2.3 Programmierung

Es gibt mehrere Vorgehensweisen für die man sich entscheiden kann, um den Arduino zu Programmieren. Wir haben uns für die gebräuchlichste und einfachste Methode entschieden: die Benutzung der Arduino-IDE. Die Arduino-IDE ist sehr spartanisch, wenn man sie mit Atmel-Studio oder Eclipse vergleicht, dafür ist sie sehr einfach gehalten und man kommt sofort klar. Dort muß man eigendlich nur drei Sachen beherrschen: Überpüfen und Hochladen, Speichern und unter Werkzeuge-Board das richtige Arduino-Board auswählen. Wenn man das verstanden hat, kann man losprogrammieren und testen. Ein weiterer Vorteil, man muß nichts Einrichten, es funktioniert alles „out of the box“.

2.3.1 die Hauptdatei

Ein kleiner Exkurs:

In C ist die Haupdatei immer die main. Beim Arduino hat man gewöhnlicherweise keinen Zugriff auf die „echte“ main. Es gibt sie unter:

$HOME/arduino/hardware/arduino/avr/cores/arduino/main.cpp

Aber diese main-Datei ist schon vorgegeben und man sollte sie nicht verändern. Die main.cpp enthält zwei Funtionen „setup()“ und „loop()“. Legt man in der Arduino-IDE nun sein Projekt an, bei uns heißt es „abschlussprojekt.ino“, sind beim ersten Start nur diese zwei Funtionen zu sehen, ansonsten ist die Datei leer. Es müssen auch beide Funtionsnamen im eigenen Projekt stehen, sonst bekommt man vom Debugger später eine Fehlermeldung. Die Befehle die im Setup() stehen, werden einmalig ausgeführt und die im loop() stehen, werden in einer Endlosschleife ausgeführt (bei uns in MCT haben wir immer while(1) genommen, welches die gleiche Funktion erfüllte).

Abschlussprojekt.ino:

Was passiert in dieser Datei.

Als erstes werden externe Librarys und ausgelagerte Funktionen includiert. Hier ist zu erwähnen, das wir für unseren Temperatursensor und i2c-LCD erst die richtige Biliothek herunterladen mußten. Unter Windows war das einfacher, da man unter Sketch-Bibliothek einbinden-Bibliotheken alles bequem per wizard erledigen konnten, da ich aber unter Linux arbeite mußte ich das Projekt ausfindig machen, die Biliothek bei Github herunterladen und manuell in den Ordner $HOME/Arduino/library/ kopieren. Aber dann war alles richtig eingerichtet und man konnte unter Linux und Windows gleichermaßen arbeiten.

Danach wurden Variablen deklariert und einige auch initialisiert. Dabei haben wir auf sprechende Namen geachtet. Sprechende Namen bedeutet, das man schon durch lesen des Variablennamen erkennen kann, was die Variable für eine Aufgabe/Funktion hat. Man sollte sich für die sprechenden Namen Zeit nehmen und alles genau durchdenken, dadurch spart man sich später Arbeit, weil man nicht wieder alles ändern muß. Wir haben die ersten Codezeilen zuerst auf Papier geschrieben, dort konnte man streichen und neuschreiben. Sprechende Namen sorgen auch dafür, dass man seinen Programmcode schon beim Lesen versteht und sich Kommentare sparen kann, zumindest im Idealfall.

Dann kämen die Funtionsdefinitionen. Ich schreibe kämen, weil dies am Anfang der Fall war, aber als wir unser Projekt modularisiert haben, sind alle selbstgeschriebenen Funktionen der Übersichthalber in extra Dateien geschrieben worden, welche jetzt am Anfang der Datei includiert werden.

Nun kommt der Setup, die erste eigentliche Arduino-Funktion. Hier wird das Display initialisiert und eine erste LCD-Ausgabe gemacht. Der Temperatursensor wird initialisiert. Dann werden die Pins konfiguriert. Dazu verwendet man die Methode pinMode() und über das Argument INPUT oder OUTPUT setzt man sie als Ein- oder Ausgang. Für den Taster und den REED-Kontakt haben wir sogar die internen Pullup-Widerstände aktiviert und uns so eine externe Beschaltung gespart. Der Timer für die Zeiten wird gesetzt. Hierzu wurde die Funktion millis() genutzt. Wenn man das Arduino-Board einschaltet startet auch ein Timer, der einfach die Zeit in Millisekunden zählt. Durch Verwendung von millis(), kann man den Wert abrufen bzw in eine Variable schreiben und den Wert verarbeiten. Durch genaues Lesen der Arduino-Referenz erfährt man, dass die Funktion nur 50 Tage zählt und dann wieder auf Null zurück geht. Das muß man im Hinterkopf behalten.

Zum Schluß kommt noch eine LCD-Ausgabe „...Complete“. Das ist gleichzeitig eine Überprüfung, ob alles korrekt im Setup durchlaufen wurde. Erscheint „Complete“ nicht, hängt irrgendwas im Setup fest und man muß den Fehler suchen.

Jetzt kommt die zweite Arduino-Funktion „loop“. Hier werden die Aufgaben programmiert, die endlos abgearbeitet werden sollen. Zuerst werden die Potentiometer eingelesen, die an den analogen Ports angeschlossen sind. Mit ihnen stellt man die Schwellen ein. Näheres unter 2.3.2. Nun wird der Tasterzustand eingelesen. Wird der Taster gedrückt, ändert sich das Verhältnis zwischen voriger_taster und aktueller_taster und die Nummer des Displays schaltet zwischen 0 und 1 hin und her. Als nächstes kommt die Display-Ausgabe. Wenn 2s verstrichen sind (dt_display_ms = 2000) geht man in die Anweisung und ändert die Display-Anzeige. In der nächsten Anweisung liest man die Messwerte der Sensoren ein (Temperatur, Erdfeuchte und Luftfeuchte). Danach wird die Raumtemperatur mit der Schwellentemperatur verglichen, ist die Raumtemperatur zu niedrig, geht die Heizung an. Ist die Temperatur zu Hoch geht die Heizung aus. Das gleiche Verfahren wird beim Lütfer angewand, bloß umgekehrt. Es ist aber auf eine gewisse Totzeit zwischen Heizen und Lüften zu achten, da sonst der Lüfter gegen die Heizung arbeitet. Das wäre kontraproduktiv und Energietechnisch gesehen eine Katastrophe. Danach wird der Helligkeitsensor ausgewehrtet. Ist es draußen Dunkel gehen die LEDs an, sonst sind sie aus.

Zum Schluß werden die Pumpenfunktionen aufgerufen. Es sind zwei, ein für die Bewässerung von oben und eine für die Bewässerung von unten. Diese Pumpkreise arbeiten unabhängig von einander.

2.3.2 Die Unterprogramme

Um den Code übersichtlicher zu gestalten, haben wir unser Programm modularisiert. Im folgenden sind die einzelnen ausgelagerten Funktionen aufgeführt:

makros.c
make_string.c
pumpe_1.c
pumpe_2.c
tuer_zu.c
update_lcd.c
update_limits.c
update_messwerte.c

Wir möchten gleich zu Beginn erwähnen, das man normaler weise in C zwei Dateien pro ausgelagerte Funktion hat. Eine C-Datei in der der Code steht, die Funktionsdefinition und eine Header-Datei in der der Prototyp steht, die Deklaration. Aber leider ist es uns nicht gelungen, die Arduino-IDE zu überzeugen, dass sie das auch möchte. Also mußten wir es entgegen dem C-Standard, alles in eine C-Datei schreiben und diese includieren.

Auch hier haben wir wieder auf sprechende Dateinamen geachtet, damit sofort klar wird, was in der Datei für eine Funktion steckt.

In der Datei makros.c, stehen alle Makros drin, welche im Wesendlichen die Pinbelegung des Arduino durch Makronamen ersetzt werden, die dann wiederum im Code stehen. Dadurch erkennt man sofort, welche Hardeware an welchem Pin angeschlossen werden muss.

Die Datei make_string.c macht etwas sehr interessantes. Sie misst die Länge des Strings, der auf unserem Display ausgegeben wird. Ist der String kürzer als 20 Zeichen, wird hinten solange ein Leerzeichen angefügt, bis die 20 erreicht ist. Somit können wir dynamisch alte Zeichen löschen, wenn das Display aktualisiert wird.

Pumpe_1.c und Pumpe_2.c sind vom Aufbau gleich. Der Ablauf des Pumpens wird hier durch einen Zustandsautomaten (case-Struktur) gelöst. Das Problem ist, die Einsickerzeit zu berücksichtigen, denn das Wasser braucht Zeit um sich im Boden zu verteilen. Im ersten Zustand (zustand_warten_zu_trocken) wird geprüft ob die Tür zu ist und die Erde feucht oder trocken ist. Ist die Erde feucht, springt man raus aus der case und kommt im nächsten Pollingzyklus wieder im ersten Zustand an. Ist es diesmal trocken, springt die Pumpe an und der nächste Zustand (zustand_pumpt) wird gesetzt. Hier wird geprüft, ob die Zeit die die Pumpe pumpen soll, abgelaufen ist (dt_pumpendauer_1_ms = 5000;) oder ob die Tür in der Zwischenzeit aufgemacht wurde. In beiden Fällen wird die Pumpe ausgeschaltet und der nächste Zustand (zustand_warten_durchfeuchtung) wird gesetzt. In diesem und letzten Zustand wird gewartet, dass sich das Wasser im Boden richtig verteilt. Diese Zeit kann man im Code einstellen und wir haben hier 10s gewählt (dt_warten_durchfeuchtung_1_ms = 10000).

Die Funktion tuer_zu.c wertet den REED-Kontakt aus. Ist der Kontakt geschlossen wird der Arduino-Pin „REEDKONTAKT“ gegen Masse gezogen und die Funktion liefert ein „true“ zurück, wenn nicht ein „false“. Das wird dann in den Pumpenfunktionen ausgewertet. Für den Arduino-Pin „REEDKONTAKT“ verwenden wir an dieser Stelle den internen Pullup-Widerstand, um einen definierten Pegel zu haben (und auch nicht mit dem Brumfinger zu schalten).

Die letzten drei Dateien, update_lcd.c, update_limits.c und update_messwerte.c, machen im Prinzip das Gleiche. Sie werden in jedem Programmzyklus abgefragt. Update_limits wird immer abgefragt und update_messwerte und update_lcd nur, wenn eine gewisse Zeit abgelaufen ist. Interessant war die map-Funktion. Damit wurden die ADC-Werte des Arduino in lesbare Werte in Grad Celsius-Werte umgewandelt (gemappt). Die zweite interessante und wichtige Sache ist die Verwendung von „extern“ bei den Variablennamen. Durch Verwendung des Deklarationsspezifizierers „extern“ kann man gleiche Variablennamen mehrfach verwenden.

Quellen:
https://www.mikrocontroller.net/articles/Umstieg_von_Arduino_auf_AVR
https://www.arduino.cc/reference/de/language/functions/time/millis/
„Mikrocontroller verstehen und Anwenden“ von Clemes Valens