UNIVERSITY OF APPLIED SCIENCES

FACHHOCHSCHULE GIESSEN FRIEDBERG

Entwicklung eines Neuronen–Demonstrators für das Huber–Braun–Modell

Studienarbeit

Fachhochschule Gießen–Friedberg
Fachbereich Elektro- und Informationstechnik

Vorgelegt von

Alexander Bergen und Fabian Kring

Zeitraum:	01.12.2009 – 31.05.2010
Referent:	Prof. Dr.-Ing. Werner Bonath
Betreuer/Doktorand:	Dipl.-Ing. (FH) Marcel Beuler

Vorwort

Dieser Bericht beschreibt unser Projekt „Neuronen-Demonstrator für das Huber–Braun–Modell“ im Rahmen einer Studienarbeit bei Dipl.-Ing. (FH) Marcel Beuler. Die Studienarbeit wurde überwiegend im Labor für Mikroelektronik G115 entwickelt und getestet.

Im Zuge der Studienarbeit bestand unsere Aufgabe darin, eine Aufsteckplatine für das FPGA–Board zu entwickeln. Diese Aufsteckplatine soll die Erfassung der Umgebungstemperatur sowie die Vorgabe von Rauschparameter und Injektionsstrom ermöglichen. Unter anderem sollten die erfassten Werte auf einem Grafik–Display und der Membranpotentialverlauf auf einem TFT-Monitor angezeigt werden.

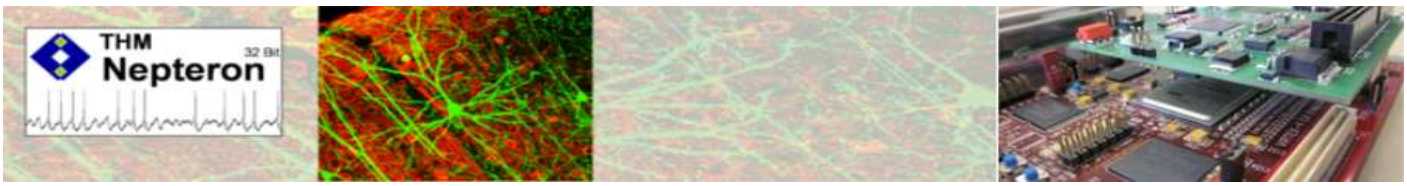
Die Studienarbeit wurde in die folgenden zwei Entwicklungsbereiche unterteilt:

- Hardwareentwicklung
- Softwareentwicklung

Um die Entwicklung der Hardware bzw. der Software zu realisieren, wurden zwei Tools verwendet: Altium Designer 6 und CodeVisionAVR. Mit dem Altium Designer 6 wurde der Schaltplan und das Layout erstellt. Mit dem CodeVisionAVR wurde der Source Code für den µController entwickelt.

Im diesen Bericht werden zunächst die verwendeten Baukomponenten erläutert. Danach wird auf die Planung und Umsetzung im Bereich Hardware, Layout und Software eingegangen und anschließend der Testlauf beschrieben.

Bedanken möchten wir uns bei unserem Betreuer Herrn Dipl.-Ing. (FH) Marcel Beuler, der uns den Einsatz in diesem Projekt ermöglicht und uns immer unterstützt hat, so dass die Arbeit interessant und zielgerichtet war. Außerdem bedanken wir uns bei Herrn Prof. Dr.-Ing. Werner Bonath für die Zusammenarbeit. Des Weiteren möchten wir uns bei Herrn Manfred Klein für jegliche technische Unterstützung bedanken.



Inhaltsverzeichnis

1 Einleitung

Auf der Hannover Messe und der „Straße der Experimente“ in Gießen sollte der FPGA-basierte Neuronen-Rechnerkern des Nepteron-Projekts vorgestellt werden. Von den 400 Neuronen nach dem Huber-Braun-Modell (nur Elektrorezeptor-Variante implementiert), die in Echtzeit berechnet werden können, wurde ein Neuron herausgegriffen und dessen Membranpotentialverlauf grafisch auf einem Oszilloskop mit angeschlossenem TFT-Bildschirm dargestellt. Für die Spike-Generierung sind insgesamt drei Parameter relevant:

- Injektionsstrom
- Rauschparameter
- Temperatur

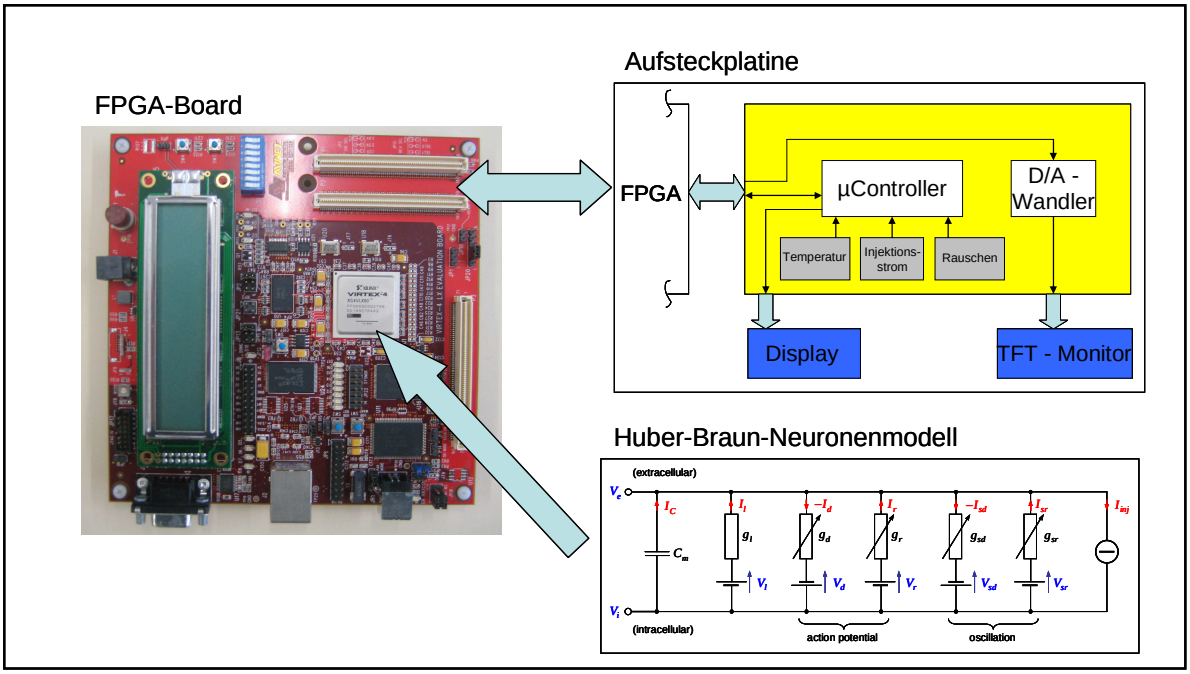


Abb.1: *Systemaufbau*

Für die individuelle Einstellung dieser Parameter zur Laufzeit des Systems dient die von uns entwickelte Aufsteckplatine für das FPGA-Board. Injektionsstrom und Rauschparameter können über Potentiometer variiert werden. Ein Mikrocontroller digitalisiert die eingelesenen Werte in gleichen Zeitabständen und wandelt sie in das 32-Bit-Gleitkommaformat um. Die Temperatur wird über einen ZACwire-Sensor erfasst und ebenfalls zum Controller gesendet, der schließlich alle drei Parameter auf ein Grafik-Display ausgibt und über eine asynchrone Schnittstelle zum FPGA überträgt.

2 Hardware

In diesem Abschnitt wird nur auf die relevanten Baukomponenten eingegangen, wie z.B. µController, D/A-Wandler und Connectoren.

2.1 Spannungsversorgung

Auf diesem Sheet wurden die verschiedenen Spannungen, +3.3V, +5V, +10V und +15V, erzeugt. Diese unterschiedlichen Spannungen sind auf der Platine verarbeitet worden, um die unterschiedlichen Bauteile mit Spannung zu versorgen. Die Elektronik wurde mit einem Brückengleichrichter geschützt.

Die Spannung von +3.3V für den µController von Atmel wurde durch einen LD1086-3.3 Spannungswandler realisiert. Am Vin-Eingang des LD1086-3.3 lagen +15V an. Der LD1086-3.3 wurde sowohl eingangs- als auch ausgangsseitig mit einem Kondensator von 10µF gestützt.

Die Spannung von +5V wurde für das Display benötigt und durch den µA7805 realisiert. Dieser Baustein wurde eingangsseitig mit einem Kondensator von 0.33µF gestützt. Ausgangsseitig erfolgte die Stützung des µA7805 durch einen Kondensator von 0.1µF.

Der ADR01 hatte eine Eingangsspannung von +15V die er zu +10V umwandelt. Diese Spannung war sehr wichtig, da der D/A-Wandler, welcher eines der zentralen Bauteile der Schaltung war, damit versorgt wurde. Die Eingangsseite des Spannungswandlers wurde mit zwei unterschiedlich großen Kondensatoren, 1µF und 0,1µF, gestützt.

2.2 µController: Atmel AT90CAN128

Das Herzstück der Platine ist der AT90CAN128. Die 64-Pins des Controllers wurden für die Ansteuerung des Displays, der Kommunikation zum FPGA und dem Einstellen der drei Stellgrößen benötigt. Die Spannungsversorgung des Controllers beträgt +3,3V. Diese Spannungsversorgung ist mit mehreren 100nF Kondensatoren stabilisiert worden. Der Takt von 8MHz erfolgte durch einen externen Quarz. Die 22pF-Resonanzkondensatoren sind nahe des Quarzes platziert worden. Die drei Stellgrößen wurden über drei 3-polige Steckverbinder mit der Platine verbunden. Der Injektionsstrom und der Rauschparameter sind durch Potentiometer nachgebildet worden. Um die Temperatur aufzunehmen ist ein Temperatursensor (LM335Z) in einem TO-92-Gehäuse ausgewählt worden. Der Sensor lieferte den Temperaturwert digital in 2-Byte-codierter Form zurück. Über den Atmel-Controller sind die drei Eingangsgrößen, der Rauschparameter, der Injektionsstrom und die Temperatur für den FPGA erfasst worden. Die Datenleitungen zum FPGA wurden terminiert, da es zu Beginn bei der Übertragung zu Problemen kam.

Abb.2: *µController*

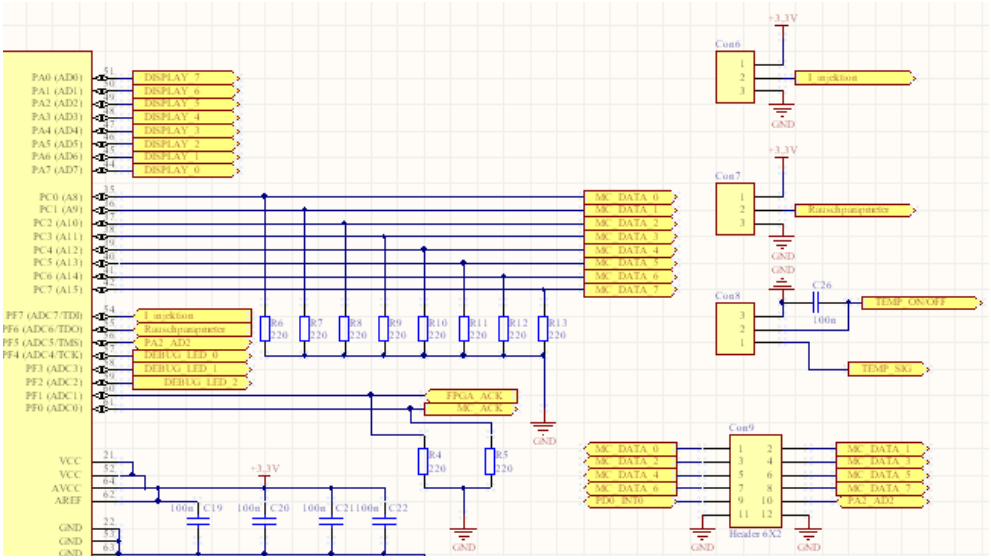


Abb.3: *Beschaltung µController*

2.3 **D/A Wandler**

Die Digital/Analog-Wandlung erfolgte durch einen AD5547 des Herstellers Analog Devices. Diese war notwendig, um die Signale bzw. Daten des FPGAs zu verarbeiten. Der D/A-Wandler wurde mit einer BNC-Buchse verbunden, womit die Membranpotentialverläufe auf einem Oszilloskop sowie auf einem Monitor dargestellt werden konnten. Die Versorgung des Bausteins erfolgte mit unterschiedlichen Betriebsspannungen, wobei Analog- und Digitalground getrennt ausgeführt waren. Zur Stützung des D/A-Wandlers kamen verschiedene Kondensatoren wie z.B. 100nF, 1µF und 3,3pF zum Einsatz. Die Anbindung der Kondensatoren wurde im später erstellten Layout möglichst kurz gehalten.

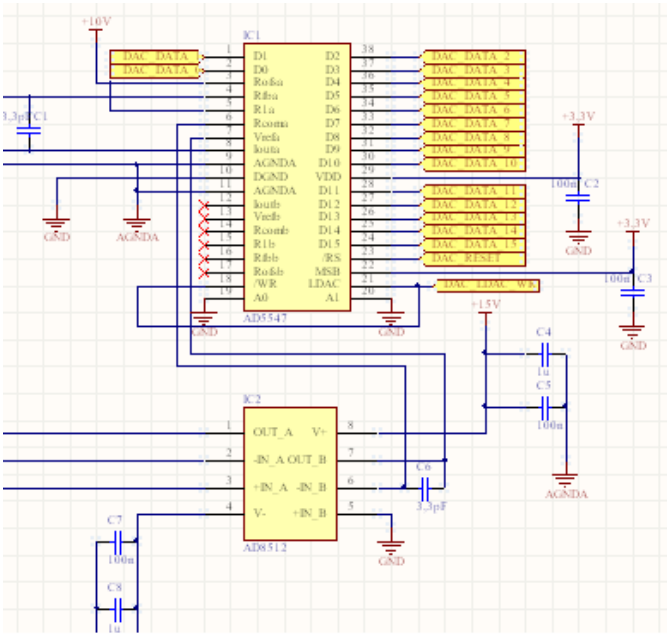


Abb.4: *Beschaltung des D /A – Wandler*

2.4 **Connector**

Auf der Schaltplanseite „Connector“ sind zwei 140-polige Steckverbinder, ein 40-poliger Wannenstecker und eine BNC-Buchse aufgeführt. Es wurde nur einer der beiden 140-poligen Stecker für die Datenübertragung des µControllers benötigt, der zweite diente ausschließlich der Stabilität der Aufsteckplatine. Die Stecker mussten verwendet werden, da auf dem FPGA-Board diese Steckverbinder bereits verbaut waren. Somit war das Problem der Stabilität der entwickelten Aufsteckplatine gelöst. Das Rastermaß dieser Steckverbinder betrug 2,54mm. Ihre Beschaffung war sehr problematisch, da dieser Stecker nicht bei den gewöhnlichen Versandhäusern zu erhalten war. Der 40-polige Wannenstecker wurde zur Verbindung zum Display verwendet. Hier wurden die acht Datenleitungen sowie Steuersignale für das Display angeschlossen.

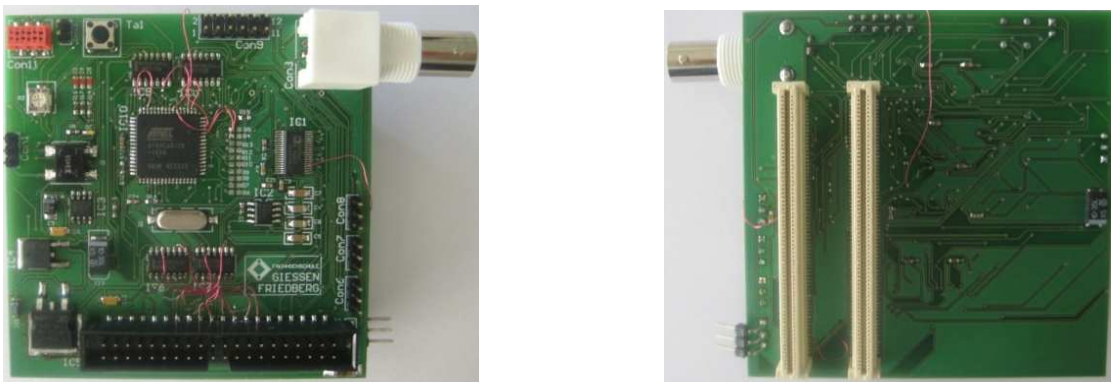


Abb.5: *Aufsteckplatine links von oben(a) rechts von unten(b)*

3 Layout

Im Rahmen der Hard- und Softwareentwicklung kamen die beiden Tools Altium Designer 6 und CodeVisionAVR zum Einsatz.

3.1 Altium Designer

Das Schaltplan-Tool des Altium Designer hatte bereits implementierte Standard Bibliotheken, aus denen die PCB-Decal für die zum Teil bekannten Bauteile verwendet wurden. Viele der benötigten Schaltsymbole mussten jedoch von Hand erstellt werden.

Der Schaltplan wurde in 4 Bereiche unterteilt:

- *Spannungsversorgung*
- *µController*
- *D/A-Wandlung*
- *Connector*

Die Seiten wurden über ein sogenanntes Mastersheet verbunden. Dies ermöglichte das Verbinden der verschiedenen Netze der 4 verschiedenen Bereiche.

3.1.1 Layout

Um das Layout zu designen, wurde der Project Wizard verwendet. Mit diesem Tool wurden die grundlegenden Vorgaben für das Design erstellt. Aus Kostengründen ist ein zweilagiges Layout festgelegt worden. Die Größe der Vias und Bohrungen sind minimal gewählt worden, da der Platz auf der Platine sehr begrenzt war. Die Breite der Leiterbahnen musste später angepasst werden, da der Strom durch die Leiterbahnen unterschiedlich groß war.

Folgende Leiterbahnbreiten kamen zum Einsatz:

- Datenleitungen => 0,2mm
- Stromversorgung Display => 0,6mm
- Stromversorgung IC´s => 0,3mm; 0,4mm; 0,5mm

Die unterschiedliche Breite der Leiterbahnen bei der Stromversorgung der IC´s liegt an dem unterschiedlichen Stromverbrauch der Bauteile.

3.1.2 Bauteilplatzierung

Die Spannungswandler wurden alle im linken unteren Bereich der Platine platziert, da durch sie große Wärme entsteht. Der µA7805 (IC5) musste auf eine große Groundfläche gelegt werden, um die entstehende Wärme besser abführen zu können. Die Kondensatoren, die zur Beschaltung der Spannungswandler notwendig waren, wurden nahe den ICs platziert. Die Größe der in der Spannungswandlung zum Einsatz kommenden Kondensatoren wurde den Datenblättern entnommen.

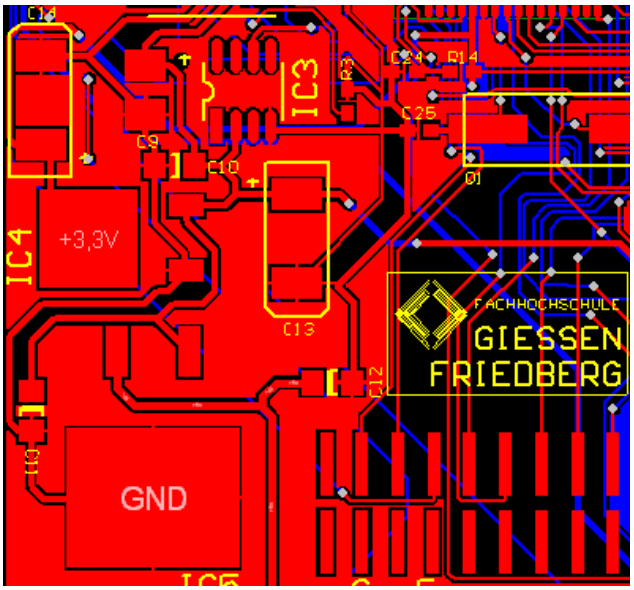


Abb.6: *Spannungswandler*

Das Herzstück der Platine, der µController AT90CAN128, sitzt in der Mitte der Platine. Von diesem Baustein aus wurden die Datenleitungen zum Display und zum FPGA geroutet. Die Kondensatoren zur Stabilisierung der Betriebsspannung wurden nahe des µControllers platziert. Dies ist von großer Wichtigkeit, da die Anbindung der Kondensatoren an den Baustein immer möglichst kurz sein muss. Somit wird die Induktivität der Leiterbahnen gering gehalten. Hier wurde der Bottom-Layer (blau) genutzt. Es sind 2 Kondensatoren auf der Unterseite des µControllers platziert, somit ist der Anbindungsweg sehr kurz. Optional hätte auch ein weitere Lage, eine Potentiallage, erstellt werden können. Dies wäre dann eine noch bessere aber auch sehr kostspielige Variante gewesen.

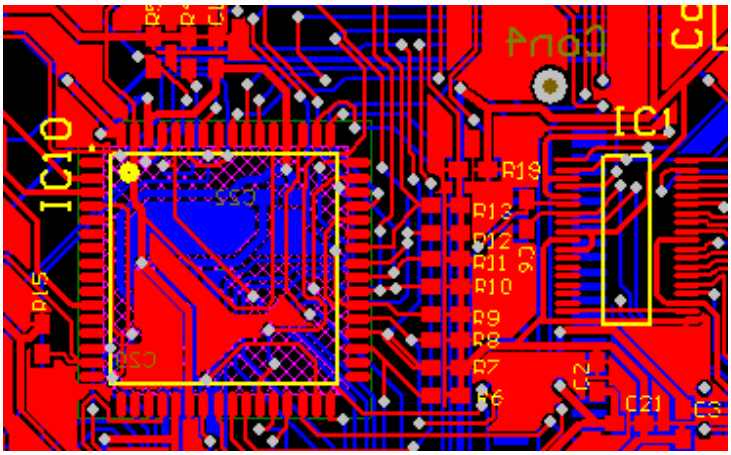


Abb.7: *µController und D/A – Wandler*

Der Quarz, der den Takt für den µController vorgibt, wurde nahe des µControllers platziert. Die zwei Resonanzkondensatoren wurden direkt neben den Pads des Quarzes platziert.

Der D/A-Wandler, Bild 7 rechts, war das kleinste und am schwersten zu routende Bauteil, da hier sehr oft Probleme mit den Design-Rules des Altium Designers auftraten. Das verwendete TSSOP-Gehäuse (Thin Shrink Small Outline Package) war durch seine sehr kleine Dimensionierung eine Herausforderung, und es erforderte viel Arbeit, bis die Anbindung des D/A-Wandlers den Design-Rules entsprach. Auch hier wurde darauf geachtet, dass die zum D/A-Wandler gehörigen Kondensatoren keine langen Anbindungswege hatten.

Die Steckkontakte, die auf der Platine verwendet wurden, konnten aus Platzgründen nicht auf einer Seite der Platine platziert werden. Dies ist aus EMV-Gesichtspunkten nicht sehr gut, da die Steckkontakte viele Antennen am Rand der Platine bilden. Die einkoppelnden Störsignale könnten das Schaltverhalten beeinträchtigen. Da die Größe der Platine aus Gründen der Stabilität begrenzt war, konnte hier keine andere Lösung gefunden werden.

4 Software

4.1 CodeVisionAVR in Programmiersprache C

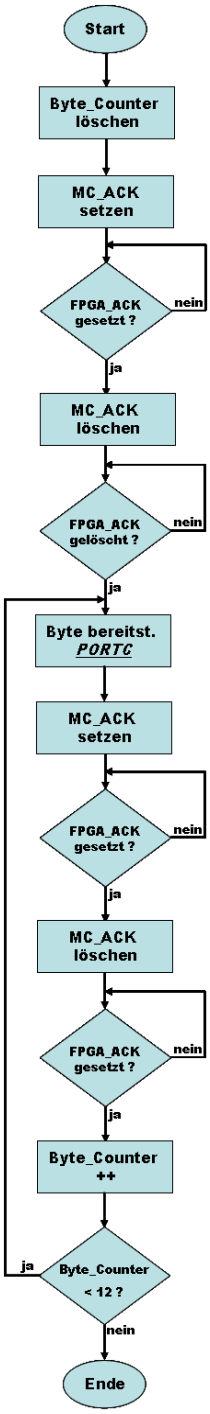
Für die Programmierung des µControllers wurde die vom Betreuer empfohlene Entwicklungsplattform CodeVisionAVR verwendet. Da der Betreuer sich sehr gut mit der Software auskannte, fiel die Einarbeitung sehr leicht.

In diesem Abschnitt wird nur auf die relevanten Quellcodeauszüge eingegangen. Um diese nachvollziehen zu können, wurde für jeden Quellcodeauszug ein Ablaufdiagram erstellt.

4.1.1 Kommunikation zwischen FPGA und µController

Hierbei wird das bekannte Handshake-Verfahren angewandt. Rechts ist das Ablaufdiagramm und links der dazugehörige Quellcode dargestellt.

```
void PARAMETER_TRANSFER(char PARAMETER[])
{
    int Byte_Counter = 0;
    MC_ACK = 1;

    while(FPGA_ACK == 0){};
    MC_ACK = 0;
    while(FPGA_ACK == 1){};
    for(Byte_Counter = 0; Byte_Counter < 12; Byte_Counter++)
    {
        MC_DATA = PARAMETER[Byte_Counter];
        MC_ACK = 1;
        while(FPGA_ACK == 0){};
        MC_ACK = 0;
        while(FPGA_ACK == 1){};
    };
    MC_DATA = 0x00;
    return;
}
```



4.1.2 A / D Wandler für Potentiometer

Das Potentiometer ist ein veränderbarere Widerstand (Abb.8). Er besteht aus einem elektrischen nicht leitenden Träger, auf dem ein Widerstandsmaterial aufgebracht ist, zwei Anschlüssen an den beiden Enden und einem beweglichen Gleitkontakt, der den festen Gesamtwiderstand elektrisch in zwei Teilwiderstände aufteilt. Dort findet auch der Abgriff der Teilspannung statt.

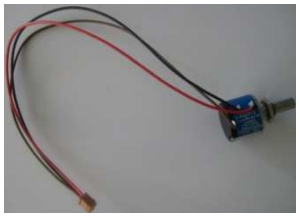


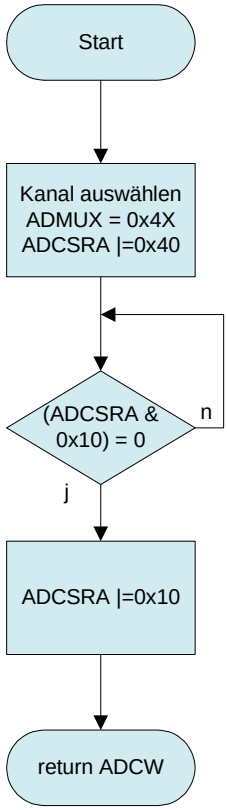
Abb.8: *Potentiometer*

Um den im µController integrierten A/D-Wandler nutzen zu können, muss der A/D–Wandler zuerst initialisiert bzw. aktiviert werden.

```
// ADC initialization
// ADC Clock frequency: 125,000 kHz
// ADC Voltage Reference: AREF pin  ADC High Speed Mode: Off
// Digital input buffers on ADC0: On, ADC1: On, ADC2: On, ADC3: On
// ADC4: On, ADC5: On, ADC6: On, ADC7: On
DIDR0=0x00;
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x86;
ADCSRB&=0x7F;
```

Der Injektionsstrom und der Rauschparameter werden nach dem gleichen Prinzip ausgelesen. Der einzige Unterschied ist die Auswahl des Kanals. Der Injektionsstrom hat den Kanal PF7(0x47) und der Rauschparameter den Kanal PF6(0x46). Rechts ist das Ablaufdiagramm und links der Quellcode der beiden Funktionen dargestellt.

```
unsigned int get_I_inj()
{
    // Injektionsstrom einlesen (Kanal: PF7)
    ADMUX = 0x47;  // Kanal PF7 ist ausgewählt
    ADCSRA |= 0x40;
    while((ADCSRA & 0x10) == 0);
    ADCSRA |= 0x10;
    return ADCW;
}
unsigned int get_nD()
{
    // Rauschparameter einlesen (Kanal: PF6)
    ADMUX = 0x46; // Kanal PF6 ist ausgewählt
    ADCSRA |= 0x40;
    while((ADCSRA & 0x10) == 0);
    ADCSRA |= 0x10;
    return ADCW;
}
```



4.1.3 Temperatursensor TSic mit ZACwire

Abb.9: *ZACwire-Sensor*

Der hier verwendete TSic–Temperatursensor (Abb.9) gibt seine Temperaturmessdaten automatisch in einem festen Intervall aus. Daher muss der µController mit der Abtastung abwarten, bis die nächsten Messdaten rausgeschickt werden. Zur Übertragung wird das ZACwire-Protokoll benutzt. Es handelt sich um eine einfache zwei-Byte-Übertragung (Abb.10). Diese zwei Byte repräsentieren den digital gewandelten Temperaturwert. Es gibt vier verschiedene Bitcodierungen, die in einem festen Zeitfenster von 125µs dargestellt werden, siehe Abb.11.

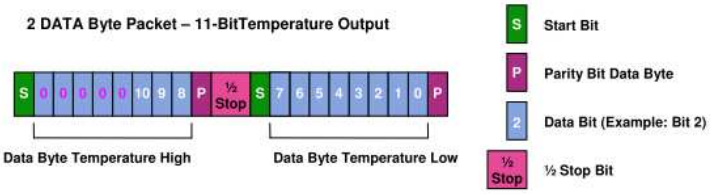


Abb.10: *Zwei Byte Übertragung*

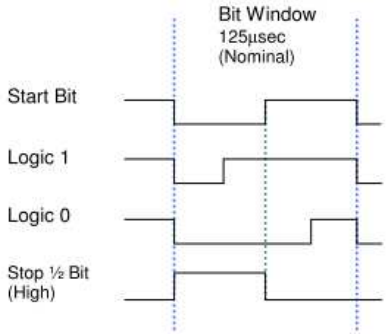


Abb.11: *Bit Encoder*

Da der TSic–Sensor einen sehr geringen Stromverbrauch hat, kann dieser direkt an den µController (siehe Abb.12) angeschlossen werden. Dadurch lässt sich der Sensor bei Bedarf einschalten und nach einem Delay für die eigentliche Temperaturmessung deren 2-Byte-Wert über den µController empfangen.

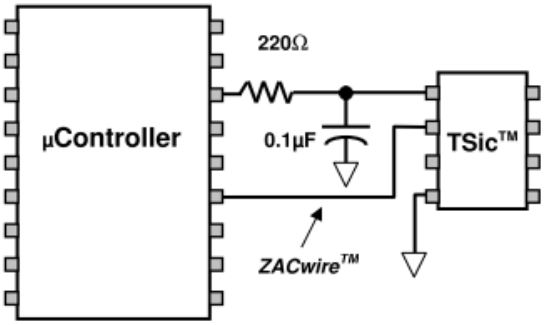
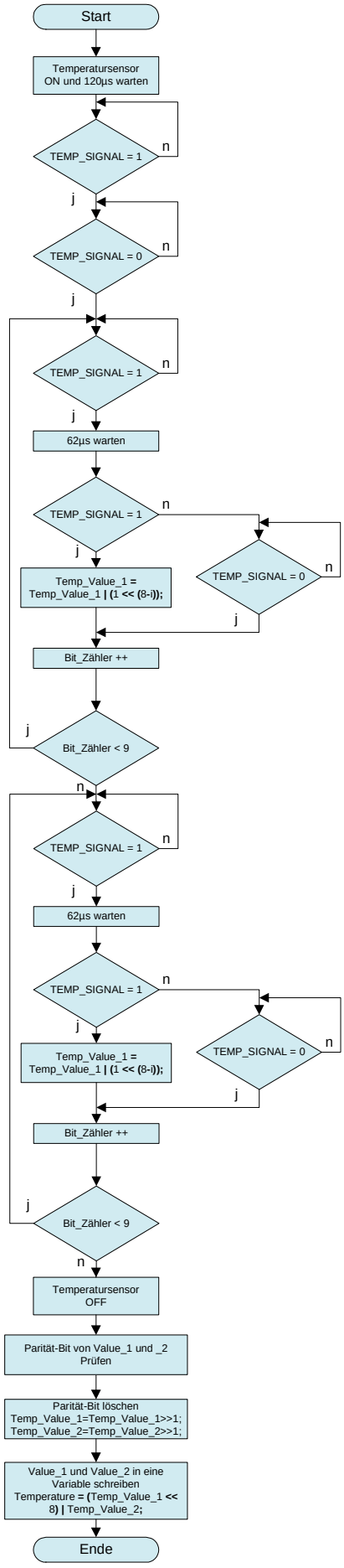


Abb.12: *Beschaltung des TSic mit dem µController*

Die Abtastung und Verarbeitung der einzelnen Bits wird sowohl im nachfolgenden Ablaufdiagramm als auch im Quellcode dargestellt und dokumentiert.

```
unsigned char get_temp_value(unsigned int *Temp_Value)
{
    char Parity;
    unsigned int Temp_Value_1 = 0, Temp_Value_2 = 0;
    unsigned int Temperature;
    int i;

    TEMP_POWER_ON;  // Temperatursensor einschalten
    delay_us(120);

    // 1. Startbit
    while(TEMP_SIGNAL == 1);  // fallende Flanke
    while(TEMP_SIGNAL == 0);  // steigende Flanke

    // 1. Datenbyte einlesen
    // 8 Datenbits und ein Paritaetbit
    for (i = 0; i < 9; i++)
    {
        while(TEMP_SIGNAL == 1);  // auf fallende
                                  // Flanke warten
        delay_us(62);

        if (TEMP_SIGNAL == 1)
        {
          Temp_Value_1 = Temp_Value_1 | (1 << (8-i));
        }
        else
        {
            while(TEMP_SIGNAL == 0);  // warten, bis
        Signalleitung wieder auf H-Pegel
        }
    }

    // 1/2 Stop-Bit bleibt auf H-Pegel

    // 2. Startbit
    while(TEMP_SIGNAL == 1);  // fallende Flanke
    while(TEMP_SIGNAL == 0);  // steigende Flanke


    // 2. Datenbyte einlesen
```



```
    // 8 Datenbits und ein Paritaetbit
    for (i = 0; i < 9; i++)
    {
        while(TEMP_SIGNAL == 1);  // auf fallende Flanke warten
        delay_us(62);

        if (TEMP_SIGNAL == 1)
        {
            Temp_Value_2 = Temp_Value_2 | (1 << (8-i));
        }
        else
        {
            while(TEMP_SIGNAL == 0);   // warten, bis Signalleitung wieder
auf H-Pegel
        }
    }

    TEMP_POWER_OFF;  // Temperatursensor ausschalten

TEMP_// Paritaet fuer Byte 1 pruefen (gerade Paritaet)

    Temp_Value_1 = Temp_Value_1 >> 1;  // Paritaet-Bit loeschen
    Temp_Value_2 = Temp_Value_2 >> 1;  // Paritaet-Bit loeschen
    Temperature = (Temp_Value_1 << 8) | Temp_Value_2;
    *Temp_Value = Temperature;
    return TRUE;  // Paritaet ist OK
}
```

4.1.4 Grafikdisplay → FH-Logo

Das LCD-Grafikdisplay (Abb.13) hat 240x128 Pixel und verfügt über einen T6963C-Controller mit integriertem komplettem und selbstdefinierbarem Zeichensatz. Die Kommunikation geschieht über einen 8-Bit-Datenbus. Text und Grafik können gleichzeitig auf dem Display dargestellt werden. Das Grafikdisplay wird mit 5 V versorgt und hat einen Stromverbrauch von 50mA bei einer Betriebstemperatur von -20 bis +70℃. Die LED–Hintergrundbeleuchtung hat einen Stromverbrauch von max. 135mA, zuzüglich der 50mA ergibt dies einen Gesamtstromverbrauch von 185mA.

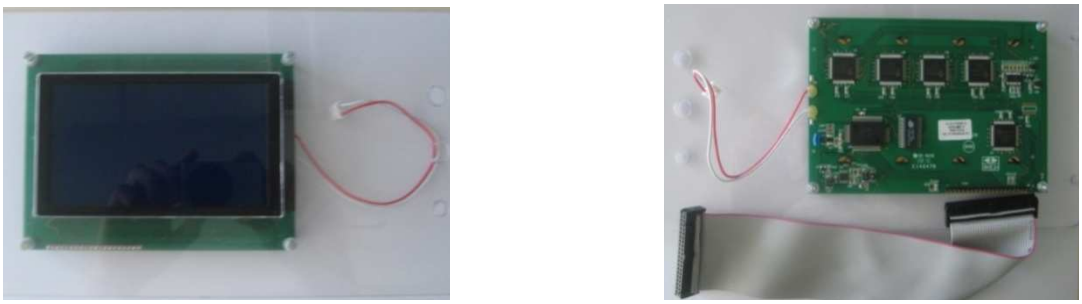


Abb.13: *Grafikdisplay links(a) von oben rechts(b) von unten mit Datenbusleitun*

Die Programmierung des LCD–Grafikdisplays wurde in drei übergeordnete Funktionsprototypen unterteilt.

1. Die Initialisierung wurde wie im Datenblatt empfohlen durchgeführt.

```
void lcd_init();        // Initialisierung des LCD – Grafikdisplays
```

2. Nach der Initialisierung wird das FH-Logo als erstes auf dem Display in der festgelegten Position und die unten dargestellte graue Schrift angezeigt.

```
void display_grundeinstellung(void); // Komplette Beschriftung und
{                                    //FH – Logo
        FH_logo();
        lcd_print(5,1,"NEURONS DEMONSTRATOR");
        lcd_print(0,8,"Injection Current:");
        lcd_print(0,10,"Noise Parameter:");
        lcd_print(0,12,"Temperature:");
        lcd_print(4,15,"<University of applied sciences>");
}
```

3. Die erfassten Werte der beiden A/D–Wandler und des Temperatursensors werden jeweils über die nachfolgenden drei Funktionen so umgewandelt, dass das Display die Werte richtig anzeigen kann.

```
void Display_I_inj(float I_inj); //Übergabe der drei Parameter
void Display_nD(float nD_display); //die dann zum LCD gesendet werden
char Display_Temp(float f_Temp_value, char Temp_Index_old);
```

Der komplette Quellcode ist im Anhang beigefügt.

5 Testlauf

Für den Testlauf des Aufsteckboards wurde über die BNC-Buchse ein Oszilloskop angeschlossen. Durch das Oszilloskop ließen sich die Spikes, welche die Neuronenaktivität wiedergaben, darstellen. Während dieser Testphase gab es Probleme mit dem Display. Die nach einigem Suchen gefundene Fehlerquelle waren die Pegelwandler, welche nur unidirektional senden konnten und ursprünglich die 3,3V-Spannungspegel des Controllers auf die 5V des Displays anheben sollten. Somit konnte das vom Display an den µController gesendete Acknowledge (ACK) nicht empfangen werden. Nachdem die Pegelwandler entfernt wurden, konnten auf dem Display auch die drei Stellgrößen, der Injektionsstrom, der Rauschparameter und die Temperatur dargestellt werden. Die Temperatur wurde durch ein Kältespray verringert, somit war zu erkennen, dass die Aktivität der Neuronen sank. Die Änderung des Rauschparameters sowie des Injektionsstroms konnten durch die angeschlossenen Potentiometer durchgeführt werden. Bei näherem Betrachten war hier zu erkennen, dass die erzeugten Signale mit den simulierten Signalen aus MatLab übereinstimmten. Zuletzt musste noch das Gehäuse für die anstehende Hannover Messe gebaut werden. Hier wurden für die Front Plexiglasscheiben verwendet. Die Verarbeitung des Plexiglases war schwierig, da es sehr schnell brechen konnte. Nachdem die Fertigstellung der Vorder- und Rückwand erfolgt war, konnten das Display und das FPGA-Board mit dem entwickelten Aufsteck-Board eingebaut werden.

Abb. 14 zeigt schließlich den fertig aufgebauten Neuronen-Demonstrator im Gehäuse mit angeschlossenem Oszilloskop und TFT-Bildschirm.

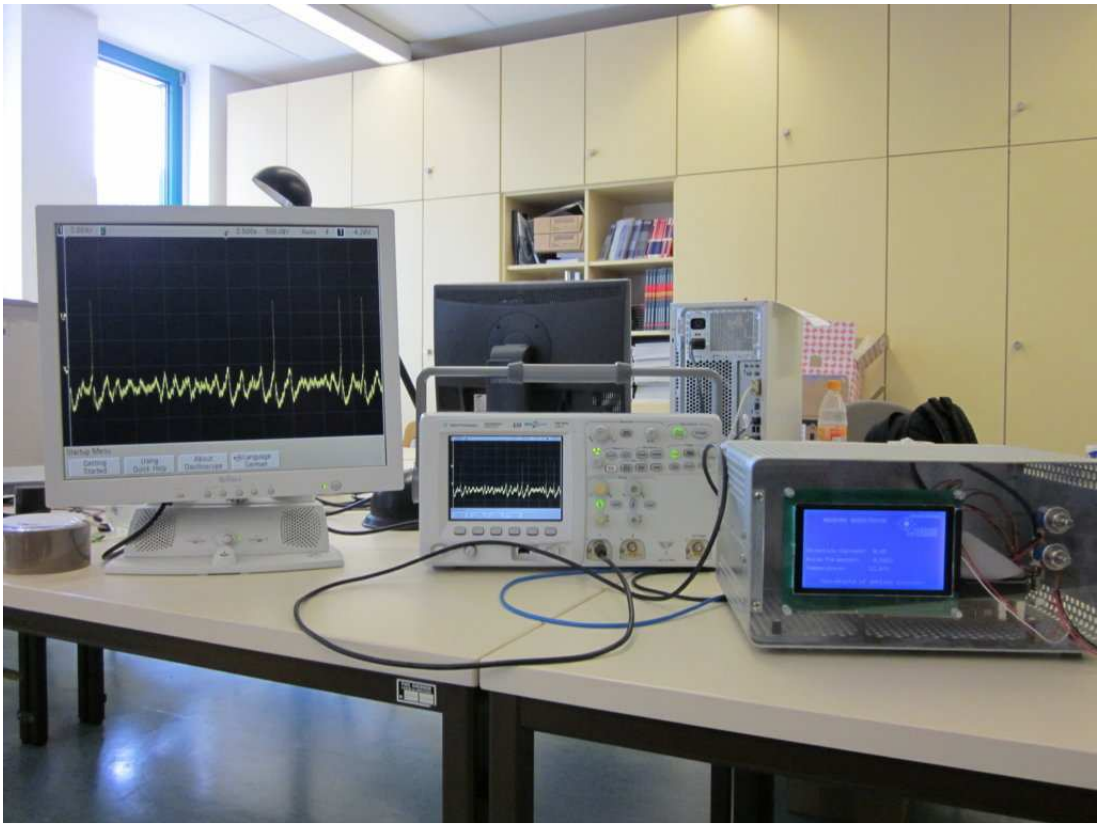


Abb.14: *Neuronen-Demonstrator in Aktion*

6 Zusammenfassung und Fazit

Die Studienarbeit war in der uns zur Verfügung stehenden Zeit sehr umfangreich. Durch die begrenzte Zeit der bevorstehenden Hannover Messe erfolgte die Einarbeitung in Altium-Designer und CodeVisionAVR unter enormem Zeitdruck. Die Entwicklung der Hard- und Software fand parallel statt. Der stetige Zeitdruck ermöglichte es leider nicht, manche Gebiete der Studienarbeit noch weiter zu vertiefen.

Die Studienarbeit hat sehr viel Spaß gemacht, ebenso die Zusammenarbeit mit Herrn Beuler. Die Studienarbeit war sehr lehrreich und hat unsere Kenntnisse in der Entwicklung vertieft.

7 Quellverzeichnis

www.thm-nepteron.de/
http://www.atmel.com
http://www.mikrocontroller.net
http://www.elektor.de
http://www.reichelt.de
http://www.farnell.de
http://www.computerkabelversand.de/