Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Bachelorarbeit

im Studiengang Physik

vorgelegt von

Alexander Gorel

geboren in Karaganda, Kazachstan

März 2013

Integration einer automatisierten analogen und erweiterbaren Testumgebung zur Validierung und Überwachung von Hardware und Software Frameworks

Diese Bachelorarbeit wurde von Alexander Gorel ausgeführt am

Kirchhoff-Institut für Physik

Ruprecht-Karls-Universität Heidelberg

unter der Betreuung von

Prof. Dr. Karlheinz Meier

Integration of an automated analog and extensible test environment for validation and monitoring of hardware and software frameworks

In this thesis a system for digitalization and recording of analog signals was integrated into an Emulation Framework. In addition an automated test environment for verification and monitoring of Neuromorphic Hardware was put into operation.

Integration einer automatisierten analogen und erweiterbaren Testumgebung zur Validierung und Überwachung von Hardware und Software Frameworks

Im Rahmen dieser Arbeit wurde ein System zur Digitalisierung und Aufnahme analoger Signale in ein Emulations Framework integriert und es wurde eine automatisierte Testumgebung zur Verifikation und Überwachung neuromorpher Hardware in Betrieb genommen.

Inhaltsverzeichnis

1. Einleitung

Die Natur brachte in einem kontinuierlichen Evolutionsprozess ein einzigartiges biologisches System hervor. Das Gehirn.

Neurophysik beschäftigt sich nicht mit der biochemischen Konstitution des Gehirns, sondern versucht physikalische Eigenschaften und Dynamik neuronaler Systeme und deren Bestandteile, den Neuronen, auf mesoskopischer und makroskopischer Skala zu beschreiben und vorherzusagen.

Durch Organisation von Neuronen zu Neuronalen Netzen entstehen fehlertolerante emergente Systeme, die zur Abstraktion, Mustererkennung, Lernen und Entscheidungsfindung fähig sind. Im Rahmen des interdisziplinären *BrainScaleS-Projekts*[4] werden die Selbstorganisationsprinzipien Neuronaler Netzwerke auf unterschiedlichen Skalen durch verschiedene Ansätze erforscht und Möglichkeiten ergründet, diese auf elektronische Systeme zu übertragen.

Mit den gängigen Methoden der Biologie ist das *Neural Sampling*, also die Untersuchung der Signale eines Neuronalen Netzes, in großem Umfang noch nicht möglich. Ferner ist diese mit der Zerstörung oder einer enormen Beeinträchtigung verbunden und nichtinvasive Methoden bieten noch keine ausreichende Auflösung. Durch Modellierung der Dynamik Neuronaler Systeme auf Basis von bereits aggregierten biologischen Messungen und Emulation, also Nachahmung der Dynamik neuronaler Systeme auf Hardware, können aber biologisch relevante Resultate in reproduzierbaren Large Scale Experimenten erzielt und die Selbstorganisationsprinzipien Neuronaler Netzwerke erforscht werden. Diese können dann dazu herangezogen werden, die Multi-Skalen Dynamik Neuronaler Netze zu erklären.

Im Rahmen des *BrainScaleS Projektes* wird durch die *Electronic Vision(s) Group*[5] am *Kirchhoff-Institut für Physik* in einer Kontinuierlichen Integration ein hybrides System zur Numerischen Simulation und Emulation von Neronalen Netzwerken , die *Hybrid Multiscale Facility*, entwickelt, mit der die Funktion des Gehirns erforscht werden soll.

Ziel dieser Bachelorarbeit ist die Integration einer automatisierten analogen und erweiterbaren Testumgebung zur Validierung und Überwachung von Hardware und Software Frameworks, die die Funktionalität der *Hybrid Multiscale Facility* prototypisieren. Dies beinhaltet auch die Integration eines *Analog Readout System*, mit dem die emulierten Signale ausgelesen werden können.

2. Methoden

In diesem Kapitel werden Methoden erläutert, die bei der Durchführung dieser Bachelorarbeit angewandt wurden. Es wird zunächst der Aufbau, die Organisation und die Funktionen der verwendeten Hardware Frameworks skizziert.

Dann werden einige Software Frameworks genannt, die die Konfiguration und Steuerung dieser Hardware Frameworks ermöglichen.

Zuletzt werden Prinzipien der *Kontinuierlichen Integration* erläutert sowie die zur Umsetzung dieses Software Entwicklungs Prozesses verwendeten Tools.

2.1. Hardware Framework

Näheres zur Funktion des *Vertical Setup*, *Field Programmable Gate Array*, *System Emulator Board* und *HICANN Chip* ist in [13] und [12] beschrieben. Im Folgenden werden die Funktionalitäten und Aufgaben des Hardware Frameworks beschrieben.

2.1.1. HICANN Chip

Der *High Input Count Analog Neural Network* Chip dient der *Emulation*[1] neuronaler Membranspannungen und bildet den Grundbaustein für das FACETS Wafer Scale System.

Er gehört zu der Klasse der mixed-signal *Application Specific Integrated Circuit*[2](ASIC) und ermöglicht die parallele, beschleunigte Emulation des *Adaptive Exponential Integrate and Fire* Modells (AdEx)[3]. Die Emulation des synaptischen Inputs und des postsynaptischen neuronalen Membranpotenzials findet analog statt, die präsynaptischen Aktionspotenziale werden hingegen digital als *Neuronales Event* in Form eines Pakets repräsentiert.

[1] Software oder Hardware basierte Nachahmung eines Systems durch ein anderes System mit ähnlicher oder grundverschiedener Funktionsweise vgl. Wikipedia

[2] Anwendungsspezifische integrierte Schaltung

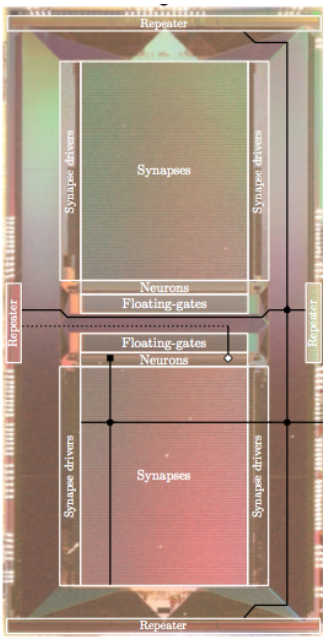


Abbildung 2.1.: HICANN Chip[13]

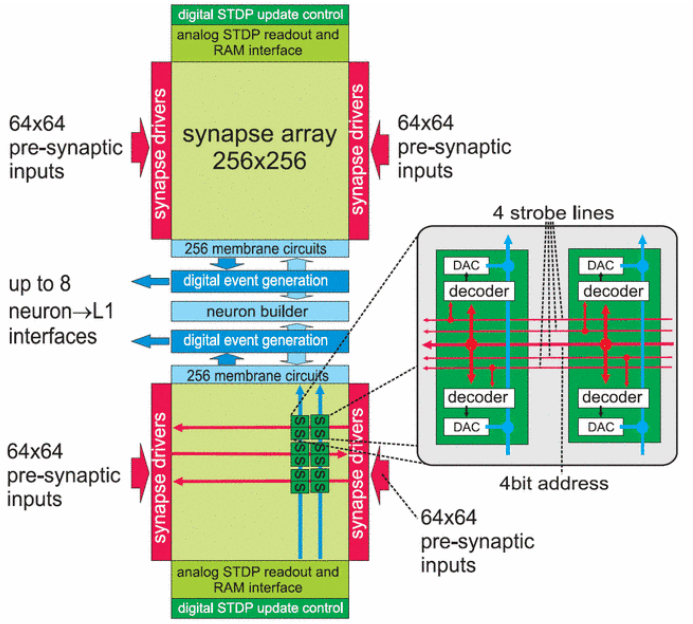


Abbildung 2.2.: HICANN Chip Schema[5]

Emulation Neuronaler Netze auf Hardware

Ein HICANN Chip verfügt über zwei *Denmen*-Blocks, die jeweils aus 256 *Denmem* bestehen. Diese Schaltkreise emulieren das AdEx Modell und können zu größeren Neuronen gruppiert werden. Bestehend aus 64 *Denmems*, kann ein Neuronen mit maximal 16k Synapsen emulieren werden.

Die emulierte Membranspannung wird vom *Neuron L1 Interface*[1] im Falle eines Spikes in ein digitales *Neuronales Event* umgewandelt, welches durch eine *Bus Struktur*[2] über *Repeater* zu den *Synapsen-Treibern* propagiert wird.

Repeater dienen der Signalregeneration und der Signalweiterleitung der *Neuronalen Events* auf dem HICANN Chip oder deren Einspeisung von außen.

Emulation Synaptischer Plastizität

Es befinden sich jeweils 56 *Synapsen-Treiber* an den Seiten der 2 *Analog Network Cores* (*ANNCORE*). Das *ANNCORE* dient der Erzeugung der emulierten synaptischen Signale und deren Weiterleitung in die *Denmen* Schaltungen. Ein *Denmem* ist mit 256 Synapsen-Emulations-Schaltungen verbunden.

[1] L1 steht für Layer 1 und bezieht sich auf die Kommunikationshierarchie auf dem Chip
[2] Schaltkreissystem für den parallelen Datentransport

Die Amplitude und die Dauer des synaptischen Signals wird im *ANNCORE* programmiert und wird zur Laufzeit des Experiments durch einen zuschaltbaren *STDP*[1] Controller angepasst, wodurch synaptische Plastizität emuliert wird.

Analoge Parametrisierung durch Floating Gates

Floating Gates sind programmierbare, nicht flüchtige Transistoren, die auf dem HICANN Chip als Speicher für analoge Parameter des AdEx Modells bei der Emulation der Membranspannung dienen.

Sie sind in 4 Arrays mit 24 Zeilen und 129 Spalten organisiert, von denen jeweils einer durch einen eigenen Controller programmiert wird.

Nicht alle *Floating Gates* werden zum Speichern der analogen Parameter verwendet, da zur Einstellung des AdEx Modells ledeglich 21 Parameter benötigt werden, von denen 12 Stromparameter und 9 Spannungparameter sind. Die *globalen Parameter*[2] des AdEx Modells werden ferner in der 0ten *Floating Gate Array Zeile* untergebracht.

Ein *Floating Gate* kann digital im Bereich von 0 bis 1023 programmiert werden, was einem spezifizierten *Floating Gate* Spannungsbereich von 0V bis 1.8V entspricht. Die Programmierung dieser Gates basiert auf dem quantenmechanischen Tunneleffekt und findet pulsweise statt, wobei die Effektivität eines Pulses mit jeder Wiederholung abnimmt. Bei der Programmierung wird während eines *Accelerator Step* die Programmierspannung erhöht, um diesen Effekt zu kompensieren.

Die Abhängigkeit der Programmierpräzision von der **maximalen Anzahl der Programmier-Pulse**, der **Pulsdauer**, der **Auslesezeit des Komparator** (welcher feststellt, ob der Programmiervorgang abgeschlossen ist) wurde in [13] und [12] untersucht.

Floating Gates, die Strom-Parameter representieren, unterscheiden sich schaltungsbedingt in der Programmiergenauigkeit von den Floating Gates, die Spannungs-Parameter representieren.

Die Kalibration der *Floating Gate* Programmierung ist eine Vorraussetzung für die Emulation des AdEx Modells auf dem HICANN Chip.

Die Programmierung der *Floatings Gate* Spannung hängt von systemweiten Parametern (Versorungsspannung, System Clock) ab und ist deshalb ein guter Indikator für die Funktionsfähigkeit des Gesamtsystems. Deshalb wird diese in einem analogen Test überprüft.

[1] Spike-Timing Dependent Plasticity
[2] Schaltungsspezifische Einstellungen

2.1.2. *System Emulator Board* und *HICANNmodule*

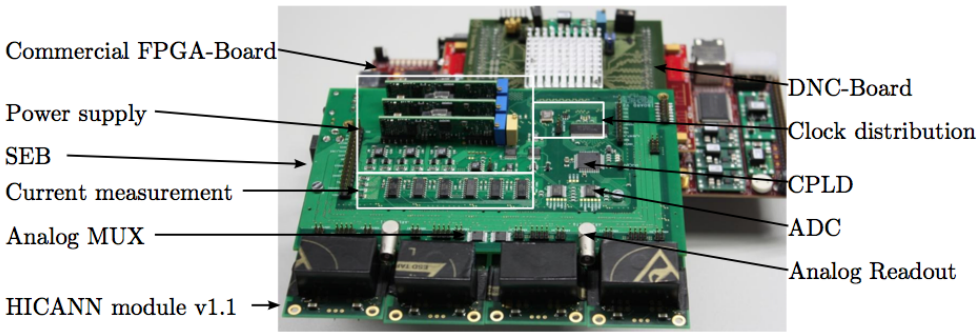


Abbildung 2.3.: System Emulator Board[13]

Das *System Emulator Board*[1] dient dazu, bis zu 8 HICANN Chips in Betrieb zu nehmen, um die Spannungsversorgung zu testen und deren Funktion zu überprüfen.

Stromversorgung

Es befinden sich 3 Gleichspannungswandler auf dem System Emulator Board. In einer überarbeiteten Version wurde nur ein Gleichspannungswandler eingebaut, da jeder von ihnen die Signalqualität des Analog Output des System Emulator Board negativ beeinflussen.

System Emulation

Das System Emulator Board wird dazu verwendet, die Topologie des FACETS Waver Scale System zu emulieren. Bis zu zwei HICANN Chips werden auf einem *HICANNmodule*[2] PCB[3] gebonded, Bis zu vier solcher Module können an das System Emulator Board angeschlossen werden.

Analoges Sampling

Auf jedem HICANN Chip sind 2 analoge Output Leitungen vorgesehen, auf die Outputs von Schaltelementen, wie Floating Gates oder aber von Schaltungen wie dem Denmem, zwecks Kalibration oder Evaluation gelegt werden können. Diese werden von zwei 50Ω terminierten Ausgangsverstärkern nach außen getrieben.

[1] Das System Emulator Board wurde von Dr. Sebastian Millner und Dr. Andreas Grübl an der Universität Heidelberg entworfen

[2] Das HICANNmodule wurde von Dr. Sebastian Millner an der Universität Heidelberg entworfen.

[3] Printed Circuit Board

Es können bis zu 8 HICANN Chips mit dem System Emulator Board in Betrieb genommen werden. Jeder der insgesammt 16 Analog Out kann auf einen der 2 analogen Ausgänge des System Emulator Boards durch eine Multiplex-Einheit gelegt werden.

Diese Outputs werden mit einem *Analog Readout System* digitalisiert und aufgenommen. Im Verlaufe dieser Bachelorarbeit wurde ein solches System in das bestehende Hardware und Software Framework integriert. Durch *analoges Sampling*[1] der Spannungssignale von Schaltelementen kann die Funktionsfähigkeit des Gesamtsystems überprüft werden. Ferner können wichtige Informationen während eines Neurophysikalischen Experiments auf dem HICANN Chip über das *Sampling* der emulierten Neuronen gesammelt werden.

[1] Stichprobenartige Aufnahme und Auswertung eines Signals

2.1.3. *FPGA Board*

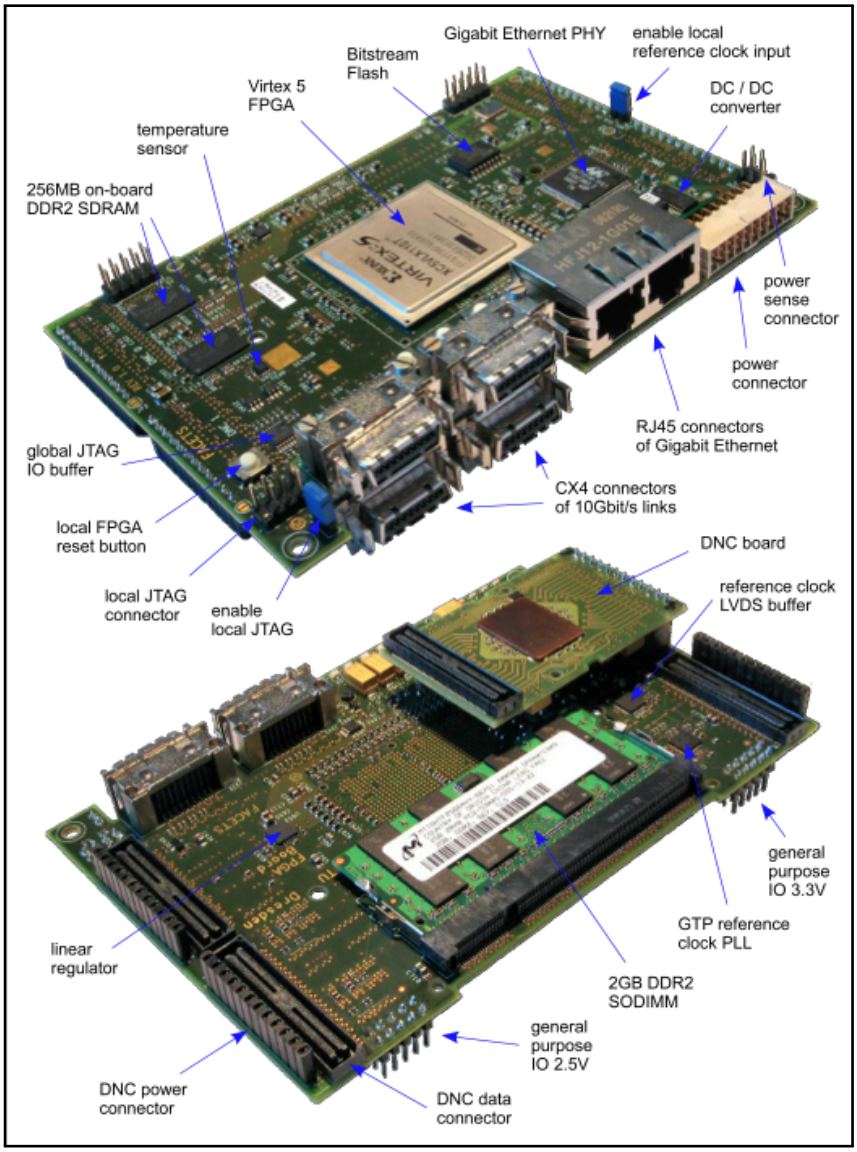


Abbildung 2.4.: Spezifikation des FPGA Boards[10]

Das *FPGA Board* beherbergt neben bis zu 4 *Digital Network Chips*[1] (*DNC*) ein Xilinx Virtex-5 LXT *Field Programmable Gate Array*[2] (*FPGA*).

Dieses Board übernimmt Aufgaben der Hardware-Konfiguration und Zeitablaufsteuerung sowie Kommunikation und Routing.

[1] Eine anwendungsspezifische integrierte Schaltung, die an der TU Dresden entwickelt wurde.
[2] Ein rekonfigurierbarer Chip, der unterschiedliche Steueraufgaben übernimmt.

Kommunikation und Routing

Der *FPGA* implementert Logik zur paketorientierten Kommunikation. Auf der Rückseite des *FPGA Boards* befinden sich 4 *DNC*, die der Vermittlung von Konfigurations- und Event-Paketen[1] zwischen HICANN Chip und FPGA dienen.

Jeder HICANN Chip verfügt über ein *DNC Interface*, welches Pakete zum *FPGA* schicken oder von diesem empfangen und weiterleiten kann.

Konfiguration

Der *FPGA* implementiert die Logik zur Konfiguration aller integrierten Schaltkreise auf dem Board. Ferner wird angeschlossene Hardware, insbesondere der HICANN Chip, durch Konfigurationspakete instruiert.

Der FPGA implementiert auch einen *JTAG-Controller* und ermöglicht damit die Validierung aller Hardware Komponenten, die als Glieder in der *JTAG-Kette*[2] geschaltet sind.

Ferner wird mit der *JTAG-Kette* eine langsamere und redundante Konfigurations-Schnittstelle bereitgestellt.

Zeitablaufsteuerung und Aufnahme

Der *FPGA* übernimmt bei der getriggerten Aufnahme emulierter Membransspannungen eine koordinierende Rolle, da er durch das Speisen der Neuronalen Events in den HICANN Chip und das Aussenden des Triggers zum *Analog Readout System* den Beginn des Experiments festlegt.

Er implementiert 2 *FIFO*[3] Strukturen, die zum Speichern von Neuronalen Events verwendet werden.

- **Puls-FIFO** dient dem Speichern von Paketen, die Neuronale Events auf dem HICANN Chip auslösen, sobald sie übertragen worden sind.
- **Trace-FIFO** dient dem Speichern von Neuronalen Events, die im Verlaufe der Emulation auf dem HICANN Chip enstehen.

[1] Enthält Neuronale Events, die entweder auf dem HICANN Chip in Synaptischen Output übersetzt werden oder im FPGA gespeichert werden

[2] Joint Test Action Group.

[3] First In First Out Buffer. In diesem Zusammenhang sind damit Strukturen gemeint, die Datenstrukturen speichern.

2.1.4. *Vertical Setup*

Das FPGA Board und das System Emulator Board bilden zusammen das *Vertial Setup*. Es dient der Prototypisierung des FACETS Wafer Scale Systems und wird für die Analyse und Verifikation der Hardware Funktion und Hardware Programmierung verwendet, sowie der Durchführung Neurophysikalischer Experimente.

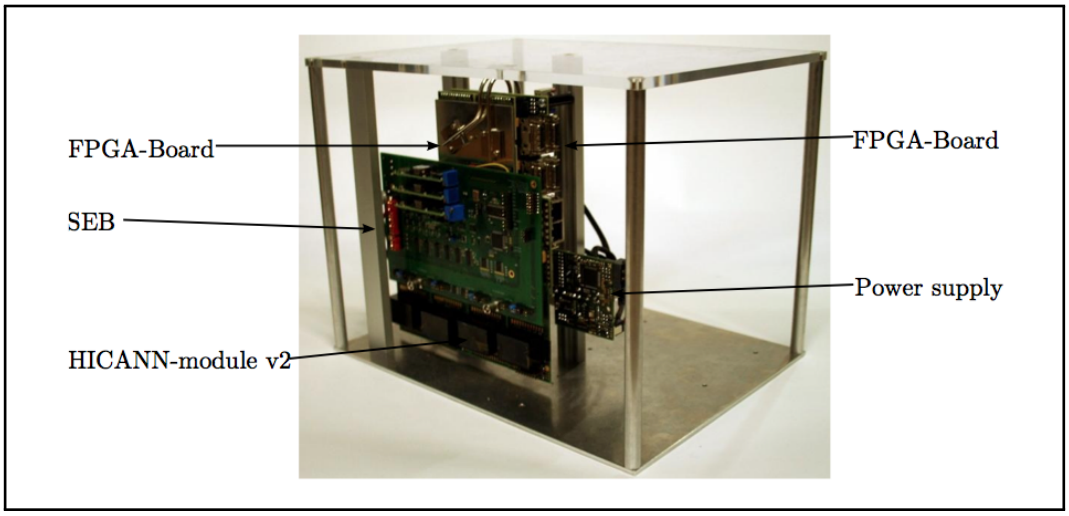


Abbildung 2.5.: Vertical Setup [13]

2.1.5. *Analog to Digital Converter Board*

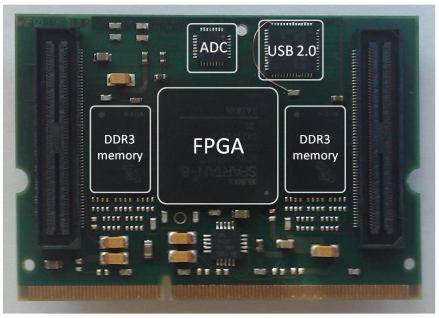


Abbildung 2.6.: ADC FPGA Board Vorderseite[1]

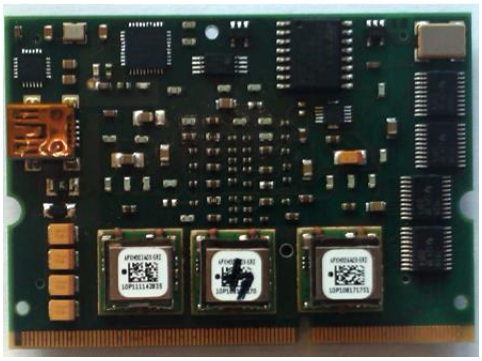


Abbildung 2.7.: ADC FPGA Board Rückseite[1]

Abbildung 2.8.: ADC Board Vorderseite

Das *Analog to Digital Converter Board* (*ADC Board*) ist ein für sich abgeschlossenes System, welches aus 3 Printed Circuit Boards (PCB) zusammengesetzt ist und bei der Evaluation des im HICANN Chip emulierten Spannungs-Signals die Aufgabe eines digitalen Oszilloskops übernimmt.

PCB	Schaltkreise	Aufgaben
Flyspi Board[a]	FPGA	Kommunikation, Konfiguration, Steuerung
	EEPROM	permanente Programmierung des FPGA
	ADC	Analog Digital Wandlung
	USB Chip	Kommunikation
	USB Interface	Kommunikation, Stomversorgung
	DDR 3 RAM	Speicherung von digitalisierten Signalen
Aufsteckboard	8 zu 1 Multiplexeinheit[b]	Signalselektion
	single-to-differential Signalverstärker	Arbeitsbereich Einstellung für den ADC
Breakoutboard	JTAG Interface	Programmierung des FPGA
	Analog Readout Interface	8 Input Kanäle, 2 Trigger Kanäle

Tabelle 2.1.: Printed Circuit Boards des ADC Boards

[a] Dieses Board wurde von Dr. Johannes Schemmel an der Universität Heidelberg entworfen.

[b] Selektionsschaltung zur Auswahl zwischen mehreren Eingangssignalen. Besteht hier aus 3 3-zu-1 Multipex Schaltelementen

Digitalisierung

Auf dem *Flyspi Board* befindet sich ein differentieller 12 bit *Pipeline Analog zu Digital Wandler*[4] (*ADC*) ADS6125 von Texas Instruments, der ein differentielles[5] analoges Signal digitalisiert. Für die Umwandlung des einfachen analogen Signalsi[6] in ein differentielles wird ein *Single-to-Differential Verstärker* vorgeschaltet, für dessen Betrieb der Eingangsreferenzwert des *ADC Boards* um 100mV angehoben wurde. Dies wird in der Kalibration des ADC Output berücksichtigt. Die Samplingfrequenz des Pipeline ADC beträgt 98 MSPS[7].

Konfiguration

Auf dem *Flyspi Board* befindet sich ein Xilinx Spartan6 FPGA, der unterschiedliche Funktionalität des *ADC Boards* implementiert. Dazu gehören unter anderen die Implementierung des *Protokoll Stack* für die USB Kommunikation, die Implementierung des

[4] Analog to Digital Converter. Pipeline bezieht sich auf die Umsetzung der Digitalisierung.

[5] Das Signal wird auf zwei Leitungen transportiert. Das einfache Signal ergibt sich durch Differenzbildung der beiden differenziellen Signale.

[6] An die 8 analoge Inputkanäle werden einfache Signale angelegt und müssen erst in differentielle umgewandelt werden, damit der ADC die Digitalisierung vornehmen kann

[7] Mega Sample Per Second

OCP Interface[1] zur Konfiguration der Integrierten Schaltkreise und die Logik zum Lesen und Schreiben des RAM.

Die Konfiguration und Ansteuerung der integrierten Schaltelemente sowie der Logik des FPGA wird durch Schreiben in eigens dafür vorgesehene Konfigurationsregister umgesetzt, die ebenfalls als FPGA Logik implementiert werden. Ein Konfigurationsregister ist 8 Bit breit und wird in einer atomaren Anweisung geschrieben.

Aufnahme und Auslesen

Die Speicherung der digitalisierten Signale wird durch das Schreiben in die zwei vorhandenen DDR RAM auf dem *Flyspi Board* realisiert. Der Schreibvorgang wird durch eine DMA[2] *Statemachine*[3] im FPGA durchgeführt, und kann konfiguriert und gesteuert werden. Die entsprechenden Konfigurationsregister werden in Anhang "Konfigurationsregister des Analog Readout Systemsäufgeführt.

Wenn der Schreibvorgang durch ein Software-Event[4] oder ein externes digitales Trigger-Signal ausgelöst wird, werden alle anderen eingehenden Trigger-Signale bis zur Beendigung des Schreibvorgangs ignoriert, um Dateninkonsistenz zu vermeiden. Durch ein *OCP Paket* kann der aufgenommene Datensatz angefordert und über USB übertragen werden.

Dieses Schreib- und Lese-Logik ist in einem *Bitfile*[5] implementiert, mit dem der FPGA programmiert wird.

Triggering

Das getriggerte Aufnehmen mit dem *ACD Board* ist im FPGA auf dem *Flyspi Board* durch zwei *Statemachines* umgesetzt.

Wird auf dem Trigger-Channel eine Flanke[6] detektiert, initialisiert die Trigger-State Machine das Schreiben in den RAM, indem es die Statemachine für das Schreiben startet, und blockiert die Detektion eines zweiten Trigger Signals, bis der Schreibvorgang abgeschlossen ist. Die Trigger-Statemachine kann so konfiguriert werden, dass sie einmalig oder repettitiv nach einer Trigger-Signal-Detektion den Schreibvorgang auslöst. Die Schreib-Statemachine kann durch ein *OCP Paket* manuell gestartet werden, welches

[1] Open Core Protokol
[2] Direct Memory Access
[3] Als FPGA Logik implementierter Automat, der Steuersignale erzeugt.
[4] Durch ein OCP Paket kann das Konfigurationsregister für Triggering geschrieben werden, sodass die Statemachine zum Schreiben in den RAM manuell gestartet werden kann
[5] Eine bereits übersetzte Datei mit Programmieranweisungen
[6] Übergang eines Logik Pegels von High zu Low oder umgekehrt

die entsprechenden Bits im Trigger-Konfigurationsregister setzt. Der Aufbau des Trigger-Konfigurationsregisters ist in Anhang "Trigger Konfigurationsregister"dargestellt. Das einstellbare Triggering-Verhalten ist in Anhang "Trigger Konfiguration"dokumentiert.

minimaler Trigger Pegel	2 V
minimale Trigger Dauer	20 ns (laut Simulation)

Tabelle 2.2.: Anforderungen an das Trigger-Signal

Multiplexing

Das *ADC Board* verfügt über 8 analoge Input Kanäle, von denen durch Konfiguration von drei 3-zu-1-Multiplexer, einer zur Digitalisierung ausgewählt werden kann.

Ein *ADC Board* kann zwischen zwei Triggerkanälen wählen, die das Trigger-Signal zum Starten tragen, was beim Betrieb von mehreren *ADC Boards* in größeren Systemen Anwendung finden soll.

Messbereichserweiterung

Aufgrund der Spezifikation des verwendeten ADC Schaltkreises können nur Signale im Bereich von 0 V bis 1 V digitalisiert werden. Der Spannungsbereich des analogen Ausgangs am System Emulator Board beträgt aber 0 V bis 1.8 V. Die Eingangsimpedanz des ADC Boards und die Ausgangsimpedanz des SEB[1] Ausgangstreibers betragen 50Ω. Da die Ausgangsspannung des Ausgangstreibers die Versorgungsspannung bei der Serienschaltung von SEB Ausgangsverstärker und ADC Board darstellt, ergibt sich die Spannungsteilerformel $\frac{U_{\mathrm{SEB}}}{R_{\mathrm{SEB}}+R_{\mathrm{ADC}}} = \frac{U_{\mathrm{ADC}}}{R_{\mathrm{ADC}}}$ und daraus $\frac{U_{\mathrm{SEB}}}{2} = U_{\mathrm{ADC}}$. Dadurch wird der Messbereich des ADC auf 0 V bis 2 V erweitert. Fertigungsbedingt kann das genaue Verhältnis davon abweichen und muss bei der Kalibration des ADC Output berücksichtigt werden.

[1] System Emulator Board

2.2. Hardware Schnittstellen

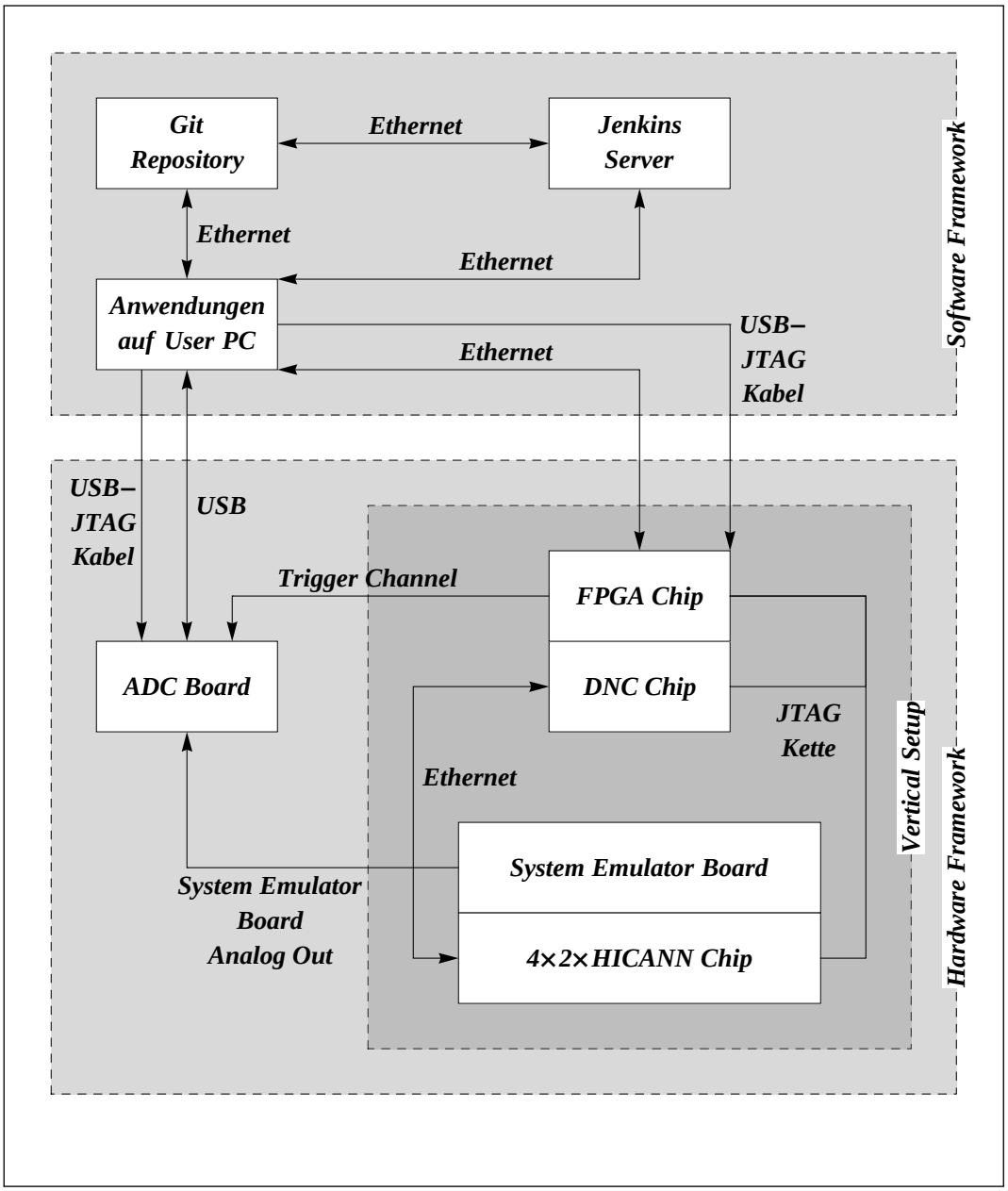


Abbildung 2.9.: Organigram der Hardware Schnittstellen

Abbildung 2.9 veranschaulicht die Organisation der verwendeten Hardware im Experiment. Dabei wird die Kommunikation zwischen den einzelnen Hardware Komponenten über unterschiedliche Schnittstellen realisiert.

2.2.1. Schnittstellen

USB (Universal Serial Bus) Ist ein serielles Bussystem, das zum Verbinden eines Computers mit externer Hardware verwendet wird. Es werden Hardware-Konfigurationspakete und Datenpakete übertragen. Zur Programmierung der FPGA wird ein *USB-zu-JTAG Platform Cable von XILINX* verwendet.

JTAG (Joint Test Action Group) Ist eine Schnittstelle, die zum Hardware-Debugging und zur Hardware-Konfiguration verwendet wird, indem zusätzliche Hardware Register in Kette geschaltet, geschrieben und gelesen werden können.

ETHERNET Ist eine Spezifikation, die die Hardware und Software für den Transport von Daten in einem Netzwerk beschreibt. Es werden Konfigurationspakete der Hardware Komponenten und Datenpakete übertragen.

2.3. Software Frameworks

In diesem Kapitel werden die Organisation und Aufgaben einiger Frameworks erläutert.

2.3.1. *Vmodule* Framework

Vmodule ist ein *Abstraktions Layer* des *ADC Boards* und stellt unterschiedliche Methoden zur Verfügung, die für den Betrieb des ADC Boards notwendig sind.

Die Konfiguration der Integrierten Schaltungen auf dem ADC Board findet über Funktionen statt, die verschiedene Konfigurationsregister schreiben.

Die Folgenden Dateien des Vmodule Frameworks definieren wichtige Funktionen für den Betrieb des ADC Boards

Headerdatei	**Funktionen**
Vmux_board.h	Schalten der Inputchannels
Vmodule_adc.h	Auslösen des Triggers, einstellen der Aufnahmedauer
Vmemory.h	Lese- und Schreib-Zugriff auf ADC Board RAM
Vmoduleusb.h Vusbstatus.h Vusbmaster.h Vocpfifo.h	Implementieren des Protokoll Stack: Packen und Entpacken von Kommunikations Paketen

Tabelle 2.3.: Wichtige Vmodule Dateien

2.3.2. *Halbe* Framework

Das *Hardware Abstraction Layer Back End* soll einen uneingeschränkten Zugriff auf die *Hybrid Multiscale Facility* (HMF) gewähren und die Konfiguration aller Hardware Komponenten ermöglichen.

Dieses Framework soll eine Grundlage für Anwendungsprogramme darstellen, die Neurophysikalische Experimete auf der HMF durchführen.

Headerdatei	**Funktionen**
HMFEnum	
HMFBackend	User Funktionen zum Arbeiten mit der Hardware
HMFUtil	Hardware Spezifische Zähltypen
HMFRun	
HMFGeometry	User orientierte Nummerierung der Hardware Komponenten
HMFConfig	

2.4. Inter Software Schnittstellen

2.4.1. Wrapping

Unter *Wrapping* versteht man das Einbinden der Funktionalität einer Programmiersprache in eine andere, was zum Vereinen der Vorteile unterschiedlicher Sprachen führen soll.

Bei einem *C++-Python-Wrapping* profitiert die Softwareentwicklung von der hohen Ausführungsgeschwindigkeit der kompilierten C++ Funktionen und von der Modularität und der Schnellinkeit des Programmierens und Testes mit Python.

Alle wichtigen Funktionen des *Hardware Abstraction Layers* werden durch einen Algorithmus bei der Kompilierung des *Halbe Projekts* automatisch gewrappt.

2.4.2. Scripting

Shell Scripting

Sowohl das Waf-Utility als auch die Google-Test-Main-Funktion können in Shellscripten zum Gestalten komplexerer Buids und Testdurchläufen verwendet werden. Durch parametrisiertes Ausführen dieser Anwendungen, kann das Ausführungsverhalten an die Anforderungen angepasst werden.

2.5. Kontinuierliche Integration

Die Implementation großer Software Projekte wird durch eine *Kontinuierliche Integration*[1] relisiert, sodass permanent Softwareänderungen, Rekonfiguration bereits integrierter Hardware oder Integration neuer (ggf. ungetesteter) Software bzw. Hardware vorgenommen werden.

Dadurch wird die bereits implementierte Funktionalität gefährdet, sodass in zeitaufwendigem und subjektivem Debugging die Wirkung der eingebrachten Änderung abgeschätz werden muss.

Die Anzahl involvierter Enwickler, begrenzter Systemüberblick und die zur Verfügung stehende Zeit setzen der Effektivität und Umsetzung des Debugging Grenzen.

Durch die Inbetriebnahme von Systemen zur Versionskontrolle, Standardisiertem Build Standardisiertem Testen und Automatisierung können diese Probleme der parallelen Entwicklung gelöst werden.

Dies erfordert allerdings auch, dass ein erweiterbares Testframework zur Verfügung gestellt wird, welches in einem *Test Driven Development*[2] durch Entwickler erweitert wird. Eine Standardisierung und Automatisierung des systematischen Testprozesses ermöglicht dann, das gesammte Projekt zu testen und Ausführungsverhalten im Dauerbetrieb zu analysieren.

Ferner ermöglicht die Automatisierung auch außerhalb der Arbeitszeiten, Projektbestandteile zu überprüfen, wodurch die effektive Entwicklungszeit erhöht wird.

Im folgenden werden die Umsetzungen dieser Software Entwicklungsparadigmen erläutert und auf die Hardwareentwicklung und -Integration übertragen.

2.5.1. Versionskontrolle und Distribution durch *Git*

Git ist ein verteiltes Versionsverwaltungssystem, das parallele Softwareentwicklung ermöglicht. Dabei findet die Softwareentwicklung lokal statt und die Resulate werden in ein globales Repository[3] submittiert, von wo aus sie an andere Entwickler verteilt werden können.

Der Entwickler kann mit diesem System

- eigene Änderungen submittieren
- die letzte stabile Version auschecken
- eine experimentelle Variante des bestehenden Codes erzeugen

[1] Ein Begriff aus der Software Entwicklung, der die kontinuierliche Erweiterung eines bestehenden Projektes bezeichnet.

[2] Die Softwareentwicklung wird in parallelem Testen und Implementieren durchgeführt.

[3] Ein verwaltetes Verzeichnis zur Speicherung und Beschreibung digitaler Objekte

- den Code in einen früheren stabilen Zustand zurückversetzen
- Konflikte beim Zusammenführen von Code Varianten auffinden und auflösen

Im Rahmen der Kontinuierlichen Integration, wird *Git*[6] dazu verwendet, Sourcecode eines ausgewählten Projekts bereitzustellen, welcher von einem Build-Tool in eine Anwendung übersetzt und innerhalb einer Testumgebung überprüft wird.

2.5.2. Standardisierter Build mit *Waf*-Utility

Von einem *Build* eines Projektes spricht man, wenn alle Bestandteile zu einem oder mehreren Anwendungsprogrammen kompiliert werden. Das Ergebnis eines solchen Vorgangs ist abhängig von seiner Umgebung, also den verwendeten Bibliotheken, den gesetzten Systemvariablen und dem verwendeten Übersetzungswerkzeug[1]. Damit das resultierende Anwendungsprogramm sich so verhält, wie vom Entwickler vorgesehen, muss die Build-Umgebung festgelegt und zur Verfügung bereit gestellt werden.

Waf[9] ist eine Python basierte Anwendung, die dazu dient, größere Projekte automatisch zu kompilieren. Sie versucht auch sicherzustellen, dass die Build-Umgebung für die Kompilierung mit der Umgebung übereinstimmt, unter der das jeweilige Projekt entwickelt und getestet wurde. Sollten *Dependencies*[2] (Bibliotheken, Tools oder Ähnliches) nicht auf der lokalen Installation vorhanden sein, so kann *Waf* dazu konfiguriert werden, vor dem Kompilierungsvorgang diese herunterladen und zu installieren.

Zusätzlich wird kann *Waf* konfiguriert werden, nach einer Kompilation und Veränderung des Projekts nur veränderte Projektbestandteilde neu zu kompilieren, um den Vogang der Übersetzung zu beschleunigen.

In der Kontinuierlichen Integration wird *Waf* dazu verwendet, die Komplilierung des Projektes auf verteilten Systemen durchzuführen.

2.5.3. Standardisiertes Testen im *Google C++ Testing Framework*

Das von Google entwickelte open source *Google C++ Testing Framework* standardisiert die Beschreibung und Ausführung von Tests. Durch Einbinden eines Interface zum *Device Under Test*[3] kann eine erweiterbare und portable Testumgebungen für die verteilte Entwicklung von Hardware und Software implementiert werden.

Im Google Test Primer[8][7] werden die folgenden Anforderungen an die Implementierung der *Google Tests* gestellt.

[1] Compiler
[2] Source Code Abhängigkeiten
[3] software seitiges oder hardware seitiges Testobjekt

1. Tests sollten unabhängig und wiederholbar sein.
2. Sollten gut organisiert sein und die Struktur des getesteten Codes wiederspiegeln.
3. Test sollten portabel und wiederverwendbar sein.
4. Tests sollten beim Fehlschlagen so viel Informationen wie möglich über das Problem mitteilen.
5. Test sollen automatisch von dem Framework erkannt und organisiert werden.
6. Test sollen schnell laufen.

Ein *Google Test* wird durch die Überprüfung von *Assert Statements*[1] realisiert und schlägt dann fehl, wenn sich bei der Durchführung das Resultat einer Operation von dem Erwarteten über eine Toleranzgrenze abweicht.

Mehrere implementierte *Google Tests* werden in ein Test Programm, welches bereits durch das Framework bereitgestellt wird, eingefügt. Das Erkennen, Organisieren und Ausführen der Tests, wird durch das Program automatisch übernommen.

Stufe	**Aufgaben**
Test Fixture	stellt Methoden für die Test-Durchführung bereit konfiguriert das *Device Under Test* in einen Anfangszustand räumt nach der Test-Durchführung auf
Test	verwendet Methoden der Test Fixtures besteht aus *Asserts Statements*, die evaluiert werden
Test Case	besteht aus mehreren Tests
Test Program	führt die Test Cases aus bietet Konfigurationsmöglichkeiten zur Ausführung an erzeugt einen XML formatierten Testbericht

Tabelle 2.4.: Hierarchischer Aufbau eines Test Programms

Die Ausführung und die Konfiguration des Test Programms kann in einem Terminal stattfinden. Zu den einstellbaren Optionen zählen:

- Ausführen von einzelnen Tests oder von Test Cases (Test Filter)
- Festlegung einer Ausführungsreihenfolge (seriell, gemischt)
- wiederholtes Ausführen von Tests
- parametrisiertes Ausführen (Auswählen eines *Device Under Test*)

[1] Ein Assert Statement ist eine Behauptung, in diesem Fall die Behauptung, dass das erwartete Ergebnis einer Operation mit dem tatsächlichen Ergebnis derselbigen übereinstimmt.

Damit können *Google Tests* zum manuellen Debugging benutzt, oder aber komplexere Debugging Routinen in Shell Scripten implementiert werden.

Ferner können andere Anwendungsprogramme die Ausführung von *Google Tests* triggern, was in einen Automatisierungsschritt stattfindet.

2.5.4. Automatisierung mit *Jenkins*-Anwendung

Jenkins[14] ist eine serverseitige Anwendung für die routinierte Ausführung von Aufgaben.

Die Konfiguration dieser Anwendung ermöglicht

- die Ausführung eines getriggerten[1] oder anberaumten Build
- routiniertes Testen des Kompilats
- Automatisiertes Reporten bei Fehlschlägen des Kompiliervorgangs oder der Tests an einen Adressat

Bei einer fortgeschrittenen Automatisierung kann auch die Veröffentlichung von verifiziert stabiler Software durch ein solches System übernommen werden.

Intern greift die *Jenkins*-Anwendung auf die Funktinalität des *Git*-Systems und des *Waf*-Utility zurück.

[1] (zum Beispiel durch Einreichen neuer Commits in ein Repository)

3. Ergebnisse

Zur Integration der analogen Testumgebung wurde die bestehende Hardware-Abstraktion erweitert, ein Hardware Setup in Betrieb genommen und kalibriert, eine Modellierung des Device Under Test durchgeführt, ein Test implementiert, der die Analog-zu-Digal Wandlung verwendet, und die Durchführung von digitalen und analogen Tests automatisiert.

Im Folgenden werden diese Schritte im Einzelnen beschrieben.

3.1. Erweiterung des *Hardware Abstraction Layer*

Durch das *Vmodule* Software Framework werden unterschiedliche Funktionen implementiert, die die softwareseitige Konfiguration des *Analog-zu-Digital-Wandler* Boards (*ADC Board*) ermöglichen. Für die Erweiterung des *Hardware Abstraction Layers*[1], wurden diese in ein objektorientes *Application Programming Interface*[2] integriert.

Die Erweiterung des *Hardware Abstraction Layers* wurde in zwei Schritten durchgeführt: Das *Application Programming Interface* von *Vmodule* Framework wurde im ersten Schritt an die Anforderungen des *Hardware Abstraction Layers* angepasst und im zweiten Schritt in dem *Hardware Abstraction Layer Back End* (*Halbe*) integriert.

Zunächst wurde eine *Koordinate* für das *Analog zu digital Wandler Board* (*ADC Board*) eingeführt. Dieses Konstrukt dient dazu, nach zukünftiger anpassung der Firmware des *ADC-Board-FPGA* eine eindeutige softwareseitige Zuordnung der *ADC Boards* umzusetzen, sodass eine simultane Bedienung von mehreren *ADC Boards* möglich wird.

Ein *Handle*[3] für den Kommunikationsaufbau und Kommunikationserhalt mit dem *ADC Board* wurde implementiert. Dieser erzeugt bei jeder Instanziierung eine neue Verbindung mit einem *ACD Board*. Dadurch wird sichergestellt, dass kein anderer Anwender zum gleichen Zeitpunkt auf das ADC Board zugreifen kann. Zusätzlich wurden Funktionen im *Handle* implementiert, die die Funktionalität des *Vmodule* Frameworks für die Konfiguration und Steuerung *des ADC Boards* verwenden.

[1] Software, die Hardware Funktionen abstrahiert und damit eine Schnittstelle für allgemeinere Anwendungen bietet

[2] Funktionen und Datenstrukturen, die dem Entwickler beim Programmieren neuer Software zur Verfügung stehen

[3] Datenstruktur zur Referenzierung einer vom Betriebssystem verwalteten Systemressource, in diesem Fall dem ADC Board

Hier findet auch die Umrechnung der Aufnahmezeit in Anzahl der *Samples*[1] des Analog zu Digital Wandlers statt.

Da dieser Code auch in Python basierten Anwendungsprogrammen durch *Wrapping*[2] zur Verfügung stehen soll, wurde bei der Implementierung auf die Typsicherheit geachtet, sodass nur Funktionsparemeter in vorgeschriebenen Parameter Bereichen zugelassen sind. Es ist also zum Beispiel nicht möglich, einer Funktion, die den Trigger Channel des ADC Boards einstellt, einen dritten Trigger Channel als Argument zu übergeben.(ADC Boards haben nur zwei Trigger Channel) Die Umsetzung einer *Hardware Abstraction Layer* konformen Integration wurde mit Christoph Koke koordiniert.

Folgende Funktionen wurden implementiert:

`ADC`	Instanziieren eines Handle und Aufbauen der Verbindung zum Board
`trigger_now`	Manuelles Aufnehmen (nur bei DC Signalen sinnvoll)
`prime`	Wiedereinschalten des Triggers
`Config`	Konfiguration des Input Channel, Trigger Channel, Trigger Modus, Aufnahmedauer in μs
`get_trace`	Auslesen der aufgenommenen Datei

Nach dieser Erweiterung stehen dem *Hardware Abstraction Layer* Funktionen zur Verfügung, die für die Aufnahme und Digitalisierung analoger Signale notwendig sind.

- Herstellen einer USB Verbindung zu einem ADC Board
- Umschalten der Input Kanäle
- Umschalten der Trigger Kanäle
- Manuelles Triggering
- Konfiguration zum externen Triggering
- Einstellen der Aufnahmedauer
- Auslesen des DDR RAM auf dem ADC Board

Bei der Erweiterung des Hardware Abstraction Framework stellte sich die Verwaltung der Kalibrationsdaten für die Analog-Digital-Wandlung als nich sinnvoll heraus. Zukünftig sollen die Ergebnisse automatisierter Kalibrationen für die *ADC Boards* in einer Datenbank hinterlegt werden, die in Python basierten Anwenderprogrammen zur Umrechnung des *ADC Output*[3] in Volt Verwendung finden soll.

Der Betrieb mehrerer ADC Boards in einer Software Anwendung ist momentan noch nicht möglich, da dazu die Programmierung des FPGA im ADC Board geändert werden muss.

[1] Messvorgang eines Signals

[2] Einbinden der Funtionalität einer Programmiersprache in eine andere

[3] Digitaler 12 bit breiter Wert, der durch den Analog Digitalwandler bei der Digitalisierung erzeugt wird.

Ferner sollten weitere Anpassungen der Firmware des FPGA für eine robustere softwareseitige Steuerung implementiert werden, sodass zum Beispiel innerhalb einer Anwendung festgestellt werden kann, ob ein getriggerter Schreibvorgang abgeschlossen wurde.

3.2. Inbetriebnahme des Setup

In diesem Kapitel wird die Vorbereitung der Hardware und Software auf die Implementierung analoger Tests beschrieben. Dazu wurden die implementierten Software-Hardware Schnittstellen an der Hardware validiert. In einem weiteren Schritt wurde eine Kalibration für die Digitalisierung analoger Signale durchgeführt.

3.2.1. Testen der Halbe Funktionalität

Zum Testen der neu implementierten sowie der bereits bestehenden Programmier-Schnittstellen wurden von mir geschriebene[1] Python basierte Kontroll-[2], Visualisierungs-[3] und Kalibrationstools[4] verwendet.

Testen der *ADC Board Halbe* Programmierschnittstelle

Durch die oben genannten Tools wurde das ADC Board konfiguriert und gesteuert, sodass Signale eines *Arbitrary Waveform Generator*[5] (*AWG*) aufgenommen, ausgelesen und visualisiert werden konnten. Mit diesem Signalgenerator konnte auch manuell ein Trigger-Signal erzeugt werden, dessen Pegel der Spezifikation des ADC Boards entsprach.

Dadurch konnten das Umschalten der Input-Channels und der Trigger-Channels, die Aufnahme-Funktion und die manuelle sowie die externe Trigger-Funktion verifiziert werden.

[1] Im vorher durchgeführten Praktikum wurden die Tools auf Grundlage der **Python Schnittstelle zum Vmodule Abstraction Layer** implementiert und validiert und währen der Bachelorarbeit auf **pyhalbe**, der Python Schnittstelle zum Halbe Framework umgestellt und um Funktionen zwecks Kalibration erweitert.

[2] Kann im interaktivem Modus alle wichtigen Funktionen des ADC Boards steuern. Ferner implementiert es Arbeitsabläufe, mit denen interaktiv Kalibrationsdaten aufgenommen und gespeichert werden können

[3] Visualisiert die Aufnahme, die momentan in dem RAM des ADC Boards gespeichert ist

[4] Führt eine Parameteroptimierung für nichtlineare Modelle durch

[5] Arbiträr-Funktionsgenerator

Testen der *Vertical Setup* Programmierschnittstelle

Zur Inbetriebnahme des Vertical Setup wurde der FPGA im Vertical Setup mit der neuesten Firmware programmiert und mit einem Shell Script reinitialisiert. Das ADC Board wurde dann als Analog Readout System des Vertical Setup angeschlossen[1].

Die Konfiguration und Steuerung des Vertical Setup erfolge mit C++ basierten Programmen, da die in der Python Schnittstelle gefundenen Fehlfunktionen nicht ohne Weiteres behoben werden konnten.

Damit konnten die Software-Funktionen zum Programmieren der Floating Gates und zum Umleiten des auf dem HICANN Chip generierten Spannungssignals auf einen der beiden analogen Outputs des System Emulator Boards verifiziert werden.

Der FPGA im Vertical Setup erzeugt beim Starten eines Experiments ein 2 V Trigger-Signal, das für die Dauer des Experiments gehalten wird. Um ein solches Trigger-Event softwareseitig zu produzieren, muss der *Puls-FIFO* (siehe "Zeitablaufsteuerung und Aufnahme" im Kapitel "FPGA Board") mit einem Dummy-Event-Paket geladen werden.

Mit einem Oszilloskop wurden der Pegel und der Ausgangsport des vom FPGA erzeugten Trigger-Signals überprüft. Der Ausgangsport des Trigger-Signals auf dem FPGA Board wird von der Firmware festgelegt.

3.2.2. Kalibration des ADC Outputs

Zur digitalen Verarbeitung der analogen Signale ist eine Analog-Digital-Wandlung notwendig. Aufgrund der Produktionsungenauigkeit der Integrierten Schaltkreise, differiert das Ergebnis der Digitalisierung zwischen verschiedenen Analog-zu-Digital-Wandlern. Ferner muss auch beachtet werden, dass diese in unterschiedlichen Hardwareumgebungen integriert sind und innerhalb verschiedener Systemzustände[2] betrieben werden.

Um Vergleichbarkeit der Messergebnisse zu erreichen, müssen die digitalisierten Spannungssignale in Volt umgerechnet werden. Im Folgenden werden der Idealfall und der Realfall einer Digitalisierung mit dem ADC Board und die dabei erhaltenen ADC-Output-zu-Volt Kalibrationen diskutiert.

Es wird auch die Impedanzanpassung des ADC Boards und des analogen Ausganges des System Emulator Boards diskutiert.

[1] Es können auch ein digitales Oszilloskop angeschlossen werden

[2] System-Temperatur und System-Spannungsversorgung unterliegen Schwankungen

Gleichspannungssignale

Die Kalibration des ADC Output wurde mit Gleichspannungssignalen durchgeführt, die allerdings durch unterschiedliche Effekte gestört wurden.

Widerstandsrauschen: thermischen Bewegung der Ladungsträger führt Stört die elektrische Leitung und Ruft ein Rauschen in elektrischen Schaltkreisen hervorgeht.

Rauschen des Ausgangsverstärkers: Störungen der Versorgungsspannung:

Crosstalk: bezeichnet die Elektromagnetische Einkopplung von Signalen, die auf benachbarten Leitungen getrieben werden. Es können periodische oder aperiodische Signale einkoppeln.

Jitter: Schwankung in der Signalbreite der *ClockDigitales Regelsignal*. Diese Störung kann sich auf unterschiedliche Weise auf die Signalqualität auswirken. In diesem Fall wird durch Jitter die Samplingzeit des ADC verrauscht, was eine Unsicherheit in die Digitalisierung einbringt.

Fehlanpassung: Da Wechselspannungssignale in die System-Emulator-Board-zu-ADC-Board Leitung einkoppeln, können diese bei einer Fehlanpassung reflektiert werden, sodass sie mehrmals das Gleichspannungssignal stören können.

Ferner muss natürlich berücksichtigt werden, dass das Gleichspannungssignal der Spannungs-Quelle ebenfalls Störungen unterliegt.

Zur Ermittelung des Gleichspannungspegels, wurde angenommen, dass die meisten Störungen normalverteilt sind und die übrigen einseitigen Störungen selten und kurz sind, sodass der Mittelwert des Signals nur leicht von dem Pegel abweicht und als Schätzwert verwendet werden kann.

Zur Rechtfertigung dieser Annahmen sind die Störungen eines Gleichspannungssignals in Abbildung 3.1 und Abbildung 3.2 dargestellt.

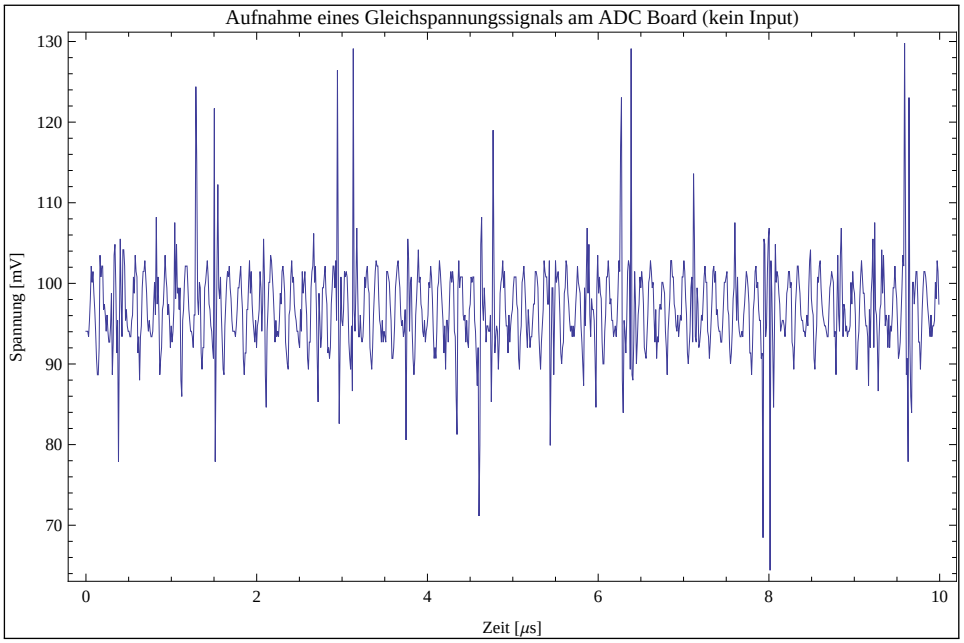


Abbildung 3.1.: Gestortes Gleichspannungssignal, digitalisiert und aufgenommen mit dem ADC Board. Dies stellt den besten anzunehmenden Fall dar, weil durch Zuschaltung einer externen Spannungsquelle zusätzliche Störungen des Signals stattfinden.

Abbildung 3.1 zeigt ein gestörtes digitalisiertes Gleichspannungssignal. Gemessen wurde mit offenem ADC Input (keine externe Spannungsquelle), um die Einflüsse von Crosstalk auf dem *Printed Circuit Board*[1], des Differentialverstärkers, Jitter der Clock und des Schaltreglers[2] des Gleichspannungswandlers sowie die Störung der Versorgungsspannung abzuschätzen.

[1] Eine Platine, die elektrische Verbindungen zwischen aufgelöteten Schaltelementen herstellt
[2] Erkennbar als kleine Spannungsspitzen

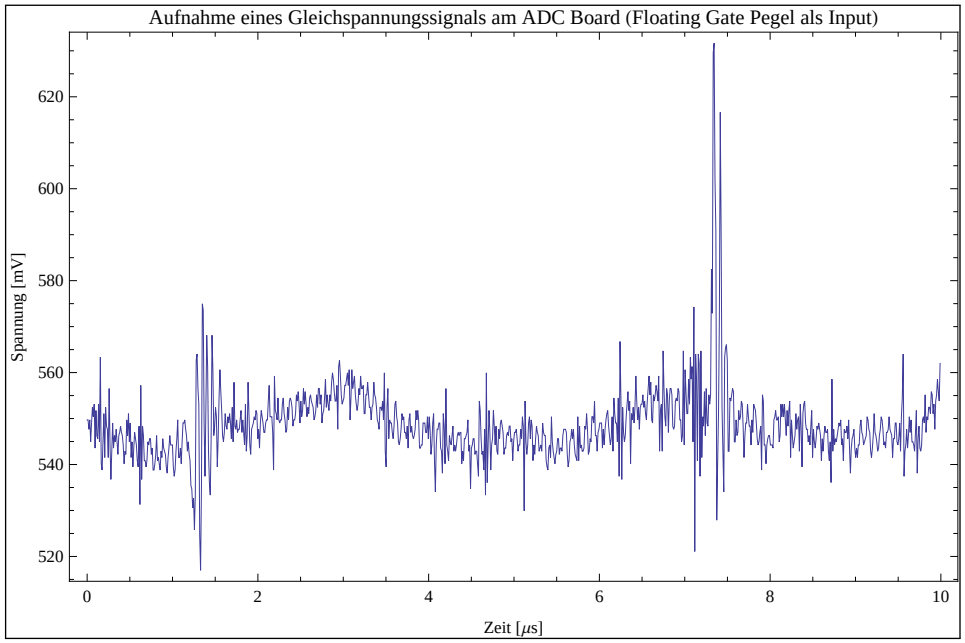


Abbildung 3.2.: Gestörtes Gleichspannungssignal am Ausgang des System Emulator Board, digitalisiert und aufgenommen mit dem ADC Board. Dies stellt den realistischen Fall dar, weil durch Signalleitung und Signalverstärkung weitere Störungen stattfinden.

Abbildung 3.2 zeigt ein gestörtes Gleichspannungssignal des System Emulator Board Ausgangsverstärkers, der das Gleichspannungssignal eines *Floating Gate Transistor, zur Speicherung von arbiträren Spannungswerten*herausleitet, welches aus dem HICANN Chip zum Analog Out des System Emulator Board getrieben wurde. Zusätzlich zu den Störungen auf dem ADC Board, wird das Signal durch Crosstalk auf dem HICANN Chip und durch das Rauschen des Ausgangsverstärkers sowie durch die Schaltregler[1] auf dem System Emulator Board und dem HICANN Chip gestört.

Kalibration mit 50Ω serienterminierter Spannungsquelle

Idealerweise sollte sich der getriebene analoge Ausgang des System Emulator Boards wie eine 50Ω serienterminierte reale Spannungsquelle verhalten, sodass bei einer Serienschaltung mit dem 50Ω terminierten ADC Board ein Spannungsteiler vorliegt.(siehe auch Kapitel "Messbereichserweiterung")

Deshalb wurden zur Erstellung der Kalibration aller Input Kanäle des ADC Boards eine DC Spannungsquelle mit seriengeschaltetem 50Ω Widerstand verwendet.

Dazu wurde das ADC Board als Gleichspannungsmesser der Spannungsquelle mit vorgeschaltetem 50Ω widerstand betrieben. Es wurden Datenpunkte für eine Gleichspannungs-

[1] Erkennbar als große Spannungsspitzen

kalibration im Bereich von 0V bis 2V gegen die ensprechenden mittleren ADC Outputs und deren Standardabweichungen aufgenommen.

Im Folgenden werden ADC Output Werte zu einem Gleichspannungssignal durch Digitalisierung und Aufnahme eines zeitbegrenzten Spannungssignals und Mittelung der resultierenden *Trace*[1] geschätzt.

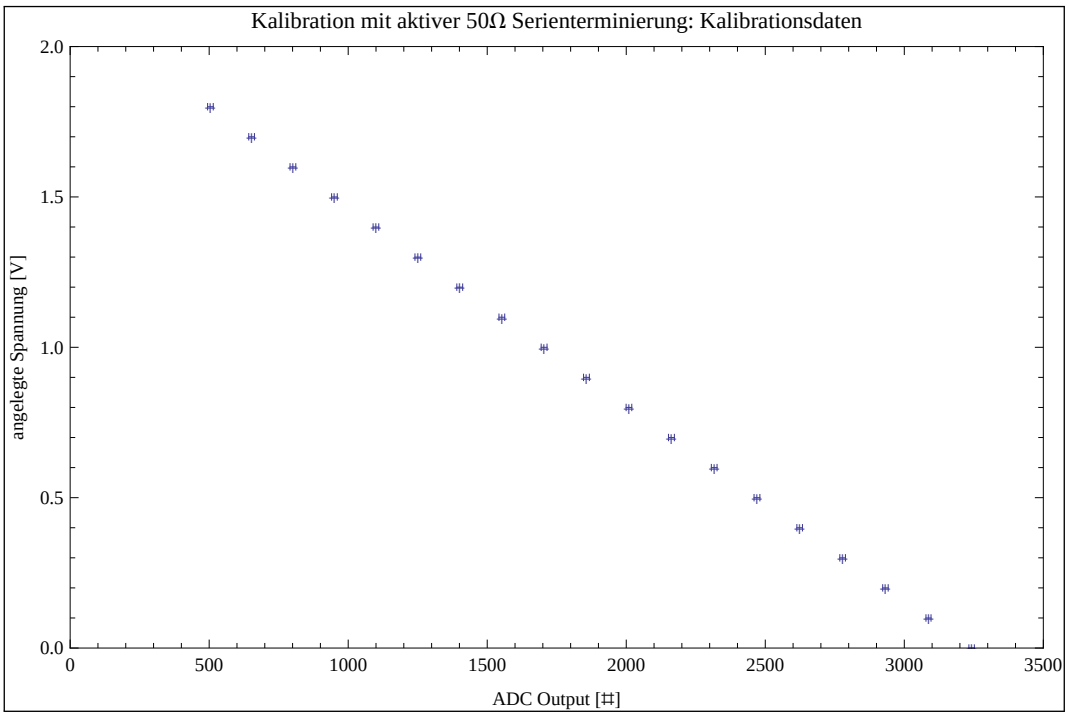


Abbildung 3.3.: Aufgenommene Daten für eine Kalibration des ADC-Input (in Volt) gegen mittleren ADC-Output. Aufgetragen ist der angelegte Gleichspannungswert (gemessen mit einem Multimeter), über dem Mittelwert des ADC-Output. Der geringe Fehler des abgelesenen Multimeterwerts ist aus darstellungsgründen nicht eingetragen. Der Fehler des ADC-Output wird als Standardabweichung der Trace geschätzt.

Zum Bestimmen der Kalibrationsfunktion ergeben sich an dieser Stelle zwei Möglichkeiten:

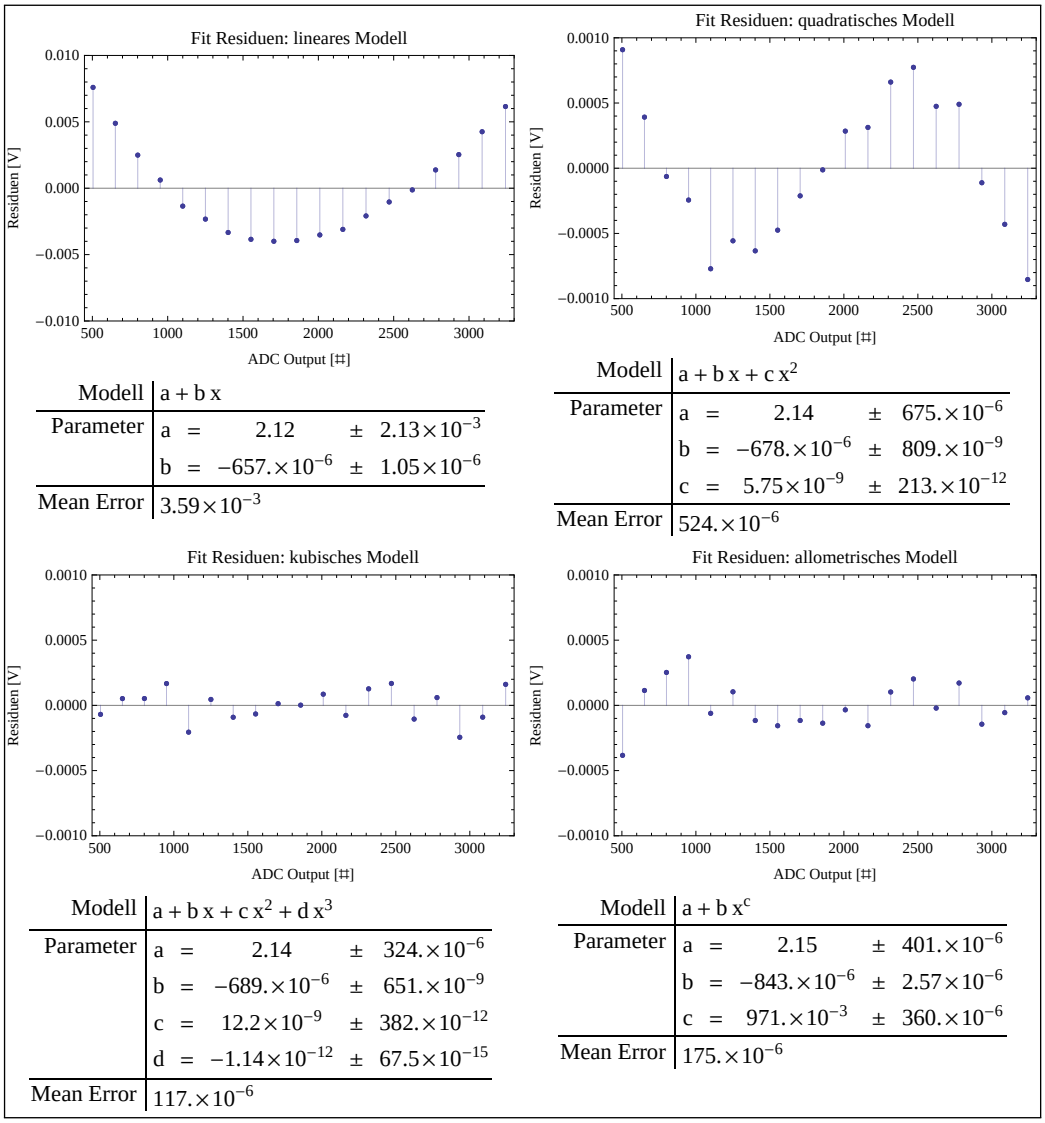
Zum einen kann der mittlere ADC-Output als Funktion des ADC-Input bestimmt werden und durch Anwenden der Umkehrfunktion eine ADC-Output zu ADC-Input Kalibration gefunden werden. Diese Umrechnung würde aber bewirken, dass neben dem Fitting Fehler der Modellparameter weitere Präzision durch die für die Umrechnung notwendige Division verloren ginge. Ferner ist die Umkehrfunktion bei Polynomiellen Modellen höheren Grades nicht trivial.

[1] Datenspur, aus mehreren ADC Outputs besteht

Die zweite Möglichkeit ist das direkte Fitten des ADC-Input als Funktion des ADC-Output.

Die letztgenannte Variante wurde bevorzugt, da sie eine direkte Bestimmung der ADC-Input zu ADC-Output Kalibrationsfunktion zulässt.

Es wurden vier Modelle gegen die Kalibrationsdaten getestet. Als Metrik für die Qualität des Fitergebnisses wurden der Mittlere Fehler herangezogen. Zusätzlich wird die Verteilung der Residuen betrachtet, um den Bereich, in dem das Modell und die Daten gut übereinstimmen, abzuschätzen. Die Ergebnisse des Modelluntersuchung sind in 3.4 zusammengetragen. Das Fitting der Modelle gegen die Kalibrationsdaten wurde mit einem selbstgeschriebenen Kalibrationstool durchgeführt, welches unter Verwendung von Python Bibliotheken (numpy, pylab und scipy) eine Optimierung von Modellparametern durchführt.



Modell	$a + b\,x$		
Parameter	a =	2.12	± 2.13×10^{-3}
	b =	$-657.\times10^{-6}$	± 1.05×10^{-6}
Mean Error	3.59×10^{-3}		

Modell	$a + b\,x + c\,x^2$		
Parameter	a =	2.14	± $675.\times10^{-6}$
	b =	$-678.\times10^{-6}$	± $809.\times10^{-9}$
	c =	5.75×10^{-9}	± $213.\times10^{-12}$
Mean Error	$524.\times10^{-6}$		

Modell	$a + b\,x + c\,x^2 + d\,x^3$		
Parameter	a =	2.14	± $324.\times10^{-6}$
	b =	$-689.\times10^{-6}$	± $651.\times10^{-9}$
	c =	12.2×10^{-9}	± $382.\times10^{-12}$
	d =	-1.14×10^{-12}	± 67.5×10^{-15}
Mean Error	$117.\times10^{-6}$		

Modell	$a + b\,x^c$		
Parameter	a =	2.15	± $401.\times10^{-6}$
	b =	$-843.\times10^{-6}$	± 2.57×10^{-6}
	c =	$971.\times10^{-3}$	± $360.\times10^{-6}$
Mean Error	$175.\times10^{-6}$		

Abbildung 3.4.: Zusammentragung der Modelluntersuchung. Dargestellt sind Fitresiduen und mittlerer Fehler sowie Fit-Parameter und deren Fehler

Die Betrachtung der Fitresiduen und des mittleren Fitting Fehlers lassen den Schluss zu, dass ein Polynom dritten Grades oder eine Allometrische Funktion die Kalibrationsdaten am besten beschreiben.

Kalibration mit Floating Gate Output

Im Anwendungsfall wird das ADC Board das Spannungsignal am analogen 50Ω terminierten Ausgang des System Emulator Boards digitalisieren und aufnehmen. Der Gleichspannungspegel der programmierten Floating Gates kann durch Stromspiegel und Multiple-

xing zu diesem analogen Ausgang geleitet werden. Duch Programmierung eines Floatings Gates, dessen Spannung zum Analog Out herausgeleitet wurde, lassen sich unterschiedliche Gleichspannungspegel erzeugen. Die Ausgangsspannung am Analog Out des System Emulator Board kann mit einem Multimeter abgegriffen werden. Durch wiederholt abwechselndes Messen mit dem ADC Board und einem Multimeter können Daten für eine zweite Kalibration aggregiert werden.

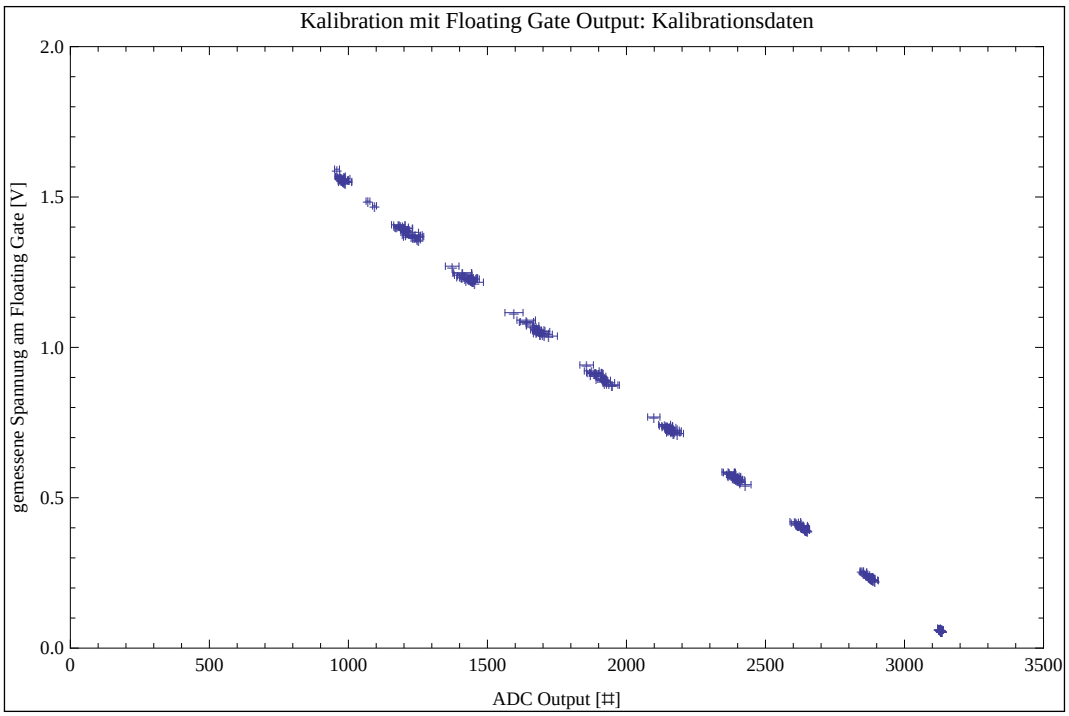
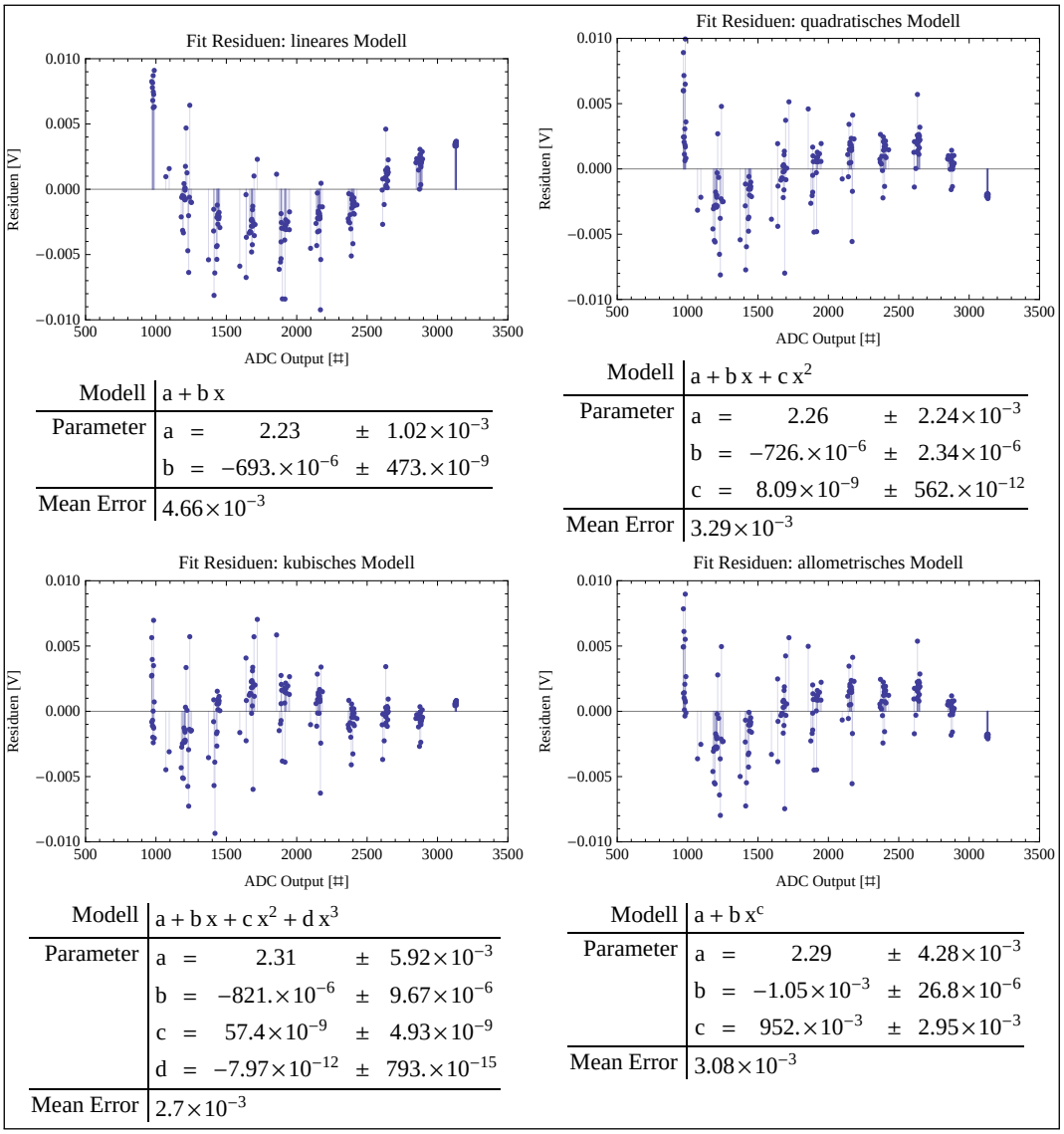


Abbildung 3.5.: Aufgenommene Daten für eine Kalibration des ADC-Input (in Volt) gegen mittleren ADC-Output. Aufgetragen ist der Gleichspannungswert am Analog Out des System Emulator Board (gemessen mit einem Multimeter), über dem Mittelwert des ADC-Output. Der geringe Fehler des abgelesenen Multimeterwerts ist aus darstellungsgründen nicht eingetragen. Der Fehler des ADC-Output wird als Standardabweichung der Trace geschätzt. Die Gleichspannung entstammte programmierten Floating Gates, deren Signal zum Analog Out des System Emulator Board getrieben wurde



Modell	$a + b\,x$		
Parameter	a =	2.23	± 1.02×10^{-3}
	b =	$-693.\times10^{-6}$	± $473.\times10^{-9}$
Mean Error	4.66×10^{-3}		

Modell	$a + b\,x + c\,x^2$		
Parameter	a =	2.26	± 2.24×10^{-3}
	b =	$-726.\times10^{-6}$	± 2.34×10^{-6}
	c =	8.09×10^{-9}	± $562.\times10^{-12}$
Mean Error	3.29×10^{-3}		

Modell	$a + b\,x + c\,x^2 + d\,x^3$		
Parameter	a =	2.31	± 5.92×10^{-3}
	b =	$-821.\times10^{-6}$	± 9.67×10^{-6}
	c =	57.4×10^{-9}	± 4.93×10^{-9}
	d =	-7.97×10^{-12}	± $793.\times10^{-15}$
Mean Error	2.7×10^{-3}		

Modell	$a + b\,x^c$		
Parameter	a =	2.29	± 4.28×10^{-3}
	b =	-1.05×10^{-3}	± 26.8×10^{-6}
	c =	$952.\times10^{-3}$	± 2.95×10^{-3}
Mean Error	3.08×10^{-3}		

Abbildung 3.6.: Zusammentragung der Modelluntersuchung. Dargestellt sind Fitresiduen und mittlerer Fehler sowie Fit-Parameter und deren Fehler

Auch hier stellen sich nach Betrachtung der Fitresiduen und des mittleren Fitting Fehlers feststellen, dass ein Polynom dritten Grades oder eine Allometrische Funktion die Kalibrationsdaten am besten beschreiben.

Vergleich der Kalibrationsergebnisse und Modellwahl

Die Ergebnisse beider Kalibrationen lassen den Schluss zu, dass ein quadratisches Modell bereits ausreicht, um eine Messungenauigkeit von 1 mV zu erreichen. Bei den folgenden

Algorithmen, die eine Umrechnungn des ADC Output in Volt benötigen, wird das polynomielle Modell dritten Grade verwendet, weil es den geringsten mittleren Fehler aufweist und gegenüber dem allometrischen Modell in Software schneller berechnet werden kann.

Das Overfitting, also das Mit-Fitting Statistischer Schwankungen, wurde aus Zeitgründen nicht untersucht.

Die Terminierung des ADC boards wurde mit einem Tischmutimeter auf 48.2Ω bestimmt. Der serielle Widerstand der verwendeten Spannungsquelle betrug 50Ω.

Da die Kalibration des ADC Output mit einer 50Ω serienterminierten Spannungsquelle. als auch in Schaltung mit dem System Emulator Board Ausgangsverstärker durchgeführt wurde und in beiden Kalibrationen des ADC Board und die Spannungsquelle in einer Spannungsteilerschaltung geschaltet wurde, kann die. reale Terminierung des Ausgangsverstärkers über einen Dreisatz bestimmt werden. Mit der Spannugsteilerformel gilt demnach für beide Schaltungen:

50Ωterminierte Spannungsquelle: $\frac{U_{50\Omega}}{R_{50\Omega}+R_{\mathrm{ADC}}} = \frac{U_{\mathrm{ADC}}}{R_{\mathrm{ADC}}}$

Analog Out des System Emulator Boards $\frac{U_{\mathrm{analogOut}}}{R_{\mathrm{analogOut}}+R_{\mathrm{ADC}}} = \frac{U_{\mathrm{ADC}}}{R_{\mathrm{ADC}}}$

und damit: $\frac{U_{\mathrm{analogOut}}}{U_{50\Omega}} = \frac{R_{\mathrm{analogOut}}+R_{\mathrm{ADC}}}{R_{50\Omega}+R_{\mathrm{ADC}}}$

Es wurden jeweils die polynomiellen Modelle dritten Grades verwendet um das Verhältnis der Spannungen zu bestimmen. Das Ergebnis ist in Abbildung 3.7 dargestellt.

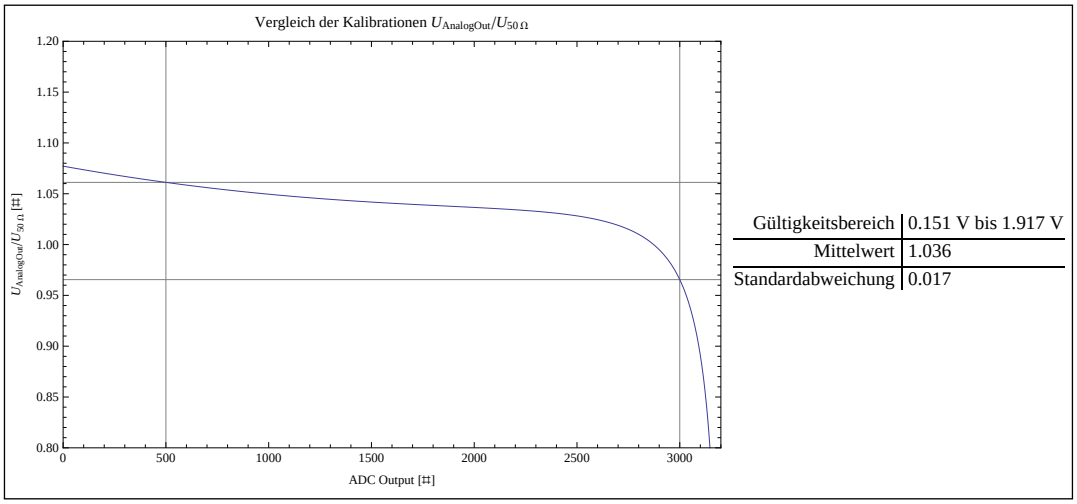


Abbildung 3.7.: Veerhältnis der Kalibrationsfunktionen für ausgewählte ADC Output. Das mittlere Verhältnis der Kalibrationsfunktionen wird durch Mittelung über dem ausgewählten Bereich zwischen 500 (entspricht 1.917V) und 3000 (entspricht 0.151V) gebildet.

Mit der letztgenannten Formel und den bestimmten Abschlusswiderstand des ADC Boards sowie des vor die Spannungsquelle vorgeschaltetem Wiederstand lässt sich der

Abschlusswiderstand zu 53.5Ω bestimmen. Bei den folgenden Anwendungen der Kalibrationsergebnisse der ersten Kalibration wird dieser Wert berücksichtigt.

Grenzen der Kalibration: Die Kalibration ist für Gleichspannungssignale anwendbar. Bei schnell alternierenden Signalen, sind der Gültigkeit der Kalibration Grenzen gesetzt, da bei höheren Signalfrequenzen Dämpfung auftritt. Eine Kalibration für alternierende Spannungssignale erfordert eine weitere zeitaufwendige Wechselspannungs Kalibration durch Aufnahme einer Transmissionscharakteristik für jeden ADC Input Channel.

3.3. Testen der neuromorphen Hardware

Neben der emulierten Membranspannung, können Spannungssignale einiger Schaltungen auf dem HICANN Chip zum Analog Out des System Emulator Board geleitet und durch ein *Analog Readout SystemEin digitales Oszilloskop oder ein ADC Board* aufgenommen werden. Dazu gehören unter anderem die Floting Gates, die als Speicher für analoge Modellparameter des AdEx Modells dienen (vergleiche dazu mit Kapitel "Analoge Parametrisierung durch Floating Gates").

Auf Grundlage dessen, lassen sich Tests implementieren, die die analoge Funktionalität der Hardware überprüfen, sodass Fehlbedienung und Defekte festgestellt werden können.

Ein Test, der die Genauigkeit der Floating-Gate-Programmierung aller Floating Gates überprüft, würde dann zum Beispiel bei zu hoher Programmierfrequenz, Unterschreiten der Versorgungsspannung des System Emulator Bords oder falsch eingestellter Programmier-Spannung fehlschlagen.

Im Folgenden wird die Implementierung eines Signifikanz-Tests und dessen Integration in ein *Google C++ Test Framework* erläutert.

3.3.1. Abschätzung der Auslesefehlers mit dem ADC Board

Das Testen der Floating Gates bedarf der Bestimmung von Spannungspegeln. Das ADC Board wird dazu verwendet, kontinuierliche Spannungssignale zu digitalisieren, aufzunehmen und deren Gleichspannungsanteil zu schätzen. Die Genauigkeit dieses Schätzwertes hängt maßgeblich von der Aufnahmedauer ab. Da die Ausführungsdauer des Floating-Gate-Tests unter anderem von der Dauer der Bestimmung eines Spannungspegels abhängt, muss die Aufnamedauer gegen die Schätzunsicherheit abgewogen werden.

Dazu wird eine lange Aufnahme eines Gleichspannungssignals, das durch eine Spannungsquelle erzeugt wurde, in kürzere gleiche Fenster bestimmter Zeitlänge partitioniert und der Spannungswert aus diesen Fenstern ermittelt. Die Standardabweichung dieser Werte ergibt den geschätzten Gleichspannungs-Auslesefehler zu einer Aufnahmezeitlänge.

Dieser Vorgang wurde für verschiedene Aufnahmezeiten durchgeführt, jedoch wurde keine externe Spannungsquelle angeschlossen, um nur das Rauschen des vorgeschalteten differentiellen Verstärkers im ADC Board und der Störungen auf dem ADC Board abzuschätzen. Das Ergebnis ist in Abbildung 3.8 dargestellt.

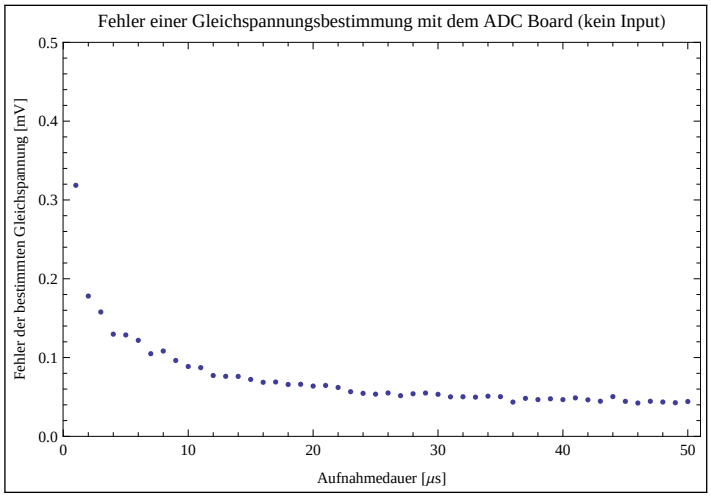


Abbildung 3.8.: Fehler einer Gleichspannungsbestimmung mit dem ADC Board als Funktion der Aufnahmedauer: Dargestellt ist der intrinische Fehler, der ohne externe Spannungs-Quelle auftritt. Für höhere Aufnahmezeiten konvergiert er gegen den Auflösungsfehler des ADC

Dann wurde ein Floating Gate auf einen Wert von 500 programmiert, dessen Gleichspannungssignal auf den Analog Out des System Emulator Board gelegt und das ADC Board an das System Emulator Board angeschlossen. Der Auslesefehler wurde auf gleiche Weise bestimmt. Das entsprechende Ergebnis ist in Abbildung 3.9 dargestellt. Dadurch wurden die Störungen des herausgeleiteten Signals eines Floating Gate abgeschätzt.

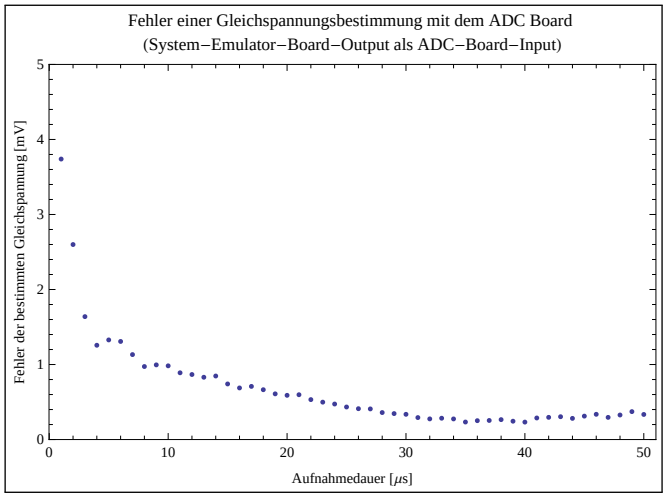


Abbildung 3.9.: Fehler einer Gleichspannungsbestimmung mit dem ADC Board als Funktion der Aufnahmedauer. Dargestellt ist der Fehler, der bei der Messung einer Externen Spannungsquelle auftritt. Für höhere Aufnahmezeiten konvergiert er gegen den Auflösungfehler des ADC

Bei einer Aufnahmedauer von 30 μs liegt der Bestimmungsfehler unter 1 mV. Diese Aufnahmedauer wird bei der Bestimmung der programmierten Floating Gate Spannung in einem Floating-Gate-Test verwendet.

3.3.2. Statistik der Floating Gate Programmierung und Signifikanz

Zur Festlegung des Toleranzbereiches für einen Signifikanz-Test bedarf es Wissen über die Statistik der Floating-Gate-Programmierung.

Die Programmierung der Floating Gates findet blockweise statt und ist derzeit ein zeitkritisches Element bei der Ausführungsdauer entsprechender Software Anwendungen. Die Ermittelung der Programmierstatistik jedes Floating Gates würde daher einen enormen Zeitaufwand darstellen. Zudem müssten viele Daten hinterlegt werden, was die Handhabung eines solchen Tests erschweren würde. Daher wird die Statistik der programmierten Floating Gate Spannungen zeilenweise ermittelt. Dies hat auch zum Vorteil, dass dieser Signifikanztest gegenüber wenigen defekten Floating Gates robust wäre und bei verschleißbedingtem Ausfall einiger Floating Gates der Systemzustand (richtige Versorgungsspannung, richtige Einstellungen der Floating Gate Programmierparameter und Ähnliches) des Hardware Setup immer noch geprüft werden kann.

Wie bereits im Kapitel "Analoge Parametrisierung durch Floating Gates" erwähnt, unterscheiden sich Spannungs-Parameter-Floating-Gates schaltungsbedingt von den Strom-Parameter-Floating-Gates in der Programmiergenauigkeit. Eine beispielhafte Verteilung der programmierten Spannungen innerhalb einer Zeile nach einem einmaligen Programmiervorgang auf den gleichen Programmierwert ist in Abbildung 3.10 dargestellt.

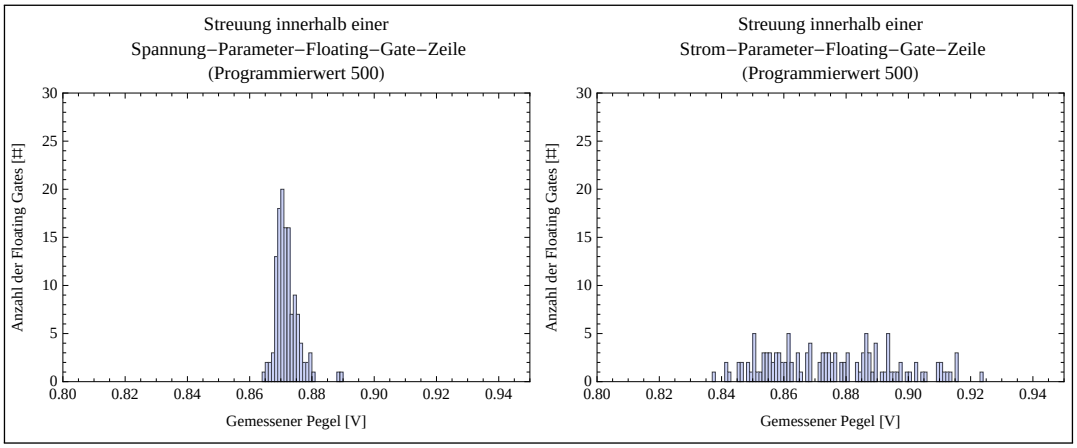


Abbildung 3.10.: Verteilung der programmierten Spannung innerhalb einer Spannungs-Parameter-Floating-Gate-Zeile und einer Strom-Parameter-Floating-Gate-Zeile. Gleichspannungsschätzung über 100μs

Der Programmierfehler der Floating Gates nimmt mit der Anzahl der Programmiervorgänge ab und mit der Differenz der Nominalspannung zur zu programmierenden Spannung zu. (weitere Programmiertartefakte werden ausführlich in [12] und [13] untersucht).

Um die Programmier-Ungenauigkeit zu schätzen, wurden die Floating Gates im Bereich von 0 bis 1023 programmiert, was einer Programmierung im Bereich 0 V bis 1.8 V entspricht, und die Verteilung der zeilenweise gemittelten Floating Gate Spannungen untersucht.

Um den Programmierfehler hierbei zu erhöhen, wurden die Abstände zwischen zwei aufeinander folgenden Programmier-Werten maximiert, indem die programmierten Werte geriffelt wurden (also 0, 510, 10, 520 ...). In mehreren Durchläufen wurde die geriffelte Programmierung von 0 bis 1010 mehrmals wiederholt und in die Statistik einbezogen. Die erhaltenen Zeilenmittelwerte zu einem Programmierwert aus mehreren Durchläufen wurden nach der Art des Floating Gates getrennt, gemittelt und es wurde die Standardabweichung bestimmt.

Das Ergebnis dieser Messung sortiert nach Spannungs-Parameter-Floating-Gates und Strom-Parameter-Floating-Gates ist in Abbildung 3.11 und Abbildung 3.12 dargestellt.

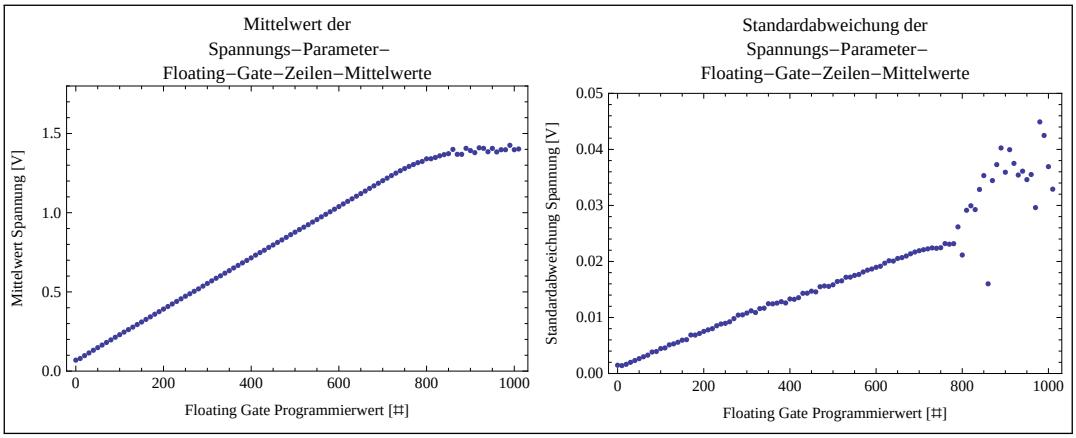


Abbildung 3.11.: Kalibrationsdaten für Spannungs-Parameter-Floating-Gates: Auftragung der programmierten Spannung und deren Fehler als Funktion des Programmierwerts

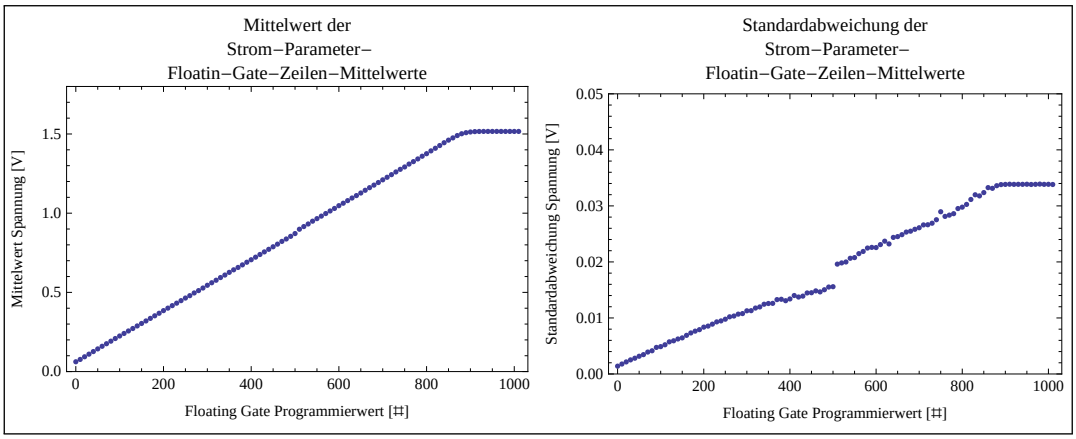


Abbildung 3.12.: Kalibrationsdaten für Strom-Parameter-Floating-Gates: Auftragung der programmierten Spannung und deren Fehler als Funktion des Programmierwerts

Während der Programmierung wird im Acceleration Step die Programmierspannung[1] ab einem bestimmten Wert stark erhöht, sodass bei größeren Differenzen zwischen Nominalspannung des Floating Gates und der zu programmierenden Spannung die Zielspannung überschossen wird. Bei dieser Messung ist dieser Effekt deutlich daran zu erkennen, dass die aufgenommene Programmierkalibration eine Unstetigkeit (siehe Abbildung 3.12) bei einem Programmierwert von ungefähr 500 aufweist, der allerdings bei mehrmaligem Programmieren kleiner ausfällt, weil die Floating Gates dann vorprogrammiert sind und im zweiten Programmiervorgang die Spannungsdifferenz kleiner ausfällt.

Um eine präzisere Programmierung der Floating Gates zu erreichen, empfiehlt es sich

[1] Die Spannung, mit der das Floating Gate programmiert wird, nicht die Spannung, auf die das Gate programmiert wird

also, den Programmierschritt mindestens 2 mal durchzuführen.

Unter der vereinfachenden Annahme einer Normalverteilung der Zeilenmittelwerte, kann das Toleranzniveau für einen Signifikanztest als 3σ (Standardabweichung) Bereich um den kalibrierten Wert gesetzt werden. Die oben bestimmte Stanardabweichung modelliert in diesem Fall zum einen die systematischen Abweichungen der aufgenommenen Spannungssignale, die aufgrund unterschiedlicher Leitungslängen und damit unterschiedlicher Signaldämpfung im HICANN Chip auftreten, zum anderen die Ungenauigkeit der Gleichspannungsbestimmung mit dem ADC Board und ferner den Fehler der Floating Gate Programmierung.

Es muss allerdings beachtet werden, dass die Standartabweichung unter Umständen unterschätzt wird und es einer größeren Datenmenge bedarf, um diese besser zu bestimmen.

Signifikanztest auf Grundlage dieser Statistik können nur bei Bedienungsfehlern fehlschlagen, die die Programmierung aller Floating Gates beeinflussen.

3.3.3. Test Implementierung

Der strukturelle Aufbau eines *Google-Tests* wurde bereits in Kapitel "Standardisiertes Testen im Google C++ Testing Framework" besprochen.

Ein bereits bestehendes *Google C++ Test Framework*, welches digitale Hardwaretests implementierte, wurde um einen analogen Test erweitert. Eine bereits definierte Methode, um das Hardware Setup in einen Ausgangszustand für einen Test zu versetzten, wurde auch in den analogen Test integriert.

Im ersten Schritt des Tests wird mithilfe der Methoden des erweiterten Hardware Abstraction Framework eine Verbindung zum ADC Board hergestellt und dessen Konfiguration vorgenommen. Es wird die Aufnahmezeit, das Triggeringverhalten und ein Input Channel eingestellt, der mit dem System Emulator Board verbunden ist.

Das Floating Gate Array wird dann auf einen Wert Programmiert. Iterativ wird das Spannungssignal eines ausgewählten Floating Gates auf den Ausgang des System Emulator Board gelegt, welches dann, nach softwareseitigem Triggern des ADC Boards, digitalisiert und aufgenommen wird. Für die Dauer der Aufnahme wird die Ausführung des Tests pausiert. Die Aufnahme des ADC Boards wird ausgelesen und in Volt umgerechnet. Es wird der Gleichspannungsanteil durch Mittelung bestimmt, was als Schätzwert der programmierten Floting Gate Spannung genommen wird. Für jedes Floating Gate der selben Zeile, wird dieser Vorgang wiederholt. Es wird dann der Mittelwert dieser Spannungen gebildet und in einem Signifikanztest gegen den Erwartungswert innerhalb der erwarteten Unsicherheit verglichen. Falls eine signifikante Abweichung festgestellt wird, schlägt der Test fehl. Ansonsten wird mit der nächsten Zeile fortgefahren. Dieser Vorgang wird für alle Zeilen eines Quadranten des Floating Gate Arrays und einige Programmierwerte innerhalb des linearen Bereiches der Floating Gate Programmierung wiederholt.

Das Signifikanzniveau wurde als 3σ Bereich festgelegt, wobei σ der Stanardabweichung des Mittelwerts der programmierten Floating Gate Zeilen-Mittelwert-Spannung entspricht.

Bei der Ausführung analoger Hardware Tests muss auch beachtet werden, dass die Hardware eine endliche Reaktionszeit hat, während derer diese nicht verstellt oder programmiert werden sollte. Dazu gehören die Dauer der Aufnahme des ADC Boards und die Umschlaltzeit der Kanäle auf dem HICANN Chip sowie die Einschwingungszeit der zugeschalteten Signale.

Die Umsetzung dieser Auswertung bedurfte der Einbeziehung der Kalibrationen des ADC Output und der Floating Gate Programmierung sowie der Implementierung der entsprechenden Konversionsfunktionen, die im Quellcode des Tests untergebracht sind. Dieser Test ist daher speziell nur auf einem ausgewählten Input Channel des verwendeten ADC Boards durchführbar. Durch Speicherung und zur Verfügung Stellung der Kalibrationsfunktionen aller verwendeten ADC Boards und derer Input Channels sowie der Implementierung der softwareseitigen Identifizierung eines ADC Boards sollte Portabilität erreicht werden.

Der Algorithmus ist denkbar simpel, demostriert aber, wie analoge Tests umgesetzt werden können.

Der implementierte Test wird erwartungsgemäß bei Schwankungen in der Spannungsversorgung, Software Fehlern und veränderten (schlechteren) Programmierparametern für die Floating Gate Programmierung fehlschlagen.

Dieser Test wurde in das *Google C++ Test Framework* integriert und wird von einer Automaisierungs-Anwendung innerhalb eines Zeitablaufplans periodisch durchgeführt.

3.4. Automatisierung

Die Aufgaben und Konfigurationsmöglichkeiten des Jenkins Servers wurden bereits in Kapitel "Automatisierung mit Jenkins-Anwendung" besprochen.

Das erweiterte Testframework wurde als Bestandteil einer Version des *Halbe* Projekts in ein *Git-Repository* hochgeladen.

Dann wurde die Jenkins-Anwendung für eine Automatisierung der Ausführung von Hardwaretest konfiguriert. Dazu wurde das *symap2ic* Metaprojekt als Ziel eines Build eingestellt und die entsprechenden Projektabhängigkeiten (unter anderen das *Halbe* Framework) angegeben, die zum Bauen des Projektes notwendig sind.

Die Durchführung des *Build* wurde durch ein Shell Script vorgegeben, welches Jenkins zu einem vorgegebenen Zeitpunkt nach dem Herunterladen der angegebenen Abhängigkeiten ausführt.

In diesem Script werden Systemvariablen zum Ausführen des *Waf-Utility* gesetzt, welches ebenfalls Bestandteil des *symap2ic* Metaprojektes ist. Es wird die Ausführund des *Waf-Utility* konfiguriert, fehlende Softwareabhängigkeiten durch das *Waf-Utility* festgestellt und heruntergeladen und der *Build* des Projektes durchgeführt.

Anschließend wird das *Google Test Program* parametrisiert durchgeführt. Dabei werden die Ip des Vertical Setup, die auszuführenden Tests und der Ort für die Speicherung des XML-Formattierten Testergebnisses angegeben.

Diese Testergebnisse werden dann nach der Durchführung dieser Routinen gespeichert und visualisiert. Dadurch kann die Funktionsfähigkeit der Hardware und die Kompilierbarkeit der Software automatisch überprüft werden.

4. Schlusswort und Ausblick

Die Aufnahme und Analyse des emulierten Membranspannungssignals konnte leider aus zeitlichen Gründen nicht durchgeführt werden. Zum einen kam es bei der Inbetriebnahme des Vertical Setup sowie der Einarbeitung in das Software und Hardware Framework zu Verzögerungen. Zum anderen bedarf es für die Emulation neuronaler Signale einer Kalibration der Modellparameter des emulierten AdEx Modells, die ebenfalls aus zeilichen Gründen nicht durchgeführt wurde.

Allerdings wurden alle Mechanismen, die für die getriggerte Aufnahme analoger Signale notwendig sind, verifiziert, sodass der Durchführung des noch ausstehenden nichts mehr im Wege steht.

Es wurde erfolgreich eine erweiterbare automatisierte analoge Testumgebung zur Validierung und Überwachung von Hardware und Software Frameworks integriert.

5. Appendix

A. Konfigurationsregister des Analog Readout Systems

Konfigurationsregister	Konfiguration
cfg0	Startaddresse für RAM Schreibvorgang
cfg1	Startaddresse für RAM Schreibvorgang
cfg2	Startaddresse für RAM Schreibvorgang
cfg3	Startaddresse für RAM Schreibvorgang
cfg4	Endaddresse für RAM Schreibvorgang
cfg5	Endaddresse für RAM Schreibvorgang
cfg6	Endaddresse für RAM Schreibvorgang
cfg7	Endaddresse für RAM Schreibvorgang
cfg8	Reset Konfiguration
cfg9	Trigger Konfiguration

Konfigurationsregister des Analog Readout Systems

B. Trigger Konfigurationsregister

BitNr.	Name	Bedeutung	Anmerkung
0	nicht belegt		
1	nicht belegt		
2	nicht belegt		
3	Trigger-Channel-Bit	Einstellen des Trigger-Channel	(0 oder 1)
4	Trigger-Enable-Bit	Einschalten des externen Triggers	(1==True)
5	Single-Mode-Bit	Einmaliger Trigger	(1==True)
6	Stop-Bit	Anhalten der Aufzeichnung	(1==True)
7	Start-Bit	Starten der Aufzeichnung	(1==True)

Konfigurationsbits für Aufnahmevorgang

C. Trigger Konfiguration

Trigger-Enable-Bit	**Single-Mode-Bit**	**Aufnahmeverhalten**
0	0	Standardzustand, keine Aufnahme
0	1	Trigger ist blockiert, keine Aufnahme möglich
1	0	Jedes Trigger Signal löst eine Aufname aus, sofern keine bereits läuft
1	1	nächstes Trigger Signal löst Aufnahme aus, sofern keine bereits gestartet wurde. Alle darauffolgenden Trigger lösen keine Aufnahme aus.

Bitweise Triggerkonfiguration

Literaturverzeichnis

[1] Brainscales 1st year report (internal document).

[2] Lecture notes - perception, sensing & instrumentation lab. http://psi.cse.tamu.edu/teaching/lecture_notes/, dez 2012.

[3] Adaptive exponential integrate-and-fire model. http://www.scholarpedia.org/article/Adaptive_exponential_integrate-and-fire_model, jan 2013.

[4] BrainScaleS. http://brainscales.kip.uni-heidelberg.de/index.html, jan 2013.

[5] Electronic Vision(s) Group. http://www.kip.uni-heidelberg.de/cms/groups/vision/home/, jan 2013.

[6] git - der einfache einstieg. http://rogerdudler.github.com/git-guide/index.de.html, jan 2013.

[7] Google c++ testing framework advancedguide. http://code.google.com/p/googletest/wiki/V1_6_AdvancedGuide, jan 2013.

[8] Google c++ testing framework primer. http://code.google.com/p/googletest/wiki/Primer, jan 2013.

[9] Waf tutorial. http://docs.waf.googlecode.com/git/apidocs_16/tutorial.html, jan 2013.

[10] Dan Husmann de Oliveira (Universität Heidelberg) und Holger Zoglauer (Universität Heidelberg) und Stefan Schiefer (Technische Universität Dresden) und Stephan Hartmann (Technische Universität Dresden). Facets wafer-scale integration. Internes Dokument.

[11] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience*. Massachusetts Institute of Technology, 2007.

[12] Alex Kononov. Testing of an analog neuromorphic network chip. Master's thesis, 2011.

[13] Millner Sebastian. *Development of a Multi-Compartment Neuron Model Emulation*. PhD thesis, 2012.

[14] John Ferguson Smart. *Jenkins The Definitive Guide*. O'Reilly Media, 2011.

Statement of Originality (Erklärung):

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, March 22, 2013

.....
(signature)