ΓΑΛΑΝΗΣ ΦΩΤΙΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΔΗΜΙΟΥΡΓΙΑ ΣΥΣΤΗΜΑΤΟΣ ΛΗΨΗΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΣΥΣΤΗΜΑ ΨΥΞΗΣ-ΑΦΥΓΡΑΝΣΗΣ ΜΕ ΑΠΟΡΡΟΦΗΣΗ

Τομέας: ΘΕΡΜΟΤΗΤΑΣ

Επιβλέπων: Ειρήνη Κορωνάκη, Καθηγήτρια ΕΜΠ

Αθήνα 2024

Έχω διαβάσει και κατανοήσει τους κανόνες για τη λογοκλοπή και τον τρόπο σωστής αναφοράς των πηγών που περιέχονται στον οδηγό συγγραφής Διπλωματικών Εργασιών. Δηλώνω ότι, από όσα γνωρίζω, το περιεχόμενο της παρούσας Διπλωματικής Εργασίας είναι προϊόν δικής μου εργασίας και υπάρχουν αναφορές σε όλες τις πηγές που χρησιμοποίησα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτή τη Διπλωματική εργασία είναι του συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις της Σχολής Μηχανολόγων Μηχανικών ή του Εθνικού Μετσόβιου Πολυτεχνείου.

ΓΑΛΑΝΗΣ ΦΩΤΙΟΣ

Ευχαριστώ θερμά τον κύριο Σωτήρη Υφαντή ο οποίος με στήριξε σε όλη την διάρκεια εκπόνησης της διπλωματικής μου εργασίας καθώς και τον κύριο Γεώργιο Αντωνάκο ο οποίος μου έδωσε την ευκαιρία να εκπονήσω την συγκεκριμένη διπλωματική. Ακόμα θα ήθελα να ευχαριστήσω τους γονείς μου οι οποίοι με στήριξαν αδιάκοπα σε όλα τα χρόνια των σπουδών μου.

Σύνοψη

Τα συστήματα ψύξης με απορρόφηση αποτελούν μία μεγάλη καινοτομία στον τομέα του κλιματισμού καθώς είναι φιλικά προς το περιβάλλον και καταναλώνουν πολύ λιγότερη ενέργεια από τα συμβατικά κλιματιστικά. Τέτοια συστήματα χρησιμοποιούνται κυρίως σε βιομηχανίες λόγω μεγάλου μεγέθους προκειμένου να επιτευχθεί μεγάλος δείκτης απόδοσης. Η μονάδα που βρίσκεται στο εργαστήριο Γ5 του Τομέα Θερμότητας αποτελεί ένα σύστημα ψύξης-αφύγρανσης ανοικτού κύκλου χρησιμοποιώντας το Χλωριούχο Λίθιο ως απορροφητικό μέσο. Στόχος της εργασίας είναι η δημιουργία ενός συστήματος παρακολούθησης και λήψης δεδομένων από τους αισθητήρες που βρίσκονται στο εσωτερικό της μονάδας καθώς και η υλοποίηση του ήδη υπάρχοντος μη αδιαβατικού υπολογιστικού μοντέλου σε γλώσσα python. Απαραίτητη είναι η δημιουργία ενός δίαυλου επικοινωνίας των αισθητήρων με τον υπολογιστή, που απαιτεί χρήση εξοπλισμού λήψης δεδομένων αλλά και λογισμικού επεξεργασίας αυτών. Για την επίτευξη των παραπάνω χρησιμοποιείται το LabView το οποίο αποτελεί ένα φθηνό αλλά ισχυρό εργαλείο όσον αφορά την λήψη δεδομένων και την δημιουργία συστημάτων αλληλεπίδρασης ανθρώπου-μηχανών. Ακόμα για την δημιουργία του μοντέλου χρησιμοποιούνται αρκετές βιβλιοθήκες της python όπως η IAPWS και η COOLPROPS. Κατά τη σύγκριση μοντέλου και πειράματος παρατηρούνται κάποιες αποκλίσεις, οι οποίες οφείλονται σε παράγοντες όπως η πιθανή λανθασμένη βαθμονόμηση των αισθητήρων, σε πιθανές δυσλειτουργίες της μονάδας ή και σε εσφαλμένες παραδοχές που πάρθηκαν κατά τη δημιουργία του μοντέλου.

Abstract

Absorption refrigeration systems constitute a major breakthrough in the field of air-conditioning as they are environmentally friendly and consume less energy than conventional air-conditioners. Such systems are mainly used in industries due to their large size in order to achieve a high efficiency indicator. The unit situated in the Applied Thermodynamic Sector Γ5 laboratory is an open cycle cooling-dehumidifying system using Lithium Chloride as a liquid desiccant medium. The target of this diploma thesis is the creation of a supervisory control and data acquisition system using the sensors located inside the unit as well as the transformation of the existing non-adiabatic model into python language for a future parametric system analysis. It is essential to create a communication channel between the sensors and the computer which needs the use of data acquisition equipment plus software to process the data. In order to achieve all the above LABVIEW is used, which is a cheap but powerful tool regarding data acquisition and the creation of a human-machine interface. Additionally, for the creation of the model several PYTHON libraries are used such as IAPWS and COOLPROPS. During the comparison of the model and the experiment some deviations are detected, which are attributed to several reasons such as a possibly wrong sensor calibration, possible malfunctioning of the unit or inaccurate assumptions taken during the creation of the model.

Περιεχόμενα

1.ΕΙΣΑΓΩΓΗ ΣΤΑ ΣΥΣΤΗΜΑΤΑ ΚΛΙΜΑΤΙΣΜΟΥ

Με τον όρο κλιματισμό εννοείται η τεχνολογία που χρησιμοποιείται για τη διαχείριση και τη ρύθμιση της θερμοκρασίας, της υγρασίας , της ποιότητας αλλά και της ροής του αέρα. Τα συστήματα κλιματισμού μεγάλης ισχύος έχουν ποικίλες εφαρμογές σε διάφορους τομείς, κύριοι εκ των οποίων είναι ο κλάδος της υγείας ,της διανομής τροφίμων και η βιομηχανία, όπου απαιτούνται σωστές συνθήκες θερμοκρασίας και υγρασίας. Επιπλέον ,παρόμοια συστήματα μικρότερης ισχύος χρησιμοποιούνται ευρέως και σε επαγγελματικούς χώρους ή κατοικίες με σκοπό την δημιουργία ενός άνετου και υγιούς περιβάλλοντος για τα άτομα που διαμένουν σε αυτούς.

Η εξέλιξη των τεχνολογιών κλιματισμού έχει προχωρήσει σημαντικά τα τελευταία χρόνια μέσω της χρήσης βιώσιμων ψυκτικών αερίων ενώ έχουν ενσωματωθεί και ηλεκτρικά ή ηλεκτρονικά συστήματα ελέγχου τα οποία αυξάνουν σημαντικά την απόδοση τέτοιων συστημάτων, ενώ συμβάλλουν στη μείωση της κατανάλωσης ηλεκτρικής ενέργειας. Ακόμα ένα πολύ σημαντικό βήμα στον τομέα αυτό αποτέλεσε η ανάπτυξη των συστημάτων ψύξης με απορρόφηση , τα οποία χρησιμοποιούν θερμική ενέργεια αντί για μηχανική-ηλεκτρική για την δημιουργία ψύξης.

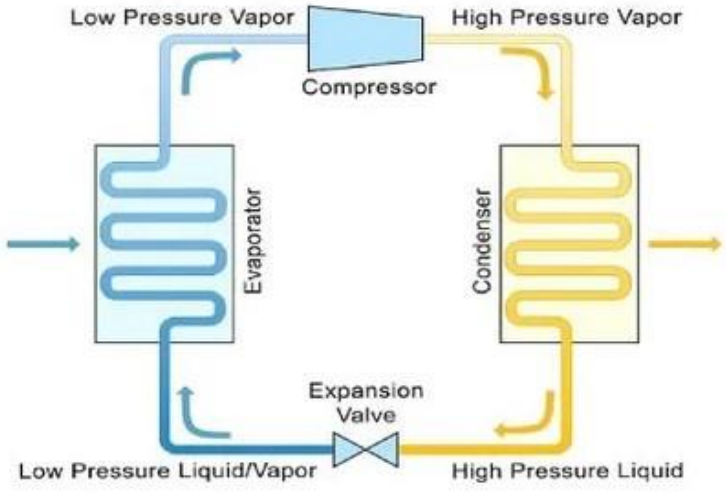
1.1 Απλός Ψυκτικός Κύκλος Συμπίεσης Ατμών

Σημαντική θα ήταν η παράλειψη αναφοράς στην πλέον κλασσική και πιο διαδεδομένη μέθοδο ψύξης , τη μέθοδο «Μηχανικής Συμπίεσης Ατμών», όπου η απαραίτητη ενέργεια για τη λειτουργία της δίνεται στη μορφή μηχανικού έργου.

Η παραπάνω μέθοδος βασίζεται στον απλό ψυκτικό θερμοδυναμικό κύκλο και τα βασικά της μέρη είναι τα παρακάτω:

- Ο εξατμιστής: Δεν είναι παρά ένας εναλλάκτης μέσα στον οποίο κυκλοφορεί το ψυκτικό μέσο και απορροφά θερμότητα από τον αέρα του περιβάλλοντα χώρου. Έτσι ο χώρος ψύχεται ενώ το μέσο περνά στην αέρια φάση διατηρώντας όμως χαμηλή πίεση.
- Ο συμπιεστής: Αποτελεί μία διάταξη στην οποία επιτυγχάνεται η αύξηση της πίεσης του ψυκτικού μέσου, το οποίο είναι εξολοκλήρου σε αέρια πλέον μορφή. Με την αύξηση της πίεσης αυξάνεται ταυτόχρονα και η θερμοκρασία του.
- Ο συμπυκνωτής: Είναι ένας εναλλάκτης θερμότητας στον οποίο γίνεται συναλλαγή θερμότητας του μέσου με τον αέρα με αποτέλεσμα αυτό να περνά ξανά στην υγρή φάση έχοντας όμως υψηλή πίεση.

- Η εκτονωτική βαλβίδα: Μειώνει την πίεση του υγρού ψυκτικού μέσου, με αποτέλεσμα την ταυτόχρονη μείωση της θερμοκρασίας του. Μετά την εκτονωτική βαλβίδα το μέσο επιστρέφει στην αρχική του κατάσταση και εισέρχεται ξανά στον εξατμιστή.



Εικόνα 1.Ψυκτικός Κύκλος

Περιβαλλοντικά –Ενεργειακά Θέματα

Επιτακτική ανάγκη αποτελεί η ανάπτυξη πιο οικονομικών, από άποψη κατανάλωσης ενέργειας, αλλά και πιο φιλικών προς το περιβάλλον συστημάτων κλιματισμού. Συγκεκριμένα τα συνήθη κλιματιστικά απαιτούν μεγάλη κατανάλωση ενέργειας κατά τη συμπίεση των ψυκτικών μέσων, ενώ τα τελευταία έχουν καταστροφικές συνέπειες για το περιβάλλον σε περίπτωση που απελευθερωθούν στην ατμόσφαιρα. Ειδικότερα τα μέσα Υδροχλωροφθοράνθρακα όπως το R-22 συμβάλλουν στην υπερθέρμανση του πλανήτη αλλά και στην καταστροφή του όζοντος, λόγω της μακράς διάρκειας παραμονής τους στην ατμόσφαιρα. Ήδη έχουν εφαρμοσθεί αρκετά περιοριστικά νομοσχέδια σχετικά με την απαγόρευση της χρήσης τέτοιων ψυκτικών μέσων ενώ διεξάγεται συνεχώς έρευνα με σκοπό την δημιουργία πρωτότυπων τεχνολογιών με μικρότερο περιβαλλοντικό αποτύπωμα.

1.2 Ψύξη με Απορρόφηση

Η συγκεκριμένη διαδικασία ψύξης αποτελεί μία μεγάλη καινοτομία η οποία δίνει λύση στα προαναφερθέντα ζητήματα. Βασικό της χαρακτηριστικό αποτελεί η απορρόφηση θερμότητας του αέρα από ένα απορροφητικό μέσο , δίχως μηχανική συμπίεση. Συστήματα τέτοιου είδους χρησιμοποιούνται σε βιομηχανικούς χώρους ή αποθήκες όπου το φορτίο θερμότητας είναι χαμηλό αλλά απαιτείται ακριβής ρύθμιση της υγρασίας του αέρα.

Τα βασικά μέρη ενός συστήματος απορρόφησης είναι ο απορροφητής, ο αναγεννητής, ο συμπυκνωτής, η εκτονωτική βαλβίδα και ο εξατμιστής. Τα συστήματα ψύξης με απορρόφηση χωρίζονται σε δύο βασικές κατηγορίες: Τα συστήματα κλειστού και τα συστήματα ανοιχτού κύκλου. Βασική διαφορά των δύο συστημάτων αποτελεί ότι στα συστήματα ανοιχτού κύκλου, σε αντίθεση με του κλειστού κύκλου, το απορροφητικό μέσο έρχεται σε άμεση επαφή με τον αέρα στους εναλλάκτες του απορροφητή και του αναγεννητή.

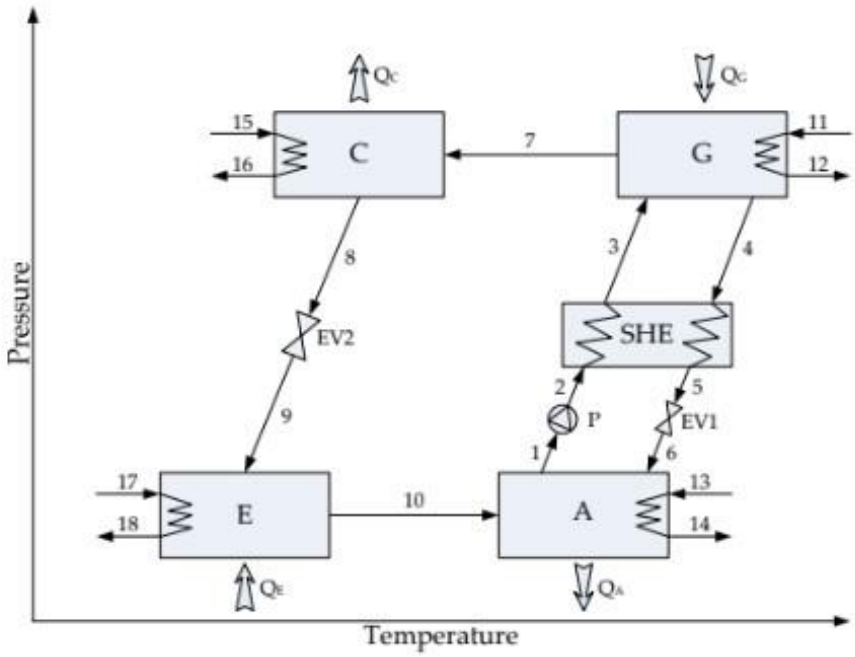
Παρακάτω περιγράφεται συνοπτικά ένα σύστημα κλειστού κύκλου :

- Εξατμιστής: Στη διάταξη του εξατμιστή εισέρχεται νερό προκειμένου να ψυχθεί καθώς και το ψυκτικό μέσο (συνήθως νερό) ,τα οποία όμως δεν έρχονται σε άμεση επαφή. Το νερό κυκλοφορεί στους σωλήνες ενός εναλλάκτη ενώ το ψυκτικό μέσο ψεκάζεται πάνω στις σωλήνες του εναλλάκτη. Στη συγκεκριμένη διάταξη η πίεση είναι εξαιρετικά χαμηλή με αποτέλεσμα κατά τη συνδιαλλαγή θερμότητας το ψυκτικό μέσο να εξατμίζεται. Σε πολλές περιπτώσεις, για να εξασφαλισθεί η πλήρης εξάτμιση του ψυκτικού μέσου εγκαθίσταται ένα δοχείο το οποίο συλλέγει το υγρό ψυκτικό μέσο το οποίο μέσω μίας αντλίας οδηγείται ξανά στους ψεκαστήρες.
- Απορροφητής: Το αέριο ψυκτικό μέσο οδηγείται στον απορροφητή όπου αναμιγνύεται με το απορροφητικό μέσο,το οποίο ψεκάζεται εσωτερικά της διάταξης του απορροφητή μέσω καταιονιστήρων. Η ανάμιξη των τελευταίων γίνεται πολύ εύκολα καθώς τα μόρια του απορροφητικού διαλύματος έλκουν σχεδόν μαγνητικά αυτά του αέριου ψυκτικού μέσου. Κατά την απορρόφηση απελευθερώνεται θερμότητα ίση με την λανθάνουσα θερμότητα συμπύκνωσης του αερίου. Η εγκατάσταση ενός εναλλάκτη στον οποίο κυκλοφορεί νερό επιλύει το πρόβλημα αφού η θερμότητα αυτή θα απορροφάται από το νερό.Ταυτόχρονα η θερμοκρασία όλου του απορροφητή θα μειώνεται με αποτέλεσμα η διαδικασία της απορρόφησης να γίνεται πιο αποτελεσματικά.

- Αναγεννητής: Στον αναγεννητή εισέρχεται ζεστό νερό ή ατμός προερχόμενο από κάποια πηγή θερμότητας και κυκλοφορεί στους σωλήνες ενός εναλλάκτη Ταυτόχρονα εισέρχεται και το διάλυμα ψυκτικού μέσου-απορροφητικού υγρού προερχόμενο από τον απορροφητή χωρίς να έρχεται σε άμεση επαφή με το ζεστό νερό(ή ατμό). Με τη θέρμανση του διαλύματος επιτυγχάνεται ο διαχωρισμός του απορροφητικού υγρού και του ψυκτικού μέσου μέσω εξάτμισης του τελευταίου. Σημαντικό είναι να τονιστεί πως ο εύκολος διαχωρισμός των δύο μέσων απαιτεί μεγαλύτερη πτητικότητα στο ψυκτικό απ'ότι στο απορροφητικό μέσο. Το απορροφητικό υγρό λαμβάνει την αρχική του συγκέντρωση και είναι έτοιμο να επαναχρησιμοποιηθεί για την διαδικασία της απορρόφησης.
- Συμπυκνωτής: Ο συμπυκνωτής δεν είναι παρά ένας εναλλάκτης νερού-ψυκτικού μέσου. Το νερό προερχόμενο από τον απορροφητή βρίσκεται ακόμα σε κατάλληλη θερμοκρασία και πίεση ώστε μέσω της συνδιαλλαγής θερμότητας με το ζεστό αέριο να εξασφαλίσει την συμπύκνωση του τελευταίου. Το ψυκτικό μέσο πλέον σε υγρή φάση οδηγείται στην εκτονωτική βαλβίδα ενώ το νερό οδηγείται σε κάποια ψυκτική διάταξη(σε κάποιο πύργο ψύξης) ώστε να ψυχθεί και να επαναχρησιμοποιηθεί στον απορροφητή .

Παρατηρήσεις:

- Αφού το σύστημα είναι κλειστού κύκλου το απορροφητικό μέσο δεν έρχεται σε άμεση επαφή με το νερό που πρόκειται να ψυχθεί και να χρησιμοποιηθεί αλλά κυκλοφορεί σε κλειστό κύκλωμα μεταξύ του απορροφητή και του αναγεννητή.
- Η εκτονωτική βαλβίδα επιτελεί την ίδια ακριβώς λειτουργία με πριν, δηλαδή την μείωση της πίεσης και της θερμοκρασίας του ψυκτικού μέσου, έτσι ώστε να οδηγηθεί στον εξατμιστή και ο κύκλος να επαναληφθεί .
- Η εγκατάσταση ενός εναλλάκτη διαλύματος του απορροφητικού μέσου θα μπορούσε να οδηγήσει στη μείωση της ενέργειας που χρειάζεται για την «αναγέννηση» ενώ μειώνει και την θερμότητα που εκλύεται κατά την απορρόφηση, η οποία πρέπει να απορροφηθεί από το ψυκτικό νερό. Στον εναλλάκτη αυτό θα εισέρχεται ψυχρό διάλυμα πλούσιο σε υγρασία(με χαμηλή συγκέντρωση) από τον απορροφητή και το θερμό αλλά υψηλής συγκέντωσης μέσο που εξέρχεται από τον αναγεννητή.



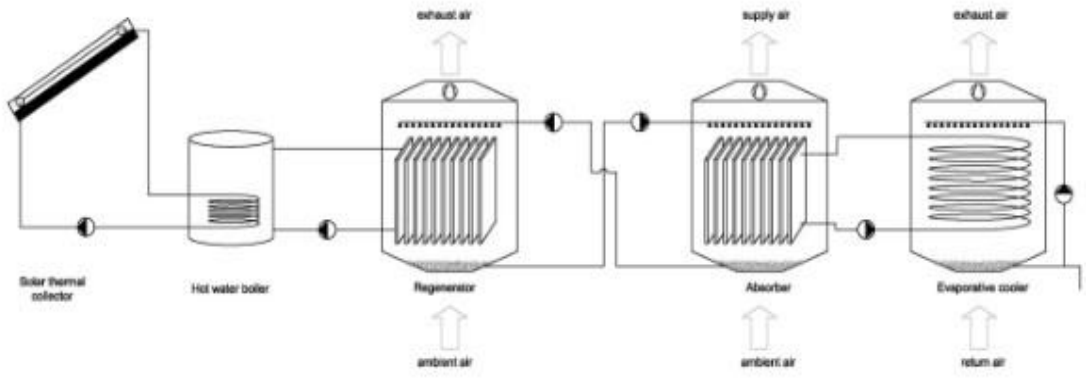
Εικόνα 2.Σύστημα Ψύξης με Απορρόφηση Κλειστού Κύκλου

Παρακάτω ακολουθεί η περιγραφή ενός συστήματος ανοιχτού κύκλου. Τα δύο βασικότερα μέρη ενός τέτοιου συστήματος είναι ο απορροφητής και ο αναγεννητής.

- Απορροφητής : Στην διάταξη ενός απορροφητή εισέρχεται αέρας από το περιβάλλον και έρχεται σε άμεση επαφή με το απορροφητικό μέσο. Αυτό έχει την ιδιότητα να απορροφά την υγρασία του αέρα, αλλά κατά την διαδικασία απορρόφησης της υγρασίας απελευθερώνεται θερμότητα. Για αυτό το λόγο ταυτόχρονα με τον εναλλάκτη αέρα-απορροφητικού μέσου εγκαθίσταται εναλλάκτης στον οποίο κυκλοφορεί νερό το οποίο ψύχει τον αέρα αλλά και το αφυγραντικό μέσο. Ο αέρας ,ξηρός και ψυχρός, εξέρχεται στο χώρο ενώ το απορροφητικό μέσο συλλέγεται σε ένα δοχείο και οδηγείται στον αναγεννητή.
- Αναγεννητής: Η διάταξή του περιλαμβάνει τα ίδια στοιχεία με τον απορροφητή αλλά ο τρόπος λειτουργίας του είναι ακριβώς αντίθετος. Θερμός αέρας εισέρχεται από το περιβάλλον , ενώ ταυτόχρονα κυκλοφορεί σε άλλο εναλλάκτη και θερμό νερό , προερχόμενο από μια εξωτερική πηγή θέρμανσης.Το αφυγραντικό μέσο έρχεται σε άμεση επαφή με τον αέρα θερμαίνεται και απελευθερώνει τους υδρατμούς που είχε προηγουμένως απορροφήσει. Ο αέρας εξέρχεται στο περιβάλλον ενώ το μέσο συλλέγεται και οδηγείται στον απορροφητή ώστε να επαναληφθεί η διαδικασία

Παρατηρήσεις :

- Στην παραπάνω διαδικασία φαίνεται πως υπάρει ανάγκη τροφοδοσίας νερού για την ψύξη του αέρα στην διάταξη του απορροφητή. Ένας διαδεδομένος τρόπος για την παροχή νερού ψύξης είναι η εγκατάσταση ενός εξατμιστικού πύργου ψύξης ο οποίος θα λειτουργεί παραλάλληλα με τις υπόλοιπες διατάξεις.
- Η πρόσδοση θερμότητας στον αναγεννητή δεν εξασφαλίζεται πλήρως από τον θερμό αέρα αλλά καθίσταται αναγκαία η χρήση μιας εξωτερικής πηγής θερμότητας.Πολύ συχνά χρησιμοποιείται ένα ηλιακός , για την ελαχιστοποίηση του κόστους ρεύματος λειτουργίας της μονάδας.
- Θα μπορούσε να εγκατασταθεί ένας εναλλάκτης υγρού διαλύματος ο οποίος θα συνέβαλε στην προθέρμανση του κρύου διαλύματος (προερχόμενο από τον απορροφητή) πριν την είσοδό του στον αναγεννητή και έτσι αυξάνει την συνολική απόδοση του συστήματος.



Εικόνα 3.Σύστημα Ψύξης με Απορρόφηση σε Συνδυασμό με Ψύκτη και Πηγή Θερμότητας προερχόμενη από ηλιακό

Τα οφέλη των συστημάτων ψύξης με απορρόφηση διακρίνονται ήδη στην παραπάνω συνοπτική περιγραφή. Συγκεκριμένα:

- Το κόστος λειτουργίας τους είναι ελάχιστο συγκριτικά με τα συστήματα μηχανικής συμπίεσης, αφού δεν υπάρχει μηχανικός συμπιεστής , με αποτέλεσμα το μόνο «κόστος» να είναι το ρεύμα των κυκλοφορητών.
- Τα συστήματα αυτά αποτελούν ένα μεγάλο βήμα προς την εκμετάλλευση των ανανεώσιμων πηγών ενέργειας, όπως η ηλιακή που προαναφέρθηκε.
  Ακόμα η απουσία ψυκτικών μέσων σε συνδυασμό με το γεγονός ότι τα αφυγραντικά είναι πλήρως φιλικά προς το περιβάλλον ελαχιστοποιεί το περιβαλλοντικό τους αντίκτυπο.
- Η ευελιξία στην επιλογή εξωτερικής πηγής θερμότητας τα καθιστά κατάλληλα για εγκαταστάσεις συμπαραγωγής θερμότητας-ενέργειας ,καθώς η απορριπτόμενη θερμότητα θα μπορεί να εκμεταλλευτεί από τον αναγεννητή. Με αυτό τον τρόπο εξασφαλίζονται τα κατάλληλα επίπεδα υγρασίας, όπου χρειάζεται, ενώ αυξάνεται και ο συνολικός βαθμός απόδοσης των εγκαταστάσεων.

Ένα σημαντικό μειονέκτημά τους συγκριτικά με τα συμβατικά συστήματα κλιματισμού αποτελεί ο χαμηλός δείκτης COP. Για αυτό το λόγο τέτοια συστήματα έχουν πολύ μεγάλες διαστάσεις με σκοπό να έχουν την απαραίτητη απόδοση ,γεγονός που τα καθιστά ικανά για χρήση αποκλειστικά σε βιομηχανία ή πολύ μεγάλους χώρους. Ακόμα το αρχικό κόστος εγκατάστασης τους τα καθιστά ασύμφορα για κατ'οίκον χρήση.

1.3 Υγρά Αφυγραντικά Μέσα-Χλωριούχο Λίθιο

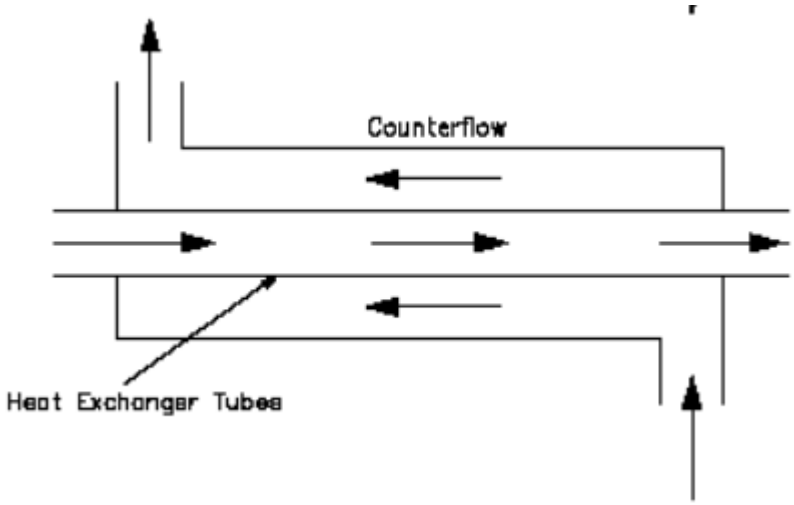
Η κατάλληλη επιλογή του αφυγραντικού μέσου είναι απαραίτητη για την σωστή και αποδοτική λειτουργία των συστημάτων ψύξης με απορρόφηση. Τα υγρά αφυγραντικά ξεχωρίζουν λόγω της ικανότητας απορρόφησης μεγάλων ποσοτήτων υγρασίας. Επιπλέον μερικά από αυτά ,όπως το Χλωριούχο Λίθιο και Βρωμιούχο Λίθιο μπορούν να αποδώσουν την υγρασία που έχουν απορροφήσει σε σχετικά χαμηλές θερμοκρασίες κατά την διαδικασία της αναγέννησης. Ένα ακόμα βασικό προτέρημα τους αποτελεί η δυνατότητα για συνεχή λειτουργία, γεγονός που οδηγεί σε ακριβή ρύθμιση της υγρασίας σε χώρους όπου αυτή είναι απαραίτητη.

Το Χλωριούχο Λίθιο ξεχωρίζει για την μεγάλη απορροφητική του ικανότητα αλλά και για την χαμηλή απαιτούμενη θερμοκρασία αναγέννησης του. Επιπρόσθετα χαρακτηρίζεται και από αρκετά μεγάλη «θερμική σταθερότητα». Με τον όρο αυτό εννοείται η ικανότητα του στη διατήρηση των θερμοδυναμικών του χαρακτηριστικών σε μεγάλο εύρος θερμοκρασιών δίχως να υφίσταται σημαντική αποσύνθεση ή αλλοίωση λόγω υψηλών θερμοκρασιών. Αυτό είναι επιθυμητό κατά την διάρκεια των κύκλων απορρόφησης και αναγέννησης ώστε να εξασφαλίζεται η αξιόπιστη και

σταθερή λειτουργία του συστήματος. Αξίζει ακόμα να αναφερθεί ότι το Χλωριούχο Λίθιο (LiCl) προκαλεί μικρότερη διάβρωση στα μέρη του συστήματος συγκριτικά με τα υπόλοιπα μέσα συμβάλλοντας έτσι στην μεγαλύτερη διάρκεια ζωής του συστήματος , ενώ ταυτόχρονα μειώνει σημαντικά και το κόστος συντήρησης.

1.4 Εναλλάκτες Θερμότητας Αντιρροής

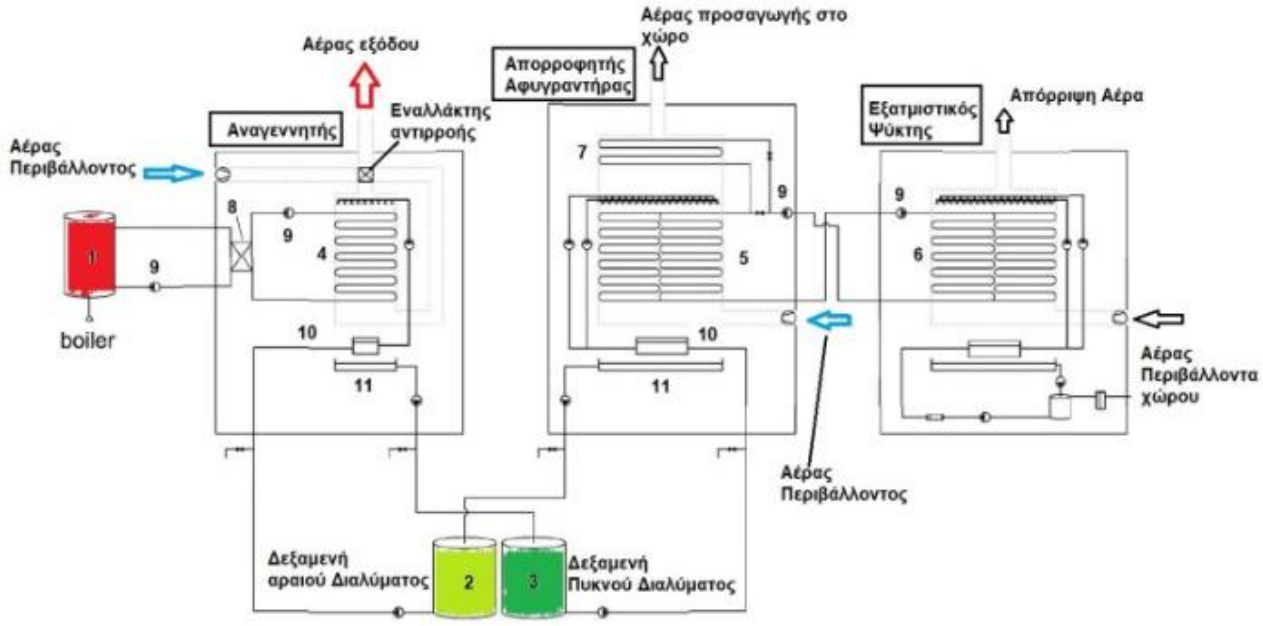
Στα συστήματα ανοιχτού κύκλου χρησιμοποιούνται κυρίως εναλλάκτες θερμότητας αντιρροής. Έρευνες έχουν δείξει ότι οι συγκεκριμένοι εναλλάκτες έχουν την μεγαλύτερη απόδοση όσον αφορά την αφύγγρανση του αέρα. Επίσης αυτό το είδος εναλλακτών επιτρέπει τη μέγιστη συναλλαγή θερμότητας μεταξύ του αέρα και των ψυκτικών μέσων, αυξάνοντας ακόμα περισσότερο την απόδοση του συστήματος. Οι συγκεκριμένοι εναλλάκτες που προορίζονται για συστήματα ψύξης με απορρόφηση ανοικτού κύκλου κατασκευάζονται κυρίως από πλαστικά και πολυμερή ,τα οποία παρουσιάζουν μεγάλη αντοχή στην διάβρωση ενώ ταυτόχρονα έχουν αρκετά μικρότερο βάρος και μέγεθος συγκριτικά με τα υπόλοιπα υλικά όπως το τιτάνιο ή τα κράματα αλουμινίου.



Εικόνα 4. Εναλλάκτης Θερμότητας Αντιρροής

2.ΠΕΡΙΓΡΑΦΗ ΒΙΟΜΗΧΑΝΙΚΟΥ ΨΥΚΤΙΚΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΠΟΡΡΟΦΗΣΗΣ

Τα βασικά μέρη του συστήματος αποτελούν ο Απορροφητής-Αφυγραντήρας, ο Αναγεννητής καθώς και ο (Εξατμιστικός) Πύργος Ψύξης. Ως ψυκτικό μέσο χρησιμοποιείται το νερό ενώ ως απορροφητικό μέσο χρησιμοποιείται το Χλωριούχο Λίθιο σε υγρή φάση ($LiCl$). Επιπλέον στο σύστημα έχουν εγκατασταθεί δύο δεξαμενές αποθήκευσης μία για το πυκνό και μία για το αραιό διάλυμα $H_2O - LiCl$.

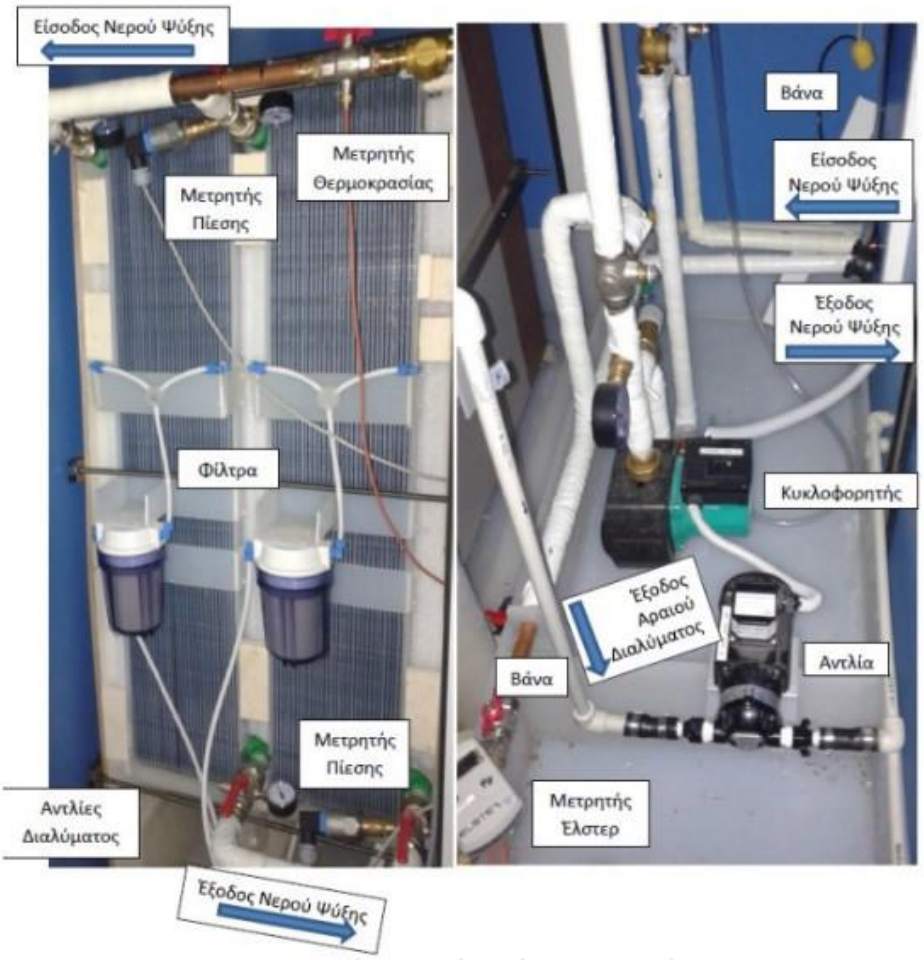


Εικόνα 5.Σχέδιο Πειραματικής Διάταξης

2.1 Απορροφητής

Από την δεξαμενή πυκνού διαλύματος εισέρχεται πυκνό διάλυμα $H_2O - LiCl$ στον εναλλάκτη αντιρροής και ψεκάζεται από τα 2/3 του ύψους του εναλλάκτη με κατεύθυνση από πάνω προς τα κάτω. Έτσι το διάλυμα δεν παρασύρεται από την ροή του αέρα ενώ αυτό το ύψος επιφέρει τα βέλτιστα αποτελέσματα όσον αφορά την αφύγρανση και ψύξη του αέρα. Στη διάταξη θα εισέλθει και αέρας περιβάλλοντος με αντίθετη κατεύθυνση από αυτή του αφυγραντικού μέσου. Με την άμεση επαφή του αέρα και του μέσου επιτυγχάνεται η απορρόφηση της υγρασίας του αέρα. Κατά την ανάμιξη των υδρατμών με το πυκνό διάλυμα απελευθερώνεται θερμότητα ίση με τη λανθάνουσα θερμότητα της συμπύκνωσης του νερού και την ενθαλπία ανάμιξης των δύο στοιχείων. Η θερμότητα αυτή θα προκαλούσε την αύξηση της θερμοκρασίας του

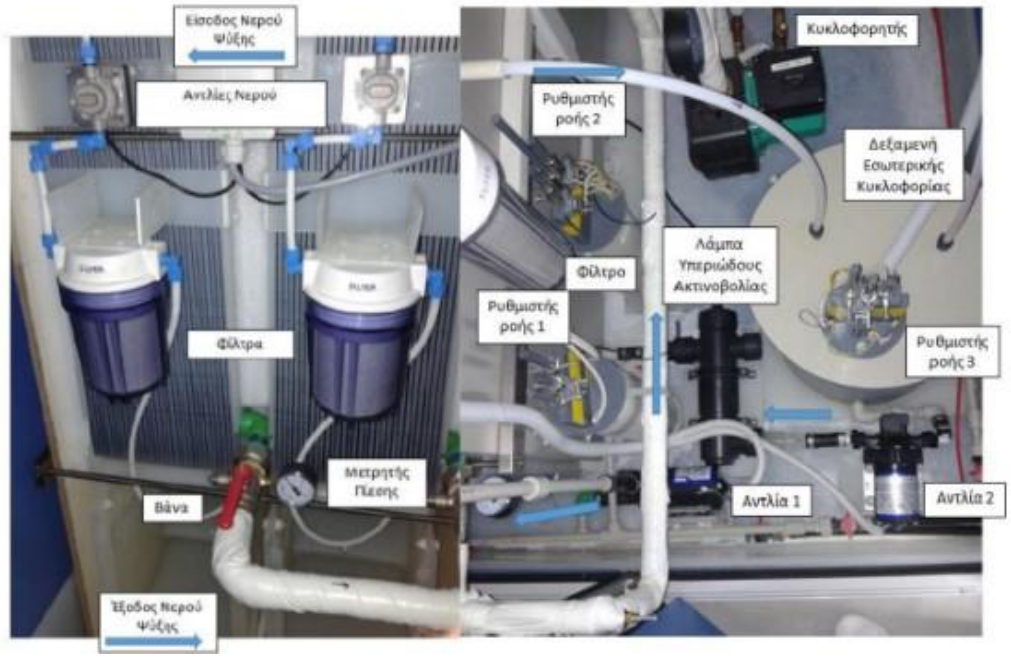
αφυγραντικού μέσου, μειώνοντας την ικανότητα απορρόφησης υδρατμών. Για αυτό το λόγο , για την εξασφάλιση της μείωση της θερμοκρασίας τόσο του διαλύματος όσο και του αέρα στον απορροφητή, κυκλοφορεί στον απορροφητή μέσω οριζόντιων πλακών πολυπροπυλενίου νερό το οποίο απορροφά την εκλυόμενη θερμότητα της διεργασίας. Στο τέλος της διαδικασίας ο ψυχρός και ξηρός αέρας εξέρχεται στο δωμάτιο ενώ το διάλυμα είναι πλέον αραιό και αφού συλλεχθεί στη δεξαμενή συμπυκνωμάτων οδηγείται μέσω μιας αντλίας στην δεξαμενή χαμηλής συγκέντρωσης. Αξίζει να αναφερθεί ότι για την περεταίρω ψύξη του αέρα χρησιμοποιείται ένας επιπλέον εναλλάκτης αέρα-νερού, στον οποίο εισέρχεται νερό από τον εξατμιστικό-ψύκτη προτού εισέλθει στον κύριο εναλλάκτη του απορροφητή. Η παροχή του νερού στον εναλλάκτη ελέγχεται μέσω ρυθμιστικών βανών.



Εικόνα 6. Εσωτερικό Απορροφητή

2.2 Εξατμιστής-Ψύκτης

Ο βασικός του ρόλος είναι η παροχή της απαραίτητης ποσότητας νερού στον απορροφητή. Η διαδικασία περιλαμβάνει την απορρόφηση αέρα από τον περιβάλλοντα χώρο (δωμάτιο) και την είσοδό του στον εναλλάκτη με κατεύθυνση από κάτω προς τα πάνω καθώς και την τροφοδοσία-ψεκασμό νερού δικτύου μέσω κατιονηστήρων για την εξασφάλιση περεταίρω ψύξης. Ταυτόχρονα υπάρχει ξεχωριστό (κλειστό) κύκλωμα στο οποίο κυκλοφορεί το νερό που εξέρχεται και εισέρχεται από και προς τον απορροφητή. Στο τέλος της διαδικασίας ο αέρας εξέρχεται από τον ψύκτη στο περιβάλλον. Θεωρητικά ως ελάχιστη θερμοκρασία για το παραγόμενο νερό ψύξης ορίζεται η θερμοκρασία υγρού βολβού του αέρα του δωματίου. Το νερό δικτύου πρέπει να απολυμανθεί μέσω UV ακτινοβολιών αλλά και ειδικών φίλτρων. Η συγκεκριμένη μονάδα έχει κατασκευαστεί από "αντιμικροβιακά» υλικά με υψηλή αντοχή στη διάβρωση. Το νερό που ψεκάζεται και δεν εξατμίζεται κατά τη διαδικασία συλλέγεται σε ένα δοχείο και οδηγείται σε μία δεξαμενή αποθήκευσης η οποία λειτουργεί ως buffer με σκοπό την άμεση παροχή νερού δικτύου.

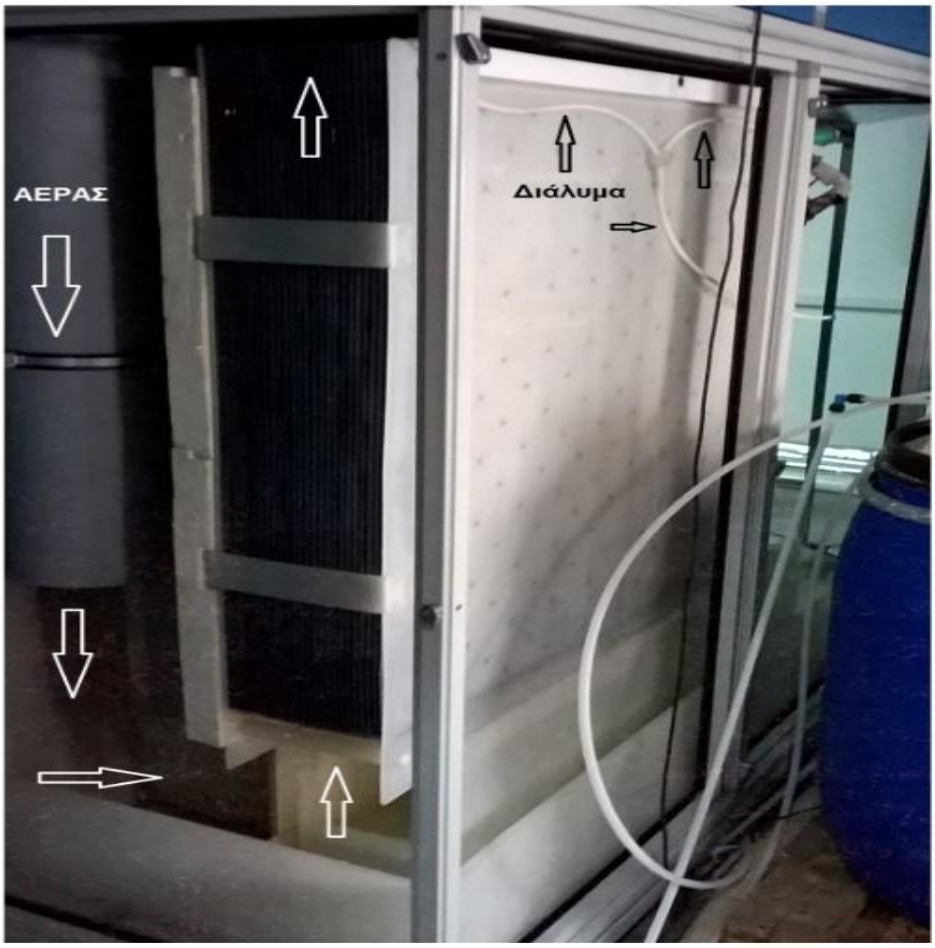


Εικόνα 7. Εσωτερικό Ψύκτη

Εικόνα 8 Εξωτερικό Ψύκτη

2.3 Αναγεννητής

Το αραιό διάλυμα οδηγείται στη μονάδα του αναγεννητή, όπου ψεκάζεται με κατεύθυνση από πάνω προς τα κάτω στα 2/3 του ύψους της μονάδας. Ταυτόχρονα εισέρχεται αέρας από το περιβάλλον με κατεύθυνση αντίθετη από του διαλύματος. Αξίζει να αναφερθεί ότι πριν την εισαγωγή του αέρα στον κύριο εναλλάκτη υπάρχει ένας επιπλέον εναλλάκτης αέρα-αέρα με σκοπό την προθέρμανση του εισαγόμενου αέρα και την μείωση της θερμοκρασίας του εξερχόμενου αέρα με στόχο την αύξηση της απόδοσης του συστήματος. Μέσω οριζόντιων πλακών πολυπροπυλενίου κυκλοφορεί ζεστό νερό το οποίο αποδίδει την απαραίτητη θερμότητα στο διάλυμα. Καθώς αυξάνεται η θερμοκρασία του αραιού διαλύματος , απελευθερώνονται από αυτό υδρατμοί οι οποίοι «αναμιγνύονται» με το ρεύμα του εισερχόμενου αέρα και εξέρχονται από τον αναγεννητή στο περιβάλλον. Καθώς απορροφάται η υγρασία από το αραιό διάλυμα, αυτό αποκτά σταδιακά την αρχική του συγκέντρωση και αφού συλλεχθεί στο δοχείο οδηγείται μέσω μιας αντλίας στη δεξαμενή υψηλής συγκέντρωσης.



Εικόνα 9. Ροή διαλύματος-αέρα στον αναγεννητή

2.4 Δεξαμενές αποθήκευσης

Ο σκοπός της χρήσης ξεχωριστών δεξαμενών αποθήκευσης αφυγραντικού μέσου είναι ο διαχωρισμός των διεργασιών της αφύγγρανσης και της αναγέννησης, έτσι ώστε η μία να είναι ανεξάρτητη της άλλης. Ένα ακόμη βασικό προτέρημα της χρήσης των δύο δεξαμενών αποτελεί το γεγονός ότι η απαραίτητη ενέργεια για την λειτουργία της μονάδας αποθηκεύεται σε μορφή άλατος (LiCL) μεγάλης συγκέντρωσης δίχως να υπάρχουν απώλειες ενέργειας.

Για την αποφυγή της υπερβολικής αύξησης της στάθμης των δεξαμενών έχουν εγκατασταθεί διακόπτες υπερχείλισης οι οποίοι ενεργοποιούνται από ένα «φλοτέρ» και διακόπτουν τη λειτουργία όλων των αντλιών του συστήματος μόλις η στάθμη ξεπεράσει τα προκαθορισμένα όρια.

Εικόνα 10. Δοχεία Διαλύματος και εξωτερικό αναγεννητή

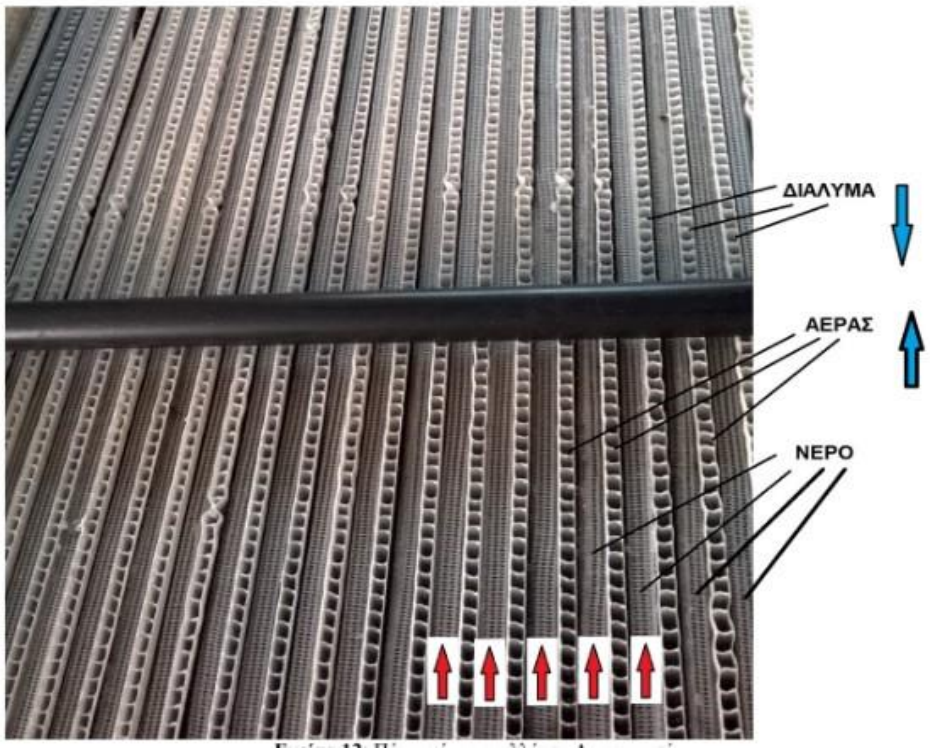
2.5 Boiler

Η απαραίτητη ενέργεια για την θέρμανση του νερού το οποίο κατευθύνεται προς τον Αναγεννητή παρέχεται από το boiler ισχύος 24 kW. Στο boiler υπάρχει εγκατεστημένος υδροστάτης ο οποίος ρυθμίζει την λειτουργία του boiler ανάλογα με τις απαιτήσεις ζεστού νερού. Η κυκλοφορία του νερού από το boiler προς στον αναγεννητή γίνεται μέσω ενός κυκλοφορητή και ύστερα ακολουθεί πλακοειδής εναλλάκτης ώστε το υδραυλικό τμήμα του Αναγεννητή να μην έρχεται σε άμεση επαφή με το υδραυλικό τμήμα της πηγής θερμότητας.

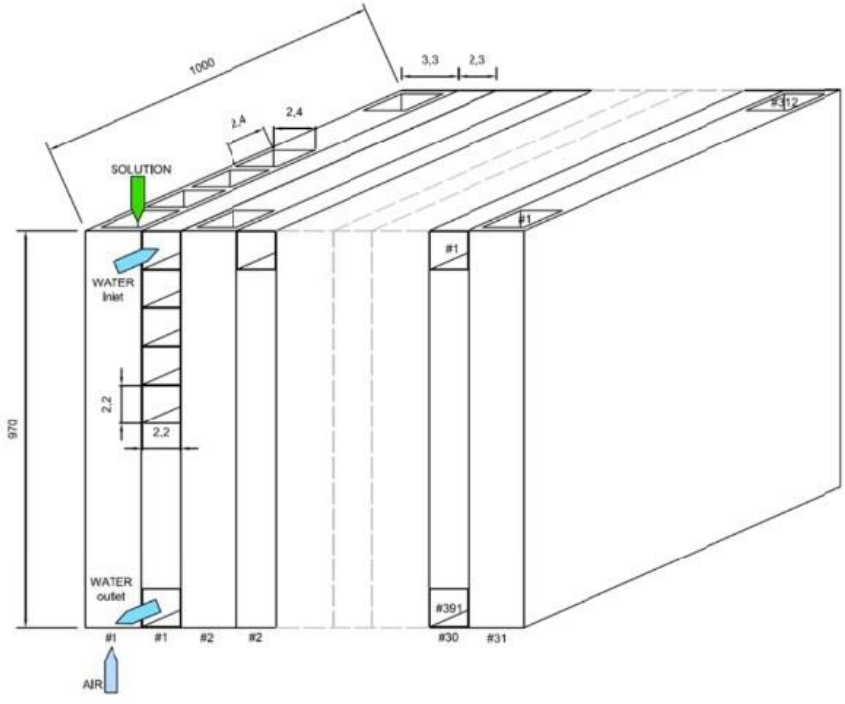
Εικόνα 11. Διάταξη του Boiler

2.6 Γεωμετρικά Χαρακτηριστικά των Διατάξεων

Ο εναλλάκτης του απορροφητή-αφυγραντήρα αποτελείται από 60 μαύρα φύλλα πολυπροπυλενίου με οριζόντιους σωλήνες στους οποίους κυκλοφορεί το νερό. Ακόμα αποτελείται από 62 λευκά φύλλα με κατακόρυφους σωλήνες στους οποίους ρέει το αφυγραντικό μέσο και ο αέρας. Οι διαστάσεις των φύλλων είναι 2.3*1000*970 mm για τα μαύρα και 3.3*1000*970 για τα λευκά. Το κάθε μαύρο φύλλο διαθέτει 391 αγωγούς με διαστάσεις 2.2*2.2 mm και το κάθε λευκό 312 αγωγούς διαστάσεων 2.4*2.4 mm.Ο εναλλάκτης του εξατμιστικού ψύκτη έχει ακριβώς την ίδια γεωμετρία με τον απορροφητή ενώ ο αναγεννητής έχει 25 και 24 μαύρα και λευκά φύλλα πολυπροπυλενίου αντίστοιχα.



Εικόνα 12. Εσωτερικό εναλλάκτη



Εικόνα 13.Γεωμετρία του εναλλάκτη

3.ΠΕΡΙΓΡΑΦΗ ΥΛΙΚΟΥ ΕΞΟΠΛΙΣΜΟΥ

3.1 Αισθητήρες

3.1.1 Αισθητήρας Θερμοκρασίας-PT100

Ο συγκεκριμένος αισθητήρας είναι τύπου μεταβλητής αντίστασης ανάλογα με τη θερμοκρασία.(RTD). Ο αριθμός 100 δείχνει την ονομαστική τιμή της αντίστασης στους 0 βαθμούς Κελσίου ενώ το PT υποδεικνύει την Πλατίνα ως υλικό κατασκευής του σύρματος του αισθητήρα. Σημαντικό είναι να αναφερθεί πως η τιμή της αντίστασης μεταβάλλεται γραμμικά ανάλογα με τη θερμοκρασία. Στην εγκατάσταση χρησιμοποιείται η διάταξη δύο καλωδίων η οποία αποτελεί την απλούστερη μορφή του συγκεκριμένου αισθητήρα.

Συνοπτικά:

PT100 sensor	
Όνομα Κατασκευαστή	KIMO
Όρια Μέτρησης	-50 έως 160 ℃
Σφάλμα Μέτρησης	$\pm$0.3 ℃.
Τρόπος Σύνδεσης	2-wire connection
Σήμα Εξόδου	4-20 mA

Πίνακας 1. Χαρακτηριστικά του αισθητηρίου PT100

Εικόνα 14.Τυπικός Αισθητήρας Pt100

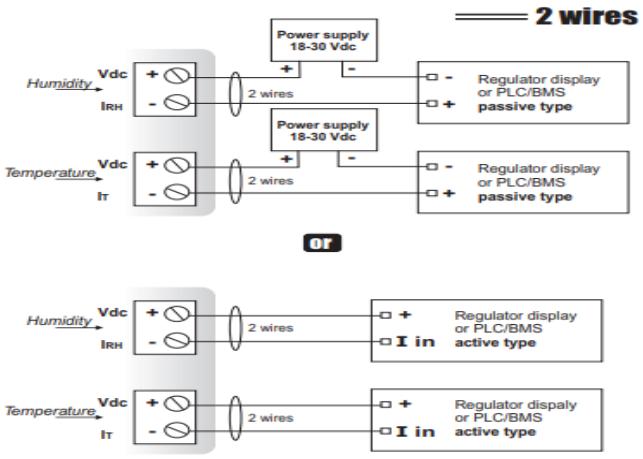
3.1.2 Αισθητήρας Μέτρησης Σχετικής Υγρασίας-Kimo TH100

Ο αισθητήρας Kimo TH100 χρησιμοποιεί συνήθως ένα στοιχείο μέτρησης υγρασίας τύπου πυκνωτή. Αυτό το στοιχείο αλλάζει την χωρητικότητά του ανάλογα με τη σχετική υγρασία του περιβάλλοντος αέρα. Τα ηλεκτρονικά του αισθητήρα μετατρέπουν αυτή την αλλαγή χωρητικότητας σε ηλεκτρικό σήμα που αντιστοιχεί στο επίπεδο της σχετικής υγρασίας.

TH100 sensor	
Όνομα Κατασκευαστή	KIMO
Όρια Μέτρησης	5 έως 95% RH
Σφάλμα Μέτρησης	$\pm 2\%$
Τρόπος Σύνδεσης	2-wire connection
Σήμα Εξόδου	4-20 mA

Πίνακας 2. Χαρακτηριστικά του αισθητηρίου TH100

Εικόνα 15. Αισθητήρας TH100



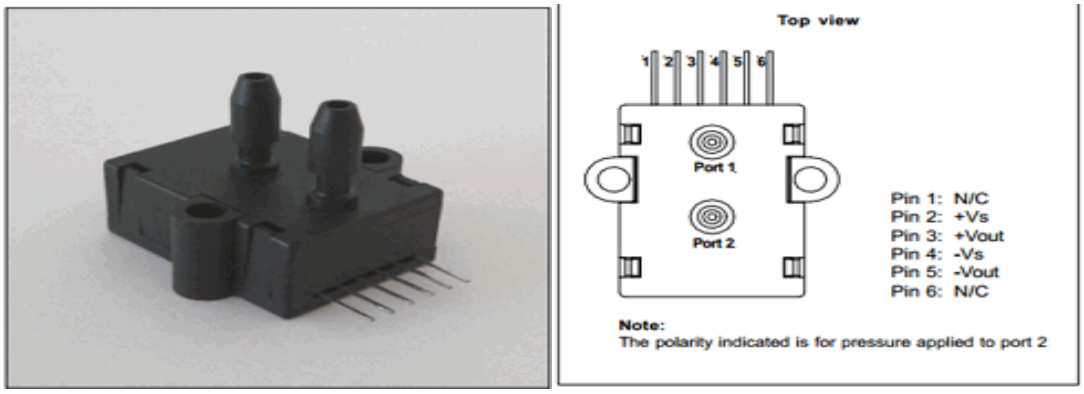
Εικόνα 16. Συνδεσμολογία αισθητήρα

3.1.3 Αισθητήρας Μέτρησης Διαφοράς Πίεσης-DCXL10DS

Ο αισθητήρας DCXL10DS είναι ένας μικρο ηλεκτρομηχανικός αισθητήρας πίεσης που χρησιμοποιείται για την μέτρηση χαμηλής πίεσης με υψηλή ακρίβεια και αξιοπιστία. Ο αισθητήρας διαθέτει δύο θύρες πίεσης που συνδέονται με ξεχωριστούς θαλάμους. Αυτοί οι θάλαμοι εκτίθενται στις πιέσεις που πρέπει να συγκριθούν. Διαθέτει διαχωριστική μεμβράνη, ευαίσθητη στις διαφορές πίεσης η οποία παραμορφώνεται ανάλογα με τη διαφορά πίεσης. Η παραμόρφωση της μεμβράνης ανιχνεύεται από έναν μετασχηματιστή και μετατρέπεται σε ηλεκτρικό σήμα

DCXL10DS	
Όνομα Κατασκευαστή	HONEYWELL
Όρια Μέτρησης	0-10 inches of water
Σφάλμα Μέτρησης	$\pm 0.05\%$
Σήμα Εξόδου	4-20 mA
Τάση Τροφοδοσίας	12 Vdc

Πίνακας 3. Χαρακτηριστικά του αισθητηρίου DCXL10DS



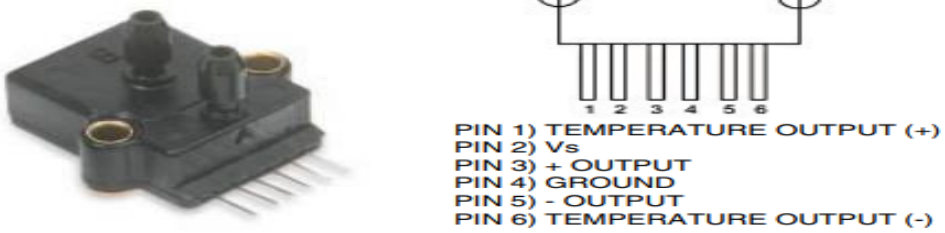
Εικόνα 17.Εξωτερικό Περίβλημα και Pinout του αισθητήρα DCXL10DS

3.1.4 Αισθητήρας Μέτρησης Πίεσης-SCX15AN

Ο αισθητήρας SCX15AN είναι ένας πιεζοηλεκτρικός αισθητήρας χαμηλής πίεσης, σχεδιασμένος για εφαρμογές που απαιτούν ακριβή και αξιόπιστη μέτρηση πίεσης. Η αρχή λειτουργίας του είναι παρόμοια με του DCXL10DS με τη διαφορά ότι διαθέτει μόνο μία θύρα μέτρησης.

SCX15AN	
Όνομα Κατασκευαστή	HONEYWELL
Όρια Μέτρησης	0-150 PSI
Σφάλμα Μέτρησης	$\pm 0.25\%$
Σήμα Εξόδου	4-20 mA
Τάση Τροφοδοσίας	12 Vdc

Πίνακας 4. Χαρακτηριστικά του αισθητηρίου SCX15AN



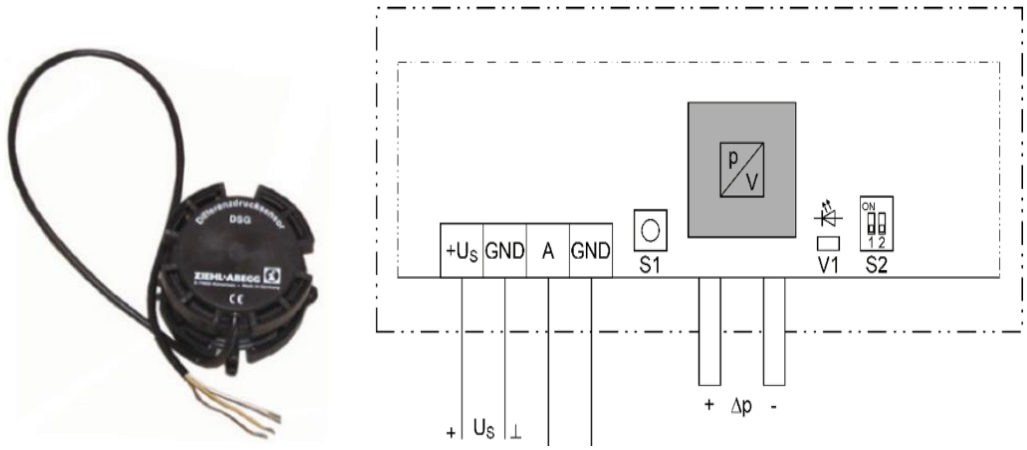
Εικόνα 18.Εξωτερικό Περίβλημα και Pinout του αισθητήρα SCX15AN

3.1.5 Μετρητές Διαφοράς Πίεσης-DSG500,DSG2000

Η λειτουργία τους είναι παρόμοια με τον αισθητήρα DCXL10DS

Sensor	DSG500	DSG2000
Όνομα Κατασκευαστή	Delta Electronics	
Όρια Μέτρησης	0-500 Pascal	0-2000 Pascal
Σήμα Εξόδου	0-10 Vdc	0-10 Vdc
Τάση Τροφοδοσίας	24 Vdc	24 Vdc

Πίνακας 5. Χαρακτηριστικά του αισθητηρίου DSG2000 και DSG500



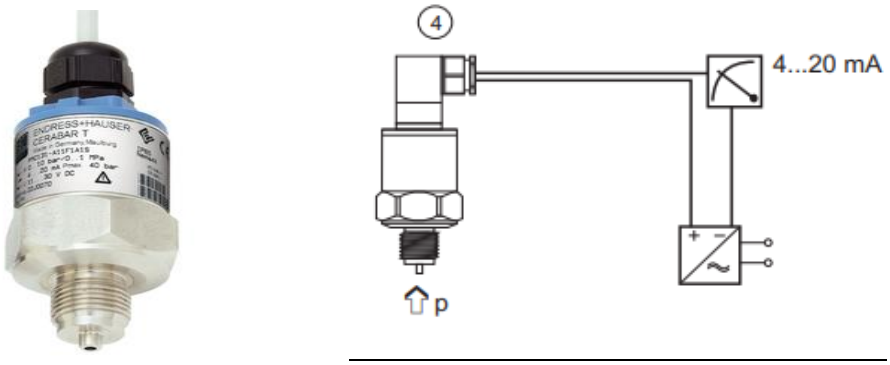
Εικόνα 19.Μορφή και Οδηγίες Συνδεσμολογίας αισθητήρα DSG

3.1.6 Αισθητήρας Μέτρησης Πίεσης-T PMC131

Παρόμοια λειτουργία με τον αισθητήρα SCX15AN

TPMC131	
Όνομα Κατασκευαστή	CERABAR
Όρια Μέτρησης	1.5-600 PSI
Όριο υπερπίεσης	900 PSI
Σφάλμα Μέτρησης	$\pm 0.5\%$
Σήμα Εξόδου	4-20 mA
Τάση Τροφοδοσίας	24 Vdc

Πίνακας 6. Χαρακτηριστικά του αισθητηρίου TPMC131



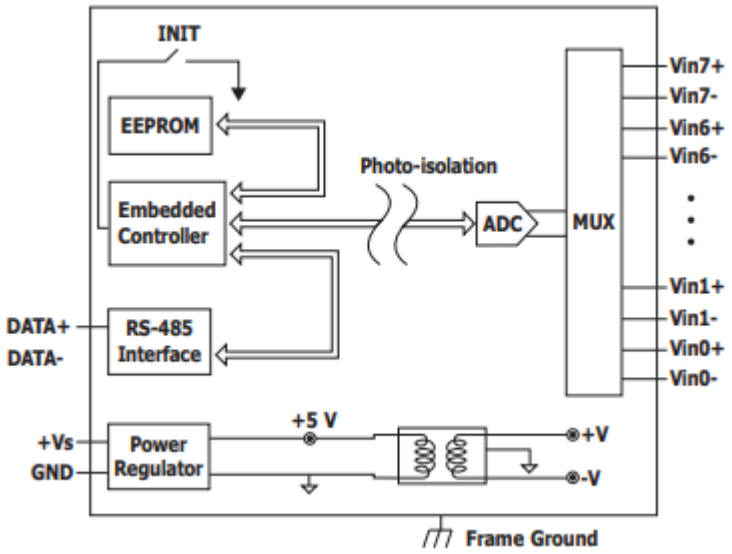
Εικόνα 20. Εξωτερικό περίβλημα και οδηγίες συνδεσμολογίας αισθητήρα PMC131

3.2 Modules Λήψης Δεδομένων

3.2.1 I7018

Το συγκεκριμένο module διαθέτει 8 κανάλια Αναλογικής εισόδου ρεύματος, τάσης ή και θερμοκρασίας μέσω θερμοζεύγους.

Εικόνα 21. Περίβλημα και εσωτερικό του module I7018



Εικόνα 22. Ηλεκτρονικό Διάγραμμα του I7018

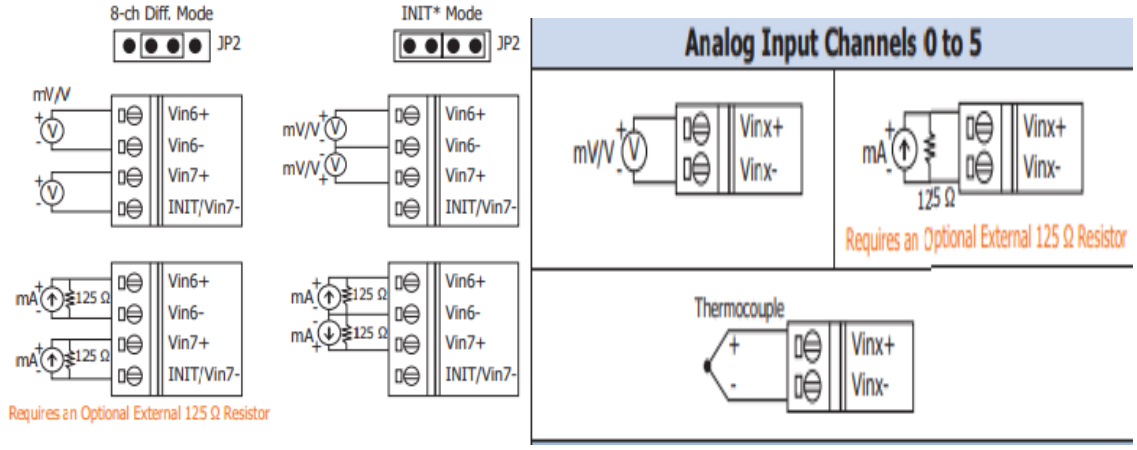
Τα βασικά τεχνικά χαρακτηριστικά του συγκεκριμένου module συγκεντρώνονται στον παρακάτω πίνακα

Module	I7018
BAUD RATE	1200-115200 bps
DATA FORMAT	(N,8,1), (N,8,2), (E,8,1), (E,8,2)
PORTS	Rs-485
PROTOCOL	DCON
INPUT RANGE	10-30 Vdc
OVERVOLTAGE PROTECTION	$\pm$120 Vdc
DIMENSIONS	76x120x42($WxLxH$)
CONSUMPTION	1 W
CHANNELS	8 DIFFERENTIAL/6 DIFFERENTIAL 2 SINGLE ENDED(INIT)
TYPE	Voltage, Current, Thermocouple
Voltage Range	$\pm 15mV \pm 50mV \pm 100mV \pm 500\ mV \pm 1\ V \pm 2.5\ V$
Current Range	$\pm 20mA$ $0-20mA$ $4-20mA$ $(requires\ an\ optional\ external\ 125\ \Omega\ resistor)$
Accuracy	0.1% of Full span Range
Resolution-Sampling Rate	16-bit-10 Hz

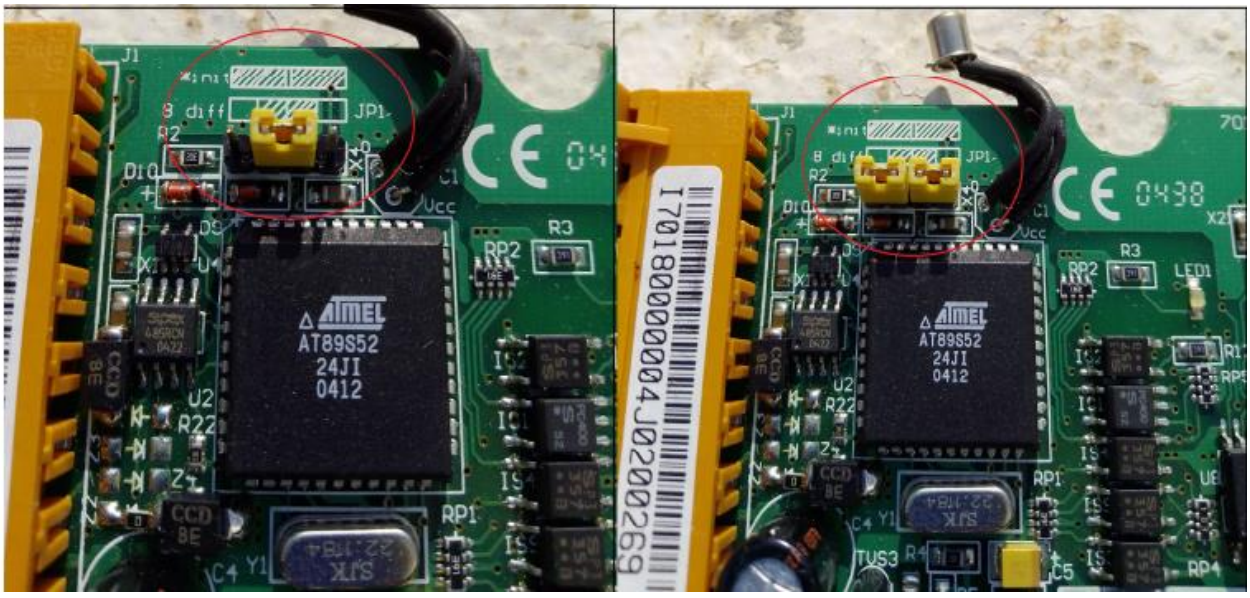
Πίνακας 7. Χαρακτηριστικά του module I7018

Το συγκεκριμένο module δίνει την δυνατότητα διαφορικής λειτουργίας ή λειτουργίας INIT (Κοινού σημείου) στα κανάλια 6 και 7 ,ενώ τα κανάλια 0 έως 5 είναι ρυθμισμένα

μόνο για διαφορική λειτουργία. Στη διαφορική λειτουργία γίνεται μέτρηση της διαφοράς τάσης των δύο άκρων κάθε καναλιού ενώ στη λειτουργία κοινού σημείου γίνεται μέτρηση της διαφοράς τάσης μίας εισόδου και μίας κοινής γείωσης. Στη διαφορική λειτουργία η τιμή της διαφοράς τάσης μπορεί να μετατραπεί σε μέτρηση της έντασης του ρεύματος με χρήση μίας εξωτερικής αντίστασης 125 Ω. Το βασικό πλεονέκτημα της διαφορικής λειτουργίας είναι η ακρίβεια και για αυτό το λόγο θα πρέπει να χρησιμοποιηθεί. Για να γίνει αυτό θα πρέπει να γεφυρωθούν τα δύο από τα 4 pins που βρίσκονται εσωτερικά του module όπως φαίνεται και στις παρακάτω εικόνες.



Εικόνα 23. Επιλογές συνδεσμολογίας αισθητήρων και λειτουργίας του module



Εικόνα 24. Οδηγίες δημιουργίας "εσωτερικής γέφυρας" για διαφορική λειτουργία (αριστερά) ή λειτουργία INIT (δεξιά)

3.2.2 I7019R

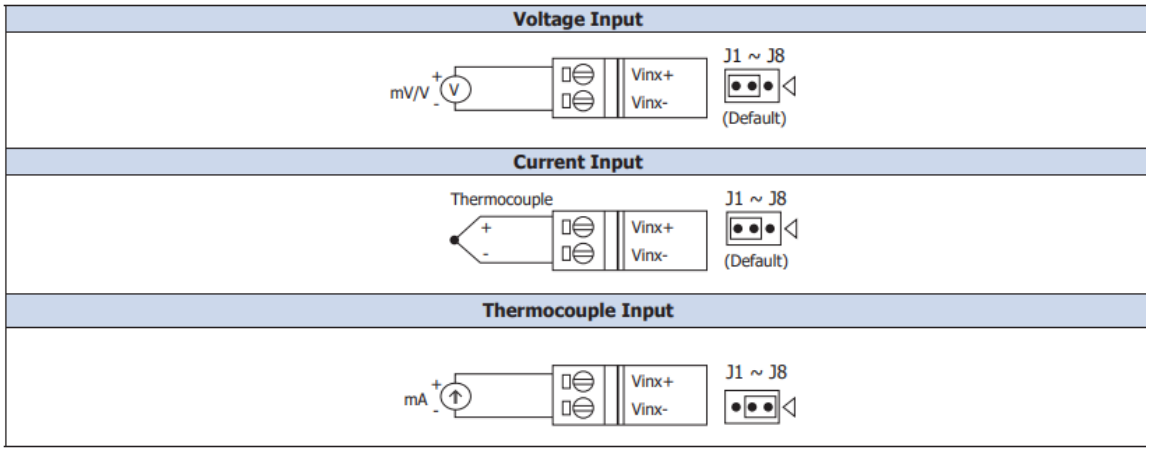
Εικόνα 25.I7019R

Το συγκεκριμένο module έχει σχεδόν τα ίδια τεχνικά χαρακτηριστικά με το I7018 αλλά δίνει την δυνατότητα μέτρησης τάσης ή ρεύματος και στα 8 κανάλια ανάλογα όπως φαίνεται παρακάτω

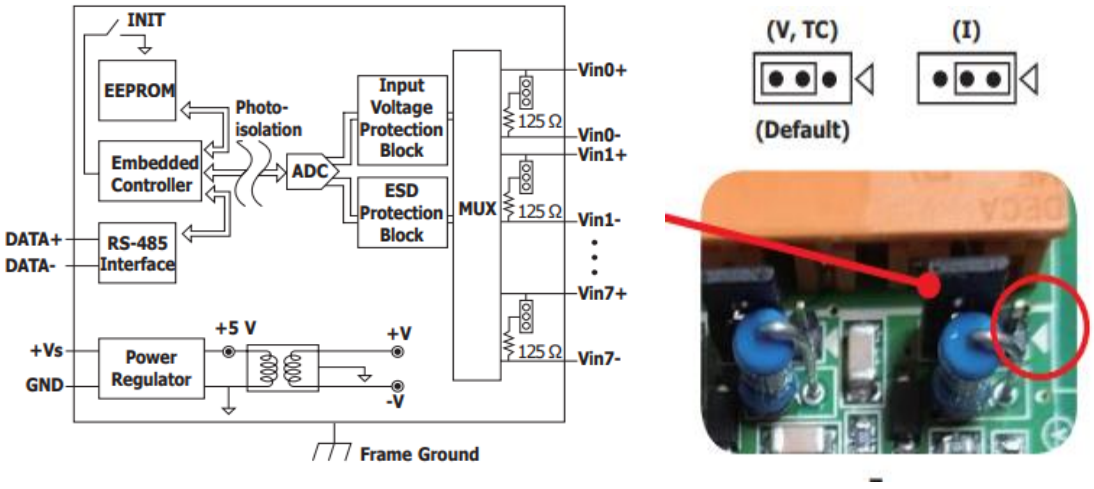
Module	I7019R
BAUD RATE	1200-115200 bps
DATA FORMAT	(N,8,1), (N,8,2), (E,8,1), (E,8,2)
PORTS	Rs-485
PROTOCOL	DCON
INPUT RANGE	10-30 Vdc
OVERVOLTAGE PROTECTION	± 200 Vdc
DIMENSIONS	76x120x42($WxLxH$)
CONSUMPTION	1 W
CHANNELS	8 DIFFERENTIAL/6 DIFFERENTIAL 2 SINGLE ENDED(INIT)
TYPE	Voltage, Current, Thermocouple
Voltage Range	$\pm 15mV \pm 50mV \pm 100mV \pm 500\ mV \pm 1\ V \pm 2.5\ V \pm 5V \pm 10V$
Current Range	$\pm 20mA$ $0-20mA$ $4-20mA$ $(requires\ an\ optional\ external\ 125\ \Omega\ resistor)$
Accuracy	0.1% of Full span Range

Resolution-Sampling Rate	16-bit-8 Hz

Πίνακας 8. Χαρακτηριστικά του module I7019R



Εικόνα 26. Επιλογές Λειτουργίας των Καναλιών



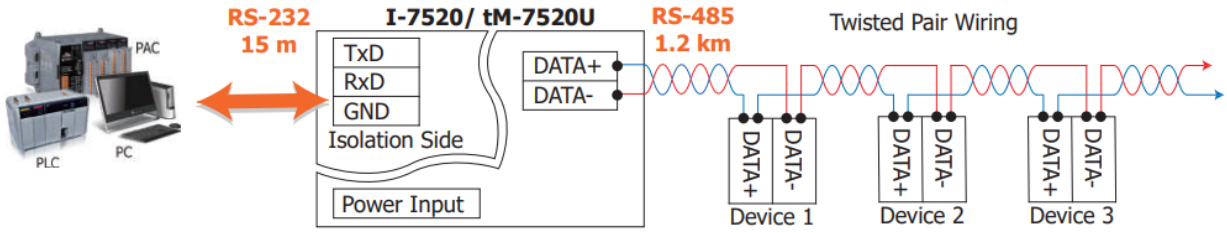
Εικόνα 27.Ηλεκτρονικό Διάγραμμα I7019R και εσωτερική γέφυρα για ρύθμιση λειτουργίας του Module

3.2.3 I7520-Συλλέκτης Δεδομένων Και μετατροπέας Rs485 σε Rs232

Εικόνα 28.Module I7520

MODULE	I7520
POWER SUPPLY	10-30 Vdc
Baud Rate	Up to 115200 bps
Max distance	1.2km @115200 bps 15m for Rs232 cable

Πίνακας 9. Χαρακτηριστικά του module I7520



Εικόνα 29. Συνδεσμολογία για μεταφορά δεδομένων σύμφωνα με το πρότυπο RS485

Λίγα λόγια για το πρότυπο Rs485:

Αποτελεί ένα πρότυπο για σειριακή επικοινωνία και μετάδοση δεδομένων. Μετράται η διαφορά τάσης μεταξύ δύο αγωγών αντί η διαφορά τάσης ενός αγωγού συγκριτικά με τη Γη. Η χρήση της διαφορικής τάσης για τη μετάδοση σημάτων, το καθιστά ανθεκτικό σε ηλεκτρομαγνητικές παρεμβολές και θόρυβο. Στηρίζει μόνο τη σειριακή επικοινωνία (ένα προς ένα ή πολλαπλά προς ένα), και χρησιμοποιεί συνήθως twisted pair καλώδιο για τη σύνδεση των συσκευών.

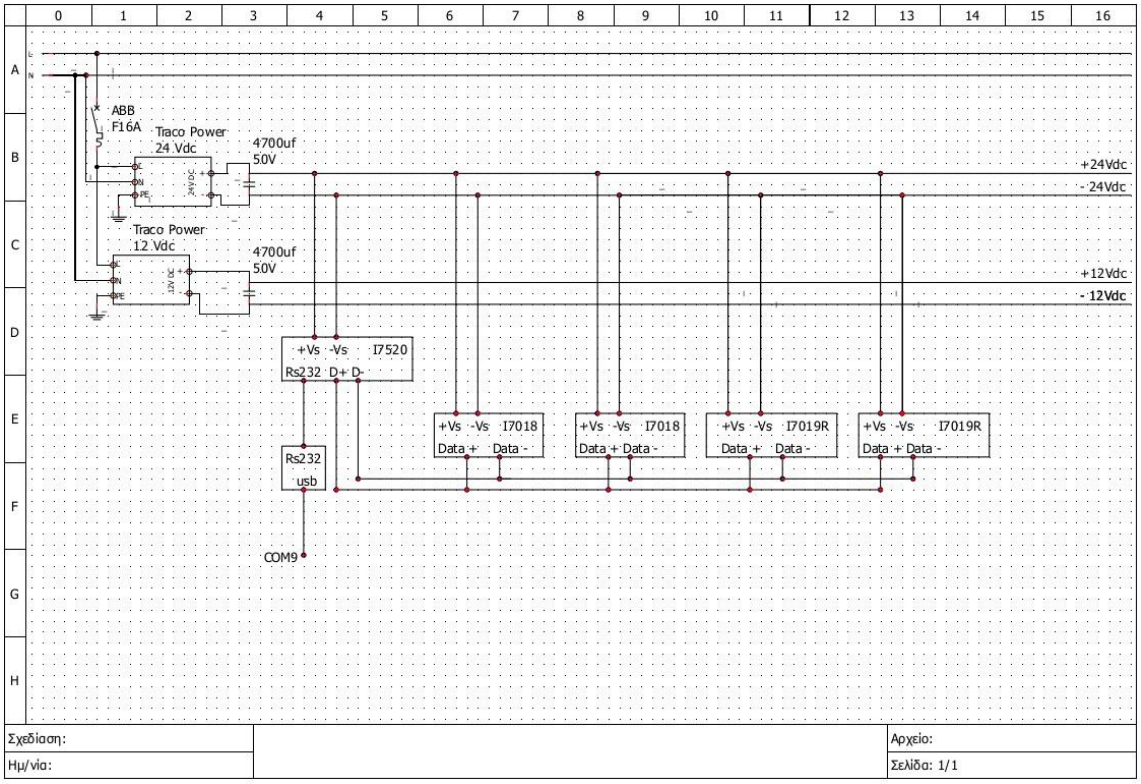
Το καλώδιο τύπου twisted pair αποτελείται από δύο αγωγούς που έχουν περιστραφεί μαζί σε ζεύγη. Μπορεί να διαθέτει και ένα πρόσθετο επίπεδο προστασίας, όπως μια θωράκιση από μεταλλικό υλικό, για να μειώσει τις ηλεκτρομαγνητικές παρεμβολές.

Λίγα λόγια για το πρότυπο Rs232:

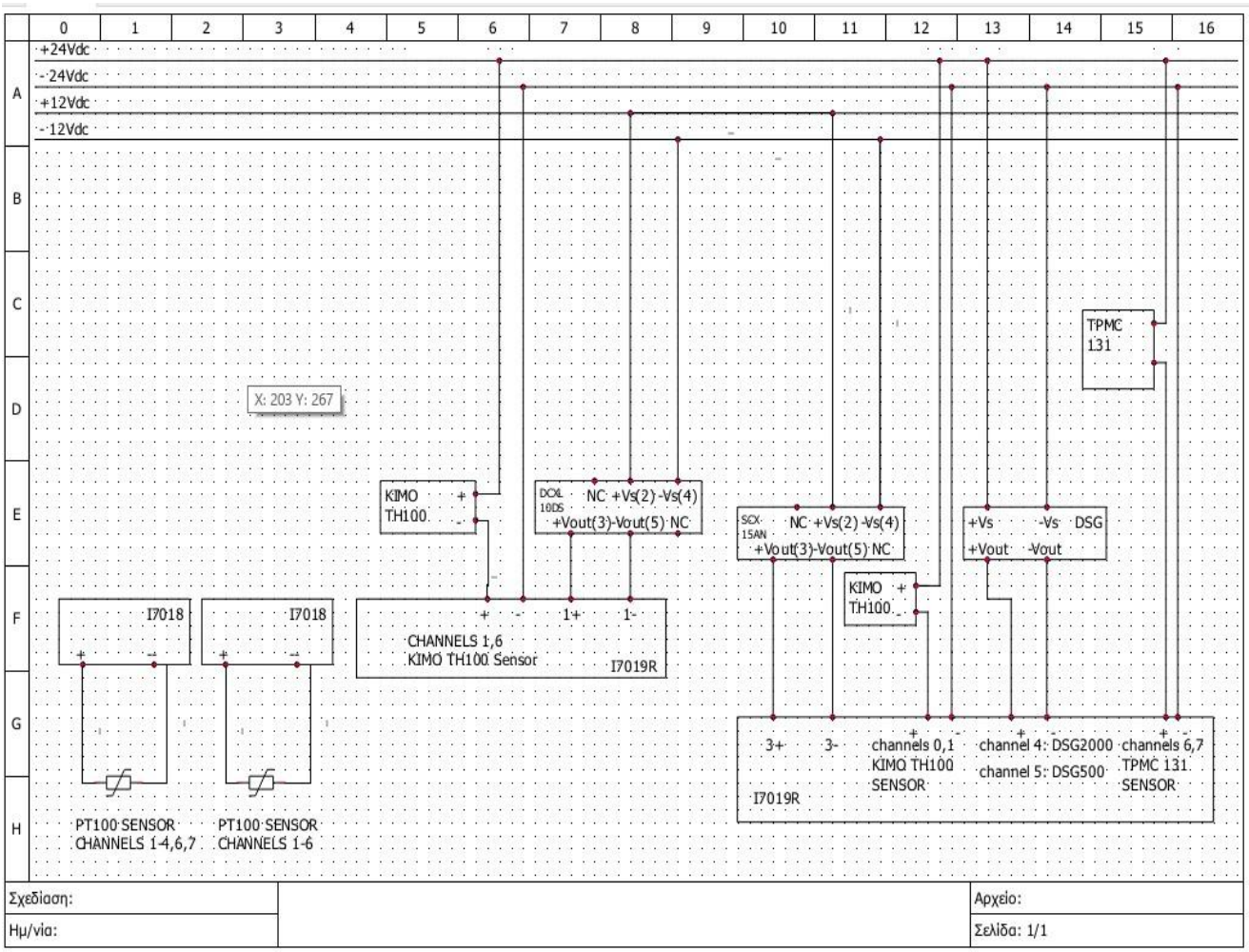
Το RS-232 χρησιμοποιεί σειριακή επικοινωνία, που σημαίνει ότι τα δεδομένα μεταδίδονται ένα bit τη φορά μέσω ενός μόνο καναλιού επικοινωνίας. Το RS-232 δεν καθορίζει ένα αυστηρό πρωτόκολλο επικοινωνίας. Υποστηρίζει τη βασική μεταφορά δεδομένων και απαιτεί εξωτερική συμφωνία για τη μορφοποίηση των δεδομένων και τον έλεγχο ροής. Απαιτείται ρύθμιση της μορφής δεδομένων (starting bit,stop bit, parity bit). Χρησιμοποιεί καλώδιο 9 ή 25 αγωγών (DB9 DB25). Κύριοι αγωγοί είναι οι TX,RX,GND(transmit, receive, ground). Ακόμα περιέχονται αγωγοί ρύθμισης ροής δεδομένων.

Για την ολοκλήρωση του «υλικού» λήψης μετρήσεων χρησιμοποιείται ένας απλός μετατροπέας rs232 σε usb ώστε να γίνει σύνδεση με την θύρα του υπολογιστή επομένως.

Στο σχεδιάγραμμα της παρακάτω εικόνας φαίνεται ο «γενικός» ασφαλειοδιακόπτης του πίνακα καθώς και τα τροφοδοτικά 24 Vdc και 12 Vdc Επιπλέον στο σχέδιο φαίνονται οι δύο πυκνωτές που χρησιμοποιούνται ως φίλτρα για την μείωση του θορύβου και την εξασφάλιση σταθερού μέτρου συνεχούς τάσης.

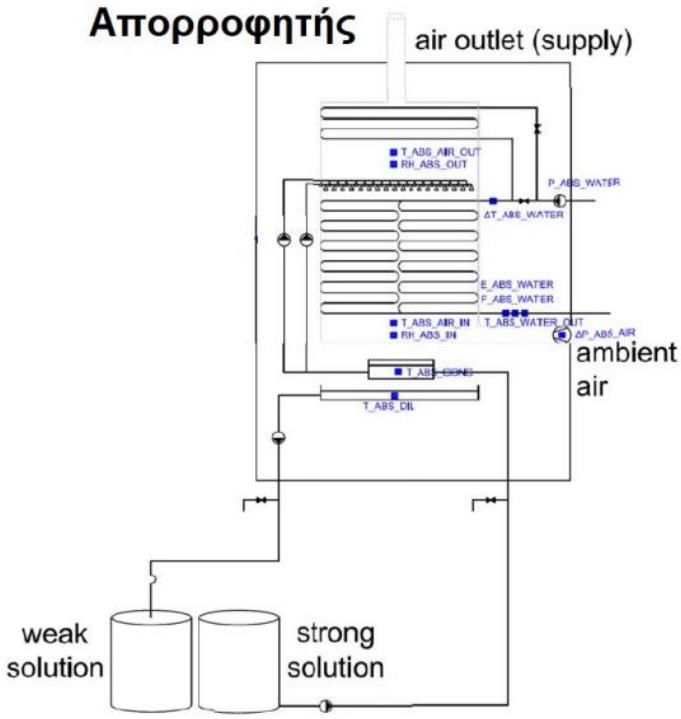


Εικόνα 30. Σχεδιάγραμμα Συνδεσμολογίας

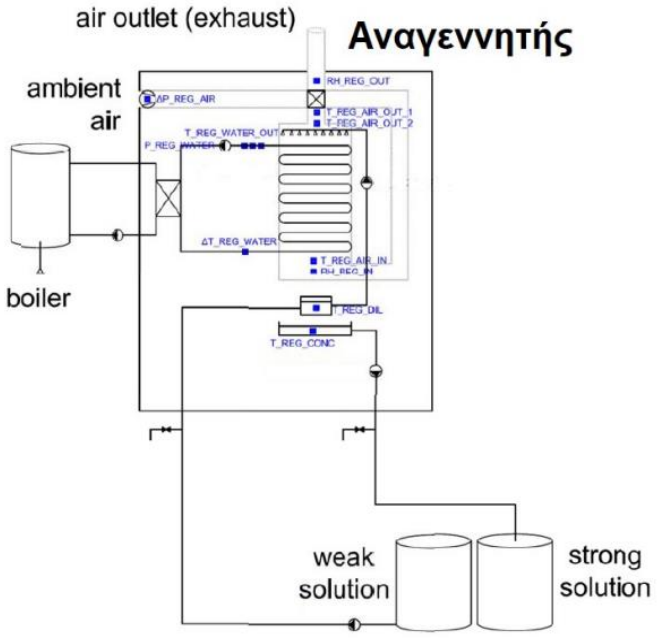


Εικόνα 31.Συνδεσμολογία αισθητήρων-modules

Οι θέσεις των αισθητήρων φαίνονται στο σχέδιο που ακολουθεί:



Εικόνα 32.Θέσεις Αισθητήρων στον Απορροφητή



Εικόνα 33.Θέσεις Αισθητήρων στον Αναγεννητή

3.3 Πρωτόκολλο Επικοινωνίας DCON

Το συγκεκριμένο πρωτόκολλο αποτελεί ένα απλό πρωτόκολλο τύπου request-reply βασισμένο στη γλώσσα ASCII.Η εντολή στέλνεται σε δεκαεξαεδική μορφή από το χρήστη σύμφωνα με την παρακάτω διάταξη

Leading Character	Module Adress	Command	[CHKSUM]	CR

Πίνακας 10.Μορφή Πρωτοκόλλου Dcon

Leading Character: Είναι ο χαρακτήρας που καθορίζει τον τύπο της εντολής

Module Adress: Το κάθε module έχει τη δική του «διεύθυνση» η οποία είναι ένας αριθμός σε μορφή ASCII. Η εντολή στέλνεται από το χρήστη σε όλα τα modules αλλά μόνο το module με την καθορισμένη διεύθυνση (από το χρήστη) θα απαντήσει.

CHKSUM: Ο έλεγχος ακεραιότητας (checksum) είναι μια μέθοδος επαλήθευσης των δεδομένων που μεταφέρονται ή αποθηκεύονται. Χρησιμοποιείται για να διασφαλίσει ότι τα δεδομένα δεν έχουν υποστεί αλλοίωση κατά τη διάρκεια της μεταφοράς ή αποθήκευσης. Αποτελείται από δύο χαρακτήρες.

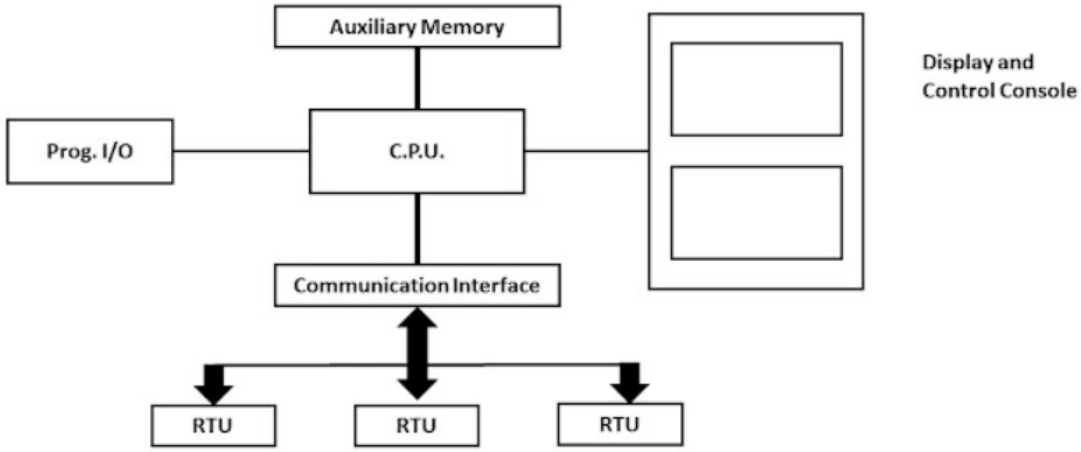
Προστίθενται όλοι οι χαρακτήρες της εντολής και εφαρμόζεται η μάσκα 0FFh. Εάν το αποτέλεσμα του αθροίσματος είναι μεγαλύτερο από τον αριθμό που υποδεικνύει η μάσκα (το 255) τότε το αποτέλεσμα θα είναι το υπόλοιπο της διαίρεσης του αθροίσματος με τον αριθμό αυτό. Έτσι εξασφαλίζεται ότι το checksum θα «πιάνει χώρο» ίσο με 1 byte. Το αποτέλεσμα της διαίρεσης αναγράφεται στην εντολή.

Ο όρος checksum δεν αναλύεται περεταίρω καθώς δε χρησιμοποιείται στην εγκατάσταση.

CR: Το Carriage Return χρησιμοποιείται για να "μεταφέρει" τον κέρσορα πίσω στην αρχή της γραμμής, χωρίς να προχωρήσει στην επόμενη γραμμή. Στην κωδικοποίηση ASCII, το Carriage Return αντιπροσωπεύεται με τον αριθμό 13 (ή 0x0D σε δεκαεξαδική μορφή

4.ΣΥΣΤΗΜΑΤΑ SCADA

4.1 Εισαγωγή στα συστήματα Scada



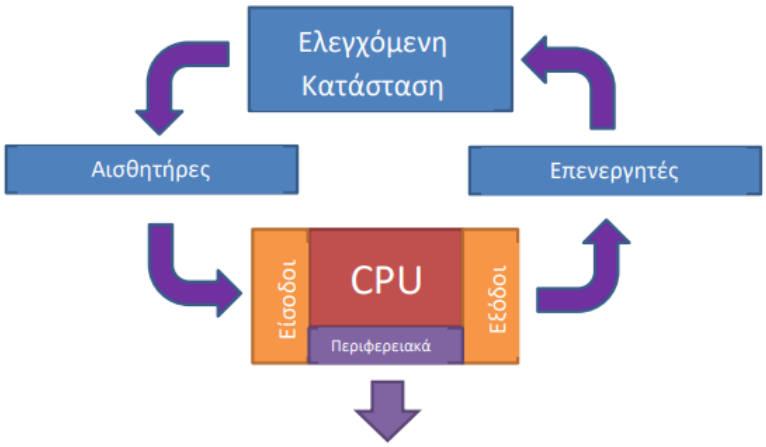
Εικόνα 34. Ροή Δεδομένων σε σύστημα Scada

Τα SCADA είναι συστήματα λογισμικού και υλικού που επιτρέπουν τον έλεγχο διεργασιών σε πραγματικό χρόνο αλλά και τη συλλογή δεδομένων από διάφορα απομακρυσμένα σημεία. Η βασική λειτουργία των συστημάτων αυτών περιλαμβάνει τη συλλογή δεδομένων από αισθητήρες και συσκευές, την επεξεργασία αυτών των δεδομένων και την προβολή τους σε χειριστές μέσω γραφικών διεπαφών.

Βασικά Στοιχεία των Συστημάτων SCADA:

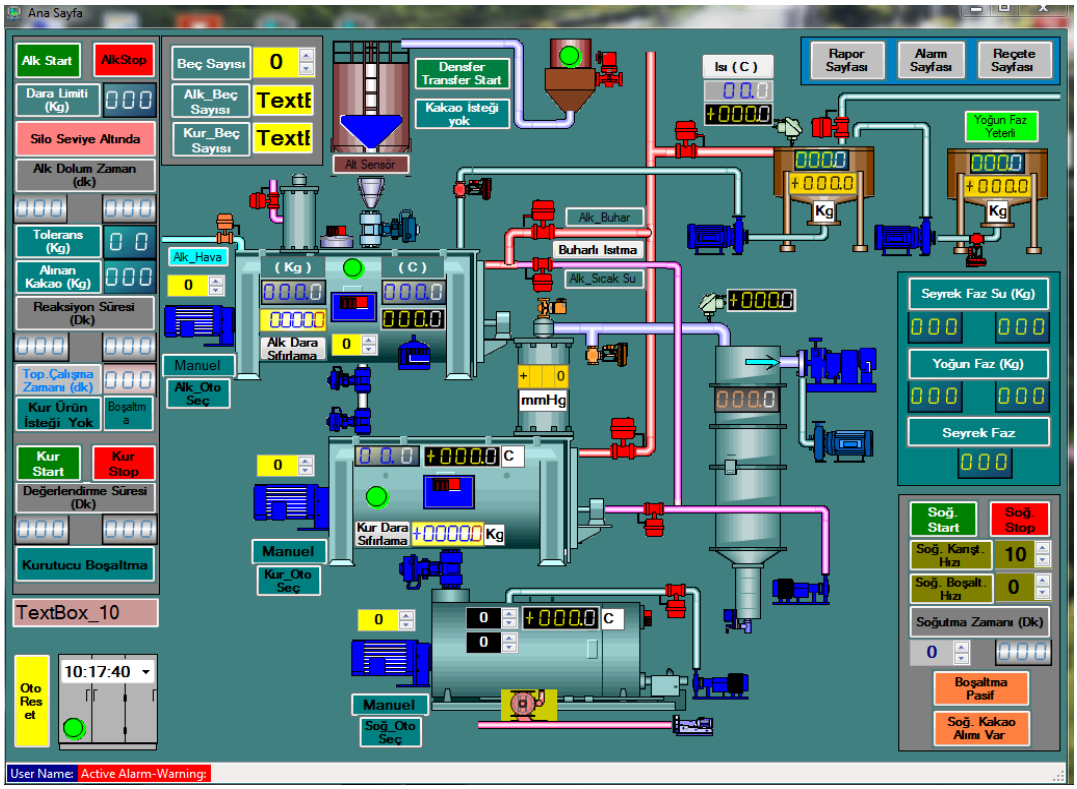
1. RTUs (Remote Terminal Units): Αυτές οι μονάδες συλλέγουν δεδομένα από αισθητήρες και μεταδίδουν τις πληροφορίες στο κεντρικό σύστημα.
2. PLC (Programmable Logic Controllers): Οι προγραμματιζόμενοι λογικοί ελεγκτές είναι βιομηχανικοί υπολογιστές που χρησιμοποιούνται για τον έλεγχο αυτοματισμών. Μπορούν να εκτελούν περίπλοκους ελέγχους και να συνδέονται με το σύστημα SCADA.

Εικόνα 35.Τυπικές Μορφές Plc και οθόνης ελέγχου



Εικόνα 36. Διάγραμμα Ροής συστήματος ελέγχου με χρήση PLC(CPU)

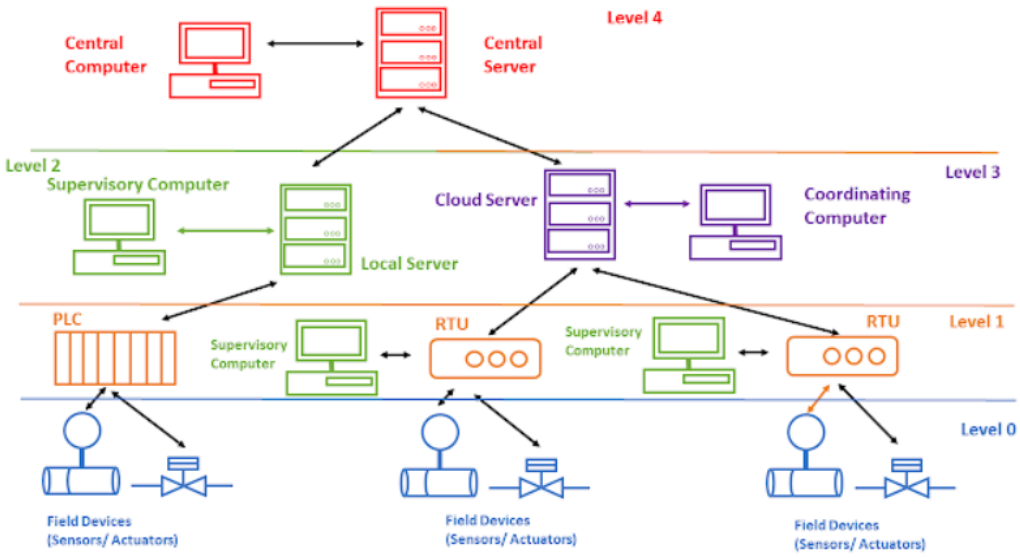
3. HMI (Human-Machine Interface): Αυτές οι διεπαφές επιτρέπουν στους χειριστές να αλληλεπιδρούν με το σύστημα SCADA, να παρακολουθούν τη λειτουργία του συστήματος και να λαμβάνουν αποφάσεις σε πραγματικό χρόνο



Εικόνα 37.Οθόνη αλληλεπίδρασης ανθρώπου-μηχανής

4. Επικοινωνιακά Δίκτυα: Τα δίκτυα αυτά είναι υπεύθυνα για τη μεταφορά δεδομένων μεταξύ των RTUs, PLC και των κεντρικών συστημάτων. Χρησιμοποιούνται διάφορες τεχνολογίες, όπως ενσύρματα και ασύρματα δίκτυα, για τη διασφάλιση της αξιόπιστης μεταφοράς των δεδομένων.

5. Κεντρικοί Υπολογιστές και Λογισμικό: Οι κεντρικοί υπολογιστές λαμβάνουν, επεξεργάζονται και αποθηκεύουν τα δεδομένα που συλλέγονται από τις RTUs και τους PLC. Το λογισμικό SCADA αναλύει τα δεδομένα αυτά και παρέχει ειδοποιήσεις, αναφορές και γραφήματα στους χειριστές.



Εικόνα 38.Δομή Δικτύου επικοινωνίας σε συστήματα ελέγχου

Το βασικό πλεονέκτημα των συστημάτων scada είναι η συμβολή στην αύξηση της αποδοτικότητας των ελεγχόμενων διεργασιών μέσω της συνεχούς παρακολούθησης και αυτόματης ρύθμισης των παραμέτρων λειτουργίας τους. Ένα ακόμα σημαντικό τους πλεονέκτημα αποτελεί η μεγάλη συμβολή τους στον παράγοντα εργασιακή ασφάλεια, μέσω της άμεσης διάγνωσης βλαβών και ανωμαλιών κατά τη λειτουργία των ελεγχόμενων συστημάτων. Σημαντική παράλειψη θα αποτελούσε η μη αναφορά στην ευκολία χρήσης και στην ευελιξία των συστημάτων scada. Συγκεκριμένα τα παραπάνω συστήματα δίνουν την δυνατότητα δημιουργίας ενός γραφικού περιβάλλοντος προσαρμοσμένο στις ανάγκες κάθε χρήστη. Σε αυτό ο χρήστης βλέπει σε πραγματικό χρόνο τη λειτουργία του συστήματος και μπορεί πολύ εύκολα να αλλάξει ή να παρακολουθήσει οποιαδήποτε ρύθμιση ή παράμετρο λειτουργίας.

4.2 Λογισμικό Labview

Το LabVIEW (Laboratory Virtual Instrument Engineering Workbench) είναι ένα ισχυρό περιβάλλον ανάπτυξης «λογισμικού» που δημιουργήθηκε από την National Instruments. Χρησιμοποιείται ευρέως για την κατασκευή προσαρμοσμένων εφαρμογών δοκιμών, μέτρησης και ελέγχου σε διάφορους τομείς, όπως η μηχανική, η επιστήμη και η βιομηχανία. Η μοναδική γραφική προσέγγιση του LabVIEW στον προγραμματισμό το ξεχωρίζει από τις παραδοσιακές γλώσσες προγραμματισμού, καθιστώντας το ένα ευέλικτο και φιλικό προς το χρήστη εργαλείο για μηχανικούς και επιστήμονες.

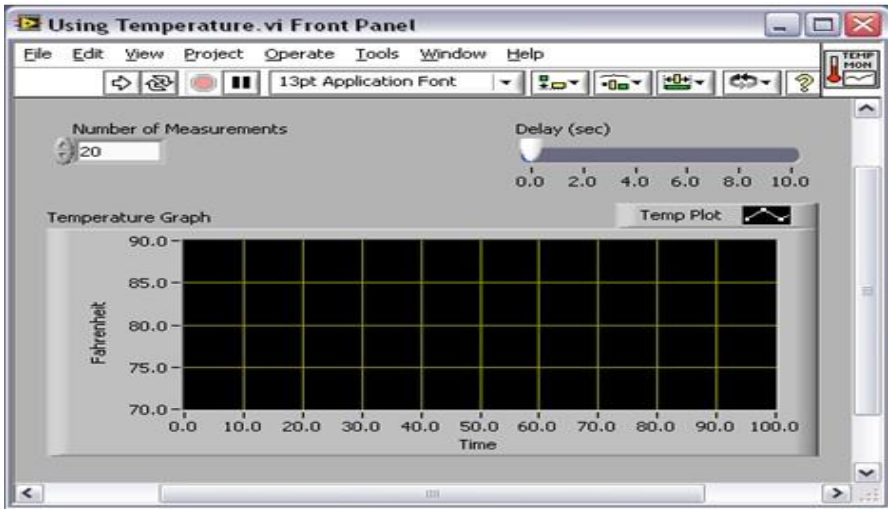
4.2.1 Κύρια Χαρακτηριστικά του LabVIEW

1. **Γραφικός Προγραμματισμός**:

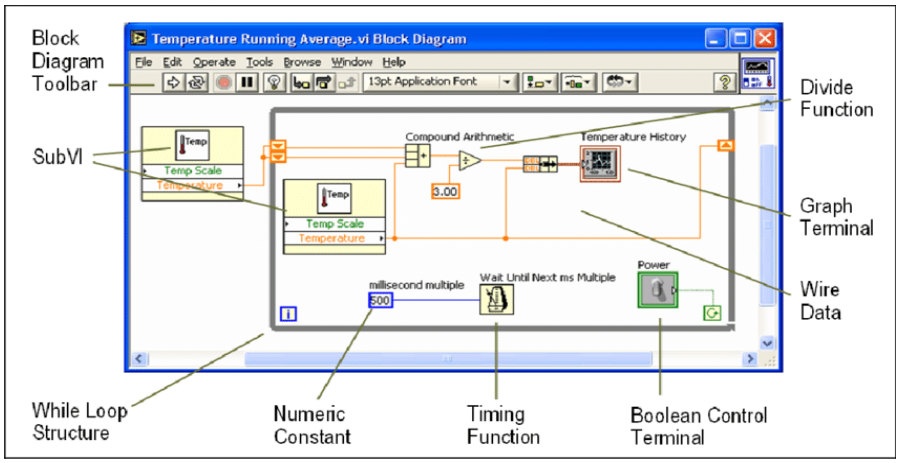
Το LabVIEW χρησιμοποιεί μια γραφική γλώσσα προγραμματισμού γνωστή ως "G". Σε αντίθεση με τον παραδοσιακό προγραμματισμό, όπου οι προγραμματιστές γράφουν κώδικα γραμμή προς γραμμή, οι χρήστες του LabVIEW δημιουργούν προγράμματα συνδέοντας λειτουργικά μπλοκ με καλώδια σε ένα διάγραμμα. Αυτή η οπτική αναπαράσταση του κώδικα επιτρέπει στους χρήστες να συλλαμβάνουν και να σχεδιάζουν σύνθετα συστήματα πιο διαισθητικά

2. **Εικονικά Όργανα (Virtual Instruments - VIs)**:

Τα βασικά δομικά στοιχεία στο LabVIEW ονομάζονται Εικονικά Όργανα (VIs). Κάθε VI έχει μια μπροστινή επιφάνεια (front panel) και ένα διάγραμμα μπλοκ (block diagram). Η μπροστινή επιφάνεια χρησιμεύει ως διεπαφή χρήστη, εμφανίζοντας χειριστήρια και ενδείκτες, ενώ το διάγραμμα μπλοκ περιέχει τον γραφικό κώδικα. Αυτός ο διαχωρισμός της διεπαφής και της λογικής απλοποιεί τη διαδικασία ανάπτυξης και εντοπισμού σφαλμάτων.



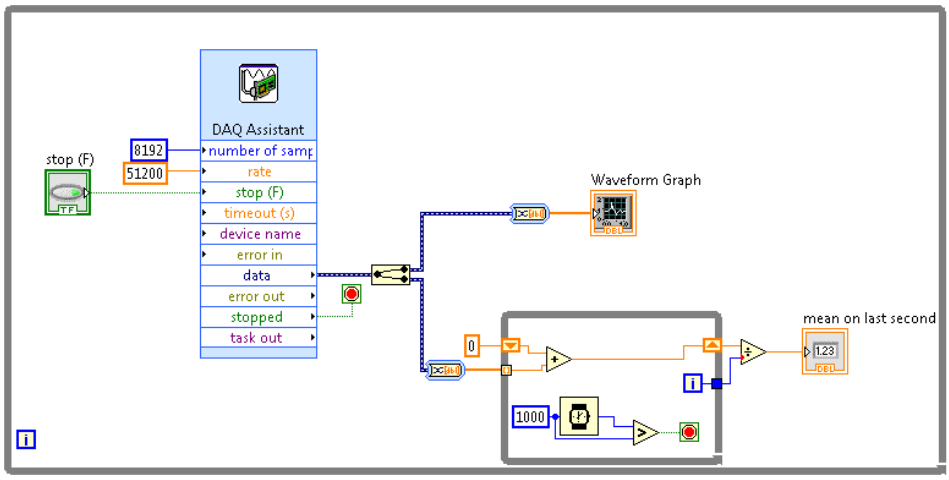
Εικόνα 39.Data chart στο front panel



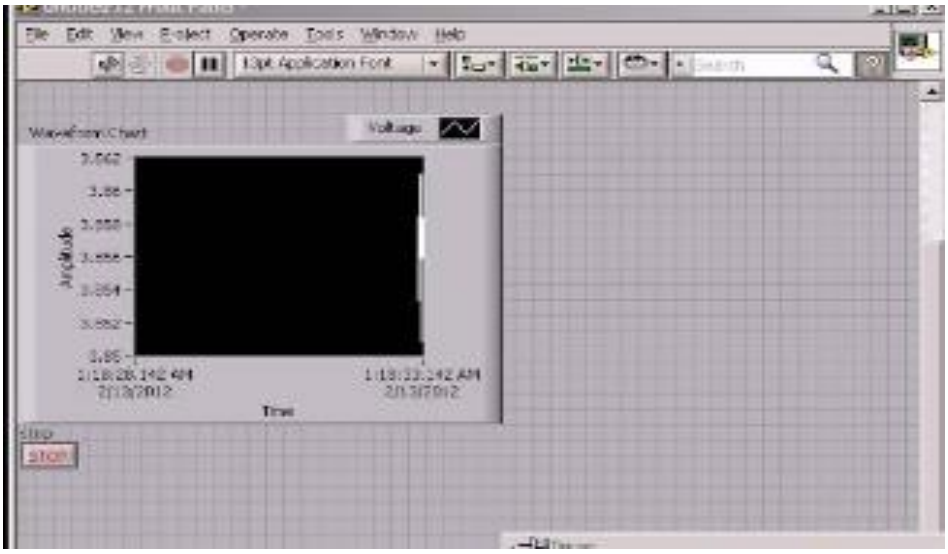
Εικόνα 40.Τυπική Μορφή Block Diagram

3. **Ενσωμάτωση με Υλικό**:

Το LabVIEW επιτρέπει την ενσωμάτωση με μια μεγάλη γκάμα συσκευών υλικού, όπως κάρτες Απόκτησης Δεδομένων (DAQ), αισθητήρες, επενεργητές και άλλα όργανα. Η National Instruments παρέχει εκτεταμένες βιβλιοθήκες και οδηγούς για απρόσκοπτη επικοινωνία με αυτές τις συσκευές, επιτρέποντας στους χρήστες να αναπτύσσουν ισχυρά συστήματα απόκτησης δεδομένων και ελέγχου.



Εικόνα 41.Block Diagram της βιβλιοθήκης DAQ



Εικόνα 42.Τυπική Μορφή Front Panel με χρήση DAQ

4. **Εκτεταμένες Βιβλιοθήκες και Εργαλεία**:

Το LabVIEW περιλαμβάνει ένα πλούσιο σύνολο βιβλιοθηκών για ανάλυση δεδομένων, επεξεργασία σήματος, έλεγχο και αυτοματισμό. Αυτές οι βιβλιοθήκες απλοποιούν την υλοποίηση σύνθετων αλγορίθμων και λειτουργιών, επιτρέποντας στους προγραμματιστές να επικεντρωθούν στη λογική της εφαρμογής και όχι στις λεπτομέρειες του προγραμματισμού χαμηλού επιπέδου.

4.2.2 Πλεονεκτήματα του LabVIEW

1. **Ευκολία Χρήσης**:

Η γραφική προσέγγιση στον προγραμματισμό καθιστά το LabVIEW προσβάσιμο σε χρήστες με περιορισμένη εμπειρία στον προγραμματισμό. Η διαισθητική διεπαφή και η λειτουργία drag-and-drop επιτρέπουν στους χρήστες να δημιουργούν προγράμματα γρήγορα και αποτελεσματικά.

2. **Ταχεία Ανάπτυξη**:

Οι εκτεταμένες βιβλιοθήκες και τα προεγκατεστημένα modules του LabVIEW επιταχύνουν τη διαδικασία ανάπτυξης. Οι χρήστες μπορούν να αξιοποιήσουν υπάρχουσες λειτουργίες και παραδείγματα για να δημιουργήσουν εφαρμογές, μειώνοντας τον χρόνο και την προσπάθεια που απαιτούνται για την ανάπτυξη από το μηδέν.

3. **Ευελιξία**:

Η δυνατότητα του LabVIEW να διασυνδέεται με μεγάλη ποικιλία υλικού λήψης δεδομένων αλλά και η δυνατότητα διασύνδεσής του με γλώσσες προγραμματισμού όπως η C και η python το καθιστούν κατάλληλο για ένα ευρύ φάσμα εφαρμογών. Από μικρά πειράματα στο εργαστήριο μέχρι μεγάλα βιομηχανικά συστήματα, το LabVIEW μπορεί να προσαρμοστεί σε διαφορετικές απαιτήσεις.

4.2.3 Προκλήσεις και Περιορισμοί

1. **Καμπύλη Εκμάθησης**:

Παρόλο που ο γραφικός προγραμματισμός είναι γενικά πιο προσιτός, η απόκτηση εξειδίκευσης στο LabVIEW για σύνθετες εφαρμογές μπορεί να απαιτήσει σημαντική προσπάθεια και χρόνο. Οι χρήστες πρέπει να επενδύσουν χρόνο για να κατανοήσουν τα εκτεταμένα χαρακτηριστικά και τις λειτουργίες του.

2. **Περιορισμένες Γραφικές Λειτουργίες**

Με την εισαγωγή νέων προγραμμάτων scada, με γρηγορότερα και όλο και πιο φιλικά προς το χρήστη γραφικά περιβάλλοντα καθιστούν το LabView κατώτερο λογισμικό. Η έλλειψη ποικιλίας εικόνων, πινάκων ή animations καθιστούν τη δημιουργία μίας οθόνης “HMI” αρκετά επίπονη και χρονοβόρα διαδικασία. Με την όλο και μεγαλύτερη μείωση του κόστους των προγραμμάτων scada μεγάλων εταιριών όπως η Siemens ή η Rockwell Automation το μη εκσυγχρονισμένο LabView θα αποτελεί αποκλειστικά μία προσωρινή «δωρεάν» λύση και δε θα προτιμάται στον τομέα της οπτικοποίησης λειτουργιών διάφορων συστημάτων.

5.ΔΗΜΙΟΥΡΓΙΑ SCADA

Στο συγκεκριμένο σημείο θα γίνει περιγραφή του λογισμικού λήψης δεδομένων καθώς και οπτικοποίησης της πειραματικής εγκατάστασης. Ακόμα θα γίνει περιγραφή κάποιων alarms που δημιουργήθηκαν με σκοπό την προστασία της εγκατάστασης. Όλα τα παραπάνω έγιναν με χρήση του λογισμικού LabView και της βιβλιοθήκης DSC η οποία χρησιμοποιήθηκε μόνο στη διαδικασία της οπτικοποίησης.

5.1 Υποπρόγραμμα Επικοινωνίας και συλλογής δεδομένων

Βήμα 1. Ρύθμιση σειριακής επικοινωνίας μεταξύ των modules και του labview

Στην εγκατάσταση υπάρχουν 4 modules , δύο I7018 και δύο I7019R. Το κάθε module έχει μία ξεχωριστή διεύθυνση αλλά όλα τα modules «στέλνουν» τα δεδομένα με τον ίδιο τρόπο. Επομένως θα πρέπει να γίνει κατάλληλη ρύθμιση της θύρας του υπολογιστή έτσι ώστε τα δεδομένα να μεταφέρονται δίχως προβλήματα.

Οι ρυθμίσεις φαίνονται στον παρακάτω πίνακα

COM PORT	9
BAUD RATE	9600 bps
DATA BITS	8
STOP BIT	1
PARITY BIT	NONE
TERMINATION CHARACTER	13
Timeout	10 sec

Πίνακας 11. Ρυθμίσεις Σειριακής Επικοινωνίας

COM PORT: Όνομα της θύρας του υπολογιστή. Ρυθμίζεται ανάλογα με το πού συνδέεται το usb

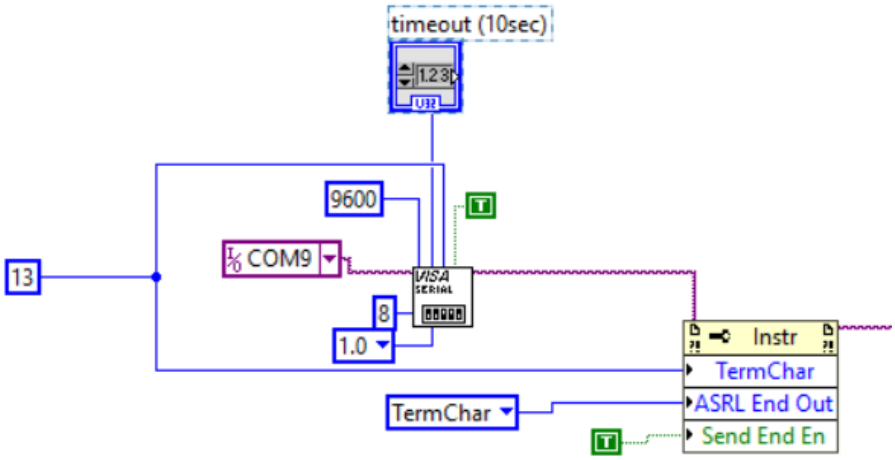
Data bits: Το κάθε module στέλνει τις μετρήσεις ως “data frames”. Το κάθε frame θα περιέχει 8 bits τα οποία αναπαριστούν την συγκεκριμένη μέτρηση

Stop bit: Το συγκεκριμένο bit χρησιμοποιείται για να υποδηλώσει το τέλος του κάθε πακέτου δεδομένων.

Parity bit: Αποτελεί bit ελέγχου σφάλματος αλλά στην προκειμένη περίπτωση δε θα χρησιμοποιηθεί.

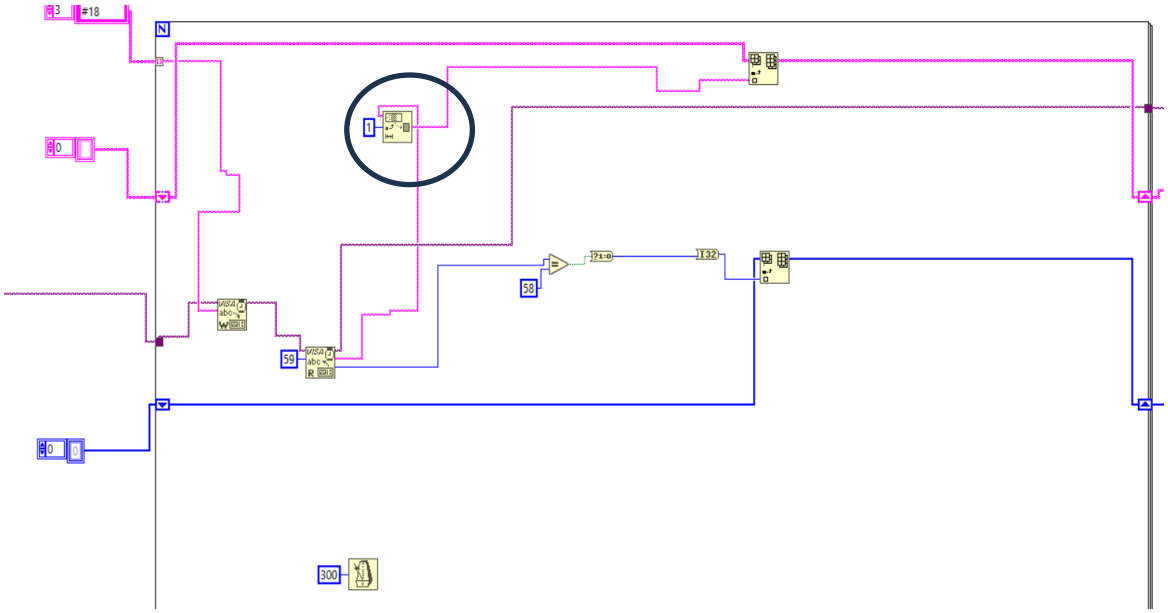
Termination Character: Όπως έχει προαναφερθεί, σύμφωνα με το πρωτόκολλο επικοινωνίας Dcon, πρέπει στο τέλος κάθε εντολής ή απάντησης να υπάρχει το σύμβολο CR (carriage return-επιστροφή κέρσορα στην αρχή της γραμμής). Το σύμβολο αυτό σε δεκαεξαδική μορφή συμβολίζεται με τον αριθμό 13.

Timeout: Συμβολίζει τον χρόνο αναμονής της θύρας προκειμένου να λάβει κάποιο πακέτο δεδομένων. Μπορεί να τροποποιηθεί ανάλογα με τις απαιτήσεις του χρήστη. Προτιμάται ο χρόνος να είναι 10 δευτερόλεπτα έτσι ώστε τα δεδομένα και από τα 4 modules να μεταδίδονται «έγκαιρα». Πρέπει όμως να τονιστεί πως λόγω κάποιο σφάλματος η συγκεκριμένη ρύθμιση δεν εγκαθίσταται και χρειάζεται εκ νέου ορισμός του χρόνου αναμονής. Παρόλαυτά απαιτείται να εισαχθεί μία ενδεικτική τιμή σε αυτό το στάδιο. Η μείωση του χρόνου αναμονής απαιτεί γρηγορότερα πιο σύγχρονα modules με μεγαλύτερη συχνότητα μετάδοσης δεδομένων.



Εικόνα 43.Port configuration

Βήμα 2. Εντολές Write, Read και αποθήκευση των δεδομένων



Εικόνα 44. Προγραμματισμός των εντολών Read,Write και αρχικός διαχωρισμός δεδομένων

Όπως έχει προαναφερθεί το κάθε module έχει τη δική του διεύθυνση και για αυτό θα πρέπει να τροποποιηθεί κατάλληλα η εντολή Read έτσι ώστε να ληφθεί απάντηση.

Σύμφωνα με το πρωτόκολλο Dcon η εντολή Read Analog Input All διαβάζει τις ενδείξεις και από τα 8 κανάλια του module και συμβολίζεται με τον παρακάτω τρόπο:

$$\#AA, \text{όπου } AA \text{ η διεύθυνση του module σε δεκαεξαδική μορφή}$$

Module	*Decimal Address*	*Hexadecimal Adress*
I7018	21	15
I7018	30	1E
I7019R	23	17
I7019R	24	18

Πίνακας 12. Διευθύνσεις των Modules

Αξίζει να αναφερθεί ότι αφού η σειριακή επικοινωνία έχει ρυθμιστεί δε χρειάζεται να δοθούν περισσότερες πληροφορίες στην παραπάνω εντολή.

Η απάντηση αποθηκεύεται σε μορφή string text και θα έχει μέγεθος 58 bytes. Επομένως απαιτείται ρύθμιση του εκτιμώμενου μεγέθους της απάντησης στην εντολή Read , το οποίο θα είναι 59 bytes.

Ύστερα από κάθε ανάγνωση η εντολή read θα πρέπει να επιστρέφει το μέγεθος της απάντησης το οποίο αναμένεται να είναι 58 bytes.Χρησιμοποιώντας το block "=" δημιουργείται ένα "status" beat το οποίο θα είναι 1 όταν το μέγεθος είναι 58 και 0 όταν αυτό έχει διαφορετική τιμή.

Για να επιτευχθεί λήψη δεδομένων και από τα 4 modules χρειάζεται να δημιουργηθεί ένα for loop το οποίο θα εκτελεί την παραπάνω διαδικασία και για τα 4 modules Τα status beat και οι απαντήσεις αποθηκεύονται σε μορφή «array». Απαιτείται χρόνος 300 ms πριν κάθε επανεκτέλεση για ομαλότερη λειτουργία.

Η μορφή των απαντήσεων και των status bits θαναι η παρακάτω:

Index	Απαντήσεις	Status beat
0	>AAA.AAAAAA.AAA….	1 (or 0)
1	>AAA.AAAAAA.AAA….	1 (or 0)
2	>AAA.AAAAAA.AAA….	1 (or 0)
3	>AAA.AAAAAA.AAA…	1 (or 0)

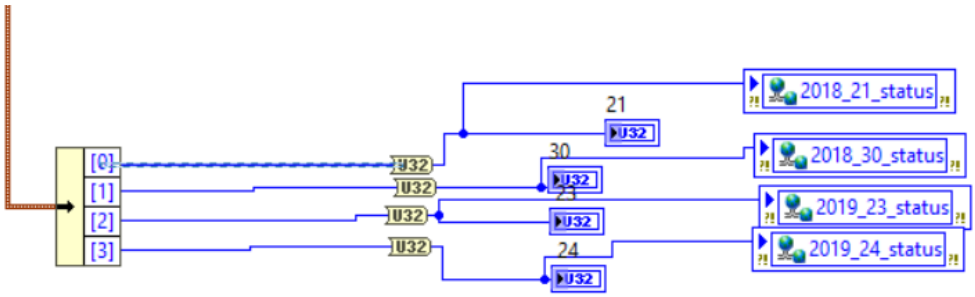
Πίνακας 13. Απαντήσεις σε "raw" μορφή

Πρέπει να σημειωθεί πως η εντολή read αγνοεί τον χαρακτήρα > μέσω ενός offset=1 που δίνεται κατά την ανάγνωση των δεδομένων. Επομένως η μορφή τους θα είναι η παρακάτω:

Index	Απαντήσεις	Status beat
0	AAA.AAAAAA.AAA….	1 (or 0)
1	AAA.AAAAAA.AAA….	1 (or 0)
2	AAA.AAAAAA.AAA….	1 (or 0)
3	AAA.AAAAAA.AAA…	1 (or 0)

Πίνακας 14. Απαντήσεις σε επεξεργασμένη μορφή

Τα status beat δεν χρειάζονται περεταίρω επεξεργασία και μπορούν να διαχωριστούν (array to cluster) και ύστερα να αποθηκευτούν ως μεταβλητές (double precision-U32).



Εικόνα 45. Status bits

Βήμα 3. Διαχωρισμός των μετρήσεων(Data parsing)

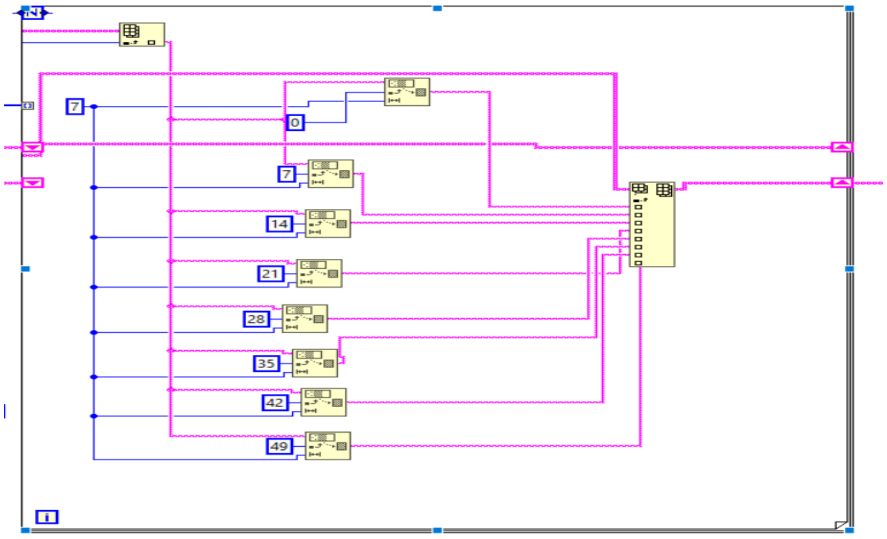
Η κάθε απάντηση περιλαμβάνει 8 μετρήσεις μήκους 7 ψηφίων. Επομένως μπορεί να γίνει ο διαχωρισμός της κάθε μέτρησης δίνοντας ένα offset καθώς και τον αριθμό των χαρακτήρων-ψηφίων. Ακολουθεί ένα παράδειγμα για να γίνει πιο κατανοητή η παραπάνω διαδικασία

020.320032.255 Για να γίνει ο διαχωρισμός θα πρέπει να δοθεί offset 0 και μήκος 7 και offset 7 και μήκος 7. Το αποτέλεσμα θα είναι το παρακάτω

020.320

032.255

Επομένως γίνεται αντιληπτό πως το offset θα είναι 0,7,14,21,28,35,42,49 και το μήκος σταθερό στα 7 ψηφία για μία απάντηση μήκους 58 χαρακτήρων. Η κάθε μέτρηση αποθηκεύεται σε ένα array ξεχωριστά από τις υπόλοιπες.



Εικόνα 46. Προγραμματισμός για Διαχωρισμό των μετρήσεων

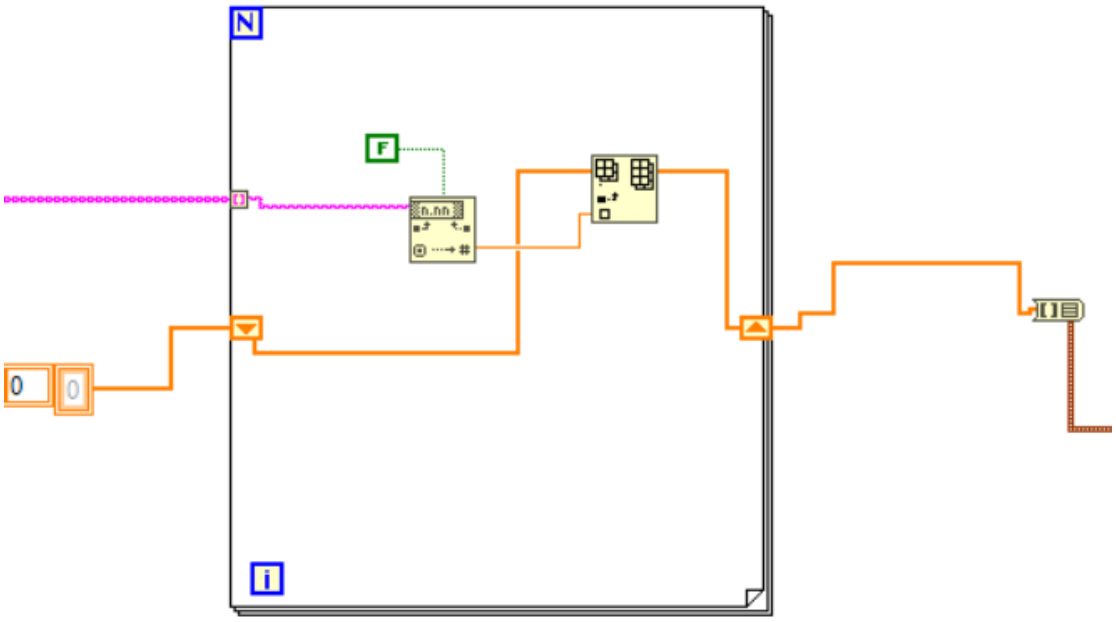
Μέσω ενός for loop το οποίο θα επαναλαμβάνεται για κάθε μία από τις 4 απαντήσεις(ανάλογα με το index) μπορεί να γίνει ο διαχωρισμός και η αποθήκευση των μετρήσεων σε μορφή string text σε ένα νέο array. Η μορφή τους ύστερα από την παραπάνω διαδικασία θα είναι ως εξής.

0	AAA.AAA
1	AAA.AAA
2	AAA.AAA
3	AAA.AAA
…….	…..
31	AAA.AAA

Πίνακας 15. Απαντήσεις σε “text” μορφή

Βήμα 4. Μετατροπή σε αριθμό

Το κάθε στοιχείο του παραπάνω πίνακα μετατρέπεται σε αριθμό και αποθηκεύεται εκ νέου. Ύστερα ακολουθεί το clustering με το οποίο κάθε μέτρηση μπορεί να φανεί στο front panel ξεχωριστά



Εικόνα 47. Πρόγραμμα για μετατροπή text σε αριθμό

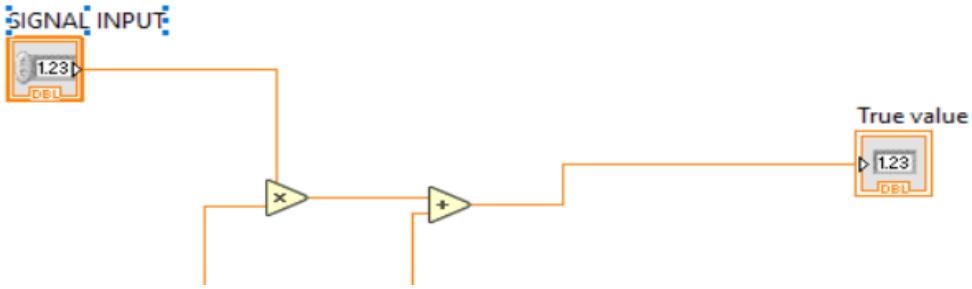
5.2. Βαθμονόμηση μετρητικών οργάνων

Τα μετρητικά όργανα της εγκατάστασης μεταβάλλουν την τιμή του ηλεκτρικού σήματος (είτε σε mA είτε σε V) γραμμικά σε σχέση με το μετρούμενο μέγεθος ξεκινώντας από μία αρχική τιμή.

Επομένως η συνάρτηση «κέρδους»(Gain) και «αρχικής απόκλισης»(offset) είναι η εξής

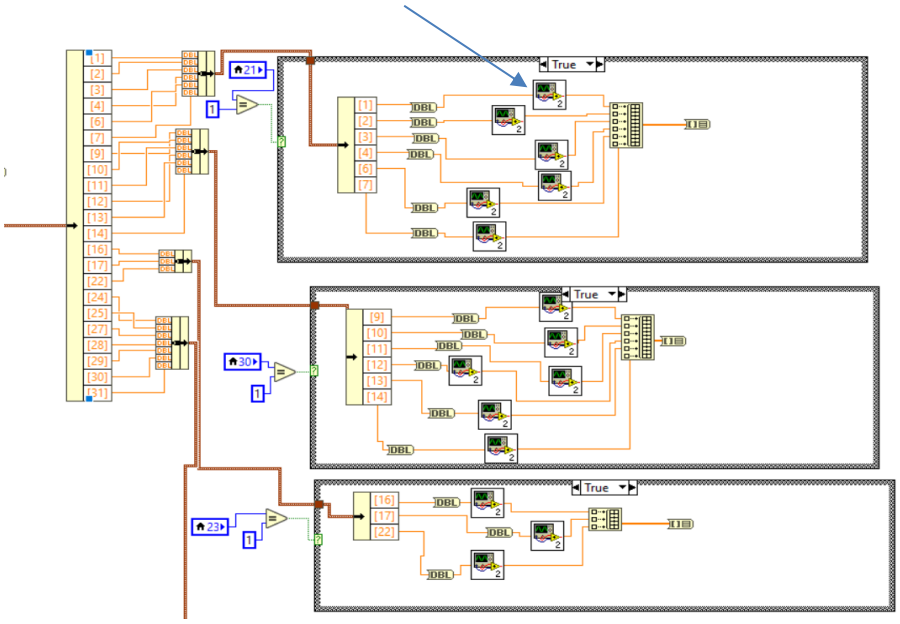
$$y = Gain * x + Offset$$

Στο περιβάλλον Labview η συγκεκριμένη διαδικασία πραγματοποιείται όπως φαίνεται στην παρακάτω φωτογραφία και είναι ίδια για κάθε μέτρηση.



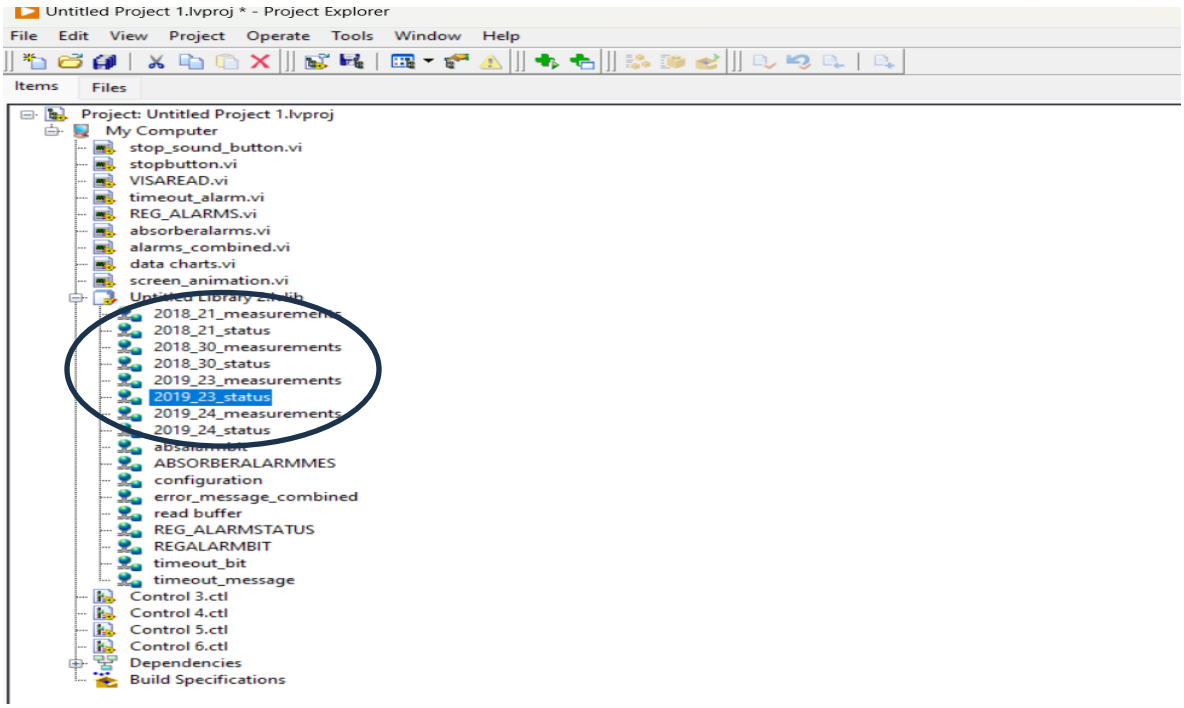
Εικόνα 48.Πρόγραμμα για ρύθμιση Gain Offset

Το παραπάνω χρησιμοποιείται ως υποπρόγραμμα(sub-vi) ενσωματωμένο στο βασικό Vi .



Εικόνα 49.Τελικός διαχωρισμός μετρήσεων και αποθήκευση μετρήσεων

Αξίζει να σημειωθεί πως οι μετρήσεις θα ανανεώνονται και θα αποθηκεύονται μόνο εάν τα status bits θα είναι ίσα με 1.Όλες οι μετρήσεις θα πρέπει να αποθηκεύονται ως “shared” μεταβλητές στον φάκελο που δημιουργείται το λογισμικό λήψης δεδομένων. (με σκοπό να μπορούν να χρησιμοποιηθούν σε όλα τα υποπρογράμματα)



Εικόνα 50. Μεταβλητές μετρήσεων και status bits

Ο παρακάτω πίνακας συγκεντρώνει τα μετρητικά όργανα και τις μετρήσεις καθώς και τα αντίστοιχα gain,offset.

Unit	Sensor Type	Name	Module	Channel	Operating Voltage	Output Signal	Gain	Offset
Absorber	KIMO PT100	T_ABS_OUT_WATER	I7018 (30)	1	-	4-20mA	553.191	-263.021
Absorber	KIMO PT100	T_ABS_IN_AIR	I7018 (30)	2	-	4-20mA	553.191	-263.021
Absorber	KIMO PT100	T_ABS_OUT_AIR	I7018 (30)	3	-	4-20mA	553.191	-263.021
Absorber	KIMO PT100	T_ABS_IN_CONC	I7018 (30)	4	-	4-20mA	553.191	-263.021
Absorber	KIMO PT100	T_ABS_IN_DIL	I7018 (30)	5	-	4-20mA	553.191	-263.021
Absorber	KIMO PT100	T_ABS_IN_WATER	I7018 (30)	6	-	4-20mA	553.191	-263.021
Absorber	KIMO TH100	RH_ABS_OUT	I7019R (24)	0	24Vdc	4-20mA	18.7640	19.7835

Absorber	KIMO TH100	RH_ABS_IN	I7019R (24)	1	24Vdc	4-20mA	18.7640	19.7835
Absorber	SCX15AN	P_AMBIENT	I7019R (24)	3	12Vdc	4-20mA	-44.333	0.202
Absorber	DSG2000	ΔP_ABS_AIR(overall)	I7019R (24)	4	24Vdc	0-10 V	0.0116	-0.0011
Absorber	DSG500	ΔP_ABS_PACK	I7019R (24)	5	24Vdc	0-10 V	0.0121	-0.0018
Absorber	CERABAR TPMC131	P_ABS_WATER_BOT	I7019R (24)	6	24Vdc	4-20mA	0.1116	-0.6343
Absorber	CERABAR TPMC131	P_ABS_WATER_TOP	I7019R (24)	7	24Vdc	4-20mA	0.0991	-0.4443
Regenerator	KIMO PT100	T_REG_OUT_WATER	I7018 (21)	1	-	4-20mA	-553.191	-263.021
Regenerator	KIMO PT100	T_REG_OUT_AIR_1	I7018 (21)	2	-	4-20mA	-553.191	-263.021
Regenerator	KIMO PT100	T_REG_IN_AIR	I7018 (21)	3	-	4-20mA	-553.191	-263.021
Regenerator	KIMO PT100	T_REG_OUT_AIR_2	I7018 (21)	4	-	4-20mA	-553.191	-263.021
Regenerator	KIMO PT100	T_REG_IN_DIL	I7018 (21)	6	-	4-20mA	553.191	-263.021
Regenerator	KIMO PT100	T_REG_OUT_CONC	I7018 (21)	7	-	4-20mA	553.191	-263.021
Regenerator	KIMO TH100	RH_REG_OUT	I7019R (23)	0	24Vdc	4-20mA	18.7640	19.7835
Regenerator	DCXL 10DS	ΔP_REG_AIR	I7019R (23)	1	12Vdc	4-20mA	-15.294	208.87
Regenerator	KIMO TH100	RH_REG_IN	I7019R (23)	6	24Vdc	4-20mA	18.7640	19.7835

Πίνακας 16. Βαθμονόμηση Μετρητικών Οργάνων και βασικές πληροφορίες λειτουργίας αυτών

5.3. Υποπρόγραμμα Alarms

Η διαδικασία δημιουργίας των alarms έχει ως εξής.

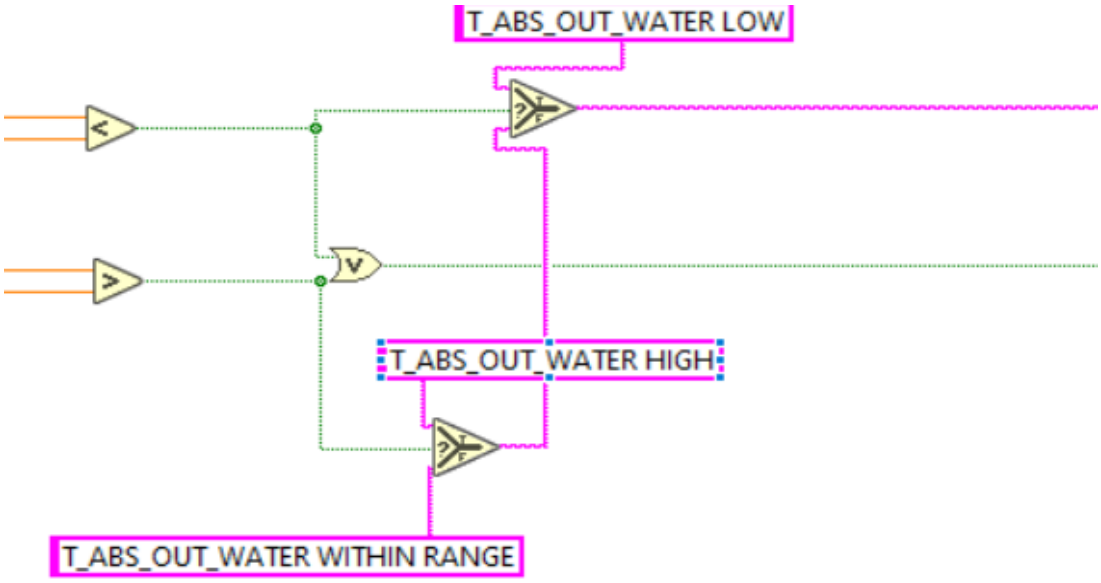
Στην αρχή χρησιμοποιούμε τις μετρήσεις (shared variables) για τον absorber και τις ονομάζουμε σύμφωνα με τον παραπάνω πίνακα. Ύστερα θέτουμε ένα άνω και κάτω όριο των τιμών της κάθε μέτρησης.

Αρχικά συγκρίνεται η τιμή με το άνω όριο και ανάλογα δίνεται ένα output bit. Εάν αυτό είναι 1 τότε εμφανίζεται το μήνυμα “…..HIGH” ενώ αν αυτό είναι 0 τότε εμφανίζεται “WITHIN RANGE”.

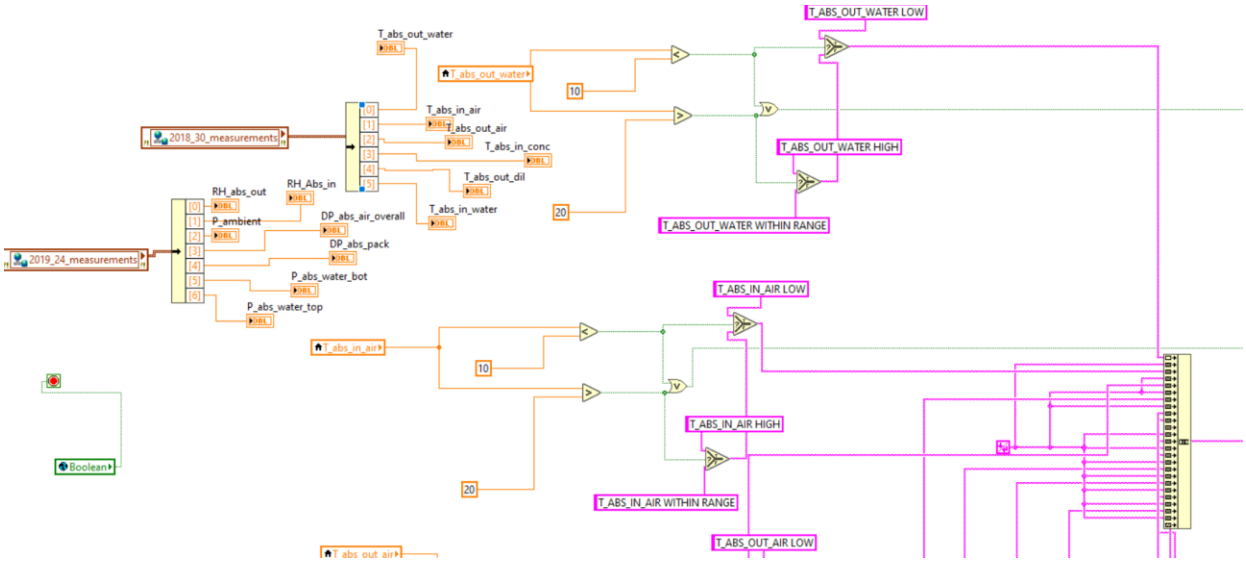
Ύστερα εξετάζεται το κάτω όριο της τιμής και δίνει ένα output bit. Εάν αυτό είναι 1 τότε εμφανίζεται το μήνυμα “LOW” ενώ αν αυτό είναι 0 τότε εμφανίζεται το μήνυμα που προέκυψε από το bit του άνω ορίου. Με αυτό τον αποφεύγεται η διπλή ένδειξη “within

range” που θα προέκυπτε εάν δεν δίναμε σαν input το μήνυμα του άνω ορίου στο κάτω όριο.

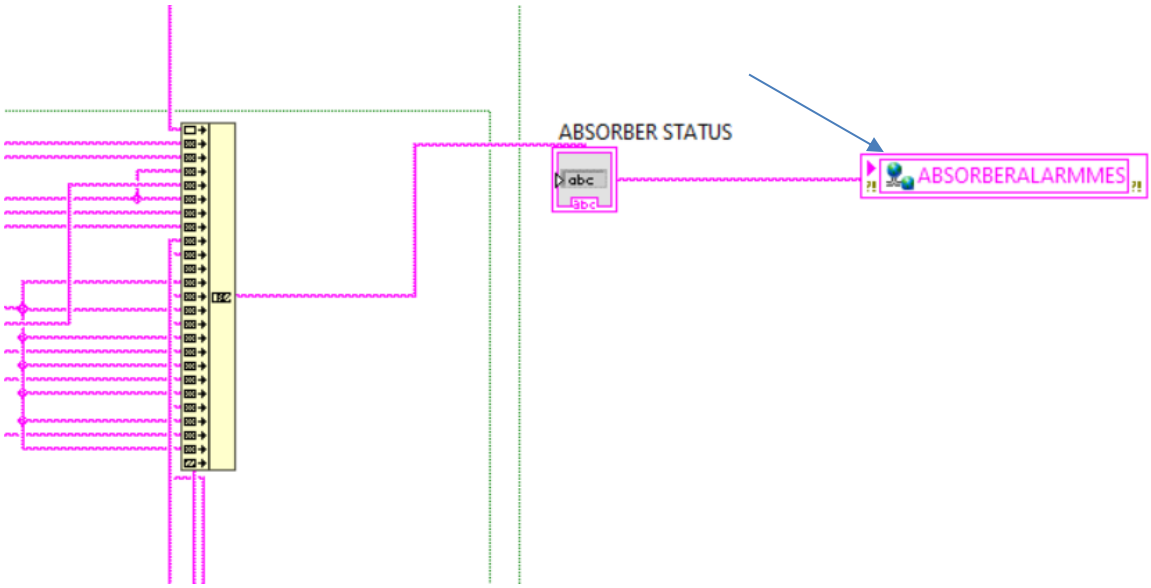
Η ίδια διαδικασία ακολουθείται για όλες τις μετρήσεις και τα μηνύματα αποθηκεύονται σε ένα “array of string text” αλλά σε ξεχωριστές γραμμές. Το συγκεκριμένο array έχει τη μορφή “Shared Variable” με όνομα “….Alarms” ώστε να μπορεί να χρησιμοποιηθεί (να φαίνεται) στα υπόλοιπα υποπρογράμματα.



Εικόνα 51.Προγραμματισμός ένδειξης alarm

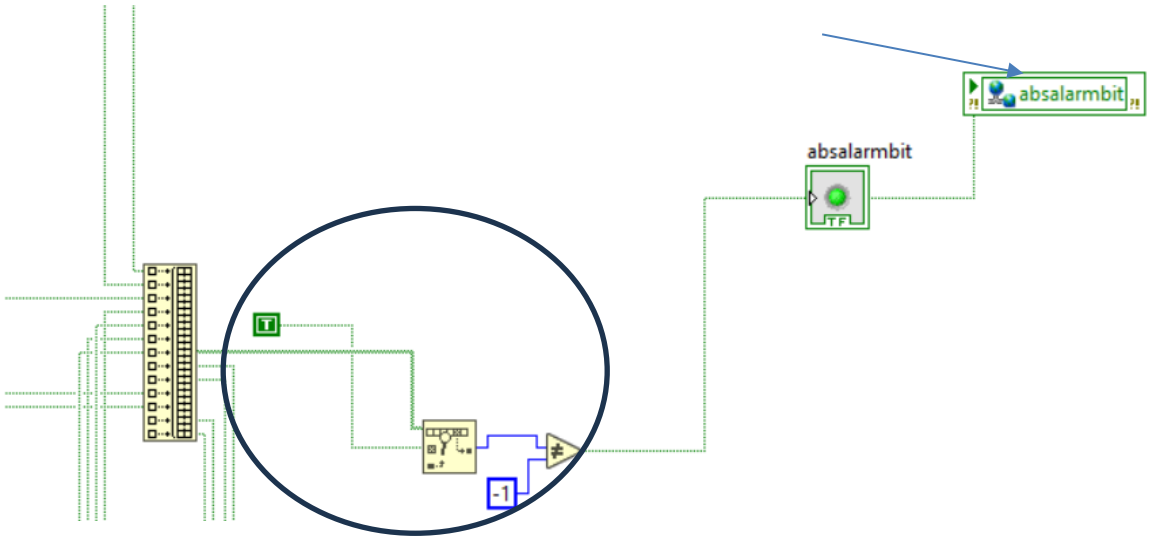


Εικόνα 52. Συνολική εικόνα προγράμματος των alarms



Εικόνα 53. Δημιουργία Πίνακα alarms και αποθήκευση

Επιπλέον όπως φαίνεται στις παραπάνω φωτογραφίες σε κάθε μέτρηση χρησιμοποιείται μία πύλη OR η οποία εξετάζει εάν κάποιο από τα 2 bits είναι 1.Το output της πύλης θα είναι ένα bit 0 ή 1 ανάλογα με τα inputs. Τα output των πυλών OR αποθηκεύονται σε ένα "Boolean array". Ύστερα εξετάζεται εάν σε αυτό το array υπάρχει κάποιο bit με τιμή 1(True). Ως output ορίζεται πλέον 1 bit το οποίο θα είναι 1 εάν στο array υπάρχει έστω και 1 True και 0 εάν όλες οι τιμές του array είναι false. Το bit αυτό αποθηκεύεται ως «shared variable» με όνομα "…alarmbit"



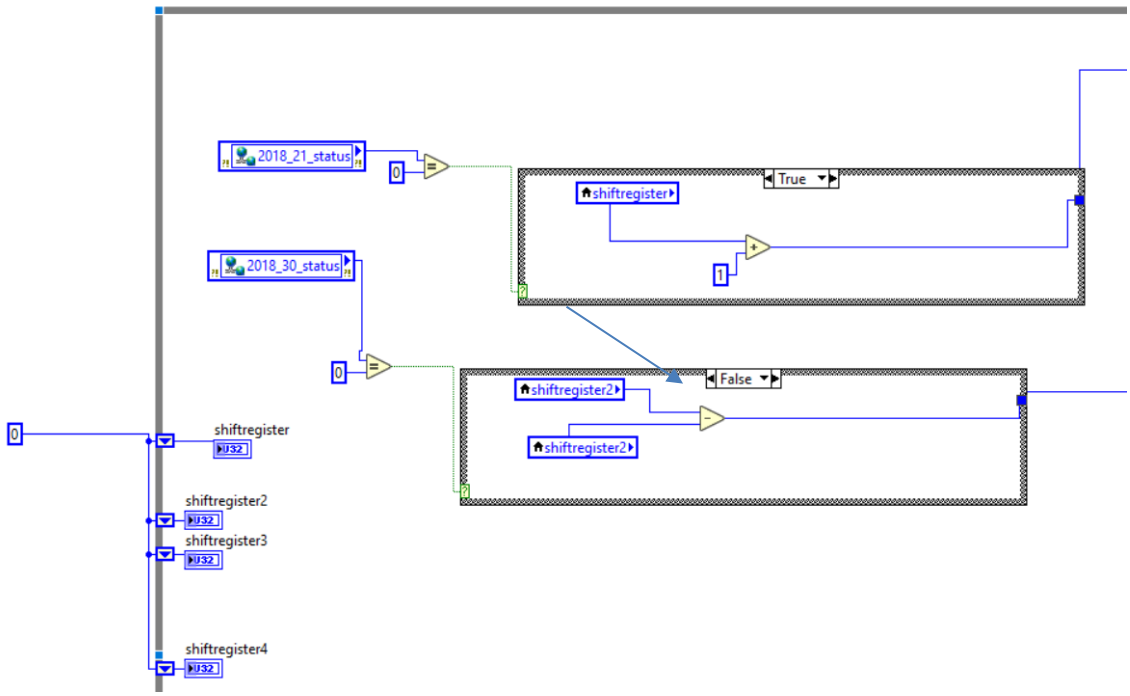
Εικόνα 54. Προγραμματισμός alarmbit

Για τον αναγεννητή ακολουθούνται ακριβώς τα ίδια βήματα για αυτό δεν θα ακολουθήσει περεταίρω ανάλυση.

5.4 Υποπρόγραμμα «Timeout»

Το συγκεκριμένο πρόγραμμα δημιουργήθηκε με σκοπό να γίνει "bypass" το σφάλμα λήξης χρόνου αναμονής το οποίο έχουν τα modules από τον κατασκευαστή. Πιθανότατα λόγω χαμηλής συχνότητας δειγματοληψίας αλλά και ταχύτητα συλλογής δεδομένων από το Module I7520 χρειάζεται περισσότερος χρόνος για να συλλεχθούν επιτυχώς όλα τα δεδομένα. Αυτό υποδεικνύει την εκ νέου δημιουργία ενός timeout alarm το οποίο θα αφήνει μεγαλύτερο χρονικό περιθώριο για τη συλλογή των δεδομένων.

Ο χρόνος που έχει ορισθεί για την συλλογή δεδομένων για το κάθε module είναι 300 ms . Χρησιμοποιώντας τα status bits των modules μπορεί να οριστεί ένα loop το οποίο θα αυξάνει μία μεταβλητή κατά 1-ανά επανάληψη όσο τα status bits είναι 0 . Εάν αυτά αλλάξουν και λάβουν την τιμή 1 τότε θα πρέπει να γίνει "reset" της μεταβλητής. Το reset μπορεί να γίνει αφαιρώντας την τιμή της μεταβλητής της προηγούμενης επανάληψης από τη νέα.

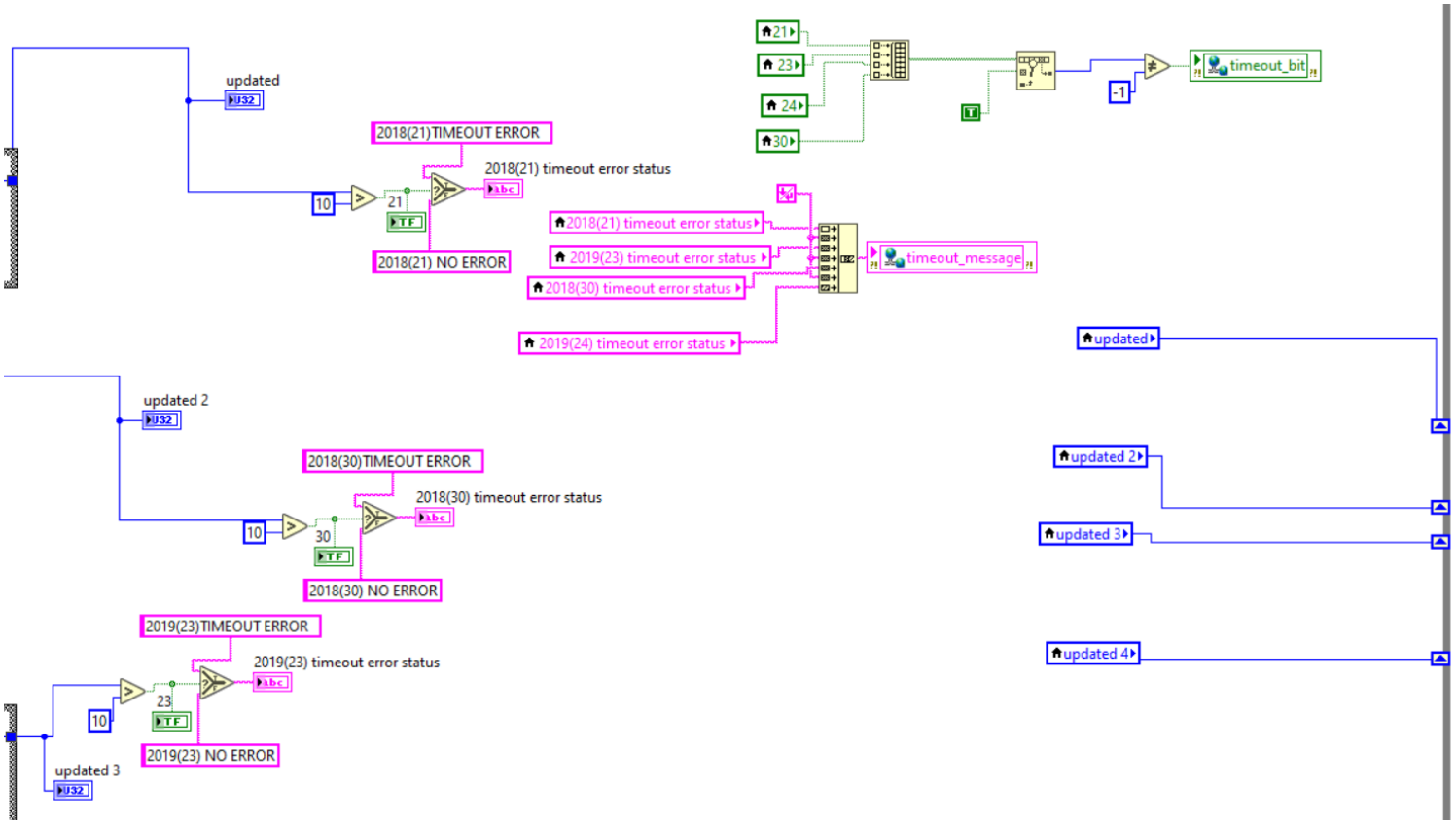


Εικόνα 55. Προγραμματισμός timeout alarm

Οι μεταβλητές shiftregister ξεκινάνε από την τιμή 0 και αυξάνουν κατά 1 όσο τα status bits είναι 0 ενώ πραγματοποιείται reset σε περίπτωση που γίνουν 1.

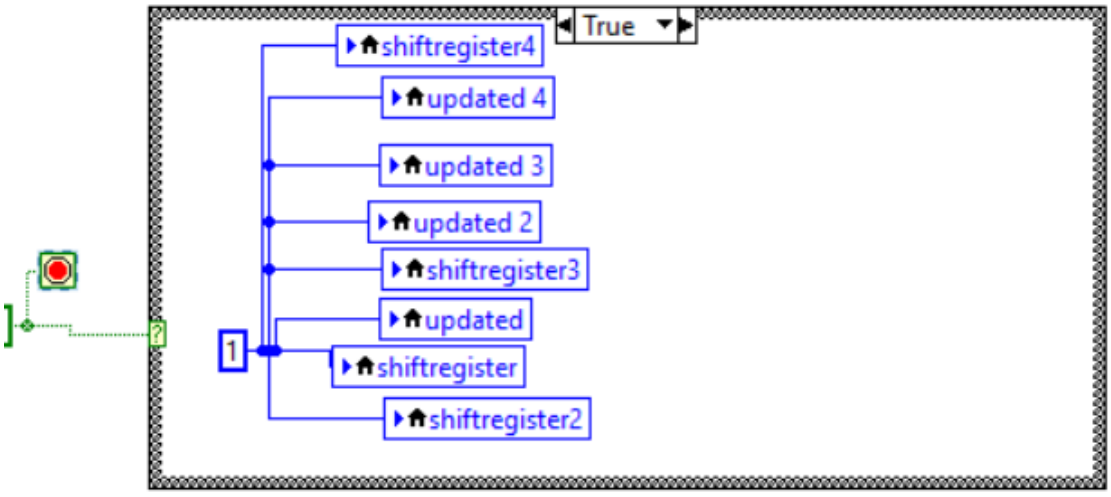
Ύστερα πρέπει να οριστεί ένα όριο στις μεταβλητές το οποίο έχει επιλεχθεί να είναι 10. Επομένως σε κάθε επανάληψη γίνεται σύγκριση των μεταβλητών με την τιμή 10. Το αποτέλεσμα της σύγκρισης είναι ένα bit (0 ή 1 False ή True αντίστοιχα). Σε περίπτωση που το bit είναι 1 τότε στην οθόνη θα εμφανιστεί σφάλμα “timeout error”.

Πρέπει να τονιστεί πως τα παραπάνω πραγματοποιούνται ξεχωριστά για κάθε module και τα μηνύματα των σφαλμάτων καθώς και τα bits συλλέγονται σε arrays και αποθηκεύονται ως «κοινές μεταβλητές» προκειμένου να φαίνονται σε όλα τα υποπρογράμματα καθώς και στην κεντρική οθόνη.



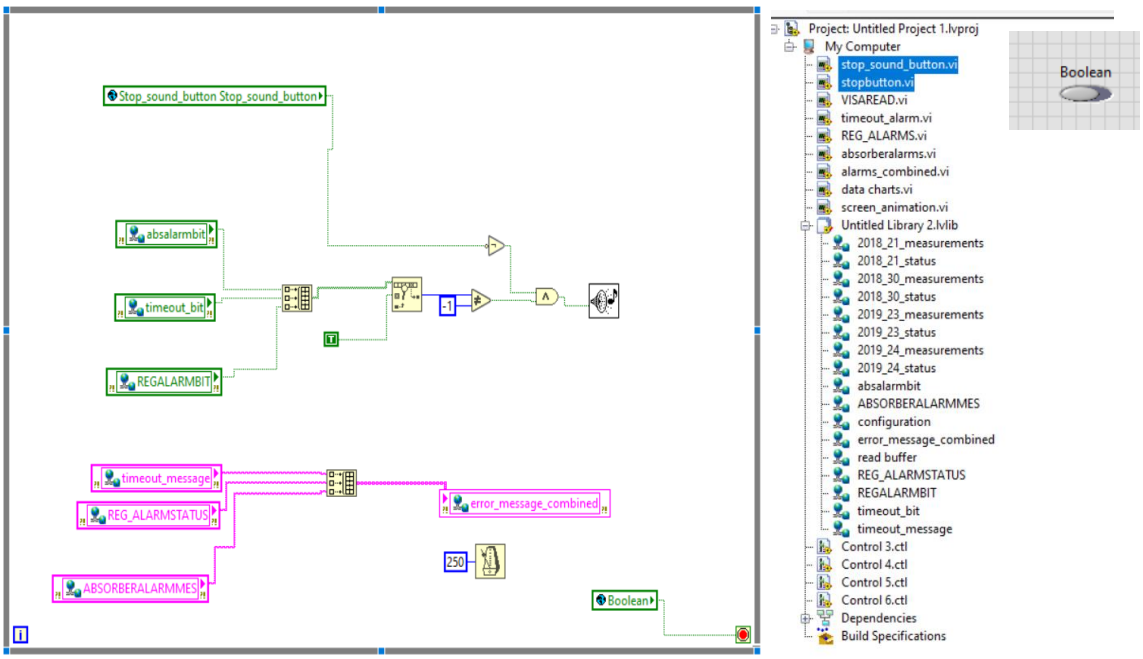
Εικόνα 56. Προγραμματισμός εμφάνισης μηνύματος timeout,alarmbit και αποθήκευση

Σε περίπτωση που ο χρήστης σταματήσει (force stop) το πρόγραμμα θα πρέπει να γίνει reset των μεταβλητών στην τιμή 1.



Εικόνα 57. Reset στην αρχική τιμή σε περίπτωση force stop

5.5. Υποπρόγραμμα «Συναγερμού»



Εικόνα 58. Συνολική εικόνα προγράμματος alarms και ήχου(αριστερά). Global μεταβλητή stop sound button

Στο συγκεκριμένο υποπρόγραμμα χρησιμοποιούνται οι «κοινές μεταβλητές» των μηνυμάτων που προκύπτουν από τα alarms του απορροφητή ,του αναγεννητή ,από το χρόνο αναμονής καθώς και τα αντίστοιχα «alarm bits».

Τα «μηνύματα συγκεντρώνονται» σε μορφή “string text array” σε μία νέα “shared variable” με όνομα “errormessagecombined”.

Τα alarm bits συγκεντρώνονται σε ένα νέο array από το οποίο θα εξεταστεί εάν υπάρχει κάποια τιμή «True» . Σαν output θα οριστεί ένα bit το οποίο θα έχει την αντίστοιχη τιμή. (1 εάν υπάρχει True και 0 εάν όλα είναι False).

Ταυτόχρονα δημιουργείται ένας διακόπτης σε μορφή “Global Variable” με όνομα “Stop sound button”, καθώς μόνο με αυτό τον τρόπο μπορεί να χρησιμοποιηθεί σε άλλα υποπρογράμματα. Όταν η διακόπτης είναι “On” τότε η μεταβλητή θα έχει τιμή 1.

Η μεταβλητή αυτή θα περάσει από μία πύλη NOT ώστε να αναστραφεί η τιμή της και ύστερα θα συνδυαστεί με το Output bit μέσω μίας πύλης AND. Το output της πύλης AND θα ανοίξει ή θα κλείσει τον ήχο του συναγερμού.

Για να γίνει πιο κατανοητή η παραπάνω διαδικασία ακολουθούν μερικοί πίνακες «αληθείας» .

Absorber bit	Reg bit	timeoutbit		Output bit
0	0	0	Check if true	0
0	0	1		1
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Πίνακας 17. Πίνακας Αληθείας

Διακόπτης	Not	Διακόπτης_Not
1		0
0		1

Πίνακας 18. Πίνακας Αληθείας Πύλης Not

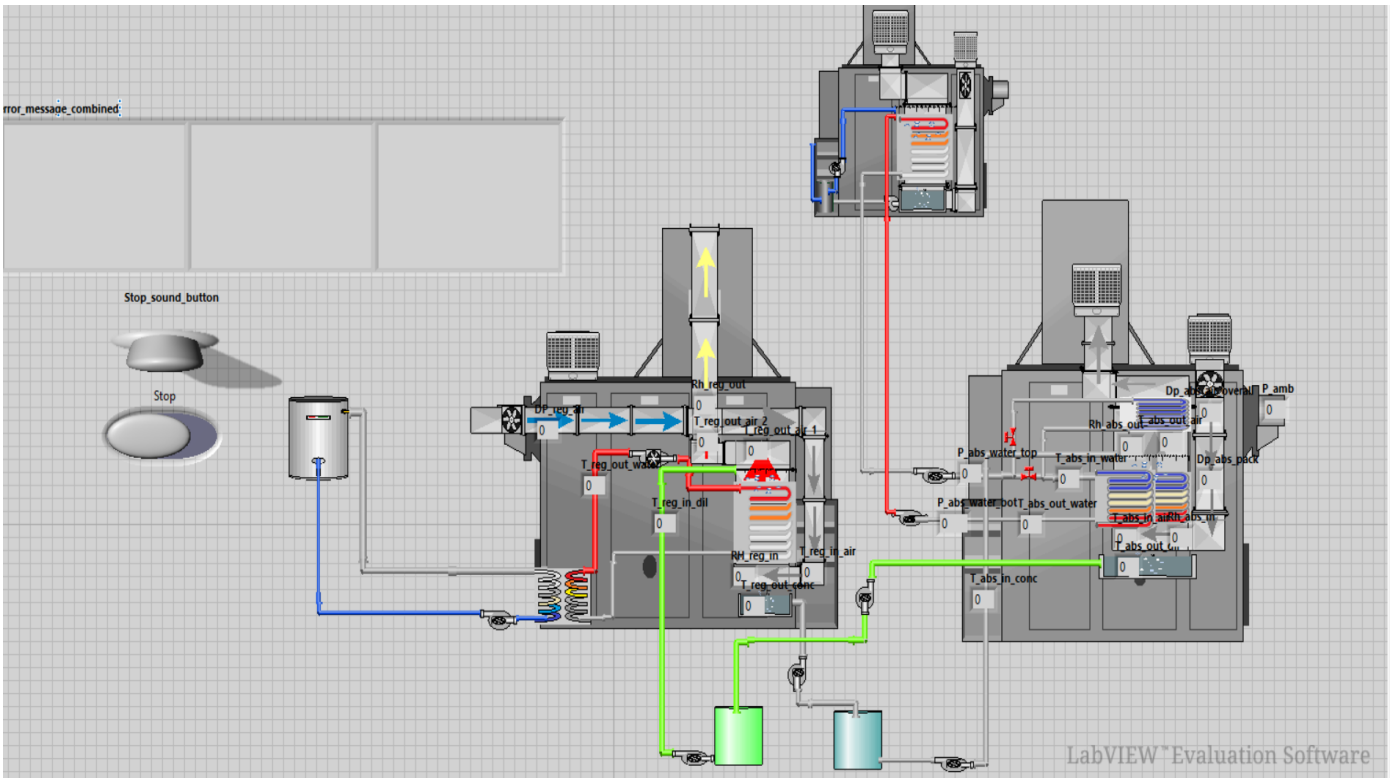
Διακόπτης	Διακόπτης_Not	Output bit		Beep_sound
0	1	0	AND	0
1	0	1		0
0	1	1		1
1	0	0		0

Πίνακας 19. Γενικευμένος Πίνακας Αληθείας

Από τον τελευταίο πίνακα φαίνεται πως ο ήχος θα ακουστεί μόνο όταν ο διακόπτης είναι “Off” και έχει εντοπιστεί κάποια βλάβη.

Η παραπάνω «λογική» είναι αρκετά χρήσιμη καθώς δίνει την δυνατότητα στο χρήστη να κλείσει τον ήχο του συναγερμού ενώ ακόμα θα μπορεί να γνωρίζει πως υπάρχει βλάβη μέσω του μηνύματος της μεταβλητής “errormessagecombined”.

5.6. Οπτικοποίηση-Δημιουργία Οθόνης



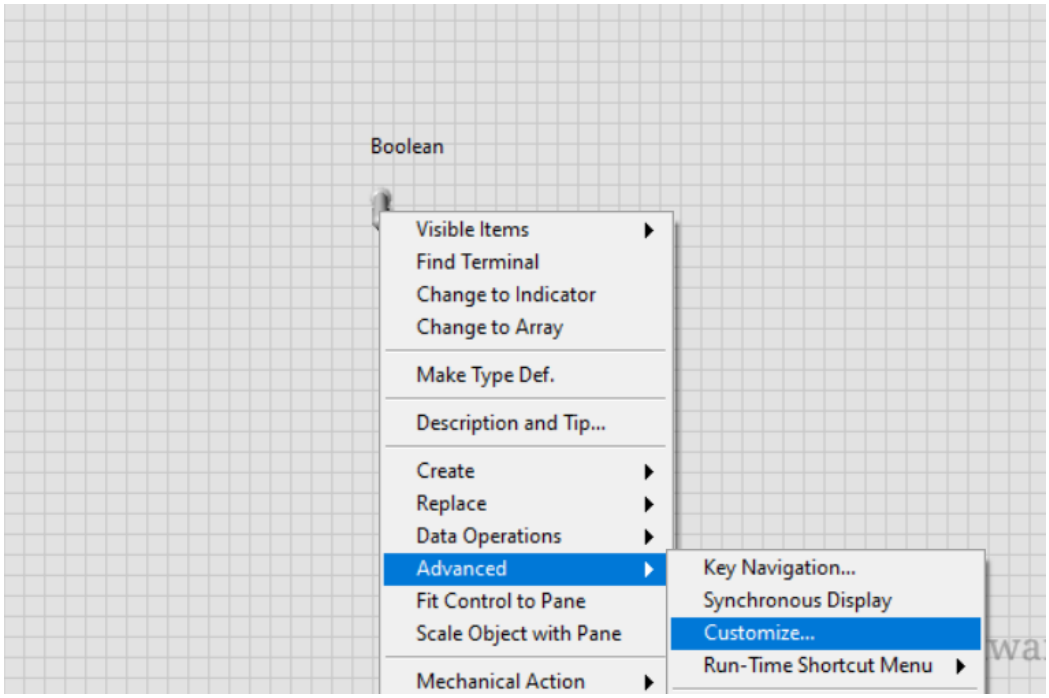
Εικόνα 59. Οθόνη Ελέγχου

Έναν πολύ βασικό περιορισμό του LabView αποτελεί η έλλειψη κάποιου “animation” προκειμένου να φαίνεται η λειτουργία του συστήματος σε πραγματικό χρόνο.

Παρόλαυτά δίνεται η δυνατότητα δημιουργίας της εναλλαγής δύο εικόνων με την αλλαγή της τιμής μιας Boolean μεταβλητής.

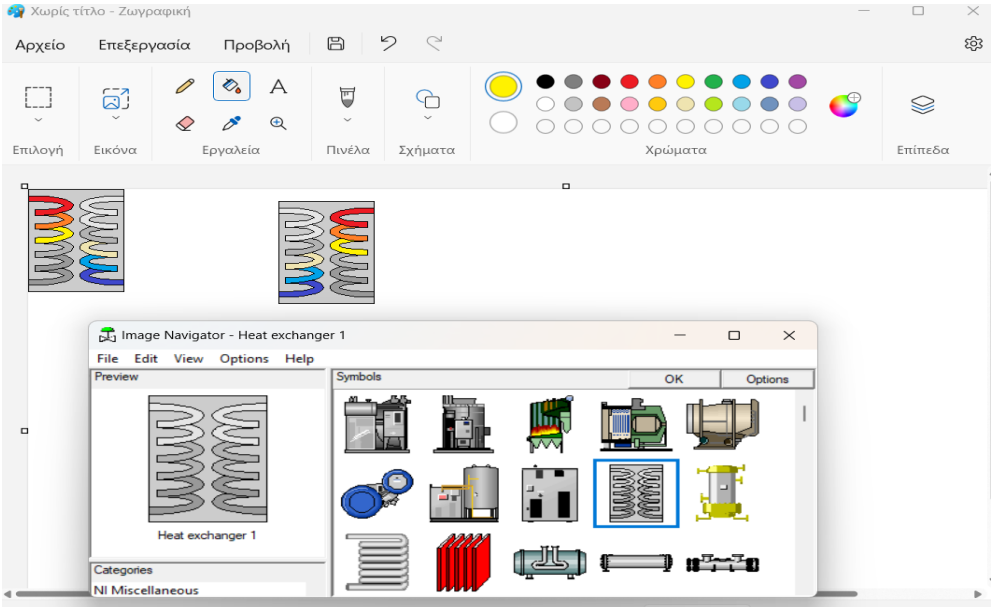
Παρακάτω θα περιγραφεί η διαδικασία δημιουργίας animation χρησιμοποιώντας ως παράδειγμα τον εναλλάκτη νερού-νερού του αναγεννητή.

Βήμα 1. Στο front panel εισάγεται ένας «Boolean Διακόπτης» . Με δεξί κλικ μπορεί να γίνει επεξεργασία αυτού του διακόπτη



Εικόνα 60. Επεξεργασία του boolean διακόπτη

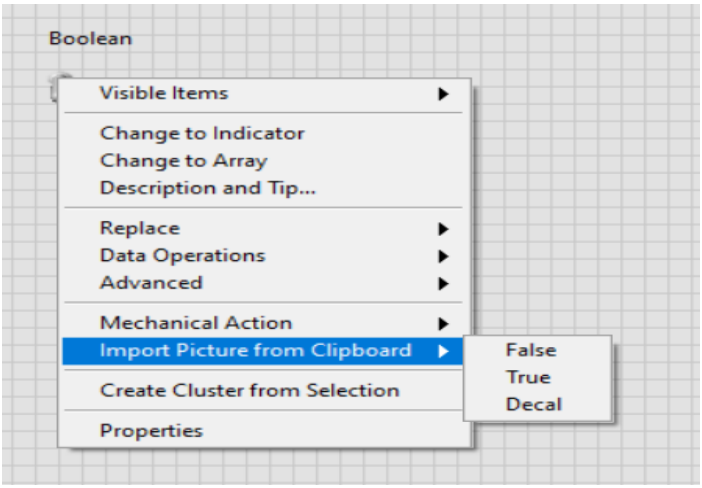
Βήμα 2. Μέσω κατάλληλης επιλογής εικόνας από τη βιβλιοθήκη DSC module και με χρήση του Paint των windows δημιουργούνται οι δύο εικόνες



Εικόνα 61.Δημιουργία εικόνων

Βήμα 3. Επικόλληση των εικόνων αντίστοιχα για τιμές true(1) και false(0).

Μετά την επικόλληση πρέπει να γίνει save ως “Control” στον φάκελο του project.



Εικόνα 62.Επικόλληση εικόνων

Με βάση την παραπάνω διαδικασία δημιουργείται το “animation” των παρακάτω

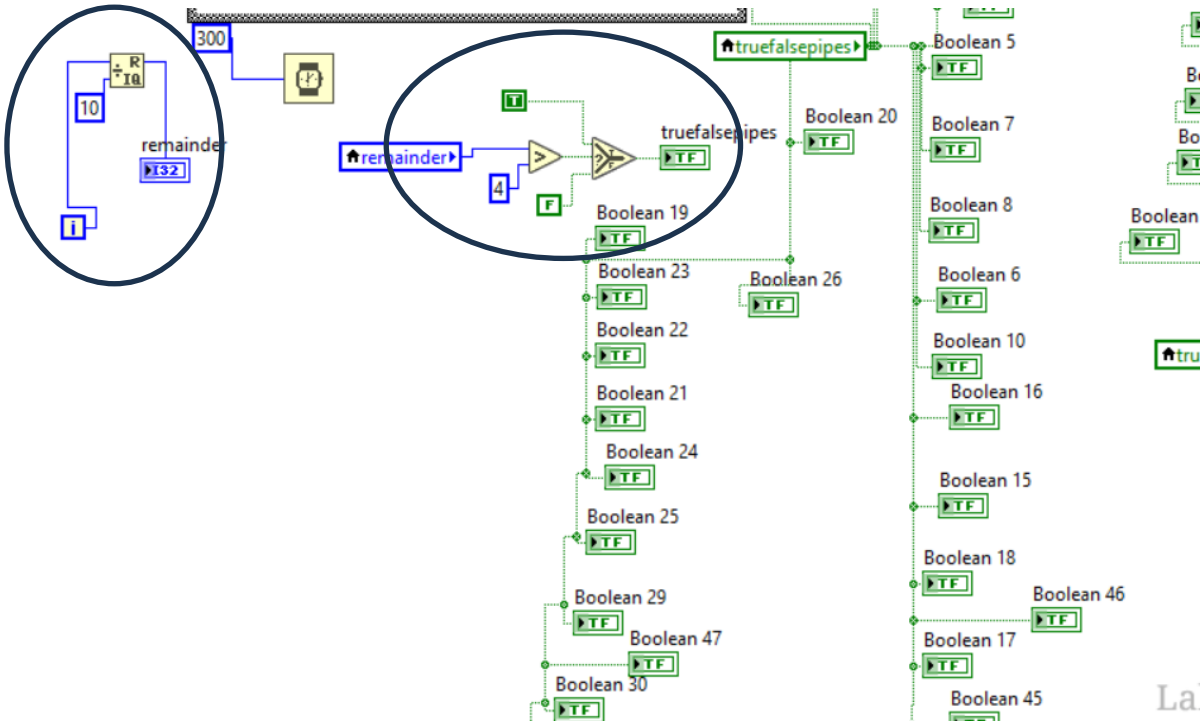
- Εναλλάκτες: Μεταβολή της θερμοκρασίας του νερού.
- Αντλίες-Ανεμιστήρες : Περιστροφή της πτερωτής

- Αγωγοί : Αλλαγή Χρώματος ανάλογα με την πυκνότητα του διαλύματος ή τη θερμοκρασία του νερού που κυκλοφορεί σε αυτούς
- Αέρας: Αλλαγή κατεύθυνσης του αέρα με βέλη , καθώς και αλλαγή χρώματος των τελευταίων ανάλογα με τη θερμοκρασία του αέρα σε κάθε σημείο
- Διάλυμα: Εκκίνηση ή παύση ψεκασμού διαλύματος.

Πρέπει να σημειωθεί ότι σε αυτό το στάδιο δεν γίνεται κάποια αλλαγή των εικόνων καθώς τα αποθηκευμένα control χρειάζονται μία Boolean μεταβλητή ως input προκειμένου να λειτουργήσουν.

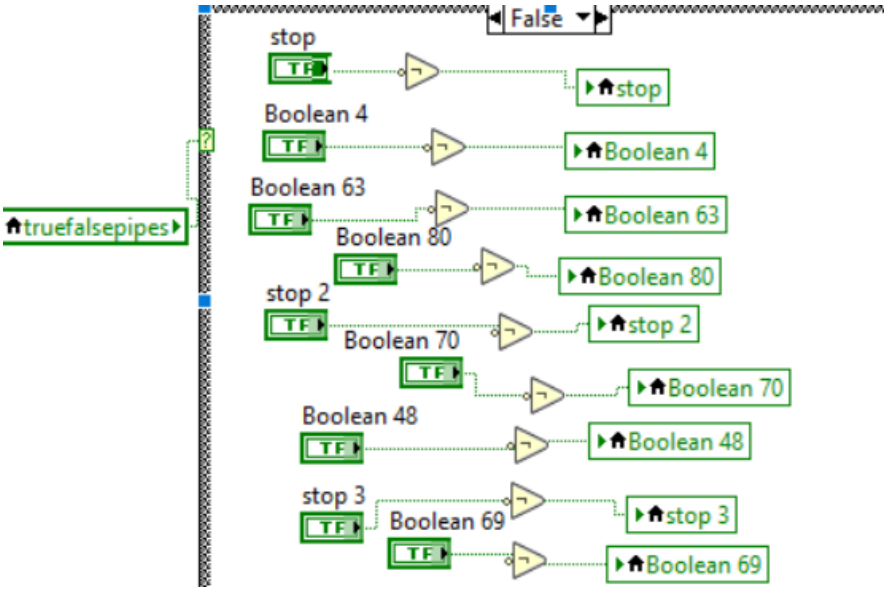
Βήμα 4. Δημιουργία Boolean Μεταβλητής

Δημιουργούμε ένα while loop το οποίο επαναλαμβάνεται κάθε 300 ms. Προσμετράμε τον αριθμό των επαναλήψεων και τον διαιρούμε με το 10 σε κάθε επανάληψη. Ύστερα το υπόλοιπο της διαίρεσης συγκρίνεται με τον αριθμό 4.Το αποτέλεσμα της σύγκρισης θα είναι ένα output bit με τιμές true ή false οι οποίες θα εξαρτώνται από το αν το υπόλοιπο είναι μεγαλύτερο ή μικρότερο του 4. Έτσι τροφοδοτώντας το bit στα controls ολοκληρώνεται η διαδικασία του animation.



Εικόνα 63. Προγραμμα για δημιουργία animation

Πρέπει να σημειωθεί πως στην οθόνη ο απορροφητής και ο αναγεννητής θα λειτουργούν «αντίστροφα». Αυτό στην πραγματικότητα μπορεί να μην συμβαίνει καθώς η λειτουργία του ενός δεν εξαρτάται από την λειτουργία του άλλου , λόγω της ύπαρξης των δύο ξεχωριστών δεξαμενών διαλύματος. Στην οθόνη προτιμήθηκε όμως η αντίστροφη λειτουργία προκειμένου το κύκλωμα να γίνεται πιο εύκολα κατανοητό από τον χρήστη. Η αντίστροφη λειτουργία επιτυγχάνεται με χρήση μίας πύλης NOT στα animations του απορροφητή προκειμένου να λειτουργούν αντίθετα από τον αναγεννητή.

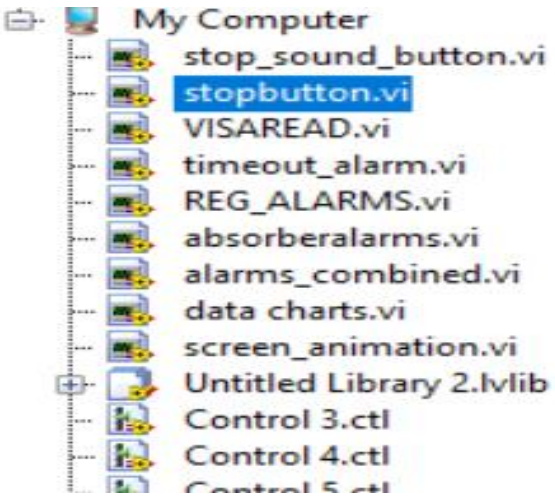


Εικόνα 64. Χρήση πύλης not για αντίστροφη λειτουργία

Επομένως με όλα τα παραπάνω, ο χρήστης βλέποντας την οθόνη είναι σε θέση να αντιληφθεί σε πολύ μεγάλο μέρος τη λειτουργία της εγκατάστασης, η οποία έχει περιγραφεί σε προηγούμενο κεφάλαιο, καθώς και να έχει εικόνα για την εγκυρότητα των μετρήσεων που λαμβάνει.

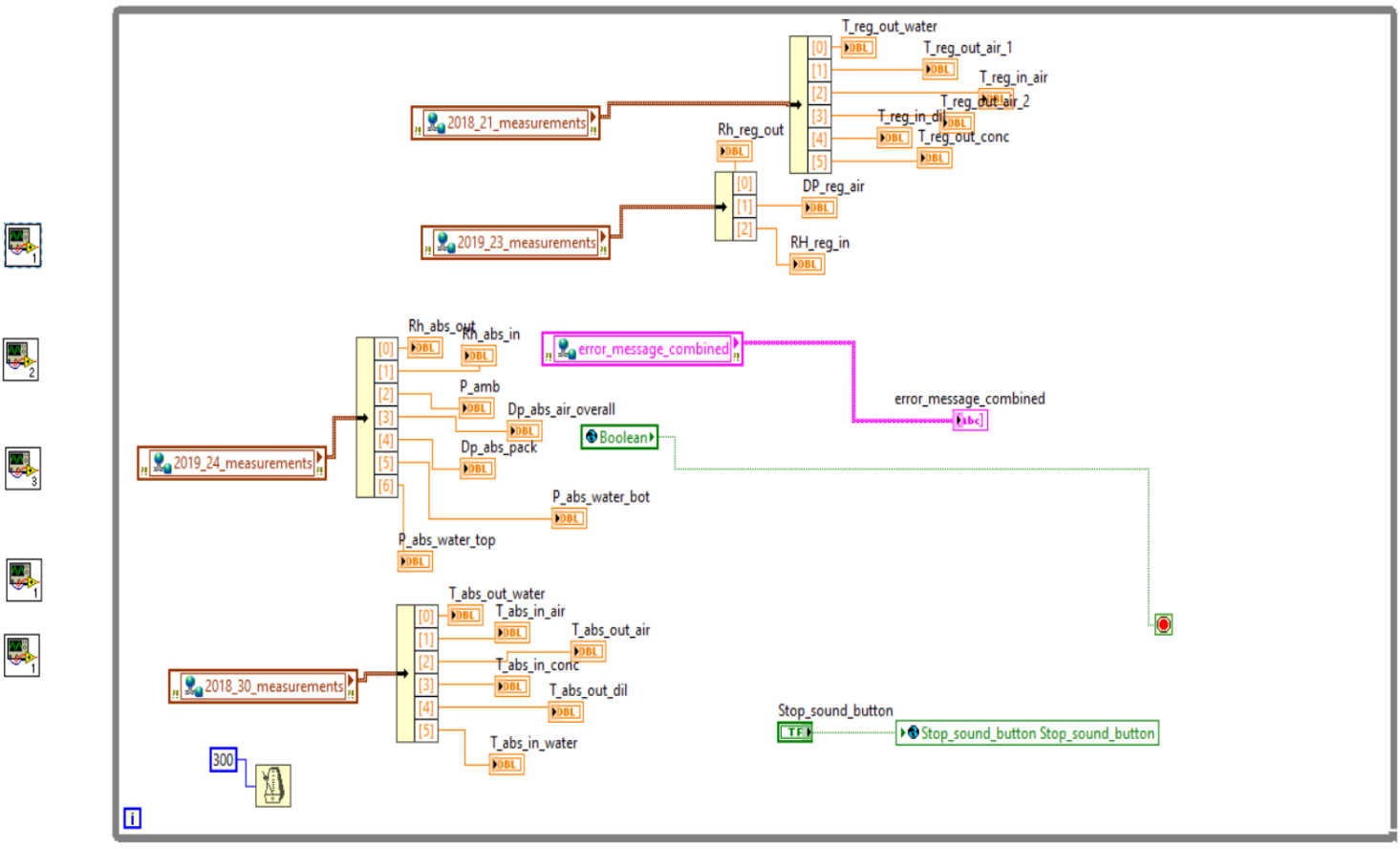
Τελευταίο βήμα για την ολοκλήρωση της οθόνης είναι η εισαγωγή όλων των υποπρογραμμάτων καθώς και των «κοινών μεταβλητών» ενώ είναι απαραίτητη και η δημιουργία ενός διακόπτη ο οποίος θα λειτουργεί ως “force stop”.

Ο διακόπτης δημιουργείται ως «Global Variable» και θα σταματάει την οθόνη καθώς και όλα τα υποπρογράμματα.



Εικόνα 65. Global variable force stop

Οι μετρήσεις και τα υποπρογράμματα εισάγονται όπως φαίνεται παρακάτω :

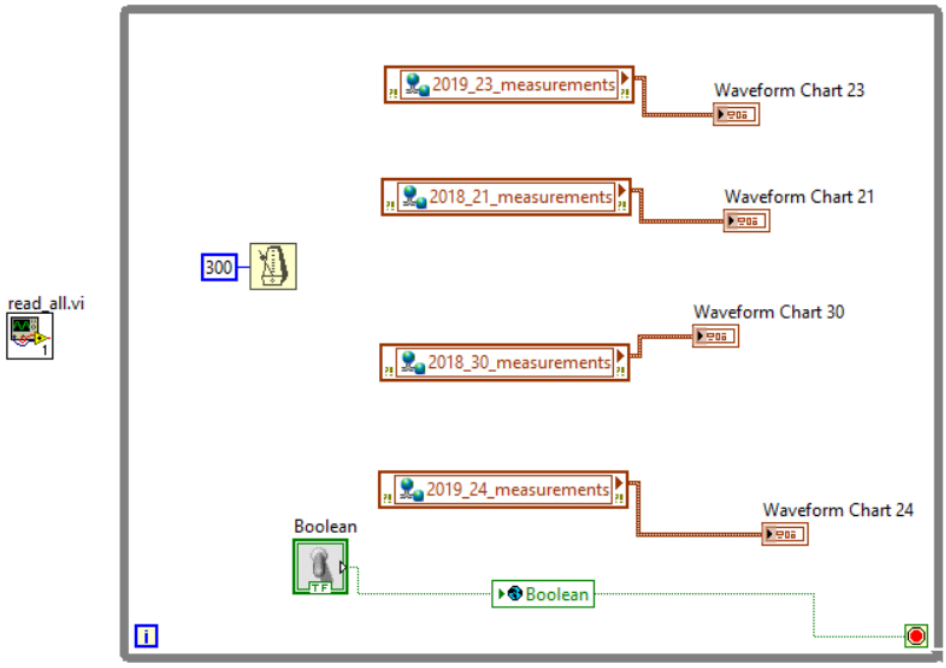


Εικόνα 66.Block diagram οθόνης για εμφάνιση των μετρήσεων

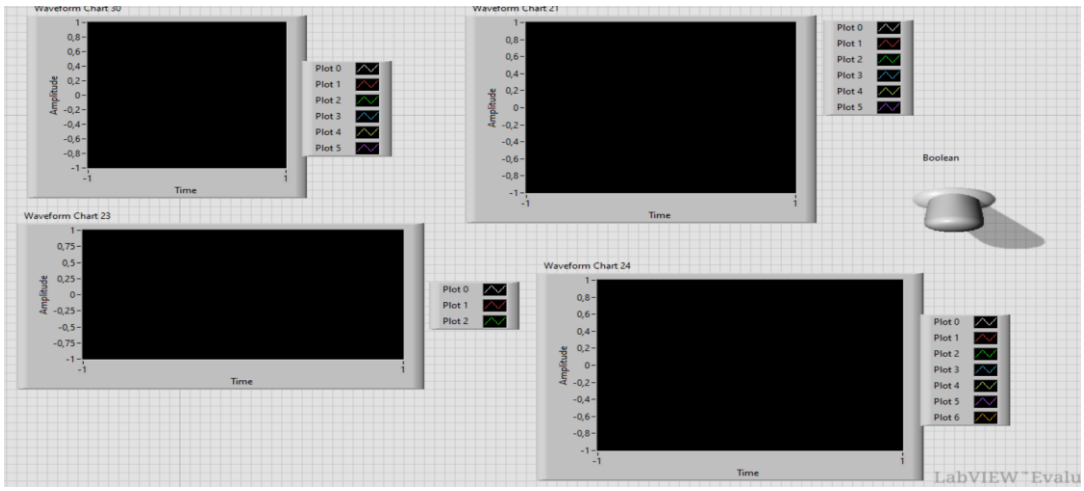
5.7. Γραφική απεικόνιση δεδομένων και δυνατότητα αποθήκευσης

Δημιουργείται εκ νέου ένα πρόγραμμα το οποίο θα χρησιμοποιεί το υποπρόγραμμα «Read» για τη συλλογή δεδομένων και την αποθήκευσή τους.

Χρησιμοποιούμε τις «κοινές» μεταβλητές που περιέχουν τις μετρήσεις του κάθε module και τις προβάλλουμε σε waveform charts τα οποία είναι διαθέσιμα από το LabVIew. Η απεικόνιση σταματάει μέσω της χρήσης του διακόπτη “force stop” που έχει δημιουργηθεί προηγουμένως.

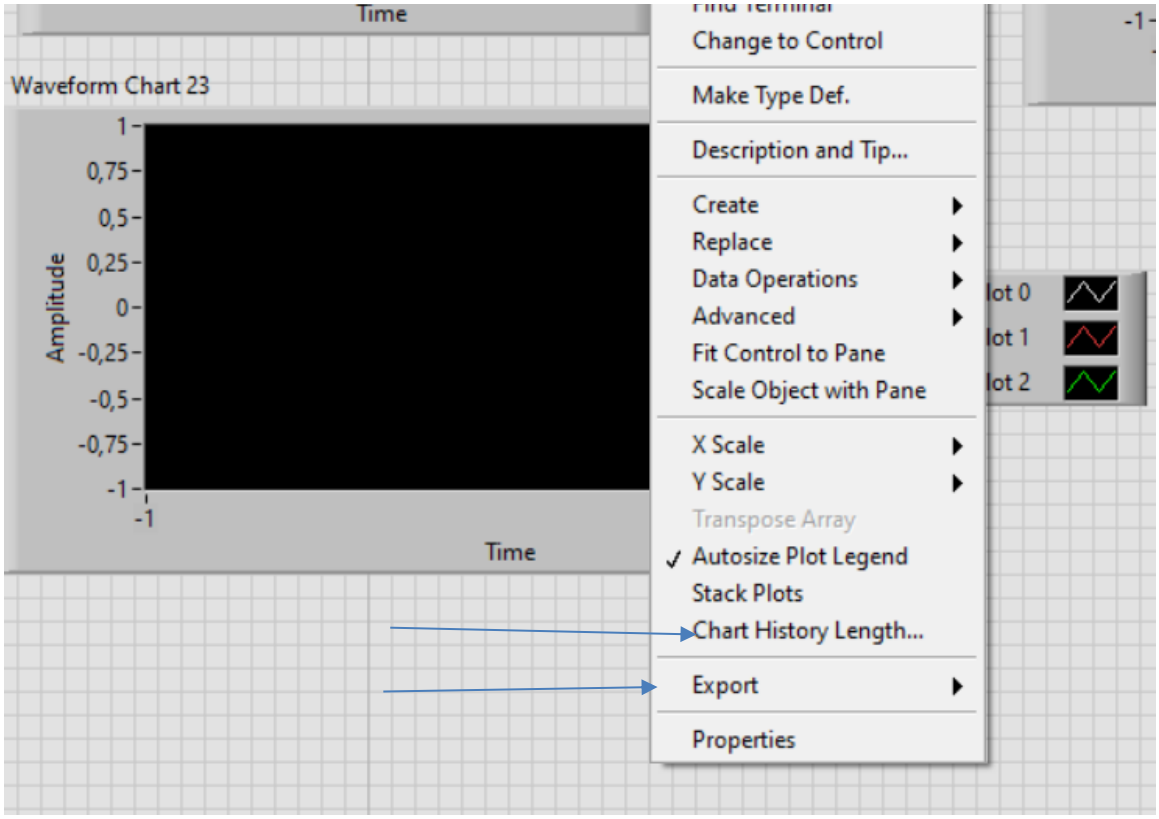


Εικόνα 67. Block diagram των data charts



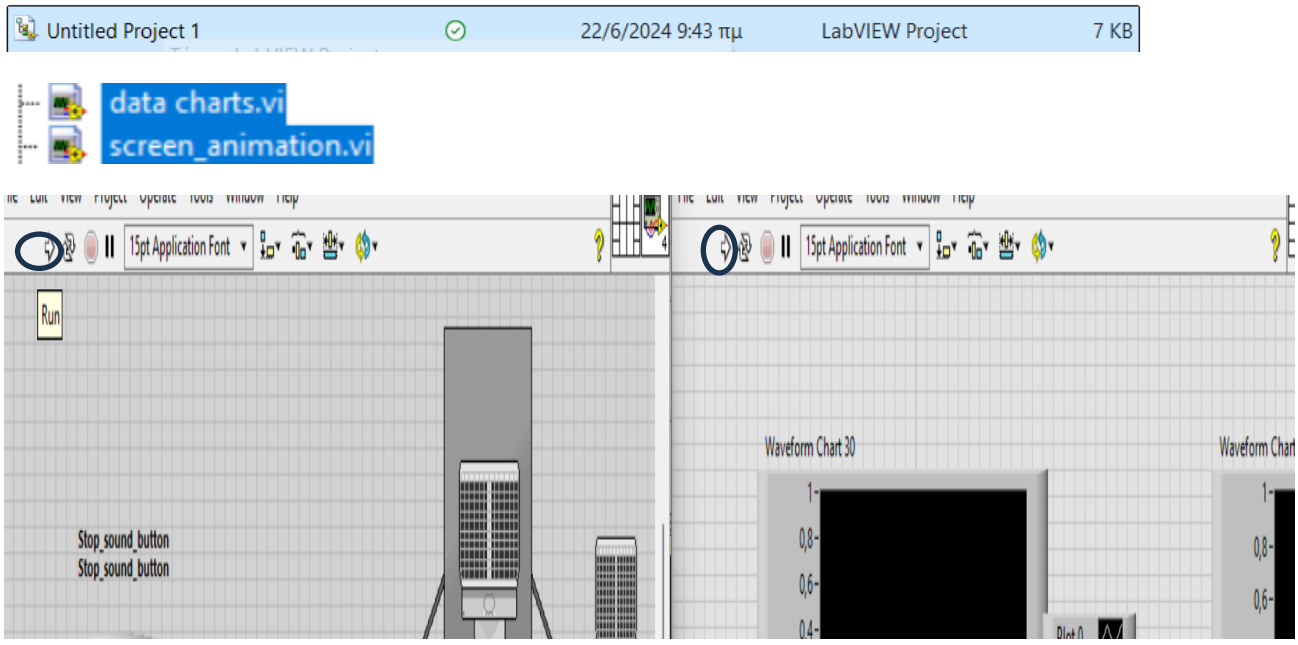
Εικόνα 68.Front Panel

Τα charts έχουν ρυθμιζόμενο πλήθος μετρήσεων καθώς και δυνατότητα export των δεδομένων τους. Ανάλογα με το χρόνο του πειράματος που εκτελείται μπορεί να ρυθμιστεί και το πλήθος των μετρήσεων. Πρέπει να τονιστεί πως κατά την αλλαγή του πλήθους μετρήσεων ο χρήστης οφείλει να λαμβάνει υπόψιν πως οι μετρήσεις καταγράφονται κάθε 300 ms .



Εικόνα 69. Ρύθμιση των charts

Το σύστημα λήψης δεδομένων μπορεί εύκολα να χρησιμοποιηθεί ανοίγοντας τον φάκελο του Project,επιλέγοντας το βασικό πρόγραμμα της οθόνης και των γραφικών απεικονίσεων και πατώντας το κουμπί της εκκίνησης.



Εικόνα 70. Οδηγίες Εκκίνησης Προγράμματος Scada

6.ΘΕΩΡΗΤΙΚΟ ΜΟΝΤΕΛΟ

Στο κεφάλαιο αυτό θα γίνει μία σύντομη περιγραφή του θεωρητικού μοντέλου καθώς και των εργαλείων τα οποία χρησιμοποιούνται για την δημιουργία του. Συγκεκριμένα θα περιγραφεί το μη αδιαβατικό μοντέλο το οποίο έχει αναπτυχθεί από την Ρόζα Χριστοδουλάκη τόσο για τον Απορροφητή όσο και για τον Αναγεννητή το οποίο θα υλοποιηθεί σε γλώσσα προγραμματισμού Python.

6.1 Εισαγωγή στην Python και στις βιβλιοθήκες IAPWS,CoolProps

6.1.1 Γλώσσα Προγραμματισμού Python

Η Python είναι μια ευέλικτη και ισχυρή γλώσσα προγραμματισμού που χρησιμοποιείται ευρέως για επιστημονικές και μηχανικές εφαρμογές, συμπεριλαμβανομένης της θερμοδυναμικής ανάλυσης. Τα κύρια οφέλη της περιλαμβάνουν:

- Ευκολία Χρήσης: Η απλή σύνταξη και η αναγνωσιμότητα της Python την καθιστούν προσιτή τόσο για αρχάριους όσο και για έμπειρους χρήστες.
- Εκτεταμένες Βιβλιοθήκες: Η Python διαθέτει έναν πλούσιο οικοσύστημα βιβλιοθηκών για επιστημονικούς υπολογισμούς, ανάλυση δεδομένων και οπτικοποίηση, όπως οι NumPy, SciPy και Matplotlib.
- Υποστήριξη Κοινότητας: Μια μεγάλη και ενεργή κοινότητα εξασφαλίζει συνεχή βελτίωση, εκτενή τεκμηρίωση και εύκολα διαθέσιμη βοήθεια.
- Διαλειτουργικότητα: Η Python μπορεί εύκολα να συνδέεται με άλλες γλώσσες και λογισμικά, ενισχύοντας την ευελιξία της σε σύνθετα έργα.

6.1.2 IAPWS: Διεθνής Ένωση για τις Ιδιότητες του Νερού και του Ατμού

Η IAPWS (International Association for the Properties of Water and Steam) παρέχει οδηγίες και διατυπώσεις για τις θερμοδυναμικές ιδιότητες του νερού και του ατμού. Το πρότυπο IAPWS-IF97, συγκεκριμένα, χρησιμοποιείται ευρέως στην ηλεκτροπαραγωγή και τη βιομηχανία διεργασιών.

Οφέλη της IAPWS για Θερμοδυναμική Ανάλυση:

1. Ακρίβεια: Οι διατυπώσεις της IAPWS βασίζονται σε εκτεταμένα πειραματικά δεδομένα και είναι εξαιρετικά ακριβείς για ένα ευρύ φάσμα θερμοκρασιών και πιέσεων.

2. Τυποποίηση: Αναγνωρισμένες παγκοσμίως, εξασφαλίζουν συνέπεια και αξιοπιστία σε βιομηχανικές και ερευνητικές εφαρμογές.

3. Ολοκληρωμένη Κάλυψη: Παρέχει λεπτομερείς ιδιότητες, όπως ενθαλπία, εντροπία, πυκνότητα, ειδική θερμότητα και πολλά άλλα, απαραίτητα για ολοκληρωμένη θερμοδυναμική ανάλυση.

4. Εξειδικευμένες Λειτουργίες: Οι διατυπώσεις της IAPWS είναι προσαρμοσμένες για το νερό και τον ατμό, καθιστώντας τες ιδιαίτερα κατάλληλες για εφαρμογές σε εργοστάσια παραγωγής ενέργειας, συστήματα HVAC και άλλες βιομηχανίες όπου το νερό και ο ατμός είναι σημαντικά.

6.1.3 CoolProp: Ανοικτού Κώδικα Βιβλιοθήκη Θερμοφυσικών Ιδιοτήτων

Η CoolProp είναι μια ανοικτού κώδικα βιβλιοθήκη που σχεδιάστηκε για να παρέχει ακριβείς και αποδοτικούς υπολογισμούς θερμοφυσικών ιδιοτήτων για μια μεγάλη ποικιλία ρευστών, συμπεριλαμβανομένων των ψυκτικών, των υδρογονανθράκων και του νερού/ατμού.

Οφέλη της CoolProp για Θερμοδυναμική Ανάλυση:

1. Ευρύ Φάσμα Ρευστών: Υποστηρίζει μια μεγάλη ποικιλία ρευστών πέρα από το νερό και τον ατμό, όπως ψυκτικά, υδρογονάνθρακες και βιομηχανικά αέρια.

2. Θερμοδυναμικές Ιδιότητες: Παρέχει πρόσβαση σε ιδιότητες όπως ενθαλπία, εντροπία, ειδικό όγκο, ειδική θερμότητα, ιξώδες και θερμική αγωγιμότητα.

3. Φιλικότητα προς τον Χρήστη: Απλή στη χρήση με καλά τεκμηριωμένο API, καθιστώντας την προσιτή τόσο για αρχάριους όσο και για έμπειρους χρήστες.

4. Αποδοτικότητα: Βελτιστοποιημένη για απόδοση, εξασφαλίζοντας γρήγορους υπολογισμούς, κάτι που είναι κρίσιμο για εφαρμογές σε πραγματικό χρόνο και μεγάλης κλίμακας προσομοιώσεις.

5. Κοινότητα με Συμμετοχή: Ως έργο ανοικτού κώδικα, ωφελείται από τις συνεισφορές και τις συνεχείς βελτιώσεις από μια παγκόσμια κοινότητα χρηστών και προγραμματιστών.

Στην θερμοδυναμική ανάλυση, η Python σε συνδυασμό με βιβλιοθήκες όπως η IAPWS και η CoolProp παρέχει ένα ισχυρό εργαλείο. Η ευκολία χρήσης της Python και το εκτεταμένο οικοσύστημά της υποστηρίζουν την αποδοτική και αποτελεσματική ανάπτυξη θερμοδυναμικών μοντέλων και προσομοιώσεων. Η IAPWS εξασφαλίζει

ακριβείς και τυποποιημένους υπολογισμούς για τις ιδιότητες του νερού και του ατμού, ενώ η CoolProp επεκτείνει αυτή τη δυνατότητα σε ένα ευρύτερο φάσμα ρευστών με υψηλή αποδοτικότητα και φιλικότητα προς τον χρήστη. Μαζί, σχηματίζουν μια στιβαρή βάση για την αντιμετώπιση σύνθετων θερμοδυναμικών προβλημάτων σε βιομηχανικά και ερευνητικά περιβάλλοντα.

6.2 Μη Αδιαβατικό Μοντέλο

Προτού παρατεθούν οι εξισώσεις του μοντέλου θα πρέπει να υπολογιστούν κάποιες βασικές ιδιότητες του Διαλύματος Χλωριούχου Λιθίου συναρτήσει της συγκέντρωσης και της θερμοκρασίας. Επίσης απαραίτητοι είναι και οι υπολογισμοί μερικών θερμοδυναμικών ιδιοτήτων του νερού του κορεσμένου ατμού καθώς και του ξηρού και υγρού αέρα .

- Πυκνότητα νερού ρ_w(Μέσω IAPWS)
- Ειδική Θερμική Αγωγιμότητα Νερού k_w(Μέσω Coolprops ή IAPWS)
- Δυναμικό ιξώδες Νερού μ_w(Μέσω Coolprops ή IAPWS)
- Ειδική θερμοχωρητικότητα Νερού cp_w(Μέσω Coolprops ή IAPWS)
- Ειδική θερμοχωρητικότητα Κορεσμένου Ατμού cp_{st}^{sat} (Μέσω Coolprops ή IAPWS)
- Πυκνότητα Διαλύματος LiCL ρ_s(Παράρτημα)
- Ειδική Θερμική Αγωγιμότητα LiCL k_s(Παράρτημα)
- Δυναμικό ιξώδες LiCl μ_s(Παράρτημα)
- Ειδική Θερμοχωρητικότητα LiCL cp_s(Παράρτημα)

- Συντελεστής Διάχυσης LiCL
  Ο σχετικός συντελεστής διάχυσης υπολογίζεται από τη σχέση

$$\frac{D_{licl}}{D_{H_2O}} = 1 - \left(1 + \left(\frac{\sqrt{X}}{0.52}\right)^{-4.92}\right)^{-0.56}$$

  Παραδοχή: Ο συντελεστής διάχυσης του νερού λαμβάνεται περίπου σταθερός και ίσος με $D_{H_{2O}} = 2.3 * 10^{-9}$
- Λανθάνουσα θερμότητα ατμοποίησης/συμπύκνωσης(Παράρτημα)
- Απόλυτη υγρασία Κορεσμού Αέρα σε ισορροπία με το διάλυμα Χλωριούχου Λιθίου

$$w^{SAT}(T_s) = 0.622 * \frac{p_{st}^{sat}(T_s)}{p_{atm} - p_{st}^{sat}(T_s)}$$

Όπου $P_{atm} = 101325\ Pa$ και $p_{st}^{sat}(T_s)$ η πίεση κορεσμού του ατμού συναρτήσει της θερμοκρασίας η οποία βρίσκεται μέσω της βιβλιοθήκης Coolprops

- Θερμική Αγωγιμότητα Αέρα k_a(Μέσω CoolProps)
- Δυναμικό Ιξώδες Αέρα μ_α(Μέσω CoolProps)
- Πυκνότητα Αέρα ρ_α(Μέσω CoolProps)
- Ειδική Θερμοχωρητικότητα Αέρα cp_a(Μέσω Coolprops)
- Ειδική Θερμοχωρητικότητα Αέρα με υγρασία cp_{m_a}(Μέσω CoolProps)
- Αριθμός Nusselt Νερού $N_w = 7.54$

ΣΥΝΤΕΛΕΣΤΕΣ ΜΕΤΑΦΟΡΑΣ ΜΑΖΑΣ

Για το νερό :

$$a_w = \frac{Nu_w * k_w}{Dh_{in}}$$

Όπου Dh_{in} είναι η εσωτερική υδραυλική διάμετρος της πλάκας

Για το διάλυμα:

Οι αριθμοί Reynolds και Prandtl υπολογίζονται ως εξής:

$$Re_s = \frac{4 * m_s}{L * \mu_s}$$

Όπου $\mathrm{m_s}$ και L η παροχή μάζας και το μήκος του αγωγού αντίστοιχα

$$Pr_s = \frac{cp_s * \mu_s}{k_s}$$

Ο αριθμός Nusselt υπολογίζεται ως εξής :

$$Nu_s = 0.680 * Re_s^{0.5} * Pr_s^{\frac{1}{3}}$$

$$\alpha_s = \frac{Nu_s * k_s}{Dh_s}$$

Όπου $Dh_s = D_{tube} - 2\delta$ είναι η διάμετρος του αγωγού στον οποίο κυκλοφορεί το διάλυμα και δ το πάχος του ρευστού

$$\delta = \left(3 * \frac{v_s^2}{9.81}\right)^{\frac{1}{3}} * Re^{\frac{1}{3}}$$

Όπου $v_s = \frac{\mu_s}{\rho_s}$ $(κινηματικό\ ιξώδες\ του\ διαλύματος)$

Για τον αέρα:

Αριθμός Reynolds :

$$Re_a = \frac{V_a * Dh_{out}}{\rho_\alpha * v_a}$$

Όπου $V_a = \frac{m_a}{\pi * \frac{D^2}{4}}$ και $v_a = \frac{\mu_\alpha}{\rho_\alpha}$

Αριθμός Prandtl :

$$Pr_\alpha = \frac{cp_\alpha * \mu_\alpha}{k_\alpha}$$

Αριθμός Nusselt :

$$Nu_a = 0.023 * Re_a^{0.8} * Pr_a^{\frac{1}{3}}$$

Συντελεστής μεταφοράς μάζας :

$$\alpha_\alpha = \frac{Nu_a * k_a}{Dh_{out}}$$

Ο συντελεστής θερμοπερατότητας:

$$U = \left[\frac{1}{a_w} + \frac{d}{k_p} + \frac{1}{a_s}\right]^{-1}$$

Όπου d το πάχος της πλάκας του εναλλάκτη και k_p η θερμική αγωγιμότητα της πλάκας του εναλλάκτη (τυπικά 0.15-0.25 για PVC).

Ο συνολικός συντελεστής μεταφοράς μάζας υπολογίζεται παρακάτω :

$$K_G = \frac{a_a}{cp_{m_a} * Le}$$

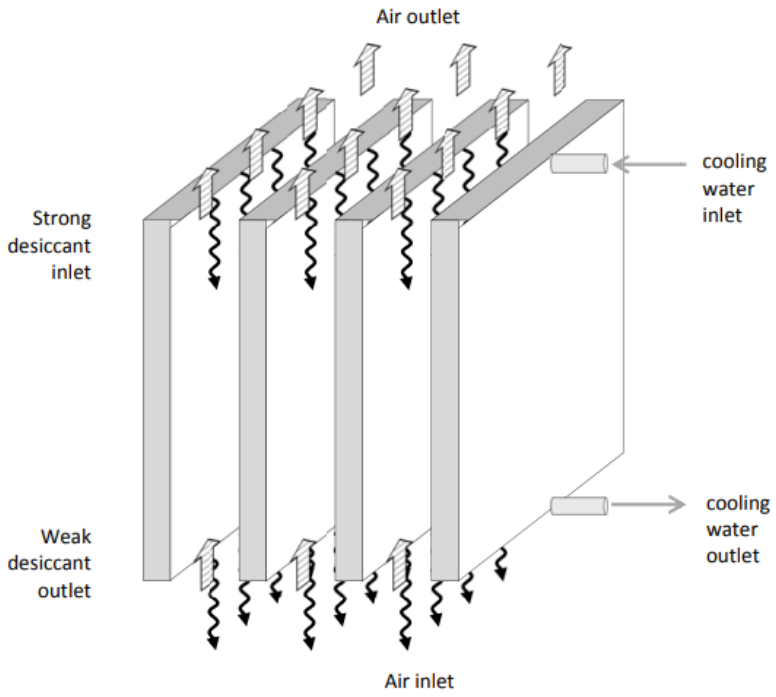
Ο αριθμός Lewis (Le) είναι ένας αδιάστατος αριθμός που χρησιμοποιείται για να περιγράψει τη σχέση μεταξύ της διάχυσης θερμότητας και της διάχυσης μάζας σε ένα ρευστό. Αποτελεί ένα χρήσιμο εργαλείο στη μελέτη των διαδικασιών μεταφοράς θερμότητας και μάζας, όπως σε εναλλάκτες θερμότητας, καύσεις και χημικές αντιδράσεις

$$Le = \frac{a}{D} \rightarrow Le = \frac{k_{licl}}{\rho_{licl} c_{p_{licl}} D_{licl}}$$

Στην παρούσα εργασία θα ληφθεί ίσος με τη μονάδα

6.2.1 Μαθηματικό μοντέλο Απορροφητή

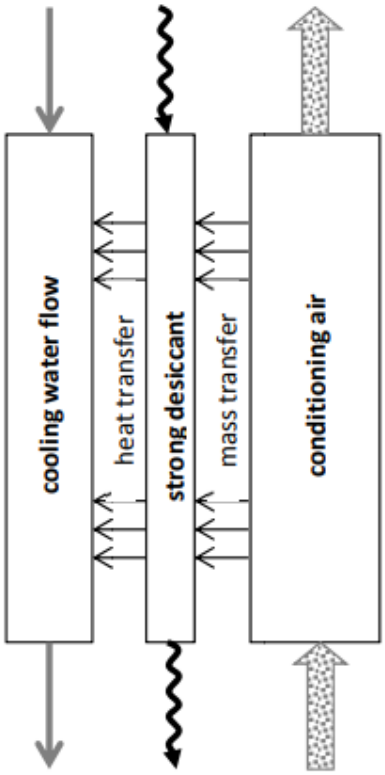
Τα βασικά γεωμετρικά χαρακτηριστικά του απορροφητή είναι : ύψος πλάκας εναλλάκτη 0.5 m, μήκος πλάκας εναλλάκτη 1 m, πάχος πλάκας 4.87 mm , πάχος τοιχώματος 0.35 mm και απόσταση μεταξύ δύο διαδοχικών πλακών 3.3 mm.



Εικόνα 71. Ροή στον εναλλάκτη του απορροφητή

Στη συνέχεια θα ακολουθήσουν μερικές παραδοχές προκειμένου να απλοποιηθούν οι υπολογισμοί.

- Μονοδιάστατη ανάλυση , δηλαδή η συνδιαλλαγή μάζας και θερμότητας γίνονται μόνο στην διεύθυνση της ροής.
- Η θερμοκρασία του διαλύματος θα είναι ίση με τη θερμοκρασία των πλακών
- Ομοιόμορφη «πλήρωση» του απορροφητή ενώ οι περιοχές συνδιαλλαγής μάζας και θερμότητας είναι ίσες με την ειδική επιφάνεια του εναλλάκτη
- Ισορροπία πιέσεων μεταξύ των υδρατμών και του διαλύματος
- Όλη η θερμότητα κατά την απορρόφηση εκλύεται στην διεπιφάνειεα
- Αμελητέα μεταφορά θερμότητας από το υγρό στον υδρατμό, καθώς και αμελητέα μεταφορά θερμότητας λόγω ακτινοβολίας,ιξώδους διασποράς, μεταβολής πίεσης, συγκέντρωσης ή βαρυτικής επίδρασης



Εικόνα 72. Ροή μάζας στον απορροφητή

Η απόλυτη υγρασία του άερα ορίζεται ως το πηλίκο μάζας νερού προς τη συνολική μάζα του αέρα.

$$W = \frac{m_w}{m_a} \rightarrow \frac{dm_w}{dA} = m_a \frac{dW}{dA}$$

Κατά την αφύγρανση η υγρασία απορροφάται από το διάλυμα LiCl. Επομένως

$$dm_s = dm_w \rightarrow \boldsymbol{\frac{dm_s}{dA} = m_a * \frac{dW}{dA}}$$

Σύμφωνα με την αρχή διατήρησης της μάζας προκύπτει ότι

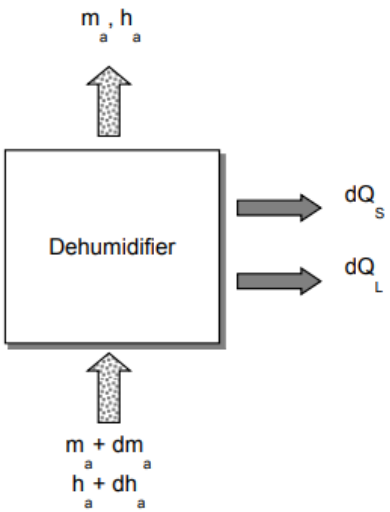
$$d(m_s X) = 0 \rightarrow \boldsymbol{\frac{dX}{dA} = -\frac{m_a}{m_s} X * \frac{dW}{dA}}$$

Και η μεταβολή της υγρασία η οποία εκφράζει το ποσό υγρασίας που απορροφάται από το διάλυμα εκφράζεται με την παρακάτω σχέση

$$\boldsymbol{\frac{dW}{dA} = -\frac{K_G}{m_a}(W_{sat} - W_{in})}$$

Σύμφωνα με το νόμο διατήρησης της ενέργειας για τον αέρα προκύπτει ότι η συνολική μεταφερόμενη ενέργεια από τον αέρα είναι :

$$m_a dh_a = dQ_s + dQ_{Latent}$$



Εικόνα 73. Ροή θερμότητας για τον αέρα

Όπου dQs είναι η θερμότητα που μεταφέρεται από τον αέρα στο διάλυμα λόγω της διαφοράς θερμοκρασίας τους→ $dQ_s = a_a(T_a - T_s)dA$

και dQ_{latent} η θερμότητα που μεταφέρεται στο διάλυμα λόγω της συμπύκνωσης υδρατμών.→ $dQ_{Latent} = dm_s * (cp_{st}^{sat} T_s + \Delta h_{abs})$

Η ενθαλπία του αέρα συμπεριλαμβανόμενης της υγρασίας του εκφράζεται ως :

$$h_a = cp_{da} * T_a + W * (cp_{st}^{sat} T_a + \Delta h_{abs})$$

Συνδυάζοντας τις παραπάνω εξισώσεις και λαμβάνοντας υπόψη ότι

$cp_{ma} = cp_{da} + W * cp_{st}^{sat}$ προκύπτει η εξίσωση που περιγράφει την μεταβολή της θερμοκρασίας του αέρα

$$\boldsymbol{\frac{dT_a}{d_A} = -\frac{T_a - T_s}{cp_{ma}}\left(a_a - cp_{st}^{sat} * \frac{dW}{dA}\right)}$$

Για το διάλυμα η μεταβολή της «ενέργειας» του ισούται με

$$m_s dh_s + h_s dm_s = dQ_{sw} + dQ_{Latent}$$

Επίσης η ενθαλπία του διαλύματος δίνεται από την εξίσωση

$$h_s = -cp_s * \left(T_s - T_{s_{in}}\right) \rightarrow dh_s = cp_s dT_s$$

Ενώ η θερμότητα που μεταφέρεται από το διάλυμα στο νερό είναι :

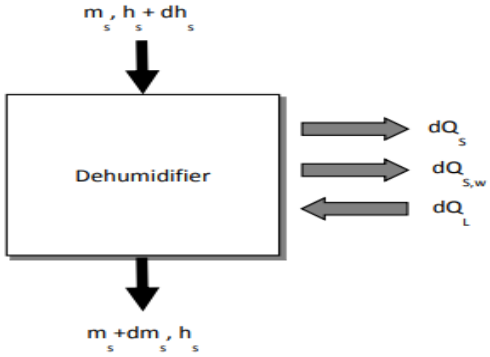
$$dQ_{s_W} = U * (T_s - T_w)dA = m_w * cp_w * dT_w$$

Επομένως η μεταβολή της θερμοκρασίας του διαλύματος προκύπτει συννδυάζοντας τις παραπάνω εξισώσεις ως

$$\mathbf{\frac{dT_s}{dA} = -\frac{1}{m_s cp_s}\left[m_a\left[cp_s\left(T_s - T_{s_{in}}\right) + cp_{st}^{sat} T_s + \Delta h_{abs}\right]\frac{dW}{dA} + a_a(T_s - T_a) + U(T_w - T_s)\right]}$$

Επίσης προκύπτει και η μεταβολή της θερμοκρασίας του νερού ως εξής :

$$\boldsymbol{\frac{dT_w}{dA} = \frac{U(T_s - T_w)}{m_w cp_w}}$$



Εικόνα 74. Ροή θερμότητας για το διάλυμα

Το σύστημα των εξισώσεων που τίθεται προς επίλυση είναι το παρακάτω

$$\begin{cases} \dfrac{dT_a}{d_A} = -\dfrac{T_a - T_s}{cp_{ma}}\left(a_a - cp_{st}^{sat} * \dfrac{dW}{dA}\right) \\ \dfrac{\mathbf{dT_s}}{\mathbf{dA}} = -\dfrac{\mathbf{1}}{\mathbf{m_s cp_s}}\left[\mathbf{m_a}\left[\mathbf{cp_s}\left(\mathbf{T_s} - \mathbf{T_{s_{in}}}\right) + \mathbf{cp_{st}^{sat}T_s} + \mathbf{\Delta h_{abs}}\right]\dfrac{\mathbf{dW}}{\mathbf{dA}} + \mathbf{a_a}(\mathbf{T_s} - \mathbf{T_a}) + \mathbf{U}(\mathbf{T_w} - \mathbf{T_s})\right] \\ \dfrac{dT_w}{dA} = \dfrac{U(T_s - T_w)}{m_w cp_w} \\ \dfrac{dW}{dA} = -\dfrac{K_G}{m_a}(W_{sat} - W_{in}) \\ \dfrac{dm_s}{dA} = m_a * \dfrac{dW}{dA} \\ \dfrac{dX}{dA} = -\dfrac{m_a}{m_s}X * \dfrac{dW}{dA} \end{cases}$$

Οι συνοριακές συνθήκες του προβλήματος έχουν ως εξής:

$$\begin{cases} (T_a)_{z=0} = T_{a_{in}} \\ (T_s)_{z=L} = T_{s_{in}} \\ (T_w)_{z=L} = T_{w_{in}} \\ (W)_{z=0} = W_{in} \\ (m_s)_{z=L} = m_{s_{in}} \\ (X)_{z=L} = X_{in} \end{cases}$$

Ακόμα θα πρέπει να οριστεί η παροχή μάζας του αέρα και η παροχή νερού. Στον παρακάτω πίνακα συνοψίζονται όλες οι τιμές των παραπάνω συνοριακών συνθηκών και τα γεωμετρικά χαρακτηριστικά που χρησιμοποιούνται στους υπολογισμούς

Θερμοκρασία Εισόδου Διαλύματος	$T_{s_{in}} = 22℃$
Παροχή Μάζας Διαλύματος στην Είσοδο	$m_{s_{in}} = 0.013 \frac{kg}{s}$
Συγκέντρωση Πυκνού Διαλύματος	$X_s = 0.4$
Θερμοκρασία Εισόδου Αέρα	$T_{a_{in}} = 30℃$
Παροχή Αέρα	$m_a = 0.5 \frac{kg}{s}$
Θερμοκρασία Εισόδου Νερού	$T_w = 15℃$
Παροχή Νερού	$m_w = 0.25 \frac{kg}{s}$
Εμβαδό Εισόδου (Για υπολογισμό ταχύτητας αέρα)	A=0.219
Διάμετρος Εισόδου	$D_{h_{in}} = 0.016\ m$
Διάμετρος Εξόδου	$D_{h_{out}}$=0.02 m
Πάχος Τοιχώματος	d=0.00035 m
Μήκος	L=1m
kp	0.19 W/mK

Πίνακας 20. Αρχικές Συνθήκες για μοντέλο Απορροφητή

Για τον υπολογισμό της απόλυτης υγρασίας του αέρα ακολουθείται η παρακάτω διαδικασία θεωρώντας γνωστή την σχετική του υγρασία(RH) καθώς και την θερμοκρασία του.

- Αρχικά γίνεται υπολογισμός της πίεσης κορεσμού του αέρα (p^{sat}) μέσω της βιβλιοθήκης COOLPROPS συναρτήσει της θερμοκρασίας.
- Η πίεση των υδρατμών θα είναι $p_{vapor} = \frac{RH}{100} * p^{sat}$
- Η απόλυτη υγρασία υπολογίζεται μέσω της βιβλιοθήκης Coolprops σε ατμοσφαιρική πίεση ως εξής

from CoolProp.HumidAirProp import HAProp

AH = HAPropsSI('W', 'T', T, 'P', 101325, 'RH', RH/100.0)

Ή μέσω της παρακάτω σχέσης

$$W = 0.622 * \frac{p_{vapor}}{P_{atm} - p_{vapor}}$$

Από τα παραπάνω προκύπτει με αρχική θεώρηση RH=50% και θερμοκρασία Ta=30°C ότι η υγρασία εισόδου $\boldsymbol{W_{in} = 0.012 \frac{kg_{νερού}}{kg_{αέρα}}}$

Το παραπάνω πρόβλημα λύνεται ως πρόβλημα συνοριακών συνθηκών με τη βοήθεια της βιβλιοθήκης scipy.integrate . Πρέπει να σημεωθεί πως η συγκεκριμένη επίλυση

απαιτεί μία αρχική τιμή των μεταβλητών κατά μήκος όλου το εύρους όπου πρόκειται να γίνει η ολοκλήρωση. Οι τιμές αυτές θα ανανεώνονται κατάλληλα έτσι ώστε να ικανοποιούνται οι συνοριακές συνθήκες. Οι αρχικές εκτιμήσεις των μεταβλητών θα είναι “arrays” συνολικού μήκους όσα και τα βήματα ολοκλήρωσης τα οποία θα περιέχουν σε κάθε θέση τις τιμές των μεταβλητών σύμφωνα με τις συνοριακές συνθήκες.

Ακολουθεί απόσπασμα του κώδικα σε python το οποίο πραγματοποιεί πολύ εύκολα την παραπάνω διαδικασία.

```
def initial_guess(A):
   return np.array([
      np.full_like(A, T_a_in),
      np.full_like(A, T_s_in),
      np.full_like(A, T_w_in),
      np.full_like(A, W_in),
      np.full_like(A, m_s_in),
      np.full_like(A, X_s])
```

6.2.2 Μαθηματικό Μοντέλο Αναγεννητή

Η διαδικασία που ακολουθείται για τον αναγεννητή είναι ακριβώς ίδια με αυτή που ακολουθείται για τον απορροφητή. Οι εξισώσεις δηλαδή προκύπτουν εφαρμόζοντας τις εξισώσεις διατήρησης ενέργειας και μάζας τόσο για το διάλυμα όσο και για τον αέρα και διαφορίζοντας κατά μήκος του αναγεννητή.

Οι απαραίτητες παράμετροι για την εύρεση των συντελεστών μεταφοράς μάζας και θερμότητας καθώς και για την επίλυση του συστήματος παρατίθενται παρακάτω.

Θερμοκρασία Εισόδου Διαλύματος	$T_{s_{in}} = 22℃$
Παροχή Μάζας Διαλύματος στην Είσοδο	$m_{s_{in}} = 0.013 \frac{kg}{s}$
Συγκέντρωση Πυκνού Διαλύματος	$X_s = 0.34$
Θερμοκρασία Εισόδου Αέρα	$T_{a_{in}} = 30℃$
Παροχή Αέρα	$m_a = 0.2 \frac{kg}{s}$
Θερμοκρασία Εισόδου Νερού	$T_w = 55℃$

Παροχή Νερού	$m_w = 0.15 \frac{kg}{s}$
Εμβαδό Εισόδου (Για υπολογισμό ταχύτητας αέρα)	A=0.219
Διάμετρος Εισόδου	$D_{h_{in}} = 0.016\ m$
Διάμετρος Εξόδου	$D_{h_{out}}$=0.02 m
Πάχος Τοιχώματος	d=0.00035 m
Μήκος	L=1m
kp	0.19 W/mK
Απόλυτη υγρασία Αέρα εισαγωγής	$W_{in} = 0.012 \frac{kg_{νερού}}{kg_{αέρα}}$

Πίνακας 21. Αρχικές Συνθήκες για μοντέλο Αναγεννητή

Το σύστημα εξισώσεων έχει ως εξής:

$$\begin{cases} \dfrac{dT_a}{d_A} = -\dfrac{T_a - T_s}{cp_{ma}}\left(a_a - cp_{st}^{sat} * \dfrac{dW}{dA}\right) \\ \dfrac{\mathbf{dT_s}}{\mathbf{dA}} = \dfrac{\mathbf{1}}{\mathbf{m_s cp_s}}\left[\mathbf{m_a}\left[-\mathbf{cp_s}\left(\mathbf{T_s} - \mathbf{T_{s_{in}}}\right) + \mathbf{cp_{st}^{sat}T_s} + \mathbf{\Delta h_{abs}}\right]\dfrac{\mathbf{dW}}{\mathbf{dA}} - \mathbf{a_a}(\mathbf{T_s} - \mathbf{T_a}) - \mathbf{U}(\mathbf{T_w} - \mathbf{T_s})\right] \\ \dfrac{dT_w}{dA} = -\dfrac{U(T_s - T_w)}{m_w cp_w} \\ \dfrac{dW}{dA} = \dfrac{K_G}{m_a}(W_{sat} - W_{in}) \\ \dfrac{dm_s}{dA} = m_a * \dfrac{dW}{dA} \\ \dfrac{dX}{dA} = -\dfrac{m_a}{m_s}X * \dfrac{dW}{dA} \end{cases}$$

Ενώ οι συνοριακές συνθήκες παραμένουν ίδιες

$$\begin{cases} (T_a)_{z=0} = T_{a_{in}} \\ (T_s)_{z=L} = T_{s_{in}} \\ (T_w)_{z=L} = T_{w_{in}} \\ (W)_{z=0} = W_{in} \\ (m_s)_{z=L} = m_{s_{in}} \\ (X)_{z=L} = X_{in} \end{cases}$$

6.3 Αποτελέσματα Θεωρητικού Μοντέλου

Τα αποτελέσματα του παραπάνω υπολογιστικού μοντέλου το οποίο υλοποιήθηκε σε γλώσσα Python δίνονται στον παρακάτω πίνακα.

Απορροφητής	
Αέρας	$T_{a_{in}} = 30\ ℃$ $T_{a_{out}} = 25℃$
Διάλυμα	$T_{s_{in}} = 22℃$ $T_{s_{out}} = 18.7℃$
	$X_{in} = 0.41$ $X_{out} = 0.328$
Νερό	$T_{w_{in}} = 15\ ℃$ $T_{w_{out}} = 21℃$
Απόλυτη Υγρασία	$W_{in} = 0.008$ $W_{out} = 0.0015$
Παροχή Μάζας Διαλύματος	$m_{s_{in}} = 0.013$ $m_{s_{out}} = 0.0162$

Αναγεννητής	
Αέρας	$T_{a_{in}} = 30\ ℃$ $T_{a_{out}} = 38℃$
Διάλυμα	$T_{s_{in}} = 22℃$ $T_{s_{out}} = 49.5℃$
	$X_{in} = 0.33$ $X_{out} = 0.415$
Νερό	$T_{w_{in}} = 55℃$ $T_{w_{out}} = 50℃$
Απόλυτη Υγρασία	$W_{in} = 0.008$ $W_{out} = 0.021$
Παροχή Μάζας Διαλύματος	$m_{s_{in}} = 0.013$ $m_{s_{out}} = 0.0103$

Πίνακας 22.Αποτελέσματα Υπολογιστικού Μοντέλου

7.ΠΕΙΡΑΜΑ

Προκειμένου να γίνει επαλήθευση του μοντέλου αλλά και για να εξετασθεί η εγκυρότητα των μετρήσεων που λαμβάνονται και να ελεγχθούν πιθανά λάθη στην βαθμονόμηση των μετρητικών οργάνων πρέπει να τεθεί σε λειτουργία η πειραματική μονάδα Ψύξης-Αφύγρανσης.

Παρακάτω ακολουθούν οι οδηγίες ,που έχουν συνταχθεί από τον κύριο Σωτήρη Υφαντή ,οι οποίες ακολουθήθηκαν προκειμένου να τεθεί το σύστημα σε λειτουργία και να ισορροπήσει προκειμένου να ληφθούν σωστές μετρήσεις.

ΕΚΚΙΝΗΣΗ ΛΕΙΤΟΥΡΓΊΑ ΚΑΙ ΑΠΟΘΕΡΜΑΝΣΗ ΣΥΣΤΉΜΑΤΟΣ LDCS (LiBr)

Ανεβάζουμε τις ασφάλειες στον κεντρικό ηλεκτρολογικό πίνακα και πατάμε τα μπουτόν start (power) σε όλα τα μηχανήματα.
(Οι αντλίες επιστροφής πυκνού και αραιού διαλύματος προς τις δεξαμενές θα λειτουργήσουν αν χρειάζεται).

Πριν ξεκινήσουμε ελέγχουμε την πίεση του νερού στο εσωτερικό μανόμετρο του Αναγεννητή REG και του ψύκτη IEC.

Αν η πίεση είναι κάτω από 0,5bar συμπληρώνουμε νερό από το δίκτυο.

ΕΝΑΡΞΗ
-Ανεβάζουμε τις ασφάλειες των αντιστάσεων και του κυκλοφορητή κυκλώματος μπόιλερ.(Η θερμοκρασία ρυθμίζεται μεταξύ 50 και 60^{o}C αυτόματα από τους θερμοστάτες).
-Πατάμε το μπουτόν αέρα και νερού στον Αναγεννητή REG και τον προθερμαίνουμε.
-Ταυτόχρονα πατάμε τα μπουτόν αέρα και νερού στον ψύκτη IEC και στον Απορροφητή (ABS) και τους προψύχουμε.
-Περιμένουμε να ζεσταθεί το νερό (REG)έως τους 40^{o}C(ένδειξη θερμομέτρου μπόιλερ).
-Πατάμε τα μπουτόν διαλύματος (solution) στον Abs και στον REG.
-Μετά από 30 λεπτά το σύστημα ισορροπεί και μπορούμε να πάρουμε μετρήσεις.
ΑΠΟΘΕΡΜΑΝΣΗ
-Πρώτα κατεβάζουμε τις ασφάλειες των αντιστάσεων.

-Όταν η θερμοκρασία πέσει στους 40^{o}C κλείνουμε τα διαλύματα (REG-ABS) και τους ανεμιστήρες (IEC –Abs).

Όταν η θερμοκρασία πέσει στους 25^{o}C κλείνουμε τα πάντα.

Το πείραμα είχε διάρκεια 3 ωρών και παρακάτω θα παρατεθεί ο πίνακας που περιέχει τον μέσο όρο κάθε μέτρησης.

Οι μετρήσεις της σχετικής υγρασίας μετατρέπονται κατάλληλα ώστε να μπορούν να συγκριθούν με το μοντέλο

7.1 Σύγκριση Αποτελεσμάτων Πειράματος-Θεωρητικού Μοντέλου

Απορροφητής	
Αέρας	$T_{a_{in}} = 30\,℃$ $T_{a_{out}} = 21℃$ $T_{a_{out_{sim}}} = 25℃$
Διάλυμα	$T_{s_{in}} = 22℃$ $T_{s_{out}} = 18℃$ $T_{s_{out_{sim}}} = 19℃$
	$X_{in} = 0.41$ $X_{out} = 0.34$ $X_{out_{sim}} = 0.328$
Νερό	$T_{w_{in}} = 15\,℃$ $T_{w_{out}} = 25℃$ $T_{w_{out_{sim}}} = 21℃$
Απόλυτη Υγρασία	$W_{in} = 0.008$ $W_{out} = 0.0023$ $W_{out_{sim}} = 0.0015$
Παροχή Μάζας Διαλύματος	$m_{s_{in}} = -$ $m_{s_{out}} = -$

Αναγεννητής	
Αέρας	$T_{a_{in}} = 30\,℃$ $T_{a_{out}} = 36℃$ $T_{a_{out_{sim}}} = 38℃$
Διάλυμα	$T_{s_{in}} = 22℃$ $T_{s_{out}} = 44℃$ $T_{s_{out_{sim}}} = 50℃$
	$X_{in} = 0.33$ $X_{out} = 0.4$ $X_{out_{sim}} = 0.415$
Νερό	$T_{w_{in}} = -$ $T_{w_{out}} = 65℃$

Απόλυτη Υγρασία	$W_{in} = 0.0085$ $W_{in_{sim}} = 0.008$ $W_{out} = 0.0228$ $W_{out_{sim}} = 0.0213$
Παροχή Μάζας Διαλύματος	$m_{s_{in}} = -$ $m_{s_{out}} = -$

Πίνακας 23.Αποτελέσματα Πειράματος και Υπολογιστικού Μοντέλου

Λόγω μη ύπαρξης μετρητικών οργάνων παροχής διαλύματος καθώς και μη ύπαρξης μετρητικού οργάνου θερμοκρασίας εισαγωγής νερού στον αναγεννητή δεν μπορεί να γίνει μία πλήρης σύγκριση του μοντέλου με τις μετρήσεις.

7.2 Αποκλίσεις

Παρακάτω ακολουθεί ο πίνακας αποκλίσεων των μεγεθών για τα οποία δύναται να γίνει σύγκριση

Απορροφητής				
$T_{a_{out}}$	$T_{s_{out}}$	X_{out}	$T_{w_{out}}$	W_{out}
19%	5%	3.5%	16%	24%

Πίνακας 24.Αποκλίσεις Απορροφητή

- Η θερμοκρασία του αέρα η οποία υπολογίζεται στο μοντέλο δεν λαμβάνει η υπόψην την περαιτέρω μείωση της θερμοκρασίας του αέρα μέσω του εισερχόμενου νερού από τον εξατμιστικό ψύκτη στον εναλλάκτη που βρίσκεται πάνω από τον κύριο εναλλάκτη του απορροφητή. Επομένως μία τέτοια απόκλιση είναι αναμενόμενη
- Η θερμοκρασία του διαλύματος φαίνεται να έχει μικρή απόκλιση η οποία ίσως οφείλεται στην βαθμονόμηση των μετρητικών και όχι αποκλειστικά στο μοντέλο. Μια άλλη πιθανή αιτία είναι η θεώρηση του αριθμού Lewis ως σταθερό και ίσο με τη μονάδα
- Η συγκέντρωση του διαλύματος δεν παρουσιάζει σημαντική απόκλιση καθώς οι τιμές προέρχονται από στρογγυλοποιήσεις. Επομένως η απόκλιση που φαίνεται δεν ανταποκρίνεται πλήρως στα πραγματικά δεδομένα
- Η θερμοκρασία του νερού παρουσιάζει απόκλιση ίσως λόγω υποεκτίμησης του συντελεστή U που προκύπτει από λανθασμένη εκτίμηση της θερμικής αγωγιμότητας των πλακών αλλά και του συντελεστή μεταφοράς θερμότητας του νερού
- Η υγρασία του αέρα παρουσιάζει σημαντική απόκλιση ενώ φαίνεται ότι το μοντέλο υπερεκτιμά την δυνατότητα αφύγρανσης της μονάδας. Αυτό ίσως

οφείλεται στην θεώρηση σταθερού αριθμού Lewis. Ακόμα πρέπει να σημειωθεί πως στο πείραμα γίνεται μέτρηση της σχετικής υγρασίας του αέρα η οποία μπορεί να αλλάξει με μεταβολή της θερμοκρασίας. Αυτό εξηγείται ως εξής: Δεδομένου του ότι ο αέρα εξέρχεται πιο ψυχρός λόγω της ύπαρξης επιπλέον εναλλάκτη μειώνεται η ικανότητα του όσον αφορά τη συγκράτηση υδρατμών και άρα η σχετική του υγρασία αυξάνεται. Αυτό έχει ως αποτέλεσμα την αύξηση της απόλυτης υγρασίας λόγω του τρόπου υπολογισμού που έχει παρατεθεί παραπάνω.

Αναγεννητής			
$T_{a_{out}}$	$T_{s_{out}}$	X_{out}	W_{out}
5%	13%	2.5%	6.5%

Πίνακας 25.Αποκλίσεις Αναγεννητή

- Φαίνεται πως στον Αναγεννητή η θερμοκρασία εξόδου αέρα παρουσιάζει μικρή απόκλιση που πιθανότατα οφείλεται σε απόκλιση στη βαθμονόμηση του μετρητικού οργάνου. Το γεγονός αυτό βέβαια υποδεικνύει πως η απόκλιση στον απορροφητή ίσως προέρχεται από κάποια δυσλειτουργία του οργάνου κατά τη διάρκεια του πειράματος,
- Η θερμοκρασία του διαλύματος παρουσιάζει απόκλιση που οφείλεται σε μεγάλο βαθμό στην θεώρηση σταθερού αριθμού Lewis.
- Η συγκέντρωση του διαλύματος στην έξοδο δεν παρουσιάζει ουσιαστική απόκλιση
- Η απόκλιση της υγρασίας του αέρα οφείλεται στο γεγονός ότι στο μοντέλο δεν λαμβάνεται υπόψιν η ύπαρξη ενός επιπλέον εναλλάκτη προθέρμανσης του εισερχόμενου αέρα ο οποίος έρχεται σε επαφή με τον εξερχόμενο αέρα. Και σε αυτή την περίπτωση ο εξερχόμενος αέρας θα έχει μικρότερη θερμοκρασία από αυτή που δείχνει ο αισθητήρας ο οποίος βρίσκεται πριν τον επιπλέον εναλλάκτη. Άρα με μείωση της θερμοκρασίας η σχετική υγρασία αυξάνεται γεγονός που οδηγεί σε αύξηση της απόλυτης υγρασίας λόγω του τρόπου υπολογισμού της σύμφωνα με το μοντέλο.

Οι παραπάνω αποκλίσεις φαίνεται πως μπορούν να αποφευχθούν με κατάλληλη τοποθέτηση και καλύτερη βαθμονόμηση των αισθητήρων και ταυτόχρονα με επαναϋπολογισμό του αριθμού Lewis σε κάθε επανάληψη στο μοντέλο. Ακόμα θα μπορούσε να δημιουργηθεί εκ νέου ένα μοντέλο για όλη την εγκατάσταση βασισμένο στο προϋπάρχον , προκειμένου να συμπεριληφθούν οι επιπλέον εναλλάκτες αλλά και ο εξατμιστικός ψύκτης. Έτσι θα μπορέσει να πραγματοποιηθεί ένας ενεργειακός

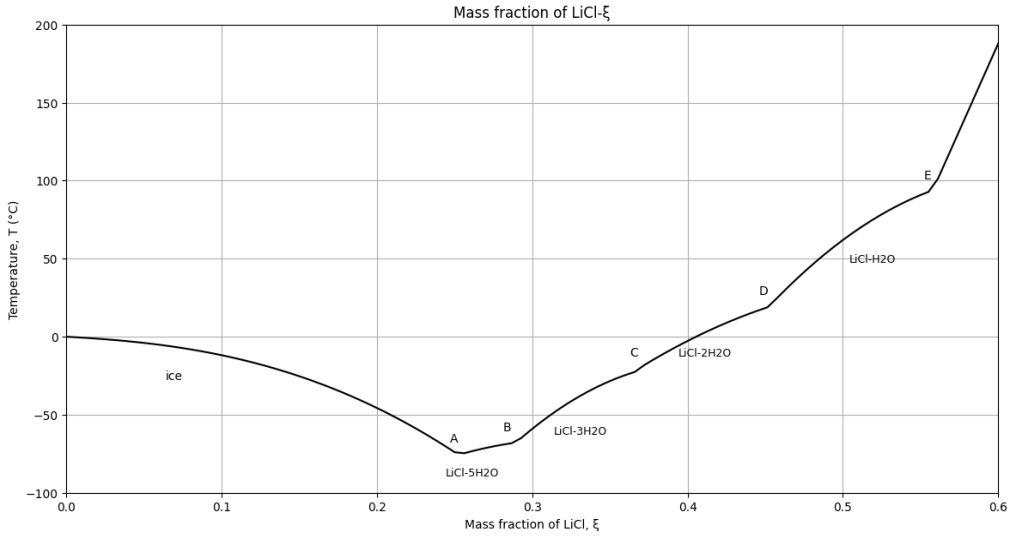
ισολογισμός της μονάδας και πιθανότατα μία παραμετροποίηση του συστήματος με στόχο τη βελτιστοποίηση της λειτουργίας του.

7.3 Ιδιότητες Χλωριούχου Λιθίου και Διάγραμμα Ισορροπίας

Στα πλαίσια της διπλωματικής εργασίας δημιουργήθηκε και ο κώδικας για εύκολο υπολογισμό των ιδιοτήτων του Χλωριούχου Λιθίου ανάλογα με την συγκέντρωση και την θερμοκρασία. Τα σημαντικότερα από αυτά παρατίθενται σε αυτό το κεφάλαιο ενώ στο Παράρτημα παρατίθενται και κάποια επιπλέον «βοηθητικά» διαγράμματα των ιδιοτήτων του διαλύματος.

Διάγραμμα Ορίου Διαλυτότητας

Το παρακάτω διάγραμμα Ορίου Διαλυτότητας απεικονίζει τα όρια σχηματισμού κρυστάλλων πάγου ,για χαμηλές συγκεντρώσεις και πολύ χαμηλές θερμοκρασίες, ή στερεών κρυστάλλων άλατος για υψηλότερες τιμές θερμοκρασιών και συγκεντρώσεων.

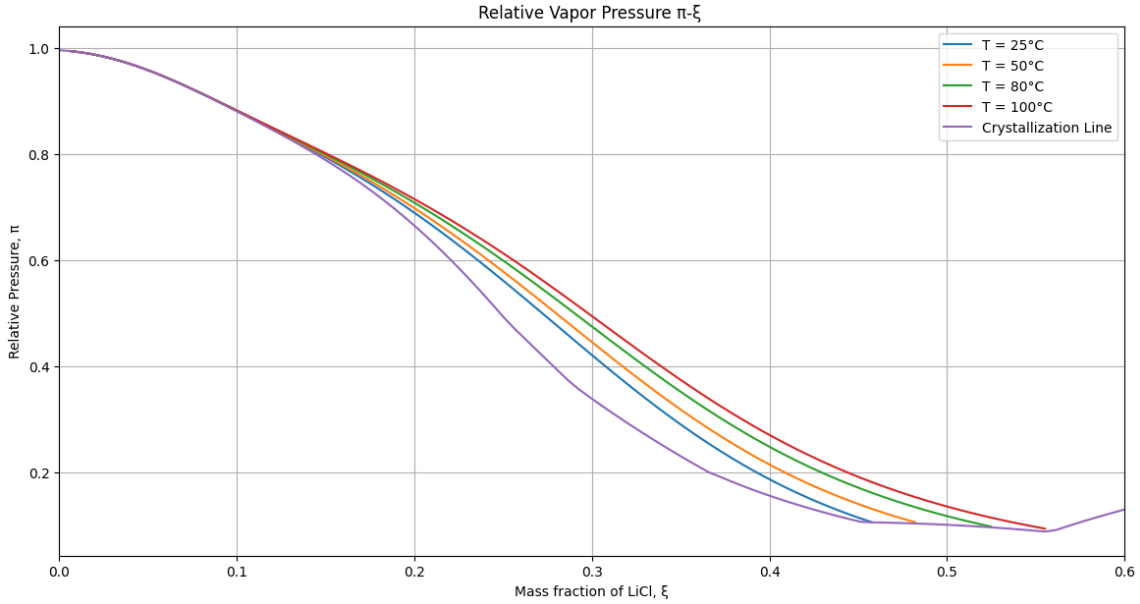


Διάγραμμα 1.Διάγραμμα Ορίου Διαλυτότητας

Διαγράμματα Τάσεων Ατμών και Διάγραμμα Ισορροπίας

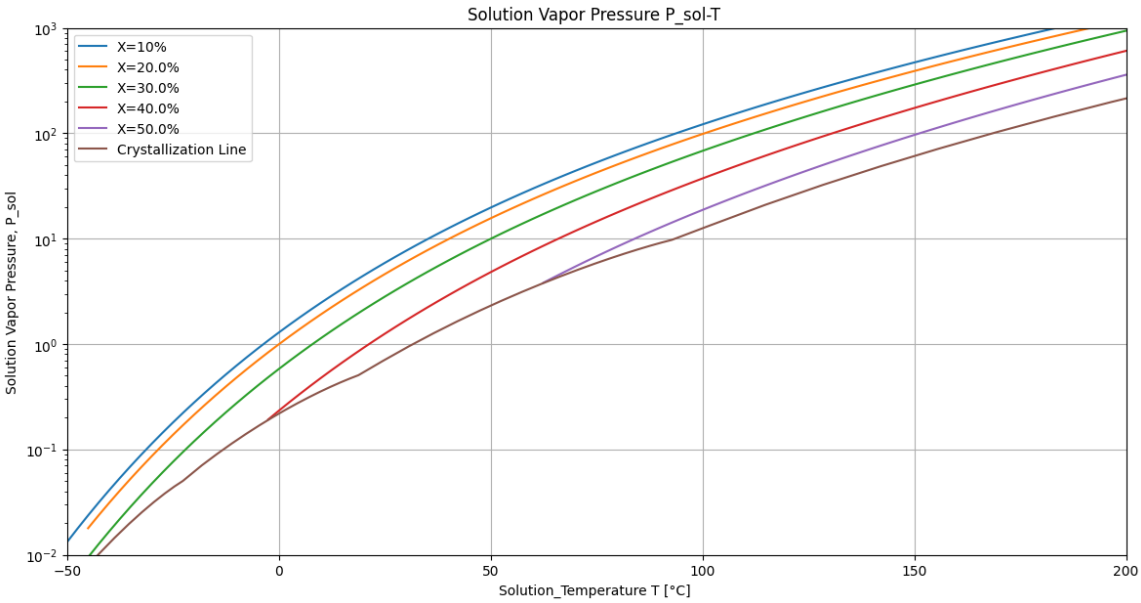
Η πίεση ισορροπίας του κορεσμένου ατμού σε υδατικά διαλύματα αποτελεί μία από τις πιο μελετημένες ιδιότητές τους. Παρά την πληθώρα πειραματικών δεδομένων δεν υπάρχει κάποια ακριβής περιγραφή της ιδιότητας για μεγάλο εύρος θερμοκρασιών και συγκεντρώσεων. Τα δύο πιο συνήθη διαγράμματα ,τα οποία αναπαριστούν την τάση ατμών για το Χλωριούχο Λίθιο είναι αυτά των Othmer και Dühring τα οποία απαιτούν

πρώτα τον υπολογισμό του σχετικού λόγου πίεσης υδρατμών διαλύματος –νερού για την ίδια θερμοκρασία.



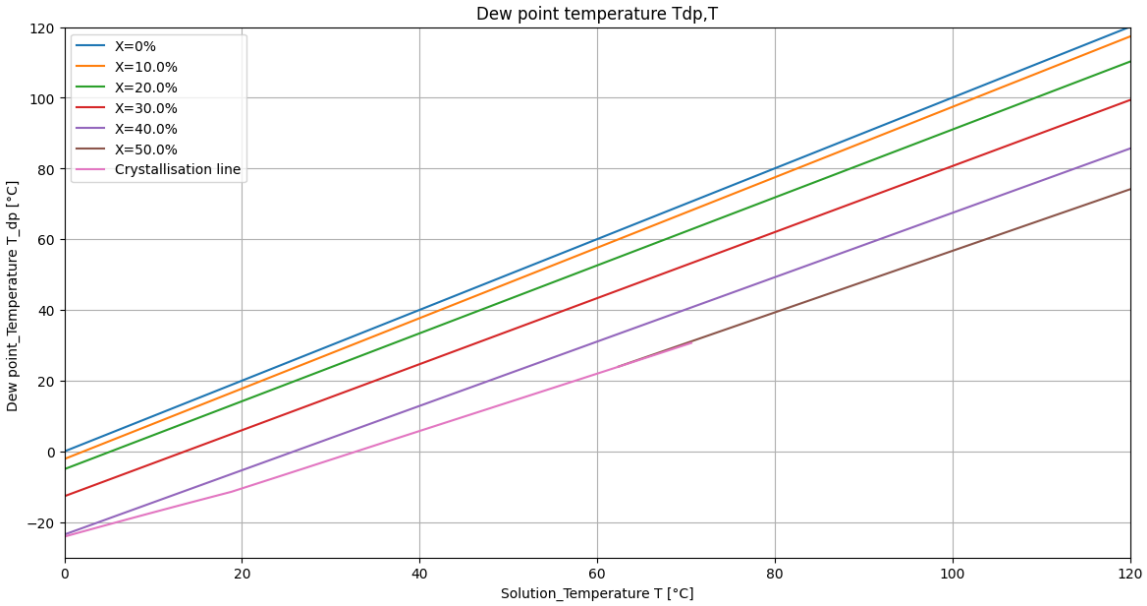
Διάγραμμα 2. Διάγραμμα Λόγου Πίεσης Διαλύματος-Νερού

Το διάγραμμα Othmer ουσιαστικά αναπαριστά την τάση υδρατμών σε ένα εύρος θερμοκρασιών αλλά για σταθερή συγκέντρωση.



Διάγραμμα 3. Διάγραμμα Othmer για το διάλυμα LiCl-H20

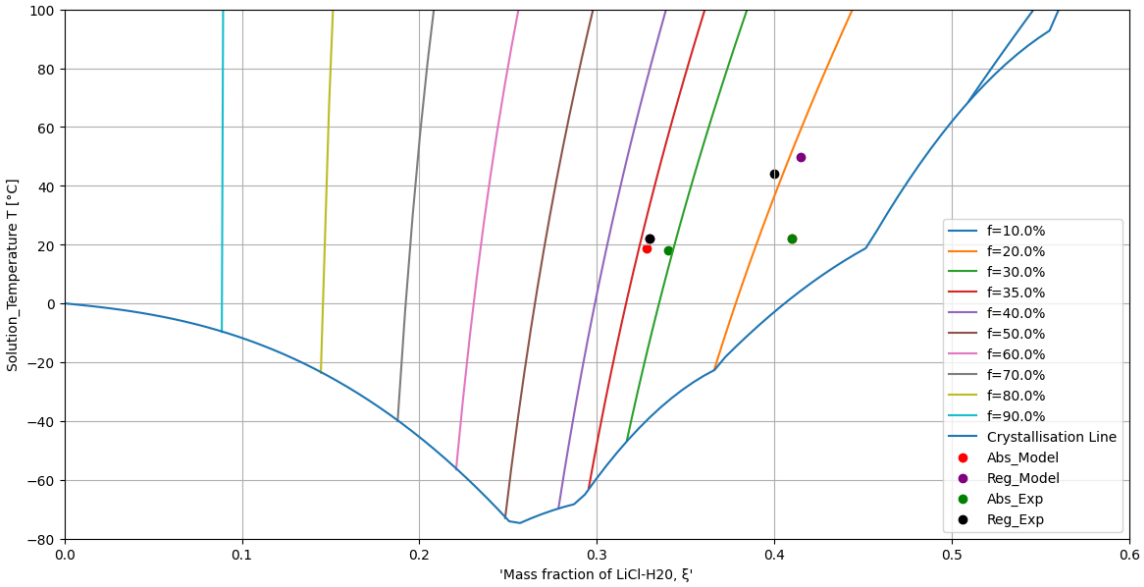
Το διάγραμμα Dühring αναπαριστά το σημείο δρόσου συγκριτικά με το σημείο όπου σχηματίζεται η πρώτη φυσαλίδα κατά την θέρμανση του διαλύματος. Το σημείο δρόσου είναι η θερμοκρασία στην οποία πρέπει να ψυχθεί ο αέρας ώστε να κορεστεί με υδρατμούς, υπό σταθερή πίεση και περιεκτικότητα υδρατμών. Αξίζει να αναφερθεί πως σε ένα υδατικό διάλυμα άλατος που βρίσκεται σε ισορροπία με υδρατμούς η θερμοκρασία κορεσμού εξαρτάται όχι μόνο από την πίεση των υδρατμών αλλά και από τη συγκέντρωση του άλατος. Η παρουσία άλατος στο διάλυμα εμποδίζει την εξάτμιση του νερού, γεγονός που αυξάνει τη θερμοκρασία κορεσμού σε σχέση με την θερμοκρασία των κορεσμένων υδρατμών. Όσο αυξάνεται η συγκέντρωση του άλατος, τόσο αυξάνεται και η θερμοκρασία κορεσμού. Το ακριβώς αντίθετο συμβαίνει κατά τη συμπύκνωση των υδρατμών όπου η θερμοκρασία συμπύκνωσης μειώνεται με την αύξηση της συγκέντρωσης. Ο νόμος του Dühring περιγράφει την γραμμική σύνδεση του σημείου κορεσμού ενός διαλύματος με το σημείο κορεσμού του διαλύτη υπό σταθερή πίεση. Παρόλαυτά δεν υπάρχει διατυπωμένη συνάρτηση που να περιγράφει την παραπάνω σχέση και το διάγραμμα προκύπτει από «μεθόδους παρεμβολής» σε πειραματικά δεδομένα.



Διάγραμμα 4. Διάγραμμα Dühring για το διάλυμα LiCl-H20

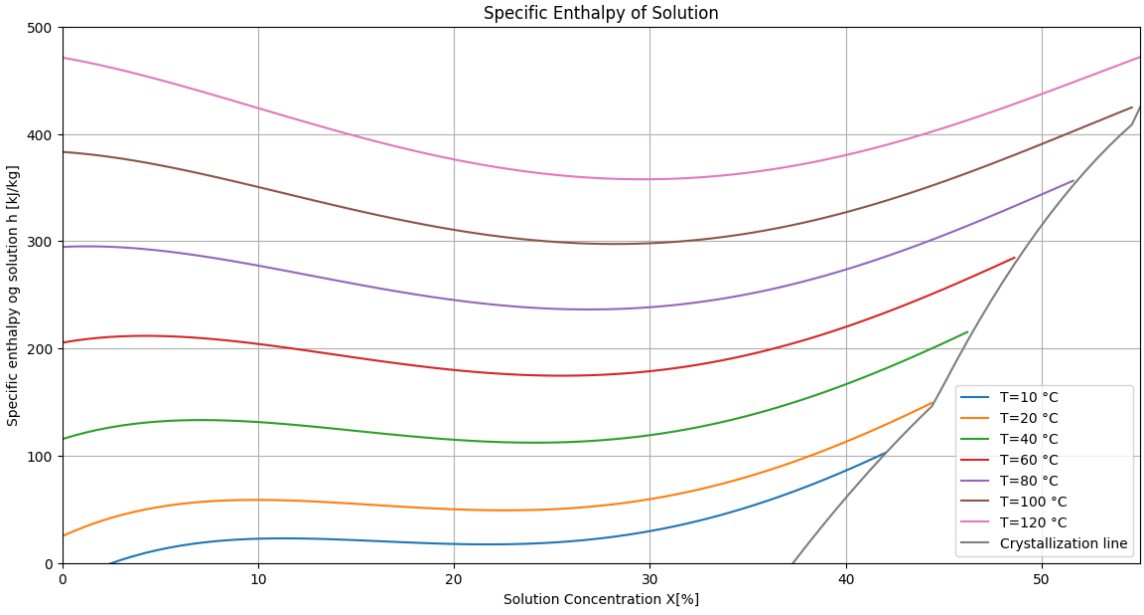
Το διάγραμμα ισορροπίας συνδέει τις γραμμές ισορροπίας της τάσης των ατμών του διαλύματος με την μερική πίεση των υδρατμών στον αέρα. Το παρακάτω διάγραμμα σχεδιάστηκε για σταθερή πίεση ίση με την ατμοσφαιρική. Επίσης περιλαμβάνονται και ο γραμμές σχετικής υγρασίας , που βοηθούν στην εύρεση και την οπτικοποίηση της απαιτούμενης αραίωσης του διαλύματος για δεδομένο ποσοστό αφύγρανσης , όταν

είναι γνωστή η απόδοση του αφυγραντικού συστήματος. Στο παρακάτω διάγραμμα φαίνονται τα πειραματικά δεδομένα καθώς και τα αποτελέσματα του μοντέλου.



Διάγραμμα 5. Διάγραμμα Ισορροπίας

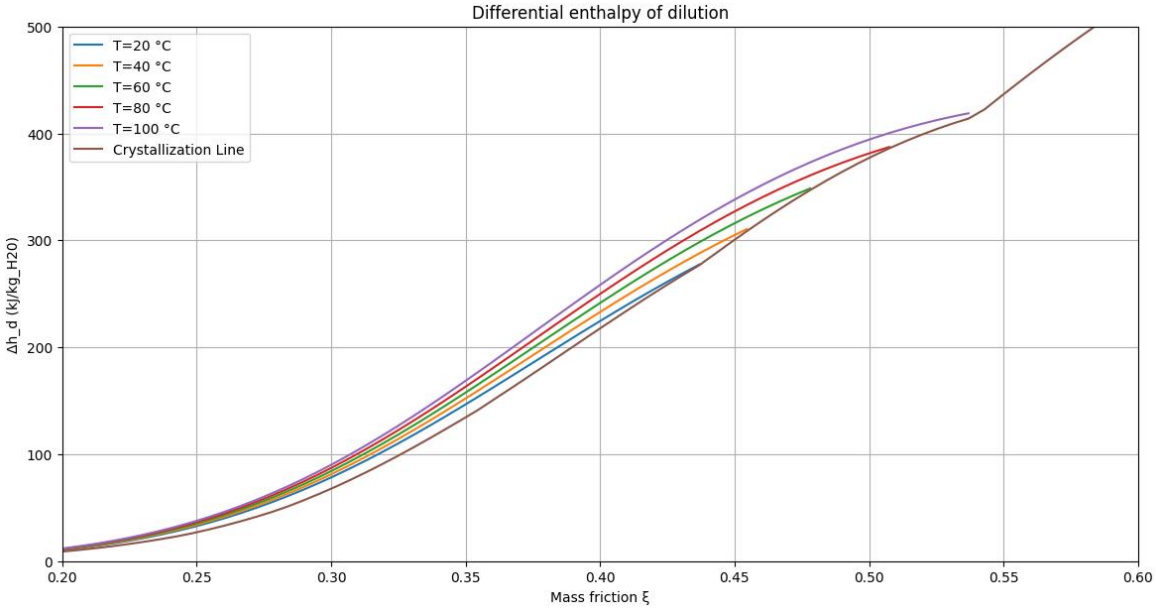
Ειδική Ενθαλπία Διαλύματος



Διάγραμμα 6.Ειδική Ενθαλπία Διαλύματος

Διάγραμμα Διαφοράς Ενθαλπιών

Η διαδικασία της απορρόφησης αποτελεί μία εξώθερμη αντίδραση ενώ η διαδικασία της αναγέννησης χρειάζεται πρόσδοση επιπλέον θερμότητας. Αυτή η ενέργεια θα είναι μεγαλύτερη συγκριτικά με την ενέργεια εξάτμισης ή συμπύκνωσης του νερού. Συγκεκριμένα κατά την απορρόφηση η ενέργεια που απελευθερώνεται είναι ανάλογη της συγκέντρωσης του διαλύματος και αντιστρόφως ανάλογη της θερμοκρασίας. Το παρακάτω διάγραμμα δείχνει αυτή τη διαφορά ,δηλαδή τη διαφορά μεταξύ της μερικής ενθαλπίας του νερού του διαλύματος και της ενθαλπίας του κορεσμένου νερού για την ίδια θερμοκρασία.



Διάγραμμα 7. Διάγραμμα Διαφοράς Ενθαλπιών

.

8.ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Koronaki, I., Christodoulaki, R., Papaefthimiou, V., & Rogdakis, E. (2013). Thermodynamic analysis of a counter flow adiabatic dehumidifier with different liquid desiccant materials. *Applied Thermal Engineering*, *50*(1), 361–373.

2. Koronaki, Christodoulaki (2015). Counter Flow Adiabatic Regenerator and comparative study.*International Journal of Thermodynamics*, *18*(3), 180.

3. Χριστοδουλάκη, Ρ. (2021). *Thermodynamic analysis of open cycle evaporative cooling systems with liquid desiccant mediums* [PhD].

4. Διπλωματική Εργασία, Μεριγκλέν Μπεντίνι, Εθνικό Μετσόβιο Πολυτεχνείο,(2020). Αποκατάσταση της λειτουργίας και πειραματική διερεύνηση Liquid desiccant mediums συστήματος

5. Εμμανουήλ Ρογδάκης , Νεόφυτος Κομνηνός,(2020),Εκδόσεις Τζιόλα,ISBN 878-960-418-855-0. Εφαρμοσμένη Θερμοδυναμική Μιγμάτων.pp.187-192,pp.214-222

6. Κραμβίδης, Σ., Kramvidis, S., & Εθνικό Μετσόβιο Πολυτεχνείο. (2016). *Σχεδιασμός και ανάπτυξη ολοκληρωμένου συστήματος εποπτείας και ελέγχου (SCADA) πειραματικής εγκατάστασης ηλιακής ψύξης*

7. Διπλωματική Εργασία, Αραβή, Χ. Γ., & Aravi, C. G. (2014). Πειραματική μελέτη συστήματος υγρού ξηραντικού για αφύγρανση και ψύξη στην Αθήνα.

8. Conde, M. R. (2004b). Properties of aqueous solutions of lithium and calcium chlorides: formulations for use in air conditioning equipment design. *International Journal of Thermal Sciences*, *43*(4), 367–382.

9. Ghatos, S., Janan, M. T., & Mehdari, A. (2021). Thermodynamic model of a single stage H2O-LiBr absorption cooling. *E3S Web of Conferences*, *234*, 00091.

10. Lab view examples and guides (in application)
11. Λογισμικό Θερμοδυναμικής-Παραδείγματα (μάθημα 7ου εξαμήνου)
12. Scx15an, Dcxl10ds, Dsg, Tpmc131, pt100, th100 sensor datasheets
13. https://www.icpdas.com/web/product/download/io_and_unit/rs-485/document/data_sheet/I-7018(R)_M-7018(R)_en.pdf
14. https://www.icpdas.com/web/product/download/io_and_unit/rs-485/document/data_sheet/I-7019R_M-7019R_en.pdf
15. https://www.icpdas.com/web/product/download/io_and_unit/rs-485/document/manual/7000/I-7017_I-7018_I-7019_M-7017_M-7018_M-7019_en.pdf
16. http://www.coolprop.org/coolprop/examples.html

9.ΠΑΡΑΡΤΗΜΑ

9.1 Μοντέλο Απορροφητή

```
import numpy as np
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.integrate import solve_bvp
import matplotlib.pyplot as plt
###############WATER PROPERTIES################
########density############
def water_density(T):
    T_K=T+273.15
    density_liquid_water = CP.PropsSI('D', 'T', T_K, 'Q', 0, 'Water')
    return density_liquid_water
water_density_vec=np.vectorize(water_density)
####################Thermal Conductivity##############################
def thermal_conductivity_water(T):
    L_water=CP.PropsSI('L', 'T', T+273.17, 'Q', 0, 'water')
    return L_water
thermal_conductivity_water_vec=np.vectorize(thermal_conductivity_water)
#######dynamic viscosity############################
def dynamic_viscosity_water(T):
    T_Kelvin = T + 273.15
    dynamic_viscosity_liquid_water = CP.PropsSI('V', 'T', T_Kelvin, 'Q', 0, 'Water')
    return dynamic_viscosity_liquid_water
dynamic_viscosity_water_vec=np.vectorize(dynamic_viscosity_water)
####################Specific Thermal Capacity##############################
def cp_h20(T):
    cp_liquid_water = CP.PropsSI('C', 'T', T+273.15, 'Q', 0, 'Water')
    return cp_liquid_water
cp_h20_vec=np.vectorize(cp_h20)
#############cp of saturated steam##########
def cp_steam_saturated(T):
    CP_steam=CP.PropsSI('D', 'T', T+273.15, 'Q', 1, 'Water')
    return CP_steam
cp_steam_saturated_vec=np.vectorize(cp_steam_saturated)
############SOLUTION PROPERTIES#####################################
########DENSITY#############
def density(J,T):
    return water_density_vec(T)*(1+0.540966*(J/(1-J))-0.303792*(J/(1-J))**2+0.100791*(J/(1-J))**3)
density_vec=np.vectorize(density)
################THERMAL CONDUCTIVITY####################
def L_sol(J,T):
```

```
    a_R=10.8958*10**(-3)-11.7882*10**(-3)*J
    z_eq=J*density_vec(J,T)/42.39
    return (thermal_conductivity_water_vec(T)-z_eq*a_R)
L_sol_vec=np.vectorize(L_sol)
##########DYNAMIC VISCOSITY#####################
def dynamic_viscosity(J,T):
    theta=(T+273)/Tc_H2O
    zeta=J/((1-J)**(1/0.6))
    return
(dynamic_viscosity_water_vec(T)*np.exp(0.090481*zeta**3.6+1.390262*zeta+0.675875*zeta/theta-
0.583517*zeta**2))
dynamic_viscosity_vec=np.vectorize(dynamic_viscosity)
####################THERMAL CAPACITY#################################
def cp_sol(J,T):
    theta=(T+273)/228-1
    f_2=58.5225*theta**(0.02)-105.6343*theta**(0.04)+47.7948*theta**(0.06)
    if J<=0.31:
        f_1=1.43980*J-1.24317*J**2-0.12070*J**3
    else:
        f_1=0.12825+0.62934*J
    return (cp_h20_vec(T)*(1-f_1*f_2))
cp_sol_vec=np.vectorize(cp_sol)
#############Diffusion Coefficient###############

def dd0(J):

    return 1-(1+(np.sqrt(J)/0.52)**(-4.92))**(-0.56)
dd0_vec=np.vectorize(dd0)

#######differential enthalpy of absorption ################

def dh_abs(J,T):

    steam = IAPWS97(x=1,T=T+273.16)
    enthalpy_vapor = steam.h
    temp=T+273.15
    h_w=0.22156863+4.19690058*temp-4.8993808e-4*temp**2+4.00756794e-6*temp**3
    theta=(T+273)/Tc_H2O
    dh_d_0=169.105+457.850*theta
    zeta=J/(0.6000-J)
    dh_d=dh_d_0*(1+zeta**(-1.965))**(-2.265)
    h_ws=h_w-dh_d
    return (enthalpy_vapor-h_ws)*1000
dh_abs_vec=np.vectorize(dh_abs)
```

```
##############################AIR PROPERTIES##########################
########SATURATION ABSOLUTE HUMIDITY#####################
def W_sat_s(T):
    temperature_K=T+273.15
    saturation_pressure = CP.PropsSI('P', 'T', temperature_K, 'Q', 1, 'Water')
    return 0.62197*saturation_pressure/(101325-saturation_pressure)
W_sat_s_vec=np.vectorize(W_sat_s)
############thermal conductivty of air###################################
def kappa_a(temperature):

    k_air = CP.PropsSI('L', 'T', temperature+273.15, 'P', pressure, 'Air')  # W/m·K
    return k_air
kappa_a_vec=np.vectorize(kappa_a)
########dynamic viscosity of air##################
def visc_a(temperature):
    visc_air = CP.PropsSI('V', 'T', temperature+273.15, 'P', pressure, 'Air')   # Pa·s
    return  visc_air
visc_a_vec=np.vectorize(visc_a)
########air density#########
def density_air(temperature):

    density_a = CP.PropsSI('D', 'T', temperature+273.15, 'P', pressure, 'Air')             # kg/m³
    return density_a
density_air_vec=np.vectorize(density_air)
#######air heat capacity################
def cp_air(temperature):
    specific_heat_capacity_air = CP.PropsSI('C', 'T', temperature+273.15, 'P', pressure, 'Air')  #
J/kg·K
    return specific_heat_capacity_air
cp_air_vec=np.vectorize(cp_air)
#############cp of moist air###############
def cp_m_air(T,W):
    cpma = CP.HAPropsSI('C', 'T', T+273.15, 'W', W, 'P', pressure)
    return cpma
cp_m_air_vec=np.vectorize(cp_m_air)
def coefficients(T_a,T_s,T_w,W,m_s,X):

    ####water#########

    Nu_w=7.54#####Nusselt number

    cp_water=cp_h20_vec(T_w)

    k_w=thermal_conductivity_water_vec(T_w)#####thermal conductivity######
```

```
a_w=Nu_w*k_w/Dh_in  #########Water side heat transfer coefficient

###############solution###############

mi_s=dynamic_viscosity_vec(X,T_s)######dynamic viscosity

Re_s=4*m_s/(L*mi_s)#####Reynolds for solution

cp_s=cp_sol_vec(X,T_s)####solution thermal capacity

k_s=L_sol_vec(X,T_s)####solution thermal conductivity

ro_s=density_vec(X,T_s)######solution density

v_s=mi_s/ro_s###solution kinematic viscosity

delta_s=(3*v_s**2/9.81)**(1/3)*Re_s**(1/3)###film thickness
#H=0.0008########unknown parameter
#Pr_s=cp_s*mi_s*delta_s/(k_s*H)#########solution prandlt number
Pr_s=cp_s*mi_s/(k_s)
Nu_s=0.680*Re_s**0.5*Pr_s**(1/3)#########solution nusselt number

a_s=Nu_s*k_s/delta_s##############SOLUTION HEAT TRANSFER COEFFICIENT

dh_vap=dh_abs_vec(X,T_s)#######latent heat of vaporization

####################air##################

k_a=kappa_a_vec(T_a)###thermal conductivity of air

mi_a=visc_a_vec(T_a)##########dynamic viscosity of air

cp_a=cp_air_vec(T_a)### specific heat capacity of air

ro_a=density_air_vec(T_a)#######density of air

v_a=mi_a/ro_a######kinematic viscosity of air

V_a=m_a_in/area###cross sectional velocity ####NEEDS ADJUSTMENTS######

Re_a=V_a*Dh_out/(ro_a*v_a)######reynolds number of air

Pr_a=cp_a*mi_a/k_a#######Prandlt number of air

Nu_a=0.023*Re_a**0.8*Pr_a**(1/3)#####nusselt number of air
```

```
    a_a=Nu_a*k_a/Dh_out#####heat trnasfer coefficient of air#######

    cp_m_a=cp_m_air_vec(T_a,W)#######heat capacity of moist air#####

    cp_st_sat=cp_steam_saturated_vec(T_a)#####heat capacity of saturated steam

    W_sat=W_sat_s_vec(T_a)

    #######Lewis number ###############

    relative_d= 1-(1+(np.sqrt(X)/0.52)**(-4.92))**(-0.56)##############relative diffusivity coef of
solution

    d_w=2.3e-9##########self diffusivity of water

    d_s=relative_d*d_w#########diffusivity coef of solution

    #Le=(k_a/(ro_a*cp_a*d_s))######Lewis number
    Le=1

    ###################overall coefficients

    KG=a_a/(cp_m_a*Le)###mass transfer coefficient

    U_air=(1/a_w+d/k_p+1/a_s)**(-1)#######overall heat transfer coef



    coefficients_dict = {
        'Nusselt_water': Nu_w,
        'k_water': k_w,
        'cp_water':cp_water,
        'a_water': a_w,
        'mi_sol': mi_s,
        'Re_sol': Re_s,
        'cp_sol': cp_s,
        'k_sol': k_s,
        'ro_sol': ro_s,
        'v_sol': v_s,
        'delta_sol': delta_s,
        'Pr_sol': Pr_s,
        'Nu_sol': Nu_s,
        'a_sol': a_s,
        'k_air': k_a,
```

```
        'mi_air': mi_a,
        'cp_air': cp_a,
        'ro_air': ro_a,
        'v_air': v_a,
        'V_air': V_a,
        'Re_air': Re_a,
        'Pr_air': Pr_a,
        'Nu_air': Nu_a,
        'a_air': a_a,
        'cp_m_air': cp_m_a,
        'relative_d': relative_d,
        'd_water': d_w,
        'd_solution': d_s,
        'Le_number': Le,
        'KG': KG,
        'U_overall': U_air,
        'Latent_heat':dh_vap,
        'cp_st_sat':cp_st_sat,
        'W_sat':W_sat}
    return coefficients_dict


def boundary_conditions(ya, yb):
    return np.array([
        ya[0] - T_a_in,      # T_a(0) = T_a_in
        ya[3] - W_in,        # W(0) = W_in
        yb[1] - T_s_in,      # T_s(L) = T_s_in
        yb[2] - T_w_in,      # T_w(L) = T_w_in
        yb[4] - m_s_in,         # m_s(L) = m_s
        yb[5] - X_s       # X(L) = X_s
    ])
T_s_in=22##solution inlet temp
m_s_in=0.013#solution flow
X_s=0.41## concentration
T_a_in=30###air inlet temp
W_in=0.008
m_a_in=0.5###air flow
pressure=101325
T_w_in=15
m_w=0.25
Tc_H2O=647.096
Dh_in=0.016
L=1
area=0.219####
Dh_out=0.02
```

```
d=0.00035
k_p=0.19
def system_of_equations(A,y):


    T_a, T_s, T_w, W, m_s, X= y

    # Compute coefficients
    coeff = coefficients(T_a, T_s, T_w, W, m_s, X)
    # Extract coefficients
    cp_m_a = coeff['cp_m_air']

    a_air = coeff['a_air']

    cpst_sat = coeff['cp_st_sat']

    KG = coeff['KG']

    W_sat = coeff['W_sat']

    cp_s = coeff['cp_sol']

    dh_evap = coeff['Latent_heat']

    U_air = coeff['U_overall']

    cp_w = coeff['cp_water']

    # Define the system of ODEs
    dT_a_dA = -(T_a - T_s) / cp_m_a * ((a_air / m_a_in) -cpst_sat * (-KG / m_a_in * (W_sat -
W_in)))
    dT_s_dA = -1 / (m_s * cp_s) * (m_a_in * (-KG / m_a_in * (W_sat - W_in))* (cp_s * (T_s -
T_s_in) + cpst_sat * T_s + dh_evap) +a_air * (T_s - T_a)+U_air * (T_w - T_s))
    dT_w_dA = U_air * (T_s - T_w) / (m_w * cp_w)
    dW_dA = -KG / m_a_in * (W_sat - W_in)
    dm_s_dA = m_a_in * (-KG / m_a_in * (W_sat - W_in))
    dX_dA = -m_a_in / m_s * X * (-KG / m_a_in * (W_sat - W_in))
    return np.vstack([dT_a_dA, dT_s_dA, dT_w_dA, dW_dA, dm_s_dA, dX_dA])

def initial_guess(A):

    return np.array([
        np.full_like(A, T_a_in),
        np.full_like(A, T_s_in),
        np.full_like(A, T_w_in),
```

```
        np.full_like(A, W_in),
        np.full_like(A, m_s_in),
        np.full_like(A, X_s)
    ])
# Integration bounds

A_span = np.linspace(0, 15,15)

# Solve

solution = solve_bvp(system_of_equations, boundary_conditions, A_span, initial_guess(A_span))

# Access the solution
A_values = solution.x
state_values = solution.y
print("State values:")
print("Ta=",state_values[0,-
1],"T_s=",{state_values[1,0]},"T_w=",{state_values[2,0]},"W=",{state_values[3,-
1]},"m_s=",{state_values[4,0]},"X_w=",{state_values[5,0]})
```

9.2 Μοντέλο Αναγεννητή

```
import numpy as np
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.integrate import solve_bvp
import matplotlib.pyplot as plt
###############WATER PROPERTIES################
########density############
def water_density(T):
    T_K=T+273.15
    density_liquid_water = CP.PropsSI('D', 'T', T_K, 'Q', 0, 'Water')
    return density_liquid_water
water_density_vec=np.vectorize(water_density)
####################Thermal Conductivity##############################

def thermal_conductivity_water(T):

    L_water=CP.PropsSI('L', 'T', T+273.17, 'Q', 0, 'water')
    return L_water
thermal_conductivity_water_vec=np.vectorize(thermal_conductivity_water)
#######dynamic viscosity############################
def dynamic_viscosity_water(T):
    T_Kelvin = T + 273.15
    dynamic_viscosity_liquid_water = CP.PropsSI('V', 'T', T_Kelvin, 'Q', 0, 'Water')
```

```
    return dynamic_viscosity_liquid_water
dynamic_viscosity_water_vec=np.vectorize(dynamic_viscosity_water)
####################Specific Thermal Capacity##############################
def cp_h20(T):
    cp_liquid_water = CP.PropsSI('C', 'T', T+273.15, 'Q', 0, 'Water')
    return cp_liquid_water
cp_h20_vec=np.vectorize(cp_h20)
#############cp of saturated steam##########
def cp_steam_saturated(T):
    CP_steam=CP.PropsSI('D', 'T', T+273.15, 'Q', 1, 'Water')
    return CP_steam
cp_steam_saturated_vec=np.vectorize(cp_steam_saturated)
############SOLUTION PROPERTIES#################################
########DENSITY#############
def density(J,T):
    return water_density_vec(T)*(1+0.540966*(J/(1-J))-0.303792*(J/(1-J))**2+0.100791*(J/(1-J))**3)
density_vec=np.vectorize(density)
################THERMAL CONDUCTIVITY####################
def L_sol(J,T):
    a_R=10.8958*10**(-3)-11.7882*10**(-3)*J
    z_eq=J*density_vec(J,T)/42.39
    return (thermal_conductivity_water_vec(T)-z_eq*a_R)
L_sol_vec=np.vectorize(L_sol)
##########DYNAMIC VISCOSITY####################
def dynamic_viscosity(J,T):
    theta=(T+273)/Tc_H2O
    zeta=J/((1-J)**(1/0.6))
    return
(dynamic_viscosity_water_vec(T)*np.exp(0.090481*zeta**3.6+1.390262*zeta+0.675875*zeta/theta-
0.583517*zeta**2))
dynamic_viscosity_vec=np.vectorize(dynamic_viscosity)
####################THERMAL CAPACITY#################################
def cp_sol(J,T):
    theta=(T+273)/228-1
    f_2=58.5225*theta**(0.02)-105.6343*theta**(0.04)+47.7948*theta**(0.06)
    if J<=0.31:
        f_1=1.43980*J-1.24317*J**2-0.12070*J**3
    else:
        f_1=0.12825+0.62934*J
    return (cp_h20_vec(T)*(1-f_1*f_2))
cp_sol_vec=np.vectorize(cp_sol)
#############Diffusion Coefficient###############

def dd0(J):
```

```
    return 1-(1+(np.sqrt(J)/0.52)**(-4.92))**(-0.56)
dd0_vec=np.vectorize(dd0)
############differential enthalpy of absorption #########

def dh_abs(J,T):

    steam = IAPWS97(x=1,T=T+273.16)
    enthalpy_vapor = steam.h
    temp=T+273.15
    h_w=0.22156863+4.19690058*temp-4.8993808e-4*temp**2+4.00756794e-6*temp**3
    theta=(T+273)/Tc_H2O
    dh_d_0=169.105+457.850*theta
    zeta=J/(0.6000-J)
    dh_d=dh_d_0*(1+zeta**(-1.965))**(-2.265)
    h_ws=h_w-dh_d
    return (enthalpy_vapor-h_ws)*1000


dh_abs_vec=np.vectorize(dh_abs)

##############################AIR PROPERTIES##########################
########SATURATION ABSOLUTE HUMIDITY#######################
def W_sat_s(T):
    temperature_K=T+273.15
    saturation_pressure = CP.PropsSI('P', 'T', temperature_K, 'Q', 1, 'Water')
    return 0.62197*saturation_pressure/(101325-saturation_pressure)

W_sat_s_vec=np.vectorize(W_sat_s)
############thermal conductivty of air###############################################
def kappa_a(temperature):

    k_air = CP.PropsSI('L', 'T', temperature+273.15, 'P', pressure, 'Air')  # W/m·K
    return k_air

kappa_a_vec=np.vectorize(kappa_a)
########dynamic viscosity of air###################
def visc_a(temperature):
    visc_air = CP.PropsSI('V', 'T', temperature+273.15, 'P', pressure, 'Air')   # Pa·s
    return  visc_air
visc_a_vec=np.vectorize(visc_a)
#######air density#########
def density_air(temperature):

    density_a = CP.PropsSI('D', 'T', temperature+273.15, 'P', pressure, 'Air')             # kg/m³
    return density_a
```

```
density_air_vec=np.vectorize(density_air)

#######air heat capacity################
def cp_air(temperature):
    specific_heat_capacity_air = CP.PropsSI('C', 'T', temperature+273.15, 'P', pressure, 'Air')  #
J/kg·K
    return specific_heat_capacity_air
cp_air_vec=np.vectorize(cp_air)

#############cp of moist air###############
def cp_m_air(T,W):
    cpma = CP.HAPropsSI('C', 'T', T+273.15, 'W', W, 'P', pressure)
    return cpma
cp_m_air_vec=np.vectorize(cp_m_air)
def coefficients(T_a,T_s,T_w,W,m_s,X):

        ####water#########

    Nu_w=7.54#####Nusselt number

    cp_water=cp_h20_vec(T_w)

    k_w=thermal_conductivity_water_vec(T_w)#####thermal conductivity######

    a_w=Nu_w*k_w/Dh_in  #########Water side heat transfer coefficient

    ###############solution###############

    mi_s=dynamic_viscosity_vec(X,T_s)######dynamic viscosity

    Re_s=4*m_s/(L*mi_s)#####Reynolds for solution

    cp_s=cp_sol_vec(X,T_s)####solution thermal capacity

    k_s=L_sol_vec(X,T_s)####solution thermal conductivity

    ro_s=density_vec(X,T_s)######solution density

    v_s=mi_s/ro_s###solution kinematic viscosity

    delta_s=(3*v_s**2/9.81)**(1/3)*Re_s**(1/3)###film thickness
    #H=0.0008########unknown parameter
    #Pr_s=cp_s*mi_s*delta_s/(k_s*H)#########solution prandlt
    Pr_s=cp_s*mi_s/(k_s)
    Nu_s=0.680*Re_s**0.5*Pr_s**(1/3)#########solution nusselt number
```

```
    a_s=Nu_s*k_s/delta_s##############SOLUTION HEAT TRANSFER COEFFICIENT

    dh_vap=dh_abs_vec(X,T_s)#######latent heat of vaporization

    ####################air###################

    k_a=kappa_a_vec(T_a)###thermal conductivity of air

    mi_a=visc_a_vec(T_a)##########dynamic viscosity of air

    cp_a=cp_air_vec(T_a)### specific heat capacity of air

    ro_a=density_air_vec(T_a)#######density of air

    v_a=mi_a/ro_a######kinematic viscosity of air

    V_a=m_a_in/area###cross sectional velocity

    Re_a=V_a*Dh_out/(ro_a*v_a)######reynolds number of air

    Pr_a=cp_a*mi_a/k_a#######Prandlt number of air

    Nu_a=0.023*Re_a**0.8*Pr_a**(1/3)#####nusselt number of air

    a_a=Nu_a*k_a/Dh_out#####heat trnasfer coefficient of air#######

    cp_m_a=cp_m_air_vec(T_a,W)#######heat capacity of moist air#####

    cp_st_sat=cp_steam_saturated_vec(T_a)#####heat capacity of saturated steam

    W_sat=W_sat_s_vec(T_a)

    #######Lewis number ###############

    relative_d= 1-(1+(np.sqrt(X)/0.52)**(-4.92))**(-0.56)##############relative diffusivity coef of
solution

    d_w=2.3e-9##########self diffusivity of water

    d_s=relative_d*d_w#########diffusivity coef of solution

    #Le=(k_a/(ro_a*cp_a*d_s))######Lewis number
    Le=1
```

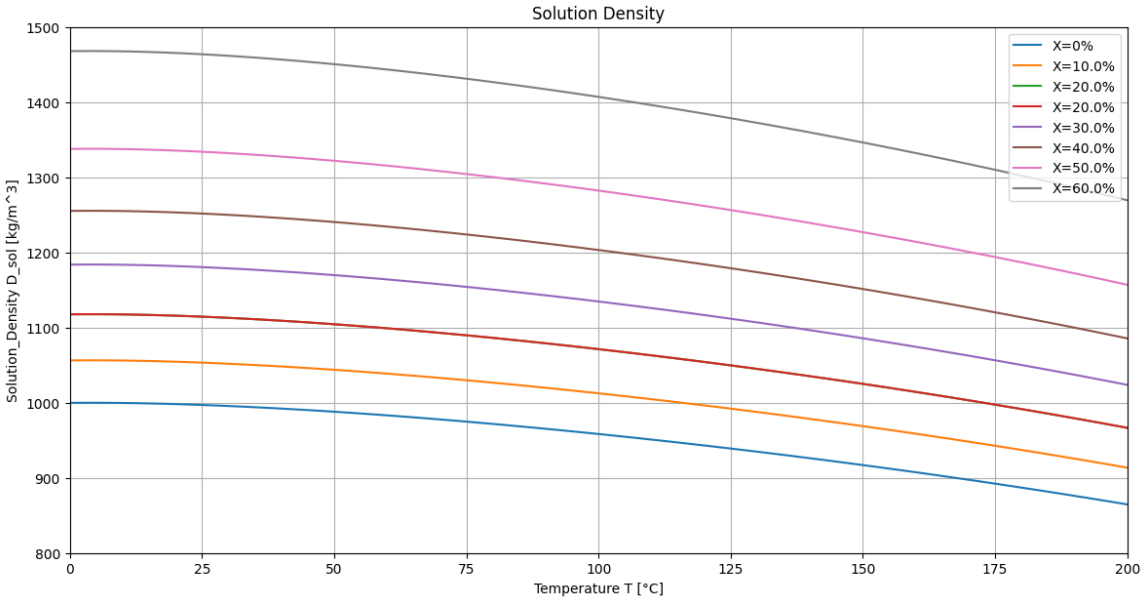
```
    ###################overall coefficients

    KG=a_a/(cp_m_a*Le)###mass transfer coefficient

    U_air=(1/a_w+d/k_p+1/a_s)**(-1)#######overall heat transfer coef



    coefficients_dict = {
        'Nusselt_water': Nu_w,
        'k_water': k_w,
        'cp_water':cp_water,
        'a_water': a_w,
        'mi_sol': mi_s,
        'Re_sol': Re_s,
        'cp_sol': cp_s,
        'k_sol': k_s,
        'ro_sol': ro_s,
        'v_sol': v_s,
        'delta_sol': delta_s,
        'Pr_sol': Pr_s,
        'Nu_sol': Nu_s,
        'a_sol': a_s,
        'k_air': k_a,
        'mi_air': mi_a,
        'cp_air': cp_a,
        'ro_air': ro_a,
        'v_air': v_a,
        'V_air': V_a,
        'Re_air': Re_a,
        'Pr_air': Pr_a,
        'Nu_air': Nu_a,
        'a_air': a_a,
        'cp_m_air': cp_m_a,
        'relative_d': relative_d,
        'd_water': d_w,
        'd_solution': d_s,
        'Le_number': Le,
        'KG': KG,
        'U_overall': U_air,
        'Latent_heat':dh_vap,
        'cp_st_sat':cp_st_sat,
        'W_sat':W_sat}
    return coefficients_dict
```

```
def boundary_conditions(ya, yb):
    return np.array([
        ya[0] - T_a_in,      # T_a(0) = T_a_in
        ya[3] - W_in,        # W(0) = W_in
        yb[1] - T_s_in,      # T_s(L) = T_s_in
        yb[2] - T_w_in,      # T_w(L) = T_w_in
        yb[4] - m_s_in,         # m_s(L) = m_s
        yb[5] - X_s       # X(L) = X_s
    ])

T_s_in=22##solution inlet temp
m_s_in=0.013#solution flow
X_s=0.34## concentration

T_a_in=30###air inlet temp
W_in=0.008
m_a_in=0.20###air flow
pressure=101325

T_w_in=55
m_w=0.15
Tc_H2O=647.096

Dh_in=0.016
L=1
area=0.219####
Dh_out=0.02
d=0.00035
k_p=0.19
def system_of_equations(A,y):
    # Unpack state variables
    T_a, T_s, T_w, W, m_s, X= y

    # Compute coefficients
    coeff = coefficients(T_a, T_s, T_w, W, m_s, X)
    # Extract coefficients
    cp_m_a = coeff['cp_m_air']

    a_air = coeff['a_air']

    cpst_sat = coeff['cp_st_sat']

    KG = coeff['KG']
```

```
    W_sat = coeff['W_sat']

    cp_s = coeff['cp_sol']

    dh_evap = coeff['Latent_heat']

    U_air = coeff['U_overall']

    cp_w = coeff['cp_water']

    # Define the system of ODEs
    dT_a_dA = -(T_a - T_s) / cp_m_a * ((a_air / m_a_in) -cpst_sat * (KG / m_a_in * (W_sat -
W_in)))
    dT_s_dA = 1 / (m_s * cp_s) * (m_a_in * (KG / m_a_in * (W_sat - W_in))* (cp_s * (T_s - T_s_in)
- cpst_sat * T_s + dh_evap) -a_air * (T_s - T_a)-U_air * (T_w - T_s))
    dT_w_dA = -U_air * (T_s - T_w) / (m_w * cp_w)
    dW_dA = KG / m_a_in * (W_sat - W_in)
    dm_s_dA = m_a_in * (KG / m_a_in * (W_sat - W_in))
    dX_dA = -m_a_in / m_s * X * (KG / m_a_in * (W_sat - W_in))
    return np.vstack([dT_a_dA, dT_s_dA, dT_w_dA, dW_dA, dm_s_dA, dX_dA])

def initial_guess(A):

    return np.array([
        np.full_like(A, T_a_in),
        np.full_like(A, T_s_in),
        np.full_like(A, T_w_in),
        np.full_like(A, W_in),
        np.full_like(A, m_s_in),
        np.full_like(A, X_s)
    ])
# Integration bounds
A_span = np.linspace(0, 15,15)
# Solve
solution = solve_bvp(system_of_equations, boundary_conditions, A_span, initial_guess(A_span))
# Access the solution
A_values = solution.x
state_values = solution.y
print("State values:")
print("Ta=",state_values[0,-
1],"T_s=",{state_values[1,0]},"T_w=",{state_values[2,0]},"W=",{state_values[3,-
1]},"m_s=",{state_values[4,0]},"X_w=",{state_values[5,0]})
```

9.3 Ιδιότητες Χλωριούχου Λιθίου

Διάγραμμα Πυκνότητας διαλύματος

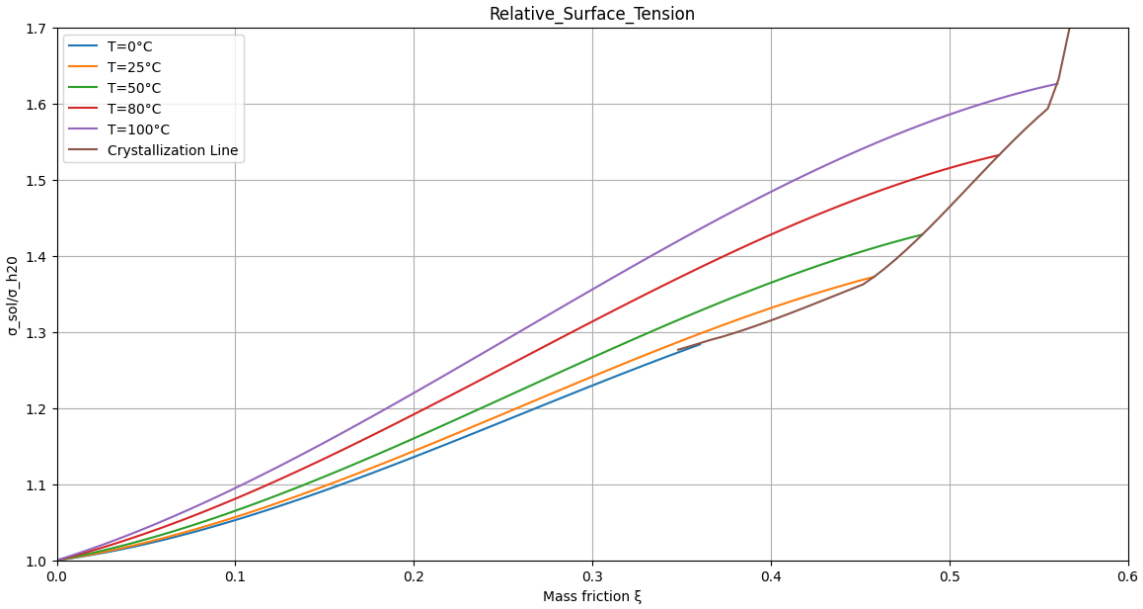


Διάγραμμα 8. Πυκνότητα Διαλύματος

```
import numpy as np
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
```

```
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
# Vectorize
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
#############Density#####################
def water_density(T):
    if T<0:
        t = 1-(T + 273.15)/Tc_H2O
        density=322*(1+1.993771843*t**(1/3)+1.0985211604*t**(2/3)-0.5094492996*t**(5/3)-
1.761912427*t**(16/3)-44.9005480267*t**(43/3)-723692.2618632*t**(110/3))
    else:
        T_K=T+273.15
        water=IAPWS97(T=T_K,x=0)
        density=water.rho
    return density
water_density_vec=np.vectorize(water_density)
def density(J,T):
    return water_density_vec(T)*(1+0.540966*(J/(1-J))-0.303792*(J/(1-J))**2+0.100791*(J/(1-J))**3)
T=np.linspace(0,200,101)
Conc=[0,0.1,0.2,0.2,0.3,0.4,0.5,0.6]
density_values={J:density(J,T) for J in Conc}

plt.figure(figsize=(14, 7))
for J in Conc:
    plt.plot(T,density_values[J],label=f"X={J*100}%")
plt.title("Solution Density")
plt.ylim((800,1500))
plt.xlim((0,200))
plt.grid(True)
plt.xlabel("Temperature T [°C]")
plt.ylabel('Solution_Density D_sol [kg/m^3]')
plt.legend()
plt.show
```

Διάγραμμμα Επιφανειακών τάσεων



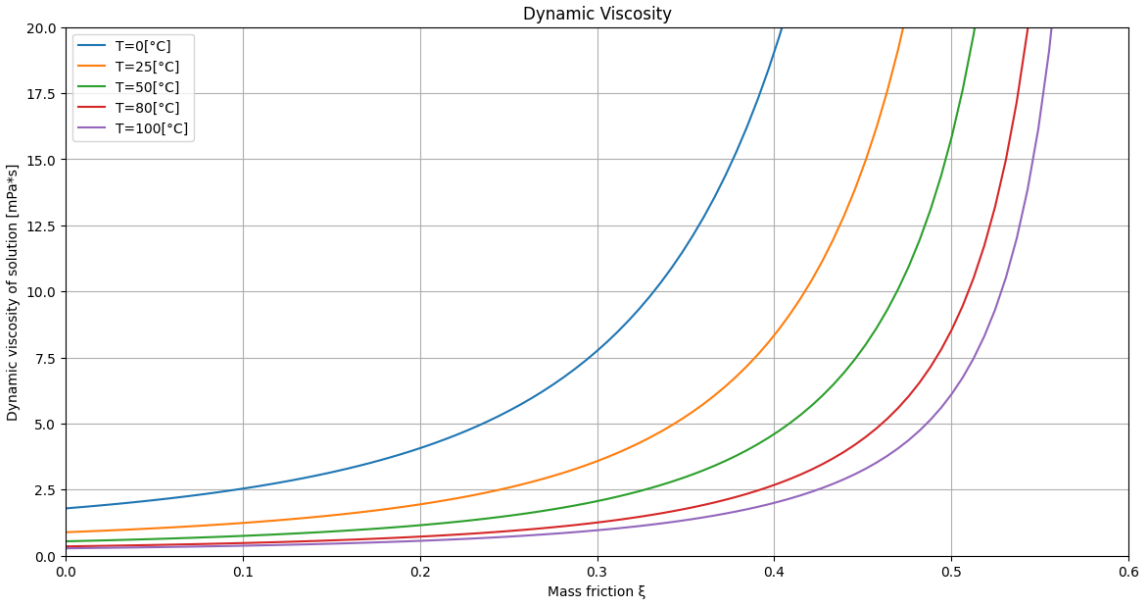
Διάγραμμα 9. Λόγος Επιφανειακών τάσεων

```
import numpy as np
from scipy.optimize import ridder
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
# Constants
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
###################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
```

```
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
# Vectorize
calculate_Th_vec = np.vectorize(calculate_Th)
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
##########surface tension#####################

def sigma_h20(T):

    theta=(T+273.15)/Tc_H2O
    return 235.8*(1+0.625*(1-theta))*(1-theta)**(1.256)
def sigma(J,T):
    theta=(T+273.15)/Tc_H2O
    return (1+2.757115*J-12.011299*J*theta+14.751818*J*theta**2+2.443204*J**2-3.147739*J**3)
def sigma_h20_T(T):
    return 235.8*(1-0.625*(1-T))*(1-T)**(1.256)
temperatures=[0,25,50,80,100]
J=np.linspace(0.001,0.61,101)
# Function to find root for given J
def find_J_for_constant_T(T):
    func = lambda J: sigma(J, T) - sigma(J,Tcrit(J))
    result =  ridder(func,0.3500, 0.5800)  # Adjust bracket as necessary
    return result
sigma_values={T:sigma(J,T) for T in temperatures}

plt.figure(figsize=(14, 7))

plt.plot(J[0:60],sigma_values[0][0:60],label=f"T={0}°C")

for T in temperatures:

    if T==0:
        continue
    reslt=find_J_for_constant_T(T)
    J_values=np.linspace(0.001,reslt,101)
    sigma_vals_above=sigma(J_values,T)
    plt.plot(J_values,sigma_vals_above,label=f"T={T}°C")
```

```
plt.plot(J[57:],sigma(J,Tcrit(J))[57:],label="Crystallization Line")
plt.title("Relative_Surface_Tension")
plt.xlabel("Mass friction ξ")
plt.ylabel("σ_sol/σ_h20")
plt.legend()
plt.xlim(0,0.6)
plt.ylim(1,1.7)
plt.grid(True)
plt.show()
```

Συντελεστής Ιξώδους Διαλύματος



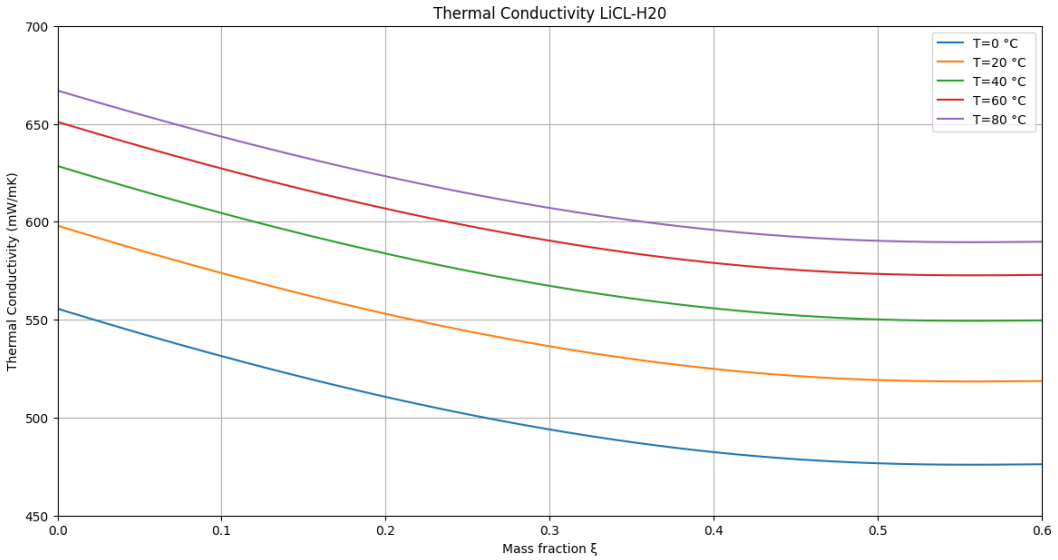
Διάγραμμα 10.Συντελεστής Ιξώδους Διαλύματος

```
import numpy as np
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
Tc_H2O = 647.096  # Critical temperature of water in Kelvin

Tc_H2O_C=374.096  # Critical temperature of water in Celsius

# Mass fraction array

J = np.linspace(0, 0.61, 101)
```

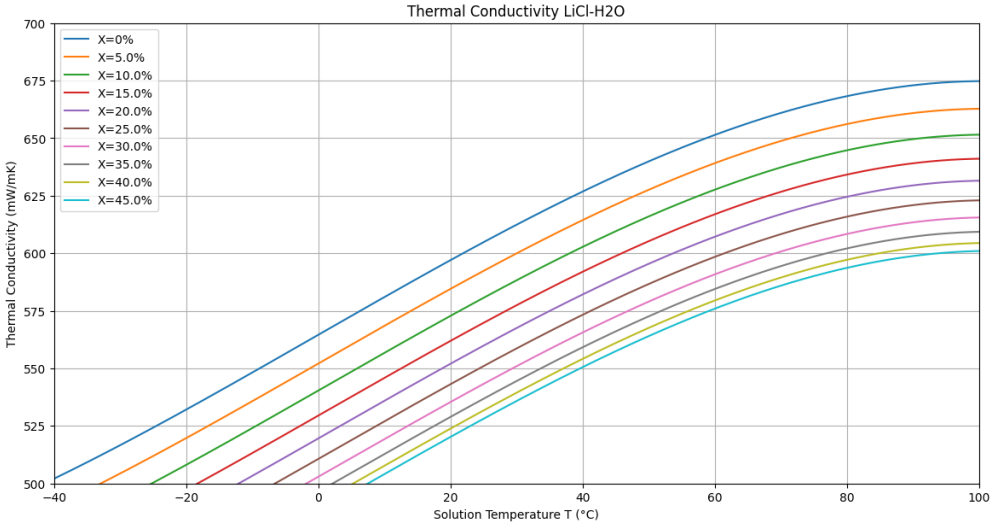
```
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
# Vectorize
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
########Dynamic Viscosity#########################

J=np.linspace(0.001,0.61,101)

def dynamic_viscosity_water(T):

    T_Kelvin = T + 273.15
    water = IAPWS97(T=T_Kelvin, x=0)
    mu = water.mu
    return mu*1000
def dynamic_viscosity(J,T):
    theta=(T+273)/Tc_H2O
    zeta=J/((1-J)**(1/0.6))
    return dynamic_viscosity_water(T)*np.exp(0.090481*zeta**3.6+1.390262*zeta+0.675875*zeta/theta-
0.583517*zeta**2)
temperatures=[0,25,50,80,100]
dynamic_viscosity_values={T:dynamic_viscosity(J,T) for T in temperatures}
plt.figure(figsize=(14, 7))
for T in temperatures:
```

```
    plt.plot(J,dynamic_viscosity_values[T],label=f"T={T}[°C]")
plt.title("Dynamic Viscosity")
plt.legend()
plt.ylim((0,20))
plt.xlim((0,0.6))
plt.xlabel("Mass friction ξ")
plt.ylabel("Dynamic viscosity of solution [mPa*s]")
plt.grid(True)
plt.show()
```

Συντελεστής Θερμικής Αγωγιμότητας



Διάγραμμα 11.Συντελεστής Θερμικής Αγωγιμότητας σε σχέση με την συγκέντρωση του διαλύματος



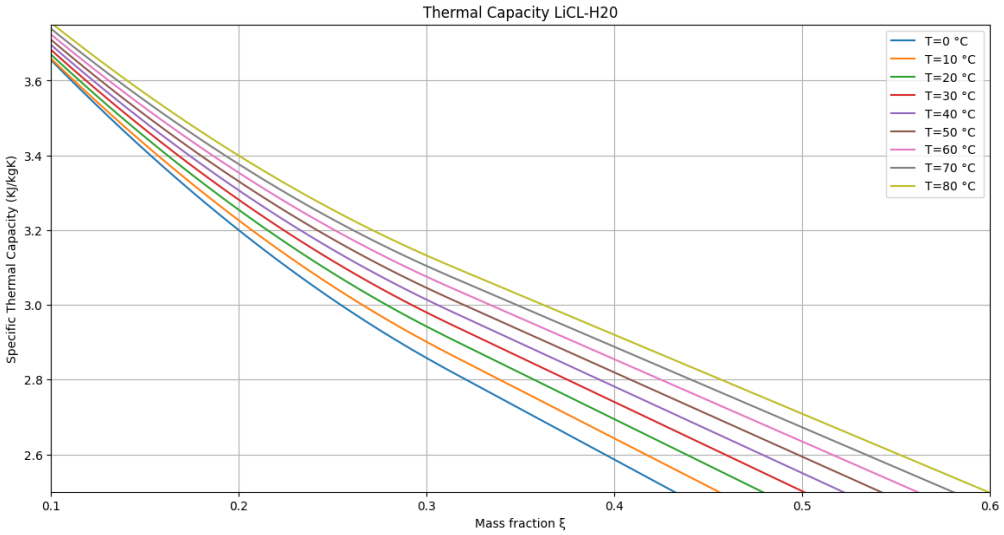
Διάγραμμα 12. Συντελεστής Θερμικής Αγωγιμότητας σε σχέση με την θερμοκρασία του διαλύματος

```
import numpy as np
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
# Vectorize
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
#############Density######################
def water_density(T):
    if T<0:
        t = 1-(T + 273.15)/Tc_H2O
        density=322*(1+1.993771843*t**(1/3)+1.0985211604*t**(2/3)-0.5094492996*t**(5/3)-
1.761912427*t**(16/3)-44.9005480267*t**(43/3)-723692.2618632*t**(110/3))
    else:
        T_K=T+273.15
        water=IAPWS97(T=T_K,x=0)
        density=water.rho
```

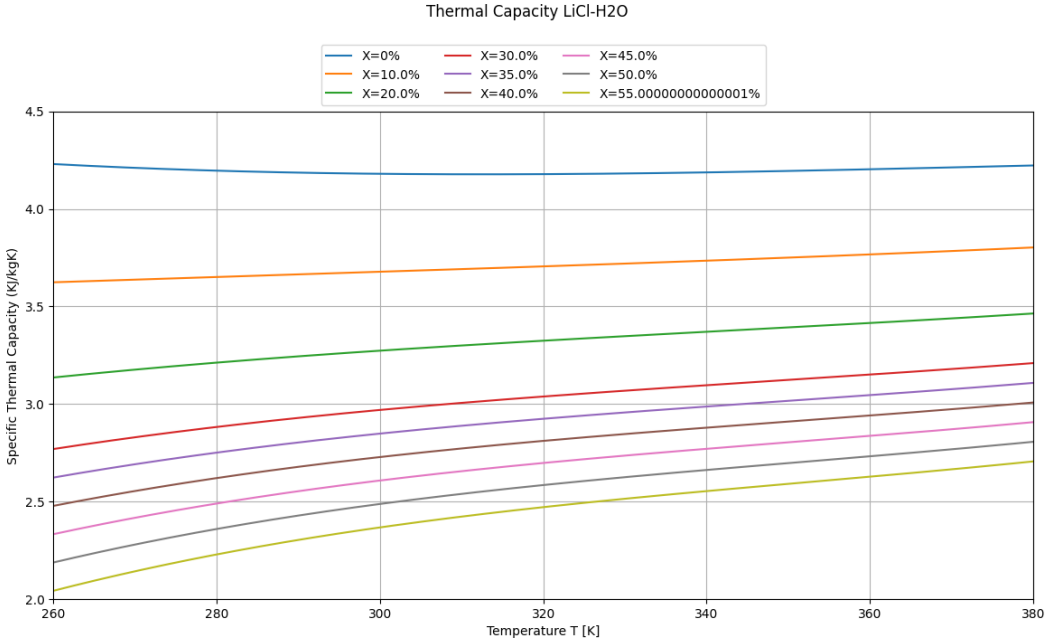
```
    return density
water_density_vec=np.vectorize(water_density)
def density(J,T):
    return water_density_vec(T)*(1+0.540966*(J/(1-J))-0.303792*(J/(1-J))**2+0.100791*(J/(1-J))**3)
####################Thermal Conductivity##############################
def thermal_conductivity_water(T):
    if T>=0:
        L_water=CP.PropsSI('L', 'T', T+273.17, 'Q', 0, 'water')
    else:
        L_water_20=CP.PropsSI('L','T',293.17,'Q',0,'water')
        L_water=L_water_20*(0.208495+1.747278*((T+273)/Tc_H2O))
    return L_water
thermal_conductivity_water_vec=np.vectorize(thermal_conductivity_water)
def L_sol(J,T):
    a_R=10.8958*10**(-3)-11.7882*10**(-3)*J
    z_eq=J*density(J,T)/42.39
    return 1000*(thermal_conductivity_water_vec(T)-z_eq*a_R)

temperatures = [0,20,40,60,80]

L_sol_values={T:L_sol(J,T) for T in temperatures}
plt.figure(figsize=(14, 7))
for T in temperatures:
    plt.plot(J,L_sol_values[T],label=f"T={T} °C ")

plt.xlabel('Mass fraction ξ')
plt.ylabel('Thermal Conductivity (mW/mK)')
plt.title('Thermal Conductivity LiCL-H20')
plt.legend()
plt.grid(True)
plt.ylim(450,700)
plt.xlim(0,0.6)
plt.show()
Conc=[0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45]
T=np.linspace(-40,100,101)
L_sol_values_2={J:L_sol(J,T) for J in Conc}
plt.figure(figsize=(14, 7))
for J in Conc:
    L_values = L_sol_values_2[J]
    spline = UnivariateSpline(T, L_values, s=1100)  #s for smoothing
    T_smooth = np.linspace(-40, 100, 600)
    L_smooth = spline(T_smooth)
    plt.plot(T_smooth, L_smooth, label=f"X={J*100}%")
plt.xlabel('Solution Temperature T (°C)')
plt.ylabel('Thermal Conductivity (mW/mK)')
```

```
plt.title('Thermal Conductivity LiCl-H2O')
plt.legend()
plt.grid(True)
plt.ylim(500, 700)
plt.xlim(-40, 100)
plt.show()
```

Ειδική Θερμοχωρητικότητα Διαλύματος



Διάγραμμα 13. Ειδική Θερμοχωρητικότητα Διαλύματος για σταθερή θερμοκρασία και μεταβλητή συγκέντρωση



Διάγραμμα 14. Ειδική Θερμοχωρητικότητα Διαλύματος για σταθερή συγκέντρωση και μεταβλητή θερμοκρασία

```
import numpy as np
```

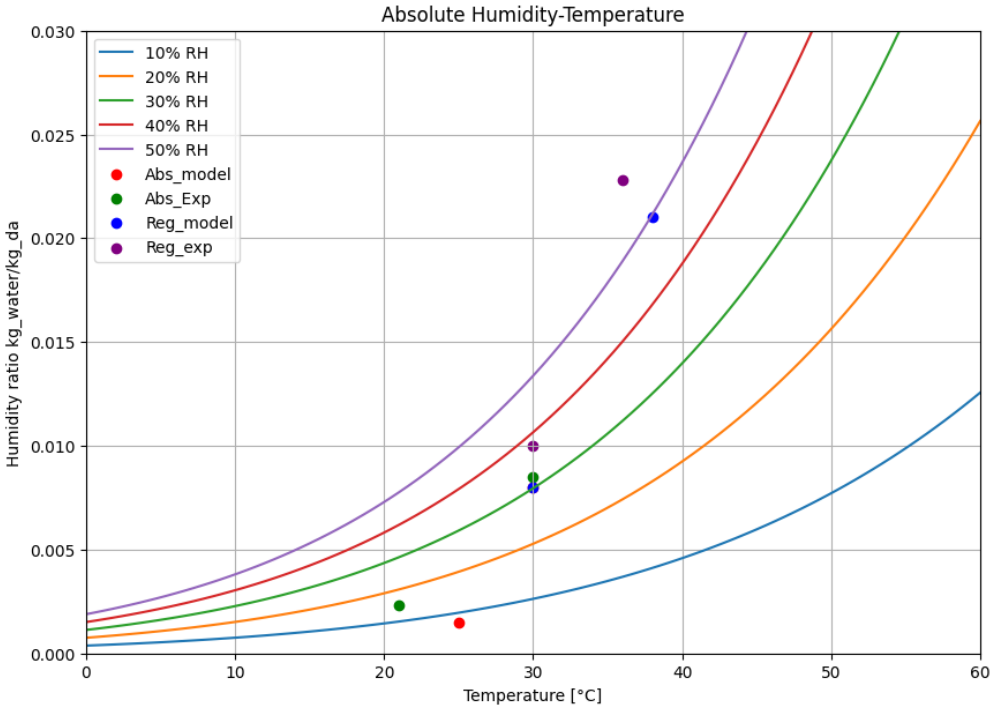
```
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline


Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
# Vectorize
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
####################Specific Thermal Capacity############################
def cp_h20(T):
    cp_water=IAPWS97(T=T+273.15,x=0).cp
    return cp_water
cp_h20_vec=np.vectorize(cp_h20)
def cp_sol(J,T):
    theta=(T+273)/228-1
    f_2=58.5225*theta**(0.02)-105.6343*theta**(0.04)+47.7948*theta**(0.06)
```

```
    if J<=0.31:
        f_1=1.43980*J-1.24317*J**2-0.12070*J**3
    else:
        f_1=0.12825+0.62934*J
    return cp_h20_vec(T)*(1-f_1*f_2)
cp_sol_vec=np.vectorize(cp_sol)
temperatures = [0,10,20,30,40,50,60,70,80]
J=np.linspace(0.001,0.61,101)
cp_sol_values={T:cp_sol_vec(J,T) for T in temperatures}
plt.figure(figsize=(14, 7))
for T in temperatures:
    plt.plot(J,cp_sol_values[T],label=f"T={T} °C ")
plt.xlabel('Mass fraction ξ')
plt.ylabel('Specific Thermal Capacity (KJ/kgK)')
plt.title('Thermal Capacity LiCL-H20')
plt.legend()
plt.ylim(2.50,3.75)
plt.xlim(0.1,0.6)
plt.grid(True)
plt.show()
#####THERMAL CAPACITY PLOT WITH TEMPERATURE BEING AT KELVIN##############
conc=[0,0.1,0.2,0.3,0.35,0.4,0.45,0.5,0.55]
T=np.linspace(260,380,101)
def cp_h20_2(T):
    if T<274:
        theta=T/228-1
        cp_water =830.54602-1247.52013*theta**0.02-68.60350*theta**0.04+491.27650*theta**0.06-
1.8692*theta**1.8-137.51511*theta**8
    elif T>=274:
        cp_water=IAPWS97(T=T,x=0).cp
    return cp_water
cp_h20_vec_2=np.vectorize(cp_h20_2)
def cp_sol_2(J,T):
    theta=T/228-1
    f_2=58.5225*theta**(0.02)-105.6343*theta**(0.04)+47.7948*theta**(0.06)
    if J<=0.31:
        f_1=1.43980*J-1.24317*J**2-0.12070*J**3
    else:
        f_1=0.12825+0.62934*J
    return cp_h20_vec_2(T)*(1-f_1*f_2)
cp_sol_vec_2=np.vectorize(cp_sol_2)
cp_sol_values_2={J:cp_sol_vec_2(J,T) for J in conc}

plt.figure(figsize=(14, 7))
for J in conc:
```

```
    values = cp_sol_values_2[J]
    spline = UnivariateSpline(T, values, s=110)
    T_smooth = np.linspace(260, 380, 600)
    cp_smooth = spline(T_smooth)
    plt.plot(T_smooth, cp_smooth, label=f"X={J*100}%")
plt.xlabel('Temperature T [K]')
plt.ylabel('Specific Thermal Capacity (KJ/kgK)')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15), ncol=3)
plt.suptitle('Thermal Capacity LiCl-H2O', y=1.05)
plt.xlim(260,380)
plt.ylim(2,4.5)
plt.grid(True)
plt.show()
```

Διάγραμμα Απόλυτης Υγρασίας-Θερμοκρασίας

Στο ψυχρομετρικό διάγραμμα φαίνονται τα πειραματικά και τα θεωρητικά αποτελέσματα. Στο διάγραμμα αυτό γίνεται εύκολα αντιληπτό το γεγονός πως το θεωρητικό μοντέλο υπερεκτιμά την ικανότητα απορρόφησης και αναγέννησης του συστήματος. Ταυτόχρονα βοηθάει στην κατανόηση των αιτιών αποκλίσεων υγρασίας που αναλύθηκαν στο κεφάλαιο 7.2.



Διάγραμμα 15.Διάγραμμα Απόλυτης Υγρασίας Αέρα και θερμοκρασίας

```
import numpy as np
from scipy.optimize import root_scalar
```

```
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
from CoolProp.HumidAirProp import HAPropsSI
# Constants
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##########################SOLUBILITY BOUNDARY#####################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan  # Handle out of range values if necessary
# Vectorize the function to apply it to arrays
calculate_Th_vec = np.vectorize(calculate_Th)
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
##################Psychrometric Chart#####################
# Function to calculate f_J_T based on J
def f_J_T(J,T):
    A = 2 - (1 + (J / 0.28)**4.3)**0.6
    B = (1 + (J / 0.21)**5.1)**0.49 - 1
    return A + B * ((T + 273.15) / Tc_H2O)
# Function to calculate P_25 based on J
def P_25(J):
    return 1 - (1 + (J / 0.362)**-4.75)**-0.4 - 0.03 * np.exp(-(J - 0.1)**2 / 0.005)
```

```
# Function to calculate P_rel based on J and T
def P_rel(J, T):
    return P_25(J) * f_J_T(J,T)
def vapor_pressure(T):
    T_kelvin = T + 273.15

    A = 8.07131
    B = 1730.63
    C = 233.426
    return 10**(A - B / (C + T)) * 133.322  # Converts mmHg to Pa
def P_sol(J,T):
    return P_rel(J,T)*vapor_pressure(T)*10**(-3)
################################Wsat dessicant#####################
def W(J,T):
    temp=T
    return 0.622*P_sol(J,temp)/(101.325-P_sol(J,temp))
W_vec=np.vectorize(W)
temperature_C = np.linspace(0, 60, 101)
temperature_K = temperature_C + 273.15
relative_humidity = [10,20,30,40,50]
def Absolute_humidity_air(T,Rh):
    AH = HAPropsSI('W', 'T', T+273.15, 'P', 101325, 'RH', Rh/100.0)
    return AH
plt.figure(figsize=(10, 7))
AH={Rh:Absolute_humidity_air(temperature_C,Rh) for Rh in relative_humidity}
for RH in relative_humidity:
    plt.plot(temperature_C, AH[RH], label=f'{RH}% RH')
Conc=[0.1,0.2,0.3,0.4,0.5]
W_values={J:W_vec(J,temperature_C) for J in Conc}
#for J in Conc:
    #plt.plot(T,W_values[J],label=f"{J*100}%LiCl")
##########MODEL#############
W_air_abs=[0.008,0.0015]
Temp_air_abs=[30,25]
Conc_sol_abs=[0.41,0.328]
Temp_sol_abs=[22,18.7]
W_air_reg=[0.008,0.021]
Temp_air_reg=[30,38]
Conc_sol_reg=[0.33,0.415]
Temp_sol_reg=[22,49.8]
########EXPERIMENT###########
W_air_abs_exp=[0.0085,0.0023]
Temp_air_abs_exp=[30,21]
Conc_sol_abs_exp=[0.41,0.34]
Temp_sol_abs_exp=[22,18]
```

```
W_air_reg_exp=[0.01,0.0228]
Temp_air_reg_exp=[30,36]
Conc_sol_reg_exp=[0.33,0.4]
Temp_sol_reg_exp=[22,44]
#Absorber
plt.scatter(Temp_air_abs,W_air_abs,color='red',label='Abs_model')#Absorber_model
plt.scatter(Temp_air_abs_exp,W_air_abs_exp,color='green',label="Abs_Exp")#Absorber_exp
##Regenerator
plt.scatter(Temp_air_reg,W_air_reg,color='blue',label="Reg_model")##Reg_model
plt.scatter(Temp_air_reg_exp,W_air_reg_exp,color='purple',label="Reg_exp")###Reg_exp
plt.xlabel("Temperature [°C]")
plt.ylabel("Humidity ratio kg_water/kg_da")
plt.title("Absolute Humidity-Temperature")
plt.xlim(0,60)
plt.ylim(0,0.03)
plt.legend()
plt.grid(True)
plt.show()
```

Ειδική Ενθαλπία Διαλύματος

```
import numpy as np
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
Tc_H2O = 647.096  # Critical temperature of water in Kelvin

Tc_H2O_C=374.096  # Critical temperature of water in Celsius

# Mass fraction array

J = np.linspace(0, 0.61, 101)#########

################SOLUBILITY BOUNDARY##############

# Function to calculate Th based on J

def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
```

```
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
# Vectorize
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
###############Specific enthalpy of solution#########################

def specific_h(J, T):
    A = (-66.2324 + 11.2711 * J - 0.79853 * J**2 + 2.1534e-2 * J**3 - 1.66352e-4 * J**4)
    B = (4.5751 - 0.146914 * J + 6.307226e-3 * J**2 - 1.38054e-4 * J**3 + 1.06690e-6 * J**4)
    C = (-8.09689e-4 + 2.18145e-4 * J - 1.36194e-5 * J**2 +3.20998e-7 * J**3 - 2.64266e-9 * J**4)

    h = A + B * T + C * T**2
    return h
temperatures=[10,20,40,60,80,100,120]
J=np.linspace(0,60,101)
specific_h_values={T:specific_h(J,T) for T in temperatures}
specific_h_crystal=specific_h(J,Tcr)
plt.figure(figsize=(14, 7))
for T in temperatures:
    specific_h_vals=specific_h_values[T]
    h_vals_above = [h if h > specific_h_crystal[idx] else np.nan for idx, h in
enumerate(specific_h_vals)]
    plt.plot(J,h_vals_above,label=f"T={T} °C ")
plt.plot(J,specific_h(J,Tcr),label=f"Crystallization line")
plt.title("Specific Enthalpy of Solution")
plt.xlabel(f"Solution Concentration X[%]")
plt.ylabel("Specific enthalpy og solution h [kJ/kg]")
plt.grid(True)
plt.ylim(0,500)
plt.xlim(0,55)
plt.legend()
plt.show()
```

Θερμοκρασία Κρυστάλλωσης

```
import numpy as np
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
# Constants
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
calculate_Th_vec = np.vectorize(calculate_Th)
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
plt.figure(figsize=(14, 7))
plt.plot(J, Tcr, '-k')
#annotations
annotations = [
    {"text": "ice", "xy": (0.07, -35)},
    {"text": "LiCl-5H2O", "xy": (0.25, -97),"fontsize": 9},
    {"text": "A", "xy": (0.253, -75.5)},
    {"text": "B", "xy": (0.287, -68.2)},
    {"text": "LiCl-3H2O", "xy": (0.32, -70),"fontsize": 9},
    {"text": "C", "xy": (0.369, -19.9)},
    {"text": "LiCl-2H2O", "xy": (0.4, -20),"fontsize": 9},
    {"text": "D", "xy": (0.452, 19.1)},
    {"text": "LiCl-H2O", "xy": (0.51, 40),"fontsize": 9},
```

```
    {"text": "E", "xy": (0.558, 93.6)},
]
for ann in annotations:
    fontsize=ann.get("fontsize", 10)
    plt.annotate(ann["text"], xy=ann["xy"], fontsize=fontsize,xytext=(-8,+10),textcoords='offset
points')
plt.title('Mass fraction of LiCl-ξ')
plt.xlabel('Mass fraction of LiCl, ξ')
plt.ylabel('Temperature, T (°C)')
plt.ylim((-100,200))
plt.xlim((0,0.6))
plt.grid(True)
plt.show()
```

Λόγος Πιέσεων

```
import numpy as np

from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan  # Handle out of range values if necessary
# Vectorize the function to apply it to arrays
calculate_Th_vec = np.vectorize(calculate_Th)
```

```
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
##############################RELATIVE PRESSURE##############################
J=np.linspace(0.0001,0.61,101)
temperatures_C=[25 ,50,80 ,100]
# Function to calculate f_J_T based on J
def f_J_T(J,T):
    A = 2 - (1 + (J / 0.28)**4.3)**0.6
    B = (1 + (J / 0.21)**5.1)**0.49 - 1
    return A + B * ((T + 273.15) / Tc_H2O)
# Function to calculate P_25 based on J
def P_25(J):
    return 1 - (1 + (J / 0.362)**-4.75)**-0.4 - 0.03 * np.exp(-(J - 0.1)**2 / 0.005)
# Function to calculate P_rel based on J and T
def P_rel(J, T):
    return P_25(J) * f_J_T(J,T)
P_crystallization_line=P_rel(J,Tcr)
P_rel_values={T:P_rel(J,T) for T in temperatures_C}
plt.figure(figsize=(14, 7))
for T in temperatures_C:
    P_vals=P_rel_values[T]
    P_vals_above = [P if P > P_crystallization_line[idx] else np.nan for idx, P in
enumerate(P_vals)]
    plt.plot(J,P_vals_above,label = f"T = {T}°C" )
plt.plot(J,P_rel(J,Tcr),label=f"Crystallization Line")
plt.legend()
# Set title and labels
plt.title('Relative Vapor Pressure π-ξ ')
plt.xlabel('Mass fraction of LiCl, ξ')
plt.ylabel('Relative Pressure, π')
plt.grid(True)
plt.xlim((0,0.6))
plt.show()
```

Πίεση Ατμών

```
import numpy as np
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline

Tc_H2O = 647.096  # Critical temperature of water in Kelvin
```

```
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan  # Handle out of range values if necessary
# Vectorize the function to apply it to arrays
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
##############################RELATIVE PRESSURE##############################
J=np.linspace(0.0001,0.61,101)
temperatures_C=[25 ,50,80 ,100]
# Function to calculate f_J_T based on J
def f_J_T(J,T):
    A = 2 - (1 + (J / 0.28)**4.3)**0.6
    B = (1 + (J / 0.21)**5.1)**0.49 - 1
    return A + B * ((T + 273.15) / Tc_H2O)
# Function to calculate P_25 based on J
def P_25(J):
    return 1 - (1 + (J / 0.362)**-4.75)**-0.4 - 0.03 * np.exp(-(J - 0.1)**2 / 0.005)
# Function to calculate P_rel based on J and T
def P_rel(J, T):
    return P_25(J) * f_J_T(J,T)
##################################SOLUTION VAPOR PRESSURE#########################
T=np.linspace(-50,200,101)
```

```
Conc=[0.1,0.2,0.3,0.4,0.5]
def vapor_pressure(T):
    return (10*760*0.133)/(1.013)*np.exp((-3968.06/(T-39.5735+273.15))+10.4592-4.04897*10**(-3)*T-
4.1752*10**(-5)*T**2+3.6851*10**(-7)*T**3-1.0152*10**(-9)*T**4+8.6531*10**(-
13)*T**5+9.03668*10**(-16)*T**6-1.9969*10**(-18)*T**7+7.79287*10**(-22)*T**8+1.91482*10**(-
25)*T**9)
def P_sol(J,T):
    return P_rel(J,T)*vapor_pressure(T)
P_sol_values={J:P_sol(J,T) for J in Conc}
plt.figure(figsize=(14, 7))
T_crit=[]
for J in Conc:
    if J==0.1:
        plt.semilogy(T,P_sol_values[0.1],label=f"X={10}%")
    else:
        P_sol_vals=P_sol_values[J]
        P_sol_crit=P_sol(J,Tcrit(J))
        P_sol_vals_above=[p if p>P_sol_crit else np.nan for p in P_sol_vals]
        plt.semilogy(T,P_sol_vals_above,label=f"X={J*100}%")
J=np.linspace(0.001,0.61,101)
plt.semilogy(Tcr[35:],P_sol(J,Tcr)[35:],label=f"Crystallization Line")
plt.title('Solution Vapor Pressure P_sol-T ')
plt.xlabel("Solution_Temperature T [°C] ")
plt.ylabel('Solution Vapor Pressure, P_sol')
plt.legend()
plt.grid(True)
plt.ylim((0.01,1000))
plt.xlim(-50,200)
plt.show()
```

Θερμοκρασία Δρόσου

```
import numpy as np
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline
# Constants
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY################################
# Function to calculate Th based on J
def calculate_Th(J):
```

```
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan  # Handle out of range values if necessary
# Vectorize the function to apply it to arrays
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
###################DEW POINT TEMPERATURE ###################
T=np.linspace(0,120,101)
Conc=[0,0.1,0.2,0.3,0.4,0.5]
def Dew_point_temp(J,T):
    JJ=J*100
    A=-0.02162-0.45308*JJ+0.04233*JJ**2-0.00208*JJ**3+2.34376*10**(-5)*JJ**4
    B=1.00071+0.00406*JJ-6.80357*10**(-4)*JJ**2+2.676*10**(-5)*JJ**3-4.48265*10**(-
7)*JJ**4+2.63846*10**(-9)*JJ**5
    return A+B*T
Dew_point_temp_values={J:Dew_point_temp(J,T) for J in Conc}
x=np.linspace(0,0.61,101)
plt.figure(figsize=(14, 7))
for J in Conc:
    dew_point_vals=Dew_point_temp_values[J]
    dew_crit=Dew_point_temp(J,Tcrit(J))
    dew_vals_above = [t if t > dew_crit else np.nan for  t in dew_point_vals]
    plt.plot(T,dew_vals_above,label=f"X={J*100}%")
plt.plot(Tcr[0:85],Dew_point_temp(x,Tcr)[0:85],label=f"Crystallisation line")
plt.title('Dew point temperature Tdp,T')
plt.ylabel("Dew point_Temperature T_dp [°C] ")
plt.xlabel('Solution_Temperature T [°C]')
plt.legend()
```

```
plt.grid(True)
plt.ylim((-30,120))
plt.xlim((0,120))
plt.show()
############DEW POINT T-X#######################################
temperatures=[0,10,20,30,40,50,60,70,80,90,100,110,120]
J=np.linspace(0,0.61,101)
Dew_point_temp_values_2={T:Dew_point_temp(J,T) for T in temperatures}
plt.figure(figsize=(14, 7))
for T in temperatures:
    plt.plot(J,Dew_point_temp_values_2[T],label=f"T={T} °C ")
plt.legend()
plt.grid(True)
plt.title('Dew point temperature Tdp,X')
plt.xlabel("X LiCl-H20 [%]")
plt.ylabel('Solution_Temperature T [°C]')
plt.show()
```

Διάγραμμα Ισορροπίας

```
import numpy as np
from scipy.optimize import ridder
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
from scipy.interpolate import UnivariateSpline

Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
#################################SOLUBILITY BOUNDARY#################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
```

```
    else:
        return np.nan
calculate_Th_vec = np.vectorize(calculate_Th)
def Tcrit(J):
    Th = calculate_Th_vec(J)
    return Th * Tc_H2O - 273.15
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
############################RELATIVE PRESSURE#############################
# Function to calculate f_J_T based on J
def f_J_T(J,T):
    A = 2 - (1 + (J / 0.28)**4.3)**0.6
    B = (1 + (J / 0.21)**5.1)**0.49 - 1
    return A + B * ((T + 273.15) / Tc_H2O)
# Function to calculate P_25 based on J
def P_25(J):
    return 1 - (1 + (J / 0.362)**-4.75)**-0.4 - 0.03 * np.exp(-(J - 0.1)**2 / 0.005)
# Function to find root for given J
def find_T_for_constant_f(f):
    func = lambda J: equilibrium(J, f) - Tcrit(J)
    result =  ridder(func,0.001, 0.555)  # Adjust bracket as necessary
    return result
#########################EQUILIBRIUM##################
J=np.linspace(0.001,0.61,101)
def equilibrium(J,f):
    A = 2 - (1 + (J / 0.28)**4.3)**0.6
    B = (1 + (J / 0.21)**5.1)**0.49 - 1
    return ((f/P_25(J)-A)/B)*Tc_H2O-273.15
f_values=[0.1,0.2,0.3,0.35,0.4,0.5,0.6,0.7,0.8,0.9]
equilibrium_values={f:equilibrium(J,f) for f in f_values}
plt.figure(figsize=(14, 7))
for f in f_values:
    boundary=find_T_for_constant_f(f)
    J_values=np.linspace(boundary,0.61,101)
    eq_vals_above=equilibrium(J_values,f)
    plt.plot(J_values,eq_vals_above,label=f"f={f*100}%")
plt.plot(J,Tcr,label=f"Crystallisation Line")
##########MODEL#############
W_air_abs=[0.008,0.0015]
Temp_air_abs=[30,25]
Conc_sol_abs=[0.41,0.328]
Temp_sol_abs=[22,18.7]
W_air_reg=[0.008,0.021]
```

```
Temp_air_reg=[30,38]
Conc_sol_reg=[0.33,0.415]
Temp_sol_reg=[22,49.8]
########EXPERIMENT###########
W_air_abs_exp=[0.0085,0.0023]
Temp_air_abs_exp=[30,21]
Conc_sol_abs_exp=[0.41,0.34]
Temp_sol_abs_exp=[22,18]
W_air_reg_exp=[0.01,0.0228]
Temp_air_reg_exp=[30,36]
Conc_sol_reg_exp=[0.33,0.4]
Temp_sol_reg_exp=[22,44]
##theoretical###
##absorber##
plt.plot(Conc_sol_abs,Temp_sol_abs,color='red',label="Abs_Model")
##regenerator#####
plt.plot(Conc_sol_reg,Temp_sol_reg,color='purple',label="Reg_Model")
###experiment###
###absorber###
plt.plot(Conc_sol_abs_exp,Temp_sol_abs_exp,color='green',label="Abs_Exp")
###regenerator#####
plt.plot(Conc_sol_reg_exp,Temp_sol_reg_exp,color='black',label="Reg_Exp")
plt.legend()
plt.ylim((-80,100))
plt.xlim((0,0.6))
plt.grid(True)
plt.xlabel("'Mass fraction of LiCl-H20, ξ'")
plt.ylabel('Solution_Temperature T [°C]')
plt.show()
```

Διαφορά Ενθαλπιών

```
import numpy as np
import matplotlib.pyplot as plt
from iapws import IAPWS97
import CoolProp.CoolProp as CP
Tc_H2O = 647.096  # Critical temperature of water in Kelvin
Tc_H2O_C=374.096  # Critical temperature of water in Celsius
# Mass fraction array
J = np.linspace(0, 0.61, 101)
##################################SOLUBILITY BOUNDARY################################
# Function to calculate Th based on J
def calculate_Th(J):
    if 0 <= J < 0.253:
        return 0.422088 - 0.09041 * J - 2.93635 * J ** 2.5
    elif 0.253 <= J < 0.287:
```

```
        return -0.005340 + 2.01589 * J - 3.114590 * J ** 2
    elif 0.287 <= J < 0.369:
        return -0.56306 + 4.72308 * J - 5.81105 * J ** 2
    elif 0.369 <= J < 0.452:
        return -0.31522 + 2.88248 * J - 2.62433 * J ** 2
    elif 0.452 <= J < 0.558:
        return -1.31231 + 6.17767 * J - 5.03479 * J ** 2
    elif 0.558 <= J <= 0.61:
        return -1.3568 + 3.44854 * J
    else:
        return np.nan
calculate_Th_vec = np.vectorize(calculate_Th)
# Calculate Th for each J
Th = calculate_Th_vec(J)
# Calculate Tcr
Tcr = Th * Tc_H2O - 273.15
##################################Differential Enthalpy of dilution##############################
J=np.linspace(0.001,0.59,101)
def h_w(T):
    temp=T+273.15
    return 0.22156863+4.19690058*temp-4.8993808e-4*temp**2+4.00756794e-6*temp**3
def dh_d(J,T):
    theta=(T+273)/Tc_H2O
    dh_d_0=169.105+457.850*theta
    zeta=J/(0.6000-J)
    return dh_d_0*(1+zeta**(-1.965))**(-2.265)
temperatures=[20,40,60,80,100]
plt.figure(figsize=(14, 7))
dh_d_values={T:dh_d(J,T) for T in temperatures}
dhd_cr=dh_d(J,Tcr)
for T in temperatures:
    dh_d_vals=dh_d_values[T]
    dh_d_vals_above=[dhd if dhd>dhd_cr[idx] else np.nan for (idx,dhd) in enumerate(dh_d_vals)]
    plt.plot(J,dh_d_vals_above,label=f"T={T} °C ")
plt.plot(J,dh_d(J,Tcr),label='Crystallization Line')
plt.legend()
plt.xlim(0.2,0.6)
plt.ylim(0,500)
plt.title("Differential enthalpy of dilution")
plt.grid(True)
plt.xlabel("Mass friction ξ")
plt.ylabel("Δh_d (kJ/kg_H20)")
plt.show()
```