**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.

**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 einen 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 unter 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.