Facharbeit

Bau und Programmierung eines mikrocontroller - gesteuerten Weckers, welcher die Weckzeit an sensorbestimmte Wetterdaten anpasst

verfasst von
Moritz Kasemann

Leistungskurs Informatik Q1
Betreuer: Herr Fassbender
Abgabetermin: 28.4.2017

Inhaltsverzeichnis

1 Einleitung

Man kennt es, man möchte mit dem Fahrrad zur Arbeit oder Schule fahren, aber es regnet. Bei Regen möchte man nicht mehr mit dem Fahrrad fahren, aber der Bus, mit dem man dann fahren muss, ist schon weg. Oder man möchte mit dem Auto fahren, aber es hat in der Nacht gefroren und man muss noch Eis kratzen und kommt deswegen zu spät zur Arbeit. Folglich wäre es sinnvoll immer zur richtigen Zeit geweckt zu werden. Wenn es regnet oder gefroren hat, müsste man etwas früher geweckt werden, damit man früh genug aufsteht, um den Bus noch zu erreichen oder sein Auto von Schnee und Eis zu befreien. Sollte aber schönes Wetter sein, möchte man so lange schlafen wie möglich, also sollte man so spät wie möglich geweckt werden. Genau dies soll mit Hilfe eines Mikrocontrollers umgesetzt werden. Der Mikrocontroller soll anhand von Sensordaten bestimmen können, ob es nötig ist den Nutzer früher als normal zu wecken.

2 Aufbau/Übersicht

Um dieses Projekt umsetzten zu können wurden folgende Bauteile verwendet:

- ein Arduino Mega 2560
- ein 3,2 Zoll Display von HVGA
- ein Temperatursensor von Analog Devices
- eine Real Time Clock (RTC) von STMicroelectronics
- ein DCF77-Empfänger
- ein Piezo Summer
- zwei normale Druckknöpfe
- ein Drehimpulsgeber, der auch ein weiterer Druckknopf ist
- ein Arduino Mega 2650 R3 Proto Shield

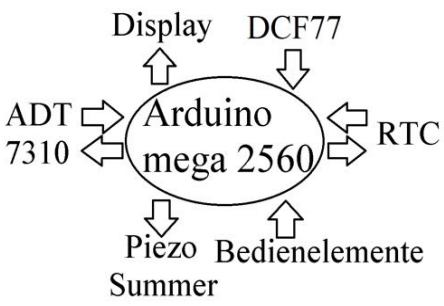

Abb1: Blockschaltbild des Weckers

In Abb1 ist der Aufbau des Weckers vereinfacht dargestellt. Die Pfeile geben an, ob Informationen zu dem jeweiligen Bauteil fließen oder das Bauteil

Informationen weitergibt. Der Mikrocontroller (Arduino Mega 2560) ist der Mittelpunkt, er muss alles steuern.

Er muss:

- dem Display sagen, was es anzeigen soll
- die Signale des DCF77-Empfängers interpretieren
- die Real Time Clock (RTC) auslesen und ggf. setzen
- auf Eingaben des Nutzers reagieren
- den Nutzer mit dem Piezo Summer wecken
- den Temperatursensor (ADT7310) konfigurieren und auslesen

Der Temperatursensor und der DCF77-Empfänger befinden sich in einer kleinen Box, die mit einem D-Sub Steckverbinder versehen ist. (10.1 Bild 3) Über ein Kabel wird diese mit dem Arduino verbunden. Um alle nötigen Widerstände, Kondensatoren und die RTC unterzubringen, wurde ein Shield, das genau für diesen Arduino gebaut ist, verwendet. (10.1 Bild 2) Der Vorteil diesen Shields ist, dass das Display einfach auf dieses Shield gesteckt werden kann. Alle Bedienelemente befinden sich auf einer Lochrasterplatine die über Drähte mit dem Arduino verbunden ist. (10.1 Bild 4)

Pin am Arduino	Ziel	Variablenname
A6	Chip Select Temperatursensor	cs
A3	Clock Temperatursensor	clk
A5	DIN Temperatursensor	DOUT
A4	DOUT Temperatursensor	DIN
3	Knopf für Einstellungen	scrW
2	Knopf des Drehimpulsgebers	drehKnopf
A8	Piezo Summer	piezo
A1	Knopf für Graphen/Uhr	wechselKnopf
A7	Ausgang des DCF77-Moduls	dcfPin
A0	Drehimpulsgeber Leitung 1	hoch
A2	Drehimpulsgeber Leitung 2	runter
20	SDA Real Time Clock	-
21	SCL Real Time Clock	-
22-53	Display	-

Tabelle 1: PIN-Belegung Arduino (DIN: Dateneingang, DOUT Datenausgang, SDA: serielle Datenleitung, SCL: Clock der seriellen Schnittstelle)

3 Herzstück Arduino Mega 2560

Als Steuereinheit des Weckers wurde ein Arduino Mega 2560 gewählt. (10.1 Bild 1) Dieser gehört, wie der Name schon verrät, der Arduino Mikrocontroller Reihe an. Das Schöne ist, dass der Mikrocontroller bereits auf einer Platine verlötet ist und eine USB-Schnittstelle besitzt, über die es möglich ist ihn direkt zu programmieren. Er kann entweder über die USB-Schnittstelle oder über den verlöteten Powerstecker mit Strom versorgt werden. Die nötige Software ist OpenSource und es gibt eine große Community, die immer bereit ist bei Fehlern oder Problemen zu helfen. Es wurde der Arduino Mega 2560 gewählt, da er mit 256Kb Speicher, von dem effektiv 248Kb nutzbar sind, auf jeden Fall genug Speicher hat um den kompletten Programmsketch zu beinhalten. Neben dem großen Flash Speicher war auch der 4Kb große EEPROM ein Grund für den Arduino, da viele Arduinos gar keinen EEPROM haben, oder so wenig haben, dass nicht alle Daten, die nach einem Stromausfall behalten werden sollen, abgespeichert werden könnten. Auch von Vorteil sind die vielen Pins, die zur Verfügung stehen, da man so nicht eingeschränkt wird, falls man den Wecker noch erweitern möchte.

4 Uhrzeit

4.1 Real Time Clock

Der Wecker braucht natürlich die Uhrzeit, damit er weiß wann er den Nutzer wecken muss. Es wäre möglich den internen Zähler des Arduinos dafür zu nutzen, aber dieser hat einige Nachteile. Zum einen ist dies ein Zähler der die Millisekunden seit dem Einschalten des Arduinos zählt. Somit fängt dieser wieder bei null an, wenn die Stromversorgung einbrechen sollte. Ein weiterer Nachteil ist, dass dieser Zähler „nur“ ein 32-bit Register zur Verfügung hat. Wenn das Maximum von 2^{32} Millisekunden erreicht ist, was ca. 49 Tage und 17 Stunden dauert, fällt der Zähler wieder auf null. Der größte Nachteil des internen Zählers ist aber, dass er stoppt wenn ein Interrupt ausgelöst wurde. Man hätte die anderen Nachteile beheben können, aber da unbedingt Interrupts genutzt werden sollten, kam der interne Zähler nicht infrage. Die Alternative ist eine Real Time Clock(RTC). In diesem Fall wurde die RTC M41T00CAP von STMicroelectronics verwendet. Diese besitzt einen internen Akku, der die RTC mit Strom versorgt, sobald sie keine externe Stromversorgung mehr hat. Sie hat acht

Register, die über einen I²C Bus ausgelesen und gesetzt werden können. In diesen Registern wird das Datum, die Uhrzeit und die Kalibrierung gespeichert. Mithilfe des Kalibrierungsregisters ist es möglich den Quarz auf die Raumtemperatur anzupassen. Zudem kann man anhand der Kalibrierung feststellen, ob der Quarz zum Stillstand gekommen ist. Das heißt, dass der Arduino weiß, ob die Zeit der RTC verlässlich ist, oder nicht. [6]

4.2 DCF77-Empfänger

Die Uhrzeit der Real Time Clock(RTC) soll regelmäßig nachgestellt werden. Dazu kann entweder ein GPS-Empfänger genutzt werden oder ein DCF77-Empfänger. Der GPS-Empfänger hätte den Vorteil, dass er überall auf der Welt die Uhrzeit eines Satelliten bekommen würde, im Gegensatz zu dem DCF77-Empfänger, der sein Signal aus Mainflingen bei Frankfurt am Main bekommt. Da der DCF77-Empfänger deutlich weniger kostet als der GPS-Empfänger wurde dieser gewählt. Der Empfänger wandelt bereits das analoge Signal aus Mainflingen in ein digitales Signal um. Eine gesamte Übertragung aller Informationen dauert eine Minute und beinhaltet sowohl die Uhrzeit, das Datum als auch Katastrophenwarnungen. Die Informationen, die der Empfänger ausgibt, müssen noch als Einsen und Nullen interpretiert werden. In jeder Sekunde liegt genau einmal eine logische Eins an, es entscheidet lediglich die Dauer des Anliegens, ob es eine Eins oder eine Null ist. Sollte sie 100ms anliegen gilt dies als Null, sollte sie 200ms anliegen gilt dies als Eins. Damit erkennbar ist, wann eine neue Übertragung aller Informationen startet, wird eine Sekunde lang keine logische Eins angelegt.
Die Bits

- 21-27 geben die Minuten an
- 29-34 geben die Stunden an
- 36-41 geben den Kalendertag an
- 42-44 geben den Wochentag an
- 45-49 geben die Monatsnummer an
- 50-57 geben das Jahr an.

Alle restlichen Bits sind nicht relevant. [5]

5 Sensordaten

5.1 Temperatursensor ADT7310

Um die Temperatur zu messen wurde der Temperatursensor ADT7310 von der Firma Analog Devices genutzt. Er hat eine maximale Auflösung von 16-Bit, sodass die kleinste Einheit 0.0078125°C entspricht. Seine absolute Genauigkeit beträgt +/- 0.5°C, womit er der beste Temperatursensor dieser Firma ist, den man von Hand verlöten kann. Der Sensor wird über einen SPI-Bus ausgelesen. Nach Anlegen der Betriebsspannung muss dem Temperatursensor mithilfe des Registers 0x01 mitgeteilt werden, in welcher Auflösung er messen soll, entweder 13 oder 16-Bit. In diesem Projekt wurde immer 16-Bit verwendet. Bei der ersten Messung benötigt der Temperatursensor 6ms, sonst 240ms. Danach darf man das entsprechende Register auslesen, in diesem Fall 0x02. Wobei man das Ergebnis der allerersten Messung verwerfen sollte, da dieses Ergebnis eine absolute Ungenauigkeit von +/- 5°C hat. Das allererste Bit, das man von dem Temperatursensor erhalten hat, gibt an ob die Temperatur positiv(0) oder negativ(1) ist. Die weiteren 15-Bit sind binär codiert. Wenn die Temperatur positiv ist, muss man alle Werte dieser 15-Bit addieren und anschließend mit der Auflösung, in diesem Fall 0.0078125, multiplizieren. Dann erhält man die Temperatur in °C. Sollte die Temperatur negativ sein, so muss man von dem Binärwert der 15-Bit eins subtrahieren und anschließend jedes Bit negieren. Danach darf man genauso vorgehen, als ob es eine positive Temperatur wäre. [1]

5.2 Datenverwaltung und Datenauswertung

Jede gemessene Temperatur wird mit einem Zeitstempel versehen und dann in einem Array abgespeichert. Dieses Array fasst maximal 110 Werte; sollte nun der 111-te Wert kommen, so wird der älteste Wert gelöscht und der neue Wert wird an den Anfang gesetzt. Das Array fasst genau 110 Werte, da alle fünf Minuten die Temperatur neu gemessen wird und alle gemessenen Temperaturen der letzten neun Stunden dort abgespeichert werden sollen. Alle Werte können in einem Graphen dargestellt werden, damit der Nutzer sich ein Bild des Temperaturverlaufs machen kann. Zudem hat der Nutzer die Möglichkeit mit einem Cursor den Graphen abzutasten und sich jeden gemessenen Wert genau ausgeben zu lassen. (10.1 Bild 9) Die Darstellung des Temperaturverlaufs ist natürlich nicht die wesentliche Aufgabe des Weckers. Die

eigentliche Aufgabe besteht darin diese Temperaturen auszuwerten und zu entscheiden, ob es in der Nacht gefroren hat oder nicht. Der Nutzer kann angeben, wie lange die Temperatur unter einem bestimmten Wert gelegen haben muss, bevor die alternative Weckzeit als tatsächliche Weckzeit übernommen wird. So kann der Nutzer den Wecker seinen persönlichen Bedürfnissen anpassen. Der Wecker selbst muss nur noch entscheiden, ob das von dem Nutzer eingegebene Kriterium erfüllt ist oder nicht.

6 User Interface

6.1 Display

Zum Anzeigen aller notwendigen Dinge wurde ein 3,2‘ IPS TFTLCD Display der Firma HVGA genutzt. Dieses hat eine Auflösung von 480x320 Pixeln und besitzt einen SD-Kartenslot, um die Verwendung von Grafiken zu vereinfachen. Zudem besitzt es einen kleinen Druckknopf, der zur Verwendung als Reset-Knopf gedacht ist. Das Display wurde genau für diesen Arduino gebaut, somit stimmen alle Maße mit dem Arduino überein und das Display lässt sich durch einfaches Aufstecken auf den Arduino anschließen. Es gibt zudem eine Library, die alle Grundfunktionen, die nötig sind um das Display anzusteuern, bietet. Außerdem gibt es noch einige weitere Libraries, die als Addon dienen und viele nützliche Funktionen bieten.

6.2 Bedienelemente

Zum bedienen des Weckers gibt es zwei normale Druckknöpfe und einen Drehimpulsgeber, der auch zugleich ein weiterer Druckknopf ist. (10.1 Bild 4) Es sollte in irgend einer Form ein Drehknopf verwendet werden, da es mit einem Drehknopf deutlich einfacher ist zu scrollen, zum einem durch die Einstellungsmöglichkeiten und zum anderen durch den Temperaturgraphen. Man hätte für diese Aufgabe sowohl ein Drehpotentiometer als auch einen Drehimpulsgeber nehmen können. Ein Drehpotentiometer wurde in diesem Fall aber nicht verwendet, da es erstens ein analoges Signal ausgibt und dieses von dem Mikrocontroller zu einem digitalen Signal umgewandelt werden muss, was meistens eine Ungenauigkeit mit sich bringt. Zweitens verschleißen Drehpotentiometer sehr schnell und drittens hat ein Drehpotentiometer einen Anschlag, es ist also nicht möglich immer weiter in eine Richtung zu drehen. In

all diesen Gesichtspunkten ist der Drehimpulsgeber immer im Vorteil. Zum einen hat dieser keinen Anschlag und zum anderen gibt dieser immer ein eindeutig zu interpretierendes Signal aus. Der Drehimpulsgeber hat zwei Leitungsausgänge die zu beobachten sind. Wenn an ihm gedreht wird, springt zuerst die eine Leitung auf Logisch eins und erst danach die andere Leitung. Wird nun anders herum gedreht, springt die Leitung, die eben als zweites auf Logisch eins gesprungen ist, nun zuerst auf Logisch eins und die andere Leitung folgt in einem gewissen Abstand.[4] Somit kann immer eindeutig entschieden werden, ob an dem Drehimpulsgeber rechtsherum oder linksherum gedreht wurde. Der Druckknopf des Drehimpulsgebers und einer der beiden weiteren Druckknöpfe belegen nicht normale Dateneingänge des Arduinos, sondern Interrupts. Dies sind Eingänge die, je nachdem wie sie konfiguriert sind, bei bestimmten Events, zum Beispiel bei dem Wechsel von Logisch null zu Logisch eins, die aktuelle Tätigkeit des Mikrocontrollers unterbrechen und zu einer bestimmten Stelle im Programm springen. Nachdem die Befehle abgearbeitet wurden geht der Arduino wieder seinen eigentlichen Tätigkeiten nach. Die beiden Druckknöpfe belegen Interrupts, da die Eingabe des Nutzers absoluten Vorrang hat und zuerst bearbeitetet werden soll. Das Belegen von Interrupts hat den Vorteil, dass die Eingabe des Nutzer sofort bearbeitetet wird und der Nutzer nicht warten muss, zum Beispiel auf das Finden der Uhrzeit.

7 Programmierung des Weckers

Ein wesentlicher Bestandteil des Programms sind die vielen verschiedenen Libraries. Sie machen das Programm übersichtlich und es ist deutlich einfacher Fehler zu suchen. Es werden sechs verschiedene Libraries genutzt: „UTFT“, „UTFT_CTE“, „auslesen“, „rtc“, „speicherVerwaltung“ und „dcfReader“. Die Libraries „UTFT“ und „UTFT_CTE“ dienen dem Ansteuern des Displays. „UTFT“ bietet alle wichtigen Grundfunktionen um das Display ansteuern zu können. „UTFT_CTE“ ist ein Addon für „UTFT“ und bietet Funktionen, wie zum Beispiel das schreiben von Texten oder das zeichnen von Kreisen. Beide Libraries wurden von der Arduino Community geschrieben und dürfen frei benutzt werden. Alle weiteren Libraries wurden selbst geschrieben.

„auslesen“ beherbergt eine Methode, die den Temperatursensor mit Hilfe der SPI-

Schnittstelle ansteuern kann. Sie kann dem Temperatursensor sowohl sagen in welcher Auflösung dieser messen soll, als auch die Temperatur auslesen. Zudem wandelt sie die empfangenen Bits direkt in eine Temperatur um und gibt diese zurück.
Die Library „rtc“ dient dem Ansteuern der Real Time Clock(RTC). Dafür nutzt sie die Library „Wire“, die direkt von der Arduino Entwicklerumgebung zur Verfügung gestellt wird. „Wire“ wird benötigt, weil sie alle nötigen Methoden zum nutzen eines I²C-Busses bereit stellt. „rtc“ besitzt zwei Methoden, eine um eine neue Uhrzeit zu setzen und eine um die momentane Uhrzeit auszulesen.
Um die Verwaltung des EEPROMS möglichst einfach zu halten, wurde die Library „speicherVerwaltung“ geschrieben. Sie greift auf die Library „EEPROM“ zurück, die bereits von der Arduino Entwicklerumgebung mitgeliefert wird. „speicherVerwaltung“ bietet zwei Methoden, die eine speichert ein übergebenes Array ab, in diesem Fall die Weckzeiten, und die andere liest den Speicher aus und speichert das Ausgelesene in einem übergebenen Array ab.
Die letzte Library „dcfReader“ gibt die Möglichkeit die Signale des DCF77-Empfängers auszuwerten. Sie besitzt eine Methode, die zuerst die Signale als eins oder null interpretiert und danach diese Einsen und Nullen in die Uhrzeit und das Datum umwandelt.
Die beiden wichtigsten Methoden des Hauptprogramms sind die Methode „setup“ und die Methode „loop“. Beide Methoden müssen in jedem Programm, das für Arduinos geschrieben wird, vorhanden sein, selbst wenn sie nichts beinhalten.
„setup“ wird einmalig bei dem Start des Arduinos ausgeführt und ist für die Initialisierung zuständig. In diesem Fall muss sie die Interrupts, und deren Funktion, ankündigen, die Pinbelegungen zuweisen, zum Beispiel, dass der Piezo Summer ein OUTPUT ist, erstmalig die Uhrzeit auslesen und falls die Real Time Clock gestoppt haben sollte die Uhrzeit neu finden, die Weckzeiten aus dem EEPROM auslesen, das Display initialisieren und das Array, das alle Temperaturwerte beinhalten soll, mit einem Defaultwert belegen.
Die Methode „loop“ ist, wie der Name schon verrät, eine Schleife, aber keine normale Schleife, sondern eine Endlosschleife, die man niemals dauerhaft verlassen kann. In dieser Anwendung, wartet diese Schleife auf Events. Es gibt vier größere Events unter denen unterschieden werden muss.

1. Es sind fünf Minuten vergangen.
2. Es ist 4Uhr nachts oder der Nutzer hat die Suche nach der Uhrzeit angeordnet.
3. Der Nutzer muss geweckt werden.
4. Der Nutzer tätigt eine Eingabe.

Sollte das erste Event eintreten, so muss die Temperatur erneut gemessen werden und diese muss in dem dafür vorgesehenen Array abgespeichert werden. Es ist bewusst die Uhrzeit von 4Uhr nachts gewählt worden, da die Zeitumstellung um 3Uhr nachts geschieht und somit die Zeitumstellung nicht extra bedacht werden muss.

Bei dem zweiten Event muss die Uhrzeit mithilfe des DCF77-Empfängers gesucht werden und die RTC korrigiert werden.

Das dritte Event muss aufgeteilt werden in zwei mögliche Fälle. Die eine Möglichkeit ist, dass der Nutzer geweckt werden möchte und die angegebene Weckzeit erreicht ist und die andere Möglichkeit ist, dass der Nutzer geweckt werden möchte, die Weckzeit aber noch nicht erreicht ist, sondern das Kriterium zum früher wecken erfüllt ist und die angegebene frühere Weckzeit erreicht ist. In beiden Fällen muss der Nutzer geweckt werden, also muss der Piezo Summer angesteuert werden, bis der Nutzer dies beendet.

Das letzte Event muss auch differenziert betrachtet werden, schließlich gibt es zwei Druckknöpfe und einen Drehimpulsgeber, der auch als Druckknopf fungiert. Zur Einfachheit werden die beiden Druckknöpfe A und B genannt und der Druckknopf des Drehimpulsgebers wird C genannt. Sollte der Knopf A gedrückt werden, wird auf der Anzeige zwischen der Uhr und dem Temperaturgraphen gewechselt, je nachdem welches gerade angezeigt wird. Das Drücken der beiden Knöpfe B und C wird nicht als Event gehandelt, da beide Knöpfe jeweils einen Interrupt belegen. Auch das Drehen des Drehimpulsgebers wird nicht als Event gehandelt, da das Drehen, nur innerhalb der Methoden die durch die Interrupts ausgelöst werden, eine Funktion erhält.

Sollte Knopf B gedrückt werden, wird ein Interrupt ausgelöst und der Arduino springt sofort in die Methode „screenEinstellungen“. Dort wird eine weitere Methode aufgerufen, die zuerst einmal das Aussehen des Menüs aufbaut. Nachdem dies geschehen ist, kann der Nutzer durch drehen des Drehimpulsgebers durch die einzelnen Unterpunkte scrollen. Er kann wählen zwischen den allgemeinen Einstellungsmöglichkeiten, den einzelnen Wochentagen und dem Verlassen des Menüs. Durch drücken des Knopfes C kann er dann bestätigen und er wird in den entsprechenden Unterpunkt weitergeleitet.

In den allgemeinen Einstellungsmöglichkeiten, kann er dann die Funktion des früher Weckens aktivieren oder deaktivieren. Außerdem kann er die minimale Temperatur und die Dauer des Unterschreitens dieser Temperatur angeben. Folglich wird der Nutzer früher geweckt, wenn die minimale Temperatur um die angegebene Dauer unterschritten wird. Zudem kann der Nutzer entscheiden, ob die Uhrzeit digital oder analog angezeigt werden soll. Als letzte Einstellungsmöglichkeit, in dem Unterpunkt, kann der Nutzer die Suche nach der aktuellen Uhrzeit anordnen.
In den jeweiligen Einstellungsmöglichkeiten der einzelnen Wochentage, kann er angeben, wann und ob er an diesem Wochentag geweckt werden möchte. Zudem kann er angeben um wie viele Minuten er früher geweckte werden möchte, falls das Kriterium, was er in den allgemeinen Einstellungsmöglichkeiten angegeben hat, erfüllt ist. Nachdem der Nutzer die allgemeinen Einstellungsmöglichkeiten oder Einstellungsmöglichkeiten der einzelnen Wochentage wieder verlässt, kommt er wieder in das Menü. Sollte der Nutzer das Menü verlassen, so werden alle seine Änderungen, mithilfe der Library „speicherVerwaltung“, im EEPROM abgespeichert. Danach wird wieder die Uhr oder der Temperaturgraph angezeigt, je nachdem wo der Nutzer vorher war.
Sollte der Nutzer den Knopf C drücken, so wird ein Interrupt ausgelöst und der Arduino springt sofort in die Methode „graphAnzeigen“. Diese Methode wird nur dann nicht sofort wieder verlassen, wenn gerade der Temperaturgraph angezeigt wird. Wenn der Temperaturgraph gerade angezeigt wird, werden der Temperaturwert und die Uhrzeit der letzten Messung genau angezeigt und der Nutzer kann mithilfe des Drehimpulsgebers durch alle gemessenen Werte scrollen. Ein roter Strich stellt dabei den Cursor dar. Durch drücken des Knopfes B wird diese Methode wieder verlassen.
Es ist noch zu erwähnen, dass das Design, zum Beispiel im Menü, zu Fuß programmiert werden musste, da ein Arduino keine vorgefertigten Befehle für dies hat bzw. versteht.

8 Probleme und deren Behebung

Es sind zwei größere Probleme aufgetreten. Das eine bestand darin, dass, obwohl man einen Knopf eindeutig nur einmal gedrückt hat, der Arduino reagiert, als ob man dreimal oder öfter gedrückt hätte. Das andere Problem äußerte sich in der Form, dass der DCF77-Empfänger, trotz dem Abwarten von über 10 Übertragungszyklen, keine

realistischen Ergebnisse lieferte.

Um bei dem ersten Problem herauszufinden, ob der Knopf oder der Arduino das Problem darstellt, wurden die Signale des Knopfes mithilfe eines Oszilloskops überprüft. Abb1 stellt dabei die verwendete Schaltung dar. In Abb2 kann man die gemessenen Daten sehen. Die rote Linie stellt die Schwellspannung dar, die überschritten werden muss, damit eindeutig eine logische Eins erkannt wird. Die grüne Linie stellt die Schwellspannung dar, die unterschritten werden muss, damit eindeutig eine logische Null erkannt wird. [3] Es fällt sofort auf, dass gleich mehrmals beide Schwellspannungen überschritten als auch unterschritten werden. Da der Arduino deutlich schneller auf Änderungen der anliegenden Spannung reagiert, als sie sich ändert, arbeitet er ihn, in diesem Beispiel, gleich viermal ab. Dieses Verhalten eines Knopfes nennt man prellen und es ist nicht untypisch. Je nach Qualität des Knopfes tritt dies mehr oder weniger stark auf. Um dieses Problem zu beheben gibt es zwei Möglichkeiten, entweder fängt man es mittels der Software ab oder mithilfe von zusätzlicher Hardware. Möchte man dies durch Software lösen, muss man die Abstände zwischen den Statusänderungen messen. Liegt nur sehr wenig Zeit zwischen ihnen, kann man sich sicher sein, dass dies auf das Prellen zurück geführt werden kann. Möchte man es mit zusätzlicher Hardware lösen, wie es hier getan wurde, muss man die Schaltung, wie in Abb3, aufbauen. Der Zeitraum in dem das Prellen auftritt entscheidet, wie groß der Kondensator sein muss. Der Zeitraum des

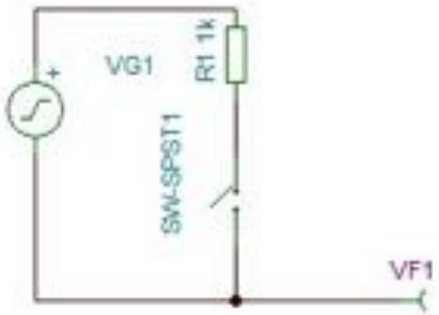

Abb1: Beschaltung des Knopfes

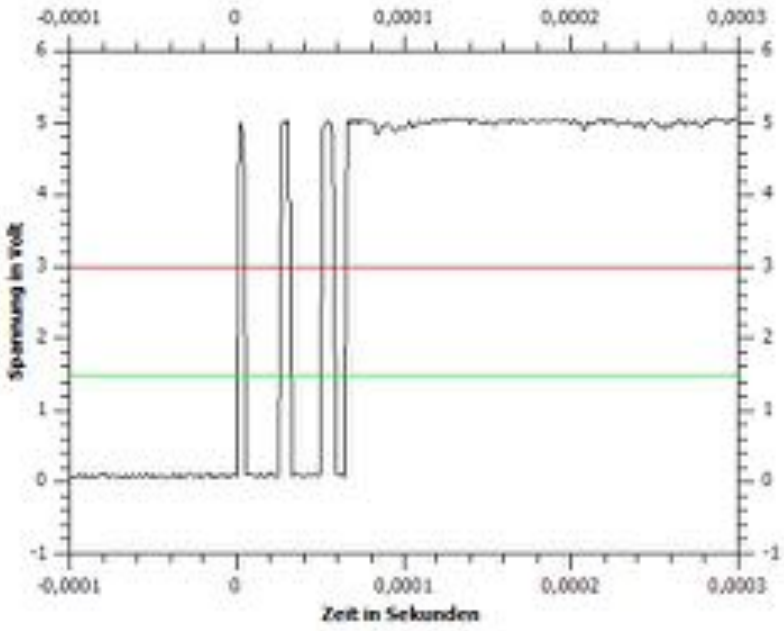

Abb2: Spannungsverlauf beim drücken des Knopfes (Schaltung nach Abb1)

Prellens entscheidet auch über den Widerstand R1, da dieser die Ladegeschwindigkeit des Kondensators angibt. Mithilfe des Widerstandes R2 kann entschieden werden, wie schnell der Kondensator wieder entladen wird. Auch der Zeitraum des Entladens sollte nicht zu klein sein, denn das Prellen kann auch beim loslassen des Knopfes auftreten. In Abb3 sieht man nun wie die Spannung, unter Verwendung der neuen Schaltung, langsam ansteigt. Nun verschwindet das Prellen komplett und die Schwellspannung wird nur noch einmal überschritten. Somit reagiert der Arduino nur noch einmal.

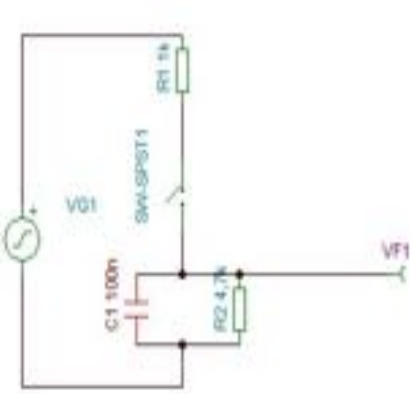

Abb3: neue Beschaltung des Knopfes

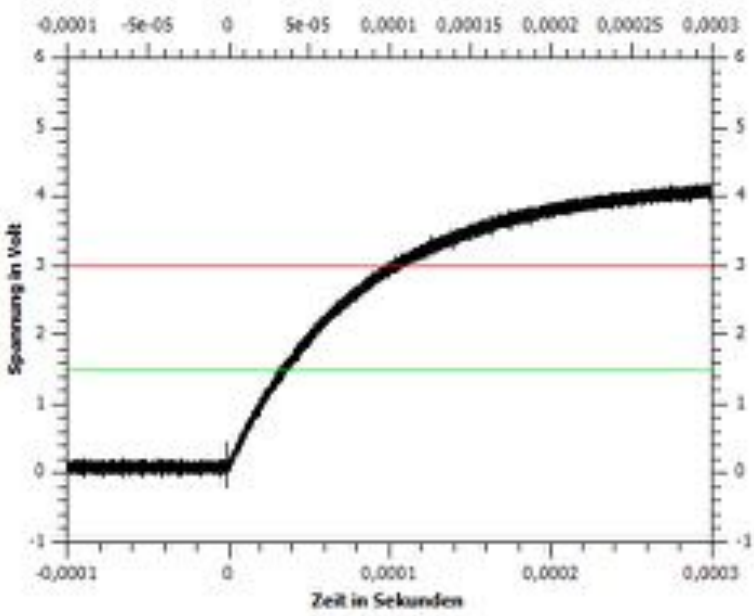

Abb4: Spannungsverlauf beim drücken des Knopfes (Schaltung nach Abb3)

Bei dem zweiten Problem wurde das Signal des DCF77-Empfängers ebenfalls mit einem Oszilloskop überprüft, um entscheiden zu können ob die empfangenen Signale nicht in Ordnung sind, oder der Arduino die Signale falsch interpretiert. Dabei stellte sich heraus, dass die empfangenen Signale nur sehr schwer zu erkennen waren. Es ist aufgefallen, dass die Signale besser wurden, wenn man den Empfänger weiter weg von dem Arduino legte. Auch die Computermaus die in der Nähe lag hatte sehr starken Einfluss auf die Signalqualität. Anscheinend erzeugen der Arduino und die Computermaus sehr starke Störstrahlung, weshalb die Signale so schwer zu interpretieren waren. Um diese Störstrahlung zu umgehen, wurde der Empfänger in das gleiche Gehäuse wie der Temperatursensor gebaut. Da dieses Gehäuse sich außerhalb des Hauses befindet, hat der Empfänger keine Probleme mehr die Signale zu empfangen. (10.1 Bild 3)

9 Fazit

Nachdem alle aufgetretenen Probleme behoben werden konnten, kann sich der Nutzer nun temperaturabhängig wecken lassen. Durch die Einbindung eines Feuchtigkeitssensors und eines kapazitiven Regensensors könnte der Funktionsumfang des Weckers sinnvoll erweitert werden. Auch das Umrüsten auf eine kabellose Verbindung, anstelle eines Kabels, das nach draußen gelegt werden muss, wäre eine lohnenswerte Weiterentwicklung.

10 Anhang

10.1 Bilder des Weckers

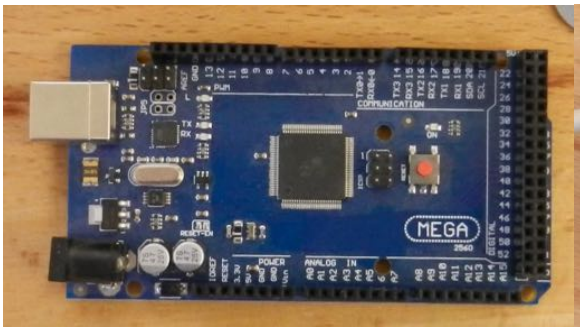

Bild1: der Arduino

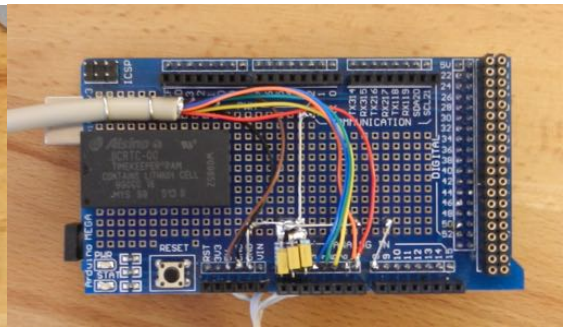

Bild2: Arduino mit aufgestecktem Shield

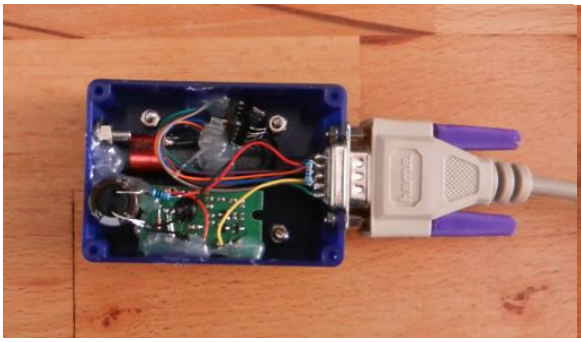

Bild3: Temperatursensor und DCF77-Empfänger

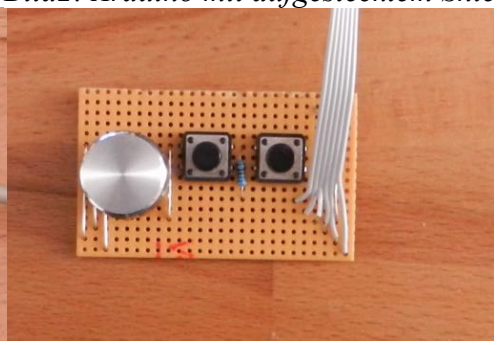

Bild4: Bedienelemente

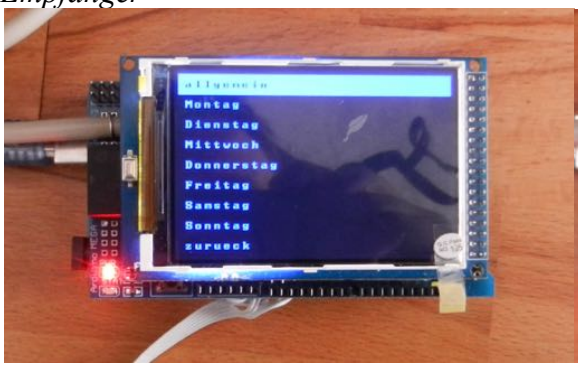

Bild5: Menü

Bild6: allgemeine Einstellungsmöglichkeiten

Bild7: Einstellungsmöglichkeiten eines Wochentages

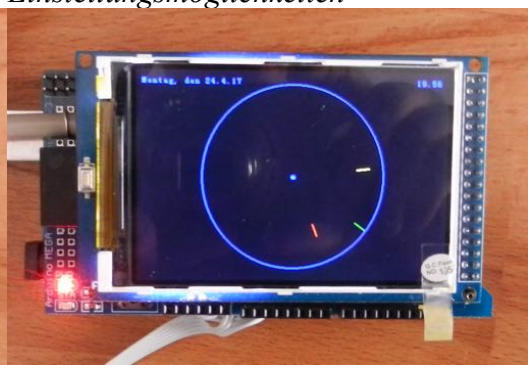

Bild8: Uhrzeit, Datum und aktuelle Temperatur

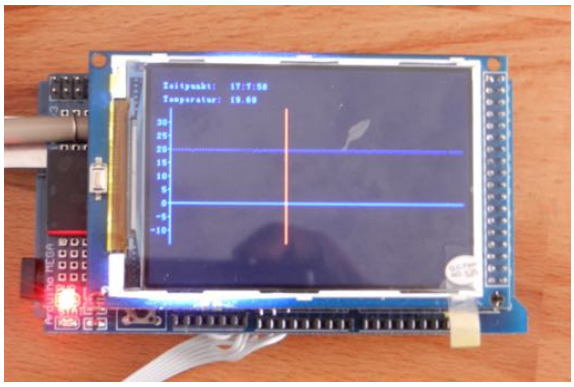

Bild9: Temperaturgraph mit Cursor

10.2 Hilfsmittel

Es wurden folgende Hilfsmittel genutzt:

- alle nötigen Werkzeuge zum löten
- Oszilloskop
- Voltmeter
- QtiPlot http://www.qtiplot.com/download.html – 26.04.2017
- LTspice XVII http://de.freedownloadmanager.org/Windows-PC/LTspice-XVII-KOSTENLOS.html – 26.04.2017
- Arduino IDE https://www.arduino.cc/en/Main/Software – 26.04.2017
- Library „UTFT“http://www.rinkydinkelectronics.com/library.php?id=51 – 26.04.2017
- Library „UTFT_CTE“ https://code.google.com/archive/p/cte-lcd-modules-arduino-library/downloads – 26.04.2017

11. Literaturverzeichnis

1. Analog Devices, Datenblatt ADT7310:

http://www.analog.com/media/en/technical-documentation/data-sheets/ADT7310.pdf – 11.04.2017

2. Arduino, Arduino Mega 2560 Schema:

https://www.arduino.cc/en/uploads/Main/arduino-mega2560_R3-sch.pdf – 11.04.2017

3. Arduino, „constants“:

https://www.arduino.cc/en/Reference/Constants – 11.04.2017

4. Conrad, Datenblatt Drehimpulsgeber:

http://www.produktinfo.conrad.com/datenblaetter/700000-724999/700708-da-01-en-ENCODER_STEC12E08.pdf – 11.04.2017

5. ELV Journal, „Elektronikwissen zu: Das DCF77-Zeitsignal“:

https://www.elv.de/elektronikwissen/das-dcf77-zeitsignal.html – 11.04.2017

6. STMicroelectronics, Datenblatt M41T00CAP:

http://www.st.com/content/ccc/resource/technical/document/datasheet/46/d2/5a/22/b8/30/42/0f/CD00189496.pdf/files/CD00189496.pdf/jcr:content/translations/en.CD00189496.pdf -11.04.2017

7. UTFT – Manual: https://code.google.com/archive/p/cte-lcd-modules-arduino-library/downloads – 26.04.2017

12 Erklärung der eigenständigen Verfassung

Ich versichere, dass ich die vorliegende Arbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

_____, den _____
(Ort) (Datum)

_____