ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών

ΠΑΡΟΥΣΙΑΣΗ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Θέμα

‘Έλεγχος ηλεκτρονόμων με την βοήθεια PLC’

Εισηγητής:
Κατσούλης Στέλιος

Εξεταστική επιτροπή:

Επιβλέπων καθηγητής
Γεώργιος Σταυρακάκης

καθηγητής
Σταυρουλάκης Πέτρος

καθηγητής
Καλαϊτζάκης Κων/νος

Χανιά
2004

Στην γυναίκα μου Γιάννα που με
στήριξε τόσα χρόνια, και με βοήθησε να
μην χάσω το δρόμο μου,

Στον κύριο Ηλία που μου έδειξε
λόγους για τους οποίους αξίζει να μάχεται
κανείς σήμερα,

Στα ‘παιδιά’ μου, επειδή υπάρχουν και
με γεμίζουν χαρά και δύναμη για να
προχωρήσω,

Στους φίλους μου που με στήριξαν
όταν το είχα περισσότερο ανάγκη.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω την ABB και για την ευκαιρία που μου έδωσε να γνωρίσω την φιλοσοφία με την οποία λειτουργεί καθώς και να χρησιμοποιήσω τον εξοπλισμό και τις εγκαταστάσεις της, και ιδιαίτερα τον κύριο Βοσυνιώτη Σταύρο υπεύθυνο του τομέα των αυτοματισμών, για τον τρόπο με τον οποίο με βοήθησε σε κάθε δυσκολία που συνάντησα κατά την διεκπεραίωση της εργασίας μου και για το ότι φρόντισε να αισθάνομαι άνετα σε ένα μέχρι τότε ξένο περιβάλλον.

Θα ήθελα επίσης να ευχαριστήσω τον καθηγητή μου κύριο Γεώργιο Σταυρακάκη για την ευκαιρία που μου έδωσε να χρησιμοποιήσω τις γνώσεις μου στην αγορά καθώς και για την καθοδήγηση που μου παρείχε καθ' όλη την διάρκεια της εκπόνησης της εργασίας.

Τέλος θέλω να ευχαριστήσω την οικογένειά μου για την υπομονή που έδειξε, αλλά και την γυναίκα μου, την 'καινούρια μου' οικογένεια, και τους φίλους μου για την αμέριστη συμπαράστασή τους όλα αυτά τα χρόνια.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΑΡΟΥΣΙΑΣΗ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ 1
ΕΥΧΑΡΙΣΤΙΕΣ 3
Λίγα στοιχεία για την εταιρία. 5
ΕΙΣΑΓΩΓΗ 5
ΣΚΟΠΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ 7
ΕΡΓΑΛΕΙΑ (COMPONENTS) ΤΗΣ ΕΡΓΑΣΙΑΣ 8
ΠΑΡΟΥΣΙΑΣΗ – ΑΝΑΠΤΥΞΗ ΤΩΝ ΕΡΓΑΛΕΙΩΝ 10
Ηλεκτρονόμοι της εταιρίας ABB (μοντέλα της σειράς SPAJ) 10
PLC ABB AC 1131 (με επεξεργαστή 07KT97B) 11
ABB AC 1131 Programming Software 12
ABB SPA Bus communication protocol 14
Λίγες λεπτομέρειες και σχηματική επεξήγηση της πειραματικής διάταξης 20
ΑΝΑΛΥΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ 24
Στάδιο 1^{ο}: 24
Στάδιο 2^{ο}: 24
Στάδιο 3^{ο}: 25
Σχολιασμός του κώδικα. 32
Στάδιο 4^{ο} 34
Η πρώτη κατηγορία 34
ΑΝΑΠΤΥΞΗ ΓΕΝΙΚΩΝ ΕΡΓΑΛΕΙΩΝ 39
Αντικείμενο. 39
Συναρτήσεις (functions) 40
Πρόγραμμα (program). 40
ΕΠΕΞΗΓΗΣΗ ΚΩΔΙΚΑ 42
ΛΙΣΤΑ ΑΝΤΙΚΕΙΜΕΝΩΝ 42
SPA_INIT 42
SPA_REC 44
SPA_SEND 46
MESCOMPOSE 50
MESDECOMPOSE 53
ERROR_HANDLING 58
SUSPEND 59
SLAVE_DEFINITION 61
NORMAL_SEQUENCE 64
CLEANING 66
SPAJ140C 68
PLC_PRG 71
ΕΠΙΛΟΓΟΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ 75
ΚΕΙΜΕΝΟ ΒΟΗΘΕΙΑΣ ΤΗΣ ΒΙΒΛΙΟΘΗΚΗΣ 77
Βιβλιογραφία 100

Λίγα στοιχεία για την εταιρία.

Η ABB είναι η εταιρία που προήλθε το 1987 από την συγχώνευση δύο κολοσσών στον τομέα της ηλεκτρολογίας με ιστορία στο χώρο, της Σουηδικής ASEA και της Ελβετικής Brown Boveri.

Στην Ελλάδα η ABB έχει το δικό της παράρτημα το οποίο διαθέτει γνώση και εμπειρία περισσότερο από οποιονδήποτε άλλο στην Ελλάδα στα θέματα ηλεκτρολογίας και αυτοματισμού.

ΕΙΣΑΓΩΓΗ

Η ραγδαία εξέλιξη της βιομηχανίας καθώς και η ανάπτυξη της αντίληψης της μαζικής παραγωγής μέσα στον εικοστό αιώνα, δημιούργησαν προϋποθέσεις για την δημιουργία και την εξέλιξη του αυτομάτου ελέγχου σε όλους τους τομείς. Ένας από αυτούς είναι και η προστασία ηλεκτρολογικών εγκαταστάσεων.

Οι ηλεκτρονόμοι προστασίας ή αλλιώς «ρελέ» όπως έχει συνηθιστεί να λέγονται, έχουν γίνει αναπόσπαστο μέρος των ηλεκτρολογικών εγκαταστάσεων ισχύος μέσης αλλά πολλές φορές και χαμηλής τάσης, όπου απαιτείται έλεγχος και προστασία από υπερένταση και υπέρταση της τάσης και έντασης του ρεύματος που διαπερνά μια συγκεκριμένη συσκευή. Από εδώ και πέρα ο συγγραφέας θα χρησιμοποιεί μόνο την λέξη «ηλεκτρονόμος» καθότι εκτός του ότι είναι πιο σωστή σαν λέξη, αποτρέπει και τον αναγνώστη από συγχύσεις με ελεγκτές άλλου είδους όπως λόγου χάρη στάθμης νερού, αφού ο σκοπός του περιγράφεται ακριβώς από το όνομα ήλεκτρο – νόμος.

Οι ηλεκτρονόμοι αρχικά περιορίζονταν στο να ανοίγουν το κύκλωμα όταν για κάποιον λόγο τους διαπερνούσε υπερβολική τάση ή ένταση, και υπερθερμαίνονταν. Αυτό απαιτούσε ή την αλλαγή του ηλεκτρονόμου, ή αναμονή έτσι ώστε ο ηλεκτρονόμος να κρυώσει.

Το ένα από τα παραπάνω δεν αποτελούσε πλέον λύση καθώς ηλεκτρονόμοι ελέγχου τάσης και έντασης σε βιομηχανίες, δεν ήταν δυνατό να «παγώνουν» την αλυσίδα παραγωγής κάθε φορά που χρειάζονταν αλλαγή.

Επίσης παρατηρήθηκε η δυνατότητα της συλλογής σημαντικών πληροφοριών μέσο των ηλεκτρονόμων που αφορούν στο εν λόγω κύκλωμα. Έτσι οι ηλεκτρονόμοι άρχισαν να έχουν ενσωματωμένα βολτόμετρα και αμπερόμετρα, καθώς και άλλα όργανα που με το πέρασμα του χρόνου ο κατασκευαστής μπορούσε πιο εύκολα να ενσωματώσει στον ηλεκτρονόμο, αλλά και ο αγοραστής θεωρούσε ως απαραίτητο προσόν.

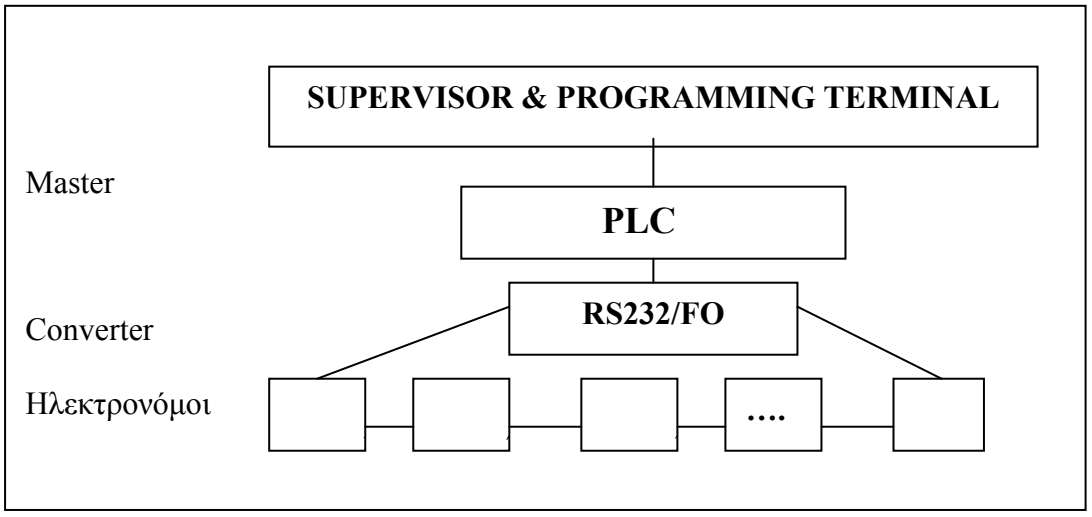
Με τον καιρό λοιπόν, οι ηλεκτρονόμοι αναπόφευκτα εξελίχθηκαν σε μονάδες, που μπορούσαν να δεχθούν από εξωτερικό χρήστη τις συνθήκες κάτω από τις οποίες θα ανοίγεται το κύκλωμα μέχρι να φτάσουν στην

σημερινή τους μορφή που έχουν δυνατότητα να προγραμματιστούν, να αποθηκεύουν σε ειδικούς καταχωρητές (registers) τιμές τάσης και έντασης του κυκλώματος που τους διαρρέει, αλλά και χαρακτηριστικές κυματομορφών των παραπάνω τιμών. Έτσι ένας σύγχρονος «ακριβός» ηλεκτρονόμος, μπορεί να διαθέτει δική του οθόνη έτσι ώστε να μπορεί εύκολα να προγραμματιστεί να αποθηκεύει τιμές της τάσης και της έντασης του ρεύματος κάθε λόγου χάρη πέντε δευτερόλεπτα και να τα καταχωρεί σε ένα πίνακα, αλλά και να ανοίγει το κύκλωμα σε περίπτωση που πάρει συγκεκριμένη είσοδο με συγκεκριμένη κυματομορφή για συγκεκριμένο χρονικό διάστημα, και όταν αυτό συμβεί να καταγράψει στην μνήμη του την κυματομορφή του ρεύματος, της τάσης, της ισχύς καθώς και όποιων άλλων παραμέτρων ο χειριστής θεωρεί αναγκαίο για λόγου χάρη πέντε δευτερόλεπτα πριν και πέντε μετά το εν λόγω σφάλμα.

ΣΚΟΠΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Στην εκπόνηση της εργασίας, επιδιώκεται η επικοινωνία ηλεκτρονόμων οι οποίο υπακούν σε ένα συγκεκριμένο πρωτόκολλο επικοινωνίας, με ένα συγκεκριμένο PLC. Για την επίτευξη του σκοπού αυτού, απαιτούνται και κάποια άλλα στοιχεία που αναπτύσσονται στην συνέχεια αυτού του κειμένου και τα οποία σχηματικά αναπαρίστανται στο σχήμα 1.

Για την επικοινωνία PLC – ηλεκτρονόμου, χρησιμοποιείται συγκεκριμένο πρωτόκολλο, ή καλύτερα συγκεκριμένο μέρος πρωτοκόλλου.

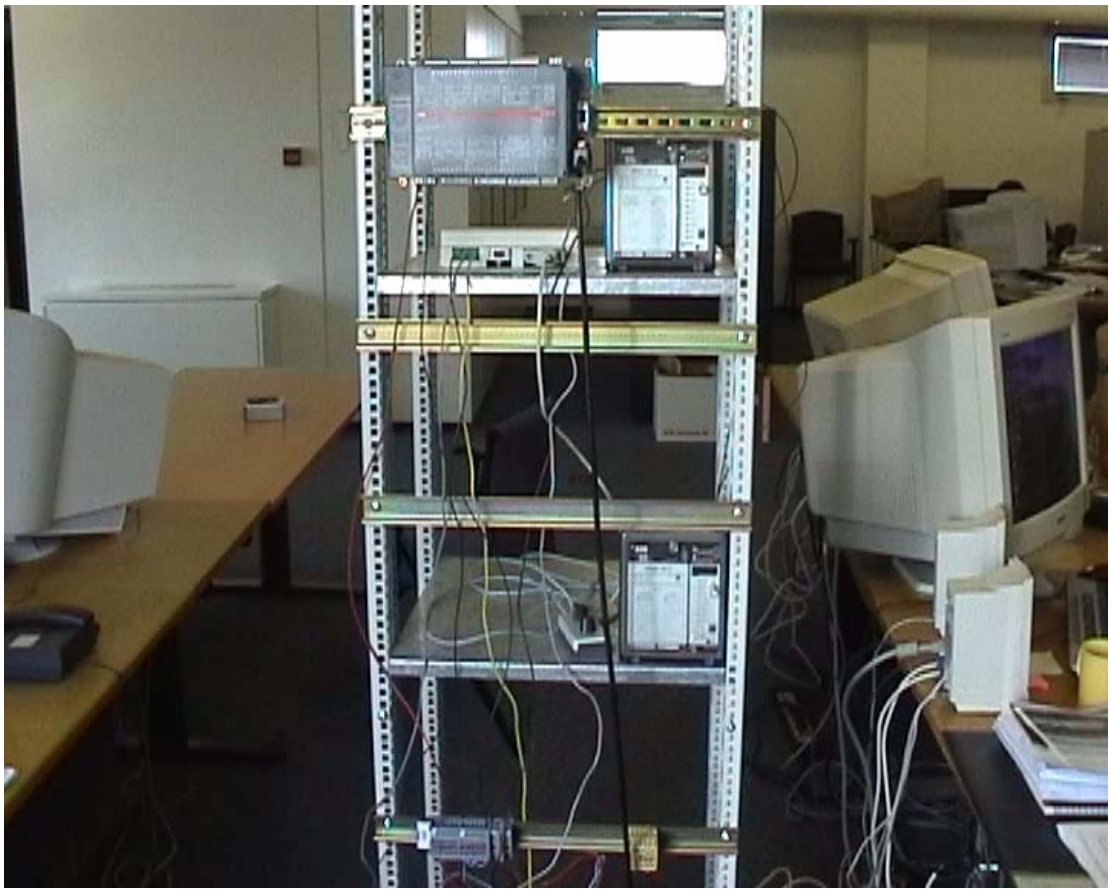


Σχήμα 1. Διάταξη τυπικής σύνδεσης πρωτοκόλλου SPA Bus

ΕΡΓΑΛΕΙΑ (COMPONENTS) ΤΗΣ ΕΡΓΑΣΙΑΣ

Τα εργαλεία (components) που θα παρουσιασθούν παρακάτω και θα χρησιμοποιηθούν κατά την εκπόνηση της εργασίας παρατίθενται παρακάτω.

- Ηλεκτρονόμοι της εταιρίας ABB (μοντέλα της σειράς SPAJ)
- PLC ABB AC 1131 (με επεξεργαστή 07KT97B)
- ABB AC 1131 Programming Software
- Πρωτόκολλο επικοινωνίας ABB SPA Bus.



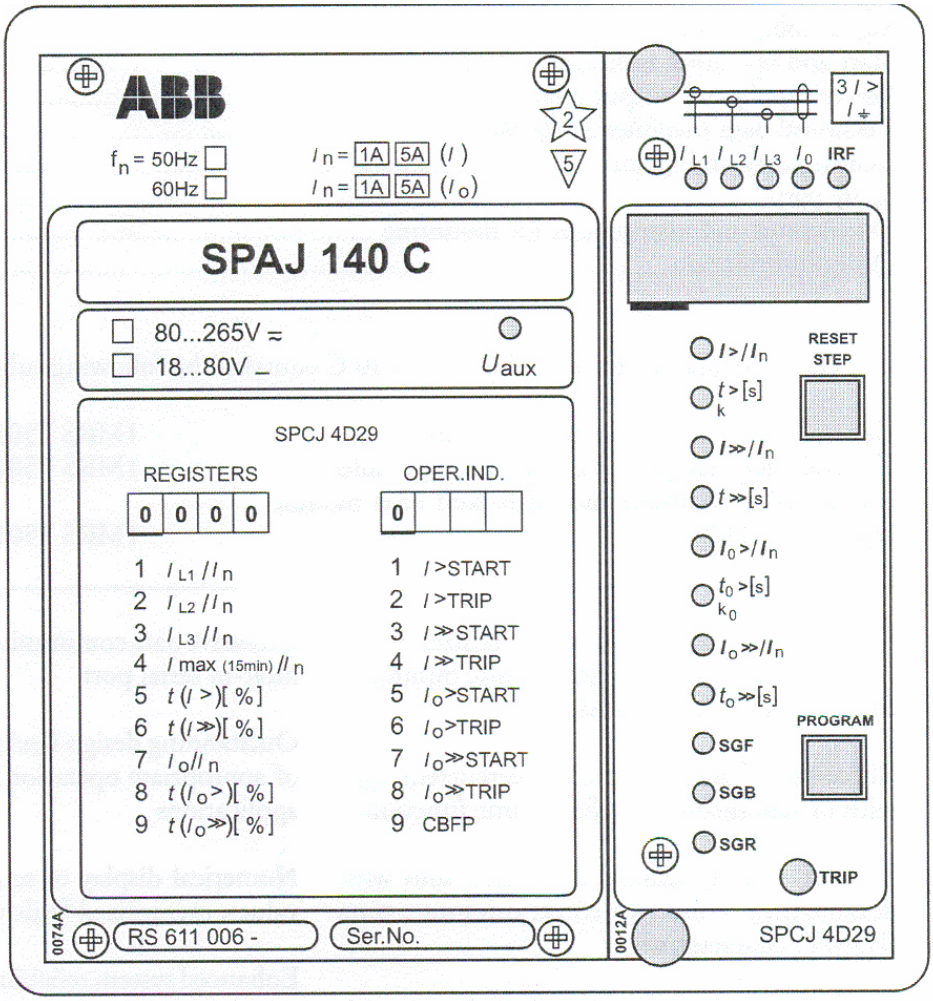
Εικόνα 1. Διακρίνεται η σχετική πειραματική διάταξη του σχήματος 1. Διακρίνονται δύο ηλεκτρονόμοι, ένα PLC, ένας βοηθητικός ροοστάτης, και στα δεξιά ένας υπολογιστής ο οποίος χρησιμοποιήθηκε σαν μέσο προγραμματισμού του PLC.

ΠΑΡΟΥΣΙΑΣΗ – ΑΝΑΠΤΥΞΗ ΤΩΝ ΕΡΓΑΛΕΙΩΝ

Ο συγγραφέας κρίνει σκόπιμο να ξεκινήσει την ανάπτυξη των εργαλείων από τα εργαλεία υλικού (hardware) του project και να συνεχίσει στο λογισμικό (software) όπως το ABB AC1131 programming software ή το πρωτόκολλο επικοινωνίας ABB SPA Bus, επειδή έτσι θα δοθεί στον αναγνώστη μια αντίληψη του «σκελετού» πάνω στον οποίο θα χτισθεί το περιβάλλον επικοινωνίας από την πλευρά του PLC.

Προς αποφυγή ασαφειών, ο συγγραφέας επίσης αναφέρει πως από εδώ και πέρα στο κείμενο ως ‘slaves’ θα αναφέρονται οι ηλεκτρονόμοι και ως ‘master’ το PLC.

Ηλεκτρονόμοι της εταιρίας ABB (μοντέλα της σειράς SPAJ)



Σχήμα 4. Μπροστινή όψη ηλεκτρονόμου της ABB μοντέλου SPAJ 140C

Στο σχήμα 4, φαίνεται ένας ηλεκτρονόμος της σειράς SPAJ. Τα μοντέλα αυτά θα χρησιμοποιηθούν κατά κόρον στην εργασία του συγγραφέα, για αυτό και παρουσιάζουμε το συγκεκριμένο. Φυσικά υπάρχουν διάφορα μοντέλα ηλεκτρονόμων αλλά αυτό μπορεί να χρησιμοποιηθεί ως 'benchmark' για το πρόγραμμα αφού είναι το πλέον χρησιμοποιούμενο σε απλές εφαρμογές προστασίας στοιχείων .

Το μοντέλο αυτό (αυτό ισχύει και για το 140 και 140C) ελέγχει μόνο ένταση του ρεύματος και σφάλμα γείωσης(overcurrent and earth fault relay) , γεγονός που το καθιστά σαν το φτωχό συγγενή σε σχέση με τα υπόλοιπα μοντέλα της εταιρείας. Παρ' όλα ταύτα και παρά το γεγονός πως το μοντέλο αυτό διαθέτει μόνο ένα κανάλι, το οποίο είναι και το κανάλι '0' (μηδέν) που όπως προαναφέρθηκε είναι απαραίτητο για επιτευχθεί επικοινωνία, παρέχει στον προγραμματιστή του μεγάλο αριθμό καταχωρητών, γεγονός που το καθιστά αρκετά ευέλικτο. Η δυνατότητά του, να προγραμματίζεται είτε από τον πίνακα ελέγχου του ή μέσο του SPA Bus από το PLC, το καθιστά και εύκολο στις επανεκκινήσεις αφού δεν χρειάζεται η παρουσία τεχνικού μετά από επανεκκίνηση.

Αντίστοιχες συσκευές της σειράς SPA προστατεύουν κινητήρες μέσης τάσης, μετασχηματιστές, γεννήτριες κ.τ.λ.

PLC ABB AC 1131 (με επεξεργαστή 07KT97B)

Ο ελεγκτής προγραμματιζόμενης λογικής (που στο εξής θα αποκαλείται χάριν συντομίας απλά 'PLC'), που χρησιμοποιήθηκε στην παρούσα εργασία είναι ο AC31 της ABB με επεξεργαστή 07KT97.

Οι δυνατότητες που παρέχει είναι ποικίλες. Από αυτές αναφέρονται χαρακτηριστικά μόνο εκείνες που αφορούν την παρούσα εργασία.

Επιγραμματικά, το PLC μπορεί να υποστηρίξει μέσο δυο σειριακών θυρών των 9 pin που διαθέτει, επικοινωνία με ηλεκτρονικό υπολογιστή για τον προγραμματισμό του, αλλά και για την περαιτέρω επίβλεψη του προγράμματος κατά την ώρα εκτέλεσής του (real time) καθώς και για την επικοινωνία του με άλλες συσκευές μέσω MODBUS ή οποιουδήποτε ASCII πρωτοκόλλου επικοινωνίας.

Εκμεταλλευόμενος το τελευταίο, ο συγγραφέας ανέπτυξε το ASCII πρωτόκολλο επικοινωνίας SPABUS.

Το PLC αυτό κάθε αυτό μπορεί να υποστηρίξει πολύ υψηλές ταχύτητες επικοινωνίας (έως και 115200 Kb/s), όμως περιορίζεται στο συγκεκριμένο project από το όριο των 9600 Kb/s στα οποία πρέπει να υπάρχει επικοινωνία για να λειτουργήσει το SPA Bus protocol, το οποίο δεν υποστηρίζει υψηλότερη ταχύτητα.

ABB AC 1131 Programming Software

Για την ανάπτυξη του πρωτοκόλλου SPABUS χρησιμοποιείται μέρος των δυνατοτήτων του λογισμικού προγραμματισμού του PLC εφόσον όπως προαναφέρθηκε το πρόγραμμα αυτό μέρος ή εργαλείο μελλοντικών εφαρμογών.

Επιγραμματικά παρουσιάζονται όλες οι μορφές κώδικα που υποστηρίζει το AC1131 το οποίο είναι πλήρως συμβατό με το IEC 61131 standard.

1. Προγραμματισμός υψηλού επιπέδου (IL, instruction list)

Πρόκειται για κώδικα που έχει παρόμοια μορφή με γλώσσα προγραμματισμού υψηλού επιπέδου.

2. Προγραμματισμός δομημένου κειμένου (ST, structured text)

Η μορφή αυτή είναι χρήσιμη μεγάλων εφαρμογών. Γενικά αξίζει να σημειωθεί πως η μορφή της είναι μάλλον πιο κατανοητή καθώς όλο το help του AC1131 programming software είναι κατασκευασμένο σύμφωνα με τους κώδικες ST και FBD(αναφέρεται παρακάτω) με αποτέλεσμα να μειώνεται η σημασία του IL τύπου κώδικα.

3. Προγραμματισμός σε σχηματικό κώδικα (FBD, function block diagram).

Εδώ έχουμε τον προγραμματισμό μικρών Function Block. Αν και αυτή η γλώσσα για μεγάλο κώδικα θα γινόταν εντελώς αχανής, εντούτοις για σχετικά μικρό μέγεθος κώδικα, έχει τα καλύτερα αποτελέσματα από όλες στον compiler, δεν δημιουργεί warnings, και ο κώδικας που δημιουργείται από αυτή είναι ο γρηγορότερος κατά την εκτέλεση. Η μορφή του βοηθά πολύ στην ανάπτυξη εφαρμογών αυτοματισμού οπού απαιτείται περίπλοκη αλλά και επαναλαμβανόμενη λογική.

4. SFC
5. LD

Οι δύο τελευταίες γλώσσες δεν βρίσκονται σε ευρεία χρήση και ο συγγραφέας δεν έχει συναντήσει παραδείγματά τους ακόμα. Για περισσότερες λεπτομέρειες για αυτές ο αναγνώστης θα μπορεί να ανατρέξει στο IEC 61131 standard.

Το λογισμικό μπορεί να προγραμματίσει όλα τα μοντέλα PLC της σειράς AC31 της ABB.

Παρόλα ταύτα, το συγκεκριμένο σύγγραμμα θα περιοριστεί στον προγραμματισμό των σειρών 07KT97.

Η δομή ενός προγράμματος είναι πολύ αυστηρή και συγκεκριμένη. Μέσα σε ένα project μπορούν να φιλοξενούνται πολλά ‘objects’, (αντικείμενα). Τα ‘objects’ αυτά μπορούν να είναι programs, functions, είτε function blocks.

Τα function blocks είναι το κομμάτι του προγράμματος το οποίο προγραμματίζει ένα από τα αντίστοιχα εσωτερικά function blocks που διαθέτει σε hardware στο εσωτερικό του το PLC. Έτσι υλοποιείται μια μορφή προγραμματιζόμενου hardware, κάτι σαν FPGA. Ο συνδυασμός αυτός είναι πολύ γρήγορος. Τα function blocks συνήθως περιέχουν μικρά κομμάτια κώδικα (για αποφευχθεί overflow) και έτσι συνδυάζονται με προγραμματισμό σε FBD οπότε το αποτέλεσμα είναι κάτι πολύ γρήγορο. Μικρά μέλη ενός μεγάλου προγράμματος που επαναλαμβάνονται συνεχώς προγραμματίζονται έτσι από τον έμπειρο προγραμματιστή.

Ένα project μπορεί να φιλοξενήσει objects διαφόρων τύπων και προγραμματισμένα σε διάφορες μορφές. Εκεί που η γλώσσα είναι πολύ αυστηρή, είναι στην δήλωση μεταβλητών. Όλες οι μεταβλητές δηλώνονται στην αρχή κάθε action. Οι internal μεταβλητές δηλώνονται χωριστά, οι μεταβλητές εισόδου χωριστά, οι μεταβλητές εξόδου χωριστά, οι εισόδου –εξόδου χωριστά, και οι global χωριστά σε κοινό domain που υπάρχει ειδικά για την δήλωση global μεταβλητών. Έξω από ένα action,

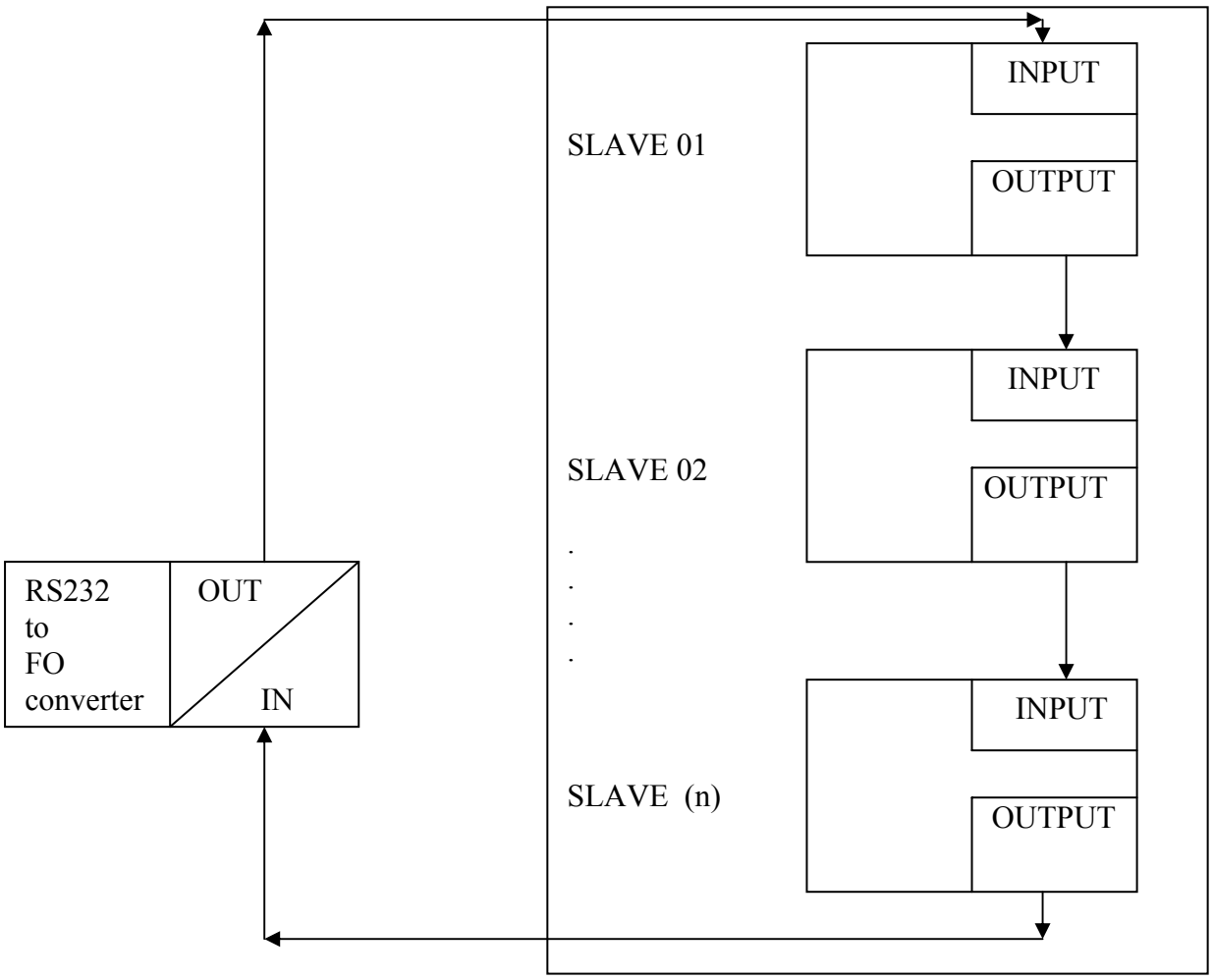
το action αυτό εμφανίζεται στα υπόλοιπα actions καθώς και στο main program ως ένα κλειστό κουτί με εισόδους τις μεταβλητές εισόδου και το ένα μέρος των εισόδων –εξόδων και εξόδους τις μεταβλητές εξόδου και το άλλο σκέλος των μεταβλητών εισόδου -εξόδου. Επίσης υπάρχει ειδικό domain στο οποίο δηλώνονται οι μεταβλητές τύπου constant. Κατά τα άλλα το πρόγραμμα υποστηρίζει τις γνωστές σε γενικές γραμμές μεταβλητές (string, Boolean, integer, char κ.τ.λ.) όμως με κάποια ιδιαίτερα χαρακτηριστικά. Έτσι λόγου χάρη, σε if –then –else loop σαν συνθήκη μπορούν να συμμετάσχουν μόνο μεταβλητές που μπορούν να μετατραπούν σε συνθήκη Boolean.

ABB SPA Bus communication protocol.

Είναι ένα ‘master - slave’ πρωτόκολλο, που αποσκοπεί στην επικοινωνία μιας συσκευής (master) με πολλές συσκευές(slaves).

Η μορφή αυτή της επικοινωνίας όμως θα χρειάζεται και ελάχιστους πόρους αφού η μέγιστη ταχύτητα που το πρωτόκολλο υποστηρίζει είναι 9600 Kbps.

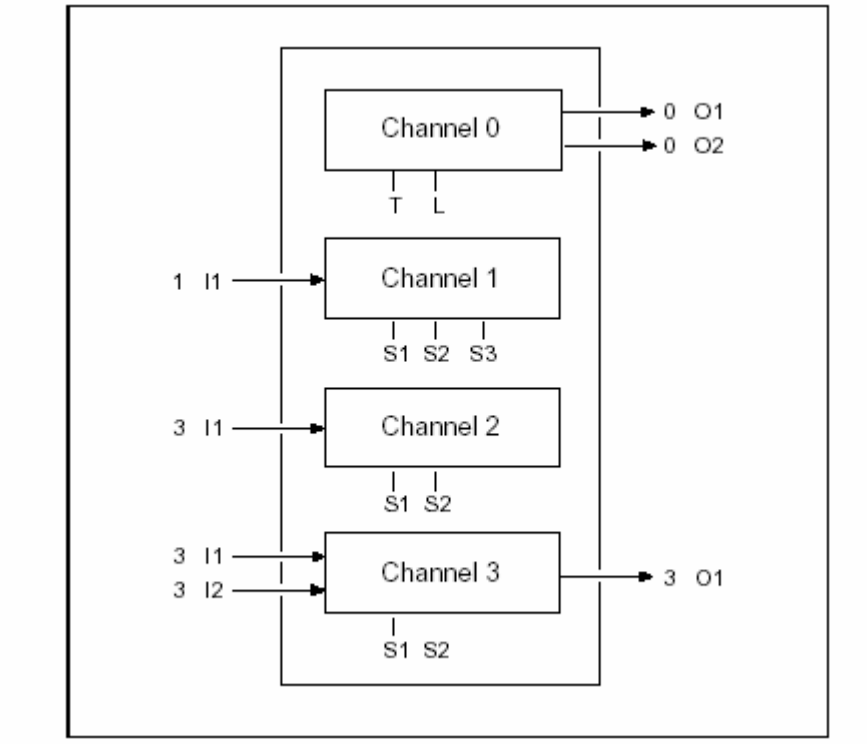
Έτσι προκειμένου το δίκτυο να μπορεί να ανταποκριθεί σε κάθε περίσταση, χωρίς περίπτωση σφάλματος, γίνεται η παραδοχή του ότι, ο κάθε SLAVE δεν έχει την ανάγκη αλλά ούτε και το δικαίωμα να πάρει την πρωτοβουλία, για να ανοίξει συνομιλία με τον MASTER, αλλά απαντά στα μηνύματα αυτού μόνο όταν του ζητείται. Παρακάτω δίνεται η δομή και η μορφή της συνδεσμολογίας των ηλεκτρονόμων – SLAVES μεταξύ των.



Σχήμα 2. Αναπαράσταση του δικτύου των SLAVE.
Παραπάνω δίνεται η τοπολογία του δικτύου επικοινωνίας SPA Bus.

Ο κάθε SLAVE έχει μια οπτική είσοδο και μια οπτική έξοδο, τις οποίες και χρησιμοποιεί για να επικοινωνεί με τον MASTER. Ο τελευταίος, στο παραπάνω σχήμα υποθετικά βρίσκεται κάπου πιο αριστερά από τον RS232 to Fiber Optical converter και επικοινωνεί αμφίδρομα με αυτόν.

Αφού δόθηκε μια γενική εικόνα του όλου συστήματος PLC(master) – SPAJ (slaves), παρακάτω το κείμενο θα εστιάσει στην ανάπτυξη του slave συγκεκριμένα έτσι ώστε να μπορέσει να αναλυθεί το πρωτόκολλο επικοινωνίας του.



Σχήμα 3. Λογική αναπαράσταση ενός ηλεκτρονόμου με 3 κανάλια.

Στο σχήμα 3 αναπαρίσταται η δομή των παραμέτρων που μπορεί να διαθέσει για το δίκτυο SPA Bus ένας ηλεκτρονόμος/slave. Παρατηρείται πως ο slave διαθέτει κάποιες παραμέτρους.

Λεπτομερέστερα:

Οι πληροφορίες εισόδου επιβλέπονται θέτοντας οριακές τιμές και delays στα σήματα εισόδου. Ο κάθε slave (και φυσικά και ο συγκεκριμένος) συνεργάζεται με ένα ρολόι πραγματικού χρόνου (real time clock) για να σημαδεύει τα καταγεγραμμένα συμβάντα.

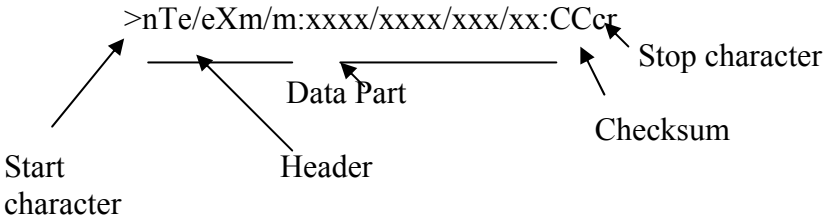
Ένας slave μπορεί να χωριστεί σε κανάλια (channels). Αυτά είναι σχεδόν ανεξάρτητες λειτουργικές οντότητες. Τυπικά, κάθε κανάλι έχει μόνο μία είσοδο και μερικές εξόδους. Η αρίθμηση των καναλιών ξεκινά από το νούμερο 1. Το κανάλι 0 χρησιμοποιείται για την αποθήκευση κάθε κοινής πληροφορίας του συγκεκριμένου (και κάθε slave).

Στο σχήμα 3 φαίνεται ένας slave τριών καναλιών όπως τον «βλέπει» το πρωτόκολλο επικοινωνίας SPA Bus αφού έχουν τεθεί οι είσοδοι I1, I2, κ.τ.λ. και οι έξοδοι O1, O2 κ.τ.λ. στα σχηματισμένα κανάλια.

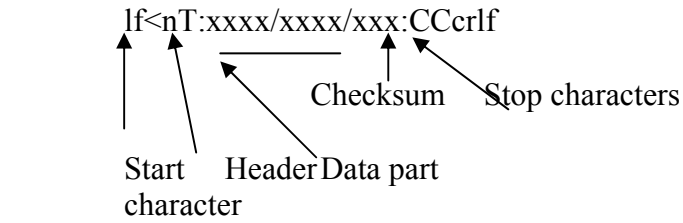
Το δείγμα εδώ έχει τέσσερις εισόδους εκ των οποίων μία έχει δοθεί στο κανάλι 1, μία στο δύο και δύο στο κανάλι 3. Από τις τρεις εξόδους, οι δύο είναι κοινές για όλα τα κανάλια και για αυτό εμφανίζονται στο κανάλι 0 (όπως προαναφέρθηκε), ενώ η τελευταία ελέγχεται από το κανάλι 3. Στο κανάλι 1 έχουν τεθεί τρεις setting values (S1, S2, S3) ενώ στα κανάλια 2 και 3 έχουν τεθεί από 2. Στο κανάλι 0 έχουν επίσης δοθεί οι T και L.

Τα σήματα που δέχονται και αναγνωρίζουν οι ηλεκτρονόμοι σύμφωνα με το SPA Bus communication protocol έχουν την παρακάτω μορφή.

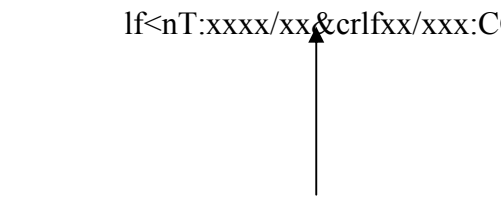
Ο master στέλνει ένα μήνυμα με την μορφή:



και αντίστοιχα δέχεται μια απάντηση της μορφής;



ή



όπου:

n = ο αριθμός του slave

T = ο κωδικός του τύπου μηνύματος

e = ο αριθμός του καναλιού
e/e = ή αριθμός πρώτου καναλιού/ αριθμός τελευταίου καναλιού

x = κωδικός κατηγορίας δεδομένων (τύπος δεδομένων)

m = αριθμός δεδομένων
m/m = πρώτο/ τελευταίο δεδομένο.

Τα ‘:’ χρησιμοποιούνται για τον διαχωρισμό του δείκτη του μηνύματος από το κομμάτι δεδομένων και για τον διαχωρισμό του κομματιού δεδομένων από το checksum αντίστοιχα.

Τα μηνύματα από και προς τους slaves, μπορούν να αποτελούνται μόνο από εκτυπώσιμους χαρακτήρες ASCII (0AH, 0DH, 2OH…,7EH). Πάντα τα μηνύματα του master ξεκινούν με ‘>’ ενώ των slaves με ‘lf<’. Τα μηνύματα του master λήγουν με τους χαρακτήρες cr (0DH) ενώ αυτά των slaves με crlf (0DH και 0AH). Επειδή το πάνω όριο της μιας γραμμής μηνύματος είναι 255 χαρακτήρες, οι slaves χρησιμοποιούν τον χαρακτήρα ‘&’ για να δηλώνουν την συνέχεια του μηνύματός τους στην επόμενη γραμμή εάν αυτό δεν χωράει σε αυτούς τους 255 χαρακτήρες.

Οι χαρακτήρες εκκίνησης και λήξης πρέπει να περιέχονται σε κάθε μήνυμα που στέλνει ο master καθώς και ο κάθε slave. Διαφορετικά, το μήνυμα δεν αναγνωρίζεται από τον παραλήπτη, εξαιτίας της αντίληψης του πρωτοκόλλου επικοινωνίας. Η επικοινωνία γίνεται με βασική ιδέα το πρωτόκολλο round – robin. Έτσι, μη πλήρες μήνυμα, δεν αναγνωρίζεται από τον παραλήπτη και ‘αιωρείται’ στο δίκτυο με την ιδέα ότι απευθύνεται σε κάποιον άλλο. Σε περίπτωση τέτοιου μηνύματος δεν αποστέλλεται καν μήνυμα λάθους αφού φαινομενικά δεν απευθύνεται σε κανέναν.

Παρακάτω δίδονται παραδείγματα τυπικού μηνύματος.

Παράδειγμα 1:
Ο master αποστέλλει ένα μήνυμα ανάγνωσης της μορφής:

>2R1:XXcr
Εδώ ζητούνται από τον slave 2 οι τιμές που έχει στο κανάλι 1 .

Ο slave απαντά:

lf<2D:10.1:XXcrlf

Παράδειγμα 2:

Ο master ζητά από τον slave 1 να του επιστρέψει το μοντέλο του.:

>1RF:XX

Ο slave απαντά με ένα μήνυμα:
lf<1D: SPAJ 142C:XXcrlf

Τύποι δεδομένων σημάτων:

Τα σήματα που μπορούν να αποσταλούν από τον master είναι βασικά δυο μορφών:

- Ανάγνωσης (R)
- Εγγραφής (W)

Αυτά χωρίζονται ως εξής:
Ανάγνωσης:
Χωρίζονται σε ανάγνωσης δεδομένων και ανάγνωσης πληροφοριών συσκευών.

Εγγραφής:

Χωρίζονται σε σήματα απευθείας εγγραφής, μαζικής εγγραφής, και εγγραφής χρόνου. Οι slaves έχουν real time clock και αυτό αρχικοποιείται από τον master. Στις περιπτώσεις μαζικής εγγραφής, και εγγραφής χρόνου, ο master δεν λαμβάνει acknowledgements από τους slaves γιατί τότε σε ένα μεγάλο δίκτυο θα είχαμε πτώση του συστήματος από το μέγεθος των πληροφοριών. Υπάρχει ειδική διεύθυνση για το broadcast σημάτων μαζικής εγγραφής.

Τα σήματα του slave χωρίζονται σε:

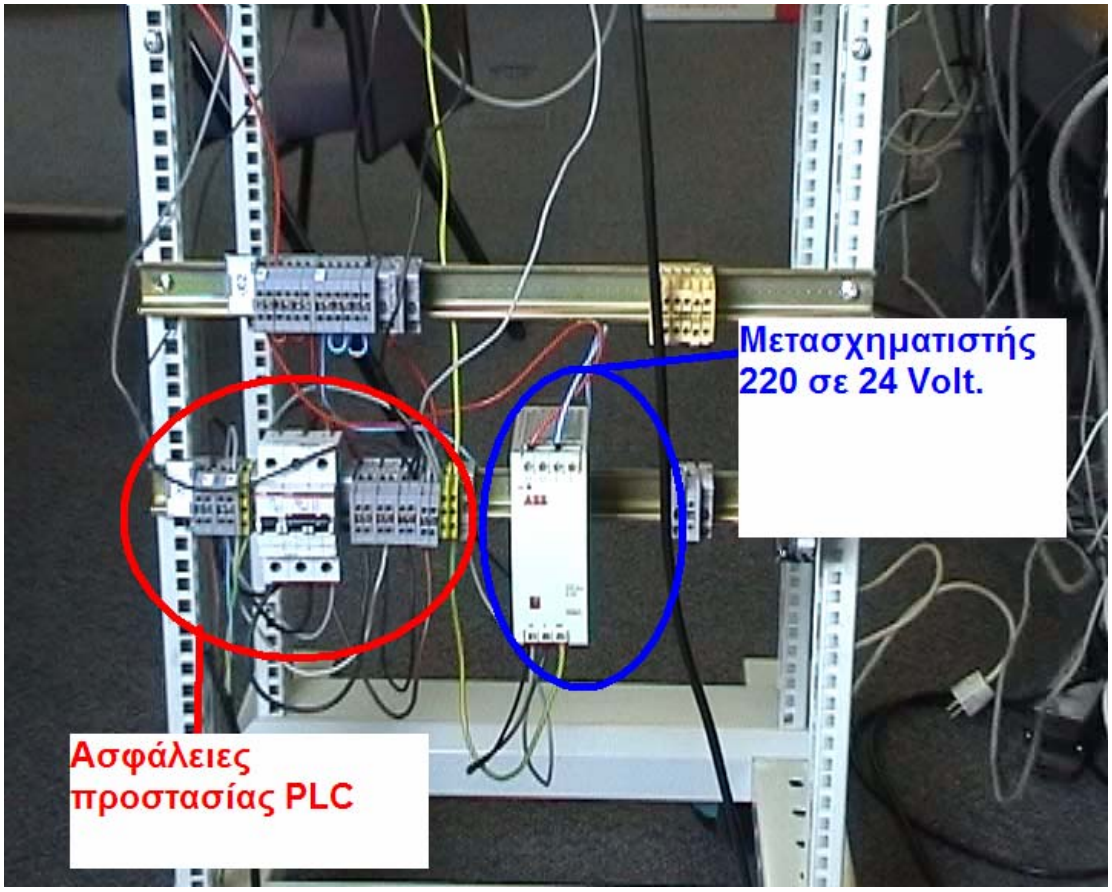
- Σήματα δεδομένων (D data message)
- Αρνητική απάντηση (N Nack)
- Θετική απάντηση. (A Acknowledge)

Δόθηκε η βασική ιδέα της επικοινωνίας με το πρωτόκολλο SPA Bus της ABB. Περισσότερες πληροφορίες παρατίθενται στο ίδιο το εγχειρίδιο χρήσης, καθώς το να παρατεθούν εδώ μάλλον θα αποπροσανατόλιζε το κείμενο από τον στόχο του.

Λίγες λεπτομέρειες και σχηματική επεξήγηση της πειραματικής διάταξης

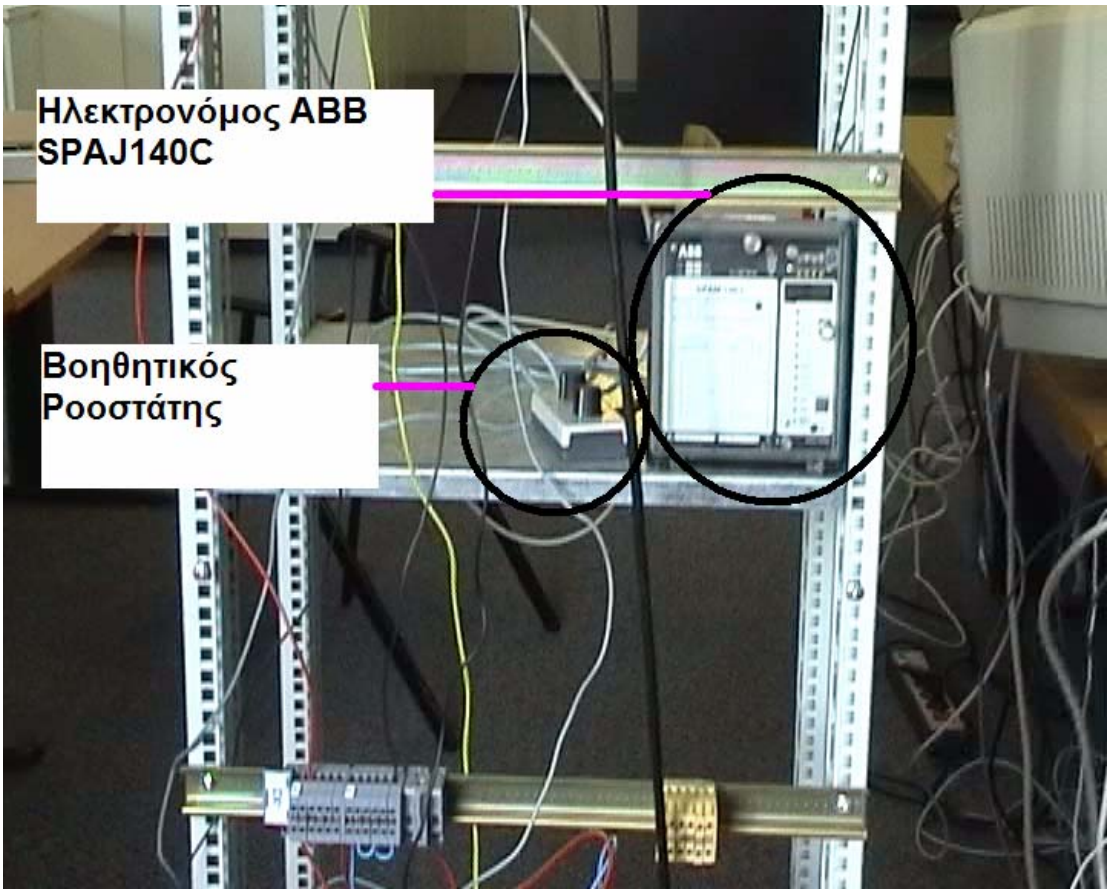
Μέχρι τώρα ο αναγνώστης μπορούσε να φανταστεί την εργαστηριακή διάταξη. Μία εικόνα της δόθηκε στην αρχή του κειμένο, όμως εδώ την χωρίζουμε σε τρία λογικότερα μέρη.

Μέρος 1^{ο}. Διανομή τροφοδοσίας.



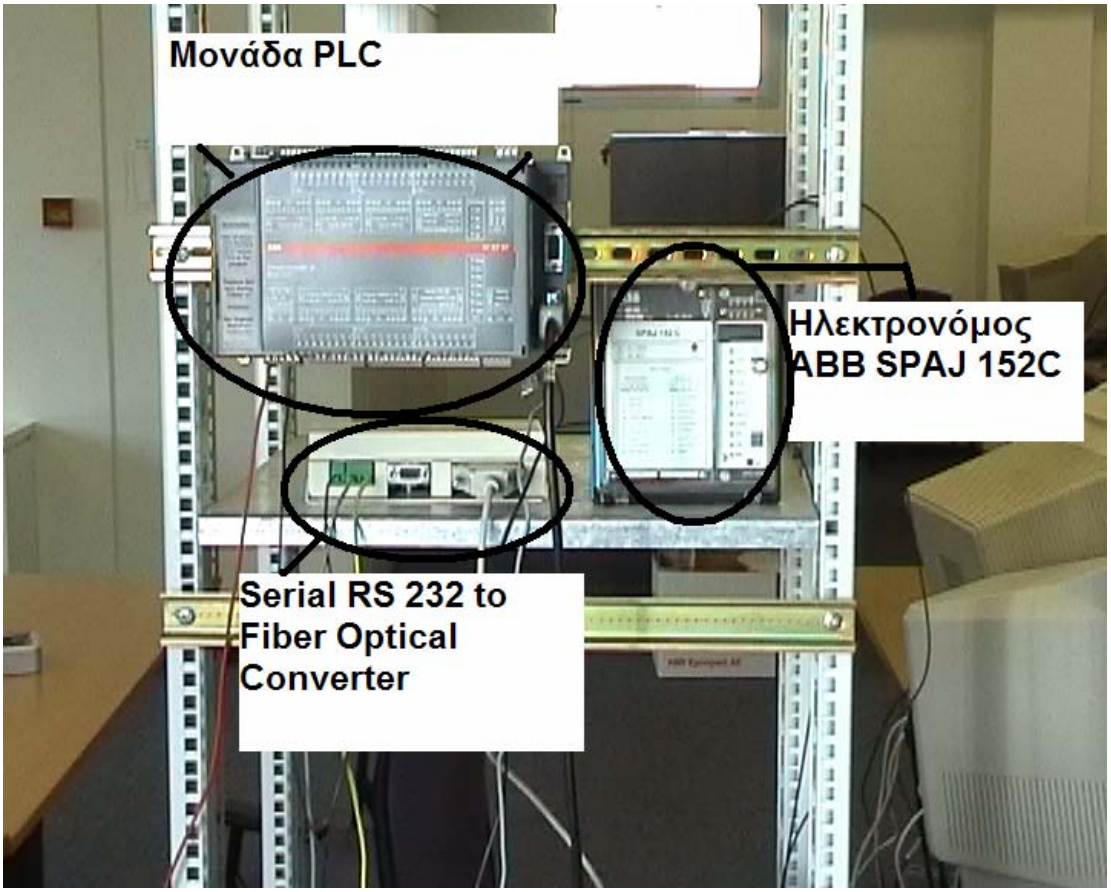
Εικόνα 2. Διανομή της τροφοδοσίας στο πειραματικό κύκλωμα. Αριστερά διακρίνονται οι ασφάλειες προστασίας του PLC ενώ δεξιά ο μετασχηματιστής μετατροπής των 220 σε 24 Volt που το συγκεκριμένο μοντέλο AC1131 (07KT97B) απαιτεί για την λειτουργία του. Οι ηλεκτρονόμοι λειτουργούν απευθείας με εναλλασσόμενη τάση 220Volt.

Μέρος 2^{ο}. Ηλεκτρονόμος με οδηγημένες τιμές



Εικόνα 3. Πάνω από τον τομέα τροφοδοσίας, διακρίνεται ο πρώτος ηλεκτρονόμος της διάταξης. Το ιδιαίτερο σε αυτόν είναι πως είναι συνδεδεμένος στους ακροδέκτες του με έναν ροοστάτη ο οποίος τροφοδοτείται επίσης με 220 Volt. Έτσι μεταβάλλοντας τις ρυθμίσεις του ροοστάτη, θα πρέπει να παρατηρούνται και μεταβολές στις ενδείξεις του ηλεκτρονόμου και φυσικά ανάλογες μεταβολές στις μετρήσεις θα πρέπει να επιτυγχάνει και το τελικό project του συγγραφέα.

Μέρος 3^{ο}. Μετατροπέας σήματος, και PLC



Εικόνα 4. Εδώ επάνω αριστερά φαίνεται το PLC που χρησιμοποιήθηκε για την αποπεράτωση της εργασίας, δεξιά ένας ακόμα ηλεκτρονόμος, ενώ κάτω αριστερά διακρίνεται ο μετατροπέας της επικοινωνίας σειριακής θύρας σε οπτική ίνα. Η λογική συνδεσμολογία αυτών έχει δοθεί παραπάνω.

Διακρίνονται επάνω στο PLC και κάτω αριστερά του, η τροφοδοσία του, ενώ στα δεξιά του και επάνω το σειριακό καλώδιο επικοινωνίας με τους ηλεκτρονόμους, ενώ ακριβώς από κάτω το σειριακό καλώδιο προγραμματισμού του. Εδώ αξίζει να σημειωθεί πως το καλώδιο προγραμματισμού του PLC είναι ειδικό και δεν μπορεί να χρησιμοποιηθεί απλά οποιοδήποτε για την δουλειά αυτή. Επίσης το καλώδιο επικοινωνίας με τους ηλεκτρονόμους ήταν μία διαδικασία από μόνο του. Και οι δύο θύρες εισόδου/εξόδου του PLC είναι των 9 ακροδεκτών. Η μόνη θύρα εισόδου/εξόδου του μετατροπέα όμως είναι των 25pin. Αφού ένας απλός μετατροπέας 9 σε 25 ακροδέκτες δεν απέδωσε και έτσι άλλη αιτία έλλειψης επικοινωνίας άρχισε να αναζητείται. Μετά από μέρες ο συγγραφέας διαπίστωσε πως προκειμένου το PLC να προστατευθεί από τυχόν βραχυκύκλωμα, κάποιες λειτουργίες κάποιων ακροδεκτών είχαν αντιστραφεί. Έτσι για να συνεργαστεί το σύνολο με άλλη σειριακή συσκευή έπρεπε πρώτα το σήμα αυτό να ξανά αντιστραφεί.

Ακριβώς κάτω από το PLC διακρίνεται η συνδεσμολογία του μετατροπέα. Στα δεξιά του ένα σειριακό καλώδιο 25 ακροδεκτών, είναι εφαρμοσμένο στην αντίστοιχη είσοδό του, ενώ αριστερά διακρίνονται τα καλώδια τροφοδοσίας του. Οι ακροδέκτες των καλωδίων οπτικών ινών βρίσκονται στην πίσω πλευρά του μετατροπέα.

ΑΝΑΛΥΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Όπως είναι ήδη γνωστό, ο σκοπός της εργασίας ήταν η επίτευξη επικοινωνίας μεταξύ του PLC ABB AC31 και των ηλεκτρονόμων της σειράς SPA μέσω σειριακής θύρας. Οι δύο αυτές σειρές προϊόντων αποτελούν low ends της ABB και ο συνδυασμός τους θα αποτελούσε μια καλή λύση για την Ελληνική αγορά.

Αφού ανατέθηκε στον συγγραφέα να μελετήσει το SPA Bus protocol manual, οι εργασίες που έπρεπε να φέρει εις πέρας ήταν οι εξής:

Στάδιο 1^{ο}:

Επίτευξη επικοινωνίας ενός ηλεκτρονόμου μοντέλου SPAJ 140C με ηλεκτρονικό υπολογιστή μέσο του προγράμματος 'Hyper Terminal'. Σκοπός ήταν όχι μόνο η επιτυχής επικοινωνία μεταξύ ηλεκτρονόμου και ηλεκτρονικού υπολογιστή αλλά και η συνεχής επιτυχημένη ανταλλαγή σωστών μηνυμάτων. Αυτός ο στόχος επιτεύχθηκε σχετικά εύκολα αφού και το SPA Bus protocol manual είναι ένα εγχειρίδιο περίπου πενήντα σελίδων και απολύτως κατανοητό ακόμα και για κάποιον που πρώτη φορά ασχολούταν με την τεχνολογία των ηλεκτρονόμων, αλλά και σχετική γνώση υπήρχε στους συναδέλφους στην εταιρεία σχετικά με την επίλυση πρακτικών ζητημάτων.

Στάδιο 2^{ο}:

Το στάδιο αυτό αποτελούνταν από δύο σκέλη. Το τελικό ζητούμενο ήταν η επίτευξη επικοινωνίας του PLC 07KT97 με τον ηλεκτρονικό υπολογιστή μέσο του προγράμματος 'Hyper Terminal'.

Η πρώτη φάση είχε ως σκοπό τον σχεδιασμό και την κατασκευή ενός προγράμματος, που θα επικοινωνούσε με τον ηλεκτρονικό υπολογιστή αποστέλλοντας αρχικά ένα μικρό κείμενο της επιλογής του χρήστη, το οποίο και ο ηλεκτρονικός υπολογιστής θα λάμβανε με επιτυχία σε ικανοποιητικό χρονικό διάστημα, και στη συνέχεια ενός προγράμματος που θα μπορούσε να λάβει ένα μήνυμα που θα έστελνε ο χρήστης του ηλεκτρονικού υπολογιστή στο PLC μέσο του 'Hyper Terminal', αναλλοίωτο και επίσης σε ικανοποιητικό χρόνο. Αφού και τα δύο αυτά επιτεύχθηκαν, το στάδιο αυτό περνούσε στην δεύτερη φάση του.

Η δεύτερη φάση είχε ως στόχο την συνένωση των δύο προηγούμενων υποπρογραμμάτων που θα λειτουργούσαν πλέον ως ένα και ο χρήστης του ηλεκτρονικού υπολογιστή αλλά και του PLC θα μπορούσαν πλέον να αποστείλουν αλλά και να λάβουν ακέραια μηνύματα και προς τις δύο κατευθύνσεις. Όσο απλό και αν ακούγεται αυτό το βήμα ήταν από τα πιο δύσκολα, αφού δημιουργήθηκαν ζητήματα ταυτοχρονισμού για τα οποία ο συγγραφέας δεν είχε απαντήσεις. Γενικά, ο συγγραφέας μετά από πολύ καιρό διαπίστωσε πως το AC1131 programming software υποστηρίζει δύο ειδών δομές (mode) επικοινωνίας: την edge triggered mode και την free mode.

Όλα τα παραδείγματα των manuals ήταν γραμμένα για edge triggered mode, όμως το help και τα manual ήταν γραμμένα με βάση την free mode.

Έτσι συνεχώς παρουσιαζόταν ζήτημα λανθασμένου χρονισμού του δικτύου και ήταν λόγου χάρη δυνατό το πρόγραμμα να ξεκινούσε την συνάρτηση υποδοχής δεδομένων από την θύρα com, όταν δεν είχε στείλει κανείς δεδομένα, ενώ ήταν γνωστό πως τα δεδομένα έπρεπε να φεύγουν από το PLC στην ακμή του ρολογιού, όμως πουθενά δεν αναφερόταν πως ο χρηστής υλοποιεί διαδικασία ρολογιού ή πως καλεί την διαδικασία ρολογιού εάν αυτή υπήρχε. Το όλο πρόβλημα προέκυπτε από το γεγονός πως για τα υπόλοιπα πρωτόκολλα επικοινωνίας που είναι πολύ ανώτερα σε ταχύτητα και δυνατότητες και χρησιμοποιούνται κατά κόρον στις εφαρμογές αυτοματισμού (λ.χ. MODBUS και PROFIBUS), το AC1131 programming software, έχει έτοιμα τα settings αλλά και την υλοποίηση των πακέτων αυτών. Αντίθετα για το SPA Bus δεν υπήρχε τίποτα τέτοιο προφανώς επειδή καλύπτει χαμηλότερες ανάγκες με χαμηλότερο κόστος γεγονός χρήσιμο για την αγορά της Ελλάδας. Έτσι μετά από πολύ προσπάθεια τα δύο προγράμματα ενώθηκαν επιτυχώς

Στάδιο 3^{ο}:

Στο στάδιο αυτό το πρόγραμμα που προέκυψε από το προηγούμενο στάδιο θα έπρεπε να προσαρμοστεί έτσι ώστε να δουλέψει επιτυχώς σε ένα δίκτυο PLC-SPAJ 140C. Αφού είχε διαπιστευτεί η λειτουργία και των δύο components με τον μεσολαβητή που ήταν ο ηλεκτρονικός υπολογιστής, καιρός ήταν ο μεσολαβητής αυτός να φύγει και να λειτουργήσει το σύστημα αυτόνομο. Οι δυσκολίες του σταδίου αυτού ήταν πως εφόσον το AC1131 programming software δεν υλοποιούσε το πρωτόκολλο επικοινωνίας SPA Bus, ο συγγραφέας έπρεπε να το γράψει από την αρχή, προκειμένου ο ηλεκτρονόμος- slave να αναγνωρίζει τα εξερχόμενα μηνύματα από το PLC- master. Σε αυτό το στάδιο λύθηκαν μετά από πολύ δουλειά και όλα τα προβλήματα συγχρονισμού καθώς και τα προβλήματα overflow του προηγούμενου βήματος.

Στη συνέχεια παρατίθεται ο κώδικας που αναπτύχθηκε στο στάδιο αυτό.

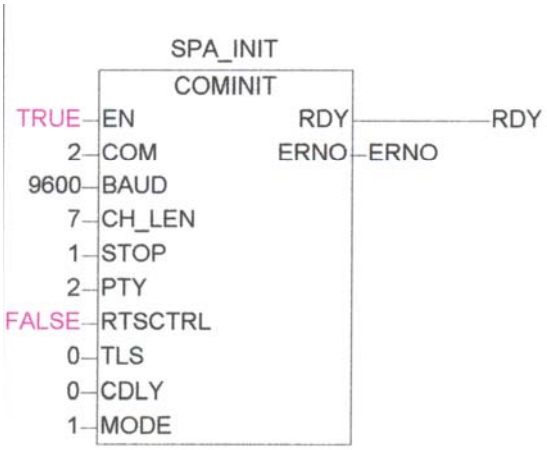
PROGRAM INITIALIZATION

ΔΗΛΩΣΕΙΣ

```
VAR_INPUT
        ENINIT              :BOOL;
    COM                     :INT:=2;
END_VAR
 VAR
```

```
        COM_INIT: COMINIT;          (*      TYPE      OF
PROCEDURE*)
END_VAR
VAR_OUTPUT
        RDY: BOOL;          (*COMINIT READY STATE. *)
        ERNO: INT;          (*COMINIT ERROR NUMBER*)
END_VAR
```

ΚΩΔΙΚΑΣ



FUNCTION_BLOCK MASTERMESSAGE

ΔΗΛΩΣΕΙΣ

```
VAR_INPUT
        SNUM            :STRING;        (*NUMBER      OF
SLAVE*)
        MTYPE: STRING;  (*MESSAGE TYPE*)
        CNUM: STRING;   (*CHANNEL NUMBER (OF SLAVE)*)
        DCAT: STRING;   (*DATA CATEGORY (R/W/D...)*)
        DNUM: STRING;   (*NUMBER OF DATA*)
        DATA: STRING;   (*THE ACTUAL DATA*)
```

```
        CSUM: STRING;      (*CHECK SUM*)
END_VAR

VAR_OUTPUT
        OPV: STRING;
END_VAR

VAR

        copv: STRING;
        copv1: STRING;
        copv2: STRING;
        copv3: STRING;

        copv4: STRING;
        copv5: STRING;
        copv6: STRING;

END_VAR
```

ΚΩΔΙΚΑΣ

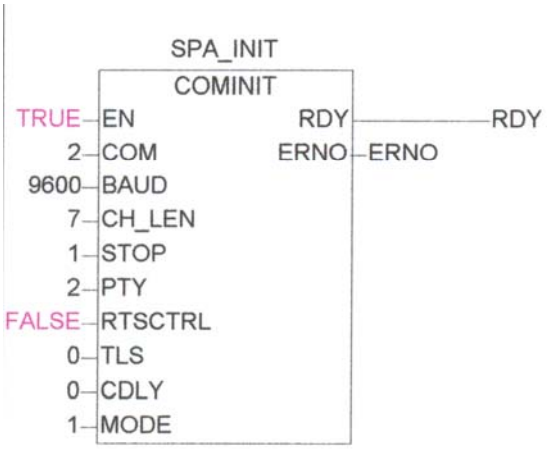
```
        copv            :=CONCAT('>',SNUM);
        copv1           :=CONCAT(MTYPE,CNUM);
        copv2           :=CONCAT(copv,copv1);
        copv3           :=CONCAT(DCAT,DNUM);
        copv4           :=CONCAT(copv2,copv3);
        copv5           :=CONCAT(DATA,CSUM);
        copv6           :=CONCAT(copv4,copv5);
        OPV             :=(copv6 );
```

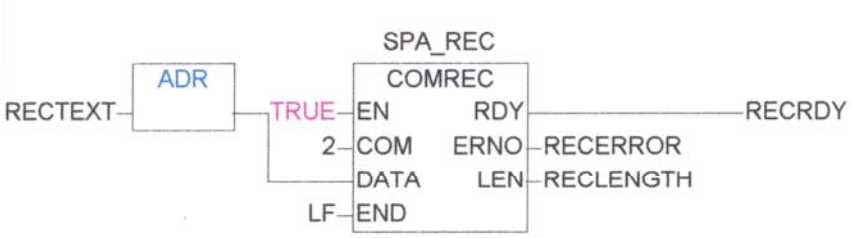
```
PROGRAM PLC_PRG
```

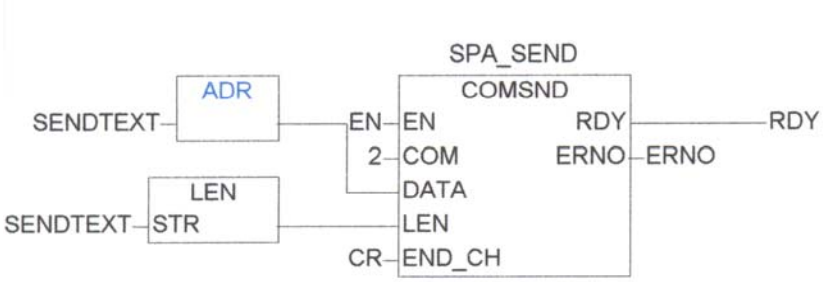
ΔΗΛΩΣΕΙΣ

```
VAR
        FINAL                   :STRING;
        SENDRDY         : BOOL;
        (*SEND READY STATE*)
```

```
        SENDERNO        : INT;
        (*SEND ERROR NUMBER IN MAIN PROGRAM*)
        SLNUM           : STRING:='1';
        (*SLAVE NUMBER*)
        MSGTYPE         : STRING:='RF';
        (*MESSAGE TYPE*)
        CHNUM           : STRING:=':';
        (*CHANNEL NUMBER*)
        DATCAT          : STRING:='XX';
        (*DATA CATEGORY*)
        DATNUMBER       : STRING;
        (*DATA NUMBER*)
        ADATA                   : STRING;
                        (*ACTUAL DATA*)
        CHSUM           : STRING;
        (*CHECK SUM*)
        OUTPUT          : MASTERMESSAGE;        (*
THE CONSTRUCTION OF OUTPUT*)
        COM2: INT := 2;
        END_VAR

VAR_OUTPUT

        RECEIVEDTEXT: STRING;   (*RECEIVED   TEXT
FROM SLAVES*)
        RECEIVERDY: BOOL:=TRUE;         (*RECEIVE  RDY
BOOLEAN (FOR MAIN PROGRAM)*)
        RECORDERROR: INT := 0;


END_VAR
```







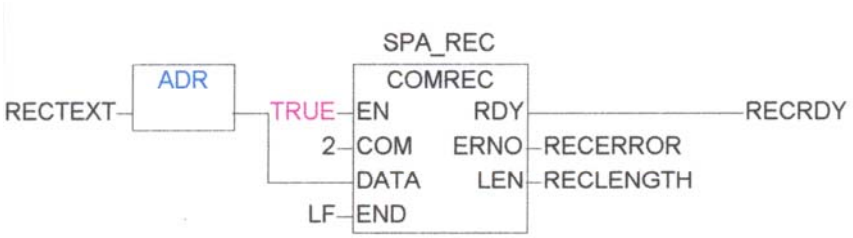
PROGRAM RECEIVE

ΔΗΛΩΣΕΙΣ
VAR

COM_REC :COMREC;
COM : INT:=2;
(*COMMUNICATION PORT (IN THE INIT STEP ALLREADY SET AS 1)*)
RECLENGTH : INT; (*COUNTER WHICH STORES THE LENGTH OF RECEIVED MESSAGE*)
END_VAR
VAR_OUTPUT
RECTEXT : STRING; (*RECORDED TEXT*)

```
        RECRDY          : BOOL;          (*BOOLEAN
USED TO COMPARE THE READY STATE OF THE COM_REC*)

        RECERROR: INT := 0;              (*RECORD        ERROR
VALUE*)
END_VAR
VAR_INPUT
        EN: BOOL:=TRUE;   (*ENABLING OF STARTING INPUT*)
END_VAR
```

ΚΩΔΙΚΑΣ



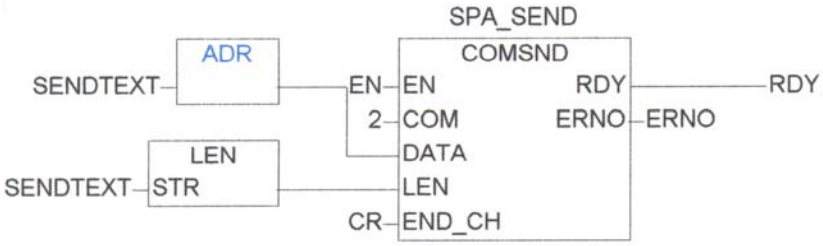
SEND

ΔΗΛΩΣΕΙΣ

```
PROGRAM SEND
VAR
        COM_SEND: COMSND;          (*TYPE OF DEVICE*)
END_VAR
VAR_INPUT
        SENDEN: BOOL:=TRUE;        (*  ENABLING  SIGNAL
FOR THE SEND FUNCTION*)
        SENDTEXT: STRING;          (* TEXT TO BE SENT*)
END_VAR
VAR_OUTPUT
        RDY: BOOL;        (*OUTPUT       BOOLEAN       FOR
READINESS OF FUNCTION*)
        ERNO: INT;        (*ERROR NUMBER. OUTPUT*)
```

END_VAR

ΚΩΔΙΚΑΣ



Παρατίθενται και οι Global μεταβλητές οι οποίες κρίθηκαν χρήσιμες στην φάση αυτή.

```
VAR_GLOBAL
        COM_INIT        :COMINIT;
        MAINEN          : BOOL;         (*      GLOBAL
BOOLEAN ENABLING ALL DEVICES*)

        COMRDY          : BOOL;         (*BOOLEAN   FOR
THE TOTAL COMMUNICATION PACKAGE*)

        COMERNO         : INT;          (*ERROR NUMBER FOR
THE TOTAL COMMUNICATION PACKAGE*)
        STEP            :INT;

        CR              : WORD := 16#0D;    (*
        TRANSLATED TO  CR = 0Dhex = $R *)

        CR_LF           : WORD := 16#0D0A;      (*
        TRANSLATED TO CR/LF = 0D0Ahex = $R$L *)
```

```
        LF                      : WORD := 16#0A;            (*
TRANSLATED TO   LF   = 0Ahex =$L$F*)

        BLINKSEND  AT %MX255.1          :BOOL;
                                (*      Blinktakt 1 sec *)
        BLINKREC    AT %MX255.0         :BOOL;

END_VAR
```

Σχολιασμός του κώδικα.

Ο κώδικας αποτελείται από πέντε objects.

Το πρώτο action είναι το λεγόμενο initialization. Σε αυτό καλείται η συνάρτηση cominit η οποία είναι υπεύθυνη στο να εγκαθιστά επικοινωνία μεταξύ μιας σειριακής θύρας του PLC και μιας άλλης συσκευής. Εδώ καθορίζονται πληροφορίες όπως το comport, το parity, η ταχύτητα, κ.τ.λ.

Το δεύτερο object, ονομάζεται mastermessage. Δουλειά του είναι να παίρνει το κείμενο που θέλει να στείλει το PLC, και να το συναρμολογεί έτσι ώστε τα πάντα να βρίσκονται στη σωστή θέση και το τελικό μήνυμα να είναι αναγνώσιμο από τον ηλεκτρονόμο.

Το τρίτο σκέλος είναι το main program. Αυτό πρέπει αναγκαστικά να ονομάζεται PROGRAM PLC_PRG. Αποτελείται από πολλά χωριστά function blocks, τα οποία είναι προγραμματισμένα σε FBD. Στην ουσία καλούνται τα υπόλοιπα actions. Το action initialization είναι υποχρεωτικά ανοιχτό συνεχώς.

Το τέταρτο σκέλος, είναι το action εισαγωγής δεδομένων από την θύρα επικοινωνίας, ενώ το πέμπτο είναι το σκέλος αποστολής δεδομένων μέσο της θύρας.

Επίσης παρατίθεται το domain καταχώρησης των global μεταβλητών.

Παρατηρούνται οι μεταβλητές που μετατρέπουν τα lf, cr και crlf σε printable ASCII.

Η μεταβλητή BLINKSEND AT %MX255.1 έχει κατασκευαστεί έτσι ώστε να ανοιγοκλείνει' την action SEND για ένα δευτερόλεπτο κάθε 2.55 δευτερόλεπτα.

Αφού το συγκεκριμένο πρόγραμμα πέτυχε το σκοπό του, μπορούμε να έχουμε επικοινωνία μεταξύ του PLC και των ηλεκτρονόμων. Επόμενος στόχος ήταν η επικοινωνία PLC με ηλεκτρονόμους κάτω όμως από συνθήκη αυτομάτου ελέγχου. Το PLC δηλαδή θα ελέγχει και θα καταγράφει τιμές και events από τους ηλεκτρονόμους και θα τα καταχωρεί σε πίνακες χωρίς την παρουσία χειριστή. Η τελική ιδέα είναι η κατασκευή βιβλιοθήκης από τον συγγραφέα ειδικά για το πρωτόκολλο SPA Bus. Η εργασία από εδώ και πέρα είναι ξεκούραστη και πιο 'βατή' αφού στην αρχή ο συγγραφέας κλίθηκε να χρησιμοποιήσει δύο εντελώς άγνωστα προς αυτόν εργαλεία προγραμματίζοντας τα με μία γλώσσα προγραμματισμού επίσης άγνωστη προς αυτόν.

Στάδιο 4^{ο} .

Το 4^{ο} είναι και το τελικό στάδιο της εργασίας και της ολοκλήρωσης του προγράμματος οδήγησης των ηλεκτρονόμων.

Για να επιτευχθεί η δημιουργία ενός αντικειμένου τέτοιου επιπέδου, ήταν απαραίτητος ο σωστός αρχικός σχεδιασμός. Κακοσχεδιασμένος κώδικας μπορεί να οδηγήσει αργότερα τον συγγραφέα σε σχεδιαστικά αδιέξοδα και στην σημαντική απώλεια χρόνου και εργασίας προκειμένου να κατορθώσει διορθώσει στην πράξη, σχεδιαστικά σφάλματα. Έτσι μετά από διάφορα μοντέλα που προηγήθηκαν ο συγγραφέας κατέληξε σε αυτό που εμφανίζεται στην παρακάτω σελίδα.

Στην συνέχεια θα αναπτυχθεί η ιδέα του τρόπου λειτουργίας του σχεδίου αυτού από το οποίο προέκυψε το πρότυπο πρόγραμμα.

Η όλη ιδέα του προγράμματος ήταν να χωριστούν οι απαιτούμενες διεργασίες προς την επίτευξη του προγράμματος σε δύο κατηγορίες:

Η πρώτη κατηγορία

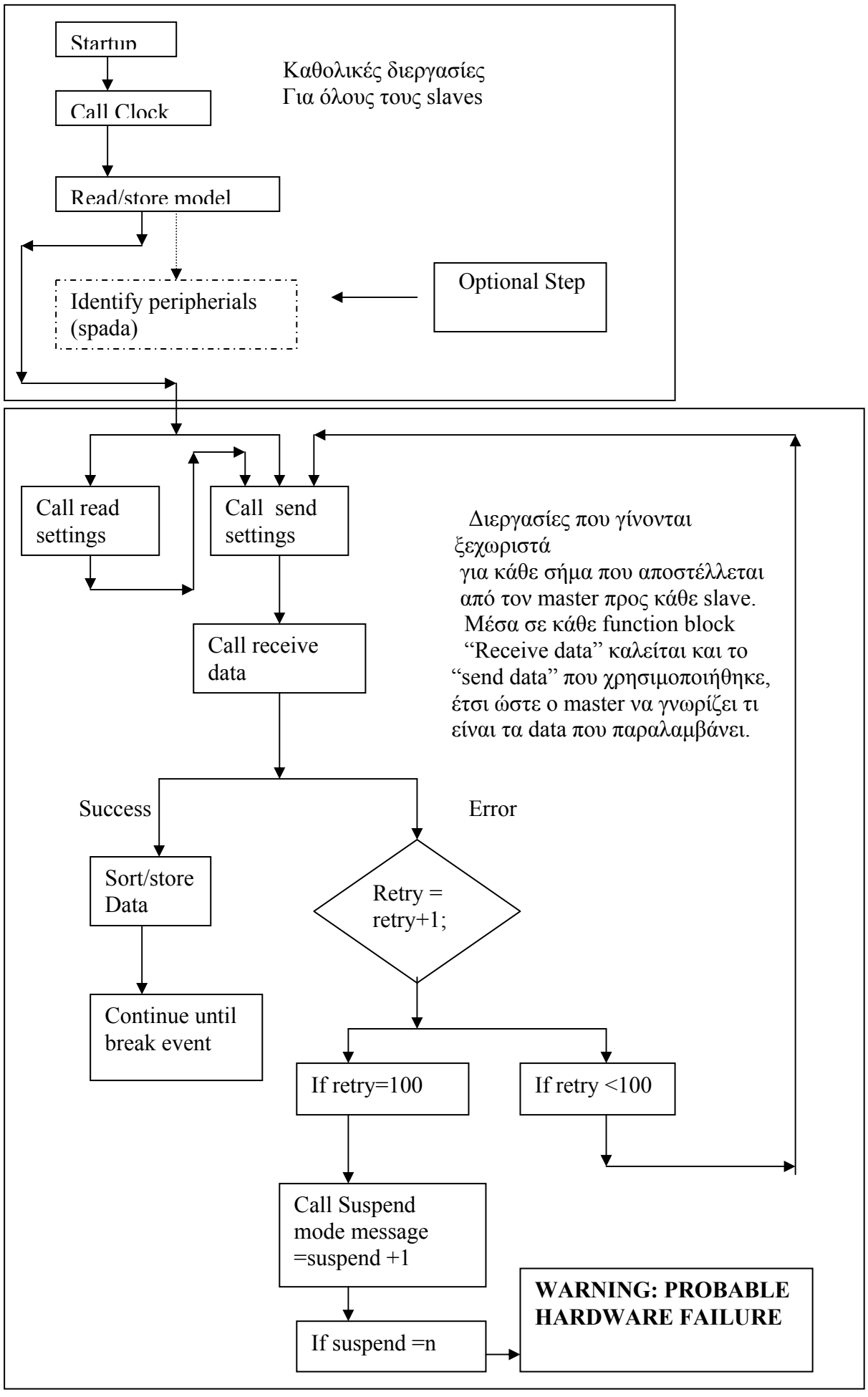
Ασχολείται με διεργασίες οι οποίες θα έπρεπε να γίνουν προκειμένου ο master να γνωρίζει με πόσους και ποιους slaves βρίσκεται σε σύνδεση έτσι ώστε να εδραιώσει επικοινωνία.

Τέτοιες διεργασίες είναι η startup, η call clock και η read/store model.

Στην startup υπάγονται τα διάφορα δεδομένα που πρέπει να φορτωθούν κατά την εκκίνηση του PLC για την αρχικοποίηση των διαφόρων συνθηκών για να διασφαλιστεί η ομαλή λειτουργία του όπως λόγου χάρη το κουτί διαδικασίας COMINIT. Το αντικείμενο και ο τρόπος λειτουργίας της, θα αναπτυχθεί παρακάτω μαζί με όλο το υπόλοιπο μοντέλο.

Στο στάδιο call clock ανήκουν οι διεργασίες που τότε (πριν τη δημιουργία του προτύπου) ο συγγραφέας πίστευε πως θα χρειάζονταν προκειμένου να κληθεί η διαδικασία του ρολογιού (η οποία είναι απαραίτητη σε όλα τα δομημένα προγράμματα) για την ακριβή καταμέτρηση και αποθήκευση των εισερχομένων και εξερχόμενων μηνυμάτων.

Το λογικό κουτί read/store model αντιπροσωπεύει τις διεργασίες που πρέπει να είναι φορτωμένες σε μορφή κουτιών διεργασίας προκειμένου ανά πάσα στιγμή να καλούνται για να εξασφαλίζεται η σωστή ανταλλαγή δεδομένων μεταξύ master και slaves. Όπως και για τις υπόλοιπες διεργασίας, έτσι και για αυτές, η ανάπτυξη και ο τρόπος λειτουργίας τους θα αναπτυχθεί παρακάτω, προκειμένου να διατηρηθεί μια λογική συνέχεια και να αποφευχθεί η μορφή του χαοτικού κειμένου που θα κουράσει τον αναγνώστη.


Startup
Call Clock
Read/store model
Identify peripherials (spada)
Optional Step
Καθολικές διεργασίες
Για όλους τους slaves
Call read settings
Call send settings
Call receive data
Διεργασίες που γίνονται ξεχωριστά
για κάθε σήμα που αποστέλλεται από τον master προς κάθε slave.
Μέσα σε κάθε function block "Receive data" καλείται και το "send data" που χρησιμοποιήθηκε, έτσι ώστε ο master να γνωρίζει τι είναι τα data που παραλαμβάνει.
Success
Error
Sort/store Data
Retry = retry+1;
Continue until break event
If retry=100
If retry <100
Call Suspend mode message =suspend +1
If suspend =n
WARNING: PROBABLE HARDWARE FAILURE

Η δεύτερη κατηγορία

Περιέχει διεργασίες που πρέπει μεν να εκτελεστούν για τους περισσότερους (αν όχι για όλους) slaves, όμως η ιδιαιτερότητα της μορφής των δεδομένων που απαιτούν διαφορετικά στοιχεία και πληροφορία για κάθε slave καθιστά απαραίτητη την επεξεργασία των εισερχομένων και εξερχόμενων από και προς τον master μηνυμάτων.

Στην αρχή διακρίνουμε τα λογικά κουτιά read settings και send settings. Το πρώτο χρησιμοποιείται όταν κριθεί σκόπιμο να διαβαστούν οι προηγούμενες τιμές που ήταν αποθηκευμένες στους slaves/ηλεκτρονόμους, και στην αρχική ιδέα υπήρχε η λογική εάν χρησιμοποιείται, αυτό να γίνεται κατά την εκκίνηση. Υπενθυμίζεται στον αναγνώστη πως εδώ αναλύεται η λογική που υπήρχε στο προσχέδιο του προγράμματος χωρίς αυτό αναγκαστικά να σημαίνει πως οι λύσεις που αναφέρονται ακολουθήθηκαν ή όχι κατά γράμμα. Το δεύτερο είναι στην ουσία αντιπροσώπευε την διαδικασία που θα ακολουθούνταν κατά την απόπειρα αποστολής δεδομένων σε έναν τυχαίο slave.

Receive data. Εδώ σκοπός ήταν η δημιουργία ενός λογικού κουτιού διεργασίας το οποίο θα ανέλυε το εισερχόμενο μήνυμα, και θα αναγνώριζε εάν πρόκειται για σωστό ή μήνυμα λάθους, και αντίστοιχα θα καλούσε δύο άλλες διεργασίας ανάλογα με το αν το μήνυμα ήταν σωστό ή όχι.

Έτσι η sort/store data θα καλούνταν από την receive data εάν το μήνυμα ήταν αληθές. Η sort/store data, θα ήταν υπεύθυνη για την σωστή αρχειοθέτηση, ταξινόμηση του μηνύματος καθώς και για την αποθήκευσή του σε stable storage εάν αυτό κρινόταν επιθυμητό ή αναγκαίο.

Μετά από την διεργασία αυτή υπάρχει η λεγόμενη **continue until break** event. Αυτή η δήλωση απλά σημαίνει πως εδώ θα γινόταν η δημιουργία ενός μηχανισμού, ο οποίος στην περίπτωση που δεν υπήρχε σφάλμα, θα έπρεπε να κρατά το σύστημα σε ομαλή κυκλική κίνηση έτσι ώστε ανά τακτά χρονικά διαστήματα ακόμα και αν δεν υφίσταται η ανάγκη ελέγχου κάποιου συγκεκριμένου slave, να τρέχει ένα προληπτικό data request σε όλους, έτσι ώστε ο μηχανικός ανά συγκεκριμένη μονάδα χρόνου να μπορεί να έχει μια συνολική εικόνα του συστήματος, και στην ουσία αντιπροσωπεύει την φάση ηρεμίας.

Η Retry είναι μια διαδικασία η οποία στην περίπτωση που το εισερχόμενο μήνυμα στην receive data διαπιστωθεί πως είναι αρνητική απάντηση (Not Acknowledged) ή μετά από κάποιο χρονικό διάστημα δεν υπάρξει απάντηση, ξαναστέλνει το εξερχόμενο προς τον slave μήνυμα. Αυτό επαναλαμβάνεται μέχρι να έρθει θετική απάντηση (Acknowledged) από τον εν λόγο slave. Όπως φαίνεται εάν η Retry κλιθεί εκατό φορές χωρίς να έρθει για καμία από αυτές θετικό σήμα (Acknowledged), τότε

καλείται το κουτί διεργασίας Suspend mode, ο ρόλος του οποίου εμφανίζεται παρακάτω.

Suspend mode. Επειδή προφανώς θα ήταν αρκετά επώδυνο από άποψης ταχύτητας και υπερφόρτωσης για το σύστημα να βομβαρδίζεται από μηνύματα προς έναν slave που δεν ανταποκρίνεται, σε αυτό το λογικό κουτί διεργασίας, γίνεται προσπάθεια αποκατάστασης της επικοινωνίας κάθε περίπου δυόμισι δευτερόλεπτα. Εάν και πάλι δεν υπάρχει καμία ανταπόκριση μετά από ένα δεδομένο αριθμό μηνυμάτων, το σύστημα θεωρεί πως για κάποιο άγνωστο λόγο ο slave είναι είτε σβηστός ή εκτός δικτύου οπότε και στέλνει στον χρήστη το μήνυμα:

WARNING: PROBABLE HARDWARE FAILURE ή κάποιο άλλο αντίστοιχης σημασίας, και επιστρέφει στην ομαλή λειτουργία θεωρώντας αυτό το slave ανύπαρκτο. Αξίζει να σημειωθεί πως καθ' όλη την διαδικασία της αντιμετώπισης λάθους το σύστημα δεν παραμένει άπραγο. Οι προγραμματισμένες διεργασίες με τους υπόλοιπους slaves συνεχίζουν κανονικά.

Όπως φαίνεται και στο διάγραμμα, η ομαλή λειτουργία θα συνεχιζόταν ώσπου να παρεμβληθεί κάποιο break event, το οποίο θα μπορούσε να είναι internal (failure ή κάτι αντίστοιχο) ή external (εντολή χειριστή).

Πολλές από τις παραπάνω διεργασίες μετονομάστηκαν ή άλλαξαν μορφή καθηκόντων. Αυτό κρίθηκε αναγκαίο κυρίως για διαφόρους λόγους.

Ο πρώτος είναι η αναγκαιότητα προσαρμογής των ζητούμενων και του τρόπου σχεδιασμού, στις δυνατότητες που παρείχε ο compiler του AC1131.

Ο δεύτερος λόγος είναι πως με κάποιες μικρές αλλαγές (που θα φανούν παρακάτω), ο κώδικας γινόταν πολύ πιο ευέλικτος μέσα στην πραγματική ροή του προγράμματος της εφαρμογής και έτσι τα διάφορα αντικείμενα του προγράμματος (objects) συνεργαζόντουσαν μεταξύ τους αρτιότερα και γρηγορότερα.

Άλλος ένας λόγος, ο οποίος είναι και ένας που θα εμφανίζεται πολύ συχνά, είναι η σε μεγάλο βαθμό διαφορετικότητα του τρόπου λειτουργίας και της αντίληψης και του AC1131 programming software, αλλά και της όλης αντίληψης λειτουργίας ενός PLC τέτοιας γενιάς. Έτσι το PLC κάθε πέντε millisecond επαναεκτελεί όλο το πρόγραμμα οπότε υπάρχει μεγάλος κίνδυνος επανεκκίνησης μιας εφαρμογής του προγράμματος που ήδη τρέχει και δεν έχει τελειώσει ακόμα. Αυτό και άλλα πολλά θέματα που έχουν να κάνουν με την φύση της εκτέλεσης ενός προγράμματος για PLC θα αναφερθούν όταν θα χρειαστεί.

Έτσι λοιπόν η οργάνωση της μορφής του κανονικού προγράμματος διέπεται από μια διαφορετική με τον αρχικό σχεδιασμό μορφή, η οποία κατά την άποψη του συγγραφέα, ήταν λειτουργικότερη και εξυπηρετεί καλύτερα τους σκοπούς της εργασίας. Ο τρόπος ανάπτυξης και επεξήγησης του κώδικα θα ακολουθεί γενικά την παρακάτω σειρά.

1. Επωνυμία αντικειμένου
2. Γενικές επεξηγήσεις γύρω από το είδος του αντικειμένου
3. Τρόπος λειτουργίας του αντικειμένου
4. Απαραίτητες διευκρινήσεις σε εύλογες ερωτήσεις που πιθανόν να δημιουργηθούν στον αναγνώστη (λόγου χάρη «γιατί έτσι και όχι αλλιώς;»)

Αρχικές διευκρινήσεις.

Το αρχικό σχέδιο, όπως φαίνεται περιείχε μόνο κομμάτια λογικών διεργασιών (function blocks) και αυτό γιατί σύμφωνα με τις οδηγίες που ο συγγραφέας είχε λάβει, θα έπρεπε τα κομμάτια των λογικών αυτών διεργασιών (function blocks) να μπορούν να ενσωματωθούν σε μια βιβλιοθήκη αναγνώσιμη από τον compiler του AC1131, για μελλοντική εύκολη χρήση από την εταιρία. Έτσι ο συγγραφέας προσπάθησε στην αρχή να δημιουργήσει ένα πρότυπο (πρώιμη σειρά Big ABB project) του οποίου όλες οι διεργασίες να είναι λογικά κομμάτια (function blocks). Αυτό προσέδιδε μια ομοιομορφία στο πρόγραμμα γεγονός ικανοποιητικό για κώδικα, όμως σύντομα ο συγγραφέας διαπίστωσε πως αυτό ήταν αδύνατο να ισχύσει για ολόκληρο το project καθ' ότι τα λογικά κομμάτια διεργασιών (function blocks) στερούνταν δυνατοτήτων έναντι άλλων μορφών αντικειμένων του προγράμματος. Σε πολλές περιπτώσεις διαδικασιών κρίνονταν μέχρι και δύσχρηστα όταν ο συγγραφέας για μία απλή κλίση ρουτίνας έπρεπε να συντάξει κώδικα που πολλές φορές ξεπερνούσε και του εκατό χαρακτήρες.

Αυτό έκανε το σύστημα ακόμα και στο επίπεδο της προσομοίωσης (simulation) πολύ αργό. Επιπλέον δημιουργήθηκαν και φαινόμενα αστάθειας καθώς στο περίπου δέκα πέντε επί τοις εκατό των κλίσεων (οκτώ φορές στις πενήντα κλίσεις)του προγράμματος στον προσομοιωτή, παρουσιάστηκε φαινόμενο overflow εξαιτίας των πολλών δεδομένων. Στο τελικό στάδιο της ιδέας αυτής, τα συνολικά δεδομένα πριν την απόπειρα τρεξίματος έφτασαν το απαγορευτικό ποσό των τετρακοσίων (400 Kbyte) γεγονός που έκανε το πρόγραμμα εξωφρενικά δύσχρηστο, χωρίς καν αυτό να έχει τελειώσει.

Κατά της εκτίμηση του συγγραφέα ένα τέτοιο πρόγραμμα θα οδηγούσε σε πάρα πολύ συχνά overflow λόγω μεγέθους και στην ουσία θα ήταν άχρηστο. Έτσι, πολλά κομμάτια λογικών διεργασιών αλλάχθηκαν σε δεύτερη φάση με αντικείμενα άλλου τύπου τα οποία μπορούσαν να ανταπεξέλθουν καλύτερα σε αυτές τις εργασίες. Τελικά όμως ο συγγραφέας εφαρμόζοντας μια Divide n' Conquer λογική κατάφερε να ανάγει όλα τα αντικείμενα σε FUNCTION BLOCKS μικρότερου μεγέθους. Έτσι μπορεί για την επίτευξη ενός σκοπού να χρειάζεται πολλές φορές η κλίση δύο ή τριών function blocks, όμως πλέον γίνεται λόγος για ένα καλοσχεδιασμένο, ταχύτατο, ευέλικτο, και πάνω από όλα σταθερό κώδικα.

ΑΝΑΠΤΥΞΗ ΓΕΝΙΚΩΝ ΕΡΓΑΛΕΙΩΝ

Αντικείμενο.

Η λέξη αντικείμενο, στον κώδικα AC1131 αντιπροσωπεύει ένα αυτόνομο σκέλος του όλου project το οποίο εκτελεί μια συγκεκριμένη εργασία, ανάλογα με το πως αυτό θα προγραμματιστεί. Υπάρχουν τρεις τύποι αντικειμένων: τα λογικά κομμάτια διεργασιών (function blocks), οι συναρτήσεις (functions) και τα προγράμματα (programs). Οι δυνατότητες του καθενός θα αναπτυχθούν αμέσως μετά.

Λογικά κομμάτια διεργασιών (function blocks).

Τα αντικείμενα αυτά είναι ιδανικά για τον προγραμματισμό αυτόνομων κομματιών κώδικα τα οποία θα πρέπει να καλούνται σχετικά συχνά μέσα σε ένα πρόγραμμα. Αποδίδουν καλύτερα (το μέγιστο των δυνατοτήτων τους) όταν ο σχεδιασμός γίνεται σε σχηματικό κώδικα (FDB), επειδή δημιουργούν ένα πολύ σταθερό μοντέλο κώδικα το οποίο πλησιάζει ταχύτητες hardware. Σε πολύπλοκους κώδικες όμως το σχηματικό μοντέλο προγραμματισμού μπορεί να γίνει πολύ κουραστικό και δύσχρηστο για τον συγγραφέα. Επίσης όταν ένα πρόγραμμα μέσα σε ένα project καλεί ένα function block, τότε πρέπει να δηλώνονται όλες οι μεταβλητές που έχουν δηλωθεί στο function block αυτό σαν είσοδοι και έξοδοι. Το αποτέλεσμα είναι πως εάν ένα function block έχει γίνει μεγάλο και χρησιμοποιεί πολλές μεταβλητές, όλες αυτές οι μεταβλητές θα πρέπει να δηλώνονται στο πρόγραμμα που το καλεί μαζί με άλλες τόσες οι οποίες πιθανότατα θα φέρουν τοπικά τις επιθυμητές τιμές για τις μεταβλητές αυτές. Έτσι εάν θέλουμε να καλέσουμε ένα function block με δεκαπέντε μεταβλητές, θα πρέπει να δηλώσουμε στο πρόγραμμα μας για αυτό το σκοπό περίπου 25 μεταβλητές. Η συνέπεια μιας τέτοιας ενέργειας είναι χαμένος χώρος σε data, περίπου 100 και πάνω χαρακτήρες για μία απλή

κλίση ενός function block και το κυριότερο, κουραστικός κώδικας για τον συγγραφέα και τον μελλοντικό αναγνώστη. Από την άλλη όμως ένα καλοσχεδιασμένο και μικρό function block αποτελεί τον σταθερότερο και γρηγορότερο τρόπο υλοποίησης κώδικα. Αυτό οφείλεται κυρίως στο ότι το PLC ABB 07KT97 παρέχει στον προγραμματιστή την δυνατότητα, να αποθηκεύσει τον κώδικα των λογικών κομματιών διεργασιών (function blocks), σε φυσική Flash EEPROM που το PLC διαθέτει εσωτερικά, ειδικά για αυτό το σκοπό, και τα δεδομένα που είναι έξοδοι των λογικών κομματιών διεργασιών (function blocks), σε Registers, που επίσης το PLC διαθέτει για αυτόν αλλά και άλλους σκοπούς. γεγονός που ανεβάζει την ταχύτητα αλλά και την ευστάθεια του προγράμματος σε πάρα πολύ υψηλά επίπεδα.

Για τους παραπάνω λόγους, τα function blocks πρέπει να χρησιμοποιούνται με μικρά, ευέλικτα κομμάτια κώδικα, και κατά προτίμηση προγραμματισμένα σχηματικά. Σε αυτή την μορφή είναι πολύ δυνατά και αποτελεσματικά.

Συναρτήσεις (functions).

Οι συναρτήσεις έχουν τελείως διαφορετικό τρόπο λειτουργίας. Μια συνάρτηση εκτελεί μια λογική διεργασία και επιστρέφει μία τιμή μόνο σαν έξοδο. Όταν καλούμε μια συνάρτηση δηλώνουμε τις εισόδους και αυτή μας δίνει το αποτέλεσμα της μιας και μοναδικής εξόδου. Πάρα πολύ καλή μορφή αντικειμένου για εκτέλεση ισχυρών μαθηματικών πράξεων. Δεν είναι ιδιαίτερα ευέλικτη λόγω του γεγονότος της επιστροφής ενός μόνο αποτελέσματος, όμως είναι πολύ γρήγορή σε αυτό που κάνει. Πολλά παραδείγματα κλίσεως έτοιμων συναρτήσεων από άλλες ήδη υπάρχουσες βιβλιοθήκες θα δοθούν κατά την διάρκεια της ανάπτυξης του κώδικα (ο συγγραφέας εδώ υπενθυμίζει πως σε αυτήν την παράγραφο αναφέρονται απλά επιγραμματικά τα διάφορα είδη αντικειμένων) .

Πρόγραμμα (program).

Τα προγράμματα αποτελούν την πλέον ευέλικτη μορφή αντικειμένων. Μόνο ένα πρόγραμμα έχει δικαίωμα να καλέσει άλλο πρόγραμμα. Επίσης κάθε πρόγραμμα με μία μικρή δήλωση μπορεί να καλέσει μεταβλητή άλλου προγράμματος οποιαδήποτε στιγμή, (ακόμα και τοπικές μεταβλητές). Για να κλιθεί ένα πρόγραμμα απαιτείται μόνο η αναφορά τους ονόματος του προγράμματος το οποίο πρέπει να κλιθεί, χωρίς άλλες διευκρινήσεις.

Τα προγράμματα είναι η καλύτερη μορφή αντικειμένου για δημιουργία μεγάλου και πολύπλοκου κώδικα, καθώς και για κώδικα σε συμπεριφεριακή μορφή (ST, structured text). Μειονεκτήματα των προγραμμάτων είναι πως δεν γίνεται να αποθηκευτούν σε βιβλιοθήκες για μελλοντική χρήση και είναι σαφώς πιο αργά στην εκτέλεσή τους από ένα λογικό κομμάτι διεργασίας (function block) που εκτελεί την ίδια εργασία.

Σύμφωνα με τα παραπάνω στοιχεία, ο συγγραφέας όποτε μπορούσε κατασκεύασε αυστηρά δομημένο και συγκεκριμένο κώδικα έτσι ώστε να μπορέσει να τον ανάγει σε function blocks, προσπάθησε να χρησιμοποιήσει functions όποτε αυτό γινόταν δυνατό, και δημιούργησε τις κυρίως διεργασίες και τα πολύπλοκα κομμάτια αρχικά κατασκευάστηκαν με την δημιουργία προγραμμάτων έτσι ώστε να ελεγχθεί με ένα ευέλικτο σχήμα η εγκυρότητα του εκάστοτε project, και στην συνέχεια ο συγγραφέας εστίασε την προσπάθειά του στο να μετατρέψει τον μακρύ και πολυσύνθετο κώδικά των προγραμμάτων σε πολλά μικρά και ευέλικτα λογικά κομμάτια διεργασιών (function blocks), έτσι ώστε το τελικό project να αποτελείται από πολλά μικρά και ευέλικτα module έτσι ώστε το όλο project να υλοποιείται με τον καλύτερο δυνατό τρόπο.

Από εδώ και κάτω θα αναλυθεί ο τρόπος με τον οποίο λειτουργεί το τωρινό πρότυπο. Ο τρόπος αυτός θα δοθεί πάντα σε σύγκριση με το παλαιότερο πρότυπο το οποίο είχε τελειώσει γύρω στα μέσα του Ιουνίου του έτους 2004. Έτσι θα φανούν οι βασικές διαφορές που έλαβαν χώρα από την πραγματοποίηση μέχρι την τελειοποίηση του κώδικα, καθώς και όπου θεωρείται σημαντικό θα δοθούν και οι λόγοι που έκαναν τις αλλαγές αυτές επιτακτικές. Όλες οι παραπάνω διευκρινήσεις δόθηκαν επειδή ο συγγραφέας έκρινε απαραίτητο πως ο αναγνώστης πρέπει να γνωρίζει αυτά τα ζητήματα, έτσι ώστε να μπορεί να κρίνει το project για να μπορεί να σχηματίσει μια δική του άποψη γύρω από αυτό ανά πάσα στιγμή.

Περισσότερες διευκρινήσεις θα δοθούν κατά την στιγμή της επεξήγησης του κάθε σκέλους του κώδικα έτσι ώστε ο αναγνώστης να αναπτύξει μια συγκεκριμένη και σε βάθος εικόνα για το όλο της λειτουργίας του κώδικα.

ΕΠΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

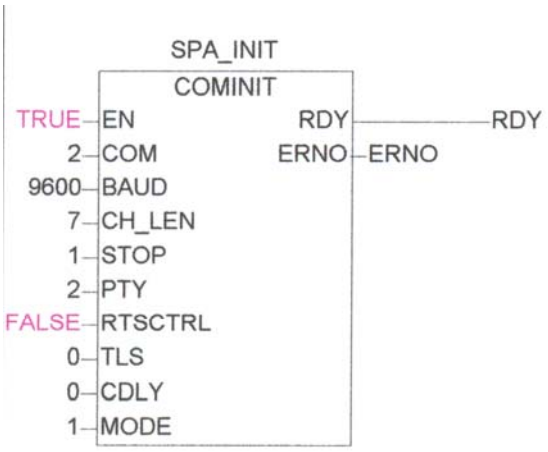
ΛΙΣΤΑ ΑΝΤΙΚΕΙΜΕΝΩΝ

SPA_INIT

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK SPA_INIT

VAR_INPUT

END_VAR

 VAR

  SPA_INIT: COMINIT;                (* TYPE OF PROCEDURE*)

END_VAR


VAR_OUTPUT

        RDY: BOOL;              (*COMINIT READY STATE. *)
  ERNO: INT;            (*COMINIT ERROR NUMBER*)

END_VAR
```

ΚΩΔΙΚΑΣ



Το αντικείμενο αυτό είναι ένα λογικό κομμάτι διεργασίας (function block). Στις παλαιότερες εκδόσεις ο συγγραφές το είχε ονομάσει INITIALIZATION. Ο σκοπός της μετονομασίας του σε SPA_INIT θα φανεί παρακάτω. Καλεί το function block COMINIT που ανήκει στην βιβλιοθήκη Com_S90_V41.lib. Η βιβλιοθήκη αυτή περιέχει έτοιμα function blocks που είναι σχετικά με σειριακή επικοινωνία. Η COMINIT είναι υπεύθυνη για την αρχικοποίηση σειριακής επικοινωνίας. Περιέχει όλες τις πληροφορίες που αφορούν το δίκτυο όπως δίαυλο επικοινωνίας, (COM port), parity, ταχύτητα επικοινωνίας(baud rate) κ.τ.λ.

Οι είσοδοι του function block INITIALIZATION ήταν μόνο μία. Η ENABLE. Η μεταβλητή αυτή είναι κατηγορίας BOOL (BOOLEAN) και στην ουσία δηλώνει το κατά πόσο ο χρήστης θέλει το function block να λειτουργεί ή όχι. Στην επόμενη έκδοση, SPA_INIT, ο συγγραφέας απλά έθεσε στην ENABLE την τιμή 'TRUE' γιατί μόνο έτσι ήταν δυνατό να υπάρχει συνεχής επικοινωνία, και επίσης έτσι η σχεδίαση γλίτωνε τον δεσμευμένο χώρο από μια άχρηστη μεταβλητή.

Οι δύο έξοδοι είναι οι RDY και ERNO συντομογραφίες για READY και ERROR NUMBER αντίστοιχα. Η πρώτη είναι τύπου BOOLEAN και εκεί το Function block δηλώνει εάν είναι έτοιμο για λειτουργία, ενώ η δεύτερη είναι τύπου INT (INTEGER) και εκεί το function block επιστρέφει τον αριθμό σφάλματος, ενώ επιστρέφει '0' για κανένα σφάλμα.

Ο κώδικας είναι γραμμένος σχηματικά (FBD, function block diagram) γιατί έτσι είναι πιο καθαρός και ευανάγνωστος για τον συγγραφέα, ενώ ως function block ανταποκρίνεται και πιο γρήγορα και σταθερά.

Οι μεταβλητές μέσα στο λογικό κουτί είναι οι είσοδοι που function block ενώ οι εξωτερικές τιμές είναι αυτές που ο χρήστης τους δίνει. Αυτές μπορεί να παίρνουν τις τιμές τους από σταθερές ή από άλλες μεταβλητές. Στην περίπτωση του SPA_INIT όλες οι είσοδοι είναι ήδη αρχικοποιημένες έτσι ώστε να μην χρειάζεται δέσμευση χώρου σε μεταβλητές καθώς και η συνεχής αναφορά στις τιμές αυτές. Το function block είναι ονομασμένο έτσι επειδή όλες οι τιμές είναι αρχικοποιημένες ακριβώς όπως χρειάζεται για να εδραιωθεί σειριακή επικοινωνία σύμφωνα με το πακέτο SPABus.

Οι είσοδοι RTSCTRL, TLS και CDLY σχετίζονται μεταξύ τους και οι δύο τελευταίες χρησιμοποιούνται μόνο όταν η RTSCTRL είναι TRUE. Σε αυτή την περίπτωση, ο χρήστης πρέπει να ρυθμίσει τις δύο επόμενες μεταβλητές σε millisecond που είναι οι carrier leading time και carrier lagging time αντίστοιχα. Αναφέρεται πως ο κατασκευαστής των function blocks αυτών προειδοποιεί για συχνά σφάλματα σε αυτή την περίπτωση, οπότε και αυτή η λύση αποφεύχθηκε, και προτιμήθηκε η λειτουργία σε ‘free mode’ όπως αναφέρεται στο εγχειρίδιο χρήσης της βιβλιοθήκης.

Η μεταβλητή mode είναι πολύ σύνθετη στην λειτουργία της και αλληλεπιδρά και στα δύο επόμενα λογικά κομμάτια διεργασιών (function blocks), SPA_REC και SPA_SEND. Για τούτο, ο συγγραφέας κρίνει απαραίτητο να επεξηγήσει τον τρόπο λειτουργίας της, αφού έχει κάνει το ίδιο και για τα δύο παραπάνω function blocks.

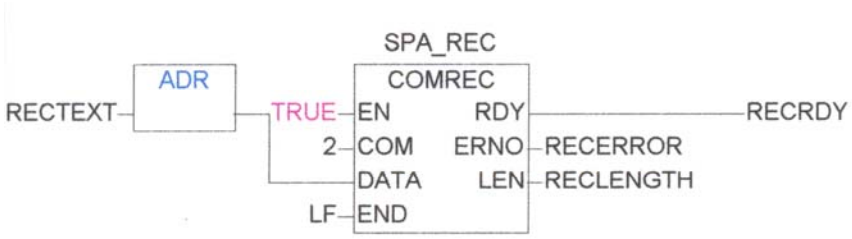
SPA_REC

ΔΗΛΩΣΕΙΣ

```
  FUNCTION_BLOCK SPA_REC
VAR

COM_REC      :COMREC;
RECLENGTH : INT;        (*COUNTER WHICH STORES THE LENGTH OF RECEIVED MESSAGE*)
SPA_REC: COMREC;

END_VAR
```

```
VAR_OUTPUT

RECTEXT     : STRING;          (*RECORDED TEXT*)
RECRDY      : BOOL;            (*BOOLEAN USED TO
COMPARE THE READY STATE OF THE COM_REC*)

RECERROR: INT := 0;        (*RECORD ERROR VALUE*)

END_VAR


VAR_INPUT

END_VAR
```

ΚΩΔΙΚΑΣ



Αυτό το αντικείμενο είναι ένα function block το οποίο καλεί της COMREC, ένα function block που πάλι ανήκει στην Com_S90_V41.lib. Το function block είναι υπεύθυνο για την υποδοχή και καταγραφή των εισερχομένων μηνυμάτων. Αυτό το function block υπάρχει για να καλείται όποτε πρέπει να καταγραφεί εισερχόμενο μήνυμα.

Εδώ υπάρχει μία είσοδος, η ENABLE της οποίας ο ρόλος είναι ο ίδιος με αυτόν της SPA_INIT. Για να λειτουργεί το function block αυτό σε 'free mode' πρέπει η ENABLE να είναι συνέχεια 'TRUE'.

Επίσης και η έξοδος είναι μία, η RDY της οποίας ο ρόλος είναι όμοιος φαινομενικά, με αυτό της ομώνυμης της SPA_INIT όμως στην ουσία η χρήση τους διαφέρει ριζικά. Η ουσία είναι πως όταν η SPA_REC λάβει ένα μήνυμα από τον receive buffer το οποίο θεωρεί ολοκληρωμένο, κάνει την RDY από FALSE ⟶ TRUE για έναν μόλις κύκλο. Αυτό δίνει την δυνατότητα στην ουσία στον χρήστη να γνωρίζει πότε υπάρχει μήνυμα στην σειριακή θύρα.

Για τους ίδιους λόγους, και αυτό το αντικείμενο είναι function block, και γραμμένο σε σχηματικό κώδικα (FDB).

Το πότε ένα μήνυμα θεωρείται ‘σωστό’ θα αναλυθεί παρακάτω.

Αξίζει ο αναγνώστης να στρέψει για λίγο της προσοχή του, στην είσοδο DATA. Όπως φαίνεται, αυτή δεν έχει μια μεταβλητή να δείχνει σε αυτή. Έχει ενδιάμεσα, έναν λογικό operand, τον ADR (ADDRESS) ο οποίος δείχνει στην θέση μνήμης που είναι αποθηκευμένη η μεταβλητή RECTEXT.

Η είσοδος END δηλώνει το εάν πρέπει το function block να περιμένει κάποια συστοιχία δεδομένων ή άλλη ένδειξη σαν το τέλος του τηλεγραφήματος. Η μεταβλητή αυτή σχετίζεται με την μεταβλητή mode του SPA_INIT και θα αναλυθεί παρακάτω.

Η έξοδος LEN επιστρέφει το μήκος του εισερχόμενου μηνύματος (και αυτή για έναν μόνο κύκλο).

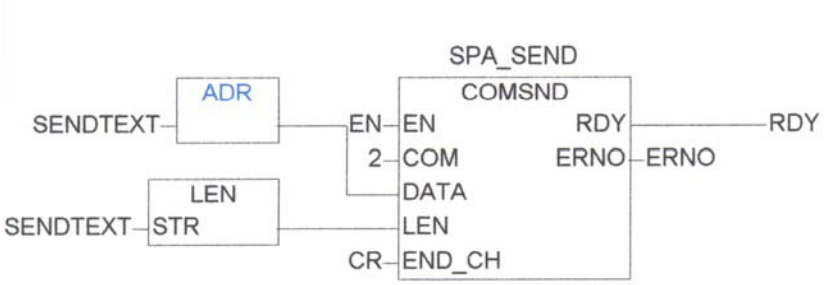
SPA_SEND

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK SPA_SEND

VAR
SPA_SEND        : COMSND;        (*TYPE OF DEVICE*)
END_VAR

VAR_INPUT
        EN: BOOL:=TRUE;          (* ENABLING SIGNAL
FOR THE SEND FUNCTION*)
        SENDTEXT: STRING;        (* TEXT TO BE SENT*)
END_VAR

VAR_OUTPUT
```

```
        RDY: BOOL;          (*OUTPUT BOOLEAN FOR
READINESS OF FUNCTION*)
        ERNO: INT;          (*ERROR NUMBER.
    OUTPUT*)

    END_VAR
```

ΚΩΔΙΚΑΣ



Το παραπάνω αντικείμενο στην ουσία καλεί το function block COMSND της βιβλιοθήκης Com_S90_V41.lib. Είναι υπεύθυνο για να αποστέλλει το εκάστοτε μήνυμα που ο συγγραφέας επιθυμεί, στο δίκτυο.

Για τους λόγους που έχουν προαναφερθεί, ο συγγραφέας επέλεξε και αυτό το αντικείμενο να είναι ένα function block, και γραμμένο σε σχηματικό κώδικα (FDB).

Οι είσοδοι του αντικειμένου αυτού, είναι οι:
ENABLE. Ο σκοπός και ο τρόπος εργασίας της συγκεκριμένης μεταβλητής έχουν αναφερθεί παραπάνω. Είναι σχεδόν τα ίδια με της SPA_INIT, με μια βασική όμως διαφορά. Το συγκεκριμένο function block λειτουργεί μόνο όταν έχουμε ένα FALSE$\longrightarrow$TRUE edge. Αυτό σημαίνει πως αποστέλλεται μήνυμα μόνο όταν το EN γίνει από FALSE, TRUE. Έτσι δίνεται η δυνατότητα στον προγραμματιστή να ελέγχει πόσες φορές ελευθερώνει ένα μήνυμα στο δίκτυο. Η COM έχει να κάνει με την θύρα επικοινωνίας η οποία είναι αρχικοποιημένη στην SPA_INIT.

SENDTEXT. Σε αυτή την μεταβλητή αποθηκεύεται το κείμενο που ο χρήστης θέλει να στείλει στο δίκτυο. Παρατηρούμε πως όπως και πριν, η μεταβλητή δεν εφαρμόζεται απ΄ ευθείας στην είσοδο DATA, αλλά παρεμβάλλεται ο operand ADR ο οποίος επιστρέφει στην είσοδο την διεύθυνση μνήμης της μεταβλητής SENDTEXT. Επίσης κάτι παραπλήσιο γίνεται και στην είσοδο LEN. Η είσοδος αυτή δέχεται το μήκος του εξερχόμενου τηλεγραφήματος. Η συνάρτηση (function) LEN που έχει σαν είσοδο το STR, είναι στην ουσία μια συνάρτηση (function) η οποία επιστρέφει το μήκος μίας μεταβλητής τύπου STRING. Χρησιμοποιείται λοιπόν η παραπάνω συνάρτηση για την εξυπηρέτηση του σκοπού της εισόδου LEN του function block COMSND.

Η συνάρτηση LEN ανήκει στην βιβλιοθήκη IEC_S90_V41.Lib, που είναι και η βιβλιοθήκη που περιέχει κυρίως τέτοιες μικρές συναρτήσεις που χρειάζονται συνεχώς στις πράξεις με STRINGS και καλούνται με μεγάλη ευκολία.

Οι έξοδοι είναι και πάλι οι RDY και ERNO για τις οποίες ισχύει ότι ίσχυε και για την SPA_INIT.

Έχει μείνει η μεταβλητή END_CH. Αυτή η μεταβλητή σχετίζεται με τις MODE, και END. Η MODE της SPA_INIT παίρνει τέσσερις τιμές.

- '0' Όταν η MODE της SPA_INIT γίνεται 0, τότε οι SPA_SEND και SPA_REC δεν αναγνωρίζουν κανένα τέλος τηλεγραφήματος.
- '1' SPA_SEND και SPA_REC αναγνωρίζουν τους χαρακτήρες που θα τους δοθούν στις END και END_CH σαν τέλος τηλεγραφήματος. Έτσι η SPA_SEND βάζει στο τέλος κάθε μηνύματος τον χαρακτήρα που έχει σαν είσοδο στην END_CH ενώ η SPA_REC αναγνωρίζει σαν τέλος τηλεγραφήματος την είσοδο της END.
- '2' Εδώ δεν υπάρχει τέλος εξερχόμενου τηλεγραφήματος, όμως η SPA_REC θεωρεί πως έχει λάβει ένα ολόκληρο μήνυμα μόλις λάβει συγκεκριμένο αριθμό χαρακτήρων από τον receive buffer ο οποίος δίνεται στην μεταβλητή END.
- '3' Εδώ η SPA_REC κάνει το μήνυμα time-out μετά από κάποιο χρόνο, και ότι έχει λάβει, το αναγνωρίζει ως σωστό.

Από όλες τις παραπάνω περιπτώσεις, ο συγγραφέας ακολούθησε την λύση για MODE=1 επειδή σύμφωνα με το SPABus υπάρχουν συγκεκριμένοι stop characters σε κάθε μήνυμα, γεγονός που κάνει τα μηνύματα αναγνώσιμα.

Σημείωση.

Εδώ αξίζει να σημειωθεί το ότι όπως φαίνεται από τα παραπάνω, και η SPA_INIT, και η SPA_REC, αλλά και η SPA_SEND, έχουν μεν αρκετές μεταβλητές, αλλά ελάχιστες από αυτές δηλώνονται σαν είσοδοι, έξοδοι ή είσοδοι και έξοδοι (σε κάποιες περιπτώσεις, μπορεί να γίνει και αυτό). Αυτό γίνεται επειδή οι μεταβλητές που είναι δηλωμένες απλά σαν εσωτερικές μεταβλητές, δεν χρειάζεται να παρουσιάζονται στο υπόλοιπο project καθ' ότι αφορούν σε εσωτερικές λειτουργίες των εκάστοτε function block, και επίσης εάν δηλώνονταν σαν είσοδοι, έξοδοι ή είσοδοι και έξοδοι, αυτομάτως θα έπρεπε κάθε φορά που ο συγγραφέας θα χρειαζόταν να καλέσει ένα function block, θα έπρεπε να αναφέρει όλες τις μεταβλητές αυτές καθώς και να τις δηλώνει στο αντικείμενο που θα καλούσε το εν λόγω function block και πέραν αυτού, θα έπρεπε να δηλώνει και κάποιες μεταβλητές με τιμές που θα ήταν ίσες με αυτές που θα ήθελε να θέσει σε όλες αυτές τις μεταβλητές, γεγονός που θα ήταν μεγάλη σπατάλη χρόνου και πόρων του συστήματος.

Έτσι είναι προτιμότερο τα function block να μένουν μικρά σε μέγεθος και ευέλικτα, για να αποφεύγονται τα παραπάνω φαινόμενα.

Εδώ ο συγγραφέας αισθάνεται αναγκαίο να ξεκαθαρίσει στον αναγνώστη πως το γεγονός πως μια μεταβλητή ενός αντικειμένου μπορεί να δηλωθεί είσοδος, έξοδος ή είσοδος και έξοδος, δεν σημαίνει πως δεν είναι τοπική μεταβλητή. Η μεταβλητή παραμένει να είναι τοπική μεταβλητή (local variable) του αντικειμένου στο οποίο ανήκει. Απλά τα υπόλοιπα αντικείμενα του project, γνωρίζουν πως σε αυτές τις μεταβλητές είναι αποθηκευμένες μεταβλητές που μπορεί να παίζουν κάποιο ρόλο και σε κάποιο άλλο αντικείμενο του project ή πως το block αυτό χρειάζεται οπωσδήποτε μια αρχικοποίηση σε αυτές τις τιμές για να λειτουργήσει.. Παρ' όλα ταύτα ένα αντικείμενο που είναι πρόγραμμα, δίνει το δικαίωμα σε ένα άλλο πρόγραμμα να έχει πρόσβαση στις εσωτερικές τοπικές μεταβλητές του. Αυτό γίνεται αναφέροντας το όνομα του προγράμματος, προσθέτοντας απ' ευθείας το στοιχείο της τελείας, και αμέσως μετά το όνομα της εσωτερικής μεταβλητής που μας ενδιαφέρει.

Απλά ένα ξένο αντικείμενο δεν έχει δικαίωμα εγγραφής σε μια μεταβλητή ενός αντικειμένου, η οποία δεν είναι δηλωμένη ως είσοδος.

Οι ολικές μεταβλητές (global variables) αποθηκεύονται σε άλλο, ειδικά διαμορφωμένο κομμάτι του κώδικα, στον φάκελο πόρων (resources). Αξίζει να σημειωθεί πως καταναλώνουν πάρα πολύ μνήμη. Ένας πίνακας εκατό θέσεων αποτελούμενος από STRING ογδόντα θέσεων, καταλαμβάνει σχεδόν 1 Megabyte σε ένα μικρό πρόγραμμα.

Στην ουσία από εδώ και πέρα αρχίζει η έννοια της συγγραφής καινούριου κώδικα. Όλα τα παραπάνω αντικείμενα ήταν πάνω – κάτω γνωστά και μέχρι κάποιο σημείο τετριμμένα

Από εδώ και κάτω θα παρατίθεται πρώτα το όνομα του αντικειμένου, μετά οι δηλώσεις των μεταβλητών τις οποίες αυτό χρησιμοποιεί, ύστερα ο κώδικας που είναι γραμμένος μέσα στα πλαίσιά του και τελικά ο τρόπος λειτουργίας του ακολουθούμενος από διευκρινήσεις και παρατηρήσεις.

MESCOMPOSE

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK MESCOMPOSE


    VAR_INPUT
    SNUM    : STRING ;(*NUMBER OF SLAVE TARGET*)
    CNUM    : STRING;               (*OUTGOING          MESSAGE
CHANNEL DESTINATION*)
    SCAT    : STRING;          (*SETTING CATEGORY*)
    SETNUM  : STRING;               (*SETTING NUMBER*)
    DATA        : STRING;           (*ACTUAL DATA*)
    CSUM        : STRING;           (*CHECKSUM*)
    ONUM        : STRING;           (*OUTPUT    NUMBER    (ONLY
FOR WRITES)*)
    VALUE       : STRING;           (*VALUE TO BE SET IN SLAVE
OUTPUT (WRITE ONLY)*)
    END_VAR

    VAR_OUTPUT
        OPV    : STRING;            (*FINAL MESSAGE*)
    END_VAR

    VAR
        COPV : STRING;              (*> SING INPUT*)
        COPV1: STRING;
        COPV2: STRING;              (*JOINING OF COPV+COPV1*)
        COPV3: STRING;
        COPV4: STRING;              (*COPV2,3 JOINING*)
        COPV5: STRING;
        COPV6: STRING;              (*COPV4.5 JOINING*)

    END_VAR
```

ΚΩΔΙΚΑΣ

```
IF MTYPE ='R' THEN

COPV        :=CONCAT('>',SNUM);
COPV1       :=CONCAT(MTYPE,CNUM);
COPV2       :=CONCAT(COPV,COPV1);
COPV3       :=CONCAT(SCAT,SETNUM);
COPV4       :=CONCAT(COPV2,COPV3);
COPV5       :=CONCAT(':',CSUM);
COPV6       :=CONCAT(COPV4,COPV5);
OPV         :=CONCAT(COPV6,'$R');

    ELSE

COPV        :=CONCAT('>',SNUM);
COPV1       :=CONCAT(MTYPE,ONUM);
COPV2       :=CONCAT(COPV,COPV1);
COPV3       :=CONCAT(':',VALUE);
COPV4       :=CONCAT(COPV2,COPV3);
COPV5       :=CONCAT(':',CSUM);
COPV6       :=CONCAT(COPV4,COPV5);
OPV         :=CONCAT(COPV6,'$R');

END_IF
```

Και αυτό το αντικείμενο είναι τύπου function block.

Σκοπός αυτού του αντικειμένου είναι η κατασκευή του μηνύματος που ο συγγραφέας θέλει να αποσταλεί κάθε φορά. Όπως είναι γνωστό στον αναγνώστη, το κάθε μήνυμα είναι διαφορετικό, όμως η δομή όλων των εξερχόμενων τηλεγραφημάτων είναι ίδια ή εάν διατυπωθεί ακριβέστερα, οι βασικές δομές είναι δύο. Η μία είναι η δομή R (read) όπου ο master στέλνει μήνυμα ζητώντας από τον slave να του αποστείλει την τιμή που έχει σε κάποια συγκεκριμένη θέση του ή σε κάποιο εύρος θέσεων, και η δεύτερη είναι η W (write) όπου ο master ζητά από τον slave να καταχωρίσει σε μια συγκεκριμένη θέση, ή φάσμα θέσεων, την τιμή που του αποστέλλει, ή το φάσμα τιμών που του αποστέλλει αντίστοιχα.

Σε σχέση με το προηγούμενο project, το οποίο είχε κατασκευαστεί για ελέγξει την συνεργασία PLC – SPAJ μέσο του project αυτού, αναγκαστικά το αντικείμενο αυτό άλλαξε μορφή.

Σε εκείνο το project, το αντικείμενο αυτό απλά ανοικοδομούσε ένα μήνυμα, εδώ όμως καλείται να λειτουργήσει αυτόματα, χωρίδς την παρέμβαση του χρήστη και θα πρέπει να λειτουργεί σωστά οπότε κάποιες αλλαγές κρίθηκαν αναγκαστικές.

Στο πρώτο project κατ' αρχήν ονομαζόταν MASTERMESSAGE. Το αντικείμενο αυτό είχε σαν εισόδους κάποια δεδομένα όπως SNUM, το οποίο είναι ο αριθμός του slave στον οποίο ο master απευθύνεται, MTYPE, το οποίο είναι στην ουσία ένα γράμμα (R ή W) που δηλώνει τον τύπο του μηνύματος που ο master στέλνει στον slave, CNUM, το οποίο είναι ο αριθμός του καναλιού του slave, στο οποίο ο master απευθύνεται, DCAT που αντιπροσωπεύει την κατηγορία των δεδομένων DNUM, που είναι ο αριθμός των δεδομένων (όπως είπαμε πριν μπορούμε να στείλουμε ή να λάβουμε μια συστάδα δεδομένων), DATA, οπού εδώ είναι τα ουσιαστικά δεδομένα και τελικά CSUM, που είναι οι χαρακτήρες που δηλώνουν στον slave τέλος τηλεγραφήματος. Οι έξοδος ήταν μόνο μία, με την τελική μορφή του συναρμολογημένου μηνύματος (OPV).

Στο καινούριο project, το όνομα αυτό θα φαινόταν παραπλανητικό καθ' ότι και άλλα αντικείμενα είχαν κεντρικές εργασίες σχετικά με τα εξερχόμενα τηλεγραφήματα. Επίσης έπρεπε να διαχωριστούν οι διεργασίες σε διαφορετικό πλαίσιο όταν ο master ήθελε να κάνει write και σε άλλο όταν ήθελε να κάνει read και αυτό γιατί σε κάθε περίπτωση, το τηλεγράφημα θα έπρεπε από κάποιο σημείο και μετά να περιέχει διαφορετικές πληροφορίες. Έτσι προέκυψε το αντικείμενο MESCOMPOSE. Στην ουσία η λειτουργία του στηρίζεται στην εξής φιλοσοφία:

Διάβασε την μεταβλητή MTYPE.

Εάν η MTYPE είναι ίση με 'R' τότε εκτέλεσε
...
...
...
αλλιώς εκτέλεσε
...
...
...
τέλος εάν

Έτσι το αντικείμενο καλεί άλλες μεταβλητές για τα write και άλλες για τα read έτσι ώστε να δημιουργηθεί μια ολοκληρωμένη και σωστή δομή, που θα εγγυάται ευστάθεια.

Κακό αυτού του function block έτσι όπως είναι κατασκευασμένο είναι πως ο χρήστης πρέπει να αναφέρει πολλές μεταβλητές κάθε φορά που θέλει να το καλέσει, επειδή έχει οκτώ εσωτερικές μεταβλητές εισόδου και μία εσωτερική μεταβλητή εξόδου, γεγονός που μπορεί ανά περιόδους να γίνεται κουραστικό για τον συγγραφέα αλλά και τον αναγνώστη και τον χρήστη. Παρ' όλα ταύτα το αντικείμενο λειτουργεί σωστά, οπότε και παραμένει παρά τα προβλήματά του.

MESDECOMPOSE

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK MESDECOMPOSE

        VAR_INPUT

        MTYPE                   :STRING;
        RECTEXT  :STRING;   (*THE INCOMING MESSAGE*)
        END_VAR


        VAR_OUTPUT

    OK      : BOOL;    (*STATE          OF          TRANSACTION
COMPLETED/NOT COMPLETED*)
    FORSTORAGE     : STRING;          (*STRING WITH BOTH
SLAVE NUMBER AND DATA FOR STORAGE*)

        END_VAR


        VAR

    CLEANDATA: STRING;   (*ACTUAL RECORDED DATA*)
    FINDER          : STRING(1);
    SUSPEND   : INT;        (*UNTIL SUSPEND MODE*)
    I         : INT;        (*SIMPLE COUNTER*)
    T         : INT:=0;     (*SIMPLE COUNTER*)
    L         : INT;        (*TEMPORARY STORAGE*)
    C         : INT;        (*CASE SELECTOR*)
    COMP      : INT;        (*SIMPLE COMPARATOR*)
    POINT1,POINT2    : INT;         (*WHEN  ':' APPEARS*)
    RECTEXTR        : STRING;           (*USE FOR HELP*)
    SNUM            : STRING;           (*THE   SENDER   SLAVE
NUMBER*)
    TIMEDATA        :ARRAY[1..60000] OF STRING;
    INTERMID        : STRING;           (*TEMPORARY       USE
STRING*)

    END_VAR
```

ΚΩΔΙΚΑΣ

```
COMP := FIND(RECTEXT,'N');
                SUSPEND:=0;
                        IF COMP <> 0 THEN
                        OK:=FALSE;
                        ELSE

          IF MTYPE ='R' THEN
         C:=1;
                        ELSIF MTYPE='W' THEN
         C:=2;
          END_IF

CASE  C OF

1:
          SNUM:= MID (RECTEXT,D-1,2);
          INTERMID:=CONCAT(SNUM,'/');
          POINT1:=FIND(RECTEXT,':');
          RECTEXTR := RIGHT (RECTEXT, LEN(RECTEXT)-
POINT1);
          POINT2:=FIND(RECTEXTR,':');


          (* FOR I:=POINT1 TO (POINT1+POINT2) DO*)
              CLEANDATA:=        MID        (RECTEXT,(POINT2-
1),(POINT1+1));
          FORSTORAGE:=CONCAT(INTERMID,CLEANDATA);
              TIMEDATA[T]:=FORSTORAGE;
          (*END_FOR*)

          OK:=TRUE;

2:

              D:=FIND(RECTEXT,'R');
              SNUM:= MID (RECTEXT,D-1,2);
              OK:=TRUE;


END_CASE
END_IF
```

Αυτό το αντικείμενο είναι ένα function block.

Είναι ένα τελείως καινούριο αντικείμενο που δεν έχει ξαναπαρουσιαστεί σε παλαιότερη έκδοση ούτε με την ίδια αλλά ούτε και παραπλήσια μορφή. Η κατασκευή του κρίθηκε αναγκαία σε σχέση με το project COMT_C.pro όπου δεν υπήρχε καν, γιατί και πάλι η ανάγκη αυτόματου ελέγχου του εισερχόμενου σήματος, απαιτούσε ένα εργαλείο το οποίο να μπορεί αυτόματα να επεξεργαστεί το εισερχόμενο από τον οποιοδήποτε slave προς τον master τηλεγράφημα, και να αποφασίζει εάν το μήνυμα περιέχει χρήσιμη πληροφορία, και αν ναι να την περισυλλέξει μέσα από το υπόλοιπο μήνυμα, ενώ να ξεκινά άλλη διαδικασία σε περίπτωση μηνύματος σφάλματος.

Η MESDECOMPOSE λοιπόν κάνει ακριβώς αυτό.

Α) διαβάζει και αναλύει το εισερχόμενο τηλεγράφημα.

Β) εάν αυτό είναι σωστό, τότε με έναν αφαιρετικό μηχανισμό εξάγει τα χρήσιμα για το project δεδομένα, από το υπόλοιπο μήνυμα.

Γ) εάν το μήνυμα είναι μήνυμα λάθους καλεί τα αντικείμενο που χειρίζεται τέτοιες περιπτώσεις (ERROR_HANDLING), το οποίο όμως εδώ αναφέρεται μόνο ονομαστικά και θα αναπτυχθεί παρακάτω.

Οι είσοδοι που δηλώνονται στο αντικείμενο αυτό είναι η μεταβλητή SENDTEXT, στην οποία καταχωρείται το εξερχόμενο κείμενο, και MTYPE στην οποία καταχωρείται ο τύπος του εξερχόμενου μηνύματος από την MESCOMPOSE. Το όλο σκεπτικό είναι πως για να μπορεί το αντικείμενο να κατηγοριοποιήσει τα εισερχόμενα δεδομένα, θα πρέπει να γνωρίζει τι είδους δεδομένα ζητήθηκαν και από που. Η αλήθεια όμως είναι πως με έμμεσο τρόπο αυτές οι πληροφορίες περιέχονται στο εισερχόμενο τηλεγράφημα. Παρ' όλα ταύτα, ο συγγραφέας διατηρεί αυτή τη δήλωση, επειδή έτσι μπορεί να καλεί το function block αυτό για συγκεκριμένες συνθήκες, αλλά και επειδή πιστεύει πως σε περίπτωση μελλοντικής επέκτασης του project το πιθανότερο είναι πως η μεταβλητή αυτή θα χρειαστεί.

Οι έξοδοι είναι οι μεταβλητές OK και η CLEANDATA.

Η μεταβλητή OK είναι τύπου bool και παίρνει τιμή ‘αληθής’ (TRUE) να δηλώνει πως όλη η διαδικασία της ανάγνωσης ολοκληρωθεί με επιτυχία, ενώ γίνεται ‘αναληθής’ (FALSE) όταν το εισερχόμενο τηλεγράφημα είναι μήνυμα λάθους (Not Acknowledged).

Στην μεταβλητή CLEANDATA μετά από την επεξεργασία του εισερχομένου τηλεγραφήματος αποθηκεύονται οι χρήσιμες πληροφορίες και τα δεδομένα που αυτό περιείχε.

Το αντικείμενο αυτό επιτυγχάνει τους στόχους του με την λογική που παρουσιάζεται παρακάτω.

Διαχείριση σφάλματος.

Πριν η διαδικασία εκτελέσει οτιδήποτε άλλο, ψάχνει όλο το εισερχόμενο τηλεγράφημα, για το γράμμα ‘N’ με την εξής εντολή:

```
COMP := FIND(RECTEXT,'N');
```

Αυτό γίνεται επειδή όταν η μεταβλητή MTYPE του εισερχόμενου τηλεγραφήματος RECTEXT γίνει ίση με ‘N’, τότε σημαίνει πως το μήνυμα ήταν κατά τον slave εσφαλμένο. Το γράμμα ‘N’ δεν εμφανίζεται σε καμία άλλη θέση μέσα σε ένα μήνυμα, οπότε εάν υπάρχει, τότε σίγουρα το μήνυμα είναι εσφαλμένο. Ο αναγνώστης εύλογα θα αναρωτάτε, πως αφού το γράμμα ΄N΄ δεν εμφανίζεται σε καμία άλλη θέση, γιατί ο κώδικας δεν είναι έτσι γραμμένος ώστε να ερευνά μόνο στην συγκεκριμένη θέση;

Η απάντηση είναι πως μπορεί η θέση να είναι σταθερή όμως πριν από αυτή ο slave επιστρέφει το νούμερο του, οπότε εάν αυτό είναι διψήφιο, τότε αυτόμάτος η θέση στην οποία το γράμμα αυτό θα βρίσκεται, μεταβάλλεται. Επίσης για κάποιο ομολογουμένως περίεργο λόγο, ο compiler του AC1131 δεν επιτρέπει την επιλογή ενός μόνο στοιχείου μέσα από ένα string. Έτσι μια δήλωση της μορφής

```
RECTEXT(2):=’X’;
```

Θεωρείται εσφαλμένη με απάντηση πως δεν γίνεται να εξισώσουμε μεταβλητή τύπου char με μεταβλητή τύπου string. Επίσης μια δήλωση της μορφής:

```
IF   RECTEXT(4)=’N’ THEN
…
```

Επίσης θεωρείται εσφαλμένη, γιατί σύμφωνα με τον compiler, μόνο μεταβλητές τύπου Boolean μπορούν να μετάσχουν ως συγκριτικά μέσα σε εκφράσεις if_then_else, while, repeat-until for_to κ.τ.λ. loop.

Οι παραπάνω ιδιαιτερότητες της συγκεκριμένης γλώσσας δυσκόλεψαν τη συγγραφή σε πάρα πολλά σημεία κάνοντας προβλήματα που σε άλλες γλώσσες προγραμματισμού θα είχαν λυθεί αμέσως, πολύ δύσκολα.

Έπρεπε λοιπόν να βρεθεί ένας άλλος τρόπος για να γίνει η ίδια διεργασία. Αυτός ήταν η συνάρτηση FIND η οποία ψάχνει για ένα χαρακτήρα ή ακολουθία χαρακτήρων μέσα σε ένα string και επιστρέφει '0' εάν δεν βρει τίποτα, ή τον αριθμό της θέσης που συναντά τον χαρακτήρα αυτό ή την ακολουθία χαρακτήρων αυτή μέσα στο string, για πρώτη φορά.

Εάν η μεταβλητή COMP είναι διάφορη του '0' μετά την έρευνα, τότε καλείται το αντικείμενο. ERROR_HANDLING. Αξίζει να δοθεί προσοχή στο γεγονός πως η κλίση του γίνεται μονάχα αναφέροντας την ονομασία του. Εδώ, στην τελευταία έκδοση, ο συγγραφέας πέρασε σήμα ENABLE στην ERROR_HANDLING έτσι ώστε να μην ξεκινά εάν ο χρήστης δεν το επιθυμεί.

Αντίθετα εάν η μεταβλητή είναι μηδενική, τότε το function block προχωρά παρακάτω στον κώδικά του σε μια CASE δήλωση. Εδώ ο κώδικας ανατρέχει στην μεταβλητή MTYPE του εξερχόμενου τηλεγραφήματος. Εάν η MTYPE ήταν 'R' ακολουθείται η πρώτη συνέχεια. Εάν η MTYPE ήταν 'W'τότε το εισερχόμενο τηλεγράφημα εξετάζεται από το δεύτερο σκέλος της case.

Στην πρώτη περίπτωση έρχεται μόνο μια απάντηση πως τα δεδομένα γράφτηκαν στον slave οπότε το μόνο που έχει να κάνει ο κώδικας, είναι να κάνει την έξοδο OK ίση με TRUE.

Στην δεύτερη περίπτωση με μια ακολουθία κλίσεων διαφόρων συναρτήσεων ο κώδικας απομονώνει τη χρήσιμη πληροφορία από το υπόλοιπο τηλεγράφημα και την αποθηκεύει στην μεταβλητή CLEANDATA.

Αυτά αποθηκεύονται μέσο της μεταβλητής FORSTORAGE στον πίνακα TIMEDATA 60000 θέσεων ο οποίος αποθηκεύει σε αύξουσα σειρά τα καθαρά δεδομένα όπως έρχονται σε περίπτωση που μελλοντικά υπάρξει ιδέα για stable storage. Στο μελλοντικό σχέδιο, η TIMEDATA κάθε φορά που γεμίσει, θα μπορεί να αποθηκεύει τα δεδομένα σε ένα υπολογιστή με τον οποίο θα μπορεί να είναι συνδεδεμένος.

Πολλές τοπικές εσωτερικές μεταβλητές, υπάρχουν ως προσωρινή χώροι αποθήκευσης για να επιτευχθούν οι σκοποί της διεργασίας.

Άλλες από αυτές χρησιμοποιούνται ως counters. Επίσης πολλές μεταβλητές υπάρχουν και για μελλοντική χρήση.

Η διεργασία αυτή λειτουργεί σειριακά.

ERROR_HANDLING

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK ERROR_HANDLING
VAR_INPUT

        RECTEXT                                 :STRING;
        SENDTEXT                                :STRING;

END_VAR



VAR

        ERROR_STATE: INT;               (*ERROR CHECKING*)
        SUS: INT;

END_VAR


VAR_OUTPUT

        OK                  : BOOL:=FALSE;

END_VAR
```

ΚΩΔΙΚΑΣ

```
OK:=FALSE;

        SUS:=1;
         REPEAT

                PLC_PRG.SENDTEXT:=SENDTEXT;
                SUS:=SUS+1;
                        UNTIL
                 FIND (RECTEXT,'N') = 0
                        OR
                SUS=500
            END_REPEAT
```

```
                IF
                        SUS=500         THEN
                OK:=FALSE;
            END_IF
          IF          FIND (RECTEXT,'N') = 0 THEN
                OK:=TRUE;
  END_IF
```

Το αντικείμενο αυτό είναι υπεύθυνο για την διαχείριση λαθών.

Έχει σαν είσοδο το SENDTEXT.

Στέλνει συνεχώς το μήνυμα SENDTEXT μέχρι να έρθει θετική απάντηση από τον slave. Τα υπόλοιπα μηνύματα τίθενται εκτός λειτουργίας. Εάν το μήνυμα σταλεί ανεπιτυχώς πεντακόσιες φορές, τότε καλείται η SUSPEND της οποίας η λειτουργία επεξήγεται παρακάτω.

Παρατηρείται ο τρόπος με τον οποίο τα αντικείμενα συνεργάζονται μεταξύ τους. Το ένα δεν εμπλέκεται στις διαδικασίες του άλλου, όμως του μεταβιβάζει όλες τις χρήσιμες πληροφορίες έτσι ώστε το καθ' ένα χωριστά να μπορέσει να συντελέσει στην άρτια λειτουργία του project.

SUSPEND

ΔΗΛΩΣΕΙΣ

```
PROGRAM SUSPEND
VAR_INPUT

      SENDTEXT            :STRING;

END_VAR


VAR

      SNUM                :STRING(2);

 K                        :INT;
 SIGNAL     AT %MX255.0 : BOOL;
 MISTAKE                  : STRING;
      SNUMINT             :INT;

END_VAR
```

ΚΩΔΙΚΑΣ

```
K:=0;

REPEAT

IF SIGNAL = TRUE THEN

  SEND(DATA:=ADR(SENDTEXT));
  K:=K+1;

END_IF

UNTIL

        FIND (RECTEXT,'N') = 0
       OR
       K=100

END_REPEAT

SNUM:= MID (SENDTEXT,1,2);
SNUMINT:=STRING_TO_INT(SNUM);

        IF K=100 THEN

                        MISTAKE :='WARNING!!! PROBABLE
HARDWARE FAILURE!!! CURRENT SLAVE IS DELETED FROM
SLAVELIST';
    (*
  SLAVE_DEFINITION.SLAVEDATA[SUSPEND.SNUMINT]:=''; *)
                                ELSE
          K:=0;

     END_IF
```

Το αντικείμενο αυτό είναι πλέον πρόγραμμα.

Το πρόγραμμα αυτό αναλαμβάνει ένα σήμα, όταν αυτό έχει αποτύχει και το ξαναστέλνει στον παραλήπτη άλλες εκατό φορές με χρονοκαθυστέρηση περίπου δυόμισι δευτερολέπτων ανάμεσα σε κάθε αποστολή Εάν όλες οι προσπάθειες είναι ανεπιτυχείς, τότε ο slave διαγράφεται από την λίστα των δεκτών(προαιρετική ενέργεια) (θα

αναφερθεί παρακάτω) και αποστέλλει μήνυμα προειδοποίησης σφάλματος υλικού στο χρήστη.

Την διαδικασία διαγραφής του slave δεν την κάνει η ίδια η SUSPEND καθ' ότι δεν είναι υπεύθυνη για ενέργειες τέτοιου είδους.

Προς αποφυγή σφαλμάτων που θα μπορούσαν να προκαλέσουν σύγχυση στο σύστημα, δικαιοδοσία για εγγραφή και διαγραφή των slaves από την λίστα των δεκτών, έχει μονάχα το αντικείμενο SLAVE_DEFINITION το οποίο και θα παρατεθεί παρακάτω.

SLAVE_DEFINITION

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK SLAVE_DEFINITION

VAR
    SENDTEXT            :STRING(13);
    RECTEXT             :STRING(13);
    I                   :INT;


    TON                 :TON;
    SEND                :SPA_SEND;
    REC                 :SPA_REC;

    INSIG               :BOOL:=TRUE;
    OUTSIG              :BOOL:=FALSE;


     NUM                : STRING(2);
          (*SLAVE NUMBER*)
    TEXT                : STRING(13);

    OK                  : BOOL;
    SNUM                : STRING(2);
    SLINT               : INT;
    K                   : INT;
    J                   : INT;
    READY               : BOOL;
    SYSTIME             : DWORD;
```

```
    COMP                        : DWORD;
    D: INT;
END_VAR
VAR_OUTPUT
    SLAVEDATA           :ARRAY[1..10] OF STRING(13);
    BUSY                        : BOOL;
    RESULT                      : BOOL;
END_VAR
VAR_INPUT
    EN: BOOL;
END_VAR
```

ΚΩΔΙΚΑΣ

```
IF EN=TRUE THEN
IF I<256 THEN

        I:=I+1;

                NUM:=INT_TO_STRING(I);

SENDTEXT:=INSERT('>RF:XX',NUM,1);



        RECTEXT:=PLC_PRG.RECTEXT;
IF RESULT = FALSE THEN


                IF RECTEXT<>SLAVEDATA[I-1] THEN

                        IF FIND(RECTEXT,'N')=0 THEN

                                D:=FIND(RECTEXT,'D');
        SNUM:= MID (RECTEXT,D-1,2);

        SLINT:=STRING_TO_INT(SNUM);
                SLAVEDATA[SLINT]:=RECTEXT;



                        ELSE
                SLAVEDATA[SLINT]:='';
```

```
                    END_IF
                ELSE
                 SLAVEDATA[SLINT]:='';


                END_IF
END_IF


ELSE
        RESULT :=TRUE;
SENDTEXT:='';
END_IF


END_IF
```

Το αντικείμενο αυτό, έχει ως σκοπό του την καταγραφή όλων των slaves και την αποθήκευσή τους σε ένα πίνακα, έτσι ώστε ανά πάσα στιγμή ο χρήστης να έχει την δυνατότητα να γνωρίζει πόσοι ηλεκτρονόμοι είναι συνδεδεμένοι στο δίκτυο, ποιοι είναι αυτοί, και σε ποια θέση και με ποιον αριθμό slave είναι δηλωμένος ο κάθε ένας. Επίσης ο πίνακας κρατάει πληροφορίες σχετικά με το μοντέλο του κάθε συγκεκριμένου slave. Το τελευταίο στοιχείο ήταν και απαίτηση του όλου project.

Η SLAVE_DEFINITION είναι κατασκευασμένη με τέτοιο τρόπο ώστε εάν καλείται ανά τακτά χρονικά διαστήματα (λόγου χάρη τριάντα δευτερόλεπτα) από το κεντρικό πρόγραμμα (με την επωνυμία PLC_PRG) τότε το project υλοποιεί συνθήκη plug and play. Αυτό είναι μια πάρα πολύ σημαντική δυνατότητα ειδικά για το κόστος του υλικού το οποίο το project αυτό υποστηρίζει.

NORMAL_SEQUENCE

ΔΗΛΩΣΕΙΣ

```
PROGRAM NORMAL_SEQUENCE


VAR

MESDEC    :MESDECOMPOSE;
MESCOMP  :MESCOMPOSE;

  SNUM              :INT;
I: INT;
RECEIVER: STRING;     (*VALUE USED TO REPRESENT
SIGNAL REFFERING TO SPECIFIC SLAVE *)

COUNTDOWN: ARRAY [1..99] OF INT;
MIDDLE: STRING (2);

END_VAR



VAR_INPUT

OK: BOOL;
CLEANDATA: STRING;
FINDER: STRING;

END_VAR
```

ΚΩΔΙΚΑΣ

```
(*SLAVE_DEFINITION;*)

FOR I:= 1 TO 99 DO
  IF SLAVE_DEFINITION.SLAVEDATA [I] <>‘‘ THEN
        MESCOMP.MTYPE: ='R';

 SEND (DATA:=ADR(MESCOMP.OPV));
END_IF
            RECEIVER:=MESCOMP.SNUM;
```

```
END_FOR
  FOR I:= 1 TO 99 DO
     IF SLAVE_DEFINITION.SLAVEDATA [I]<>‘‘ THEN
     COUNTDOWN[I]:=1;
   ELSE
          COUNTDOWN[I]:=0;
   END_IF
 END_FOR

   REPEAT
     MESDEC (SENDTEXT:        =MESCOMP.OPV,     RECTEXT:
         =RECTEXT);

  I:=I+1;
        MIDDLE: = MID ('RECTEXT', 2,3);
        SNUM:= STRING_TO_INT(MIDDLE);
       COUNTDOWN [SNUM]:=0;




   UNTIL

                        COUNTDOWN [I]=0
      OR    I=99


          END_REPEAT
```

Η NORMAL_SEQUENCE παρατίθεται περισσότερο για μουσειακούς λόγους.

Το πρόγραμμα αυτό ήταν κατασκευασμένο για να εκτελεί τις διεργασίες που ο χρήστης επιθυμεί σε κατάσταση ηρεμίας. Το ζήτημα είναι πως αυτές οι απαιτήσεις έπρεπε να αντιμετωπίζονται ανά περίπτωση γιατί για κάθε είδος χρήσης του κώδικα, απαιτείται η ανάγνωση εντελώς διαφορετικών αντικειμένων και διαφορετικών δεδομένων από τους slaves οπότε αυτό το αντικείμενο δεν μπορούσε να τελειώσει χωρίς περαιτέρω διευκρινήσεις.

Εάν το επιθυμητό είναι ο συγγραφέας να δημιουργήσει τα λογικά κομμάτια διεργασιών τα οποία ο μελλοντικός προγραμματιστής να μπορεί να καλέσει οποιαδήποτε στιγμή μέσα από μια βιβλιοθήκη (που θα δημιουργήσει εδώ ο συγγραφέας) ώστε να μπορέσει με ευκολία να διεκπεραιώσει τον σκοπό της εργασίας του, τότε η σύνταξη της υπόλοιπης αυτής διεργασίας θα έπρεπε να είναι κάτι που καλείται να

φέρει εις πέρας αυτός, γιατί ο συγγραφέας αδυνατούσε στην Χρονική στιγμή της 13ης Ιουνίου να γνωρίζει το τι θα ζητούσε ο μελλοντικός χρήστης από το project.

Μέχρι αυτό το σημείο, το πρόγραμμα αποστέλλει ένα μήνυμα σε όλους τους αποθηκευμένους στον πίνακα SLAVEDATA slaves. Εν μέρει το μήνυμα κατασκευάζεται αυτόματα, όχι όμως ολοκληρωτικά αφού λείπουν πληροφορίες. Στην συνέχεια για τα εισερχόμενα τηλεγραφήματα καλείται η MESDECOMPOSE και όλα είναι σωστά εκεί τελειώνει η εκτέλεση.

Ο αναγνώστης λογικά θα αναρωτηθεί για ποιο λόγο η MESDECOMPOSE δεν καλεί την NORMAL_SEQUENCE ξανά εάν το εισερχόμενο τηλεγράφημα είναι επιτυχές. Υπάρχει περίπτωση ο κώδικας έτσι να εισέλθει σε infinite loop γεγονός που τελικά θα προκαλέσει σφάλμα συστήματος (λόγου χάρη soft crash), κάτι που είναι επιθυμητό να μην συμβεί.

Η αντικατάσταση της NORMAL_SEQUENCE ήρθε από την SPAJ140C η οποία έκανε αυτόματη οδήγηση στους ηλεκτρονόμους τύπου SPAJ που ήταν και ο τελικός σκοπός της εργασίας. Προτού αναλύσουμε την SPAJ140C αναφέρουμε πως για να λειτουργήσει χρειάζεται την MESDECOMPOSE καθώς και ένα καινούριο function block, την CLEANING η οποία και παρατίθεται παρακάτω.

CLEANING

ΔΗΛΩΣΕΙΣ

```
    FUNCTION_BLOCK CLEANING
VAR_INPUT
    RECTEXT: STRING;
END_VAR

VAR_OUTPUT
```

```
        FINAL: STRING;

    END_VAR


    VAR

        FINAL1: STRING;
        FINAL2: STRING;
        LOOK: INT;

    END_VAR
```

ΚΩΔΙΚΑΣ

```
    IF FIND(RECTEXT,'>')<>0 THEN

            FINAL:='';


        END_IF



        IF FIND(RECTEXT,'&')<>0       THEN

        FINAL1:=LEFT(RECTEXT, LEN(RECTEXT)-LOOK);
                ELSE
          FINAL2:=RECTEXT;
        END_IF


        FINAL:=CONCAT(FINAL1,FINAL2);
```

Η λειτουργία της CLEANING είναι η παρακάτω. Πολλές φορές, επειδή τα δεδομένα που ζητά ο master μπορεί να είναι πολλά σε μέγεθος, ο slave παίρνει την πρωτοβουλία να διαιρέσει το εξερχόμενο μήνυμα σε δύο, έτσι ώστε τα δεδομένα να μην ξεπεράσουν το μέγιστο επιτρεπόμενο

όριο των 256 χαρακτήρων. Η Η CLEANING λοιπόν, αναλαμβάνει τις εξής εργασίες

1. Να σβήσει το εισερχόμενο μήνυμα εάν αυτό είναι echo.
2. Να συνενώσει τα όσα κομμάτια μηνύματος ανήκουν στο ίδιο τηλεγράφημα έτσι ώστε η MESDECOMPOSE να μπορέσει να λειτουργήσει κανονικά.

Το πρώτο επιτυγχάνεται ψάχνοντας το μήνυμα για στοιχεία που μόνο ο master βάζει, ενώ το δεύτερο σαρώνοντας το μήνυμα για continuation character (&). Εάν αυτό βρεθεί, όλα τα επόμενα μηνύματα συνενώνονται μεταξύ τους μέχρι να μην βρεθεί άλλο continuation character (&) πριν από τους τελικούς χαρακτήρες lf (line feed).

SPAJ140C

ΔΗΛΩΣΕΙΣ

```
FUNCTION_BLOCK SPAJ140C

    VAR_INPUT

        EN                  :BOOL;
        L                   :INT;
        RECTEXT: STRING;
     END_VAR

    VAR_OUTPUT
        SENDTEXT                :STRING;
    I1, I2, I3, NC, CS      : REAL;
    EXODOS: INT;
    END_VAR

        VAR
        TP                                  :TP;
        RDY: BOOL;
      CLEANING              :CLEANING;
        MESDECOMPOSE    :MESDECOMPOSE;
        SPA_SEND            :SPA_SEND;
        K: INT;

       SNUM: STRING;
        CLEANDATA: STRING;
```

```
                SLASH1, SLASH2, SLASH3, SLASH4 : INT;
                  STRI1, STRI2, STRI3, STRI4, STRI5   : STRING;
                  AFTERSLASH1,    AFTERSLASH2,    AFTERSLASH3,
AFTERSLASH4 : STRING;

                  FINAL: STRING;
                END_VAR
```

ΚΩΔΙΚΑΣ

```
EXODOS:=1;
RDY:=FALSE;

IF EN= TRUE THEN
EXODOS:=0;

RDY:= FALSE;
SNUM:=INT_TO_STRING(L);
SENDTEXT:=INSERT('>RI1/5:XX',SNUM,1);
(*TP(IN:=TRUE, PT:=T#2MS);

      SPA_SEND(EN:=TP.Q,SENDTEXT:=SENDTEXT);*)



                RECTEXT:=PLC_PRG.RECTEXT;


                CLEANING(RECTEXT:=RECTEXT);
                FINAL:=CLEANING.FINAL;


      MESDECOMPOSE(MTYPE:='R',RECTEXT:=FINAL, L:=L);

      CLEANDATA:=MESDECOMPOSE.CLEANDATA;


      SLASH1:=FIND(CLEANDATA,'/');
      STRI1:=LEFT(CLEANDATA,SLASH1-1);
```

```
        AFTERSLASH1:=RIGHT(CLEANDATA, LEN(CLEANDATA)-
SLASH1);


        SLASH2:=FIND(AFTERSLASH1,'/');
        STRI2:=LEFT(AFTERSLASH1,SLASH2-1);
        AFTERSLASH2:=RIGHT(AFTERSLASH1,
LEN(AFTERSLASH1)-SLASH2);

        SLASH3:=FIND(AFTERSLASH2,'/');
        STRI3:=LEFT(AFTERSLASH2,SLASH3-1);
        AFTERSLASH3:=RIGHT(AFTERSLASH2,
LEN(AFTERSLASH2)-SLASH3);

        SLASH4:=FIND(AFTERSLASH3,'/');
        STRI4:=LEFT(AFTERSLASH3,SLASH4-1);
        AFTERSLASH4:=RIGHT(AFTERSLASH3,
LEN(AFTERSLASH3)-SLASH4);

        STRI5:=AFTERSLASH4;


        I1          :=STRING_TO_REAL(STRI1);
        I2          :=STRING_TO_REAL(STRI2);
        I3          :=STRING_TO_REAL(STRI3);
        NC          :=STRING_TO_REAL(STRI4);
        CS          :=STRING_TO_REAL(STRI5);
        EXODOS:=1;

    END_IF

    RDY:=TRUE;
```

Αυτό το function block αναλαμβάνει να πάρει την εισερχόμενη πληροφορία από ένα συγκεκριμένο μήνυμα, ζητάμε συγκεκριμένα I1, I2, I3, neutral current και configuration settings (NS, CS) και να τα προβάλει στον χρήστη του function block ο οποίος ενδιαφέρεται για αυτές τις τιμές. Τεμαχίζει την πληροφορία με τέτοιο τρόπο ώστε η σωστή πληροφορία να καταλήξει στο σωστό σημείο. Έχει σήμα ENABLE για να τίθεται σε λειτουργία, έχει έξοδο σήματος READY για να γνωρίζει ο χρήστης πότε η διεργασία έχει σταματήσει να τρέχει, και είναι έτοιμη, και έχει και έμμεση έξοδο σφάλματος την OK της MESDECOMPOSE που γίνεται FALSE σε περίπτωση σφάλματος. Έτσι εάν το READY είναι TRUE αλλά το OK είναι FALSE, σημαίνει πως ή δεν παίρνουμε απάντηση από τον συγκεκριμένο slave, ή πως μας στέλνει μήνυμα λάθους.

Χρησιμοποιώντας την έξοδο READY του ενός function block, στην είσοδο ENABLE του επόμενου, έχουμε ένα πλήρως σειριακό μοντέλο που

τρέχει συνεχώς χωρίς προβλήματα. Ένα τέτοιο παράδειγμα δίνεται στο κυρίως πρόγραμμα παρακάτω.

PLC_PRG

ΔΗΛΩΣΕΙΣ

```
PROGRAM PLC_PRG

VAR_INPUT

END_VAR


VAR

SPA_REC        :SPA_REC;
               SPAJ140C2       :SPAJ140C;
               SPAJ140C1       :SPAJ140C;
               SPA_SEND        :SPA_SEND;
               FINAL                   :STRING;
               SPA2EN          :BOOL;
               RECTEXT         : STRING;          (*RECORDED
TEXT*)
               RECRDY          : BOOL;            (*BOOLEAN
USED TO COMPARE THE READY STATE OF THE COM_REC*)
               RECERROR        : INT := 0;        (*RECORD
ERROR VALUE*)



               I: INT;
               SPA_INIT: SPA_INIT;
               BUSY: BOOL;
               RESULT: BOOL;
```

```
        I1,I2,I3,CS,NC                                  : REAL;
            I12,I22,I32,CS2,NC2         : REAL;
        TP:TP;
        RDY2: BOOL;
        EXODOS: INT;
        RDY1: BOOL;
        SENDTEXT: STRING;
        D: INT;
        SNUM: STRING;
        SL: INT;
        EN1: BOOL;
        EN2: BOOL;

END_VAR


VAR_OUTPUT

END_VAR
```

ΚΩΔΙΚΑΣ

```
        SPA_INIT;
            SPA_SEND(EN:=%MX255.1,
SENDTEXT:=SENDTEXT);


            SPA_REC;
            RECTEXT:=SPA_REC.RECTEXT;
            RECRDY:=SPA_REC.RECRDY;
            RECERROR:=SPA_REC.RECERROR;

                (*          D:=FIND(RECTEXT,'D');
                            SNUM:=MID(RECTEXT,2,D-2);
                        SL:=STRING_TO_INT(SNUM);
                            IF SL= 1 THEN*)


        SPAJ140C1(EN:=SPAJ140C2.RDY,RECTEXT:=RECTEXT,
L:=1);
            I1                  :=SPAJ140C1.I1;
            I2                  :=SPAJ140C1.I2;
            I3                  :=SPAJ140C1.I3;
            NC                  :=SPAJ140C1.NC;
            CS                  :=SPAJ140C1.CS;
            RDY1        :=SPAJ140C1.RDY;

        SENDTEXT:=SPAJ140C1.SENDTEXT;
```

```
        SPAJ140C2(EN:=SPAJ140C1.RDY,
RECTEXT:=RECTEXT, L:=2);
            I12         :=SPAJ140C2.I1;
            I22         :=SPAJ140C2.I2;
            I32         :=SPAJ140C2.I3;
            NC2   :=SPAJ140C2.NC;
            CS2   :=SPAJ140C2.CS;
            RDY2  :=SPAJ140C2.RDY;

        EXODOS:=SPAJ140C2.EXODOS;

        SENDTEXT:=SPAJ140C2.SENDTEXT;
```

Το αντικείμενο αυτό είναι το κεντρικό πρόγραμμα του project. Εδώ καλούνται όλα τα άλλα αντικείμενα ανάλογα με τις ανάγκες του χρήστη. Ο συγγραφέας επειδή δεν γνωρίζει ποιες θα είναι οι ανάγκες του χρήστη ουσιαστικά έχει κατασκευάσει ένα μικρό loop το οποίο στην αρχή καλεί τηνSPA_INIT, κάθε περίπου δυόμισι δευτερόλεπτα αλλάζει το EN της SPA_SEND, και έχει συνεχώς ανοιχτή την SPA_REC.

Υπήρχαν και κάποιες αρκετές μεταβλητές οι οποίες θα χρησιμοποιούνταν σε περίπτωση χρήσης ρολογιού στο project, όμως παρακάτω ο συγγραφέας θα αναπτύξει τις επιφυλάξεις του για αυτό.

Στην ουσία, το PLC_PRG το δηλαδή είναι ένα απλό test bench, οπότε σαν ένα test bench θα πρέπει να συμπεριφέρεται. Η δημιουργία ενός τέλεια οδηγούμενου κυρίως προγράμματος θα ήταν ανεπιθύμητη επειδή το test bench πρέπει να ελέγχει το πρόγραμμα σε μη ιδανικές συνθήκες και σε περιπτώσεις που τα σήματα ερχόντουσαν ασύγχρονα. Έπρεπε να ελέγχει το τι θα γινόταν εαν λόγου χάρη ο slave x έστελνε μήνυμα την στιγμή που το πρόγραμμα περίμενε απάντηση από τον slave y.

Συνιστάται η προσοχή του αναγνώστη στις δύο κλίσεις του function block SPAJ140C. Το ίδιο function block καλείται να λειτουργήσει σειριακά για δύο διαφορετικούς slaves (τον 1 και 2). Φαίνεται πως το ENABLE του 1 είναι feedback του SPAJ140C2.RDY σήματος, ενώ το ENABLE του 2 είναι feedback του σήματος SPAJ140C1.RDY του 1 αντίστοιχα.

Έτσι όταν τρέχει το ένα αντικείμενο, είναι σίγουρο πως δεν τρέχει το άλλο. Αυτό θα μπορούσε να υλοποιηθεί για άπειρους slaves αλλά στην πράξη μόνο 255 μπορούν να βρίσκονται ταυτόχρονα σε ένα δίκτυο.

Το εάν ο μελλοντικός χρήστης θα θελήσει να περάσει ρολόι στο project του, είναι κάτι που θα απασχολήσει αυτόν στο μέλλον αφού τα function blocks που ο συγγραφέας του κειμένου αυτού κατασκεύασε μπορούν να δουλέψουν και βασισμένα σε χρόνο (time_variant).

Στο σημείο αυτό θα πρέπει να αναφερθεί πως ο συγγραφέας των function blocks COMINIT, COMREC και COMSND που είναι τα components που αποτελούν τη βάση αυτού του project, αναφέρει μέσα στο help των function blocks αυτών, πως ο χρήστης θα πρέπει να αποφύγει να τα βάλει να λειτουργούν βάσει ρολογιού, και συνιστά (συγκεκριμένα αναφέρει 'it is strongly recommended') να καλούνται σε 'FREE MODE' δηλαδή εκτός ρολογιού(ABB, 2000).

ΕΠΙΛΟΓΟΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ

Θα πρέπει να αναφερθεί ότι όλη η παραπάνω εργασία δοκιμάστηκε σε συνθήκες εργαστηρίου με ένα PLC και δύο ηλεκτρονόμους.

Για το σκοπό αυτό αναπτύχθηκε μια τυπική εφαρμογή συλλογής δεδομένων από τους ηλεκτρονόμους.

Η εφαρμογή λειτούργησε με επιτυχία.

Οι συνθήκες εργαστηρίου εάν εξαιρεθούν τα μήκη των οπτικών ινών καθώς και το πλήθος των ηλεκτρονόμων, ταυτίζονται με τις πραγματικές ενός τυπικού υποσταθμού μέσης τάσης.

Προκειμένου να δημιουργηθούν συνθήκες πλησιέστερες στις πραγματικές, ο ένας ηλεκτρονόμος εκ των δύο, είχε συνδεθεί με ένα ροοστάτη με δύο ακροδέκτες ο οποίος τροφοδοτούνταν με ρεύμα και στην άλλη άκρη του επέστρεφε διαφορετικές τιμές έντασης ανάλογα με την ρύθμιση. Έτσι ο συγγραφέας γνώριζε εάν το πρόγραμμα ανταποκρινόταν σωστά.

Γενικά τα αντικείμενα που κατασκευάστηκαν απείχαν σε κάποια σημεία από τον αρχικό σχεδιασμό, όμως αυτός δεν θα μπορούσε να είναι άρτιος εφόσον γινόταν ως ένα πλάνο. Όπου οι συνθήκες το απαίτησαν, ο συγγραφέας θεώρησε σωστό να ξεφύγει μέσα σε λογικά πάντα πλαίσια από τον αρχικό σχεδιασμό έτσι ώστε το τελικό αποτέλεσμα να είναι λειτουργικότερο.

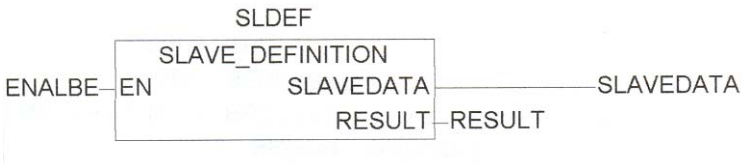
Τα αντικείμενα του project εγγυώνται στον μελλοντικό προγραμματιστή συστημάτων ελέγχου πως θα έχει πλέον τη δυνατότητα μέσα από μία προκαθορισμένη και σωστά δομημένη διαδικασία να πάρει και να στείλει δεδομένα από ένα δίκτυο ηλεκτρονόμων, χωρίς τη χρήση ειδικού εξοπλισμού. Απλά στην ίδια εφαρμογή θα πρέπει:

Α) να περιλαμβάνει τα function blocks που αναπτύχθηκαν στην παρούσα εργασία

Β) να καλεί αυτά τα function blocks σύμφωνα με τους κανόνες όπως αναφέρονται στο παρόν κείμενο.

ΚΕΙΜΕΝΟ ΒΟΗΘΕΙΑΣ ΤΗΣ ΒΙΒΛΙΟΘΗΚΗΣ

HELP SLAVE_DEFINITION



Το function block αυτό στέλνει ένα συγκεκριμένο σήμα σε όλες τις διευθύνσεις και καταχωρεί τις απαντήσεις σε ένα πίνακα βάζοντας τον κάθε slave σε μια συγκεκριμένη θέση ανάλογα με τον αριθμό επικοινωνίας του.

Ο πίνακας εισόδων εξόδων δίνεται παρακάτω.

Είσοδοι

EN	Σήμα ενεργοποίησης του function block

Έξοδοι

RESULT	Αποτέλεσμα έρευνας και λειτουργίας του function block
SLAVEDATA	Πίνακας καταχώρησης τιμών και μοντέλων slaves

EN

Η μεταβλητή αυτή είναι τύπου BOOLEAN. Πρέπει να γίνει TRUE προκειμένου να ξεκινήσει το function block να λειτουργεί.

RESULT

Η μεταβλητή αυτή είναι τύπου BOOLEAN. Η μεταβλητή αυτή γίνεται TRUE όταν το function block τελειώσει την λειτουργία του. Μέχρι τότε είναι FALSE.

SLAVEDATA

Είναι ένας πίνακας που αποτελείται από 256 θέσεις STRING των 13 θέσεων. Εδώ το function block καταχωρεί τις απαντήσεις που στέλνουν οι slaves από τους οποίους το function block ζητά να του επιστρέψουν το μοντέλο τους.

ΕΙΔΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ

Το function block αυτό έχει ιδιαιτέρα χαρακτηριστικά λειτουργίας. Αυτά δίνονται παρακάτω ώστε ο χρήστης να μπορέσει να το εκμεταλευτεί στο μέγιστο.

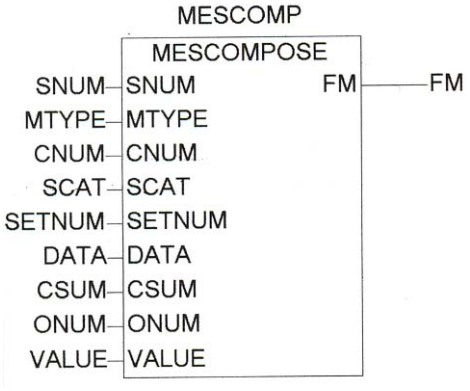
- Είναι πλήρως ανισόχρονο
- Είναι πλήρως σύγχρονο αν και μπορεί να λειτουργήσει και ασύγχρονα, με ευθύνη όμως του χρήστη

Πρέπει να καλείται σε πάρα πολύ προσεγμένο χρόνο, γιατί χρειάζεται 256 κύκλους του plc για μία εκτέλεση, άρα πρέπει να καλείται προσεκτικά και σε στιγμές που θα πρέπει να επηρεάζει το σύστημα όσο το δυνατόν λιγότερο.

Παράδειγμα κλίσης της SLAVE_DEFINITION σε ST

```
SLAVE_DEFINITION(EN:=VAR1);

VAR2:=SLAVE_DEFINITION.SLAVEDATA;
VAR3:=SLAVE_DEFINITION.RESULT;
```

HELP MESCOMPOSE



Το function block αυτό αναλαμβάνει να κατασκευάσει σωστά το εξερχόμενο μήνυμα που ο master της επικοινωνίας σύμφωνα με το SPA Bus protocol πρέπει να στείλει προκειμένου το μήνυμα αυτό να θεωρηθεί έγκυρο. Το function block αναλαμβάνει να βάλει στην σωστή σειρά τα δεδομένα που έχουν να κάνουν με τον αριθμό του χρήστη, το νούμερο του επιθυμητού καναλιού κ.τ.λ.

Παρακάτω δίνεται η σημασιολογία των μεταβλητών.

Είσοδοι

SNUM	Ο αριθμός του slave
MTYPE	Ο τύπος του τηλεγραφήματος
CNUM	Ο αριθμός του καναλιού
SCAT	Η κατηγορία των settings
SETNUM	Ο αριθμός των settings
DATA	Τα δεδομένα
CSUM	Το Checksum
ONUM	Αριθμός εξόδου (μόνο για εγγραφές)
VALUE	Η τιμή που πρέπει να πάρει η έξοδος (μόνο για εγγραφές)

Έξοδος.

FM	Τελικό μήνυμα.

SNUM.

Είναι μεταβλητή τύπου string. Δηλώνεται εδώ ο αριθμός του slave στον οποίο πρέπει να γίνει η αποστολή.

MTYPE

Είναι μεταβλητή τύπου string. Εδώ δίνεται ο τύπος του μηνύματος. Η MTYPE μπορεί να χρησιμοποιείται και από άλλα function blocks προκειμένου να γνωρίζουν τι τύπου ήταν το εξερχόμενο τηλεγράφημα.

Ανάλογα με τον τύπο του τηλεγραφήματος, άλλες είσοδοι είναι χρήσιμες, και άλλες δεν είναι. Έτσι σε περίπτωση write, οι είσοδοι CNUM, SCAT, SETNUM και DATA είναι άχρηστες, ενώ κατά τα read οι είσοδοι ONUM και VALUE δεν εξυπηρετούν σε τίποτα.

Παρότι φαινομενικά δεν θα πείραζε οι άχρηστες τιμές να παραμένουν μη μηδενικές, καλό είναι να μηδενίζονται.

CNUM

Ο αριθμός του καναλιού του slave στο οποίο ο master απευθύνεται. Ηλεκτρονόμοι της σειράς SPAJ, SPAM, κ.τ.λ. διαθέτουν μόνο ένα κανάλι, το κανάλι ‘0’ οπότε για τέτοια μοντέλα αυτή η είσοδος είναι πάντα ‘0’ δίνοντας όμως στον χρήστη την δυνατότητα αλλαγής. Είναι μεταβλητή τύπου string.

SCAT

Είναι μεταβλητή τύπου string. Εδώ δηλώνεται ο χαρακτήρας του setting το οποίο ο master επιθυμεί να γνωρίζει. Αυτή η μεταβλητή είναι αλληλένδετη με την επόμενη και πρέπει να εξετάζονται μαζί.

SNUM

Είναι και αυτή, μεταβλητή τύπου string. Εδώ εισάγεται ο αριθμός του setting. Για παράδειγμα εάν ο master θέλει να διαβάσει τι έχει η θέση V201 ενός slave, θα πρέπει να κάνει την SCAT=V και την SNUM=1.

DATA

Είναι και αυτή τύπου string. Η πραγματική έννοια αυτής της μεταβλητής είναι additional data. Δηλαδή εάν κάποιος θέλει να διαβάσει πολλαπλές τιμές, καταχωρεί την τελική τιμή (όπως γίνεται σύμφωνα με το SPA Bus protocol) σε αυτή την μεταβλητή.

CSUM

Είναι μεταβλητή τύπου string. Εδώ καταχωρείται το checksum του master. Ο συγγραφέας θεώρησε σωστό να μην καταχωρήσει το checksum σε σταθερά, γιατί υπάρχει περίπτωση σε μελλοντικές εκδόσεις να

τροποποιηθεί ή ακόμα πιθανότερα, ο μελλοντικός προγραμματιστής να το κρατήσει για μελλοντική διαφορετική χρήση.

ONUM
Είναι μεταβλητή τύπου string. Εδώ ο master δηλώνει το σημείο της διεύθυνσης – εξόδου του slave στο οποίο θα γίνει το write.

VALUE
Είναι μεταβλητή τύπου string. Εδώ δίνεται η τιμή της εγγραφής. Χρησιμοποιείται σε αλληλουχία με την προηγούμενη μεταβλητή.

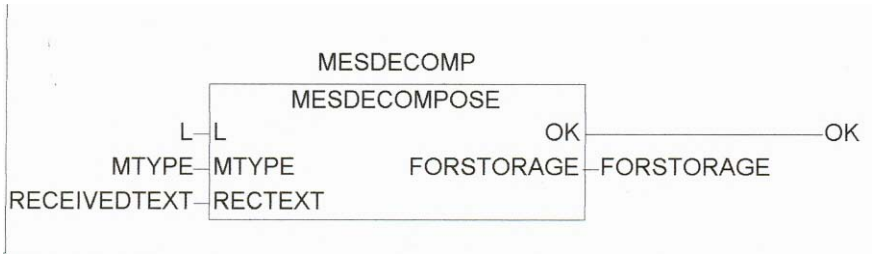
Ειδικές πληροφορίες γύρω από την μελλοντική χρήση του function block

- Το συγκεκριμένο function block δεν έχει trigger.
- Είναι ασύγχρονο έως ισόχρονο.
- Είναι εκ τούτης πολύ καλύτερο να καλείται εσωτερικά.
- Χρειάζεται πολύ προσοχή στην χρήση των πληροφοριών του.

Παράδειγμα κλήσης του Function block MESCOMPOSE σε ST.

```
MESCOMPOSE(SNUM:=VAR1, MTYPE:=VAR2, CNUM:=VAR3,
SCAT:=VAR4, SNUM:=VAR5, DATA:=VAR6, CSUM:=VAR7,
ONUM:=VAR8, VALUE:=VAR9);

VAR10:=MESCOMPOSE.FM;
```

HELP MESDECOMPOSE



Η MESDECOMPOSE είναι αντικείμενο τύπου function block που αναλαμβάνει να καθαρίσει ένα εισερχόμενο προς τον master τηλεγράφημα σύμφωνα με το SPA Bus Protocol, από τα περιττά δεδομένα αφήνοντας μόνο την χρήσιμη πληροφορία.

Η λειτουργία της επεξηγείται παρακάτω.

Είσοδοι.

L	Ο αριθμός του slave
MTYPE	Ο τύπος του προηγούμενου εξερχόμενου μηνύματος.
RECTEXT	Το καταγεγραμμένο εισερχόμενο τηλεγράφημα.

Έξοδοι

OK	Τελικό αποτέλεσμα σωστού/λάθους μηνύματος.
FORSTORAGE	Καθαρή προς αποθήκευση πληροφορία.

ΑΝΑΛΥΣΗ ΜΕΤΑΒΛΗΤΩΝ

L

Η μεταβλητή αυτή είναι τύπου INT. Εδώ καταχωρείται σε INT ο αριθμός του slave, για τον οποίο προοριζόταν το προηγούμενο εξερχόμενο τηλεγράφημα.

MTYPE

Αυτή η μεταβλητή είναι τύπου STRING. Εδώ είναι καταχωρημένος ο τύπος του προηγούμενου εξερχόμενου τηλεγραφήματος. Όπως φαίνεται, η μεταβλητή αυτή είναι όμοια με την MTYPE της MESCOMPOSE. Με

σωστό time management ο χρήστης μπορεί να την χρησιμοποιεί ταυτόχρονα.

RECTEXT
Αυτή είναι επίσης μια μεταβλητή τύπου STRING. Εδώ είσοδος είναι όλο το εισερχόμενο τηλεγράφημα έτσι ώστε το function block να μπορέσει να το επεξεργαστεί, για να εξάγει από αυτό, την χρήσιμη πληροφορία.

OK
Μια έξοδος τύπου BOOL η οποία γίνεται TRUE όταν το εισερχόμενο τηλεγράφημα ήταν μήνυμα σωστής ανάγνωσης της αίτησης του master και FALSE όταν η απάντηση του slave, είναι μήνυμα λάθους.

FORSTORAGE
Μεταβλητή τύπου STRING. Εδώ αποθηκεύεται η καθαρή, χρήσιμη (προς τον χρήστη) πληροφορία. Αυτή επιστρέφεται με τον αριθμό του slave να προπορεύεται, ακολουθούμενο από slash και την πληροφορία στην συνέχεια.

ΕΙΔΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΠΡΟΣ ΤΟΝ ΜΕΛΛΟΝΤΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ.

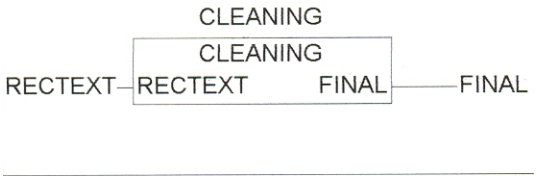
Το function block αυτό έχει τα εξής κεντρικά χαρακτηριστικά

- Είναι ισόχρονο
- Είναι τελείως ασύγχρονο
- Όπως είναι εμφανές; Από τα παραπάνω, καλό είναι να καλείται εσωτερικά.
- Η σχεδίασή του είναι τέτοια, για βέλτιστες επιδόσεις, για αυτό, πρέπει να δίνεται ιδιαίτερη προσοχή σε θέματα ταυτοχρονισμού για την αποφυγή λάθους (WAR Hazards).

Παράδειγμα κλίσης της MESDECOMPOSE σε ST.

```
MESDECOMPOSE(L:=VAR1, MTYPE:=VAR2, RECTEXT:=VAR3);

VAR4:=MESDECOMPOSE.OK;
VAR5:=MESDECOMPOSE.FORSTORAGE);
```

HELP CLEANING



Το CLEANING είναι ένα αντικείμενο τύπου function block. Εάν ο slave έχει στείλει το τηλεγράφημα της απάντησής του στον master σε πολλά συνεχιζόμενα μηνύματα, η CLEANING αναλαμβάνει να τα συνενώσει ξανά σε ένα μεγάλο ενιαίο.

Οι είσοδοι και έξοδοι είναι οι εξής:

Είσοδοι.

RECTEXT	Το εκάστοτε εισερχόμενο μήνυμα

Έξοδοι

FINAL	Το ολοκληρωμένο εισερχόμενο τηλεγράφημα

RECTEXT
Η παραπάνω μεταβλητή είναι τύπου STRING. Είναι το καταγεγραμμένο εισερχόμενο μήνυμα από την σειριακή θύρα με την βοήθεια του function block COMREC (ή την παραλλαγή του SPA_REC).

FINAL
Είναι μεταβλητή τύπου STRING. Εδώ καταχωρείται το τελικό αποτέλεσμα το οποίο περιέχει όλα τα συνενωμένα επιμέρους μηνύματα έτσι ώστε να προκύψει ολόκληρο το εισερχόμενο τηλεγράφημα.

ΕΙΔΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ.

Το παραπάνω function block, λειτουργεί πολύ καλά σε συνεργασία με το MESDECOMPOSE. Το παραπάνω χρειάζεται ολόκληρο τηλεγράφημα για να λειτουργήσει οπότε καλό είναι το CLEANING να καλείται πρώτα. Επίσης ισχύει:

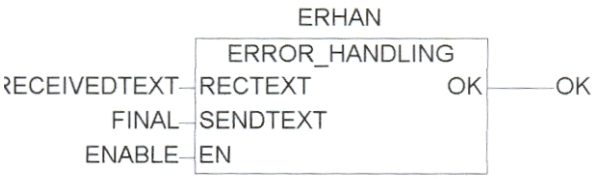
- Είναι ισόχρονο και σειριακό
- Είναι πλήρως ασύγχρονο
- Ενδείκνυται για εσωτερικές κλίσεις

Όπως και με το MESDECOMPOSE, έτσι και το CLEANING χρειάζεται προσοχή στα φαινόμενα ταυτοχρονισμού του.

Παράδειγμα κλίσης του function block CLEANING.

```
CLEANING(RECTEXT:=VAR1);
VAR2:=CLEANING.FINAL;
```

HELP ERROR_HANDLING



Το αντικείμενο ERROR_HANDLING ανήκει στην κατηγορία των function blocks.

Η εργασία που εκτελεί είναι να δεσμεύει το σύστημα για κάποιο χρονικό διάστημα μέσα στο οποίο στέλνει ένα εξερχόμενο τηλεγράφημα σε έναν συγκεκριμένο slave. Σε περίπτωση που ο slave στείλει θετικό μήνυμα ανταπόκρισης, η OK γίνεται TRUE. Ο πίνακας εισόδων-εξόδων δίνεται παρακάτω.

Είσοδοι

EN	Σήμα ενεργοποίησης του function block
RECTEXT	Το εισερχόμενο τηλεγράφημα
SENDTEXT	Τηλεγράφημα προς αποστολή

Έξοδοι

OK	Αποτέλεσμα αποστολής

EN

Αυτή είναι μια μεταβλητή τύπου BOOLEAN. Όταν η μεταβλητή αυτή δηλωθεί ίση με ‘TRUE’, τότε το function block μπορεί να κλιθεί κανονικά. Ισχύει και το αντίστροφο.

RECTEXT

Αυτή είναι μια μεταβλητή τύπου STRING. Εδώ ο προγραμματιστής πρέπει να καταχωρήσει το εισερχόμενο μήνυμα από τον receive buffer.

SENDTEXT
Αυτή είναι μια μεταβλητή τύπου STRING. Εδώ ο χρήστης καταχωρεί το μήνυμα η αποστολή του οποίου έφερε ως αποτέλεσμα την απάντηση με μήνυμα λάθους από μεριάς του slave.

OK
Αυτή είναι μια μεταβλητή τύπου BOOLEAN. Είναι η έξοδος του function block. Όταν υπάρξει θετική απάντηση από τον slave, η OK γίνεται TRUE, αλλιώς παραμένει FALSE.

ΕΙΔΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ.

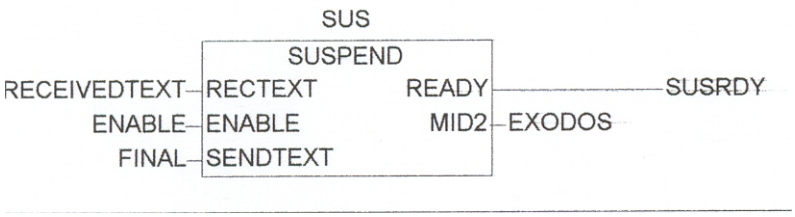
Το function block αυτό διακρίνεται από τα παρακάτω χαρακτηριστικά

- Είναι πλήρως ανισόχρονο
- Είναι πλήρως σύγχρονο
- Μπορεί να καλείται μόνο σε συγκεκριμένες στιγμές, αφού μονοπωλεί το σύστημα για περίπου 2 δευτερόλεπτα έτσι ώστε να αδειάσει όλος ο receive buffer από μηνύματα που μπορεί να έχουν απομείνει.

Παράδειγμα κλίσης της ERROR_HANDLING στην γλώσσα ST.

```
ERROR_HANDLING( EN:=VAR1,
                SENDTEXT:=VAR2,
                RECTEXT:=VAR3);

                VAR4:=ERROR_HANDLING.OK;
```

HELP SUSPEND



Το αντικείμενο SUSPEND είναι τύπου function block. Αποστέλλει ένα αποθηκεμένο εξερχόμενο τηλεγράφημα σε έναν συγκεκριμένο slave κάθε φορά που το ενεργοποιεί ο χρήστης. Μετά από 100 αποτυχημένες απόπειρες επιστρέφει στον χρήστη ένα μήνυμα αποτυχίας. Είναι κατασκευασμένη για να θέτει ηλεκτρονόμους που δεν απαντούν αμέσως σε κατάσταση SUSPEND πριν γίνουν time out.

Είσοδοι

RECTEXT	Το εισερχόμενο μήνυμα
ENABLE	Σήμα ενεργοποίησης
SENDTEXT	Το εξερχόμενο μήνυμα

Έξοδοι

READY	Σήμα κατάστασης του function block
MID2	Μήνυμα σφάλματος προς τον χρήστη

RECTEXT

Αυτή η μεταβλητή είναι τύπου STRING. Ο χρήστης εδώ πρέπει να συνδέσει εισερχόμενο από τον receive buffer μήνυμα, έτσι ώστε η SUSPEND να γνωρίζει εάν ο επιθυμητός slave απάντησε, και μάλιστα όχι με μήνυμα σφάλματος

ENABLE

Αυτή είναι μια μεταβλητή τύπου BOOLEAN. Εδώ ο προγραμματιστής θα πρέπει να θέσει την τιμή TRUE όταν θα θελήσει να χρησιμοποιήσει το function block αυτό.

SENDTEXT

Η SENDTEXT είναι άλλη μια μεταβλητή τύπου STRING. Εδώ δίνεται το εξερχόμενο τηλεγράφημα, προς τον slave που δεν απαντά.

READY

Μεταβλητή τύπου BOOLEAN. Εδώ το function block επιστρέφει TRUE εάν έχει τελειώσει να λειτουργεί και FALSE εάν για οποιονδήποτε λόγο δεν είναι έτοιμο να λειτουργήσει (κυρίως επειδή μπορεί να τρέχει ακόμα εκτελώντας μια εργασία που μπορεί να του ανατέθηκε νωρίτερα).

MID2

Τέλος αυτή είναι μια μεταβλητή τύπου STRING. Εδώ το function block, εάν μετά από όλες τις επαναλήψεις, ο slave δεν απαντήσει, επιστρέφει στον χρήστη ένα μήνυμα της μορφής:

«xxx/WARNING PROBABLE HARDWARE ERROR OF FAILURE OCCURRED»

Ο αριθμός του slave.

ΕΙΔΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ

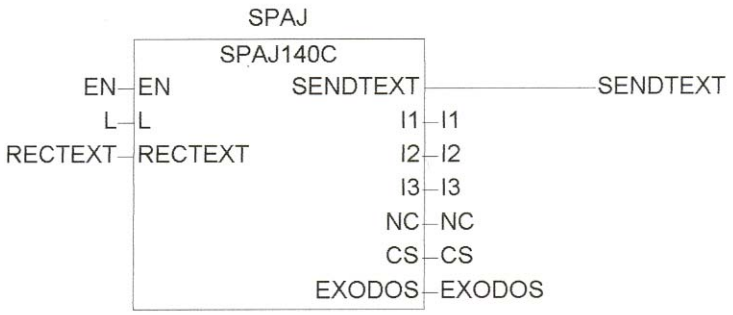
Το function block έχει τα παρακάτω χαρακτηριστικά.

- Είναι πλήρως ανισόχρονο (μια εκτέλεσή του απαιτεί τουλάχιστον 100 κύκλους)
- Υποστηρίζει πλήρως παράλληλη λογική εκτέλεσης
- Μπορεί να λειτουργεί και σειριακά με άλλα function blocks άλλου ή του ίδιου είδους

Παράδειγμα κλίσης του function block σε ST

```
SUSPEND(EN:=VAR1, RECTEXT:=VAR2, SENDTEXT:=VAR3);

VAR4:=SUSPEND.READY;
VAR5:=SUSPEND.MID2;
```

HELP SPAJ140C



Είναι ένα αντικείμενο τύπου function block. Δέχεται σαν εισόδους κάποιες πληροφορίες και επιστρέφει σαν έξοδο τις τιμές των εντάσεων στα άκρα του ηλεκτρονόμου, καθώς και την τιμή της ουδέτερης έντασης και την ένδειξη σφάλματος ή όχι.

Παρακάτω δίνεται ο πίνακας εισόδων / εξόδων.

Είσοδοι

EN	Σήμα ενεργοποίησης
L	Ο αριθμός του slave
RECTEXT	Το εισερχόμενο μήνυμα

Έξοδοι

SENDTEXT	Το κείμενο προς αποστολή
I1	Η τιμή του I1
I2	Η τιμή του I2
I3	Η τιμή του I3
NC	Η τιμή του ουδέτερου ρεύματος
CS	Η κατάσταση του κυκλώματος (κλειστό/ανοιχτό), μπλοκαρισμένο/όχι

EXODOS	Control έξοδος του function block

EN

Είναι μια μεταβλητή τύπου BOOLEAN. Το function block καλείται μόνο εάν η EN είναι TRUE ενώ εάν είναι FALSE η εκτέλεση του κώδικα συνεχίζει παρακάτω.

L

Είναι μεταβλητή τύπου INT. Εδώ είσοδος είναι ο αριθμός του slave με τον οποίο το function block πρέπει να επικοινωνεί.

RECTEXT

Αυτή είναι μεταβλητή τύπου STRING. Είναι είσοδος. Εδώ δηλώνεται το εξωτερικό εισερχόμενο τηλεγράφημα, το οποίο δέχεται επεξεργασία, έτσι ώστε να βγει από αυτό η χρήσιμη πληροφορία.

SENDTEXT

Άλλη μια μεταβλητή τύπου STRING. Εδώ το function block επιστρέφει το τηλεγράφημα που πρέπει να στείλει, προκειμένου να πάρει από τον slave τις τιμές που επιθυμεί. Το κείμενο είναι ήδη έτοιμο καθότι οι σημαντικές για τον χρήστη τιμές στους ηλεκτρονόμους των σειρών SPAJ, SPAM κ.τ.λ. είναι συγκεκριμένες.

I1, I2, I3

Αυτές είναι μεταβλητές τύπου REAL. Σε αυτές αποθηκεύονται οι τιμές I1, I2 και I3 αντίστοιχα, που καταγράφει ο ηλεκτρονόμος στα άκρα του.

NC

Μεταβλητή τύπου REAL. Εδώ αποθηκεύεται κάθε στιγμή η τιμή του της ουδέτερης έντασης (neutral current) που καταγράφει ο ηλεκτρονόμος στους ακροδέκτες του.

CS

Μεταβλητή τύπου REAL. Θα μπορούσε να είναι και μεταβλητή τύπου BOOLEAN καθότι η CS παίρνει τιμές 0 ή 1, όμως ο συγγραφέας την αφήνει με περισσότερες δυνατότητες για τυχόν μελλοντική χρήση. Εδώ καταγράφεται το κατά πόσο ο ηλεκτρονόμος έχει πάρει την πρωτοβουλία να στείλει «blocking signal» στο προστατευμένο κύκλωμα ανοίγοντας το.

EXODOS

Είναι μεταβλητή τύπου INT. Εδώ το function block επιστρέφει την κατάστασή του.

- Εάν εκτελεί επιστρέφει ‘0’.
- Εάν έχει εκτελέσει με επιτυχία, επιστρέφει ‘1’
- Εάν έχει εκτελέσει ανεπιτυχώς επιστρέφει ‘-1’

Στην ουσία υλοποιείται ένας tri state buffer όμως για πάρα πολλούς λόγους ο συγγραφέας πιστεύει πως στο μέλλον τα εξερχόμενα control σήματα μπορεί να χρειαστεί να γίνουν πολύ περισσότερα, οπότε αφήνει και την δυνατότητα για αυτό.

ΕΙΔΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΛΕΙΤΟΥΡΓΙΑΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ

Το αντικείμενο αυτό είναι κατασκευασμένο με τα παρακάτω χαρακτηριστικά

- Ισόχρονο για μία εκτέλεση
- Πλήρως συγχρονισμένο
- Πλήρως σειριοποιήσιμο με άλλα
- Πρέπει για να λειτουργήσει, να υπάρχει η παρουσία των CLEANING και MESDECOMPOSE
- Πλήρως συνεργάσιμο με τα παραπάνω.
- Οι τιμές επιστρέφονται σε real time

Όπως φαίνεται από τα παραπάνω, πολλαπλά τέτοια αντικείμενα μπορούν να συνδεθούν σειριακά, ή και με άλλους priority τρόπους προκειμένου να λειτουργούν το κάθε ένα για ένα μικρό χρονικό διάστημα ώστε στην ουσία να λειτουργούν όλα ταυτόχρονα. Δίνεται η δυνατότητα σύνδεσής του και με άλλα αντικείμενα άλλων μορφών. Το function block έχει πολλές ασφαλιστικές δικλίδες ώστε να προβλέπει και να ακυρώνει κάθε μορφή σφάλματος.

Παράδειγμα κλίσεως της SPAJ140C σύμφωνα με την γλώσσα ST.

```
SPAJ140C(EN:=VAR1, L:=VAR2, RECTEXT:=VAR3);

VAR4      := SPAJ140C.SENDTEXT;
VAR5      := SPAJ140C.I1;
VAR6      := SPAJ140C.I2;
VAR7      := SPAJ140C.I3;
VAR8      := SPAJ140C.NC;
VAR9      := SPAJ140C.CS;
VAR10     := SPAJ140C.EXODOS;
```

Ο συγγραφέας δηλώνει πως η βιβλιογραφία είναι γραμμένη και παρατίθεται σύμφωνα με το Harvard system.

Βιβλιογραφία

- ABB (2002)SPAJ 140 C overcurrent and earth-fault relay, Version B, FIN –65101 VAASA, Finland

- ABB (2000)SPA Bus communication protocol V 2.5, technical description , FIN –65101 VAASA, Finland

- ABB (2000) Advant controller 31 Intelligent Decentralized Automation System, Function Block Library 90 series, Software Description, ABB STOTZ-KONTAKT Gmph, Heidelberg, Germany

- www.ABB.com, visited at 20/10/2004,

- Καλουπτσίδης Νίκος, (1995)Σήματα Συστήματα και Αλγόριθμοι, πρώτη έκδοση, Κλεάνθους Κ., εκδόσεις ΔΙΑΥΛΟΣ, Αθήνα, ISBN 960-7140-40-0

- Παρασκευόπουλος Π. Ν., (1991)Εισαγωγή στον Αυτόματο Έλεγχο, πρώτη έκδοση, Ε.Μ.Π., Αθήνα