**Review:**

Am Beginn unserer Präsentation fassen wir kurz unser Projekt bis zur Zwischenpräsentation zusammen.

Bis dahin hatten wir unseren gewählten Chip, den ATmega 328p, auf einem Breadboard zum Laufen gebracht. Unser Code haben wir dabei ganz in C geschrieben, also ohne Arduino Framework in der Entwicklungsumgebung von Atmel Studio. Zum Programmieren unseres AVR-Chips verwenden wir den AVR-Programmer v2 von Pololu. Wir hatten bis zu diesem Zeitpunkt nur einen Temperatur- und Luftqualitätssensor verbaut den DHT11 und wir schafften eine Kommunikation mit unserem Übertragungsmodul, allerdings nur zwischen zwei CC1101 mit identischen Librarys auf identischen Chips. Zusätzlich haben wir begonnen mit dem Programm EAGLE zu arbeiten und schlossen dabei die ersten Schritte in Richtung Schaltplan ab.

Die Grundlagen zum Thema hatten wir uns bis dahin erarbeitet nun ging es zu den nächsten Schritten.

**Wetterstation 2.0**

Nachdem wir zum Zeitpunkt der Zwischenpräsentation erst einen Temperatur- und Luftfeuchtigkeitssensor hatten, machten wir uns Gedanken, mit welchen Sensoren wir unser Board erweitern können. Dazu stellten wir uns als erstes die Frage, ob wir unser Board im Freien oder im Haus nutzen möchten. Für draußen hatten wir Ideen wie Regensensor, Windstärkenmesser oder Sonnensensor, aber das war nicht Besonderes. Wir wollten etwas Ausgefalleneres. Als erstes kamen wir auf die Idee, einen Seismographen zum Messen von Erdbeben zu bauen. Aber da wussten wir nicht, ob das umsetzbar ist und haben uns deshalb nicht mehr damit beschäftigt. Während wir auf der Suche nach etwas Ausgefallenem waren, stießen wir auf ein Tutorial von Mightyohm, wo Schritt für Schritt erklärt wird, wie man einen Geigerzähler zum Messen der radioaktiven Strahlung bauen kann. Außerdem fanden wir noch einen Luftqualitätssensor, der giftige Gase in der Luft messen kann.

**Luftqualitätssensor:**

Der MQ 135 ist ein einfach zu nutzender Sensor zum Messen der Konzentration von verschiedenen Giftgasen wie Benzol, Alkohol, Rauch sowie Verunreinigungen in der Luft. Der MQ-135 misst eine Gaskonzentration von 10 bis 1000ppm (Parts per Million), das bedeutet er misst Gasanteile in der Luft schon ab 0,001 % und ist ideal zum Erkennen eines Gaslecks oder zur Überwachung der Luftqualität. Die Sensoren der MQ-Serie benutzen ein kleines Heizelement mit einem elektronisch-chemischen Sensor. Sie sind empfindlich gegenüber verschiedensten Gasen und eignen sich zur Verwendung in Räumen. Der Sensor hat eine hohe Empfindlichkeit und schnelle Reaktionszeit, benötigt allerdings einige Minuten bis er genaue Messwerte ausgibt, da der Sensor sich erst aufheizen muss. Das wirkt sich negativ auf den Stromverbrauch aus, aber das haben wir leider erst spät erfahren und konnten den Sensor nicht mehr austauschen, da Fabian dabei war, das PCB zu zeichnen und eine Neubestellung zu lange gedauert hätte. Die Messwerte des Sensors werden als analoger Wert ausgegeben, welche mit analogRead ausgelesen werden können.

**Geiger-Zähler:**

Geigerzähler Idee

Kommen wir nun zum Geigerzähler. Da wir ja eigentlich was Ausgefalleneres gesucht haben für unsere Wetterstation, sind wir auf die Suche nach weiteren Sensoren gegangen. Nach ein paar Tagen sind wir auf ein Tutorial gestoßen in dem ein portabler Geigerzähler gebaut wird. Wir haben uns die Materialiste angeschaut was für Teile verbaut sind und haben uns dann die Sachen bestellt zum Ausprobieren. Dieser Geigerzähler ist so aufgebaut, dass immer wenn radioaktive Strahlung gemessen wird, wurde von einem aktiven Buzzer ein „Piep“-Geräusch von sich geben und eine LED leuchtete kurz. Wie man es von Geigenzähler so kennt. Bei unserem Geigerzähler haben wir dass das Piepen und die LED weg gelassen das es wahrscheinlich auf Dauer ziemlich stört.

Dann nach ein wenig Zeit haben wir das Ganze auf einem Breadboard aufgebaut. Nach ein paar falsch eingesteckten Steckern und ein wenig Kaffee hat es geklappt, aber dann kam schon eigentlich unser Hauptproblem. Der Geigerzähler von diesem Tutorial hat einen AVR ATtiny Micro-Controller verbaut. Wir aber haben einen AVR ATmega Micro-Controller.

Eckdaten:

Das Herzstück des Geigerzählers ist die Zählröhre die man hier gut sehen kann. Diese Zählröhre ist eine Geiger-Müller Zählröhre STS-5 welche im Zeitraum 1960 bis 1970 in Russland gebaut wurden. Diese Röhre haben wir leicht über Ebay für ca. 15€ kaufen können. Diese Röhre erkennt Gamma und Beta Strahlung (woher?). Diese Strahlung kommt nicht nur von Kernreaktoren sondern kommt auch natürlich aus dem Weltall oder im Boden durch Gestein vor.

Die Zählerröhre kann bis zu 27 Pulse in der Sekunde erkennen und arbeitet mit ca. 360 bis 440 Volt.

Leider konnten wir vom Datasheet nicht viel herausfinden für die Zählröhre da man das Datasheet nur auf Russisch findet und keiner von uns zufällig Russisch kann.

Datenauswurf  
Radioaktive Strahlung wird in der Maßeinheit Sievert gemessen. Zwischen 0,1 und 0,2 Mikrosievert pro Stunde ist die durchschnittliche Grundstrahlung in Deutschland. Dieser Wert ist unterschiedlich je nach Beschaffenheit des Bodens an dem jeweiligen Ort. Beim bayrischen Landesamt für Umwelt kann man die aktuelle Strahlung verschiedener Regionen nachschauen. Und unser Sensor erhält ca. die gleichen Werte, so um die 0,1 Mikrosievert. Ab 2,5 Sievert innerhalb kurzer Zeit kommt es zu medizinischen Problemen. Das ist ein sehr hoher Wert, der nur bei Strahlungsunfällen erreicht wird.

Funktionsweise Zählröhre  
Hier sieht man die Zählröhre von innen. Diese Röhre ist innendrinnen mit zwei verschiedenen Gasen gefüllt. Einmal Auslösegas und ein Löschgas. Dann in der Mitte ein sehr dünner Zähldraht und an der äußeren Wand jeweils ein Glimmerfenster welche die Strahlung hinein lässt.

Die Zählrohrwand wird mit dem Minus und Plus der Quelle angeschlossen und dazwischen liegt ein Widerstand. Im Inneren des Zählrohrs herrscht ein elektrisches Feld um das Zähldraht. Gelangt radioaktive Strahlung in das Zählrohr, so wird das Auslösegas ionisiert. Aufgrund des elektrischen Feldes werden Elektronen in zur Außenwand gezogen und bei diesem Vorgang entladet sich die Zählröhre. Um diese Entladung zu messen greift man die Spannung am Widerstand R ab und verstärkt diese mit einem Verstärker und zählt die Spannungsimpulse. Damit das Zählrohr für eine neu radioaktive Strahlung zählbereit ist, muss die Entladung zum Erlöschen gebracht werden und dies geschieht mit dem enthaltenen Löschgas.

**ATtiny2313 auf unserem ATmega 328:**

Die größten Änderungen, die vorgenommen werden mussten, war der Code für Timer und Interrupts, die auf den ATtiny Chips anders funktionieren wie auf den ATmega Chips. Verschiedenen Register, die bei dem ATtiny verwendet wurden, gibt es auf den ATmega Chips nicht und mussten deshalb anpasst werden. Zusätzlich haben wir den Buzzer und das LED die bei dem Bauplan von „mightyohm“ verwendet wurden weckgelassen. Ebenfalls wurde ein Button entfernt der beim Drücken den Buzzer muted. Große Teile des Codes konnten wir mit Arduino-Code austauschen, so wie auch den Code für die serielle Schnittstelle. So wurde das Ganze kürzer und übersichtlicher.

In einem letzten Schritt wurde der Code des Geiger-Zählers, dann noch mit dem bisherigen Code zusammengefügt.

Die schwierigsten Aufgaben beim Anpassen des Codes auf unseren Chip war allerdings das Ändern von Timer- und Interrupt-Einstellungen.

Durch das Entfernen des Buttons fiel nicht nur ein Interrupt Ereignis, sondern auch ein weiterer Timer weck. Wir verwenden somit nur mehr ein Interrupt, nämlich beim Erhalt von Signalen von der Röhre und den dazugehörigen Timer.

**Back to the roots**

Nach der Zwischenpräsentation haben wir uns entschieden, von C wieder zurück zu Arduino zu wechseln, weil uns das vieles vereinfachte, wie zum Beispiel die Ausgaben der Console oder die Einbindung der Sensoren. Der Hauptgrund war aber der Luftqualitätssensor, der nur ein analogRead benötigt und unser selbst geschriebenes nicht richtig funktionierte. Atmel Studio bot uns die Möglichkeit, dort ein existierendes Arduino Projekt zu importieren und das zu bearbeiten. Die Entwicklungsumgebung wollte ich nicht wechseln, da wir bereits mit dem Pololu Programmer unseren Chip programmieren konnten und auch die Seriellen Ausgaben funktionierten. Das einzige was sonst noch geändert werden musste, war die F\_CPU in der Toolchain. Das nächste was ich gemacht habe, war den Code in Arduino umzuschreiben. Der Code für den Geigenzähler, den Gregor angepasst hatte, musste ich nur mehr in das Projekt kopieren.

**EAGLE:**

Kommen wir zum Schaltplan:

Den fertigte ich mit dem Programm EAGLE an. Wir hatten uns schon vor der Zwischenpräsentation ein bisschen mit dem Planen unseres Schaltplans auseinandergesetzt, auf dem ich nachher aufbaute.

Es kamen ja noch 2 weitere Sensoren dazu, einmal der Luftqualitätssensor und einmal der Geigerzähler. Der Part für den Geigerzähler macht dabei den weitaus größten Teil unseres Boards aus (wie wir hier auch auf der Zeichnung schnell sehen können). Auf die einzelnen Parts gehe ich gleich genauer ein. Und auch der analoge Luftqualitätssensor brachte wegen der analogen Stromversorgung ein paar Neuerungen.

Starten wir bei dem Netzteil unseres Boards:

Unser Temperatursensor verbraucht 2.5mA und was wir erst bei der Planung des Netzteils rausbekommen haben ist, dass der Luftqualitätssensor durch seine Technologie mit einer Heizspirale 150mA Strom verbraucht. Damit ist auch die Versorgung per Batterie ins Wasser gefallen. Die Geigerröhre verbraucht im Vergleich zum Luftqualitätssensor auch nur einen Bruchteil.

Unser Netzteil arbeitet mit 7 – 40 Volt Eingangsspannung und macht daraus unsere gewünschten 3,3 Volt. Unser Chip und alle Sensoren, mit Ausnahme dem Geigerzähler (dazu später mehr), arbeiten mit dieser Spannung.

Unser Board kann nur mit Gleichspannung versorgt werden, also mit Batterien oder einem normalen Netzteil.

Am Anfang haben wir eine Verpolungsschutz-Diode mit Sicherung, die unser Board bis zu einer Spannung von 7 Volt schützt. Unter 7 Volt wird es dann kritisch, weil es da die 0.5 Ampere, für die die Sicherung ausgelegt ist, übersteigt.

Das Netzteil funktioniert wie folgt:

Der Kondensator C5 stabilisiert die Eingangsspannung, dann gelangt sie in den IC2. Dabei handelt es sich um einen Spannungsregler mit 5 Pins. Interessant sind dabei Pin 2, dort kommt der Strom raus und durch Spule an den ATmega. Pin 4 dient als Feedback-Leitung, also da kommt der umgewandelte Strom nach Durchlaufen der Spule wieder zurück zum Spannungsregler, der dann die Spannung erneut anpassen kann.

Der Kondensator C6 macht dann schlussendlich aus den Impulsen des Spannungsreglers wieder eine konstante Spannung und diesmal mit 3,3 V. Das wars mit dem Netzteil.

Hier am Ende sieht man noch weitere Kondensatoren, sogenannte Abblockkondensatoren (3 für den ATmega und 1 für Geiger-Röhre) um die effektive Spannung, die jetzt zum Chip gelangt, nochmals zu stabilisieren.

Weiter zum ersten Teil des Geigerzählers:

Sowohl das Netzteil als auch den Geigerzähler habe ich so gut wie 1 zu 1 vom Internet entnommen und habe nur kleine Änderungen vorgenommen.

Hier handelt es sich um die Stromversorgung der Röhre. Wie vorher schon einmal angesprochen arbeitet die Röhre mit einer Spannung von 300 bis 600 Volt, die natürlich erst erzeugt werden müssen.

Dazu gehe ich jetzt nicht genauer ein. Die Widerstände dienen für die Einstellungen, die bei dem IC vorgenommen werden müssen und er besitzt auch noch einen Dreher (zeigen), wo es möglich ist die Ausgangsspannung anzupassen. Der IC3 ist ein reiner Zeitgeber.

Die ganz geringe Menge an Strom, aber mit fast 400 Volt gelangt jetzt an die Röhre. Wie diese arbeitet haben wir bereits besprochen.

Auf der anderen Seite muss jetzt der Impuls der Röhre verarbeitet werden:

Dazu kommen Kondensatoren und Widerstände ins Spiel, um das Signal überhaupt zu entdecken und ein Transistor Q3. Der Transistor ist, je nach Impuls von der Geiger-Röhre, leitend oder nichtleitend. Bekommt er keinen Impuls ist er nichtleitend und der Strom geht, vom 10k Pull-Up-Widerstand, direkt zum Interrupt-Pins des ATmega-Chips. Kommt ein Impuls legt der Transistor um und der Strom sucht sich den kürzeren Weg und gelangt durch den Transistor direkt zu Ground. Der Interrupt-Pin des Chips hat eine fallende Kurve und zeichnet den Impuls auf.

Kommen wir zu Letzt zu den weniger spannenden Sensoren:

Diese arbeiten, wie vorher erwähnt, alle mit einer Spannung von 3.3 Volt.

Der Luftqualitätssensor und die ISP-Programmierschnittstelle sind direkt mit dem ATmega verbunden. Der CC1101 hat auf den Pins die er zusammen mit dem Programmer verwendet, also MISO, MOSI und CLK, zusätzlich Widerstände, sodass wenn beide senden immer der Programmer das stärkere Signal schickt.

Zum besseren Debuggen haben wir noch eine Serielle Schnittstelle eingerichtet und hier sehen wir noch den DHT11-Temperatursensor. Beide sind ebenfalls direkt mit dem Chip verbunden.

Analoge Stromversorgung:

Da wir mit dem Luftqualitätssensor auch einen analogen Sensor besitzen, musste ich mich auch um die analogen Stromversorgungen kümmern, dabei verwenden wir AVCC.

Die Drosselspule L1 und der Kondensator C2 ergeben zusammen einen Tiefbassfilter (Quarz schwingt bisschen auf Leitung mit, damit werden Überschwingungen und Störungen eliminiert), um eine genauere und stabilere Stromversorgung für analoge Sensoren zur Verfügung zu stellen und bei dem AREF-Pin habe ich einen Kondensator C1 verbaut der dazu dient die Spannung nochmal zu stabilisieren.

Hier noch ein kurzer Blick auf das PCB-Layout, dass ihr das auch kurz seht:

**Präsenzwoche:**

**PCB-Anpassung**

Als wir in der Praktikumswoche unser PCB programmieren und testen wollten, funktionierte gar nichts. Um sicher zu gehen, dass nicht beim Löten etwas kaputt gegangen ist, testeten wir ein einfachen blink Sketch und der funktioniert. Wir fügten immer wieder neue Sensor hinzu die auch funktionierten. Der Fehler lag beim CC1101. Aber wo genau, wussten wir nicht. Da wir aber am Code nichts geändert hatten, musste es beim PCB liegen. Wir glichen alle Ein- und Ausgänge mit unserem Breadboard ab und fanden da 2 Fehler. Der eine war, dass wir den GD0 Pin an den PB7 Pin vom Chip und den CSN vom CC1101 anstatt an den Slave Select Pin an den PB6 angeschlossen haben. Warum das passiert ist, wissen wir selbst nicht, da wir alles vor der Bestellung des PCB nachkontrolliert haben. Das Erste was wir ändern mussten, war, dass wir überhaupt die Pins PB6 und 7 verwenden können. Diese sind standardmäßig für die externe Clock vorgesehen und können nicht einfach so angesteuert werden. Um dieses Problem zu lösen, haben wir die pin\_arduino.h Datei etwas verändert. Was genau geändert werden musste, haben wir nach einer Anleitung gemacht, welche wir im Internet gefunden haben. Aber hauptsächliche wurde nur die Anzahl der Digitalen Pins um 2 erhöht, und das an mehreren Stellen. Das nächste Problem war, dass wir den Slave Select Pin des Chips nur mit ein paar Änderungen des Codes verlegen mussten. Das war etwas schwieriger herauszufinden. Zuerst müssen wir in der Init den SS Pin, als Pin 10 bei uns als Output setzten und zudem noch digitalWrite mit High. Um den PB7, Pin 14 als SS Pin festzulegen, wird in der pins\_arduino.h bei #define PIN-SPI\_SS der Wert von 10 auf 14 verändert. Das wäre alles gewesen, nun funktionierte auch der TICC1101 und der Spektrum Analyzer zeigte an, dass etwas gesendet wird. Wir waren froh, dass es funktionierte, ansonsten hätten wir die Leitungen auf dem PCB verlegen müssen.

**Programmierung**

In der Praktikumswoche habe ich dann den Rest des Codes geschrieben. Das heißt, ich versuchte die sendData Methode des CC1101, die wir verwendeten, zu verstehen, um zu wissen, wie das Format des Pakets, welches wir an das Station Team schicken, auszusehen hat. Ich passte das dann so an, dass es reicht, ein Array aus den Sensorwerten zu erstellen. Dieser Array hat die Länge 6 und besteht aus 6 Bytes. Das erste Byte ist, wie mit den anderen Teams abgesprochen, die Adresse der Station und das zweite ist unsere Adresse. Als drittes Byte schicken wir die Temperatur vom DHT11, welches einen Wert von 0 bis 50 Grad annehmen kann. Wie wir die Werte des Temperatursensor erhalten, haben wir bereits in der Zwischenpräsentation genauer erklärt. Nämlich erhalten wir ein 40 Bit langes Array, wo wir die einzelnen Bits an der richtigen Stelle auslesen und diese als int speichern. Die Luftfeuchtigkeit besetzt das 4. Byte im Paket und alles funktioniert gleich wie bei der Temperatur, nur an einer anderen Stelle im Array. Diese zwei Werte werden dann im Paket Array gespeichert und überschreiben den alten Wert.

Um den Wert vom Luftqualitätssensor zu erhalten, reicht es, wie oben schon gesagt, einfach die Methode analogRead aufzurufen. Der Wert, den wir erhalten, liegt bei guter Luftqualität zwischen 120 und 140. Aber wenn man diesen Wert auf der Website des Station Teams lesen würde, dann hätte niemand eine Ahnung was dieser Wert bedeutet. Zudem kann man den Sensor auch mit einem Schraubenschlüssel anders kalibrieren und die Werte wären wieder anders. Deshalb haben wir 2 Grenzen hinzugefügt, eine bei 150 und die andere bei 180, bei denen wir gesagt haben, ab diesem Wert ist die Luft nicht mehr so gut, und bei der anderen wäre die Luft schon schlecht. Je nach Bereich, in welchen der erhaltene Wert fällt, wird an das Station Team eine 0, 1 oder 2 geschickt.

An der fünften Stelle im Paket kommt dieser Wert vom Luftqualitätssensor. Der würde nur 2 Bit benötigen, aber aus vereinfachungsgründen für uns und das Station Team schicken wir es als ganzes Byte.

So ähnlich funktioniert das auch beim Geigenzähler. Der Wert 0,1 Mikrosievert/Stunde würde den wenigsten etwas sagen. Deshalb haben wir uns entschieden, bei mehr als 10-facher Strahlung des Durchschnittes an das Station Team eine 1 zu schicken, was gefährlich bedeutet. Anderenfalls senden wir eine 0. Auch hier senden wir ein ganzes Byte, obwohl wir nur ein Bit benötigen würden.

Wir haben uns entschieden, die Sensorwerte alle 30 Sekunden an die Station zu senden, da sich die Temperatur und Luftfeuchtigkeit nicht schlagartig ändern. Eine Ausnahme bildet der Geigerzähler. Sobald dieser einen Wert erhält, welcher über dem Schwellenwert liegt, wird das Paket sofort an die Station geschickt.

**CC1101 – Kommunikation:**