modbus-TCP-Protokoll schreibt.

Durch das Übersetzen der nicht standarisierten Werteanordnungen des Modbuses zu IEC 61850, die die Werte erwartet und strukturiert ablegt, sollte es zu einer Transparenz und Qualität kommen, die es erlaubt, mehrere Anlagen verteilt, aber trotzdem geregelt anzusteuern und zu überwachen.

Da es aber leider nicht möglich war die erwarteten Werte der IEC 61850 selber vorzuschreiben ist das Hantieren von Stellgrößen und das Archivieren der Simulationswerte meist von der Datenbank übernommen worden. Veränderungen der Stellgrößen wurden direkt durch die Überarbeitung des Fahrplans in der Datenbank vorgenommen, und das Auswerten der Simulationsergebnisse erfolgte ebenfalls durch das Herunterladen der Werte aus der Datenbank.

Um eine breite, vielfältig Angelegte Massensimulation mehrerer Erzeuger zu steuern ist das Anlegen einer gut strukturierten Datenbank als Verwaltungssystem

nicht nur hilfreich, sondern auch zwingend notwendig.

Das betrifft natürlich auch die genormte Kommunikation, jedoch kann durch den Zugriff einer effizienten, konsistenten und dauerhaft speichernden Datenbank das Grundgerüst der Kommunikation liegen, wie es der Kommunikationsstack der RWTH Aachen und der FGH Mannheim realisiert hat.

Anhang

A Generierter C-Code

A.1 ert-main.c:

```
#include <stdio.h>
#include "Liegenschaft.h"
#include "Liegenschaft.c"
#include "rtwtypes.h"
/*
void rt_OneStep(void)
{
  static boolean_T OverrunFlag = 0;
  if (OverrunFlag) {
    rtmSetErrorStatus(Liegenschaft_M, "Overrun");
    return;
  }

  OverrunFlag = TRUE;
  Liegenschaft_step();
  OverrunFlag = FALSE;
}
*/
int_T main(int_T argc, const char *argv[])
{
  (void)(argc);
  (void)(argv);
  Liegenschaft_initialize();
  printf("Warning: The simulation will run forever. "
         "Generated ERT main won't simulate model step behavior. "
         "To change this behavior select the 'MAT-file logging' option.\n");
  fflush((NULL));
  while (rtmGetErrorStatus(Liegenschaft_M) == (NULL)) {
  }

  Liegenschaft_terminate();
  return 0;
}
```

A.2 Liegenschaft.c

```
#include "Liegenschaft.h"
#include "Liegenschaft_private.h"

B_Liegenschaft_T Liegenschaft_B;
DW_Liegenschaft_T Liegenschaft_DW;
ExtU_Liegenschaft_T Liegenschaft_U;
ExtY_Liegenschaft_T Liegenschaft_Y;
RT_MODEL_Liegenschaft_T Liegenschaft_M_;
RT_MODEL_Liegenschaft_T *const Liegenschaft_M = &Liegenschaft_M_;
real_T Ausgabe[39];
void Liege_IfActionSubsystem_Disable(B_IfActionSubsystem_Liegensch_T *localB)
{
  localB->In1[0] = 0.0;
  localB->In1[1] = 0.0;
}

void Liegenschaft_IfActionSubsystem(const real_T rtu_0[2],
  B_IfActionSubsystem_Liegensch_T *localB)
{
  localB->In1[0] = rtu_0[0];
  localB->In1[1] = rtu_0[1];
}

void Liegenschaft_EnabledSubsystem(boolean_T rtu_0, boolean_T rtu_In1,
  B_EnabledSubsystem_Liegenscha_T *localB)
{
  if (rtu_0) {
    localB->Product = (real_T)rtu_In1 * 2500.0;
  }
}

//void Liegenschaft_step(void)
```

```
real_T * Liegenschaft_step(real_T arg_P_BHKW_el_soll[2], real_T
  arg_P_BHKW_th_soll[2], real_T arg_Modellspezifisch, real_T arg_P_el_Last_zus,
  real_T arg_Betriebsmodus, real_T arg_P_SLK_th_soll, real_T arg_Tu, real_T
  arg_Tu_Speicher, real_T arg_P_HS_el_soll)
{
  real_T rtb_Add3_nl[2];
  real_T rtb_Add3_aq[2];
  […]
  boolean_T rtb_LogicalOperator1_k_idx;
  boolean_T rtb_LogicalOperator1_k_idx_0;
  //Paramter in die Eingaenge schreiben
  Liegenschaft_U.P_BHKW_el_soll[0]=arg_P_BHKW_el_soll[0];
  Liegenschaft_U.P_BHKW_el_soll[1]=arg_P_BHKW_el_soll[1];
  Liegenschaft_U.P_BHKW_th_soll[0]=arg_P_BHKW_th_soll[0];
  Liegenschaft_U.P_BHKW_th_soll[1]=arg_P_BHKW_th_soll[1];
  Liegenschaft_U.Modellspezifisch=arg_Modellspezifisch;
  Liegenschaft_U.P_el_Last_zus=arg_P_el_Last_zus;
  Liegenschaft_U.Betriebsmodus=arg_Betriebsmodus;
  Liegenschaft_U.P_SLK_th_soll=arg_P_SLK_th_soll;
  Liegenschaft_U.Tu=arg_Tu;
  Liegenschaft_U.Tu_Speicher=arg_Tu_Speicher;
  Liegenschaft_U.P_HS_el_soll=arg_P_HS_el_soll;

  rtb_Add = 1.0 + Liegenschaft_U.Betriebsmodus;
  memcpy(&rtb_Memory[0], &Liegenschaft_DW.Memory_PreviousInput[0], 15U * sizeof
         (real_T));
  […]
  Liegenschaft_DW.Initialisierung_PreviousInput = 0.0;
  int32_T w;
  Ausgabe[0]=Liegenschaft_Y.Arg_P_BHKW_el_ist[0];
  Ausgabe[1]=Liegenschaft_Y.Arg_P_BHKW_el_ist[1];
  Ausgabe[2]=Liegenschaft_Y.Arg_P_BHKW_th_ist[0];
  Ausgabe[3]=Liegenschaft_Y.Arg_P_BHKW_th_ist[1];
  Ausgabe[4]=Liegenschaft_Y.Arg_P_HS_el_ist;
  Ausgabe[5]=Liegenschaft_Y.Arg_P_SLK_th_ist[0];
  Ausgabe[6]=Liegenschaft_Y.Arg_P_SLK_th_ist[1];
  Ausgabe[7]=Liegenschaft_Y.Arg_P_Prim[0];
  Ausgabe[8]=Liegenschaft_Y.Arg_P_Prim[1];
  Ausgabe[9]=Liegenschaft_Y.Arg_P_Last_el_ist;
  Ausgabe[10]=Liegenschaft_Y.Arg_P_Last_th_ist;
  for(w = 1; w < 16; w++){
    Ausgabe[10+w]=Liegenschaft_Y.Arg_T_Schicht[w];
  }
  Ausgabe[26]=Liegenschaft_Y.Arg_Fehlerindex;
  Ausgabe[27]=Liegenschaft_Y.Arg_Display;
  Ausgabe[28]=Liegenschaft_Y.Arg_P_Prim_SLK[0];
  Ausgabe[29]=Liegenschaft_Y.Arg_P_Prim_SLK[1];
  Ausgabe[30]=Liegenschaft_Y.Arg_P_HS_th_ist;
  Ausgabe[31]=Liegenschaft_Y.Arg_Restzeit_MRT_BHKW[0];
  Ausgabe[32]=Liegenschaft_Y.Arg_Restzeit_MRT_BHKW[1];
  Ausgabe[33]=Liegenschaft_Y.Arg_Restzeit_MST_BHKW[0];
  Ausgabe[34]=Liegenschaft_Y.Arg_Restzeit_MST_BHKW[1];
  Ausgabe[35]=Liegenschaft_Y.Arg_Restzeit_MRT_SLK[0];
  Ausgabe[36]=Liegenschaft_Y.Arg_Restzeit_MRT_SLK[1];
  Ausgabe[37]=Liegenschaft_Y.Arg_Restzeit_MST_SLK[0];
  Ausgabe[38]=Liegenschaft_Y.Arg_Restzeit_MST_SLK[1];

  return Ausgabe;
}

void Liegenschaft_initialize(void)
[…]
```

A.3 Liegenschaft.h

```
[…]
extern ExtY_Liegenschaft_T Liegenschaft_Y;
extern void Liegenschaft_initialize(void);
//extern void Liegenschaft_step(void);
extern real_T * Liegenschaft_step(real_T arg_P_BHKW_el_soll[2], real_T
  arg_P_BHKW_th_soll[2], real_T arg_Modellspezifisch, real_T arg_P_el_Last_zus,
  real_T arg_Betriebsmodus, real_T arg_P_SLK_th_soll, real_T arg_Tu, real_T
  arg_Tu_Speicher, real_T arg_P_HS_el_soll);
extern void Liegenschaft_terminate(void);
extern RT_MODEL_Liegenschaft_T *const Liegenschaft_M;
#endif
```

B Python Skript Simulation

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-

#-------------------------------------------------------------------------------#
# 202 Mdl-Simulation 202
# Eine unter Simulink gebaute BHKW-Simulation wird als C-kompilierte,
# dynamische Programmbibliothek mit diesem Python-Skript gestartet.
# Das Skript bekommt aktuelle Stellgrößen unter Modbus vorgegeben.
# Die Simulationsergebnisse werden dann als Rückgabewerte in diesem Modbus
# bereitgestellt.
#-------------------------------------------------------------------------------#

import time                                 # Takt
import sys                                  # Exit

import ctypes                               # Bibliothek benutzen
from ctypes import CDLL
from numpy.ctypeslib import ndpointer       # Rückgabevektor

import modbus_tk                            # Modbus Modul
import modbus_tk.defines as mdef
import modbus_tk.modbus_tcp as modbus_tcp

#-------------------------------------------------------------------------------#
# Variablen initialisieren
#-------------------------------------------------------------------------------#
global connection
connection = 0;

zeit = 1;
simwerte = 100*[0.0]
simwerte_int = 100*[0]

P_BHKW_el_soll = [0.0, 0.0]
P_SLK_th_soll = 0.0
P_HS_el_soll = 0.0
P_BHKW_th_soll = [0.0, 0.0]
Modellspezifisch = 500.0
P_el_Last_zus = 200.0
Betriebsmodus = 2.0
Tu = 0.0
Tu_Speicher = 15.0
Tageszeit = 1

#-------------------------------------------------------------------------------#
# Ausgabe des Fahrplans auf dem Monitor
#-------------------------------------------------------------------------------#
def PrintOutStellgroeszen(P_BHKW_el_soll, P_BHKW_th_soll, Modellspezifisch,
P_el_Last_zus, Betriebsmodus, P_SLK_th_soll, Tu, Tu_Speicher, P_HS_el_soll):
      print "\nEmpfangene Stellgrößen:"
      print "_____________________________________________________"
      print "\t P_BHKW1_el_soll_kW\t\t%2.1f"   %(P_BHKW_el_soll[0])
      print "\t P_BHKW2_el_soll_kW\t\t%2.1f"   %(P_BHKW_el_soll[1])
      print "\t P_BHKW1_th_soll_kW\t\t%2.1f"   %(P_BHKW_th_soll[0])
      print "\t P_BHKW2_th_soll_kW\t\t%2.1f"   %(P_BHKW_th_soll[1])
      print "\t Modellspezifisch\t\t%2.1f"     %(Modellspezifisch)
      print "\t P_el_Last_zus_kW\t\t%2.1f"     %(P_el_Last_zus)
      print "\t Betriebsmodus\t\t\t%2.1f"      %(Betriebsmodus)
      print "\t P_SLK_th_soll_kW\t\t%2.1f"     %(P_SLK_th_soll)
      print "\t Tu\t\t\t%2.1f"                 %(Tu)
      print "\t Tu_Speicher\t\t\t%2.1f"        %(Tu_Speicher)
      print "\t P_HS_el_soll_kW\t\t%2.1f"      %(P_HS_el_soll)

#-------------------------------------------------------------------------------#
# Ausgabe der Rueckgabewert auf dem Monitor
#-------------------------------------------------------------------------------#
def PrintOutWerte(Werte):
      print "\nSimulationswerte:"
      print "_____________________________________________________"
      print "\t P_BHKW_el_ist\t %2.1f\t %2.1f\t"  % (Werte[0], Werte[1])
      print "\t P_BHKW_th_ist\t %2.1f\t %2.1f\t"  % (Werte[2], Werte[3])
      print "\t P_HS_el_ist\t %2.1f\t"  % (Werte[4])
      print "\t P_SLK_th_ist\t %2.1f\t %2.1f\t"  % (Werte[5], Werte[6])
      print "\t P_Prim\t\t %2.1f\t %2.1f\t"  % (Werte[7], Werte[8])
```

```
      print "\t P_Last_el_ist\t %2.1f\t"  % (Werte[9])
      print "\t P_Last_th_ist\t %2.1f"  % (Werte[10])
      print "\t T_Schicht\t 2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f"
% (Werte[11], Werte[12], Werte[13], Werte[14], Werte[15], Werte[16], Werte[17],
Werte[18])
      print "\t \t\t %2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t" %
(Werte[19], Werte[20], Werte[21], Werte[22], Werte[23], Werte[24], Werte[25])
      print "\t Fehlerindex\t %2.1f"  % (Werte[26])
      print "\t Display\t %2.1f"  % (Werte[27])
      print "\t P_Prim_SLK\t %2.1f\t %2.1f\t"  % (Werte[28], Werte[29])
      print "\t P_HS_th_ist\t %2.1f\t" % (Werte[30])
      print "\t Restzeit_MRT_BHKW\t %2.1f\t %2.1f"  % (Werte[31], Werte[32])
      print "\t Restzeit_MST_BHKW\t %2.1f\t %2.1f"  % (Werte[33], Werte[34])
      print "\t Restzeit_MRT_SLK\t %2.1f\t %2.1f"  % (Werte[35], Werte[36])
      print "\t Restzeit_MST_SLK\t %2.1f\t %2.1f"  % (Werte[37], Werte[38])

#-------------------------------------------------------------------------------#
# TCP_Modbus Server wird gestartet
#-------------------------------------------------------------------------------#
logger = modbus_tk.utils.create_logger(name="console",
record_format="%(message)s")
server = modbus_tcp.TcpServer(address='141.22.122.202')
slave1 = server.add_slave(1)                          # erstellt slave mit id 1
slave1.add_block("BHKW", mdef.HOLDING_REGISTERS, 0, 100)   # address 0, length 100
slave1.set_values("BHKW", 0, 0)                        # Registerwerte auf Null
server.start()
print "Modbus-Server gestartet"

#-------------------------------------------------------------------------------#
# Bibliothek wird eingebunden
#-------------------------------------------------------------------------------#
LIBRARY_PATH = "/home/pi/Liegenschaft_ert_rtw/libLiegenschaft.so"
lib = ctypes.CDLL(LIBRARY_PATH)
lib.Liegenschaft_step.argtypes = [ctypes.POINTER(ctypes.c_double),
ctypes.POINTER(ctypes.c_double), ctypes.c_double, ctypes.c_double,
ctypes.c_double, ctypes.c_double, ctypes.c_double, ctypes.c_double,
ctypes.c_double]

lib.Liegenschaft_step.restype = ndpointer(dtype=ctypes.c_double, shape=(39,))
print "Lib geladen"
# Modell initialisieren
lib.Liegenschaft_initialize()

#-------------------------------------------------------------------------------#
# Loop
#-------------------------------------------------------------------------------#
while (1):
   #while (connection == 1)
   #   print "connection interrupts\n"
   #   time.sleep(6.)
   try:
      # get time
      now = time.time()

      #-------------------------------------------------------------------------#
      # Lesen der Stellgroeszen aus dem Modbus
      #-------------------------------------------------------------------------#
      P_BHKW_el_soll[0] = slave1.get_values("BHKW", 80)[0]
      P_BHKW_el_soll[1] = slave1.get_values("BHKW", 81)[0]
      P_SLK_th_soll = slave1.get_values("BHKW", 82)[0]
      P_HS_el_soll = slave1.get_values("BHKW", 83)[0]
      P_BHKW_th_soll[0] = slave1.get_values("BHKW", 84)[0]
      P_BHKW_th_soll[1] = slave1.get_values("BHKW", 85)[0]
      Modellspezifisch = slave1.get_values("BHKW", 86)[0]
      P_el_Last_zus = slave1.get_values("BHKW", 87)[0]
      Betriebsmodus = slave1.get_values("BHKW", 88)[0]
      Tu = slave1.get_values("BHKW", 89)[0]
      Tu_Speicher = slave1.get_values("BHKW", 90)[0]

      Tageszeit = slave1.get_values("BHKW", 99)[0]

      #-------------------------------------------------------------------------#
      # Stellgroeszen in cytpes
      #-------------------------------------------------------------------------#

      Ct_Modellspezifisch = ctypes.c_double(Modellspezifisch)
      Ct_P_el_Last_zus = ctypes.c_double(P_el_Last_zus)
      Ct_Betriebsmodus = ctypes.c_double(Betriebsmodus)
```

```
      Ct_P_SLK_th_soll = ctypes.c_double(P_SLK_th_soll)
      Ct_Tu = ctypes.c_double(Tu)
      Ct_Tu_Speicher = ctypes.c_double(Tu_Speicher)
      Ct_P_HS_el_soll = ctypes.c_double(P_HS_el_soll)

      # P_BHKW_el und P_BHKW_th sind Pointer
      Ct_P_BHKW_el_soll = (ctypes.c_double * len(P_BHKW_el_soll))()
      for index, value in enumerate(P_BHKW_el_soll):
         Ct_P_BHKW_el_soll[index] = value
      Ct_P_BHKW_th_soll = (ctypes.c_double * len(P_BHKW_th_soll))()
      for index, value in enumerate(P_BHKW_th_soll):
         Ct_P_BHKW_th_soll[index] = value

      #---------------------------------------------------------------------#
      # Step-Funktion wird ausgeführt, Rwerte werden in simwerte gespeichert
      #---------------------------------------------------------------------#
      simwerte = lib.Liegenschaft_step(Ct_P_BHKW_el_soll, Ct_P_BHKW_th_soll,
Ct_Modellspezifisch, Ct_P_el_Last_zus, Ct_Betriebsmodus, Ct_P_SLK_th_soll, Ct_Tu,
Ct_Tu_Speicher, Ct_P_HS_el_soll)
      # Werte als int für den Modbus
      for j in range(0, 38):
        try:
         simwerte_int[j] = int(abs(simwerte[j]))
        except ValueError:
         simwerte_int[j] = 0

      #---------------------------------------------------------------------#
      # Modbus-Register mit Rueckgabewerten beschreiben
      #---------------------------------------------------------------------#
      for k in range (0, 38):
         slave1.set_values("BHKW",  k, simwerte_int[k])

      #---------------------------------------------------------------------#
      # Monitorausgabe
      #---------------------------------------------------------------------#
      # Zeitangabe
      print "\nAktuelle Zeit:"
      print Tageszeit
      # Monitorausgabe empfangene Stellgroeszen
      PrintOutStellgroeszen(P_BHKW_el_soll, P_BHKW_th_soll, Modellspezifisch,
P_el_Last_zus, Betriebsmodus, P_SLK_th_soll, Tu, Tu_Speicher, P_HS_el_soll)
      # Monitorausgabe Simulationswerte
      PrintOutWerte(simwerte);

      #---------------------------------------------------------------------#
      # Berechnung der Zeit und warten auf den naechsten Schritt
      #---------------------------------------------------------------------#
      while ((60. - (time.time() - now)) > 0):
         time.sleep(0.001) # 1ms warten, bis volle 60 sec. vergangen sind

    #-------------------------------------------------------------------------#
    # EXIT
    #-------------------------------------------------------------------------#
   except(KeyboardInterrupt, SystemExit):
      server.stop()
      sys.exit("KeyboardInterrupt! quit!")
# raise SystemExit(0)
```

C Python Skript Gateway

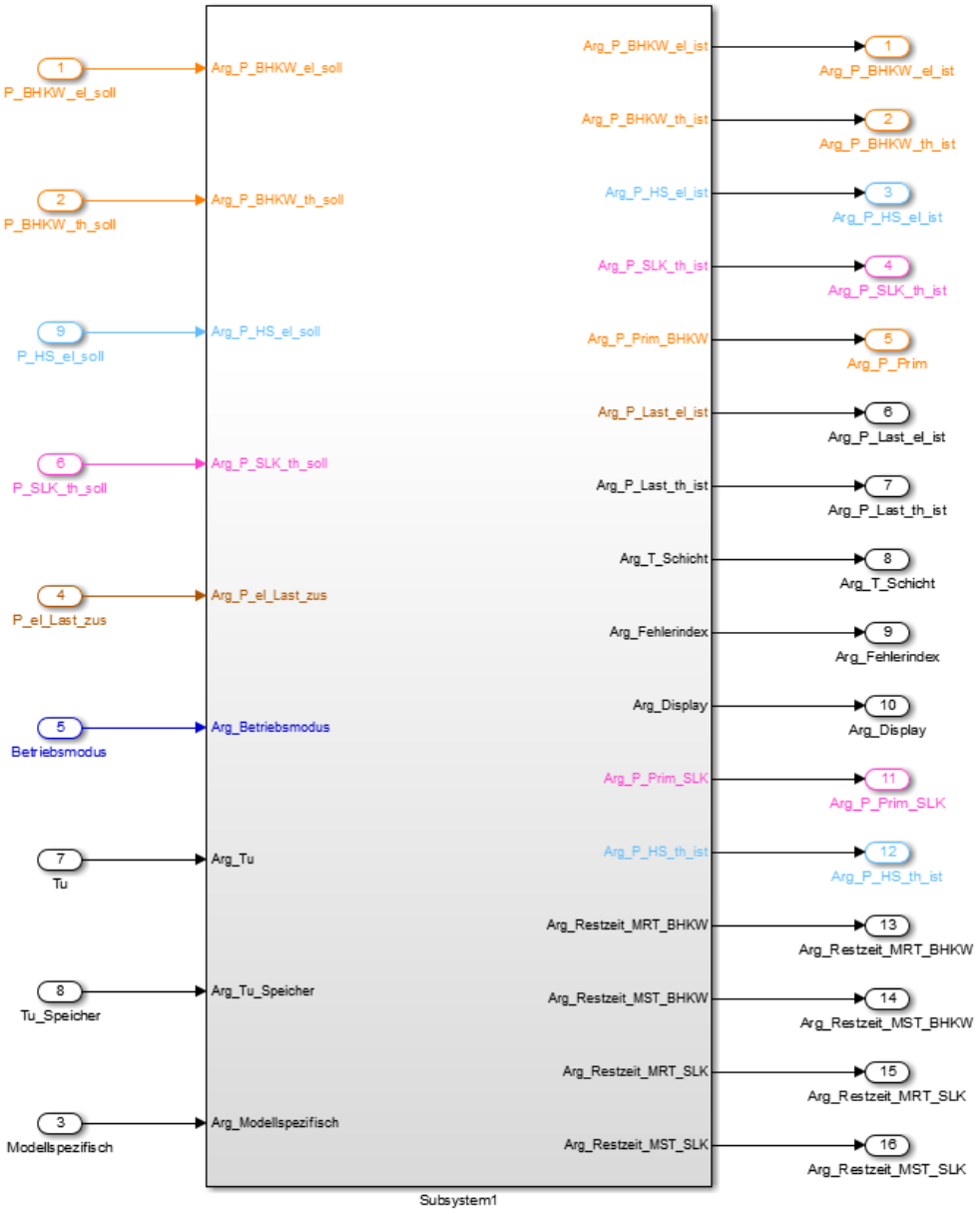
```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-

#-------------------------------------------------------------------------------#
# 201 GATEWAY 201
# Das Gateway dienst als Schnittstelle zwischen Leitwarte und Simulation
# Es werden Stellgrößen aus einer Datenbank gelesen und
# via Modbus dem Modell zu Verfügung gestellt.
# Die Rückgabewerte der Simulation werden ebenfalls via Modbus gelesen
# und per IEC 61850 der Leitwarte bereit gestellt
#-------------------------------------------------------------------------------#

import time                             # pulse
import sys                              # exit interrupt

import modbus_tk                        # Modbus Modul
import modbus_tk.defines as cst
import modbus_tk.modbus_tcp as modbus_tcp

import MySQLdb                          # mySQL

#-------------------------------------------------------------------------------#
# Variablen initialisieren
#-------------------------------------------------------------------------------#
global connection
connection = 0;

zeit = 1;
Werte = 100*[0]
filename = "%s_Fahrplan_2014.csv" %str(time.localtime().tm_yday)

P_BHKW_el_soll = [0.0, 0.0]
P_SLK_th_soll = 0.0
P_HS_el_soll = 0.0
P_BHKW_th_soll = [0.0, 0.0]
Modellspezifisch = 0.0
P_el_Last_zus = 0.0
Betriebsmodus = 2.0
Tu = 10.0
Tu_Speicher = 10.0

#-------------------------------------------------------------------------------#
# Ausgabe der  Stellgr□□ auf dem Monitor
#-------------------------------------------------------------------------------#
def PrintOutStellgroeszen(P_BHKW_el_soll, P_BHKW_th_soll, Modellspezifisch,
P_el_Last_zus,
                          Betriebsmodus, P_SLK_th_soll, Tu, Tu_Speicher,
P_HS_el_soll):
        print "\nVorgegebene Stellgr□□:"
        print "                                                                "
        print "\t P_BHKW1_el_soll_kW\t\t%2.1f"  %(P_BHKW_el_soll[0])
        print "\t P_BHKW2_el_soll_kW\t\t%2.1f"  %(P_BHKW_el_soll[1])
        print "\t P_BHKW1_th_soll_kW\t\t%2.1f"  %(P_BHKW_th_soll[0])
        print "\t P_BHKW2_th_soll_kW\t\t%2.1f"  %(P_BHKW_th_soll[1])
        print "\t Modellspezifisch\t\t%2.1f"    %(Modellspezifisch)
        print "\t P_el_Last_zus_kW\t\t%2.1f"    %(P_el_Last_zus)
        print "\t Betriebsmodus\t\t\t%2.1f"     %(Betriebsmodus)
        print "\t P_SLK_th_soll_kW\t\t%2.1f"    %(P_SLK_th_soll)
        print "\t Tu\t\t\t%2.1f"                %(Tu)
        print "\t Tu_Speicher\t\t\t%2.1f"       %(Tu_Speicher)
        print "\t P_HS_el_soll_kW\t\t%2.1f"     %(P_HS_el_soll)

#-------------------------------------------------------------------------------#
# Ausgabe der empfangene Simulationswerte auf dem Monitor
#-------------------------------------------------------------------------------#
def PrintOutWerte(Werte):
        print "\nEmpfangene Simulationswerte:"
        print "                                                                "
        print "\t P_BHKW_el_ist\t %2.1f\t %2.1f\t"  % (Werte[0], Werte[1])
        print "\t P_BHKW_th_ist\t %2.1f\t %2.1f\t"  % (Werte[2], Werte[3])
        print "\t P_HS_el_ist\t %2.1f\t"  % (Werte[4])
        print "\t P_SLK_th_ist\t %2.1f\t %2.1f\t"  % (Werte[5], Werte[6])
        print "\t P_Prim\t\t %2.1f\t %2.1f\t"  % (Werte[7], Werte[8])
        print "\t P_Last_el_ist\t %2.1f\t"  % (Werte[9])
```

```
        print "\t P_Last_th_ist\t %2.1f"  % (Werte[10])
        print "\t T_Schicht\t
%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f"  % (Werte[11], Werte[12],
Werte[13], Werte[14], Werte[15], Werte[16], Werte[17], Werte[18])
        print "\t \t\t %2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t" %
(Werte[19], Werte[20], Werte[21], Werte[22], Werte[23], Werte[24], Werte[25])
        print "\t Fehlerindex\t %2.1f"  % (Werte[26])
        print "\t Display\t %2.1f"  % (Werte[27])
        print "\t P_Prim_SLK\t %2.1f\t %2.1f\t"  % (Werte[28], Werte[29])
        print "\t P_HS_th_ist\t %2.1f\t" % (Werte[30])
        print "\t Restzeit_MRT_BHKW\t %2.1f\t %2.1f"  % (Werte[31], Werte[32])
        print "\t Restzeit_MST_BHKW\t %2.1f\t %2.1f"  % (Werte[33], Werte[34])
        print "\t Restzeit_MRT_SLK\t %2.1f\t %2.1f"  % (Werte[35], Werte[36])
        print "\t Restzeit_MST_SLK\t %2.1f\t %2.1f"  % (Werte[37], Werte[38])

#-------------------------------------------------------------------------------#
# Werte aus der Datenbank lesen
#-------------------------------------------------------------------------------#
def FahrplanDB(zeit):
  try:
    db = MySQLdb.connect("141.22.122.14","babu","stern","simulationsdaten")
    cursor = db.cursor()
    cursor.execute("SELECT * from Fahrplan where Tageszeit = %s", zeit)
    Fahrplan = cursor.fetchall()
    # speichern der Stellgr□□ in Fahrplan[]
    PrimKey, P_BHKW1_el_soll, P_BHKW2_el_soll, P_SLK_th_soll, P_HS_el_soll,
P_BHKW1_th_soll, P_BHKW2_th_soll, Modellspezifisch, P_el_Last_zus, Betriebsmodus,
Tu, Tu_Speicher = Fahrplan[0]

    cursor.close()
    db.commit()
  except db.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
    connection = 1;
  finally:
    if db:
      db.close()
      return P_BHKW1_el_soll, P_BHKW2_el_soll, P_SLK_th_soll, P_HS_el_soll,
P_BHKW1_th_soll, P_BHKW2_th_soll, Modellspezifisch, P_el_Last_zus, Betriebsmodus,
Tu, Tu_Speicher

#-------------------------------------------------------------------------------#
# Werte in die Datenbank schreiben
#-------------------------------------------------------------------------------#
def ErgebnisseDB(Werte):
  ergeb=24*[0] #ergeb sind ModBusWerte ohne die Schichttemperaturen
  ergeb[0:11]=Werte[0:11]
  ergeb[12:23]=Werte[27:38]
  try:
    # Open database connection
    db = MySQLdb.connect("141.22.122.14","babu","stern","simulationsdaten" )
    # prepare a cursor object using cursor() method
    cursor = db.cursor()
    # Prepare SQL query
    cursor.execute("INSERT INTO simulationsergebnisse
(zeit,P_BHKW1_el_ist,P_BHKW2_el_ist,P_BHKW1_th_ist,P_BHKW2_th_ist,P_HS_el_ist,P_SL
K1_th_ist,P_SLK2_th_ist,P_Prim1,P_Prim2,P_Last_el_ist,P_Last_th_ist,Fehlerindex,Di
splay,P Prim SLK1,P Prim SLK2,P HS th ist,Restzeit MRT BHKW1,Restzeit MRT BHKW2,Re
stzeit MST BHKW1,Restzeit MST BHKW2,Restzeit MRT SLK1,Restzeit MRT SLK2,Restzeit M
ST_SLK1,Restzeit_MST_SLK2) VALUES (NOW(), %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)", ergeb)
    cursor.close()
    db.commit()
  except db.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
    connection = 1;
  finally:
    # disconnect from server
    if db:
      db.close()

#-------------------------------------------------------------------------------#
# 61850
#-------------------------------------------------------------------------------#
def Ergebnisse61850(Werte):
  ergeb=24*[0]
  ergeb[0:11]=Werte[0:11]
  ergeb[12:23]=Werte[27:38]
```

```
  try:
    # Open database connection
    db = MySQLdb.connect("141.22.122.14","babu","stern","chp_test" )
    # prepare a cursor object using cursor() method
    cursor = db.cursor()
    ## SQL query LDCTRL1/DRCT.MaxWLim
      ## cursor.execute("UPDATE chp01 data attribut SET value = %s WHERE id =
5",200)

    # SQL query LDCTRL1/MMXU.TotW
    # Total active power (total P)
    cursor.execute("UPDATE chp01_data_attribut SET value = %s WHERE
ref='LDCTRL1/MMXU.TotW.mag.f'",ergeb[0])
    cursor.execute("UPDATE chp01_data_attribut SET value = NOW() WHERE
ref='LDCTRL1/MMXU.TotW.t'")

    # SQL query LDCTRL1/DCTS.ThrmOut
    # Instantaneous thermal energy output from storage

    cursor.execute("UPDATE chp01_data_attribut SET value = %s WHERE
ref='LDCTRL1/DCTS.ThrmOut.mag.f'",ergeb[2])
    cursor.execute("UPDATE chp01_data_attribut SET value = NOW() WHERE
ref='LDCTRL1/DCTS.ThrmOut.t'")

    # SQL query LDCTRL1/DRCT.StrDITms
    # Nominal time delay before starting or restarting

    cursor.execute("UPDATE chp01_data_attribut SET value = %s WHERE
ref='LDCTRL1/DRCT.StrDITms.mag.f'",ergeb[16])
    cursor.execute("UPDATE chp01_data_attribut SET value = NOW() WHERE
ref='LDCTRL1/DRCT.StrDITms.t'")

    # SQL query LDCTRL1/DRCT.
    # Nominal time delay before stopping

    cursor.execute("UPDATE chp01_data_attribut SET value = %s WHERE
ref='LDCTRL1/DRCT.StopDlTms.mag.f'",ergeb[18])
    cursor.execute("UPDATE chp01_data_attribut SET value = NOW() WHERE
ref='LDCTRL1/DRCT.StopDlTms.t'")

    cursor.close()
    db.commit()
  except db.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
    connection = 1;
  finally:
    # disconnect from server
    if db:
      db.close()

#--------------------------------------------------------------------------------#
# Werte in die Log-Datei schreiben
#--------------------------------------------------------------------------------#
def ErgebnisseCSV(Werte):
    f = open(filename,'a')

f.write("\n%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%2.1f\t%
2.1f\t" %(P_BHKW_el_soll[0], P_BHKW_el_soll[1], P_SLK_th_soll, P_HS_el_soll,
P_BHKW_th_soll[0], P_BHKW_th_soll[1], Modellspezifisch, P_el_Last_zus,
Betriebsmodus, Tu, Tu_Speicher))
    for i in range(len(Werte)):
        f.write("%2.1f\t" %(Werte[i]))
    f.close()

#--------------------------------------------------------------------------------#
# TCP-Modbus Master wird gestartet
#--------------------------------------------------------------------------------#
print "Modbus Client wird gesucht..."
try:
    time.sleep(6.)          # Connect to the slave
    master = modbus_tcp.TcpMaster(host="141.22.122.202")
except modbus_tk.modbus.ModbusError, e:
    print "Modbus error ", e.get_exception_code()
    connection = 1
except Exception, e2:
    print "Error ", str(e2)
    #connection = 1
```

```
while (1):
    while (connection == 1):
        print "connection interrupts\n"
        time.sleep(6.)
    try:
        now = time.time()    # Zeitabfrage
        if zeit==1440:       # Neuer Tag, neue Logdatei
            zeit=1
            filename = "%s_Fahrplan_2014.csv" %str(time.localtime().tm_yday)

        # Fahrplan aus der Datenbank lesen
        P_BHKW_el_soll[0], P_BHKW_el_soll[1], P_SLK_th_soll, P_HS_el_soll,
P_BHKW_th_soll[0], P_BHKW_th_soll[1], Modellspezifisch, P_el_Last_zus,
Betriebsmodus, Tu, Tu_Speicher = FahrplanDB(zeit)

        # Modbus auslesen
        ModBusWerte = master.execute(1, cst.READ_HOLDING_REGISTERS, 0, 100)

        # Stellgroeszen in den Modbus schreiben
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 80,
output_value=int(P_BHKW_el_soll[0]))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 81,
output_value=int(P_BHKW_el_soll[1]))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 82,
output_value=int(P_SLK_th_soll))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 83,
output_value=int(P_HS_el_soll))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 84,
output_value=int(P_BHKW_th_soll[0]))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 85,
output_value=int(P_BHKW_th_soll[1]))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 86,
output_value=int(Modellspezifisch))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 87,
output_value=int(P_el_Last_zus))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 88,
output_value=int(Betriebsmodus))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 89, output_value=int(Tu))
        master.execute(1, cst.WRITE_SINGLE_REGISTER, 90,
output_value=int(Tu_Speicher))

        master.execute(1, cst.WRITE_SINGLE_REGISTER, 99, output_value=int(zeit))

        # Zeitangabe
        print "\nAktuelle Zeit:"
        print zeit
        # Monitorausgabe empfangene Simulationswerte
        PrintOutWerte(ModBusWerte)
        # Monitorausgabe vorgegebene Stellgroeszen
        PrintOutStellgroeszen(P_BHKW_el_soll, P_BHKW_th_soll, Modellspezifisch,
P_el_Last_zus, Betriebsmodus, P_SLK_th_soll, Tu, Tu_Speicher, P_HS_el_soll)
        # Werte speichern
        ErgebnisseDB(ModBusWerte)
        ErgebnisseCSV(ModBusWerte)
        Ergebnisse61850(ModBusWerte)

        Zeit+=1
    #--------------------------------------------------------------------------#
    # Berechnung der Zeit und warten auf den naechsten Schritt
    #--------------------------------------------------------------------------#
        while ((60. - (time.time() - now)) > 0):
            time.sleep(0.001) # 1ms warten, bis volle 60 sec. vergangen sind
  #----------------------------------------------------------------------------#
  # EXIT
  #----------------------------------------------------------------------------#
    except(KeyboardInterrupt, SystemExit):
        master.close()
        sys.exit("KeyboardInterrupt! quit!")
# raise SystemExit(0)
```

D Matlab Simulink Modell



E Inhalt der DVD

Teil dieser Arbeit ist ein Anhang in elektronischer Form, der auf der beiliegenden CD enthalten ist. Die Tabelle A.1 listet die enthaltenen Dateien mit einer kurzen Beschreibung auf.

Tabelle A.1: Der Inhalt der Beiliegenden DVD

Datei	Beschreibung
./Master_Gwy.py	Python-Datei des Gateways
./Server_MDL.py	Python-Datei des Simulationsmodells
./IEC61850SPH.zip	IEC 61850 Stack der RWTH Aachen
./Modell/Liegenschaft.slx	Matlab-Simulink Modell der Simulation
./Modell/Testbett_Liegenschaft.slx	Matlab-Simulink Modell der Simulation mit Testbett
./Liegenschaft_ert_rtw_Matlab.zip	Simulationsmodell als C-Code
./Liegenschaft_ert_rtw_RPi.zip	Bearbeitetet Simulationsmodell als C-Code
./Chart.py	Python-Datei zur Visualisierung der Simulationsdateien

Diese DVD ist bei Prof. Dr. Wolfgang Renz einsehbar.

Literaturverzeichnis

[Bar12] Bartmann, Erik: *Durchstarten mit dem Raspberry Pi*. 1. Auflage, Köln. O´Reilly Verlag, 2012

[Bra14] Bradbury, Alex; Everard, Ben: *Learning Python with Raspberry Pi.* 1. Auflage, Hoboken, New Jersey. John Wiley & Sons, 2014

[BSU11] Behörde für Stadtentwicklung und Umwelt, Unternehmen für Ressourcenschutz (2011): *BHKW-Check.* 2. Auflage Internet:*https://www.hamburg.de/contentblob/2490150/data/bhkwcheck-handbuch.pdf*, abgerufen am 17.09.2014

[Cam06] Camille Bauer AG (2006): *Modbus-Grundlagen* Internet:*http://www.camillebauer.com/src/download/Modbus_Grundlagen.pdf*, abgerufen am 17.09.2014

[Cra09] Crastan, Valentin: *Elektrische Energieversorgung*. 2. Auflage, Berlin. Springer, 2009

[Fra13] Rundel, Paul (2013): *Frauenhofer Umsicht: Speicher für die Energiewende.* Internet: *http://www.umsicht-suro.fraunhofer.de/content/dam/umsicht-suro/de/documents/studien/studie_speicher_energiewende.pdf*, abgerufen am 19.09.2014

[Ger11] Gers, Juan; Holmes, Edward J.: *Protection of Electricity Distribution Networks*. 3. Auflage, Herts, UK. The Institution of Engineering and Technology, 2011

[IDA06] Modbus Organization (2006): *MODBUS Messaging on TCP/IP Implementation Guide*.V1.0b Internet:*http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf*, abgerufen am 17.09.2014

[Nie20] Niehöster, Klaus (2010): *Energiewirtschaftliche Tagesfragen: Virtuelle Kraftwerke*. 60. Jg. Heft 9, S.20 ff.

[Sch10] Schaumann, Gunter; Schmitz, Karl W. (Hrsg.): *Kraft-Wärme-Kopplung*. 4., vollständig bearbeitete und erweiterte Auflage, Heidelberg. Springer, 2010

[Scn06] Schnell, Gerhard (Hrsg.); Wiedemann, Bernhard: *Busysteme der Automatisierungs- und Prozesstechnik*. 6., überarbeitete und aktualisierte Auflage, Wiesbaden. Vieweg, 2006

[SPH11] Hamburg Energie, HAW Hamburg, RWTH Aachen, BSU (2011): *Smart Power Hamburg: Das Projekt stellt sich vor.* Internet: *http://www.smartpowerhamburg.de/index.php/das-projekt.html*, abgerufen am 19.09.2014

[Sut08] Suttor Wolfgang; Weisenberger, Dietmar; Johler, Matthias: *Das Mini-Blockheizkraftwerk*. 4., durchgesehene Auflage, Heidelberg. Müller C.F, 2008

[Sut09] Suttor, Wolfgang: *Blockheizkraftwerke.* 7., vollständig überarbeitete Auflage, Karlsruhe. BINE Informationsdienst, 2010

[Zah10] Allelein, Hans; Bollin, Elmar; Oehler, Helmut; Schelling, Udo; Zahoransky, Richard (Hrsg.): *Energietechnik*. 5., überarbeitete und erweiterte Auflage, Wiesbaden. Vieweg+Teubner, 2010

Erklärung zur Bachelorarbeit

Ich versichere, dass ich meine Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Hamburg, 10.10.2014

_____

Sebastian Farrenkopf