

**Debreceni Egyetem
Informatikai Kar**

**Baromfi inkubátor fejlesztése PIC
mikroprocesszorral**

**Témavezető: Dr. Végh
János
Beosztása: Egyetemi
Tanár**

**Készítette: Forgács
Richárd
Programtervező
Informatikus M.Sc.**

Debrecen 2014

Tartalomjegyzék

Bevezetés	3
1. A baromfi keltetésről általában.....	6
1.2 Optimális keltetési hőmérséklet	9
1.3 Optimális páratartalom a keltetés alatt	10
1.4 Forgatás és permetezés keltetés alatt	10
2. A rendszerrel szemben felállított követelmények	12
3. A hardver	14
3.1 A PIC18F4523	21
3.2 A PIC24FJ64GA002	22
4. A rendszer funkcionális bemutatása	27
5. A rendszer programszintű felépítése	29
6.1 Az inicializálás.....	32
6.2 A főciklus	33
7. Hőmérséklet	36
7.1 Hőmérséklet érzékelés	36
7.2 A hőmérséklet adatok feldolgozása	40
7.3 A hőmérséklet vezérlése	40
8. PID	42
9. Páratartalom	47
9.1 Páratartalom érzékelés	48
9.2 A páratartalom adatok feldolgozása	49
9.3 A páratartalom vezérlése	49
10. Forgatás.....	53
11. Nedvesítés.....	55
12. A rendszer monitorozása	56
13. Befejezés, összefoglalás, önértékelés	59
Mellékletek:.....	60
Irodalomjegyzék	61

Bevezetés

Szakdolgozatom témájának kiválasztásában több tényező is közre játszott. Elsősorban gyermekkori tapasztalataim, érdeklődésem a baromfitenyésztés iránt. Mindig kíváncsi voltam és erős érdeklődést mutattam a kelési folyamat részletes megismerése felé. Amikor a baromfi elül tojásain és addig fel sem kel, amíg kiscsibéit világra nem hozza, a természet egyik legszebb és legérdekesebb folyamatai közé tartozik. A tojások hőmérsékletének, páratartalmának szabályozása, a tojások napi rendszerességű forgatása, rossz tojások, elpusztult embriók kiválogatása a tyúkanyó által egy olyan folyamat, amely egy mérnök figyelmét mindenképp felkelti. Mivel az elkotlási folyamatot nehéz előidézni, rávenni a baromfit, hogy a szaporodási ösztöneire hallgatva a tojásokon maradjon és rajtuk ülve világra hozza kicsinyeit. Az emberiségség táplálékigénye rohamosan növekszik, egyre több és több élelmiszerre van szükség, amely természetesen érinti a baromfitartást és tenyésztést is. A hús előállítása ráadásul egy egész éves folyamatot igényel, nem elégséges az, hogy akkor keljenek ki újszülött csibék, baromfik, amikor az anya kotlós elül. A mai kor megigényli a folyamatos keltetést és nevelést, ezért a keltetési folyamatot megpróbálják az emberek lemásolni, lemodellezni. A modellezés során rengeteg apróságra figyelniünk kell, ami a baromfinak természetesen adódik. A legapróbb dolgok is befolyásolhatják a kelési százalékot. Ez az egyik legfontosabb mutató szám, amellyel mérni tudjuk rendszerünk hatékonyságát. Minél nagyobb kelési százalékot érünk el egy-egy ciklus során, rendszerünk annál jobb, hatékonyabb. Természetesen figyelembe kell vennünk a modellen kívül, a rendszerünk fogyasztását, a kelésre ráfordított energiát is. Mindenképp gazdaságosnak kell lennie, ha magának a baromfinak a hatékonyságát nem is tudjuk felülmúlni. Ezt a hatékonyságot bővebben kifejtem a dolgozatom releváns fejezetében.

A modellünkben a legfontosabb dolgok, amit figyelembe kell vennünk:

- Hőmérséklet szabályozása, állandó hőmérséklet biztosítása az embriók számára
- Páratartalom szabályozása, állandó páratartalom biztosítása a rendszeren belül
- Tojások forgatása rendszeresen, egy napon belül egy megadott számban
- Rendszer szellőztetése, oxigén biztosítása, szén-dioxid eltávolítása
- Tojások szelektálása, elpusztult embriók eltávolítása
- Tojások helyes fekvésének ellenőrzése és felügyelése

Rendszeren a dolgozat kereti között a keltetőgépet magát fogom érteni, erről fogok beszélni, írni ezen címszó alatt.

A fejlesztést önállóan kezdtem el, egy volt kollégám segítségével. A programozási feladatokat én végeztem el, az elektronikai kérdésekben nagyon sok segítséget kaptam Tőle. Törekedtünk a modellt minél jobban matematikai alapokra fektetni, de ahogy az az életben általában lenni szokott, sokszor a túlságosan bonyolult képletek helyett empirikusan állapítottunk meg dolgokat. Így sokkal gyorsabban eredményt értünk el, tudtam tovább haladni a fejlesztési munkával. Nem volt időm minden részletet hiánytalanul kidolgozni, de biztos vagyok benne, hogy a rendszer fejlődése nem áll meg egy adott ponton, a lehető legtökéletesebben szeretném a kelési folyamatot reprodukálni. Kutatásom során néhány irodalmi jegyzetet is segítségül hívtam, de erősen támaszkodtam a falusi hagyományokra és szokásokra is. Szülőfalumban, Erdőtelken rengeteg kelési folyamatnak szemtanúja voltam, mind a mai napig keltetek csibéket.

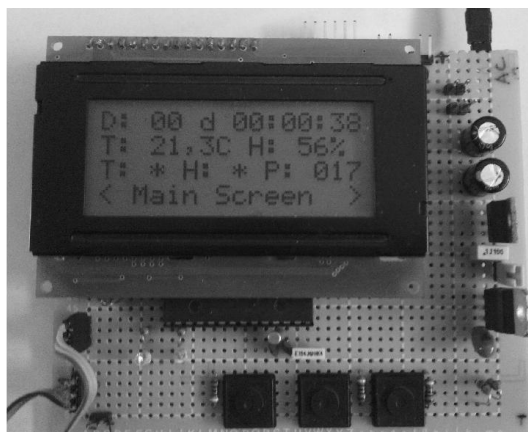
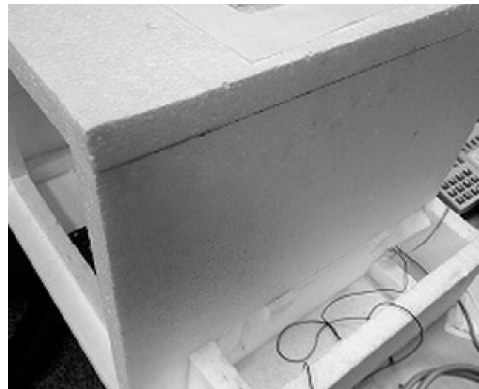
A rendszer több részből áll össze, minden egység külön-külön szerepkörrel. A dolgozat tárgyalásának első részében a rendszerrel szemben felállított követelményeket igyekszem feltárni, majd egyre mélyebben, a konkrét megvalósításokat fogom részletezni. Bár a dolgozatom témájaként elsősorban a szoftver oldal bemutatását tűztem ki célul, elengedhetetlen, hogy a hardver felépítését is tárgyaljam, annak működését és elvi felépítését. A szoftvert igyekeztem úgy megírni, hogy láthatólag elkülönüljenek az egy-egy funkcionális megvalósító részek.

A projekt célja a természet minél élet hűbb modellezése és mindez nagyon alacsony költségvetésen. A kivitelezés során a rendszer nagyon sok részegysége saját ötletből megvalósított egyedi megoldásokból született. Ilyen például a fűtőegység ellenálláshálóból felépített szerkezete vagy az optimális páratartalmat biztosítani hivatott nedvesség befűző alrendszer. Sokszor az elektronikai alkatrészek is a fiókból kerültek elő és annak ellenére, hogy a precíz számításaim miatt mást igényelt volna a rendszer, ráhagyatkoztam arra, ami éppen kéznél volt. Ez természetesen elhanyagolható hibákat és tévedéseket is hagy a rendszerben, de még olyan szinten tartja ezt, hogy az elhanyagolható a célunk szempontjából. Dolgozatomban igyekszem szétválasztani a hardver bemutatását a szoftver ismertetésétől és ezeket minél jobban szemléltetni grafikonokkal és ábrákkal. Sok helyen szeretném a konkrét kódot is bemutatni, hogy az olvasó jobban megértse hogyan is került implementálásra egy-egy funkcionális.

A projekt sikeres volt, kialakításra került egy olyan rendszer, amely a fentebb megjelölt modellt élethűen megvalósítja. Ahhoz, hogy megértsük a dolgozat egészét, összefoglalnám pár pontban mik voltak a konkrét célok, amelyeket megvalósítani kívántam ezen projekt keretein belül:

- hőmérséklet szabályzása, egy ponton tartása megadott időszakok alatt
- páratartalom szabályzása, megtartása adott időszakok alatt
- egy olyan közeg kialakítása, amelyben ez a két tényező maradéktalanul megvalósítható (megfelelő szigetelés, megfelelő helymennyiség)
- tálcák tojások számára
- szellőztetési lehetőség
- automatizált folyamatok (főleg hőmérséklet és páratartalom szabályzás)
- életvédelmi szempontok figyelembe vétele
- felhasználói követelmények figyelembe vétele (interaktivitás)
- a rendszer folyamatos monitorozásának lehetősége

Alább látható pár kép a megvalósított rendszerről:



1. ábra: Az elkészült rendszer

1. A baromfi keltetésről általában

A dolgozat célja, hogy bemutasson egy olyan automatizált eljárást, amellyel utánozzuk a keltetés természetes folyamatát. De anélkül, hogy alaposan meg ne értenénk a természetes folyamat minden alapvető tényezőjét, nem számíthatunk áttörő sikerre. Nagyon fontos biológiai folyamatok játszódnak le a keltetés során. Ennek a fejezetnek a célja, hogy megismertessem ezeket a tényezőket és a kikelésig lezajló folyamatokat.

A baromfi fajtájától függően eltérőek a kelési idők, a tojások formája és tömege. Amíg egy átlagos tyúk tojásának tömege 50-70 gramm között lehet, addig egy lúd tojása 155 grammot is elérheti. Ezt figyelembe kell vennünk a keltető tervezésnél: el kell, hogy férjen egy átlagos tyúktójas is, de ha ludat akarunk keltetni, akkor annak is el kell férnie rendszerünkben. A 2. ábra szemlélteti az egyes baromfifajok tojásának tulajdonságait.

Baromfifaj	A tojás tömege, g	Az alkotórészek aránya, %		
		szik	fehérje	héj és héjhártyák
Pulyka (nagy testű)	80—90	31—35	54—58	8,5—10,5
Tyúk	50—70	25—33	57—65	8,5—10,5
Gyöngytyúk	35—45	25—35	50—60	15
Fácán	29—32	30—32	52—55	9,0—10,5
Galamb	18	18—22	65—75	7—9
Fürj (tojótípus)	8—10	30—33	52—60	7—9
Lúd (szürke, májhasznú)	155	30—33	55—58	11—13
Lúd (fehér, húshasznú)	150	36	51,5	12,5
Házikacsa (pekingi)	92	33	57,5	9,5
Házikacsa (khaki campbell)	55—65	33—36	53—56	9,5—11
Pézsmakacsa	75—85	33—37	50—53	11—13

2. ábra: Egyes tojások tömege és az alkotórészek aránya

Forrás: Dr. Bogenfürst Ferenc, 2000

Az embriófejlődés azonnal elkezdődik a megtermékenyítés után. Miután a tojásrakás megtörténik nagyon fontos, hogy a tojást megfelelő környezetben tároljuk addig, amíg nem kerülnek be rendszerünkbe. A kelés sikerességét befolyásolja a tojások tárolása és azoknak a tojásrakás utáni kezelése. Ha túl gyorsan hűtjük le a tojásokat azonnal megáll az embriófejlődés, amely károsan hat a későbbi kelési folyamatra. Ugyanúgy érzékeny az embrió

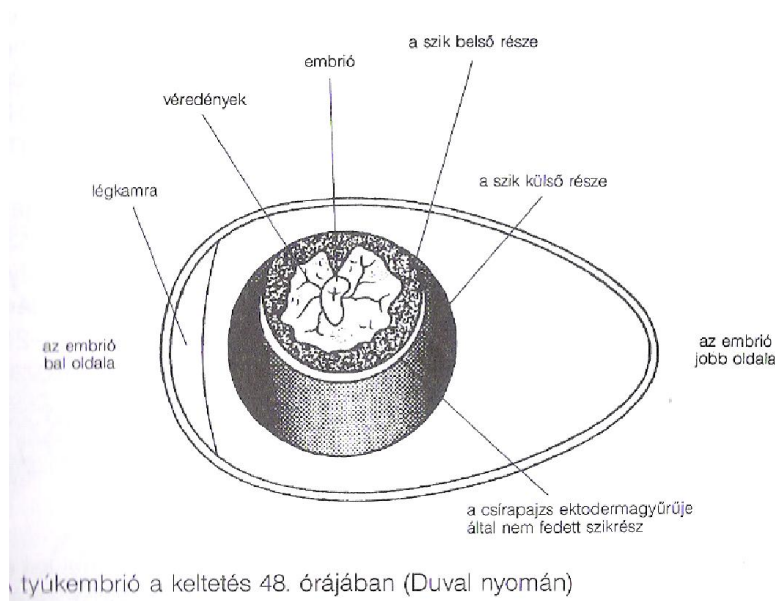
a magas hőmérsékletre. A tojás a megtojást követően lehűlik és amikor hőmérséklete 24°C alá csökken a fejlődés megáll, majd egy nyugalmi állapot következik.

A baromfi fajtája válogatja azt az alsó hőmérsékleti értéket, amely az alsó határa az embriófejlődéshez szükséges hőmérsékletnek. Egy átlagos értéket állapítottak meg vizsgálatok segítségével, amely az összes baromfifaj embriófejlődéséhez szükséges. Ez az érték 23,9 °C (Dr. Bogenfurst, 2000)

A keltetés során több szakaszt is megkülönböztethetünk. Rendszeremben én három szakaszra osztottam fel a kelési ciklust, de általánosan két szakaszt különböztetnek meg. Dr. Bogenfurst Ferenc például két szakaszt ismertet „Keltetés” című művében:

- a 8-9. napig tartó embrionális szakaszt
- az ezt követő magzati szakaszt

Az első szakasz a gyors fejlődés időszaka. Ekkor alakulnak ki a végtagok, valamennyi szerv és azok kezdeménye. Első hét után átvilágítással már láthatóak a fejlődés első jelei. Nagyon fontos szerepet játszik ez az átvilágítás. Azokat a tojásokat, amelyek megálltak a fejlődésben és nem látjuk az érhalózatot és végtagokat, eltávolítjuk a rendszerből. Ezt egyébként a tyúk is megteszi a saját tojásaival.



3. ábra: Az embrió a keltetés 48. órájában

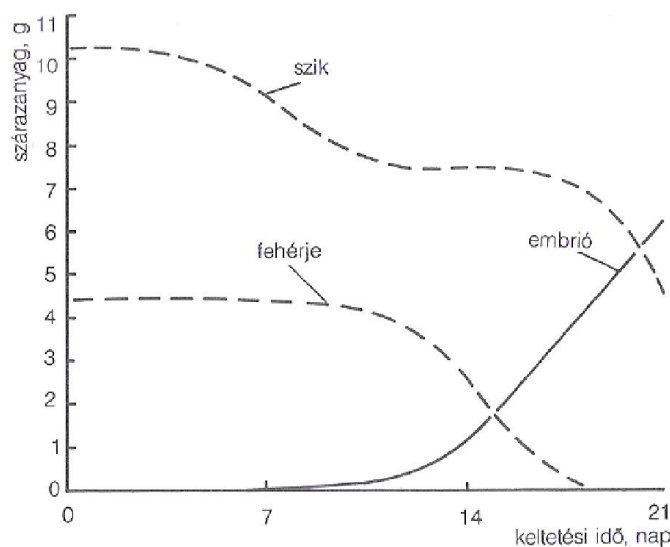
Forrás: Dr. Bogenfurst Ferenc, 1987

Az embrió fejlődése során folyamatosan alakul ki annak fekvési pozíciója, ezért a tojások forgatásánál nagyon ügyelni kell annak helyes kivitelezésére. A helytelen fekvési pozíció a keltetés során az embrió elpusztulásához vezethet. Általános bevett szokás, hogy a tojásokat

legfeljebb 45°-os szögben forgatjuk el, mivel az ennél nagyobb elforgatási szög akár a köldökzsinór elzáródásához is vezethet. Naponta többször is helyet cserélnek a tojások a kotlós alatt, folyamatosan forgatja azokat az anya tyúk. Szintén alapvető élettani fontosságú, hogy az embrió kapjon megfelelő mennyiségű oxigén utánpótlást. Mivel az embrió széndioxidot termel a légzése során, ezért a levegő cseréje a rendszerben alapvető szerepet kell, hogy játsszon. A légzés a tojás burkolatán keresztül történik. Szintén a tojás héján keresztül távozik a hő, amelyet az embrió termel. „A legkritikusabb környezeti feltétel nem minden baromfifajban egyforma. A pulyka pl. a hőmérsékletre a legérzékenyebb, a kacsza a relatív páratartalomra, a fácán a légmozgásra.” (Dr. Bogenfürst, 1987)

Ahhoz, hogy megfeleljünk az egyes baromfifajták igényeinek fontos, hogy a rendszerünk flexibilisen tudjon alkalmazkodni egy-egy ilyen igényhez. Ismernie kell egy-egy faj hőmérséklet, páratartalom és oxigén igényét, a napi forgatások minimális számát.

A tojásban található anyagok (fehérje, szik, zsírok) a fejlődés során folyamatosan elhasználódnak. Az 5.-ik ábrán láthatjuk, hogy az idő előrehaladtával és az embrió fejlettségének függvényében, hogyan használódnak el ezek a tápanyagok.



4. ábra: Az embrió, a szik és a fehérje szárazanyagának változása a tyúktojásban a keltetés alatt

Forrás: Romanoff, 1967

Az embrió a keltetés során bármelyik pillanatban elpusztulhat az elégtelen külső környezeti tényezők hatásai miatt. A fejlődés korai szakaszában a túl magas hőmérséklet, a helytelen forgatás, a rázkódás és a helytelen gázcsere (CO_2 , O_2) azok a tényezők, amelyek legnagyobb

mértékben vezethetnek az embrió elhalásához. A későbbi szakaszban az embrió fejlődésének visszamaradása, meggyengülése szintén ahhoz vezet, hogy az embrió nem képes áttérni a tüdőlélegzésre és világra jönni, kibújni a tojásból. Nagy erőfeszítést igényel az újszülött baromfitól tojásának feltörése és abból való kibújása. Ilyenkor a baromfit igen nagy stressz is éri.

A fejezet további részében szeretném részletezni a keltetés során alapvető szerepet játszó környezeti tényezők (hőmérséklet, páratartalom, stb.) optimális egyensúlyát.

1.2 Optimális keltetési hőmérséklet

Az embrió fejlődése kizárólag egy bizonyos hőmérséklettartományban tartható fenn. Helytelen hőmérséklet elnyújtja a kelési időt, gátolja a sejtek osztódását. Több forrás szerint is ez a hőmérséklet a 37°C és a 38°C közé esik. A természetben a kotlás alatt a tojások felszínének hőmérséklete meghaladhatja a 39°C-ot is, viszont a tojások belsejében átlagosan megmért hőmérséklet az 37,8°C. Mesterséges környezetben nem érintkezik a fűtőtest a tojások felszínével. A fűtőtest elhelyezhető a keltető több pontján is. Helyezhetjük a gépünk mennyezetére és aljára is. Ha az aljára helyezzük, számításba kell vennünk a meleg levegő fölfelé történő áramlását. Szokás ventillációt alkalmazni a hő egyenlő elosztására, elkerülve ezzel a nagy hőkülönbségeket a keltetőn belül. A keltetés első szakaszában a szükséges hőmérséklet a tojás belsejében 37.5°C-37.8°C között van. Ez a második szakaszban valamivel kevesebb, 37.0°C-37.5°C között mozog.

Ha túl magas a hőmérséklet az embrió túl gyorsan fog fejlődni, ami szintén olyan káros, mintha lassan fejlődne. Az 1.sz. táblázatban láthatjuk, hogy egyes hőmérsékleti fokokhoz milyen kelési időhossz tartozik.

Hőmérséklet:	38,9 °C	37,8 °C	37,2 °C	36,7 °C	35,6 °C
Kelési idő:	19,5 nap	20,2 nap	20,8 nap	21,6 nap	23,6 nap

1.sz. táblázat: Keltetési alatti hőmérséklethez tartozó kelési időtartam

Forrás: Dr. Bogenfürst Ferenc, 2000

A hőmérséklet ingadozására, amennyiben az rövid ideig tart, viszonylag ellenálló az embrió. Maximálisan 0.3°C-0.5°C fokos ingadozás megengedett, ez még nem veszélyezteti az embrió fejlődését. Előfordulhat mesterséges keltetési környezetben, hogy pl. áramszünet miatt a tojások lehűlnek és hosszú ideig nem is melegednek vissza az optimális hőmérsékletre. Ha ez

a lehűlt állapot viszonylag tartósan fennmarad, akkor az embrió elhalásához vezet.

1.3 Optimális páratartalom a keltetés alatt

A páratartalom több tényezőtől is függ a keltető belsejében. Egyrészt maguk a tojások is párologtatnak nedvességet a kelési folyamat során, másrészt a levegő hőmérséklete is befolyásolja a relatív páratartalmat. Az alacsony páratartalom a tojáshéjon található pórusok kiszáradásához, a túl magas nedvességtartalom ezen pórusok eltömődéséhez vezethet. Mindkét eset abnormális embriófejlődést okoz. A keltetés első szakaszában 65%-70% a megkívánt páratartalom, ami később 50%-55% csökken. Ennek a páratartalomnak a fenntartása a keltetőgép feladata.

1.4 Forgatás és permetezés keltetés alatt

A hőmérséklet és a páratartalom mellett még két alapvető tényezőt említek meg: a forgatás és a permetezés. A forgatás az embrió számára nélkülözhetetlen, élettani szempontok miatt fontos. Az embrió folyamatos úszkálása a tojáson belül lehetővé teszi, hogy hozzájusson a számára fontos tápanyagokhoz. A permetezés a tojáshéj puhulását eredményezi, amely segítséget nyújt a csibének a tojáshéj feltörésében.

Az 5. számú ábrán láthatjuk összefoglalva azokat a legfontosabb technológiai tényezőket, amelyet megkíván egy baromfi embrió a fejlődésének egyes szakaszaiban.

A keltetés napja	Hőmérséklet		Páratartalom			F o r g a t á s	Hűtés gépen kívül 20 °C-os teremhőmérsékleten	Permetezés gépen kívül 40 °C-os vízzel	Szellőzőnyílás Petersime —168 típusú keltetőgépre
	°F	°C	°F	°C	%				
Előkeltetés									
1									
2			90,9	32,7	70				0
3							nincs	nincs	
4									0,5
5									
6									
7	100	37,8							1,0
8									
9			84,0	28,9	52				
10									
11							naponta egyszer 10—15 perc, emelkedő tartammal		1,5
12									
13									
14									
15									
16									
17									
18	99,5	37,5					naponta egyszer 15—20 (emelkedő) perc	minden hűtés után harmatszerűen	2,0
19									
20									
21									
22									
23	99,0	37,2					naponta egyszer 20—25 perc a pattogzás kezdetéig		2,5
24									
25 Átrakás									
Bújtatás									
26	99,0	37,2	88*	31,1	66				
27			91,4—	33,0—	75—				
28			92,3**	33,5	78				3,0

5. ábra: A baromfi keltetésének legfontosabb technológiai előírásai

Forrás: Dr. Bogenfürst Ferenc, 2000

2. A rendszerrel szemben felállított követelmények

A dolgozatban használt rövidítéseket az 1.sz. melléklet tartalmazza.

Ha egy mondatban szeretném megfogalmazni, hogy mi a célja a rendszernek, akkor a következőt mondanám: **a rendszerbe behelyezett tojások keltetése.**

Ahogy azt már egy előző fejezetben tárgyaltam, a keltetés során rengeteg környezeti tényezőt figyelembe kell venni és azt a lehető legpontosabban lemásolni a természetről és megvalósítani azt a rendszerünkben. A legfontosabb dolgok, amelyeket mindenképp meg kell tudnia valósítani rendszerünknek:

- a keltetéshez megadott optimális hőmérséklet fenntartása minden körülmények között
- a hőmérséklet ingadozás minimalizálása, lehető legkevesebb kilengéssel való működés
- páratartalom megfelelő szinten tartása
- levegőcsere biztosítása, oxigén bejuttatása és széndioxid kijuttatása a rendszerből
- napi háromszori forgatása a tojásoknak
- az utolsó kelési stádiumban lévő tojások nedvesítése, permetezése

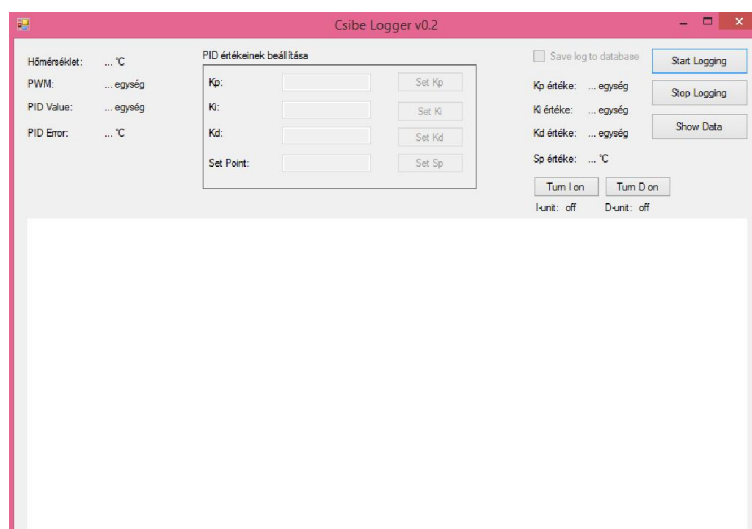
A fentebb említett pontokból igyekeztem minél többet megvalósítani, de a feladat komplexitása és az idő hiánya miatt ez csak részben valósult meg. A tojások forgatását nem sikerült automatizálni eddig a pontig, a kezdeti tervek viszont rendelkezésre állnak. Szintén nem került megvalósításra a permetezés sem, ez is túl komplex feladatnak bizonyult ahhoz, hogy ebben az időkeretben megvalósításra kerüljön. Ám ezeken kívül minden mást sikerült megvalósítani és ezeknek lépéseiről és megoldásairól részletesen írni fogok dolgozatomban.

Egyéb követelmények is felmerülnek egy ilyen rendszer tervezésekor. Nem szabad figyelmen kívül hagyni, hogy a fűtőegység 230 V-ról üzemel, amely életvédelmi szempontból kritikus. A teljes szigetelés és galvanizálás elsődleges szempontok között szerepelt.

Mindezekén túl gondolni kellett a felhasználóra is. A rendszer kezelhetősége, konfigurálhatósága és működése kiemelkedően fontos a felhasználó szempontjából. Igényeinek megfelelően be kell tudnia állítani a rendszert attól függően, hogy épp milyen baromfit szeretne keltetni.

A rendszer jelenlegi állapotáról mindig visszajelzést kell adnunk a felhasználónak és az esetleges hibákról is. Ehhez egy LCD kijelzőt helyeztem a rendszerbe, amelyen keresztül a felhasználó követni tudja a rendszerben zajló folyamatokat, a rendszer állapotáról információt

kap. Ezen kívül (talán még komolyabb eszköz) írtam egy PC oldali kezelő felületet a rendszerhez, amely soros porton keresztül lép kapcsolatba a keltetővel és azon keresztül kapunk nagyon látványos módon információkat, sőt innen nagyon könnyen tudjuk vezérelni a rendszert, működését befolyásolni. A PC oldali programról még fogok egy külön fejezetben beszélni és annak alapvető funkcionálisait és a kommunikációs csomagot bemutatni.



6. ábra: A PC oldali program

A vázat polisztirolból alkottam meg, amely köztudott, hogy nagyon jó hőszigeteléssel rendelkezik. Számomra is bebizonyosodott, hogy nagyon jó a hő tartása és ezzel együtt nagyon könnyen megmunkálható.

3. A hardver

Rendszeremben két mikrokontroller is található, egy PIC18F4523 és egy PIC24FJ64GA002. Alapvetően azért döntöttem úgy, hogy két egységet is fogok használni, mert szerettem volna megtanulni és kitapasztalni a két egység közötti kommunikációt.

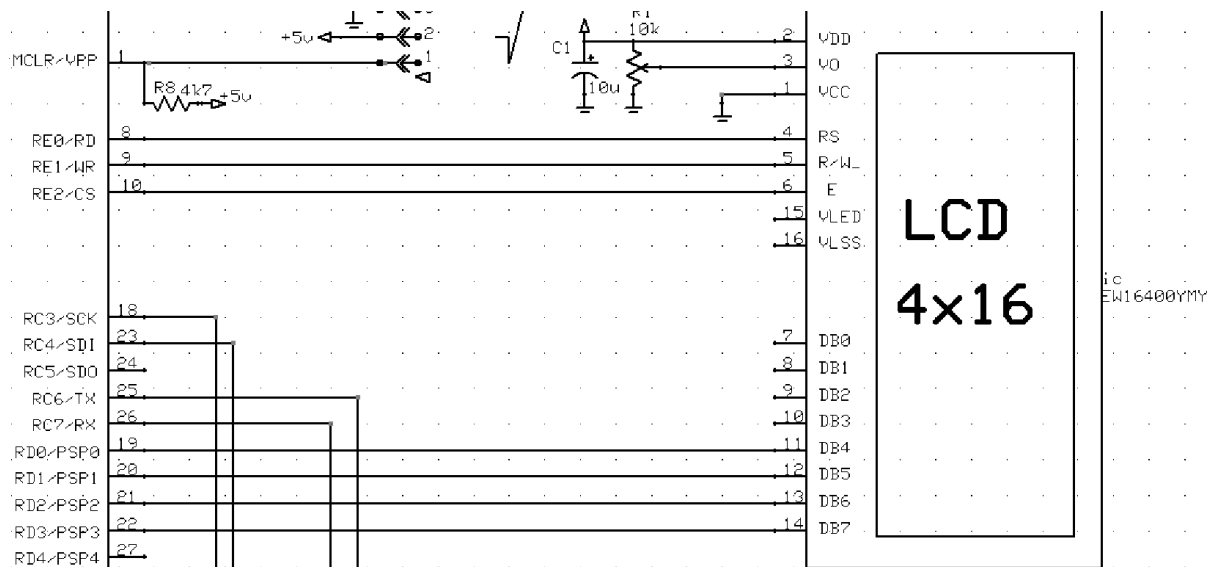
A kommunikáció I²C buszon történik. A PIC24-es feladata csupán annyi, hogy a PIC18-tól kapott PWM értéket fogadja, majd azt feldolgozva vezérelje a fűtőtestet PWM jellel. A PWM vezérlőjelem, amelyet a PIC18-tól küldök [0-200] tartományba esik. Arról, hogy miért ezt az intervallumot választottam egy későbbi fejezetben fogok bővebben írni.

A másik funkcionalitás, amit megvalósít a PIC24 azaz úgynevezett RTCC, valós idejű óra. Ez az egység tartja nyilván a bekapcsolástól eltelt időt. Ez alapján tudja a PIC18, hogy a keltetés éppen melyik fázisában van. Három fázisra bontottam fel a keltetési folyamat, ezek pedig eltérnek paramétereikben egymástól. A PWM küldésként valósul meg, kézfogás nélkül. Az idő lekérdezésként kerül a PIC18 oldalára.

A PIC18-as egységünk lényegében a lelke a rendszernek. Ezen a mikrokontrolleren valósul meg a lényegi vezérlés és adatgyűjtés. A PIC18-as oldaláról a PIC24 is csak egy funkciót megvalósító modul.

A PIC18-hoz négy lábon csatlakozik az LCD kijelző. A kijelző egy 4x16 soros LCD, amelyet a HD44780 kompatibilis LCD vezérlő hajt. A kommunikáció ezzel a részegységgel történik meg egy megadott algoritmus szerint. Nagyon fontos az időzítés a kommunikáció során, főleg az inicializálásnál. Lehetséges nyolc és négy lábon is kommunikálni a HD44780-assal, a különbség annyi, hogy négy lábaskor két részletben kell elküldenünk az egy bájtnyi adatunkat. Mivel sok funkciót terveztem a PIC18-asnak, ezért a lábakkal szerettem volna spórolni. Így a négy lábaskor mellett döntöttem. Az LCD mikrovezérlőjének inicializálása egy igen bonyolult algoritmus szerint történik. Ennek megvalósításával nagyon sokat dolgoztam, először assemblyben, később pedig C-ben is megírtam az inicializáló eljárást. Talán az egész rendszer programjában itt a legfontosabb az időzítés. A DHT11 páratartalom mérő is nagyon érzékeny a pontos időzítésre, de ha lehet ilyet mondani, az LCD létfontosságú egysége a rendszernek. Persze lehet nélkül is működtetni a rendszert, de semmilyen számunkra érthető visszajelzést nem kapnánk a rendszertől.

Az LCD bekötését az alábbi ábra mutatja:



7. ábra: LCD bekötési rajza

Amint láthatjuk egy potméter került az LCD oldalán lévő VO lábra, amellyel tudjuk szabályozni az LCD-n megjelenő szöveg fényét, erősségét. Az RS láb állapot adja meg, hogy a sínen lévő adat az parancs vagy karakteradat. Ha ennek állapota 0, akkor parancsként értelmezi az adatot a HD44780, egyébként pedig karakteradatként. Az E láb az úgynevezett enable láb, amellyel azt mondjuk meg az LCD vezérlőjének, hogy az adatok rendelkezésre állnak a sínen és értelmezhet azokat. Amint látjuk a négy felső (DB4-DB7) portra lett kötve az adattovábbításra kijelölt négy lábunk. Az adat ilyenkor úgy értelmezendő, hogy az első bit a DB7-es portra kerül, a nyolcadik bit pedig a DB0-ra. Lényegében a vezérlés abból áll, hogy az adatot ráhelyezzük a sínre, megadjuk, hogy parancsot továbbítunk és várunk egy megfelelő időtartamot, majd enable lábat 1-es állapotba helyezzük. Minden más a késleltetés pontosságán múlik.

Az inicializáló eljárást röviden összefoglalom az alábbi pontokban:

- Minimum 15 ms –ot várjunk az áram alá helyezés után
- Előzetesen minden portot, amely az LCD-hez van kötve állítsunk kimeneti portnak
- Állítsuk úgy a lábak állapotát, hogy hexa 0x03 állapotot tükrözzenek a lábak
- Adjuk ki az *enable* jelet, majd várjunk 7 ms-t
- Ezt háromszor ismételjük meg. Ez a parancs egyébként 8 bitesre állítja a sít, aminek különösebb jelentősége nincs, csupán egy ajánlás, hogy így kezdjük. Sajnos arról nem találtam információt a fejlesztés során, hogy miért ez az előírt protokoll, a HD44780 vezérlő dokumentációja erről konkrét információt nem tartalmaz

- Most küldjünk ki egy 0x02-es parancsot, amely 4 bitesre módra állítja be a vezérlőt
- Ettől a ponttól kezdve a bájtejainkat két részben kell elküldenünk
- Így most előbb küldjük ki a 0x02 jelet, majd a 0x08 jelet. Ezzel a betűtípust 5*8-asra és két sorosra állítottuk
- Következő parancs két részletben: 0x00, majd 0x08, amely kikapcsolja a kijelzőt
- Ezután töröljük a kijelző tartalmát: 0x00, majd 0x01
- Innen már csak finomhangolások vannak, amellyel magunk számára testre szabhatjuk, hogyan szeretnénk megjeleníteni az általunk küldött adatot. Ezt nem részletezném tovább

Az adatok átvitele hasonlóan történik, mint a parancsok átvitele, csupán annyi különbséggel, hogy az RS lába 1-es állapotba kerül. Ilyenkor a karakter átkerül a LCD vezérlő memóriájába. Nagyon fontos, hogy megadjuk melyik sorba akarjuk az adott karaktert írni. Ezt persze parancsként kell továbbítanunk. Az hogy melyik karakter melyik sorba kerül úgy kerül meghatározásra, hogy az adott karakterszám elérte-e az egy sorban elhelyezhető karakterek számát. Erre írtam egy eljárást, amelyet a 8. számú ábrán láthatunk.

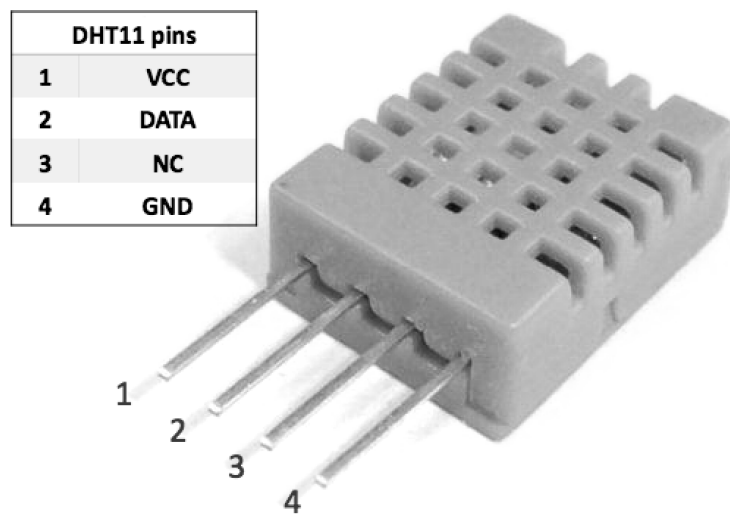
```
void putsLCD(char *buffer)
{
    int i;
    for(i = 0; *buffer; i++){
        switch(i){
            case 0:{
                WriteCmdLCD(FIRST_ROW);
                break;
            }
            case 16:{
                WriteCmdLCD(SECOND_ROW);
                break;
            }
            case 32:{
                WriteCmdLCD(THIRD_ROW);
                break;
            }
            case 48:{
                WriteCmdLCD(FOURTH_ROW);
                break;
            }
            default:
                break;
        }
        WriteDataLCD(*buffer);
        buffer++;
    }
}
```

8. ábra: A putsLCD eljárás

Azt már említettem, hogy a kijelző 4x16-os méretű, ami annyit jelent, hogy egy sorba 16

karakter fér el. Az algoritmus egyszerű, ha a 16-nak elértük az adott többszörösét, úgy átkerülünk a következő sorba. A buffer-ünk egy a PIC18 memóriájában elhelyezett tároló, amely az a szöveget tartalmazza, amelyet egyszerre átküldök a HD44780-nak. Azért tárolom a PIC18-on külön a szöveget, mert egyszerre átküldeni az egészet sokkal egyszerűbb és nem utolsó sorban könnyebben is debugolható a program így. A *WriteDataLCD(char *buffer)* eljárás valósítja meg azt a funkcionalitást, amelynek segítségével átküldjük az adatokat az LCD vezérlőjének. 0Ezzel együtt egy olyan eljárást sikerült kifejlesztenem, amellyel egyszerűen küldhetünk szöveges információkat az LCD-re.

A következő részegység amelyet ezen fejezeten belül szeretnék bemutatni, az a páratartalom érzékelő. Ez egy DHT11 típusú érzékelő. Egy nagyon olcsó szenzorról beszélünk, amelynek kezelhetősége kicsit nehézkes. Egy speciális interfészen keresztül kommunikál, úgynevezett 1-wire technológiát valósít meg. Ennek lényege, hogy a küldés és fogadás ugyanazon a lábon történik meg. Az adatok feszültség szintek változásával reprezentálhatóak.

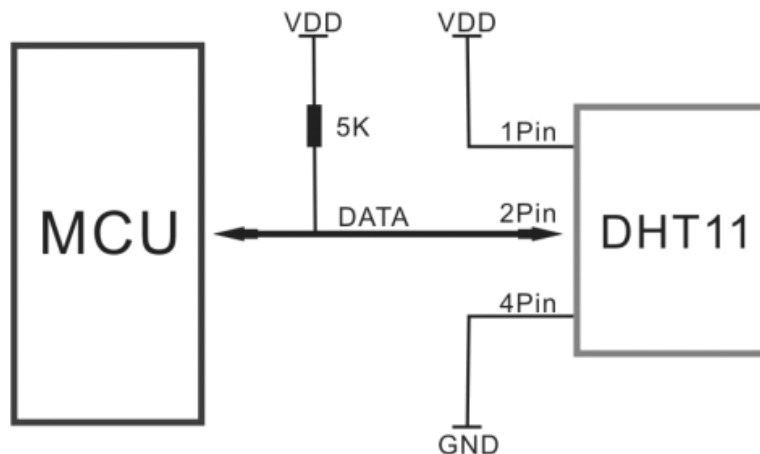


9. ábra: A DHT11 páratartalom szenzor

Forrás: 11

Ahogy azt a 9. számú ábrán is láthatjuk, a 2-es lábon keresztül tudunk kommunikálni az egységgel. A lekérdezés viszonylag bonyolult, nagyon érzékeny az időzítésre. A szenzor tulajdonságairól részletesen a páratartalom érzékelés fejezetében fogok írni.

A szenzor bekötésre a 10. számú ábra szerint történik.



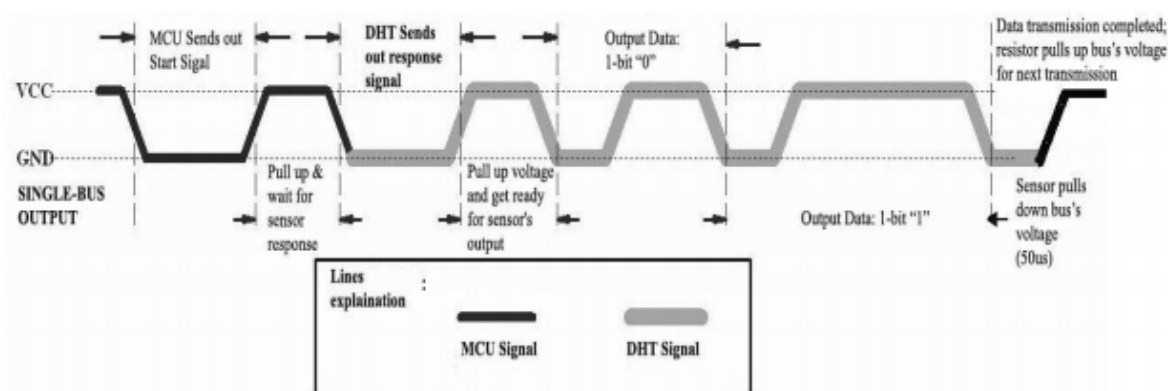
10. ábra: A DHT11 szenzor bekötése

Forrás: II

A vonal alaphelyzetben fel van húzva 5 voltra, amelyet a kommunikáció elején az MCU lehúz földre. Ezzel jelzi, hogy szeretne adatot kérni a DHT11-től. A PIC18-nak legalább 18 ms-ig lenn kell tartania 0 volton a vonalat, ahhoz hogy a szenzor azt „start” jelnek minősítse. Én 20 ms-ot várok a saját programomban. Ezután visszaállítjuk a jelet 5 voltra, majd várunk 20-40 μ s-ot, ami lényegében az én programomban abból áll, hogy átváltom a port irányát bemenetre és várom a DHT11 jelét. Ezután a szenzor átküldi a saját start jelét, ami 80 μ s-ig tart, majd felhúzza a jelet 5 voltra. ez szintén 80 μ s-ig tart. Ennek ellenőrzésére egy sajátos megoldást vezettem be. Nem foglalkoztam az időkkel konkrétan, hanem vártam 30 μ s-t miután a PIC18 oldaláról felemeltem a jelet, ami körülbelül a DHT11 start jelének időintervallumának a felébe vág bele. Ilyenkor feltételezem, hogy a jelet lehúzta a szenzor és ebből kiindulva várok, amíg a jel nem vált 5 voltra. (*while(!PORTCbits.RC0)*). Ezután szintén váraozom egy hasonló ciklussal, amíg a jel le nem kerül 0 voltra. Ezt követően már csak az adatokat kell beolvasnom hasonló módon figyelve a megadott időzítésekre. A szenzor minden bit elküldése előtt lehúzza 0 voltra a vonalat 50 μ s időtartamig, majd ezt követően küldi a bitet. Ha az átküldendő bit 0, akkor a DHT11 26-28 μ s-ig tartja a vonalat 5 volton, egyébként pedig, ha a bit 1, akkor 70 μ s-ig történik ugyanez.

Az egész kommunikációs eljárás körülbelül 4 ms-ig tart. Ezalatt a szenzor 40 bitnyi adatot küld el az MCU felé. Fontos szem előtt tartani, hogy mindig a magas helyiértéket küldi először. Az adat formátuma a következő képpen néz ki: „8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum.” – DHT11 manual.

Ezen adatokból számunkra lényegeset csak az első, integral RH adat fog jelenteni. Ez tartalmazza a páratartalom értékét. Az utolsó bájt egy **checksum**, az előző négy adat összege adja ki és a kommunikáció sikerességének ellenőrzésére szolgál. Ezt a bájtot ellenőrizve tudjuk biztosítani a programunk helyes működését. Ezt a bájtot kiolvasom, majd hiba esetén tovább ugrok a végrehajtásban és így a régi adat kerül feldolgozásra. Amennyiben a program azt érzékeli, hogy hosszú idő után (1 óra) nem változik a páratartalom a rendszerben, lekapcsolja a párásítást. Az eddig leírt kommunikációs eljárást a 11. számú ábra szemlélteti.



11. ábra: A DHT11 kommunikációs eljárása

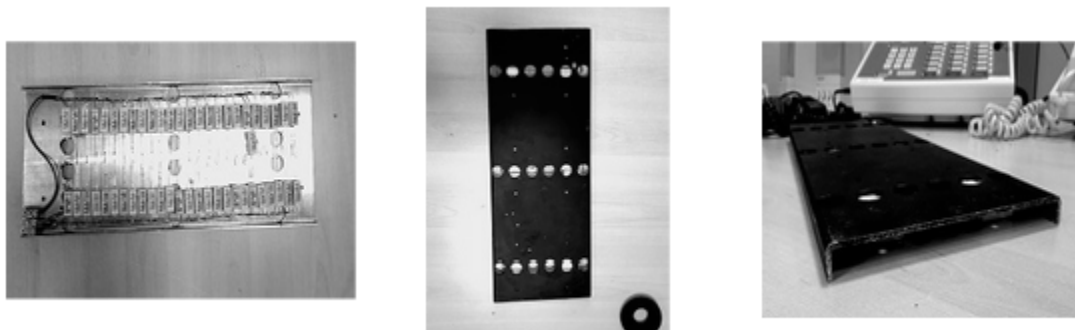
Forrás: II

A hőmérséklet érzékelése egy LM35-ös szenzorral történik. Ez az IC a hőmérséklet függvényében analóg jelet bocsát ki. Ezt felerősítjük, majd előszűrjük mielőtt a PIC18-hoz juttatjuk a jelet. Részletesen ezekről a „Hőmérséklet érzékelés” fejezetben fogok írni.

A PIC18-hoz tartozik három nyomógomb, amellyel a menü kezelése lehetséges. A gomboknál az egyedüli dolog, amire figyelni kellett, hogy a „pergést” szoftveresen megszüntessük. Ezt nagyon egyszerűen oldottam meg. Az ISR-hoz (Interrupt Service Routine) tartozó eljárások végére vettem a gombok kezelését és mielőtt a ténylegesen megtörténne a gombok állapotának beolvasása, várok egy megadott időtartamot. Bár hivatalosan ezt hardveresen szokás megoldani, de úgy tapasztaltam, hogy ezzel a megoldással is kielégítően működik a rendszer. Természetesen tervbe van véve ennek hardveres megoldása, de időhiány miatt erre még nem került sor.

A hőmérséklet beavatkozó egységként egy ellenálláshálóból kialakított fűtőegység funkcionál. Az ellenállások disszipációs tulajdonságát használtam fel arra, hogy felfűtsem a belső teret. Alapvetően az elektromos disszipáció egy káros, nem kívánt jelenség, az energia átalakulása

hőenergiává, de esetünkben pontosan hasznos célt szolgál. Az ellenállások nagy teljesítményű kerámia ellenállások, összesen körülbelül 130 W maximális teljesítményre képes az egység. Ez bőven elegendő számunkra ahhoz, hogy felfűtsük a légtérét és könnyedén tartsuk a megjelölt hőmérsékleti tartományt. Ez az egyetlen eleme a rendszernek, amely 230 V tápfeszültségről működik, ezért itt nagyon fontos volt a megfelelő szigetelés. Az ellenállások egy alumínium lapra vannak felragasztva szilikon ragasztóval. Mind a szilikon, mint az acél kiválóan vezetik a hőt. Az volt a célom, hogy minél nagyobb felületen érintkezzen a fűtőtest a levegővel és a fűtőtest olyan anyagból készüljön, amely jól vezeti a hőt és nagyon könnyen áttudja adni a levegőnek. Erre az alumínium tökéletesen megfelelt. Az ellenállásokat igyekeztem minél sűrűbben felragasztani, nem sok helyet hagyva a levegőnek, amely hűtötte volna azokat. Minél több hőmennyiségnek az alumínium lapra kellett átkerülnie. Mivel a meleg levegő fölfelé áramlik, azért a levegő keverésére és a hő elosztására is nagy figyelmet kellett szentelni. A fűtőtest fölött közvetlenül elhelyezett tojások sokkal nagyobb hőnek lettek volna kitéve, mint azok, amelyek pl. középen helyezkednek el. Viszont ha elkezdjük keverni a levegőt, akkor a hő eloszlik és minden tojás egyaránt kap ebből a hőmennyiségből. A 12. számú ábrán láthatjuk a kész fűtőtestet.



12. ábra: Az ellenállásokból összeállított fűtőegység

A levegőcseréért és ventillációért egy-egy 12 V-os ventillátor a felelős. A páratartalom szabályzásának pontos működéséről a 9. *Páratartalom* c. fejezetben fogok részletesen írni. Egy-egy tranzisztor kapcsolja ki-be a két ventillátort. Működésük egyszerű, különösebb részletezést nem igényel.

A PC-vel való kommunikáció egy USART-USB fordítón (PL2303MDL) keresztül történik. Nagyon kényelmes dolgozni vele, mivel a szintáttétellel nem kell foglalkozni, szimplán a kommunikáció implementációjával. Hardver szempontból ez sem igényel komolyabb

részletezést, különálló egységként vásároltam meg.

Végül pedig az egész elektronikai rendszer áramellátásáról ejtenék pár szót. Egy 9 V-os adapter szolgáltatja nekünk a váltóáramot, amelyet egyenirányítva, pufferelve, majd külön-külön 5 V-ra és 3.3 V-ra leszabályozva kapnak meg a részegységeink. Nagyon figyelni kellett arra, hogy ne tápláljam egyszerre PICKIT-ről és erről a panelt. Ennek elkerülése miatt belekerült egy *jumper* az áramkörbe, amelyet, ha lehúzzunk, akkor fejlesztői „üzemmódban” vagyunk, azaz leválasztjuk a tápot. Ilyenkor a PICKIT-ről történik az áramkör táplálása. Ha feltesszük a *jumpe*rt, akkor pedig a táp IC-ről történik az áramkör táplálása és figyelni kell rá, hogy véletlenül se használjuk ilyenkor a PICKIT-et.

3.1 A PIC18F4523

A PIC18F4538 egy 8 bites mikrokontroller. Azért erre a termékre esett a választásom, mert az AD átalakítójának felbontása igen magas, 12 bit szélességű. Mivel nekünk nagyon fontos a pontos hőmérséklet érzékelés, ezért ez volt a legfontosabb szempont a választásnál. Sajnos a PWM szélessége nem volt megfelelő, ezért egy másik kontroller is bekerült a rendszerbe, de erről a következő fejezetben fogok írni.

A PIC18 működési feszültsége 5 V. Belső órajelét használom, amely 4 MHz-en fut. Összesen négy Timer modul található benne, amelyekből kettőt használok fel a programomban.

A Timer0 modul a főciklusom vezérléséért felelős. A Timer0 képes 8 és 16 bites üzemmódban is futni. A főciklust vezérlő óra megköveteli azt, hogy a megszakítás minél később következzen be, ezért 16 bites módra lett bekonfigurálva az időzítő. Ezen kívül a prescaler értéke is 1:256-re van állítva, ezzel is biztosítva, a minél későbbi megszakítás generálódást. A Timer0 modult csak a főciklust vezérlő órával kapcsolatban használom, amelynek bemutatására később keríték sort.

A Timer1 modult a DHT11-es páratartalom szenzorról történő adatbeolvasáskor használom fel. A felhasználása nagyon egyszerűen történik. Amíg a DHT11 adat lába 5V-on van, addig a Timer1 fut. Amint az adatlábat a szenzor lehúzza 0V-ra, megnézem a TMR1L értékét. Ha az nagyobb, mint 40 μ s, akkor a bejövő bit 1-es volt, egyébként nulla.

```

T1CONbits.TMR1ON = 1;
TMR1H = 0;
TMR1L = 0;

while (PORTCbits.RC0) count++;

T1CONbits.TMR1ON = 0;

if (TMR1L > 40) data = data | 1;

data = data << 1;

```

13. ábra: A Timer1 modul konfigurálása

Ezen kívül még a PIC18 hardveréhez tartozik az I2C busz is, amelyet felhasználok. Ezzel kötöm össze PIC24-el ez a vezérlő. A modul részleteibe nem fogok részletesen belemenni, amit megemlíthetek erről, az annyi, hogy a PIC18 játssza a master szerepét a kommunikációban, a PIC24 pedig a slave szerepét.

Amint azt már említettem a PC-vel az USART modulon keresztül tartom a kapcsolatot. A kommunikáció 9600-as baud rate értékkel történik.

Minden más funkcionalitás szoftveresen lett megoldva a vezérlőn belül, a hardver részéről ennyit érdemes megemlítenem.

3.2 A PIC24FJ64GA002

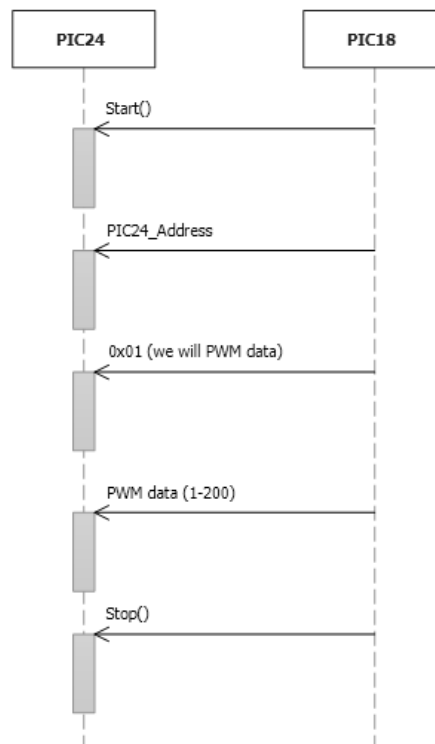
A PIC24-re azért esett a választás, mert PWM modulja skálázhatósága sokkal jobban megfelelt számomra. Mivel egy lassú jelet kell meg-meg szakítanunk (50 Hz), ezért a PWM jelünknek is megfelelően lassúnak kellett lennie. Úgy döntöttem, hogy a teljes tartomány, amelyben a jelet vezérlem, az 4 másodperc lesz. Ha 50 Hz-el számolunk a 4 másodperces tartományon, akkor egy 200 egységnyi felbontást kapunk, amelyre vezérlőjelünket leképezhetjük. ($50 \text{ Hz} \rightarrow 1 / 50 = 0.02 \rightarrow 4 \text{ sec} / 0.02 = 200 \text{ egység}$)

Ez egy nagyon egyszerűen kezelhető interfészt nyújt a PIC18 számára. A PIC18-nak az I2C buszon csupán egy olyan értéket kell elküldenie, ami az 1-200-ig terjedő intervallumra esik. Ebből a PIC24 visszafejti a szükséges paramétereket. A PIC24 PWM kimenetéhez egy optotriak csatlakozik, amely a 230 V-os feszültség vezérléséért felelős. A vezérlőjel, amely a PIC18-ből érkezik, PID algoritmussal kerül kiszámításra, majd skálázásra a 1-200-as

intervallumra. Tapasztalataim szerint a rendszer nagyon stabilan képes tartani a megkívánt hőmérsékletet és viszonylag gyorsan is képes reagálni a külső környezeti változásokra. Méréseim szerint akár 1 percen belül képes stabilizálni a hőmérsékletet egy ajtó nyitás után a rendszer.

A PIC24 egy másik funkciója, amelyet a rendszerben ellát az a valós idejű óra vezérlése. A PIC24-ben beépítve találjuk meg az RTCC (Real-Time Clock and Calendar), amely egy nagyon pontos óra modul és emellett nagyon könnyen is konfigurálható. A kezelése talán nem túl triviális és számomra kicsit nehézkes volt a lekérdezés megírása, de végül egy nagyon jól működő részegységet sikerült belőle összeállítani. Az idő értékek hexadecimális formában kerülnek kiolvasásra, nekünk kell ügyelni arra, hogy ezt megfelelően konvertáljuk át decimális, mindenki számára érthető formátumba. Az időre vonatkozó információk szintén az I2C buszon kerülnek továbbításra a PIC18 felé.

Amit még a PIC24-ről ebben a fejezetben írni érdemes, az a megszakító rutin működése. A PIC24-nek kétféle „kérést” továbbíthatunk a PIC18 oldaláról. Egyrészt küldhetünk PWM jelet, másrészt lekérhetjük az RTCC jelenlegi állapotát. Attól függően, hogy mit szeretnénk a PIC24-től, kétféle értéket küldünk ki a PIC18 oldaláról a kommunikáció elején. A PWM vezérlőjel küldésénél 0x01 hexa számot, az RTCC lekérdezésekor 0x08 hexa számot. Ezt a PIC24 oldalán ellenőrizzük, majd a megfelelő kódrészt hajtjuk végre. Az alábbi ábra szemlélteti a PWM vezérlőjel küldésének folyamatát:

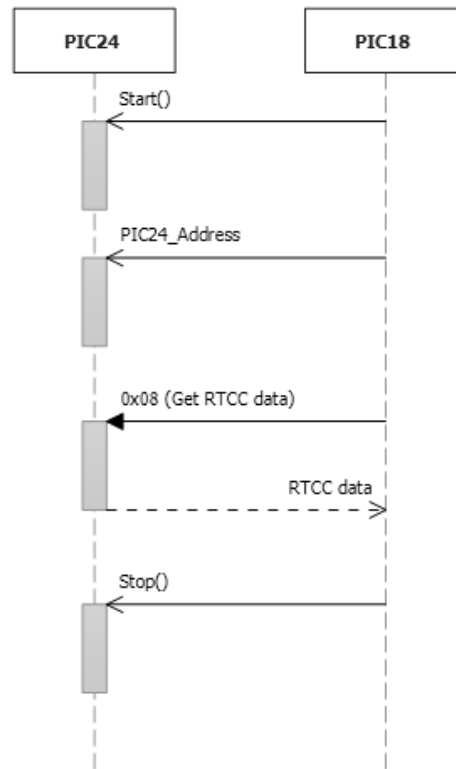


14. ábra: PWM vezérlőjel küldése PIC24-re

A PWM vezérlőjel alatt persze nem a konkrét PWM értéket értem, hanem a már fentebb részletezett 1-200-as intervallumra eső értéket. Ezt fejtí vissza a PIC24 konkrét PWM értékke. Láthatjuk, hogy itt a kommunikáció teljesen aszinkron, nem kap külön megerősítő választ ezek után a PIC18. Ennek ellenére az I2C busz ellenőrzi, hogy válaszolt a megcímzett részegység. Szóval hardver szinten a kommunikációt tekinthetjük szinkron kommunikációnak, viszont magasabb szinten semmilyen visszajelzést nem várunk azzal kapcsolatban, hogy a PWM jelet sikerült e értelmezni és azt megfelelő módon továbbítani az optotriaknak. Mivel hardveres ACK figyelés létezik, ezért figyelni kell arra, hogy ha nem elérhető a PIC24 bármilyen oknál fogva is, ne ragadjon be a program ott. Ezt úgy oldottam meg a PIC18 oldalán, hogy a kommunikáció kezdetekor egy számlálót indítok el, amely leteltekor egy megfelelő hibaüzenet íródik ki az LCD kijelzőre. A rendszer jelenlegi állapotában még nem implementáltam azt, hogy elérhetetlen részegységek esetén kapcsoljon le minden olyan részegységet, amely veszélyes lehet a környezetre, főleg a tojásokra nézve. Előfordulhat olyan eset, amikor a PWM beragad a maximális tartományában és a tojások szó szerint megsülnek. Ennek ellenőrzése mind PIC24 oldalon, mind PIC18 oldalon meg kell hogy történjen. Ha a PIC24 egy hosszú idő után nem kap jelet a PIC18-tól, le kell hogy kapcsolja a PWM egységét és ártalmatlan állapotba kell állnia. Ennek megvalósítása a terveim

közt szerepel.

Az idő lekérdezése az alábbi ábra szerint történik:



15. ábra: RTCC adat lekérdezése

Itt a kommunikáció egy ponton szinkron kommunikációvá válik. Ahogy elküldjük a 0x08 bájtot, amellyel jelezzük a PIC24 felé, hogy kérjük az RTCC adatait, úgy egy választ is várunk vissza. Ilyenkor fontos, hogy az I2C modulnak jelezzük, hogy olvasni is szeretnénk a buszról, nem csak írni rá. Ez azért fontos, mivel a PIC18 a master a kommunikációban.

A PIC24 oldalán a megszakító rutin így már egyszerűen megérthető. Ha 0x01 (PWM data) adat érkezik, akkor a következő megszakításnál tudni fogja, hogy PWM adat fog érkezni. Egyébként pedig küldenie kell az RTCC adatokat. A 16. számú ábrán láthatjuk az I2C megszakítás vektorát a PIC24-en.

```

void __attribute__((interrupt)) _SI2C1Interrupt(void){
    data = I2C1RCV;

    if(data == 0x01) next_i2c_data = 0x01; // pwm_adat fog jönni
    else if (data == 0x08) next_i2c_data = 0x08; // időt kell visszaküldeni

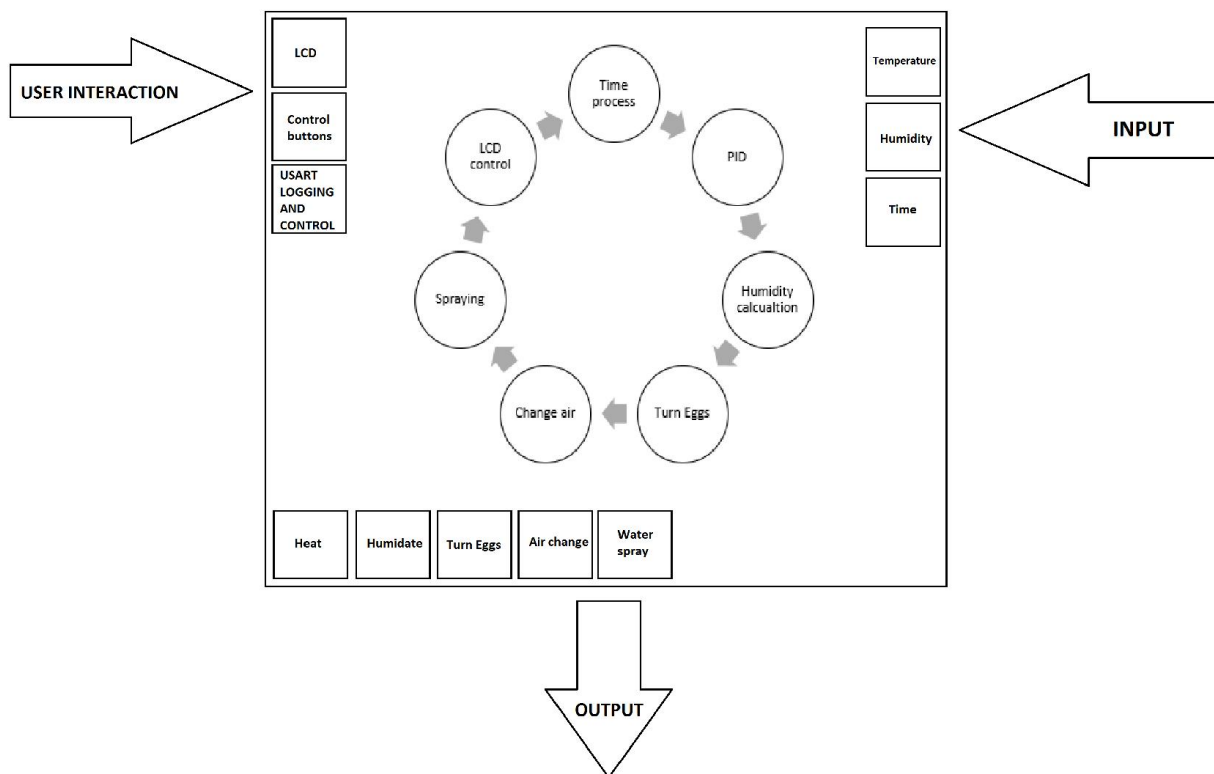
    if(next_i2c_data == 0x01){
        i2c_range = data;
        duty_cycle(i2c_range);
    } // set the duty_cycle
    else if(next_i2c_data == 0x08){
        if(_R_W && (!I2C1CONbits.SCLREL)){
            switch(current_data){
                case send_secs:{ SlaveWriteI2C1(rDate.f.mday); current_data = send_mins; break;}
                case send_mins:{ SlaveWriteI2C1(rTime.f.hour); current_data = send_hours; break;}
                case send_hours:{ SlaveWriteI2C1(rTime.f.min); current_data = send_days; break;}
                case send_days:{ SlaveWriteI2C1(rTime.f.sec); current_data = send_secs; break;}
            }
        }
    }
    IFS1bits.SI2C1IF = 0;
}

```

16. ábra: Az I2C megszakítás a PIC24 oldalán

4. A rendszer funkcionális bemutatása

A hardver bemutatása után áttérek a szoftveres követelményekre, majd azok megvalósítására. A rendszer szoftverének működését a 17. számú ábra foglalja össze.

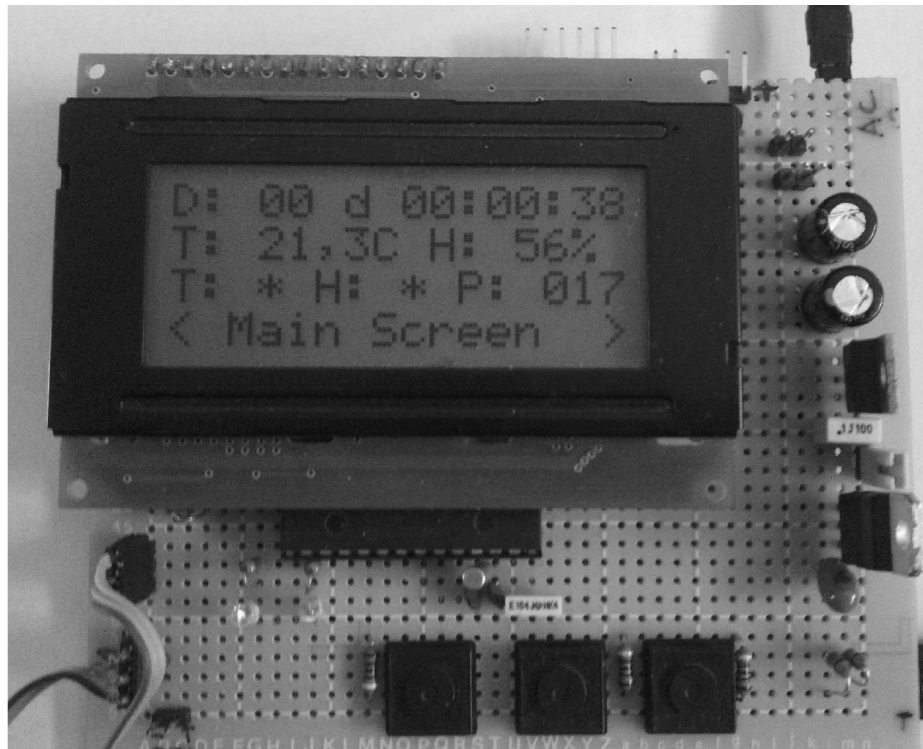


17. ábra: A szoftver működését összefoglaló ábra

Az ábra külön választja a rendszer által elvárt kimenő viselkedést, a bejövő környezeti tényezőket és a felhasználó felé szánt információkat és annak beavatkozó szerepét. Minden más a rendszeren belül valósul meg, amely rejtve van a felhasználó elől. Lényegében három külső tényezőtől függ a rendszer kimeneti viselkedése. A külső hőmérséklettől, a külső páratartalomtól és az időtől. A hőmérséklet határozza meg, hogy a fűsön-e a rendszer vagy sem, a páratartalom határozza meg, hogy szükség van-e párasításra vagy sem, az idő határozza meg, hogy mikor kell forgatni a tojásokat, mikor kell a levegőt cserélni a rendszerben és mikor kell nedvesíteni a tojások külsejét.

Ezen kívül nagyon fontos szerepe van a felhasználó által történő beavatkozásnak. A felhasználó állítja be a kívánt programot, amely magával hozza a saját paramétereit. Programok között nagy eltérés lehet. Programok alatt azt értem, hogy a felhasználónak

lehetősége van választani, milyen baromfit kíván keltetni. A felhasználónak információt kell továbbítanunk a rendszer aktuális állapotáról, aki ezek alapján képes beavatkozni a rendszerbe, szükség esetén bizonyos funkciókat ő lát el. (Forgatás, nedvesítés, levegőcsere)



18. ábra: LCD kijelzés a felhasználónak a rendszer állapotáról

A belső folyamatok, számítások kalkulálják ki a beavatkozás mértékét, annak menetét és szükségességét. Legelső lépésként az idő értékelődik ki, amely a keltetés fázisáról mond információt. Ezután a hőmérséklet értékelődik ki a bejövő értékek alapján, majd meghatározódik a kimenet. Ugyanez igaz a páratartalomra is. A tojások forgatásának, a levegő cseréjének és a tojások nedvesítésének szükségessége az idő alapján kerül meghatározásra. Ha egy bizonyos idő eltelt, akkor szükségessé válnak ezek a folyamatok. Utolsó lépésként az információkat kiírjuk LCD-re a felhasználónak.

5. A rendszer programszintű felépítése

A program felépítését tekintve egy nagy főciklusból áll, amelyet úgy konstruáltam meg, hogy csak minden másodpercben fusson le. Ez azért volt fontos szempont számomra, mert sok hibalehetőséget láttam abban, ha a rendszer ezredmásodpercenként állapotot vált a külső tényezők befolyása miatt. Ha belegondolunk a tyúkanyó sem reagál ilyen gyorsan egy-egy környezeti változásra, számára is szükséges némi idő ezen reakciókra. Így a választásom egy másodperces ciklusra esett.

A program felépítése egyre komplexebb lesz, minél mélyebbre nézünk a függvényeknek. „Felülről” nézve a főciklust az alábbi feladatokat látja el másodpercenként:

- Megnöveli a másodperc számlálót, amelyet bizonyos alegységek fognak felhasználni
- Meghívja a „CsibeTask()” eljárást, amely magában foglalja a rendszer funkcionalitásait. Ezen belül helyezkednek el az egy-egy részegységet megvalósító programsorok.
- Kezeljük a menüszerkezetet, amelyet a felhasználó az LCD kijelzőn láthat.
- Kiírjuk a bufferünk tartalmát az LCD kijelzőre.

```
while (1) {  
    if ((LONG) (TickGet() - startTime) < TICK_SECOND) continue;  
    else {  
        Seconds++;  
  
        CsibeTask();  
  
        startTime = TickGet();  
  
        handleMenuSystem(LCDBuffer);  
  
        putsLCD((char*)LCDBuffer);  
    }  
}
```

19. ábra: A főciklus

A számlálót a Microchip „TCP/IP Stack for PIC18, PIC24, dsPIC & PIC32” (16) példaalkalmazásából emeltem át. Ennek működéséről részletesen nem írnék, a legfontosabb dolog, hogy a Timer0-t használja és ezzel le is foglalja azt. A Timer0 egy 8 bites, hardveres időzítő, amely minden órajel ciklusában megnöveli a saját regiszterében tárol értékét egyel.

Amint a regiszter eléri a hexadecimális 0xFF értéket, egy megszakítás generálódik. Az átemelt időzítő működése viszonylag egy elég bonyolult eljárás alapján alapszik, sokat dolgoztam rajta, hogy átvegyem a saját környezetembe. Az óra pontos működése dokumentálva van a TCP/IP stack kódjában, amelyet az I5 hivatkozáson keresztül érhetünk el. A befektetett energia megtérült, mivel egy nagyon precíz, univerzális, jól használható eszközt sikerült átemelnem a saját programom köré. Konfigurálása viszonylag egyszerű és magáért beszél, tetszőlegesen választhatok számomra szimpatikus időközöket. Felvetődött bennem lehetőségként, hogy az üresjáratban a mikrokontrollert alvó állapotba küldjem energiatakarékosság szempontjából, de sajnos idő hiánya miatt egyelőre még nem sikerült ennek megvalósítása. A további fejezetekben mélyebbre fogok lépni a függvényekben, ezzel bemutatva az egyes funkciók megvalósítását.

Nagyon fontos ezen a ponton jeleznem, hogy a forgatás, levegőcsere és nedvesítés egyelőre nem került megvalósításra. A program vázában szerepel az interfészük, de idő hiánya miatt implementációra még nem került sor.

Még mielőtt az egyes funkciókról és a főciklusról beszélnék, szeretném bemutatni a globális paramétereket és állapotváltozókat, ezek inicializálását. De még mielőtt ezeket megtenném, megemlíteném, hogy a program MPLAB C18-as fordítóhoz lett írva és annak specifikációja szerint íródtak meg a programsorok. A Microchip azóta bevezette az XC8-as fordítót és ezt támogatja C forrású kódok megírásához, de nem ültettem át programom az új környezetbe.

A legfontosabb paramétereket ROM szinten tárolom. Ilyenek a hőmérséklet számításához és a PID vezérlőhöz szükséges paraméterek és az egyes keltetési programokhoz tartozó paraméterek. (Kelési hőmérséklet, kelési időhossz, stb.) Inicializálásnál kiolvasom ezeket az adatokat és az egyik globális struktúrában helyezem el, amelynek neve „*AppConfig*”.

```

/* Mostly read from EEPROM */
typedef struct {
    WORD divisor;
    BYTE shift;

    BYTE ConstantForCounterForDTerm;

    WORD Kp;
    WORD Ki;
    WORD Kd;

    signed long AccumulatedErrorLimit;

    BYTE SkalaTenyezo;

    unsigned char ConstantForPIDTicks;
} APP_CONFIG;

```

20. ábra: Az APP_CONFIG struktúra

Ahogy a programsorból is látszik a struktúra a hőmérséklet számításra és a PID vezérlőre vonatkozó paramétereket tárolja. A struktúra feltöltése inicializáláskor olvasódik ki a ROM-ból.

A rendszer állapotát leíró információkat másik két struktúrában tárolom, amelynek neve „AppStatus” és „PIDStatus”. Lényegében hasonló típusú információkat tárolnak, mégis fontosnak éreztem elválasztani a PID-hez kapcsolódó értékeket, mivel ezekből igen sok került bevezetésre. Az „AppStatus” struktúra az aktuális időről, hőmérsékletről, páratartalomról, a célhőmérsékletről és célpáratartalomról tárol információkat. Ezen kívül van még két boolean típusú változója a struktúrának, amely azt mondja meg kell-e fűteni vagy párásítani vagy sem.

```

typedef struct {
    TIME CurrentTime;
    WORD Temperature;
    BYTE Humidity;

    WORD DestTemp;
    WORD DestHum;

    BOOL toHeat;
    BOOL toHumidate;
} APP_STATUS;

```

21. ábra: Az APP_STATUS struktúra

A célhőmérséklet és célpáratartalom értékei változhatnak attól függően, hogy a keltetési

ciklusnak mely részében vagyunk. Ezért is nem került bele az „*AppConfig*” struktúrába. A „*toHeat*” változó a PID vezérlő mellett értelmetlennek tűnhet, de előfordulhatnak olyan esetek, amikor ki kell kapcsolni az egész fűtési mechanizmust, mert valami hiba történt, vagy éppen már kikeltek a tojások.

A „*PIDStatus*” struktúra tárolja a PID vezérléshez szükséges változókat. Ezekről a PID szabályzóról szóló fejezetben részletesen fogok írni, itt csak ennyit szeretnék megemlíteni.

6.1 Az inicializálás

Az inicializálás két részletben történik meg. Először a hardver inicializálása, aztán a program inicializálása történik meg. A hardver inicializálása magában foglalja a lábak megfelelő állapotba helyezését (input <-> output), az USART modul inicializálását, az I2C modul inicializálását, az A/D átalakító inicializálását, a megszakítások inicializálását, az oszcillátor és timerek inicializálását, majd ezek után az LCD inicializálását.

Nagyon oda kell figyelni, hogy egyetlen olyan láb sem maradjon input bemenetre állítva, amely megszakítást okozhat és nincs rákötve kívül semmi. Ez olyan hibákhoz vezethet, amelyet nagyon nehéz észrevenni és debug-olni. Ha folyamatos megszakítást kapunk egy lábról, akkor az egész program viselkedése megváltozik, lényegében nem a funkciója szerint működik. Legjobb, ha minden lábat outputra teszünk először és utána állítjuk azokat inputra, amelyeket valóban használni kívánunk.

A program inicializáló blokkjában minden változót inicializálunk, a legtöbbet 0 értékre állítjuk. Itt kerül sor a ROM-ból történő olvasásra is és az „*AppConfig*” struktúra kitöltésére is. Nem ejtettem még szót az egyes részegységekhez tartozó vezérlő struktúráról. A hőmérséklet és páratartalom egységekhez tartozik a „*TempCon*” és a „*HumCon*” nevű struktúra, amely ezen részegységek vezérléséért felelősek. Felépítésükről később fogok részletesen beszélni. Ezek inicializálása is itt történik meg. A kiválasztott keltetési programnak megfelelően töltődnek ki. Szintén ezen blokkon belül történik meg a PID és a vezérlőóra inicializálása is. A vezérlőóra inicializálása lényegében a Timer0 modul inicializálását foglalja magában. A PID inicializálása szintén a változók kezdeti értékének megadását foglalja magában, ami legtöbb esetben is azok lenullázását jelenti. Miután lefutottak az inicializáló blokkok, értesítjük a felhasználót, hogy ez megtörtént és belépünk a főciklusba.



22. ábra: LCD kijelző az inicializálás után

6.2 A főciklus

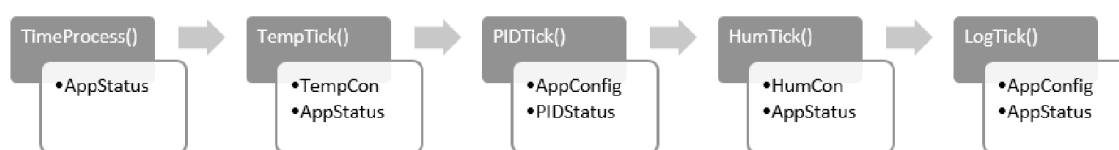
A főciklus legfontosabb elemet a „*CsibeTask()*” függvény, amely magában foglalja az összes funkcionalitást, amelyet a keltető elektronikájának meg kell valósítania. A függvényen belül további részekre vannak bontva az egyes funkcionalitások. Például a hőmérséklet érzékelés, szabályzás funkcióhoz tartozik az úgynevezett „*TempTick()*” függvény. Ez foglalja magában a hőmérséklettel kapcsolatos összes tevékenységet, eljárást. Minden funkcióhoz tartozik egy-egy „control” struktúra, amely az adott funkcionalitás konfigurációit tartalmazza. Például a hőmérséklethez hozzárendeltem az úgynevezett „*TemperatureControl*” struktúrát. Ennek tartalma az alábbi programsorokban látható:

```
typedef struct {  
    BYTE T1; // időszak 1  
    BYTE T2; // időszak 2  
    BYTE T3; // időszak 3  
  
    WORD Temp1; // temperature 1  
    WORD Temp2; // temperature 2  
    WORD Temp3; // temperature 3  
}TemperatureControl;
```

23. ábra: *TemperatureControl* struktúra

Ez az egyszerű struktúra tartalmazza azokat az időszakokat, amelyek a keltetés során

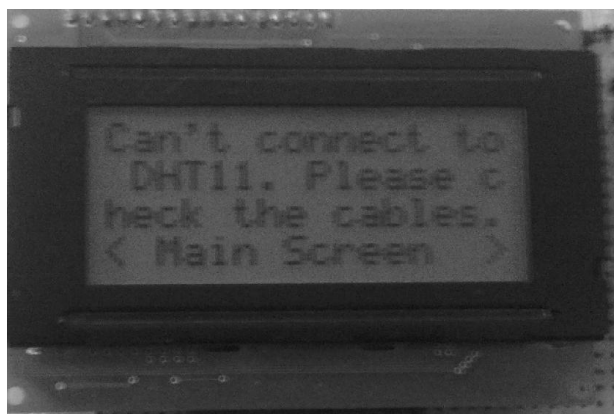
megjelennek. Beszéltünk már ezekről az időszakokról, minden időszakban más hőmérséklettel kell dolgoznunk. Ebben a konfigurációs struktúrában tároljuk el, hogy hanyadik napon lép érvénybe az adott időszak és ahhoz milyen hőmérséklet érték tartozik. Az állapot információkat nagyrészt az „*AppStatus*” struktúrában tárolom, de például a PID vezérlő állapotát egy külön struktúrában helyeztem el. Az egyes funkcionalitásokat megvalósító eljárások egymás után hívódnak meg, egymás után módosítják a megfelelő változókat. Az alábbi ábra szemlélteti köztük a sorrendet:



24. ábra: A „*CsibeTask*” belső szerkezete

Az ábra azokat a funkciókat mutatja, amelyek valóban részt is vesznek a rendszer működésében a jelenlegi állapotban. Nincs rajta a forgatást, szellőzést és nedvesítést biztosító eljárások. Az ábrán jól láthatjuk, hogy melyik funkciót melyik konfigurációs struktúra befolyásolja és ezek az eljárások hová helyezik el az állapot információkat. Első lépésként az idő kerül lekérdezésre a PIC24 mikrovezérlőtől, amely ezután befolyásolni fogja a hőmérséklet és páratartalom funkciókat. Ebből derül ki ugyanis, hogy a keltetési időszak melyik részén járunk. A hőmérséklet kiértékelése a következő lépésben történik. A leolvasott jelenlegi érték határozza meg a PID szabályzónk viselkedését, ami a megkapott értékekből előállítja a PID értéket és skálázza azt a már ismert 1-200-as intervallumra. Itt fontos megjegyezni, hogy a PID eljárás nem küldi ki a vezérlőjelet a PIC24-nek, az a következő iterációban történik meg a „*TempTick*” eljárás alatt. A következő lépésben a páratartalom kerül leolvasásra és a leolvasott értéknek megfelelően történik beavatkozás a rendszerbe. Legvégül pedig kiértékelődik, hogy a felhasználó kért-e naplózást a rendszertől és amennyiben igen, akkor kiküldésre kerülnek a megfelelő értékek az USART interfészen. Két ponton állhat meg a rendszer a folyamatban. Egyrészt az idő leolvasásánál, ha nem elérhető a PIC24 valamilyen hardveres hiba miatt, akkor az első lépésnél elakadhat a rendszer. Második hely, ahol várható, hogy hiba lép fel a kommunikációval az a páratartalom érzékelés.

Itt a DHT11 hardveres hibája okozhatja fennakadást. Mindkét esettel számoltam és egy időzítőt indítok az eljárások elején, amelynek lejártakor hibaüzenet kerül kiírásra az LCD-re. A program ezen a ponton természetesen megáll és nem szándékoztam tovább görgetni, mivel nem állnak így rendelkezésre a szükséges adatok a további működéshez. Ha hibára kerül a sor a felhasználó döntheti el, hogy újraindítja-e a rendszert vagy valamilyen javító intézkedést végez a rendszeren.



25. ábra: Hiba jelzése a felhasználónak

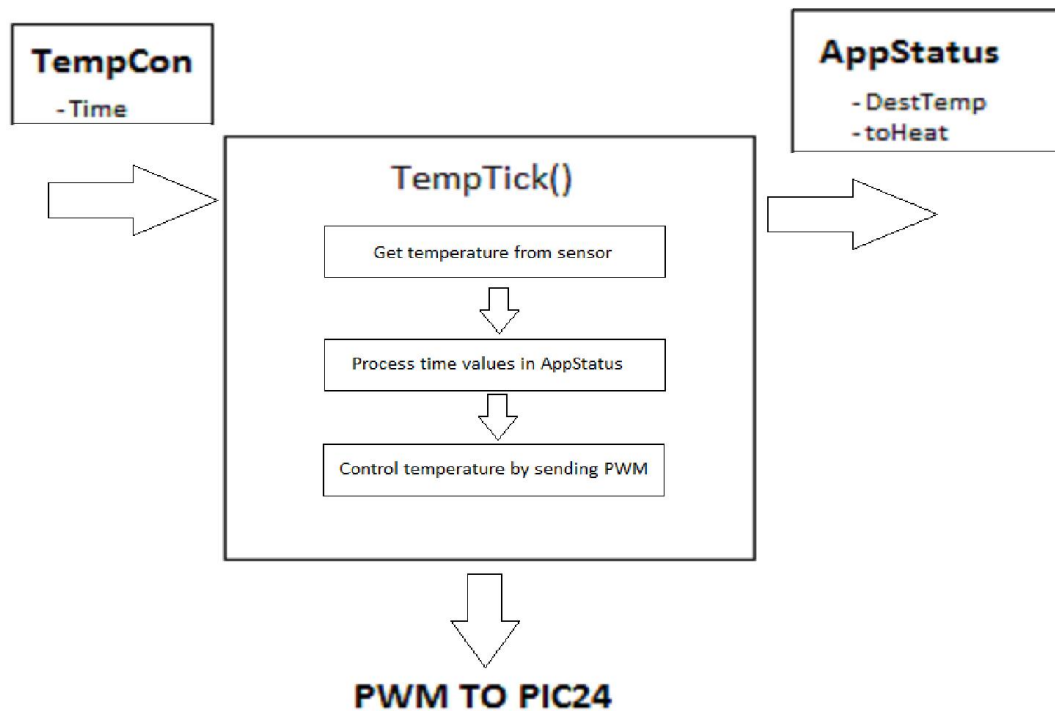
A hibát a rendszer a Timer0 megszakításában ellenőrzi, ami körülbelül 16 másodpercenként következik be.

Ebben a fejezetben szeretnék még szót ejteni a megszakításokról. Alapvetően szekvenciális a program felépítése, egyedül az esetleges hibák és természetesen a megszakítások módosíthatják a program lefutását. A rendszeremben nem használok különböző szintű megszakításokat, minden megszakítást egy szinten kezelek. Három típusú megszakításra számítok: egyrészt a Timer0 megszakítására, amely a vezérlő óra működésével kapcsolatos, az USART interfész és a nyomógombok megszakítására. A nyomógombok a menüszerkezetben való lépkedéshez és a paraméterek esetleges módosításához szükségesek. Ezeket interrupt szinten kezelem. Szintén befolyásolhatja a rendszer működését a PC felől az USART interfészen megjelenő beállítások. Például módosíthatja a felhasználó a megkívánt célhőmérsékletet az ő szokásainak megfelelően. Ezen kívül állíthatja a PID vezérlő konfigurációját is, persze megfelelő körültekintéssel.

A következő fejezetekben az egyes funkciók részletes bemutatását fogom megtenni.

7. Hőmérséklet

A hőmérséklet témakört három részre tudjuk felbontani: érzékelés, feldolgozás, vezérlés. A hőmérséklet funkció működését az alábbi ábrán szemléltetném:



26. ábra: A `TempTick()` felépítése

Szépen látszik az ábrán, hogy az eljárás három részre oszlik. Látszik ezen kívül, hogy milyen tényezők befolyásolják az eljárás működését és az hova szolgáltatja az output-ját. Egyrészt konkrét beavatkozásnak tekinthetjük a PWM elküldését a PIC24-nek, másrészt közvetett beavatkozásnak tekinthetjük, hogy az „AppStatus” struktúra változóit kezeli és így a többi részegységre is hatással lesz.

A következő három fejezetekben részletesen tárgyalom a három összetevőjét az eljárásnak, azok hogyan is valósulnak meg a gyakorlatban.

7.1 Hőmérséklet érzékelés

Mivel a hőmérséklet alapvető tényezőt játszik a kelési folyamat során, ezért nagyon fontos, hogy olyan érzékelőt használjunk, amely tized Celsius fok pontossággal képes megadni az aktuális hőmérsékletet. A legjobb választásnak a Texas Instruments LM35-ös hőmérséklet érzékelője minősült.

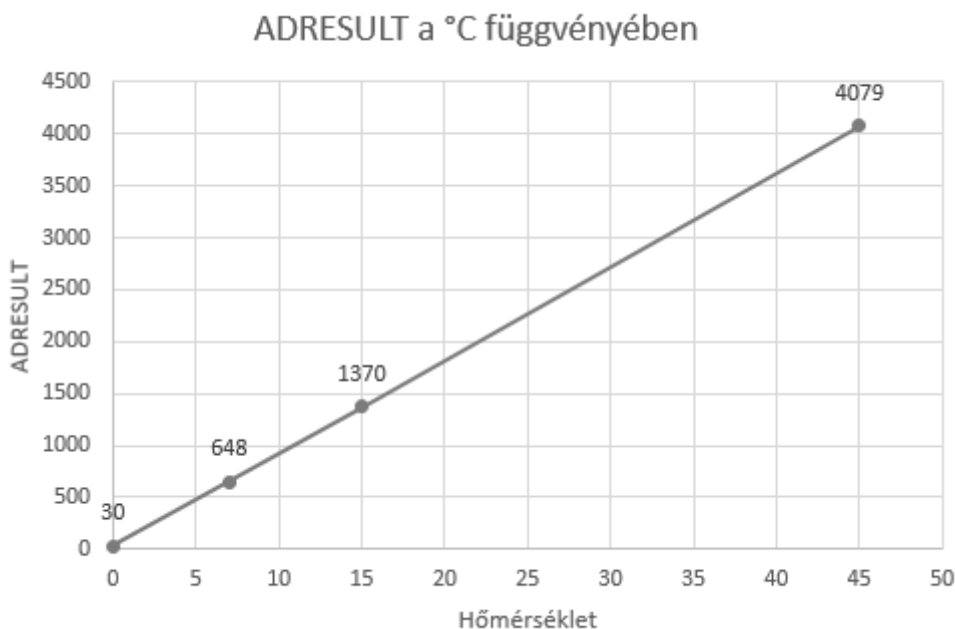
MCP617-es erősítőbe csatlakozik. Ebben az erősítőben lényegében két erősítő található. Ezekből az első erősítésre, a másodikat pedig szűrésre használom. Az erősítés ötszörös, az LM35 kimenő feszültségének az ötszörösét kapjuk meg a 2-es lábon.

Fontos megjegyezni, hogy az átalakításhoz használunk egy 2,5V-os referencia feszültséget, amelyet az RA3-as lábra kötöttem be.

Az erősítésre azért van szükségünk, hogy az AD átalakító működési tartományát sokkal hatékonyabban ki tudjuk használni. A maximális hőmérsékletet 50 °C fokra állítottuk be, ennél nagyobbra nem számítunk. Mivel az LM35 szenzor például 24,4 foknál 0.244 V feszültséget bocsát ki, amit mi ötszörösére erősítünk (1,22 V), így könnyen beláthatjuk, hogy a referencia feszültség miatt legfeljebb 50 °C mérhetünk.

$$2,5 \text{ V} / 5 = 0,5 \text{ V} \rightarrow 50 \text{ }^{\circ}\text{C}$$

Empirikus úton meghatároztam azt az értéket, amellyel az AD átalakító által szolgáltatott bináris értéket el kell osztanom. Mivel a hőmérő kimenete tartalmaz kisebb mértékű hibát, ezért nekem erre figyelni kellett. Az osztót úgy határoztam meg, hogy felvettem egy egyenest a 0°C-hoz tartozó és a 45 °C-hoz tartozó értékhez. Azért választottam a 45°C fokos értéket, mivel az AD átalakító regisztere hamarabb telítődik mint 50°C.



29. ábra: Az ADRESULT regiszter értéke a hőmérséklet függvényében

Mivel a 0°C-nál az ADRESULT értéke 30 egység, ezért minden leolvasáskor ki kell vonnunk

ezt a 30 egységet a kapott értékből. Nagyon fontos, hogy én a tényleges hőmérséklet 100-szorosával számolok annak érdekében, hogy ne kelljen tizedes jegyekkel dolgozni.

Így minden esetben az ADRESULT értéket felszorozom százzal, majd az osztónk tízszeresével osztom vissza. Habár a számítás bonyolultnak tűnhet, a kontroller idejébe még belefér és sokkal átláthatóbb és használhatóbb a programozó szempontjából. Az osztónkat nagyon egyszerűen megkaphatjuk: osszuk el a lemért értéket a hőmérséklettel. Így jön ki számunka a 90,6-os érték, amely az osztónk lesz a program futása során. A számítás a 29. számú ábrán látható.

```
void GetPIDTemperature(void) {  
    START_ADC = 1;  
    SelChanConvADC(0);  
    while (BusyADC());  
  
    ADRESULT = (WORD) ReadADC();  
  
    CloseADC();  
  
    AppStatus.Temperature=(ADRESULT - AppConfig.shift)*100/AppConfig.divisor;  
}
```

30. ábra: A hőmérséklet átszámítása

Láthatjuk, hogy először leolvassuk az analóg értéket, majd átkonvertáljuk azt digitális formába. Az AD átalakító modul a szűrőtől közvetlenül kapja meg a felerősített értéket. Az AD átalakító konfigurálást az inicializáló blokkban hajtottuk végre. A szükséges paramétereket (shift, divider) a ROM-ban tároljuk, minden reset-nél kiolvassuk és beállítjuk azok az „AppConfig” struktúra megfelelő változójába. Az AD átalakítónk egy 12 bites számmá alakítja át a bejövő analóg jelet, amelyet nekünk kell visszafejtenünk számunkra értelmes számmá.

Az előzőekben már említett *AppStatus* struktúra tárolja a rendszerünk pillanatnyi állapotát, amelyeket a szenzorokról olvastunk le, így a számított hőmérséklet is ebbe kerül.

A kódrészlet tartalmaz az MPLAB C18 által már előre definiált függvényeket és eljárásokat is. Ezek:

- a *SelChanConvADC*, amellyel azt a csatornát választjuk, ahonnan olvasni akarjuk az analóg értéket
- a *BusyADC*, amely igaz értéket ad, amíg az AD átalakító dolgozik, azaz amíg *ADCON0* regiszter *ADON* bitjében 1-es érték van.

- a *ReadADC*, amely egy 16 bites értékkel tér vissza (tartalmazza az *ADRESL* és *ADRESH* regiszterek tartalmát is), ebben tárolva az átalakítás értékét.
- a *CloseADC*, amely lezárja az AD átalakítást.

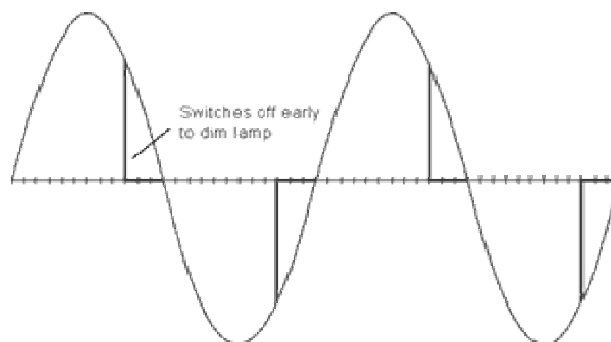
Ezután a már fent részletezett számítást végezzük el, majd eltároljuk a hőmérsékletünket. Így ezzel teljes a hőmérséklet érzékelés.

7.2 A hőmérséklet adatok feldolgozása

A következő lépésben a „*TempCon*” struktúra kiértékelődése történik a bejövő időadatok alapján. Ezen a ponton dől el, hogy éppen melyik időszakában vagyunk a keltetési folyamatnak, majd a megfelelő célhőmérséklet belekerül az „*AppStatus*” struktúrába. A tényleges mért értékek feldolgozása a PID algoritmusban történik meg, de ez egy következő fejezetünk témája. Itt még eldöntésre kerül az is, hogy egyáltalán kell-e fűteni vagy sem. Egyetlen feltételből származhat ez az érték itt, mégpedig abból, hogy túlléptük-e a megadott kelési napok számát. Amennyiben igen, a rendszer lekapcsolja a fűtést és a továbbiakban nem fog fűteni.

7.3 A hőmérséklet vezérlése

A hőmérséklet vezérlése a következő lépés az iterációban. Arról már írtam, hogyan kommunikál a PIC18 és a PIC24. A megfelelő PWM érték átkerül a PIC24-hez, ami ez alapján képes az optotriak segítségével szabályozni a 230 V-os feszültséget. Az alábbi ábra szemlélteti, hogyan tudjuk „szaggatni” az 50 Hz-es jelet:



31. ábra: Az 50 Hz-es jel „szaggatása”

Az egész lényege úgy írható le röviden, hogy a PWM a megadott értéknek megfelelően a periódus egy bizonyos szakaszában 0 V feszültségre húzza le a jelet. Így átlagolva a feszültséget egy időszak felett megkapjuk a leszabályzott teljesítményt. Például ha 50%-os

PWM jelet adunk ki, akkor a szinusz jel minden fel és lefutásánál félbe vágódik és azt átlagolva egy időszak felett kijön a 110 V. Így nagyon egyszerűen szabályozhatjuk a beavatkozó egységünket.

A PWM értéke a PID szabályzó által állapítódik meg, amelyről a következő fejezetben fogok részletesen írni.

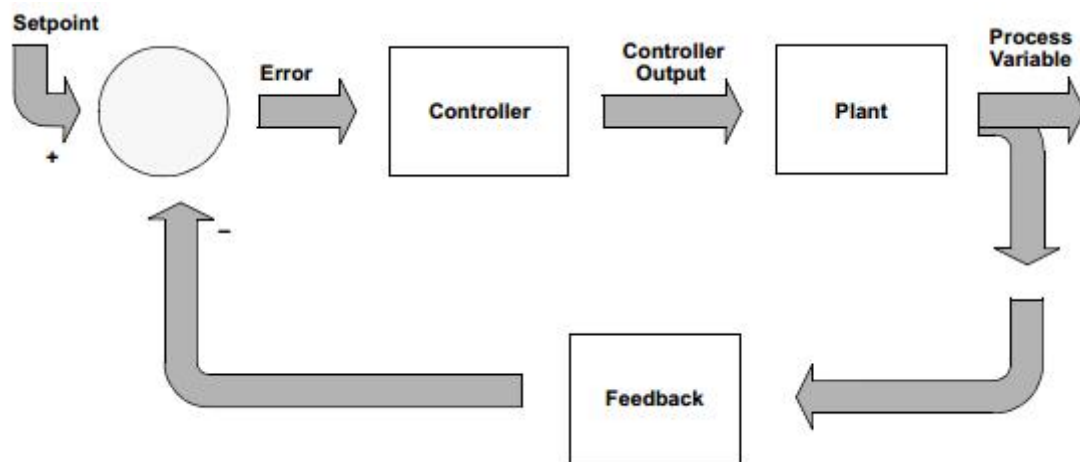
8. PID

A PID szabályzás a szabályzás technikában gyakran alkalmazott eljárás. A PID elnevezése, rövidítése a szabályzó felépítésére utal. Három tagból áll a szabályzó:

- Arányos tag (**P**: proportional)
- Integráló tag (**I**: integral)
- Deriváló tag (**D**: derivative)

A végrehajtójel a jelenlegi hiba, a múltbéli hiba és a várható hibák függvényéből adódik össze.

A PID szabályzás kiemelkedően fontos tényezője a rendszernek. Mivel nem hagyhatunk a rendszerben nagy hőmérséklet ingadozást, ezért ésszerű választásnak tűnt a PID szabályzó implementálása. Végül csak P és I tag kerül a rendszerbe, mivel a D deriváló tagnak gyorsító hatása nem hoz sok hasznót a folyamatba. Értem ez alatt, hogy a hőmérséklet ingadozása nem időkritikus, nem szükséges másodpercek alatt elérnünk a kelési hőmérsékletet. Ennek ellenére implementáltam a D tagot is kísérletezés miatt. Tesztjeim kimutatták, hogy semmilyen pozitív hatással nincs a folyamatra, így elhagyásra került. Ez a tag egyébként bekapcsolható konfiguráción keresztül, csak alapbeállításoknál nem veszi figyelembe a rendszer.



32. ábra: PID szabályzó elvi működése

Forrás: I6

A PID szabályzó három részből áll össze: az arányos tag, az integráló tag és a deriváló tag.

Mindegyikhez tartozik egy súly érték (kp, ki, kd). A kp, ki és kd tagok meghatározása empirikus úton került meghatározásra. Mivel az előzőekben már említettem, hogy a D tagot nem használtam fel, így lényeges jelentősége nincs a kd tagnak. Külön be- és kikapcsolható az integráló tag is. A deriváló és az integráló tag ki- és bekapcsolása egyrészt szervíz menüből, másrészt PC-ről is megtörténhet. Alapvetően a PID szabályzó átállítása nem ajánlott egy átlag felhasználónak, viszont olyan felhasználóknak, akik értenek a szabályozástechnikához egy nagyon flexibilis eszközt ad a kezükbe.

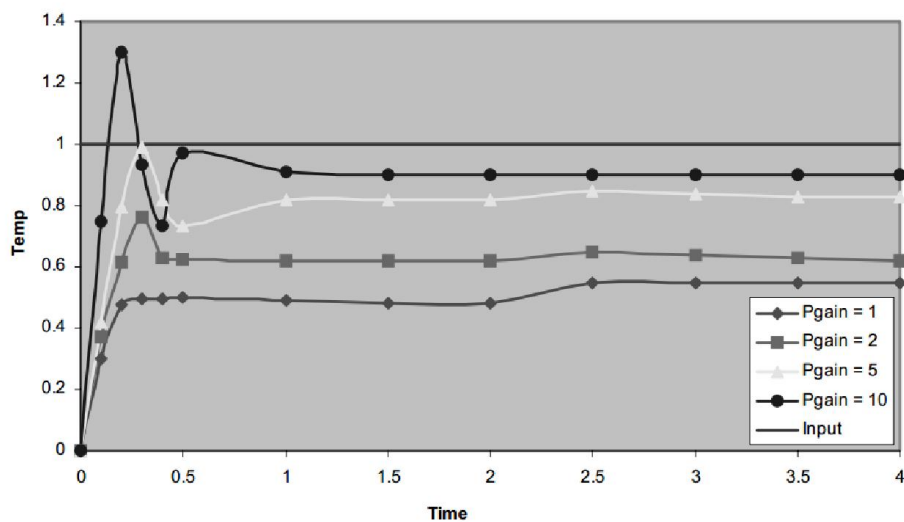
A PID szabályzó esetében beszélünk hibáról. A hiba a célhőmérséklet és az aktuális hőmérséklet közötti különbségből adódik. Ezt folyamatosan kiszámoljuk az algoritmusunk elején. Nyilván tartjuk az előző hibát is, amelyet egyrészt szummázzunk az integráló tagnak, másrészt a deriváló tagunk is ebből dolgozik.

Az arányos tag (P) a legegyszerűbb tagja a szabályzónak. Szinte minden szabályozási rendszerben megtalálható. A P érték úgy határozódik meg, hogy a kp arányossági tényezőt megszorozzuk a hibával.

```
void P_term() {  
    //prop0:prop2 = kp * error0:error1  
    pidStatus.PValue = AppConfig.Kp * pidStatus.PIDError;  
}
```

33. ábra: A P_Term eljárás

A P-tag egy maradandó hibával dolgozik, éppen ezért számunka sajnos nem elég csupán ez a tag. Megállapítható egy olyan Kp érték amellyel a rendszer igen jól megtudja közelíteni a kitűzött hőmérsékletet, de ez sok mérés vagy számolás útján lehetséges csak. A 33. számú ábrán láthatjuk, hogyan viselkedik egy rendszer a Kp érték változtatásával.



34. ábra: P szabályzó viselkedése a K_p változtatásával

Forrás: I6

Minél nagyobbra állítjuk a K_p értéküknek, a rendszerünk viselkedése annál agresszívabb lesz. Ez a fenti ábrán nagyon jól kivehető. Minél nagyobbra állítjuk a K_p tagot, annál gyorsabban érjük el a kívánt hőmérsékletet, viszont annál nagyobb kilengéssel is számolhatunk.

Az integráló tag szerepe ezután jön képbe. Az I-tag a múltbéli hibákból igyekszik következtetéseket levonni a jövő nézve. Ez gyakorlatban úgy néz ki, hogy minden iterációban az aktuális hibát hozzáadjuk az előző hibák összegéhez és ezt szorozzuk meg az I-tag arányossági tényezőjével (K_i). Nagyon figyelni kell, hogy az összeget tároló változó nehogy túlszorduljon, ezért egyrészt megfelelő méretű változót kell létrehozni neki (esetemben ez egy 4 bájtos változóban kerül tárolásra), másrészt egy olyan határértéket kell megszabni neki, amelyet ha átlép az összeg, akkor azt lenullázuk (esetemben ez az érték 100.000 egység).

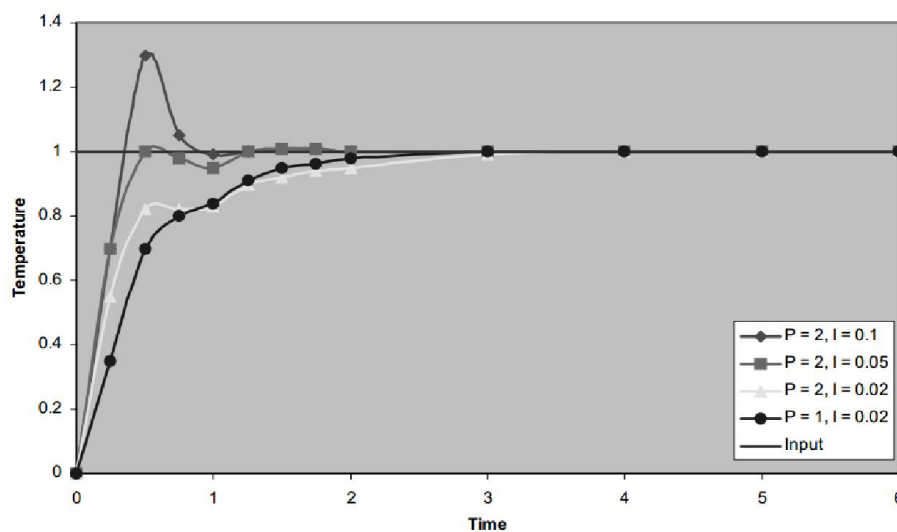
```
void I_term(){
    if(pidStatus.IntegralFlag){
        pidStatus.AccumulatedError += pidStatus.PIDError;
        pidStatus.IntegralFlag = 0;

        if(pidStatus.AccumulatedError > AppConfig.AccumulatedErrorLimit){
            pidStatus.AccumulatedError = 0;
        }
    }

    pidStatus.IValue = AppConfig.Ki * pidStatus.AccumulatedError;
}
```

35. ábra: Az I_term eljárás

A kódban található egy feltétel is. Ez annyit vizsgál meg, hogy a felhasználó kikapcsolta-e az integráló tagot. Ezt persze egy átlagos felhasználó valószínűleg nem fogja átállítani, de aki szeretne kísérletezni a rendszerrel az megteheti. Nagyon fontos még megjegyezni, hogy nem minden iterációban hajtjuk végre a számolást, mert akkor nagyon gyorsan túlsordulna a változó. Ez azért fordulhat elő, mivel a rendszer hetekig működik, nagy esély van az ilyen jellegű hibákra. Az hogy mennyi iterációt várunk egy-egy számolás között, azt a konfigurációs struktúra tárolja nekünk, de ez is állítható PC oldali program segítségével. Minél nagyobb lesz az összegzett hiba, annál nagyobb lesz az I-tag hatása.



36. ábra: Az I-tag viselkedése a K_i érték változtatásával

Forrás: I6

Jól látható a képen, hogy megfelelő K_i választás esetén a hőmérséklet alulról besimul a megkívánt hőmérséklet értékre. A lengés is megszűnik jól választott K_i esetén, ami nagyon fontos a csibék szempontjából.

Amint azt már említettem a deriváló tag az egész folyamat gyorsítására szolgál. Viszont sokkal fontosabb nekünk a pontos hőmérséklet tartása, mint a gyorsaság. Implementáltam a D-tagot is ennek ellenére, letesztelve működését valóban megfigyelhető volt a gyorsító tulajdonság, de ennek ellenére nem éreztem hasznát. Ez kikapcsolt állapotban kerül inicializálásra és ez is bekapcsolható PC oldalról. Mivel lényegében nem használom ezt a tagot, ezért nem is részletezném úgy mint az előző kettőt.

Az I-tag és a D-tag ki- és bekapcsolásra a „*PIDStatus*” struktúra, „*useIntegralUnit*” és „*useDerivalUnit*” típusú tagváltozóit használom. Ezen struktúra a *PID.h* header fájlban került

definiálásra.

Amint programunk áthaladt az egyes tagok eljárásain, már csak összegeznünk kell az általuk meghatározott eredményeket. Ez a „*PIDStatus*” struktúra *PIDValue* változójában kerül eltárolásra.

```
//PID Output = prop0:prop2 + integ0:integ2 + deriv0:deriv2  
pidStatus.PIDValue = pidStatus.PValue + pidStatus.IValue + pidStatus.DValue;
```

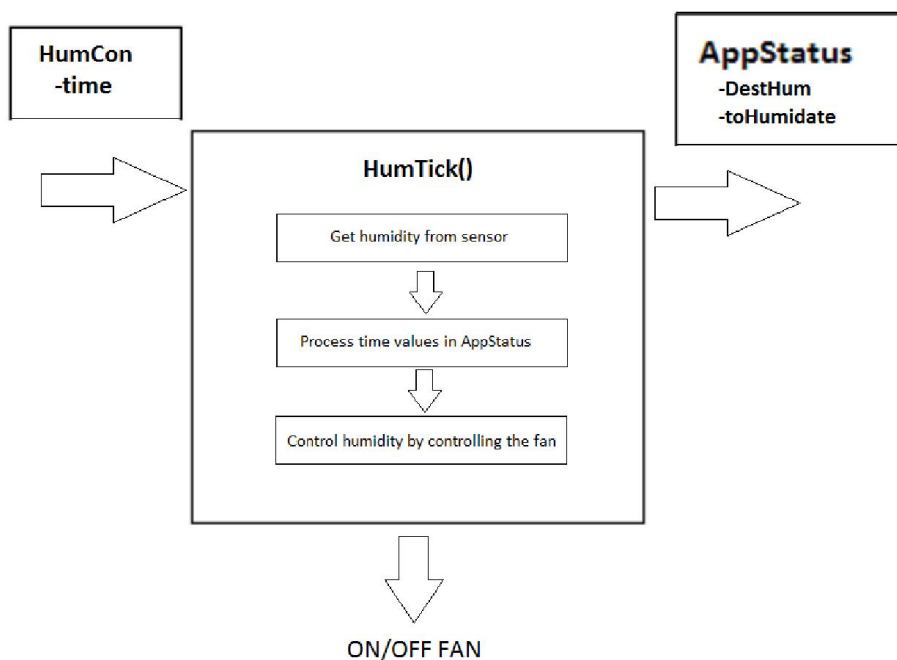
37. ábra: A PIDValue kiszámítása

Ennél a pontnál a PID szabályzó befejezi működését és már csak skálázni kell a megkapott értéket a megadott 1-200-as tartományra.

A hőmérséklet változását és a hőmérséklet-idő diagramot az általam írt PC-s programmal vagyunk képesek követni. Ennek a programnak a segítségével tudtam beállítani és ellenőrizni a PID szabályzóm működését.

9. Páratartalom

A pártartalom kezelése sokkal egyszerűbben történik a rendszerben, mint a hőmérséklet szabályozása. Bár az érzékelést láthattuk, hogy nem triviális, annak leprogramozása igen nagy kihívást jelentett, de lényegében az egész páratartalom kezelés egyszerű. A beavatkozó teljes egészében még nem készült el a rendszerhez, annak pontos kivitelezése még várat magára. Ennek ellenére már létezik egy működő modell, amelyet használok. Erről a beavatkozás részben fogok írni bővebben. A páratartalom funkciót az alábbi ábra szemlélteti:



38. ábra: A `HumTick()` felépítése

Ugyan úgy mint a hőmérsékletet érintő ábrán, ezen is látszik az eljárás tagoltsága és a funkciót befolyásoló tényezők. A beavatkozás egy ventilátor ki- és bekapcsolásából áll. Amikor párásítanunk kell a levegőt, akkor bekapcsoljuk a ventilátort, amely páras levegőt igyekszik a légtérbe juttatni, ezzel párasítva a levegőt. Ugyanúgy beállítjuk az „*AppStatus*” struktúra megfelelő változóit és ezzel közvetett hatást gyakorlunk a rendszer többi részegységére.

A következő fejeztek fogják bemutatni a részleteket a már ismert felosztásban.

9.1 Páratartalom érzékelés

A páratartalom érzékeléshez felhasznált hardver már részletesen ismertettem, annak programszintű megvalósításával együtt. Ebben a fejezetben igyekszem egy kicsit részletesebben leírni hogyan is működik az algoritmus, amely eredményeként rendelkezésre fog állni a páratartalom érték. A „HumTick” eljárás foglalja keretbe a páratartalom szabályzással kapcsolatos eljárásokat. Az alábbi programkód szemlélteti ezt az eljárást:

```
void HumTick(void) {
    DHT11Start();

    DHT11CheckResponse();

    DHT11data.HINT = DHT11ReadData();
    DHT11data.HDEC = DHT11ReadData();
    DHT11data.TINT = DHT11ReadData();
    DHT11data.TDEC = DHT11ReadData();

    AppStatus.Humidity = DHT11data.HINT;

    HumWhatToDo();

    ControlHumidity();
}
```

39. ábra: A HumTick eljárás

A DHT11-el való kommunikációt már részleteztem, itt láthatjuk annak szoftveres interfészeit. A „DHT11Start()” eljárás kezdeményezi a kommunikációt az IC-vel, majd a „DHT11CheckResponse()” eljárás ellenőrzi az IC-től kapott ACK jelet. Amint megkapom a jelet, kiolvasom a négy bájtnyi adatot, amelyet a szenzor fog küldeni. Azt már említettem, hogy csak az első bájt tartalmaz számomra fontos információt. Felmerülhet az ötlet, hogy miért nem használjuk fel a hőmérséklet adatot, amelyet küld a szenzor és akkor nem lenne szükség külön érzékelőre. Ennek nagyon egyszerű oka van: a DHT11 tévedése a páratartalomra +/- 5%, a hőmérsékletre +/- 2°C is lehet. Ekkora eltérést nem engedhetek meg egy ilyen pontosságot igénylő folyamatnál. Mindenképpen kiolvasom az adatokat, de őket nem használom fel később.

Amint rendelkezésre állnak az adatok a páratartalom értékét beleírom az „AppStatus” struktúrámba későbbi felhasználásra. Ez az érték eltérhet a tényleges páratartalom értéktől a már fent említett értékkel, de a pártartalomra nem érzékenyek olyan mértékben a csibék, mint a hőmérsékletre. Ekkora hibát eltűrnek, nem okoz problémát a kelés során.

Előfordulhat itt is az a szituáció, hogy nem kapunk választ a szenzortól és ezért elakadna a

program futása, mivel várakozna a válaszra. Ennek kezelését már ismertettem, egy hibaüzenet jelenik meg a felhasználónak erről.

9.2 A páratartalom adatok feldolgozása

A következő lépésben a „HumCon” struktúra kiértékelése történik, ugyan úgy mint történt az a hőmérséklet feldolgozásánál. A pártartalomra vonatkozóan csak két időszakot különböztettek meg. Általában minden baromfi ugyanazokat a páratartalom értékeket követeli meg a kelési folyamat előrehaladtával, ebben nem sok eltérés mutatkozik.

```
static void HumWhatToDo(void) {  
    if (AppStatus.CurrentTime.day < HumCon.H1) // if ( current day < DT1 )set HumCON,0  
    {  
        AppStatus.toHumidate = TRUE;  
        AppStatus.DestHum = HumCon.Humidity1;  
    } else {  
        AppStatus.toHumidate = TRUE;  
        AppStatus.DestHum = HumCon.Humidity2;  
    }  
}
```

40. ábra: A HumWhatToDo eljárás

A kódban láthatjuk az időszakok kezelését és azoknak megfelelő célpáratartalmak feldolgozását.

9.3 A páratartalom vezérlése

Amint azt a fejezet bevezetésénél már fejtegettem, a beavatkozás konkrét megvalósítása még nem történt meg. Egy prototípus létezik, amelynek konstrukcióját az alábbiakban részletezem. Az elgondolás az volt, hogy a meleg levegőt átfűjva egy vízzel átitatott közegen, az felvesz magába vizet és azt visszafűjva a rendszerbe párasítja a levegőt. Ennek megvalósításához egy tartályszerű kialakítást készítettem a keltető dobozához illesztve. Az alábbi képen látható ez:



41. ábra: A párasító tartály

A vizet alul tárolom a tartályban, amelyet lehetőségünk van bármikor után tölteni. A tartály zárható, így megelőzve azt, hogy a meleg kiáramoljon a rendszerből. A tartályhoz tartozik egy 12 V-os ventilátor, amelyet egy tranzisztorra ki- és bekapcsolhatunk. A közeg, amelyen átfűjjük a levegőt, egy nedvszívó törülőkendőből kivágott lapokból áll össze. Ezek a tartály tetejéből lógnak lefelé, magukba szívva a nedvességet.



42. ábra: A felhasznált törülőkendő illusztrációja

Sajnos az egyedi párasító hatékonysága nem tűnt igazán megfelelőnek, minél magasabb hőmérsékleten dolgozott a rendszer, annál gyengébbnek bizonyult a párasítási képessége. Tervbe van véve, hogy sűrűbben fogom felragasztani ezeket és így is tesztelem majd, de erre

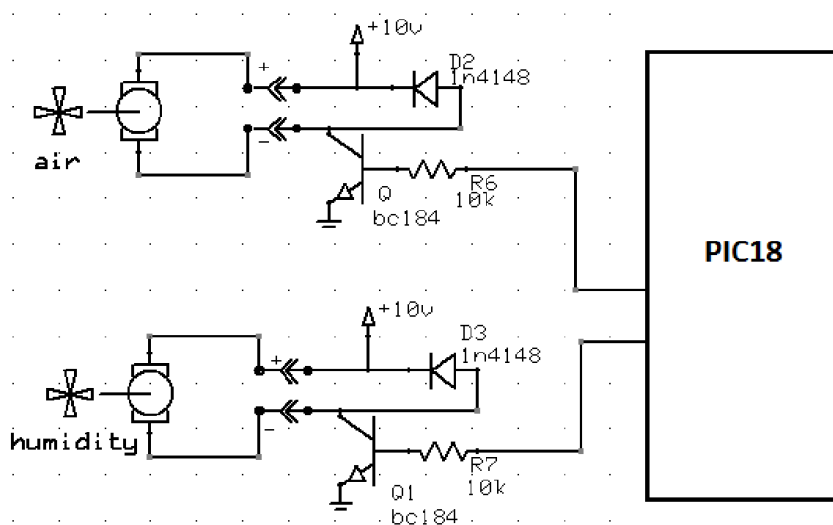
még nem született igazán jó megoldás.

A hardver részletezése után szeretném bemutatni a szoftveres kezelést a beavatkozásnak. Az alábbi programsorok foglalják össze:

```
static void ControlHumidity(void) {  
    PARASITAS = 0;  
    LEVEGOCSERE = 0;  
  
    if (AppStatus.Humidity <= (AppStatus.DestHum - 5)) {  
        PARASITAS = 1;  
    } else if (AppStatus.Humidity > AppStatus.DestHum + 5) {  
        LEVEGOCSERE = 1;  
    }  
}
```

43. ábra: A ControlHumidity eljárás

Ahogy már írtam, egy tranzisztor van a PIC18 egyik lábához kötve, amelynek ki- és bekapcsolásával tudjuk vezérelni a ventilátort. Nagyon fontos kiegészítést kell itt tennem. Nem csak a párást kezelem ezen eljárásban, hanem a levegőcserét is. A levegőcserére akkor van szükség, ha a pártartalom meghaladja a kívánt értéket. Ilyenkor levegőcserére van szükségünk, amely hatására kikerül a rendszerből a nedves levegő és csökken a pártartalom. Annak érdekében, hogy elkerüljük az állandó váltakozását a levegőcserének és a párástásnak, mindkettőnél meghatároztam egy 5%-os küszöböt. Csak akkor párástok, illetve cserélek levegőt, ha több mint 5%-al meghaladjuk vagy alatta vagyunk a megkívánt páratartalomnak. Alább látható a két ventilátor bekötési rajza:



44. ábra: A levegőcseréért és párástásért felelős ventilátorok bekötése

A ventilátorok használata követelte meg, hogy egy 9V-os váltófeszültségű adatterről tápláljam az alaplomat. Ezekről lett később leválasztva az 5V és a 3.3V.

A beavatkozó működése tesztelés közben jónak mutatkozott, hatékonyságával voltak gondok, ahogyan azt már említettem. Ennek továbbfejlesztése az egyik jövőbeni célom.

10. Forgatás

A forgatás szintén nagyon fontos szerepet játszik a keltetési folyamatban. Ipari környezetben általában 45° -ban szokták a tojásokat megdönteni a vízszinttől vagy balra vagy jobbra. Az döntés iránya változik időben. Annak oka, hogy csak 45° -al döntjük a tojásokat az az, hogy a kelési folyamat során az embriókban kialakuló köldökzsinór hajlamos a csibe nyakára tekeredni és ezzel őt megfojtani. A 45° -os forgatásnál ennek valószínűsége minimális, nem fordulhat elő ilyen jellegű halálozás.

A forgatás hardveres megvalósítása sajnos egyenlőre egyáltalán nem került kivitelezésre, így a teljes automatizálása a folyamatnak egyenlőre várat magára. A mechanikai kivitelezése a folyamatnak túlmutatott a rendelkezésre álló időn és lehetőségen. Az biztos, hogy léptető motort fogok használni a jövőbeni konstrukciónál, amelynek felprogramozása nagy kihívást fog magában rejteni.

Annak eldöntése, hogy mikor van szükség forgatásra, az időből derül ki. A PIC24 által küldött időadatok alapján tudjuk követni, hogy mennyi idő telt el a legutóbbi forgatás óta és ezzel tudjuk vezérelni a forgatást.

Mivel nincs szükség, sőt erősen ajánlott elkerülni a tojások forgatását a kelési időciklus utolsó három napjában, így ezt a paramétert is ROM szinten kell hogy tároljam.

Az alábbi képen láthatjuk, hogy egy ipari keltetőben a forgatás hogyan kerül megvalósításra:



45. ábra: Forgatás ipari keltetőben

Mivel nem áll még rendelkezésre a hardver, ezért a szoftveres implementálás is kezdeti fázisban van. Csupán az interfészek állnak rendelkezésre és azoknak meghívási helye. A jövőben ennek tényleges megvalósítása fogja jelenteni a legnagyobb kihívást.

11. Nedvesítés

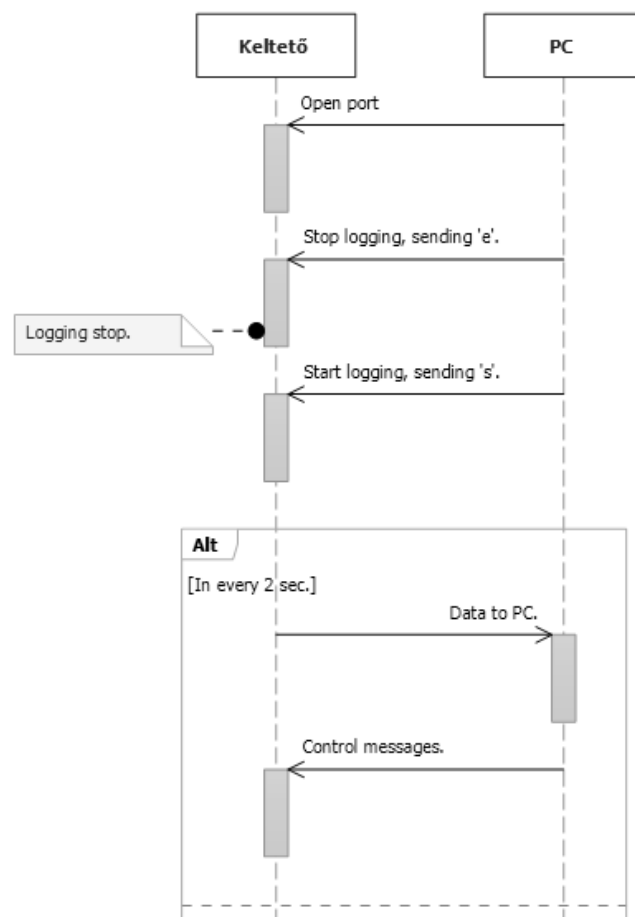
A kelési ciklus végén szükséges a tojások héjának puhítása. Mivel a tojás héja igény kemény és ezt a csibének ki kell vájnia, ha világra akar jönni, ezért ezt a folyamatot megkönnyítve nedvesíteni szokták a tojás héját. Ez a csibéknél még nem is annyira nagy probléma, mivel csőrük igen éles, viszont egy liba vagy egy kacsza esetén ez már nem mondható el. A nedvesítés nélkül nagyon rossz kelési arányt tudunk csak elérni.

A nedvesítésnek a kidolgozása egyáltalán nem került még terítékre. Csupán interfészei léteznek szoftver oldalon, hardveres megvalósításáról még nem születtek ötletek a fejemben.

12. A rendszer monitorozása

Ahogy már előző fejezetekben említettem, készítettem a keltető számára egy PC oldali programot, amely teljesíti a monitorozás kívánalmait. A program .Net Framework 4.5-ös verzióban készült, C# nyelven. A program csak Windows operációs rendszeren működik. A fejezet célja, hogy bemutassa egyes funkcionálisit és lehetőségeit ennek a programnak.

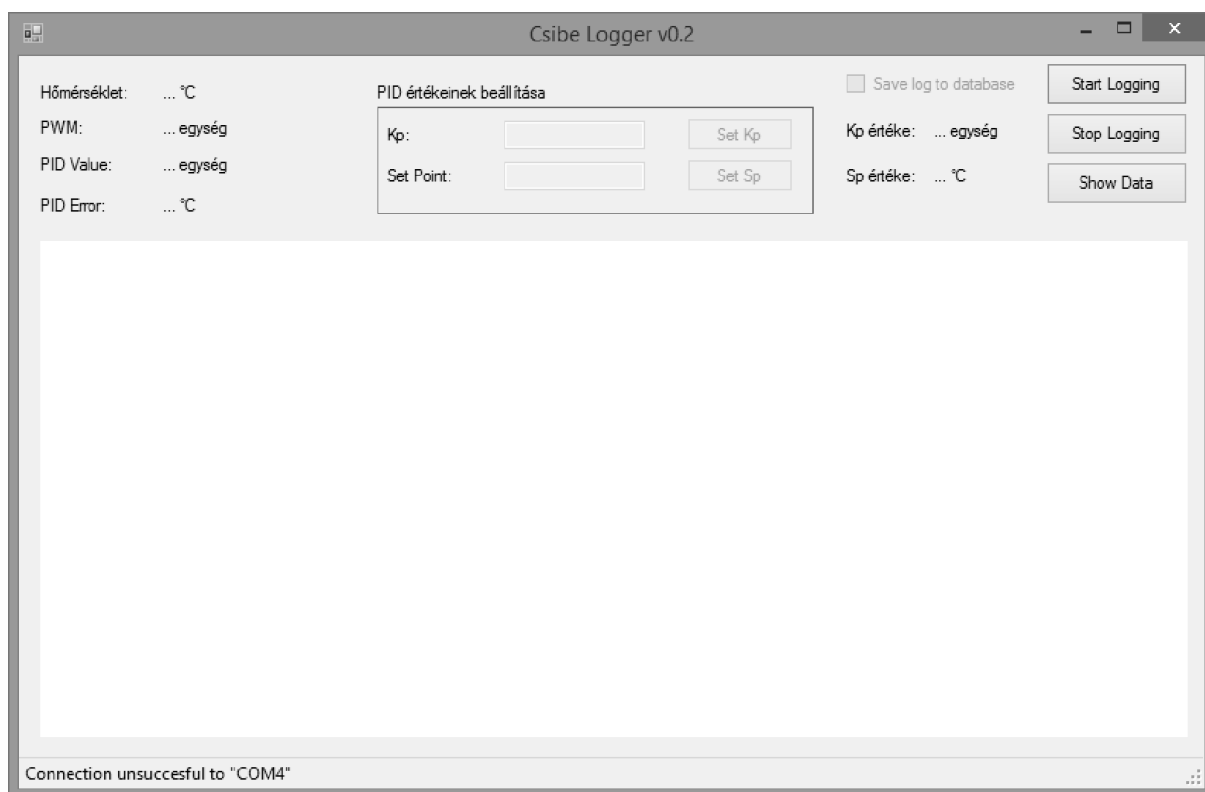
A program úgy lett kifejlesztve, hogy USART (soros) interfészen legyen képes kommunikálni a keltetővel. A soros port támogatása igen jól támogatott C# környezetben, így könnyű volt egy jól használható alkalmazást felépíteni köré. Egy kommunikációs protokollt fejlesztettem ki, amely alapján a keltető és a PC kommunikál. A 45. számú ábrán látható a kommunikáció lépései.



46. ábra: Kommunikáció a keltető és a PC között.

A program indulásakor megnyílik a soros port és ellenőrzöm, hogy sikerült-e a folyamat. Miután erről megbizonyosodtam, elküldöm a megfelelő üzenetet, amellyel kikapcsolható a

monitorozás a keltetőben. Erre azért van szükség, mert előfordulhat, hogy beragadt egy előző futásból és felesleges, hibás megszakításokat eredményezne PC oldalon. Miután kikapcsolódik a monitorozás, a PC program elindul és működésre kész. A felhasználó egy gomb segítségével tudja elindítani a monitorozást. A 46. számú ábrán láthatjuk a kezelőfelület felépítését.



47. ábra: A PC oldali program felépítése

A „Start Logging” gomb megnyomásával indíthatjuk a monitorozást, a „Stop Logging” gombbal pedig leállíthatjuk azt. Amint elindítjuk az alkalmazást a kezelőfelület megfelelő pontjain megjelennek a releváns adatok. Az esetleges hibák esetén az alsó állapotson kapunk tájékoztatást azokról. A PID szabályzónál már ismertetett paramétereket is láthatjuk és állíthatjuk a felületen. A program képes a beérkező adatokat adatbázisba tárolni az esetleges későbbi elemzések miatt. A „Show Data” gomb lenyomásával és felületet nyithatunk meg, amelyen ezeket az adatokat tudjuk böngészni.

Miután kiadjuk a „start” jelet, a keltető ettől a ponttól kezdve leállításig minden második másodpercben elküldi az adatokat a PC felé. A 46. számú ábrán láthatjuk az adatcsomag felépítését.

PWM value [1 byte]
Temperature [2 bytes]
Kp value [2 bytes]
Set Point [2 bytes]
PIDValue [2 bytes]
PIDError [2 bytes]
Ki value [2 bytes]
Kd value [2 bytes]
AccumulatedError [4 bytes]
CountOfPIDTicksOverflows [1 byte]
PIDTicks [1 byte]
useIntegralUnit [1 byte]
useDerivalUnit [1byte]
PValue [4 bytes]
IValue [4 bytes]
Dvalue [4 byte]
Humidity [1 byte]
PARASITAS [1 byte]
LEVEGOCSER [1 byte]

48. ábra: Adatcsomag felépítése

Előfordulhat, hogy az adatcsomag csak részben érkezik át PC oldalra, azaz hibás a kommunikáció. Ez előfordulhat azért, mert valamiért késik az adat vagy egyszerűen nem is fog érkezni több. Emiatt az adatok beérkezése után leellenőrzöm, hogy megkaptam-e minden szükséges tagot az adatcsomagból. Ha nem kapom meg az összes bájtot, akkor várok egy megadott ideig, majd újra megnézem, hogy megvannak-e kívánt bájtok. Ha még ekkor sem érkeznek meg, egyszerűen figyelmen kívül hagyom abban az iterációban a megszakítást. A csomag beérkezését egyébként egy megszakítás jelzi, ami Windows szinten egy „event”-nek felel meg.

A program képes valós időben diagramot rajzolni az idő és a hőmérséklet függvényében. Ez azért nagyon hasznos a fejlesztő és egy szabályzáshoz értő felhasználó számára, mert így nagyon egyszerűen konfigurálhatja a PID szabályzót. Azonnal kap visszajelzést arról, hogy ez a változtatás hogyan befolyásolta a fűtés minőségét.

13. Befejezés, összefoglalás, önértékelés

Ahogy dolgozatomban több helyen is említést tettem róla, a keltető még nem készült el teljesen. Rengeteg továbbfejlesztési lehetőséget látok, az eltervezett funkcionalitások mellett is. Igyekeztem a dolgozatban a lehető legteljesebben visszaadni azokat a problémákat, amelyekkel a fejlesztés során foglalkoztam. Kettős célom volt a projekt indulásakor: tanulni és tapasztalatot szerezni elektronika és mikrovezérlők területéről, majd kifejleszteni egy olyan rendszert, amely képes egy hidat emelni a legújabb elektronika fejlesztések és a mezőgazdaság közé.

Rengeteg hasznos tapasztalatot gyűjtöttem a fejlesztés során, nagyon gyorsan sikerült elsajátítanom az Assembly nyelvet, majd később a C nyelv sajátosságait. A keltető az egyetemi képzésem során nagy sikert aratott több kurzus keretén belül is. A téma amelyet választottam szinte mindenki fantáziáját megmozgatta, érdekes perspektívát tárt fel a hallgatók és a tanárok előtt egyaránt. Bár még nem keltek ki csibék a keltetőben, ettől függetlenül nagy sikernek tartom azt, ami eddig elkészült. A sok egyedi megoldás, a sok felhasznált tudás mind olyan dolog, amelyet csak ezen az úton járva tanulhattam meg.

Hiszek abban, hogy a keltető később még nagy sikert fog megélni a mezőgazdaságban is. Ezt a projektet csak egy biztos alapnak szántam. Erre szeretnék felépíteni egy olyan keltető hálózatot, amelyet képesek leszünk akár egy Android alapú telefonról is irányítani. Ehhez már megvannak a terveim. Mindenképpen szeretném a teljes automatizálást megoldani a forgatásra, a permetezésre vonatkozóan is.

A hőmérséklet szabályzás talán az a téma, amellyel legtöbbet foglalkoztam a projekt során. Sok nehézség adódott ezzel a témával kapcsolatban. De az eredmény úgy érzem magáért beszél. Egy nagyon pontos szabályzást sikerült kifejlesztenem, amely eleget tesz a keltetési igényeknek.

Igyekeztem a lehető legtöbb biológiai és élettani tényezőt figyelembe venni a minél jobb kelési százalék eléréséhez. Már nagyon várom, hogy kipróbálhassam valódi tojásokkal is a keltetőt és a várt siker megmutatkozzon a való életben is.

Mellékletek:

1.sz. melléklet:

A dolgozatban használt rövidítések:

PWM Pulse-Width Modulation – Impulzusszélesség moduláció

PID Proportional-Integral-Derivative controller

USART Universal asynchronous receiver/transmitter

Irodalomjegyzék

BOGENFÜRST F.(1987): A baromfikeltetés gyakorlata, Mezőgazdasági Kiadó, Bp.

DR. BOGENFÜRST F.(2000): Keltetés. Gazda Könyvkiadó, Győr.

dr. ZOLTÁN P.(1997): Baromfihús- és tojástermelők kézikönyve, Mezőgazdasági Szaktudás

I1(2014.04.05.): DHT11 manual: <http://www.micro4you.com/files/sensor/DHT11.pdf>

I2(2014.04.01.): HD44780: <https://www.sparkfun.com/datasheets/LCD/HD44780.pdf>

I3(2014.04.12.) LM35 sensor manual: <http://www.ti.com/lit/ds/symlink/lm35.pdf>

I4(2014.04.12.) PIC18F4523 manual:

<http://ww1.microchip.com/downloads/en/DeviceDoc/39755c.pdf>

I5(2014.04.13.) TCP/IP stack application note:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537041

I6(2014.04.01) Implementing a PID Controller Using a PIC18 MCU:

<http://ww1.microchip.com/downloads/en/AppNotes/00937a.pdf>

SZALAY I.(2004): Alternatív baromfitenyésztés és –tartás, Mezőgazda Kiadó, Bp.

Kiadó Kft., Bp.